A study of algorithms for parallel computers and VLSI systolic processor arrays

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:


Metadata Record: [https://dspace.lboro.ac.uk/2134/27483](https://dspace.lboro.ac.uk/2134/27483)

Publisher: © Michael P. Bekakos

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 2.5 Generic (CC BY-NC-ND 2.5) licence. Full details of this licence are available at: [http://creativecommons.org/licenses/by-nc-nd/2.5/](http://creativecommons.org/licenses/by-nc-nd/2.5/)

Please cite the published version.
This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
A STUDY OF ALGORITHMS

FOR

PARALLEL COMPUTERS AND VLSI

SYSTOLIC PROCESSOR ARRAYS

VOLUME - II

BY

MICHAEL P. BEKAKOS
B.Sc.(Hons.), M.Sc., M.H.M.S.

A Doctoral Thesis
Submitted in partial fulfilment of the requirements
for the Award of Doctor of Philosophy
of the Loughborough University of Technology

SUPERVISOR: PROFESSOR DAVID J. EVANS
B.Sc., M.Sc., Ph.D., D.Sc., F.I.M.A., F.B.C.S.
Department of Computer Studies

This Research is Sponsored by NATO and the Greek Ministry
of National Economy under the Science Fellowship Contract
No. [ΔΣ5450/TB.2391/15-7-81/AYE-(AE.347/32213/Tpow.6)].

ABSTRACT

In this Thesis the design and analysis of parallel algorithms is investigated under the framework of, either, being suitable for execution on asynchronous Multiprocessor testbeds (MIMD organizations), or, due to the recent remarkable advance of 'Very Large Scale Integrated' - VLSI circuitry, of being suitable for direct hardware implementation.

In the first three introductory Chapters a brief and taxonomically disciplined state-of-the-art survey is presented with up-to-date information on the parallel computing environment. This survey is relatively complemented by the contents of the last Chapter VIII, where most of the envisaged technological advancements are discussed.

More analytically, Chapter I is devoted to the overview of parallel computer systems and prototypes. After the exploitation of parallelism in various parallel computer structures, in terms of classifying the various architectural designs, the Chapter continues with the genealogical taxonomy of the main current multiple processor complexes.
In Chapter II the programming tools and algorithms to exploit the parallel hardware potential are introduced. In particular, concurrent programming languages motivations and general concepts for parallel processing are discussed, to continue with various methodological design and analysis aspects of parallel algorithms to appropriately map onto the different architectural categories.

In both these Chapters particular reference has been made to the 'NEPTUNE' MIMD prototype, sited at the Department of Computer Studies, at Loughborough University of Technology, on which the bulk of the experimental work contained herein was carried out.

Developments in microelectronics have revolutionized computer design. VLSI technology has enormously increased the number and complexity of components that can fit on a chip. As a result, machines-on-a-chip have emerged; these machines can be used as special-purpose devices attached to a conventional 'host' computer. In Chapter III, at first, various computational models and 'Knowledge Information Processing Systems' - KIPS are introduced, to continue with the embedding of information flow schemes on grids and in VLSI chip area and time.

An extensive investigation on the potential parallelism of a new powerful class of Group Explicit methods compared to the Standard Explicit method is carried out in Chapter IV, for the solution of parabolic partial differential equations. For the performance analysis, on the provided MIMD testbed, of all parallel implementations in the Thesis, a detailed 'Deterministic Performance Model' - DPM is established along with all the particular general formulae for the estimation of its various parameters.

A complete performance exploitation of the 'NEPTUNE' MIMD
prototype is pursued in Chapter V, by implementing several parallel algorithms using the Cyclic Odd-Even reduction technique, in combination with all the possible parallel constructs for the system, to solve Toeplitz tridiagonal linear systems for use in signal and image processing applications. The Chapter continues with the implementation of several new parallel algorithms using the same technique, to solve general periodic and non-periodic tridiagonal linear systems, following an alternate approach for the utilization of any number of processors.

In Chapter VI the research is being concentrated on algorithmically specialized systolic networks. A new powerful 'rotating' and 'folding' technique is introduced and applied on two-dimensional systolic communication geometries to solve a variety of occurring problems. In particular, at first, the matrix-vector and matrix multiplication problems are treated; then, the method is applied to tridiagonal and quindiaonal linear systems, and eventually generalized for p semi-bandwidth linear systems. To bypass the complexity arising along with the increase of the semi-bandwidth of the coefficient matrix, an alternative 'unidirectional' factorization of the central formatted submatrix is proposed and exemplified. The resulting upper and lower triangular linear systems are solved again by the new method using a linear systolic array of processors.

Finally, in Chapter VII, single stage computational dewavefronts are investigated, as an expansion of the 'rotate' and 'fold' method, for the implementation of the 'Quadrant Interlocking Factorization' - QIF parallel method on a data-driven 'Wavefront Array Processor' - WAP, for the case that the coefficient matrix of the linear system is a
compact dense (n×n) matrix.

The Thesis is concluded with general comments on future computer architectures, overviewing conclusions and a discussion for further future research topics in this area. References and Appendices with complementary theory and proofs, where needed, and a selection of optimized parallel computer programs from our experimental work are also included.
TABLE OF CONTENTS

DECLARATION

ACKNOWLEDGEMENTS

ABSTRACT

LIST OF FIGURES

LIST OF TABLES

VOLUME I

A TREE-LIKE LOGICAL SUBDIVISION OF CHAPTER I CONTENTS

CHAPTER I: AN OVERVIEW OF PARALLEL COMPUTER ARCHITECTURES 27

SECTION A: EXPLOITATION OF PARALLELISM IN VARIOUS PARALLEL COMPUTER ARCHITECTURES 28

I.A.1: The Innovation of the Parallel Notion 29

I.A.2: Classification of Designs 35

I.A.2.1: Flynn's Very High-Speed Computing Systems 35

I.A.2.2: Shore's Taxonomy 41

I.A.2.3: Other Classification Approaches 44

SECTION B: THE MAIN CURRENT MULTIPLE PROCESSOR ARCHITECTURES 46

I.B.1: Introduction 47

I.B.2: The Genealogy of the SIMD Organization 50

I.B.3: The Utilization and Application of the SIMD Systems 52
I.B.3.1: The Associative Processor Architecture

I.B.3.1.1: The Associative Memory Organization

I.B.3.1.2: Architectural Taxonomy of the Associative Processors

I.B.3.1.2.i: Fully Parallel Associative Processors

I.B.3.1.2.ii: Bit-Serial Associative Processors

I.B.3.1.2.iii: Word-Serial Associative Processors

I.B.3.1.2.iv: Block-Oriented Associative Processors

I.B.3.1.2.v: Highly Parallel Associative Processors

I.B.3.2: Parallel Processor Architectures

I.B.3.2.1: The Utilization of Parallel Memories

I.B.3.2.2: The Interconnection Networks

I.B.3.2.3: Implemented Parallel Processor Systems

I.B.3.2.3.i: The Orthogonal Computer Concept

I.B.4: The Genealogy of the Pipelined Vector Organization

I.B.4.1: Pipeline as a Fundamental Design Principle and Performance Characteristics

I.B.4.2: Vector Processing Characteristics

I.B.4.3: Implemented Pipelined Vector Computers

I.B.5: The MIMD Multiprocessing Architectures

I.B.5.1: The MIMD Hardware System Organization

I.B.5.1.1: The Time-Shared/Common Bus Interconnection Schema

I.B.5.1.2: The Crossbar Switch Matrix Interconnection Schema

I.B.5.1.3: The Multibus/Multiport Interconnection Schema
A Tree-like Logical Subdivision of Chapter II Contents

Chapter II: Programming Tools and Algorithms to Exploit the Parallel Hardware Potential

Section A: Programming Languages and Concepts for Parallel Processing

II.A.1: Introduction

II.A.2: 'Concurrent' Programming Languages Motivations and Transformations of Sequential Programs into Parallel Programs

II.A.2.1: A Level-Detection of Parallelism

II.A.2.2: Explicit Parallelism Detection Approach

II.A.2.3: Implicit Parallelism Detection Approach

II.A.3: Programming Concepts of the Loughborough MIMD Multiprocessing Architectures

II.A.3.1: The User Interface to the 'NEPTUNE' Parallel Processor System

Section B: Aspects of Parallel Algorithms Design and Analysis Methodology

II.B.1: Construction Principles for Efficient Parallel Algorithms

II.B.2: Schemes and Techniques to Design Algorithms to Map onto SIMD and Pipelined Vector Computer Architectures

II.B.2.1: Particular Concepts and Performance Features of the 'DAP' System
A Tree-like Logical Subdivision Of Chapter III Contents

Chapter III: Fifth Generation Knowledge-Based And VLSI Chip-Embodied Information Flow Computer Systems

Section A: Computational Models And 'Knowledge Information Processing Systems' - KIPS

III.A.1: Introduction

III.A.2: Applied Schemas for Describing Parallelism in Computer Systems

 III.A.2.1: Origination and Modelling Potential of 'Petri Nets'

 III.A.2.1.1: The Structure, Modelling Properties and Execution Rules of 'Petri Nets'

 III.A.2.1.2: 'Petri Nets' Analysis Approaches, Programming Constructs Representation, and Formal Languages

 III.A.2.2: Extensions, Subclasses and Related Models to 'Petri Nets'

III.A.3: Objectives of 'Fifth Generation Computer Systems' - FGCS and Novel Decentralized Machines as their Potential Architectural Basis

 III.A.3.1: Origination, Fundamental Hardware and Software Principles, Characteristics of the 'Data Flow' Machine Architectures

 III.A.3.1.1: Prototype 'Data Flow' Machine Architectures, Programming Languages, and Further Design Alternatives

SECTION B: EMBEDDING INFORMATION FLOW SCHEMES ON GRIDS AND IN CHIP AREA AND TIME

III.B.1: The Impact of the Technological Innovation on Future Architectures - The VLSI Challenge

III.B.2: Fundamental Architectural Concepts in Designing Special-Purpose VLSI Computing Structures

III.B.2.1: The Fundamental Principle, Criteria and Advantages of 'Systolic' Architectures

III.B.2.2: A Compatibility Taxonomy in the Space of 'Systolic' Computations and VLSI Structures

A TREE-LIKE LOGICAL SUBDIVISION OF CHAPTER IV CONTENTS

CHAPTER IV: AN INVESTIGATION ON THE POTENTIAL PARALLELISM OF A NEW CLASS OF GROUP EXPLICIT METHODS FOR THE SOLUTION OF PARABOLIC PARTIAL DIFFERENTIAL EQUATIONS

SECTION A: FUNDAMENTAL CONCEPTS OF DIFFERENTIAL EQUATIONS

IV.A.1: Introductory Remarks

IV.A.2: Mathematical Preliminaries of Differential Equations

IV.A.3: Canonical Classification of Partial Differential Equations

IV.A.3.1: Boundary Conditions

IV.A.3.2: Mathematical Physics and Well-Posedness of Problems

IV.A.3.3: Analytical/Numerical Approximate Methods of Solution

SECTION B: PARALLEL EXPLOITATION OF EXPLICIT METHODS FOR THE SOLUTION OF NON-LINEAR PARABOLIC EQUATIONS

IV.B.1: Preliminary Concepts and Notations of Finite-Difference Approximations to Derivatives

IV.B.1.1: Parabolic Equations in One Space Dimension and Discretizing Finite-Difference Formulae

IV.B.1.2: A Descriptive Treatment of the Convergence, Stability, and Consistency or Compatibility Concepts
IV.B.2: Various Finite-Difference Approximation Schemes to a Non-Linear Parabolic Problem

IV.B.3: The New Class of 'Group Explicit' - GE Solution Methods

IV.B.3.1: The Standard Explicit Method: Performance Model, Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

IV.B.3.2: The 'Group Explicit with Ungrouped ends' - GEU Method: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

IV.B.3.3: The 'Group Explicit Complete' - GEC Method: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

IV.B.3.4: The 'Single) Alternating Group Explicit' - (S)AGE Method: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

IV.B.3.5: The 'Double) Alternating Group Explicit' - (D)AGE Method: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

IV.B.3.6: A 'Modified Double) Alternating Group Explicit' - (MD)AGE Method: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

IV.B.3.7: Indicative Experimental Results and Performance Measurements of the GEU Method on SIMD and Pipelined Vector Computers

IV.B.4: Relative Performance Comparisons and Conclusive Remarks on the GE Methods

Volume II

A Tree-like Logical Subdivision of Chapter V Contents

Chapter V: Implicit Parallelism Exploitation of Direct Tridiagonal Linear System Solvers

Section A: Parallel Cyclic Odd-Even Reduction Algorithms for Solving Toeplitz Tridiagonal Equations

V.A.1: Introductory Remarks
V.A.2: Fundamental Concepts and Notations of Matrix Computational Algebra

V.A.3: Classification and Merits of the Methods for Solving Linear Systems of Equations

V.A.4: The Symmetric Constant-Diagonal Case

V.A.5: The Symmetric Constant-Diagonal Periodic Case

V.A.6: Algorithmic Flowchart Representation and Inherent Parallelism Detection

V.A.7: Implementation of the Parallel Symmetric Constant-Diagonal Periodic Case: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

SECTION B: PARALLEL CYCLIC ODD-EVEN REDUCTION ALGORITHMS FOR SOLVING GENERAL-TRIDIAGONAL EQUATIONS

V.B.1: The General Non-Periodic Case: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

V.B.2: The General Periodic Case: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

V.B.3: General Comments and Conclusions

A TREE-LIKE LOGICAL SUBDIVISION OF CHAPTER VI CONTENTS

CHAPTER VI: A NEW CLASS OF 'PIPELINED' ARRAY ARCHITECTURES FOR ALGORITHM SYSTOLIZATION

SECTION A: ON ALGORITHMICALLY SPECIALIZED SYSTOLIC NETWORKS USING THE 'ROTATING' AND 'FOLDING' TECHNIQUE

VI.A.1: Introduction

VI.A.2: Classification and Principles of 'Systolic' Algorithms

VI.A.3: An Abstract Mathematical Model for the Verification of 'Systolic' Networks

VI.A.4: The Data Stream 'Rotating' and 'Folding' Technique

VI.A.5: A 'Rotating' and 'Folding' Algorithm Using a Two-Dimensional 'Systolic' Communication Geometry
VI.A.6: 'Systolic' LU-Factorization Dequeues for Tridiagonal Systems

VI.A.6.1: Dequeues for Solving Triangular Linear Systems

VI.A.6.2: General Comments: The Pivoting Problem and Orthogonal Factorization

SECTION B: Concurrent Systolization For Solving General Banded Linear Systems

VI.B.1: 'Systolic' LU-Factorization Dequeues for Quindiagonal Systems

VI.B.1.1: Modified Dequeues for the Unidirectional Factorization of the 'Central' Subsystems

VI.B.2: 'Systolic' Pipelinability 'Rotating' and 'Folding' General Banded Matrices

VI.B.3: Further Research in the 'Soft-Systolic' Area, Conclusive Remarks

A Tree-like Logical Subdivision Of Chapter VII Contents

Chapter VII: Supercomputing With Data-Driven Wavefront Array Processors

VII.1: Introductory Remarks

VII.2: A Pipelinable Two-Dimensional Computational Wavefront Concept

VII.3: On the Solution of Linear Systems Applying the Quadrant Interlocking Factorization - QIF Method

VII.4: Single Stage Computational De wavefronts for the Implementation of the QIF Algorithm

VII.5: Discussion and Further Remarks

Chapter VIII: General Comments On Future Computer Architectures, Overviewing Conclusions, And Further Research

References
<table>
<thead>
<tr>
<th>Appendix C-I</th>
<th>Further Analysis of the 'Speed-up' and 'Efficiency' Formulae in (par.-I.B.4.1)</th>
<th>906</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix C-II</td>
<td>Extended Implementation Details for (par.-II.A.3, II.B.3.1)</td>
<td>909</td>
</tr>
<tr>
<td>Appendix C-IV</td>
<td>A Selection of Optimized Parallel Computer Programs for the 'GE' Methods</td>
<td>918</td>
</tr>
<tr>
<td>Appendix C-V</td>
<td>A Selection of Optimized Parallel Computer Programs for the Tridiagonal Linear System Solvers</td>
<td>940</td>
</tr>
<tr>
<td>Appendix C-VI</td>
<td>Mathematical Background and Exemplary Numerical Examples</td>
<td>991</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.A.1-f1</td>
<td>The Increment in Computational Arithmetic Speed (a factor of 10 every 5 years).</td>
<td>30</td>
</tr>
<tr>
<td>I.A.2.1-f1</td>
<td>Flynn's SISD Organization.</td>
<td>38</td>
</tr>
<tr>
<td>I.A.2.1-f2</td>
<td>Concurrency and Instruction Processing.</td>
<td>38</td>
</tr>
<tr>
<td>I.A.2.1-f3</td>
<td>A MISD Organization.</td>
<td>39</td>
</tr>
<tr>
<td>I.A.2.2-f1</td>
<td>The Configuration of the Six Machine Classes.</td>
<td>42</td>
</tr>
<tr>
<td>I.B.2-f1</td>
<td>The SIMD Genealogy.</td>
<td>51</td>
</tr>
<tr>
<td>I.B.3.1-f1</td>
<td>The General Scheme of an Associative Processor Architecture.</td>
<td>57</td>
</tr>
<tr>
<td>I.B.3.1.1-f1</td>
<td>An Example of the Operation of an Associative Memory.</td>
<td>60</td>
</tr>
<tr>
<td>I.B.3.1.2.i-f1</td>
<td>The General Structure of a Fully Parallel, Word-Organized Associative Memory and ALU (each crosspoint represents a bit-cell of the memory).</td>
<td>62</td>
</tr>
<tr>
<td>I.B.3.1.2.ii-f1</td>
<td>The General Structure of a Fully Parallel Distributed Logic Associative Processor (as proposed by Lee).</td>
<td>63</td>
</tr>
<tr>
<td>I.B.3.1.2.iii-f1</td>
<td>The Outline of a Bit-Serial Associative Memory and the ALU.</td>
<td>65</td>
</tr>
<tr>
<td>I.B.3.1.2.iv-f1</td>
<td>The Operational Concept of a STARAN Associative Array Module with a 256-Word x 256-Bits Memory.</td>
<td>66</td>
</tr>
<tr>
<td>I.B.3.1.2.v-f1</td>
<td>The Hardware Interconnections of a Word-Serial Associative Processor.</td>
<td>68</td>
</tr>
<tr>
<td>I.B.3.2-f1</td>
<td>The Associative Memory of RAPID.</td>
<td>69</td>
</tr>
<tr>
<td>I.B.3.2-f1</td>
<td>The General Model of an Array or Parallel Computer, having an identical number of Processors and Memory Banks.</td>
<td>73</td>
</tr>
</tbody>
</table>
A typical straight storage of a Two-Dimensional \((4\times 4)\) Array into 4 Memory Units.

A Skewed storage into 4 Memory Units allowing access to Rows and Columns of a Two-Dimensional \((4\times 4)\) Array.

a) One-Dimensional; b)-f) Two-Dimensional; g)-j) Three-Dimensional Networks.

1) Single stage, \(8\times 8\) Shuffle-Exchange Network; 11) Multistage, \(8\times 8\) Benes Network; 111) Crossbar Switch Network.

The ILLIAC IV System Configuration.

The Major Hardware Units of the DAP System.

The Genealogy of Pipelined Architectures.

A Pipelined Processor System.

The Modules of a Pipelined Processor.

The Module - Time Diagram.

Modules for Floating Point Operations.

The Bottleneck is in Module 2.

Subdivision of Bottleneck.

Paralleling of Bottleneck.

A MIMD Architecture.

Indirectly or Loosely Coupled Systems.

Directly or Tightly Coupled Systems.


The Crossbar Switch Matrix Interconnection Schema.

The Multibus/Multiport Interconnection Schema.

The C.mmp Multi-min1 Processor.
A Three-Cluster Cm* Network.

A MIMD Architecture with Skeleton Processors and Centralized Computation Facilities.

The Interdata Dual Processor System Configuration.

The Interdata Dual Processor System.

The Current 'NEPTUNE' System Configuration.

The 'NEPTUNE' System.

Pictorial Representations of the Memory Allocation to Various Tasks. [Obtained utilizing the 'Show Memory Map' - SMM command].

Hierarchical Representation of a Sequentially Organized Program (Each block within a level represents a single task).

Sequential and Parallel Execution of a Task.

The FORK/JOIN Technique.

Binary Tree Representations of the Arithmetic Expression: A+B+C+D+E+F+G+H.

Tree-Height Reduction (by Distributivity) of the Arithmetic Expression: A*(B*C*D+E).

The Flowchart Structure of a Program for a MIMD Computer.

The XPFCL Command.

The XPFT Command.

The Report of a 'Non-Running' Parallel Program.

A Pipelined Integer Adder (with k=3).

The 'Perfect Shuffle' Interconnection Pattern of Eight Processors.

The Evaluation Tree of the Expression An (for n=8).
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.B.2-f4</td>
<td>The 'Inner' or 'Scalar Product' of two n-vectors ( x, y ).</td>
<td>202</td>
</tr>
<tr>
<td>II.B.2-f5</td>
<td>The 'Odd-Even Transposition Sort' on a Linear Array of Processors (The even-numbered processors have been activated first).</td>
<td>203</td>
</tr>
<tr>
<td>II.B.3-f1</td>
<td>The Evaluation Tree of the Expression An of (II.B.3:2) (in the case that ( p ) is even).</td>
<td>216</td>
</tr>
<tr>
<td>II.B.3-f2</td>
<td>The Transition Tree of the ( A_2 ) Algorithm.</td>
<td>223</td>
</tr>
<tr>
<td>III.A.2.1.1-f1</td>
<td>A Simple Graph Representation of a 'Petri-Net'.</td>
<td>255</td>
</tr>
<tr>
<td>III.A.2.1.1-f2</td>
<td>The Modelling of either Firing Order 'Concurrent' Events.</td>
<td>258</td>
</tr>
<tr>
<td>III.A.2.1.1-f3</td>
<td>The Firing of either of the Transitions ( t_i, t_j ) Disables the other (i.e. in 'Conflict').</td>
<td>258</td>
</tr>
<tr>
<td>III.A.2.1.2-f1</td>
<td>Programming Language Constructs Representation via 'Petri Nets'.</td>
<td>263</td>
</tr>
<tr>
<td>III.A.2.1.2-f2</td>
<td>'Petri Net' Modelling of Parallelism.</td>
<td>264</td>
</tr>
<tr>
<td>III.A.2.1.2-f3</td>
<td>'Petri Net' Representation of Mutual Exclusion.</td>
<td>264</td>
</tr>
<tr>
<td>III.A.3.1-f1</td>
<td>Instruction Execution Mechanism in a Data Flow Machine for the Computation of ( a=(b+1)*(b-c) ).</td>
<td>279</td>
</tr>
<tr>
<td>III.A.3.1-f2</td>
<td>Three Snapshots of the Data Flow Computation for ( a=(b+1)*(b-c) ).</td>
<td>280</td>
</tr>
<tr>
<td>III.A.3.1-f3</td>
<td>A 'Static' Data Flow Machine Organization.</td>
<td>281</td>
</tr>
<tr>
<td>III.A.3.1.1-f1</td>
<td>The Dennis 'Static' Data Flow Machine Architecture at MIT.</td>
<td>286</td>
</tr>
<tr>
<td>III.A.3.2-f1</td>
<td>A General Configuration for a 'Data Base' Machine.</td>
<td>293</td>
</tr>
<tr>
<td>III.B.1-f1</td>
<td>The Relative Evolution of the 'VLSI' Component Fields.</td>
<td>298</td>
</tr>
<tr>
<td>III.B.2-f1</td>
<td>The Design Stages of a Special-Purpose VLSI Chip.</td>
<td>305</td>
</tr>
</tbody>
</table>
The Fundamental Principle of a 'Systolic' Architecture.

'Conceptual' Cost and Performance Curves [KUNG82].

A Classification of Communication Geometry of Parallel Algorithms.

Various 'Systolic' Array Configurations.

The Area of Integration $S$ and the Boundary Curve $C$.

The Open-Ended, Area of Integration $S$ and Curve $C$.

The Finite-Difference Grid.

The Solution Regions for Parabolic Equations.

Geometrical Representation of the Finite-Difference Formulae Concept.

The Molecular Diagram of the Standard Explicit Scheme for the Direct Solution of Burgers' Equation.

The Molecular Diagram of the Fully Implicit Scheme for the Iterative Solution of Burgers' Equation.

The Molecular Diagram of the Crank-Nicolson Scheme for the Direct Solution of Burgers' Equation.

The Molecular Diagram of Saul'yev's Asymmetric Formula (IV.B.3:2).

The Molecular Diagram of Saul'yev's Asymmetric Formula (IV.B.3:4).

A Variation of the Use of Saul'yev's Asymmetric Equations.

The Molecular Diagrams of the GE Formulae for Burgers' Equation.

The Representative Diagrams of the Various GE Schemes.
Figure IV.B.3.2-f1 : The Representative Diagram of this Scheme (i.e. G.E.U.).  
Page 425

Figure IV.B.3.3-f1 : The Representative Diagram of this Scheme (i.e. G.E.C.).  
Page 434

Figure IV.B.3.4-f1 : The Representative Diagram of this Scheme (i.e. S.A.G.E.).  
Page 443

Figure IV.B.3.5-f1 : The Representative Diagram of this Scheme (i.e. D.A.G.E.).  
Page 456

Figure IV.B.3.6-f1 : The Representative Diagram of this Scheme (i.e. M.D.A.G.E.).  
Page 467

Figure IV.B.4-f1 : The Time-Complexity of the Parallel Algorithms for the GE Schemes and the Standard Explicit Method.  
Page 482

Figure IV.B.4-f2 : The Speed-ups achieved by the Parallel Algorithms for the GE Schemes and the Standard Explicit Method.  
Page 483

Figure IV.B.4-f3 : The Relative (or Normalized) Speed-ups achieved by the Parallel Algorithms for the GE Schemes.  
Page 484

Figure IV.B.4-f4 : The Reference Internal Speed-ups achieved by the Parallel Algorithms for the Unconditionally Stable (for all r>0) GE Schemes.  
Page 485

Figure IV.B.4-f5 : The Efficiency achieved by the Parallel Algorithms for the GE Schemes and the Standard Explicit Method.  
Page 486

Figure IV.B.4-f6 : The Real Cost of the Parallel Algorithms for the GE Schemes and the Standard Explicit Method.  
Page 487

Figure V.A.6-f1 : The Sequential Flowchart of the Cyclic Odd-Even Reduction Algorithm for the Symmetric Constant-Diagonal Case (see Bekakos [BEK81]).  
Page 544-545

Figure V.A.6-f2 : The Sequential Flowchart of the Cyclic Odd-Even Reduction Algorithm for the Symmetric Constant-Diagonal Periodic Case (see Bekakos [BEK81]).  
Page 546-547

Figure V.A.6-f3 : Serial Evaluation Routing of the Symmetric Constant-Diagonal Periodic Case (with reference to even rows and n=8).  
Page 548
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>V.A.6-f4</td>
<td>Parallel Evaluation Routing of the Symmetric Constant-Diagonal Periodic Case ( n=3 ).</td>
<td>556</td>
</tr>
<tr>
<td>V.A.7-f1</td>
<td>The Time-Complexity of Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.</td>
<td>576</td>
</tr>
<tr>
<td>V.A.7-f2</td>
<td>The Speed-ups achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.</td>
<td>577</td>
</tr>
<tr>
<td>V.A.7-f3</td>
<td>The Relative (or Normalised) Speed-ups achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.</td>
<td>578</td>
</tr>
<tr>
<td>V.A.7-f4</td>
<td>The Efficiency achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.</td>
<td>579</td>
</tr>
<tr>
<td>V.A.7-f5</td>
<td>The Real Cost of Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.</td>
<td>580</td>
</tr>
<tr>
<td>V.B.1-f1</td>
<td>The Time-Complexity of Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Non-Periodic Case.</td>
<td>607</td>
</tr>
<tr>
<td>V.B.1-f2</td>
<td>The Speed-ups achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Non-Periodic Case.</td>
<td>608</td>
</tr>
<tr>
<td>V.B.1-f3</td>
<td>The Efficiency achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Non-Periodic Case.</td>
<td>609</td>
</tr>
<tr>
<td>V.B.1-f4</td>
<td>The Real Cost of Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Non-Periodic Case.</td>
<td>610</td>
</tr>
<tr>
<td>V.B.2-f1</td>
<td>The Time-Complexity of Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Periodic Case.</td>
<td>630</td>
</tr>
<tr>
<td>V.B.2-f2</td>
<td>The Speed-ups achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Periodic Case.</td>
<td>631</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>V.B.2-f3</td>
<td>The Efficiency achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Periodic Case.</td>
<td>632</td>
</tr>
<tr>
<td>V.B.2-f4</td>
<td>The Real Cost of Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Periodic Case.</td>
<td>633</td>
</tr>
<tr>
<td>VI.A.4-f1</td>
<td>The Architecture of Leiserson's IPS Processor.</td>
<td>659</td>
</tr>
<tr>
<td>VI.A.4-f2</td>
<td>The Architecture of Leiserson's Modified IPSP.</td>
<td>662</td>
</tr>
<tr>
<td>VI.A.4-f3</td>
<td>The Computational Steps of the Matrix-Vector Multiplication Algorithm (n=5) Using a 'Dequeue'.</td>
<td>663</td>
</tr>
<tr>
<td>VI.A.5-f1</td>
<td>The Outline of the IPS Cell in the Hexagonal Geometry.</td>
<td>665</td>
</tr>
<tr>
<td>VI.A.5-f2</td>
<td>The Dequeues of Data for the Matrix Multiplication Problem on a Hexagonal Systolic Array (for p₁=q₁=p₂=q₂=2, and n=5).</td>
<td>668</td>
</tr>
<tr>
<td>VI.A.5-f3</td>
<td>The Dequeues of Data for the Matrix Multiplication Problem on a Hexagonal Systolic Array (for p₁=2, q₁=3, p₂=3, q₂=2, and n=5).</td>
<td>670</td>
</tr>
<tr>
<td>VI.A.5-f4</td>
<td>Four Consecutive Computational Steps of the Matrix Multiplication Problem of Paradigm [VI.A.5:n₂].</td>
<td>671-672</td>
</tr>
<tr>
<td>VI.A.6-f1</td>
<td>Hexagonal Array of Processors for Pipelining the LU-decomposition of a (n×n)-Band Matrix with Bandwidth w=7.</td>
<td>675</td>
</tr>
<tr>
<td>VI.A.6-f2</td>
<td>The Dequeue of Data for the LU-factorization on a Hexagonal Systolic Array (for p=q=2, and n=9).</td>
<td>683</td>
</tr>
<tr>
<td>VI.A.6-f3</td>
<td>All the Computational Steps of the LU-factorization of Paradigm [VI.A.6:n₁].</td>
<td>684-687</td>
</tr>
<tr>
<td>VI.A.6-f4</td>
<td>The Dequeue of Data for the LU-factorization on a Hexagonal Systolic Array (for p=q=2, and n=4).</td>
<td>691</td>
</tr>
<tr>
<td>VI.A.6-f5</td>
<td>All the Computational Steps of the LU-factorization of Paradigm [VI.A.6:n₂].</td>
<td>692-694</td>
</tr>
</tbody>
</table>
VI.A.6.1-f1: The Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.A.6:π₁] on the Linearly Connected Systolic Array (for \( w=q=2 \)).

VI.A.6.1-f2: All the Computational Steps for the Solution of the Lower Triangular Linear System of Paradigm [VI.A.6:π₁].

VI.A.6.1-f3: The Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm [VI.A.6:π₂] on the Linearly Connected Systolic Array (for \( w=q=2 \)).

VI.A.6.1-f4: All the Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.A.6:π₁].

VI.A.6.1-f5: The Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.A.6:π₂] on the Linearly Connected Systolic Array (for \( w=q=2 \)).

VI.A.6.1-f6: All the Computational Steps for the Solution of the Lower Triangular Linear System of Paradigm [VI.A.6:π₂].

VI.A.6.1-f7: The Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm [VI.A.6:π₂] on the Linearly Connected Systolic Array (for \( w=q=2 \)).

VI.A.6.1-f8: All the Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.A.6:π₂].

VI.B.1-f1: The Dequeue of Data for the LU-factorization on a Hexagonal Systolic Array (for \( p=q=3 \), and \( n=5 \)).

VI.B.1-f2: Six Consecutive Computational Steps of the LU-factorization of a Quindiagonal Matrix (for \( n=5 \)).

VI.B.1-f3: The Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1:π₁] on the Linearly Connected Systolic Array (for \( w=q=3 \)).

VI.B.1-f4: Twelve Consecutive Computational Steps for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1:π₁].
The Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1:π₁] on the Linearly Connected Systolic Array (for w=q=3).

Six Consecutive Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1:π₁].

The Modified Dequeue of Data for the LU-factorization on a Hexagonal Systolic Array (for p=q=3, and n=5).

Eight Consecutive Computational Steps of the Unidirectional (for the Central Submatrix) LU-factorization of a Quindiagonal Matrix (for n=5).

The Modified Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1:π₁] on the Linearly Connected Systolic Array (for w=q=3).

Six Consecutive Computational Steps for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1:π₁] Using a Modified Dequeue of Data.

The Modified Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1:π₁] on the Linearly Connected Systolic Array (for w=q=3).

Six Consecutive Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1:π₁] Using a Modified Dequeue of Data.

The Modified Dequeue of Data for the LU-factorization on a Hexagonal Systolic Array (for p=q=3, and n=4).

Six Consecutive Computational Steps of the Unidirectional (for the Central Submatrix) LU-factorization of a Quindiagonal Matrix (for n=4).

The Modified Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1:π₁] on the Linearly Connected Systolic Array (for w=q=3).
Consecutive Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1.1:] Using a Modified Dequeue of Data.

The Modified Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1.1:] on the Linearly Connected Systolic Array (for w=q=3).

Six Consecutive Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1.1:] Using a Modified Dequeue of Data.

The Relativity of the Various Cases for a (n×n) Banded Matrix of Semi-bandwidth p.

Definition of the Binary Cell.

The Computational Steps of the Matrix-Vector Multiplication Algorithm (n=5) Using a 'Quadqueue'.

The Configuration for a (n×n)-Square Wavefront Array Processor (WAP).

Hardware Configuration for a (2×2) Linear System Solver Using Cramer's Rule.

The Propagation of Two-Dimensional Computational 'Dewavefronts'.

The Data Streams for the Modification Part of the Factorization Phase (for n=5).

The Data Streams for the Modification Part of the Forward Solution Phase (for n=5).

The Data Streams for the Modification Part of the Backward Solution Phase (for n=5).

The Overall Hardware Configuration for the Implementation of the QIF Algorithm.
### Table of Contents

<table>
<thead>
<tr>
<th>Table ID</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.B.1-t1</td>
<td>Typical Problems Solved on SIND Computers.</td>
<td>191</td>
</tr>
<tr>
<td>II.B.3.1-t1</td>
<td>Resource Provisions of the 'NEPTUNE' System.</td>
<td>237</td>
</tr>
<tr>
<td>III.B.2.2-t1</td>
<td>The Potential Utilization of 'Systolic' Array Configurations.</td>
<td>314</td>
</tr>
<tr>
<td>IV.B.3.1-t1</td>
<td>List of Parameters for the Performance Model.</td>
<td>394-400</td>
</tr>
<tr>
<td>IV.B.3.1-t2</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the Standard Explicit Method on the 'NEPTUNE' Prototype System.</td>
<td>403-404</td>
</tr>
<tr>
<td>IV.B.3.1-t3</td>
<td>A Program Dependent Performance Analysis of the Parallel Algorithm for the Standard Explicit Method.</td>
<td>408</td>
</tr>
<tr>
<td>IV.B.3.1-t4</td>
<td>A System Dependent Performance Analysis of the Parallel Algorithm for the Standard Explicit Method.</td>
<td>408</td>
</tr>
<tr>
<td>IV.B.3.2-t1</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the G.E.U. Method on the 'NEPTUNE' Prototype System.</td>
<td>429-430</td>
</tr>
<tr>
<td>IV.B.3.2-t2</td>
<td>A Program Dependent Performance Analysis of the Parallel Algorithm for the G.E.U. Method.</td>
<td>432</td>
</tr>
<tr>
<td>IV.B.3.2-t3</td>
<td>A System Dependent Performance Analysis of the Parallel Algorithm for the G.E.U. Method.</td>
<td>432</td>
</tr>
<tr>
<td>IV.B.3.3-t1</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the G.E.C. Method on the 'NEPTUNE' Prototype System.</td>
<td>437-438</td>
</tr>
<tr>
<td>IV.B.3.3-t2</td>
<td>A Program Dependent Performance Analysis of the Parallel Algorithm for the G.E.C. Method.</td>
<td>440</td>
</tr>
<tr>
<td>IV.B.3.3-t3</td>
<td>A System Dependent Performance Analysis of the Parallel Algorithm for the G.E.C. Method.</td>
<td>440</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>IV.B.3.4-t1</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the (S).A.G.E. Method (for r=1) on the 'NEPTUNE' Prototype System.</td>
<td>445-446</td>
</tr>
<tr>
<td>IV.B.3.4-t2</td>
<td>A Program Dependent Performance Analysis of the Parallel Algorithm for the (S).A.G.E. Method (for r=1).</td>
<td>450</td>
</tr>
<tr>
<td>IV.B.3.4-t3</td>
<td>A System Dependent Performance Analysis of the Parallel Algorithm for the (S).A.G.E. Method (for r=1).</td>
<td>450</td>
</tr>
<tr>
<td>IV.B.3.4-t4</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the (S).A.G.E. Method (for r=2) on the 'NEPTUNE' Prototype System.</td>
<td>451-452</td>
</tr>
<tr>
<td>IV.B.3.4-t5</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the (S).A.G.E. Method (for r=4) on the 'NEPTUNE' Prototype System.</td>
<td>453</td>
</tr>
<tr>
<td>IV.B.3.4-t6</td>
<td>A Program Dependent Performance Analysis of the Parallel Algorithm for the (S).A.G.E. Method (for r=4).</td>
<td>454</td>
</tr>
<tr>
<td>IV.B.3.4-t7</td>
<td>A System Dependent Performance Analysis of the Parallel Algorithm for the (S).A.G.E. Method (for r=4).</td>
<td>454</td>
</tr>
<tr>
<td>IV.B.3.5-t1</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the (D).A.G.E. Method (for r=1) on the 'NEPTUNE' Prototype System.</td>
<td>458-459</td>
</tr>
<tr>
<td>IV.B.3.5-t2</td>
<td>A Program Dependent Performance Analysis of the Parallel Algorithm for the (D).A.G.E. Method (for r=1).</td>
<td>461</td>
</tr>
<tr>
<td>IV.B.3.5-t3</td>
<td>A System Dependent Performance Analysis of the Parallel Algorithm for the (D).A.G.E. Method (for r=1).</td>
<td>461</td>
</tr>
<tr>
<td>IV.B.3.5-t4</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the (D).A.G.E. Method (for r=2) on the 'NEPTUNE' Prototype System.</td>
<td>462-463</td>
</tr>
<tr>
<td>IV.B.3.5-t5</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the (D).A.G.E. Method (for r=4) on the 'NEPTUNE' Prototype System.</td>
<td>464</td>
</tr>
</tbody>
</table>
### Table

<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV.B.3.5-t6</td>
<td>A Program Dependent Performance Analysis of the Parallel Algorithm for the (D).A.G.E. Method (for r=4).</td>
<td>465</td>
</tr>
<tr>
<td>IV.B.3.5-t7</td>
<td>A System Dependent Performance Analysis of the Parallel Algorithm for the (D).A.G.E. Method (for r=4).</td>
<td>465</td>
</tr>
<tr>
<td>IV.B.3.6-t1</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the (M.D).A.G.E. Method (for r=1) on the 'NEPTUNE' Prototype System.</td>
<td>469-470</td>
</tr>
<tr>
<td>IV.B.3.6-t2</td>
<td>A Program Dependent Performance Analysis of the Parallel Algorithm for the (M.D).A.G.E. Method (for r=1).</td>
<td>472</td>
</tr>
<tr>
<td>IV.B.3.6-t3</td>
<td>A System Dependent Performance Analysis of the Parallel Algorithm for the (M.D).A.G.E. Method (for r=1).</td>
<td>472</td>
</tr>
<tr>
<td>IV.B.3.6-t4</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the (M.D).A.G.E. Method (for r=2) on the 'NEPTUNE' Prototype System.</td>
<td>473-474</td>
</tr>
<tr>
<td>IV.B.3.6-t5</td>
<td>Experimental Results and Performance Measurements of the Parallel Algorithm for the (M.D).A.G.E. Method (for r=4) on the 'NEPTUNE' Prototype System.</td>
<td>475</td>
</tr>
<tr>
<td>IV.B.3.6-t6</td>
<td>A Program Dependent Performance Analysis of the Parallel Algorithm for the (M.D).A.G.E. Method (for r=4).</td>
<td>476</td>
</tr>
<tr>
<td>IV.B.3.6-t7</td>
<td>A System Dependent Performance Analysis of the Parallel Algorithm for the (M.D).A.G.E. Method (for r=4).</td>
<td>476</td>
</tr>
<tr>
<td>IV.B.3.7-t1</td>
<td>Experimental Results on the 'DAP' and 'CRAY-IS' Systems.</td>
<td>480</td>
</tr>
<tr>
<td>IV.B.4-t1</td>
<td>The Algebraic-Complexity of the GE Schemes in Comparison with the Standard Explicit and Crank-Nicolson Methods.</td>
<td>488</td>
</tr>
<tr>
<td>V.A.2-t1</td>
<td>The Definitions of Special Types of Band Matrices.</td>
<td>522</td>
</tr>
<tr>
<td>V.A.2-t2</td>
<td>Some Special Properties of Square Matrices.</td>
<td>523</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>V.A.7-t1</td>
<td>Experimental Results and Performance Measurements, for the Symmetric</td>
<td>567-570</td>
</tr>
<tr>
<td></td>
<td>Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reduction Method on the 'NEPTUNE' Prototype System, for Granularity Factors</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of Various Sizes.</td>
<td></td>
</tr>
<tr>
<td>V.A.7-t2</td>
<td>Experimental Results and Performance Measurements, for the Symmetric</td>
<td>571-572</td>
</tr>
<tr>
<td></td>
<td>Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reduction Method on the 'NEPTUNE' Prototype System, for a Granularity Factor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of Size (2×2).</td>
<td></td>
</tr>
<tr>
<td>V.A.7-t3</td>
<td>List of Local Parameters for the Performance Model.</td>
<td>575</td>
</tr>
<tr>
<td>V.A.7-t4</td>
<td>Program Dependent Performance Analyses, for the Symmetric Constant-Diagonal</td>
<td>582-583</td>
</tr>
<tr>
<td></td>
<td>Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.</td>
<td></td>
</tr>
<tr>
<td>V.A.7-t5</td>
<td>System Dependent Performance Analyses, for the Symmetric Constant-Diagonal</td>
<td>584</td>
</tr>
<tr>
<td></td>
<td>Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.</td>
<td></td>
</tr>
<tr>
<td>V.B.1-t1</td>
<td>Experimental Results and Performance Measurements, for the General Non-</td>
<td>601-603</td>
</tr>
<tr>
<td></td>
<td>Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.</td>
<td></td>
</tr>
<tr>
<td>V.B.1-t2</td>
<td>Program Dependent Performance Analyses, for the General Non-Periodic Case,</td>
<td>612-613</td>
</tr>
<tr>
<td></td>
<td>of Parallel Variants of the Cyclic Odd-Even Reduction Method.</td>
<td></td>
</tr>
<tr>
<td>V.B.1-t3</td>
<td>System Dependent Performance Analyses, for the General Non-Periodic Case,</td>
<td>614</td>
</tr>
<tr>
<td></td>
<td>of Parallel Variants of the Cyclic Odd-Even Reduction Method.</td>
<td></td>
</tr>
<tr>
<td>V.B.2-t1</td>
<td>Experimental Results and Performance Measurements, for the General Periodic</td>
<td>625-627</td>
</tr>
<tr>
<td></td>
<td>Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>V.B.2-t2</td>
<td>Program Dependent Performance Analyses, for the General Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.</td>
<td>634-635</td>
</tr>
<tr>
<td>V.B.2-t3</td>
<td>System Dependent Performance Analyses, for the General Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.</td>
<td>636</td>
</tr>
</tbody>
</table>
CHAPTER V: IMPLICIT PARALLELISM EXPLOITATION OF DIRECT TRIDIAGONAL LINEAR SYSTEM SOLVERS

SECTION A: PARALLEL CYCLIC ODD-EVEN REDUCTION ALGORITHMS FOR SOLVING TOEPLITZ TRIDIAGONAL EQUATIONS

V.A.1: Introductory Remarks
V.A.2: Fundamental Concepts and Notations of Matrix Computational Algebra
V.A.3: Classification and Merits of the Methods for Solving Linear Systems of Equations
V.A.4: The Symmetric Constant-Diagonal Case
V.A.5: The Symmetric Constant-Diagonal Periodic Case
V.A.6: Algorithmic Flowchart Representation and Inherent Parallelism Detection
V.A.7: Implementation of the Parallel Symmetric Constant-Diagonal Periodic Case: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System

SECTION B: PARALLEL CYCLIC ODD-EVEN REDUCTION ALGORITHMS FOR SOLVING GENERAL-TRIDIAGONAL EQUATIONS

V.B.1: The General Non-Periodic Case: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System
V.B.2: The General Periodic Case: Experimental Results and Performance Analysis on the 'NEPTUNE' Prototype System
V.B.3: General Comments and Conclusions
CHAPTER V

IMPLICIT PARALLELISM EXPLOITATION
OF DIRECT TRIDIAGONAL LINEAR SYSTEM SOLVERS
PARALLEL CYCLIC
ODD-EVEN REDUCTION ALGORITHMS
FOR SOLVING
TOEPLITZ TRIDIAGONAL EQUATIONS
V.A.1: **INTRODUCTORY REMARKS**

In this *Chapter* a thorough performance exposure and exploitation of the 'NEPTUNE' (MIMD) prototype system is carried out, by presenting a selection of algorithms which, utilizing different parallel strategies, or constructs available on the system, implement the cyclic odd-even reduction method and search for the optimal values of the granularity factor in terms of the size of the final sequentially solved subsystems.

In the present *Section A* we initiate the reader to the necessary fundamental concepts and notations of matrix computational Algebra and to the various methods for solving linear systems of equations.

Then, the cyclic odd-even reduction method, which possesses advantages for transforming sequential computations into highly parallel ones, is exemplified for particular problems involving coefficient matrices of Toeplitz type, i.e., symmetric constant-tridiagonal and symmetric constant-tridiagonal periodic matrices.

The sequential algorithmic flowcharts of the aforementioned technique, for both cases, are given, while its inherent potential parallelism is detected through a numerical example for the latter type of matrices above.
Finally, this Section concludes with the analytical evaluation of the algebraic-complexity of the serial and parallel versions of the method and the actual experimentation and performance analyses of its various implemented parallel invariants, in comparison with the most efficient sequential algorithm of Gaussian elimination. These performance analyses are dependent on the detailed performance model established in the previous Chapter. They are, again, substantiated by an analysis of the system's resources provided and the resources demanded by the parallel algorithms. In particular, all the selected parallel variants of the method are analyzed in a theoretical (i.e., program dependent) and experimental (i.e., system dependent) manner.

The experimental vehicle for this implementation was chosen to be the symmetric constant-diagonal periodic case, since it is more complicated and its concept indirectly includes that of the corresponding non-periodic case.

In Section B we use the same method for solving general tridiagonal equations with coefficient matrices of non-periodic and periodic type. The analytical evaluations of the algebraic-complexity of the serial and parallel versions of the method, the experimental results and the performance analyses of its parallel variants for each of these cases, on the NEPTUNE prototype system, are similarly as before presented, in comparison with a generalized version of Gaussian elimination.

The Chapter concludes with general comments and conclusions concerning various algorithmic aspects, in relation with future computer architectural designs. For further investigation, emphasis has been given on the implementation of the parallel version of the method on VLSI processor arrays utilizing a tree-structure of processing elements.
V.A.2: FUNDAMENTAL CONCEPTS AND NOTATIONS OF MATRIX COMPUTATIONAL ALGEBRA

This Chapter starts by reviewing some preliminaries of matrix algebra, and since we have already introduced matrix computations in the previous Chapter, its primary purpose is to serve as a brief, but formal, glossary of some particular and most commonly used concepts and notations.

Let us begin with the symbol $\mathbb{R}^{m \times n}$ which denotes the vector space of all the $m$-by-$n$ order real matrices, i.e.,

$$A \in \mathbb{R}^{m \times n} \Rightarrow A = [a_{ij}],$$

in fact, the capital letters are used to denote a matrix $\dagger$, while the corresponding lower case letters with the subscript $ij$ imply the $(i,j)$ component of the matrix, where $i$ refers to the rows and $j$ to the columns of the matrix.

The basic computational manipulations with matrices can be found in any relevant literature.

The $n$-by-$n$ matrices are said to be square. In specific, the $n$-by-$n$ identity matrix is denoted by $I_n$ and its $k^{th}$ column by $e_k^{(n)}$, i.e.,

$$I_n = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}, \quad e_k^{(n)} = (0, \ldots, 0, 1, 0, \ldots, 0)^T.$$

In the case that the dimension is clear from context, then we simply write $I$ and $e_k$, respectively.

$\dagger$Enclosing the general matrix element, e.g. $a_{ij}$, in square brackets is another way of representing a matrix, as shown above.

$\ddagger$One of the basic manipulations of matrices is 'Transposition' ($\mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{n \times m}$), i.e., $C = A^T, \quad c_{ij} = a_{ji}$. 
If $A$ and $B$ in $\mathbb{R}^{n \times n}$ satisfy $AB = I$, then $B$ is the inverse of $A$ and is denoted by $A^{-1}$. If $A^{-1}$ exists, then $A$ is said to be non-singular; otherwise, $A$ is singular.

If $A = (a) \in \mathbb{R}^{I \times I}$, then its determinant is given by $\det(A) = a$.

For $A \in \mathbb{R}^{n \times n}$, we have

$$\det(A) = \sum_{j=1}^{n} (-1)^{j+1} a_{1j} \det(A_{1j}) ,$$

where $A_{1j}$ is an $(n-1)$-by-$(n-1)$ matrix obtained by deleting the first row and $j^{th}$ column of $A$. In accordance, for

$$A \in \mathbb{R}^{n \times n}, \det(A) \neq 0 \Rightarrow A \text{ is non-singular.}$$

A set of vectors $\{a_1, \ldots, a_n\}$ in $\mathbb{R}^m$ is linearly independent if

$$\sum_{j=1}^{n} a_j a_j = 0 \Rightarrow a_1 = \ldots = a_n = 0;$$

otherwise, a non-trivial combination of $a_1, \ldots, a_n$ is zero and $\{a_1, \ldots, a_n\}$ is said to be linearly dependent.

A subspace of $\mathbb{R}^m$ is a subset that is also a vector space. The set of all linear combinations of $a_1, \ldots, a_n \in \mathbb{R}^m$ is a subspace referred to as the span of $\{a_1, \ldots, a_n\}$, i.e.,

$$\text{span}\{a_1, \ldots, a_n\} = \{ \sum_{j} \beta_j a_j \mid \beta_1, \ldots, \beta_n \in \mathbb{R} \}.$$ 

A subset $\{a_{i1}, \ldots, a_{ik}\}$ is a maximal linearly independent subset of $\{a_1, \ldots, a_n\}$ if $\{a_{i1}, \ldots, a_{ik}\}$ is linearly independent and is not properly contained in any linearly independent subset of $\{a_1, \ldots, a_n\}$. If $\{a_{i1}, \ldots, a_{ik}\}$ is maximal, then,

$$\text{span}\{a_1, \ldots, a_n\} = \text{span}\{a_{i1}, \ldots, a_{ik}\}$$

and $\{a_{i1}, \ldots, a_{ik}\}$ is a basis for $\text{span}\{a_1, \ldots, a_n\}$. If $S \subseteq \mathbb{R}^m$ is a subspace, then there exist independent basic vectors $a_1, \ldots, a_k$ in $S$. 
such that

\[ S = \text{span}\{a_1, \ldots, a_k\}. \]

All the bases for a subspace \( S \) have the same number of elements, which is called the dimension of \( S \) and is denoted by \( \dim(S) \).

In particular, there are two important subspaces associated with a matrix \( A \) in \( \mathbb{R}^{m \times n} \). The range of \( A \), which is defined by

\[ R(A) = \{y \in \mathbb{R}^m | y = Ax \text{ for some } x \in \mathbb{R}^n\}, \]

and the null space of \( A \) by

\[ N(A) = \{x \in \mathbb{R}^n | Ax = 0\}. \]

If \( A = [a_1, \ldots, a_n] \) then

\[ R(A) = \text{span}\{a_1, \ldots, a_n\}. \]

The rank of a matrix \( A \) is defined by

\[ \text{rank}(A) = \dim(R(A)). \]

It can be shown that \( \text{rank}(A) = \text{rank}(A^T) \), and thus, the rank of a matrix equals the maximal number of independent rows or columns.

For any \( A \in \mathbb{R}^{m \times n} \), \( \dim(N(A)) + \text{rank}(A) = n \). If \( m=n \), then the following are equivalent:

1) \( A \) is non-singular
2) \( N(A) = \{0\} \)
3) \( \text{rank}(A) = n \).

We shall now consider instances of matrices with special patterns of zero entries, also with special symmetries and properties. Let us begin by classifying matrices according to their zero/non-zero structure. We say that \( A \in \mathbb{R}^{m \times n} \) has lower bandwidth \( r \) and upper bandwidth \( s \) if \( a_{ij} = 0 \) whenever \( i > j+r \) and \( j > i+s \). In the case of \( r=s \) then \( A \) is simply said to have bandwidth \( r \). The most frequently
occurring special classes of real band matrices are shown in Table (V.A.2-t1), while analogous definitions hold for lower bidiagonal, lower triangular, strictly lower triangular, and lower Hessenberg matrices.

If most of the elements of a matrix are zero, then it is said to be a sparse matrix; in fact, banded matrices in $\mathbb{R}^{m \times n}$ whose bandwidths are much smaller than $m$ and $n$ are sparse. However, if most of the matrix elements are non-zero, then the matrix is said to be a dense matrix.

In accordance with some special properties occurring, the class of real square matrices can be further distinguished into several important types as shown in Table (V.A.2-t2). In particular, for $A \in \mathbb{R}^{m \times n}$ and if $X \in \mathbb{R}^{n \times n}$ is non-singular, then we say that $A$ and $X^TAX$ are congruent. We must point out that the properties of symmetry, skew-symmetry, and definiteness are preserved under congruence transformations (see Golub and Van Loan [GOLU83]).

<table>
<thead>
<tr>
<th>MATRIX</th>
<th>DEFINITION [$A \in \mathbb{R}^{m \times n}$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal</td>
<td>If $a_{ij} = 0$ whenever $i \neq j$</td>
</tr>
<tr>
<td>Tridiagonal</td>
<td>If $a_{ij} = 0$ whenever $</td>
</tr>
<tr>
<td>Upper bidiagonal</td>
<td>If $a_{ij} = 0$ whenever $1 &gt; j$ or $j &gt; i + 1$</td>
</tr>
<tr>
<td>Upper triangular</td>
<td>If $a_{ij} = 0$ whenever $1 &gt; j$</td>
</tr>
<tr>
<td>Strictly upper triangular</td>
<td>If $a_{ij} = 0$ whenever $1 &gt; j$</td>
</tr>
<tr>
<td>Upper Hessenberg</td>
<td>If $a_{ij} = 0$ whenever $1 &gt; j$</td>
</tr>
</tbody>
</table>

Table V.A.2-t1: The Definitions of Special Types of Band Matrices.

*Just apply these definitions to $A^T$. [522]
<table>
<thead>
<tr>
<th>MATRIX PROPERTY</th>
<th>DEFINITION ((A \in \mathbb{R}^{n \times n}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric</td>
<td>If (A^T = A)</td>
</tr>
<tr>
<td>Skew-Symmetric</td>
<td>If (A^T = -A)</td>
</tr>
<tr>
<td>Positive definite</td>
<td>If (x^T Ax &gt; 0), (0 \neq x \in \mathbb{R}^n)</td>
</tr>
<tr>
<td>Non-negative definite</td>
<td>If (x^T Ax \geq 0), (x \in \mathbb{R}^n)</td>
</tr>
<tr>
<td>Indefinite</td>
<td>If ((x^T Ax)(y^T Ay) &lt; 0), for some (x, y \in \mathbb{R}^n)</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>If (A^T A = I_n)</td>
</tr>
<tr>
<td>Nilpotent</td>
<td>If (A^k = 0), for some positive integer (k)</td>
</tr>
<tr>
<td>Idempotent</td>
<td>If (A^2 = A)</td>
</tr>
<tr>
<td>Positive</td>
<td>If (a_{ij} &gt; 0), for all (i) and (j)</td>
</tr>
<tr>
<td>Non-negative</td>
<td>If (a_{ij} \geq 0), for all (i) and (j)</td>
</tr>
<tr>
<td>Diagonally dominant</td>
<td>If (</td>
</tr>
<tr>
<td>Strictly diagonally dominant</td>
<td>If (</td>
</tr>
<tr>
<td>Permutation</td>
<td>If (A = [e_{s_1}, \ldots, e_{s_n}]), where ((s_1, \ldots, s_n)) is a permutation of ((1, 2, \ldots, n))</td>
</tr>
</tbody>
</table>

Table V.A.2-2: Some Special Properties of Square Matrices.

Many of the properties described above can be extended to block matrices (i.e., matrices whose elements are again matrices) and complex matrices whose \(m\)-by-\(n\) set is being denoted by \(\mathbb{C}^{m \times n}\) and by \(\mathbb{C}^n\) the set of complex \(n\)-vectors. In particular, for \(\{m_1, \ldots, m_p\}\) and \(\{n_1, \ldots, n_q\}\) sets of positive integers, if

\[
A_{ij} \in \mathbb{R}^{m_i \times n_j}, \quad i=1, \ldots, p, \quad j=1, \ldots, q
\]

then
A = \begin{bmatrix}
A_{11} & A_{12} & \cdots & A_{1q} \\
A_{21} & A_{22} & \cdots & A_{2q} \\
\vdots & \vdots & \ddots & \vdots \\
A_{p1} & A_{p2} & \cdots & A_{pq}
\end{bmatrix}

is a p-by-q block matrix and \(A_{ij}\) is referred to as the \((i,j)\) block.

On the other hand, if \(A \in \mathbb{C}^{m \times n}\), then its conjugate transpose \(A^H\) is defined by

\[ A^H = (a_{ij})^\ast. \]

The inner product of \(x\) and \(y\) in \(\mathbb{C}^n\) thus has the form

\[ x^H y = \sum_{i=1}^{n} x_i \bar{y}_i = y^H x. \]

Furthermore, an \(A \in \mathbb{C}^{n \times n}\) is said to be

i) unitary if \(A^H A = I_n\)

ii) Hermitian if \(A = A^H\)

iii) positive definite if \(x^H A x > 0\) for all non-zero \(x \in \mathbb{C}^n\).

We shall conclude with the concepts related with the eigenvalues of a matrix \(A \in \mathbb{C}^{n \times n}\) which are the \(n\) roots of its characteristic polynomial

\[ p(z) = \det(zI - A). \]

The set of these roots is called the spectrum and is denoted by \(\lambda(A)\).

If \(\lambda(A) = \{\lambda_1, \ldots, \lambda_n\}\), then it follows that

\[ \det(A) = \prod_{i=1}^{n} \lambda_i. \]

Moreover, if we define the trace of \(A\) by

\[ \text{trace}(A) = \sum_{i=1}^{n} a_{ii}, \quad A \in \mathbb{C}^{n \times n}, \]

\[ \lambda(A) = \{\lambda_1, \ldots, \lambda_n\}, \quad A \in \mathbb{C}^{n \times n}, \]

The eigenvalues and eigenvectors are otherwise called 'latent' roots and vectors, respectively.
then $\text{trace}(A) = \sum_{1=1}^{n} \lambda_i$.

If $\lambda \in \lambda(A)$, then the non-zero vectors $x \in \mathbb{R}^n$ that satisfy $Ax = \lambda x$
are referred to as eigenvectors. This eigenproblem may be written as $(\lambda I - A)x = 0$
which is a system of $n$ homogeneous linear equations. This system has non-trivial solutions if and only if its matrix is singular, i.e., $\det(\lambda I - A) = 0$.

The spectral radius of matrix $A$ is defined as $\rho(A) = \max_{1 \leq i \leq n} |\lambda_i|$.

Finally, in accordance with the eigenvalues concept, a real matrix is positive definite if and only if it is symmetric and all its eigenvalues are strictly positive.

†The eigenvalues and eigenvectors are otherwise called 'latent' roots and vectors, respectively.
V.A.3: **Classification and Merits of the Methods for Solving Linear Systems of Equations**

As we have seen in the previous Chapter a finite-difference method leads to a system of algebraic simultaneous equations, which in the case of linear boundary-value problems are always linear. However, the size of the resulting system is generally large and, for this reason, its efficient solution is a major problem in itself.

These problems can be expressed in matrix-vector notation by the equation

\[ Ax = b , \]  

where \( A \) is an \((n \times n)\) matrix of the coefficients, \( b \) is a known \( n \)-vector and \( x \) is an unknown \( n \)-vector whose value is to be found. The reader should note the different vector notation \( \vec{x} \) that has been introduced in Chapter II. Provided that \( \det(A) \) is non-zero, the unique solution of the equation is expressed simply as

\[ x = A^{-1}b , \]

where \( A^{-1} \), as we have previously seen, is the inverse of matrix \( A \).

In numerical practice, however, the computation of this inverse is preferably avoided, since more efficient ways of solving the problem are available. On the other hand, the occurring matrices in real-time, generally, fall into one of the two categories of matrices, i.e. dense, but not large, and sparse and perhaps very large matrices; and, it is for the latter type, to be specific, it is unlikely (except in very special cases) that the sparsity is preserved in the inverse computation.

Different solution methods are usually used for these two categories classed as direct and indirect or iterative methods.‡

†The most common notation for a \( n \)-by-\( n \) matrix.
‡Some Algebra books also advocate Cramer’s rule, but it is not indicated for large \( n \).
The options that someone has of trying to find a solution to equation (V.A.3:1), in general, are either to change the matrix \( A \) (most suitable for dense matrices via the direct methods), or simply to leave \( A \) unaltered (most suitable for sparse matrices via the iterative methods). In particular for the dense matrices, when stored in full, then \( n^2 \) locations are required and any zero elements are likely to be changed to non-zero, a fact which could be disastrous for the latter category of matrices. In that case, a matrix is likely to be generated and only the non-zero elements need to be stored.

Direct Methods are principally based on elimination\(^\dagger\) techniques and the amount of work involved is fixed and known beforehand. Furthermore, the solution process is done just once and the only errors arising in the solution are only the round-off errors introduced in the computation.

A common direct (factorization) method (a variant of the well known Gaussian elimination method) for the solution of equation (V.A.3:1) requires the decomposition of matrix \( A \) into lower and upper triangular matrices \( L,U \), respectively, of the same order as \( A \), with the \( U \) matrix having 1's on its diagonal. This method\(^\ddagger\) is known as the triangular-decomposition method, or \( LU \)-decomposition method; also named as Crout reduction, or, after another discoverer, the Cholesky method (see Bekakoş [BEKA81]) and it is feasible only if the matrix \( A \) is non-singular. Hence, equation (V.A.3:1) can then be replaced by

\[
LUX = b \quad \text{(V.A.3:2)}
\]

\(^\dagger\)We consider the 'factorization techniques' indistinguishably related with the elimination procedure (see Johnson and Riess [JOHN??], pp.34-35).

\(^\ddagger\)An equivalent method transforms \( A \) into an \( L,U \) pair in which the \( L \) matrix has 1's on its diagonal. This is called 'Doolittle's method'.
or by introducing an auxiliary vector $z$ (say) the system of equations (V.A.3:2) can be formulated as

\[
Lz = b \quad (V.A.3:3)
\]
\[
UX = z \quad (V.A.3:4)
\]

Then, the original set of equations (V.A.3:1) is solved in two stages, by solving (V.A.3:3) for $z$, followed by (V.A.3:4) for $x$, called \textit{forward elimination} and \textit{back-substitution} procedures, respectively. However, for this proposition to be viable, (V.A.3:3) and (V.A.3:4) must be easily solved.

As opposed to the direct methods for solving a set of linear equations, which are the premier and exclusive scope of this Chapter, when the coefficient matrix is sparse, then \textit{Iterative methods} are much more preferred due to the fact that they provide more rapid solutions; but in certain cases only, since the occurrence of the zeros may follow some easy pattern from which the elimination methods can take advantage.

Each iterative method is designed to generate a sequence of vectors, \textit{(iterates)}, \{$x^{(k)}$\}_{k=0}^{\infty}, which hopefully converge to a value close to the true solution, $x_t$, of the system (V.A.3:1). The iterative procedure is said to be \textit{convergent} when the difference between the exact solution and the successive approximations tends to zero, as the number of iterations increases.

The basic idea of iterative methods can be described as follows:

1) The matrix $A$ is written as the difference of two matrices $N$ and $P$, so that

\[
A = N - P \quad (V.A.3:5)
\]

This decomposition of $A$ is called a \textit{splitting}. 
ii) An initial guess \( x^{(0)} \) is made for the solution vector \( x \).

iii) A sequence \( x^{(1)}, x^{(2)}, x^{(3)}, \ldots \), of estimates to \( x \) is generated by the formula
\[
Nx^{(k+1)} = Px^{(k)} + b, \quad k = 0, 1, 2, \ldots .
\] (V.A.3.6)

The first thing to observe is that solving system (V.A.3.1) is equivalent to solving the system
\[
Nx = Px + b .
\] (V.A.3.7)

In order to get some idea of what might constitute a good choice \( N \) and \( P \), we consider formula (V.A.3.6). This formula says that if we have \( x^{(k)} \), then we can get the next iterate \( x^{(k+1)} \) provided we can solve the linear system
\[
Nx^{(k+1)} = h^{(k)},
\] (V.A.3.8)

where the vector \( h^{(k)} \) is given by
\[
h^{(k)} = Px^{(k)} + b .
\] (V.A.3.9)

Thus, it is clear that we must require \( N \) to be non-singular in order to be assured that we can implement the iteration. Furthermore, for an iterative procedure to be efficient, \( N \) should be chosen so that (V.A.3.8) is quite easy to solve; this is the case if, for instance, \( N \) is chosen to be a triangular or a diagonal matrix.

However, the total amount of work involved is not known, as the calculations continue indefinitely until the answers have converged to sufficient accuracy. In fact, the process may not even converge and therefore it is important to know of any conditions under which an iterative procedure can be guaranteed to converge (see Varga [VARG62]).

The first and simplest iterative method is the Jacobi method. For the purposes of briefly discussing this method it is convenient
to think of the matrix \( P \) as the sum of lower and upper triangular matrices. To be specific, let \( L, D, \) and \( U \) to be the lower triangular, diagonal, and upper triangular parts of the \((n \times n)\) matrix \( A \). Thus,

\[
A = L + D + U, \quad (V.A.3:10)
\]

and so the Jacobi splitting is given by

\[
N = D, \quad P = -(L+U). \quad (V.A.3:11)
\]

The **Jacobi method** for solving the system \((V.A.3:1)\) is given by

\[
Dx^{(k+1)} = -(L+U)x^{(k)} + b, \quad (V.A.3:12)
\]

while the matrix

\[
M_{ij} = -D^{-1}(L+U) \quad (V.A.3:13)
\]

is called the **Jacobi iterative matrix**.

In actual computation, equation \((V.A.3:12)\) would have to be written out element-wise. Suppose the vector \( x^{(k)} \) is given by

\[
x^{(k)} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix}, \quad k=0,1,2,\ldots \quad (V.A.3:14)
\]

Then, equation \((V.A.3:12)\) leads to the following iteration for the \( i^{th} \) component of \( x^{(k)} \):

\[
x_i^{(k+1)} = \frac{1}{a_{ii}} \left[ \sum_{j=1, j \neq i}^{n} a_{ij} x_j^{(k)} \right] - b_i, \quad i=1,2,\ldots,n, \quad (V.A.3:15)
\]

which shows that the Jacobi iteration is quite easy to program. The only real problem is to determine an efficient test for terminating the iteration. However, in order for the Jacobi method to be used, the diagonal elements of \( A \) must all be non-zero, but this requirement
causes no real difficulty.

There are many other, much faster, iterative methods besides Jacobi, the most important being, the Gauss-Seidel method, the Successive Over-Relaxation (SOR) method and the Alternating-Direction Implicit (ADI) method, for which a good advanced reference is Varga [VARG62].

Finally, arranged in tabular form, we shall summarize the merits of iterative methods compared with elimination methods:

**Advantages**

1. Probably more efficient for large order systems
2. Implementation is simpler
3. Advantage can be taken of a known approximate solution, if one exists
4. Low accuracy solutions can be obtained quickly
5. Where the equations have a repetitive form, their coefficients need not be stored but can be generated, and for sparse matrices only
6. Less storage space required for an iterative solution
7. The storage requirement is more easily defined in advance
8. The order of specification of the variables is not, usually, important.

**Disadvantages**

1. Additional right hand sides are not easily processed
2. The convergence, even if assured, may be slow and so the amount of work is not predictable
3. The time and accuracy of the result depends on a judicious choice of parameters
4. If the convergence rate is poor, the results must be interpreted with caution.

5. No advantage in time per iteration can be gained if the coefficient matrix is symmetric. For elimination the time can be halved.

All iterative methods are similar to \( \mathbf{x}^{(k+1)} = A \mathbf{x}^{(k)} + \mathbf{b} \), so that if \( A \) is a \((n \times n)\) full matrix, the number of multiplications is \( -n^2 \) per iteration. Since this number for elimination methods is \( -n^3/3 \), an iterative method for solving \((V.A.3:1)\) is likely to be viable if the number of iterations is \( <n/3 \) (or \( <n/6 \) for symmetric \( A \)).
V.A.4: **THE SYMMETRIC CONSTANT-DIAGONAL CASE**

The principal aim of the present Chapter is to investigate the so termed *implicit parallelism* of a special technique known as *Cyclic Odd-Even Reduction*\(^1\). This form of parallelism involves the discovery of independent sub-expressions in the computation capable of proceeding in parallel. The aforementioned technique possesses advantages for transforming a sequential calculation into a highly parallel one, and appears to be one of the best techniques for the *symmetric constant-tridiagonal case*.

To be more specific, we assume a set of \( n \) linear equations of the form

\[
A\mathbf{x} = \mathbf{y},
\]

where \( A \) is a tridiagonal matrix such as,

\[
A = \begin{bmatrix}
   b & a & & & & \\
   a & b & a & & & \\
   & a & b & a & & \\
   & & & \ddots & \ddots & \\
   & & & & a & b \\
   & & & & & \ddots & a
\end{bmatrix}
\]

(V.A.4:2)

which, in shorthand notation, we denote as the \((...,a,b,a,...)\) case.

As a parenthesis, matrices whose entries are constant along each diagonal arise in many applications and are called *Toeplitz matrices*. Formally, \( T \in \mathbb{R}^{n \times n} \) is Toeplitz if there exist scalars \( r_{-n+1}, ..., r_0, ..., r_{n-1} \) such that \( t_{ij} = r_{|i-j|} \), for all \( i \) and \( j \). Toeplitz matrices belong to the larger class of *persymmetric matrices*. We say that \( B \in \mathbb{R}^{n \times n} \) is persymmetric if it is symmetric about its northeast-southwest

\(^1\)Cyclic reduction was first used to solve tridiagonal equations by Hockney (1965) in collaboration with Golub (see Hockney and Jesshope \( \text{[HOCK81]} \)).

\(^\dagger\)A Toeplitz matrix is completely specified by its first row and column.
diagonal, i.e. if \( b_{ij} = b_{n-j+1-n-i+1} \) for all \( i \) and \( j \).

The definition of Toeplitz matrices extends, in a corresponding manner, to complex square matrices as well.

Finally, a particular, and most frequently occurring, subclass of Toeplitz matrices is the class of circulant matrices which are defined by the further property that \( t_{_i} = t_{n-j} \) for \( i = 1, 2, \ldots, n-1 \). A circulant matrix is completely specified by its first row; each further row may be obtained from the previous one by a right cyclic shift.

On the other hand, the absolute performance behaviour of Multi-processor testbeds will be revealed by implementing and analyzing, according to the deterministic modelling approach established in the previous Chapter, several parallel algorithms using the above solution technique in combination with most of the available parallel constructs for the experimental system in hand, i.e., the NEPTUNE prototype system.

This technique is applicable to the same class of problems, as applied to the Recursive Doubling process.

In the following discussion, we assume that \( n \) can take any of the following values, i.e., \( 2^{m-1}, 2^m, 2^{m+1} \), where \( m \) is any positive integer (see Bekakos [BEKA81]).

The general procedure of the cyclic odd-even reduction algorithm is as follows; let us consider three adjacent rows of \( A \), i.e.,

\[
\begin{array}{ccc}
i-1 & a & b & a \\
i & a & b & a \\
i+1 & a & b & a \\
\end{array}
\]  

\( (V.A.4:3) \)

A multiple of the middle row \( i \) is added to the summation of the \( i-1, i+1 \) rows, thus obtaining the form \((\ldots, a', 0, b', 0, a', \ldots)\). This operation creates a tridiagonal system consisting of \( 2^{m-1}-1 \), or \( 2^{m-1} \), or \( 2^{m-1}+1 \)
only, *even* or *odd* rows of the original matrix $A$.

For example, if we consider that, originally, the number of equations is: $n=2^m-1$, and the new system consists of the *even* rows of the original matrix $A$, $2^{m-1}-1$ the number, then although the *odd* rows have been eliminated, the *odd* unknowns can be computed from the *even* unknowns by a back-substitution process.

Repeating the process to the new system of $2^{m-1}-1$ equations, involving just the *even* unknowns, we can eliminate every other row, thus obtaining a set of $2^{m-2}-1$ equations involving unknowns with subscripts which are multiples of 4. We can repeat this process until we obtain a single equation for $x_{2^{m-1}}$, which can then be easily solved. Then, by back-substituting, we can compute the eliminated unknowns in the reverse order in which they were eliminated.

The *cyclic odd-even reduction algorithm* reaches a significant speed increase when the 'a' coefficients of matrix $A$ are initially equal to unity, i.e., the shorthand notation now has the form $(\ldots,1,b,1,\ldots)$, since they remain equal to unity throughout the computation, thus reducing the number of additions and multiplications per iteration.

The *symmetric constant-tridiagonal* system, given in (V.A.4:1), can be normalized in the $(\ldots,1,b,1,\ldots)$ form, dividing by 'a', to normalize 'b' and 'y', to produce the *symmetric constant-diagonal form*. This can be performed simultaneously, in most of the parallel computers, since the diagonal elements are a constant number.

More specifically, to describe the algorithm, let us consider the case that $n=2^m-1$, where $m=3$, i.e., the original matrix $A$ consists of 7 equations. In this case the matrix equation given in (V.A.4:1), after normalization, has the form,
When the procedure described previously is applied, e.g., for the even rows, we multiply equations $2, 4, 6$ by $-b^2$, adding the two adjacent rows to each of them. Then, the system (V.A.4:4) becomes,

\[
\begin{bmatrix}
1 & b & 1 \\
2 & 0 & b^2 & 0 & 1 \\
3 & 1 & b & 1 \\
4 & 1 & 0 & b^2 & 0 & 1 \\
5 & 1 & b & 1 \\
6 & 0 & 1 & b^2 & 0 \\
7 & 1 & b \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
\end{bmatrix}
= 
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5 \\
y_6 \\
y_7 \\
\end{bmatrix}
\]

(V.A.4:4)

where,

\[ b^{[2]} = 2 - b^2 \]  

(V.A.4:6)

\[ y_i^{[2]} = y_{i-1} - b_i y_{i+1}, \] for $i = 2, 4, 6$.  

(V.A.4:7)

Since the even rows $2, 4, 6$ in the system (V.A.4:5) are independent of the odd rows, they may be separated as follows,

\[
\begin{bmatrix}
1 & b^2 & 1 & 0 \\
2 & 1 & b^2 & 1 \\
3 & 0 & 1 & b^2 \\
\end{bmatrix}
\begin{bmatrix}
x_2 \\
x_4 \\
x_6 \\
\end{bmatrix}
= 
\begin{bmatrix}
y_2^{[2]} \\
y_4^{[2]} \\
y_6^{[2]} \\
\end{bmatrix}
\]

(V.A.4:8)
Applying the above process, once more, to the system \((V.A.4:8)\), i.e., multiplying the second row of the system \((V.A.4:8)\) by \(-b^{[2]}\), and adding to it the first and third rows, the system \((V.A.4:8)\) becomes,

\[
eq \begin{array}{ccc|ccc}
1(2) & b^{[2]} & 1 & 0 & x_2 & y_2^{[2]} \\
2(4) & 0 & b^{[3]} & 0 & x_4 & y_4^{[3]} \\
3(8) & 0 & 1 & b^{[2]} & x_6 & y_6^{[2]}
\end{array}
\]

where

\[
b^{[3]} = 2 - (b^{[2]})^2 \tag{V.A.4:10}
\]

\[
y_4^{[3]} = y_2^{[2]} - b^{[2]}y_4^{[2]} + y_6^{[2]} \tag{V.A.4:11}
\]

From the system \((V.A.4:9)\), by separating the second row, we obtain

\[
b^{[3]}x_4 = y_4^{[3]} \tag{V.A.4:12}
\]

which can be easily solved, thus finding \(x_4\).

By a process of back-substitution, and in terms of \(x_4\), the first and third rows of the system \((V.A.4:9)\) may be written as,

\[
b^{[2]}x_2 = y_2^{[2]} - x_4 \tag{V.A.4:13}
\]

\[
b^{[2]}x_6 = y_6^{[2]} - x_4 \tag{V.A.4:14}
\]

Therefore, we can easily calculate \(x_2\) and \(x_6\).

By continuing the back-substitution process, in the same way, to the system \((V.A.4:5)\), we can calculate \(x_1, x_3, x_5, x_7\) in terms of \(x_2, x_4, x_6\), thus finding the solution of the matrix equation \((V.A.4:4)\).

To conclude, the above described process can be applied for any of the values of \(n\), (i.e. \(2^m - 1, 2^m, 2^m + 1\), where \(m\) is any positive integer), choosing each time the even or odd rows of the system to work with.
V.A.5: The Symmetric Constant-Diagonal Periodic Case

Herein, we shall again examine the previously described, efficient for parallelism exploitation, technique, but now operating on a set of \( n \) linear equations of the form

\[
Ax = y, 
\]

where the matrix \( A \) has the representation

\[
A = \begin{bmatrix}
  b & a & a \\
  a & b & a \\
  a & b & a \\
  a & 0 & 0 \\
  0 & a & b \\
  0 & 0 & a
\end{bmatrix},
\]

which, in shorthand notation, we denote as the \((a, ..., a, b, a, ..., a)\) case.

In the discussion which follows, we assume, for greatest efficiency, that \( n = 2^m \), where \( m \) is any positive integer. Although the general procedure for solving the system \((V.A.5:1)\), applying the cyclic odd-even reduction technique, is similar to that exhibited previously, the first and last equations are special cases because of the periodic nature of the problem. These cases will be explained later in this paragraph, during the solution procedure of the given example.

Let us, again, consider three adjacent rows of \( A \), i.e.,

\[
i-1 \quad a \quad b \quad a \\
i \quad a \quad b \quad a \\
i+1 \quad a \quad b \quad a
\]

\((V.A.5:3)\)

The aim is to obtain a form like \((a, ..., a', 0, b', 0, a', ..., a)\), by adding rows \( i+1, i-1 \), to a multiple of the middle \( i \) row, and this for every
three adjacent rows of \( A \). The resulting tridiagonal system consists of \( 2^{m-1} \) only, even or odd rows of the original matrix \( A \), retaining at the same time the periodicity of the original system.

We can repeat this process that many times until we obtain a single equation for \( x_{2^m} \) or \( x_{2^{m-1}+1} \), if we are applying the method by choosing the even or odd rows, respectively.

Finally, the eliminated unknowns can be computed by a back-substitution process, exactly in the reverse order to the elimination process.

Again, similarly to the simple case, because of the significant speed increase obtained when the 'a' coefficients are initially equal to unity (since that way the number of operations is reduced), we can normalize the system (V.A.5:1) to the \((1, \ldots, 1, b, 1, \ldots, 1)\) form, dividing by 'a', to normalize 'b' and 'y', to produce the symmetric constant-diagonal periodic form.

For a clearer insight of the algorithm, in the symmetric constant-diagonal periodic case, let us consider the instance that \( n=8 \), i.e., the original matrix \( A \) consists of 8 equations. Thus, the system (V.A.5:1) becomes,

\[
\begin{align*}
\text{eq.} & & 1 & \begin{array}{ccc} b & 1 & 1 \end{array} & \begin{array}{c} x_1 \\ y_1 \end{array} \\
1 & \begin{array}{ccc} 1 & b & 1 \end{array} & \begin{array}{c} x_2 \\ y_2 \end{array} \\
2 & \begin{array}{ccc} 1 & b & 1 \end{array} & \begin{array}{c} x_3 \\ y_3 \end{array} \\
3 & \begin{array}{ccc} 1 & b & 1 \end{array} & \begin{array}{c} x_4 \\ y_4 \end{array} \\
4 & \begin{array}{ccc} 1 & b & 1 \end{array} & \begin{array}{c} x_5 \\ y_5 \end{array} \\
5 & \begin{array}{ccc} 1 & b & 1 \end{array} & \begin{array}{c} x_6 \\ y_6 \end{array} \\
6 & \begin{array}{ccc} 1 & b & 1 \end{array} & \begin{array}{c} x_7 \\ y_7 \end{array} \\
7 & \begin{array}{ccc} 1 & b & 1 \end{array} & \begin{array}{c} x_8 \\ y_8 \end{array} \\
8 & \begin{array}{ccc} 1 & b & 1 \end{array} & \begin{array}{c} x_8 \\ y_8 \end{array} \\
\end{align*}
\]
In a similar manner as before, having chosen the even rows to work with when applying the method, we multiply rows 2, 4, 6, 8 by \(-b\) and add the two adjacent rows to each of them to obtain the following system,

\[
\begin{align*}
1 & : b & 1 & 1 \quad x_1 & \quad y_1 \\
2 & : 0 & b^{[2]} & 0 & 1 \quad x_2 & \quad y_2 \\
3 & : 1 & b & 1 \quad x_3 & \quad y_3 \\
4 & : 1 & 0 & b^{[2]} & 0 & 1 \quad x_4 & \quad y_4 \\
5 & : 1 & b & 1 \quad x_5 & = y_5 \\
6 & : 0 & b^{[2]} & 0 & 1 \quad x_6 & \quad y_6 \\
7 & : 0 & 1 & 1 \quad x_7 & \quad y_7 \\
8 & : 0 & 1 & 0 & b^{[2]} & 1 \quad x_8 & \quad y_8 \\
\end{align*}
\]

where \(b^{[2]} = 2b^2\) \quad (V.A.5:6)

\[
y_i^{[2]} = y_{i-1} - by_i + y_{i+1}, \text{ for } i=2,4,6,8. \quad (V.A.5:7)
\]

Noticeably, different from the simple case is that every time for the last row of the system, due to the periodicity of the problem, the next row is taken to be the first row of the system (i.e. \(n+1 \mod n = 1\)).

Because of the independency between the even and odd rows, in the system (V.A.5:5) the even rows may be separated as follows,

\[
\begin{align*}
1(2) & : b^{[2]} & 1 & 0 & 1 \quad x_2 & \quad y_2 \\
2(4) & : 1 & b^{[2]} & 1 & 0 \quad x_4 & \quad y_4 \\
3(6) & : 0 & 1 & b^{[2]} & 1 \quad x_6 & = y_6 \\
4(8) & : 1 & 0 & 1 & b^{[2]} \quad x_8 & \quad y_8 \\
\end{align*}
\]

Again, applying the above process, this time multiplying by \(-b^{[2]}\), the rows 2(4), 4(8), we obtain,
\[
\begin{align*}
1(2) & \quad \begin{bmatrix} b[2] & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ y_4 \end{bmatrix}, \\
2(4) & \quad \begin{bmatrix} 0 & b[3] & 0 & 2 \end{bmatrix} \begin{bmatrix} x_4 \\ y_4 \end{bmatrix} = \begin{bmatrix} y_4 \\ y_6 \end{bmatrix}, \\
3(6) & \quad \begin{bmatrix} 0 & 1 & b[2] & 1 \end{bmatrix} \begin{bmatrix} x_6 \\ y_6 \end{bmatrix} = \begin{bmatrix} y_6 \\ y_8 \end{bmatrix}, \\
4(8) & \quad \begin{bmatrix} 0 & 2 & 0 & b[3] \end{bmatrix} \begin{bmatrix} x_8 \\ y_8 \end{bmatrix}
\end{align*}
\]

\[\text{where,} \quad b[3] = 2 - (b[2])^2 \]
\[\text{and} \quad y_i[3] = y_{i-2}[2] - b[2]y_{i+1}[2] + y_i[2], \quad \text{for } i=4,8.\]

Separating again the even rows, in the system \((V.A.5:9)\), we obtain,
\[
\begin{align*}
1(4) & \quad \begin{bmatrix} b[3] & 2 \end{bmatrix} \begin{bmatrix} x_4 \\ y_4 \end{bmatrix} = \begin{bmatrix} y_4 \\ y_8 \end{bmatrix}, \\
2(8) & \quad \begin{bmatrix} 0 & b[3] \end{bmatrix} \begin{bmatrix} x_8 \\ y_8 \end{bmatrix}
\end{align*}
\]

A final application of the process, multiplying by \(-b[3]\), row \(2(8)\), produces a system of the form,
\[
\begin{align*}
1(4) & \quad \begin{bmatrix} b[3] & 2 \end{bmatrix} \begin{bmatrix} x_4 \\ y_4 \end{bmatrix} = \begin{bmatrix} y_4 \\ y_8 \end{bmatrix}, \\
2(8) & \quad \begin{bmatrix} 0 & b[4] \end{bmatrix} \begin{bmatrix} x_8 \\ y_8 \end{bmatrix}
\end{align*}
\]

where the two adjacent rows added to the second row coincide with the first row of the system \((V.A.5:13)\).

This time, exceptionally,
\[b[4] = 4 - (b[3])^2\]

From the system \((V.A.5:13)\) we can easily calculate \(x_8\), and then by back-substituting in systems \((V.A.5:13)\), \((V.A.5:9)\), \((V.A.5:5)\) we obtain the values for \((x_4, x_2, x_6), (x_1, x_3, x_5, x_7)\), respectively, in terms of the previously found values for the \(x_i\)'s.
To conclude, in the case that we choose the odd rows to work with, applying the cyclic odd-even reduction algorithm, then the special case occurs in the $i^{st}$ row of the original system (V.A.5:4) and every subsequent system resulting during the process, the last row, each time, being considered as the top adjacent row to it.
V.A.6: ALGORITHMIC FLOWCHART REPRESENTATION AND INHERENT PARALLELISM DETECTION

In an attempt to identify sections of parallelism in a serial algorithm a pragmatic approach would seem reasonable: The serial algorithms are analyzed for patterns of frequently occurring basic elements which are then arranged into a parallel formation.

In our case, although both of the presented algorithms are very efficient as serial algorithms, however, because they have been constructed following the 'serial way of thinking', indirectly transforming them into parallel ones would result in algorithms which are not the optimum from the efficiency point of view.

For this pure sequential implementation of both algorithmic cases, and the stated values for $n$, the flowcharts are illustrated in Figures (V.A.6-f1,f2) respectively, and were coded and run on the VAX 750 computer, at Loughborough University (see Bekakos [BEKA81]).

As we have discussed in previous Chapters, the more natural and efficient approach to constructing a parallel algorithm is to follow the 'parallel way of thinking' directly from the problem itself. In the following discussion we shall exemplify this approach considering the symmetric constant-diagonal periodic case to work with, since it is more broad in concept covering the non-periodic case.

In order to illustrate diagrammatically the serial algorithm described in (par.-V.A.5) we present the scheme depicted in Figure (V.A.6-f3). Therein, we have considered the even rows to work with, and eventually we end up with a single equation having the $x_8$ as unknown, which can be easily solved. Then, by back-substitution we can evaluate the remaining unknowns in both, even and odd rows.
read \ r, n, m, b

INITIALIZATION
\[ x(m+1) = 0.0 \]
\[ b(i) = 0.0, \text{ for } i = 1, 3m/2 \]
\[ w(1) = r, \quad z(1) = r \]
\[ t = 1 \]

\[ m = 2^{*n} \]

\[ m = (2^{*n}) - 1 \]

\[ z(t+1) = 2 - w(t) * z(t) \]

\[ t = 1, \ n - 1 \]

\[ w(t+1) = 2 - (w(t)^2) \]

\[ m = (2^{*n}) - 1 \]

\[ z(t+2) = 1 - w(t+1) * z(t+1) \]

\[ j = 2^{*t}, m, 2^{*t} \]

\[ b(j) = b(j - (2^{*(t-1)})) - w(t) * b(j) + b(j + (2^{*(t-1)})) \]

A
Figure V.A.6-f1: The Sequential Flowchart of the Cyclic Odd-Even Reduction Algorithm for the Symmetric Constant- Diagonal Case (see Bekakos [BEKA81]).
read r, n, m, b

INITIALIZATION
\[ w(1) = r \]

\[ t = 1, n-1 \]

\[ w(t+1) = 2 - (w(t)^2) \]

\[ j = 2^t, m, 2^t \]

\[ j = 2^t, m, 2^t \]

\[ b(j) = b(j-2^t) - w(t)\times b(j) + b(2^t) \]

\[ w(n+1) = 4 - (w(n)^2) \]

\[ b(2^n) = b(2^n) - w(n) + 2^t b(2^n-1) \]

\[ x(2^n) = b(2^n) / w(n) \]

i = 1, n-1

\[ t = n - i \]

\[ j = (2^t + 2^{t+1}), m, 2^t \]

\[ x(j) = (b(j) - x(j-2^t) - x(j+2^t)) / w(t+1) \]

\[ k = 2^t \]

\[ k = 2^{n-1} \]

\[ x(k) = (b(k) - x(2^t+k) - x(2^n)) / w(t+1) \]
Figure V.A.6-f2: The Sequential Flowchart of the Cyclic Odd-Even Reduction Algorithm for the Symmetric Constant-Diagonal Periodic Case (see Bekkos [BEKA81]).
Figure V A.5 - f_3  Serial Evaluation Routing of the Symmetric Constant - Diagonal Periodic Case (with reference to 'even' rows and 'n' = 8)
In a similar manner, we could have chosen the odd rows to work with instead, thus eliminating the even rows, and finally, again, by back-substituting to obtain similar results.

However, by approaching this problem in a parallel manner, since either process, i.e., the even or odd rows, develops independently of the other, we could consider both choices applying the described algorithm for each of them, simultaneously. Let us now consider an example applying this concept and to conclude we shall present diagrammatically the process of the new evaluation.

We shall make use of the same system (for \( n = 8 \)) as we had in (par.-V.A.6) i.e.,

\[
\begin{align*}
1 & \quad \begin{bmatrix} b & 1 & 1 \\ 2 & 1 & b & 1 \\ 3 & 1 & b & 1 \\ 4 & 1 & b & 1 \\ 5 & 1 & b & 1 \\ 6 & 1 & b & 1 \\ 7 & 1 & b & 1 \\ 8 & 1 & b & 1 \\ \end{bmatrix} \\
& \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ \end{bmatrix}.
\end{align*}
\]

(V.A.6:1)

At first we choose the even rows to apply the cyclic odd-even reduction algorithm, i.e., multiply the rows 2, 4, 6, 8, by \(-b\) and add the two adjacent rows to each of them. Thus, we obtain the following system,
where  
\[ b^{[2]} = 2 - b^2 \quad (V.A.6:3) \]
\[ y_1^{[2]} = y_{i-1} - b_i y_{i+1}, \text{ for } i = 2, 4, 6, 8 \quad (V.A.6:4) \]

where the next row to the 8th row is considered as the 1st row of the system due to the periodicity of the problem.

If, on the other hand, we again apply the cyclic odd-even reduction algorithm to the original system, but this time considering the odd rows to work with, we obtain the following system,

\[ b^{[2]} = 2 - b^2 \quad (V.A.6:5) \]
\[ b^{[2]} = 2 - b^2 \]  \hspace{1cm} (V.A.6:6)

\[ y_1^{[2]} = y_{i-1} - by_i + y_{i+1} \]  \hspace{1cm} for i=1,3,5,7 \hspace{1cm} (V.A.6:7)

and as the top adjacent row for the \( i^{th} \) row is considered the \( 8^{th} \) row of the system.

Thus, separating the even from the odd rows and vice-versa, in the systems \((V.A.6:2),(V.A.6:5)\), respectively, we obtain two systems of the form,

\[
\begin{align*}
1(2) & \quad \begin{bmatrix} b^{[2]} & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \end{bmatrix} = \begin{bmatrix} y_2^{[2]} \end{bmatrix} \\
2(4) & \quad \begin{bmatrix} 1 & b^{[2]} & 1 & 0 \end{bmatrix} \begin{bmatrix} x_4 \end{bmatrix} = \begin{bmatrix} y_4^{[2]} \end{bmatrix} \\
3(6) & \quad \begin{bmatrix} 0 & 1 & b^{[2]} & 1 \end{bmatrix} \begin{bmatrix} x_6 \end{bmatrix} = \begin{bmatrix} y_6^{[2]} \end{bmatrix} \\
4(8) & \quad \begin{bmatrix} 1 & 0 & 1 & b^{[2]} \end{bmatrix} \begin{bmatrix} x_8 \end{bmatrix} = \begin{bmatrix} y_8^{[2]} \end{bmatrix}
\end{align*}
\]

and

\[
\begin{align*}
1(1) & \quad \begin{bmatrix} b^{[2]} & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \end{bmatrix} = \begin{bmatrix} y_1^{[2]} \end{bmatrix} \\
2(3) & \quad \begin{bmatrix} 1 & b^{[2]} & 1 & 0 \end{bmatrix} \begin{bmatrix} x_3 \end{bmatrix} = \begin{bmatrix} y_3^{[2]} \end{bmatrix} \\
3(5) & \quad \begin{bmatrix} 0 & 1 & b^{[2]} & 1 \end{bmatrix} \begin{bmatrix} x_5 \end{bmatrix} = \begin{bmatrix} y_5^{[2]} \end{bmatrix} \\
4(7) & \quad \begin{bmatrix} 1 & 0 & 1 & b^{[2]} \end{bmatrix} \begin{bmatrix} x_7 \end{bmatrix} = \begin{bmatrix} y_7^{[2]} \end{bmatrix}
\end{align*}
\]

We can now apply the cyclic odd-even reduction algorithm again, independently to each of the systems \((V.A.6:8),(V.A.6:9)\), multiplying by \(-b^{[2]}\), the appropriate rows each time, to obtain two (2x2) subsystems from each system, one from the even case and one from the odd case.

In specific, from the system \((V.A.6:8)\), we obtain,

\[
\begin{align*}
1(4) & \quad \begin{bmatrix} b^{[3]} & 2 \end{bmatrix} \begin{bmatrix} x_4 \end{bmatrix} = \begin{bmatrix} y_4^{[3]} \end{bmatrix} \quad \text{(even case)} \hspace{1cm} (V.A.6:10) \\
2(8) & \quad \begin{bmatrix} 2 & b^{[3]} \end{bmatrix} \begin{bmatrix} x_8 \end{bmatrix} = \begin{bmatrix} y_8^{[3]} \end{bmatrix}
\end{align*}
\]
where \[ b^{[3]} = 2 - (b^{[2]})^2 \] \[ (V.A.6:11) \]
\[ y_i^{[3]} = y_{i-2}^{[2]} - b y_i^{[2]} + y_{i+2}^{[2]} \], for \( i = 4, 8 \) \[ (V.A.6:18) \]

and

\[
\begin{align*}
1(2) & \begin{bmatrix} b^{[3]} & 2 \\ 2 & b^{[3]} \end{bmatrix} \begin{bmatrix} x_2 \\ x_6 \end{bmatrix} = \begin{bmatrix} y_2^{[3]} \\ y_6^{[3]} \end{bmatrix} \quad \text{(odd case),} \\
2(6) & \begin{bmatrix} 2 & b^{[3]} \\ b^{[3]} & 2 \end{bmatrix} \begin{bmatrix} x_7 \\ x_6 \end{bmatrix} = \begin{bmatrix} y_7^{[3]} \\ y_6^{[3]} \end{bmatrix} \quad \text{(even case),} \\
\end{align*}
\]

where \[ b^{[3]} = 2 - (b^{[2]})^2 \] \[ (V.A.6:14) \]
\[ y_i^{[3]} = y_{i-2}^{[2]} - b y_i^{[2]} + y_{i+2}^{[2]} \], for \( i = 2, 6 \) . \[ (V.A.6:15) \]

On the other hand, from the system \((V.A.6:9)\), we obtain,

\[
\begin{align*}
1(3) & \begin{bmatrix} b^{[3]} & 2 \\ 2 & b^{[3]} \end{bmatrix} \begin{bmatrix} x_3 \\ x_7 \end{bmatrix} = \begin{bmatrix} y_3^{[3]} \\ y_7^{[3]} \end{bmatrix} \quad \text{(even case),} \\
2(7) & \begin{bmatrix} 2 & b^{[3]} \\ b^{[3]} & 2 \end{bmatrix} \begin{bmatrix} x_3 \\ x_7 \end{bmatrix} = \begin{bmatrix} y_3^{[3]} \\ y_7^{[3]} \end{bmatrix} \quad \text{(odd case),} \\
\end{align*}
\]

where \[ b^{[3]} = 2 - (b^{[2]})^2 \] \[ (V.A.6:17) \]
\[ y_i^{[3]} = y_{i-2}^{[2]} - b y_i^{[2]} + y_{i+2}^{[2]} \], for \( i = 3, 7 \) \[ (V.A.6:18) \]

and

\[
\begin{align*}
1(1) & \begin{bmatrix} b^{[3]} & 2 \\ 2 & b^{[3]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_1^{[3]} \\ y_5^{[3]} \end{bmatrix} \quad \text{(odd case),} \\
2(5) & \begin{bmatrix} 2 & b^{[3]} \\ b^{[3]} & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_1^{[3]} \\ y_5^{[3]} \end{bmatrix} \quad \text{(even case),} \\
\end{align*}
\]

where \[ b^{[3]} = 2 - (b^{[2]})^2 \] \[ (V.A.6:20) \]
\[ y_i^{[3]} = y_{i-2}^{[2]} - b y_i^{[2]} + y_{i+2}^{[2]} \], for \( i = 1, 5 \). \[ (V.A.6:21) \]

Again, the periodicity of the problem is used, so we always have two adjacent rows for each considered row of the systems.

The four \((2 \times 2)\) systems obtained can be easily solved either by applying, once more, the above technique (i.e., considering for each
(2x2) system once the even and once the odd row, separately, to be multiplied by \(-b[^3]\), or by any other method (e.g., Cramer's rule, etc.).

In order to retain as much parallelism as possible throughout the reduction phase, and for further exemplification purposes, let us follow the same technique as above. Thus, from the system (V.A.6:10) we obtain,

\[
\begin{align*}
\text{eq.} & \\
\begin{bmatrix} 1(4) & 2 \\ 2(8) & 0 \end{bmatrix} \begin{bmatrix} b[^3] \\ b[^4] \end{bmatrix} & = \begin{bmatrix} y[^3] \\ y[^4] \end{bmatrix} \quad \text{(even case), } \text{(V.A.6:22)} \\
\end{align*}
\]

and

\[
\begin{align*}
\text{eq.} & \\
\begin{bmatrix} 1(4) & 2 \\ 2(8) & b[^3] \end{bmatrix} & = \begin{bmatrix} y[^4] \\ y[^3] \end{bmatrix} \quad \text{(odd case), } \text{(V.A.6:23)} \\
\end{align*}
\]

where

\[
\begin{align*}
b[^4] & = 4 - (b[^3])^2 \quad \text{(V.A.6:24)} \\
\end{align*}
\]

thus evaluating simultaneously \(x^8\) and \(x^4\) from systems (V.A.6:22), (V.A.6:23), respectively; from the system (V.A.6:13) we obtain,

\[
\begin{align*}
\text{eq.} & \\
\begin{bmatrix} 1(2) & 2 \\ 2(6) & 0 \end{bmatrix} \begin{bmatrix} b[^3] \\ b[^4] \end{bmatrix} & = \begin{bmatrix} y[^3] \\ y[^4] \end{bmatrix} \quad \text{(even case), } \text{(V.A.6:27)} \\
\end{align*}
\]

and

\[
\begin{align*}
\text{eq.} & \\
\begin{bmatrix} 1(8) & 2 \\ 2(6) & b[^3] \end{bmatrix} & = \begin{bmatrix} y[^4] \\ y[^3] \end{bmatrix} \quad \text{(odd case), } \text{(V.A.6:28)} \\
\end{align*}
\]

where
\[ b[4] = 4 - (b[3])^2 \]

Thus evaluating simultaneously \( x_6 \) and \( x_2 \) from systems (V.A.6:27), (V.A.6:28), respectively; from the system (V.A.6:16) we obtain,

\[
\begin{align*}
\text{eq.} & \\
1(3) & \begin{bmatrix} b[3] & 0 \\ 0 & b[4] \end{bmatrix} \begin{bmatrix} x_3 \\ x_7 \end{bmatrix} = \begin{bmatrix} y_3[3] \\ y_7[3] \end{bmatrix} \quad \text{(even case), (V.A.6:32)}
\end{align*}
\]

and

\[
\begin{align*}
\text{eq.} & \\
1(3) & \begin{bmatrix} b[4] & 2 \\ 0 & b[3] \end{bmatrix} \begin{bmatrix} x_3 \\ x_7 \end{bmatrix} = \begin{bmatrix} y_3[4] \\ y_7[3] \end{bmatrix} \quad \text{(odd case), (V.A.6:33)}
\end{align*}
\]

where

\[ b[4] = 4 - (b[3])^2 \]

Thus evaluating simultaneously \( x_7 \) and \( x_3 \) from systems (V.A.6:32), (V.A.6:33), respectively; and, finally, from the system (V.A.6:19) we obtain,

\[
\begin{align*}
\text{eq.} & \\
1(1) & \begin{bmatrix} b[3] & 0 \\ 0 & b[4] \end{bmatrix} \begin{bmatrix} x_1 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_1[3] \\ y_5[4] \end{bmatrix} \quad \text{(even case), (V.A.6:37)}
\end{align*}
\]

and

\[
\begin{align*}
\text{eq.} & \\
1(1) & \begin{bmatrix} b[4] & 2 \\ 0 & b[3] \end{bmatrix} \begin{bmatrix} x_1 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_1[4] \\ y_5[3] \end{bmatrix} \quad \text{(odd case), (V.A.6:39)}
\end{align*}
\]

where
\begin{align*}
    b[4] &= 4 - (b[3])^2 \quad (V.A.6:39) \\
\end{align*}

thus evaluating simultaneously \(x_5\) and \(x_1\) from systems \((V.A.6:37)\), \((V.A.6:38)\), respectively.

Therefore, we apparently reach the conclusion that by following the 'parallel way of thinking' we have created independent computational streams, concerning the even and odd rows of the system(s) each time, which can proceed concurrently. Hence, the whole second part of the previously described serial algorithm (i.e., the back-substitution process) can be eliminated, because this way we end up with all the solutions of the original system \((V.A.6:1)\) in one step, simultaneously, thus saving almost \(\frac{1}{2}\) (theoretically) of the total computing time. Diagrammatically the parallel process described above (i.e., for \(n=8\)) is shown in Figure \((V.A.6-f4)\), below.
Figure V A.6 - $f_{4}$: Parallel Evaluation Routing of the Symmetric Constant - Diagonal Periodic Case (‘n’ = 8).

where: \[
\begin{align*}
q_i &= (1, b, 1, y_i) \\
q_i^{[1]} &= q_i
\end{align*}
\]
V.A.7: Implementation of the Parallel Symmetric Constant-Diagonal Periodic Case: Experimental Results and Performance Analysis on the 'Neptune' Prototype System

Herein, a thorough performance exposure and exploitation of the 'Neptune' (MIMD) computer complex is carried out again, by presenting a selection of algorithms which implement the parallel evaluation routing given in Figure (V.A.6-f4) and search for the optimal values of the granularity factor, according to specific sequential subroutines invited in some of the programs, in combination with most of the available parallel constructs on this machine.

For the cyclic odd-even reduction technique the symmetric constant-diagonal periodic case is chosen as the experimental vehicle, since it is more complicated and, as we have mentioned earlier, its concept indirectly includes that of the corresponding non-periodic case.

The assumption of sufficiently diagonally dominant, constant-tridiagonal periodic systems would definitely prevent the occurrence of ill-conditioned situations, during the solution process, which would result in not so accurate answers. In our case, however, because of the computer's limitations on the real arithmetic range \( +10^{-75} \), we were forced into a trade-off in solution accuracy with the performance exploitation of the machine and the technique used. In other words, in our desire to experiment with as large a system as possible, we have considered slightly < diagonally dominant systems, with the diagonal dominance analogously weakening along with the system size increments; this is due to the fact that the initial constant-diagonal entry increases

\[ 0 < x < 10^{-75}, \text{ including zero.} \]
quadratically during the cyclic odd-even reduction process and being system size dependent, for systems of size \( > (64 \times 64) \), reaches such values which cause error situations reported by the computer as: 'FLOATING POINT, DIVIDE BY ZERO AT', e.g., '686C IN $MAIN$'.

This method was, initially, implemented on a serial computer, the IBM 7090, and was chosen in preference to Gaussian elimination because cyclic reduction deals with periodic boundary conditions in a much neater way, avoiding the need for the calculation of auxiliary vectors.

In the subsequent experimentation, and although no restrictions are necessary, for the method, for convenience in balancing the computational load amongst the utilized combinations of the machine's processors, the size of the considered systems was, always, taken to be a power of two.

Let us now work out the algebra of the serial cyclic odd-even reduction method, in a general form, considering the symmetric constant-diagonal periodic system given by (V.A.5:4), with \( n=2^m \) (where \( m \) is any positive integer) number of equations.

In accordance with (par.-V.A.5), at every reduction level, this number is being halved; clearly this process will be repeated recursively until, after \( \log_2 n \) levels of reduction, we obtain a single equation for \( x_n \).

The reduction process formulae, for reduction levels \( \ell=1,2,\ldots,\log_2 n \), are given by

\[
b[\ell] = 2 - (b[\ell-1])^2 \quad (V.A.7:1)
\]

and

\[
y_{1\ell} = y_{1\ell-1} - b[\ell-1]y_{1\ell-1} + y_{1\ell-1} \quad (V.A.7:2)
\]

\( \text{The superscript } \ell \text{ between } [ ] \text{ indicates the level of reduction, while the initial values are: } b[0] = b, y[0] = y, \)
where \( i = 2^k \) (step \( 2^k \)) until \( 2^m \).

In particular, the reader should bear in mind the periodicity of the problem when the first and last rows of each system are considered in the reduction process. It is due to this fact that, exceptionally for the last reduction level, formula (V.A.7:1) is slightly modified as the formula (V.A.5:14).

The solution for the final equation is obtained by the division

\[
x_n = y_n^{[r]}/b^{[r]},
\]

where \( r = \log_2 n \).

The unknowns at level \( r-1 \) can now be found from a backfilling in procedure using the equation

\[
x_1 = (y_1^{[r-1]} - 2x_n^{[r-1]})/b^{[r-1]}, \quad \text{for } r = \frac{n}{2}.
\]

This backfilling procedure is repeated until, finally, all the odd unknowns are found using the original equations.

The recursive backfilling of the solution, in general, for \( i = \log_2 n + 1, \log_2 n, \ldots, 2, 1 \), is given by the formula,

\[
x_1 = (y_1^{[i-1]} - x_1^{[i-2]}(i-1) - x_1^{[i+2]}(i-1))/b^{[i-1]},
\]

where \( i = 2^{(r-1)} \) (step \( 2^{(r-1)} \)) until \( 2^m \).

Note that, in the implementation of the above formula, when the subscripts of the \( x_i \)'s take values lying outside the defined range \( 1 \leq i \leq n \), then as their correct values are considered the boundary values of the corresponding system each time, according to the periodicity of the problem.

In particular, for the last reduction level, i.e., \( i = \log_2 n \), where we have only one equation, the out of bounds \( x_i \)'s are considered to be of zero value. Also, note that, in the analytical exemplification of all algorithms in this Chapter, including the diagrammatical routings given in
Figures (V.A.6-f3,f4), the initial values are considered of being at reduction level one (instead of level zero as implied in the previous formulae), which in fact results in $\log_2 n + 1$ deceptive levels of reduction, and this is for convenient reasons in programming.

The algebraic-complexity sum of formulae (V.A.7:1,2), considering a theoretical equivalence in the arithmetic (see below), is

$$A_C [R] = 2 \log_2 n + 3 \sum_{k=1}^{\log_2 n} \frac{n}{2^k},$$

\hspace{1cm} \text{(V.A.7:6)}

while for the back-substitution phase, i.e., through formula (V.A.7:5), it is

$$A_C [S] = 1 + 3 \sum_{k=1}^{\log_2 n} \frac{n}{2^k}.$$ 

\hspace{1cm} \text{(V.A.7:7)}

Note that, all formulae have been written down according to the operations sequence appearing in the computational process. Hence, the total algebraic-complexity of the serial evaluation routing of the cyclic odd-even reduction method is

$$A_C [R] + A_C [S] = 1 + 2 \log_2 n + 6 \sum_{k=1}^{\log_2 n} \frac{n}{2^k}.$$ 

\hspace{1cm} \text{(V.A.7:8)}

On the other hand, it is obviously quite complex and parallel machine dependent to evaluate, on a per parallel path basis, the total algebraic-complexity of the parallel variant of this method (where no back-substitution phase occurs), taking into account the reduction (for Pipelined Vector or SIMD computers) or increment\(^\dagger\) (for Multi-processors) in parallelism at each reduction level of the process.

In particular, for the last kind of computers, the total algebraic-complexity of the parallel algorithm using the cyclic odd-even reduction

\(^\dagger\)Depending on the, in terms of processors, potential of the parallel machine in hand and until the optimal 'granularity factor' is obtained.
technique, for the same problem, diagrammatically shown in Figure (V.A.6-f4), on a p-processor system is

\[ A_c = 2\log_2 n + \sum_{k=1}^{\log_2 n} \left( \frac{2^k 3n}{p} \right) + \left( \frac{2^k}{p} \right) \]. \quad (V.A.7:9)

In terms of the time-complexity (which conceptually is being considered as a superset of the algebraic-complexity), considering each basic operation requiring the same executional time-step length, we obtain the following theoretical Speed-up (i.e., internal acceleration) and Efficiency (i.e., utilization of the parallel machine) ratios of the parallel variant compared to the serial one

\[
S_p = \frac{T_s^{(t)}}{T_p^{(t)}} = \frac{1 + 2\log_2 n + \sum_{k=1}^{\log_2 n} \left( \frac{2^k 3n}{p} \right) + \left( \frac{2^k}{p} \right)}{1 + 2\log_2 n + \log_2 n + \sum_{k=1}^{\log_2 n} \left( \frac{2^k}{p} \right)} = \frac{1 + 2\log_2 n + \sum_{k=1}^{\log_2 n} \left( \frac{2^k 3n}{p} \right) + \left( \frac{2^k}{p} \right)}{1 + 2\log_2 n + \log_2 n + \sum_{k=1}^{\log_2 n} \left( \frac{2^k}{p} \right)} \approx O(p) \quad (\text{since } \lceil x \rceil \geq x) \quad (V.A.7:10)
\]

and

\[
E_p = \frac{S_p}{p} \leq \frac{1 + 2\log_2 n + \sum_{k=1}^{\log_2 n} \left( \frac{2^k 3n}{p} \right) + \left( \frac{2^k}{p} \right)}{p(1 + 2\log_2 n + \log_2 n + \sum_{k=1}^{\log_2 n} \left( \frac{2^k}{p} \right))} \leq 1^+. \quad (V.A.7:11)
\]

These should not, however, be considered as the true performance ratios, since, as we have repeatedly mentioned, the capabilities of the parallel variant should be compared with the best (i.e., most efficient) existing sequential algorithm for the specific instance, which in our case happens to be Gaussian elimination.

Let us now proceed with the actual experimentation of the cyclic odd-even reduction method at first considering a brief description of

\[ ^+ \text{Which are their optimal theoretical values.} \]
the selected characteristic programs, each making use of a different parallel strategy, or construct available on the NEPTUNE system.

These programs are included in the Appendix C-V, in the same order and under the following meaningful names:

- (i) MB$2^+$.GAUSCDP : GAUS algorithm for the symmetric Constant-Diagonal Periodic case.
- (ii) MB$2^+$.POEGSBP : Parallel cyclic Odd-Even reduction algorithm calling the Gauss Subroutine for the symmetric constant-diagonal Periodic case.
- (iii) MB$2^+$.POERSBP : Parallel cyclic Odd-Even reduction algorithm calling the cyclic odd-even Reduction Subroutine for the symmetric constant-diagonal Periodic case.
- (iv) MB$2^+$.POEXLDP : Parallel cyclic Odd-Even reduction algorithm to make use of the XPFCLD command for the symmetric constant-diagonal Periodic case.
- (v) MB$2^+$.POEDPRP : Parallel cyclic Odd-Even reduction algorithm utilizing solely the $DOPAR/$SPAREND parallel construct for the symmetric constant-diagonal Periodic case.
- (vi) MB$2^+$.POECSCP : Parallel cyclic Odd-Even reduction algorithm utilizing Critical Sections for the symmetric constant-diagonal Periodic case.

\[\text{Directory name.}\]
More analytically, program (i), representing the most efficient sequential algorithm existing, implements the well known Gauss elimination method. As the Relative (or normalized) Speed-up \( R_{SP} \) will be considered the ratio between the experimental time-complexity of this uniprocessor standard solution and the experimental time-complexities of the cyclic odd-even reduction parallel algorithms achieved in a uniprocessor and parallel implementation.

In program (ii) the number of created paths\(^\dagger\) is always equal to the number of available processors each time. Then, after a pre-set number of reduction steps, we continue by applying the Gauss elimination sequential procedure to solve the resulting subsystems, in each of the created parallel paths, simultaneously.

The notion behind program (iii) is similar to that of program (ii) except that, after the pre-set number of reduction steps, we continue by applying the cyclic odd-even reduction technique again, but sequentially this time, to solve the resulting subsystems, in each of the created parallel paths, simultaneously.

In either of programs (ii) and (iii) the parallel paths are created/terminated utilizing the \$DOPAR\$/\$PAREND parallel construct. The principal aim behind these two implementations is, on the one hand, to determine the optimal granularity factor of parallelism and, on the other hand, to compare their parallel performance with that of the following programs.

Program (iv) performs \( \log_2 n \) reduction steps in total, while it creates as many parallel paths as it is possible, i.e., in a manner independent of the number of available processors; but there is an

\(^\dagger\)At each reduction level.
important restriction, that is, the maximum number of parallel paths
that can be generated on this particular MIMD prototype, utilizing
the XPFCL command, is 75. If, however, an alternate command is
utilized, i.e., the XPFCLD command, then as many as 32,767 parallel
paths can be generated. Certainly, in our case, this number is
directly dependent upon the reduction level each time, while the
$DOPAR/$PAREND parallel construct is, again, utilized.

In program (v) the number of created paths† is, as before,
always equal to the number of available processors each time. This
program performs a total number of $\log_2 n$ reduction steps as well,
using solely the $DOPAR/$PAREND parallel construct.

Finally, program (vii) creates the parallel paths in a semi-
asynchronous manner, utilizing critical sections only (i.e., the
$ENTER/$EXIT parallel construct), depending on the reduction level
and the number of available processors each time. In other words,
this program, initially, creates two parallel paths and then, each time,
the next available processor creates the next parallel path. The
total number of reduction levels performed is similarly as before $\log_2 n$.

In terms of the programming strategy followed, in all these
parallel variants, we have declared the following shared arrays:

\begin{align*}
W & : \quad \text{It stores the multipliers of the constant-diagonal} \\
& \quad \text{entry of matrix } A \text{ during the elimination process.} \\
B & : \quad \text{It stores the r.h.s. entries of the considered system} \\
& \quad \text{each time. On this array will be applied the even} \\
& \quad \text{stream of the cyclic odd-even reduction procedure.}
\end{align*}

†At each reduction level.
$C$ : It stores a copy of the array $B$. On this array will be applied the odd stream of the cyclic odd-even reduction procedure.

$X$ : It stores the solution of the system.

$\textsc{inde}$ : It stores the indices of the Even stream of the cyclic odd-even reduction procedure.

$\textsc{indo}$ : It stores the indices of the Odd stream of the cyclic odd-even reduction procedure.

$\textsc{itime}$ : It stores the timing information.

Each program can be, theoretically, distinguished into five different parts, where at the end of the parallel sections all the processors are forced to synchronize. All parts, except the last, are included in a DO-loop the execution of which is related to the pre-set depth of recursion, which determines the granularity factor.

In the first part the odd and even computational streams of the cyclic odd-even reduction procedure are created and the corresponding elements are calculated separately.

In the next three consecutive parts, correspondingly, we interchange the computed r.h.s. values via the appropriate arrays of indices, shuffle the even and odd used indices to the top of the respective arrays, and finally, copy the odd and even used indices at the rear half of each other's array. Note that, all physical 'removals' of the elements between the arrays, after every reduction cycle, have been avoided by the use of an indexing procedure, which experimentally has proved to be far cheaper (in computing time) compared to the former approach. In addition, extensive experimentation proved that
the work involved in these three parts was not worthwhile being performed in parallel and hence the sequential approach was followed (in fact, it proved much faster than the parallel one).

Finally, in the last part, the original system has been reduced to a number of subsystems of pre-determined size, which, in a theoretical environment of a machine with an appropriate potential (in terms of the number of processors), could be solved in one time-step, simultaneously. Note that, in the case that the granularity factor is a (2x2) subsystem, the solution is obtained by repeating the cyclic odd-even reduction procedure once more, as exemplified in (par.-V.A.6).

- Experimental Results

The experimental results obtained for all the parallel variants of the cyclic odd-even reduction algorithm (and for the standard Gaussian elimination) on the NEPTUNE system, along with the values of some other parameters of the deterministic performance model estimated statically, are presented in Tables (V.A.7-t1,t2).

As we mentioned previously, the intention of the experimentation by being inclined, mainly, towards the best achievable parallel performance, in conjunction with the smallest possible uniprocessor time-complexity, led us literally to the upper limits of the NEPTUNE machine, i.e., to the maximum allowed experimental system size for this particular problem which was (2048x2048).

The combinations of the utilized processors (N_PROCS) have been analytically and in the given order stated in the above Tables, due to

\[1\] For a perfect balancing, in the sake of parallelism, between the problem's computational load and the utilized processors we have experimented with a power of 'two' combinations of the latter.
### Table V.A. 7-1: Experimental Results and Performance Measurements, for the Symmetric Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.

<table>
<thead>
<tr>
<th>$M_S$</th>
<th>$G_F$</th>
<th>$N_{PROCS}$</th>
<th>$T_o(e)$ (secs)</th>
<th>$C_P$</th>
<th>$S_P$</th>
<th>$R_S$</th>
<th>$E_P$</th>
<th>$F_P T_o(e)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$\frac{M_S}{2}$</td>
<td>$\emptyset$</td>
<td>4.310</td>
<td>4.310</td>
<td>1</td>
<td>0.578</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\emptyset, 1$</td>
<td>2.260</td>
<td>4.520</td>
<td>1.907</td>
<td>1.102</td>
<td>0.954</td>
<td>1.818</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>1.210</td>
<td>4.840</td>
<td>3.562</td>
<td>2.058</td>
<td>0.890</td>
<td>3.172</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{M_S}{2^3}$</td>
<td>$\emptyset$</td>
<td>4.860</td>
<td>4.860</td>
<td>1</td>
<td>0.512</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\emptyset, 1$</td>
<td>2.570</td>
<td>5.140</td>
<td>1.891</td>
<td>0.969</td>
<td>0.946</td>
<td>1.788</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>1.400</td>
<td>5.600</td>
<td>3.471</td>
<td>1.779</td>
<td>0.868</td>
<td>3.013</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{M_S}{2^4}$</td>
<td>$\emptyset$</td>
<td>5.360</td>
<td>5.360</td>
<td>1</td>
<td>0.465</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\emptyset, 1$</td>
<td>2.820</td>
<td>5.640</td>
<td>1.901</td>
<td>0.883</td>
<td>0.950</td>
<td>1.806</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>1.540</td>
<td>6.160</td>
<td>3.481</td>
<td>1.617</td>
<td>0.870</td>
<td>3.029</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{M_S}{2^5}$</td>
<td>$\emptyset$</td>
<td>5.720</td>
<td>5.720</td>
<td>1</td>
<td>0.435</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\emptyset, 1$</td>
<td>3.030</td>
<td>6.060</td>
<td>1.888</td>
<td>0.822</td>
<td>0.944</td>
<td>1.782</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>1.690</td>
<td>6.760</td>
<td>3.385</td>
<td>1.473</td>
<td>0.846</td>
<td>2.864</td>
</tr>
</tbody>
</table>

- **[POEGSBP]:** Gauss Elimination Subroutine
- **[POERSBP]:** Odd-Even Reduction Subroutine

The resulting subsystems each time are sequentially solved by the corresponding subroutines.
### Table V.A.7-1(cont.d): Experimental Results and Performance Measurements, for the Symmetric Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.

<table>
<thead>
<tr>
<th>$N_{STEP}$</th>
<th>$G_F$</th>
<th>$N_{PROCS}$</th>
<th>$T_{c(e)}$ (secs)</th>
<th>$C_p$</th>
<th>$S_p$</th>
<th>$R_{sp}$</th>
<th>$E_p$</th>
<th>$F_p$</th>
<th>$F_p . T_{c(e)}$ (secs)</th>
<th>$T_{c(e)}$ (secs)</th>
<th>$C_p$</th>
<th>$S_p$</th>
<th>$R_{sp}$</th>
<th>$E_p$</th>
<th>$F_p$</th>
<th>$F_p . T_{c(e)}$ (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$M_S / 2^2$</td>
<td>$\emptyset$</td>
<td>8.670</td>
<td>8.670</td>
<td>1</td>
<td>0.578</td>
<td>1</td>
<td>1</td>
<td>5.460</td>
<td>5.460</td>
<td>1</td>
<td>0.918</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>4.470</td>
<td>8.940</td>
<td>1.940</td>
<td>1.121</td>
<td>0.970</td>
<td>1.881</td>
<td></td>
<td>2.850</td>
<td>5.700</td>
<td>1.916</td>
<td>1.756</td>
<td>1.958</td>
<td>1.835</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>2.340</td>
<td>9.360</td>
<td>3.705</td>
<td>2.141</td>
<td>0.926</td>
<td>3.432</td>
<td></td>
<td>1.550</td>
<td>6.200</td>
<td>3.523</td>
<td>3.232</td>
<td>0.801</td>
<td>3.102</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$M_S / 2^3$</td>
<td>$\emptyset$</td>
<td>9.810</td>
<td>9.810</td>
<td>1</td>
<td>0.511</td>
<td>1</td>
<td>1</td>
<td>6.650</td>
<td>6.650</td>
<td>1</td>
<td>0.753</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>5.090</td>
<td>10.180</td>
<td>1.927</td>
<td>0.984</td>
<td>0.964</td>
<td>1.857</td>
<td></td>
<td>3.490</td>
<td>6.950</td>
<td>1.905</td>
<td>1.436</td>
<td>0.953</td>
<td>1.815</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>2.680</td>
<td>10.720</td>
<td>3.660</td>
<td>1.869</td>
<td>0.915</td>
<td>3.350</td>
<td></td>
<td>1.890</td>
<td>7.560</td>
<td>3.519</td>
<td>2.651</td>
<td>0.880</td>
<td>3.095</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$M_S / 2^4$</td>
<td>$\emptyset$</td>
<td>10.870</td>
<td>10.870</td>
<td>1</td>
<td>0.461</td>
<td>1</td>
<td>1</td>
<td>7.910</td>
<td>7.910</td>
<td>1</td>
<td>0.633</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>5.660</td>
<td>11.320</td>
<td>1.920</td>
<td>0.885</td>
<td>0.960</td>
<td>1.844</td>
<td></td>
<td>4.170</td>
<td>8.340</td>
<td>1.897</td>
<td>1.201</td>
<td>0.948</td>
<td>1.799</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>3.020</td>
<td>12.080</td>
<td>3.599</td>
<td>1.659</td>
<td>0.900</td>
<td>3.239</td>
<td></td>
<td>2.270</td>
<td>9.080</td>
<td>3.485</td>
<td>2.207</td>
<td>0.871</td>
<td>3.036</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$M_S / 2^5$</td>
<td>$\emptyset$</td>
<td>11.920</td>
<td>11.920</td>
<td>1</td>
<td>0.420</td>
<td>1</td>
<td>1</td>
<td>9.120</td>
<td>9.120</td>
<td>1</td>
<td>0.549</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>6.420</td>
<td>12.400</td>
<td>1.923</td>
<td>0.808</td>
<td>0.961</td>
<td>1.848</td>
<td></td>
<td>4.800</td>
<td>9.600</td>
<td>1.900</td>
<td>1.044</td>
<td>0.950</td>
<td>1.805</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>3.340</td>
<td>13.360</td>
<td>3.569</td>
<td>1.500</td>
<td>0.892</td>
<td>3.184</td>
<td></td>
<td>2.650</td>
<td>10.600</td>
<td>3.442</td>
<td>1.891</td>
<td>0.860</td>
<td>2.961</td>
<td>1</td>
<td>5.010</td>
</tr>
</tbody>
</table>

The resulting subsystems each time are sequentially solved by the corresponding subroutines.

The granularity factor in terms of the size of the final sequentially solved subsystems.

[GAUSCDP] $T_{c(e)}$ (secs)
The table below presents experimental results and performance measurements for the symmetric constant-diagonal periodic case of parallel variants of the cyclic odd-even reduction method on the 'NEPTUNE' prototype system, for granularity factors of various sizes.

<table>
<thead>
<tr>
<th>$N_{S}$</th>
<th>$G_F$</th>
<th>$N_{PROCS}$</th>
<th>$T_{C}(e)$ (secs)</th>
<th>$S_P$</th>
<th>$R_S$</th>
<th>$E_P$</th>
<th>$F_{P}, T_{S}(e)$ (secs)</th>
<th>$T_{C}(e)$ (secs)</th>
<th>$S_P$</th>
<th>$R_S$</th>
<th>$E_P$</th>
<th>$F_{P}, T_{S}(e)$ (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$\frac{M_S}{2^2}$</td>
<td>$\varnothing$</td>
<td>17.470</td>
<td>1</td>
<td>0.575</td>
<td>1</td>
<td>10.880</td>
<td>10.880</td>
<td>1</td>
<td>0.923</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1$</td>
<td>8.920</td>
<td>1</td>
<td>1.126</td>
<td>0.979</td>
<td>1.918</td>
<td>5.600</td>
<td>11.200</td>
<td>1.943</td>
<td>1.793</td>
<td>0.971</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1, 2, 3$</td>
<td>4.610</td>
<td>3.790</td>
<td>2.178</td>
<td>0.947</td>
<td>3.590</td>
<td>2.970</td>
<td>11.880</td>
<td>3.663</td>
<td>3.380</td>
<td>0.916</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{M_S}{2^3}$</td>
<td>$\varnothing$</td>
<td>19.760</td>
<td>1</td>
<td>0.508</td>
<td>1</td>
<td>13.250</td>
<td>13.250</td>
<td>1</td>
<td>0.758</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1$</td>
<td>10.140</td>
<td>1.949</td>
<td>0.990</td>
<td>0.974</td>
<td>1.899</td>
<td>6.870</td>
<td>13.740</td>
<td>1.929</td>
<td>1.461</td>
<td>0.964</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1, 2, 3$</td>
<td>5.310</td>
<td>3.721</td>
<td>1.891</td>
<td>0.930</td>
<td>3.462</td>
<td>3.680</td>
<td>14.720</td>
<td>3.601</td>
<td>2.728</td>
<td>0.900</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{M_S}{2^4}$</td>
<td>$\varnothing$</td>
<td>21.950</td>
<td>1</td>
<td>0.457</td>
<td>1</td>
<td>15.760</td>
<td>15.760</td>
<td>1</td>
<td>0.637</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1$</td>
<td>11.330</td>
<td>1.937</td>
<td>0.886</td>
<td>0.969</td>
<td>1.877</td>
<td>8.190</td>
<td>16.380</td>
<td>1.924</td>
<td>1.226</td>
<td>0.962</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1, 2, 3$</td>
<td>5.950</td>
<td>3.689</td>
<td>1.687</td>
<td>0.922</td>
<td>3.402</td>
<td>4.410</td>
<td>17.640</td>
<td>3.574</td>
<td>2.277</td>
<td>0.893</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{M_S}{2^5}$</td>
<td>$\varnothing$</td>
<td>24.210</td>
<td>1</td>
<td>0.415</td>
<td>1</td>
<td>18.170</td>
<td>18.170</td>
<td>1</td>
<td>0.553</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1$</td>
<td>12.500</td>
<td>1.937</td>
<td>0.803</td>
<td>0.968</td>
<td>1.876</td>
<td>9.480</td>
<td>18.960</td>
<td>1.917</td>
<td>1.059</td>
<td>0.958</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1, 2, 3$</td>
<td>6.620</td>
<td>3.657</td>
<td>1.517</td>
<td>0.914</td>
<td>3.344</td>
<td>5.130</td>
<td>20.520</td>
<td>3.542</td>
<td>1.957</td>
<td>0.885</td>
</tr>
</tbody>
</table>

Table V.A.7-t1(cont.d): Experimental Results and Performance Measurements, for the Symmetric Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.
The granularity factor in terms of the size of the final sequentially solved subsystems.

The resulting subsystems each time are sequentially solved by the corresponding subroutines.

<table>
<thead>
<tr>
<th>$N_{\text{STEP}}$</th>
<th>$G_F$</th>
<th>$N_{\text{PROC}}$</th>
<th>$N_{\text{PROC}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$M_S$</th>
<th>$G_F$</th>
<th>$N_{\text{PROC}}$</th>
<th>$\tau_C^{(e)}$ (secs)</th>
<th>$C_p$</th>
<th>$S_p$</th>
<th>$R_S^{(e)}$</th>
<th>$E_p$</th>
<th>$F_p$</th>
<th>$F_p \cdot T_C^{(e)}$ (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{M_S}{2^2}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>35.000</td>
<td>35.000</td>
<td>1</td>
<td>0.575</td>
<td>1</td>
<td>1</td>
<td>21.850</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>$\emptyset, 1, 2, 3$</td>
<td>17.830</td>
<td>35.660</td>
<td>1.963</td>
<td>1.128</td>
<td>0.981</td>
<td>1.927</td>
<td>11.220</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>$\emptyset, 1, 2, 3$</td>
<td>9.170</td>
<td>36.680</td>
<td>3.817</td>
<td>2.193</td>
<td>0.954</td>
<td>3.642</td>
<td>5.890</td>
</tr>
<tr>
<td>$\frac{M_S}{2^3}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>39.740</td>
<td>39.740</td>
<td>1</td>
<td>0.506</td>
<td>1</td>
<td>1</td>
<td>26.670</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>$\emptyset, 1, 2, 3$</td>
<td>20.330</td>
<td>40.660</td>
<td>1.955</td>
<td>0.989</td>
<td>0.977</td>
<td>1.911</td>
<td>13.730</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>$\emptyset, 1, 2, 3$</td>
<td>10.560</td>
<td>42.240</td>
<td>3.763</td>
<td>1.904</td>
<td>0.941</td>
<td>3.541</td>
<td>7.250</td>
</tr>
<tr>
<td>$\frac{M_S}{2^4}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>44.330</td>
<td>44.330</td>
<td>1</td>
<td>0.454</td>
<td>1</td>
<td>1</td>
<td>31.490</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>$\emptyset, 1, 2, 3$</td>
<td>22.780</td>
<td>45.560</td>
<td>1.946</td>
<td>0.883</td>
<td>0.973</td>
<td>1.893</td>
<td>16.330</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>$\emptyset, 1, 2, 3$</td>
<td>11.930</td>
<td>47.720</td>
<td>3.716</td>
<td>1.686</td>
<td>0.929</td>
<td>3.452</td>
<td>8.690</td>
</tr>
<tr>
<td>$\frac{M_S}{2^5}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>48.840</td>
<td>48.840</td>
<td>1</td>
<td>0.412</td>
<td>1</td>
<td>1</td>
<td>36.410</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>$\emptyset, 1, 2, 3$</td>
<td>25.150</td>
<td>50.300</td>
<td>1.942</td>
<td>0.800</td>
<td>0.971</td>
<td>1.886</td>
<td>18.860</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>$\emptyset, 1, 2, 3$</td>
<td>13.220</td>
<td>52.880</td>
<td>3.694</td>
<td>1.521</td>
<td>0.924</td>
<td>3.412</td>
<td>10.100</td>
</tr>
</tbody>
</table>

$[\text{GAUSCDP}]$ $\tau_C^{(e)}$ (secs)

$\phi$ $\tau_C^{(e)}$ (secs) $\phi$ $20.110$

Table V.A.7-1 (cont.d): Experimental Results and Performance Measurements, for the Symmetric Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.
<table>
<thead>
<tr>
<th>Program</th>
<th>$G_P$</th>
<th>$N_{PROCS}$</th>
<th>$T_o(e)$ (secs)</th>
<th>$C_p$</th>
<th>$S_p$</th>
<th>$R_S p$</th>
<th>$F_p$</th>
<th>$F_p T_o(e)$</th>
<th>$T_o(e)$ (secs)</th>
<th>$C_p$</th>
<th>$S_p$</th>
<th>$R_S p$</th>
<th>$F_p$</th>
<th>$F_p T_o(e)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[POEDPRP]</td>
<td>$M_S/(2^{(m-1)})$</td>
<td>$\emptyset$</td>
<td>4.850</td>
<td>4.850</td>
<td>1</td>
<td>0.513</td>
<td>1</td>
<td>1</td>
<td>10.740</td>
<td>10.740</td>
<td>1</td>
<td>0.466</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>5.300</td>
<td>1.830</td>
<td>0.940</td>
<td>0.915</td>
<td>1.675</td>
<td>5.750</td>
<td>11.500</td>
<td>1.868</td>
<td>0.871</td>
<td>0.934</td>
<td>1.744</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>1.530</td>
<td>6.120</td>
<td>3.170</td>
<td>1.627</td>
<td>0.792</td>
<td>2.512</td>
<td>3.240</td>
<td>12.960</td>
<td>3.315</td>
<td>1.546</td>
<td>0.829</td>
<td>2.747</td>
<td></td>
</tr>
<tr>
<td>[POECSFP]</td>
<td>$M_S/(2^{(m-1)})$</td>
<td>$\emptyset$</td>
<td>5.090</td>
<td>5.090</td>
<td>1</td>
<td>0.489</td>
<td>1</td>
<td>1</td>
<td>11.250</td>
<td>11.250</td>
<td>1</td>
<td>0.445</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>5.580</td>
<td>1.824</td>
<td>0.892</td>
<td>0.912</td>
<td>1.664</td>
<td>6.020</td>
<td>12.040</td>
<td>1.869</td>
<td>0.832</td>
<td>0.934</td>
<td>1.746</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>1.770</td>
<td>7.080</td>
<td>2.876</td>
<td>1.407</td>
<td>0.719</td>
<td>2.067</td>
<td>3.660</td>
<td>14.640</td>
<td>3.074</td>
<td>1.369</td>
<td>0.768</td>
<td>2.362</td>
<td></td>
</tr>
<tr>
<td>[POEXLDP]</td>
<td>$M_S/(2^{(m-1)})$</td>
<td>$\emptyset$</td>
<td>5.180</td>
<td>5.180</td>
<td>1</td>
<td>0.481</td>
<td>1</td>
<td>1</td>
<td>11.440</td>
<td>11.440</td>
<td>1</td>
<td>0.438</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>5.600</td>
<td>1.850</td>
<td>0.889</td>
<td>0.925</td>
<td>1.711</td>
<td>6.050</td>
<td>12.100</td>
<td>1.891</td>
<td>0.828</td>
<td>0.945</td>
<td>1.788</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>1.720</td>
<td>6.880</td>
<td>3.012</td>
<td>1.448</td>
<td>0.753</td>
<td>2.267</td>
<td>3.640</td>
<td>14.560</td>
<td>3.143</td>
<td>1.376</td>
<td>0.786</td>
<td>2.469</td>
<td></td>
</tr>
<tr>
<td>[GAUSCDP]</td>
<td>$T_o(e)$ (secs)</td>
<td>$\emptyset$</td>
<td>2.490</td>
<td>2.490</td>
<td>1</td>
<td>0.513</td>
<td>1</td>
<td>1</td>
<td>5.010</td>
<td>5.010</td>
<td>1</td>
<td>0.466</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table V.A.7-12: Experimental Results and Performance Measurements, for the Symmetric Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the 'NEPTUNE' Prototype System, for a Granularity Factor of Size (2x2).
<table>
<thead>
<tr>
<th>Program</th>
<th>$G_F$</th>
<th>$N_{PROCS}$</th>
<th>$T_0(e)$ (secs)</th>
<th>$C_p$</th>
<th>$S_p$</th>
<th>$R_S_p$</th>
<th>$E_p$</th>
<th>$F_p \cdot T_s(e)$ (secs)</th>
<th>$T_0(e)$ (secs)</th>
<th>$C_p$</th>
<th>$S_p$</th>
<th>$R_S_p$</th>
<th>$E_p$</th>
<th>$F_p \cdot T_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[POEDPRP]</td>
<td>$\frac{M_S}{2(m-1)}$</td>
<td>$\emptyset$</td>
<td>23.890</td>
<td>23.890</td>
<td>1</td>
<td>0.420</td>
<td>1</td>
<td>1</td>
<td>52.440</td>
<td>52.440</td>
<td>1</td>
<td>0.383</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>12.620</td>
<td>25.240</td>
<td>1.893</td>
<td>0.796</td>
<td>0.947</td>
<td>1.792</td>
<td>27.540</td>
<td>55.080</td>
<td>1.904</td>
<td>0.730</td>
<td>0.952</td>
<td>1.813</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>6.950</td>
<td>27.800</td>
<td>3.437</td>
<td>1.445</td>
<td>0.859</td>
<td>2.954</td>
<td>15.170</td>
<td>60.680</td>
<td>3.457</td>
<td>1.326</td>
<td>0.864</td>
<td>2.987</td>
<td></td>
</tr>
<tr>
<td>[POECSCP]</td>
<td>$\frac{M_S}{2(m-1)}$</td>
<td>$\emptyset$</td>
<td>24.870</td>
<td>24.870</td>
<td>1</td>
<td>0.404</td>
<td>1</td>
<td>1</td>
<td>54.560</td>
<td>54.560</td>
<td>1</td>
<td>0.369</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>13.170</td>
<td>26.340</td>
<td>1.888</td>
<td>0.762</td>
<td>0.944</td>
<td>1.783</td>
<td>28.760</td>
<td>57.520</td>
<td>1.897</td>
<td>0.699</td>
<td>0.949</td>
<td>1.799</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>7.910</td>
<td>31.640</td>
<td>3.144</td>
<td>1.269</td>
<td>0.786</td>
<td>2.471</td>
<td>16.920</td>
<td>67.680</td>
<td>3.225</td>
<td>1.189</td>
<td>0.806</td>
<td>2.599</td>
<td></td>
</tr>
<tr>
<td>[POEXLDP]</td>
<td>$\frac{M_S}{2(m-1)}$</td>
<td>$\emptyset$</td>
<td>25.300</td>
<td>25.300</td>
<td>1</td>
<td>0.397</td>
<td>1</td>
<td>1</td>
<td>55.270</td>
<td>55.270</td>
<td>1</td>
<td>0.364</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1$</td>
<td>13.230</td>
<td>26.460</td>
<td>1.912</td>
<td>0.759</td>
<td>0.956</td>
<td>1.828</td>
<td>28.840</td>
<td>57.680</td>
<td>1.916</td>
<td>0.697</td>
<td>0.958</td>
<td>1.836</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset, 1, 2, 3$</td>
<td>7.810</td>
<td>31.240</td>
<td>3.239</td>
<td>1.286</td>
<td>0.810</td>
<td>2.623</td>
<td>16.830</td>
<td>67.320</td>
<td>3.284</td>
<td>1.195</td>
<td>0.821</td>
<td>2.696</td>
<td></td>
</tr>
<tr>
<td>[GAUSCDP]</td>
<td>$\frac{M_S}{2(m-1)}$</td>
<td>$\emptyset$</td>
<td>25.300</td>
<td>25.300</td>
<td>1</td>
<td>0.397</td>
<td>1</td>
<td>1</td>
<td>55.270</td>
<td>55.270</td>
<td>1</td>
<td>0.364</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table V.A.7-t2 (cont.d): Experimental Results and Performance Measurements, for the Symmetric Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the 'NEPTUNE' Prototype System, for a Granularity Factor of Size (2x2).
the occurring variations in their relative speeds which would certainly lead to different sets of results if other processor combinations were to be utilized.

The reader should note that it was of no use estimating the Reference internal Speed-up \( \overline{R}_s \), due to the obvious effect of the granularity factor upon the time-complexities obtained.

Some conclusions that can apparently be drawn from the examination of the above Tables are that, in terms of the \( S_p \) and \( E_p \) parameters, all programs exhibit results which improve analogously to the system size experimented with, which at the computer's limits reach values very close to the optimum theoretical ones, i.e., \( S_p \) and \( \leq I \), respectively. In addition, the \( F_p T_s^{(e)} \) parameter also exhibits optimum results of \( O(p) \). These observations, however, are directly dependent upon the granularity factor and are generally analogous to its decrement.

In terms of the real Cost of each parallel variant, as was normally expected, it increases along with the increase of the number of utilized processors.

To accomplish the principally set target, concerning the determination of the optimal granularity factor, a comparison between the different parallel variants of the method would prove that its most efficient value is, always, the size of the systems obtained at the earliest reduction level that the number of created parallel paths becomes equal to the number of utilized processors of the machine. More analytically, we observe that the implementations inviting the sequential cyclic odd-even reduction and the Gauss elimination
subroutines, after a pre-set number of reduction steps, are far superior, and in that priority sequence, compared to the other implementations which proceed until the last reduction level. This can apparently be seen from the time-complexities achieved in a uni-processor and parallel implementation, since the more reduction levels which are performed, the more expensive the implementation becomes.

In particular for the latter implementations, they have been presented in Table (V.A.7-t2) in the priority sequence of the best parallel performance achieved, in conjunction with their time-complexities.

From the aspect of the real life 'true' Relative or normalized Speed-up ($R_S$) results, achieved when compared with the most efficient serial implementation of the Gauss elimination, we observe that the above comments, regarding the optimal granularity factor and the 'best' parallel variant of the cyclic odd-even reduction method, apply equally well for them. In fact for that granularity factor, the results obtained from the MB$^2$.POERSBP parallel variant are always of $O(p)$ and are continuously improving up to the NEPTUNE system's limits. It would certainly be of great interest in experimenting with very-large scale systems (i.e., of size $>10^4$) and a MIMD computer complex of more processors.

The parallel behaviour of all these versions of the cyclic odd-even reduction method, since the symmetric constant-diagonal periodic case consists of the ideal problem for this technique to be applied, is diagrammatically depicted in the Figures below. In particular, Figures (V.A.7-f1,f2) correspondingly exhibit the experimental Time-complexities and the respective Speed-ups achieved on the NEPTUNE prototype system. In Figures (V.A.7-f3,f4) the Relative (or normalized) $^\dagger$ for that matrix size for which the performance analyses are carried out.
Speed-ups and the Efficiences achieved are illustrated, while the real Costs of all these versions are given in Figure (V.A.7-f5).

Finally, let us formally introduce, in Table (V.A.7-t3), the complementary new **local** parameters set for this particular method and already utilized in the theoretical evaluation of its algebraic-complexity and the Tables with the experimental results.

**LOCAL PARAMETERS**

\[
\begin{align*}
A_c[R] & : \text{The Algebraic-complexity of the Reduction process in terms of flops.} \\
A_c[S] & : \text{The Algebraic-complexity of the recursive filling in Solution procedure in terms of flops.} \\
A_c[@] & : \text{The total Algebraic-complexity of the parallel algorithm using the cyclic odd-even reduction technique in terms of flops.} \\
M_S & : \text{The Size of the original Matrix.} \\
G_F & : \text{The Granularity Factor in terms of the size of the final sequentially solved subsystems.} \\
N_{STEP} & : \text{The Number of reduction STEPS.}
\end{align*}
\]

**Table V.A.7-t3:** List of Local Parameters for the Performance Model.

- Performance Analysis

The program and system dependent performance analyses of the parallel variants of the cyclic odd-even reduction method are correspondingly given in Tables (IV.A.7-t4,t5). Note, the modification of the local parameters \(A_c(i,j)\) and \(T_c(i,j)\), appearing in the performance analyses in Chapter IV, to \(A_c_{in}\) and \(T_c_{in}\), respectively,
Figure V.A.7-f1: The Time-Complexity of Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.
Figure V.A.7-2: The Speed-ups achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.
Figure V.A.7-f3: The Relative (or Normalized) Speed-Ups achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.
Figure V.A.7-f4: The Efficiency achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.
Figure V.A.7-f5: The Real Cost of Parallel Variants of the Cyclic Odd-Even Reduction Method for the Symmetric Constant-Diagonal Periodic Case.
implying the Algebraic-complexity (in terms of flops) and the Time-complexity per line of the linear system of equations.

In terms of the \( R_{a(s)} \) global parameter, when the processing-to-shared memory module access ratio is very high, then the cost of calling and setting the \( F\$RITP \) subroutine (i.e., \(-252.6\mu s\)) is, generally, ignorable. In cases similar to the present one, however, this overhead can be considered as an additional real-time flop.

In particular for the implementation involving the mutual exclusion mechanism, we found that, at the period of experimentation and after extensive experiments\(^\dagger\), for the execution of the mechanism itself, i.e., the \$ENTER/$EXIT parallel construct, required a time \((t_{cs})\) of \(-561\mu s\), instead of \(-600\mu s\) as given in the (Appendix C-\(\Pi /\Pi \). II.B.3.1), having used the 'XPFCFL' command to generate the load modules; the processors cycle time \((t_{cy}')\), when waiting access to a critical section resource, was \(-1070\mu s\), while the blocked time \((t_{b}')\) again exhibited some fluctuation in values around \(-319\mu s\).

As we observe from the program dependent performance analyses all programs produce an apparently considerable overhead when accessing the shared data, parallel path scheduling and critical sections resources. Although the best out of this asynchronous MIMD system was achieved by producing long independent tasks in the programs calling the subroutines for the sequential solution part, however, a cross-comparison between the theoretical and experimental results obtained proves that, albeit it was not possible to verify\(^\ddagger\), an optimization mechanism, by means of local registers, must operate in the system taking care of multiple

\(^\dagger\)With the old ('parity') memory installed.
\(^\ddagger\)This is internal information of Texas software not available.
Table 5A/7-14 Program Dependent Performance Analyses, for the Symmetric Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.

<table>
<thead>
<tr>
<th>Processors (p)</th>
<th>A(n)</th>
<th>s</th>
<th>Tc(t)</th>
<th>Tl(t)</th>
<th>Ap</th>
<th>A(p)</th>
<th>Td(p)</th>
<th>(log2 n - 1)</th>
<th>t(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048 x 2048</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
<tr>
<td>1024 x 1024</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
<tr>
<td>512 x 512</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
<tr>
<td>256 x 256</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
<tr>
<td>128 x 128</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
<tr>
<td>64 x 64</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
<tr>
<td>32 x 32</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
<tr>
<td>16 x 16</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
<tr>
<td>8 x 8</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
<tr>
<td>4 x 4</td>
<td>P</td>
<td>Nt</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
<td>Np</td>
<td>P*S</td>
<td>P*S</td>
<td>Sp</td>
</tr>
</tbody>
</table>

**Notes:**
- P: Parallel
- Nt: Number of processors
- s: Shared memory size
- Tc(t): Cyclic Odd-Even Reduction Time
- Tl(t): Serial Odd-Even Reduction Time
- Ap: Parallel efficiency
- A(p): Parallel efficiency
- Td(p): Program dependent performance analysis
- t(n): Execution time

Program Dependent Performance Analyses for the Symmetric Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.
<table>
<thead>
<tr>
<th>PARALLEL PATH</th>
<th>CRITICAL SECTION</th>
<th>LIMITS TO PERFORMANCE</th>
<th>$d_{l}^{(t)}$</th>
<th>$w_{st}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{a(///)}$</td>
<td>$O_{st(///)}^{(t)}$</td>
<td>$R_{a(cs)}$</td>
<td>$O_{st(cs)}^{(t)}$</td>
<td>$S_{d(r)}$</td>
</tr>
<tr>
<td>1: $6.144 \times 10^{3} \frac{3}{p}$ flops</td>
<td>0.051p%</td>
<td>-</td>
<td>-</td>
<td>$m_{p}=152$</td>
</tr>
<tr>
<td>1: $6.144 \times 10^{3} \frac{3}{p}$ flops</td>
<td>0.051p%</td>
<td>-</td>
<td>-</td>
<td>$m_{p}=152$</td>
</tr>
<tr>
<td>1: $6.144 \times 10^{3} \frac{3}{p}$ flops</td>
<td>0.051p%</td>
<td>-</td>
<td>-</td>
<td>$m_{p}=152$</td>
</tr>
<tr>
<td>1: $11.264 \times 10^{3} \frac{3}{p}$ flops</td>
<td>0.028p%</td>
<td>-</td>
<td>-</td>
<td>$m_{p}=254$</td>
</tr>
<tr>
<td>1: $6.144 \times 10^{3} \frac{3}{p}$ flops</td>
<td>0.051p%</td>
<td>1: $6.144 \times 10^{3} \frac{3}{p}$ flops</td>
<td>0.024$2^{\frac{3}{p}}$</td>
<td>$m_{p}=152$</td>
</tr>
<tr>
<td>1: $11.264 \times 10^{3} \frac{3}{p}$ flops</td>
<td>0.028p%</td>
<td>1:11 flops</td>
<td>13.247p%</td>
<td>$m_{p}=254$</td>
</tr>
<tr>
<td>1: $6.144 \times 10^{3} \frac{3}{p}$ flops</td>
<td>0.051p%</td>
<td>1:11 flops</td>
<td>13.247p%</td>
<td>$m_{p}=254$</td>
</tr>
<tr>
<td>1:11 flops</td>
<td>30.697p%</td>
<td>-</td>
<td>-</td>
<td>$m_{p}=254$</td>
</tr>
</tbody>
</table>

Table V.A.7-{t4 (cont.d): Program Dependent Performance Analyses, for the Symmetric Constant-Diagonal Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.
MS
(2048X2048)

(:;::)

s

p

PARALLEL PATH

I

tcy

c

ld(e)

t

tb

lo~e)

Program

!XPFCL]

Ill ,1 ~,1,2,3 J.=l y l.

[POEGSBP]

35.000

1.963 3.817

156

-10,800

~4.6%

IY686

0.052%

[POERSBP]

21.850

l. p

3. 710

151

-10,800

-7%

Hs6

[POEDPRP]

I52.440

1.904 3.457

767

j-10 ,800 ~13. 7%

f--686

[POECSCP]

54.560

1.897 3.225

[POEXLDP]

!XPFCLD]

55.270

1.916 3.284

(j.lsecs)

1212
(cs.) 24' l(~;y
1111

,070

f-o.-o ,8oo

(j.!secs)

~19.7%

-696
(tb)-319

r-17 .8% ~686

(//)

o(e)

0.292%

-

-

0.082%

0.443%

-

-

0.095%

0.868%

-

-

0.085%

1.229%

2.584%

5.935%

1.134%

-

A
-·-·
-·
~·

o,(e)
0

cn(cs)

(~p~~'

)

(//)

st(cs)

T (e)

Table V.A.?-t5: System Dependent Performance Analyses,
for the Symmetrl.c Constant-Dl.agonal
Perl.odl.C Case, of Parallel Var1ants
of the Cycll.c Odd-Even Reductl.On
• • •
Method.

lo~e)

CRITICAL SECTION

PARALLEL
CONTROL
(e)

T (e)
(

~P~"<)

0.115%

SHARED
DATA
(e)

~~
~~

.,

••

.• • •

'.,•,_

t!J
\~de

!XPFCLNJ

otl(//)

!XPFCLSJ 0

tl(s)

(secs)

34.985

0.04%

34.870

0.33%

1.685

21.840

0.05%

21.. 710

0.60%

1.631.

52.320

0.23%

51.783

1.03%

8.284

52.630

3.54%

52.150

0.91% 1'-3 .351

52.560

4.90%

52.040

0.99%

1.999


accesses and transfers to, and from the same shared structures. Indirectly, this was assured through some of our experiments in which we eliminated the excess of the accesses to the shared data resource, by declaring 'dummy' local variables for the multi-used shared structures. The results obtained, though, were more or less the same, if not worse. For future investigation, it would be very interesting to study a general Redundancy Analysis and reveal such optimal aspects for MIMD asynchronous Multiprocessors testbeds, depending upon and thus integrating the 'Deterministic Performance Model' - DPM established in the previous Chapter.

The conclusions presented in the numerical experimentation are, also, reflected from the results of the generalized performance analyses given in Tables (V.A.7-t4,t5), which additionally hint possibilities of different implementation structures in terms of using the available processing power. Note, however, that unpredictable dynamic (internal and external) factors, affecting the behaviour status of the NEPTUNE prototype system, have caused, similarly to the previous experimental results, the particular, deceptive though, declinations from the generally drawn line of conclusions.

With respect to the programs calling the sequential subroutines to perform the solution part, although the overheads of the program involving the Gauss subroutine are less than the other's, its overall higher executonal cost is due to the alteration from cyclic odd-even reduction to Gaussian elimination, which involves an extensive copying procedure of the already obtained results at that reduction level. In fact, the larger the size of the sequentially solved subsystems, or

†It results in better internal acceleration results.
otherwise, the lesser the refinement steps of analysis, the higher the relative time-complexity achieved.

This observation can be proved by considering the ratios between the corresponding time-complexities of these implementations, which call the subroutines to perform serial Gaussian elimination and cyclic odd-even reduction, respectively, for different granularity factors; these ratios are decreasing along with the increase of the reduction steps.

On the other hand, because of these extensive sequential parts of the involved subroutines, the very low rates of accesses resulted in unmeasurable shared memory and parallel path scheduling losses for the corresponding solution parts and therefore their performance analysis has been omitted.

In addition, for all implementations, except the shared resources' contribution to performance degradation, by means of 'waiting' accesses, the unavoidable sequential parts performing the *interchanging* of the computed r.h.s. values via the appropriate arrays of indices, and the *shuffling* and *copying* of the *odd* and *even* used indices to the top of the corresponding arrays and at the rear half of each other's array, respectively, also negatively affect the overall parallel performance.

Consequently, due to all these factors, the potential performance of an algorithm is always affected, thus the balance between the number of parallel paths and their algebraic-complexity and the number of sequential paths and their algebraic-complexity determines the maximum parallel performance obtainable.

Let us now add some more explanatory comments about the implementation utilizing critical sections, to assist in a better understanding of the program dependent performance analysis.
The number of the generated parallel paths, in terms of the utilization cost of the parallel mechanism, depends upon the number of available processors each time; while, in terms of the utilization cost of the mutual exclusion mechanism, the sectioning of the problem's lines, on which the cyclic odd-even reduction technique is applied, depends upon the current reduction level. The allocation of these various created sections to the available processors takes place in an asynchronous manner to benefit from their different relative speeds. Due to this difference in relative speeds, however, the number of implementation cycles executed by each processor (for \( p > 1 \)) may vary; in other words, the faster processor may execute more cycles compared to the slower one, but this is only for cases where there are many sections of small relatively length.

The fact that at the end of each reduction level all utilized processors are forced to synchronize characterizes the implementation as a semi-asynchronous one.

The complementary information to that given in (Appendix C-II/par. II.B.3.I) is as it was presented in (par.-IV.B.3.I). In particular, for the information obtained from the shared array \( \text{ITIME} \) when testing the parallel variants of the cyclic odd-even reduction method, many experimental runs were carried out always considering the average figures accordingly. This information being essential for the estimation of their experiment dependent performance analysis parameters, in the case of a maximum utilization of the \( \text{NEPTUNE} \) system and in the corresponding implementation sequence as it appears in Tables (V.A.7-t4,t5), was as follows:

1) The smallest run-times \( T^{(e)}_p \) to be utilized in formula (IV.B.3.I:25)
were (in secs) 9.160, 5.870, 15.160, 16.900, 16.820;

ii) the total numbers of wait cycles to be utilized in formula
(IV.B.3.1:26) were 156, 151, 767, 1212, 1111, which implied
average numbers of wait cycles per processor of 39, -38, -192, 303,
-278, respectively;

iii) the average experimental timings of all cooperating processors
were (in secs) -9.168, 5.880, -15.168, 16.910, -16.823; and,

iv) the numbers of parallel paths run by each processor, considering
the average of all cooperating processors but $P_0$, were 4, 4, 12,
12, 768.

In particular, for the semi-asynchronous parallel variant the
average numbers of accesses to critical sections and wait cycles to
access resource 1 were -779 and 61, respectively.

The times the system was not used productively ($W$), being estimated
through the formula (IV.B.3.1:22) by using the average experimental
timings in iii), were (in secs) -1.672, 1.670, -8.232, 13.080, -12.022;
a good approximation to these total wasted times can be obtained from
the sum of the wasted times statically and dynamically given in the
performance analysis Tables.

Finally, for the implementation using mutual exclusion, we may
consider a performance limitation in terms of a theoretical upper
bound on the number of cooperating processors, in connection with the
cycle time of the critical sections resources. The corresponding
estimation formula is

$$ S_{cs}(r) = \frac{R_{a(cs)}'f_{pt}}{t_{cs}} \quad \text{(V.A.7:12)} $$

which, in this case, produced ($m_p$) values of $4,216/2^6$ and 7, respectively.
In addition, for the same implementation, the formula (IV.B.3.1:25), for the static estimation of the average Idle time \( \text{id}_{t}^{(t)} \), was modified to include the idle time due to the execution of the mutual exclusion mechanism as

\[
\text{id}_{t}^{(t)} = \frac{\text{maxps-minps} \cdot \sum_{k=1}^{nph} (\text{opt}_{k} \cdot \text{tr}_{k} \cdot \text{pt}_{k} \cdot \text{pt}(p) \cdot \text{lo}(/)) + \text{tcs} \cdot \text{lo}(cs)) \cdot 100}{\text{tp} \cdot 10^6}.
\]

\[\text{(V.A.7:13)}\]

Furthermore, despite the generality of the estimating formulae of the performance model, some very specific implementations of the parallel constructs consist of exceptions of the followed prediction line, thus imposing some minor alterations to the existing formulae. In our case, this takes place for the POESCSCP and the POEXLDP parallel variants.

In particular for the former implementation, the formula (IV.B.3.1:23) is modified as follows,

\[
W_{st} = p \cdot (q \cdot \text{opt}_{p} \cdot \text{lo}(/)) \cdot t + t \cdot N(p \cdot \text{lo}(/)) + t \cdot \text{pt}(p \cdot \text{lo}(cs)) = N(p) \cdot (q \cdot \text{opt}_{p} \cdot \text{lo}(/)) \cdot \sum_{l=1}^{p} (s \cdot cy_{l} - 2) + p \cdot s \cdot \text{lo}(/)) + t \cdot \text{lo}(cs)),
\]

\[\text{(V.A.7:14)}\]

to take care of the semi-asynchronous structure.

Note, for this variant, in Table (V.A.7-t4), the implied use of the \( \text{lo}(cs) \) parameter with values for the reduction and solution parts of \( 2^{l} \) and 1024, respectively.

For both the above implementations \( 2^{l} \) implies the value of

\[
\log_{2}^{n-1} \frac{N_{\text{STEP}}}{2} = 2 (\log_{2}^{n-1} - 1),
\]

\[\text{(V.A.7:15)}\]

\[\text{For the figure given in 'Table (V.A.7-t4)'} \text{ we have considered the case that all four processors of the 'NEPTUNE' system were cooperating.}\]
which for this particular experimental case was 2046; whilst the approximations given for some of the predicted values are mainly due to the existing differences in the processors relative speeds, which are absolutely vital for these types of implementation.

On the other hand, for the POEXLDP parallel variant, the use of the \( L_0(///) \) parameter in formulae (IV.B.3.1:23,25) is superfluous, since the parallel paths are generated independently to the number of utilized processors.

To conclude, in respect to the Table (V.A.7-\#4), note that the normalization and integer rounding of the number of accesses to the shared data resource would introduce slight discrepancies in the results, if they were to be utilized where it was necessary. The correct results are obtained if the real processing-to-access ratios are used, i.e., 3 and 11 flops over 10 and 22 accesses to the shared data resource, respectively, for each implementation cycle.

In accordance to what was discussed in (par.-IV.B.3.1)\(^*\) most of the performance analyses figures can be verified by a cross-examination of the program and system dependent Tables, while, generally, it should be noted that the executional cost of all integer operations, involving shared or local variables, \( DO \)-loop increments, etc., has been considered as ignorable.

\[^{*}\text{Namely, for the same, in terms of processors utilization, implementation instances.}\]
SECTION B

PARALLEL CYCLIC
ODD-EVEN REDUCTION ALGORITHMS
FOR SOLVING
GENERAL-TRIDIAGONAL EQUATIONS
V.B.1: The General Non-Periodic Case: Experimental Results and Performance Analysis on the 'Neptune' Prototype System

In this Section we investigate the tridiagonal equation solvers in their most general forms and obtain their algebraic-complexity counts. The cyclic odd-even reduction algorithms we describe are direct generalizations of the previous Section's parallel algorithms, solving non-constant coefficient systems.

Herein, we shall examine the case of a tridiagonal non-periodic coefficient matrix and experiment, for greatest efficiency when balancing the computing power available from the MIMD parallel system in hand, with a number of equations taken to be a power of 'two'.

For a theoretical simplicity, however, we shall assume that \( n = n' - 1 \), where \( n = 2^m \) and \( m \) is any positive integer, and solve the general tridiagonal set of linear algebraic equations

\[
\begin{bmatrix}
  b_1 & c_1 & & & \\
  a_2 & b_2 & c_2 & & \\
    & a_{n-1} & b_{n-1} & c_{n-1} & \\
    & & a_n & b_n & \\
    & & & a_{n-1} & b_{n-1}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{bmatrix}
= 
\begin{bmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_n
\end{bmatrix}
\]

\[(V.B.1:1)\]
or, in matrix-vector notation
\[ Ax = y \]  \hspace{1cm} (V.B.1:2)

To exemplify the cyclic odd-even reduction process consider the following three adjacent equations from (V.B.1:1),
\[
\begin{align*}
   a_{i-1}x_{i-2} + b_{i-1}x_{i-1} + c_{i-1}x_i &= y_{i-1} \\
   a_i x_{i-1} + b_i x_i + c_i x_{i+1} &= y_i, \\
   a_{i+1}x_i + b_{i+1}x_{i+1} + c_{i+1}x_{i+2} &= y_{i+1}
\end{align*}
\]
for \( i=2,4,\ldots,n'-2 \).

If the first of these equations is multiplied by \( \alpha = -a_i/b_{i-1} \) and the last by \( \gamma = -c_i/b_{i+1} \), and the three equations are added, all reference to the variables \( x_{i-1}, x_{i+1} \) is eliminated, to obtain
\[
\begin{align*}
   a_i^{[1]}x_{i-2} + b_i^{[1]}x_i + c_i^{[1]}x_{i+2} &= y_i^{[1]},
\end{align*}
\]
where
\[
\begin{align*}
   a_i^{[1]} &= a_i \alpha_i^{-1} \\
   c_i^{[1]} &= \gamma_i c_i + 1 \\
   b_i^{[1]} &= b_i + \alpha_i c_i^{-1} + \gamma_i a_i^{[1]} \\
   y_i^{[1]} &= y_i + \alpha_i y_i^{[1]} + \gamma_i y_{i+1}
\end{align*}
\]
\hspace{1cm} (V.B.1:4, 5)

Note that, for the special end equations, the out of bounds \( x_i \)'s are considered of zero value.

The equations (V.B.1:4, 5) relate every second variable and, if written for \( i=2,4,\ldots,n'-2 \), again consist of a tridiagonal set of equations of the same form as the original equations (V.B.1:3), but with different coefficients, i.e., \( a_i^{[1]}, b_i^{[1]}, c_i^{[1]} \). The number of equations in (V.B.1:1) has, thus, been roughly halved.

Obviously the above process can be repeated recursively until, after \( \log_2 n' - 1 \) levels of reduction, only the central equation for
\[ b_{n'/2}^{[r]} x_{n'/2}^{[r]} = y_{n'/2}^{[r]} \quad \text{(V.B.1:6)} \]

where, again, the superscript \( r=\log_2 n' - 1 \) indicates the level of reduction. The solution for the equation (V.B.1:6) is simply obtained by division, i.e.,

\[ x_{n'/2}^{[r]} = y_{n'/2}^{[r]} / b_{n'/2}^{[r]} \quad \text{(V.B.1:7)} \]

The remaining unknowns can now be found from a back filling in procedure. In actual fact, the unknowns at level \( r-1 \) can be found using the equation

\[ x_i = (y_1^{[r-1]} - a_1^{[r-1]} x_{n' - i}/4 - c_i^{[r-1]} x_{i + n' - 4}/4) / b_1^{[r-1]} \quad \text{(V.B.1:8)} \]

for \( i = n'/4 \) and \( 3n'/4 \).

This filling in procedure is repeated until, finally, all the odd unknowns are found using the original equations.

Therefore, the cyclic reduction procedure involves the recursive computation of new coefficients and right-hand sides, for levels \( \ell=1,2,\ldots,m-1 \), from

\[
\begin{align*}
a_1^{[\ell]} &= a_{1-2}^{[\ell-1]} \\
c_1^{[\ell]} &= c_{1+2}^{[\ell-1]} \\
b_1^{[\ell]} &= b_1^{[\ell-1]} + a_1^{[\ell-1]} x_{1-2}^{[\ell-1]} + y_1^{[\ell-1]} x_{1+2}^{[\ell-1]} \\
y_1^{[\ell]} &= y_{1-2}^{[\ell-1]} + a_1^{[\ell-1]} y_{1-2}^{[\ell-1]} + c_1^{[\ell-1]} y_{1+2}^{[\ell-1]} \\
\end{align*}
\]

where,

\[
\begin{align*}
a_i &= -a_{i-2}^{[\ell-1]} / b_{1-2}^{[\ell-1]} \\
y_i &= -c_{i+2}^{[\ell-1]} / b_{1+2}^{[\ell-1]} \\
\end{align*}
\]
and \( i=2^k \) (step \( 2^k \)) until \( n'-2^k \),
with the initial values \( a^{[0]}=a_1, b^{[0]}=b_1 \) and \( c^{[0]}=c_1 \).
For the solution a recursive filling in process is followed, for
\( k=m, m-1, \ldots, 2, 1 \), from
\[
 x_1 = (y_1^{[k-1]} - a_1^{[k-1]} x_1 - c_1^{[k-1]} x_1^{1+2(k-1)})/b_1^{[k-1]}, (V.B.1:11)
\]
where
\[
 i = 2^k (\text{step } 2^k) \text{ until } n'-2^k
\]
and \( x_0 = x_1 = 0 \) when they occur. The routing diagram for this algorithm can be straight-forwardly derived from Figure (V.A.6-f3) bearing in mind the non-periodic nature of the problem.

Also, note that, in this diagrammatical routine the initial values are considered of being at reduction level one (instead of level zero as implied in the previous formulae), which in fact would result in \( \log_2 n' \) deceptive levels of reduction, and this is for convenient reasons in programming.

The algebraic-complexity sum of formulae (V.B.1:8,10), considering as before a theoretical equivalence in the arithmetic, is
\[
 A_{[R]} = 12 \sum_{k=1}^{\log_2 n'-1} \left\lfloor \frac{n}{2^k} \right\rfloor ^{+} , \quad (V.B.1:18)
\]
while for the back-substitution phase, i.e. through formula (V.B.1:11), it is approximately\(^+\)
\[
 A_{[S]} = 1 + 5 \sum_{k=1}^{\log_2 n'-1} \left\lfloor \frac{n}{2^k} \right\rfloor \quad . \quad (V.B.1:18)
\]
Hence, the total algebraic-complexity of the general serial evaluation

\(^+\)To take care of the not exactly divisible number of considered equations.
\(^+\)This approximation is due to the non-periodic treatment of the special end equations at each reduction level.
routing of the cyclic odd-even reduction method is approximately

\[ A^r_{[R]} + A^r_{[S]} \approx 1 + 12 \sum_{l=1}^{\log_2 n' - 1} \left\lfloor \frac{n}{2^l} \right\rfloor + 5 \sum_{l=1}^{\log_2 n' - 1} \left\lfloor \frac{n}{2^l} \right\rfloor. \]  

(V.B.1:14)

On the other hand, as for the symmetric constant-diagonal periodic case, it is obviously quite complex and parallel machine dependent to evaluate, on a per parallel path basis, the total algebraic-complexity of the parallel variant of this method (where no back-substitution phase occurs).

For the particular type of the parallel system in hand, the total algebraic-complexity of the parallel algorithm using the cyclic odd-even reduction technique, for the same problem as previously, whose diagrammatic routing can be straight-forwardly derived from Figure (V.A.6-f4), on a hypothetical p-processor system is approximately

\[ A^r_{[///]} \approx 12 \sum_{l=1}^{\log_2 n' - 1} \left( \frac{2^l}{p} \right)^2 \left( \frac{n}{2^l} \right) - 1) + \frac{2}{p} \left( \frac{\log_2 n'}{p} \right). \]  

(V.B.1:15)

This approximation is due to the fact that the previously considered number of equations does not assist in producing symmetrical workloads\(^+\) for every available processor, each time; in fact, this formula approximates the longest, in terms of arithmetic operations, parallel path, while it should be noted that the number of operations required for the special end equations is only half the number required for the rest. Also, the number of reduction levels, in the parallel implementation, has been increased by one.

In terms of the time-complexity, considering each basic operation

---

\(^+\)Bear in mind the particular construction of the parallel programs.
requiring the same executional time-step length, we obtain the following theoretical Speed-up (i.e., internal acceleration) and Efficiency (i.e., utilization of the parallel machine) ratios of the parallel variant compared to the serial one

\[ S_p = \frac{T_s(t)}{T_p(t)} = \frac{\log_2 n' - 1}{12 \sum_{k=1}^{\left\lfloor \frac{n}{2^k} \right\rfloor} n + 5 \sum_{k=1}^{\left\lfloor \frac{n}{2^k} \right\rfloor} \left( \frac{2^k}{p} \right)^{\frac{1}{2}}} \leq \frac{\log_2 n'}{p \sum_{k=1}^{\left\lfloor \frac{n}{2^k} \right\rfloor} n} \]

and

\[ E_p = \frac{S_p}{p} \leq \frac{\log_2 n'}{p \sum_{k=1}^{\left\lfloor \frac{n}{2^k} \right\rfloor} n} \leq 1. \]

Certainly, the respective optimal values of \( p \) and \( \text{one} \) for the Speed-up and Efficiency factors, as for the symmetric constant-diagonal periodic case, are achieved for very large values of \( n \). Again, note that, these are not the true performance ratios, since the parallel variant should be compared with the most efficient existing sequential algorithm of Gaussian elimination.

Let us now proceed with the actual experimentation of the cyclic odd-even reduction method on the NEPTUNE parallel system, at first considering a brief description of the selected characteristic programs, each making use of a different parallel strategy. These programs are included in the Appendix C-V under the following meaningful names:

\[-(i)\] **MB\&D** \[^\dagger\] **GAUSSGNP**: **GAUSS** algorithm for the General Non-Periodic case.

\[^\dagger\] Which are their optimal theoretical values.

\[^\ddagger\] Directory name.

-(iii) MB$4^+$.POEGSGNP : Parallel cyclic Odd-Even reduction algorithm calling the Gauss Subroutine for the General Non-Periodic case.

-(iv) MB$4^+$.POERSGNP : Parallel cyclic Odd-Even reduction algorithm calling the cyclic odd-even Reduction Subroutine for the General Non-Periodic case.

More analytically, program (i), representing the most efficient sequential algorithm existing, implements the well known Gauss elimination method. As the Relative (or normalized) Speed-up ($R_{sp}$) will be considered the ratio between the experimental time-complexity of this uniprocessor standard solution and the experimental time-complexities of the cyclic odd-even reduction parallel algorithms achieved in a uniprocessor and parallel implementation.

Program (ii) performs $\log_2 n$ reduction steps in total, while the number of created parallel paths, at each reduction level, is always equal to the number of available processors each time.

The structure of programs (iii),(iv) is similar to that of program (ii) except that, after a pre-set number of reduction steps, we continue sequentially by applying the Gauss elimination and the cyclic odd-even reduction techniques, respectively, to solve the resulting subsystems, in each of the created parallel paths, simultaneously.

$^+$Directory name.
In terms of the programming strategy followed, in all these parallel variants, we have declared the following *shared* arrays:

- **EA**: It stores the sub-diagonal entries of the coefficient matrix.
- **EB**: It stores the diagonal entries of the coefficient matrix.
- **EC**: It stores the super-diagonal entries of the coefficient matrix.
- **RHSE**: It stores the r.h.s. entries of the system.

On the above arrays will be applied the *E*ven stream of the cyclic odd-even reduction procedure.

- **INDEX**: It stores the *I*Ndices of the *E*ven stream of the cyclic odd-even reduction procedure.
- **OA**: It stores a *co*y of the array EA.
- **OB**: It stores a *co*y of the array EB.
- **OC**: It stores a *co*y of the array EC.
- **RHSO**: It stores a *co*y of the array RHSE.

On these arrays will be applied the *O*dd-stream of the cyclic odd-even reduction procedure.

- **INDEX**: It stores the *I*Ndices of the *O*dd stream of the cyclic odd-even reduction procedure.
- **WE**: It stores the *mul*tipliers for the *e*ven stream of the cyclic odd-even reduction procedure.
- **WO**: It stores the *mul*tipliers for the *o*dd stream of the cyclic odd-even reduction procedure.
- **X**: It stores the *so*lution of the system.
- **ITIME**: It stores the *ti*ming *i*formation.
- **A**: It stores the computed *ret*urn *po*ints of the cyclic odd-even reduction procedure.
Finally, the theoretical structural subdivision of each program, in general, follows that of the symmetric constant-diagonal periodic case, while the parallel paths are created/terminated utilizing the $DOPAR$/$SPAREND$ parallel construct. For a better balancing of the workload, however, the potential of the parallel system (in terms of the number of utilized processors) is applied on the same computational stream (odd or even) each time. In addition, another particular difference appears in the program calling the cyclic odd-even reduction subroutine, in which we have avoided the natural back-substitution phase by performing the parallel process of the main program serially, in each of the created parallel paths, simultaneously.

- Experimental Results

The experimental results obtained on the NEPTUNE parallel system, for the previous parallel variants of the cyclic odd-even reduction algorithm (and for the standard Gaussian elimination), along with the values of some other parameters of the DPM estimated statically, are presented in Table (V.B.1-t1).

Although the intention of the experimentation was the exploitation of the parallel machine and the method itself, however, the natural programming length and complexity of the implementations (the generality of the problem imposed the plethora of used arrays) forced us, on the one hand, into the solution of strictly diagonally dominant systems with the maximum allowed coefficient matrix size of $(256 \times 256)$, while, on the other hand, to subdivide the actual programs into smaller, appropriately interrelated, subroutines. Certainly,

$^\dagger$By the parallel machine.
The granularity factor in terms of the size of the final sequentially solved subsystems

<table>
<thead>
<tr>
<th>( M_S )</th>
<th>64 x 64</th>
</tr>
</thead>
</table>

The resulting subsystems each time are sequentially solved by the corresponding subroutines

[POEGSGNP]: Gauss Elimination Subroutine

<table>
<thead>
<tr>
<th>( N_{STEP} )</th>
<th>( G_F )</th>
<th>( N_{PROCS} )</th>
<th>( T_c(e) ) (secs)</th>
<th>( C_p )</th>
<th>( S_p )</th>
<th>( R_S )</th>
<th>( E_p )</th>
<th>( F_p \cdot T_s(e) ) (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>( \frac{M_S}{2^3} )</td>
<td>( \emptyset )</td>
<td>1.970</td>
<td>1.970</td>
<td>1</td>
<td>( 0.183 )</td>
<td>1</td>
<td>2.440</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1 )</td>
<td>1.130</td>
<td>2.260</td>
<td>1.743</td>
<td>( 0.319 )</td>
<td>0.872</td>
<td>1.520</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1, 2 )</td>
<td>0.900</td>
<td>2.700</td>
<td>2.189</td>
<td>( 0.400 )</td>
<td>0.730</td>
<td>1.597</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1, 2, 3 )</td>
<td>0.680</td>
<td>2.720</td>
<td>2.897</td>
<td>( 0.529 )</td>
<td>0.724</td>
<td>2.098</td>
</tr>
<tr>
<td>4</td>
<td>( \frac{M_S}{2^4} )</td>
<td>( \emptyset )</td>
<td>2.430</td>
<td>2.430</td>
<td>1</td>
<td>( 0.148 )</td>
<td>1</td>
<td>2.880</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1 )</td>
<td>1.380</td>
<td>2.760</td>
<td>1.761</td>
<td>( 0.261 )</td>
<td>0.880</td>
<td>1.550</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1, 2 )</td>
<td>1.030</td>
<td>3.090</td>
<td>2.359</td>
<td>( 0.350 )</td>
<td>0.786</td>
<td>1.855</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1, 2, 3 )</td>
<td>0.840</td>
<td>3.360</td>
<td>2.893</td>
<td>( 0.429 )</td>
<td>0.723</td>
<td>2.092</td>
</tr>
<tr>
<td>5</td>
<td>( \frac{M_S}{2^5} )</td>
<td>( \emptyset )</td>
<td>2.800</td>
<td>2.800</td>
<td>1</td>
<td>( 0.129 )</td>
<td>1</td>
<td>3.440</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1 )</td>
<td>1.600</td>
<td>3.200</td>
<td>1.750</td>
<td>( 0.225 )</td>
<td>0.875</td>
<td>1.531</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1, 2 )</td>
<td>1.200</td>
<td>3.600</td>
<td>2.333</td>
<td>( 0.300 )</td>
<td>0.778</td>
<td>1.815</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1, 2, 3 )</td>
<td>0.980</td>
<td>3.920</td>
<td>2.857</td>
<td>( 0.367 )</td>
<td>0.714</td>
<td>2.041</td>
</tr>
</tbody>
</table>

[GAUSSGMP]: THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE IS APPLIED UNTIL \( \frac{M_S}{2} \) (2x2) SUBSYSTEMS ARE OBTAINED

<table>
<thead>
<tr>
<th>( M_S )</th>
<th>( \frac{M_S}{2} )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>( N_{STEP} )</th>
<th>( G_F )</th>
<th>( N_{PROCS} )</th>
<th>( T_c(e) ) (secs)</th>
<th>( C_p )</th>
<th>( S_p )</th>
<th>( R_S )</th>
<th>( E_p )</th>
<th>( F_p \cdot T_s(e) ) (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>( \frac{M_S}{2^3} )</td>
<td>( \emptyset )</td>
<td>2.800</td>
<td>2.800</td>
<td>1</td>
<td>( 0.129 )</td>
<td>1</td>
<td>3.440</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1 )</td>
<td>1.620</td>
<td>3.240</td>
<td>1.728</td>
<td>( 0.222 )</td>
<td>0.864</td>
<td>1.494</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1, 2 )</td>
<td>1.200</td>
<td>3.600</td>
<td>2.333</td>
<td>( 0.300 )</td>
<td>0.778</td>
<td>1.815</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \emptyset, 1, 2, 3 )</td>
<td>0.980</td>
<td>3.920</td>
<td>2.857</td>
<td>( 0.367 )</td>
<td>0.714</td>
<td>2.041</td>
</tr>
</tbody>
</table>

Table V.B.1-t1: Experimental Results and Performance Measurements, for the General Non-Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.
### Table V.B.1-1 (cont.d):

Experimental Results and Performance Measurements, for the General Non-Periodic Case, of Parallel Variants of the Cyclic Odd-Even Method on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.

#### Table V.B.1-1 (cont.d):

<table>
<thead>
<tr>
<th>$M_S$</th>
<th>$N_{\text{STEP}}$</th>
<th>$G_F$</th>
<th>$N_{\text{PROCS}}$</th>
<th>$T_o(e)$ (secs)</th>
<th>$C_P$</th>
<th>$S_P$</th>
<th>$R_S$</th>
<th>$E_P$</th>
<th>$F \cdot T_o(e)$</th>
<th>$T_o(e)$ (secs)</th>
<th>$C_P$</th>
<th>$S_P$</th>
<th>$R_S$</th>
<th>$E_P$</th>
<th>$F \cdot T_o(e)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varnothing$</td>
<td>3</td>
<td>$\frac{M_S}{2}$</td>
<td>5.030</td>
<td>5.030</td>
<td>1</td>
<td>0.143</td>
<td>1</td>
<td>1</td>
<td>5.950</td>
<td>5.950</td>
<td>1</td>
<td>0.121</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\varnothing, 1$</td>
<td>4</td>
<td>$\frac{M_S}{2}$</td>
<td>2.730</td>
<td>5.460</td>
<td>1.842</td>
<td>0.264</td>
<td>0.921</td>
<td>1.697</td>
<td>3.180</td>
<td>6.360</td>
<td>1.871</td>
<td>0.226</td>
<td>0.936</td>
<td>1.750</td>
<td></td>
</tr>
<tr>
<td>$\varnothing, 1, 2$</td>
<td>5</td>
<td>$\frac{M_S}{2}$</td>
<td>1.970</td>
<td>5.910</td>
<td>2.553</td>
<td>0.365</td>
<td>0.851</td>
<td>2.173</td>
<td>2.340</td>
<td>7.020</td>
<td>2.543</td>
<td>0.308</td>
<td>0.848</td>
<td>2.155</td>
<td></td>
</tr>
<tr>
<td>$\varnothing, 1, 2, 3$</td>
<td>5</td>
<td>$\frac{M_S}{2}$</td>
<td>1.510</td>
<td>6.040</td>
<td>3.331</td>
<td>0.477</td>
<td>0.833</td>
<td>2.774</td>
<td>1.760</td>
<td>7.040</td>
<td>3.381</td>
<td>0.409</td>
<td>0.845</td>
<td>2.857</td>
<td></td>
</tr>
</tbody>
</table>

**[GAUSSGNP]**

$T_o(e)$ (secs) = 0.720

**[POEGSNP]:** Gauss Elimination Subroutine

**[POERSGNP]:** Odd-Even Reduction Subroutine

---

**THE GRANULARITY FACTOR IN TERMS OF THE SIZE OF THE FINAL SEQUENTIALLY SOLVED SUBSYSTEMS**

The resulting subsystems each time are sequentially solved by the corresponding subroutines.

---

**THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE IS APPLIED UNTIL $\frac{M_S}{2}(2\times2)$ SUBSYSTEMS ARE OBTAINED**
The granularity factor in terms of the size of the final sequentially solved subsystems

<table>
<thead>
<tr>
<th>N_STEPCP</th>
<th>G_P</th>
<th>N_PROCS</th>
<th>[POEGSGNP]: Gauss Elimination Subroutine</th>
<th>[POERSGNP]: Odd-Even Reduction Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(\phi)</td>
<td>8.080</td>
<td>8.080</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(\phi,1)</td>
<td>4.230</td>
<td>8.460</td>
<td>1.910</td>
</tr>
<tr>
<td></td>
<td>(\phi,1,2)</td>
<td>3.180</td>
<td>9.540</td>
<td>2.541</td>
</tr>
<tr>
<td></td>
<td>(\phi,1,2,3)</td>
<td>2.250</td>
<td>9.000</td>
<td>3.591</td>
</tr>
<tr>
<td>4</td>
<td>(\phi)</td>
<td>10.220</td>
<td>10.220</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(\phi,1)</td>
<td>5.380</td>
<td>10.760</td>
<td>1.900</td>
</tr>
<tr>
<td></td>
<td>(\phi,1,2)</td>
<td>3.770</td>
<td>11.310</td>
<td>2.711</td>
</tr>
<tr>
<td></td>
<td>(\phi,1,2,3)</td>
<td>3.890</td>
<td>11.560</td>
<td>3.536</td>
</tr>
<tr>
<td>5</td>
<td>(\phi)</td>
<td>12.270</td>
<td>12.270</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(\phi,1)</td>
<td>6.450</td>
<td>12.900</td>
<td>1.902</td>
</tr>
<tr>
<td></td>
<td>(\phi,1,2)</td>
<td>4.520</td>
<td>13.560</td>
<td>2.715</td>
</tr>
<tr>
<td></td>
<td>(\phi,1,2,3)</td>
<td>3.480</td>
<td>13.920</td>
<td>3.526</td>
</tr>
</tbody>
</table>

[GAUSSGNP] \(\tau_0^{(e)}\) (secs) \(\phi\) \(1.450\)

[POEGGNP]: The cyclic odd-even reduction technique is applied until \(\frac{R_S}{2}\) (2x2) subsystems are obtained

<table>
<thead>
<tr>
<th>N_STEPCP</th>
<th>G_P</th>
<th>N_PROCS</th>
<th>(\phi)</th>
<th>(\phi,1)</th>
<th>(\phi,1,2)</th>
<th>(\phi,1,2,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m-1)</td>
<td>(\frac{M_S}{2^{(m-1)}})</td>
<td></td>
<td>(\frac{M_S}{2^{(m-1)}})</td>
<td>(\frac{M_S}{2^{(m-1)}})</td>
<td>(\frac{M_S}{2^{(m-1)}})</td>
<td>(\frac{M_S}{2^{(m-1)}})</td>
</tr>
<tr>
<td>5</td>
<td>(\phi)</td>
<td>15.600</td>
<td>15.600</td>
<td>1</td>
<td>0.093</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(\phi,1)</td>
<td>8.200</td>
<td>16.400</td>
<td>1.902</td>
<td>0.177</td>
<td>0.951</td>
</tr>
<tr>
<td></td>
<td>(\phi,1,2)</td>
<td>5.680</td>
<td>17.040</td>
<td>2.746</td>
<td>0.255</td>
<td>0.915</td>
</tr>
<tr>
<td></td>
<td>(\phi,1,2,3)</td>
<td>4.450</td>
<td>17.800</td>
<td>3.506</td>
<td>0.326</td>
<td>0.876</td>
</tr>
</tbody>
</table>

Table V.B.1-1t (cont.d): Experimental Results and Performance Measurements, for the General Non-Periodic Case, of Parallel Variants of the Cyclic Odd-even Reduction Method on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.
the fact of involving so many subroutines has contributed the unavoidable and superfluous overhead to 'call' them.

An advantage, due to the sufficient diagonal dominance condition sought from the tridiagonal system, is that the cyclic reduction algorithm may be stopped before completion without loss of accuracy. Let us define the diagonal dominance of the original system of equations \((V.B.1:1)\) as \(\delta\), the minimum over all the equations of the ratios \(|b_1|/|a_1|\) and \(|b_1|/|c_1|\). We can then consider the solution of the simpler set of constant coefficient equations:

\[
ax_{i-1} + bx_i + ax_{i+1} = y_i, \quad i=1,2,...,n, \tag{V.B.1:18}
\]

with \(|b/a| = \delta\). This, in fact, is equally or less diagonally dominant than the original. Consequently, if the system \((V.B.1:18)\) can be solved to a certain approximation, then the original set of equations will be solved more accurately. The cyclic reduction recurrences in \((V.B.1:9)\) for this case become

\[
a^{[l]} = -(a^{[l-1]})^2/b^{[l-1]}, \tag{V.B.1:19}
\]

\[
b^{[l]} = b^{[l-1]} - 2(a^{[l-1]})^2/b^{[l-1]}, \tag{V.B.1:20}
\]

for \(l=1,2,...,\log_2 n'-1\), where \(a^{[0]}=a\), \(b^{[0]}=b\), \(c^{[l]}=a^{[l]}\) and the subscript 'i' has been dropped because the coefficients are the same for all equations. The recurrence relation for the diagonal dominance, obtained dividing equation \((V.B.1:20)\) by equation \((V.B.1:19)\), is

\[
\delta^{[l]} = |b^{[l]}|/|a^{[l]}| = |(\delta^{[l-1]})^2|, \tag{V.B.1:21}
\]

for \(l=1,2,...,\log_2 n'-1\), where \(\delta^{[0]} = \delta\).

Hence, if the initial diagonal dominance \(\delta > 2\), the diagonal dominance

\[\text{\textsuperscript{†}The reduction process.}\]
will grow quadratically at least as fast as equation (V.B.1:21),
and
\[ \delta^{|L|} = \delta^L , \quad \text{for } \delta > 2. \]  
\((V.B.1:22)\)

Some conclusions that can apparently be drawn from the examination of the Table with the experimental results are that, in terms of the internal acceleration and machine utilization parameters, all programs exhibit results which improve analogously to the system size experimented with.

Despite the fact that the computer's hardware restrictions were too severe to not allow our method to be tested for very-very large system sizes, the performance characteristics obtained from our experiments were encouraging, following a similar pattern as that for the symmetric constant-diagonal periodic case.

All performance measurements, in the sake of parallelism, are directly dependent and, in general, although it is not quite obvious\(^\dagger\) due to the imposed experimentation with small size systems, deteriorate analogously with the granularity factor's decrement; since the nature of the method itself assists in a perfect balancing between the problem's computational load and the number of utilized processors only and only if a power of 'two' combinations of the latter is used, this fact justifies the alteration of the performance pattern in the case that three processors are cooperating.

In terms of the real cost of each parallel variant, it, normally, increases along with the increase of the number of utilized processors.

With respect to the determination of the 'best' implementation and the optimal granularity factor, a comparison of the experimental results

\(^\dagger\)From some of the parametric figures.
obtained from all parallel variants proves that the most efficient is the implementation calling the Gauss subroutine at the largest, in terms of the $g_p$ parameter, reduction level experimented with.

Finally, from the aspect of the Relative or normalised Speed-up ($r_s$) results†, achieved when the implementations calling the sequential subroutines are compared with the implementation of Gaussian elimination‡, the improvement in values obtained as the system size increases proves the similar potential of the method, as that for the symmetric constant-diagonal periodic case, and the necessity for experimentation with very-very large systems (i.e., of size at least $10^4$).

The parallel behaviour of all these versions of the cyclic odd-even reduction method, for that matrix size for which the performance analyses are carried out, is diagrammatically depicted in the Figures below.

In particular, Figures (V.B.1-$f_1,f_2$) correspondingly exhibit the experimental Time-complexities and the respective Speed-ups achieved on the NEPTUNE prototype system, while Figures (V.B.1-$f_3,f_4$) display the Efficiencies obtained and the occurring real Costs. All these diagrammatic representations refer only to the optimal results achieved in accordance with the potential of the hardware in hand.

Note that, the local parameters, for the DPM, utilized for the general non-periodic case, stand exactly as they were introduced in Table (V.A.?-t3). As for the symmetric constant-diagonal periodic case,

†Note that, the respective concepts of the Relative or normalised Efficiency and the Reference internal Efficiency (where it applies) may be introduced as $E_{rl} = \frac{r_s}{p}$, $E_{rf} = \frac{r_s}{p}$.

‡It is widely accepted as the most efficient serial method.
Figure V.B.1-f1: The Time-Complexity of Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Non-Periodic Case.
Figure V.B.1-f2: The Speed-ups achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Non-Periodic Case.
Figure V.B.1-f3: The Efficiency achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Non-Periodic Case.
Figure V.B.1-f4: The Real Cost of Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Non-Periodic Case.
however, an appropriate consideration must be given to the definition of the $Ae_{ln}$ and $Te_{ln}$ parameters for the analyzed solution parts of each implementation.

- Performance Analysis

The program and system dependent performance analyses, of the parallel variants of the cyclic odd-even reduction method introduced herein, are correspondingly given in Tables (V.B.1-t2,t3). In the sake of accuracy, a 'ceiling' function should be applied on some of the figures in the former Table, when a $mod\neq0$ division occurs.

In terms of the unavoidable and superfluous overhead of calling every time the various subroutines in each program, extensive experimentation made possible the estimation of this considerable overhead as -216μs (per call); however, note that, this value reflects the particular experimental case in hand, since the overhead analogously increases with the number of arguments involved in the subroutine call.

On the other hand, we should also recall the observation made in the previous Section about the overhead to call and set the $F\&RITP$ subroutine, which can be considered as an additional real-time flop in cases similar to these appearing in the present Chapter. This is quite important in order to justify any discrepancies occurring between the theoretical and experimental results given.

With respect to the loss due to the accessing of the shared data resource, similar indications as before were obtained about the existence of that optimization mechanism, which takes care of the multiple accesses and transfers to, and from the same shared structures.

\[ \text{Namely, when a very 'poor' processing-to-shared module access ratio occurs.} \]
<table>
<thead>
<tr>
<th>$P_n$</th>
<th>$P_{H}$</th>
<th>$G_{F}$</th>
<th>PROCESSORS (p)</th>
<th>$A_c$, ln</th>
<th>$T_c(t)$, ln</th>
<th>$I_0$</th>
<th>$A_p$</th>
<th>$T_c(t)$, p</th>
<th>$L_0$</th>
<th>$R_{a}(s)$</th>
<th>$O_{st}(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{M_{S}}{2^3}$</td>
<td>n</td>
<td>$0(p)$</td>
<td>15 flops</td>
<td>0.0059</td>
<td>128</td>
<td>$\frac{1.92 \times 10^3}{p}$</td>
<td>0.739</td>
<td>1</td>
<td>3</td>
<td>2:1 flop</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{M_{S}}{2^3}$</td>
<td>$p \leq 2^3$</td>
<td>$0(p)$</td>
<td>-</td>
<td>-</td>
<td>$\frac{4}{p}$</td>
<td>Serial Gauss Elimination</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>$\frac{M_{S}}{2^3}$</td>
<td>$p \leq 2^3$</td>
<td>$0(p)$</td>
<td>15 flops</td>
<td>0.0058</td>
<td>128</td>
<td>$\frac{1.92 \times 10^3}{p}$</td>
<td>0.739</td>
<td>1</td>
<td>3</td>
<td>2:1 flop</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{M_{S}}{2^3}$</td>
<td>$p \leq 2^3$</td>
<td>$0(p)$</td>
<td>-</td>
<td>-</td>
<td>$\frac{4}{p}$</td>
<td>Serial Odd-Even Reduction</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>$\frac{M_{S}}{(\log_2 n-1)}$</td>
<td>$n \leq 2$</td>
<td>$0(p)$</td>
<td>15 flops</td>
<td>0.0058</td>
<td>128</td>
<td>$\frac{1.92 \times 10^3}{p}$</td>
<td>0.739</td>
<td>1</td>
<td>$(\log_2 n-1)$</td>
<td>2:1 flop</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{M_{S}}{(\log_2 n-1)}$</td>
<td>$p \leq 2$</td>
<td>$0(p)$</td>
<td>15 flops</td>
<td>0.0058</td>
<td>64</td>
<td>$\frac{0.96 \times 10^3}{p}$</td>
<td>0.370</td>
<td>1</td>
<td>1</td>
<td>1:1 flop</td>
</tr>
</tbody>
</table>

Table V.B.1-t2: Program Dependent Performance Analyses, for the General Non-Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.
### Table V.B.I-t2 (cont.d): Program Dependent Performance Analyses, for the General Non-Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.

<table>
<thead>
<tr>
<th>PARALLEL PATH</th>
<th>LIMITS TO PERFORMANCE</th>
<th>$d(t)$</th>
<th>$w_{st}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_a(///)$</td>
<td>$o_{st}(///)$</td>
<td>$s_d(r)$</td>
<td>$s'_d(r)$</td>
</tr>
<tr>
<td>1.92x10^3 fops $/p$</td>
<td>0.162$p%$</td>
<td>$m_p=212$</td>
<td>$m_p=170$</td>
</tr>
<tr>
<td>1.92x10^3 fops $/p$</td>
<td>0.162$p%$</td>
<td>$m_p=212$</td>
<td>$m_p=170$</td>
</tr>
<tr>
<td>1.92x10^3 fops $/p$</td>
<td>0.162$p%$</td>
<td>$m_p=212$</td>
<td>$m_p=170$</td>
</tr>
<tr>
<td>96x10^3 fops $/p$</td>
<td>0.325$p%$</td>
<td>$m_p=478$</td>
<td>$m_p=382$</td>
</tr>
<tr>
<td>M</td>
<td>s</td>
<td>T(s)</td>
<td>S</td>
</tr>
<tr>
<td>P</td>
<td>o</td>
<td>e</td>
<td>g</td>
</tr>
<tr>
<td>P</td>
<td>o</td>
<td>e</td>
<td>r</td>
</tr>
<tr>
<td>P</td>
<td>o</td>
<td>e</td>
<td>g</td>
</tr>
</tbody>
</table>

Table V.B.1-t3: System Dependent Performance Analyses, for the General Non-Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.
The conclusions derived from the numerical experimentation can be, also, verified from the generalized performance analyses introduced in Tables (V.B.1-t2,t3), which in turn reveal different implementing strategies in terms of using the available processing potential. Unpredictable dynamic (internal and external) factors, however, directly affecting the behavioural status of the NEPTUNE testbed, have caused some, deceptive though, declinations from the generally drawn pattern of conclusions.

In particular for the programs calling the sequential subroutines to perform the solution part, the observations made for the symmetric constant-diagonal periodic case are reversed in favour of the program calling the Gauss subroutine; this is due to the alteration in the sequential procedure, in the cyclic odd-even reduction subroutine, which results in higher executional costs.

In addition, the very low rates of accesses, due to the extensive sequential parts of the involved subroutines, resulted in unmeasurable shared memory and parallel path scheduling losses, for the corresponding solution parts and therefore their performance analysis has been omitted.

Furthermore, for every implementation, a considerable amount of the performance degradation, besides that resulting from the shared resources' demands, is contributed by the unavoidable sequential parts performing the interchanging of the modified elements and the shuffling and copying of the used indices in the appropriate arrays.

Consequently, similarly as before, the balance between the number of parallel paths and their algebraic-complexity and the number of sequential paths and their algebraic-complexity determines
the maximum parallel performance obtainable.

The complementary information to that given in Appendix C-II/par.-II.B.3.1 is as it was presented in (par.-IV.B.3.1). With respect to the information obtained from the shared array ITIME when testing the parallel variants of the cyclic odd-even reduction method, many experimental runs were carried out always considering the average figures accordingly. This essential information for the estimation of their experiment dependent performance analysis parameters, in the case of a maximum utilization of the NEPTUNE prototype and in the corresponding implementation sequence as it appears in Tables (V.B.1-t2,t3), was as follows:

i) The smallest run-times \( T_p^{(e)} \) to be utilized in formula (IV.B.3.1:25) were (in secs.) 2.250, 2.740, 4.440;

ii) the total numbers of wait cycles to be utilized in formula (IV.B.3.1:27) were 71, 80, 180, which implied average numbers of wait cycles per processor of 18, 20, 45, respectively;

iii) the average experimental timings of all cooperating processors were (in secs.) 2.250, ~2.748, 4.445; and,

iv) the numbers of parallel paths run by each processor, considering the average of all cooperating processors but \( P_0 \), were 9,9,17.

The times the system was not used productively (\( W \)), being estimated through the formula (IV.B.3.1:22) by using the average experimental timings in iii), were (in secs.) 0.920, ~1.022, 2.180; again, the sum of the wasted times statically and dynamically (see performance analysis Tables) can give us a good approximation to these total wasted times.
It should be noted that most of the performance analyses figures can be verified by an appropriate examination of the program and system dependent Tables, while taking into account that the executional cost of all integer operations, involving shared or local variables, DO-loop increments, etc., has been considered as ignorable.

Furthermore, to obtain the accurate results presented in Table (V.B.1-t2) the real processing-to-access ratios should be used, i.e., $15$ flops over $36$ and $10^4$ correspondingly accesses to the shared data resource, for each implementation cycle. This is due to the fact that the normalization and integer rounding of the numbers of accesses to the shared data resource would, otherwise, introduce slight discrepancies in the results.

In conclusion, the utilization of the parallel constructs for each set of odd and even lines implies that the parametric figures per parallel path, given in Table (V.B.1-t2), should be doubled.

\[^{\dagger}\text{For the }'\text{fifth}'\text{ phase of the FOEGENNP program.}\]
V.B.2: THE GENERAL PERIODIC CASE: EXPERIMENTAL RESULTS AND
PERFORMANCE ANALYSIS ON THE 'NEPTUNE' PROTOTYPE SYSTEM

We shall continue our investigation on the solution of non-
constant coefficient systems, using the generalized form of the cyclic
odd-even reduction solver, to fully exploit the case of a tridiagonal
periodic coefficient matrix.

As we implied in (par.-V.B.1), although the cyclic odd-even
reduction method is generally classified as a direct method, under
special conditions it behaves like an iterative method.
The values of intermediate quantities converge to final values and may
reach the final values to within machine accuracy well before the full
number of 'iterations' has been performed. The convergence can be
tested, and the algorithm can be terminated early when full machine
accuracy is attained.

Therefore, the cyclic odd-even reduction method can be, literally,
considered as a semi-direct method (or perhaps, should be called a
semi-iterative method).

For an algorithm to be convergent, some dominance conditions must
hold for the tridiagonal system of equations. Most often it is
convenient to assume the system is diagonally dominant.

For our purpose, and under the severe computer's hardware limitations,
we have assumed a strict form of dominance in the systems experimented
with, while, again for greatest efficiency when balancing the computing
power available from the MIMD parallel prototype in hand, the number
of equations was, always, taken to be a power of 'two'.

We wish to solve a general tridiagonal set of linear algebraic
equations, similar to that given in (V.B.1:1) but, with the coefficient
matrix A now having the periodic form

\[
A = \begin{bmatrix}
  b_1 & c_1 & \cdots & a_1 \\
  a_2 & b_2 & c_2 & \cdots & a_{n-1} \\
  & \ddots & \ddots & \vdots & \vdots \\
  & & \ddots & b_{n-1} & c_{n-1} \\
  c_n & & & \cdots & a_n \\
\end{bmatrix}.
\]

\[ (V.B.2:1) \]

Since the theoretical analysis of the cyclic reduction technique, introduced in (par. -V.B.1), indirectly covers the general periodic case, we shall describe the parallel cyclic odd-even reduction process, for this particular case, through the following numerical example (for \( n=8 \)):

\[
\begin{bmatrix}
  4 & 1 \\
  1 & 4 & 2 \\
  1 & 3 & 1 & 0 \\
  2 & 5 & 2 \\
  1 & 3 & 1 \\
  0 & 2 & 6 & 3 \\
  1 & 4 & 2 \\
  1 & 2 & 4 \\
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5 \\
  x_6 \\
  x_7 \\
  x_8 \\
\end{bmatrix}
= \begin{bmatrix}
  7 \\
  7 \\
  5 \\
  9 \\
  5 \\
  11 \\
  7 \\
  7 \\
\end{bmatrix}.
\]

\[ (V.B.2:2) \]

Note, the strict form of diagonal dominance holding and that, for tidy purposes, the solution sought has been pre-arranged to be \( x^T=(1,1,\ldots,1) \).

(i) - The Even Computational Stream

\[
\begin{bmatrix}
  4 & 1 \\
  1 & 4 & 2 \\
  1 & 3 & 1 & 0 \\
  2 & 5 & 2 \\
  1 & 3 & 1 \\
  0 & 2 & 6 & 3 \\
  1 & 4 & 2 \\
  1 & 2 & 4 \\
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5 \\
  x_6 \\
  x_7 \\
  x_8 \\
\end{bmatrix}
= \begin{bmatrix}
  7 \\
  7 \\
  5 \\
  9 \\
  5 \\
  11 \\
  7 \\
  7 \\
\end{bmatrix}.
\]

\[ (V.B.2:3) \]

We just need to take into account the periodic treatment of the special end equations at each reduction level.
1st Reduction Level

\[
\begin{bmatrix}
37/12 & -2/3 & 0 & -1/2 \\
-2/3 & 11/3 & -2/3 & 0 \\
0 & -2/3 & 55/12 & -3/2 \\
-1/4 & 0 & -1/2 & 5/2
\end{bmatrix}
\begin{bmatrix}
x_2 \\
x_4 \\
x_6 \\
x_8
\end{bmatrix}
\begin{bmatrix}
23/12 \\
7/3 \\
29/12 \\
7/4
\end{bmatrix}
\]

\[
\begin{bmatrix}
\alpha_1's \\
\gamma_1's
\end{bmatrix}
\begin{bmatrix}
8/37 \\
3/37
\end{bmatrix}
\]

(V.B.2:4)

\[
\begin{bmatrix}
13/4 & -1/2 & 0 & -1 \\
-1/4 & 21/10 & -2/5 & 0 \\
0 & -2/5 & 34/15 & -1/2 \\
-1/2 & 0 & -1/3 & 5/2
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_3 \\
x_5 \\
x_7
\end{bmatrix}
\begin{bmatrix}
7/4 \\
29/20 \\
41/30 \\
5/3
\end{bmatrix}
\]

\[
\begin{bmatrix}
\alpha_1's \\
\gamma_1's
\end{bmatrix}
\begin{bmatrix}
1/13 \\
2/13
\end{bmatrix}
\]

(V.B.2:5)

2nd Reduction Level

\[
\begin{bmatrix}
6971 & -664 \\
2035 & 2035 \\
-258 & 4672 \\
2035 & 2035 \\
440 & 442 \\
221 & 2009 \\
-30 & 884 \\
221 & 884
\end{bmatrix}
\begin{bmatrix}
x_4 \\
x_8
\end{bmatrix}
\begin{bmatrix}
6307 \\
4414 \\
807 \\
1889
\end{bmatrix}
\begin{bmatrix}
258 \\
3 \\
3
\end{bmatrix}
\]

(V.B.2:6)

\[
\begin{bmatrix}
440 & -73 \\
221 & 442 \\
-30 & 2009 \\
221 & 884
\end{bmatrix}
\begin{bmatrix}
x_3 \\
x_7
\end{bmatrix}
\begin{bmatrix}
807 \\
442 \\
1889 \\
884
\end{bmatrix}
\begin{bmatrix}
3 \\
44
\end{bmatrix}
\]

(V.B.2:7)

\[
\begin{bmatrix}
961 & -73 \\
330 & 330 \\
179 & 2747 \\
660 & 660
\end{bmatrix}
\begin{bmatrix}
x_2 \\
x_6
\end{bmatrix}
\begin{bmatrix}
148 \\
214 \\
179 \\
1922
\end{bmatrix}
\begin{bmatrix}
55 \\
55
\end{bmatrix}
\]

(V.B.2:8)

\[
\begin{bmatrix}
314 & -72 \\
105 & 315 \\
-31 & 223 \\
210 & 105
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_5
\end{bmatrix}
\begin{bmatrix}
174 \\
83 \\
174 \\
83
\end{bmatrix}
\begin{bmatrix}
63 \\
42
\end{bmatrix}
\]

(V.B.2:9)
3rd Reduction Level

\[
\begin{align*}
2.28374695x_8 &= 2.28374695 \quad \Leftrightarrow \quad x_8 = 1 & (V.B.2:10) \\
2.26136364x_7 &= 2.26136364 \quad \Leftrightarrow \quad x_7 = 1 & (V.B.2:11) \\
4.14151925x_6 &= 4.14151925 \quad \Leftrightarrow \quad x_6 = 1 & (V.B.2:12) \\
2.11252654x_5 &= 2.11252654 \quad \Leftrightarrow \quad x_5 = 1 & (V.B.2:13)
\end{align*}
\]

(ii) - The Odd Computational Stream

\[
\begin{array}{cccc}
4 & 1 & 2 & 7 \\
1 & 4 & 2 & x_1 \\
1 & 3 & 1 & x_2 \\
2 & 5 & 2 & x_3 \\
1 & 3 & 1 & x_4 = 9 \\
\hline
0 & 2 & 6 & 3 & x_5 = 5 \\
1 & 4 & 2 & x_6 = 11 \\
2 & 4 & 1 & x_7 = 7 \\
2 & 4 & 1 & x_8 = 7
\end{array}
\]

\[
\begin{array}{cccc}
\alpha_i's & \gamma_i's \\
-1/4 & -1/4 & (V.B.2:14) \\
-1/5 & -1/5 \\
-1/6 & -1/6 \\
-1/2 & -1/2
\end{array}
\]

1st Reduction Level

\[
\begin{align*}
\begin{bmatrix}
13/4 & -1/2 & 0 & -1 \\
-1/4 & 21/10 & -2/5 & 0 \\
0 & -2/5 & 34/15 & -1/2 \\
-1/2 & 0 & -1/3 & 5/2
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\begin{bmatrix}
7/4 \\
29/20 \\
41/30 \\
5/3
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\begin{bmatrix}
37/12 & -2/3 & 0 & -1/2 \\
-2/3 & 11/3 & -2/3 & 0 \\
0 & -2/3 & 55/12 & -3/2 \\
-1/4 & 0 & -1/2 & 5/2
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\begin{bmatrix}
23/12 \\
7/3 \\
29/12 \\
7/4
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\begin{bmatrix}
4/21 & 5/21 & (V.B.2:15) \\
2/5 & 1/5 \\
2/11 & 2/11 & (V.B.2:16) \\
1/5 & 3/5
\end{bmatrix}
\end{align*}
\]
<table>
<thead>
<tr>
<th>2nd Reduction Level</th>
<th>[1]'s</th>
</tr>
</thead>
</table>
| \[
\begin{bmatrix}
314 \\
105 \\
-72
\end{bmatrix}
\begin{bmatrix}
31 \\
210 \\
223
\end{bmatrix}
\frac{x_1}{174}
\begin{bmatrix}
63 \\
83 \\
42
\end{bmatrix}
\frac{72}{669}
\]
| \(\text{(V.B.2:1)}\) |
| \[
\begin{bmatrix}
961 \\
330 \\
-73
\end{bmatrix}
\begin{bmatrix}
179 \\
2747 \\
660
\end{bmatrix}
\frac{x_2}{148}
\begin{bmatrix}
55 \\
214 \\
2747
\end{bmatrix}
\frac{146}{2747}
\]
| \(\text{(V.B.2:18)}\) |
| \[
\begin{bmatrix}
440 \\
221 \\
-73
\end{bmatrix}
\begin{bmatrix}
30 \\
221 \\
2009
\end{bmatrix}
\frac{x_3}{807}
\begin{bmatrix}
442 \\
1889 \\
884
\end{bmatrix}
\frac{146}{2009}
\]
| \(\text{(V.B.2:19)}\) |
| \[
\begin{bmatrix}
6971 \\
2035 \\
-664
\end{bmatrix}
\begin{bmatrix}
258 \\
4672 \\
2035
\end{bmatrix}
\frac{x_4}{6307}
\begin{bmatrix}
2035 \\
4414 \\
2035
\end{bmatrix}
\frac{664}{4672}
\]
| \(\text{(V.B.2:20)}\) |

<table>
<thead>
<tr>
<th>3rd Reduction Level</th>
</tr>
</thead>
</table>
| \[
2.97458894x_1 = 2.97458894 \iff x_1 = 1 \quad \text{(V.B.2:21)}
\]
| \[
2.89770659x_2 = 2.89770659 \iff x_2 = 1 \quad \text{(V.B.2:22)}
\]
| \[
1.98108512x_3 = 1.98108512 \iff x_3 = 1 \quad \text{(V.B.2:23)}
\]
| \[
3.40753425x_4 = 3.40753425 \iff x_4 = 1 \quad \text{(V.B.2:24)}
\]

Note that, the systems \((\text{V.B.2:5}), (\text{V.B.2:8},9)\) and \((\text{V.B.2:16}), (\text{V.B.2:19},20)\) have been, respectively, derived through the complementary, at each reduction level, computational stream.

The routing diagrams for both, serial and parallel implementations can be easily deduced from Figures \((\text{V.A.6-f3,f4})\), respectively.
The recursive computational formulae (for the reduction and back filling in parts), as well as the corresponding algebraic-complexity formulae, taking into account the remark made about the starting level of the initial values, remain as were introduced in (par.-V.B.1). For the former recursive formulae, however, when any of the subscripts takes a value lying outside the defined range \(l \leq n\), then as its correct value is considered to be the boundary value of the corresponding system each time, according to the periodicity of the problem. In particular, for the last reduction level, i.e. \(i = \log_2 n' - 1\), where we have only one equation, the out of bounds \(x_i\)'s are taken to be of zero value. In the actual programming, however, the cyclic odd-even procedure for the last reduction level is apparently more costly than the direct solution of the \((2 \times 2)\) resulting subsystems. On the other hand, in respect of the algebraic-complexity formulae, some of the approximating relations (due to the non-periodic treatment of the special end equations) in the present case should be altered to equalities.

From the actual experimentation aspect, the selected characteristic programs, for the general periodic case, are included in the Appendix C-V under the following meaningful names:


\textsuperscript{+}Directory name.
- (iii) MB$4.\textsuperscript{t}POEGSGP : Parallel cyclic Odd-Even reduction algorithm calling the Gauss Subroutine for the General Periodic case.

- (iv) MB$4.\textsuperscript{t}POERSGP : Parallel cyclic Odd-Even reduction algorithm calling the odd-even Reduction Subroutine for the General Periodic case.

The general structure of the above programs, the parallel strategies followed, the parallel constructs and even the shared arrays utilized, are correspondingly similar to those of the programs introduced in (par.-V.B.1).

Furthermore, in particular for the observations made therein about the better balancing of the workload and the sequential execution, in the cyclic odd-even reduction subroutine, of the parallel process of the main program, still apply to the present periodic case.

- Experimental Results

The experimental results obtained on the NEPTUNE parallel system, for the parallel variants of the cyclic odd-even reduction algorithm (and for the standard Gaussian elimination) introduced herein, along with the values of some other parameters of the DPM estimated statically, are presented in Table (V.B.2-t1).

For the same reasons, as for the general non-periodic case, the maximum coefficient matrix size allowed by the parallel machine to experiment with was of (256x256).

The involvement of so many interrelated subroutines in each

\textsuperscript{t}Directory name.
The granularity factor is too small to apply the Gauss procedure.
### Table V.B.2-t1 (cont.d): Experimental Results and Performance Measurements, for the General Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.

<table>
<thead>
<tr>
<th>$M_S$</th>
<th>$G_F$</th>
<th>$N_PROCS$</th>
<th>$T_a(e)$ (secs)</th>
<th>$T_r(e)$ (secs)</th>
<th>$R_S$</th>
<th>$E_P$</th>
<th>$P_T(e) p_s$</th>
<th>$T_e(e)$ (secs)</th>
<th>$C_P$</th>
<th>$S_P$</th>
<th>$R_S$</th>
<th>$E_P$</th>
<th>$P_T(e) p_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td></td>
<td></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>5.230</td>
<td>1.777</td>
<td>0.563</td>
<td>0.941</td>
<td>1.793</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td>$\emptyset$, 1</td>
<td>2.610</td>
<td>1.885</td>
<td>0.598</td>
<td>1.188</td>
<td>0.761</td>
<td>1.739</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td>$\emptyset$, 1, 2</td>
<td>2.310</td>
<td>1.885</td>
<td>0.732</td>
<td>1.778</td>
<td>0.944</td>
<td>1.739</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td>$\emptyset$, 1, 2, 3</td>
<td>1.420</td>
<td>1.885</td>
<td>1.099</td>
<td>3.001</td>
<td>0.761</td>
<td>1.739</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td>$\emptyset$, 1, 2</td>
<td>3.160</td>
<td>2.558</td>
<td>1.490</td>
<td>2.182</td>
<td>1.777</td>
<td>1.739</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td>$\emptyset$, 1, 2, 3</td>
<td>2.000</td>
<td>2.558</td>
<td>1.490</td>
<td>2.182</td>
<td>1.777</td>
<td>1.739</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td>$\emptyset$, 1, 2</td>
<td>3.630</td>
<td>2.558</td>
<td>1.490</td>
<td>2.182</td>
<td>1.777</td>
<td>1.739</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td>$\emptyset$, 1, 2, 3</td>
<td>2.000</td>
<td>2.558</td>
<td>1.490</td>
<td>2.182</td>
<td>1.777</td>
<td>1.739</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The resulting subsystems each time are sequentially solved by the corresponding subroutines.

[POEGSGP]: Gauss Elimination Subroutine

[POERSGP]: Odd-Even Reduction Subroutine

---

**THE GRANULARITY FACTOR IN TERMS OF THE SIZE OF THE FINAL SEQUENTIALLY SOLVED SUBSYSTEMS**

The resultung subsystems each time are sequentially solved by the corresponding subroutines.
### Table V.B.2-1 (cont.d): Experimental Results and Performance Measurements, for the General Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method on the 'NEPTUNE' Prototype System, for Granularity Factors of Various Sizes.

<table>
<thead>
<tr>
<th>$N_{\text{STEP}}$</th>
<th>$G_F$</th>
<th>$N_{\text{PROCS}}$</th>
<th>$T_c(e)$ (secs)</th>
<th>$C_p$</th>
<th>$S_p$</th>
<th>$R_p$</th>
<th>$F_p$</th>
<th>$F_p.T_c(e)$</th>
<th>$T_c(e)$ (secs)</th>
<th>$C_p$</th>
<th>$S_p$</th>
<th>$R_p$</th>
<th>$F_p$</th>
<th>$F_p.T_c(e)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$\frac{M_S}{2^3}$</td>
<td>$\varnothing, 1$</td>
<td>9.870</td>
<td>9.870</td>
<td>1</td>
<td>0.315</td>
<td>1</td>
<td>1</td>
<td>10.410</td>
<td>10.410</td>
<td>1</td>
<td>0.299</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1, 2$</td>
<td>11.990</td>
<td>11.990</td>
<td>1</td>
<td>0.259</td>
<td>1</td>
<td>1</td>
<td>12.590</td>
<td>12.590</td>
<td>1</td>
<td>0.247</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{M_S}{2^4}$</td>
<td>$\varnothing, 1$</td>
<td>6.250</td>
<td>12.500</td>
<td>1.918</td>
<td>0.498</td>
<td>0.959</td>
<td>1.840</td>
<td>6.560</td>
<td>13.120</td>
<td>1.919</td>
<td>0.474</td>
<td>0.960</td>
<td>1.842</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1, 2$</td>
<td>4.430</td>
<td>13.290</td>
<td>2.707</td>
<td>0.702</td>
<td>0.902</td>
<td>2.442</td>
<td>4.660</td>
<td>13.980</td>
<td>2.702</td>
<td>0.667</td>
<td>0.901</td>
<td>2.433</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1, 2, 3$</td>
<td>3.360</td>
<td>13.440</td>
<td>3.568</td>
<td>0.926</td>
<td>0.892</td>
<td>3.183</td>
<td>3.510</td>
<td>14.040</td>
<td>3.587</td>
<td>0.886</td>
<td>0.927</td>
<td>3.216</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{M_S}{2^5}$</td>
<td>$\varnothing, 1$</td>
<td>7.310</td>
<td>14.620</td>
<td>1.911</td>
<td>0.425</td>
<td>0.956</td>
<td>1.826</td>
<td>7.730</td>
<td>15.460</td>
<td>1.920</td>
<td>0.402</td>
<td>0.960</td>
<td>1.843</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1, 2$</td>
<td>5.120</td>
<td>15.360</td>
<td>2.729</td>
<td>0.607</td>
<td>0.910</td>
<td>2.482</td>
<td>5.490</td>
<td>16.470</td>
<td>2.703</td>
<td>0.566</td>
<td>0.910</td>
<td>2.436</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\varnothing, 1, 2, 3$</td>
<td>3.900</td>
<td>15.600</td>
<td>3.582</td>
<td>0.797</td>
<td>0.896</td>
<td>3.203</td>
<td>4.130</td>
<td>16.520</td>
<td>3.593</td>
<td>0.753</td>
<td>0.898</td>
<td>3.228</td>
</tr>
</tbody>
</table>

**Note:** All times in seconds. The results are obtained sequentially by the corresponding subroutines. The cyclic odd-even reduction technique is applied until $\frac{M_S}{2^2}$ subsystems are obtained.

The resulting subsystems each time are sequentially solved by the corresponding subroutines.

**[GAUSSGP]**: Gauss Elimination Subroutine

**[POECEGP]**: The Cyclic Odd-Even Reduction Technique is Applied Until $\frac{M_S}{2^2}$ Subsystems are Obtained.

**Table V.B.2-1**: The granularity factor in terms of the size of the final sequentially solved subsystems.
Implementation has, again, caused the same superfluous overhead experienced in the previous case.

The conclusions drawn from the examination of the Table with the experimental results, in general, follow the same line as that for the previous cases examined in this Chapter. All performance measurements, from the aspect of parallelism, in a similar manner as for the general non-periodic case, deteriorate analogously with the granularity factor's decrement. This is not, however, apparently shown by some of the parametric figures\(^\dagger\) in the above Table, due to the restricted experimentation with relatively very small size systems. Again, the phenomenon of the alteration in the performance pattern, due to an unavoidably inefficient workload balance, appears in the case that three processors are cooperating.

With respect to the determination of the 'best' implementation, taking into account the real cost figures obtained, this is the implementation calling the Gauss subroutine at the earliest reduction level experimented with, which, therefore, implies the optimal granularity factor value.

Finally, the same observations, as those made for the general non-periodic case, about the Relative or normalized Speed-up\( (R_{SP})\) results, achieved, in specific, when comparing the implementations calling the sequential subroutines with that performing Gaussian elimination, still apply to the present case.

The parallel behaviour of all these versions of the cyclic odd-even reduction method, for that matrix size for which the performance

\(^\dagger\)The occurring fluctuations will smooth out when very-very large size systems are tested.
analyses are carried out, is diagrammatically depicted in the Figures below.

In particular, Figures (V.B.2-f1,f2) correspondingly exhibit the experimental Time-complexities and the respective Speed-ups achieved on the NEPTUNE prototype system, while Figures (V.B.2-f3,f4) display the Efficiencies obtained and the occurring real Costs. These diagrammatic representations refer to the optimal results achieved in accordance with the hardware potential of the parallel machine in hand.

Lastly, note that, the local parameters for the DFM of the present case remain identical to those utilized in (par.-V.B.1).

- Performance Analysis

The program and system dependent performance analyses, of the parallel variants of the cyclic odd-even reduction method introduced herein, are correspondingly given in Tables (V.B.2-t2,t3).

With respect to the programs calling the sequential subroutines, it can be observed, from Table (V.B.2-t3), that although the program calling the Gauss subroutine presents relatively greater static and contention overheads than the other program, however, the alteration in the sequential procedure, in the cyclic odd-even reduction subroutine, results in considerably higher executonal costs for this implementation.

In both these programming cases, for similar reasons to those for the general non-periodic problem, the program dependent performance analysis of their solution parts has been omitted.

Finally, let us consider the essential information for the
Figure V.B.2-f1: The Time-Complexity of Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Periodic Case.
Figure V.B.2-f2: The Speed-ups achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Periodic Case.
Figure V.B.2-f3: The Efficiency achieved by Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Periodic Case.
Figure V.B.2-f4: The Real Cost of Parallel Variants of the Cyclic Odd-Even Reduction Method for the General Periodic Case.
Table IV.B.2-t2: Program Dependent Performance Analyses, for the General Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.

<table>
<thead>
<tr>
<th>Program</th>
<th>Processor</th>
<th>64 processors</th>
<th>128 processors</th>
<th>256 processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(t)</td>
<td>P</td>
<td>-</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>f(t)</td>
<td>p</td>
<td>0.370</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>g(t)</td>
<td>p</td>
<td>0.299</td>
<td>2.1 flop</td>
<td>2.1 flop</td>
</tr>
<tr>
<td>h(t)</td>
<td>p</td>
<td>0.411</td>
<td>0.411</td>
<td></td>
</tr>
<tr>
<td>T0(t)</td>
<td>p</td>
<td>0.799</td>
<td>0.209</td>
<td></td>
</tr>
<tr>
<td>T1(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T2(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T3(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T4(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T5(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T6(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T7(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T8(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T9(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T10(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T11(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T12(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T13(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T14(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T15(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T16(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T17(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T18(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T19(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T20(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T21(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T22(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T23(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T24(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T25(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T26(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T27(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T28(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T29(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T30(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T31(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T32(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T33(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T34(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T35(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T36(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T37(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T38(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T39(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T40(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T41(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T42(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T43(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T44(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T45(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T46(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T47(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T48(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T49(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T50(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T51(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T52(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T53(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T54(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T55(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T56(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T57(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T58(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T59(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T60(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T61(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T62(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T63(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T64(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T65(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T66(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T67(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T68(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T69(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T70(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T71(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T72(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T73(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T74(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T75(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T76(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T77(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T78(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T79(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T80(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T81(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T82(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T83(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T84(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T85(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T86(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T87(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T88(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T89(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T90(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T91(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T92(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T93(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T94(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T95(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T96(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T97(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T98(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T99(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T100(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T101(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T102(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T103(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T104(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T105(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T106(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T107(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T108(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T109(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T110(t)</td>
<td>p</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Note: The table continues with more rows and columns, but they are not visible in the image.
4
'
'_,

I

•••

LIMITS TO PERFORMANCE

PARALLEL PATH

(t)

Ra( I I)

l:1.92x10
p

ostrll)

,1.92xl0
p

d(r)

8

8

d(r)

3
flop' o.l62p%

1

8

-

m =212
p

-

m =170
p

m
p

-

-

-

3
1 :2 .048xlo flop• O.l52p%
p

m =212
p

-

mp=226

m =170
p

m
p

616
p

~

616
p

~

-

mp=l81

h(r)

-

3
flop• 0.162p%

I it)
t

wst
(secs)

...0.4%

0.025

-

-

-0.4%

0.025

-

-

657
m --p p

0.058
-0.5%

3

1:

o.96xlo fl
ops
p

o.325p%

m·=478
p

m =382
p

m
p

308
p

~

0.006

Table V.B.2-t2 (cont.d): Program Dependent Performance Analyses, for the General
Per1od1c Case, of Parallel Var1ants of the Cycl1c OddEven Reduct1on Method.


| $M_S$ | $T_e^e$ (secs) | $S_p$ | $P$ | $c_y_i$ | $t_{ey}$ (micros) | $I_d^{e}$ | $t_b$ (micros) | PARALLEL PATH $O^{e}_{st}$(///) | $O^{e}_{cn}$(///) | $O^{e}_{til}$(///) | PARALLEL CONTROL $T_e^e$ (secs) | $T_e^e$ (secs) | SHARED DATA $W_{d0}$ (secs) |
|--------|----------------|-------|-----|---------|----------------|----------|-------------|----------------------|----------------|--------------------|----------------------|---------------|----------------|----------------|
| [POEGSGP] | 9.870 | 1.917 | 2.419 | 3.589 | 85 | -10,800 | -8.3% | -686 | 0.39% | 0.53% | 9.850 | 0.20% | 9.810 | 0.41% | 0.918 |
| [POERSGP] | 10.410 | 1.924 | 2.410 | 3.615 | 84 | -10,800 | -7.9% | -686 | 0.38% | 0.50% | 10.380 | 0.29% | 10.330 | 0.48% | 0.907 |
| [POEGENP] | 18.160 | 1.908 | 2.760 | 3.554 | 178 | -10,800 | -9.5% | -686 | 0.40% | 0.60% | 18.105 | 0.30% | 18.010 | 0.52% | 1.922 |

*Table V.B.2-t3: System Dependent Performance Analyses, for the General Periodic Case, of Parallel Variants of the Cyclic Odd-Even Reduction Method.*
estimation of all the experiment dependent performance analysis parameters, for the parallel variants of the cyclic odd-even reduction method, obtained from the shared array \textit{ITIME}. This information, having averaged the respective figures of many experimental runs, in the case of a maximum utilization of the \textit{NEPTUNE} prototype and in the corresponding implementation sequence as it appears in \textit{Tables (V.B.2-t2,t3)}, was as follows:

\begin{itemize}
  \item[i)] The smallest run-times $T^{(e)}_p$ to be utilized in formula (IV.B.3.1:25) were (in secs.) 2.730, 2.870, 5.110;
  \item[ii)] the total numbers of wait cycles to be utilized in formula (IV.B.3.1:27) were 85, 84, 178, which implied average numbers of wait cycles per processor of -21, 21, -45, respectively;
  \item[iii)] the average experimental timings of all cooperating processors were (in secs.) 2.740, 2.875, 5.110; and,
  \item[iv)] the numbers of parallel paths run by each processor, considering the average of all cooperating processors but $P_0$ were 9, 9, 17.
\end{itemize}

The times the system was not used productively ($\bar{W}$), being estimated through the formula (V.B.3.1:22) by using the average experimental timings in \textit{iii)}, were (in secs.) 1.090, 1.090, 2.280. Note that, the sum of the wasted times statically and dynamically can give us a good approximation to these total wasted times.

For the verification of the results presented in \textit{Table (V.B.2-t2)} the real processing-to-access ratios should be used, i.e., 15 $\textit{flops}/36^\dagger$, 16 $\textit{flops}/36^\dagger$ and 15 $\textit{flops}/16^\ddagger$ correspondingly accesses to the shared

\textit{\dagger} For the programs calling the sequential subroutines.
\textit{\ddagger} For the reduction and solution parts, respectively, of the last implementation.
data resource, for each implementation cycle. Due to the same reason, however, as that in (par.-V.B.1), the parametric figures per parallel path, given in the above Table, should be doubled.

To conclude, the reader should, once more, bear in mind, on the one hand, the overheads contributed by the unavoidable calls of the various interrelated subroutines in each program and the call and set of the F$RITP subroutine every time; while, on the other hand, the deduced existence of an optimization mechanism, which takes care of the multiple accesses and transfers to, and from the same shared structures.

Furthermore, one should take into account the additional performance degradation contributed by the unavoidable sequential parts carrying out the *interchanging* of the modified elements and the *shuffling* and *copying* of the used indices in the appropriate arrays. As usual, the minor losses due to all sorts of integer operations, *DO*-loop increments, etc., have been considered as ignorable.
V.B.3 GENERAL COMMENTS AND CONCLUSIONS

The discussion in the previous Sections shows that parallel computation is not a simple and direct adaptation of the serial computation, but that it is an area with its own set of characteristic problems over and above any attributes inherited from serial computation.

Most of the results obtained have generated cause for optimism about the cost-effectiveness of parallel computers for selected classes of computations. Since the construction of parallel computers has still a long way to advance, further research in parallel computation stands to influence the architecture of parallel computers, as much as the programming of parallel computers.

It is relatively easy to invent algorithms. In practice, however, one wants not only algorithms, one wants effective algorithms. Thus, the objective is to invent effective algorithms and prove their effectiveness. The effectiveness of an algorithm can be appraised by a variety of criteria. One of the most important is the time taken to execute it. There are several aspects of such a time criterion, one being the execution time required by different algorithms for the solution of a particular problem on a particular computer architecture. Such an empirical measure, however, is strongly dependent upon both the program and the machine used to implement it.

A useful alternative to such empirical measurements is a mathematical analysis of the intrinsic difficulty of solving a problem computationally, which, judiciously used, can provide an important means of evaluating the executional cost of the algorithm.

In general, the cost of obtaining a solution increases with the problem size. If its value is sufficiently small, even an inefficient
algorithm will not cost much to run; consequently, the choice of an algorithm for a small problem is not critical (unless the problem is to be solved many times).

In analyzing algorithms for solving numerical problems, the accuracy of the computed results is another very important criterion for distinguishing between effective and ineffective algorithms, in relation to the computer in hand.

In particular for the complexity analysis amongst other things, it is concerned with obtaining upper and lower bounds on the performance of algorithms. The existence of complexity bounds for the algorithms available can serve as a basis for classifying problems. In terms of optimality, the goal of complexity analysis is to show that in order to solve a certain problem computationally, one requires a certain number of operations of a certain type. This is a very difficult objective; for the majority of practical problems we still have to rely on experience to judge the effectiveness of an algorithm.

In particular for the various presented versions of the cyclic odd-even reduction method, the conclusions that we arrive at, after the detailed tests and comparisons of the results, are not innovative, but they consist of a verification of what has been discussed already. The most important conclusion is that, in order to obtain the best possible performance on an asynchronous parallel computer architecture, we should construct the unavoidable sequential paths of a program as short as possible, whereas, to the contrary, the parallel paths as long as possible, minimizing at the same time the accesses to the shared resources both in number and duration.

On the other hand, testing all these different parallel algorithmic
variants of the method, we do not only test and solve the problem itself, but at the same time we examine and test the parallel system, proving its weaknesses by means of overheads and idle losses due to the various relative speeds of the processors. Furthermore, the way that the parallel routing of the cyclic odd-even reduction method has been constructed, allows us, in future investigation, a straightforward implementation on VLSI processor arrays utilizing a tree-structure of processing elements.

Finally, the results of the experimental attempts herein should assist hardware designers, in the near future, to provide or invent a parallel architecture which will be in a position to meet the resources demands of most algorithms; but, this can only succeed if and only if they will be guided from such or similar results, by grouping them in sets presenting similar resources demands.
Chapter VI: A New Class of 'Pipeline' Array Architectures for Algorithm Systolization

Section A: On Algorithmically Specialized Systolic Networks Using the 'Rotating' and 'Folding' Technique

VI.A.1: Introduction

VI.A.2: Classification and Principles of 'Systolic' Algorithms

VI.A.3: An Abstract Mathematical Model for the Verification of 'Systolic' Networks

VI.A.4: The Data Stream 'Rotating' and 'Folding' Technique

VI.A.5: A 'Rotating' and 'Folding' Algorithm Using a Two-Dimensional 'Systolic' Communication Geometry

VI.A.6: 'Systolic' LU-Factorization Dequeues for Tridiagonal Systems

VI.A.6.1: Dequeues for Solving Triangular Linear Systems

VI.A.6.2: General Comments: The Pivoting Problem and Orthogonal Factorization

Section B: Concurrent Systolization for Solving General Banded Linear Systems

VI.B.1: 'Systolic' LU-Factorization Dequeues for Quindiagonal Systems

VI.B.1.1: Modified Dequeues for the Unidirectional Factorization of the 'Central' Subsystems

VI.B.2: 'Systolic' Pipelinability 'Rotating' and 'Folding' General Banded Matrices

VI.B.3: Further Research in the 'Soft-Systolic' Area, Conclusive Remarks
A NEW CLASS OF
'PIPELINED' ARRAY ARCHITECTURES
FOR ALGORITHM SYSTOLIZATION
On Algorithmically Specialized Systolic Networks Using the 'Rotating' and 'Folding' Technique
VI.A.1: INTRODUCTION

"I have not kept the square, but that to come
Shall all be done by the rule."

William Shakespeare.

In any given technology formation follows function in a particular way. The most efficient first step towards understanding the architectural possibilities of a technology is the study of carefully selected existing designs.

In order to realize, however, how the evolution of the VLSI technology has changed the nature of the game we must briefly discuss the progress carried on in Integrated Circuits (IC) technology.

All IC work depends on the foundation of semiconductor-device physics that provides the essential knowledge of how the IC's elements function. Logically built on top of this knowledge is that of semiconductor fabrication technology, which allows the designed IC's to be physically constructed, and above this in turn is the circuit- and logic-design knowledge that the IC designer brings to his part of the overall engineering task.
At present, device physics is a mature area of knowledge in which the number of dramatically new ideas being added is small, while fabrication technology, although still developing, is reaching the point at which we have essentially all the fundamental knowledge that will be required.

On the other hand, circuit- and logic-design techniques are still developing rapidly - there is still room left for 'some cleverness', but that too will soon saturate.

The three component-skills of current IC design work are thus all very well established, but are they any longer all that is required? In fact, all the evidence available strongly suggests that these traditional skills are not enough. Throughout the world, design teams working on VLSI projects have all found the need for an even higher level set of skills to coordinate their lower level design efforts, skills that can generally be summed-up as forming a large-systems design methodology. The knowledge and techniques required for future VLSI systems are only just beginning to be acquired and many fundamental ideas have yet to be discovered.

Inevitably, the many different groups working in this field all have their own particular approach to the problems of establishing an effective formal design methodology, but certain ideas seem to have been almost universally adopted, and these seem likely to form the basic elements of whatever fully developed methodologies emerge in future.

While the IC design has traditionally been a highly specialized task, and certainly the design of a typical custom LSI chip is far from being a task for an 'amateur', the changed nature of VLSI work means
that specialists in other areas of technology are increasingly realizing as possible the design and manufacture of their own VLSI devices, at least for experimental and prototyping purposes.

The VLSI technology has made one thing clear. Simple and regular interconnections lead to cheap implementations and high densities, and high density implies both high performance and low overhead for support components.

The system architecture and design, however, like any art, can only be learned by attempt. To carry a small design from conception through to successful completion provides the confidence necessary to undertake larger designs.

Further on, and due to the fact that an application is efficiently implemented on a VLSI circuit chip if any large problem can be tackled on a considerably small network of processors, let us abstractly introduce here the three approaches considered by H.T. Kung, in [KUNG80], one can follow for solving a large problem on a small network:

i) Use algorithms with large modular granularity, i.e., each processor handles a large group of elements, rather than a few elements. This approach is suitable for SIMD machines, where processors can have relatively large local memories.

ii) Decompose the problem, so that the resulting subproblems will be small enough to be solved on the small network of processors.

iii) Decompose an algorithm that originally requires a large network, i.e., there is a further analysis of the simultaneous operations invoked in every step of the original algorithm, so they will be performed in more than one step by fewer processors of the smaller network.
The idea behind the three approaches is a functional dependency amongst the number of the processors involved in the network and the time that they will be kept busy. The longer the time kept busy and the sequences of data, the better, from the efficiency point of view, the VLSI chip.

In this Chapter a new class of 'pipelined' array structures for algorithm systolization is introduced.

In the present Section A we initiate the reader to the classification and fundamental principles of 'systolic' algorithms, along with the presentation of an abstract mathematical model for the verification of 'systolic' networks.

The principal part of this Section covers the investigation of a new data stream 'rotating' and 'folding' technique on algorithmically specialized 'systolic' networks.

At first, the matrix-vector and matrix multiplication problems are tackled using two-dimensional 'systolic' communication geometries. Then, 'systolic' LU-factorization dequeues for tridiagonal systems are presented and both cases, when n-odd and n-even, are investigated, along with the introduction of the necessary mathematical background of the double Gaussian elimination streams for full matrices. Further on, the complementing dequeues for solving the resulting triangular linear systems are correspondingly given.

Finally, this Section concludes with general comments about the existing pivoting problem and the 'orthogonal' factorization based on Givens's transformation.

In Section B the investigation is extended to concurrent systolization for solving general banded linear systems.
In particular, 'systolic' LU-factorization dequeses are given for quindiagonal systems and the complexity problems occurring are exemplified. This fact led to the introduction of modified dequeses for the unidirectional elimination of the 'central' subsystems, thus bypassing the superfluous complexity caused by the overlapping of the opposite factorization streams.

Finally, the boundaries of the central formatted submatrix for the general banded case are investigated, along with the introduction of the complementary background theory simplifying the variety of the occurring cases.

The Chapter concludes with a part of the work given in [BEKA85a], which extends the research into the 'soft-systolic' area by proceeding to higher level foldings and proving the superiority of this new 'systolic' solution technique compared with the other approaches discussed earlier herein.
VI.A.2: CLASSIFICATION AND PRINCIPLES OF 'SYSTOLIC' ALGORITHMS

Systolic processors are a new class of 'pipelined' array architectures, pioneered by H.T. Kung, which are becoming increasingly attractive because of continuous advances in VLSI technology.

As we have already mentioned in Chapter III, a systolic system is a 'network of processors which rhythmically compute and pass data through the system'. The fundamental operation on systolic arrays is a multiply-and-add performable by an 'Inner-Product-Step' - (IPS) cell. It is shown in [KUNG78] that some of these basic cells can be locally connected together to perform digital filtering, matrix multiplication, and other related operations.

The systolic array features the important properties of modularity, regularity, local interconnection, a high degree of pipelining, and highly synchronized multiprocessing. The data movements in a systolic array are often described in terms of the 'snapshots' of the activities. As the unit of time is considered the time necessary to achieve a multiply-and-add operation.

One of the major challenging research items, therefore, has become the development of algorithms that can be mapped into and executed efficiently by a special-purpose computer system. Algorithms that match with systolic systems, utilizing extensive pipelining and multiprocessing, are called systolic algorithms. As we have previously mentioned, systolic algorithms, in a general comparison with SIMD and MIMD algorithms, are the most structured and MIMD algorithms are the least structured. More specifically, systolic algorithms deal with simple and frequently interacting task modules, while the situation is reversed for MIMD algorithms.
Recent developments in programming languages along with the chip technology has made it possible to classify systolic algorithms into broad groups dependent on their properties.

We classify systolic algorithms into two main sets (see Bekakos and Evans [BEKA85]):

i) **Hard-systolic algorithms**: denoted $S_H$, and

ii) **Soft-systolic algorithms**: denoted $S_S$.

**Hard-systolic algorithms**

These are the traditional algorithms which, as well as observing the general features given above, are subjected to further restrictions placed on their designs. In other words, the graph model representation must be planar, broadcasting to cells to be avoided, or a limited\(^1\) amount to be allowed (Semi-hard-systolic algorithms), and the least amount of area in a chip design to be required.

**Soft-systolic algorithms**

These algorithms are more flexible than the Hard-systolic algorithms, since non-planar graphs may be represented and area is not, directly, a major consideration (this translates to storage used in a program). In addition, they do not have to be fabricable (but must be simulatable in some appropriate programming language, e.g. OCCAM, CONCURRENT PROLOG), and broadcasting is not to be avoided. Intuitively such an algorithm may not be suitable for chip implementation, but it can be performed on a suitable parallel computing structure.

It is evident that all Hard-systolic algorithms are special cases of Soft-systolic ones and so can also be simulated in the same

\(^1\)If over long distances clock skew occurs and data can become unsynchronized.
programming languages. Further it is also apparent that some Soft-systolic algorithms will be very close to being Hard-systolic, but under the strict definitions would not be purely classed as such, but as Hybrid-systolic algorithms: denoted $S_{HY}$.

Hybrid-systolic algorithms

They will represent a grey area of algorithms being in a state of 'migration' between Soft and Hard and altering technological conditions over time. In specific, algorithms which allow local broadcasting (not necessarily between nearest-neighbour cells), limited non-planarity or large amounts of non-planarity (but in a control sense, with regular connection structures), could be considered as candidates for this category of algorithms.

All the above definitions will become increasingly important as FGCS evolve. The relations between these classes of algorithms, in a set theory manner, are:

1) $S_H \cup S_S = S$: All systolic algorithms
2) $S_H \subseteq S_{HY} \subseteq S_S$

The important question arising is whether $S_H = S_S$, because if this is the case then, all Soft-systolic algorithms can be in essence fabricable.

In the following paragraph a mathematical model for the verification of systolic networks will be introduced (see Melhem and Rheinboldt [MELH84]), along with the data sequences to represent the data appearing on the communication links at successive time intervals, and the causal operators which model the computations performed by a cell of the network. The latter concept was primarily inspired by corresponding approaches in systems theory (see Faurre and Depeyrot [FAUR??]).
VI.A.3: AN ABSTRACT MATHEMATICAL MODEL FOR THE VERIFICATION OF 'SYSTOLIC' NETWORKS

In all theoretical models of VLSI circuits two parameters are of vital importance, size and speed. Since VLSI is essentially two-dimensional, the size of a circuit is best expressed in terms of its area. Sufficient area must be provided in a circuit layout for each gate and each wire. Gates are not allowed to overlap each other at all, and only two (or perhaps three) wires can pass over the same point.

The speed of a synchronous VLSI circuit can be measured by the number of clock pulses it takes to complete its computation. The actual size of this time unit, however, is a technological variable.

The speed of the VLSI circuit may be adversely affected by the presence of very long wires, unless special measures are taken. In many VLSI processes, a minimum-sized transistor cannot send a signal from one end of the chip to the other in one clock period. Today, to accomplish such unit-delay cross-chip communication, and to achieve large fan-outs, special 'driver' (amplifier) circuits are employed.

In general, an efficient systolic array should exhibit a linear-rate pipelinability, i.e., it should achieve \( O(m) \) speed-up, in terms of processing rates, where \( m \) is the number of PE's. The term Efficiency (E) will denote the fraction of processor cycles during which a typical processor is actively employed in the array.

Let us now proceed with the definition of the main elements of the mathematical model for the verification of systolic networks.

- Abstract Model of Data, Causal Relations

We define a data sequence to be an infinite sequence whose elements

---

\(^{\dagger}\)For the superconducting technology of Josephson junctions, a clock period of \(1-3\)ns is achievable today, using a process for which the area unit is \(25\mu \text{m}^2\) (see Ketchen [KETCH80]).
are members of the set \( R_0 = R \cup \{ \delta \} \).

**Notation:**
\[
\begin{align*}
R & \equiv \text{Real numbers} \\
\delta & = \text{'don't care element' (or, 'dummy element')} \notin R.
\end{align*}
\]

We extend any operator defined on \( R \) to \( R_0 \) either:

1. By adding the rule that the result of any operator involving \( \delta \) is \( \delta \) (class of \( \delta \)-regular operators), or
2. by treating \( \delta \) as a special symbol that affects the result of the operation (class of non-\( \delta \)-regular operators).

**Definition:** Operations

1. \( \delta \)-regular operators, e.g., \( \delta \cdot \text{op' } x = x \cdot \text{op' } \delta = \delta \) \( \forall x \in R_0 \),
2. non-\( \delta \)-regular operators, e.g., binary operator \( \oplus \) such that for any \( x, y \in R_0 \), \( x \oplus y = x + y \) if \( x, y \neq \delta \), \( x \oplus \delta = \delta \oplus x = x \).

**Definition:** Let \( N \) be the set of positive integers. Then any data sequence \( \eta \) is defined as a mapping from \( N \) to \( R_0 \); that is, the image element \( \eta(l), l \in N \), is the \( l \)-th element in the sequence. The set of all data sequences, that is the set of all such mappings, will be denoted by \( R^*_0 = \{ \eta | \eta : N \rightarrow R_0 \} \).

**Remark:** Any arithmetic operation on \( R_0 \) is extended to \( R^*_0 \) by applying the operation elementwise to the elements of the sequences, with \( \delta \) being the result of any undefined operation, e.g.,

if 'op' is a binary operation defined on \( R_0 \), then \( \forall \eta_1, \eta_2 \in R^*_0, \eta_1 \cdot \text{op' } \eta_2 = \eta_3 \), where \( \forall l \in N \)

\[
\eta_3(l) = \begin{cases} 
\eta_1(l) \cdot \text{op' } \eta_2(l), & \text{if } \eta_3(l) \text{ is defined}, \\
\delta, & \text{otherwise}.
\end{cases}
\]

**Definitions:**

1. \( d_j \): We can also use scalar operations on sequences, e.g.,

**scalar product:**
for \( \eta \in R_\delta \land \omega \in R_0, \ \zeta = \omega \eta \in R^*_\delta \) for which \( \zeta(1) = \omega(1), \ i \in N \).

\( d_2: \) Bounded Data Sequence set: \( R^* \supseteq R^*_\delta = \{ \text{All sequences having only} \ a \ \text{finite number of non-}\delta\text{-elements} \} \).

\( d_3: \) Termination Function: \( T_f: R_\delta \rightarrow N \) such that, for \( \eta \in R_\delta, T_f(\eta) \) is the position of the last non-\(\delta\)-element in \( \eta \); in other words:

for any \( \eta \in R_\delta, T_f(\eta) = i \iff \eta(1) \neq \delta \) and \( \eta(j) = \delta \) for \( j > i \).

In addition to the operators extended from \( R_\delta \) to \( R^*_\delta \), we may also define operators directly on \( R^*_\delta \).

\( d_4: \) The \( n \)-ary Sequence Operator \( (\Gamma) \): is a transformation

\( \Gamma: (R_\delta)^n \rightarrow R^*_\delta, \) where \( (R_\delta)^n = R_\delta \times R_\delta \times \ldots \times R_\delta \) is the cartesian product space of \( n \) copies of \( R_\delta \).

\( d_5: \) Shift and Spread Operators \( (\Omega^k, \theta^r): \)

\[ \Omega^k \zeta = \eta \text{ and } \theta^r \zeta = \xi, \]

where

\[ \eta(1) = \begin{cases} \delta & \text{if } i \leq k \\ \xi(1-k) & \text{if } i > k, \end{cases} \]

\[ \xi(i) = \begin{cases} \xi(\frac{i+r}{r+1}) & i=1, r+2, 2r+3, \ldots, (n-1)r+n, \ldots, \\ \delta & \text{otherwise}. \end{cases} \]

More descriptively, \( \Omega^k \) inserts \( k \) \( \delta \)-elements at the beginning of a sequence, while \( \theta^r \) inserts \( r \) zero-elements between every two elements of a sequence. For example, if \( \xi = a_1, a_2, a_3, a_4, \delta, \delta, \ldots \), then \( T_f(\xi) = 4 \) and

\[ \xi(1) = a_1, \ \ 1 \leq i \leq T_f(\xi), \]

\[ \Omega^3 \xi = \delta, \delta, \delta, a_1, a_2, a_3, a_4, \delta, \delta, \delta, \ldots, \]

\[ \theta^2 \zeta = a_1, \delta, \delta, a_2, \delta, \delta, a_3, \delta, \delta, a_4, \delta, \delta, \ldots . \]
It is clear that we can define a sequence operator by combining previously defined sequence operators. For example, we might define an operator \( \Gamma: [R_0]^n \rightarrow [R_0] \) as follows:

\[
\Gamma([x, n, \xi]) = \Omega([x+n, \xi]),
\]

where square brackets are used for grouping and parentheses for enclosing the arguments of the operator.

**Causal Operators:** Any n-ary sequence operator \( \Gamma: [R_0]^n \rightarrow [R_0] \), which satisfies the causality property in the sense that the \( i \)-th element of any of its operands can only affect the \( j \)-th element of its image for \( j > i \). More formally, assume that for any \( n_r \in [R_0], r=1,2,\ldots,n \), the image under \( \Gamma \) is \( \xi=r(n_1,...,n_r,...,n_n) \). Then \( \Gamma \) is a causal operator if by replacing any operands \( n_r \) by another sequence \( n'_r \) satisfying

\[
\eta'_r(t) = n_r(t), 1 \leq t < 1,
\]

the resulting image \( \xi'=\Gamma(n_1',...,n_r',...,n_n) \) satisfies

\[
\xi'(t) = \xi(t), 1 \leq t < 1.
\]

Namely, the value of \( \xi(1) \) depends only on the first 1-1 elements of \( n_r, 1 \leq r \leq n \). In the case that the \( i \)-th element of the image sequence \( \xi(1) \) depends only on the first \( i \) elements of the operands \( n_r, 1 \leq r \leq n \), then we are talking about weakly causal operators.

**Abstract Systolic Network Model**

The systolic model is defined to be composed of the following components:

1. A loopless multigraph \( G(V,E,\phi,\phi^+) \), which in turn is composed of
   a) \( V=\{\text{Nodes or cells}\} \)
   b) \( E=\{\text{Directed edges}\} \)
c) two functions $\phi_-, \phi_+: E \rightarrow V$ satisfying the condition that for any (edge) $e \in E$, $\phi_-(e) \neq \phi_+(e)$ (i.e., prevents direct loops). The nodes $\phi_-(e)$ and $\phi_+(e)$ are the 'source' and 'destination' node, respectively, of edge $e \in E$.

The notation $V_S, V_T, V_I$ will be used for the subsets of $V$ defined as:

1) $V_S = \{\text{Source nodes (no edges directed IN)}\}$
2) $V_T = \{\text{Sink nodes (no edges directed OUT)}\}$
3) $V_I = \{\text{Interior nodes (not a source or sink)}\}$;

certainly, the condition $V_S \cup V_T \cup V_I = V$ is always satisfied.

ii) A colouring function $\text{col}: E \rightarrow C_E$, where $C_E$ is a given finite set of colours.

Essentially, input edges to the same node receive different colours, as do output edges, e.g., $y = \text{col}(e)$ denotes edge $e$ has colour $y$.

iii) For each edge $e \in E$, a sequence $\xi_e \in \mathbb{R}_\delta$ is specified.

iv) For each interior node $v \in V$ with IN-degree $n$ and OUT-degree $m$, $n$ causal $m$-ary operators $\Gamma_v^1:\mathbb{R}_\delta^m \rightarrow \mathbb{R}_\delta$ are given, which specify the 'node I/O description'. More specifically, if $\eta^j$, $j=1,2,\ldots,m$, and $\xi^1, i=1,2,\ldots,n$, are the sequences associated with the IN and OUT edges of $v$, respectively, then the $n$ relations

$$\xi_i^1 = \Gamma_v^1(\eta^1, \eta^2, \ldots, \eta^m), i=1,2,\ldots,n,$$

are the I/O description of $v$. The different IN and OUT edges of $v$ are distinguished in the I/O description by their colours.
Remark: Each interior node represents a computational cell\textsuperscript{+}/ processor and each source/sink node corresponds to an input/output cell for the overall network. Each edge $e \in E$ is a unidirectional communication link or channel.\[ 

Finally, given a systolic network based on the graph $G = (V, E, \phi, \phi')$, a subset $V'_I \subseteq V_I$ of interior nodes is said to be a homogeneous set if:

i) All the nodes in $V'_I$ have identical IN- and OUT-degrees, say $m$ and $n$, respectively.

ii) The $m$ colours of the IN edges of any $v \in V'_I$ are identical and so are the $n$ colours of the OUT edges of $v$.

iii) The node I/O descriptions of any $v \in V'_I$ are generic [MELH84].

To conclude, a network is said to be homogeneous if the set of interior nodes $V_I$, in its graph $G$, is a homogeneous set. More generally, a network is said to be $k$-partially homogeneous if there exists a $k$ partition $\bigcup_{i=1}^{k} V^I_i$ of $V_I$ into $k$ non-empty homogeneous subsets $V^I_i$, $i=1,2,\ldots,k$.

\textsuperscript{+}The computations performed by the cells are modelled by a system of difference equations involving operations on the various data sequences. The input/output descriptions, which describe the global effect of the computations performed by the network, are obtained by solving this system of difference equations [MELH84].
VI.A.4: THE DATA STREAM 'ROTATING' AND 'FOLDING' TECHNIQUE

Matrix-Vector Multiplication

Definition of the Problem

Given a \((n \times n)\) matrix \(A\) and a \(n\)-vector \(x\), compute a \(n\)-vector \(y\), from \(Ax = y\). This problem was tackled for a matrix with bandwidth \(w = p + q - 1\) by Leiserson [LEIS81], from which we get the following Theorem.

Theorem [VI.A.4:0.1]

For a \((n \times n)\)-band matrix \(A = (a_{ij})\), of bandwidth \(w\), the elements in the matrix-vector product \(y = (y_1, \ldots, y_n)^T\) can be computed by the following recurrences:

\[
y_1^{(1)} = 0 \\
y_1^{(k+1)} = y_1^{(k)} + a_{i_k}x_k, \quad k = 1, 2, \ldots, n \\
y_1^{(n+1)} = y_1^{(n)}
\]
on a systolic array of \(w\) linearly connected processors, in \(2n + w\) time-units.

As we have previously defined, a time-unit is the cost of one IPS (see Figure (VI.A.4-f1)).

Remark: The efficiency of the linear array of processors is: \(E = 1/2\); in other words, the Efficiency really evaluates the intrinsic degree of parallelism which the array can achieve. □

We shall utilize the same type of IPS as illustrated in Figure (VI.A.4-f1), making use of the 'delays' appearing in Leiserson's implementation. More specifically, in that implementation after \(w\) units (cycles) of time the components of the product \(y = Ax\) start shifting out from the left-end processor in the \(w\)-linear array of processors, at the rate of one output every two units of time. Therefore, the \(O(nw)\) time needed for the sequential algorithm on a uniprocessor was reduced to \(2n + w\).
Figure VI.A.4-f': The Architecture of Leiserson's IPS Processor.

Remark: A good measure for the potential of the systolic array is the well defined by now Speed-up factor, in this case, however, being estimated in terms of processing rates reflecting the pipelining potential of the systolic algorithm. □

To make use of the gaps ('delays') appearing in the data stream of the above implementation we can apply any of the following three ways:
a) Since at any given time alternating processors are 'idle', by 
*coalescing* pairs of adjacent processors it is possible to use \(\frac{w}{2}\) 
processors in the systolic network, for a general banded matrix 
with bandwidth \(w\). This can be achieved by interleaving the diagonals 
of the matrix, while Leiserson's time-units are kept invariant 
(mathematical approach);

b) by using a *double pipe* construction, i.e., by creating a second 
pipeline consisting of the alternative processors in the linear 
array, to eliminate their occurring 'dormant' situations (see 
Robert [ROBE85]) (hardware approach);

c) by *folding* the opposite extreme ends of the matrix band to form a 
dequeue (or deostream, or dewave) with the elements of the lower-half 
of the matrix band interleaved in the previous delay spaces 
(mathematical approach).

A direct comparison of these three approaches, bearing in mind the 
present state of technology, would prove the last approach far superior 
for any type of systolic algorithms, despite the mathematical 
complications occurring, since they can be easily tackled by elementary 
hardware or programming.

In the following we shall exemplify the latter approach, considering 
the above defined problem for the simple case of \(A\) being a tridiagonal 
matrix, i.e., \(p=q=2, \ w=3\).

**Theorem [VI.A.4.3]**

The \((n\times n)\)-band matrix-vector multiplication problem with bandwidth 
\(w=p+q-1\) can be solved in \(n+w+1\) time-units, using Leiserson's linear 
array\(^\dagger\) of \(w\) IPSP's.

\(^\dagger\)With the modifications displayed in Figure (VI.A.4-f2).
Proof:

(By construction of the array). □

Remark: The Efficiency of the linear array of processors is: \( E = 1.0 \)

Consider the tridiagonal matrix \( A \) (for \( n=5 \))

\[
\begin{bmatrix}
\begin{array}{c}
\alpha_{11} & \alpha_{12} \\
\alpha_{21} & \alpha_{22} & \alpha_{23} \\
\alpha_{32} & \alpha_{33} & \alpha_{34} & \alpha_{35} \\
\alpha_{43} & \alpha_{44} & \alpha_{45} \\
\alpha_{54} & \alpha_{55} & \alpha_{51}
\end{array}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{bmatrix} =
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5
\end{bmatrix}
\]

The dequeue resulting when applying the folding technique is the following:

\[
\begin{bmatrix}
\alpha_{33} \\
\alpha_{43} & \delta & \alpha_{34} \\
\alpha_{23} & \alpha_{44} & \alpha_{32} \\
\alpha_{54} & \alpha_{22} & \alpha_{45} \\
\alpha_{12} & \alpha_{55} & \alpha_{21} \\
\alpha_{11}
\end{bmatrix}
\]

notation: \( \delta \) = dummy element

Note that, the middle element of the \( x \)-vector, and consequently the corresponding element in the r.h.s. vector, have to be kept in the

\[\text{This technique is not a simple folding of the band of the matrix, but involves a simultaneous 'rotation' of the off-diagonals.}\]
Figure VI.A.4-f2: The Architecture of Leiserson's Modified IPSP.
respective opposite-end processors for two clock ticks, which can be easily tackled either by programming, or by altering slightly the structure of the IPSP (see Figure (VI.A.4-f2)).

Remark: We have chosen the tridiagonal case for simple exemplification purposes. The reader should bear in mind, however, that as the semi-bandwidths of matrix A increase, irrespective of the size of the matrix, in the middle of the dequeue we have the formation of a full submatrix the size of which depends on the semi-bandwidths (p,q).

Finally, all the computational steps of the systolic algorithm are illustrated in Figure (VI.A.4-f3).

*Figure VI.A.4-f3: The Computational Steps of the Matrix-Vector Multiplication Algorithm (n=5) using a Dequeue.*
VI.A.5: A 'Rotating' AND 'Folding' Algorithm Using A Two-Dimensional 'Systolic' Communication Geometry

How can processors be distributed in a two-dimensional area so that they can be mesh-connected in a simple and regular way, in the sense that the connections are all symmetric and of the same length? This problem is related to that of finding regular figures that can be closely packed to completely cover a two-dimensional area.

It turns out that one of the most preferable solutions to this problem, possessing the above property, is the hexagon. In the following we shall demonstrate a modified rotating and folding algorithm, utilizing such a systolic communication network.

Matrix Multiplication on a Hex-Connected† Systolic Array

Theorem [VI.A.5:θ₁]

Let \( A = (a_{ij}) \) and \( B = (b_{ij}) \) be \((n \times n)\)-band matrices of bandwidths \( w_1 \) and \( w_2 \), respectively. Their product \( C = (c_{ij}) \) can be computed by the following recurrences:

\[
\begin{align*}
  c_{ij}^{(1)} &= 0 \\
  c_{ij}^{(k+1)} &= c_{ij}^{(k)} + a_{ik} b_{kj}, \quad k = 1, 2, \ldots, n \\
  c_{ij} &= c_{ij}^{(n+1)}
\end{align*}
\]

on a systolic network of \( w_1 w_2 \) hex-connected IPSP's, in \( 3n + \min(w_1, w_2) \) time-units (see Leiserson [LEIS81]).

Remark: The Efficiency of the diamond-shaped hexagonal array is: \( E = \frac{1}{3} \); this is due to the fact that in any row or column of the network, out of every three consecutive processors, only one is active at any given time. □

We shall apply the 'rotate' and 'fold' algorithmic concept to this problem, making use of the same systolic network of processors with †

†An abbreviation for Hexagonally mesh-connected.
identical cell specifications. The outline of the IPS hexagonal cell is given in Figure (VI.A.5-f1). To review the general IPS operational procedure irrespectively of the used geometry, each processor has three registers \( R_A, R_B, R_C \) and six external connections, three for input and three for output. In each time-unit interval, the processor shifts the data on its input lines denoted by A, B and C, into \( R_A, R_B \) and \( R_C \), respectively, computes \( R_C + R_C + R_A \times R_B \), and makes the input values for \( R_A \) and \( R_B \), together with the new value of \( R_C \), available as outputs on the output lines denoted by A, B and C respectively.

![Figure VI.A.5-f1: The Outline of the IPS cell in the Hexagonal Geometry.](image)

Again, all outputs are latched and the logic is clocked so that, when one processor is connected to another, the changing output of one during a time-unit interval will not interfere with the input to another during this time interval. This is not, however, the only processing element we shall make use of, but it will be the work horse.

**Comment:** The geometrical type of the IPS cell introduced in (par.VI.A.4) is most suitable to be used for matrix-vector multiplication and the solution of triangular linear systems. The cell geometry introduced
Theorem [VI.A.5:e 2]

The (n×n)-band matrix multiplication problem: C=A×B, with band-widths $w_1$ and $w_2$, respectively, can be solved in $\left\lceil \frac{3n}{2} \right\rceil + \min(w_1, w_2)$ time-units, using Leiserson's systolic network of $w_1w_2$ hex-connected IPSP's.

**Proof:**
(By construction of the array).

**Remark:** The Efficiency (E) of the hexagonal array is not constant, but exhibits a fluctuation between $1/2$ and $2/3$. The reader should bear in mind that these results can be verified by considering the well known Efficiency formula: $E_p = \frac{s}{p \times \eta(e)}$, where $p$ is the number of PE's in the systolic network.

Prior to illustrating the 'rotate' and 'fold' concept, as we have noted, the band-shape of the matrices to be multiplied plays a significant role in the formation of the appropriate dequesues. In particular, the difficulties arise if one of the matrices has not equal semi-bands. Then, the 'rotating' and 'folding' technique must be carefully applied on the elements of those super- or sub-diagonals whose presence alters the band symmetry. We shall examine both the symmetric and unsymmetric semi-band cases by considering appropriate Paradigms.

**Paradigm [VI.A.5:p_{\frac{1}{2}}]**

Let us consider the following (n×n)-band matrix multiplication problem (for $n=5$), which is a symmetric semi-band example: i.e.,

$p_1 = q_1 = p_2 = q_2 = 2, w_1 = w_2 = 3$.

\[ \text{It may change to} \left\lceil \frac{3n}{2} \right\rceil + \min(w_1, w_2) \text{ depending on the band-shape of the matrices to be multiplied.} \]
The *dequeues* resulting when applying the new technique to each of the involved matrices, together with the systolic hexagonal array, are given in *Figure (VI.A.5-2)*. Note that, each $c_{ij}$ is initialized to zero as it enters the network through the bottom boundaries.

**Remark:** The top data stream, in each factor matrix of the multiplication, slightly overlaps the bottom originated data stream for the sake of the tail elements (note the elements in circles). Besides this, each of the dequeues with the data of the factor matrices is identical to the dequeue given for the matrix-vector multiplication problem.

To measure accurately the time-complexity of systolic arrays we should sum all the steps required for loading inputs, processing, and retrieving outputs. For this specific Paradigm the matrix multiplication is computed in $\left\lceil \frac{3n}{2} \right\rceil + \min(3,3) = 10$ time-units, instead of $3n+\min(3,3)=18$ time-units that would be required normally.
Figure VI.A.5-f2: The Dequeues of Data for the Matrix Multiplication Problem on a Hexagonal Systolic Array (for $p_1 = q_1 = p_2 = q_2 = 2$, and $n=5$).
Paradigm [VI.A.5:π]  

Let us now consider the following \((n \times n)\)-band matrix multiplication problem (for \(n=5\)), which is an unsymmetric semi-band example:

\[
\begin{pmatrix}
- \text{p}_1 & \text{p}_2 \\
\text{q}_1 & \end{pmatrix} \quad \begin{pmatrix}
\text{a}_{11} & \text{a}_{12} \\
\text{a}_{21} & \text{a}_{22} & \text{a}_{23} \\
\text{a}_{31} & \text{a}_{32} & \text{a}_{33} & \text{a}_{34} \\
0 & \text{a}_{42} & \text{a}_{43} & \text{a}_{44} & \text{a}_{45} \\
0 & \text{a}_{53} & \text{a}_{54} & \text{a}_{55} \\
\end{pmatrix} \quad \begin{pmatrix}
\text{b}_{11} & \text{b}_{12} & \text{b}_{13} \\
\text{b}_{21} & \text{b}_{22} & \text{b}_{23} & \text{b}_{24} \\
\text{b}_{32} & \text{b}_{33} & \text{b}_{34} & \text{b}_{35} \\
\text{b}_{43} & \text{b}_{44} & \text{b}_{45} \\
\text{b}_{54} & \text{b}_{55} \\
\end{pmatrix} = \begin{pmatrix}
\text{c}_{11} & \text{c}_{12} & \text{c}_{13} & \text{c}_{14} & 0 \\
\text{c}_{21} & \text{c}_{22} & \text{c}_{23} & \text{c}_{24} & \text{c}_{25} \\
\text{c}_{31} & \text{c}_{32} & \text{c}_{33} & \text{c}_{34} & \text{c}_{35} \\
\text{c}_{41} & \text{c}_{42} & \text{c}_{43} & \text{c}_{44} & \text{c}_{45} \\
\text{c}_{52} & \text{c}_{53} & \text{c}_{54} & \text{c}_{55} \\
\end{pmatrix}
\]

In particular for the factor matrices, an 'on-the-fly' modification of the 'rotating' and 'folding' concept is necessary, and this is due to the lack of band symmetry. In respect to the product matrix \(C\), again a slight advancement of the top data stream against the bottom originated one would take place normally, for the sake of the factor dequeses' tail elements.

The dequeses of data resulting when applying this 'on-the-fly' modified/normal 'rotate' and 'fold' to the correspondingly involved matrices, together with the systolic hexagonal array, are given in Figure (VI.A.5-f3). Note that, all the in-between gaps are filled in with 'don't care elements', which for clean diagrammatical purposes have been omitted.

In Figure (VI.A.5-f4) are displayed four consecutive computational steps on this hex-connected systolic array, while the matrix multiplication is computed in \(\left\lfloor \frac{3n}{2} \right\rfloor + \min(4,4) = 12\) time-units in total, instead of \(3n + \min(4,4) = 19\) time-units that would be required normally.

The reader is invited to study the data flow (indicated by 'arrows')
Figure VI.A.5-f:
The Decomposition of Data for the Matrix Multi-
lication Problem on a Hexagonal Systolic Array (for $P_1 = 2$, $Q_1 = 3$, $P_2 = 3$, $Q_2 = 2$, and $n = 5$).

[VI/SEC. A : 670]
continued...→
Figure VI.A.5-f4: Four Consecutive Computational Steps of the Matrix Multiplication Problem of Paradigm [VI.A.5:πₜ].
of this problem more closely by making transparencies of the banded matrices, and moving them over the network picture as previously described.

We shall conclude by introducing the appropriate formulae for the estimation of the (semi)-bandwidth(s) of the product matrix in the matrix multiplication problem.

**Corollary [VI.A.5-ρ₁]**

Given two \((n \times n)\)-band matrices \(A=(a_{ij})\) and \(B=(b_{ij})\) of semi-bandwidths \(p_A, q_A\) and \(p_B, q_B\) respectively, the corresponding semi-bandwidths \(p_C, q_C\) of their product matrix \(C=(c_{ij})\) are estimated by formulae:

\[
\begin{align*}
\text{if } \quad & p_C = p_A + p_B - 1 \\
\text{and } \quad & q_C = q_A + q_B - 1
\end{align*}
\]

which implies that: \(w_C = p_A + q_A + p_B + q_B - 3\).

**Condition**

\(i) \quad p_A \leq n - p_B \)

\(ii) \quad q_A \leq n - q_B\).

**Corollary [VI.A.5-ρ₂]**

In the complementary condition cases:

\(i) \quad p_A > n - p_B \quad \wedge \quad q_A > n - q_B\)

\(ii) \quad p_A > n - p_B \quad \wedge \quad q_A \leq n - q_B\)

\(iii) \quad p_A \leq n - p_B \quad \wedge \quad q_A > n - q_B\),

the bandwidth of the product matrix \(C=(c_{ij})\) is estimated by the corresponding formulae given for each case:

\[
\begin{align*}
\text{if } \quad & w_C = 2n - 1 \\
\text{if } \quad & w_C = n + q_A + q_B - 2 \\
\text{if } \quad & w_C = n + p_A + p_B - 2
\end{align*}
\]

\[(VI.A.5:2)\]
VI.A.6: 'SYSTOLIC' LU-FACTORIZATION DEQUEUES FOR TRIDIAGONAL SYSTEMS

The problem of factoring a matrix $A=(a_{ij})$ into lower and upper triangular matrices $L$ and $U$ (i.e., LU-decomposition) has been proved (see [KUNG80], [LEISS81]) that it can be done naturally on hexagonal systolic arrays. It is assumed, however, that the matrix $A$ has the property that its LU-decomposition can be done by Gaussian elimination without pivoting. This is true, for example, when $A$ is a symmetric positive-definite, or an irreducible, diagonally dominant matrix. Usually this condition is not a problem, since most of the systems encountered in practice are diagonally dominant; on the other hand, however, there is currently no efficient way of incorporating a pivot strategy into existing vector or parallel algorithms.

Once the $L$ and $U$ factor matrices are computed, it is relatively easy to solve the resulting triangular linear systems.

The hex-connected systolic array of processors to implement the LU-decomposition, displayed in Figure (VI.A.6-f1), is constructed as follows. The processors below the upper boundaries are the standard IPSP's and are hex-connected exactly the same as the matrix multiplication computing network presented in (par.-VI.A.5). The processor at the top, denoted by a circle, is a special processor. It computes the reciprocal of its input and pumps the result southwest, and also pumps the same input northwards unchanged. The other processors on the upper boundaries are again IPSP's, but their orientation is changed: the processors on the upper left boundary are rotated 120 degrees clockwise; the processors on the upper right boundary are rotated 120 degrees counterclockwise. Again, the flow of data in the array is indicated by arrows.
Figure VI.A.6-f1: Hexagonal Array of Processors for Pipelining the LU-decomposition of a $(n \times n)$-Band Matrix with Bandwidth $w=7$.

**Remark:** A major part of the LU-decomposition network is formed from the matrix multiplication network due to the similarity of the defining recurrences. In other words, this implies that the rotating and folding technique should be successfully applied on the present problem and systolic network, to increase, in a similar manner, the overall efficiency of the array. □

Prior to proceeding to the formation of the systolic LU-factorization dequeue we shall introduce the following Theorem and Lemma obtained from [LEIS81].

**Theorem [VI.A.6.6]**

Let $A=(a_{ij})$ be a $(n \times n)$-band matrix with bandwidth $w=p+q-1$, then a processor array having no more than $pq$ hex-connected processors can compute the LU-decomposition of $A$ in $3n+\min(p,q)$ time-units.
Lemma [VI.A.6: λ₁]

Let $A=(a_{ij})$ be a $(n \times n)$-dense matrix, then $n^2$ hex-connected processors can compute the $L$ and $U$ matrices in $4n$ time-units.

Remark: It is not to be forgotten that these complexities include I/O, control, and data movement.

When we apply the new 'rotate' and 'fold' concept we obtain two LU-factorization streams functioning concurrently and in opposite directions, one from the top downwards and the other vice versa. Certainly, the two factorization streams should confront each other in the center of the matrix and the degree of difficulty in handling the factorizing procedures in this part is directly dependent upon the size and the semi-bandwidths of the matrix. Herein, we shall exemplify the concept of the systolic LU-factorization dequeue for tridiagonal matrices.

Theorem [VI.A.6:θ₂]

Let $A=(a_{ij})$ be a $(n \times n)$-band matrix with semi-bandwidths $p=q=2$, then by applying the 'rotate' and 'fold' technique the factorization of $A$ into the $L$ and $U$ matrices can be done in $\left\lceil \frac{3n}{2} \right\rceil + \min(p, q)$ time-units, using a hex-connected systolic network of $pq$ processors.

Proof:

(By the construction of the array in the Paradigms which follow).

Paradigm [VI.A.6:π₁]

$n$-odd

Let us consider the following $(n \times n)$ tridiagonal matrix (for $n=9$):

\[\text{The ceiling function is used to take care of the case that } n \text{ is an odd number.}\]
The general recurrences for the pipelined systolic evaluation of the triangular matrices $L=(l_{ij})$ and $U=(u_{ij})$, for any $(n \times n)$-band matrix with bandwidth $w=p+q-1$, are the following:

\[
\begin{align*}
    a_{ij}^{(1)} &= a_{ij} \\
    a_{ij}^{(k+1)} &= a_{ij}^{(k)} + l_{ik} (u_{kj})^{-1} \\
    l_{ik} &= \begin{cases} 0 & \text{if } i < k \\ 1 & \text{if } i = k \\ a_{ik} (u_{kk})_{-1} & \text{if } i > k \end{cases} \\
    u_{kj} &= \begin{cases} 0 & \text{if } k > j \\ a_{kj}^{(k)} & \text{if } k \leq j. \end{cases}
\end{align*}
\]

The resulting matrices for the particular case of $p=q=2$, and $n=9$ have the form
Prior to illustrating the LU-factorization dequeue let us discuss the mathematical alterations imposed on the recurrences (VI.A.6:2) for the application of the 'rotate' and 'fold' technique. In particular, the matrix $A$ will be rewritten as two separate matrices $A_1, A_2$ of half-band length each, i.e., 

$$L = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\ell_{21} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\ell_{32} & \ell_{43} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\ell_{43} & 1 & \ell_{54} & 1 & 0 & 0 & 0 & 0 & 0 \\
\ell_{54} & 1 & \ell_{65} & 1 & 0 & 0 & 0 & 0 & 0 \\
\ell_{65} & 1 & \ell_{76} & 1 & 0 & 0 & 0 & 0 & 0 \\
\ell_{76} & 1 & \ell_{87} & 1 & 0 & 0 & 0 & 0 & 0 \\
\ell_{87} & 1 & \ell_{98} & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix},$$

$$U = \begin{bmatrix}
\begin{array}{cccccccc}
\tilde{u}_{11} & \tilde{u}_{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\tilde{u}_{22} & \tilde{u}_{23} & \tilde{u}_{33} & \tilde{u}_{34} & 0 & 0 & 0 & 0 & 0 \\
\tilde{u}_{33} & \tilde{u}_{34} & \tilde{u}_{44} & \tilde{u}_{45} & \tilde{u}_{55} & \tilde{u}_{56} & 0 & 0 & 0 \\
\tilde{u}_{44} & \tilde{u}_{45} & \tilde{u}_{55} & \tilde{u}_{56} & \tilde{u}_{66} & \tilde{u}_{67} & \tilde{u}_{77} & \tilde{u}_{78} & \tilde{u}_{88} \\
\tilde{u}_{55} & \tilde{u}_{56} & \tilde{u}_{66} & \tilde{u}_{67} & \tilde{u}_{77} & \tilde{u}_{78} & \tilde{u}_{88} & \tilde{u}_{89} & \tilde{u}_{99} \\
\end{array}
\end{bmatrix}. $$


\[ A_1 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} \\ & a_{32} & a_{33} & a_{34} \\ & & a_{43} & a_{44} & a_{45} \\ & & & a_{54} & a_{55} \end{bmatrix} = \begin{bmatrix} 1 \\ \ell_{21} & 1 \\ \ell_{32} & 1 \\ \ell_{43} & 1 \\ \ell_{54} & 1 \end{bmatrix} \]

\[ A_2 = \begin{bmatrix} a_{99} & a_{98} \\ a_{89} & a_{88} & a_{87} \\ a_{78} & a_{77} & a_{76} \\ & a_{67} & a_{66} & a_{65} \\ & & a_{56} & a_{55} \end{bmatrix} = \begin{bmatrix} 1 \\ \ell_{89} & 1 \\ \ell_{78} & 1 \\ \ell_{67} & 1 \\ \ell_{56} & 1 \end{bmatrix} \]

Remark: For this specific example, where the semi-bandwidths are \( p = q = 2 \), we shall use a \((2 \times 2)\) systolic array of hex-connected processors.
The recurrences given in (VI.A.6:2) will be modified to the following:

\[
\begin{align*}
\text{(1)} & \quad a_{ij}^{(1)} = a_{ij} \\
\text{(2)} & \quad a_{ii}^{(2)} = a_{ii}^{(1)} + \ell_{i,i-1}^{(1)} (-u_{i-1,i}) \\
\end{align*}
\]

\[
\ell_{ir} = \begin{cases} 
0 & \text{if } i<r \\
1 & \text{if } i=r \\
a_{ir}^{(1)} u_{rr} & \text{if } i>r 
\end{cases} 
\]

\[
u_{rj} = \begin{cases} 
0 & \text{if } r>j \\
a_{rj}^{(2)} & \text{if } r=j \neq 1 \\
a_{rj}^{(1)} & \text{if } r<j, r=j=1 
\end{cases} 
\]

\text{(1st stream) (VI.A.6:3)}

and

\[
\begin{align*}
\text{(1)} & \quad a_{ij} = a_{ij} \\
\text{(2)} & \quad a_{ii} = a_{ii}^{(1)} + \ell_{i,i+1}^{(1)} (-u_{i+1,i}) \\
\end{align*}
\]

\[
\ell_{ir} = \begin{cases} 
0 & \text{if } i>r \\
1 & \text{if } i=r \\
a_{ir}^{(1)} u_{rr} & \text{if } i<r 
\end{cases} 
\]

\[
u_{rj} = \begin{cases} 
0 & \text{if } r>j \\
a_{rj}^{(2)} & \text{if } r=j \neq n \\
a_{rj}^{(1)} & \text{if } r<j, r=j=n 
\end{cases} 
\]

\text{(2nd stream) (VI.A.6:4)}

\textbf{Comment:} Since \( n \) is chosen to be odd, each stream in the destream procedure modifies the center element of the matrix middle row; and to be more specific (in the above example) the \( a_{55} \) element (only) is modified twice.\footnote{This implies that after its first modification this element has to be collected 'on-the-fly' from the output of the cell and brought back into the serial stream to re-enter that cell (in the same time-unit step) for the next modification.}

Let us now compute the corresponding elements of the resulting triangular matrices, by using the above recurrences for the opposite
factorization streams.

1st Stream

\[
\begin{align*}
\alpha^{(1)}_{1j} &= \alpha_{1j} \\
\bar{u}^{(1)}_{1j} &= \alpha_{1j} \Rightarrow u_{11} = \alpha_{11}, \quad u_{12} = \alpha_{12} \text{ (initially)} \\
\bar{u}^{(2)}_{2j} &= \alpha_{2j} \Rightarrow u_{22} = \alpha_{22} \\
\bar{u}^{(2)}_{3j} &= \alpha_{3j} \Rightarrow u_{33} = \alpha_{33} \\
\bar{u}^{(2)}_{4j} &= \alpha_{4j} \Rightarrow u_{44} = \alpha_{44} \\
\bar{u}^{(2)}_{5j} &= \alpha_{5j} \Rightarrow u_{55} = \alpha_{55} \\
\end{align*}
\]

\[
\begin{align*}
\lambda_{11} &= \alpha^{(1)}_{11} - 1 \Rightarrow \lambda_{21} = \alpha^{(1)}_{21} - 1 \\
\lambda_{12} &= \alpha^{(1)}_{12} - 1 \Rightarrow \lambda_{32} = \alpha^{(1)}_{32} - 1 \\
\lambda_{13} &= \alpha^{(1)}_{13} - 1 \Rightarrow \lambda_{43} = \alpha^{(1)}_{43} - 1 \\
\lambda_{14} &= \alpha^{(1)}_{14} - 1 \Rightarrow \lambda_{54} = \alpha^{(1)}_{54} - 1 \\
\end{align*}
\]

\[
\begin{align*}
\alpha^{(2)}_{11} &= \alpha^{(1)}_{11} + \lambda_{11} ( -u_{11} ) \Rightarrow \alpha^{(2)}_{22} = \alpha^{(1)}_{22} + \lambda_{21} ( -u_{12} ) \\
\alpha^{(2)}_{11} &= \alpha^{(1)}_{11} + \lambda_{12} ( -u_{21} ) \Rightarrow \alpha^{(2)}_{33} = \alpha^{(1)}_{33} + \lambda_{32} ( -u_{23} ) \\
\alpha^{(2)}_{11} &= \alpha^{(1)}_{11} + \lambda_{13} ( -u_{31} ) \Rightarrow \alpha^{(2)}_{44} = \alpha^{(1)}_{44} + \lambda_{43} ( -u_{34} ) \\
\alpha^{(2)}_{11} &= \alpha^{(1)}_{11} + \lambda_{14} ( -u_{41} ) \Rightarrow \alpha^{(2)}_{55} = \alpha^{(1)}_{55} + \lambda_{54} ( -u_{45} ).
\end{align*}
\]

2nd Stream

\[
\begin{align*}
\alpha^{(1)}_{1j} &= \alpha_{1j}, \quad \alpha^{(3)}_{1j} = \alpha_{1j} \\
\bar{u}^{(1)}_{9j} &= \alpha_{9j} \Rightarrow u_{99} = \alpha_{99}, \quad u_{98} = \alpha_{98} \text{ (initially)} \\
\bar{u}^{(2)}_{8j} &= \alpha_{8j} \Rightarrow u_{88} = \alpha_{88} \\
\bar{u}^{(2)}_{7j} &= \alpha_{7j} \Rightarrow u_{77} = \alpha_{77} \\
\bar{u}^{(2)}_{6j} &= \alpha_{6j} \Rightarrow u_{66} = \alpha_{66} \\
\bar{u}^{(2)}_{5j} &= \alpha_{5j} \Rightarrow u_{55} = \alpha_{55} \\
\end{align*}
\]

\[
\begin{align*}
\bar{u}^{(2)}_{8j} &= \alpha_{8j} \Rightarrow u_{88} = \alpha_{88} \\
\bar{u}^{(2)}_{7j} &= \alpha_{7j} \Rightarrow u_{77} = \alpha_{77} \\
\bar{u}^{(2)}_{6j} &= \alpha_{6j} \Rightarrow u_{66} = \alpha_{66} \\
\bar{u}^{(2)}_{5j} &= \alpha_{5j} \Rightarrow u_{55} = \alpha_{55} \\
\end{align*}
\]
\[
\begin{align*}
\ell_{19} &= a_{19}^{(1)} - 1, \quad \ell_{89} = a_{89}^{(1)} - 1, \\
\ell_{18} &= a_{18}^{(1)} - 1, \quad \ell_{78} = a_{78}^{(1)} - 1, \\
\ell_{17} &= a_{17}^{(1)} - 1, \quad \ell_{67} = a_{67}^{(1)} - 1, \\
\ell_{16} &= a_{16}^{(1)} - 1, \quad \ell_{56} = a_{56}^{(1)} - 1.
\end{align*}
\]

\[(L_2)\]

\[
\begin{align*}
a_{11}^{(2)} &= a_{11}^{(1)} + \ell_{19}^{(-u_{91})} = a_{88}^{(1)} + \ell_{89}^{(-u_{98})}, \\
a_{11}^{(2)} &= a_{11}^{(1)} + \ell_{18}^{(-u_{81})} = a_{77}^{(1)} + \ell_{78}^{(-u_{87})}, \\
a_{11}^{(2)} &= a_{11}^{(1)} + \ell_{17}^{(-u_{71})} = a_{66}^{(1)} + \ell_{67}^{(-u_{76})}, \\
a_{11}^{(2)} &= a_{11}^{(1)} + \ell_{16}^{(-u_{61})} = a_{55}^{(1)} + \ell_{56}^{(-u_{65})}. \\
\end{align*}
\]

The Efficiency achieved in [KUNG80] from the single stream LU-decomposition scheme is: \(E = 1/3\), since in any row or column of the hex-connected systolic array, similar to the matrix multiplication case, only one out of every three consecutive processors is active at a given time.

The dequeue of data resulting when applying the 'rotate' and 'fold' concept to the two half-A matrices, to take advantage of the processors 'dormant' instances occurring in the above implementation, together with the systolic array, are given in Figure (VI.A.6-f2).

**Remark:** The Efficiency achieved has now been increased to \(E = 1/2\), whereas the number of IPS cells is the same as for the usual LU-decomposition array.

For exemplary purposes of the new concept, in Figure (VI.A.6-f3) are displayed all the computational steps required on this hex-connected systolic array of processors. For this specific Paradigm the LU-factorization is computed in \(\left\lceil \frac{3n}{2} \right\rceil + \min(2,2) = 16\) time-units, instead of \(3n + \min(2,2) = 29\) time-units that would be required normally.
Figure VI.A.6-f2: The Dequeue of Data for the LU-factorization on a Hexagonal Systolic Array (for p=q=2, and n=9).
The prime denotes that this element has been modified by the previous stream.

Figure VI.A.6-f3: All the Computational Steps of the LU-factorization of Paradigm [VI.A.6:π₁].
Again, the reader is invited to verify the computational snapshots given in the above Figure by making transparencies either of the compound data stream, or the network itself, and moving them one over the other appropriately.

Paradigm [VI.A.6:π₂]

n-even

Let us consider the following \((nxn)\) tridiagonal matrix (for \(n=4\)):

\[
A = \begin{bmatrix}
 a_{11} & a_{12} & 0 \\
 a_{21} & a_{22} & a_{23} \\
 0 & a_{32} & a_{33} & a_{34} \\
 0 & 0 & a_{43} & a_{44}
\end{bmatrix}.
\]

The matrix \(A\) will be rewritten as two separate matrices \(A_1, A_2\) one of which is greater than a half-band length, i.e.,

\[
A_1 = \begin{bmatrix}
 a_{11} & a_{12} & 0 \\
 a_{21} & a_{22} & a_{23} \\
 0 & a_{32} & a_{33}
\end{bmatrix} = \begin{bmatrix}
 1 & 0 \\
 z_{21} & 1 \\
 0 & 1
\end{bmatrix} \begin{bmatrix}
 u_{11} & u_{12} & 0 \\
 0 & u_{22} & u_{23} \\
 0 & 0 & u_{33}
\end{bmatrix}
\]

\[
A_2 = \begin{bmatrix}
 a_{34} & a_{43} \\
 a_{34} & a_{33}
\end{bmatrix} = \begin{bmatrix}
 1 & 0 \\
 z_{34} & 1
\end{bmatrix} \begin{bmatrix}
 u_{44} & u_{43} \\
 0 & u_{33}
\end{bmatrix}.
\]

Remark: Again, we shall use a \((2\times2)\) systolic array of hex-connected processors. □

The recurrences that we shall use in this case are:
Let us now compute the corresponding elements of the resulting triangular matrices, by using the above recurrences for the opposite factorization streams.

**1st Stream**

\[
\begin{align*}
\alpha_{ij}^{(1)} &= \alpha_{ij}, \quad \alpha_{ii}^{(3)} = \alpha_{11}^{(2)} \\
\alpha_{ii}^{(2)} &= \alpha_{ii}^{(1)} + \xi_{i,i-1}(-u_{i-1,i}) \\
\xi_{ir} &= \begin{cases} 
0 & \text{if } i < r \\
1 & \text{if } i = r \\
(1)_{ir}u_{rr}^{-1} & \text{if } i > r
\end{cases} \\
\mu_{rj} &= \begin{cases} 
0 & \text{if } r > j \\
(2)_{rj} & \text{if } r = j \\
(1)_{rj} & \text{if } r < j, r = j = n
\end{cases}
\end{align*}
\]

and

\[
\begin{align*}
\alpha_{ij}^{(1)} &= \alpha_{ij}^{(1)} \\
\alpha_{ii}^{(2)} &= \alpha_{ii}^{(1)} + \xi_{i,i+1}(-u_{i+1,i}) \\
\xi_{ir} &= \begin{cases} 
0 & \text{if } i > r \\
1 & \text{if } i = r \\
(1)_{ir}u_{rr}^{-1} & \text{if } i < r
\end{cases} \\
\mu_{rj} &= \begin{cases} 
0 & \text{if } r < j \\
(2)_{rj} & \text{if } r = j \\
(1)_{rj} & \text{if } r > j, r = j = n
\end{cases}
\end{align*}
\]

Let us now compute the corresponding elements of the resulting triangular matrices, by using the above recurrences for the opposite factorization streams.
The dequeue of data resulting when applying the 'rotate' and 'fold' concept to the matrices $A_1$ and $A_2$, to benefit from the processors inactive instances occurring in the single LU-decomposition stream, together with the systolic array, are given in Figure (VI.A.6-f4).

**Remark:** The Efficiency achieved has again been increased to: $E=1/2$, whereas the number of IPS cells is the same as for the usual LU-decomposition array.

Again, for exemplary purposes of the 'rotate' and 'fold' concept for this case, in Figure (VI.A.6-f5) are displayed all the computational steps required on this hex-connected systolic array of processors. For this specific Paradigm the LU-factorization is computed in $\frac{3n}{2}\min(2,2)+1=9$ time-units, instead of $3n+\min(2,2)=14$ time-units that would be required normally.

**Comment:** Note that, the $a_{33}$ element after its first modification has
to be collected 'on-the-fly' from the output of the cell and brought back into the serial stream to re-enter that cell, with a time-unit delay, for the next modification by the 1st wave.

For a better assistance in understanding the 'rotate' and 'fold' concept, in Appendix C-VI, the mathematical background of the double Gaussian elimination streams\(^+\) is introduced, accompanied by an appropriate general numerical example, i.e. for a full \((n \times n)\) matrix, to demonstrate the efficiency of the method.

For the specific instance of the present case, however, let us present a simple numerical example illustrating this necessary 'on-the-fly' modification discussed above.

\(^+\)For full matrices.
continued...
The prime denotes that this element has been modified by the previous stream.
Figure VI.A.6-f5: All the Computational Steps of the LU-factorization of Paradigm [VI.A.6:n3].
Consider the following (4x4) tridiagonal linear system:

\[
\begin{bmatrix}
2 & 1 & 0 & \circ \\
1 & 3 & 1 & 0 \\
-1 & 2 & 1 & \circ \\
\circ & 1 & 2 & \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
=
\begin{bmatrix}
3 \\
5 \\
2 \\
3 \\
\end{bmatrix}
\]  
(VI.A.6:8)

Notice that, the coefficient matrix is diagonally dominant.

By applying the folding algorithmic process we obtain the following solution steps:

\[
\begin{bmatrix}
2 & 1 & 0 & \circ \\
1 & 3 & 1 & 0 \\
-1 & 2 & 1 & \circ \\
\circ & 1 & 2 & \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
=
\begin{bmatrix}
3 \\
5 \\
2 \\
3 \\
\end{bmatrix}
\]

multipliers (top-bottom streams)

\[m_t \quad m_b\]

\[\begin{bmatrix}
5/2 & 1 & 0 \\
-1 & 3/2 & \circ \\
\circ & 1 & 2 \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
\]

\[1^{st \ step}\]

\[\begin{bmatrix}
2 & 1 & 0 & \circ \\
1 & 3 & 1 & 0 \\
-1 & 2 & 1 & \circ \\
\circ & 1 & 2 & \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
\]

\[\begin{bmatrix}
3 \\
5 \\
2 \\
3 \\
\end{bmatrix}
\]

\[\begin{bmatrix}
5/2 & 1 & 0 \\
-1 & 3/2 & \circ \\
\circ & 1 & 2 \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
\]

\[2/5 \quad 2/5\]

\[\begin{bmatrix}
2 & 1 & 0 & \circ \\
1 & 3 & 1 & 0 \\
-1 & 2 & 1 & \circ \\
\circ & 1 & 2 & \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
\]

\[\begin{bmatrix}
3 \\
7/2 \\
1/2 \\
3 \\
\end{bmatrix}
\]

The circle and square schemes underline the modification sequence of that element by the bottom and top streams, respectively. The solution vector \(x\) was pre-arranged to be \(x=(1,1,1,1)^T\). Further, note that, since \(n\) is even the folding procedure ends-up with the formation of a
(2x2) central submatrix. On this matrix, only the top stream eliminates the corresponding element and hence the backward, forward substitutions start from unsymmetric matrix positions.

To conclude, in respect to the utilized systolic array for the Paradigms herein, there are several other equivalent networks that reflect only minor changes compared with it. For example, the elements of L and U can be retrieved as output in a number of different ways. Also, the '-1' input to the network can be changed to a '+1' if the special processor at the top of the network computes minus the reciprocal of its input.

VI.A.6.1: 'DEQUEUES' FOR SOLVING TRIANGULAR LINEAR SYSTEMS

This paragraph concerns itself with the solution of the corresponding triangular linear systems resulting from the factorization of the numerical examples of Paradigms [VI.A.6: π₁, π₂], by applying the 'rotate' and 'fold' concept again, to obtain a similar increased efficiency as that for the factorizing part.

Suppose that we want to solve a linear system: Ax = b. In fact, after the LU-decomposition we have to solve two triangular linear systems:

\[
\begin{align*}
Ly &= b \\
Ux &= y
\end{align*}
\]

(VI.A.6;1)

An upper triangular linear system can always be rewritten as a lower triangular linear system without any loss of generality.

Herein, we shall investigate both problems individually, making use of the same systolic network introduced in [KUNG80]; but, instead of a single data stream we shall form an appropriate dequeue for each of the problems above, to solve two triangular linear systems in the
same amount of time required for the solution of one in the above implementation.

Further, note that, we shall tackle both these problems for the correspondingly resulting triangular linear systems of the above Paradigms, in order to cover the instances of \( n \) being odd and even.

**CASE A - (of Paradigm [VI.A.6: \( \pi_1 \)])**

*(i) - Lower Triangular Linear Systems*

Let \( A = (a_{i,j}) \) be a non-singular \((9 \times 9)\)-band lower triangular matrix. Suppose that \( A \) and vector \( b = (b_1, \ldots, b_9)^T \) are given. The problem is to compute \( x = (x_1, \ldots, x_9)^T \) such that: \( Ax = b \). The vector \( x \) can be computed by the following recurrences:

\[
\begin{align*}
y_1^{(1)} &= 0 \\
y_2^{(2)} &= y_1^{(1)} + a_{1,1}x_1 = a_{1,1}x_1 \\
x_1 &= (b_1 - y_1^{(2)})/a_{11} \quad \text{(when } 1=1, \text{ then } y_1^{(2)} = y_1^{(1)}). 
\end{align*}
\]

More specifically, let us consider the lower triangular linear system and compute the solution vector \( x \), using the above recurrences, i.e.,

\[
\begin{bmatrix}
a_{11} & a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & a_{76} & a_{87} & a_{98} \\
a_{22} & a_{33} & a_{44} & a_{55} & a_{66} & a_{77} & a_{88} & a_{99} \\
a_{32} & a_{43} & a_{54} & a_{65} & a_{76} & a_{87} & a_{98} & \\
a_{44} & a_{55} & a_{66} & a_{77} & a_{88} & a_{99} & \\
a_{54} & a_{65} & a_{76} & a_{87} & a_{98} & & \\
a_{65} & a_{76} & a_{87} & a_{98} & & & \\
a_{76} & a_{87} & a_{98} & & & & & \\
a_{87} & a_{98} & & & & & & & \\
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
x_8 \\
x_9 \\
\end{bmatrix} = \begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5 \\
b_6 \\
b_7 \\
b_8 \\
b_9 \\
\end{bmatrix}. \quad \text{(VI.A.6.1:2)}
\]
The solution will be:

\[
\begin{align*}
x_1 &= b_1/a_{11} \\
a_{21}x_1 + a_{22}x_2 &= b_2 \iff x_2 = (b_2 - a_{21}x_1)/a_{22} \\
a_{32}x_2 + a_{33}x_3 &= b_3 \iff x_3 = (b_3 - a_{32}x_2)/a_{33} \\
a_{43}x_3 + a_{44}x_4 &= b_4 \iff x_4 = (b_4 - a_{43}x_3)/a_{44} \\
a_{54}x_4 + a_{55}x_5 &= b_5 \iff x_5 = (b_5 - a_{54}x_4)/a_{55} \\
a_{65}x_5 + a_{66}x_6 &= b_6 \iff x_6 = (b_6 - a_{65}x_5)/a_{66} \\
a_{76}x_6 + a_{77}x_7 &= b_7 \iff x_7 = (b_7 - a_{76}x_6)/a_{77} \\
a_{87}x_7 + a_{88}x_8 &= b_8 \iff x_8 = (b_8 - a_{87}x_7)/a_{88} \\
a_{98}x_8 + a_{99}x_9 &= b_9 \iff x_9 = (b_9 - a_{98}x_8)/a_{99}
\end{align*}
\]

In this case (i.e., tridiagonal matrix), the bandwidth of the matrix is \( w=q=2 \). The above given recurrences, in (VI.6.1:1), can be evaluated by the algorithm and network almost identical to those used for the band matrix-vector multiplication problem. The outline of this systolic network is illustrated in Figure (VI.6.1-f1) further on.

On this network, the \( y_i \), which are initially zero, move leftwards through the network, while the \( x_i, a_{ii} \), and \( b_1 \) are moving as indicated in Figure (VI.6.1-f1). The left-end processor is special in that it performs \( x_1 = (b_1 - y_1)/a_{11} \). In fact, the special processor introduced in the LU-decomposition problem is a special case of this more general processor.

Each \( y_1 \) accumulates an inner product term in the square processor as it moves to the left. At the time \( y_1 \) reaches the left-end processor, it has, in general, the value \( a_{11}x_1 + a_{12}x_2 + \ldots + a_{i-1,i-1}x_{i-1} \), and consequently the \( x_1 \) computed by the formula above at the processor will have the correct value. From [KUNG80] we obtain the following Theorem.
Theorem [VI.A.6.1: 61]

Let \( A=(a_{ij}) \) be a non-singular \((n \times n)\)-band lower triangular matrix of bandwidth \( w=q \). Suppose that \( A \) and an \( n \)-vector \( b=(b_1, \ldots, b_n)^T \) are given. Then, an \( n \)-vector \( x=(x_1, \ldots, x_n)^T \) such that: \( Ax=b \), can be computed in \( 2n+q \) time-units on a linearly connected systolic network of \( w \) IPSP's.

Again, we shall take advantage from the fact that the number of processors required by the network can be reduced to \( w/2 \). In particular, we shall make use of the gaps (i.e., 'idle' processors) by coalescing with the first stream of data, a second stream, thus halving the total solution time.

More specifically, we have to solve two lower triangular linear systems, i.e.,

\[
\begin{bmatrix}
   a_{11} & a_{21} & a_{32} & a_{33} & a_{43} & a_{44} & a_{54} & a_{55} \\
   a_{21} & a_{22} & a_{32} & a_{33} & a_{43} & a_{44} & a_{54} & a_{55} \\
   & a_{32} & a_{33} & a_{44} & a_{54} & a_{55} \\
   & & a_{44} & a_{54} & a_{55} \\
   & & & a_{54} \\
   & & & & a_{55}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
x_8 \\
x_9
\end{bmatrix}
= \begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5 \\
b_6 \\
b_7 \\
b_8 \\
b_9
\end{bmatrix}
\]

(Resulting from the 1st stream of the LU-decomposition)

and

\[
\begin{bmatrix}
   a_{99} & a_{89} & a_{78} & a_{77} & a_{67} & a_{66} & a_{56} & a_{55} \\
   a_{89} & a_{88} & a_{78} & a_{77} & a_{67} & a_{66} & a_{56} & a_{55} \\
   & a_{78} & a_{77} & a_{66} & a_{56} & a_{55} \\
   & & a_{67} & a_{66} & a_{56} & a_{55} \\
   & & & a_{66} & a_{56} & a_{55} \\
   & & & & a_{56} \\
   & & & & & a_{55} \\
   & & & & & & a_{55}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
x_8 \\
x_9
\end{bmatrix}
= \begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5 \\
b_6 \\
b_7 \\
b_8 \\
b_9
\end{bmatrix}
\]

(Resulting from the 2nd stream of the LU-decomposition)
The recurrence formulae for the system: $A_1 x_1 = b_1$ will be:

$$y_1^{(1)} = 0$$

$$y_1^{(2)} = y_1^{(1)} + a_{1,i-1} x_{i-1} = a_{1,i-1} x_{i-1}$$

$$x_1 = (b_1 - y_1^{(2)}) / a_{1i}, \, (\text{when } i=1=y_1^{(2)} = y_1^{(1)})$$

(1st stream)

and for the system: $A_2 x_2 = b_2$:

$$y_1^{(1)} = 0$$

$$y_1^{(2)} = y_1^{(1)} + a_{1,i+1} x_{i+1} = a_{1,i+1} x_{i+1}$$

$$x_1 = (b_1 - y_1^{(2)}) / a_{1i}, \, (\text{when } i=n=y_n^{(2)} = y_n^{(1)})$$

(2nd stream)

Let us now compute the solution for the above systems, so later, in the step-by-step analysis, we shall be able to follow the results in each step.

1st Stream

: Solution for the system: $A_1 x_1 = b_1$

$$x_1 = b_1 / a_{11}$$

$$a_{21} x_1 + a_{22} x_2 = b_2 \quad \Rightarrow \quad x_2 = (b_2 - a_{21} x_1) / a_{22}$$

$$a_{32} x_2 + a_{33} x_3 = b_3 \quad \Rightarrow \quad x_3 = (b_3 - a_{32} x_2) / a_{33}$$

$$a_{43} x_3 + a_{44} x_4 = b_4 \quad \Rightarrow \quad x_4 = (b_4 - a_{43} x_3) / a_{44}$$

$$a_{54} x_4 + a_{55} x_5 = b_5 \quad \Rightarrow \quad x_5 = (b_5 - a_{54} x_4) / a_{55}$$

2nd Stream

: Solution for the system: $A_2 x_2 = b_2$

$$x_9 = b_9 / a_{99}$$

$$a_{99} x_9 + a_{88} x_8 = b_8 \quad \Rightarrow \quad x_8 = (b_8 - a_{89} x_9) / a_{88}$$

$$a_{79} x_9 + a_{77} x_7 = b_7 \quad \Rightarrow \quad x_7 = (b_7 - a_{78} x_8) / a_{77}$$

$$a_{67} x_7 + a_{66} x_6 = b_6 \quad \Rightarrow \quad x_6 = (b_6 - a_{67} x_7) / a_{66}$$

$$a_{56} x_6 + a_{55} x_5 = b_5 \quad \Rightarrow \quad x_5 = (b_5 - a_{56} x_6) / a_{55}$$
Theorem [VI.A.6.1: \( \theta_2 \)]

An \((n \times n)\)-band lower triangular linear system: \(Ax=b\), with coefficient matrix bandwidth \(w=q=2\), can be solved in \(n+2p+q\) time-units applying the 'rotate' and 'fold' concept on a linearly connected systolic network of \(w\) IPSP's.

**Proof:**

(See experimental results of Figure (VI.A.6.1-f2).)

The dequeue of data resulting when applying the above concept to the two half-parts \(A_1\) and \(A_2\) of the original matrix \(A\), together with the systolic array, are given in Figure (VI.A.6.1-f1).

**Remark:** The Efficiency achieved has been increased from: \(E=1/2\) to 1, whereas the number of IPSP's is the same as in [KUNG80].

In Figure (VI.A.6.1-f2) are illustrated all the computational steps required on this linear systolic network of processors. Note from this Figure that the common elements to both streams, denoted by circles in the Figure (VI.A.6.1-f1), are subject to special handling. For this specific Paradigm, the solution of the lower triangular linear system is computed in 13 time-units, since \((n,p)\) are \((odd,even)\), instead of 20 time-units that would be required normally. It is obvious that the new technique, for \(n\) very large, pipelines the solution twice as fast.

**CASE B - (of Paradigm [VI.A.6: \( \pi_1 \)])**

(ii) - Upper Triangular Linear Systems

As we have mentioned, an upper triangular linear system can be rewritten as a lower triangular linear system and then solved using the

\[\text{Note that, } p \text{ is either } p-1 \text{ or } p \text{ (} p \text{ is the semi-bandwidth of the original matrix) depending on the combination (odd/even) of } (n,p). \text{ The related theory is given in Section B.} \]

\[\text{Hence, } p=p-1.\]
Figure VI.A.6.1-f1: The Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.A.6: π₁] on the Linearly Connected Systolic Array (for \(w=2\)).

following recurrence formulae which consist of a generalization of the formulae (VI.A.6.1:3,4), respectively:

For the system: \(A_1 x_1 = b_1\)
1. \( x_1 = \frac{b_1 - y_1}{a_{11}} \) (since \( y_1 = 0 \)).

2. \( y_2 = a_{21} x_1 \cdot x_2 = \frac{b_2 - y_2}{a_{22}} = \frac{(b_2 - a_{21} x_1)}{a_{22}} \). x_1 is output.

3. \( y_3 = a_{32} x_2 \). x_3 is output.

4. \( y_4 = a_{43} x_3 \). x_4 is output.

5. continued...
Figure VI.A.6.1-f2: All the Computational Steps of the Solution of the Lower Triangular Linear System of Paradigm [VI.A.6: $\pi_1$].
\((k)\)
\[ y^{(k)}_1 = 0 \]
\[ y^{(k+1)}_1 = y^{(k)}_1 + a_{1k} x_k \]
\[ x_1 = \frac{b_1 - y^{(1)}_1}{a_{11}} \]

and

for the system: \( A_2 x_2 = b_2 \)

\[ y^{(k)}_t = 0 \]
\[ y^{(k+1)}_t = y^{(k)}_t + a_{tr} x_r \]
\[ x_t = \frac{b_t - y^{(1)}_t}{a_{tt}} \]

where
\[ t = n-1+1 \]
\[ r = n-k+1 \]

In particular, the two upper systems rewritten as lower triangular linear systems to be solved concurrently by the two opposite elimination streams are:

\[
\begin{bmatrix}
  a_{55} & a_{45} & 0 \\
  a_{44} & a_{34} & a_{33} \\
  0 & a_{23} & a_{22} \\
  0 & 0 & a_{12} \\
  0 & 0 & 0 & a_{11}
\end{bmatrix}
\begin{bmatrix}
  x_5 \\
  x_4 \\
  x_3 \\
  x_2 \\
  x_1
\end{bmatrix}
= \begin{bmatrix}
  b_5 \\
  b_4 \\
  b_3 \\
  b_2 \\
  b_1
\end{bmatrix}
\]

(Resulting from the 1st stream of the LU-decomposition)

and

\[
\begin{bmatrix}
  a_{55} & a_{65} & 0 \\
  a_{66} & a_{76} & a_{77} \\
  0 & a_{87} & a_{88} \\
  0 & 0 & a_{98} \\
  0 & 0 & 0 & a_{99}
\end{bmatrix}
\begin{bmatrix}
  x_5 \\
  x_4 \\
  x_3 \\
  x_2 \\
  x_1
\end{bmatrix}
= \begin{bmatrix}
  b_5 \\
  b_4 \\
  b_3 \\
  b_2 \\
  b_1
\end{bmatrix}
\]

(Resulting from the 2nd stream of the LU-decomposition)
Let us now compute the solution vector for the above systems, so that we can follow the results given in the subsequent step-by-step systolic computation.

1\textsuperscript{st} Stream

: Solution for the system: $A_1x_1 = b_1$

$x_5 = b_5/a_{55}$

$\begin{align*}
  a_{45}x_5 + a_{44}x_4 &= b_4 \quad \Rightarrow \quad x_4 = \frac{(b_4-a_{45}x_5)}{a_{44}} \\
  a_{34}x_4 + a_{33}x_3 &= b_3 \quad \Rightarrow \quad x_3 = \frac{(b_3-a_{34}x_4)}{a_{33}} \\
  a_{23}x_3 + a_{22}x_2 &= b_2 \quad \Rightarrow \quad x_2 = \frac{(b_2-a_{23}x_3)}{a_{22}} \\
  a_{12}x_2 + a_{11}x_1 &= b_1 \quad \Rightarrow \quad x_1 = \frac{(b_1-a_{12}x_2)}{a_{11}}
\end{align*}$

2\textsuperscript{nd} Stream

: Solution for the system: $A_2x_2 = b_2$

$x_5 = b_5/a_{55}$

$\begin{align*}
  a_{65}x_5 + a_{66}x_6 &= b_6 \quad \Rightarrow \quad x_6 = \frac{(b_6-a_{65}x_5)}{a_{66}} \\
  a_{76}x_6 + a_{77}x_7 &= b_7 \quad \Rightarrow \quad x_7 = \frac{(b_7-a_{76}x_6)}{a_{77}} \\
  a_{87}x_7 + a_{88}x_8 &= b_8 \quad \Rightarrow \quad x_8 = \frac{(b_8-a_{87}x_7)}{a_{88}} \\
  a_{98}x_8 + a_{99}x_9 &= b_9 \quad \Rightarrow \quad x_9 = \frac{(b_9-a_{98}x_8)}{a_{99}}
\end{align*}$

The dequeue of data resulting when applying the 'rotate' and 'fold' concept to the two half-parts $A_1$ and $A_2$ of the original matrix $A$, together with the systolic array, are given in Figure (VI.A.6.1-f3).

The remark made in the previous case, about the Efficiency achieved, applies to the present case as well.

Finally in Figure (VI.A.6.1-f4) are illustrated all the computational steps required on this linear systolic network of processors.

Again, the common elements to both streams, denoted by circles in Figure (VI.A.6.1-f3), are subject to special handling. The solution was obtained, as expected, in 13 time-units\(^\dagger\).

\(^\dagger\)In fact, it can be obtained in even fewer time-units.
Figure VI.A.6.1-f3: The Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm [VI.A.6:π₁] on the Linearly Connected Systolic Array (for \( w=q=2 \)).
1. 
\[ y_4 = \frac{a_{45} x_5}{a_{65}} \]
\[ x_5 = \frac{(b_5 - y_5)}{a_{55}} = \frac{b_5}{a_{55}} \text{(since } y_5 = 0) \]

2. 
\[ y_3 = a_{34} x_4 \]
\[ x_4 = \frac{(b_4 - y_4)}{a_{44}} = \frac{(b_4 - a_{45} x_5)}{a_{44}} \]
\[ x_5 \text{ is output.} \]

3. 
\[ y_6 = a_{65} x_5 \]
\[ x_6 = \frac{(b_6 - y_6)}{a_{66}} = \frac{(b_6 - a_{65} x_5)}{a_{66}} \]
\[ x_5 \text{ is output.} \]

4. 
\[ y_5 \text{ enters processor 2.} \]
\[ y_5 \]

5. 
\[ y_5 \text{ enters processor 2.} \]
\[ y_5 \]

continued...
9. continued...
Figure VI.A.6.1-f4: All the Computational Steps of the Solution of the Upper Triangular Linear System of Paradigm [VI.A.6: π₁].

11. \( x_9 = \frac{(b_9 - y_9)}{a_{99}} = \frac{(b_9 - a_{98}x_8)}{a_{99}} \). 
   \( x_8 \) is output.

12. \( x_1 \) is output.

13. \( x_9 \) is output.
CASE A - (of Paradigm [VI.A.6:π2])

(6) Lower Triangular Linear Systems

The lower triangular linear systems to be solved are the following:

\[
\begin{bmatrix}
a_{11} & 0 \\
a_{21} & a_{22} \\
0 & a_{32} & a_{33}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix} =
\begin{bmatrix}
b_1 \\
b_2 \\
b_3
\end{bmatrix}
\]

(Resulting from the 1\textsuperscript{st} stream of the LU-decomposition)

and

\[
\begin{bmatrix}
a_{44} & 0 \\
a_{34} & a_{33}
\end{bmatrix}
\begin{bmatrix}
x_4 \\
x_3
\end{bmatrix} =
\begin{bmatrix}
b_4 \\
b_3
\end{bmatrix}
\]

(Resulting from the 2\textsuperscript{nd} stream of the LU-decomposition)

The recurrence solution formulae for these systems are those given in (VI.A.6.1:3,4), or the general ones in (VI.A.6.1:5,6), respectively.

The corresponding solution for each of these systems is:

\textbf{1\textsuperscript{st} Stream}

: Solution for the system: \( A_1 x_{1-1} = b_1 \)

\[
x_1 = b_1 / a_{11}
\]

\[
a_{21} x_1 + a_{22} x_2 = b_2 \quad \Rightarrow x_2 = (b_2 - a_{21} x_1) / a_{22}
\]

\[
a_{32} x_2 + a_{33} x_3 = b_3 \quad \Rightarrow x_3 = (b_3 - a_{32} x_2) / a_{33}
\]

\textbf{2\textsuperscript{nd} Stream}

: Solution for the system: \( A_2 x_2 = b_2 \)

\[
x_4 = b_4 / a_{44}
\]

\[
a_{34} x_4 + a_{33} x_3 = b_3 \quad \Rightarrow x_3 = (b_3 - a_{34} x_4) / a_{33}
\]

The dequeue of data for the solution of the above systems, together with the systolic network, are given in Figure (VI.A.6.1-f5). All the
computational steps required on this systolic array are displayed in Figure (VI.A.6.1-f6); in specific, the solution was obtained in 8 time-units, instead of 10 time-units that would be required normally. The efficiency achieved has been increased from: $E=1/2$ to 1, while the number of cells has remained the same as in [KUNG80].

Note, the special handling of the common elements to both streams, denoted by circles in Figure (VI.A.6.1-f5), and that for $n$ very large the solution will be pipelined twice as fast as in the above implementation.

Figure VI.A.6.1-f5: The Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.A.6: $\pi_n$] on the Linearly Connected Systolic Array (for $w=q=2$).
I. \( y_1 \) enters processor 2.

\[ x_1 = \frac{(b_1 - y_1)}{a_{11}} = \frac{b_1}{a_{11}} \text{ (since } y_1 = 0) \]

II. \( y_2 = a_{21} x_1 \),

\[ x_2 = \frac{(b_2 - y_2)}{a_{22}} = \frac{(b_2 - a_{21} x_1)}{a_{22}} \]

\( x_1 \) is output.

III. \( y_3 = a_{34} x_4 \),

\[ x_3 = \frac{(b_3 - y_3)}{a_{33}} = \frac{(b_3 - a_{34} x_4)}{a_{33}} \]

\( x_4 \) is output.

IV. \( y_4 \) enters processor 2.

\[ x_4 = \frac{(b_4 - y_4)}{a_{44}} = \frac{b_4}{a_{44}} \text{ (since } y_4 = 0) \]

V. Continued...
Figure VI.A.6.1-f6: All the Computational Steps of the Solution of the Lower Triangular Linear System of Paradigm [VI.A.6: π₂].
CASE B - (of Paradigm [VI.A.6:π₆])

(ii) - Upper Triangular Linear Systems

Since an upper triangular linear system can be rewritten as a lower triangular linear system, the systems to be solved are:

\[
\begin{bmatrix}
  a_{33} & a_{23} & 0 \\
  a_{22} & a_{12} & a_{11} \\
  0 & a_{12} & a_{11}
\end{bmatrix}
\begin{bmatrix}
  x_3 \\
  x_2 \\
  x_1
\end{bmatrix}
= \begin{bmatrix}
  b_3 \\
  b_2 \\
  b_1
\end{bmatrix}
\]

(Resulting from the 1\textsuperscript{st} stream of the LU-decomposition)

and

\[
\begin{bmatrix}
  a_{33} & 0 & a_{23} \\
  a_{43} & a_{44} & 0 \\
  0 & a_{12} & a_{11}
\end{bmatrix}
\begin{bmatrix}
  x_3 \\
  x_4 \\
  x_1
\end{bmatrix}
= \begin{bmatrix}
  b_3 \\
  b_4 \\
  b_1
\end{bmatrix}
\]

(Paradigm [VI.A.6:π₆]

We shall make use of the recurrence solution formulae given in (VI.A.6.1:3,4), or the general ones in (VI.A.6.1:5,6 respectively.

The corresponding solution for each of these systems is:

\textit{1\textsuperscript{st} Stream}

: Solution for the system: \( A_1 x_1 = b_1 \)

\[
x_3 = b_3/a_{33}
\]

\[
a_{23}x_3 + a_{22}x_2 = b_2 \iff x_2 = (b_2 - a_{23}x_3)/a_{22}
\]

\[
a_{12}x_2 + a_{11}x_1 = b_1 \iff x_1 = (b_1 - a_{12}x_2)/a_{11}
\]

\textit{2\textsuperscript{nd} Stream}

: Solution for the system: \( A_1 x_1 = b_1 \)

\[
x_3 = b_3/a_{33}
\]

\[
a_{43}x_3 + a_{44}x_4 = b_4 \iff x_4 = (b_4 - a_{43}x_3)/a_{44}
\]
The dequeue of data for the solution of the above systems, together with the systolic array of processors, are given in Figure (VI.A.6.1-f7). Note, the common elements to both streams denoted by circles.

Finally, all the computational steps required on this network are displayed in Figure (VI.A.6.1-f8); in specific, the solution was obtained in 8 time-units, while the comments made in the previous case, about the Efficiency and the matrix size, still apply to the present case.

![Diagram](image)

*Figure VI.A.6.1-f7: The Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm [VI.A.8:*] on the Linearly Connected Systolic Array (for \( w=q=2 \)).

\( ^{+} \)In fact, it can be obtained in even fewer time-units.
1. $y_3$ enters processor 2.

2. $x_3 = (b_3 - y_3) / a_{33} = b_3 / a_{33}$ (since $y_3 = 0$).

3. $y_2 = a_{23} x_3$.

4. $x_2 = (b_2 - y_2) / a_{22} = (b_2 - a_{23} x_3) / a_{22}$.

5. $x_3$ is output.

[Ch. VI/Sec. A: 718] continued...
Figure VI.A.6.1-f8: All the Computational Steps of the Solution of the Upper Triangular Linear System of Paradigm [VI.A.6: π₂].
VI.A.6.2: **GENERAL COMMENTS: THE PIVOTING PROBLEM, AND ORTHOGONAL FACTORIZATION**

Research in interconnection networks and algorithms has been traditionally motivated by large scale parallel Array computers, such as the ILLIAC IV. The technique presented herein was motivated by the advance in VLSI, albeit this is certainly applicable to any parallel computer complex. We have exemplified that many basic computations can be performed very efficiently by special-purpose multiple processor networks, which may be built very cheaply using the evolving VLSI technology. The important feature, common to all algorithms presented, is that their data flows are very simple and regular, and they are pipelinable algorithms.

In respect of the mathematical side, in everything that has been discussed previously, we have assumed that the matrices have the property that there is no need of using pivoting when the Gaussian elimination is applied to them. What, however, should one do if the matrices do not have this 'nice' property? Note that, the Gaussian elimination becomes very inefficient on mesh-connected processors, if pivoting is necessary.

This question has motivated the consideration of Givens's transformation (see Hammarling [HAMM74]) for triangularizing a matrix, which is known to be a numerically stable method. It turns out that, like Gaussian elimination without pivoting, the orthogonal factorization based on Givens's transformation can be implemented naturally on mesh-connected processors, although a pipelined implementation appears to be more complex. Sameh and Kuck, in [SAME78], considered parallel linear system solvers based on Givens's transformation, but they did not give solutions to the processor communication problem considered here.
SECTION B

Concurrent Systolization
For Solving
General Banded Linear Systems
VI.B.1: 'SYSTOLIC' LU-FACTORIZATION DEQUES FOR QUINDIAGONAL SYSTEMS

Again, we ought to examine two cases when $n$ is odd and when $n$ is even.

The general form of the matrix $A$ is:

$$
A = \begin{bmatrix}
\ldots & \ldots & \ldots & \ldots & a_{n-2,n} & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & \ldots & \ldots & \ldots & \ldots \\
a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & \ldots & \ldots & \ldots & \ldots \\
a_{21} & a_{22} & a_{23} & a_{24} & \ldots & \ldots & \ldots & \ldots & \ldots \\
a_{11} & a_{12} & a_{13} & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{bmatrix}
$$

We shall consider a small (in size) example to apply the 'rotate' and 'fold' method, since irrespective of the band-length of the considered matrix the procedural steps are the same. The only difficulty arises when we reach the final elimination stage to obtain the peak of the two streams in order to start the backward and forward substitutions.

In the penultimate stage of the elimination the middle element is
modified twice and then the final stage commences. In this final stage we have a (3\times3)-full matrix and the elimination process cannot be performed in parallel by both streams. So, obviously we have an overhead in the whole method, but it is infinitively small comparing it with the total amount of achievable parallelism in terms of speed-up.

Hence, conclusively, the two streams will unavoidably confront each other at the final stage of the elimination process.

Paradigm [VI.B.1:n\_1]

n-odd

Let us consider a (5\times5) quindiagonal matrix, since this case will demonstrate the critical cases in particular, i.e., the last confrontation stages above are the only elimination stages to be carried out. The matrix A will be:

\[
A = \begin{bmatrix}
    a_{11} & a_{12} & a_{13} & & & \\
    a_{21} & a_{22} & a_{23} & a_{24} & & \\
    a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & \\
    a_{42} & a_{43} & a_{44} & a_{45} & & \\
    a_{53} & a_{54} & a_{55} & & & \\
\end{bmatrix}
\]  

(VI.B.1:2)

The middle row is the third row, so for each of the two streams the two submatrices will include one more row than the middle one (i.e. the last/first row of the central (3\times3) subsystem for the top/bottom elimination streams, respectively). Consequently, the matrix A, according to the 'rotate' and 'fold' concept diagrammatically shown in (VI.B.1:2), will be rewritten as the following two submatrices $A_1$ and $A_2$: 
We assume that matrix $A$ has the property that each LU-decomposition can be done by Gaussian elimination without pivoting.

Remarks: The elimination process for the elements surrounded by the solid lines in the matrices $A_1$ and $A_2$ is to be carried out concurrently. On the other hand, the elimination process for the elements in the circles is to be carried out sequentially, i.e., part of the final elimination of the $(3 \times 3)$ central submatrix. The elements in the squares are completely ignored by the bottom originated elimination stream.

The corresponding triangular matrices $L=(l_{ij})$ and $U=(u_{ij})$ are evaluated according to the following general recurrences:

**1st Stream**

$$a_{1j}^{(1)} = a_{1j}$$

$$a_{1j}^{(k+1)} = a_{1j}^{(k)} + l_{ik}(-u_{kj})$$

**2nd Stream**

$$a_{1j}^{(1)} = a_{1j}$$

$$a_{1j}^{(k+1)} = a_{1j}^{(k)} + l_{it}(-u_{tj})$$
\[ \ell_{ik} = \begin{cases} 
0 & \text{if } i < k \\
1 & \text{if } i = k \\
\ell_{ik}^{(k)-1} u_{kk} & \text{if } i > k 
\end{cases} \]

\[ \ell_{it} = \begin{cases} 
0 & \text{if } i > t \\
1 & \text{if } i = t \\
\ell_{it}^{(k)-1} u_{tt} & \text{if } i < t 
\end{cases} \]  

\[ u_{kj} = \begin{cases} 
0 & \text{if } k > j \\
The matrix $A$, i.e., $p=q=3$, we have

for the $1^{st}$ stream:

\[ a_{i+2,i}^{(k)} = a_{i+2,i} \] for $1 \leq k \leq 1$, $1 \geq 2$ .

\[ a_{i,1+2}^{(k)} = a_{i,1+2} \]

and for the $2^{nd}$ stream:

\[ a_{i-2,i}^{(k)} = a_{i-2,i} \] for $1 \leq k \leq n-1+1, 1 \leq n-1$ .

\[ a_{i,1-2}^{(k)} = a_{i,1-2} \]

Remark: The above recurrences certainly can be applied to the

tridiagonal case of (par.-VI.A.6) as is proved in Appendix C-VI.□

Let us now compute the corresponding elements of the resulting

triangular matrices, by using the above recurrences for the opposite

factorization streams, so we shall have an analytical view of the

sequence of the modifications.

$1^{st}$ Stream ($k$ denotes steps and rows)

\[ a_{1j}^{(1)} = a_{ij} \]

\[ (k=1) \quad u_{1j}^{(1)} = a_{1j}^{(1)} \rightarrow u_{11}^{(1)} = a_{11}^{(1)}, u_{12}^{(1)} = a_{12}^{(1)}, u_{13}^{(1)} = a_{13}^{(1)} \]

\[ (k=2) \quad u_{2j}^{(2)} = a_{2j}^{(2)} \rightarrow u_{22}^{(2)} = a_{22}^{(2)}, u_{23}^{(2)} = a_{23}^{(2)}, u_{24}^{(2)} = a_{24}^{(2)} \]
\[
\begin{align*}
(k=3) & \quad u_{3j} = a_{3j}^{(3)} \Rightarrow u_{33} = a_{33}^{(3)}, \quad u_{34} = a_{34}^{(3)} \\
(k=4) & \quad u_{4j} = a_{4j}^{(4)} \Rightarrow u_{44} = a_{44}^{(4)} \quad \left(\rightarrow a_{44}^{(3)}\right)
\end{align*}
\]

\[
\begin{align*}
(k=1) & \quad \xi_{11} = a_{11}^{(1)} u_{11}^{-1} \Rightarrow \xi_{21} = a_{21}^{(1)} u_{11}^{-1}, \quad \xi_{31} = a_{31}^{(1)} u_{11}^{-1} \\
(k=2) & \quad \xi_{12} = a_{12}^{(2)} u_{22}^{-1} \Rightarrow \xi_{32} = a_{32}^{(2)} u_{22}^{-1}, \quad \xi_{42} = a_{42}^{(2)} u_{22}^{-1} \\
(k=3) & \quad \xi_{13} = a_{13}^{(3)} u_{33}^{-1} \Rightarrow \xi_{43} = a_{43}^{(3)} u_{33}^{-1}
\end{align*}
\]

\[
(k=1) \quad a_{1j}^{(2)} = a_{1j}^{(1)} + \xi_{1j}^{(1)} (-u_{1j}) \Rightarrow \left\{
\begin{align*}
a_{22} &= a_{22} + \xi_{21}^{(1)} (-u_{12}), & a_{23} &= a_{23} + \xi_{21}^{(1)} (-u_{13}) \\
& + \xi_{21}^{(1)} (-u_{12}), & a_{33} &= a_{33} + \xi_{31}^{(1)} (-u_{13}) \\
& a_{32} &= a_{32} + \xi_{31}^{(1)} (-u_{12}), & a_{33} &= a_{33} + \xi_{31}^{(1)} (-u_{13})
\end{align*}\right.
\]

\[
(k=2) \quad a_{1j}^{(3)} = a_{1j}^{(2)} + \xi_{1j}^{(2)} (-u_{2j}) \Rightarrow \left\{
\begin{align*}
a_{33} &= a_{33} + \xi_{32}^{(2)} (-u_{23}), & a_{34} &= a_{34} + \xi_{32}^{(2)} (-u_{24}) \\
& a_{43} &= a_{43} + \xi_{42}^{(2)} (-u_{23}), & a_{44} &= a_{44} + \xi_{42}^{(2)} (-u_{24})
\end{align*}\right.
\]

Remarks: The elements in the circles have to be collected 'on-the-fly' from the outputs of the corresponding cells and brought back into the serial stream for a further modification by the second wave. In particular, the element in the rectangle despite its modification to an \(a_{23}^{(3)}\) by the second factorization stream belongs to the present stream for the solution process (i.e., backward substitution). The underlined element is ignored, since the element \(a_{43}\) is not eliminated. \(\Box\)

Second Stream (\(k\) denotes steps and rows)
\[ a_{1j}^{(1)} = a_{1j} \]

\[(k=1) \Rightarrow t=5 \quad \Rightarrow u_{5j}^{(1)} = a_{5j}^{(1)} \Rightarrow u_{55} = a_{55}^{(1)} , u_{54} = a_{54}^{(1)} , u_{53} = a_{53}^{(1)} \]

\[(k=2) \Rightarrow t=4 \quad \Rightarrow u_{4j}^{(2)} = a_{4j}^{(2)} \Rightarrow u_{44} = a_{44}^{(2)} , u_{43} = a_{43}^{(2)} \Rightarrow u_{42} = a_{42}^{(2)} \]

\[(k=3) \Rightarrow t=3 \quad \Rightarrow u_{3j}^{(3)} = a_{3j}^{(3)} \Rightarrow u_{33} = a_{33}^{(3)} \Rightarrow \{ a_{33}^{(4)} \} , u_{32} = a_{32}^{(3)} \]

\[(k=4) \Rightarrow t=2 \quad \Rightarrow u_{2j}^{(4)} = a_{2j}^{(4)} \Rightarrow u_{22} = a_{22}^{(4)} \]

\[(k=1) \Rightarrow t=5 \quad \Rightarrow \ell_{15} = a_{15}^{(1)} \Rightarrow \ell_{45} = a_{45}^{(1)} \Rightarrow \ell_{35} = a_{35}^{(1)} \]

\[(k=2) \Rightarrow t=4 \quad \Rightarrow \ell_{14} = a_{14}^{(2)} \Rightarrow \ell_{34} = a_{34}^{(2)} \Rightarrow \ell_{24} = a_{24}^{(2)} \]

\[(k=3) \Rightarrow t=3 \quad \Rightarrow \ell_{13} = a_{13}^{(3)} \Rightarrow \ell_{23} = a_{23}^{(3)} \]

\[(k=1) \Rightarrow t=5 \quad \Rightarrow a_{1j}^{(2)} = a_{1j}^{(1)} \ell_{15}^{(-1)} \Rightarrow a_{44}^{(2)} = a_{43}^{(2)} + \ell_{45}^{(-1)} (-u_{54}^{(-1)}) , a_{43}^{(2)} = a_{44}^{(2)} + \ell_{45}^{(-1)} (-u_{54}^{(-1)}) , a_{43}^{(2)} = a_{44}^{(2)} + \ell_{45}^{(-1)} (-u_{54}^{(-1)}) \]

\[(k=2) \Rightarrow t=4 \quad \Rightarrow a_{1j}^{(3)} = a_{1j}^{(2)} \ell_{14}^{(-1)} \Rightarrow a_{32}^{(3)} = a_{33}^{(3)} + \ell_{34}^{(-1)} (-u_{43}^{(-1)}) , a_{33}^{(3)} = a_{32}^{(3)} + \ell_{34}^{(-1)} (-u_{43}^{(-1)}) , a_{33}^{(3)} = a_{32}^{(3)} + \ell_{34}^{(-1)} (-u_{43}^{(-1)}) \]

**Remarks:** The elements in the circles have to be collected 'on-the-fly' from the outputs of the corresponding cells and brought back into the serial stream for a further modification by the 1st wave. In particular, the elements in the rectangles will be finally outputted by the bottom originated factorization stream as: \( u_{43}^{(3)} = a_{43}^{(3)} \), \( \ell_{34}^{(3)} = a_{34}^{(3)} \ell_{44}^{(-1)} \), \( a_{33}^{(3)} = a_{33}^{(3)} + \ell_{34}^{(-1)} (-u_{43}^{(-1)}) \), whilst, on the other hand, the underlined elements are completely ignored. \( \Box \)
The dequeue of data resulting when the 'rotate' and 'fold' concept is applied to the two subparts of the matrix A, together with the systolic network of the hex-connected processors, are illustrated in Figure (VI.B.1-f1). Note that, the labelling of data denotes the sequence of modifications by the opposite factorization streams.

We must notice the difficulties arising when we attempt to represent the data stream, due to the 'on-the-fly' modifications imposed, which in addition make it impossible to keep pace with the $k$ increment (especially for the $2^{nd}$ stream). In other words, the formulae (VI.B.1:3) are applied correctly, but the $k^{th}$ step, because of the multiple modification of the same data (only in the elimination part of the central $(3 \times 3)$ submatrix), alters without being controlled by these formulae.

Comments: Again, the hex-connected processors have the properties defined in (par.-VI.A.6). The sequential bidirectional elimination process in the central $(3 \times 3)$ submatrix commences when (for $n$-odd) the middle element, in this example the element $a_{33}$, first enters the systolic network as an element of the $2^{nd}$ stream. Then, all the elements of the $2^{nd}$ stream already in the network (ready to exit or exiting it in that time-unit) have to be collected 'on-the-fly' as modified $a_{ij}$'s, instead of $l_{ij}$'s and $u_{ij}$'s, and brought back into the serial flow again. This procedure certainly includes the center element (i.e., the element $a_{33}$), as well as the element $a_{23}$ which belongs to the $1^{st}$ stream and is the only element (except the middle one) of that stream which will be modified again by the $2^{nd}$ stream.

Due to the sequential nature of the factorization process in that part.

These elements are denoted by 'double' circles in Figure (VI.B.1-f2).
Figure VI.B.1-f1: The Dequeue of Data for the LU-factorization on a Hexagonal Systolic Array (for p=q=3, and n=5).
From the speed-up point of view, the sequential elimination process for the (3x3) central submatrix contributes an unavoidable (but constant, independent of the original matrix size) overhead of 7 time-units to the time anticipated by Theorem [VI.A.6:0].‡ The previous elimination stages, including the penultimate stage of the twice modified middle element \(a_{33}\), are carried out in half the time required by the single stream LU-decomposition.

Finally, since the presented example demonstrated only the critical situations arising in quindia
gonal banded matrices, the timing obtained applying the 'rotate' and 'fold' technique was identical to that of the single stream LU-decomposition. However, for large (nxn) quindia
gonal banded matrices (e.g. n=4096) the speed-up achieved approximates the optimal value of 2. In terms of the Efficiency (E) achieved it has been normally as before increased from: E=1/3 to 1/2, whereas the number of IFS cells is the same as for the usual LU-decomposition array.

In Figure (VI.B.1-f2) are displayed six consecutive computational steps on this hex-connected systolic array.

In conclusion, for the specific instance of the present critical case, let us present a simple numerical example illustrating the necessary 'on-the-fly' modifications discussed above.

Consider the following (5x5) quindia
gonal linear system:

\[
\begin{bmatrix}
4 & 1 & -1 & 0 \\
1 & 4 & -1 & 1 \\
-1 & 1 & 5 & -1 & 1 \\
0 & 1 & 2 & 8 & -2 \\
0 & -1 & 1 & 4
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{bmatrix}
= 
\begin{bmatrix}
4 \\
5 \\
5 \\
9 \\
4
\end{bmatrix}
\]

(VI.B.1:6)
1. continued

2. continued...
The labelling denotes that this element has been modified by the 2nd stream.

Figure VI.B.1-f2: Six Consecutive Computational Steps of the LU-factorization of a Quindiagonal Matrix (for n=5).
By applying the folding algorithmic process we obtain the following solution steps:

\[\begin{bmatrix}
4 & 1 & -1 & 0 & x_1 \\
1 & 4 & -1 & 1 & x_2 \\
-1 & 1 & 5 & -1 & x_3 \\
1 & 2 & 8 & -2 & x_4 \\
0 & -1 & 1 & 4 & x_5
\end{bmatrix} = \begin{bmatrix}
4 \\
5 \\
5 \\
9 \\
4
\end{bmatrix}
\]

Note the sequential presentation of the parallel computational steps.

\[\begin{bmatrix}
4 & 1 & -1 & 0 & x_1 \\
1 & 4 & -1 & 1 & x_2 \\
1 & 21/4 & -5/4 & x_3 \\
1 & 3/2 & 17/2 & x_4 \\
0 & -1 & 1 & 4 & x_5
\end{bmatrix} = \begin{bmatrix}
4 \\
5 \\
4 \\
11 \\
4
\end{bmatrix}
\]

\[\begin{bmatrix}
4 & 1 & -1 & 0 & x_1 \\
15/4 & -3/4 & 1 & x_2 \\
5/4 & 5 & -5/4 & x_3 \\
1 & 3/2 & 17/2 & x_4 \\
0 & -1 & 1 & 4 & x_5
\end{bmatrix} = \begin{bmatrix}
4 \\
5 \\
4 \\
11 \\
4
\end{bmatrix}
\]

\[\begin{bmatrix}
4 & 1 & -1 & 0 & x_1 \\
15/4 & -3/4 & 1 & x_2 \\
21/4 & -19/12 & x_3 \\
17/10 & 247/30 & x_4 \\
-1 & 1 & 4 & x_5
\end{bmatrix} = \begin{bmatrix}
4 \\
4 \\
11/3 \\
149/15 \\
4
\end{bmatrix}
\]

Note: The sequential presentation of the parallel computational steps.
The solution vector $x$ was pre-arranged to be $x = (1, 1, 1, 1, 1)^T$. The elements in the circles are the elements to be eliminated at the next solution step.

**Triangular Linear Systems**

Let us now investigate the solution of the four resulting triangular linear systems of Paradigm [VI.B.1:π₁], considering, again, that an upper triangular linear system can always be rewritten as a lower triangular linear system without loss of generality.

We shall make use of the same systolic network required for the single LU-decomposition, coalescing two data streams to solve two lower triangular linear systems concurrently.

(i) - Lower Triangular Linear Systems

The systems to be solved are:

\[
\begin{bmatrix}
4 & 1 & 1 & & \\
15/4 & -945/988 & & & 0 \\
2755/494 & & & & \\
0 & 17/10 & 247/30 & & 0 \\
& -1 & 1 & 4 & \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
\end{bmatrix}
= \begin{bmatrix}
4 \\
690/247 \\
2755/494 \\
149/15 \\
4 \\
\end{bmatrix}
\]

(RESULTING FROM THE 4TH step)

The solution vector $x$ was pre-arranged to be $x = (1, 1, 1, 1, 1)^T$. The elements in the circles are the elements to be eliminated at the next solution step.

**Triangular Linear Systems**

Let us now investigate the solution of the four resulting triangular linear systems of Paradigm [VI.B.1:π₁], considering, again, that an upper triangular linear system can always be rewritten as a lower triangular linear system without loss of generality.

We shall make use of the same systolic network required for the single LU-decomposition, coalescing two data streams to solve two lower triangular linear systems concurrently.

(i) - Lower Triangular Linear Systems

The systems to be solved are:

\[
\begin{bmatrix}
a_{11} & & & & \\
a_{21} & a_{22} & & & 0 \\
a_{31} & a_{32} & a_{33} & & \\
0 & a_{42} & a_{43} & a_{44} & \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
= \begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
\end{bmatrix}
\]

(RESULTING FROM THE 1ST stream of the LU-decomposition)

and

\[
\begin{bmatrix}
a_{55} & & & & \\
a_{45} & a_{44} & & & 0 \\
a_{35} & a_{34} & a_{33} & & \\
0 & a_{24} & a_{23} & a_{22} & \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
= \begin{bmatrix}
b_5 \\
b_4 \\
b_3 \\
b_2 \\
\end{bmatrix}
\]

(RESULTING FROM THE 2ND stream of the LU-decomposition)
The recurrence solution formulae for these systems are those given in 
(VI.A.6.1:5,6), respectively, with the initialization:
\[
y_1^{(1)} = y_t^{(1)} = 0.
\]

For later convenience in following the systolic computational steps, let us proceed with the notational computation of the r.h.s. vector \( b \), during the solution of the original system with the coefficient matrix given in (VI.B.1:2), by applying the bidirectional Gaussian elimination, i.e.,

\[
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & \cdots & 0 \\
  a_{21} & a_{22} & a_{23} & a_{24} & \cdots \\
  a_{31} & a_{32} & a_{33} & a_{34} & \cdots \\
  a_{42} & a_{43} & a_{44} & a_{45} & \cdots \\
  \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
\end{pmatrix}
=
\begin{pmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5 \\
\end{pmatrix}
\]

The 'exponential' labelling in the r.h.s. vector \( b \) denotes as before the modification sequence by the two elimination streams. More analytically, in the 1\textsuperscript{st} step this sequence will be: \( b_1^{1}, b_2^{12}, b_3^{2}, b_4^{1} \), to continue in the 2\textsuperscript{nd} step by: \( b_2^{2}, b_3^{12}, b_4^{1}. \) Consequently, the final r.h.s. vector \( b \), which will be used in the solution of the upper triangular linear system, will have the form:

\[\text{Since, due to the interleaving opposite elimination streams, it is not necessary to complete the solution process implied from each of the above systems.}\]
In particular, according to the above denoted modification sequence, the interleaved $x_i$ values expected during the systolic computational steps are:

\[
\begin{align*}
    x_2 &= \frac{(b_2 - a_{21}x_1)}{a_{22}} \\
    x_3 &= \frac{(b_3 - a_{31}x_1)}{a_{33}} \\
    x_4 &= \frac{(b_4 - a_{45}x_5)}{a_{44}} \\
    x_3 &= \frac{(b_3 - a_{35}x_5)}{a_{33}} \\
    x_2 &= \frac{(b_2 - a_{24}x_4)}{a_{22}} \\
    x_3 &= \frac{(b_3 - a_{34}x_4)}{a_{33}} \\
    x_2 &= \frac{(b_2 - a_{23}x_3)}{a_{22}} \\
    x_3 &= \frac{(b_3 - a_{32}x_3)}{a_{33}} \\
    x_4 &= \frac{(b_4 - a_{42}x_2)}{a_{44}} \\
    x_3 &= \frac{(b_3 - a_{31}x_2)}{a_{33}} \\
    x_2 &= \frac{(b_2 - a_{21}x_1)}{a_{22}}
\end{align*}
\]

Furthermore, note that, every time the $y_i$'s enter the systolic linear array they are initialized to zero.

The dequeue of data resulting from the application of the 'rotate' and 'fold' concept to those subparts $A_1$ and $A_2$ of the original matrix $A$, together with the systolic array, are given in Figure (VI.B.1-4). Exceptionally for this case the 'don't care elements' do appear for convenience in the data stream, while the elements in the squares are ignored completely being considered as additional delays. Note the multiply 'on-the-fly' modified elements denoted by the circles.
Figure VI.B.1-f3: The Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1:π₁] on the Linearly Connected Systolic Array (for w=q=3).
In Figure (VI.B.1-f4) are displayed twelve consecutive computational steps required on this linear systolic network of processors.

This specific Paradigm (i.e., critical case) cannot assist in a fair comparison of the time-units required using the 'rotate' and 'fold' technique against the normal single stream solution process. In other words, the former technique requires 21 time-units as compared to the 13 time-units for the latter. Consequently, the Theorem [VI.A.6.1:e 2] does not apply to the present case due to the increased complexity in the 'on-the-fly' multiple modification of common elements, which arises from the concept of seeking for a central peak (in the forward elimination procedure) for both elimination streams. In actual fact both streams perform the elimination process on the \((p \times p)\) central subsystem sequentially, and in equal depth, which especially for this example \((i.e., n=5, q=3)\) proves to be critical.

For \(n \gg q\), however, the 'rotate' and 'fold' technique has again an advantage in speed-up compared to the normal process; but, there is no reason to follow such an uneconomical process, since the modified dequeues for unidirectional factorization of the central subsystems, to be introduced in (par.-VI.B.1.1), will be proved to be much more efficient even for small size examples. In addition, by using the 'rotate' and 'fold' concept as it stands herein, the 'tail' in the data stream is much more complicated compared to that of the unidirectional process.

Remark: The Efficiency achieved for the parallel elimination part has been increased from: \(E=1/2\) to 1 (i.e., one output every time-unit), \(\dagger\)

\(\dagger\)Therefore, the case when \(n\) is 'even' will be investigated therein in accordance with the new concept.
1. $x_1 = \frac{(b_1 - y_1)}{a_{11}} = \frac{b_1}{a_{11}}$ (since $y_1 = 0$).

2. $x_5 = \frac{(b_5 - y_5)}{a_{55}} = \frac{b_5}{a_{55}}$ (since $y_5 = 0$).

3. $x_3 = \frac{(b_3 - y_3)}{a_{33}} = \frac{b_3 - a_{33}x_3}{a_{33}}$.

4. $x_4 = \frac{(b_4 - y_4)}{a_{44}} = \frac{b_4 - a_{45}x_5}{a_{44}}$.

5. $x_1$ is output.

6. $x_2 = \frac{(b_2 - y_2)}{a_{22}} = \frac{b_2 - a_{21}x_1}{a_{22}}$.

This $x_2$ is an intermediate value.

continued...
Figure VI.B.1-f4: Twelve Consecutive Computational Steps for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1:π₁].
whereas when the opposite streams reach the central subsystem delays occur due to the multiple modification of the common elements. □

(ii) - Upper Triangular Linear Systems

For the completion of the presented example, the two upper triangular linear systems rewritten as lower triangular linear systems to be solved concurrently by the two opposite elimination streams are:

\[
\begin{bmatrix}
  a_{44} & 0 \\
  a_{34} & a_{33} \\
  a_{24} & a_{23} & a_{22} \\
  a_{14} & a_{13} & a_{12} & a_{11}
\end{bmatrix}
\begin{bmatrix}
  x_4 \\
  x_3 \\
  x_2 \\
  x_1
\end{bmatrix}
= \begin{bmatrix}
  b_4 \\
  b_3 \\
  b_2 \\
  b_1
\end{bmatrix}
\]

(Comparing from the 1st stream of the LU-decomposition)

\[
\begin{bmatrix}
  a_{22} & 0 \\
  a_{32} & a_{33} \\
  a_{42} & a_{43} & a_{44} \\
  a_{52} & a_{53} & a_{54} & a_{55}
\end{bmatrix}
\begin{bmatrix}
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{bmatrix}
= \begin{bmatrix}
  b_2 \\
  b_3 \\
  b_4 \\
  b_5
\end{bmatrix}
\]

(Comparing from the 2nd stream of the LU-decomposition)

Comment: In this phase we have the final backward substitution process commencing from the central element of the original matrix, the peak of the two streams. In this substitution process, only the elements included in the triangles are required and hence these will form the dequeue of data for the systolic computation. □

The recurrence solution formulae for these systems, taking into consideration the peculiarity of the case imposed by the 'rotate' and 'fold' technique, are those given in (VI.A.6.1:5,6)†, respectively; the solution vector to be obtained is the following:

†With the initialization: \( y_t^{(1)} = y_t^{(2)} = 0 \).
1st Stream

Solution for the system: $A_1x_1=b_1$

$x_3 = b_3 1212 \div a_{33}$

$a_{23}x_3 + a_{22}x_2 = b_2 12 \iff x_2 = (b_2 - a_{23}x_3) / a_{22}$

$a_{13}x_3 + a_{12}x_2 + a_{11}x_1 = b_1 \iff x_1 = (b_1 - a_{13}x_3 - a_{12}x_2) / a_{11}$.

2nd Stream

Solution for the system: $A_2x_2=b_2$

$x_3 = b_3 1212 \div a_{33}$

$a_{43}x_3 + a_{44}x_4 = b_4 21 \iff x_4 = (b_4 - a_{43}x_3) / a_{44}$

$a_{53}x_3 + a_{54}x_4 + a_{55}x_5 = b_5 \iff x_5 = (b_5 - a_{53}x_3 - a_{54}x_4) / a_{55}$.

The dequeue of data resulting when the 'rotate' and 'fold' concept is applied to those subparts of the original matrix A surrounded by the triangles, together with the systolic array, are given in Figure (VI.B.1-f9). Again, the common elements to both streams, denoted by circles, are subject to special handling.

Finally, in Figure (VI.B.1-f6) are displayed six consecutive computational steps on this linear systolic network of processors. The solution was obtained in 11 time-units.

Remarks: This number of time-units was closely anticipated by the Theorem [VI.A.6.1-6.2], so conclusively the whole delay is caused from the forward elimination process due to the multiple modification of the common elements. The Efficiency achieved has been increased from: E=1/2 to 1, whereas the number of IPSP's is the same as for the normal solution of triangular linear systems.
Figure VI.B.1-f5: The Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1:w_1] on the Linearly Connected Systolic Array (for w=q=3).
Figure VI.B.1-f8: Six Consecutive Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1:π].
In conclusion, with the new concept following we avoid the double sequential part in the center of the matrix and bring a balance between the otherwise should be identical timings in solving the triangular linear systems, by reducing, considerably, the timing for the lower system, and increasing, slightly, the timing for the upper system, to verify the anticipated timings by the above Theorem.

VI.B.1.1: Modified DeQueues For The Unidirectional Factorization of The 'Central' Subsystems

Let us consider again the case when \( n \) is odd and the matrix \( A \) as given in the previous Paradigm [VI.B.1.1], i.e.,

\[
A = \begin{bmatrix}
    \text{1st stream} & p \\
    \text{2nd stream} \\
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} & a_{24} \\
    a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\
    a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\
    a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\
\end{bmatrix} \quad (VI.B.1.1:1)
\]

In accordance with the new concept, the parallel part of the opposite eliminating streams remains the same as in the previous process; however, when the 'central' subsystem is reached (i.e., the (3×3) system denoted by the rectangle above) then instead of two sequential parts there is only one for the top stream, which settles a peak inside the area belonging to the bottom stream according to the previous concept (i.e., the element \( a_{44} \)). When the center element (i.e., the element \( a_{33} \)) first enters the systolic network as an element of the \( 2^{nd} \) stream,

\[\text{This term is indistinguishably related with the Gaussian elimination procedure.}\]
in a similar manner as before, all the elements of the 2nd stream already in the network (ready to exit or exiting it in that time-unit) have to be collected 'on-the-fly' as modified $a_{ij}$'s, instead of $l_{ij}$'s and $u_{ij}$'s, and brought back into the serial flow again. This procedure does not include the center matrix element (i.e., the element $a_{33}$).

Note the elimination process of the top/bottom streams denoted by squares and circles, respectively.

Another difference is that, in the backward substitution process, the 1st stream from the top commences ahead of the 2nd stream from the bottom, because it has to go through the sequential part at first, before the 2nd stream engages in the parallel process.

Consequently, according to the new concept diagrammatically shown in (VI.B.1.1:1), the matrix $A$ will be rewritten as the following two submatrices $A_1$ and $A_2$:

**1st Stream**

\[
A_1 = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} & 0 \\
  q & a_{21} & a_{22} & a_{23} & a_{24} \\
  a_{31} & a_{32} & a_{33} & a_{34} \\
  0 & a_{42} & a_{43} & a_{44}
\end{bmatrix}
\]

\[
A_1 = \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  l & 1 & 0 & 0 \\
  0 & l & 1 & 0 \\
  0 & 0 & l & 1
\end{bmatrix}
\]

\[
L_1 A_1 = \begin{bmatrix}
  u_{11} & u_{12} & u_{13} & 0 \\
  u_{22} & u_{23} & u_{24} \\
  0 & u_{33} & u_{34} \\
  u_{44}
\end{bmatrix}
\]

**2nd Stream**

\[
A_2 = \begin{bmatrix}
  a_{55} & a_{54} & a_{53} \\
  a_{45} & a_{44} & a_{43} \\
  a_{35} & a_{34} & a_{33}
\end{bmatrix}
\]

\[
A_2 = \begin{bmatrix}
  1 & 0 & 0 \\
  l & 1 & 0 \\
  l & 1 & 1
\end{bmatrix}
\]

\[
L_2 A_2 = \begin{bmatrix}
  u_{55} & u_{54} & u_{53} \\
  u_{44} & u_{43} \\
  0 & u_{33}
\end{bmatrix}
\]
Remark: Note that, the element $a_{34}$ (in circle) of the 3\textsuperscript{rd} stream is ignored. □

The corresponding triangular matrices $L=(l_{ij})$ and $U=(u_{ij})$ are similarly evaluated according to the recurrences and relations given in (VI.B.1:3, 4, 5).

Let us now compute the corresponding elements in both elimination streams, by using these recurrences, for a better assistance in following the sequence of modifications.

\textbf{1\textsuperscript{st} Stream ($k$ denotes steps and rows)}

\begin{equation}
a_{1j}^{(1)} = a_{1j}
\end{equation}

\begin{align*}
(k=1) & : u_{1j} = a_{1j}^{(1)} \Rightarrow u_{11} = a_{11}^{(1)}, u_{12} = a_{12}^{(1)}, u_{13} = a_{13}^{(1)} \\
(k=2) & : u_{2j} = a_{2j}^{(2)} \Rightarrow u_{22} = a_{22}^{(2)}, u_{23} = a_{23}^{(2)}, u_{24} = a_{24}^{(2)} \\
(k=3) & : u_{3j} = a_{3j}^{(3)} \Rightarrow u_{33} = a_{33}^{(3)}, u_{34} = a_{34}^{(3)} \\
(k=4) & : u_{4j} = a_{4j}^{(4)} \Rightarrow u_{44} = a_{44}^{(4)}
\end{align*}

\begin{align*}
(k=1) & : l_{11} = a_{11}^{(1)} - u_{11} \Rightarrow l_{21} = a_{21}^{(1)} - u_{21}, l_{31} = a_{31}^{(1)} - u_{31} \\
(k=2) & : l_{12} = a_{12}^{(2)} - u_{12} \Rightarrow l_{32} = a_{32}^{(2)} - u_{32}, l_{42} = a_{42}^{(2)} - u_{42} \\
(k=3) & : l_{13} = a_{13}^{(3)} - u_{13} \Rightarrow l_{43} = a_{43}^{(3)} - u_{43}
\end{align*}

\begin{align*}
(k=1) & : a_{1j}^{(2)} = a_{1j}^{(1)} + l_{11}(-u_{1j}) \Rightarrow \\
& \begin{cases} 
  a_{22}^{(2)} = a_{22}^{(1)} + l_{21}(-u_{12}), a_{23}^{(2)} = a_{23}^{(1)} + l_{21}(-u_{13}) \\
  a_{32}^{(2)} = a_{32}^{(1)} + l_{31}(-u_{12}), a_{33}^{(2)} = a_{33}^{(1)} + l_{31}(-u_{13}) \\
  a_{43}^{(2)} = a_{43}^{(1)} + l_{43}(-u_{13})
\end{cases} \\
(k=2) & : a_{1j}^{(3)} = a_{1j}^{(2)} + l_{12}(-u_{2j}) \Rightarrow \\
& \begin{cases} 
  a_{33}^{(3)} = a_{33}^{(2)} + l_{33}(-u_{32}), a_{34}^{(3)} = a_{34}^{(2)} + l_{33}(-u_{34}) \\
  a_{43}^{(3)} = a_{43}^{(2)} + l_{43}(-u_{34})
\end{cases} \\
(k=3) & : a_{1j}^{(4)} = a_{1j}^{(3)} + l_{13}(-u_{3j}) \Rightarrow \\
& \begin{cases} 
  a_{44}^{(4)} = a_{44}^{(3)} + l_{44}(-u_{34})
\end{cases}
\end{align*}
The modified dequeue of data resulting when the 'rotate' and 'fold' concept is applied to the two subparts of the matrix $A$, together with the systolic network of the hex-connected processors, are depicted in Figure (VI.B.1.l-f1). Note, again, the labelling of data which denotes the sequence of modifications by the opposite factorization streams.

In Figure (VI.B.1.l-f2) are displayed eight consecutive computational steps on this systolic array.

Comments: The timing formula for this case, a modification of the formula given in Theorem [VI.A.6; Theorem], is: $$3 \left\lfloor \frac{n}{2} \right\rfloor + 4 \tilde{p},$$ where $\tilde{p}$ is either $p-1$ or $p$. For $\tilde{p}=p-1$, this formula verifies the total number of time-units (i.e., 14 time-units) required for the factorization of the previous $(5\times5)$ example using the new concept. A comparison with the timing required using the previous concept (i.e., 18 time-units) proves that a considerable reduction in time-units was obtained albeit the size of the considered Paradigm was very small. Note that, for the parallel part the Efficiency was increased to two outputs every three time-units,
Figure VI.B.1.1-f1: The Modified Dequeue of Data for the LU-factorization on a Hexagonal Systolic Array (for p=q=3, and n=5).
Figure VI.B.1.1-f2: Eight Consecutive Computational Steps of the Unidirectional (for the Central Submatrix) LU-factorization of a Quindiagonal Matrix (for n=5).
whereas for the sequential part the Efficiency still remained: $E=1/3$.

In respect to the $k$-step numbering, each factorization stream maintains its own counter independent of the other's.

A complete discussion about the general timing formula for the LU-factorization of general banded linear systems will be considered in the next paragraph along with the related theory.

Finally, let us now exemplify the new concept for the specific instance of the present critical case, considering the $(5 \times 5)$ quindagonal linear system of (VI.B.1.6), i.e.,

$$
\begin{bmatrix}
4 & 1 & -1 & 0 \\
1 & 4 & -1 & 1 \\
-1 & 1 & 5 & -1 & 1 \\
1 & 2 & 8 & -2 \\
0 & -1 & 1 & 4
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{bmatrix}
= 
\begin{bmatrix}
4 \\
5 \\
5 \\
9 \\
4
\end{bmatrix}.
$$

(VI.B.1.1:2)

By applying the folding algorithmic process according to this concept we obtain the following solution steps†:

\begin{align*}
\begin{bmatrix}
4 & 1 & -1 & 0 \\
1 & 4 & -1 & 1 \\
-1 & 1 & 5 & -1 \\
1 & 2 & 8 & -2 \\
0 & -1 & 1 & 4
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{bmatrix}
&= 
\begin{bmatrix}
4 \\
5 \\
5 \\
9 \\
4
\end{bmatrix} \\
\quad \text{(top-bottom streams)}
\end{align*}

\begin{align*}
\begin{bmatrix}
4 & 1 & -1 & 0 \\
1 & 4 & -1 & 1 \\
-1 & 1 & 5 & -1 \\
1 & 2 & 8 & -2 \\
0 & -1 & 1 & 4
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{bmatrix}
&= 
\begin{bmatrix}
4 \\
5 \\
5 \\
9 \\
4
\end{bmatrix} \\
\quad \text{(top-bottom streams)}
\end{align*}

Note the sequential presentation of the parallel computational steps.
The solution vector $\mathbf{x}$ was pre-arranged to be $\mathbf{x}=(1,1,1,1,1)^T$. The elements in the circles are the elements to be eliminated at the next solution step.

**Triangular Linear Systems**

We shall again investigate the solution of the four resulting triangular linear systems according to the new unidirectional factorization of the central subsystem. The systolic network to be used is identical to that used for the solution of the systems resulting from the single LU-decomposition, while a compound data stream is
pipelined through it for the concurrent solution of the pairs of lower triangular linear systems.

(i) - Lower Triangular Linear Systems

The systems to be solved are:

\[
\begin{bmatrix}
    a_{11} & 0 \\
    a_{21} & a_{22} \\
    a_{31} & a_{32} & a_{33} \\
    a_{41} & a_{42} & a_{43} & a_{44}
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    x_3 \\
    x_4
\end{bmatrix}
= 
\begin{bmatrix}
    b_1 \\
    b_2 \\
    b_3 \\
    b_4
\end{bmatrix}
\]  
(Resulting from the 1st stream of the LU-decomposition)

and

\[
\begin{bmatrix}
    a_{55} & 0 \\
    a_{45} & a_{44} \\
    a_{35} & a_{34} & a_{33}
\end{bmatrix}
\begin{bmatrix}
    x_5 \\
    x_4 \\
    x_3
\end{bmatrix}
= 
\begin{bmatrix}
    b_5 \\
    b_4 \\
    b_3
\end{bmatrix}
\]  
(Resulting from the 2nd stream of the LU-decomposition)

The recurrence solution formulae for these systems are those given in (VI.A.6.1.5,6)\(^\dagger\), respectively. Again, for later convenience in following the systolic computational steps, let us proceed with the notational computation of the r.h.s. vector \(b\), during the solution of the original system with the coefficient matrix given in (VI.B.1.2), by applying the bidirectional Gaussian elimination, i.e.,

\[\text{1st stream}\]

\[\text{1st step} \rightarrow \begin{bmatrix}
    a_{11} & a_{12} & a_{13} \circled{0} \\
    a_{21} & a_{22} & a_{23} & a_{24} \\
    a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\
    a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\
    a_{51} & a_{52} & a_{53} & a_{54} & a_{55}
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    x_3 \\
    x_4 \\
    x_5
\end{bmatrix}
= 
\begin{bmatrix}
    b_1 \\
    b_2 \\
    b_3 \\
    b_4 \\
    b_5
\end{bmatrix}
\]  
\[\dagger\text{With the initialization: } y^{(1)}_t = y^{(1)}_t = 0.\]
Note, again, the 'exponential' labelling in the r.h.s. vector \( \mathbf{b} \), denoting the modification sequence by the opposite elimination streams. Consequently, the final r.h.s. vector \( \mathbf{b} \), which will be used in the solution of the upper triangular linear system, will have the form:

\[
\mathbf{b} = \begin{bmatrix}
    b_1 \\
    b_2 \\
    b_{211} \\
    b_3 \\
    b_{211} \\
    b_4 \\
    b_5
\end{bmatrix}.
\]

In particular, according to the double modification stream, the interleaved \( x \) values expected during the systolic computational steps are:

\[
x_2 = \left( b_2 - a_{21}x_1 \right)/a_{22} \quad \text{(1st stream)}
\]

\[
x_3 = \left( b_3 - a_{31}x_1 \right)/a_{33} \quad \text{(1st stream)}
\]

\[
x_3 = \left( b_3 - a_{31}x_1 \right)/a_{33} \quad \text{(1st stream)}
\]
\[ \begin{align*}
    x_4 &= \left( b_4 - a_{45} x_5 \right) / a_{44} \\
    x_3 &= \left( b_3 - a_{35} x_5 \right) / a_{33} \\
    x_4 &= \left( b_4^{21} - a_{42} x_2 \right) / a_{44} \\
    x_4 &= \left( b_4^{21} - a_{43} x_3 \right) / a_{44} 
\end{align*} \]

Again, the \( y \)'s are initialized to zero each time they enter the systolic linear array.

The modified dequeue of data resulting from the application of the new unidirectional (for the central submatrix) 'rotate' and 'fold' concept to those subparts \( A_1 \) and \( A_2 \) of the original matrix \( A \), together with the systolic array, are given in Figure (VI.B.1.1-f3). The delays are, again, denoted by 'don't care elements', the element in the square is ignored completely being considered as an additional delay, while the elements in circles are subject to special handling.

**Remark:** Note that, in the parallel part the Efficiency (E) of the array has been increased to one output every time-unit, whereas in the sequential part it has maintained the value 1/2.

In Figure (VI.B.1.1-f4) are displayed six consecutive computational steps required on this linear systolic array of processors. The total number of time-units required for the solution of the previous systems, anticipated by the Theorem [VI.A.6.1:8], equals the number of time-units (i.e., 13 time-units) required by the single stream solution process. The considered case, however, is considered to be a critical case, and hence, for \( n \gg p \), the new 'rotate' and 'fold' concept proves to be twice as fast compared to the above normal process.
Figure VI.B.1.1-f3: The Modified Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1:π₁] on the Linearly Connected Systolic Array (for w=q=3).
Figure VI.B.1.1-f4: Six Consecutive Computational Steps for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1-n_1] Using a Modified Dequeue of Data.
(ii) - Upper Triangular Linear Systems

In this part we shall solve the second pair of systems, i.e. the upper triangular linear systems, by converting them to lower ones; thus, the systems to be solved are:

\[
\begin{bmatrix}
a_{44} & \circ \\
a_{34} & a_{33} \\
a_{24} & a_{23} & a_{22} \\
\circ & a_{13} & a_{12} & a_{11}
\end{bmatrix}
\begin{bmatrix}
x_4 \\
x_3 \\
x_2 \\
x_1
\end{bmatrix}
= 
\begin{bmatrix}
b_4 \\
b_3 \\
b_2 \\
b_1
\end{bmatrix} \quad \text{(Resulting from the 1st stream of the LU-decomposition)}
\]

and

\[
\begin{bmatrix}
a_{33} & \circ \\
a_{43} & a_{44} \\
a_{53} & a_{54} & a_{55}
\end{bmatrix}
\begin{bmatrix}
x_3 \\
x_4 \\
x_5
\end{bmatrix}
= 
\begin{bmatrix}
b_3 \\
b_4 \\
b_5
\end{bmatrix} \quad \text{(Resulting from the 1st stream of the LU-decomposition)}
\]

Comment: The elements of the matrix \( A_2 \) denoted by the circles are ignored, since the solution process of these subsystems is a straight backward substitution process due to the lack of interference between the two opposite streams, according to the unidirectional concept applied on the central submatrix.\( \square \)

The recurrence solution formulae for these systems are those given in (VI.A.6.1:5,6)\(^*\), while the corresponding solution for each of these systems is:

1\textsuperscript{st} Stream

: Solution for the system: \( A_1 x = b_1 \)

\( ^* \)With the initialization: \( y_1^{(1)} = y_2^{(1)} = 0. \)
\[ x_4 = b_4^{211}/a_{44} \]
\[ a_{34}x_4 + a_{33}x_3 = b_3^{211} \implies x_3 = (b_3^{211} - a_{34}x_4)/a_{33} \]
\[ a_{24}x_4 + a_{23}x_3 + a_{22}x_2 = b_2^1 \implies x_2 = (b_2^1 - a_{24}x_4 - a_{23}x_3)/a_{22} \]
\[ a_{13}x_3 + a_{12}x_2 + a_{11}x_1 = b_1 \implies x_1 = (b_1 - a_{13}x_3 - a_{12}x_2)/a_{11} \].

**2nd Stream**

: Solution for the system: \( A_2x_2 = b_2 \)

\[ a_{53}x_3 + a_{54}x_4 + a_{55}x_5 = b_5 \implies x_5 = (b_5 - a_{53}x_3 - a_{54}x_4)/a_{55} \].

The modified dequeue of data for the solution of the above systems, together with the systolic linear array, are given in Figure (VI.B.1.1-f5). Note the interchange of positions of the elements of matrix \( A_2 \) denoted by squares, and the duplication of the elements denoted by circles in this Figure to avoid the special 'on-the-fly' handling requisite.

Finally, in Figure (VI.B.1.1-f6) are displayed six consecutive computational steps on this linear systolic network of processors.

**Remark:** The total number of time-units required, and anticipated by Theorem [VI.A.6.1:2] (for \( p=p-1 \)), was 12, while the observations made for the previous pair of systems, i.e., about the Efficiency of the array and the potential of the new concept, well apply to the present case.

Prior to proceeding with the next Paradigm to summarize the new concept, when the central submatrix has been reached we do not seek for a center element, or two opposite stream peaks, in order to start the destream backward substitution from, but the elimination process is a single, top originated, LU-decomposition. The acceptability of this new concept is supported by the facts that: a) The size of the central submatrix is very small in comparison with the size of the
Figure VI.B.1.1-f5: The Modified Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1:π] on the Linearly Connected Systolic Array (for w=q=3).
\[
x_4 = \frac{(b_4 - y_4)}{a_{44}} \quad (\text{since } y_4 = 0).
\]

\[
x_3 = \frac{(b_3 - y_3)}{a_{33}} \quad \Rightarrow \quad y_2 = a_{24}x_4 \quad \Rightarrow \quad x_3 \text{ is output.}
\]

\[
x_2 = \frac{(b_2 - y_2)}{a_{22}} \quad \Rightarrow \quad y_5 = a_{54}x_4 + a_{53}x_3 \quad \Rightarrow \quad x_2 \text{ is output.}
\]

The duplicate \(x_4\) is output.

\[y_1 = a_{13}x_3\]

\[y_5 = a_{54}x_4 + a_{53}x_3\]

\[x_1 = \frac{(b_1 - y_1)}{a_{11}} \quad \Rightarrow \quad y_5 = \frac{(b_5 - y_5)}{a_{55}} \quad \Rightarrow \quad x_1 \text{ is output.}
\]

\[y_3 = a_{34}x_4\]

\[y_3 = a_{34}x_4\]

\[y_5 = a_{54}x_4\]

\[y_2 = a_{24}x_4 + a_{23}x_3\]

\[x_4 \text{ is output.}\]

\[y_5 = a_{54}x_4\]

\[y_1 = a_{13}x_3 + a_{12}x_2\]

\[x_3 \text{ is output.}\]

\[y_3 = a_{34}x_4\]

\[y_2 = a_{24}x_4 + a_{23}x_3\]

\[x_4 \text{ is output.}\]
original matrix (in real life problems); b) the degree of difficulty in transforming the 'tail' of the data stream is considerably smaller than previously; and c) the time-complexity has been reduced.

Paradigm [VI.B.1.1:n]

Let us consider the following (4×4) qudiagonal matrix, which similarly as before consists of a critical case, i.e.,

1st stream

\[
A = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} & 0 \\
  a_{21} & a_{22} & a_{23} & a_{24} \\
  a_{31} & a_{32} & a_{33} & a_{34} \\
  0 & a_{42} & a_{43} & a_{44}
\end{bmatrix}
\]

\[\text{modified twice by both streams in this bidirectional procedure, so what remains is the elimination of the element } a_{32} \text{ by the 1st stream.}\]

Consequently, we have:

1st Stream

\[
A_1 = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 \\
  l_{21} & 1 \\
  l_{31} & l_{32} & 1
\end{bmatrix}
\begin{bmatrix}
  u_{11} & u_{12} & u_{13} \\
  u_{22} & u_{23} \\
  0 & u_{33}
\end{bmatrix}
\begin{bmatrix}
  L_1 \\
  & 1
\end{bmatrix}
\]

n-even
\[ a_2 = \begin{bmatrix} a_{44} & a_{43} & a_{42} \\ a_{34} & a_{33} & a_{32} \\ a_{24} & a_{23} & a_{22} \end{bmatrix} = \begin{bmatrix} \ell_{34} & 1 \\ \ell_{24} & \ell_{23} & 1 \end{bmatrix} \begin{bmatrix} u_{44} & u_{43} & u_{42} \\ u_{33} & u_{32} \\ 0 & 0 & u_{22} \end{bmatrix}. \]

**Remark:** For the elements surrounded by the circles note that, the multiplier \( \ell_{23} \) is not needed and consequently the element \( u_{22} \) is not modified again for the elimination of the element \( a_{23} \). \( \square \)

Again, we assume that the matrix \( A \) has the property that its LU-decomposition can be done by Gaussian elimination without pivoting.

The triangular matrices \( L=(\ell_{ij}) \) and \( U=(u_{ij}) \) are evaluated using the recurrences and relations given in (VI.B.1:3,4,5), whereas for a better assistance in following the sequence of modifications we shall compute the corresponding elements in both factorization streams.

**2nd Stream**

\[ a_{ij}^{(1)} = a_{ij} \]

\[(k=1) \quad \Rightarrow \quad u_{1j} = a_{1j}^{(1)} \Rightarrow u_{11} = a_{11}^{(1)} , u_{12} = a_{12}^{(1)} , u_{13} = a_{13}^{(1)} \]

\[(k=2) \quad \Rightarrow \quad u_{2j} = a_{2j}^{(2)} \Rightarrow u_{22} = a_{22}^{(2)} , u_{23} = a_{23}^{(2)} \]

\[(k=3) \quad \Rightarrow \quad u_{3j} = a_{3j}^{(3)} \Rightarrow u_{33} = a_{33}^{(3)} \]

\[(k=1) \quad \Rightarrow \quad \ell_{11} = a_{11}^{(1)} u_{11}^{-1} \Rightarrow \ell_{21} = a_{21}^{(1)} u_{11}^{-1} , \ell_{31} = a_{31}^{(1)} u_{11}^{-1} \]

\[(k=2) \quad \Rightarrow \quad \ell_{12} = a_{12}^{(2)} u_{22}^{-1} \Rightarrow \ell_{22} = a_{22}^{(2)} u_{22}^{-1} \]

\[(k=1) \quad \Rightarrow \quad a_{1j}^{(2)} = a_{1j}^{(1)} + \ell_{11} u_{1j} \]

\[(k=2) \quad \Rightarrow \quad a_{1j}^{(3)} = a_{1j}^{(2)} + \ell_{12} u_{1j} \]

\[ \begin{cases} a_{22} = a_{22}^{(1)} + \ell_{21} (-u_{12}) , a_{23} = a_{23}^{(1)} + \ell_{21} u_{13} \\ a_{32} = a_{32}^{(1)} + \ell_{31} (-u_{12}) , a_{33} = a_{33}^{(1)} + \ell_{31} u_{13} \end{cases} \]
2nd Stream (k denotes steps and rows)

\[ a_{1j}^{(1)} = a_{1j} \]

\[ (k=1) \Rightarrow t=4 \quad a_{4j} = a_{4j}^{(1)} \Rightarrow a_{44} = a_{44}^{(1)}, a_{43} = a_{43}^{(1)}, a_{42} = a_{42}^{(1)} \]

\[ (k=2) \Rightarrow t=3 \quad a_{3j} = a_{3j}^{(2)} \Rightarrow a_{33} = a_{33}^{(2)}, a_{32} = a_{32}^{(2)} \]

\[ (k=3) \Rightarrow t=2 \quad a_{2j} = a_{2j}^{(2)} \Rightarrow a_{22} = a_{22}^{(2)} \]

\[ (k=1) \Rightarrow t=4 \quad a_{1j} = a_{1j}^{(1)} - a_{1j}^{(2)} \Rightarrow a_{14} = a_{14}^{(1)} - a_{14}^{(2)} \]

Remarks: Note that, since there are common elements to both streams (these are the elements consisting of the central submatrix) they are computed correctly through the given recurrences, but irrespective of the index \( k \), which cannot follow the pace of the alternate modifications, since its increment is unique to each elimination stream. The elements consisting of the central submatrix will be defined by the appropriate theory introduced for the general case in (par.-VI.B.2), so it will be known beforehand what exemptions in terms of the index \( k \) should be expected in the factorization formulae.

The modified dequeue of data resulting when the new concept for the above factorization is applied, together with the systolic network of the hex-connected processors, are illustrated in Figure (VI.B.1.1-f7).

In Figure (VI.B.1.1-f8) are displayed six consecutive computational
Figure VI.B.1.1-f?: The Modified Dequeue of Data for the LU-factorization on a Hexagonal Systolic Array (for $p=q=3$, and $n=4$).
Figure VI.B.1.1-f8: Six Consecutive Computational Steps of the Unidirectional (for the Central Submatrix) LU-factorization of a Quindiagonal Matrix (for n=4).
steps on this systolic array. In particular, we should underline the special 'on-the-fly' handling of the element $a_{33}$, and the fact that when it first enters the systolic network as an element of the $1^{st}$ stream, then all the elements already in the network at that snapshot (or exiting it in that time-unit) have to be collected 'on-the-fly' as modified $a_{lj}$'s, instead of $l_{lj}$'s and $u_{lj}$'s and brought back into the serial flow again.

Comments: The timing formula is as the one given for the case when $n$ is odd, i.e., $3\left\lfloor \frac{n}{2} \right\rfloor + 4\tilde{p}$. Again, for $\tilde{p} = p - 1$, this formula verifies the total number of time-units (i.e., 14 time-units) required for the factorization of the considered $(4 \times 4)$ example. The comments made for the above case (i.e., $n$ is odd) regarding the Efficiency achieved apply to the present case as well. Note that, when $n >> p$, the above timing formula becomes a relation depending only up to $n$, which implies a twice as fast execution timing compared to the normal single stream LU-decomposition. □

Finally, let us now exemplify the new concept for the specific instance of the present critical case, considering the $(4 \times 4)$ quindiaogonal linear system:

\[
\begin{bmatrix}
4 & 1 & 1 & 0 \\
1 & 4 & -1 & 1 \\
2 & 1 & 8 & -1 \\
0 & -1 & 1 & 2 \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
= 
\begin{bmatrix}
6 \\
5 \\
10 \\
2 \\
\end{bmatrix}
\]

(VI.B.1.1:4)

By applying the folding algorithmic process according to this concept we obtain the following single solution step:

\[\text{The element } a_{33} \text{ is certainly included.}\]
The solution vector \( x \) was pre-arranged to be \( x = (1,1,1,1)^T \).

## Triangular Linear Systems

The next thing to investigate is the solution of the four resulting triangular linear systems applying the 'rotate' and 'fold' concept.

(i) - Lower Triangular Linear Systems

The systems to be solved are:

\[
\begin{bmatrix}
    a_{11} & a_{21} & a_{31} & a_{41} \\
    0 & a_{22} & a_{32} & a_{42} \\
    0 & 0 & a_{33} & a_{43} \\
    0 & 0 & 0 & a_{44}
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    x_3 \\
    x_4
\end{bmatrix}
= 
\begin{bmatrix}
    b_1 \\
    b_2 \\
    b_3 \\
    b_4
\end{bmatrix} \tag{5}
\]

(Resulting from the 1\textsuperscript{st} stream of the LU-decomposition)

\[
\begin{bmatrix}
    a_{11} & a_{21} & a_{31} & a_{41} \\
    0 & a_{22} & a_{32} & a_{42} \\
    0 & 0 & a_{33} & a_{43} \\
    0 & 0 & 0 & a_{44}
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    x_3 \\
    x_4
\end{bmatrix}
= 
\begin{bmatrix}
    b_1 \\
    b_2 \\
    b_3 \\
    b_4
\end{bmatrix} \tag{6}
\]

(Resulting from the 2\textsuperscript{nd} stream of the LU-decomposition)

The element denoted by a circle is not used, according to the elimination process, and hence it will be considered as a 'don't care element' in the modified dequeue of data which will be given further on.
The recurrence solution formulae are as those for the case when n is odd, whilst the corresponding solution for each of the above systems is:

1st Stream

: Solution for the system: \( \text{A}_1 \text{x}_1 = \text{b}_1 \)

\[
\begin{align*}
\text{x}_1 &= \frac{\text{b}_1}{\text{a}_{11}} \\
\text{a}_{21} \text{x}_1 + \text{a}_{22} \text{x}_2 &= \text{b}_2 \quad \Rightarrow \quad \text{x}_2 = \frac{(\text{b}_2 - \text{a}_{21} \text{x}_1)}{\text{a}_{22}} \\
\text{a}_{31} \text{x}_1 + \text{a}_{32} \text{x}_2 + \text{a}_{33} \text{x}_3 &= \text{b}_3 \quad \Rightarrow \quad \text{x}_3 = \frac{(\text{b}_3 - \text{a}_{31} \text{x}_1 - \text{a}_{32} \text{x}_2)}{\text{a}_{33}}.
\end{align*}
\]

2nd Stream

: Solution for the system: \( \text{A}_2 \text{x}_2 = \text{b}_2 \)

\[
\begin{align*}
\text{x}_4 &= \frac{\text{b}_4}{\text{a}_{44}} \\
\text{a}_{34} \text{x}_4 + \text{a}_{33} \text{x}_3 &= \text{b}_3 \quad \Rightarrow \quad \text{x}_3 = \frac{(\text{b}_3 - \text{a}_{34} \text{x}_4)}{\text{a}_{33}} \\
\text{a}_{24} \text{x}_4 + \text{a}_{23} \text{x}_3 + \text{a}_{22} \text{x}_2 &= \text{b}_2 \quad \Rightarrow \quad \text{x}_2 = \frac{(\text{b}_2 - \text{a}_{24} \text{x}_4 - \text{a}_{23} \text{x}_3)}{\text{a}_{22}}.
\end{align*}
\]

Comment: The underlined elements denote the intermediate values obtained during the multiple modification of these common elements, which will substitute the corresponding r.h.s. \( \text{b}_i \)'s for the computation of the final solution value. □

The modified dequeue of data for the solution of the above systems, together with the systolic linear array, are given in Figure (VI.B.1.1-f9).

In Figure (VI.B.1.1-f10) are displayed six consecutive computational steps on this linear systolic network of processors.

Remarks: The total number of time-units required, and anticipated by Theorem [VI.A.6.1:θ₂] (for \( \tilde{p} = p - 1 \)), was 11, which is justified by the fact that the examined case is considered to be a critical case. The Efficiency (E) of the array has been similarly as before increased to one output every time-unit for the parallel part, whereas in the
sequential part it has maintained the value 1/2. Hence, for \( n \gg p \), the modified dequeue will prove to be twice as fast compared to the single stream solution process. \( \square \)

\((ii)\) - Upper Triangular Linear Systems

The pair of the resulting upper triangular systems to be solved, converted to a pair of lower triangular systems, are the following:

\[
\begin{bmatrix}
  a_{33} & 0 \\
  a_{23} & a_{22} \\
  a_{13} & a_{12} & a_{11}
\end{bmatrix}
\begin{bmatrix}
  x_3 \\
  x_2 \\
  x_1
\end{bmatrix}
= 
\begin{bmatrix}
  b_3 \\
  b_2 \\
  b_1
\end{bmatrix}
\quad \text{(Resulting from the 1st stream of the LU-decomposition)}
\]

and

\[
\begin{bmatrix}
  0 & a_{33} \\
  a_{22} & a_{21} \\
  a_{13} & a_{12} & a_{11}
\end{bmatrix}
\begin{bmatrix}
  x_3 \\
  x_2 \\
  x_1
\end{bmatrix}
= 
\begin{bmatrix}
  b_3 \\
  b_2 \\
  b_1
\end{bmatrix}
\quad \text{(Resulting from the 2nd stream of the LU-decomposition)}
\]

Comments: The elements of the matrix \( A_2 \) denoted by the circles are ignored, since the solution process of these subsystems is a straight backward substitution process due to the lack of interference between the two opposite streams, according to the unidirectional concept applied on the central submatrix. The elements denoted by the squares have interchanged positions in the modified dequeue of data to be presented in the following. Also, note the duplication of the elements in circles in that Figure to avoid the special 'on-the-fly' handling requisite. \( \square \)

The recurrence solution formulae for these systems are those given in (VI.A.6.1:5,6), with the initialization: \( y_1^{(1)} = y_t^{(1)} = 0 \), while the corresponding solution for each of these systems is:
Figure VI.B.1.1-f2: The Modified Dequeue of Data for the Solution of the Lower Triangular Linear System of Paradigm [VI.B.1.1:f1] on the Linearly Connected Systolic Array (for w=q=3).
Six Consecutive Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1.1:n 1 Using a Modified Dequeue of Data.

1. \[ x_4 = \frac{(b_4 - y_4)}{a_{44}} \]
   \[ = \frac{b_4}{a_{44}} \] (since \( y_4 = 0 \)).

2. \[ x_1 = \frac{(b_1 - y_1)}{a_{11}} \]
   \[ = \frac{b_1}{a_{11}} \] (since \( y_1 = 0 \)).

3. \[ y_2 = \frac{a_{21} x_1}{a_{22}} \]
   \[ = \frac{(b_2 - y_2)}{a_{22}} \]
   \[ = \frac{(b_2 - a_{24} x_4)}{a_{22}} \]

4. \[ y_3 = \frac{a_{31} x_1 + a_{32} x_2}{a_{33}} \]
   \[ = \frac{(b_3 - y_3)}{a_{33}} \]
   \[ = \frac{(b_3 - a_{34} x_4)}{a_{33}} \]

5. \[ x_2 = \frac{(b_2 - y_2)}{a_{22}} \]
   \[ = \frac{(b_2 - a_{24} x_4)}{a_{22}} \]

6. \[ x_3 = \frac{(b_3 - y_3)}{a_{33}} \]
   \[ = \frac{(b_3 - a_{31} x_1 + a_{32} x_2)}{a_{33}} \]

This \( x_3 \) is an intermediate value.

---

Figure VI.B.1.1-f10: Six Consecutl.ive Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1.1:n 1 Using a Modified Dequeue of Data.
**1st Stream**

Solution for the system: \( A_1x_1 = b_1 \)

\[
x_3 = \frac{b_3}{a_{33}}
\]

\[
a_{23}x_3 + a_{22}x_2 = b_2 \iff x_2 = \frac{b_2 - a_{23}x_3}{a_{22}}
\]

\[
a_{13}x_3 + a_{12}x_2 + a_{11}x_1 = b_1 \iff x_1 = \frac{b_1 - a_{13}x_3 - a_{12}x_2}{a_{11}}.
\]

**2nd Stream**

\[
a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4 \iff x_4 = \frac{b_4 - a_{42}x_2 - a_{43}x_3}{a_{44}}.
\]

The modified dequeue of data for the solution of these systems, together with the systolic linear array, are given in Figure (VI.B.1.1-f11).

Finally, in Figure (VI.B.1.1-f12) are illustrated six consecutive computational steps on this linear systolic network of processors.

**Remarks:** The total number of time-units required, and anticipated by Theorem [VI.A.6.1:6;] (for \( \tilde{p} = p-1 \)), was 11, while the observations made for the previous pair of systems, i.e., about the Efficiency of the systolic array of processors and the potential of the modified 'rotate' and 'fold' concept, apply equally well to the present case. In particular, by equalizing the timing formulae given by the above Theorem and Theorem [VI.A.6.1:9;] we obtain: \( \tilde{p} = n/2 \), which, in reverse, indicates the instance that these timings will coincide. As an example, however, if we consider the quindagonal case for \( n=4096 \), then the solution process using a modified dequeue proves to be 1.997... faster than the single stream solution process. Overall, and in more practical terms, for the parallel solution parts we apply the timing formula given by Theorem [VI.A.6.1:8;], whereas for the sequential parts imposed due to the formation of the \((p\times p)\) central submatrix the timing formula given by Theorem [VI.A.6.1:8;] is applied. □
Figure VI.B.1.1-f11: The Modified Dequeue of Data for the Solution of the Upper Triangular Linear System of Paradigm (VI.B.1.1:π₁) on the Linearly Connected Systolic Array (for w=q=3).
1. $x_3 = \frac{b_{31} - y_3}{a_{33}} = \frac{b_{31}}{a_{33}}$ (since $y_3 = 0$).

2. $y_2 = a_{23}x_3$.

3. $y_4 = a_{43}x_3$.

4. $y_4 = a_{43}x_3 + a_{42}x_2$.

5. $x_2 = (b_{12}y_2)/a_{22}$.

6. $x_2 = (b_1y_1)/a_{11}$.

The duplicate $x_3$ is output. $x_2$ is output.

Figure VI.B.1.1-12: Six Consecutive Computational Steps for the Solution of the Upper Triangular Linear System of Paradigm [VI.B.1.1:π₁] Using a Modified Dequeue of Data.
VI.B.2: 'SYSTOLIC' PIPELINABILITY 'ROTATING' AND 'FOLDING' GENERAL BANDED MATRICES

After the detailed investigation of the 'rotate' and 'fold' concept in the previous paragraphs, and since the difficulties\(^\dagger\) arising in the implementation of this technique were bypassed by the established and exemplified unidirectional procedure, herein we shall examine the boundaries of the central formatted submatrix for the general banded case and introduce the complementary\(^\ddagger\) background theory simplifying the variety of the occurring cases.

The problem of solving a banded system of linear equations

\[
Ax = b
\]

occurs frequently in the numerical solution of partial and ordinary differential equations. In the subsequent analysis we shall consider, again, the case that the coefficient matrix A has the property that its LU-decomposition can be done by Gaussian elimination without pivoting, and it is of the following symmetric semi-band type:

\(^\dagger\)Due to the interference of the two opposite factorization streams.

\(^\ddagger\)To the theory for full matrices introduced in (Appendix C-VI/par.-VI.A.6).
In particular, for simpler exemplification purposes, let us consider the case that the matrix $A$ has a semi-bandwidth $p=4$. The LU-decomposition of this general heptadiagonal matrix using the 'rotate' and 'fold' concept for the opposite factorization streams will be presented as:
which implies that:

\[
A_1 = \begin{bmatrix}
1 \\
\ell_{21} & 1 \\
\ell_{31} & \ell_{32} & 1 \\
\ell_{41} & \ell_{42} & \ell_{43} & 1 \\
\ell_{51} & \ell_{52} & \ell_{53} & \ell_{54} & 1 \\
\end{bmatrix}
\]

\[
L_1 = \begin{bmatrix}
\ell_{t,t-1} \\
\ell_{t+\bar{p}-1,t+\bar{p}-4} \\
\ell_{t+\bar{p}-1,t+\bar{p}-2} \\
\end{bmatrix}
\]

\[
U_1 = \begin{bmatrix}
u_{11} & u_{12} & u_{13} & u_{14} \\
u_{22} & u_{23} & u_{24} & u_{25} \\
u_{33} & u_{34} & u_{35} & u_{36} \\
u_{tt} & \ell_{t,t+\bar{p}-1} \\
\end{bmatrix}
\]

and

\[
A_2 = \begin{bmatrix}
1 \\
\ell_{n-1,n} \\
\ell_{n-2,n} & \ell_{n-2,n-1} & 1 \\
\ell_{n-3,n} & \ell_{n-3,n-2} & 1 \\
\ell_{n-4,n-1} & \ell_{n-4,n-2} & \ell_{n-4,n-3} & 1 \\
\ell_{r,r+1} & \ell_{r+1,r+2} & \ell_{r+2,r+3} & \ell_{r+3,r+4} & 1 \\
\end{bmatrix}
\]

\[
L_2 = \begin{bmatrix}
\ell_{r-r+1} \\
\ell_{r+1,r+p+1} \\
\ell_{r+p+1,r+p+2} \\
\end{bmatrix}
\]

\(1^{st} \text{ stream}\)
The triangular matrices \( L = \{l_{ij}\} \) and \( U = \{u_{ij}\} \) are evaluated according to the recurrences given in (VI.B.1:3). However, because of the bandwidth of the matrix \( A \), i.e., \( p = 4 \), we have the relations (VI.B.1:4,5) respectively modified as:

**1\(^{st}\) Stream**

\[
\begin{align*}
\{k\} & \quad a_{i+3,1}^{(k)} = a_{1+3,1} \\
\{k\} & \quad a_{1,i+3}^{(k)} = a_{1,i+3}
\end{align*}
\]

for \( 1 \leq \xi \leq i \) and \( i \geq 2 \). \( (VI.B.2:3) \)

and for the

**2\(^{nd}\) Stream**

\[
\begin{align*}
\{k\} & \quad a_{i-3,1}^{(k)} = a_{1-3,1} \\
\{k\} & \quad a_{1,i-3}^{(k)} = a_{1,i-3}
\end{align*}
\]

for \( 1 \leq \xi \leq [n-1] \) and \( 1 \leq \xi \leq [n-1] \). \( (VI.B.2:4) \)

**Comments:** Although the central formatted submatrix is unique, the existence of different counters in each of the factorization streams imposes the general expression of its elements as each of the opposite elements.

\[\uparrow \] These elements consist of the exemption for the formulae giving at each \( k \)-step the general matrix elements modification.
streams is 'facing' them, for any semi-bandwidth \( p \) (i.e., odd or even) and any value of \( n \) (i.e., odd or even). In other words, for the 1st stream the central submatrix appears as:

\[
\begin{bmatrix}
  a_t, & a_{t+p-1} \\
  a_{t+p-1}, & a_{t+p-1,t+p-1}
\end{bmatrix}
\]

where \( t = \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{p-1}{2} \right\rfloor \);

whereas for the 2nd stream it appears as:

\[
\begin{bmatrix}
  a_r & a_{r-p+1} \\
  a_{r-p+1}, & a_{r-p+1,r-p+1}
\end{bmatrix}
\]

where \( r = \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{p-1}{2} \right\rfloor \).

The value of \( p \) depends on the size of the central submatrix which is known beforehand as \((p \times p)\) or \([(p-1) \times (p-1)]\).

Let us now solve a numerical example considering the following

(5x5) heptadiagonal linear system, i.e.,

\[
\begin{bmatrix}
  4 & 1 & -1 & 1 & 0 \\
  1 & 4 & -1 & 1 & 1 \\
  -1 & 1 & 5 & -1 & 1 \\
  1 & 2 & 1 & 8 & -2 \\
  0 & -1 & 1 & -1 & 4
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{bmatrix}
= \begin{bmatrix}
  5 \\
  6 \\
  5 \\
  10 \\
  3
\end{bmatrix}
\]

(VI.B.2:5)

By applying the folding algorithmic process according to the original concept we obtain the following solution steps:

multipliers (top-bottom streams)

\[
\begin{bmatrix}
  4 & 1 & -1 & 1 & 0 \\
  1 & 4 & -1 & 1 & 1 \\
  -1 & 1 & 5 & -1 & 1 \\
  1 & 2 & 1 & 8 & -2 \\
  0 & -1 & 1 & -1 & 4
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{bmatrix}
= \begin{bmatrix}
  5 \\
  6 \\
  5 \\
  10 \\
  3
\end{bmatrix}
\]

\[
\begin{bmatrix}
  4 & 1 & -1 & 1 & 0 \\
  1 & 4 & -1 & 1 & 1 \\
  -1 & 1 & 5 & -1 & 1 \\
  1 & 2 & 1 & 8 & -2 \\
  0 & -1 & 1 & -1 & 4
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{bmatrix}
= \begin{bmatrix}
  5 \\
  6 \\
  5 \\
  10 \\
  3
\end{bmatrix}
\]

\[
\begin{bmatrix}
  4 & 1 & -1 & 1 & 0 \\
  1 & 4 & -1 & 1 & 1 \\
  -1 & 1 & 5 & -1 & 1 \\
  1 & 2 & 1 & 8 & -2 \\
  0 & -1 & 1 & -1 & 4
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{bmatrix}
= \begin{bmatrix}
  5 \\
  6 \\
  5 \\
  10 \\
  3
\end{bmatrix}
\]

\[
\begin{bmatrix}
  4 & 1 & -1 & 1 & 0 \\
  1 & 4 & -1 & 1 & 1 \\
  -1 & 1 & 5 & -1 & 1 \\
  1 & 2 & 1 & 8 & -2 \\
  0 & -1 & 1 & -1 & 4
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{bmatrix}
= \begin{bmatrix}
  5 \\
  6 \\
  5 \\
  10 \\
  3
\end{bmatrix}
\]
The solution vector \( \mathbf{x} \) was pre-arranged to be \( \mathbf{x} = (1,1,1,1,1)^T \). The elements in the circles are the elements to be eliminated at the next solution step.

With the unidirectional procedure investigated previously we avoided the superfluous complexity caused by the overlapping of the opposite
factorization streams; since the degree of their interference, however, gradually increases along with the increase of the semi-bandwidth $p^+$, to conclude with the formation of the central full submatrix, it is necessary to introduce, briefly, the relative background theory.

We shall apply similar techniques as in (Appendix C-VI/par.-VI.A.6) to transform the matrix $A$ with the form given by (VI.B.2:2).

For the case when $n$ is odd we proceed as follows:

(i) The elements $a_{i,j}$ are eliminated by the top stream for $1,j$ in the range:

$$j = 1 \left(1 - \left\lfloor \frac{n+1}{2} \right\rfloor \right), \quad i = [j+1](1)[j+p-1] \ , \quad (VI.B.2:6)$$

which implies that the elements $a_{n-1+1,n-j+1}$ are eliminated by the bottom stream. The transformed matrix $A$ has the form:

$$\begin{bmatrix}
0 & \frac{n+1}{2} & 0 \\
\downarrow & & \downarrow \\
0 & p & 0
\end{bmatrix}$$

\begin{equation}
(VI.B.2:7)
\end{equation}

(ii) From this stage onwards we need to introduce slightly more sophisticated factorization techniques as the opposite streams start interfering since they have to eliminate elements of the same row. The elements $a_{i,j}$ to be eliminated are for $i,j$ in the range $^\dagger$:

\[\begin{align*}
\text{In accordance with the cases exemplified previously in this Chapter it is apparent that any problem tends to become critical for degenerate cases of } p-n. \\
^\dagger A \text{ certain degree of flexibility should be allowed in the application of these formulae when same degenerated cases (i.e., almost full matrices) are to be considered.}
\end{align*}\]
The resulting transformed matrix $A$ will have the form:

\[
\begin{bmatrix}
0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix}
\]

\[
\begin{cases}
\frac{(n-p)}{2}, & \text{p-odd} \\
\frac{(n-p+1)}{2}, & \text{p-even}
\end{cases}
\]

\text{for p-odd} \quad (VI.B.2:10)

and

\[
\begin{bmatrix}
0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix}
\]

\[
\begin{cases}
\frac{(n-p+2)}{2}, & \text{p-odd} \\
\frac{(n-p+1)}{2}, & \text{p-even}
\end{cases}
\]

\text{for p-even.} \quad (VI.B.2:11)
(iii) Finally, at this stage, in accordance with the original (not the unidirectional) procedure, each pivoting row eliminates first the appropriate element of the opposite process, so that each elimination does not recreate values in the zero elements created by the other. The elimination procedure continues now exactly as for the case of a full matrix resulting with the matrix $A$ having the form:

$$
\begin{bmatrix}
0 & \cdots & \cdots & \cdots & \cdots \\
\cdot & \ddots & \cdots & \cdots & \cdots \\
\cdot & \cdot & \ddots & \cdots & \cdots \\
\cdot & \cdot & \cdot & \ddots & \cdots \\
\cdot & \cdot & \cdot & \cdot & \ddots \\
\end{bmatrix}
$$

(VI.B.2:12)

The case when $n$ is even can also be treated in a similar manner by following the previous steps with the following differences at each step:

(i) Instead of (VI.B.2:6), the top stream eliminates the elements $a_{1j}$ such that

$$j = 1(1)[\frac{n}{2} - p + 1], \quad l = (j+1)(1)[j+p-1], \quad (VI.B.2:13)$$

which implies again that the elements $a_{n-1+l, n-j+1}$ are eliminated by the bottom stream. The transformed matrix $A$ has the form:
Then, instead of (VI.B.2:8,9), the elements $a_{ij}$ which are correspondingly eliminated are for $i, j$ in the range:

$$j = \left\lfloor \frac{n}{2} - p + 1 \right\rfloor (l+1), \quad (2\text{ stream})$$

and

$$j = \left\lceil \frac{n}{2} + p - 1 \right\rceil (l+1), \quad (3\text{ stream})$$

The resulting transformed matrix $A$ will have the form (VI.B.2:11) for $p$-odd and the form (VI.B.2:10) for $p$-even.

(iii) Finally, the matrix $A$ has the easily solvable form:
which is treated similarly to the even case as exemplified for the instance when matrix A is full.

In accordance with the previous background theory, the conclusion emanating is that we must always investigate four different cases, namely, two subcases when n-odd, one for p-odd and the other for p-even, and two corresponding subcases when n-even. This variety of cases, however, may be simplified to only one case despite the values of n and p.

It can be noticed easily that the cases: (n-even/p-even) and (n-odd/p-odd) conclude with the formation of a central subsystem of size \((p \times p)\). On the other hand, the remaining complementary cases, i.e., (n-even/p-odd) and (n-odd/p-even), conclude with the formation of a central subsystem of size \(((p-1) \times (p-1))\). Hence, the four different cases have been reduced to two cases.

Furthermore, in every case the formation of the central full submatrix commences at \(\left\lfloor \frac{n-p}{2} \right\rfloor + 1\) lines depth for both factorization streams, counting from the corresponding first line for each wave. Apparently, we may consider the case: (n-odd/p-odd) as a subcase of the case: (n-odd/p-even), as well as the case: (n-even/p-even) a subcase of the case: (n-even/p-odd). Hence, we end up with only one case.

All these relations between the variety of cases occurring are diagrammatically illustrated in Figure (VI.B.2-f1).

Comments: In this Figure the number of lines per factorization stream is considered according to the original 'rotate' and 'fold' concept. We must notice that when n-odd/p-odd or n-even/p-even, then the range for each of the opposite streams does not reach the last line of the central submatrix, but stops one line before it. To the contrary, when we consider the superset cases of the above pairs, i.e., when
n-odd/p-even or n-even/p-odd, then each stream overlaps the other the full length of the central submatrix. Hence, due to the relation \( \subseteq \) between the former and latter pairs of cases always we may consider the latter range of overlap. □

Since the size of the matrix in real life problems is bound to be \( n \gg p \), we may consider that the number of lines treated by each stream is \( \left\lfloor \frac{n}{2} \right\rfloor \). Consequently, in accordance with the modified 'rotate' and 'fold' concept, the timing formula is: \( 3\left\lfloor \frac{n}{2} \right\rfloor + \text{the time for the central full matrix} \), i.e., \( 4\bar{p} \), where \( \bar{p} = p-1 \) or \( p \).

Theorem [VI.B.2.2]

The systolic pipelinability of the LU-factorization of \((n \times n)\) symmetric semi-bandwidth matrices by the 'rotating' and 'folding' technique is directly dependent upon the relation between their size and semi-bandwidth.

Proof:

By equalizing the above timing formula with the single stream LU-

![Diagram](image)

**Figure VI.B.2-f1:** The Relativity of the Various Cases for a \((n \times n)\) Banded Matrix of Semi-bandwidth \( p \).
decomposition formula given by Theorem [VI.A.8:2] we obtain:

\[
3 \left\lfloor \frac{n}{2} \right\rfloor + 4 \bar{p} = 3n + p. \quad (VI.B.2:18)
\]

We must examine the cases: \( \bar{p} = p - 1 \) and \( \bar{p} = p \). Hence, for \( \bar{p} = p - 1 \) we have:

\[
4(p-1) - p = 3n - 3 \left\lfloor \frac{n}{2} \right\rfloor \implies
3p - 4 = 3 \left\lfloor \frac{n}{2} \right\rfloor \implies
p - 1 = \left\lfloor \frac{n}{2} \right\rfloor. \quad (VI.B.2:19)
\]

On the other hand, for \( \bar{p} = p \), we have:

\[
3p = 3 \left\lfloor \frac{n}{2} \right\rfloor \implies
p = \left\lfloor \frac{n}{2} \right\rfloor. \quad (VI.B.2:20)
\]

Consequently, from formulae (VI.B.2:19,20) we may write:

\[
\bar{p} = \left\lfloor \frac{n}{2} \right\rfloor. \quad (VI.B.2:21)
\]

The last relation determines the critical bounds for \( p \) and \( n \).

In other terms, if we have a problem where \( \bar{p} \) and \( n \) verify the relation (VI.B.2:21), then the expected timings from both schemes will be identical. If \( \bar{p} < \left\lfloor \frac{n}{2} \right\rfloor \), then the 'rotate' and 'fold' technique proves to be superior, whereas due to the enlargement of the formatted central submatrix exactly the opposite applies for \( \bar{p} > \left\lfloor \frac{n}{2} \right\rfloor \).

Remarks: In practice it is bound that \( n \gg \bar{p} \), hence the superiority of the 'rotate' and 'fold' technique is apparent. If we consider an arithmetic example, i.e., \( p = 5 \) and \( n = 4096 \), then the speed-up achieved by this technique is approximately 2. Also, take into account that due to the relation \( \subseteq \) illustrated in Figure (VI.B.2-f1) we can practically consider that \( \bar{p} = p - 1 \) for every occurring case. \( \square \)
To conclude, the reader is once more invited to follow any of the solved examples in the present Chapter, by making transparencies either of the compound data stream, or the network itself, and moving them one over the other appropriately. In addition, as a further exercise, the verification of the timings obtained in these examples (for the LU-factorization and the solution of the resulting triangular systems) is proposed, by applying the appropriate general formulae established, respectively.
VI.B.3: **FURTHER RESEARCH IN THE 'SOFT-SYSTOLIC' AREA, CONCLUSIVE REMARKS**

The data stream 'rotating' and 'folding' technique can be naturally extended to higher level foldings (see Bekakos and Evans [BEKA85a]. In specific, if we consider, for simplicity, the problem and the instance defined in (par.-VI.A.4), then we get the following Theorem.

**Theorem [VI.B.3: θ]**

The \((n\times n)\)-band matrix-vector multiplication problem with bandwidth \(w=p+q-1\) can be solved in \(\lceil n/2 \rceil + w\) time-units, using a systolic array of \(w\) double IPSP's (with only a simple modification for the \(x\)'s), and one adder, applying the fourfolding technique.

**Proof:**

(By construction of the array). \(\square\)

In other words, each cell will consist of 2 IPSP's (*binary cell*) and the adder is to be used for the middle point of the matrix, since it will be evaluated concurrently in both streams. In Figure (VI.B.3-f1) is illustrated the definition of the binary cell.

*Figure VI.B.3-f1: Definition of the Binary Cell.*
Consider again the tridiagonal matrix $A$ (for $n=5$)

\[
\begin{bmatrix}
  a_{11} & a_{12} & & & \\
  a_{21} & a_{22} & a_{23} & & \\
  & a_{32} & a_{33} & a_{34} & \\
  & & a_{43} & a_{44} & a_{45} \\
  & & & a_{54} & a_{55}
\end{bmatrix}
\]

The quadruple resulting when applying the fourfolding technique is the following:

\[
\begin{bmatrix}
  a_{23} & a_{43} \\
  a_{32} & a_{34} & a_{22} & a_{44} \\
  a_{12} & a_{54} & a_{33} & \phi & a_{21} & a_{45} \\
  a_{11} & a_{55} & \delta & \delta
\end{bmatrix}
\]

\[
x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5
\]

\[
y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5
\]

\text{ notation, } \delta = \text{ dummy element}

In Figure (VI.B.3-f2) are illustrated all the computational steps of the algorithm.

\textbf{Remark:} The adjacent elements to the middle element of the $x$-vector have to be kept in the left-end binary cell for two clock ticks, as well as a copy of them to propagate to their right-neighbour. This can be easily achieved by performing a simple modification in the cell structure given in Figure (VI.A.4-f2), for each individual IPSP in the.

\textit{This technique involves a 'rotation' of the corresponding off-diagonals for each folding of the matrix band.}
binary cell, i.e., connect the [NOT-AND] logic element to the output line for x's, cancel the clock for the y's.

Since we have entered the space of the soft-systolic algorithms when talking about binary cells (although it could be constructed a simple cell with 6 registers and an adder), in [BEKAS5a], we have made use of the flexibility of the area to solve another problem involving the diagonal and anti-diagonal banded matrix, otherwise known as X-band matrix. Many problems give a matrix of this general structure,

\[
\begin{align*}
V_1 &= a_{11}x_1 + a_{12}x_2 + a_{13}x_3, \\
V_2 &= a_{21}x_1 + a_{22}x_2 + a_{23}x_3, \\
V_3 &= a_{31}x_1 + a_{32}x_2 + a_{33}x_3, \\
V_4 &= a_{41}x_1 + a_{42}x_2 + a_{43}x_3, \\
V_5 &= a_{51}x_1 + a_{52}x_2 + a_{53}x_3.
\end{align*}
\]

Figure VI.B.3-f2: The Computational Steps of the Matrix-Vector Multiplication Algorithm (n=5) using a 'Quadrequeue'.

if we are flexible in allowing some of the sparseness to be included in the X-bands.

Certainly the 'rotate' and 'fold' method as exemplified in [BEKAB55a] can be generalized retaining the quadrequeue as the fundamental granularity factor. This process, however, implies the utilization of more complicated binary cell networks with always a doubling number of IPSP's, compared to the immediately previous folding level, but with a directly analogous computational speed-up. The number of cells may be considered as increasing rapidly, however, since we are in the space of the soft-systolic algorithms, we can have a non-planar (multi-layer) structure accommodating on each layer a modified Leiserson's linear systolic array with local broadcasting between them.

In this work the 'rotating' and 'folding' technique has been used and considered up to the fourfold level. Theoretically we can continue these foldings until we end up with a (5x5) sub-band partition of the original band, which is the minimum boundary allowing a fourfolding step. This is currently under investigation.

The superiority of the 'rotating' and 'folding' technique compared with the other approaches discussed in (par.-VI.A.4) has already (by constructing the arrays) become apparent. In specific, assuming a constraint-free technological status, there is no comparison with the interleaving of the diagonals approach. On the other hand, in the soft-systolic space, a direct comparison of this mathematical approach with the hardware double pipe approach would prove that the timing results obtained are of the same order. However, in the latter approach the number of IPSP's increases quadratically, while in our case it doubles for each new fourfolding level. This is even more obvious in the
single-band matrix-vector multiplication case, where with the double pipe approach twice as many IPSP's are required compared to our case in order to obtain similar timing results.

We shall conclude this Chapter overviewing some vital points made in earlier Chapters about the design of special-purpose VLSI computer systems. In specific, the cost-effectiveness of such systems has always been a major concern; their fabrication cost must be low enough to justify their specialized, and consequently limited, applicability.

Cost can be distinguished in non-recurring design and recurring part costs. Any fall of the latter's cost is equally applied for the merit of both, special-purpose and general-purpose computer systems. Furthermore, this cost is even less significant than the design cost, since the production of special-purpose computer systems in large quantities is quite a rare phenomenon.

Hence, the design cost of such a system should be relatively small for it to be more attractive compared to a general-purpose computer, and this can be achieved by the utilization of appropriate architectures. More explanatory, if the decomposition of a structure into a few types of simple substructures, which are respectively utilized with simple interfaces, is feasible, then significant cost savings can be achieved.

In addition, special-purpose computer systems based on simple and regular designs are likely to be modular and, therefore, adjustable to various performance goals - that is, systems' cost can be made analogous to the performance required. This fact reveals that accomplishing the architectural challenge for simple and regular designs yields cost-effective special-purpose computer systems.
CHAPTER VII: SUPERCOMPUTING WITH DATA-DRIVEN WAVEFRONT ARRAY PROCESSORS

VII.1: Introductory Remarks

VII.2: A Pipelinable Two-Dimensional Computational Wavefront Concept

VII.3: On the Solution of Linear Systems Applying the Quadrant Interlocking Factorization - QIF Method

VII.4: Single Stage Computational DeWavefronts for the Implementation of the QIF Algorithm

VII.5: Discussion and Further Remarks
CHAPTER VII

SUPERCOMPUTING
WITH DATA-DRIVEN
WAVEFRONT ARRAY PROCESSORS
VII.1: INTRODUCTORY REMARKS

As it has become apparent from the previous Chapter, the introduction of the VLSI circuit technology has offered a promising scenario for processing large scale data by a multiprocessor array in a highly parallel mode. Nevertheless, prior to the efficient utilization of such an advanced technology in large processor array structures, let us summarize several of the fundamental problems imposed by the above technology:

i) **Interconnection:** In massively parallel array processors this is the most critical issue of the system design, since communication is very expensive in terms of area, power, and time consumption. Therefore, communication has to be restricted to 'localized interconnections'.

ii) **Description:** A novel descriptive tool is required to assist in the visualization, description and verification of parallel algorithms in a large computing network.

iii) **Clocking:** The clocking scheme is another very critical issue. In the globally synchronous scheme, there is a global clock network to broadcast the clock signal over the entire array. For very large systems, however, the clock skew unavoidably incurred in this signal distribution
is a non-trivial factor, causing unnecessary slowdown in the clock rate (see S.Y. Kung and Gal-Ezer [KUNG82]. Hence, a self-timed scheme appears to be more preferable.

**iv) Design Complexity:** Large design of layout costs suggest the utilization of repetitive modular structures, i.e., a few different types of simple (and often standard) cells.

**v) Programmability:** Programmable processor modules (as opposed to dedicated modules) are preferable due to cost-effectiveness considerations. The high cost of designing such modules may be amortized over a broader range of applications. Indeed, a major portion of scientific computations can be reduced to a basic set of matrix operations and other related algorithms, which should be carefully investigated in the attempt to simplify the hardware module.

All these constraints which are imposed by VLSI will render the general-purpose processor array very inefficient, sometimes almost impossible, a fact which should dictate the scope of the algorithms which can be tackled by this array.

In this *Chapter* a special-purpose network of processors is introduced, which was also conceived and further developed by S.Y. Kung, et al [KUNG82a]. The important feature of this *wavefront array processor* is its computational notation which removes the need for global synchronization and also proves to be useful for programming the machine and describing the algorithm.

The principal theme herein is the direct hardware implementation of an alternate matrix procedure for the solution of linear systems: \(Ax=b\), where \(A\) is a compact dense \((n \times n)\) matrix. The method is based on the factorization of the coefficient matrix into components which are of
butterfly form, i.e., interlocking matrix quadrants, and for its implementation the concept of computational *de*wavefronts is investigated.

The *Chapter* concludes with remarks on the simulation of the phenomenon of the propagation of the waves and a further discussion about some improved issues of the QIF method.

In addition, further investigating hints are given for a possible, area efficient, combined utilization of both systolic and wavefront conceptual computational tools for a fully parallel solution of general banded linear systems.
VII.2 A PIPELINABLE TWO-DIMENSIONAL COMPUTATIONAL WAVEFRONT CONCEPT

It has recently been indicated (see Speiser and Whitehouse [SPEI80]) that a significant part of the computational requirements for signal processing and other applied mathematical problems can, in fact, be reduced to a basic set of matrix operations and other related algorithms. These algorithms mainly involve the repeated application of relatively simple operations with regular data flow.

The systolic array configurations, investigated in the previous Chapter, exploit these properties for the parallel execution of those recursive algorithms in a synchronized and regular mode.

The systolic array of processors, however, requires global synchronization, a fact which would generally cause difficulties in its VLSI implementation. The introduced approach herein bypasses such global synchronization requirements by adopting the notion of continuously advancing waves of data and computational activity, resembling a physical wave propagation phenomenon.

In the previous Chapter the hex-connected network of processors was utilized to solve the matrix multiplication problem. We may consider a different scheme for matrix multiplication involving an orthogonal network of IFSP's. In particular, let us consider \( A = (a_{ij}) \) and \( B = (b_{ij}) \), and their product \( C = A \times B = (c_{ij}) \) all be \((n \times n)\) matrices. If we decompose both factor matrices correspondingly into columns \((A_i)\) and rows \((B_j)\), then,

\[
C = A_1 B_1 + A_2 B_2 + \ldots + A_n B_n .
\]  

(VII.2:1)

Thus, the matrix multiplication can be carried out in \(n\) recursions, executing

\[
C^k = C^{(k-1)} + A_k B_k .
\]

(VII.2:2)
recursively for \( k=1,2,\ldots,n \).

It becomes apparent how the parallelism can be exploited having \((n \times n)\) processing elements available, and hence it is almost trivial the parallel algorithm for this case.

In terms of VLSI design, the topology of such an algorithm, with this degree of localized interconnections and data flow, can be naturally mapped to a square \((n \times n)\) matrix array, illustrated in Figure (VII.2-f1), by using the computational wavefront concept.

---

**Figure VII.2-f1:** The Configuration for a \((n \times n)\)-Square Wavefront Array Processor (WAP).
For the purpose of the matrix multiplication problem each wavefront in the network of processors will correspond to a mathematical recursion in the algorithm. Hence, successive pipelining of the wavefronts will accomplish the computation of all recursions in the algorithmic process.

Let us now exemplify the first recursion for this problem supposing that the registers of all the PE's are initially set to zero, i.e., $C_{1j}^{(0)} = 0$, for all $(i,j)$. The elements of matrix $A$ are stored in the memory modules on the left (in columns), while the elements of matrix $B$ in the memory modules on the top (in rows).

The process commences within the top northwest PE(1,1) as:

$$C_{11}^{(1)} = C_{11}^{(0)} + a_{11}b_{11} = a_{11}b_{11}.$$ 

The end of this computation activates the successor neighbours: PE(1,2) and PE(2,1), which will execute (in parallel):

$$C_{12}^{(1)} = C_{12}^{(0)} + a_{11}b_{12} = a_{11}b_{12}$$

and

$$C_{21}^{(1)} = C_{21}^{(0)} + a_{21}b_{11} = a_{21}b_{11}.$$ 

After this computation, their corresponding successor neighbours: PE(1,3), PE(2,2), and PE(3,1) will be activated, thus creating a wave of hierarchical computations travelling down the orthogonal network of processors. It may be noted, however, that wave propagation implies localized data flow.

Once the wavefront sweeps through all the cells, the first recursion has been completed.

The inherent parallelism lies in the fact that synchronously with the unfolding of the first wave and immediately after the emergence of the first front of computation we can propagate the second wave (i.e., recursion of the algorithm), and then the third wave, and so on, until the matrix multiplication problem is computed.
For instance, in the second wave the PE(1,j) will execute:
\[ C^{(2)}_{ij} = C^{(1)}_{ij} + a_{12}B_{2j}, \]
and so on.

The pipelining of these computations is feasible because the wavefronts will never intersect (Huyghen's wavefront principle), assuming that they will be using different processors and bypassing any contention situations. The overall machine architecture is essentially the multi-processor lattice with additional memory modules on the north and west edges of the lattice. From the PE's architectural aspect, they are being built out of conventional LSI modules as illustrated in [KUNG82a].

Finally, to summarize the key advantages claimed by S.Y. Kung and his co-workers for the wavefront concept are:

1) It drastically reduces the complexity of describing parallel algorithms for matrix computations,

2) the wavefront language (developed for the machine) allows the WAP to be programmable and increases its applicability range,

3) the wavefront language makes it possible to simulate and hence verify parallel algorithms, and

4) the processors have an asynchronous waiting capability which obeys the Huyghen's principle that wavefronts can never intersect.

To conclude, the initiative of this network of processors in our investigation, independently to S.Y. Kung and his co-workers, was motivated as the means for the direct hardware implementation of the alternate butterfly matrix procedure, established by Evans and Hatzopoulos in [EVANG9] as an efficient parallel linear system solver. To be more specific, the attempt to complement the systolic implementation of the

\[ \text{Namely, for compact dense (n\times n) matrices.} \]
LU-factorization of general banded linear systems, investigated in the previous Chapter, applying the 'rotate' and 'fold' technique, led us directly to the single stage dewavefront concept (note the similarity in glossary) and a simpler issue of the previous technique.

\[\text{It should be reminded that the degree of interference of the opposite factorization streams, and hence the complexity of the process, was increasing analogously with the increase of the semi-bandwidth} \ p.\]
VII.3 ON THE SOLUTION OF LINEAR SYSTEMS APPLYING THE QUADRANT INTERLOCKING FACTORIZATION - QIF METHOD

Consider the following set of linear equations

\[ Ax = b, \quad (VII.3:1) \]

where \( A \) is a non-singular compact dense \((n \times n)\) matrix, \( x \) is an unknown \((n \times 1)\) column matrix and \( b \) is a given \((n \times 1)\) column matrix.

The alternate matrix procedure will now be introduced, which is based on the factorization of the coefficient matrix into components which are of butterfly form, i.e., interlocking matrix quadrants. More specifically, in this direct method, which is equivalent to a \((2 \times 2)\) block LU-factorization, we consider the matrix \( A \) is factorized into two matrices \( W \) and \( Z \), of the form,

\[
W = \begin{bmatrix}
1 & 1 & 0 & \cdots & 0 \\
0 & w_{21} & 1 & \cdots & 0 \\
0 & w_{31} & 0 & \cdots & 1 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & w_{n-2,1} & 0 & \cdots & 1 \\
w_{n-1,1} & 0 & \cdots & 1 & 0 \\
0 & 1 & \cdots & 1 & 1 \\
0 & 0 & \cdots & 1 & 1
\end{bmatrix}, \quad (VII.3:2)
\]

and

\[
Z = \begin{bmatrix}
z_{11} & z_{12} & z_{13} & \cdots & z_{1,n-2} & z_{1,n-1} & z_{1n} \\
z_{22} & z_{23} & 0 & \cdots & z_{2,n-2} & z_{2,n-1} & 0 \\
z_{33} & 0 & \cdots & \cdots & \cdots & 0 & 0 \\
0 & 0 & \cdots & \cdots & 0 & 0 & 0 \\
z_{n-2,3} & \cdots & z_{n-2,n-2} \\
z_{n-1,2} & z_{n-1,3} & \cdots & z_{n-1,n-2} & z_{n-1,n-1} & 0 \\
z_{n1} & z_{n2} & z_{n3} & \cdots & z_{n,n-2} & z_{n,n-1} & z_{nn}
\end{bmatrix}, \quad (VII.3:3)
\]
and the following relationship holds, i.e.,

\[ A = WZ. \]  

(VII.3:4)

The matrices can be seen to possess an interlocking quadrant appearance with matrix structures similar to the capital letters W and Z.

The matrices W and Z can be written in the compact form shown below,

\[ W = [W_1, W_2, \ldots, W_n] \]

and

\[ Z^T = [Z_1, Z_2, \ldots, Z_n], \]

where \( W_1 \) and \( Z_1 \), \( i = 1(1)n \) are the column vectors of the matrices W and \( Z^T \), which are of the following general forms:

(i) For \( n \)-odd

\[ W_1 = \begin{cases} [0,0,\ldots,0,1,w_{1+1},\ldots,w_{n-1},1,O,\ldots,0]^T, i=1(1)\left[\frac{n-1}{2}\right], \\ 1 \\ \end{cases} \]

\[ (VII.3:6) \]

\[ Z_1 = \begin{cases} [0,0,\ldots,0,1,0,\ldots,0]^T, i=1(1)\left[\frac{n+1}{2}\right], \\ 1 \\ \end{cases} \]

\[ (VII.3:7) \]

(ii) for \( n \)-even,

\[ W_1 = \begin{cases} [0,0,\ldots,0,1,w_{1+1},\ldots,w_{n-1},1,O,\ldots,0]^T, i=1(1)\left[\frac{n}{2}-1\right], \\ 1 \\ \end{cases} \]

\[ (VII.3:8) \]

\[ Z_1 = \begin{cases} [0,0,\ldots,0,1,0,\ldots,0]^T, i=1(1)\left[\frac{n}{2}\right], \\ 1 \\ \end{cases} \]
and

\[
Z \equiv \begin{cases} 
[0,0,\ldots,0,z_{11},\ldots,z_{1,n-1+1},0,\ldots,0]^T, & l=1(1)\frac{n}{2} \\
[0,0,\ldots,0,z_{1,n+1},\ldots,z_{11},0,\ldots,0]^T, & l=[\frac{n+1}{2}]+1(1)n.
\end{cases}
\]  

(VII.3:9)

The elements of the matrices \( W \) and \( Z \) can be evaluated in \( \lfloor (n-1)/2 \rfloor \) distinct stages. From the given forms of the matrices \( W \) and \( Z \) in (VII.3:6,7,8,9) and the equality (VII.3:4), it can easily be observed that the values of the elements of the first and last rows of the matrix \( Z \) are as follows:

\[
Z_{1l} = a_{1l} \quad \text{and} \quad Z_{nl} = a_{nl}
\]

for \( l=1(1)n \).  

(VII.3:10)

The elements of the first and last columns of the matrix \( W \) are then evaluated by solving \( (n-2) \) sets of \( (2\times2) \) linear systems given by

\[
\begin{align*}
Z_{ll}w_{ll} + Z_{nl}w_{ln} &= a_{ll} \\
Z_{ln}w_{ll} + Z_{nn}w_{ln} &= a_{ln}
\end{align*}
\]

for \( l=2(1)[n-1] \).  

(VII.3:11)

This then completes the first stage of the factorization process, whereas in preparation for the next stage the elements of the matrix \( A \) are modified according to the following formula:

\[
a_{ij} = a_{ij} - w_{il}z_{lj} - w_{ln}z_{nj}, \quad \text{for } i,j=2(1)[n-1].
\]

(VII.3:12)

In general, at the \( i^{th} \) stage of the factorization process we have the relationships,

\[
Z_{ij} = a_{ij} \quad \text{and} \quad Z_{n-i+1,j} = a_{n-i+1,j}
\]

for \( j=i(1)[n-1+1] \).  

(VII.3:13)

and the solution of the \((2\times2)\) linear systems:
\[
\begin{align*}
Z_{i1}w_{j1} + Z_{n-i+1,1}w_{j1,n-i+1} &= a_{j1} \\
Z_{1,n-i+1}w_{j1} + Z_{n-i+1,n-i+1}w_{j1,n-i+1} &= a_{j,n-i+1}
\end{align*}
\]
for \( j = [i+1](1)[n-1] \),
\[\text{(VII.3:14)}\]

to give the unknown quantities \( w_{j1}, w_{j1,n-i+1} \), for \( j = [i+1](1)[n-1] \),
and finally the modified \( a_{j} \)'s are evaluated from the formula:
\[
a_{k\ell} = a_{k\ell} - w_{k1}z_{1\ell} - w_{k,n-i+1}z_{n-i+1,\ell}, \text{ for } k, \ell = [i+1](1)[n-1].
\]
\[\text{(VII.3:15)}\]

In order to solve the system \( (\text{VII.3:1}) \) by the QIF method it can be seen from \( (\text{VII.3:4}) \) that the system
\[
(WZ)x = b,
\]
\[\text{(VII.3:16)}\]
can be solved instead of \( (\text{VII.3:1}) \). Hence, we need to solve two related and simpler linear systems of the form:
\[
W_y = b\quad \text{(VII.3:17)}
\]
and
\[
Zx = y.\quad \text{(VII.3:18)}
\]

The system \( (\text{VII.3:17}) \) is first solved for the intermediate vector \( y \) and then the final solution \( x \) of the system \( (\text{VII.3:1}) \) can be obtained by solving the linear system \( (\text{VII.3:18}) \).

The solution of the linear system \( (\text{VII.3:17}) \) can be obtained in \( [(n-1)/2] \) steps, with the evaluation procedure carried out in pairs from the top and bottom of the vector \( y \), i.e., the \( y_1 \) and \( y_n \) are evaluated first, then the \( y_2 \), \( y_{n-1} \), and so on. To typify the computational process, in general, at the \( i \)th stage \( (i=1, 2, \ldots, [(n-1)/2]) \) we have:
\[
\begin{align*}
y_1 &= b_1 \\
y_{n-1+1} &= b_{n-1+1}
\end{align*}
\]
\[\text{(VII.3:19)}\]
and
$(ii) \quad b_j = b_j - w_j y_{j} - w_{j,n-l+1} y_{n-l+1}^j, \quad \text{for } j = [l+1](-1)l[1-n], \quad (VII.3:20)$

and then we proceed to the next stage.

For the solution of the linear system $(VII.3:18)$ we distinguish again the cases that $n$ is an odd or even number.

If $n$ is odd, then we can find that

$$x_k^j = y_k^j/z_k^j, \quad \text{for } k = (n+1)/2,$$  

$(VII.3:21)$

and in preparation for the next stage we compute:

$$y_j = y_j^j - x_k^j z_k^j, \quad \text{for } j = [l](1)n \text{ and } j \neq (n+1)/2. \quad (VII.3:22)$$

The remaining elements of the vector $x$ can again be evaluated in pairs by solving $(n-1)/2$ sets of $(2 \times 2)$ linear systems in $(n-1)/2$ distinct stages.

In general, at the $i^{th}$ stage we solve the following $(2 \times 2)$ linear system:

$$\begin{align*}
&z_{11} x_{1i} + z_{i,n-1+1} x_{n-i+1} = y_i, \\
&z_{n-1+i,1} x_{1i} + z_{i,n-1+1} x_{n-i+1} = y_{n-i+1},
\end{align*}$$

$(VII.3:23)$

for $i = [l](-1)l$.

To compute $x_{1i}$ and $x_{n-i+1}$. We then set

$$y_j = y_j^j - x_k^j z_k^j - x_{n-l+1}^j z_{j,n-l+1}, \quad \text{for } j = [l](1)[l-1] \text{ and } [n-l+2](1)n.$$

$(VII.3:24)$

and proceed to the next stage.

On the other hand, if $n$ is even, then all the components of the vector $x$ are found in pairs. To find all the pairs, the linear system $(VII.3:23)$ and the formula $(VII.3:24)$ are executed, respectively, for $i = n/2(-1)l$.

To conclude, in the hardware implementation of the QIF method utilizing two-dimensional single stage computational dewavefronts, we shall refer to some alternative solution strategies for the $(2 \times 2)$ linear systems and discuss the appropriate hardware configurations to carry them out.
VII.4 SINGLE STAGE COMPUTATIONAL DEWAVEFRONTS FOR THE IMPLEMENTATION OF THE QIF ALGORITHM

As a summarizing prologue to the computational dewavefront concept introduced herein, a WAP is a computing network possessing the following features:

i) **Self-Timed, Data-Driven Computation:** No global clocking synchronization is necessary due to the self-timing property.

ii) **Modularity and Local Interconnection:** This is basically the same as in a systolic network of processors. The WAP, however, can be extended indefinitely without having to deal with the global synchronization problem.

iii) **Pipelining:** This is similar to the systolic network of processors.

Hence, the principal difference between the WAP and the systolic array is the data-driven property of the former network. Consequently, the temporal locality condition required in systolic arrays (i.e., it has to be at least one time-unit delay allotted, so that signal transactions from one node to the next can be completed) is no longer necessary, since there is no explicit timing reference in the wavefront arrays. By relaxing the strict timing requirement, there are many advantages to be gained, such as speed and programming simplicity.

As can be observed from (par.-VII.3), each computational stage of the factorization and solution parts of the QIF method can be generally distinguished into two sub-stages correspondingly performing:

a) The solution of \((2 \times 2)\) linear system(s), and

b) the modification of the appropriate matrix element(s).

More analytically, the QIF algorithm can be expressed in a
recursively (single stage) regular manner for its VLSI implementation to obtain the following homogenized steps:

i) Factorization process: At each computational stage 1 we have to solve \((n-2i)\) \((2\times2)\) linear systems to evaluate the corresponding \(w_{1j}\)'s (moving, at each step, one column inwards from each side of the matrix \(A\)), and to perform the modification of the \((n-2i)^2\) elements \(a_{1j}\) of the inner square (at every stage) of the coefficient matrix \(A\), ending up with a central peak matrix element (i.e., \(n\) is odd), or a \((2\times2)\) central submatrix (i.e., \(n\) is even).

ii) Solution of the system: \((\text{VII.3.17})\): In this part of the butterfly algorithm we have the analogous modification of \((n-2i)\) \(b_i\)'s (moving, at each computational step 1, one row inwards from the top and bottom of the r.h.s. vector \(b\)).

iii) Solution of the system: \((\text{VII.3.18})\): For the final solution \(x\) of the system \((\text{VII.3.1})\) we have to solve at every stage a \((2\times2)\) linear system and to modify all the outer-positioned \(y_i\)'s (moving, at each computational step 1, one row outwards to the top and bottom of the r.h.s. vector \(y\)).

Prior to exemplifying numerically the above process, let us introduce, in brief, two alternative solution strategies for the \((2\times2)\) linear systems:

i) **Cramer's Rule:** Each of the \((2\times2)\) linear systems can be solved by the following procedure (applied, for instance, to the \((2\times2)\) systems given by \((\text{VII.3.14})\)):

\[ R_1 = \frac{z_{11} z_{n-1+1,n-1+1} - z_{n-1+1,l} z_{l,n-1+1}}{R_3, R_2} \]

\(\text{With an exception for the case that } n\ \text{is odd.}\)
\[ R_2 = a_{j_1} z_{n-1,l,n-1+l} - z_{n-1,l,n-1+l} \]
and
\[ R_3 = z_{i_1} a_{j_1} z_{n-1,l,n-1+l} - a_{j_1} z_{n-1,l,n-1+l}. \]

Then, the quantities \( w_{ji} = R_2 / R_1 \) and \( w_{j,n-1+1} = R_3 / R_1 \) are computed.

Apparently, from the definition of \( R_1 \), we have that if any of the quantities
\[
R_1 = \begin{bmatrix}
z_{i_1}
& z_{n-1,l,i}
z_{1,n-1+i}
& z_{n-1,l,n-1+i}
\end{bmatrix} = 0, \quad (VII.4:1)
\]
then the method will break down. Although there are two well known pivotal strategies to avoid such an event (i.e., partial/complete pivoting), the inefficiency caused in the VLST implementation, if pivoting is to be necessary, imposes a non-singularity condition to be satisfied all the time.

ii) Gaussian Elimination: This method was fully exploited in the previous Chapter from both the software and hardware aspects.

Let us now present a numerical example where due to the similarity of the process for \( n\text{-odd}/n\text{-even} \), without loss of generality, we have chosen \( n \) to be odd.

Consider the following (5x5)-dense linear system:
\[
\begin{bmatrix}
6 & 1 & 1 & 2 & 1 \\
1 & 8 & 2 & 1 & 2 \\
1 & 1 & 5 & 1 & 1 \\
2 & 1 & 1 & 6 & 1 \\
1 & 2 & 1 & 1 & 6
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{bmatrix}
= \begin{bmatrix}
11 \\
14 \\
9 \\
11 \\
11
\end{bmatrix} \quad (VII.4:2)
\]

**Factorization Process**

In accordance with the method, the matrix \( A \) will be factorized
into two matrices $W$ and $Z$ of the following form:

$$
W = \begin{bmatrix}
1 & 0 & 0 & w_{21} & 1 & 0 & 0 & w_{25} \\
w_{31} & w_{32} & 1 & w_{34} & w_{35} \\
w_{41} & 0 & 0 & 1 & w_{45} \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

(VII.4:3)

and

$$
Z = \begin{bmatrix}
z_{11} & z_{12} & z_{13} & z_{14} & z_{15} \\
z_{22} & z_{23} & z_{24} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
z_{42} & z_{43} & z_{44} & 0 & 0 & 0 & 0 & 0 \\
z_{51} & z_{52} & z_{53} & z_{54} & z_{55}
\end{bmatrix}
$$

(VII.4:4)

**Stage 1**

The first and last rows of the matrix $Z$ are as follows:

- $z_{11} = a_{11} = 6$, $z_{12} = a_{12} = 1$, $z_{13} = a_{13} = 1$, $z_{14} = a_{14} = 2$, $z_{15} = a_{15} = 1$
- $z_{51} = a_{51} = 1$, $z_{52} = a_{52} = 2$, $z_{53} = a_{53} = 1$, $z_{54} = a_{54} = 1$, $z_{55} = a_{55} = 6$.

For the matrix $W$, i.e., the evaluation of the elements of its first and last columns, we need to solve three $(2 \times 2)$ linear systems, for $j=2(1)4$:

**Case 1 ($j=2$):**

$$
\begin{align*}
z_{11}w_{21} + z_{51}w_{25} &= a_{21} \\
z_{15}w_{21} + z_{55}w_{25} &= a_{25}
\end{align*}
$$

$$
\begin{align*}
&\Rightarrow \\
&6w_{21} + w_{25} = 1 \\
&\rightarrow
\end{align*}
$$

- $w_{21} = 4/35$
- $w_{25} = 11/35$

**Case 2 ($j=3$):**

$$
\begin{align*}
z_{11}w_{31} + z_{51}w_{35} &= a_{31} \\
z_{15}w_{31} + z_{55}w_{35} &= a_{35}
\end{align*}
$$

$$
\begin{align*}
&\Rightarrow \\
&6w_{31} + w_{35} = 1 \\
&\rightarrow
\end{align*}
$$

- $w_{31} = 1/7$
- $w_{35} = 1/7$
j=4:
\[
\begin{align*}
\begin{pmatrix}
z_{11}w_{41} & + & z_{51}w_{45} & = a_{41} \\
\end{pmatrix} & + \begin{pmatrix}
6w_{41} & + & w_{45} & = 2 \\
\end{pmatrix} \Rightarrow w_{41} = 11/35 \\
\begin{pmatrix}
z_{15}w_{41} & + & z_{55}w_{45} & = a_{45} \\
\end{pmatrix} & + \begin{pmatrix}
w_{41} & + & 6w_{45} & = 1 \\
\end{pmatrix} \Rightarrow w_{45} = 4/35 .
\end{align*}
\]

Modification of the \(a_{k\ell}'s\) [for \(k,\ell=2(1)4\)]

\(k=2/\ell=2(1)4:\)
\[
\begin{align*}
a_{22} & = a_{22} - w_{21}z_{12} - w_{25}z_{52} \iff a_{22} = 254/35 \\
a_{23} & = a_{23} - w_{21}z_{13} - w_{25}z_{53} \iff a_{23} = 55/35 \\
a_{24} & = a_{24} - w_{21}z_{14} - w_{25}z_{54} \iff a_{24} = 16/35 \\
\end{align*}
\]

\(k=3/\ell=2(1)4:\)
\[
\begin{align*}
a_{32} & = a_{32} - w_{31}z_{12} - w_{35}z_{52} \iff a_{32} = 4/7 \\
a_{33} & = a_{33} - w_{31}z_{13} - w_{35}z_{53} \iff a_{33} = 33/7 \\
a_{34} & = a_{34} - w_{31}z_{14} - w_{35}z_{54} \iff a_{34} = 4/7 \\
\end{align*}
\]

\(k=4/\ell=2(1)4:\)
\[
\begin{align*}
a_{42} & = a_{42} - w_{41}z_{12} - w_{45}z_{52} \iff a_{42} = 16/35 \\
a_{43} & = a_{43} - w_{41}z_{13} - w_{45}z_{53} \iff a_{43} = 20/35 \\
a_{44} & = a_{44} - w_{41}z_{14} - w_{45}z_{54} \iff a_{44} = 184/35 . \\
\end{align*}
\]

**Stage 2**

The first and last rows of the inner formatted matrix of \(Z\) are as follows,
\[
\begin{align*}
z_{22} & = a_{22} = 254/35, \quad z_{23} = a_{23} = 55/35, \quad z_{24} = a_{24} = 16/35 \\
z_{42} & = a_{42} = 16/35, \quad z_{43} = a_{43} = 20/35, \quad z_{44} = a_{44} = 184/35 . \\
\end{align*}
\]

For the corresponding inner first and last columns of matrix \(W\) we need to solve the following \((2\times2)\) linear system \(j=3(1)3\):

\(j=3:\)
\[
\begin{align*}
z_{22}w_{32} + z_{42}w_{34} & = a_{32} \quad \begin{pmatrix}
\frac{254}{35} & w_{32} + \frac{16}{35}w_{34} = \frac{4}{7} \\
\end{pmatrix} \Rightarrow w_{32} = 762/10541 \\
z_{24}w_{32} + z_{44}w_{34} & = a_{34} \quad \begin{pmatrix}
\frac{16}{35}w_{32} + \frac{184}{35}w_{34} = \frac{4}{7} \\
\end{pmatrix} \Rightarrow w_{34} = 17/166 .
\end{align*}
\]
Modification of the $a_{k\ell}$'s [for $k, \ell=3(1)3$]

$k=3/\ell=3$ (middle element):

$$a_{33} = a_{33} - w_{32}z_{23} - w_{34}z_{43} \quad \Rightarrow \quad a_{33} = \frac{47879}{10541}.$$ 

Hence, the matrices $W$ and $Z$ after the evaluation of their elements are as follows:

$$W = \begin{bmatrix}
1 & 0 & 0 \\
4/35 & 1 & 0 \\
1/7 & 762/10541 & 17/166 \\
11/35 & 0 & 1 \\
0 & 1
\end{bmatrix} \quad (\text{VII.4:5})$$

and

$$Z = \begin{bmatrix}
6 & 1 & 1 & 2 & 1 \\
254/35 & 55/35 & 16/35 \\
0 & \frac{47879}{10541} & 0 \\
16/35 & 20/35 & 184/35 \\
1 & 2 & 1 & 1 & 6
\end{bmatrix} \quad (\text{VII.4:6})$$

**Solution Process**

We need to solve the two resulting and related linear systems given by (VII.3:17,18), respectively. The former system is first solved for the intermediate vector $y$ and then the final solution $x$ of the original system (VII.3:1) can be obtained by solving the linear system (VII.3:18).

- Solution of the system: $Wy=b$

**Stage 1**

In accordance with (VII.3:18) we have:

$$y_1 = b_1 = 11, \quad y_5 = b_5 = 11.$$
Modification of the b_j's [for j=2(1)4]

\[ j=2 : b_2 = b_2 - w_1 y_1 - w_5 y_5 \quad \Rightarrow \quad b_2 = 65/7 \]

\[ j=3 : b_3 = b_3 - w_1 y_1 - w_5 y_5 \quad \Rightarrow \quad b_3 = 41/7 \]

\[ j=4 : b_4 = b_4 - w_1 y_1 - w_5 y_5 \quad \Rightarrow \quad b_4 = 44/7. \]

**Stage 2**

\[ y_2 = b_2 = 65/7, \quad y_4 = b_4 = 44/7. \]

Modification of the b_j's [for j=3(1)3]

\[ j=3 : b_3 = b_3 - w_2 y_2 - w_4 y_4 \quad \Rightarrow \quad b_3 = \frac{47879}{10541}. \]

: Solution of the system: Zx=y

**Stage 1**

Since \( n \) is odd, we have that

\[ j=3 : \quad x_3 = \frac{47879}{10541} / \frac{47879}{10541} \quad \Rightarrow \quad x_3 = 1. \]

Modification of the y_j's [for j=1(1)5 & j\neq3]

\[ j=1 : \quad y_1 = y_1 - x_3 z_1 \quad \Rightarrow \quad y_1 = 10 \]

\[ j=2 : \quad y_2 = y_2 - x_3 z_2 \quad \Rightarrow \quad y_2 = \frac{270}{35} \]

\[ j=4 : \quad y_4 = y_4 - x_3 z_4 \quad \Rightarrow \quad y_4 = \frac{200}{35} \]

\[ j=5 : \quad y_5 = y_5 - x_3 z_5 \quad \Rightarrow \quad y_5 = 10. \]

Now, for each value of \( i=2(-1)1 \), we need again to solve a \((2\times2)\) linear system and modify the corresponding outerlying y_j's.

**Stage 2**

\[ i=2 : \]

\[
\begin{align*}
\frac{z_{22} x_2 + z_{24} x_4}{35} &= y_2 \\
\frac{z_{42} x_2 + z_{44} x_4}{35} &= y_4
\end{align*}
\]

\[ \frac{254}{35} x_2 + \frac{16}{35} x_4 = \frac{270}{35} \quad \Rightarrow \quad x_2 = 1 \]

\[ \frac{16}{35} x_2 + \frac{184}{35} x_4 = \frac{200}{35} \quad \Rightarrow \quad x_4 = 1. \]
Modification of the $y_j$'s [for $j=1(1)1$ & $j=5(1)5$]

$\begin{align*}
\text{Stage 3} \\
\text{1=1:} & \quad y_1 = y_1 - x_2 z_12 - x_4 z_{14} \iff y_1 = 7 \\
\text{1=5:} & \quad y_5 = y_5 - x_2 z_{52} - x_4 z_{54} \iff y_5 = 7.
\end{align*}$

Note that, the solution vector $x$ was pre-arranged to be $x=(1,1,1,1,1)^T$.

In the same procedural sequence we shall now introduce the appropriate hardware configuration to solve the $(2 \times 2)$ linear systems and the dewavefront concept to take care of the modification at each computational stage of the appropriate matrix element(s).

The simpler way to solve the $(2 \times 2)$ linear systems is by applying Cramer's rule under the non-singularity constraint imposed for their determinants. For the clear understanding of the appropriate hardware to be introduced and, without loss of generality, for the instance of the $(2 \times 2)$ linear systems given by (VII.3:14) we have in matrix form:

\[
\begin{bmatrix}
z_{11} & z_{n-i+1,n} \\
z_{i,n-1+1} & z_{n-i+1,n-1+1}
\end{bmatrix}
\begin{bmatrix}
w_{j1} \\
w_{j,n-1+1}
\end{bmatrix}
= 
\begin{bmatrix}
a_{j1} \\
a_{j,n-i+1}
\end{bmatrix},
\]

for $j=[1+1)(1)[n-1]$.

In accordance with Cramer's rule

\[
D_i = \det \begin{bmatrix}
z_{ii} & z_{n-i+1,n} \\
z_{i,n-1+1} & z_{n-i+1,n-1+1}
\end{bmatrix} \neq 0,
\]

and

\[
\begin{bmatrix}
w_{j1} \\
w_{j,n-1+1}
\end{bmatrix}
= \frac{1}{D_i} \begin{bmatrix}
z_{n-i+1,n-i+1} & -z_{n-i+1,n} \\
-z_{i,n-1+1} & z_{ii}
\end{bmatrix}
\begin{bmatrix}
a_{j1} \\
a_{j,n-i+1}
\end{bmatrix},
\]

for $j=[1+1)(1)[n-1]$.
For the direct hardware computation of the involved determinants we shall utilize a special type of cells, which we call determinant cells. A determinant cell is a cell with four inputs $a,b,c,d$, which can compute the determinant '$ad-bc$'. For the physical realization of such a cell, two multipliers, one subtractor and some latches are needed. The two multipliers are controlled by the same internal clock and hence the subtraction can be pipelined with the two multiplications, i.e., it can commence when the first bits of '$ad$' and '$bc$' are available. Thus, the cycle delay of a determinant cell is exactly the same as the one of an IPS cell.

In Figure (VII.4-f1) is presented the general hardware configuration (note the high degree of interconnection regularity) to perform Cramer's rule, illustrating as an example the solution for the system (VII.4:9).

Now, returning to the QIF process, the difficulty was in the modification at the end of each computational stage of the general matrix elements $a_{ij}$. This was because we had to form two products, which implied a recursive procedure (very complicated) inside the same stage if we were to use the same type of hex-connected systolic network of processors that was used for the destream LU-factorization. By simplifying, however, the matrix multiplication process using the wavefront concept we obtained the means to perform the double product very easily.

The proposed network is similar to that used for matrix multiplication, but it will consist of $(n-2)^2$ processors (instead of $n^2$) and all its boundary cells will be locally interconnected with dedicated memory modules (registers) communicating with the host computer system.

For the instance of the QIF factorization process, since the action,
Figure VII.4-f1: Hardware Configuration for a (2x2) Linear System Solver Using Cramer's Rule.
at each computational stage, is moving inwards by one row and column from each side of the square matrix, each time the appropriate entries of the matrix W will be accordingly stored in the vertical memory modules and the appropriate entries of the matrix Z accordingly stored in the horizontal memory modules. Hence, any broadcasting either from the computation of the w's or the z's will be driven to the appropriate dedicated registers.

There is a slight difference, however, between the modules dedicated to the w values (in this case) compared with the rest, in that these registers will change to negative every value entering the network of processors.

The dewavefront term is self-explanatory and implies that the single stage process concept is similar to that of the matrix multiplication, but instead of a single wave originating from the top northwest cell and travelling downwards, we have an additional concurrent wave\(^\dagger\) originating from the bottom southeast cell and travelling in the opposite direction. The pictorial representation of the opposite travelling action, having flexibly gathered computational dewavefronts at different stages in a pipelining sequence on the orthogonal network of processors, is illustrated in Figure (VII.4-f2). Note the 'dummy' occurrence of each cell for the sake of the clear perception of the dewavefronts travelling action, and the variability of \(^\ddagger\) due to the different overall time requirement of each (2×2) linear systems solution and modifications stage.

As an example, let us consider the instance of a (5×5) matrix, i.e.,

\(^\dagger\) As for the dewave LU-factorization.

\(^\ddagger\) It is not the time-unit of the preceding arithmetic operation in this case.
FIRST WAVE :    -----    
SECOND WAVE :    ------    

\[ \Delta = \text{Time-Unit of Data Transfer} \]
\[ T = \text{Time-Complexity (2x2 Linear System Solver)} \]

*Figure VII.4-f2*: The Propagation of Two-Dimensional Computational 'Dewavefronts'.
of odd size, and examine paradigmatically for the remaining occurring cases the computational dewavefront for the first stage of the factorization process.

The values of the z's in the first and last rows of the matrix 2 are known according to the formulae (VII.3:10), whereas the w's for this stage have been computed by using the hardware configuration of Figure (VII.4-f1). Hence, the horizontal memory modules will be ready to forward into the network the values of $z_{1,1}$ (top registers) and $z_{5,1}$ (bottom registers), for $i=2(1)4$, respectively. On the other hand, the vertical memory modules will be ready to forward into the network the values of $-w_{1,1}$ (left registers) and $-w_{1,5}$ (right registers), for $i=2(1)4$, respectively. Note that, the $(3\times3)$ orthogonal network of processors will be initialized (once only) with the corresponding inner $(n-2)^2$ elements $a_{ij}$ of matrix A. Also, each of the opposite computational wavefronts will have the capability to activate the successor neighbouring processors in its own propagating direction.

If we present the modified elements by $C_{ij}^{(k)}$, where $k$ is the recursion stage, then we have:

**1st Wavefront (top)**

The process starts with PE(1,1) which will execute:

$$C_{11}^{(1)} = a_{22} - w_{21} z_{12}$$

(VII.4.10)

It then activates its successor neighbouring PE's(1,2) and (2,1), which will execute:

$$C_{12}^{(1)} = a_{23} - w_{21} z_{13}$$

(VII.4.11)

and

$$C_{21}^{(1)} = a_{32} - w_{31} z_{12}$$

(VII.4.11)

They, in turn, will activate their successor neighbouring PE's(1,3), (2,2) and (3,1), creating a computational wavefront travelling down the processor array.
2nd Wavefront (bottom)

Concurrently with the commencement of the top wavefront this process starts with PE(3,3) which will execute:

\[ C_{33}^{(1)} = a_{44} - w_{45} z_{54} \]  

(VII.4:12)

The completion of this computation activates the successor neighbouring PE's(2,3) and (3,2) which will execute:

\[ C_{23}^{(1)} = a_{34} - w_{35} z_{54} \]  

(VII.4:13)

and

\[ C_{32}^{(1)} = a_{43} - w_{45} z_{53} \]  

(VII.4:14)

Hence, once the dewavefront sweeps through all the cells the modifications of the first stage will have been completed, each PE containing two negative products added to the original matrix element stored in that cell. For example, the PE(1,1) will contain:

\[ C_{11}^{(2)} = a_{22} - w_{21} z_{12} - w_{25} z_{52} \]  

(VII.4:15)

We note that the computational wavefronts are similar to electromagnetic wavefronts (they both obey Huygens' principle), since each processor acts as a secondary source and is responsible for the propagation of the wavefront. In addition, wave propagation implies localized data flow as well as localized control (handshaking). The pipelining is feasible because the wavefronts of two successive recursion stages will never intersect (Huygens' wavefront principle), thus avoiding any contention problems. From the hardware perspective, the desired 'separation' between two consecutive wavefronts, a fundamental principle for the matrix multiplication problem reaffirmed by the 'separators' with proper handshaking, is not necessary for the

\[ \text{A handshaked separator is a device, usually symbolized by a rhombus, whose role is to prevent any incoming data from directly passing through, until the handsaking flag signals a 'pass'.} \]
present problem due to the natural distinction between the wavefronts of different computational stages. In particular, the 'confrontation' stage of the opposite travelling wavefronts on the same cell can be resolved by using appropriate arithmetic logic unit latches or dedicated I/O control latches which will impose, at the worst case, the delay of one time-unit overall for every modification stage.

Remarks: It is possible to have wavefront propagation in several different fashions. In the extreme case of non-uniform clocking, the wavefronts are actually crooked. Hence, for the QIF algorithm to avoid such a situation due to the variability of $T$, we may consider, as a unique value for it, the value of the longest first factorization stage. What is most significant, however, is that the order of task sequencing must be correctly followed.

Since the input data streams (for each phase of the algorithm) to the hardware configuration of the (2x2) linear system solver can be straightforwardly derived from Figure (VII.4-f1), in the following we shall introduce the input data streams for the modification parts of the three different phases of the QIF algorithm for the case of the (5x5) considered example.

For the modification part of the factorization phase the data streams are depicted in Figure (VII.4-f3).
**Figure VII.4-f3:** The Data Streams for the Modification Part of the Factorization Phase (for n=5).

**Remark:** Note the initialization snapshot (once for this phase) of the processors in the network with the inner \((n-2)^2\) matrix elements \(a_{1j}\). The circle denotation in all the phases will imply the multiple modification of those elements by successive dewavefronts.

For the modification part during the solution phase of the system: \(Wy=b\), the corresponding data streams along with the initialization snapshot are depicted in Figure (VII.4-f4).
Figure VII.4-f4: The Data Streams for the Modification Part of the Forward Solution Phase (for n=5).

Remark: For the particular case when n is odd the solution for the middle element of matrix W can be obtained by using a 'single' divide cell.

Finally, in Figure (VII.4-f5) are given the data streams, along with the initialization snapshot, for the modification part during the final solution phase of the system: $Z_{x=y}$. Note that, the pairs of elements in the network of processors indicate the required initialization sequence for the successive computational stages.

$^+$Note the definition of a 'double' divide cell in Figure (VII.4-f1).
Figure VII.4-f5: The Data Streams for the Modification Part of the Backward Solution Phase (for $n=5$).

Comment: Since we have considered the notion of a programmable network of processors, the basic wavefront array can be either a square array of $(n\times n)$ PE's, a linear array of $(1\times n)$ PE's, or a bilinear array of $(2\times n)$ PE's, which are all identical and orthogonally connected. This fact can certainly reduce the amount of computational travelling especially for the degenerated cases occurring where the $(n-2)^2$ mesh of processors is partly used.

The estimation of the overall time-complexity $^+$ for the VLSI implementation of the QIF method is a very difficult task due to the distinction of this direct method into several different phases. Since

$^+$In terms of the number of time-units required.
the most complex phase to analyze is the factorization phase, we shall restrict our attention solely to that. In fact, in the worst bound case the remaining degenerated phases of the forward and backward solutions of the resulting subsystems, due to the similarity of the occurring stages, can be roughly considered as of the same time-order to this phase.

The time-complexity for the solution of all the occurring \((2 \times 2)\) linear systems on the hardware configuration introduced by Figure \((VII.4-f1)\) is given by

\[
T_{\text{sol}} = \frac{n-1}{2} \sum_{i=1}^{L} (n-2i) + 5 \left\lfloor \frac{n-1}{2} \right\rfloor .
\]  
\((VII.4:16)\)

On the other hand, for the modification part interfered in each computational stage we have a time-complexity given by

\[
T_{\text{mod}} = \frac{n-1}{2} \sum_{i=1}^{L} (2n-4).
\]  
\((VII.4:17)\)

However, the summation of formulae \((VII.4:16,17)\) would result in a quite misleading timing, since these timings refer to distinct stages added together; namely, for the orthogonal network of processors, the programming flexibility provided by the wavefront concept (similar to the flexibility of the soft-systolic space) has not yet been considered.

In particular for the network of processors, at the end of every computational stage pairs of two rows and two columns of processors become 'dormant', which implies an artificial (in programming terms) reduction, or rather 'shrinking' of the effective network size and hence computational timing. To be specific, making use of this flexibility, the formula \((VII.4:17)\) becomes
\[ T_{\text{mod}} = \sum_{i=1}^{n-1} 2^{(n-1i)} \]  

(VII.4:18)

Hence, the total number of time-units required for the completion of the factorization stage will be given by

\[ T_{\text{sol}} + T_{\text{mod}} = \sum_{i=1}^{n-1} 3^{(n-1i)} + 5 \left\lfloor \frac{n-1}{2} \right\rfloor. \]  

(VII.4:19)

Due to the simplicity of the hardware structure implementing Cramer's rule its replication would be inexpensive to take advantage of the pipelining capabilities provided by the QIF method. In other words, the second hardware structure of the \((2 \times 2)\) linear system solver will be dedicated to the bottom originated wavefront for the solution (in parallel with the top wavefront) of approximately half the number of \((2 \times 2)\) linear systems at each computational stage; thus, the overall time-complexity for the factorization phase will be sufficiently reduced by considering the overlapping, at each stage of the same phase, between the solution output of each \((2 \times 2)\) linear system and the dewavefront propagation prior to the confrontation instant. In this case the formula (VII.4:19) is modified to

\[ T_{\text{sol}} + T_{\text{mod}} = \sum_{i=1}^{n-1} 3^{(n-1i)} + 5 \left\lfloor \frac{n-1}{2} \right\rfloor + \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{3(n-1i)}{2} \right\rfloor. \]  

(VII.4:20)

*Remarks:* Note that, the I/O interface with the host computer has been excluded from our estimations since we assume as relatively ignorable the contributed time overheads. Also, we have considered direct communication links (bypassing the delay of storing the values in the memory modules) between the outputs from the hardware structure of the
(2×2) linear system solvers and the corresponding boundary PE's.

To conclude, a rough picture of the overall proposed hardware configuration for the complete implementation of the QIF algorithm is given in Figure (VII.4-f6). □
Figure VII.4-f6: The Overall Hardware Configuration for the Implementation of the QIF Algorithm.
VII.5 Discussion and Further Remarks

To simulate the phenomenon of the propagation of the waves, the processors in the orthogonal network must be 'dormant' (idle) until the computational activity and the required data via each wavefront arrive. In other terms, each PE can be considered as capable of being in any of the three states: active, dormant, and disabled.

The dormant state implies that the PE in that state is 'waiting' to be activated, and thus change to the active state, by an oncoming computational wavefront, since processing can be performed only by active PE's.

A disabled PE remains unaffected by the activity wavefronts, being in effect 'dead', until it is 'woken' out of that state by a special control wavefront. Note the significance of the last state due to the fact that some applications (like the QIF algorithm) need a 'shrinking' in the effective size of the array of PE's.

Since each PE in the wavefront, in accordance with Huyghen's first principle, behaves as a secondary source capable of activating its neighbouring two processors in the travelling direction of the wave, implies the alteration of the neighbours' state from dormant to active. At the same time the PE is able to deactivate itself and thus cause a 'movement' of the computational front.

In accordance with Huyghen's second principle the wavefronts must never intersect, so to ensure that a PE will not attempt the activation of an already active neighbouring PE. Instead, the PE will 'wait' for the previous (or opposite in the confrontation instances of the QIF algorithm) wavefront to leave the neighbours and make them dormant, before it propagates the new wavefront and reactivates them. Thus, the
wavefront concept can be preserved. Note that, all these capabilities must be built into the PE hardware (see S.Y. Kung, et al [KUNG82a]), as well as the interpretation of its instruction set.

Albeit a PE may be active, it has to 'wait' for the data front to arrive before it starts the actual processing. For the implementation of such a wait state, the PE's must be provided with data transfer buffers. Therefore, a 'fetching' of data involves an inherent 'waiting' for the buffer to be filled by the adjacent PE. On the other hand, the latches on each entrance to the PE's will ensure that a processor cannot send new data to the buffer unless the old data has been used by the receiver neighbour. The waiting property for wavefronts of data and activation allow for globally asynchronous operation of the PE's in the network.

Finally, in concern with the dynamic 'shrinking' of the effective size of the array of PE's, mentioned in (par.-VII.4), it can be achieved by a special 'global' program in the 'Matrix Data-Flow Language' - MDFL, a wavefront-oriented language introduced by S.Y. Kung, et al in [KUNG82a].

From the aspect of the QIF method itself two other versions of it have been presented in an attempt to further improve it.

The first modified QIF method (see Evans and Hadjidimos [EVAN80]) although it possesses many of the basic characteristics of the previous one, it can also cope successfully with the parallel numerical solution of linear systems with a real symmetric positive-definite coefficient matrix A, by using an analogous procedure to the square root free Choleski type factorization. This feature is not one of the characteristics of the QIF method investigated herein. In addition, the new method can very easily produce a parallel algorithm for the evaluation
of the determinant of a matrix $A$.

A further improvement of the above modified QIF method was introduced in (Evans, Noutsos, and Hadjidimos [EVAN81]). The new method was in many respects at least as good as the one introduced by Sameh and Kuck [SAME75], which, in turn, was based on a parallel LU-type factorization.

To conclude, in future research on direct hardware systolic/wavefront implementations these versions of the QIF method will be investigated. We shall be primarily seeking for a possible soft or hardware reconfiguration of the appropriate central part of the hex-connected systolic network of processors, for an area efficient dewavefront-like factorization of the central compact dense formatted square submatrix, due to the 'rotate' and 'fold' of the original coefficient matrix $A$, for a fully parallel solution of general banded linear systems.
GENERAL COMMENTS ON FUTURE COMPUTER ARCHITECTURES, OVERVIEWING CONCLUSIONS, AND FURTHER RESEARCH
From the very outset, the central role of information processing systems in all walks of life has been the major driving force behind the evolution of computer generations.

The popular concepts of a computer as an information processing machine, of data as strings of symbols manipulated by a computer at speeds not emulatable by the people and interpreted by the user, and of programs which transform a general-purpose machine into one dedicated to a particular range of tasks, are the products of some of the most brilliant minds of the twentieth century. Thus, we have come from the Entschiedungsproblem - the problem of a definite method - to the proliferation of computing in the early 1980s.

In the past computers have developed more processing power, firstly, by the increase in the density of integration, and, secondly, by the operational acceleration of the basic switching elements. Both these methods, however, lead to a higher power dissipation per unit area to which there are fundamental thermodynamic limits.

For a further improvement of the performance attainable by present day computers technologically, it is likely that radical new developments
must take place. Some of the envisaged technological advancements include logic circuits from Gallium Arsenide and Superconducting Josephson Junctions.

_Gallium arsenide_ can give a ten to a hundredfold speed enhancement compared to current silicon based systems. The basic material, however, is more expensive and new techniques will be necessary to obtain the large, pure crystals required to make semiconductors. Many manufacturers and universities are developing this material, which exhibits a quite high heat dissipation level for each basic transistor, and simple logic gates can now be reliably built.

_Josephson junctions_ operate only with superconducting materials (at temperatures below -268 Centigrades, 5 Kelvin), but retain a very low heat dissipation and a very simple fabrication leading to a potentially large increase in density and speed of logic circuits. Research in supercomputing technology has been undertaken by only a few manufacturers because of the very great difficulty of operating at such low temperatures (the temperature of Outer Space!). Due to this physical constraint it is only likely to be used in very costly, high-speed computer architectures at least in the foreseeable future.

_Parallel processing_, therefore, is widely viewed as the only natural and feasible way forward to achieve a significant (hundredfold or even more) increase in processing power. The basic idea is attractive - the cost of a Z80 microprocessor is about 1/100,000^{th} of the cost of a Cray Supercomputer, but only 1/5,000^{th} of the latter's processing speed. The occurring overheads, however, due to the required communication and synchronization between all the processors contribute to the difficulties arising for the parallel implementation
of certain algorithms. It is as difficult to organize a parallel computer efficiently as it is to persuade groups of people to work together efficiently.

Parallel computers, as it has been discussed in the introductory Chapters of the Thesis, are classified in various different types, each of which has an effective range of problems to which it is most suited to be capable of solving.

There are some other alternatives to attempting the parallel organization of computers built with existing technology.

In the so-called Distributed Systems, parallel processing is achieved by spreading the work between geographically separated computer systems where the intercommunication is carried out via 'Local Area Networks' - LANs and 'Wide Area Networks' - WANs, according to whether the range is less than about a mile, or hundreds/thousands of miles. In particular, two technologies are in use for LANs, the Cambridge ring and the Ethernet, but the latter is by far the most widely used commercially. The Cambridge ring system is in fact intended to comprise very many specialized services provided by computers distributed around the ring (e.g., process servers, file servers, compilers, printers, etc.).

Two much more optimistic alternatives are the Optical Computers and the Biocomputers.

The Optical Computers use light beams instead of electrons flowing along wires. Simple circuits can be built by using a material that can be set to become either transparent or opaque (e.g. Indium Antimonide). Holographic technology can be used to control the 'interconnection' of light beams between these simple switching devices and
potentially offers a very high level of parallelism, initially at least of SIMD type; it is limited only by the number of individual beams that can be focussed on one InSb wafer, and not by a complex etching process as for silicon systems.

With respect to Biotechnology the ultimate goal will be the fabrication of a remarkably special device called the biochip. The switching elements in this case will be organic substances, probably simple proteins. The switching speed will be inherently slower than for electronic systems as the protein will switch by changing either its molecular structure, or its topology. The achievable level of integration will be extremely high with a straightforward application in the three-dimensional space, rather than in two dimensions as for the current silicon systems. In fact, it was originally claimed that a biochip could accommodate the logic circuitry of all the currently existing computers worldwide, and all these in a volume of just one sugar cube; but, nowadays, they only claim about a cubic foot of volume. Such a computer will be very difficult to design and build, but its achievement may provide Computer Science with a new meaning.

To-date, in concern with the imminent next generation of computer systems, the epitome of contemporary computing - the Supercomputer - consists of banks of high-speed processors, operating in parallel on arrays of numbers or closely coupled in a pipeline, with further processors to pass the information to and from secondary storage. The total assembly achieves a very high throughput of data, and makes use of densely packed VLSI chips. The processing power of these Supercomputers continues to grow exponentially, their physical volume and power requirements decrease with each new model, and their prices reduce in real terms.
However, these machines cannot 'realize' the data they process. They do not take initiatives, nor can they make sense of fuzzy, incomplete, or contradictory information. They cannot cope with information in a natural language, and albeit they produce spectacular graphical displays, they cannot interpret information in a visual form. Their prowess is no more than the product of the sequences of instructions which control them and the data on which these instructions operate.

To-date computers have come to be the standard tools of the 'exact' sciences, i.e., Mathematics, Physics, Chemistry, and to some extent Biology. They are indispensable in all branches of Engineering, and the advances in Aeronautics and space travel in the last forty years could not have taken place without them. However, they bear no resemblance at all to 'Fifth Generation Computer Systems' - FGCS, which are meant to be used by people who are not necessarily computer specialists. Their scope will be totally different to that of the computers today, since they will be intended to move to the center of the stage in such fields as Sociology, Economics and, above all, Medicine, where the fundamental knowledge is not so precise and easily quantifiable. In these fields the highly qualified and experienced expert is the ultimate decision maker.

Much of this will change if even some of the aspirations of the FGCS are achieved.

In broad terms, a FGCS is conceived as a 'series of interconnected data base and parallel processing machines, accessed by means of an intelligent inference machine which can (amongst other things) accept problem statements in a natural language' (see Bramer [BRAM84]). Its
main function is not information processing, in the conventional sense, but drawing inferences from knowledge bases. It is to incorporate a much higher degree of intelligence than contemporary computers, approaching that of a human expert in certain circumstances. The main application area of FGCS is expected to be the solution of highly complex problems, ones which require a considerable measure of reasoning, intelligence and expertise when carried out by people.

Although the architecture of FGCS is basically different from what is in use today, the fundamental hardware technology remains the same - Very Large Scale Integration of semiconductor components. Gallium arsenide is being investigated as an alternative to silicon, and novel ways of cooling very large, densely packed chips are being tried out.

The constant reduction in the size of chip elements gives rise to enormous engineering and quality control problems at every stage of chip fabrication. At present the constraint is the width of a conducting path on a chip - currently of the order of half a micron, but likely to reduce to a quarter of a micron during the time-scale of the FGCS programme. The precision of the chip fabrication stages has to increase continually to cope with these requirements, and for future research some measure of redundancy has to be devised and incorporated into chip designs to compensate for the inevitable flaws. The ultimate limiting factor is the speed of light: The speed with which electrical signals propagate along a conductor. In order to achieve nanosecond performance, all the PE's of a computer have to be contained within a volume of a 30 cm cube (light travels approximately 30 cm in one nanosecond).

One challenging VLSI technique is "Wafer Scale Integration" - WSI
which attempts the production, onto a single wafer, of systems which are 'better' than conventional ones (realized with commercially available ICs or classical dedicated VLSI) according to the criteria of performance, cost, reliability, power consumption and compaction.

It is well known, however, that the most critical problem of WSI is yield.

The conventional chip fabrication process is to form a few hundred identical chips on a circular slice - a wafer - of silicon crystal. A seemingly obvious step forward is to utilize the entire silicon wafer for a single chip, some 5 cm square. However, the conventional chips are tested before they are separated from their wafer, and defective ones are discarded. In practice the yield varies considerably, but is seldom over 90%. On the other hand, if it is a realistic assumption to put on a wafer systems comprising 150-200 PE's each having the complexity of a 16-bit microprocessor, and if a yield of 30% is also a realistic assumption for those PE's, it is apparent that the yield of the system on the whole wafer is catastrophic. In other words, in order to be usable, a chip made by WSI must be close enough to perfection for the redundancy in its design to cope.

To summarize, WSI requires adequate design styles (regular structure, standby resources), intelligent multi-level reconfiguration facilities, simple and efficient test approaches. Regular, highly parallel data-flow systems and memories remain the most reasonable targets. If the progress in technology (switches) goes on, WSI has a good chance of being a reality in the foreseeable future. In fact, some first attempts have already been made in the 'Laboratory of Circuits and Systems' - LCS, at IMAG, Grenoble University, with the
study of the HYETI - a high yield defect-tolerant processor and the
breadboard of a systolic machine, the SYSTOLIMAG I machine, designed
to implement 'any' systolic algorithm, including matrix transformation
(e.g. Choleski and Jacobi algorithms). However, the matrix node was
rather complex causing integration difficulties, so the breadboarding
of the SYSTOLIMAG II machine is imminent at LCS/IMAG with the specifi-
cation of a new matrix with less complex nodes and more signal processing
oriented.

Also, CMU has built a 32-bit floating-point systolic array, the
so-called Warp processor, that can efficiently program many essential
computations in signal processing like the FFT and convolution. This
is a one-dimensional systolic array that, in general, takes inputs from
one end cell and produces outputs at the other end, with data and
control all flowing in one direction at a very high-speed. The CMU
prototype has 10 cells, each of which is capable of performing 10
million floating-point operations per second (i.e. 1OM flops/s) and is
built on a single board using only off-the-shelf components. This 10-
cell processor, for example, can process 1024-point complex FFTs at a
rate of one FFT every 614.4μs. Under program control it can also
perform (amongst other things) two-dimensional convolution and complex
matrix multiplication, at a rate of 100M flops.

Another key to FGCS, and as it is agreed by all concerned the
most important one, is parallel processing, but certainly in a much
higher degree than is incorporated into computers at present. It is
likely that there will be a number of layers of parallelism: Closely
coupled processing elements reflecting the parallelism inherent in
inference or knowledge base processing operations, looser coupling
between the various subsystems in a FGCS, and distributed processing across LANs and WANs.

The central theme of the Thesis is the interrelated realization of the parallel processing concepts as applied on parallel computer complexes and VLSI structures.

In general, programming parallel systems is more difficult than programming uniprocessor systems, and this has led to the parallelism being concealed on most existing MIMD prototypes. This kind of machine consisted of the primary testbed for all numerical algorithms designed and analyzed herein. The actual prototype that carried out the extensive bulk of our experiments and measures was the NEPTUNE MIMD four processors' complex at Loughborough University of Technology.

The techniques for programming this type of computers for efficient parallel operations are much less developed than the corresponding techniques for SIMD systems. This, however, does not imply that the category of problems suitable for the former type of systems can be easily implemented.

The problem which arises here lies in making the p computers consisting of the testbed to cooperate, so that one problem can be appropriately partitioned amongst them to be solved with greater speed than it could be solved on a uniprocessor.

In order to make the Multiprocessor complex effective it is vital that the speed increase is substantial, hopefully of \( \Theta(p) \), in comparison with the smallest possible sequential time-complexity achievable for the same problem when solving it by any of the relatively 'best' considered existing methods.

This can be achieved when specific attention is paid to the problem of
minimizing the synchronizations of the participant processors and the data sharing amongst them, which are directly dependent upon the overall computational scheduling.

The performance analysis of an algorithm is very significant from various aspects. Basically, it can help one to understand the algorithm better and sometimes to reveal necessary further improvements. In other words, the careful search required for a proper performance analysis often leads to more efficient and more correct implementations of algorithms. However, the more complicated the algorithm, the more difficult its performance analysis.

This fact was the major driving force for the establishment of a general DPM for the analysis of any type of algorithm on such as the NEPTUNE Multiprocessor testbed.

The principle behind this analysis is that parallel processing involves the sharing of some resources which have a limited availability. This has the consequence that there is a limit to the number of demands that can be satisfied and some of them must wait if there are some competing ones. These demands are determined by the program, while the availability and allocation algorithm are properties of the system. Obviously, the capability of a given parallel computer to meet the resource demands of a program limits only the effectiveness of the program on that computer. When the results of the analytic study come to an acceptable agreement with those of the corresponding empirical study, one can then be convinced of the validity of the algorithm, as well as the correctness of the process of analysis.

From the aspect of hardware algorithmic implementation on VLSI structures many alternatives exist for the implementation of systolic
algorithms at both chip and board levels. We can classify them along two dimensions: The interconnection topology discussed in the introductory Chapters of the Thesis (i.e., linear systolic arrays, 2-D systolic arrays), and the flexibility.

Along the latter dimension we can distinguish single-purpose systolic arrays, multi-purpose systolic arrays, non-programmable/programmable building-blocks, and programmable systolic arrays.

The most straightforward way to implement a systolic algorithm is to construct a special-purpose systolic array processor just for that specific algorithm. This approach seems reasonable if one or more of the following conditions hold: i) The performance of the processor is of ultimate importance and the use of the processor is well-understood; ii) the processor will be used in large quantities despite the fact that it is single-purpose; and iii) the design and implementation cost of the processor is low - this in fact is the case for those systolic arrays consisting of only a few types of very simple cells like some of the pattern matching and correlation arrays.

The multi-purpose systolic array processor is intended for the implementation of a predefined set of systolic algorithms. This approach is based on the observation that many systolic algorithms like those for convolution and matrix multiplication can be executed on systolic arrays of very similar structures. Hence, such systolic processors with little overheads for providing the necessary flexibility, can perform a number of functions under some simple controls.

Most cells of a large number of systolic arrays can perform common functions like 'multiply-accumulate'. Thus, it is possible to construct non-programmable building-block processors capable of executing
a few predefined and commonly used functions, and then connect them
to form a variety of systolic processors of various sizes and shapes.
An efficient building-block should meet the needs from current as well
as future systolic algorithms.

In the *programmable building-blocks* approach a large number of
systolic cells can be implemented by programming. The reader should
recall the classification of systolic algorithms into the two main sets
of $S_H$ and $S_S$. Certainly, the programmable approach is not as efficient
as the non-programmable one, due to the overheads for supporting the
programmability. Nevertheless it fulfills the need of implementing
those systolic arrays, each of which is not significant enough for
warranting individual, custom hardware implementation, but in aggregate
can be implemented cost-effectively by some programmable building-blocks.
In addition, in some systolic designs, the instruction that the cell
executes in a cycle depends, in a complicated way, on the inputs to the
cell and the cell's state during that cycle. The programmable approach
seems to be the only effective way to implement those complicated, data­
dependent systolic arrays.

Finally, in the *programmable systolic array processors* approach
a fixed number of programmable PEs are interconnected in a certain
manner, possibly with other control circuits. In particular, when a
number of programmable building-blocks mentioned above are connected
into a fixed array, they then form such a programmable systolic array.
However, it is possible for PEs not intended to be programmable building­
blocks for any other systolic array to assemble such a programmable
systolic array. Programmable systolic arrays are more flexible (most
often characterized as deceivable by present Technology) than the multi­
purpose systolic arrays in the sense that the PEs are programmable (simulatable) and sometimes even their interconnections can be configured by software control before a computation starts.

So far, the design of VLSI architectures has been more of an 'art and trial and error' activity, rather than a systematic process which, starting from the description of an algorithm, yields the specification of the optimal VLSI architecture for that algorithm (or class of algorithms). A lot of research has to be performed in exactly the opposite direction for the devise of methodologies to transform original algorithms, which can hardly be mapped under their present state onto array configurations, into other 'equivalent' algorithms for which an easy mapping exists and it allows for maximal parallelism using a minimum number of processors.

As a future investigational hint, the set of data dependencies will consist of the most important component for our VLSI design procedure based on algorithm transformations. Always driven by the final desirable form of the data communication, the matricial representations of algorithms dependencies should be closely investigated for the conception of their appropriate (unidirectional?) executional re-ordering to explore the potentiality of the VLSI structures.

The introduction of partial transformation models for the different existing classes of algorithms will be apparently the next investigation goal to be pursued, to continue with the theoretical optimization of these various models and, if possible, the unification of all models and the establishment of a general algorithm transformation model.

If such an investigation proves to be fruitful, then the next imposable step will be the simulation of all multiple processor
architectures, departing from MIMD machines, by VLSI systolic (or wavefront) array processor structures.

From the aspect of the wavefront computational concept, the major flexibility offered by its programming capabilities is that software reconfigurability can be used to map a linear or bilinear array onto a square array hardware. Therefore, a hardwired square array may be used for the purpose of linear (or bilinear) wavefront array processing.

In particular for the QIF implementation, this method was mainly pursued for future investigation in the space of optical array processors. In other words, the advantage of this powerful linear system solver is its capability to directly map onto the orthogonal network of processors. Hence, since there is a rapidly growing interest in the developments of optical array processors, from an optical interconnection perspective each computational stage of the method could be instantaneously(1) implemented (see Caulfield, et al [CAUL81] and Goodman, et al [GOOD84]).

In conclusion, it can be foreseeable that both the systolic and data-driven computing will play a major role in future supercomputing, especially for number 'crunching' problems. Most computing networks will be systematically converted into systolic or wavefront arrays following some of the already well established procedures. This fact should encourage more scientists to develop advanced hardware and software for massively parallel-array processors. The impact of the novel architectures upon future supercomputer designs cannot be overestimated.


[AVRI66] - Avriel, M., and Wilde, D.J., Optimal Search for a Maximum
with Sequences of Simultaneous Function Evaluations,

[AYRE74] - Ayres Frank, Jr., Theory and Problems of Differential
Equations, Schaum's outline series, McGraw-Hill Book Company,

[BACK78] - Backus, J., Can Programming Be Liberated from the von Neumann
Style? A Functional Style and Its Algebra of Programs, CACM,
Vol. 21, No. 8, August 1978, pp. 613-641.

[BAER68] - Baer, J.-L., and Bovet, D.P., Compilation of Arithmetic
Expressions for Parallel Computations, Proc. IFIP Congr. 1968,

[BAER73] - Baer, J.-L., A Survey of Some Theoretical Aspects of Multi-
processing, ACM Computing Surveys, Vol. 5, No. 1, March 1973,
pp. 31-80.

[BAER77] - Baer, J.-L., and Jensen, J., Simulation of Large Parallel
Systems: Modelling of Tasks, Proc. 3rd Int.Symp. on Modelling

[BAER80] - Baer, J.-L., Computer Systems Architecture, Computer Science

[BAER82] - Baer, J.-L., "Techniques to Exploit Parallelism", in Parallel
Processing Systems, ed. by D.J. Evans, Cambridge University


presented in Workshop at L.U.T., Comp. Studies Dept.,

[BARL82c] - Barlow, et al: Barlow, R.H., Evans, D.J., and Shahnch1, J.,
Performance Analysis of Algorithms on Asynchronous Parallel
233-236.

Computer, IEEE Transactions on Computers, Vol. C-17, No. 8,
August 1968, pp. 746-757.

[BATC72] - Batcher, K.E., Flexible Parallel Processing and STARAN,
1972 WESCON Tech. Papers, Session 1-Parallel Processing
Systems, Sept. 1972, pp. 1/5-1 - 1/5-3.

[BATC74] - Batcher, K.E., Multi-Dimensional Access Solid State Memory,

[BATC74a] - Batcher, K.E., STARAN Parallel Processor System Hardware,
in Proc. AFIPS 1974 National Computer Conf., Vol. 43, AFIPS

[BAUD78] - Baudet, G.M., The Design and Analysis of Algorithms for
University, Dept. of Comp. Science, (Univ. Microfilm Inter-
national, 300 N. Zeeb Rd., Ann Arbor, MI 48106- Ref. 78-15,
195), April 1978.


[CORD83] - Cordy, J.R., and Holt, R.C., Specification of Concurrent Euclid, Computer Systems Research Group, University of


[EVAN79] - Evans, D.J., and Hatzopoulos, M., A Parallel Linear System
pp. 227-238.

[EVAN80] - Evans, D.J. and Hadjidimos, A., A Modification of the
Quadrant Interlocking Factorization Parallel Method, Intern.

Solution of Banded Linear Equations by the New Quadrant
Interlocking Factorization (Q.I.F.) Method, Intern. J.

for the Solution of Burgers' Equation, Internal Report No.

[EVAN83a] - Evans, et al: Evans, D.J., Shanehchi, J., and Abdullah,
A.R.B., The Numerical Solution of Non-Linear Parabolic
Equations on Parallel Computers, Internal Report No. 184,
Dept. of Computer Studies, L.U.T., April, 1983.

[EVAN83b] - Evans, D.J., and Abdullah, A.R.B., Group Explicit Methods
for Parabolic Equations, Intern. J. Comp. Math., Vol. 14,
1983, pp. 73-105.

[EVAN83c] - Evans, D.J., New Parallel Algorithms in Linear Algebra,
E.D.F. - Bulletin de la Direction des Etudes et Recherches -


        and Athale, R., Optical Interconnections for VLSI Systems, in Proc. of the
        IEEE, Vol. 72, No. 7, July 1984, pp. 850-866; also prepared as an ARO
        Palantir Meeting Report.

[GOOD68] - Goodyear Aerospace Corp., Handbook of Operating and
        Maintenance Instructions for the Associative Memory, GAC

[GOOD74] - Goodyear Aerospace Corp., STARAN APPLE Programming Manual,

[GORS80] - Gorsline, G.W., Computer Organization, Prentice-Hall,

[GOSD66] - Gosden, J.A., Explicit Parallel Processing Description and
        Control in Programs for Multi- and Uni-processor Computers,

[GOST71] - Gostelow, K.P., Flow of Control, Resource Allocation and
        the Proper Termination of Programs, Ph.D. Thesis, Computer
        Science Dept., Univ. of California, Los Angeles, Dec. 1971,
        219pp.

[GURD83] - Gurd, J., and Watson, I., "Preliminary Evaluation of a Proto-
        type Data Flow Computer", in Information Processing 1983,


[HOLT70] - Holt, A.W., and Commoner, F., Events and Condition, Applied Data Research, N.Y., 1970; also in Record of the Project
MAC Conf. on Concurrent Systems and Parallel Computation,  


[KUNG82] - Kung, S.Y., and Gal-Ezer, R.J., Synchronous vs. Asynchronous


[LIPT75] - Lipton, R.J., *The Reachability Problem and the Boundedness Problem for Petri Nets are Exponential Space-Hard*, presented...


[MAYS84] - May, D., and Shepherd, R., "OCCAM and the Transputer", Inmos Ltd., presented in the Workshop on Hardware Supported

[MCAT64] - McAteer, J.E., Capobiano, J.A., and Koppel, R.L.,

Associative Memory System Implementation and Characteristics,


[STEV75] - Stevens, K.G., Jr., CFD - A FORTRAN-like Language for the


Kadowaki, Y., Yamamoto, M., Shimada, T., Sohma, Y.,
Takizawa, M., Ito, N., Takeuchi, A., Kitsuregawa, M., and
Goto, A., "The Preliminary Research of Data Flow Machine and
Data Base Machine as the Basic Architecture of Fifth
Generation Computer Systems", in Fifth Generation Computer
Systems, ed. by T. Moto-Oka, North-Holland Publishing Company,

Vision: Machine Perception Through Hierarchical Computation


[Texas Instruments, II & IV] - "Texas Instruments DX10 Operating System
Loughborough University of Technology, U.K.

[Texas Instruments, VI] - "Texas Instruments DX10 Operating System
Loughborough University of Technology, U.K.

[THOM77] - Thompson, C.D., and Kung, H.T., Sorting on a Mesh-Connected

[THOM80] - Thompson, C.D., A Complexity Theory for VLSI, Ph.D.
Dissertation, Carnegie-Mellon University, Computer Science
Department, August 1980, Pittsburgh, Pennsylvania.


[TUTT63] - Tuttle, G.T., How to Quiz a Whole Memory at Once, Electronics, Nov. 1963, pp. 43-46.


[VONN66] - Von Neumann, J., Theory of Self-Reproducing Automata, 

and Gehringer, E., Performance Prediction and Calibration for 
a Class of Multiprocessor Systems, Dept. of Computer Science, 


[WANG80] - Wang, P.-S., and Liu, M.T., Parallel Processing of High-
Level Language Programs, in Proceedings of the 1980 Intern. 
Conf. on Parallel Processing, Computer Society Press, 1980, 
pp. 17-25.

with Token Labelling, Proc. National Computer Conference, 

[WEID77] - Weide, B., A Survey of Analysis Techniques for Discrete 
Algorithms, Computing Surveys, Vol. 9, No. 4, December 1977, 
pp. 291-313.

[WEIT80] - Weitzman, C., Distributed Micro/Minicomputer Systems: 
Structure, Implementation, and Application, Prentice-Hall, 

[WELS79] - Welsh, J., and Bustard, D.W., Pascal-Plus—Another Language 
for Modular Multiprogramming, Software-Practice and 


APPENDIX C-I

FURTHER ANALYSIS OF
THE 'SPEED-UP' AND 'EFFICIENCY' FORMULAE
IN (par. - I.B.4.1)
Paragraph - I.B.4.1

The Speed-up, generally for a Parallel system, is defined as the quotient of the sequential run-time over the time obtained when performed in parallel. In formula (I.B.4.1:2), $T_s$ is the sequential time and $T_{pipe}$ is the pipeline time. $\sum_{i=1}^{p} \tau_i$ is the time taken for a process to be extended sequentially and consequently this is the time required for the completion of the first process in the pipeline; in addition, there is always a set-up time involved with the first process flowing in, thus allowing the pipe to become full. The bottleneck time $\tau_j$ is considered as the time limiting the output of the remaining processes, and therefore the formula:

$$\frac{T_s}{T_{pipe}} = \frac{k \cdot \sum_{i=1}^{p} \tau_i}{\sum_{i=1}^{p} \tau_i + (k-1) \cdot \tau_j}$$

is obtained.

In the ideal case, where no bottleneck exists and $\tau_j = \tau$, for $i \in [1,p]$, this formula simplifies to:

$$\frac{T_s}{T_{pipe}} = \frac{p \cdot k \cdot \tau}{p \cdot \tau + (k-1) \cdot \tau} = \frac{p \cdot k}{p+k-1}$$

(A.C-I:1)

A division of (A.C-I:1) by $k$ implies: $\frac{p}{P+k-1}$; taking the limit $(k \to \infty)$ we obtain:

$$\lim_{k \to \infty} \frac{T_s}{T_{pipe}} = \lim_{k \to \infty} \left( \frac{p}{p+k-1} \right) = \frac{p}{p+k} \cdot \frac{1}{k} \cdot \lim_{k \to \infty} \left( \frac{1}{p+k} + \frac{1}{k} \right) = \frac{p}{p+k} \cdot \frac{1}{k} \cdot \lim_{k \to \infty} \left( \frac{1}{p+k} + \frac{1}{k} \right)$$

due to $(k \to \infty)$ $\frac{P}{k}, \frac{1}{k} \to 0$.

The Efficiency for a Parallel system is, in general, defined as the quotient of the achieved Speed-up over the corresponding number of utilized processors, i.e.,
Parallel Efficiency = \frac{\text{Speed-up}}{\text{No. of proc.}}

If \( S_{\text{pipe}} \) is the achieved pipeline Speed-up, then the Efficiency of the pipeline is:

\[ E_{\text{pipe}} = \frac{S_{\text{pipe}}}{\text{No. of modules}} = \frac{\sum_{i=1}^{p} \tau_{i}}{p} = \frac{k \sum_{i=1}^{p} \tau_{i}}{p \left[ \sum_{i=1}^{p} \tau_{i} + (k-1) \tau_{j} \right]} \]

\[ E_{\text{pipe}} = \frac{k \sum_{i=1}^{p} \tau_{i}}{p \left[ \sum_{i=1}^{p} \tau_{i} + (k-1) \tau_{j} \right]} \]

\[ E_{\text{pipe}} = \frac{k \sum_{i=1}^{p} \tau_{i}}{p \left[ \sum_{i=1}^{p} \tau_{i} + (k-1) \tau_{j} \right]} \]  \hspace{1cm} (A.C-I:2)

Again, in the ideal case, where no nottleneck exists and \( \tau_{i} = \tau \), for \( i \in \{1, p\} \), formula (A.C-I:2) simplifies as:

\[ E_{\text{pipe}} = \frac{k \tau}{p \left( \tau + (k-1) \tau \right)} = \frac{k \tau}{p \tau + (k-1) \tau} = \frac{k \tau}{p + k - 1} \]  \hspace{1cm} (A.C-I:3)

At the limit \((k \to \infty)\), the Efficiency approaches the theoretical value 1, since \( \frac{p}{k}, \frac{1}{k} \to 0 \).

Another way to view the Efficiency of the pipeline is to consider the Efficiency over individual modules, i.e.,

\[ \text{Efficiency of a module} = \frac{\text{Time module } i \text{ is active}}{\text{Total time pipeline is active}} = \frac{k \tau_{i}}{\sum_{i=1}^{k} \tau_{i} + (k-1) \tau_{j}} \]

Consequently the pipeline Efficiency is:

\[ \text{Efficiency of the whole pipeline} = \frac{\sum_{i=1}^{p} k \tau_{i}}{\sum_{i=1}^{p} \tau_{i} + (k-1) \tau_{j}} = \frac{k \sum_{i=1}^{p} \tau_{i}}{p \left[ \sum_{i=1}^{p} \tau_{i} + (k-1) \tau_{j} \right]} \]

\[ \text{Efficiency of the whole pipeline} = \frac{k \sum_{i=1}^{p} \tau_{i}}{p \left[ \sum_{i=1}^{p} \tau_{i} + (k-1) \tau_{j} \right]} \]  \hspace{1cm} (A.C-I:4)
APPENDIX C-II

EXTENDED IMPLEMENTATION DETAILS

FOR (par. - II.A.3, II.B.3.1)
Paragraph - II.A.3

i) The 'NEPTUNE' parallel path execution scheme for XPFC1'

This scheme, implemented by Dr. R.H. Barlow, utilizes a shared array of path descriptor blocks† to represent the parallel paths. Each path descriptor block contains the following information:

- Start of path address
- Address of parent path descriptor block
- Index variable address
- Index variable value
- Which processor to execute a path (zero, if any)
- Count of active children;

there are 75 of these information blocks.

The initialization routine INIT (from $USEPAR construct) sets all the blocks 'empty' and sets-up a single block to describe the first sequential path in the program. All processors then enter the scheduling routine to choose a new path to execute, if no path is available, for a processor to run, it enters an 'idle' loop within the scheduler and then rescans the array (the 'wait' cycle time is 10μs, at present).

When a processor encounters a 'CALL FORK' instruction (from $DOPAR/ $DOALL/$FORK), a 'free' path descriptor block is taken from the array for each path created; if there are no 'free' blocks, then an error message is reported and the caller enters an 'idle' loop and rescans the array until sufficient new blocks have been found. The number of paths is fixed when the 'FORK' is executed, rather than in normal FORTRAN where the end and increment value, as well as, the index variable itself

†The so-called 'Task Control Blocks' - 'TCBs'.
‡A computed 'FORK'.
can be modified during the execution of the DO-loop.

For the $DOALL construct a path block is initialized for each processor executing the program, with its id set on one of the blocks so that each processor has a single path to execute.

The 'DOPAR' creates a path block for each value of the index variable and sets the value in the block. Any processor may choose a 'DOPAR' path block, and because the path descriptor array is scanned in order, the index values will be chosen in the same order as for the sequential DO-loop. When choosing a DOPAR path, the address of the loop index variable is set to the index value held on the block; this variable must, therefore, be in private memory or several processors would try to set the same location.

Because of the storage of the index variable in private memory, a problem exists with nested DOPAR constructs, because the parent path index variable may not hold the same value as that of the parent path of the current path (see example - A.C.-II:1).

```plaintext
INTEGER I,J,K
$DOPAR 30 I=1,3
$DOPAR 20 J=1,3
$DOPAR 10 K=1,2
WRITE (6,1000) I,J,K
$PAREND

(A.C-II:1)
```

This program is not correct in that a write statement is not permitted in a parallel construct, but if it was, all the combinations of I,J,K should be printed. When a processor chooses a path for the K-$DOPAR, it must set the values of I and J in its local memory, appropriate to that path. This is a result of the fact that a processor can choose

---

*Identifier

*Either '$SOPAR' or $FORK.
any 'free' path from a 'DOPAR', irrespective of the previous path executed by that processor. This being said, nested 'DOPAR' constructs are so rarely found in actual parallel programs that the overhead of setting the parent values is not important.

When a processor executes a 'CALL JOIN' (from $PARENZ/$JOIN), the count of active children is decremented on the parent block. If the count is zero, then there are no outstanding child paths and the processor that executed the 'CALL FORK' can now, when it next chooses a path, execute the code following the 'CALL JOIN'. The path descriptor block is set 'empty' and the processor enters the scheduling routine to choose a new block.

Finally, all the operations that modify the descriptor blocks are made within a critical region, controlled by the same 'CALL GETRES' (from $ENTER) and 'CALL PUTRES' (from $EXIT) that are utilized by the FORTRAN program. Resource number nine is utilized, this has the benefit that the TIMOUT routine will return the number of accesses and 'wait' cycles involved in controlling scheduling, as well as, the resources utilized by the FORTRAN programmer.

ii) The 'NEPTUNE' parallel path execution scheme for 'XPFCLD'

Although 'XPFCL' has been utilized for the majority of parallel programs, there were some deficiencies that suggested an amended version may sometimes be useful. These problems were as follows:

1) The limit of 75 path scheduling blocks was too low. Even though it has been seen that the most efficient parallel algorithms tend to utilize a much smaller number of paths, it

\*The 'termination' counter.
is useful to be able to measure the actual performance degradation as more paths are utilized.

2) The layout of the blocks as an array, rather than a list, meant that the parallel path scheduling overhead contains a term quadratic in the number of paths created. This happens because all the previously executed and now 'empty' blocks are scanned when looking for a new path to execute. A 'queue' organization of two separate lists, of 'allocated' and 'free' blocks, would reduce this effect.

3) The critical resource routines GETRES/PUTRES utilize the indivisible 'test' and 'set' hardware instruction 'ABS'. This instruction locks the shared memory for a longer time than other instructions and has led to several system 'memory time-out' faults. A combined resource sharing algorithm developed in the Department of Computer Studies, removes the need to utilize an indivisible instruction, albeit with an increased execution time.

4) It is not necessary to have a path descriptor block for each parallel path. If a single block is utilized for a 'DOPAR', the block management operations would be removed and so reduce the overhead of executing a parallel path.

Accordingly, a new version was written in September 1982, to incorporate these new facilities. None of the changes should effect the behaviour of a parallel program, but will obviously affect the actual timing obtained.

The new path descriptor block holds the following information:

<table>
<thead>
<tr>
<th>See (A.C-II:3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSolute value</td>
</tr>
</tbody>
</table>
- Start of path address
- Pointer to next block in the list (-1 if end of list)
- Address of parent path descriptor block (as in XPFCL)

[ ] Type of block ('DOPAR', 'DOALL', 'JOIN', 'Empty 'DOPAR')
- Count of active paths on this block
- Index value of parent block (if 'DOPAR')
- The identifier of the processor that executed the
  'FORK'

'parameters'

- Index variable address
- Current index value
- Index end value
- Index increment

Flags for
  each
  for 'DOPAR' or processor
  for '$DOALL'

Each block occupies 20 bytes*, instead of 12 bytes for 'XPFCL',

and there are 60 blocks as opposed to 78 in 'XPFCL'. The two lists of
blocks are referenced by list headers, a value of -1 indicates an 'empty'
list.

When a 'CALL FORK' is executed, a 'free' block is taken from the head
of the 'free' list (sometimes the current block can be re-utilized
instead) and initialized according to the parameters, the processor
number and the current index value of the parent. The processor then
enters the normal path scheduling routine, scanning the list of active
blocks for one with outstanding work. This will either be a '$DOALL'
with this processor yet to run it, a 'JOIN' block to be executed, or a
'DOPAR' with outstanding paths.

If it is a 'DOPAR', the next index value is calculated (i.e. current

*The type of block and the count of active paths occupy a half word each,
whilst the rest one word each, altogether a total of ten words (i.e. 20
bytes).

+ increment) and set into local memory; if this is the last path, the 'DOPAR' is set as an 'empty' DOPAR block and the set of nested 'DOPAR' index values is set as in the case for 'XPFCL'.

The 'JOIN' operation is even more complicated. When a path terminates, the count of active paths on that block is decremented; if it is zero and the block is an 'empty' DOPAR, then the block is finished and it is modified into a 'JOIN' block, which resembles a $DOALL that must be executed by only one processor, the one that executed the original 'FORK'.

A similar fate befalls a $DOALL when all the processors have completed their paths. Only if the block is a 'JOIN' block type is it released to the 'free' list. It is these 'JOIN' blocks that are simply re-utilized when a 'CALL FORK' occurs, as the only information required for such a block is the address of the parent and the executing processor.

These, rather obscure, rules ensure that the number of block link and delink operations are kept to a minimum. Indeed, for a program with no nested parallel 'FORK' constructs, such as in the example-A.C-II:2), only a single scheduling block is utilized throughout the program.

```
$USEPAR
  ...
  $DOALL 10
    ...
    10 $PAREND
    ...
    $DOPAR 20 I=1,50
    ...
    20 $PAREND
    ...
    $DOALL 30
      ...
      30 $PAREND
      ...
```

(A.C-II:2)

The utilization of a single block, for each 'DOPAR', has enabled the
maximum number of paths to be limited by the word length\(^*\) and not the path scheduling workspace size. Performance measurements show that, although the path block set-up time is about the same, as for 'XPFCL', the new version has a significantly lower parallel path overhead. These overheads, for a 'DOPAR' to create \(N\) paths for the two versions, are as follows:

\[
\begin{align*}
\text{'XPFCL':} & \quad 1.2 + 1.115N + 0.024N^2 \text{ ms} \\
\text{'XPFCLD':} & \quad 1.3 + 0.87N \text{ ms}
\end{align*}
\]

\(A.C-II:3\)

In conclusion, some additional up-to-date information about the NEPTUNE system is that, as mentioned earlier, it utilizes (for the 'XPFCLD' command only), a new 'coordination' algorithm, the so-called 'hybrid' algorithm (designated WARR), thus substituting the indivisible 'test' and 'set' hardware instruction 'ABS' (of the 'XPFCL' command). The WARR combined sharing algorithm, behaves as a bartering algorithm for periods of low resource utilization and as resource master algorithm at high levels of resource utilization. The concept behind this algorithm was that, since the performance characteristics of the two classes of algorithms were (in some sense) complementary, then if an algorithm which would dynamically alter its nature could be developed, the best performance of each class would be achieved, without the corresponding poor performance associated with each class separately.

Finally, a new 'XPFCLX' command has been implemented, similarly, to the 'XPFCLD' one, but with the significant advantage of extending the 'nesting' levels of the 'XPFCL/XPFCLD' commands, from 'four' for any 'FORK', to a 'hundred' for $FORK$ and unlimited 'nesting' for $DOPAR/$DOALL.

\(^*\)It is 15 bits + 1 'sign' bit, i.e.: \(2^{15} - 1 = 33767\) parallel paths per DOPAR block.
**APPENDIX - II.B.3.1**

**APPROXIMATE NEW TIMINGS**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Processor</th>
<th>( P_0 )</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Speeds (^\dagger)</td>
<td></td>
<td>1.000</td>
<td>1.014</td>
<td>1.006</td>
<td>1.019</td>
</tr>
<tr>
<td>Floating point(^*)</td>
<td></td>
<td>(-700)</td>
<td>(-700)</td>
<td>(-700)</td>
<td>(-700)</td>
</tr>
<tr>
<td>Integer(^*)</td>
<td></td>
<td>(-20)</td>
<td>(-20)</td>
<td>(-20)</td>
<td>(-20)</td>
</tr>
<tr>
<td>Local Memory Access(^*)</td>
<td></td>
<td>0.98</td>
<td>0.95</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td>Shared Memory Access(^*)</td>
<td></td>
<td>1.73</td>
<td>1.70</td>
<td>1.68</td>
<td>1.68</td>
</tr>
<tr>
<td>Mutual Exclusion Mechanism(^*)(for 'XPFCL')</td>
<td></td>
<td>(-600)</td>
<td>(-600)</td>
<td>(-600)</td>
<td>(-600)</td>
</tr>
<tr>
<td>Mutual Exclusion Mechanism(^*)(for 'XPFCLD')</td>
<td></td>
<td>(-800)</td>
<td>(-800)</td>
<td>(-800)</td>
<td>(-800)</td>
</tr>
<tr>
<td>Parallel Path Mechanism(^*)(for 'XPFCL')</td>
<td></td>
<td>(-1200)</td>
<td>(-1200)</td>
<td>(-1200)</td>
<td>(-1200)</td>
</tr>
<tr>
<td>Parallel Path Mechanism(^*)(for 'XPFCLD')</td>
<td></td>
<td>(-900)</td>
<td>(-900)</td>
<td>(-900)</td>
<td>(-900)</td>
</tr>
<tr>
<td>Overhead of executing a $DOPAR(^*)(for 'XPFCL')</td>
<td></td>
<td>(-1200)</td>
<td>(-1200)</td>
<td>(-1200)</td>
<td>(-1200)</td>
</tr>
<tr>
<td>Overhead of executing a $DOPAR(^*)(for 'XPFCLD')</td>
<td></td>
<td>(-1300)</td>
<td>(-1300)</td>
<td>(-1300)</td>
<td>(-1300)</td>
</tr>
</tbody>
</table>

\(^\dagger\) excluding access to shared memory

\(^*\) times in microseconds

Note that processors \( P_1 \), \( P_2 \), and \( P_3 \) have parity memory, while processor \( P_0 \) has two different blocks of error correcting memory. Finally, it is apparent from the above Table that the differences between 'XPFCL' and 'XPFCLD' will affect the parallel programs execution times and, consequently, the timings achieved respectively cannot be directly compared. However, all parallel programs that run with 'XPFCL' will function under 'XPFCLD' and it is intended to phase out the utilization of the former.
A SELECTION OF
OPTIMIZED PARALLEL COMPUTER PROGRAMS
FOR THE 'GE' METHODS
C THIS PROGRAM IMPLEMENTS THE STANDARD EXPPLICIT METHOD TO SOLVE
BURGERS' 1-D NON-LINEAR PARABOLIC PDE.
THE NATURE AND COMPLEXITY OF THE METHOD INEVITABLY INTRODUCE
SYNCHRONIZATION ACTIVITIES, WITHOUT ANY PERFORMANCE LOSS THOUGH.
DUE TO THE FAST SYNCHRONIZATION TOOL OF THE 'NEPTUNE' Prototype
SYSTEM.

C SET THE REQUIRED ARRAYS.
C **************
C U : IT HOLDS TEMPORARILY AT EACH TIME-LEVEL, THE PDE'S
C APPROXIMATE VALUES AT THE INTERNAL POINTS COMPUTED
C USING THE STANDARD EXPPLICIT FINITE-DIFFERENCE FORMULA.
C W : IT HOLDS THE PDE'S EXACT THEORETICAL VALUES AT ALL THE
C BOUNDARY AND INTERNAL POINTS AT THE MAXIMUM TIME-LEVEL,
C COMPUTED USING THE CHOSEN EXACT SOLUTION FORMULA.
C ERROR : IT HOLDS THE DIFFERENCES BETWEEN THE PDE'S EXACT AND
C APPROXIMATE VALUES AT THE INTERNAL POINTS AT THE MAXIMUM
C TIME-LEVEL.
C Z : IT HOLDS AT EACH TIME-LEVEL THE PDE'S EXACT AND APPROXIMATE
C VALUES AT ALL THE BOUNDARY AND INTERNAL POINTS COPIED FROM THE
C ARRAY 'U' AND THE WORK-ARRAY 'F' RESPECTIVELY.
C ITIME : IT HOLDS THE TIMING INFORMATION.
C F : IT HOLDS THE PDE'S EXACT VALUES AT THE POINTS ON BOTH
C BOUNDARIES FOR ALL TIME-LEVELS COMPUTED USING THE CHOSEN
C EXACT SOLUTION FORMULA.
C
C REAL 10, IT, IM, IK, LTD
C SET THE SHARED DATA.
C *SHARED U,NPROC,ITIME,EPSLN, R
C C** SHARED DX,DT,IL,Z,IL,F
C SET CRITICAL SECTIONS.
C C** REGION NOPROC
C INITIALIZE PARALLELISM.
C USEPAR
C NPROC=0
C GENERATE PARALLEL PATHS TO SET DYNAMICALLY THE NUMBER OF
C PROCESSORS TO BE UTILIZED EACH TIME IN THE PROGRAM.
C SDALL 1000
C ENTER CRITICAL SECTION.

C READ THE NUMBER OF INTERNAL POINTS AND TIME-STEPS.
READ(5,99990) NPOINT, ISTEP
EPSLN=0.003
C SET THE VALUE FOR THE GRID RATIO: R = DT/(DX**2).
R=0.5
C COMPUTE THE GRID'S X-SPACING UNIT, FOR THE RANGE [0,1].
DX=1./(NPOINT+1)
C COMPUTE THE GRID'S T-SPACING UNIT, FOR THE RANGE [0, +oo).
DT=R/(DX**2)
C WRITE(6,99991) NPOINT, ISTEP, EPSLN, R, DX, DT
C IN=NPOINT/NPROC
C IL=ISTEP/NPROC
C GENERATE PARALLEL PATHS TO SET LOCAL COPIES TO DIMINISH THE
C OVERHEADS DUE TO SHARED MEMORY ACCESS.
C SDALL 1500
C EPSLN=EPSLN
C RL=R
C DLX=DX
C DTL=DT
C IML=IM
C ILL=IL
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C 1500 $PAREND
C
C <<<<<<<<<<<<<<<<<<< COMPUTATION OF THE EXACT THEORETICAL VALUES
C C PHASE 2 << AT ALL THE BOUNDARY AND INTERNAL POINTS AT
C C THE MAXIMUM TIME-LEVEL, USING THE CHOSEN EXACT
C C SOLUTION FORMULA.
C
C DO 2000 I=1, IN
X=(I-1)*DXL
SDT=ISTEP*DTL
X1=EXP(-A)
X2=EXP(-B)
X3=EXP(-C)
IT=0.1*X1+0.5*X2+X3
2000 CONTINUE
109 \text{IO} \times X1 + X2 + X3
110 W1 = IT/10
111 \text{COPY THE COMPUTED EXACT VALUES AT BOTH BOUNDARY POINTS AT}
112 \text{THE MAXIMUM TIME-LEVEL INTO THE APPROPRIATE ARRAY.}
113 ISP1 = ISP1 + 1
114 \text{F(ISP1,1)} = W1
115 \text{F(ISP1,2)} = W(IN)
116 \text{WRITE(6, 99992) ISP1, W(IN, I = 1, IN)}
117 \text{START TIME THE COMPUTATIONAL PROCEDURE.}
118 C
119 C \text{COMPUTE EXACT SOLUTION FORMULA.}
120 C
121 C P H A S E 3 \text{: COMPUTATION OF THE EXACT VALUES AT ALL THE}
122 C \text{INTERNAL POINTS, USING THE CHOSEN INITIAL VALUES.}
123 C P H A S E 4 \text{: AT THE INTERNAL POINTS, USING THE CHOSEN}
124 C \text{INITIAL VALUES.}
125 \text{IT} = 0.0
126 \text{ID} = 0.0
127 C \text{GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED}
128 C \text{SUBSETS OF INTERNAL POINTS (*ZERO TIME-LEVEL*).}
129 \text{DO} 5000 J = 1, NPROC
130 \text{C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.}
131 C \text{EXACT SOLUTION FORMULA.}
132 C
133 C \text{START TIME THE COMPUTATIONAL PROCEDURE.}
134 C
135 \text{COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED EXACT VALUES.}
136 C \text{AT BOTH BOUNDARY POINTS (*ZERO TIME-LEVEL*).}
137 \text{IT} = 0.0
138 \text{ID} = 0.0
139 C \text{GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED}
140 C \text{SUBSETS OF BOUNDARY POINTS.}
141 \text{DO} 6000 J = 1, NPROC
142 \text{C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.}
143 C \text{COMPUTATION OF THE APPROPRIATE VALUES AT}
144 C \text{THE INTERNAL POINTS, AT EVERY TIME-LEVEL.}
145 C \text{FOR ALL TIME-STEP, USING THE STANDARD}
146 C \text{EXPLICIT FINITE-Difference FORMULA.}
147 C
148 \text{COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED EXACT VALUES.}
149 C \text{AT BOTH BOUNDARY POINTS (*ZERO TIME-LEVEL*).}
150 \text{IT} = 0.0
151 \text{ID} = 0.0
152 C \text{GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED}
153 C \text{SUBSETS OF INTERNAL POINTS.}
154 \text{DO} 7000 K = 1, ISTEP
155 \text{C \text{END TIME-LEVEL. GENERATE 'NPROC' PARALLEL PATHS AND ASSING TO THEM THE}
156 C \text{CREATED SUBSETS OF INTERNAL POINTS.}
157 \text{DO} 8000 J = 1, NPROC
158 \text{C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.}
159 C \text{EXACT SOLUTION FORMULA.}
160 C
161 \text{COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED EXACT VALUES.}
162 C
C VALUES AT BOTH BOUNDARY POINTS. FOR THE PRESENT TIME-LEVEL.

KP1 = KP1 + 1
Z(1) = F(KP1, 1)
Z(IN) = F(KP1, 2)

C COPY THE COMPUTED APPROXIMATE VALUES AT THE INTERNAL POINTS.

DO 6500 I = 2, 1R
Z(1) = U(I)
6500 CONTINUE

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

CALL TIMOUT(I)

C TERMINATE PROGRAM.

SEND

C P H A S E 6 COMPUTATIONAL PROCEDURE AND COMPUTE THE MAXIMUM ABSOLUTE ERROR AND THE MAXIMUM PERCENTAGE ERROR.

WRITE(6, 99993) ITIME
WRITE(6, 99994) ISTEP, Z(1), I = 2, 1R

C COMPUTE AND PRINT OUT THE DIFFERENCES BETWEEN THE EXACT AND APPROXIMATE VALUES AT THE INTERNAL POINTS AT THE MAXIMUM TIME-LEVEL.

DO 8000 K = 2, 1R
ERROR(K) = W(K) - Z(K)
8000 CONTINUE

WRITE(6, 99995) (ERROR(I), I = 2, 1R)

C COMPUTE THE MAXIMUM "ABSOLUTE ERROR" - "A.E."

DO 9000 I = 2, 1R
IF (ABS(ERROR(I)), 0.0, T1) GO TO 8500
GO TO 9000
8500 T1 = ABS(ERROR(I))
9000 CONTINUE

C COMPUTE THE MAXIMUM "PERCENTAGE ERROR" - "P.E."

PERR = 100 * T1 / T2
C PRINT OUT THE MAXIMUM "A.E." THE MAXIMUM "P.E."
WRITE(6, 99996) T1, PERR

C SET THE FORMATS
C ***************

99990 FORMAT(16, 1X, 16)
C SET CRITICAL SECTIONS.
C SET SYSTEM.
C SYNCHRONIZATION ACTIVITIES, WITHOUT ANY PERFORMANCE LOSS THOUGH,
C UNGROUPED ENDS 'G.E.U.' METHOD TO SOLVE
C NON-LINEAR PARABOLIC P.D.E.
C THIS PROGRAM IMPLEMENTS THE "GROUP EXPLICIT WITH BOTH
C UNGROUPED ENDS" 'G.E.U.' METHOD TO SOLVE BURGERS' 1-D
C THE NATURE AND COMPLEXITY OF THE METHOD INEVITABLY INTRODUCE
C TO THE FAST SYNCHRONIZATION TOOL OF THE 'NEPTUNE' PROTOTYPE
C DUE TO THE MAXIMUM TIME-LEVEL, COMPUTE USING SAUL'YEV'S 'L->R','L<-R' ASYMMETRIC FORMULAE,
C AT THE REMAINING GROUPS OF '2' INTERNAL POINTS, THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE ARE
C UTILIZED.
C U : IT HOLDS TEMPORARILY AT EACH TIME-LEVEL, THE P.D.E.'S
C APPROXIMATE VALUES AT ALL THE INTERNAL POINTS, THE VALUES
C AT THE 'LEFT' AND 'RIGHT' UNGROUPED NEAR BOUNDARY POINTS ARE
C COMPUTED USING SAUL'YEV'S 'L-OR','L-R' ASYMMETRIC FORMULAE, RESPECTIVELY, WHILE AT THE REMAINING GROUPS OF '2' INTERNAL
C POINTS THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE ARE
C USED.
C W : IT HOLDS THE P.D.E.'S EXACT THEORETICAL VALUES AT THE
C INTERNAL POINTS, AT THE MAXIMUM TIME-LEVEL, COMPUTED
C USING THE CHOSEN EXACT SOLUTION FORMULA.
C ERROR : IT HOLDS THE DIFFERENCES BETWEEN THE P.D.E.'S EXACT AND
C APPROXIMATE VALUES AT THE INTERNAL POINTS, AT THE MAXIMUM TIME-LEVEL.
C Z : IT HOLDS AT EACH TIME-LEVEL, THE P.D.E.'S EXACT AND APPROXIMATE
C VALUES AT ALL THE BOUNDARY AND INTERNAL POINTS, COPIED FROM THE
C ARRAY 'F' AND THE WORK-ARRAY 'U', RESPECTIVELY.
C ITIME : IT HOLDS THE TIMING INFORMATION.
C F : IT HOLDS THE P.D.E.'S EXACT VALUES AT THE POINTS ON BOTH
C BOUNDARIES, FOR ALL TIME-LEVELS, COMPUTED USING THE CHOSEN
C EXACT SOLUTION FORMULA.
C REAL IQ, IT, KX, KL, LTD
C SET THE SHARED DATA.
C *SHARE U,NPOINT,NPROC,ITIME, EPSLN,R.
C *REGION NOPROC
C C GENERATE PARALLEL PATHS TO SET DYNAMICALLY THE NUMBER OF
C PROCESSORS TO BE UTILIZED EACH TIME IN THE PROGRAM,
C $DOALL 1500
C C ENTER CRITICAL SECTION.
C *ENTER NOPROC
C NPROC=NPROC+1
C C RELEASE CRITICAL SECTION.
C *EXIT NOPROC
C C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C $DOALL 1000 SPAREND
C C READ THE NUMBER OF INTERNAL POINTS AND TIME-STEPS.
C READ(5,99990) NPOINT, ISTEP
C EPSLN=0.003
C C SET THE VALUE FOR THE GRID RATIO : R = (1/2)/(DX)**2.
C R=1.0
C C COMPUTE THE GRID'S X-SPACING UNIT, FOR THE RANGE [0,1).
C DX1=(1/NPOINT+1)
C C COMPUTE THE GRID'S T-SPACING UNIT, FOR THE RANGE [0, +oo).
C DT=R*(DX**2)
C WRITE(6,99991) NPOINT, ISTEP, EPSLN, R, DX, DT
C DX2=DX/2
C IN=NPOINT+2
C IR=NPOINT+1
C C IN ACCORDANCE WITH THE UTILIZED NUMBER OF PROCESSORS EACH TIME.
C C IN ACCORDANCE WITH THE UTILIZED NUMBER OF PROCESSORS EACH TIME.
C C IN ACCORDANCE WITH THE UTILIZED NUMBER OF PROCESSORS EACH TIME.
C C IN ACCORDANCE WITH THE UTILIZED NUMBER OF PROCESSORS EACH TIME.
C C IN ACCORDANCE WITH THE UTILIZED NUMBER OF PROCESSORS EACH TIME.
C C C TERMINATE PARALLEL PATHS TO SET LOCAL COPIES TO DIMINISH THE
C C OVERHEADS DUE TO SHARED MEMORY ACCESSES.
C $DOALL 1500
C EPSLNL=EPSLN
C RL=R
C DXL=DX
C DTL=DT
C DX2L=DX2
C IRL=IR
C INPTL=NPOINT
C IL=IL
C C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C $DOALL 1500 SPAREND
C C C COMPUTATION OF THE EXACT THEORETICAL VALUES
C C AT THE INTERNAL POINTS, AT THE MAXIMUM TIME-LEVEL, USING THE CHOSEN EXACT SOLUTION FORMULA.
C DO 2000 I=2, IRL
X=(X(I-1))*DXL
C SET CRITICAL SECTIONS.
C $REGION NOPROC
C C INITIALIZE PARALLELISM.
C $USEPAR
ISO

152

153

154

155

156

157

158

159

160

161

162

151

150

149

148

147

146

145

144

143

142

141

140

139

138

137

136

135

134

133

132

131

130

129

128

127

126

125

124

123

122

121

120

119

118

117

116

115

114

113

112

111

110

109

108

107

106

105

104

103

102

101

100

99

98

97

96

95

94

93

92

91

90

89

88

87

86

85

84

83

82

81

80

79

78

77

76

75

74

73

72

71

70

69

68

67

66

65

64

63

62

61

60

59

58

57

56

55

54

53

52

51

50

49

48

47

46

45

44

43

42

41

40

39

38

37

36

35

34

33

32

31

30

29

28

27

26

25

24

23

22

21

20

19

18

17

16

15

14

13

12

11

10

9

8

7

6

5

4

3

2

1

200 CONTINUE

WRITE(6,99992) ISTEP, (W(I), I=2, IRL)

C START

TIMING THE

COMPUTATIONAL PROCEDURE.

C

CALL TIMEST

2500 SPAREND

C COMPUTATION OF

THE EXACT VALUES AT ALL THE

POINTS ON BOTH BOUNDARIES, FOR

ALL TIME-LEVELS, STARTING FROM

TIME-LEVEL ONE, USING THE

EXACT SOLUTION FORMULA.

C IT=0.0

IQ=0.0

C GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE

CREATED SUBSETS OF INTERNAL POINTS (ZERO TIME-LEVEL).

DO 4500 J=1, NPROC

C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.

IS= (J-1)* IHL+1

IE= IS + IHL -1

DO 4500 I= IS, IE, 2

IM1 = I-1

IM2 = I-2

IP1 = I+1

IP2 = I+2

IP3 = I+3

DO 10500 K=1, ISTEP

C EVERY TIME-STEP, COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED

EXACT VALUES. AT BOTH BOUNDARY POINTS.

C

Z(1) = F(K,1)

Z(INL+1) = F(K,2)

DO EVERY TIME-STEP, GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN

C TO THEM THE CREATED SUBSETS OF INTERNAL POINTS.

DO 205 J=1, NPROC

C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.

IS= (J-1)*IHL+1

IE= IS + IHL -1

DO 9000 I= IS, IE, 2

IM1 = I-1

IM2 = I-2

IP1 = I+1

IP2 = I+2

IP3 = I+3

150

DMIM= IM1-0.5

LDL= K-DMIM

A= 0.05*(DMIM-4.95*DLT)/EPSLN

B= 0.25*(DMIM+0.75*DLT)/EPSLN

C= 0.5*(IM1-0.375)/EPSLN

X1= EXP(-A)

X2= EXP(-B)

X3= EXP(-C)

IT= 0.1*X1+0.5*X2+X3

IQ= X1+X2+X3

Z(U, I)= IT / IQ

200 CONTINUE

100

X= I*DXL

XM= X-0.5

A= 0.05*XM/EPSLN

B= 0.25*XM/EPSLN

C= 0.5*(X-0.375)/EPSLN

X1= EXP(-A)

X2= EXP(-B)

X3= EXP(-C)

IT= 0.1*X1+0.5*X2+X3

IQ= X1+X2+X3

Z(U, I)= IT / IQ

163 C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.

164 4000 SPAREND

165 C

166 C COMPATIIIATION OF THE EXACT INITIAL VALUES

167 C P H A S E 4 C C AT THE INTERNAL POINTS, USING THE CHOSEN

168 C EXACT SOLUTION FORMULA.

169 C

IT=0.0

IQ=0.0

170 C GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED

171 C SUBSETS OF INTERNAL POINTS (ZERO TIME-LEVEL).

172 C

173 C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.

174 C

IS= (J-1)*IHL+1

IE= IS + IHL -1

DO 4500 J=1, NPROC

C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.

IS= (J-1)*IHL+1

IE= IS + IHL -1

DO 4500 J=1, NPROC

C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.

IS= (J-1)*IHL+1

IE= IS + IHL -1

DO 4500 J=1, NPROC

C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.
C TERMINATE TIMING
C
10:500 CONTINUE

C TERMINATE PARALLEL

10000 C

C CHECK FOR THE 'LEFT' UNGROUPED NEAR BOUNDARY POINT.
IF (I.EQ.2) GO TO 8200

EP1 = B2* ( (1-BIR)*Z (1) )
EP2 = (1-AIR-A2R-AIR+2R)*Z (2)
EP3 = BIR+B2R*Z (1) + (1+AIR)*Z (1)
UI (1) = (EP1*EP2)/D

GO TO 8000

C CHECK FOR THE 'RIGHT' UNGROUPED NEAR BOUNDARY POINT.
IF (I.EQ.1) GO TO 8500

OP1 = (1+B2R+B1R+2B2R)*Z (1)
OP2 = (1-AIR)*AIR+Z (2)
OP3 = (1+B2R)*BIR+Z (1) + AIR+2R+Z (3)
UI (1) = (OP1*OP2)/D

GO TO 9000

C COMPUTE THE VALUE AT THE 'LEFT' UNGROUPED NEAR BOUNDARY POINT.
A = EPSLNL-DX2L*Z (2)
B = EPSLNL-DX2L*Z (1)
RA = RL + RA
RB = RL + RB
U1 (2) = (RB*Z (1) + (RA)*Z (2) + (1-RA)*Z (1))/ (1+RB)

GO TO 8000

C COMPUTE THE VALUE AT THE 'RIGHT' UNGROUPED NEAR BOUNDARY POINT.
A = EPSLNL-DX2L*Z (2)
B = EPSLNL-DX2L*Z (1)
RA = RL + RA
RB = RL + RB
UI (3) = (RA*Z (1) + RB*Z (2) + (1-RB)*Z (1))/ (1+RA)

GO TO 9000

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.

9000 CONTINUE

C COPY THE COMPUTED APPROXIMATE VALUES AT THE INTERNAL POINTS.
C FROM THE WORK-ARRAY 'U' INTO THE ARRAY 'Z'.
DO 10000 I = 2, IRL
Z(I) = UI

10000 CONTINUE

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
C DIRECTORy,FILENAME: MB55.GECONI

C THIS PROGRAM IMPLEMENTS THE "GROUP EXPLICIT COMPLETE"-"G.E.C."
C METHOD TO SOLVE BURGERS' 1-D NON-LINEAR PARABOLIC P.D.E.
C THE NATURE AND COMPLEXITY OF THE METHOD INEVITABLY INTRODUCE
C SYNCHRONIZATION ACTIVITIES.WITHOUT ANY PERFORMANCE LOSS THOUGH.
C DUE TO THE FAST SYNCHRONIZATION TOOL OF THE 'NEPTUNE' PROTOTYPE
C SYSTEM.
C
C SET THE REQUIRED ARRAYS.
C
C DIMENSION U(1940),W(1940),ERROR(1940),Z(1940),ITIME(100)
C DIMENSION F(500,2)
C
C SPECIFICATION OF THE ARRAYS
C
C U : IT HOLDS TEMPORARILY,AT EACH TIME-LEVEL,THE P.D.E.'S
C APPROXIMATE VALUES AT ALL THE GROUPS OF '2' INTERNAL
C POINTS,COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE
C FORMULAE.
C
C W : IT HOLDS THE P.D.E.'S EXACT THEORETICAL VALUES AT ALL
C THE BOUNDARY AND INTERNAL POINTS,AT THE MAXIMUM TIME-LEVEL,
C COMPUTED USING THE CHOSEN EXACT SOLUTION FORMULA.
C
C ERROR : IT HOLDS THE DIFFERENCES BETWEEN THE P.D.E.'S EXACT AND
C APPROXIMATE VALUES AT THE INTERNAL POINTS,AT THE MAXIMUM
C TIME-LEVEL.
C
C APPROXIMATE VALUES AT ALL THE BOUNDARY AND INTERNAL POINTS,COMPUTED FROM THE
C ARRAY 'F' AND THE WORK-ARRAY 'U',RESPECTIVELY.
C
C ITIME : IT HOLDS THE TIMING INFORMATION.
C
C F : IT HOLDS THE P.D.E.'S EXACT VALUES AT THE POINTS ON BOTH
C BOUNDARIES,FOR ALL TIME-LEVELS,COMPUTED USING THE CHOSEN
C EXACT SOLUTION FORMULA.
C
C REAL I0,IT,KK,KL,LDT
C
C $SHARED U,NPROC,ITIME,EPSSLN,R.
C $REGION DX,DT,HH,IL,F,DX2
C
C SET THE SHARED DATA.
C
C THE MAXIMUM TIME-LEVEL,USING THE CHOSEN EXACT
C SOLUTION FORMULA.
C
C DO 2000 I=1,IN
C X=(I-1)*DX
C SDT=ISTEP*DT
C XM=0.5
C A=0.05*(XM+4.95*SDT)/EPSSLN
C B=0.25*(XM+0.75*SDT)/EPSSLN
C C=0.5*(X-0.375)/EPSSLN
C X1=EXP(-A)
C X2=EXP(-B)
C
C ENTER NPROC
C NPROC=NPROC+1
C
C RELEASE CRITICAL SECTION.
C NEXT NPROC
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C DOALL 1000
C
C COMPUTE THE GRID'S X-SPACING UNIT,FOR THE RANGE [0.1],
C DX=1./(NPOINT+1),
C DT=R*(DX**2)
C WRITE(*,99991) NPOINT,ISTEP,EPSSLN,R,DX,DT
C
C ENTER CRITICAL SECTION.
C
X3 = EXP(-C)
IT = 0.1*X1 + 0.5*X2 + X3
ID = X1 + X2 + X3
W(I) = 1/T/10

2000 CONTINUE
C COPY THE COMPUTED EXACT VALUES AT BOTH BOUNDARY POINTS, AT
THE MAXIMUM TIME-LEVEL, INTO THE APPROPRIATE ARRAY.
ISTP = ISTEP + 1
F (ISTP, 1) = W(I)
WRITE (6, 99992) ISTEP, W(I), I = 1, NPROC
C COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED EXACT VALUES,
AT BOTH BOUNDARY POINTS (*ZERO TIME-LEVEL*).
F<ISTP ,1 = Z(I)
F<ISTP,2 >= FCIN>
DO 7000 K = 1, ISTEP
C EVERY TIME-STEP, GENERATE *NPROC* PARALLEL PATHS AND ASSIGN
TO THEM THE CREATED SUBSETS OF INTERNAL POINTS.
$DOPAR 6000 J = 1, NPROC
C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.
IS = (J-1)*IHL + 1
IE = IS + IHL - 1
DO 5500 L = IS, IE, 2
LP1 = L + 1
LP2 = L + 2
IM1 = L - 1
IM2 = L - 2
AL = EPSLNL * KL
A2 = EPSLNL * KK
Z<LP1> = Z1
Z<LP2> = Z2
DO 3500 K = 1, ISTEP
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
4000 $PAREND
C COMPARE THE PRE-COMPUTED EXACT VALUES, AT BOTH BOUNDARY POINTS.
C AT BOTH BOUNDARY POINTS (*ZERO TIME-LEVEL*).
Z<IT> = F<IT>
Z<IT> = FCIN>
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
5000 $PAREND
C COMPARE THE COMPUTED EXACT INITIAL VALUES, AT THE INTERNAL POINTS.
C USE THE CHosen EXACT SOLUTION FORMULA.
C COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED EXACT VALUES,
AT BOTH BOUNDARY POINTS (*ZERO TIME-LEVEL*).
F<ISTP, 1 = Z(I)
F<ISTP,2 >= FCIN>
DO 7000 K = 1, ISTEP
C EVERY TIME-STEP, GENERATE *NPROC* PARALLEL PATHS AND ASSIGN
TO THEM THE CREATED SUBSETS OF INTERNAL POINTS.
$DOPAR 6000 J = 1, NPROC
C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.
IS = (J-1)*IHL + 1
IE = IS + IHL - 1
DO 5500 L = IS, IE, 2
LP1 = L + 1
LP2 = L + 2
IM1 = L - 1
IM2 = L - 2
AL = EPSLNL * KL
A2 = EPSLNL * KK
Z<IT> = F<IT>
Z<IT> = FCIN>
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
5000 $PAREND
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
B1=EPSLN+KL
B2=EPSLN+KK
AIR=A1+RL
AR=A2+RL
BIR=B1+IR
B2R=B2+IR
D1=A1R+2R
D2=A2R+2R
OP1=(1+B2R-B1R-B2R)*Z(I)
OP2=(1-A2R)*AIR+Z(I)
OP3=(1+B2R)*AIR+AIR+2R*I(IP2)
U(I)=(OP1+OP2*OP3)/D
EP1=B2R*(1-BIR)*Z(I)
EP2=(1+AIR-A2R-AIR+2R)*Z(I)
EP3=BIR+B2R*I(IP1)+(1+AIR+2R)*Z(IP2)
5500 CONTINUE
6000 SPAREND
C COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED EXACT VALUES AT BOTH BOUNDARY POINTS, FOR THE PRESENT TIME-LEVEL.
KP1=K1
Z(I)=F(KP1,1)
Z(IN)=F(KP1,2)
C COPY THE COMPUTED APPROXIMATE VALUES AT THE INTERNAL POINTS.
C FROM THE WORK-ARRAY 'U' INTO THE ARRAY 'Z'.
DO 6500 I=2,IR
   Z(I)=U(I)
6500 CONTINUE
7000 CONTINUE
C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
C I OHSE 6 OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL PROCEDURE AND COMPUTE THE MAXIMUM ABSOLUTE ERROR AND THE MAXIMUM PERCENTAGE ERROR.
WRITE(6,99993) ITIME
WRITE(6,99994) ISTEP,(Z(I),I=2,IR)
C COMPUTE AND OUTPUT THE DIFFERENCES BETWEEN THE EXACT AND APPROXIMATE VALUES AT THE INTERNAL POINTS, AT THE MAXIMUM TIME-LEVEL.
DO 8000 K=2,IR
   ERROR(K)=W(K)-Z(K)
8000 CONTINUE
4500 SPAREND
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C SET THE FORMATS
C COMPUTE THE MAXIMUM "ABSOLUTE ERROR"-'A.E.'
DO 9000 K=2,IR
   IF(ABS(ERROR(I)).GT.T1) GO TO 8500
   T1=ABS(ERROR(I))
   T2=ABS(W(I))
8500 CONTINUE
C COMPUTE THE MAXIMUM "PERCENTAGE ERROR"-'P.E.'
PERR=100.0*T1/T2
C PRINT OUT THE MAXIMUM 'A.E.', THE MAXIMUM 'P.E.'
WRITE(6,99996) T1,PERR
C COMPUTE AND OUTPUT THE MAXIMUM ABSOLUTE ERROR AND THE MAXIMUM PERCENTAGE ERROR.
WRITE(6,99995) ERROR(I),I=2,IR
C SET THE FORMATS
FORMAT(1X,16)
FORMAT(1X,'NPOINT =',I4,4X,'ISTEP =',J4,4X,'EPSLN =',E12.4,4X,'OX =',E14.6,4X,'OT =',E14.6)
FORMAT(1X,'THE EXACT THEORETICAL VALUES AT TIME-LEVEL :',E18.9,2X)
FORMAT(1X,'THE APPROXIMATE VALUES AT TIME-LEVEL :',E18.9,2X)
FORMAT(1X,'THE ERRORS ARE ',E18.9,2X)
FORMAT(1X,'THE MAXIMUM A.E. IS ',E18.9,2X)
FORMAT(1X,'THE MAXIMUM P.E. IS ',E18.9,2X)
FORMAT(1X,'TIME-LEVEL :',E18.9)
FORMAT(1X,'THE MAXIMUM A.E. IS ',E18.9,2X)
FORMAT(1X,'THE MAXIMUM P.E. IS ',E18.9,2X)
STOP
END
PROGRAM.
C THIS PROGRAM IMPLEMENTS THE "(SINGLE) ALTERNATING GROUP EXPLICIT"-
C 'O.G.E.C.' SCHEMES AT EVERY ALTERNATE 'TIME-LEVEL' TO SOLVE BURGERS-
C 1-D NON-LINEAR PARABOLIC P.D.E.
C THE NATURE AND COMPLEXITY OF THE METHOD INEVITABLY INTRODUCE
C SYNCHRONIZATION ACTIVITIES WITHOUT ANY PERFORMANCE LOSS THOUGH.
C DUE TO THE FAST SYNCHRONIZATION TOOL OF THE 'NEPTUNE' PROTOTYPE
C SYSTEM.
C
C SET THE REQUIRED ARRAYS.
C
DIMENSION U(1940),W(1940),ERROR(1940),Z(1940),ITIME(100)

C SPECIFICATION OF THE ARRAYS
C
U : IT HOLDS TEMPORARILY AT EACH 'TIME-LEVEL' THE P.D.E.'S
C APPROXIMATE VALUES AT ALL THE INTERNAL POINTS. THE VALUES
C AT THE 'LEFT' AND 'RIGHT' UNGROUPED NEAR BOUNDARY POINTS,
C AT EVERY 'TIME-LEVEL' THAT THE 'O.G.E.' SCHEME IS APPLIED,
C ARE COMPUTED USING SAUL-YEV'S 'L->R', 'L<R' ASYMMETRIC
C FORMULAE, RESPECTIVELY. THE VALUES AT ALL OR THE REMAINING
C -FOR THE 'O.G.E.C.' SCHEME - GROUPS OF '2' INTERNAL POINTS ARE
C COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
C W : IT HOLDS THE P.D.E.'S EXACT THEORETICAL VALUES AT THE
C INTERNAL POINTS AT THE MAXIMUM 'TIME-LEVEL'. COMPUTED
C DUE TO THE CHOSEN EXACT SOLUTION FORMULA.
C ERROR : IT HOLDS THE DIFFERENCES BETWEEN THE P.D.E.'S
C EXACT AND
C APPROPRIATE VALUES AT THE INTERNAL POINTS AT THE MAXIMUM
C TIME-LEVEL.
C Z : IT HOLDS AT EACH 'TIME-LEVEL' THE P.D.E.'S EXACT AND
C APPROXIMATE VALUES AT ALL THE BOUNDARY AND INTERNAL POINTS.  COMPUTED FROM THE
C ARRAY 'F' AND THE WORK-ARRAY 'U', RESPECTIVELY.
C ITIME : IT HOLDS THE TIMING INFORMATION.
C F : IT HOLDS THE P.D.E.'S EXACT VALUES AT THE POINTS ON BOTH
C BOUNDARIES, FOR ALL 'TIME-LEVELS', COMPUTED USING THE CHOSEN
C EXACT SOLUTION FORMULA.
C
REAL 10.IT,XX,KL,LDT
C
C SET THE SHARED DATA,
C
*SHARED U,NPOINT,NPROC,ITIME,ERROR,EPSLN,R,

-DX,DT,IN,INH,Z,IL,ILH,F,IR,DX2
C
C SET CRITICAL SECTIONS,
C
*REDIJ NPROC

C INITIALIZE PARALLELISM.
C
*USEPAR
NPROC=0
C
C GENERATE PARALLEL PATHS TO SET DYNAMICALLY THE NUMBER OF
C PROCESSORS TO BE UTILIZED EACH TIME IN THE PROGRAM.
C
C ENTER CRITICAL SECTION.
C
*ENTER NOPROC
NPROC=NPROC+1
C
C RELEASE CRITICAL SECTION.
C
*EXIT NOPROC
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
1000 *PAREND
C
C READ THE NUMBER OF INTERNAL POINTS, 'TIME-LEVELS' AND THE VALUE
C FOR THE GRID RATIO R = DT/(DX**2).
C
READ(5,99990) NPOINT,R
C
EPSLN=0.003
C
C COMPUTE THE GRID'S X-SPACING UNIT FOR THE RANGE [0,1].
C
DX=1.0/NPOINT
C
C COMPUTE THE GRID'S T-SPACING UNIT FOR THE RANGE [0,100].
C
DT=R*(DX**2)
C
WRITE(6,99991) NPOINT,ISTEP,EPSLN,R,DX,DT
C
C IN ACCORDANCE WITH THE UTILIZED NUMBER OF PROCESSORS EACH TIME.
C
C SET THE SIZE OF EACH SUBSET OF INTERNAL POINTS TO BE ASSIGNED
C TO A PARALLEL PATH ('GRANULARITY FACTOR#1'.
C
I=NPOINT/NPROC
C
C IN ACCORDANCE WITH THE UTILIZED NUMBER OF PROCESSORS EACH TIME.
C
C SET THE SIZE OF EACH SUBSET OF BOUNDARY POINTS TO BE ASSIGNED
C TO A PARALLEL PATH ('GRANULARITY FACTOR#1'.
C
IL=ISTEP/NPROC
C
C GENERATE PARALLEL PATHS TO SET LOCAL COPIES TO DIMINISH THE
C OVERHEADS DUE TO SHARED MEMORY ACCESSES.
C
*DOALL 1500
I=DOALL 1500
EPSLN=EPSLN
RL=R
DXL=DX
DTL=DT
DXL=DX2
INL=IN
IRL=IR
INPL=IN
NL=IL
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
1500 *PAREND
C
C COMPUTATION OF THE EXACT THEORETICAL VALUES.
C
C PHASE 2 AT THE INTERNAL POINTS, AT THE MAXIMUM 'TIME-
C LEVEL', USING THE CHOSEN EXACT SOLUTION FORMULA.
C
109 X*(I-1)*DXL
110 SDT=ISTEP*DTL
111 XM=0.5
112 A=0.05*((XM+4.95*SDT)/EPSLN)
113 B=0.25*(XM+0.75*SDT)/EPSLN
114 C=0.5*(XM-0.375)/EPSLN
115 X1=Exp(-A)
116 X2=Exp(-B)
117 X3=Exp(-C)
118 IT=0.1*X1+0.5*X2+X3
119 IQ=I+X2+X3
120 W(I)=IT/10
121 2000 CONTINUE
122 WRITE(6,99992)ISTEP,(W(I),I=1,2,1RL)
123 C
124 C START TIMING THE COMPUTATIONAL PROCEDURE.
125 C
126 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
127 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
128 $DOLL 2500
129 CALL TIMEST
130 2500 $PAREND
131 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
132 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
133 C
134 C $DOPAR 4000 J=1,NPROC
135 C
136 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
137 C $DOPAR 4000 J=1,NPROC
138 C
139 C IT=0.0
140 CI0=0.0
141 C GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED
142 C SUBSETS OF BOUNDARY POINTS.
143 C
144 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
145 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
146 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
147 C
148 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
149 C
150 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
151 C
152 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
153 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
154 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
155 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
156 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
157 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
158 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
159 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
160 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
161 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
162 3000 CONTINUE
C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.

C COMPUTE THE VALUE AT THE 'LEFT' UNGROUPED NEAR BOUNDARY POINT.

GO TO 7000

C COMPUTE THE VALUE AT THE 'RIGHT' UNGROUPED NEAR BOUNDARY POINT.

GO TO 8500

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.

IND=IND+2

C COPY THE COMPUTED, WITH THE 'O.E.C.' SCHEME, APPROXIMATE VALUES

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.

C AT THE INTERNAL POINTS, FROM THE WORK-ARRAY 'U' INTO THE ARRAY 'Z'.

DO 9500 I=2,1RL

Z(I)=U(I)

GO TO 9500

DO 10000 CONTINUE

CALL TIMOUT(ITIME)

C AT EVERY ALTERNATE TIME-LEVEL, FOR THE 'O.E.C.' SCHEME, GENERATE C 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED SUBSETS
C OF INTERNAL POINTS.
WRITE(6,99994) ISTEP,(I(I),I=2,IRL)
C COMPUTE AND PRINT OUT THE DIFFERENCES BETWEEN THE EXACT AND
C APPROXIMATE VALUES AT THE INTERNAL POINTS, AT THE MAXIMUM
C TIME-LEVEL.
DO 11000 K=2,IRL
ERROR(K)=W(K)-Z(K)
11000 CONTINUE
WRITE(6,99995) ERROR(I),I=2,IRL)
C COMPUTE THE MAXIMUM "ABSOLUTE ERROR"-"A.E."
DO 12000 I=2,IRL
IF(ABS(ERROR(I))>T1) GO TO 11500
11500 T1=ABS(ERROR(I))
T2=ABS(W(I))
12000 CONTINUE
C COMPUTE THE MAXIMUM "PERCENTAGE ERROR"-"P.E."
PERR=100*T1/T2
C PRINT OUT THE MAXIMUM 'A.E.' THE MAXIMUM 'P.E.'
WRITE(6,99996) T1,PERR
C SET THE FORMATS
C **************
FORMAT(I6,1X,16.1X,F6.2)
FORMAT(I6,1X,13.4X,'ISTEP =',13.4X,'EPSLN =',
E12.4X,'R =',E12.4X,'DX =',E14.6X,'DT =',E14.6,
15.3X,'ARE 1/3(E18.9,2X)/;
FORMAT(/'TIMINO=(*For SAGEONI*) 1/8(I6,2X))
FORMAT(I6,1X,'THE APPROXIMATE VALUES AT TIME-LEVEL 1',
15.3X,'ARE 1/3(E18.9,2X)/;
FORMAT(/'THE ERRORS ARE 1/3(E18.9,2X)/;
FORMAT(I6,1X,'THE MAXIMUM A.E. IS =',E18.9,
5X,'THE MAXIMUM P.E. IS =',E18.9)
C TERMINATE PROGRAM.
STOP
END
C THIS PROGRAM IMPLEMENTS THE "(DOUBLE) ALTERNATING GROUP EXPLICIT" -
C (D),A.G.E., METHOD TO SOLVE BURGERS' 1-D NON-LINEAR PARABOLIC P.D.E.
C THIS SCHEME IS THE PERIODIC ROTATION OF THE 'TWO' TIME-LEVEL
C (D),A.G.E., SCHEME RESULTING IN A 'FOUR' TIME-LEVEL STEP PROCESS
C WITH THE SECOND HALF CYCLE IN REVERSE ORDER.
C THE NATURE AND COMPLEXITY OF THE METHOD INEVITABLY INTRODUCE
C SYNCHRONIZATION ACTIVITIES WITHOUT ANY PERFORMANCE LOSS THOUGH.
C DUE TO THE FAST SYNCHRONIZATION TOOL OF THE 'NEPTUNE' PROTOTYPE
C SYSTEM.
C Set the required arrays.
C
DIMENSION U(I940),W(I940),ERROR(I940),Z(I940),TIME(I00)
DIMENSION F(500,2)
C Specification of the arrays
C
C U : It holds temporarily at each time-level the P.D.E.'s
C approximate values at all the internal points, the values
C at the 'left' and 'right' ungrouped near boundary points,
C at every time-level that the 'G.E.U.' scheme is applied.
C Are computed using Saul'yev's 'L>R', 'L<R' asymmetric
C formulae respectively, the values at all or the remaining
C -for the 'G.E.U.' scheme- groups of 2 internal points are
C computed using the group explicit finite-difference formulae.
C
W : It holds the P.D.E.'s exact theoretical values at the
C internal points at the maximum time-level computed using
C the chosen exact solution formula.
C Error : It holds the differences between the P.D.E.'s exact and
C approximate values at internal points at the maximum
C time-level.
C Z : It holds at each time-level the P.D.E.'s exact and approximate
C values at all the boundary and internal points copied from the
C array 'F' and the work-array 'U', respectively.
C ITIME : It holds the timing information.
C F : It holds the P.D.E.'s exact values at the points on both
C boundaries for all time-levels computed using the chosen
C exact solution formula.
C
REAL IT,K,W,KL,LDT
C Set the shared data.
C *SHARED U,NPOINT,NPROC,ITIME,EPSLN,R.
C
C Set critical sections.
C GENERATE "NPROC" PARALLEL PATHS AND ASSIGN TO THEM THE CREATED
C SUBSETS OF INTERNAL POINTS (*ZERO TIME-LEVEL*).
181     $DOPAR 5000 J=1,NPROC
182     C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.
183     IS=(J-1)*HL+1
184     IE=IS+HL-1
185     DO 4500 I=1,IE
186        X=I*DXL
187        XM=X-0.5
188        X=0.05*(XM+X-0.75*SDTL)/EPSLNL
189        IE=IS+IHL-1
190        IT=0.1*X1+0.5*X2+X3
191        IQ=I+1
192        XI=EXP(-IT)
193        X2=EXP(-IT)
194        XI=EXP(-B)
195        X2=EXP(-B)
196        X3=EXP(-C)
197        IT=0.1*X1+0.5*X2+X3
198     END DO
199     4500 CONTINUE
200     C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
201     5000 $PAREND
203     C PROCESS WITH THE SECOND HALF CYCLE IN REVERSE ORDER.
204     IF(K.GT.ISTEP) 00 TO 11500
205     11500 $PAREND
206     K=0
207     C INITIALIZE THE 'FLAG' CONTROLLING THE ALTERNATING SEQUENCE BETWEEN
208     C THE 'O.E.U.' AND 'O.E.C.' SCHEMES TO FORM A 'FOUR' TIME-LEVEL STEP
209     5500 IFLAG=0
210     6000 CONTINUE
211     2100 $PAREND
212     C TERMINATE THE ALTERNATING SEQUENCE WHEN THE NUMBER OF TIME-STEP
213     C IS EXHAUSTED.
214     IF(K.GT.ISTEP) GO TO 11500
215     11500 $PAREND
216     11500 $PAREND
COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.

IE=IS+IHL-1

DO 8000 I=IS,IE,2

IM1=I-1

IM2=I-2

IP1=I+1

IP2=I+2

IP3=I+3

K=DX2L*I(IP1)

K2=DX2L*I(I)

A1=EPSLNL-KL

A2=EPSLNL-KK

B1=EPSLNL-KL

B2=EPSLNL-KK

AIR=AIR+RL

A2R=A2R+RL

BIR=BI+RL

B2R=B2R+RL

D=1+AIR+B2R

C CHECK FOR THE 'LEFT' UNGROUPED NEAR BOUNDARY POINT.

IF((IpEQ,1) GO TO 7000

EP1=B2R*(1-BIR)*A(IM1)

EP2=(1+AIR-A2R-AIR+A2R)*Z(I)

EP3=AIR*AIR+Z(I2)+(1+AIR+A2R)*Z(IP2)

U(I)=(EP1+EP2)/D

C CHECK FOR THE 'RIGHT' UNGROUPED NEAR BOUNDARY POINT.

IF((IEQ,1) GO TO 7500

5000 CONTINUE

OP1=(1+B2R-BIR*B2R)*Z(I1)

OP2=(1-AIR-A2R+AIR-A2R)*Z(IP2)

OP3=(1+B2R)*BIR*Z(I)+AIR*A2R*Z(IP3)

U(I)=S+OP2/OP3/D

GO TO 8000

C COMPUTE THE VALUE AT THE 'LEFT' UNGROUPED NEAR BOUNDARY POINT.

A=EPSLNL-DX2L*Z(I2)

B=EPSLNL+DX2L*Z(I2)

RA=RA+RL

RB=RB+RL

U(2)=RA*Z(I1)+RA*Z(3)+(1-RB)*Z(2)/(1+RB)

GO TO 6500

C COMPUTE THE VALUE AT THE 'RIGHT' UNGROUPED NEAR BOUNDARY POINT.

A=EPSLNL-DX2L*Z(I1)

B=EPSLNL+DX2L*Z(I1)

RA=RA+RL

RB=RB+RL

U(I)=S+RA*Z(I)+RA*Z(3)+(1-RB)*Z(2)/(1+RB)
COPY THE COMPUTED, WITH THE 'G.E.C.' SCHEME, APPROXIMATE VALUES AT THE INTERNAL POINTS, FROM THE WORK-ARRAY 'U' INTO THE ARRAY 'Z'.

DO 11000 I=2, IRL
Z(I) = U(I)
11000 CONTINUE

IF (IFLAG.EQ.1) GO TO 6000
11500 CONTINUE

TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

*DOALL 12000
CALL TIMEOUTCITIME)
12000 *PAREND

WRITEC6,99993) ITIME
WRITEC6,99994) ISTEP, EPSLN)

COMPUTE AND PRINT OUT THE DIFFERENCES BETWEEN THE EXACT AND APPROXIMATE VALUES AT THE INTERNAL POINTS AT THE MAXIMUM TIME-LEVEL.

DO 12300 K=2, IRL
ERROR(K) = W(K) - Z(K)
12500 CONTINUE

THE ERRORS ARE

THE MAXIMUM ABSOLUTE ERROR 'A.E.' IS

THE MAXIMUM PERCENTAGE ERROR 'P.E.' IS

OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL PROCEDURE AND COMPUTE THE MAXIMUM ABSOLUTE ERROR AND THE MAXIMUM PERCENTAGE ERROR.

WRITEC6,99995) ERROR(I), I=2, IRL

WRITEC6,99996) Tt, PERR

C TERMINATE PROGRAM.

C SET THE FORMATS

FORMAT(1X,'THE EXACT THEORETICAL VALUES AT TIME-LEVEL 1',/15.3X, 'ARE 1'/(E18.9,2X)/)
FORMAT(1X,'THE APPROXIMATE VALUES AT TIME-LEVEL 1',/15.3X, 'ARE 1'/(E18.9,2X)/)
FORMAT(1X,'THE ERRORS ARE 1'/(E18.9,2X)/)
FORMAT(I6,1X,I6,F6.2)
C SET CRITICAL SECTIONS.
REGION NOPROC
C INITIALIZE PARALLELISM.
USEPAR
NPROC=0
C GENERATE PARALLEL PATHS TO SET DYNAMICALLY THE NUMBER OF
C PROCESSORS TO BE UTILIZED EACH TIME IN THE PROGRAM.
$DOALL 1000
C ENTER CRITICAL SECTION.
$ENTER NOPROC
NPROC=NPROC+1
C RELEASE CRITICAL SECTION.
$EXIT NOPROC
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
1000 $PAREND
C READ THE NUMBER OF INTERNAL POINTS, TIME-STEMPS AND THE VALUE
C FOR THE GRID RATIO \( \frac{R}{Dx} \).
WRITE(6,99991) NPOINT,ISTEP,EPSLN,R
C SET the grid's \( x \)-spacing unit, for the range \([0,1] \).
DX1=(/NPOINT+1)
C COMPUTE the grid's \( t \)-spacing unit, for the range \([0,\infty) \).
DT=R*(DX2**2)
C FORMULATE PARALLEL PATHS TO SET DYNAMICALLY.
WRITE(6,99991) NPOINT,ISTEP,EPSLN,R,DX,DT
DX2=DX/2
IN=NPOINT+2
IR=NPOINT+1
C IN ACCORDANCE WITH THE UTILIZED NUMBER OF PROCESSORS EACH TIME.
C SET THE SIZE OF EACH SUBSET OF INTERNAL POINTS TO BE ASSIGNED
C TO A PARALLEL PATH \( \ast \) GRANULARITY FACTOR \( \ast \).
I=1
NOPOINT/NPROC
C IN ACCORDANCE WITH THE UTILIZED NUMBER OF PROCESSORS EACH TIME.
C SET THE SIZE OF EACH SUBSET OF BOUNDARY POINTS TO BE ASSIGNED
C TO A PARALLEL PATH \( \ast \) GRANULARITY FACTOR \( \ast \).
IL=ISTEP/NPROC
C GENERATE PARALLEL PATHS TO SET LOCAL COPIES TO DIMINISH THE
C OVERHEADS DUE TO SHARED MEMORY ACCESSES.
$DOALL 1500
EPSLN=EPSLN
RL=R
DXL=DX
DTL=DT
DX2L=DX2
INL=IN
IRL=IR
INPTL=NPOINT
C COMPUTATION OF THE MAXIMUM TIME-LEVEL USING THE CHosen EXACT
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
1500 $PAREND
C COMPUTATION OF THE EXACT THEORETICAL VALUES
C AT ALL THE BOUNDARY AND INTERNAL POINTS AT
C THE MAXIMUM TIME-LEVEL USING THE CHosen EXACT
C FORMULAE, RESPECTIVELY.
THE VALUES
AT THE INTERNAL TIME-LEVEL, THE\( O.E.U. \) SCHEME IS APPLIED,
AT THE INTERNAL TIME-LEVEL, THE \( O.E.U. \) SCHEME IS APPLIED.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
THE VALUES AT ALL THE INTERNAL POINTS ARE
COMPUTED USING THE GROUP EXPLICIT FINITE-DIFFERENCE FORMULAE.
SOLUTION FORMULA.

DO 2000 I=1,INL
   X=([I-1]*DXL
   SDT=ISTEP*DTL
   XM=X-0.5
   A=0.05*(XM+4.95*SDT)/EPSLNL
   B=0.25*(XM+0.75*SDT)/EPSLNL
   C=0.5*(X-0.375)/EPSLNL
   RH=EXP(A)
   R2=EXP(-B)
   RX=EXP(-C)
   IT=0.1*(X+0.5)*X2+X3
   IL=RX+X2*X3
   W(I)=IT/IQ
   K=K+1
2000 CONTINUE

SUBSETS OF BOUNDARY POINTS.

DO 3500 K=IS,IE
   F(ISPI,K)=WINL)
   F(ISPIK)=WINL)
   WRITE(6,99992) KSTEP,(I,1=1,INL)
   L=K-1
   A=0.05*(DMIM+5.75*LDT)/EPSLNL
   B=0.25*(DMIM+0.75*LDT)/EPSLNL
   C=0.5*(X-0.375)/EPSLNL
   X=I*L
   X1=EXP(-A)
   X2=EXP(-B)
   X3=EXP(-C)
   IT=0.1*X1+0.5*X2+X3
   IL=X1+X2*X3
   W(I)=IT/IQ
3500 CONTINUE

3000 CONTINUE

3500 CONTINUE

300 CONTINUE

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
4000 #PEND

C COMPUTE THE START AND END POINTS FOR EACH SUBSET.
5000 #PEND

C SUBSETS OF INTERNAL POINTS (*ZERO TIME-LEVEL*).

C COMPUTE THE APPROPRIATE VALUES AT EACH SUBSET.

C COMPUTE THE PENULTIMATE TIME-LEVEL, USING THE CHOSEN

C EXACT SOLUTION FORMULA.

IT=0.0

DO 4500 I=IS,IE
   X=I*L
   XM=X-0.5
   A=0.05*X*EPSLNL
   B=0.25*X*EPSLNL
   C=0.5*(X-0.375)/EPSLNL
   X1=EXP(-A)
   X2=EXP(-B)
   X3=EXP(-C)
   IT=0.1*X1+0.5*X2+X3
   IL=X1+X2*X3
   W(I)=IT/IQ
4500 CONTINUE

4500 CONTINUE

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.

C COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED EXACT VALUES.

C AT BOTH BOUNDARY POINTS (*ZERO TIME-LEVEL*).

C EXPANDED FINITE-DIFFERENCE AND SAUL-YEV'S ASYMPTOTIC

C FORMULAE ACCORDINGLY.

C COPY THE COMPUTED EXACT VALUES AT BOTH BOUNDARY POINTS, AT

C THE MAXIMUM TIME-LEVEL, INTO THE APPROPRIATE ARRAY.

C START TIMING THE COMPUTATIONAL PROCEDURE.

C SUBSETS OF BOUNDARY POINTS.

C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.

C THE PENULTIMATE TIME-LEVEL, USING THE CHOSEN

C EXACT SOLUTION FORMULA.

IT=0.0

DO 4500 I=IS,IE
   X=I*L
   XM=X-0.5
   A=0.05*X*EPSLNL
   B=0.25*X*EPSLNL
   C=0.5*(X-0.375)/EPSLNL
   X1=EXP(-A)
   X2=EXP(-B)
   X3=EXP(-C)
   IT=0.1*X1+0.5*X2+X3
   IL=X1+X2*X3
   W(I)=IT/IQ
4500 CONTINUE

4500 CONTINUE

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.

C GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED

C SUBSETS OF BOUNDARY POINTS.

C GROUP SYNCHRONIZE PROCESSORS.

C COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED EXACT VALUES.

C AT BOTH BOUNDARY POINTS (*ZERO TIME-LEVEL*).

C EXPANDED FINITE-DIFFERENCE AND SAUL-YEV'S ASYMPTOTIC

C FORMULAE ACCORDINGLY.

C GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED

C SUBSETS OF BOUNDARY POINTS.

C GROUP SYNCHRONIZE PROCESSORS.

C COMPLETE EXACT VALUES AT ALL THE TIME-LEVELS.

C COUNTING THE APPROPRIATE VALUES AT EACH SUBSET.

C AT THE PENULTIMATE TIME-LEVEL, USING THE CHOSEN

C EXACT SOLUTION FORMULA.

IT=0.0

DO 4500 I=IS,IE
   X=I*L
   XM=X-0.5
   A=0.05*X*EPSLNL
   B=0.25*X*EPSLNL
   C=0.5*(X-0.375)/EPSLNL
   X1=EXP(-A)
   X2=EXP(-B)
   X3=EXP(-C)
   IT=0.1*X1+0.5*X2+X3
   IL=X1+X2*X3
   W(I)=IT/IQ
4500 CONTINUE

4500 CONTINUE

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.

C GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED

C SUBSETS OF BOUNDARY POINTS.

C GROUP SYNCHRONIZE PROCESSORS.

C COMPLETE EXACT VALUES AT ALL THE TIME-LEVELS.

C COUNTING THE APPROPRIATE VALUES AT EACH SUBSET.

C AT THE PENULTIMATE TIME-LEVEL, USING THE CHOSEN

C EXACT SOLUTION FORMULA.

IT=0.0

DO 4500 I=IS,IE
   X=I*L
   XM=X-0.5
   A=0.05*X*EPSLNL
   B=0.25*X*EPSLNL
   C=0.5*(X-0.375)/EPSLNL
   X1=EXP(-A)
   X2=EXP(-B)
   X3=EXP(-C)
   IT=0.1*X1+0.5*X2+X3
   IL=X1+X2*X3
   W(I)=IT/IQ
4500 CONTINUE

4500 CONTINUE

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.

C GENERATE 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED

C SUBSETS OF BOUNDARY POINTS.

C GROUP SYNCHRONIZE PROCESSORS.

IF(K.GT.1STEP) GO TO 10500

C GENERATE FOR THE 'G.E.U.' SCHEME, 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED SUBSETS OF INTERNAL POINTS.

DO 6000 I=IS,IE,2

C GENERATE, FOR THE 'G.E.C.' SCHEME, 'NPROC' PARALLEL PATHS AND ASSIGN TO THEM THE CREATED SUBSETS OF INTERNAL POINTS.

SDOPAR 6500 J=I,NPROC

C COMPUTE THE 'START' AND 'END' POINTS FOR EACH SUBSET.

IS=CJ-1>•IHL+2

IE=IS+IHL-1

DO 6000 I=IS,IE,2

IPI=I+1

IP2=I+2

IM1=I-1

KL=DX2L*ZC(I)

KK=DX2L*ZC(IP1)

A1=EPSLNK-KL

A2=EPSLNK-KK

Bl=EPSLNK+KL

B2=EPSLNK+KK

AIR=A1*RL

A2R=A2*RL

BlR=Bl*RL

B2R=B2*RL

D=I+AIR+B2R

C CHECK FOR THE 'LEFT' UNGROUPED NEAR BOUNDARY POINT.

IF(I.EQ.2) GO TO 8200

EP1=82R*C1-B1R>•ZC1M1>

EP2=C1+A1R-A2R-A1R•A2R>•ZC1>

EP3=B1R•82R•ZC1M2)+(1+A1R>•A2R•ZCIP1)

UC1I>=<EP1+EP2+EP3)/0

C CHECK FOR THE 'RIGHT' UNGROUPED NEAR BOUNDARY POINT.

IF<I.EQ.INPTL) GO TO 8500

8000 CONTINUE

C COMPUTE THE VALUE AT THE 'LEFT' UNGROUPED NEAR BOUNDARY POINT.

A=EPSLNK-DX2L*Z(I)

B=EPSLNK+DX2L*Z(I)

RA=RL*A

RB=RL*B

UCIL=CRB•ZC1)+RA•ZC3)+Cl-RB)*ZCIP1))/Cl+RA>

GO TO 8000

C COMPUTE THE VALUE AT THE 'RIGHT' UNGROUPED NEAR BOUNDARY POINT.

A=EPSLNK-DX2L*Z(I)

B=EPSLNK+DX2L*Z(I)

RA=RL*A

RB=RL*B

UCIP1=(RA*Z(INPL)+RB*z(INPL)+1-RB)*Z(I)}/(1+RB)

GO TO 9800

C COMPUTE THE VALUE AT THE 'RIGHT' UNGROUPED NEAR BOUNDARY POINT.

A=EPSLNK-DX2L*Z(I)

B=EPSLNK+DX2L*Z(I)

RA=RL*A

RB=RL*B

UCIP1=(RA*Z(INPL)+RB*z(INPL)+1-RB)*Z(I)}/(1+RB)

GO TO 9800

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.

9500 END


IF(K.GT.1STEP) GO TO 10500

KPI=K+1


IF(K.GT.1STEP) GO TO 10500

C IN ACCORDANCE WITH THE COUNTER 'K', FOR THE 'G.E.U.' SCHEME, COPY INTO THE APPROPRIATE ARRAY THE PRE-COMPUTED EXACT VALUES AT BOTH BOUNDARY POINTS.

Z(I)=F(KP1,I)
COPY THE COMPUTED, WITH THE 'G.E.U.' SCHEME, APPROXIMATE VALUES AT THE INTERNAL POINTS, FROM THE WORK-ARRAY 'U' INTO THE ARRAY 'Z'.

DO 10000 I=2,IRL
   Z(I)=W(I)
10000 CONTINUE

IF(IFLAG.EQ.1) GO TO 5500

IFLAG=1
GO TO 7500

CONTINUE

TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

CALL TIMOUT(TIM
e

OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL PROCEDURE AND COMPUTE THE MAXIMUM ABSOLUTE ERROR AND THE MAXIMUM PERCENTAGE ERROR.

WRITE(6,99993) TIME
WRITE(6,99994) ISTEP, (Z(I), I=2,IRL)

WRITE(6,99995) (ERROR(I), I=2,IRL)

T1=0.0
T2=0.0

WRITE(6,99996) Ti, PERR

SET THE FORMATS

FORMAT(6,I6.1X,16,F6.2)
FORMAT(6,I4.4X,'NPOINT =',I4.4X,'ISTEP =',I3.4X,'EPSLN ='
                   + E12.4X,'R =',E12.4X,'DX =',E14.6X,'DT =',E14.6)

THE EXACT THEORETICAL VALUES AT TIME-LEVEL 1',

15.3X,'ARE : /3(E18.9,2X)/

THE ERRORS ARE : /3(E18.9,2X)/

THE MAXIMUM A.E. IS = ',E18.9,

THE MAXIMUM P.E. IS = ',E18.9)
APPENDIX
C-V

A SELECTION OF
OPTIMIZED PARALLEL COMPUTER PROGRAMS
FOR THE TRIDIAGONAL
LINEAR SYSTEM SOLVERS
C THIS PROGRAM IMPLEMENTS THE GAUSS ELIMINATION METHOD TO
C SOLVE THE MATRIX EQUATION AY = V , SEQUENTIALLY, WHERE 'A'
C IS A TRIDIAGONAL PERIODIC MATRIX WITH CONSTANT-DIAGONAL
C AND UNIT OFF-DIAGONAL ENTRIES.
C
C SET THE REQUIRED ARRAYS.
C
C DIMENSION A(2048), Y(2048), D(2048), V(2048)
C DIMENSION ITIME(100)
C
C SPECIFICATION OF THE ARRAYS
C
C A : IT HOLDS THE CONSTANT-DIAGONAL ENTRIES OF MATRIX 'A'.
C V : IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM.
C D : IT HOLDS THE PERIODIC ENTRIES OF MATRIX 'A'.
C Y : IT HOLDS THE SOLUTION OF THE SYSTEM.
C ITIME : IT HOLDS THE TIMING INFORMATION.
C
C USEPAR
C
C READ(5, 9990) IPRT
C READ(5, 9991) CONS, M
C WRITE(6, 9991) CONS, M
C
C THE SIZE OF MATRIX 'A'.
C
N=2*M
C
C THE PERIODIC ENTRIES OF MATRIX 'A'.
C
D(1)=1.0
C
C INITIALIZE OF THE CONSTANT-DIAGONAL ENTRIES OF MATRIX 'A'.
C
DO 10 I=1, N
A(I)=CONS
10 CONTINUE
C
C READ THE R.H.S. ELEMENTS OF THE SYSTEM.
C
DO 15 I=1, N
READ(5, 9992) V(I)
15 CONTINUE
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C [DOALL 23 CALL TIMEST ]
C
C [DOALL 23 ]
C
C [DOALL 24 CALL TIMEOUT(ITIME) ]
C
C CALL TIMEST
C
C C PERFORM THE ELIMINATION PROCEDURE THAT MANY TIMES UNTIL
C C THE ORIGINAL TRIDIAGONAL MATRIX IS REDUCED TO A FULL
C
C (3X3) MATRIX.
C
INN=N-3
C
DO 20 K=1, INN
FACT=1.0/A(K)
A(K+1)=A(K+1)-FACT
V(K+1)=V(K+1)-FACT*V(K)
20 CONTINUE
C
C THE PERIODICITY OF THE SYSTEM.
C
FACT=D(K)/A(K)
D(K+1)=FACT
A(N-1)=A(N-1)-FACT
V(N)=V(N)-FACT*V(K)
C
C THE ORIGINAL TRIDIAGONAL MATRIX HAS BECOME A FULL
C (3X3) MATRIX. CARRY ON THE GAUSS ELIMINATION PROCESS.
C
FACT=1.0/A(N-2)
A(N-1)=A(N-1)-FACT
D(N-1)=1.0-FACT*D(N-2)
V(N-1)=V(N-1)-FACT*V(N-2)
C
C THE PERIODICITY OF THE SYSTEM.
C
FACT=D(N-2)/A(N-2)
A(N-1)=A(N-1)-FACT*D(N-2)
V(N)=V(N)-FACT*V(N-2)
C
C THE FINAL ELIMINATION PROCESS IN THE RESULTING (2X2) MATRIX.
C
FACT2=1.0-FACT
FACT2=FACT2/A(N-1)
A(N)=A(N)-FACT2*D(N-1)
V(N)=V(N)-FACT2*V(N-1)
C
C THE SOLUTION PROCESS [Back-substitution].
C
Y(N)=V(N)/A(N)
Y(N-1)=(V(N-1)-D(N-1)*Y(N))/A(N-1)
INJ=N-2
DO 30 I=1, INJ
30 Y(N-I)=Y(N-I)-D(N-I)*Y(N)/A(N)
C
C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
C
WRITE(6, 9994)
C
C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL
C PROCEDURE.
C
WRITE(6, 9994) (V(I), I=1, N)
C SET THE FORMATS

C

9990 FORMAT(I2)
9991 FORMAT(F4.2,2X,I2)
9992 FORMAT(F8.4)
9993 FORMAT(/,'TIMING-(#For GAUSCDP#) :',8(I6,2X))
9994 FORMAT(/,'THE SOLUTION IS :',5(E12.5,2X))

C TERMINATE PROGRAM.

9999 STOP
$END
C THE PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE
C TO SOLVE THE MATRIX EQUATION: AX=B. WHERE 'A' IS A TRIANGULAR
C PERIODIC MATRIX WITH CONSTANT-DIAGONAL AND UNIT OFF-DIAGONAL
C ENTRIES.
C THE NUMBER OF CREATED PATHS IS ALWAYS EQUAL TO THE NUMBER OF
C AVAILABLE PROCESSORS EACH TIME.
C THEN, AFTER A PRE-SET NUMBER OF REDUCTION STEPS, WE CONTINUE BY
C APPLYING THE GAUSS ELIMINATION SEQUENTIAL PROCEDURE TO SOLVE THE
C RESULTING SUBSYSTEMS, IN EACH OF THE CREATED PARALLEL PATHS.
C SIMULTANEOUSLY.
C SET THE REQUIRED ARRAYS.
C SPECIFICATION OF THE ARRAYS
C ======================================================================
W : IT HOLDS THE MULTIPLIERS OF THE CONSTANT-DIAGONAL ENTRY
OF MATRIX 'A' DURING THE ELIMINATION PROCESS.
B : IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM. ON THIS
ARRAY WILL BE APPLIED THE 'EVEN' STREAM OF THE CYCLIC
ODD-EVEN REDUCTION PROCEDURE.
C : IT HOLDS A COPY OF THE ARRAY 'B', ON THIS ARRAY WILL BE
APPLIED THE 'ODD' STREAM OF THE CYCLIC ODD-EVEN REDUCTION
PROCEDURE.
X : IT HOLDS THE SOLUTION OF THE SYSTEM.
IND: IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
ODD-EVEN REDUCTION PROCEDURE.
INDO: IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
ODD-EVEN REDUCTION PROCEDURE.
CITIME: IT HOLDS THE TIMING INFORMATION.
INTEGER LS
C SET THE SHARED DATA.
C *SHARED N.R.BX.IT.CW.INDO.ITIME.NSTEP.IJ.
C SET CRITICAL SECTIONS.
C REGION NOPROC
C INITIALIZE PARALLELISM.
0 USEPAR
NPROC=0
C GENERATE PARALLEL PATHS TO SET DYNAMICALLY THE NUMBER OF
C PROCESSORS TO BE UTILIZED EACH TIME IN THE PROGRAM.
*DOALL 2000
C ENTER CRITICAL SECTION.
C RELEASE CRITICAL SECTION.
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
DO 2000 IEND
READ(5,9991) R
WRITE(6,9992) R
W(I)=R
C THE SIZE OF MATRIX 'A'.
DO 60 J=1,IT
C READ THE R.H.S. ELEMENTS OF THE SYSTEM.
DO 50 J=1,IT
READ(5,9993) B(J)
C MAKE A COPY OF THE R.H.S. ELEMENTS OF THE SYSTEM.
C(J)=B(J)
C INITIALIZE THE ARRAYS OF INDICES.
IND(E)=J
INDO(J)=J
20 CONTINUE
C READ THE "DEPTH" OF RECURSION UPON WHICH THE GRANULARITY
C FACTOR DEPENDS.
DO 80 I=1,IT
READ(7,9994) IZ
WRITE(6,9995) IZ
C START TIMING THE COMPUTATIONAL PROCEDURE.
C START TIMING THE COMPUTATIONAL PROCEDURE.
C C [ ] [ ] [ ] [ ] [ ] [ ]
C [ ] [ ] [ ] [ ] [ ] [ ]
C [ ] [ ] [ ] [ ] [ ] [ ]
C [ ] [ ] [ ] [ ] [ ] [ ]
C CALL TIMES
C 45 *PEND
C [ ] [ ] [ ] [ ] [ ] [ ]
C [ ] [ ] [ ] [ ] [ ] [ ]
C [ ] [ ] [ ] [ ] [ ] [ ]
C [ ] [ ] [ ] [ ] [ ] [ ]
C START THE REDUCTION PROCESS.
DO 92 NSTEP=1,IT
IN=2**(NSTEP-1)
J=IT/IN
IF(NSTEP.EQ.IZ) GO TO 999
W(NSTEP+1)=2-W(NSTEP)*W(NSTEP)
N=N-1
1D=2**NSTEP
1D=IT/1D
IF(NPROC.NE.1) GO TO 1002
IA=1
GO TO 1003
1003 IA=IT/IA
IF(NPROC.EQ.4) GO TO 3000
1002 IA=NPROC/2
1001 IA=IT/IA
IF(NPROC.NE.1) GO TO 1002
1003 IA=IT/IA
IF(NPROC.EQ.4) GO TO 3000
1002 IA=NPROC/2
1001 IA=IT/IA
1000 IA=IT/IA
GO TO 1003
999 W(I)=W(IZ)
1008 IU=IT/2**(I+N+1)
C PROCEDURE.
C FROM ELIMINATION SUBROUTINE TO CHECK
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C CREATE
C
C PROGRAM IS EXECUTED.
C
C GENERATE 'NPROC' PARALLEL PATHS.
C
C IN CASE THAT ALL PROCESSORS ARE UTILIZED THEN THIS PART OF THE
C PROGRAM IS EXECUTED.
C
C GENERATE 'NPROC' PARALLEL PATHS.
C
C CHECK FOR THE 'ODD' AND 'EVEN' COMPUTATIONAL STREAMS OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C
C CHECK FOR THE 'ODD' STREAM OF THE CYCLIC ODD-EVEN REDUCTION
C
C CHECK FOR THE 'ODD' STREAM OF THE CYCLIC ODD-EVEN REDUCTION
C
C CHECK FOR THE PERIODICITY OF THE SYSTEM.
C
C CHECK FOR THE PERIODICITY OF THE SYSTEM.
C
C CHECK FOR THE PERIODICITY OF THE SYSTEM.
C
C CHECK FOR THE PERIODICITY OF THE SYSTEM.
C
C CHECK FOR THE PERIODICITY OF THE SYSTEM.
C
C CHECK FOR THE PERIODICITY OF THE SYSTEM.
217 IF(L.EQ.10) GO TO 510
218 B(INDE(J))=B(INDE(J-1))-W(NSTEP)*B(INDE(J))+B(INDE(J+1))
219 GO TO 91
220 510 L=0
221 B(INDE(J))=B(INDE(J-1))-W(NSTEP)*B(INDE(J))+Z
222 CONTINUE
223 91 CONTINUE
224 400 IF(NSTEP.EQ.1) GO TO 103
225 103 K=4
226 IS=(I-1)*IB+1
227 IE=IS+I-1
228 L=0
229 DO 166 J=IS,IE,K
230 CALL CYCC(IS,IE,INDO)
231 CONTINUE
232 166 DO 3006 IC=1,IT
233 IF(I.EQ.2) GO TO 3007
234 IS=I*(IC-1)+1
235 IE=IS+IC-1
236 L=0
237 DO 3008 J=IS,IE,L
238 CALL CYC(IS,IE,INDO)
239 CONTINUE
240 3006 CONTINUE
241 3007 DO 206 IC=1,IT
242 IF(I.EQ.2) GO TO 207
243 IS=I*(IC-1)+1
244 IE=IS+IC-1
245 L=0
246 DO 208 J=IS,IE,L
247 CALL CYC(IS,IE,INDO)
248 CONTINUE
249 206 CONTINUE
250 207 DO 3006 IC=1,IT
251 IF(I.EQ.2) GO TO 3007
252 IS=I*(IC-1)+1
253 IE=IS+IC-1
254 L=0
255 DO 208 J=IS,IE,L
256 CALL CYC(IS,IE,INDO)
257 CONTINUE
258 3006 CONTINUE
259 3007 DO 206 IC=1,IT
260 IF(I.EQ.2) GO TO 208
261 D=INDO(IS+IC-1)
262 GO TO 209
263 209 121 D=INDE(IS+1)-W(NSTEP)*D+INDE(IS+1)
264 CONTINUE
265 121 CONTINUE
266 208 D=INDE(IS+2)-W(NSTEP)*D+INDE(IS+2)
267 CONTINUE
268 63 IF(NSTEP.EQ.12) GO TO 60
269 60 CONTINUE
270 520 CALL TIMOUT(ITIME)
271 525 CALL SPAREND
272 C INTERCHANGE THE COMPUTED R.H.S. VALUES VIA THE
273 C APPROPRIATE ARRAYS OF INDICES.
274 DO 140 J=2,IT,2
275 140 CONTINUE
276 C SHUFFLE THE 'EVEN' AND 'ODD' USED INDICES TO THE
277 C TOP OF THE CORRESPONDING ARRAYS.
278 IF(NSTEP.EQ.12-1) GO TO 60
279 60 CONTINUE
280 169 CONTINUE
281 60 CONTINUE
282 C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
283 WRITE(6,9996) ITIME
284 WRITE(6,9997)(X(I),I=1,IT)
285 C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL
286 C PROCEDURE.
287 WRITE(6,9998) ('THE SOLUTION IS ',/7(E15.6,2X))
325 C TERMINATE PROGRAM.
326 370   $STOP
327 $END
328 C
329 C
330 C -----------------------------
331 C * THE GAUSS ELIMINATION *
332 C -----------------------------
333 C
334 C
335 C SUBROUTINE CYCLIC(IS,IE,INDEX)
336 C
337 C
338 C THIS SUBROUTINE SOLVES THE MATRIX EQUATION \( QX=E \), SEQUENTIALLY.
339 C WHERE \( Q \) IS A TRIDIAGONAL PERIODIC MATRIX WITH CONSTANT-DIAGONAL.
340 C AND UNIT OFF-DIAGONAL ENTRIES. THE IMPLEMENTATION IS AS THAT OF THE
341 C PROGRAM 1 MBS2.GAUSSCDP BUT MAKING USE OF THE ARRAYS OF INDICES
342 C SPECIFIED IN THE MAIN PROGRAM.
343 C
344 C DIMENSION A(512),D(512),X(2048),INDO(2048)
345 C DIMENSION E(2048),W(10),B(2048),C(2048),INDE(2048)
346 C DIMENSION ITIME(100),INDEX(2048)
347 C
348 C
349 C SPECIFICATION OF THE ADDITIONAL ARRAYS
350 C ************************************************************
351 C
352 C A : IT HOLDS THE CONSTANT-DIAGONAL ENTRIES OF MATRIX \( Q \).
353 C D : IT HOLDS THE PERIODIC ENTRIES OF MATRIX \( Q \).
354 C E : IT HOLDS THE VALUES OF EITHER OF \( B \) OR \( C \) ARRAY
355 C DEPENDING ON THE STREAM OF THE CYCLIC ODD-EVEN REDUCTION
356 C PROCEDURE CALLING THE SUBROUTINE.
357 C INDEX : IT HOLDS THE INDICES OF EITHER \( \langle \) INDE \( \rangle \) OR \( \langle \) INDO \( \rangle \) ARRAY
358 C DEPENDING ON THE STREAM OF THE CYCLIC ODD-EVEN REDUCTION
359 C PROCEDURE CALLING THE SUBROUTINE.
360 C $SHARED M,R,B,X,I,J,C,W,INDO,INDEX,NSTEP,IlJ,
361 - 10,J,12,NPROC,IB,IIU,III
362 C THE SIZE OF MATRIX \( Q \).
363 N=2#M
364 C THE PERIODIC ENTRIES OF THE MATRIX.
365 D(1)=1.0
366 C INITIALIZATION OF THE CONSTANT-DIAGONAL ENTRIES OF MATRIX \( Q \).
367 DO 890 I=1,N
368 A(I)=W(I)
369 CONTINUE
370 C PERFORM THE ELIMINATION PROCEDURE THAT MANY TIMES UNTIL
371 C THE CONSIDERED MATRIX EACH TIME IS REDUCED TO A FULL
372 C (3X3) MATRIX.
373 ILK=N-3
374 DO 882 K=1,ILK
375 IS=IS+K
376 IST=IS-1
377 ISL=IS+N
378 IS=ISL-1
379 FACT=1.0/A(K)
380 A(K+1)=A(K+1)-FACT
381 E(INDEX(IS))+E(INDEX(IS))=FACT*E(INDEX(IS))
382 C THE PERIODICITY OF THE SYSTEM.
383 FACT=D(K)/A(K)
384 D(K+1)=FACT
385 C WRITE(6,9999) D(K+1)
386 A(N)=A(N)-FACT*D(K)
387 C WRITE(6,9999) A(N)
388 E(INDEX(ISJ))=E(INDEX(ISJ))-FACT*E(INDEX(ISJ))
389 C WRITE(6,9999) E(INDEX(ISJ))
390 CONTINUE
391 C THE TRIAGONAL MATRIX HAS BECOME A FULL (3X3) MATRIX. CARRY ON
392 C THE GAUSS ELIMINATION PROCESS.
393 ISK=IS+N-1
394 ISX=IS-1
395 FACT=1.0/A(N-2)
396 A(N-1)=A(N-1)-FACT
397 D(N-1)=1.0*FACT*D(N-2)
398 E(INDEX(ISJ))=E(INDEX(ISJ))-FACT*E(INDEX(ISJ))
399 C
400 C THE PERIODICITY OF THE SYSTEM.
401 FACT=D(N-2)/A(N-2)
402 A(N)=A(N)-FACT*D(N-2)
403 E(INDEX(ISJ))=E(INDEX(ISJ))-FACT*E(INDEX(ISJ))
404 C THE FINAL ELIMINATION PROCESS IN THE RESULTING (2X2) MATRIX.
405 FACT=1.0-FACT
406 FACT=FACT/2/A(N-1)
407 A(N)=A(N)-FACT*D(N-1)
408 E(INDEX(ISJ))=E(INDEX(ISJ))-FACT*E(INDEX(ISJ))
409 C
410 C THE SOLUTION PROCESS [Back-substitution].
411 C
412 X(INDEX(ISJ))=E(INDEX(ISJ))/A(N)
413 X(INDEX(ISJ))=X(INDEX(ISJ))-A(N)-1
414 - D(N-1)*X(INDEX(ISJ))/A(N-1)
415 ILK=N-2
416 DO 883 I=1,ILJ
417 I=N-1-II
418 INT=IS-1
419 INT=INT-1
420 X(INDEX(IMI))=E(INDEX(IMI))-X(INDEX(IMI))*D(I)*X(INDEX(ISJ))/A(I)
422 CONTINUE
423 C
424 C SET THE FORMATS
425 C
426 C
427 9999 FORMAT(F25.9)
428 RETURN
429 $END
C DIRECT, FILENAME: MB2, POERSBP

C

C

C THIS PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE
C TO SOLVE THE MATRIX EQUATION A.X=B, WHERE 'A' IS A TRIDIAGONAL
C PERIODIC MATRIX WITH CONSTANT-DIAGONAL AND UNIT OFF-DIAGONAL
C ENTRIES.
C THE NUMBER OF CREATED PATHS IS ALWAYS EQUAL TO THE NUMBER
C AVAILABLE PROCESSORS EACH TIME.
C THEN, AFTER A PRE-SET NUMBER OF REDUCTION STEPS, WE CONTINUE BY
C APPLYING THE ODD-EVEN REDUCTION TECHNIQUE AGAIN, BUT SEQUENTIALLY
C THIS TIME, TO SOLVE THE RESULTING SUBSYSTEMS IN EACH OF THE CREATED
C PARALLEL PATHS SIMULTANEOUSLY.
C
C
C SET THE REQUIRED ARRAYS.
C
C
C DIMENSION W(10), B(2048), C(2048), X(2048), INDE(2048)
C DIMENSION ITIME(100), INDO(2048)

C SPECIFICATION OF THE ARRAYS
C

C W: IT HOLDS THE MULTIPLIERS OF THE CONSTANT-DIAGONAL ENTRY
C OF MATRIX 'A' DURING THE ELIMINATION PROCESS.
C B: IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM. ON THIS
C ARRAY WILL BE APPLIED THE 'EVEN' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C C: IT HOLDS A COPY OF THE ARRAY 'B'. ON THIS ARRAY WILL BE
C APPLIED THE 'ODD' STREAM OF THE CYCLIC ODD-EVEN REDUCTION
C PROCEDURE.
C X: IT HOLDS THE SOLUTION OF THE SYSTEM.
C INDE: IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C INDO: IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C ITIME: IT HOLDS THE TIMING INFORMATION.
C
C INTEGER LS
C
C SET THE SHARED DATA.
C
C $SHARED N,R,B,I IT,C,W,INDE,INDO,ITIME,NSTEP,JJ
C - IO.1A, IZ,NPROC.IB,IU,IHZ

C SET CRITICAL SECTIONS.
C
C *REGION NOPROC

C INITIALIZE PARALLELISM.
C
C *USEPAR

C GENERATE PARALLEL PATHS TO SET DYNAMICALLY THE NUMBER OF
C Processors TO BE UTILIZED EACH TIME IN THE PROGRAM.
C
C ENTER CRITICAL SECTION.
C
C
C RELEASE CRITICAL SECTION.
C
C EXIT NPROC

C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C 2000 *PEND

C READ(5,9990) IPRINT
C READ(5,9991) R,N
C WRITE(6,9992) R,N

C THE SIZE OF MATRIX 'A'.
C IT=2*N
C
C READ THE R.H.S. ELEMENTS OF THE SYSTEM.
C DO 20 J=1, IT
C READ(5,9993) B(J)

C MAKE A COPY OF THE R.H.S. ELEMENTS OF THE SYSTEM.
C C(J)=B(J)

C INITIALIZE THE ARRAYS OF INDICES.
C INDE(J)=J
C INDO(J)=J

C 20 CONTINUE

C READ THE "DEPTH" OF RECURSION UPON WHICH THE GRANULARITY
C FACTOR DEPENDS.
C
C READ(7,9994) IZ
C WRITE(6,9995) IZ

C C

C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C C [I][I][I][I][I][I][I][I]
C C [I][I][I][I][I][I][I][I]
C [I][I][I][I][I][I][I][I]
C [I][I][I][I][I][I][I][I]
C
C C
C START THE REDUCTION PROCESS.
C
C DO 60 NSTEP=1, IZ
C IN=2**(NSTEP-1)
C JJ=IT/IN
C IF(NSTEP.EQ.1) GO TO 999
C W(NSTEP+1)=W(NSTEP)+W(NSTEP)
C N=N-1
C ID=2*NSTEP
C ID=IT/ID
C IF(NPROC.NE.1) GO TO 1002
C IA=1
C GO TO 1003
C IA=NPROC/2
C IB=IT/IA
C IF(NPROC.EQ.4) GO TO 3000
C GO TO 1002
C 1002 IA=NPROC/2
C 1003 IB=IT/IA
C 1004 IF(NPROC.EQ.4) GO TO 3000
C 1005 GO TO 1000
C 1006 999 I=I+1
C 1007 IU=IT/2**(N+1)
C 1008 IU=IU/2

C A

C B

C C

C C

C C

C C

C C
C CHECK FOR THE PERIODICITY OF THE SYSTEM.
IF(J.EQ.IS) GO TO 131

C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
148 100 IF(NSTEP.EQ.1) GO TO 93
149 IS=1
150 IE=IB
151 GO TO 106
152 93 DO 3002 IC=1,IC
153 IS=(IC-1)*I+J-1
154 IE=IS+I+J-1
155 CALL SYMCDPC(I,IS,IE,INDO)
156 3002 CONTINUE
157 GO TO 71
158 106 L=0
159 DO 120 J=IS,IE,2
160 L=L+1
161 C CHECKS FOR THE PERIODICITY OF THE SYSTEM.
162 IF(J.EQ.IS) GO TO 131

163 IF(L.EQ.I0) GO TO 130
164 C(INDO(J))=C(INDO(J-1))-(W(NSTEP)*C(INDO(J))+C(INDO(J+1))
165 GO TO 120
166 130 L=0
167 GO TO 132
168 131 DO 130 IS=1,IC+1
169 132 C(INDO(J))=C(INDO(J-2*L-1))
170 L=0
171 CONTINUE
172 C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
173 71 NDEND
175 GO TO 63
176 C
177 C IN CASE THAT ALL PROCESSORS ARE UTILIZED THEN THIS PART OF THE
178 C PROGRAM IS EXECUTED.
179 C
180 C GENERATE 'NPROC' PARALLEL PATHS.
181 3000 SDOPAR 70 1=1,NPROC
182 C CREATE THE 'ODD' AND 'EVEN' COMPUTATIONAL STREAMS OF THE
183 C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
184 3001 CONTINUE
185 IF(NSTEP.EQ.1) GO TO 117
186 C CHECK THE "DEPTH" OF RECURSION IN ORDER TO CALL THE
187 C SUBROUTINE PERFORMING THE ODD-EVEN REDUCTION TO SOLVE
188 C THE SUBSYSTEMS RESULTING FROM THE 'ODD' STREAM OF THE
189 C CYCLIC ODD-EVEN REDUCTION PROCEDURE IN THE MAIN PROGRAM.
190 IF(NSTEP.EQ.12) GO TO 127
191 K=2
192 IS=1-3*IB+2
193 IE=IS+1-2
194 GO TO 191
195 117 K=4
196 IS=(1-2)*IB+2
197 IE=IB
198 IF(L.EQ.I0) GO TO 157
199 GO TO 181
200 157 L=10/2
201 GO TO 182
202 127 DO 3003 IC=1,IC
203 3004 IF(L.EQ.4) GO TO 3004
204 IS=(1-2)*(IC-1)*I+1
205 IE=IS+I+1
206 GO TO 3003
207 3004 IS=IB/2*(1-3)*(IC-1)*I+1
208 IE=IS+I-1
209 3005 CALL SYMCDPC(I,IS,IE,INDO)
210 3003 CONTINUE
211 GO TO 70
212 181 L=0
213 182 DO 91 J=IS,IE,2
214 91 L=L+1
215 C CHECK FOR THE PERIODICITY OF THE SYSTEM.
216 IF(L.EQ.I0) GO TO 510
C SUBROUTINE PERFORMING THE ODD-EVEN REDUCTION TO SOLVE THE SUBSYSTEMS RESULTING FROM THE 'ODD' STREAM OF THE CYCLIC ODD-EVEN REDUCTION PROCEDURE IN THE MAIN PROGRAM.

103 IF(INSTEP.EQ.IZ) GO TO 166
166 DO 3006 IC=1,1M
   IF(IE.EQ.2) GO TO 3007
   IS=IS+IC-1+1
   IE=IE+IB-1
   L=0
3007 GO TO 3008
3008 CALL SYMCDP(C,IS,IE,INDO)
3009 CONTINUE
321 C APPROPRIATE ARRAYS OF INDICES.
322 DO 140 J=2,IT,2
323 C(INDE(J))=B(INDE(J-1)) W(NSTEP)*B(INDE(J))+B(INDE(J+1))
324 CONTINUE
325 C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
326 CALL TIMOUT(TIM)
327 WRITE(6,9997) TIM
328 WRITE(6,9990) TIM
329 FORMAT(1X,'THE SOLUTION IS :',/15E15.6,2X)
329 C TERMINATE PROGRAM.
330 C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL PROCEDURE.
331 IF(IPRINT.EQ.0) GO TO 370
332 WRITE(6,9997) (X(I),I=1,IT)
333 C CHECKS FOR THE PERIODICITY OF THE SYSTEM.
334 IF(NSTEP.EQ.1) GO TO 1021
335 L=1
336 IF(L.EQ.(IT+1)) GO TO 232
337 IF(J.EQ.IS) GO TO 231
338 DO 1022 TO 230
339 1022 IF(J.EQ.IS) GO TO 231
340 C(INDO(J))=C(INDO(J-1))+W(NSTEP)*C(INDO(J))+C(INDO(J+1))
341 GO TO 121
342 D=C(INDO(IS+J-1))
343 D=C(INDO(IS+J-2))
344 L=0+1
345 C(INDO(J))=D-W(NSTEP)*C(INDO(J))+C(INDO(J+1))
346 CONTINUE
347 C INTERCHANGE THE COMPUTED R.H.S. VALUES VIA THE
348 C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
349 70 END
SUBROUTINE SYMCDPCO, JS, IE, INDEX)

* ODD-EVEN REDUCTION FOR THE SYMMETRIC
* CONSTANT-DIAGONAL PERIODIC CASE *

THIS SUBROUTINE SOLVES THE MATRIX EQUATION
C WHERE 'Q' IS A TRIDIGONAL PERIODIC MATRIX
C AND UNIT OFF-DIAGONAL ENTRIES.

C DIMENSION W(10), B(2048), C(2048), X(2048), INDE(2048)
C DIMENSION 0(2048), INDEX(2048), INDO(2048), ITIME(100)

C SPECIFICATION OF THE ADDITIONAL ARRAYS

G : IT HELDS THE VALUES OF EITHER OF 'B' OR 'C' ARRAY
DEPENDING ON THE STREAM OF THE CYCLIC ODD-EVEN REDUCTION
PROCEDURE CALLING THE SUBROUTINE.
INDEX : IT HELDS THE INDICES OF EITHER 'INDE' OR 'INDO' ARRAY
DEPENDING ON THE STREAM OF THE CYCLIC ODD-EVEN REDUCTION
PROCEDURE CALLING THE SUBROUTINE.

INTEGER LT, N, R, EX, TT, CW, JM, JNDO, ITIME, NSTEP, IJ,
- JQ, JA, ZZ, NIX, IJ, IE, ITZ, N, M = 2**N.

SET THE "DEPTH" OF RECURSION. WHERE 'N' IS THE EXPONENT OF THE
SIZE OF MATRIX 'Q', i.e. 1 < M < 2**N.

START THE REDUCTION PROCESS.
DO 1 LT = 1, ITZ
1 W(LT) = 2 - W(LT)*W(LT)
DO 2 J = 1, LT
2 CONTINUE

RETURN

THE SOLUTION PROCESS {Back-substitution}.
C THIS PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE
C TO SOLVE THE MATRIX EQUATION A*X=B, WHERE 'A' IS A TRIDIAGONAL
C PERIODIC MATRIX WITH CONSTANT-DIAGONAL AND UNIT OFF-DIAGONAL
C ENTRIES.
C THE PROGRAM PERFORMS (N-1) REDUCTION STEPS, WHERE 'N' IS THE EXPONENT
C OF THE SIZE OF THE MATRIX. IT GENERATES AS MANY PARALLEL PATHS AS
C IT CAN, i.e., IT IS INDEPENDENT OF THE NUMBER OF AVAILABLE PROCESSORS.
C BUT THERE IS AN IMPORTANT RESTRICTION: THIS IS THAT THE MAXIMUM
C NUMBER OF PARALLEL PATHS THAT CAN BE GENERATED ON THE 'NEPTUNE'
C 'MIMD' PROTOTYPE USING THE 'XPFLD' COMMAND IS 75.
C HOWEVER, IF AN ALTERNATE COMMAND IS UTILIZED, i.e., THE 'XPFLD' COMMAND,
C THEN AS MANY AS 32,767 PARALLEL PATHS CAN BE GENERATED.
C
C SET THE REQUIRED ARRAYS.
C DIMENSION W(12),B(2048),C(2048),X(2048),INDEX(2048),INDE(2048)
C DIMENSION ITIME(100)
C
C SPECIFICATION OF THE ARRAYS
C
C W : IT HOLDS THE MULTIPLIERS OF THE CONSTANT-DIAGONAL ENTRY
C OF MATRIX 'A' DURING THE ELIMINATION PROCESS.
C B : IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM. ON THIS ARRAY
C WILL BE APPLIED THE 'EVEN' STREAM OF THE CYCLIC ODD-EVEN
C REDUCTION PROCEDURE.
C C : IT HOLDS A COPY OF THE ARRAY 'B'. ON THIS ARRAY WILL BE
C APPLIED THE 'ODD' STREAM OF THE CYCLIC ODD-EVEN REDUCTION
C PROCEDURE.
C X : IT HOLDS THE SOLUTION OF THE SYSTEM.
C INDE : IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C INDO : IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C ITIME : IT HOLDS THE TIMING INFORMATION.
C
C INTEGER T,S
C C SET THE REQUIRED DATA,
C *SHARED N.R.I,II,B,C,X,W,INDEX,INDEX,NSTEP,INDE,INDE,ITIME,IM,IN,IK.
C - I,J,II,II
C C INITIALIZE PARALLELLISM.
C *USEPAR
C READ(5,9990) IPRT
C READ(5,9991) R,N
C WRITE(6,9992) R,N
C C THE SIZE OF MATRIX 'A'.
C IT=2*N
C
C DO 10 J=1,IT
C READ(5,9993) B(J)
C C MAKE A COPY OF THE R.H.S. ELEMENTS OF THE SYSTEM.
C C(INDE)=B(J)
C C INITIALIZE THE ARRAYS OF INDICES.
C C(INDE)=J
C C(INDE)=J
C 10 CONTINUE
C C SET THE "DEPTH" OF RECURSION UPON WHICH THE GRANULARITY
C FACTOR DEPENDS.
C IM=N-1
C C START TIMING THE COMPUTATIONAL PROCEDURE.
C C [I][I][I][I][I][I][I]
C C [I][I][I][I][I][I][I]
C 14 BODY 14
C CALL TIMES
C 14 *PAREND
C C C THE 'EVEN' COMPUTATIONAL STREAM.
C C CREATE AS MANY PARALLEL PATHS AS IT IS POSSIBLE
C C DEPENDING ON THE REDUCTION LEVEL EACH TIME.
C L=0
C 20 IF(I,IE,IM) GO TO 100
C C IS THE 'EVEN' COMPUTATIONAL STREAM.
C C[l]=l+1
C C[l]=l+1
C 30 CONTINUE
C C CHECK FOR THE PERIODICITY OF THE SYSTEM.
C IF(L,EQ.10) GO TO 60
C B(INDE(J))=B(INDE(J-1))+W(NSTEP)*B(INDE(J))
C GO TO 40
C 60 Z=B(INDE(J-2*L+1))
C L=0
C C THE 'ODD' STREAM OF THE SYSTEM.
C C[l]=l+1
C 40 CONTINUE
C 100 GO TO 30
C THIS IS THE 'ODD' COMPUTATIONAL STREAM.
IS=(I-1)*I+1
IE=I*I
D=C(INDO(I)*I))
DO 70 J=IS,IE,2
C CHECK FOR THE PERIODICITY OF THE SYSTEM.
IF(J.EQ.IS) GO TO 80
C(INDO(J))=C(INDO(J+1))-W(NSTEP)*C(INDO(J))*C(INDO(J+1))
GO TO 70
80 C(INDO(J))=D=W(NSTEP)*C(INDO(J))*C(INDO(J+1))
70 CONTINUE
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
30 SPAREND
C INTERCHANGE THE COMPUTED R.H.S. VALUES VIA THE APPROPRIATE
C ARRAYS OF INDICES.
DO 141 J=2,JT,2
C(INDO(J))=B(INDO(J))
B(INDO(J))=C(INDO(J+1))
141 CONTINUE
C SHUFFLE THE 'EVEN' AND 'ODD' USED INDICES TO THE TOP OF
C THE CORRESPONDING ARRAYS.
LS=0
DO 160 J=2,JT,2
LS=LS+1
LINDO<LSl=INDO(IS)
INDO<LSl=INDEX<LSl
160 CONTINUE
IF(NSTEP.EQ.IZ) GO TO 20
C COPY THE 'ODD' AND 'EVEN' USED INDICES AT THE REAR HALF
C OF EACH OTHER'S ARRAY.
LS=0
IM=1
DO 140 M=1,IT
LS=LS+1
INDEX(M)=INDEX(M+1)
INDEX(M+1)=INDEX(M)
140 CONTINUE
C THIS IS THE 'ODD' SOLUTION STREAM.
IS=2*(I-1)*I+1
IE=I*I
B(INDE(IE))=B(INDE(IE))+(W(N))2*B(INDE(IS))
X(INDE(IS))=B(INDE(IE))+2*X(INDE(IE))/W(N)
GO TO 130
C THIS IS THE 'ODD' SOLUTION STREAM.
IS=2*(I-1)*I+1
IE=I*I
C(INDO(IE))=C(INDO(IE))+(W(N))2*C(INDO(IS))
X(INDE(IS))=C(INDO(IS))/W(N)
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
30 SPAREND
TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
CALL TIMOUT(TIME)
SPAREND
OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL
PROCEDURE.
WRITE(6,9995) TIME
IF(IPRINT.EQ.O) GO TO 151
WRITE(6,9996) (X(I),I=1,IT)
C SET THE FORMATS
9990 FORMAT(I2)
9991 FORMAT(F4.2)
9992 FORMAT(F4.2,1X)
9993 FORMAT(F8.4)
9994 FORMAT(F8.4,1X)
9995 FORMAT(F8.4,1X,F9.4,1X)
9996 FORMAT(F8.4,1X,F9.4,1X)
200 C TERMINATE PROGRAM.
STOP
END
THE PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE.

1 C
2 C DIRECTORY.FILENAME: MB2_PEDPRP.C
3 C
4 C
5 C THIS PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE,
6 C TO SOLVE THE MATRIX EQUATION: AX=B, WHERE 'A' IS A TRIDIAGONAL
7 C PERIODIC MATRIX WITH CONSTANT-DIAGONAL AND UNIT OFF-DIAGONAL
8 C ENTRIES.
9 C THE NUMBER OF CREATED PATHS IS ALWAYS EQUAL TO THE NUMBER OF
10 C AVAILABLE PROCESSORS EACH TIME.
11 C THE PROGRAM PERFORMS (N-1) REDUCTION STEPS. WHERE 'N' IS THE EXPONENT
12 C OF THE SIZE OF THE MATRIX, USING THE 'IDOPAR' CONSTRUCT.
13 C
14 C
15 C SET THE REQUIRED ARRAYS.
16 C
17 C DIMENSION W(100), B(2048), C(2048), X(2048), INDE(2048)
18 C DIMENSION ITIME(100), IND(2048)
19 C
20 C SPECIFICATION OF THE ARRAYS
21 C ..............................................
22 C
23 C
24 C W: IT HOLDS THE MULTIPLIERS OF THE CONSTANT-DIAGONAL ENTRY
25 C OF MATRIX 'A' DURING THE ELIMINATION PROCESS.
26 C B: IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM. ON THIS ARRAY
27 C WILL BE APPLIED THE 'EVEN' STREAM OF THE CYCLIC ODD-EVEN
28 C REDUCTION PROCEDURE.
29 C C: IT HOLDS A COPY OF THE ARRAY 'B'. ON THIS ARRAY WILL BE
30 C APPLIED THE 'ODD' STREAM OF THE CYCLIC REDUCTION
31 C PROCEDURE.
32 C X: IT HOLDS THE SOLUTION OF THE SYSTEM.
33 C INDE: IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
34 C REDUCTION PROCEDURE.
35 C INDO: IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
36 C REDUCTION PROCEDURE.
37 C ITIME: IT HOLDS THE TIMING INFORMATION.
38 C
39 C INTEGER LS
40 C SET THE SHARED DATA.
41 C $SHARED N.R.X.I.T.C.W.INDE,INDO,ITIME,NSTEP,IJ.
42 C - 10,1A,IZ,NPROC,IB,TH.
43 C SET CRITICAL SECTIONS.
44 C REGION NOPAR
45 C INITIALIZE PARALLELISM.
46 C $USEPAR
47 C NPROC=0
48 C GENERATE PARALLEL PATHS TO SET DYNAMICALLY THE NUMBER OF
49 C PROCESSORS TO BE UTILIZED EACH TIME IN THE PROGRAM.
50 C $DOALL 2000
51 C ENTER CRITICAL SECTION.
52 C $ENTER NOPAR
53 C NPROC=NPROC+1
54 C RELEASE CRITICAL SECTION.

*EXIT NOPAR
C TERMINATE PARALLEL PATHS AND SYNCRONIZE PROCESSORS.
C
2000 $PEND
READ(5,9990) IPRINT
READ(5,9991) R,N
WRITE(6,9992) R,N
W(1)=R
C THE SIZE OF MATRIX 'A'.
IT=2**N
C READ THE R.H.S. ELEMENTS OF THE SYSTEM.
DO 20 J=1,IT
READ(5,9993) B(J)
C MAKE A COPY OF THE R.H.S. ELEMENTS OF THE SYSTEM.
C C(J)=B(J)
C INITIALIZE THE ARRAYS OF INDICES.
C INDE(J)=J
C INDO(J)=J
20 CONTINUE
C SET THE "DEPTH" OF RECURSION UPON WHICH THE GRANULARITY
C FACTOR DEPENDS.
C IZ=IZ-1
C WRITE(6,9994)
C C START TIMING THE COMPUTATIONAL PROCEDURE.
C C
60 C [I][I][I][I][I][I][I]
61 C [I][I][I][I][I][I][I]
62 C [I][I][I][I][I][I][I]
63 C [I][I][I][I][I][I][I]
64 CALL TIMEST
65 45 $PEND
C PROCEDURE.
66 C [I][I][I][I][I][I]
67 C [I][I][I][I][I][I][I]
68 C [I][I][I][I][I][I][I]
69 C [I][I][I][I][I][I][I]
70 C C START THE REDUCTION PROCESS.
71 C DO 60 NSTEP=1,IZ
72 C IN=2**N(NSTEP-1)
73 C IJ=IJ/IN
74 C W(NSTEP)=2-W(NSTEP)\#W(NSTEP)
75 C ID=2**NSTEP
76 C IQ=IQ/ID
77 C IF(NPROC.NE.1) GO TO 1002
78 C IA=1
79 C GO TO 1003
80 C IA=NPROC/2
81 C IB=IB/IA
82 C IF(NPROC.NE.4) GO TO 2000
83 C C GENERATE 'NOPAR' PARALLEL PATHS.
84 C $DOPAR 71 I=1,NPROC
85 C CREATE THE 'ODD' AND 'EVEN' COMPUTATIONAL STREAMS OF THE
86 C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
87 C C(J)=B(J)
88 C IF(I.LE.IA) GO TO 100
89 C THIS IS THE 'EVEN' COMPUTATIONAL STREAM.
109 IS = 2
110 IE = 1B
111 L = 0
112 DO 90 J = IS, IE, 2
113 L = L + 1
114 C CHECK FOR THE PERIODICITY OF THE SYSTEM.
115 IF (L.EQ.10) GO TO 110
116 B(INDE(J)) = B(INDE(J-1)) - W(NSTEP) * B(INDE(J)) + B(INDE(J+1))
117 GO TO 90
118 110 Z = B(INDE(J-2*L+1))
119 L = 0
120 B(INDE(J)) = B(INDE(J-1)) - W(NSTEP) * B(INDE(J)) + 2
121 90 CONTINUE
122 IF (NPROC.EQ.1) GO TO 100
123 GO TO 71
124 C THIS IS THE 'ODD' COMPUTATIONAL STREAM.
125 100 IS = 1
126 IE = 1B
127 L = 0
128 DO 120 J = IS, IE, 2
129 L = L + 1
130 C CHECKS FOR THE PERIODICITY OF THE SYSTEM.
131 IF (J.EQ.IS) GO TO 131
132 IF (J.EQ.IE) GO TO 130
133 C(INDO(J)) = C(INDO(J-1)) - W(NSTEP) * C(INDO(J)) + C(INDO(J+1))
134 GO TO 120
135 130 D = C(INDO(J+2*L-1))
136 L = 0
137 GO TO 132
138 D = C(INDO(IS+1-J-1))
139 132 L = 0
140 C(INDO(J)) = D - W(NSTEP) * C(INDO(J)) + C(INDO(J+1))
141 120 CONTINUE
142 C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
143 71 *PEND
144 GO TO 63
145 C
146 C IN CASE THAT ALL PROCESSORS ARE UTILIZED THEN THIS PART OF THE
147 C PROGRAM IS EXECUTED.
148 C
149 C GENERATE 'NPROC' PARALLEL PATHS.
150 3000 $DOPAR 70 =1,NPROC
151 C CREATE THE 'ODD' AND 'EVEN' COMPUTATIONAL STREAMS OF THE
152 C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
153 C
154 C THIS IS THE 'EVEN' COMPUTATIONAL STREAM.
155 IF (NSTEP.EQ.1) GO TO 117
156 K = 2
157 IS = (1-J+1)*IB+2
158 IE = IS+1B-2
159 GO TO 181
160 117 K = 4
161 IS = (1-J+2)*2
162 IE = IT
163 163 IF (I.EQ.4) GO TO 157
164 157 L = IQ/2
165 156 GO TO 182
166 181 L = 0
167 182 DO 91 J = IS, IE, K
168 91 GO TO 71
169 179 C THIS IS THE 'ODD' COMPUTATIONAL STREAM.
170 400 IF (NSTEP.EQ.1) GO TO 103
171 K = 2
172 IS = (1-J-1)*IB+1
173 IE = IS+1B-1
174 L = 0
175 IF (NSTEP.GT.2) L = IQ
176 GO TO 206
177 103 IS = (1-J+1)*IT
178 IE = IT
179 L = 0
180 206 DO 121 J = IS, IE, K
181 120 GO TO 63
182 181 C CHECKS FOR THE PERIODICITY OF THE SYSTEM.
183 IF (NPROC.EQ.1) GO TO 1021
184 L = L + 1
185 IF (I.EQ.(J+1)) GO TO 232
186 K = 4
187 1022 C(INDO(J)) = C(INDO(J+1)) - W(NSTEP) * C(INDO(J)) + C(INDO(J-1))
188 GO TO 121
189 200 D = C(INDO(IS+1-J-1))
190 GO TO 233
191 202 C(INDO(J)) = D - W(NSTEP) * C(INDO(J)) + C(INDO(J-1))
192 200 GO TO 121
193 121 CONTINUE
194 207 C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
195 70 *PEND
196 209 C INTERCHANGE THE COMPUTED R.H.S. VALUES VIA THE
197 C APPROPRIATE ARRAYS OF INDICES.
198 63 DO 140 J = 2, IT, 2
199 140 C(INDE(J)) = B(INDE(J))
200 140 CONTINUE
201 210 C SHUFFLE THE 'EVEN' AND 'ODD' USED INDICES TO THE TOP
202 210 C OF THE CORRESPONDING ARRAYS.
DO 169 J=2,IT/2
   LS=LS+1
   INDE(LS)=INDE(J)
   INDO(LS)=INDO(J-1)
   CONTINUE
   IF(NSTEP.EQ,10) GO TO 60
   WRITE(6,9995) ITIME
   CALL TIMOUT(ITIME)
   9995 IF(IPRINT.EQ,0) GO TO 370
   WRITE(6,9996) (X(J),J=1,IT)
   STOP
   370 STOP
   380 SET THE FORMATS
   9990 FORMAT(I2)
   9991 FORMAT(F4.2,2X,I2)
   9992 FORMAT(F8.4)
   9993 FORMAT(F8.4)
   9994 FORMAT(IX,'TIME("S")="'TIMEOCS)="'+(F8.2X)"
   9995 FORMAT(IX,'THE SOLUTION IS :'+(E16.2X))
   9996 FORMAT(/,'THE SOLUTION IS :'+(E16.2X))
   9997 C TERMINATE PROGRAM.
   370 380 STEPCOMPUTATIONALPROCEDURE.
   270 C
   271 C GENERATE 'NPROC' PARALLEL PATHS.
   272 C
   273 DO 270 I=1,NPROC
   274 C CREATE THE 'ODD' AND 'EVEN' SOLUTION STREAMS.
   275 IF(NPROC.NE.1) GO TO 291
   276 INDECJ+1J=(INDECJ+1J)*(-W(N))+2*X(INDECJ)
   277 X(INDECJ)=(INDECJ+1J)/W(N)
   278 CONTINUE
   279 IF(NPROC.EQ.1) GO TO 290
   280 C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
   281 C
   282 C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL
   283 C PROCEDURE.
   284 IF(IPRINT.EQ,0) GO TO 370
   285 WRITE(6,9996) (X(J),J=1,IT)
   286 CONTINUE
   287 C
   288 C THIS IS THE 'ODD' SOLUTION STREAM.
   289 C
   290 IS=1
   291 IF(I.LE.10) GO TO 290
   292 IF(NPROC.EQ.4) GO TO 281
   293 C THIS IS THE 'EVEN' SOLUTION STREAM.
   294 C
   295 J=IS+1
   296 IS=IS+1
   297 IF(IS.GT.IH) GO TO 290
   298 295 CONTINUE
   299 C

---

[App. C-V : 955]
C THIS PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE
C TO SOLVE THE MATRIX EQUATION \( \mathbf{A}\mathbf{x} = \mathbf{b} \), WHERE \( \mathbf{A} \) IS A TRIDIAGONAL
C PERIODIC MATRIX WITH CONSTANT-DIAGONAL AND UNIT OFF-DIAGONAL
C ENTRIES.
C THE PROGRAM GENERATES THE PARALLEL PATHS IN A SEMI-ASYNCHRONOUS
C MANNER DEPENDING ON THE NUMBER OF AVAILABLE PROCESSOR(S) EACH TIME.
C IN OTHER WORDS, THE PROGRAM INITIALLY GENERATES TWO PARALLEL PATHS
C AND THEN, EACH TIME, THE NEXT AVAILABLE PROCESSOR GENERATES THE NEXT
C PARALLEL PATH.
C SET THE REQUIRED ARRAYS.
C DIMENSION W(IJ), B(IJ), X(IJ), INDI(IJ), INDO(IJ)
C DIMENSION ITIME(100)
C SPECIFICATION OF THE ARRAYS
integral IT, LS
C SET THE SHARED DATA.
C SHARED N.R.IT.B.X.INDE.ITIME.NSTEP.IT.ID.IM.IK,
- ID.IP.ID.IP.ID.IP.ID.IP.
C SET CRITICAL SECTIONS.
C REGION PATH
C INITIALIZE PARALLELISM.
C USE PAR
C THE SIZE OF MATRIX 'A'.
integer IT=2*N
C READ THE R.H.S. ELEMENTS OF THE SYSTEM.
DO 20 J=1, IT
C MAKE A COPY OF THE R.H.S. ELEMENTS OF THE SYSTEM.
c(J)=b(J)
C INITIALIZE THE ARRAYS OF INDICES.
ind(J)=J
indo(J)=J
20 CONTINUE
C START TIMING THE COMPUTATIONAL PROCEDURE.
60 C START THE REDUCTION PROCESS.
DO 70 NSTEP=1, IT
L=L+1
W(NSTEP+1)=0.0
W(NSTEP+1)=2-W(NSTEP)*W(NSTEP)
IN=0
IM=1
ID=2+NSTEP
IN=2*(NSTEP-1)
IN=IT/IN
IN=IT/2
IN=IT/2
C GENERATE PARALLEL PATHS.
DOALL 70
C ENTER CRITICAL SECTION.
80 ENTER PATH
IN=IM
IM=IM+1
C RELEASE CRITICAL SECTION.
90 EXIT PATH
IF(I.LT.ID) GO TO 70
C CREATE THE 'ODD' AND 'EVEN' COMPUTATIONAL STREAMS OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
100 IF(L.EQ.ID) GO TO 100
101 C THIS IS THE 'EVEN' COMPUTATIONAL STREAM.
102 IS=2+J*(1-L-N1)
103 IE=J*(1-L-N1)
104 L=0
105 DO 90 J=1, IE, 2
106 L=L+1
107 C CHECK FOR THE PERIODICITY OF THE SYSTEM.
108 IF(L.EQ.ID) GO TO 110
109 WRITE(6,9994) J=1, IT
C CHECK FOR THE CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C CREATE A COPY OF THE 'ODD' STREAM OF THE CYCLIC
C REDUCTION PROCESS.
C START THE 'ODD' COMPUTATIONAL STREAM.
C CYCLIC ODD-EVEN REDUCTION, 'ODD' STREAM OF THE CYCLIC
C REDUCTION PROCESS.
C START THE 'ODD' COMPUTATIONAL STREAM.
C CYCLIC ODD-EVEN REDUCTION, 'ODD' STREAM OF THE CYCLIC
C REDUCTION PROCESS.
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
110 B(INDE(J)) = B(INDE(J-1)) - W(NSTEP) * B(INDE(J)) + B(INDE(J+1))
111 GO TO 90
112 Z = B(INDE(J-2*L+1))
113 L = 0
114 B(INDE(J)) = B(INDE(J-1)) - W(NSTEP) * B(INDE(J)) + Z
115 CONTINUE
116 GO TO 90
117 C THIS IS THE 'ODD' COMPUTATIONAL STREAM.
118 IS = (I-1) * I + 1
119 IE = I + 1
120 D = C(INDO(IJ1))
121 DO 120 J = IS, IE, 2
122 C CHECK FOR THE PERIODICITY OF THE SYSTEM.
123 IF(J.EQ.IS) GO TO 130
124 C(INDO(J)) = C(INDO(J-1)) - W(NSTEP) * C(INDO(J)) + C(INDO(J+1))
125 GO TO 120
126 CONTINUE
127 GO TO 90
128 C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
129 STOP *PAREN
130 C INTERCHANGE THE COMPUTED R.H.S. VALUES VIA THE APPROPRIATE
131 C ARRAYS OF INDICES AND SHUFFLE THE 'EVEN' AND 'ODD' USED
132 C INDICES TO THE TOP OF THE CORRESPONDING ARRAYS.
133 LS = 0
134 DO 140 J = 2, IT, 2
135 LS = LS + 1
136 C(INDE(J)) = B(INDE(J))
137 INDE(LS) = INDE(J)
138 CONTINUE
139 LT = 0
140 DO 160 J = 2, IT, 2
141 LT = LT + 1
142 B(INDO(J)) = C(INDO(J))
143 C(INDO(LT)) = INDO(J)
144 CONTINUE
145 IF(NSTEP.EQ.12) GO TO 60
146 C COPY THE 'ODD' AND 'EVEN' USED INDICES AT THE REAR HALF
147 C OF EACH OTHER'S ARRAY.
148 LT = 0
149 IMM = IT / 2 + 1
150 DO 170 MM = IW, IT
151 LT = LT + 1
152 INDE(M) = INDO(LT)
153 INDO(M) = INDE(LT)
154 CONTINUE
155 170 CONTINUE
156 60 CONTINUE
157 C THE SOLUTION PROCESS [Back-substitution].
158 W(N) = 4 - W(M) * W(N)
159 IP = 2 * (N-1)
160 IK = 1
161 C GENERATE PARALLEL PATHS.
162 STOP 260
163 C ENTER CRITICAL SECTION.
164 270 *ENTER PATH
165 IK = IK + 1
166 C RELEASE CRITICAL SECTION.
167 280 *EXIT PATH
168 IF(I.IE.IP) GO TO 260
169 C CREATE THE 'ODD' AND 'EVEN' SOLUTION STREAMS.
170 IF(I.IE.IU) GO TO 280
171 C THIS IS THE 'EVEN' SOLUTION STREAM.
172 IS = 2 * (I-1) * I + 1
173 IE = IS + 1
174 D = B(INDE(E))
175 B(INDE(E)) = B(INDE(E)) * (-W(N)) + 2 * B(INDE(IS))
176 X = B(INDE(E)) / W(N)
177 X = B(INDE(IS)) / W(N)
178 GO TO 270
179 C THIS IS THE 'ODD' SOLUTION STREAM.
180 280 IS = 2 * (I-1) * I + 1
181 IE = IS + 1
182 D = C(INDO(E))
183 C(INDO(E)) = C(INDO(E)) * (-W(N)) + 2 * C(INDO(IS))
184 X = C(INDO(E)) / W(N)
185 X = C(INDO(IS)) / W(N)
186 GO TO 270
187 C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
188 260 STOP *PAREN
189 C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
190 STOP *PAREN
191 C CALL TIMOUT(ITIME)
192 265 STOP *PAREN
193 C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL
194 C PROCEDURE.
195 WRITE(6,9995) ITIME
196 IF(IPRINT.EQ.0) GO TO 99990
197 WRITE(6,9996) (X(I), I = 1, IT)
198 STOP *PAREN
199 C SET THE FORMATS
200 WRITE(6,9990) FORMAT(12)
201 WRITE(6,9991) FORMAT(F4.2,2X)
202 WRITE(6,9992) FORMAT(F4.2,2X)
203 WRITE(6,9993) FORMAT(F8.4)
204 C
205 STOP 99990
206 C
207 STOP 9991
208 FORMAT(4X) FOR POESCSP; 1 , (B(I), I = 1, IT)
209 STOP 9992
210 FORMAT(F8.4)
211 STOP 9993
212 FORMAT(F8.4)
213 STOP 9994
214 FORMAT(1X, 'THE SOLUTION IS:', 7(E15.6,2X))
215 STOP 9995
216 STOP 9996
217 STOP 9997
218 STOP 9998
219 STOP 9999
220 STOP 99990
221 STOP 99991
222 STOP 99992
223 STOP 99993
224 STOP 99994
225 STOP 99995
226 STOP 99996
227 STOP 99997
228 STOP 99998
229 STOP 99999

C THIS PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE
7 TO SOLVE THE MATRIX EQUATION: \( \mathbf{A} \mathbf{x} = \mathbf{b} \), \( \mathbf{b} \) IS A GENERAL
8 TRIDIAGONAL PERIODIC MATRIX.
9 THE PROGRAM PERFORMS (N-1) REDUCTION STEPS, WHERE N IS THE EXPONENT
10 OF THE SIZE OF THE MATRIX. THE NUMBER OF CREATED PATHS IS ALWAYS
11 EQUAL TO THE NUMBER OF AVAILABLE PROCESSORS EACH TIME, WHILE ONE ONLY
12 ARRAY WITH INDICES IS UTILIZED FOR EACH COMPUTATIONAL STREAM.
13
14 SET THE REQUIRED ARRAYS.
15 DIMENSION EA(512), EC(512), INDEX(512)
16 DIMENSION OA(512), OB(512), OC(512), INDEX(512)
17 DIMENSION RHSE(512), RHSO(512)
18 DIMENSION WE(512), WD(512), X(512)
19 DIMENSION ITIME(100), A(512)
20 INTEGER A
21
22 SPECIFICATION OF THE ARRAYS
23 -----------------------------------------------
24 EA: IT HOLDS THE SUB-DIAGONAL ENTRIES OF MATRIX G.'
25 ED: IT HOLDS THE DIAGONAL ENTRIES OF MATRIX G.'
26 EC: IT HOLDS THE SUPER-DIAGONAL ENTRIES OF MATRIX G.'
27 RHSE: IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM.
28 On the above arrays will be applied the 'even' stream of the
29 cyclic odd-even reduction procedure.
30 INDEX: IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
31 ODD-EVEN REDUCTION PROCEDURE.
32 OA: IT HOLDS A COPY OF THE ARRAY EA',
33 OB: IT HOLDS A COPY OF THE ARRAY EB',
34 OC: IT HOLDS A COPY OF THE ARRAY EC',
35 RHSO: IT HOLDS A COPY OF THE ARRAY RHSO.
36 On the above arrays will be applied the 'odd' stream of the
37 cyclic odd-even reduction procedure.
38 INDEX: IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
39 ODD-EVEN REDUCTION PROCEDURE.
40 WE: IT HOLDS THE MULTIPLIERS FOR THE 'EVEN' STREAM OF THE CYCLIC
41 ODD-EVEN REDUCTION PROCEDURE.
42 WD: IT HOLDS THE MULTIPLIERS FOR THE 'ODD' STREAM OF THE CYCLIC
43 ODD-EVEN REDUCTION PROCEDURE.
44 X: IT HOLDS THE SOLUTION OF THE SYSTEM.
45 ITIME: IT HOLDS THE TIMING INFORMATION.
46 A: IT HOLDS THE COMPUTED 'RETURN' POINTS OF THE CYCLIC
47 ODD-EVEN REDUCTION PROCEDURE.
48
49 SET THE SHARED DATA.
50 $REGION NOPROC
51 CALL INIT
52 $REGION NOPROC
53 CALL TIME
54$IREGION NOPROC
55$IREGION NOPROC
56$IREGION NOPROC
57$IREGION NOPROC
58$IREGION NOPROC
59$IREGION NOPROC
60$IREGION NOPROC
61$IREGION NOPROC
62$IREGION NOPROC
63$IREGION NOPROC
64$IREGION NOPROC
65$IREGION NOPROC
66$IREGION NOPROC
67$IREGION NOPROC
68$IREGION NOPROC
69$IREGION NOPROC
70$IREGION NOPROC
71$IREGION NOPROC
72$IREGION NOPROC
73$IREGION NOPROC
74$IREGION NOPROC
75$IREGION NOPROC
76$IREGION NOPROC
77$IREGION NOPROC
78$IREGION NOPROC
79$IREGION NOPROC
80$IREGION NOPROC
81$IREGION NOPROC
82$IREGION NOPROC
83$IREGION NOPROC
84$IREGION NOPROC
85$IREGION NOPROC
86$IREGION NOPROC
87$IREGION NOPROC
88$IREGION NOPROC
89$IREGION NOPROC
90$IREGION NOPROC
91$IREGION NOPROC
92$IREGION NOPROC
93$IREGION NOPROC
94$IREGION NOPROC
95$IREGION NOPROC
96$IREGION NOPROC
97$IREGION NOPROC
98$IREGION NOPROC
99$IREGION NOPROC
100$IREGION NOPROC
101$IREGION NOPROC
102$IREGION NOPROC
103$IREGION NOPROC
104$IREGION NOPROC
105$IREGION NOPROC
106$IREGION NOPROC
107$IREGION NOPROC
108$IREGION NOPROC

$DOPAR 11000 I=1,NPROC

IS=(I-1)*IH+2
IE=IE+IH-2
IF(I.EQ.NPROC) IE=IT
DO 10000 JS=IS,IE,2
  10000 J=JS,I+1
  10100 IF(J.EQ.NPROC) IE=IT
  10200 DO 10500 J=JS,IE,2
  10300 JM=J+1
  10400 IF(J.EQ.I) GO TO 14000
  10500 Q1=OA(JM1)+2

C THE PERIODICITY OF THE SYSTEM.

C 7000
  7100 JJ=JP1E-2*10
  7200 WE(JM1)=EC(INDEX(J))+EB(INDEX(JM1))
  7300 WE(J)=EC(INDEX(J))+EB(INDEX(JM1))
  7400 Z5=WE(J)
  7500 Z5=WE(J)
  7600 ECM(INDEX(J))+2*EC(RHSE(INDEX(JM1)))+D
  7700 RHSE(INDEX(JM1))+2*RHSE(INDEX(JM1))
  7800 TO 10000

C THE LAST PHASE OF THE 'REFINEMENT' STEPS.

C 14000
  14100 LVAR=JM1+2*10
  14200 Q4=OA(LVAR)
  14300 WO(J)=OA(INDEX(J))/OB(INDEX(LVAR))
  14400 WO(JP1)=OB(INDEX(J))
  14500 WO(J)=OB(INDEX(J))/OB(INDEX(LVAR))
  14600 TO 17000

C THE COMPUTATION OF THE 'ODD' LINES OF THE SYSTEM.

C $DOPAR 19000 I=1,NPROC

IS=(I-1)*IH+1
IE=IE+IH-1
IF(I.EQ.NPROC) IE=IT
DO 19000 JS=IS,IE,2
  19100 JM=J+1

CONTINUE

$PAREND

CALL SOLVER

TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
CALL TIMOUT(ITIME)

30000 FORMAT(*PAREND)

C [III][III][III][III][III][III][III][III][III][III]

C PROCEDURE.

WRITE(6,40000) ITIME

40000 FORMAT('TIMELESS(*)=(#For POEGEN*)',/16.2X)

IF(IPRINT.EQ.0) GO TO 60000

WRITE(6,50000) (/X,'THE SOLUTION IS ',/4E15.6,2X/)

50000 FORMAT('TIMELESS(*)=(#For POEGEN*)',/16.2X)

C TERMINATE PROGRAM.

60000 FORMAT(*STOP)

C 

C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL

C PROCEDURE.

WRITE(6,40000) ITIME

40000 FORMAT('TIMELESS(*)=(#For POEGEN*)',/16.2X)

IF(IPRINT.EQ.0) GO TO 60000

WRITE(6,50000) (/X,'THE SOLUTION IS ',/4E15.6,2X/)

50000 FORMAT('TIMELESS(*)=(#For POEGEN*)',/16.2X)

C TERMINATE PROGRAM.

60000 FORMAT(*STOP)

C 

C THIS SUBROUTINE PERFORMS THE INTERCHANGING OF THE MODIFIED

C ELEMENTS AND THE SHUFFLING AND COPYING OF THE USED INDICES

C IN THE APPROPRIATE ARRAYS.

C 

C DIMENSION EA(512),EB(512),EC(512),INDEX(512)

C DIMENSION OA(512),OB(512),OC(512),INDEX(512)

C DIMENSION RHSE(512),RHSO(512)

C DIMENSION WE(512),WO(512),XI(512)

C DIMENSION TIME(100),A(512)

C 

C SUBROUTINE INITIAL

C 

C THIS SUBROUTINE PERFORMS INITIALIZATION.

C 

C DIMENSION EA(512),EB(512),EC(512),INDEX(512)

C DIMENSION OA(512),OB(512),OC(512),INDEX(512)

C DIMENSION RHSE(512),RHSO(512)

C DIMENSION WE(512),WO(512),XI(512)

C DIMENSION TIME(100),A(512)

C 

C INTEGER A

C 

C *SHARED EA,EB,EC,INDEX,OA,OB,OC,INDEX.

C 

C - RHSE,RHSE,WE,WO,XI,IT,IO,II,NPROC.

C 

C - ITIME,NSTEP,II,HI,IPRINT,A

C 

C IF(NSTEP.EQ.12) GO TO 20300

C 

C C SUBROUTINE SHUFFLE

C 

C C COPY THE COMPUTED ELEMENTS FROM THE 'EVEN' AND 'ODD' LINES

C C TO THE CORRESPONDING POSITIONS IN THE OPPOSITE ARRAYS.

C 

C DO 20100 J=2,IT,2

C OA(INDEX(J))=EA(INDEX(J))

C OB(INDEX(J))=EB(INDEX(J))

C OC(INDEX(J))=EC(INDEX(J))

C EA(INDEX(J-1))=OA(INDEX(J-1))

C EB(INDEX(J-1))=OB(INDEX(J-1))

C EC(INDEX(J-1))=OC(INDEX(J-1))

C RHSO(INDEX(J))=RHSE(INDEX(J))

C RHSEO(INDEX(J-1))=RHSO(INDEX(J-1))

C 20100 CONTINUE

C 

C C SHUFFLE THE 'EVEN' AND 'ODD' USED INDICES TO THE TOP OF

C C THE CORRESPONDING ARRAYS.

C 

C DO 20300 J=2,IT,2

C LS=LS+1

C INDEX(LS)=INDEX(J)

C INDEX(LS)=INDEX(J-1)

C 20300 CONTINUE

C 

C C COPY THE 'ODD' AND 'EVEN' USED INDICES AT THE REAR HALF

C C OF EACH OTHER'S ARRAY IN ORDER TO PROCEED AGAIN WITH THE

C C NEW ODD-EVEN CYCLE.
IF(NSTEP.EQ.IZ) GO TO 20600
LS=0
1W=IT/2+1
DO 20500 M=1W,IT
LS=LS+1
INDEXIM=INDEXILS
INDEXILS=INDEXIM
20:500
CONTINUE
20600 RETURN
$END

SUBROUTINE SOLVER
THIS SUBROUTINE PERFORMS THE SOLUTION PART.
DIMENSION EA(512),EB(512),EC(512),INDEX(512)
DIMENSION OA(512),OB(512),OC(512),INDEX(512)
DIMENSION RHSE(512),RHSO(512)
DIMENSION WE(512),WO(512),X(512)
DIMENSION ITIME(IOOI),A(512)
INTEGER A
SHARED EA,EB,EC,INDEX,OA,OB,OC,INDEX,
- RHSE,RHSO,WE,WO,X,IT,IQ,IZ,NPROC,
- ITIME,NSTEP,IJ,IPRINT,AJ2
REGION NOPROC
C THE NUMBER OF GROUPS, i.e., THE NUMBER OF PAIRS OF THE
C FINAL (2x2) SUBSYSTEMS TO SOLVE.
IPAIRS=IT/4
IT2=IT/2
C THE ROUNING PROCEDURE TO THE NEAREST (SMALLER OR GREATER)
INTEGER A,
IH=IPAIRS/NPROC
L=2#MOD(IPAIRS,NPROC)
IF(L<GE,NPROC) IH=IH+1
IH=2#IH
C
C THE COMPUTATION OF THE VALUES ON THE 'EVEN' CONSIDERED ARRAYS,
$DOPAR 21200 I=1,NPROC
IS=(I-1)lIH+1
IE=IS+1IH+1
IF(I.EQ,NPROC) IE=IT2
DO 21300 J=IS,IE,2
JP10=J+1
IY1=INDEX(IJP10)
Y2=OC(INDEX(J))
Y3=INDEX(J)
Y4=RHSO(INDEX(J))
SL=OC(INDEX(IJP10))/OB(IY1)
IF(ABS(SL).LT.0.1E-06) SL=0.0
SR=SL*Y2+OB(IY1)
X(JY3)=SR/SK
CONTINUE
21300 RETURN
$DOPAR 21400 I=1,NPROC
IS=(I-1)lIH+1
IE=IS+1IH+1
IF(I.EQ,NPROC) IE=IT2
DO 21500 J=IS,IE,2
JP10=J+1
IY1=INDEX(IJP10)
Y2=OC(INDEX(J))
Y3=INDEX(J)
Y4=RHSO(INDEX(J))
SL=OC(INDEX(IJP10))/OB(IY3)
IF(ABS(SL).LT.0.1E-06) SL=0.0
SR=SL*Y2+OB(IY3)
X(JY3)=SR/SK
CONTINUE
21500 RETURN
$END
C THIS PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE
C TO SOLVE THE MATRIX EQUATION \( A x = b \), WHERE \( A \) IS A GENERAL
C TRIDIAGONAL PERIODIC MATRIX.
C THE NUMBER OF CREATED PATHS IS ALWAYS EQUAL TO THE NUMBER OF
C AVAILABLE PROCESSORS EACH TIME.
C THEN, AFTER A PRE-SET NUMBER OF REDUCTION STEPS, WE CONTINUE WITH
C APPLYING THE ODD-EVEN REDUCTION TECHNIQUE AGAIN BUT SEQUENTIALLY
C TILL THE RESULTING SUBSYSTEMS IN EACH OF THE CREATED
C PARALLEL PATHS, SIMULTANEOUSLY, FOR EACH COMPUTATIONAL STREAM ONLY
C ONE ARRAY WITH INDICES IS UTILIZED.
C
C SET THE REQUIRED ARRAYS.
19   DIMENSION EA(256), EB(256), EC(256), INDEX(256)
20   DIMENSION RHSE(256), RHSO(256)
21   DIMENSION WE(256), WO(256), X(256)
22   DIMENSION ITIME(100), A(256)
23   INTEGER A
24
C SPECIFICATION OF THE ARRAYS
25
26 C COMPUTATION OF
27 C *****************************************
28 C   EA : IT HOLDS THE SUB-DIAGONAL ENTRIES OF MATRIX 'G'.
29 C   EB : IT HOLDS THE DIAGONAL ENTRIES OF MATRIX 'G'.
30 C   EC : IT HOLDS THE SUPER-DIAGONAL ENTRIES OF MATRIX 'G'.
31 C   RHSE : IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM.
32 C   Rhs: THE above arrays will be applied the 'e'en' stream of the
33 C   CYCLIC ODD-EVEN REDUCTION PROCEDURE.
34 C   INDEX : IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
35 C   ODD-EVEN REDUCTION PROCEDURE.
36 C   QA : IT HOLDS A COPY OF THE ARRAY 'EA'.
37 C   OB : IT HOLDS A COPY OF THE ARRAY 'EB'.
38 C   QC : IT HOLDS A COPY OF THE ARRAY 'EC'.
39 C   RH: IT HOLDS A COPY OF THE ARRAY 'RHSE'.
40 C On the above arrays will be applied the 'oDD' stream of the
41 C   CYCLIC ODD-EVEN REDUCTION PROCEDURE.
42 C   INDEX : IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
43 C   ODD-EVEN REDUCTION PROCEDURE.
44 C   WE : IT HOLDS THE MULTIPLIERS FOR THE 'EVEN' STREAM OF THE
45 C   CYCLIC ODD-EVEN REDUCTION PROCEDURE.
46 C   WO : IT HOLDS THE MULTIPLIERS FOR THE 'ODD' STREAM OF THE CYCLIC
47 C   ODD-EVEN REDUCTION PROCEDURE.
48 C   X : IT HOLDS THE SOLUTION OF THE SYSTEM.
49 C   ITIME : IT HOLDS THE TIMING INFORMATION.
50 C   A : IT HOLDS THE COMPUTED 'RETURN' POINTS OF THE CYCLIC
51 C   ODD-EVEN REDUCTION PROCEDURE.
52 C
53 C SET THE SHARED DATA.
54 C
55 COMMON/BL2/IFLAG
56 COMMON/RHSO,NPRO,WE,MO,X,IT,NP,LT,II,IT,F,
57 COMMON/IH,JL,H1,IPRINT,A,IN,LPNT,E,KK,ICOUNT
109 GO TO 3500
110 C STORE THE COMPUTED 'RETURN' POINTS EACH TIME.
111 2700 DO 3000 K=1,IN
112 LPNTE=K*IJ
113 LPNTO=LPNTE-IJ+I
114 A(LPNTE)=LPNTE
115 A(LPNTO)=LPNTO
116 3000 CONTINUE
117 C
118 C THE COMPUTATION OF THE 'EVEN' LINES OF THE SYSTEM.
119 C
120 3500 $DOPAR 11000 I=1,NPROC
121 IF(NSTEP.EQ.12) GO TO 10100
122 IS=(I-1)*IH+2
123 IE=IS+IH-2
124 IF(IS.EQ.II) IE=IT
125 DO 10000 IS,J=IE,2
126 JM10=J-1
127 Z2=EA(LPINDOX(JM10))
128 IF(J.EQ.A(J)) GO TO 14000
129 OJ=OACINDOX(JM10)
130 WO(J)=-OACINDOX(JM10)/OBCINDOX(JM10)
131 Q5=WO(J)
132 WE(J)=EA(LPINDOX(J))/EB(LPINDOX(J))
133 Z5=WE(J)
134 EB(LPINDOX(J))=Z5*EC(LPINDOX(J))+EB(LPINDOX(J))
135 EC(LPINDOX(J))=Z5*EC(LPINDOX(J))+EB(LPINDOX(J))
136 RC(LPINDOX(J))=Z5*RHSE(LPINDOX(J))+EB(LPINDOX(J))
137 RC(LPINDOX(J))=Z5*RHSE(LPINDOX(J))+EB(LPINDOX(J))
138 TO 10000
139 C THE PERIODICITY OF THE SYSTEM.
140 7000
141 7000 J=J+1
142 WE(JM1E)=EC(LPINDOX(J))/EB(LPINDOX(J))
143 Z4=WE(J)
144 WE(J)=EA(LPINDOX(J))/EB(LPINDOX(J))
145 Z5=WE(J)
146 10000 CONTINUE
147 C THE LAST PHASE OF THE 'REFINEMENT' STEPS.
148 EAAO(LVAR)=Z5*Z4
149 2000 LVAR=JM+2*J
150 IF(J.EQ.A(J)) GO TO 14000
151 OJ=OACINDOX(JM10)
152 WO(J)=-OACINDOX(JM10)/OBCINDOX(JM10)
153 Q5=WO(J)
154 WE(J)=EA(LPINDOX(J))/EB(LPINDOX(J))
155 Z5=WE(J)
156 EB(LPINDOX(J))=Z5*EC(LPINDOX(J))
157 EC(LPINDOX(J))=Z5*EC(LPINDOX(J))
158 RC(LPINDOX(J))=Z5*RHSE(LPINDOX(J))
159 RC(LPINDOX(J))=Z5*RHSE(LPINDOX(J))
160 TO 10000
161 C THE LAST PHASE OF THE 'REFINEMENT' STEPS.
162 10100 IFLAG=0
163 IF(KK,NE.0) GO TO 10200
164 IF(J.EQ.A(J)) GO TO 10450
165 IHL=IH
166 ICOUNT=IHL
167 IF(I.EQ.NPROC) IHL=IH+KK
168 GO TO 10500
169 IHL=IH+1
170 ICOUNT=IHL
171 IF(I.EQ.NPROC) IHL=KK
172 DO 10700 J=1,NPROC
173 IS=(J-1)*IJ+1
174 IE=IS+IJ-1
175 CALL GENP(IS,IE)
176 10700 CONTINUE
177 11000 $DOPAR 19000 I=1,NPROC
178 IF(NSTEP.EQ.12) GO TO 18100
179 IS=(I-1)*IH+1
180 IE=IS+IH-1
181 IHL=IH+KK
182 DO 18000 J=IS,IE,2
183 JM10=J-1
184 IF(J.EQ.A(J)) GO TO 13000
185 OJ=OACINDOX(JM10)
186 WO(J)=-OACINDOX(JM10)/OBCINDOX(JM10)
187 Q5=WO(J)
188 WE(J)=EA(LPINDOX(J))/EB(LPINDOX(J))
189 Z5=WE(J)
190 EB(LPINDOX(J))=Z5*EC(LPINDOX(J))
191 EC(LPINDOX(J))=Z5*EC(LPINDOX(J))
192 RHSE(LPINDOX(J))=Z5*RHSE(LPINDOX(J))
193 RHSE(LPINDOX(J))=Z5*RHSE(LPINDOX(J))
194 TO 10700
195 C THE PERIODICITY OF THE SYSTEM.
196 14000 LVAR=JM+2*J
197 OJ=OACINDOX(JM10)
198 WO(J)=-OACINDOX(JM10)/OBCINDOX(JM10)
199 Q5=WO(J)
200 RHRSE(LPINDOX(JM10))=Q5*RHRSE(LPINDOX(JM10))
201 TO 10000
202 C THE PERIODICITY OF THE SYSTEM.
203 C THE LAST PHASE OF THE 'REFINEMENT' STEPS.
204 10000 CONTINUE
205 LVAR=JM10+2*J
206 OJ=OACINDOX(JM10)
207 WO(J)=-OACINDOX(JM10)/OBCINDOX(JM10)
208 Q5=WO(J)
209 RHRSE(LPINDOX(JM10))=Q5*RHRSE(LPINDOX(JM10))
210 TO 10000
211 C THE LAST PHASE OF THE 'REFINEMENT' STEPS.
CONTINUE
GO TO 19000
IF(FLAG.GT.0) GO TO 18200
IHL=IHL+1
ICOUNT=IHL
IF(I.EQ.NPROC) THEN
IHL=IHL+KK
GO TO 18500
END IF
DO 18700 JK=I,IHL
IS=CJK-1)*1J+1+1COUNT*IJ+CI-1)
IE=IS+IJ-1
CALL GENP(IS,IE)
CONTINUE
SPAREND
IF(INSTEP.EQ.IZ) GO TO 21000
CALL SHUFI
CONTINUE
C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
WRITE((,1100)
FORMAT(2,,X))
READ(5,1150)
CONTINUE
DO 1150 NSIZE=I,IT
ACNSIZE.GE.O CONTINUE
RETURN
SEND
DO 1300 J=LOI,IT
READ(5,1200) EAI(J),EB(J),EC(J)
FORMAT(31F4.2,2X))
OAI(J)=EAI(J)
OBC(J)=EB(J)
OCI(J)=EC(J)
INDEX(J)=J
CONTINUE
DO 1500 J=I,IT
READ(5,J400) RHSE(J)
FORMAT(6F6.2)
RHSO(J)=RHSE(J)
CONTINUE
DO 1300 J=I,IT
READ(5,1400) RHSE(J)
FORMAT(6F6.2)
CONTINUE
RETURN
END

SUBROUTINE INITIAl
DIMENSION EA(256),EB(256),EC(256),INDEX(256)
DIMENSION QA(256),QB(256),QC(256),INDEX(256)
DIMENSION RHSE(256),RHSO(256)
DIMENSION WE(256),WO(256),X(256)
DIMENSION ITIME(100),A(256)
INTEGER A,QA,QB,QC,JNDEX,QA,OB,OC,JNDOX,
- RHSE,RHSO,N,WE,WO,X,IT,IO,IZ,NPROC,
- INDEX,NSTEP,IT,II,IPRINT,A.IN.LPNT,AK,
- READ(5,1100) IPRINT,N
WRITE((,1100)(IPRINT,N)
FORMAT(2,,X))
READ(5,1120) IT
WRITE((,1120)(IT)
CONTINUE
DO 1300 J=1,IT
READ(5,J400) RHSE(J)
CONTINUE
DIMENSION EAC256),EBC256),EC(256)
DIMENSION OAI256),OBI256),OCI256),INDOX(256)
DIMENSION WE1256),WO1256),X(256)
DIMENSION ITIME(100),A(256)
INTEGER A,QA,QB,OC,JNDEX,QA,OB,OC,JNDOX,
- RHSE,RHSO,N,WE,WO,X,IT,IO,IZ,NPROC,
- INDEX,NSTEP,IT,II,IPRINT,A.IN.LPNT,AK,
- READ(5,1100) IPRINT,N
WRITE(6,1100) IPRINT,N
CONTINUE
DO 1150 NSIZE=I,IT
ACNSIZE.GE.O CONTINUE
RETURN
SEND
DO 1300 J=1,IT
READ(5,1200) EA(J),EB(J),EC(J)
CONTINUE
DO 1300 J=1,IT
READ(5,1400) RHSE(J)
CONTINUE
RETURN
END

SUBROUTINE SHUFI
THIS SUBROUTINE PERFORMS THE INTERCHANGING OF ELEMENTS AND THE SHUFFLING AND COPYING OF THE MODIFIED USED INDICES IN THE APPROPRIATE ARRAYS.
DIMENSION EA(256),EB(256),EC(256),INDEX(256)
DIMENSION QA(256),QB(256),QC(256),INDEX(256)
DIMENSION RHSE(256),RHSO(256)
DIMENSION WE(256),WO(256),X(256)
DIMENSION ITIME(100),A(256)
INTEGER A,QA,QB,OC,JNDEX,QA,OB,OC,JNDOX,
- RHSE,RHSO,N,WE,WO,X,IT,IO,IZ,NPROC,
- INDEX,NSTEP,IT,II,IPRINT,A.IN.LPNT,AK,
- READ(5,1100) IPRINT,N
WRITE(6,1100) IPRINT,N
CONTINUE
DO 1150 NSIZE=I,IT
ACNSIZE.GE.O CONTINUE
RETURN
SEND
DO 1300 J=1,IT
READ(5,1200) EA(J),EB(J),EC(J)
CONTINUE
DO 1300 J=1,IT
READ(5,1400) RHSE(J)
CONTINUE
RETURN
END

APPENDIX C: V. 96
INTEGER A
A*SHARED EA,EB,EC,INDEX,OA,OB,OC,INDOX,
A*RHSE,RHSO,N,WE,WO,X,JT,IQ,
A*PROC
A*ITIME,NSTEP,JJ,JH,IPRINT,A,IN,LPNTE,KK
A*IF(NSTEP.EQ.(12-1)) GO TO 74300
330 C
331 C COPY THE COMPUTED ELEMENTS FROM THE 'EVEN' AND 'ODD' LINES
332 C TO THE CORRESPONDING POSITIONS IN THE OPPOSITE ARRAYS.
333 C
334 C DO 74100 J=2,IT,2
335 OA(INDEX(J))=EA(INDEX(J))
336 OB(INDEX(J))=EB(INDEX(J))
337 OC(INDEX(J))=EC(INDEX(J))
338 EA(INDEX(J-I))=OA(INDEX(J-I))
339 EB(INDEX(J-I))=OB(INDEX(J-I))
340 EC(INDEX(J-I))=OC(INDEX(J-I))
341 RHSO(INDEX(J))=RHSO(INDEX(J-I))
342 RHSE(INDEX(J-I))=RHSE(INDEX(J))
343 CONTINUE
344 C
345 C SHUFFLE THE 'EVEN' AND 'ODD' USED INDICES TO THE TOP
346 C OF THE CORRESPONDING ARRAYS.
347 C
348 74300 LS=0
349 DO 74200 J=2,IT,2
350 LS=LS+1
351 INDEX(LS)=INDEX(J)
352 INDEXO(LS)=INDEXO(J-I)
353 CONTINUE
354 C
355 C COPY THE 'ODD' AND 'EVEN' USED INDICES AT THE REAR HALF
356 C OF EACH OTHER'S ARRAY IN ORDER TO PROCEED AGAIN WITH THE
357 C NEW ODD-EVEN CYCLE.
358 C
359 C IF(NSTEP.EQ.(12-1)) GO TO 74600
360 LS=0
361 JM=IT/2+1
362 DO 74500 M=JM,IT
363 LS=LS+1
364 INDEXO(M)=INDOX(LS)
365 INDUX(M)=INDUX(LS)
366 CONTINUE
367 74500 RETURN
368 C
369 C *END
370 C
371 C *********************************************************
372 C * Phase Three *
373 C *********************************************************
374 C
375 C SUBROUTINE GENP(IS,IE)
376 C
377 C THIS SUBROUTINE SOLVES THE MATRIX EQUATION : QX=Z, WHERE 'Q' IS
378 C A GENERAL TRIDIAGONAL PERIODIC MATRIX, SEQUENTIALLY, BY APPLYING
379 C THE ODD-EVEN REDUCTION TECHNIQUE AGAIN.
380 C THE ADDITIONALLY DECLARED ARRAYS FOR THE WORKSPACE IN THE
381 C SUBROUTINE ACCOMMODATING EACH TIME THE CORRESPONDING MODIFIED
382 C ELEMENTS OF THE SYSTEM. IN ACCORDANCE WITH THE COMPUTATIONAL
383 C STREAM THE SUBROUTINE IS CALLED FROM.
384 C
385 C DIMENSION EA(256),EB(256),EC(256),INDEX(256)
386 C DIMENSION OA(256),OB(256),OC(256),INDOX(256)
387 C DIMENSION RHSE(256),RHSO(256)
388 C DIMENSION WE(256),WO(256),X(256)
389 C DIMENSION EAG(256),EBG(256),ECG(256),INDEXG(32)
390 C DIMENSION OA(256),OB(256),OCG(256),INDOX(32)
391 C DIMENSION RHSEG(256),RHSOG(256)
392 C DIMENSION WE(32),WO(32)
393 C DIMENSION ITIME(100),A(256),AG(32)
394 C INTEGER A
395 C *SHARED EA,EB,EC,INDEX,OA,OB,OC,INDOX,
396 C RHSE,RHSO,N,WE,WO,X,JT,IQ,
397 C ITIME,NSTEP,JJ,JH,IPRINT,A,IN,LPNTE,KK
398 C COMMON/BL1/EAG,EBG,ECG,INDEXG,OA,G,OBG,OCG,INDOXG,
399 C RHSEG,RHSG0,EMG,MG,AG,ISTEP,NSTEPG
400 C COMMON/BL2/IFLAG
401 C CALL INITO(IS,IE)
402 C DO 76000 NSTEPO=ISTEP
403 C IND=2**NSTEPO-1
404 C IJG=IJ/IND
405 C IDO=2**NSTEPO
406 C COMPUTATION OF THE INCREMENT.
407 C IJG=IJ/IND
408 C STORE THE COMPUTED 'RETURN' POINTS EACH TIME.
409 C DO 61000 K=1,IND
410 C LPNTEG=K*IJG
411 C LPNTDG=LPNTEG-INDO-1
412 C AG(LPNTEG)=LPNTDG
413 C AG(LPNTDG)=LPNTDG
414 61000 CONTINUE
415 C
416 C THE COMPUTATION OF THE 'EVEN' LINES OF THE SYSTEM.
417 C
418 C DO 67000 J=2,IT,2
419 C JMIE=J-1
420 C JPIE=J+1
421 C Z=EAG(INDEXG(JMIE))
422 C IF(J.IEQ.JPIE) GO TO 64000
423 C WEG(JMIE)=EAG(INDEXG(JMIE))
424 C Z=WEG(JMIE)
425 C WEG(J)=EAG(INDEXG(J))
426 C Z=WEG(J)
427 C EBG(INDEXG(J))=Z2*ECG(INDEXG(J))+EAG(INDEXG(J))+Z*EAG(INDEXG(J))
428 C IF(IJG.EQ.4) GO TO 62000
429 C EAG(INDEXG(J))=Z2*Z
430 C D=0.0
431 C GO TO 63000
432

[App. C-IV: 965]
D=Z2*Z1

ECG(INDEX(J))=Z3*ECG(INDEX(JP1))+D

RHSOG(INDEX(J))=Z2*RHSOG(INDEX(JM1))+

RHSOG(INDEX(J))=Z3*RHSOG(INDEX(JP1))

GO TO 67000

C THE PERIODICITY OF THE SYSTEM.

C THE LAST PHASE OF THE 'REFINEMENT' STEPS.

C THE COMPUTATION OF THE 'ODD' LINES OF THE SYSTEM.

DO 74000 J=1,IJ+2

JM1=J-1

JP1=J+1

IF(J,EQ,AO(J)) GO TO 71000

Q1=AO(J)(INDEX(JM1))

WOG(J)=AO(J)(INDEX(J))/OBG(INDEX(JM1))

Q2=WOG(J)

WOG(JP1)=OBG(INDEX(J))/OBG(INDEX(JP1))

Q3=WOG(JP1)

IF(J,EQ.4) GO TO 69000

D=0.0

OAG(INDEX(J))=Q2*Q1

GO TO 70000

D=Q2*Q1

Q3=OAG(INDEX(JM1))

Q4=OAG(INDEX(J))/OBG(INDEX(JM1))

Q5=WOG(J)

WOG(JP1)=OAG(INDEX(J))/OBG(INDEX(JP1))

Q6=WOG(JP1)

Q7=OAG(INDEX(J))/OAG(INDEX(JP1))

Q8=WOG(INDEX(JM1))

Q9=ROGS0(INDEX(J))/OAG(INDEX(JM1))

Q10=ROGS0(INDEX(J))/ROGS0(INDEX(JP1))

GO TO 74000

C THE PERIODICITY OF THE SYSTEM.

LVAR=JM1*Z1

OAG(INDEX(LVAR))

WOG(J)=AO(LVAR)/OBG(LVAR)

C THE LAST PHASE OF THE 'REFINEMENT' STEPS.

IF(IJ,EQ.4) GO TO 72000

Q5=WOG(J)

WOG(JP1)=OAG(INDEX(J))/OBG(INDEX(JP1))

Q6=WOG(JP1)

D=0.0

GO TO 73000

D=Q5*Q4

GO TO 72000

CALL SHUF2

CONTINUE

C *Solution Part*

C *Phase Four*

SUBROUTINE INIT0(IN1,NE)

***************

DO 72000

DIMENSION EA(256),EB(256),EC(256),INDEX(256)

DIMENSION OA(256),OB(256),OC(256),INDEX(256)

DIMENSION RHS(256),RHSO(256)

DIMENSION WE(256),WO(256),X(256)

DIMENSION EAO(256),EBO(256),ECO(256)

DIMENSION OAG(256),OBG(256),OCG(256)

Dimensions RHO(256),RHSO(256)

DIMENSION ITIME(100),AA(256),AG(32)

INTEGER AAG

$SHARED EA,EB,EC,INDEX,OA,OB,OC,INDEX,

- RHS,RHSO,N,WE,WO,X,IT,IG,IZ,NP,CC

- ITIME,STEP,IG,IM,PRINT,AA,LP,NT,EE

COMMON/BL1/EA,EB,EC,INDEX,OA,OB,OC,INDEX,

- RHS,RHSO,WE,WO,AG,ISTEP,STEP

COMMON/BL2/FLAG

DO 133 I=1,11

AG(I)=0

CONTINUE

ISTEP=N-1Z
541 IF(IFLAG.EQ.1) GO TO 10700
542 L=0
543 DO 10600 J=IS,IE
544 INDEX(L)=INDEX(J)
545 INDEXG(L)=INDEXG(J)
546 EAG(INDEX(L))=EAG(INDEX(J))
547 OAG(INDEX(L))=OAG(INDEX(J))
548 EBO(INDEX(L))=EBO(INDEX(J))
549 OBO(INDEX(L))=OBO(INDEX(J))
550 ECO(INDEX(L))=ECO(INDEX(J))
551 OCO(INDEX(L))=OCO(INDEX(J))
552 RHSEG(INDEX(L))=RHSEG(INDEX(J))
553 RHSO(INDEX(L))=RHSO(INDEX(J))
554 CONTINUE
555 10600 GO TO 10900
556 10700 L=0
557 DO 10900 J=IS,IE
558 INDEXG(J)=INDEXG(J)
559 INDEXO(J)=INDEXO(J)
560 EAO(INDEXO(J))=EAO(INDEXG(J))
561 OAO(INDEXO(J))=OAO(INDEXG(J))
562 EBO(INDEXO(J))=EBO(INDEXG(J))
563 OBO(INDEXO(J))=OBO(INDEXG(J))
564 ECO(INDEXO(J))=ECO(INDEXG(J))
565 OCO(INDEXO(J))=OCO(INDEXG(J))
566 RHSEG(INDEXO(J))=RHSEG(INDEXG(J))
567 RHSO(INDEXO(J))=RHSO(INDEXG(J))
568 CONTINUE
569 10800 RETURN
570 10900 RETURN
571 RETURN
572 *END
573 C
574 C ***************
575 C Phase Five
576 C ***************
577 C
578 C SUBROUTINE SHUF2
579 C
580 C THIS SUBROUTINE PERFORMS THE INTERCHANGING OF THE MODIFIED ELEMENTS AND THE SHUFFLING AND COPYING OF THE USED INDICES IN THE APPROPRIATE ARRAYS.
581 C
582 C DIMENSION EA(256),EB(256),EC(256),INDEX(256)
583 C DIMENSION OA(256),OB(256),OC(256),INDO(256)
584 C DIMENSION RHSE(256),RHSO(256)
585 C DIMENSION WE(256),W0(256),X(256)
586 C DIMENSION EAG(256),EBO(256),ECO(256),INDEX(32)
587 C DIMENSION OAG(256),OBO(256),OCO(256),INDO(32)
588 C DIMENSION RHSEG(256),RHSO(256)
589 C DIMENSION RHSEG(32),RHSO(32)
590 C DIMENSION ITIME(100),A(256),AO(32)
591 C INTEGER A
592 C $SHARED EA,EB,EC,INDEX,OA,OB,OC,INDO,
DIMENSION OA(256), OB(256), OC(256), INDOX(256)
DIMENSION RHSE(256), RHSO(256)
DIMENSION WE(256), W0(256), X(256)
DIMENSION EAG(256), EBO(256), ECG(256), INDEXG(32)
DIMENSION OA(256), OB(256), OC(256), INDOX(32)
DIMENSION RHSEG(256), RHSOG(256)
DIMENSION WEO(32), WOG(32)
DIMENSION ITIME(100), A(256), AG(32)
DIMENSION RHSEC256, RHSOC256
DIMENSION WEC256, W0C256, XJC256)
DIMENSION EAGC256, EBOC256, ECGC256, INDEXGC32
DIMENSION OAG(256), OBO(256), OCG(256), INDOXG(32)
DIMENSION RHSEG(256), RHSOG(256)
DIMENSION WEOC32, WOGC32
DIMENSION ITIMEC100, AC256, AGC32)

INTEGER A

SSHARE OA, EBO, ECG, INDEX, OA, OB, OC, INDOX,
- RHSE, RHSO, WE, W0, X, IT, II, NPROC,
- TIME, NSTEP, JJ, IH, IPRINT, A, IN, LPRINT, KK
- COMMON/BLI/EAG, EBO, ECG, INDEX, OA, OB, OC, INDOX,
- RHSEG, RHSOG, WEO, WOG, AG, ISTEP

iJ2=IJ/2

C THE COMPUTATION OF THE VALUES ON THE 'EVEN' CONSIDERED ARRAYS.

DO 76100 J=1, IJ2, 2
    JPIE=J+1
    IX1=INDEXG(JPIE)
    X2=ECG(IX1)
    IX3=INDEXG(J)
    X4=RHSEG(IX3)
    SL=ECG(IX1)/EBO(IX3)
    IF(ABS(SL).LT.(0.1E-06)) SL=0.0
    SK=SL*X2+EOB(IX1)
    SR=SL*X4+RHSEG(IX3)
    XI(IX1)=SR/SK
    X(IX3)=(X4-X2*X(IX1))/EOB(IX3)
CONTINUE

C THE COMPUTATION OF THE VALUES ON THE 'ODD' CONSIDERED ARRAYS.

DO 76200 J=1, IJ2, 2
    JPIO=J+1
    IY1=INDEXO(JPIO)
    Y2=OCG(IY1)
    IY3=INDEXO(J)
    Y4=RHSOG(IY3)
    SL=OCG(IY1)/EBO(IY3)
    IF(ABS(SL).LT.(0.1E-06)) SL=0.0
    SK=SL*Y2+OBG(IY1)
    SR=SL*Y4+RHSOG(IY3)
    XIY1)=SR/SK
    X(IY3)=(Y4-Y2*X(IY1))/OBG(IY3)
CONTINUE

RETURN

END
C CYCLIC ODD-EVEN REDUCTION TECHNIQUE
C TO SOLVE THE MATRIX EQUATION: Dx=B, WHERE 'G' IS A GENERAL
C TRIDIAGONAL PERIODIC MATRIX.
C THE NUMBER OF CREATED PATHS IS ALWAYS EQUAL TO THE NUMBER OF
C AVAILABLE PROCESSORS EACH TIME.
C THEN, AFTER A PRE-SET NUMBER OF REDUCTION STEPS, WE CONTINUE BY
C APPLYING THE GAUSS ELIMINATION SEQUENTIAL PROCEDURE TO SOLVE THE
C RESULTING SUBSYSTEMS IN EACH OF THE CREATED PARALLEL PATHS.
C SIMULTANEOUSLY, FOR EACH COMPUTATIONAL STREAM ONLY ONE ARRAY WITH
C INDICES IS UTILIZED.
C
C SET THE REQUIRED ARRAYS.
C
DIMENSION EA(256), EB(256), EC(256), INDEX(256)
DIMENSION OA(256), OB(256), OC(256), INDX(256)
DIMENSION RHS(256), RSH(256)
DIMENSION WE(256), WO(256), X(256)
DIMENSION ITIME(100), AT(256)
INTEGER A
C
C SPECIFICATION OF THE ARRAYS
C
C EA: IT HOLDS THE SUB-DIAGONAL ENTRIES OF MATRIX 'G'.
C EB: IT HOLDS THE DIAGONAL ENTRIES OF MATRIX 'G'.
C EC: IT HOLDS THE SUPER-DIAGONAL ENTRIES OF MATRIX 'G'.
C RHSE: IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM.
C On the above arrays will be applied the 'even' stream of the
C cyclic odd-even reduction procedure.
C INDEX: IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C OA: IT HOLDS A COPY OF THE ARRAY 'EA'.
C OB: IT HOLDS A COPY OF THE ARRAY 'EB'.
C OC: IT HOLDS A COPY OF THE ARRAY 'EC'.
C RSHO: IT HOLDS A COPY OF THE ARRAY 'RHSE'.
C On the above arrays will be applied the 'odd' stream of the
C cyclic odd-even reduction procedure.
C INDX: IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C WE: IT HOLDS THE MULTIPLIERS FOR THE 'EVEN' STREAM OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C WO: IT HOLDS THE MULTIPLIERS FOR THE 'ODD' STREAM OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C X: IT HOLDS THE SOLUTION OF THE SYSTEM.
C ITIME: IT HOLDS THE TIMING INFORMATION.
C A: IT HOLDS THE COMPUTED 'RETURN' POINTS OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C SET THE SHARED DATA.
C
C $SHARED EA, EB, EC, INDEX, OA, OB, OC, INDX,
C - RHSE, RSHO, N, WE, WO, X, IT, ND, NPROC,
C - ITIME, NSTEP, JJ, IH, IPRINT, A, IN, LPNTE, KK, ICOUNT
C
C COMMON/BL2/IFLAG
C
C SET CRITCAL SECTIONS.
C
C INITIALIZE PARALLELISM.
C
C GENERATE PARALLEL PATHS TO SET DYNAMICALLY THE NUMBER OF
C PROCESSORS TO BE UTILIZED IN THE PROGRAM.
C
C ENTER CRITICAL SECTION.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START THE REDUCTION PROCESS.
C
C GO TO 2100
C
C NPROC=0
C $USEPAR
C
C INITIALIZE PARALLELISM.
C
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
110  2700  DO 3000  K=1,N
111     LPNTE=K*IJ
112     LPNT0=LPNTE-1*IJ+1
113     ACLPNE=LPNTE
114     A(LPNTO)=LPNT0
115     3000  CONTINUE
116
117 C  THE COMPUTATION OF THE 'EVEN' LINES OF THE SYSTEM.
118
119  3500  $DOPAR$  I=1,NPROC
120     IF(NSTEP.EQ.12)  GO TO 10100
121     IS=(I-1)*IH+2
122     IE=IS+IH-2
123     DO 10000  J=IS,IE,2
124         JME=J-1
125         JPIE=J+1
126     WE(JMIE)=EAEINDEX(JM1))}
127     WE(JPIE)=EAEINDEX(JPIE))
128     Z2=WE(JM1))
129     WE(J)=EAEINDEX(J)/EBINDEX(JMIE))
130     Z3=WE(J)
131     EINDEX(JMIE))=Z2*EINDEX(JMIE))
132     EINDEX(JPIE))=Z3*EINDEX(JPIE))
133     EINDEX(J))+Z2*EINDEX(J)+Z3*EINDEX(J))
134     Z3=WE(JMIE))
135     Z2=WE(JP1E))
136     Z1=EAEINDEX(JMIE))
137     Z2=EAEINDEX(JPIE))
138     Z3=WE(JMIE))
139     Z4=WE(J)
140     CONTINUE
141
142  10000  GO TO 10000
143
144 C  THE COMPUTATION OF THE 'ODD' LINES OF THE SYSTEM.
145
146  10100  $DOPAR$  I=1,NPROC
147     IF(NSTEP.EQ.12)  GO TO 18100
148     IS=(I-1)*IH+1
149     IE=IS+IH-1
150     DO 18000  J=IS,IE,2
151         JME=J-1
152         JPIE=J+1
153     WE(JMIE)=EAEINDEX(JM1))}
154     WE(JP1E)=EAEINDEX(JPIE))
155     Z2=WE(JM1))
156     WE(J)=EAEINDEX(J)/EBINDEX(JMIE))
157     Z3=WE(J)
158     EINDEX(JMIE))=Z2*EINDEX(JMIE))
159     EINDEX(JPIE))=Z3*EINDEX(JPIE))
160     EINDEX(J))+Z2*EINDEX(J)+Z3*EINDEX(J))
161     Z3=WE(JMIE))
162     Z2=WE(JP1E))
163     Z1=EAEINDEX(JMIE))
164     Z2=EAEINDEX(JPIE))
165     Z3=WE(JMIE))
166     Z4=WE(J)
167     CONTINUE
168
169  18000  GO TO 18000
170
171 C  THE PERIODICITY OF THE SYSTEM.
172
173  7000  J=JPE=2*10
174     WE(JMIE)=EAEINDEX(JMIE)
175     Z4=WE(JMIE)
176     WE(J)=EAEINDEX(J)/EBINDEX(JMIE))
177     Z5=WE(J)
178     CONTINUE
179
180  2000  C  THE PERIODICITY OF THE SYSTEM.
181
182  14000  LVAR=JH10=2*10
183     WE(JMIE)=EAEINDEX(JMIE)
184     Z4=WE(JMIE)
185     WE(J)=EAEINDEX(J)/EBINDEX(JMIE))
186     Z5=WE(J)
187     CONTINUE
188
189  2010 C  THE LAST PHASE OF THE 'REFINEMENT' STEPS.
190
191  14000  LVAR=JH10=2*10
192     WE(JMIE)=EAEINDEX(JMIE)
193     Z4=WE(JMIE)
194     WE(J)=EAEINDEX(J)/EBINDEX(JMIE))
195     Z5=WE(J)
196     CONTINUE
197
198  2020 C  THE PERIODICITY OF THE SYSTEM.
199
200  14000  LVAR=JH10=2*10
201     WE(JMIE)=EAEINDEX(JMIE)
202     Z4=WE(JMIE)
203     WE(J)=EAEINDEX(J)/EBINDEX(JMIE))
204     Z5=WE(J)
205     CONTINUE
206
207  2030 C  THE LAST PHASE OF THE 'REFINEMENT' STEPS.
208
209  14000  LVAR=JH10=2*10
210     WE(JMIE)=EAEINDEX(JMIE)
211     Z4=WE(JMIE)
212     WE(J)=EAEINDEX(J)/EBINDEX(JMIE))
213     Z5=WE(J)
214     CONTINUE
215
216  18000  CONTINUE
C TERMINATE 50000 FORMAT(CX,'THE OUTPUT TERMINATE TIMING	CONTINUE

***

SUBROUTINE INITIAL

THE COMPUTATIONAL PROCEDURE.

***

CONTINUE

300 1300 CONTINUE

CONTINUE

RETURN

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

***************

C THE SUBROUTINE PERFORMS THE INTERCHANGING OF THE MODIFIED
C ELEMENTS AND THE SHUFFLING AND COPYING OF THE USED INDICES
C IN THE APPROPRIATE ARRAYS.

C IN THE APPROPRIATE ARRAYS.

C THE SUBROUTINE PERFORMS INITIALIZATION.

C this subroutine performs initialization.
C ELEMENTS OF THE SYSTEM IN ACCORDANCE WITH THE COMPUTATIONAL
STREAM THE SUBROUTINE IS CALLED FROM.

DIMENSION EA(256),EB(256),EC(256),INDEX(256)
DIMENSION OA(256),OB(256),OC(256),INDEX(256)
DIMENSION RHSEC(256),RHSO(256)
DIMENSION WE(256),W0(256),X(256)
DIMENSION ITIME(100)
DIMENSION AD(256),SUBD(256),SUPD(256)
DIMENSION DU(32),DL(32),V(256),INDEX(32)
$SHARED EA,EB,EC,INDEX,OA,OB,OC,INDEX,

- RHSEC,RHSE,WO,X.IT.IQ,N.PROC.
- ITIME,NSTEP,IJ,IM,PRINT,A.IN,LNPTE,KK

C SPECIFICATION OF THE ADDITIONAL ARRAYS

***************
*Phase Three*
***************

SUBROUTINE GAUSSP(IS,IE)

THIS SUBROUTINE SOLVES THE MATRIX EQUATION : DX=V, WHERE 'Q'
IS A GENERAL TRIADIGONAL PERIODIC MATRIX, SEQUENTIALLY, BY APPLYING
THE GAUSS ELIMINATION METHOD.

THE ADDITIONALLY DECLARED ARRAYS FORM THE WORKSPACED IN THE
SUBROUTINE ACCOMMODATING EACH TIME THE CORRESPONDING MODIFIED
C THE INITIAL PERIODIC ELEMENTS.

C THIS IS (LN-3), BECAUSE WE HAVE (LN-1) SUB-DIAGONAL ELEMENTS TO ELIMINATE, AND THE LAST 2 ELEMENTS ARE INCLUDED IN THE FINAL FULL (3X3) MATRIX.

C THE ELIMINATION PROCESS STARTS.

DO 10550 K=1,INN

FACT=SUBD(INDEXG(K+1))/AD(INDEXG(K))

AD(INDEXG(K+1))=AD(INDEXG(K+1))-FACT*SUPD(INDEXG(K))

V(INDEXG(K+1))=V(INDEXG(K+1))-FACT*V(INDEXG(K))

DU(K+1)=FACT*DU(K)

IF(ABS(DU(K+1)) .LT. (0.1E-08)) DU(K+1)=0.0

FACT=DL(K)/AD(INDEXG(K))

DL(K+1)=FACT*SUPD(INDEXG(K))

IF(ABS(DL(K+1)) .LT. (0.1E-08)) DL(K+1)=0.0

AD(INDEXG(LN))=AD(INDEXG(LN))-FACT*DU(K)

V(INDEXG(LN))=V(INDEXG(LN))-FACT*V(INDEXG(K))

CONTINUE

DO 10560 II=1,KREST

I=LN-II

X(INDEXG(I))=(V(INDEXG(I))-SUP(INDEXG(I)))*

X(INDEXG(I+1))-DU(I)*X(INDEXG(LN))/AD(INDEXG(I))

CONTINUE

RETURN

*END
THIS PROGRAM IMPLEMENTS THE GAUSS ELIMINATION METHOD TO SOLVE THE MATRIX EQUATION \( QY = V \) SEQUENTIALLY, WHERE \( Q \) IS A GENERAL TRIDIAGONAL PERIODIC MATRIX.

Set the required arrays.

\[
\text{DIMENSION } A(256), \text{SUBD}(256), \text{SUPD}(256), \text{DU}(256), \text{DL}(256) \\
\text{DIMENSION } V(256), Y(256), \text{ITIME}(100)
\]

**C** SPECIFICATION OF THE ARRAYS

**A** : IT HOLDS THE DIAGONAL ELEMENTS OF MATRIX \( Q \).

**SUBD** : IT HOLDS THE SUB-DIAGONAL ELEMENTS OF MATRIX \( Q \).

**SUPD** : IT HOLDS THE SUPER-DIAGONAL ELEMENTS OF MATRIX \( Q \).

**DU** : IT HOLDS THE MODIFICATIONS OF THE UPPER PERIODIC ENTRIES OF MATRIX \( Q \).

**DL** : IT HOLDS THE MODIFICATIONS OF THE LOWER PERIODIC ENTRIES OF MATRIX \( Q \).

**V** : IT HOLDS THE R.H.S. ELEMENTS OF THE SYSTEM.

**Y** : IT HOLDS THE SOLUTION OF THE SYSTEM.

**ITIME** : IT HOLDS THE TIMING INFORMATION.

**SIZE OF MATRIX \( Q \).**

\[
N=2M
\]

DO 10 I = 1, N
READ(5,9991) SUBD(I), A(I), SUPD(I)
CONTINUE
DO 11 I = 1, N
READ(5,9992) V(I)
CONTINUE

\$USEPAR

READ(5,9990) IPRINT, M
WRITE(6,9990) IPRINT, M

**THE SIZE OF MATRIX \( Q \).**

\[
N=2M
\]

DO 10 I = 1, N
READ(5,9991) SUBD(I), A(I), SUPD(I)
CONTINUE
DO 11 I = 1, N
READ(5,9992) V(I)
CONTINUE

\$DOALL 12

\$CALL TIMEST

\$RETURN

**THE INITIAL PERIODIC ELEMENTS.**

\[
\text{DU}(1)=\text{SUBD}(1)
\]

\[
\text{DL}(1)=\text{SUPD}(N)
\]

**THIS IS \((N-3)\), BECAUSE WE HAVE \((N-1)\) SUB-DIAGONAL ELEMENTS TO ELIMINATE, AND THE LAST 2 ELEMENTS ARE INCLUDED IN THE FINAL FULL \((3x3)\) MATRIX.**

**INN = N-3**

**THE ELIMINATION PROCESS STARTS.**

\[
\text{DO } 20 K=1, \text{INN}
\]

FACT = SUBD(K+1)/A(K)
A(K+1) = A(K+1) - FACT*SUPD(K)
V(K+1) = V(K+1) - FACT*V(K)
DU(K+1) = FACT*DU(K)
IF(ABS(DU(K+1)) LT (0,1E-08)) DU(K+1) = 0.0
FACT = DL(K)/A(K)
DL(K+1) = FACT*SUPD(K)
IF(ABS(DL(K+1)) LT (0,1E-08)) DL(K+1) = 0.0
A(N) = A(N) - FACT*DU(K)
V(N) = V(N) - FACT*V(K)
CONTINUE

**THE LAST PART OF THE ELIMINATION PROCESS, WHEN THE ORIGINAL MATRIX HAS BECOME A FULL \((3x3)\) MATRIX.**

FACT = SUBD(N-1)/A(N-2)
A(N-1) = A(N-1) - FACT*SUPD(N-2)
DU(N-1) = SUPD(N-1) - FACT*DU(N-2)
IF(ABS(DU(N-1)) LT (0,1E-08)) DU(N-1) = 0.0
V(N-1) = V(N-1) - FACT*V(N-2)
FACT = DL(N-2)/A(N-2)
DL(N-1) = SUBD(N-1) - FACT*SUPD(N-2)
IF(ABS(DL(N-1)) LT (0,1E-08)) DL(N-1) = 0.0
A(N) = A(N) - FACT*DU(N-2)
A(N) = A(N) - FACT*DU(N-1)
V(N) = V(N) - FACT*V(N-1)
V(N) = V(N) - FACT*V(N-1)

**THE SOLUTION PART.**

\[
\text{DO } 30 K=1, \text{INN}
\]

FACT = DL(N-1)/A(N-1)
A(N) = A(N) - FACT*DU(N-1)
V(N) = V(N) - FACT*V(N-1)

**THE COMPUTATION OF THE REST Y's, IN A BACK-SUBSTITUTION PROCESS.**

\[
\text{DO } 30 K=1, \text{INN}
\]

FACT = DL(N-1)/A(N-1)
A(N) = A(N) - FACT*DU(N-1)
V(N) = V(N) - FACT*V(N-1)

**THE COMPUTATION OF Y(N).**

\[
Y(N) = V(N) / A(N)
\]

**THE COMPUTATION OF Y(N-1).**

\[
Y(N-1) = (Y(N-1) - DU(N-1)*Y(N)) / A(N-1)
\]

**THE COMPUTATION OF THE REST Y's, IN A BACK-SUBSTITUTION PROCESS.**

**THE COMPUTATION OF Y(N-1).**

\[
Y(N-1) = (Y(N-1) - DU(N-1)*Y(N)) / A(N-1)
\]

**THE COMPUTATION OF THE REST Y's, IN A BACK-SUBSTITUTION PROCESS.**
I=N-1-1
Y(I)=(Y(I)-SUPD(I)*Y(I+1)-DU(I)*Y(N))/A(I)

CONTINUE

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C SET THE FORMATS
C **********
C 9990 FORMAT(2(I2,1X))
C 9991 FORMAT(3(F4.2,2X))
C 9992 FORMAT(F6.2)
C 9993 FORMAT(4(E15.4,1X))
C 9994 FORMAT(//,'TIMING-#For GAUSSP# I:/',8(E16.2X))
C 9995 FORMAT(IX,'THE SOLUTION IS I:/',5(E12.5,2X)/)

C TERMINATE PROGRAM.
50 $STOP
51 $END
C THIS PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE
C TO SOLVE THE MATRIX EQUATION : AX=B WHERE 'A' IS A GENERAL
C TRIDIAGONAL MATRIX.
C THE PROGRAM PERFORMS (N-1) REDUCTION STEPS WHERE 'N' IS THE
C EXPONENT OF THE SIZE OF THE MATRIX. THE NUMBER OF CREATED PATHS
C IS ALWAYS EQUAL TO THE NUMBER OF AVAILABLE PROCESSORS EACH TIME.
C WHILE ONE ONLY ARRAY WITH INDICES IS UTILIZED FOR EACH COMPUTATIONAL
C STREAM.
C
C SET THE REQUIRED ARRAYS.
C
DIMENSION EA(512),EB(512),EC(512),INDEX(512)
DIMENSION OA(512),OB(512),OC(512),INDX(512)
DIMENSION RHSE(512),RHSO(512)
DIMENSION WE(512),WO(512),X(512)
INTEGER A

C SPECIFICATION OF THE ARRAYS
C
C EA : IT HOLDS THE SUB-DIAGONAL ENTRIES OF MATRIX 'A'.
C EB : IT HOLDS THE DIAGONAL ENTRIES OF MATRIX 'A'.
C EC : IT HOLDS THE SUPER-DIAGONAL ENTRIES OF MATRIX 'A'.
C RHSE : IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM.
C On the above arrays will be applied the 'even' stream of the
C cyclic odd-even reduction procedure.
C INDEX : IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C OA : IT HOLDS A COPY OF THE ARRAY 'EA'.
C OB : IT HOLDS A COPY OF THE ARRAY 'EB'.
C OC : IT HOLDS A COPY OF THE ARRAY 'EC'.
C RHSO : IT HOLDS A COPY OF THE ARRAY 'RHSE'.
C On the above arrays will be applied the 'odd' stream of the
C cyclic odd-even reduction procedure.
C INDEX : IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C WE : IT HOLDS THE MULTIPLIERS FOR THE 'EVEN' STREAM OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C WO : IT HOLDS THE MULTIPLIERS FOR THE 'ODD' STREAM OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C X : IT HOLDS THE SOLUTION OF THE SYSTEM.
C ITIME : IT HOLDS THE TIMING INFORMATION.
C A : IT HOLDS THE COMPUTED 'RETURN' POINTS OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C
C SET THE SHARED DATA.
C
*SHARED EA,EB,EC,INDEX,OA,OB,OC,INDX,
RHSO,RHSE,WE,WO,X,IT,IZ,NPROC,
DO 10000 J=15,1.5E2
   JMI=J-1
   JP1=J+1
   Z1=EA(IINDEX(JMI1))
   IF(J, EQ, A(J)) GO TO 7000
   WE(JMI1)=EA(IINDEX(J))/EB(INDEX(JMI1))
   Z2=WE(JMI1)
   WE(J)=EC(INDEX(J))/EB(INDEX(JPIE))
   Z3=WE(J)
   EBI(J)=Z2*EC(INDEX(J)) + EB(INDEX(J)) +
       Z3*WE(J)
   EA(J)=Z2*Z1
   Z4=EB(INDEX(J))
   RHSECINDEX(JPIE)=Z4*RHSECINDEX(JPIEJ)+
       Z3*RHSECINDEX(JPIEJ)
   RHSE(INDEX(J)) = Z4*RHSE(INDEX(J)) +
       Z3*RHSE(INDEX(J))
   GO TO 10000
10000 CONTINUE
11000 $PAEND

C THE LAST PHASE OF THE 'REFINEMENT' STEPS.
7000 WE(J)=EA(IINDEX(J))/EB(INDEX(J))
   Z5=WE(J)
   EC(INDEX(J))=0.0
   E(A(IINDEX(J)))=Z5*Z1
   EB(INDEX(J))=Z5*Z1
   EBI(J)=Z5*EC(INDEX(J)) + EB(INDEX(J)) +
       Z5*WE(J)
   EA(J)=Z4*INDEXC(J)
   RHSECINDEX(JPIE)=Z4*RHSECINDEX(JPIE)+
       Z3*RHSE(INDEX(J))
   RHSE(INDEX(J)) = Z5*RHSE(INDEX(J)) +
       Z5*RHSE(INDEX(J))
   GO TO 10000
10000 CONTINUE
11000 $PAEND

C THE LAST PHASE OF THE 'REFINEMENT' STEPS.
DO 10000 J=15,1.5E2
   JMI=J-1
   JP1=J+1
   Z1=EA(IINDEX(JMI1))
   IF(J, EQ, A(J)) GO TO 7000
   WE(JMI1)=EB(INDEX(JMI1))/EB(INDEX(JMI1))
   Z2=WE(JMI1)
   WE(J)=EC(INDEX(J))/EB(INDEX(JPIE))
   Z3=WE(J)
   EBI(J)=Z2*EC(INDEX(J)) + EB(INDEX(J)) +
       Z3*WE(J)
   EA(J)=Z2*Z1
   Z4=EB(INDEX(J))
   RHSECINDEX(JPIE)=Z4*RHSECINDEX(JPIEJ)+
       Z3*RHSECINDEX(JPIEJ)
   RHSE(INDEX(J)) = Z4*RHSE(INDEX(J)) +
       Z3*RHSE(INDEX(J))
   GO TO 10000
10000 CONTINUE
11000 $PAEND

C THE LAST PHASE OF THE 'REFINEMENT' STEPS.
14000 WO(JPI0)=OC(INDEX(J))/OB(INDEX(JPI0))
1414 C $DOALL 30000 CALL TIMOUT(ITEM)
1415 C CALL SOLVER
1416 C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
1417 C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL
1418 C PROCEDURE.
1419 WRITE(6,40000) ITEM
1420 FORMAT(/'TOTAL TIMING: TIME=', T1,'/SEC./')
1421 IF(ITEM.EQ.0) GO TO 60000
1422 WRITE(6,50000) (X(I),I=1,IT)
1423 C TERMINATE PROGRAM.
2000 CONTINUE
2000 60000 $STOP
2010 $END
202 C $DOALL 30000 CALL TIMOUT(ITEM)
203 C CALL SOLVER
204 C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
205 C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL
206 C PROCEDURE.
207 WRITE(6,40000) ITEM
208 C TERMINATE PROGRAM.
209 C THIS SUBROUTINE PERFORMS INITIALIZATION.
210 C DIMENSION EA(512),EB(512),EC(512),INDEX(512)
211 C DIMENSION OA(512),OB(512),OC(512),INDEX(512)
212 C DIMENSION RHSEC(512),RHSE(512)
213 C DIMENSION WE(512),WO(512),X(512)
214 C DIMENSION ITIME(100),A(512)
215 C DIMENSION $INTEGER A
216 C
SUBROUTINE SHUFFLE

THIS SUBROUTINE PERFORMS THE INTERCHANGING OF MODIFIED ELEMENTS AND THE SHUFFLING AND COPYING OF THE USED INDICES IN THE APPROPRIATE ARRAYS.

DIMENSION EAI(12),EB(12),EC(12),INDEX(12)

DIMENSION OA(12),QB(12),QC(12),INDEX(12)

DIMENSION RHSE(12),RHSO(12)

DIMENSION W(12),WO(12),XI(12)

DIMENSION ITIME(100),AI(12)

INTEGER A

IF(INSTEP.EQ.IZ) GO TO 20300

COPY THE COMPUTED ELEMENTS FROM THE 'EVEN' AND 'ODD' LINES TO THE CORRESPONDING POSITIONS IN THE OPPOSITE ARRAYS.

DO 20100 J=2,JT,2

OAI(INDO(X))=EAI(INDO(X))

OBI(INDO(X))=EBI(INDO(X))

OCI(INDO(X))=ECI(INDO(X))

EAI(INDO(X))=OAI(INDO(X))

EBI(INDO(X))=OBI(INDO(X))

ECI(INDO(X))=OCI(INDO(X))

RHSE(INDO(X))=RHSE(INDO(X))

RHSO(INDO(X))=RHSO(INDO(X))

CONTINUE

IF(INSTEP.EQ.IZ) GO TO 20500

LS=0

DO 20600 J=IT/2+1

INDEX(M)=INDEX(1)

CONTINUE

IF(INSTEP.EQ.IZ) GO TO 20800

LS=0

DO 20900 J=IT/2+1

INDEX(M)=INDEX(1)

CONTINUE

SUBROUTINE SOLVER

THIS SUBROUTINE PERFORMS THE SOLUTION PART.

DIMENSION EA(512),EB(512),EC(512),INDEX(512)

DIMENSION OA(512),OB(512),OC(512),INDEX(512)

DIMENSION RHSE(512),RHSO(512)

DIMENSION W(512),WO(512),XI(512)

DIMENSION ITIME(100),A(512)

INTEGER A

IF(INSTEP.EQ.IZ) GO TO 20300

COPY THE COMPUTED ELEMENTS FROM THE 'EVEN' AND 'ODD' LINES TO THE CORRESPONDING POSITIONS IN THE OPPOSITE ARRAYS.

DO 20100 J=2,JT,2

OAI(INDO(X))=EAI(INDO(X))

CONTINUE

RETURN

END
IPAIRS=IT/4
IT2=IT/2
C THE ROUNING PROCEDURE TO THE NEAREST (SMALLER OR GREATER) INTEGER.
IH=IPAIRS/NPROC
L=2*MOD(IPAIRS,NPROC)
IF(L.GE.NPROC) IH=IH+1
IH=2*IH
C THE COMPUTATION OF THE VALUES ON THE 'EVEN' CONSIDERED ARRAYS.

*DOPAR 21200 I=1,NPROC
   IS=(I-1)*IH+1
   IE=IS+IH-1
   IF(I.EQ.NPROC) IE=IT2
   DO 21100 J=IS,IE,2
      JPIE=J+1
      IX1=INDEX(JPIE)
      IX2=INDEX(J)
      IX3=INDEX(J)
      X4=RHSO(INDEX(JPIE))/EB(IX3)
      IF(ABS(SL.LT.(0.1E-06)) SL=0.0
      SK=SL*X2+EB(IX1)
      SR=SL*X4+RHSO(INDEX(JPIE))
      X(IY1)=SR/SK
      X(IY3)=(X4-X2*X(IY1))/EB(IX3)
   CONTINUE
   21100

*PAREND

C THE COMPUTATION OF THE VALUES ON THE 'ODD' CONSIDERED ARRAYS.

*DOPAR 21400 I=1,NPROC
   IS=(I-1)*IH+1
   IE=IS+IH-1
   IF(I.EQ.NPROC) IE=IT2
   DO 21300 J=IS,IE,2
      JPO=J+1
      IY1=INDEX(JPO)
      IY2=INDEX(J)
      IY3=INDEX(J)
      Y4=RHSO(INDEX(J))
      SL=0.0(INDEX(JPO))/OB(IY3)
      IF(ABS(SL.LT.(0.1E-06)) SL=0.0
      SK=SL*Y2+OB(IY1)
      SR=SL*Y4+RHSO(INDEX(JPO))
      X(IY1)=SR/SK
      X(IY3)=(Y4-Y2*X(IY1))/OB(IY3)
   CONTINUE
   21300

*PAREND

*END
C
C THIS PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE
C TO SOLVE THE MATRIX EQUATION \( AX=B \), WHERE \( A \) IS A GENERAL
C TRIDIAGONAL MATRIX.
C THE NUMBER OF CREATED PATHS IS ALWAYS EQUAL TO THE NUMBER OF
C AVAILABLE PROCESSORS EACH TIME.
C THEN, AFTER A PRE-SET NUMBER OF REDUCTION STEPS, WE CONTINUE BY
C APPLYING THE ODD-EVEN REDUCTION TECHNIQUE AGAIN, BUT SEQUENTIALLY
C THIS TIME, TO SOLVE THE RESULTING SUBSYSTEMS IN EACH OF THE CREATED
C PARALLEL PATHS SIMULTANEOUSLY, FOR EACH COMPUTATIONAL STREAM ONLY.
C ONE ARRAY WITH INDICES IS UTILIZED.
C
C SET THE REQUIRED ARRAYS.
C
DIMENSION EA(256),EB(256),EC(256),INDEX(256)
DIMENSION OA(256),OB(256),OC(256),INDEX(256)
DIMENSION RHSE(256),RHSO(256)
DIMENSION WE(256),WO(256),X(256)
DIMENSION ITIME(100),A(256)
INTEG A

C SPECIFICATION OF THE ARRAYS
C================================================================================
C EA : IT HOLDS THE SUB-DIAGONAL ENTRIES OF MATRIX 'O'.
C EB : IT HOLDS THE DIAGONAL ENTRIES OF MATRIX 'O'.
C EC : IT HOLDS THE SUPER-DIAGONAL ENTRIES OF MATRIX 'O'.
C RHSE : IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM.
C
C On the above arrays will be applied the 'even' stream of the
C cyclic odd-even reduction procedure.
C INDEX : IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C OA : IT HOLDS A COPY OF THE ARRAY 'EA'.
C OB : IT HOLDS A COPY OF THE ARRAY 'EB'.
C OC : IT HOLDS A COPY OF THE ARRAY 'EC'.
C RHSO : IT HOLDS A COPY OF THE ARRAY 'RHSE'.
C
C On the above arrays will be applied the 'odd' stream of the
C cyclic odd-even reduction procedure.
C INDEX : IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C WE : IT HOLDS THE MULTIPLIERS FOR THE 'EVEN' STREAM OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C WO : IT HOLDS THE MULTIPLIERS FOR THE 'ODD' STREAM OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C ITIME : IT HOLDS THE TIMING INFORMATION.
C A : IT HOLDS THE COMPUTED 'RETURN' POINTS OF THE CYCLIC
C ODD-EVEN REDUCTION PROCEDURE.
C
C SET THE SHARED DATA.
C
#SHARED EA,EB,EC,INDEX,OA,OB,OC,INDEX,
- RHSE,RHSO,N,WE,WO,X,IT,IZ,NPROC.
- ITIME,NSTEP,IJ,IN,PRINT,A,KK
COMMON/BL2/IFLAG
C SET CRITICAL SECTIONS.
C INITIALIZE PARALLELISM.
#USEPAR
NPROC=NPROC+1
C GENERATE PARALLEL PATHS TO SET DYNAMICALLY THE NUMBER OF
C PROCESSORS TO BE UTILIZED IN THE PROGRAM.
#DOALL 1000
C ENTER CRITICAL SECTION.
#ENTER NPROC
NPROC=NPROC+1
C RELEASE CRITICAL SECTION.
C NEXT NPROC
C TERMINATE PARALLEL PATHS AND SYNCHRONIZE PROCESSORS.
1000 #PAREND
CALL INITIAL
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C [ ] [ ] [ ] [ ] [ ] [ ]
C [ ] [ ] [ ] [ ] [ ] [ ]
C
C TERMINATE THE PROGRAM.
2000 #PAREND
C
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C TERMINATE THE PROGRAM.
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C TERMINATE THE PROGRAM.
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C TERMINATE THE PROGRAM.
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C TERMINATE THE PROGRAM.
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C TERMINATE THE PROGRAM.
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C C [ ] [ ] [ ] [ ] [ ] [ ]
C C [ ] [ ] [ ] [ ] [ ] [ ]
C
C TERMINATE THE PROGRAM.
C THE COMPUTATION OF THE ‘EVEN’ LINES OF THE SYSTEM.

109 3000 CONTINUE

110 C

111 C THE COMPUTATION OF THE ‘EVEN’ LINES OF THE SYSTEM.

112 C

113 3500 $DOPAR 11000 I=1,NPROC

114 IF(NSTEP, EQ, 12) GO TO 10100

115 IS=(I-1)*IH+2

116 IE=IS+IH-2

117 IF(I,EQ,NPROC) IE=IT

118 DO 10000 J=IS,IE,2

119 JMIE=J-1

120 JPIE=J+1

121 Z1=EA(INDEX(JMIE))

122 IF(J,EQ,A(J)) GO TO 7000

123 WE(JMIE)=EA(INDEX(J))/EB(INDEX(JMIE))

124 Z2=WE(JMIE)

125 WE(J)=EC(INDEX(J))/EB(INDEX(J))

126 Z3=WE(J)

127 EB(INDEX(J))=Z2*EC(INDEX(JMIE))+EB(INDEX(J))

128 " CONTINUE

129EA(INDEX(J))=Z2*Z1

130 EC(INDEX(J))=Z3*EC(INDEX(JMIE))

131 RHSE(INDEX(J))=Z2*RHSE(INDEX(JMIE))

132 " CONTINUE

133 DO TO 10000

134 C

135 C THE LAST PHASE OF THE ‘REFINEMENT’ STEPS.

136 C

137 7000 WE(J)=EA(INDEX(J))/EB(INDEX(J))

138 Z5=WE(J)

139 EC(INDEX(J))=0.0

140 EA(INDEX(J))=Z5*Z1

141 EB(INDEX(J))=Z2*EC(INDEX(JMIE))+EB(INDEX(J))

142 RHSE(INDEX(J))=Z2*RHSE(INDEX(JMIE))

143 " CONTINUE

144 DO TO 10000

145 10100 IFLAG=0

146 IF(I,H,E,0) GO TO 10200

147 IHL=IH

148 ICOUNT=IH

149 GO TO 10500

150 10200 IL=2*KK

151 IF(I,EQ,NPROC) GO TO 10450

152 IHL=IH

153 ICOUNT=IH

154 IF(I,EQ,NPROC) IHL=IH+KK

155 GO TO 10500

156 10450 IHL=IH+1

157 ICOUNT=IH

158 IF(I,EQ,NPROC) IHL=KK

159 10500 DO 10700 JL=1,IHL

160 IS=(JL-1)*JL+ICOUNT+J*(J-1)

161 IE=IS+J-1

162 CALL OENNP(IS,IE)

163 10700 CONTINUE

164 11000 $PAREND

165 C

166 C THE COMPUTATION OF THE ‘ODD’ LINES OF THE SYSTEM.

167 C

168 $DOPAR 19000 I=1,NPROC

169 IF(NSTEP, EQ, 12) GO TO 18100

170 IS=(I-1)*IH+1

171 IE=IS+IH-1

172 IF(I,EQ,NPROC) IE=IT

173 DO 18000 J=IS,IE,2

174 JM10=J-1

175 JPI0=J+1

176 IF(J,EQ,A(J)) GO TO 14000

177 Q1=OA(INDEX(JM10))

178 W0(J)=OA(INDEX(J))/OB(INDEX(JM10))

179 Q2=WO(J)

180 W0(JP10)=OC(INDEX(J))/OB(INDEX(JP10))

181 Q3=W0(JP10)

182 OC(INDEX(J))=Q3+OC(INDEX(JP10))

183 OB(INDEX(J))=Q3+OA(INDEX(JP10))

184 Q2=OC(INDEX(JM10))

185 OA(INDEX(J))=Q2+Q1

186 RHS0(INDEX(J))=Q2*RHS0(INDEX(JM10))

187 " CONTINUE

188 GO TO 18000

189 C

190 C THE LAST PHASE OF THE ‘REFINEMENT’ STEPS.

191 C

192 14000 W0(JP10)=OC(INDEX(J))/OB(INDEX(JP10))

193 Q5=W0(JP10)

194 Q6=Q5*OC(INDEX(JP10))

195 OB(INDEX(J))=Q5+OA(INDEX(JP10))

196 OC(INDEX(J))=Q6+OC(INDEX(JP10))

197 RHS0(INDEX(J))=RHS0(INDEX(JP10))

198 " CONTINUE

199 18000 CONTINUE

200 GO TO 19000

201 18100 IFLAG=1

202 IF(KK,NE,0) GO TO 18200

203 IHL=IH

204 ICOUNT=IH

205 GO TO 18500

206 L=2*KK

207 IF(L,EQ,NPROC) GO TO 18450

208 IHL=IH

209 ICOUNT=IH

210 IF(I,EQ,NPROC) IHL=IH+KK

211 GO TO 18500

212 IHL=IH+1

213 ICOUNT=IH

214 IF(I,EQ,NPROC) IHL=KK

215 GO TO 18500

216 IS=(JK-1)*J+1+ICOUNT+J*(J-1)
CALL GENGPIIS, IE
CONTINUE

IF(INSTEP.EQ.IZ-1) GO TO 21000
CALL STLFI
CONTINUE

C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.

C TERMINATE PROGRAM.

WRITE(6,40000) ITIME
FORMAT(/, 'TIMING(S)-*(For PERSON=*) /', /, (I6,2X))
IF(IPRINT.EQ.0) GO TO 60000
WRITE(6,40000) (X(1), I=1, IT)
FORMAT(/, 'THE SOLUTION IS /', /, (E15.6,2X))

RETURN

END

SUBROUTINE INITIAL
C THIS SUBROUTINE PERFORMS INITIALIZATION.

DIMENSION EA(256), EB(256), EC(256), INDEX(256)
DIMENSION DA(256), DB(256), DC(256), IND(256)
DIMENSION WE(256), WO(256), X(256)
DIMENSION ITIME(IOO), A(256)
INTEGER A

IF(INSTEP.EQ.(12-1)) GO TO 74300

DO 74100 NSIZE•0
READ(5,1100) IPRINT, N
WRITE(6,1100) IPRINT, N
CONTINUE

DO 74200 IT=2*N
WRITE(6,1120) IT
WRITE(6,1120) IT
CONTINUE

END

SUBROUTINE SHUFI
C THIS SUBROUTINE PERFORMS THE INTERCHANGING OF THE MODIFIED ELEMENTS AND THE SHUFFLING AND COPYING OF THE USED INDICES IN THE APPROPRIATE ARRAYS.

DIMENSION EA(256), EB(256), EC(256), INDEX(256)
DIMENSION DA(256), DB(256), DC(256), IND(256)
DIMENSION WE(256), WO(256), X(256)
DIMENSION ITIME(IOO), A(256)
INTEGER A

IF(INSTEP.EQ.(12-1)) GO TO 74300

DO 74100 NSIZE•0
READ(5,1100) IPRINT, N
WRITE(6,1100) IPRINT, N
CONTINUE

DO 74200 IT=2*N
WRITE(6,1120) IT
WRITE(6,1120) IT
CONTINUE

END
SUBROUTINE GENNP(IS,IE)

C THIS SUBROUTINE SOLVES THE MATRIX EQUATION: QX=Z, WHERE 'Q'
C IS A GENERAL TRIDIAGONAL MATRIX SEQUENTIALLY, BY APPLYING
C THE ADDITIONALLY DECLARED ARRAYS FORM THE WORKSPACE IN THE
C SUBROUTINE ACCOMMODATING EACH TIME THE CORRESPONDING MODIFIED
C ELEMENTS OF THE SYSTEM IN ACCORDANCE WITH THE COMPUTATIONAL
C STREAM THE SUBROUTINE IS CALLED FROM.

DIMENSION EA(256), ED(256), EC(256), INDEX(256)
DIMENSION OA(256), OB(256), OC(256), INDEX(256)
DIMENSION RHEG(256), RHSG(256)
DIMENSION WEC(256), WDO(256), X(256)
DIMENSION EAG(256), EBO(256), EC(256), INDEXO(32)
DIMENSION OAG(256), OBG(256), OC(256), INDEXO(32)
DIMENSION RHEG(256), RHSE(256)
DIMENSION WE(256), WDO(256)
DIMENSION ITIME(100), A(256), A0(32)
DIMENSION IEAG, IEB, IEC, INDEXA, OB, OC, INDEXO,
  RHEG, RHSG, N, W, W0, X, IT, II, NPROC,
   ITIME, NSTEP, J, JH

COMMON/BL2/EAG, EBO, EC, INDEXA, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
  BL2/EAG, EBO, EC, INDEXO, OA, OB, OC, INDEXO,
**SUBROUTINE INITO(IS,IJE)**

**Solution Part**

**Phase Four**

**SUBROUTINE INITO(IS,IJE)**

**THIS SUBROUTINE PERFORMS INITIALIZATION.**

**DIMENSION EA(256),EB(256),EC(256),INDEX(256)**

**DIMENSION OA(256),OB(256),OC(256),INDOX(256)**

**DIMENSION RHSE(256),RHSO(256)**

**DIMENSION WE(256),WO(256),X(256)**

**DIMENSION EAQ(256),EBQ(256),ECQ(256),INDEXQ(32)**

**DIMENSION OAQ(256),OBQ(256),OCQ(256),INDOXQ(32)**

**DIMENSION RHSEQ(256),RHSOQ(256)**

**DIMENSION WEO(32),WOO(32)**

**DIMENSION ITIME(100,A(256),A(32)**

**INTEGER AAO**

**SHARED EA,EB,EC,INDEX,OA,OB,OC,INDOX,**

- RHSE,RHSO,N,WE,W0,X,IT,IZ,NPROC,

- ITIME,NSTEP,I,J,H,IFRINT,A,KK

**COMMON/BLI/AEO,EBQ,ECQ,INDOXQ,OAQ,OBQ,OCQ,INDOXQ,**

- RHSEQ,RHSOQ,WEQ,WOQ,AAQ,ISTEP,NSTEQ

**CONTINUE**

**THE INTERCHANGING OF THE MODIFIED ELEMENTS AND THE SHUFFLING AND COPYING OF THE USED INDICES IN THE APPROPRIATE ARRAYS.**
DIMENSION RHSEG(256), RHS0(256)
DIMENSION WEO(32), WOG(32)
DIMENSION ITIME(100), A(256), A0(32)

INTEGER A

*SHARED EA, EB, EC, INDEX, OA, OB, OC, INDX.

- RHSEG, RHS0, NEW, X, IT, XII, NPROC,

- ITIME, NSTEP, IJ, IH, IPRINT, A, KK

COMMON/B1/EAO, EBO, ECO, INDEXJ, OAO, OBO, OCG, INDOXG,

- RHSEG, RHS0, NEW, X, A0, ISTEP, NSTEO

IF(NSTEP.EQ.ISTEP) GO TO 74300

C COPY THE COMPUTED ELEMENTS FROM THE 'EVEN' AND 'ODD' LINES
C TO THE CORRESPONDING POSITIONS IN THE OPPOSITE ARRAYS.

DO 74100 J=2, J+2
    EA(INDEXG(J)) = EA(INDEXG(J))
    OB(INDEXG(J)) = OB(INDEXG(J))
    OCG(INDEXG(J)) = OCG(INDEXG(J))
    EAQ(INDEXG(J-1)) = EAQ(INDOXG(J-1))
    EBO(INDEXG(J-1)) = EBO(INDOXG(J-1))
    ECG(INDEXG(J-1)) = ECG(INDOXG(J-1))
    RSQG(INDEXG(J)) = RHSQG(INDOXG(J))
    RHSEG(INDEXG(J)) = RHSEG(INDOXG(J))

74100 CONTINUE

C SHUFFLE THE 'EVEN' AND 'ODD' USED INDICES TO THE TOP OF
C THE CORRESPONDING ARRAYS.

74300 LS=0
DO 74400 J=2, J+2
    LS=LS+1
    INDEXG(LS)=INDEXG(J)
    INDOXG(LS)=INDOXG(J-1)

74400 CONTINUE

C COPY THE 'ODD' AND 'EVEN' USED INDICES AT THE REAR HALF
C EACH OTHER'S ARRAY IN ORDER TO PROCEED AGAIN WITH THE
C NEW ODD-EVEN CYCLE.

IF(NSTEP.EQ.ISTEP) GO TO 74600

74500 LS=0
DO 74600 M=M+1
    LS=LS+1
    INDEX(M)=INDOXG(LS)
    INDOX(M)=INDEXG(LS)

74600 CONTINUE

74700 RETURN

2. C

**************

C Phase S i x

**************

SUBROUTINE SOLVER

DIMENSION EA(256), EB(256), EC(256), INDEX(256)

DIMENSION EA(256), EB(256), EC(256), INDEX(256)

DIMENSION WEO(32), WOG(32)

DIMENSION ITIME(100), A(256), A0(32)

INTEGER A

*SHARED EA, EB, EC, INDEX, OA, OB, OC, INDX.

- RHSEG, RHS0, NEW, X, IT, XII, NPROC,

- ITIME, NSTEP, IJ, IH, IPRINT, A, KK

COMMON/B1/EAO, EBO, ECO, INDEXJ, OAO, OBO, OCG, INDOXG,

- RHSEG, RHS0, NEW, X, A0, ISTEP

IF(NSTEP.EQ.ISTEP) GO TO 74600

C COPY THE COMPUTED ELEMENTS FROM THE 'EVEN' AND 'ODD' LINES
C TO THE CORRESPONDING POSITIONS IN THE OPPOSITE ARRAYS.

DO 74100 J=1, J+2
    JPIE=J+1
    XI=INDEXG(JPIE)
    X2=ECG(INDEXG(J))
    IX3=INDEXG(J)
    X4=RHSEG(INDEXG(J))
    SL=EAQ(INDEXG(JPIE))/EBO(X3)
    IF(ABS(SL).LT.(0.1E-05)) SL=0.0
    SR=SL*X4+RHSEG(INDEXG(JPIE))
    X3=SR/SK

74100 CONTINUE

C THE COMPUTATION OF THE VALUES ON THE 'ODD' CONSIDERED ARRAYS.

76100 CONTINUE

C THE COMPUTATION OF THE VALUES ON THE 'ODD' CONSIDERED ARRAYS.

76200 CONTINUE

C
C THIS PROGRAM IMPLEMENTS THE CYCLIC ODD-EVEN REDUCTION TECHNIQUE
C TO SOLVE THE MATRIX EQUATION : Gx=b, WHERE 'O' IS A GENERAL
C TRIDIAGONAL MATRIX.
C THE NUMBER OF CREATED PATHS IS ALWAYS EQUAL TO THE NUMBER OF
C AVAILABLE PROCESSORS EACH TIME.
C THEN, AFTER A PRE-SET NUMBER OF REDUCTION STEPS, WE continue BY
C APPLYING THE GAUSS ELIMINATION SEQUENTIAL PROCEDURE TO SOLVE THE
C RESULTING SUBSYSTEMS IN EACH OF THE CREATED PARALLEL PATHS.
C SIMULTANEOUSLY, FOR EACH COMPUTATIONAL STREAM ONLY ONE ARRAY WITH
C INDICES IS UTILIZED.
C
C SET THE REQUIRED ARRAYS.
C
DIMENSION EA(256), EB(256), EC(256), INDEX(256)
DIMENSION OA(256), OB(256), OC(256), INDEX(256)
DIMENSION RHSE(256), RHSO(256)
DIMENSION WE(256), WO(256), X(256)
DIMENSION OA(256), OB(256), OC(256), INDEX(256)
DIMENSION EA(256), EB(256), EC(256)

INTEGER A

C SPECIFICATION OF THE ARRAYS
C
C EA : IT HOLDS THE SUB-DIAGONAL ENTRIES OF MATRIX 'O'.
C EB : IT HOLDS THE DIAGONAL ENTRIES OF MATRIX 'O'.
C EC : IT HOLDS THE SUPER-DIAGONAL ENTRIES OF MATRIX 'O'.
C RHSE : IT HOLDS THE R.H.S. ENTRIES OF THE SYSTEM.
C
C On the above arrays will be applied the 'even' stream of the
C cyclic odd-even reduction procedure.
C INDEX : IT HOLDS THE INDICES OF THE 'EVEN' STREAM OF THE CYCLIC
C REDUCTION PROCEDURE.
C OA : IT HOLDS A COPY OF THE ARRAY 'EA'.
C OB : IT HOLDS A COPY OF THE ARRAY 'EB'.
C OC : IT HOLDS A COPY OF THE ARRAY 'EC'.
C RHSO : IT HOLDS A COPY OF THE ARRAY 'RHSE'.
C
C On the above arrays will be applied the 'odd' stream of the
C cyclic odd-even reduction procedure.
C INDEX : IT HOLDS THE INDICES OF THE 'ODD' STREAM OF THE CYCLIC
C REDUCTION PROCEDURE.
C WE : IT HOLDS THE MULTIPLIERS FOR THE 'EVEN' STREAM OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C WO : IT HOLDS THE MULTIPLIERS FOR THE 'ODD' STREAM OF THE
C CYCLIC ODD-EVEN REDUCTION PROCEDURE.
C X : IT HOLDS THE SOLUTION OF THE SYSTEM.
C ITIME : IT HOLDS THE TIMING INFORMATION.
C A : IT HOLDS THE COMPUTED 'RETURN' POINTS OF THE CYCLIC
C REDUCTION PROCEDURE.
C
C SET THE SHARED DATA.
LPNTO=LPNTE-IJ+1
A(LPNTE)=LPNTE
A(LPNTO) = LPNTO
3000 CONTINUE
C THE COMPUTATION OF THE 'EVEN' LINES OF THE SYSTEM.

116 3500 #DOPAR 11000 I=1,NPROC
117 IF(NSTEP.EQ.1) GO TO 10100
118 IS=(I-1)*I+2
119 IE=IS+IH-2
120 IF(I.EQ.NPROC) IE=IT
121 DO 10000 J=IS,IE,2
122 JMIE=J-1
123 JPIE=J+1
124 Z1=EA(INDEX(JMIE))
125 IF(J.EQ.A(J)) GO TO 7000
126 WE(JMIE)=EA(INDEX(J))/EB(INDEX(JMIE))
127 Z2=WE(JMIE)
128 WE(J)=EC(INDEX(J))/EB(INDEX(JPIE))
129 Z3=WE(J)
130 ED(INDEX(J))+Z2*EC(INDEX(JMIE))+EB(INDEX(J))+Z3*WE(J)+
131 - Z3*EA(INDEX(JPIE))
132 EA(INDEX(J))+Z2+Z1
133 EC(INDEX(J))+Z3*EC(INDEX(JPIE))
134 RHSE(INDEX(J))+Z2*RHS(IINDEX(JMIE))
135 - RHSE(INDEX(J))+Z3*RHSE(INDEX(JPIE))
136 GO TO 10000
137 C
138 C THE LAST PHASE OF THE 'REFINEMENT' STEPS.
139 C
140 7000 WE(J)=EA(INDEX(J))/EB(INDEX(JMIE))
141 Z5=WE(J)
142 EC(INDEX(J))+0.0
143 EA(INDEX(J))+Z5+Z1
144 EB(INDEX(J))+Z5*EC(INDEX(JMIE))+EB(INDEX(J))+
145 RHSE(INDEX(J))+Z5*RHSE(INDEX(J))+Z5*WE(J)+
146 - RHSE(INDEX(JMIE))
147 10000 CONTINUE
148 GO TO 10000
149 10100 IFLAG=0
150 IF(KK.NE.0) GO TO 10200
151 IHL=IH
152 ICOUNT=1HL
153 GO TO 10500
154 10200 L=2*KK
155 IF(GL.EQ.NPROC) GO TO 10450
156 IHL=IH
157 ICOUNT=1HL
158 If(I.EQ.NPROC) IHL=1H+KK
159 GO TO 10500
160 10450 IHL=1H+1
161 ICOUNT=1HL
162 If(I.EQ.NPROC) IHL=KK
163 10500 DO 10700 JL=1,1H
164 IS=(JL-1)*IJ+1
165 IE=IS+IJ-1
166 CALL GAUSNIP(IS,IE)
167 10700 CONTINUE
168 11000 #PEND
169 C
170 C THE COMPUTATION OF THE 'ODD' LINES OF THE SYSTEM.
171 C
172 14000 CONTINUE
173 GO TO 14000
174 18000 GO TO 18000
175 18200 GO TO 18200
176 18400 GO TO 18400
177 18500 GO TO 18500
178 18700 GO TO 18700
179 18900 GO TO 18900
180 19100 GO TO 19100
181 19300 GO TO 19300
182 19500 GO TO 19500
183 19700 GO TO 19700
184 19900 GO TO 19900
185 20100 GO TO 20100
186 20300 GO TO 20300
187 20500 GO TO 20500
188 20700 GO TO 20700
189 20900 GO TO 20900
190 21100 GO TO 21100
191 21300 GO TO 21300
192 21500 GO TO 21500
193 21700 GO TO 21700
194 21900 GO TO 21900
195 22100 GO TO 22100
196 22300 GO TO 22300
197 22500 GO TO 22500
198 22700 GO TO 22700
199 22900 GO TO 22900
200 23100 GO TO 23100
201 23300 GO TO 23300
202 23500 GO TO 23500
203 23700 GO TO 23700
204 23900 GO TO 23900
205 24100 GO TO 24100
206 24300 GO TO 24300
207 24500 GO TO 24500
208 24700 GO TO 24700
209 24900 GO TO 24900
210 25100 GO TO 25100
211 25300 GO TO 25300
212 25500 GO TO 25500
213 25700 GO TO 25700
214 25900 GO TO 25900
215 26100 GO TO 26100
216 26300 GO TO 26300

C THE LAST PHASE OF THE 'REFINEMENT' STEPS.
ICOUNT = IHL
IF (I.EQ. NPROC) IHL = KK
DO 18700 JK = 1, IHL
IS = (JK-1)*I+1
ICOUNT = JK
IE = IS + I - 1
CALL GAUSSNP(IS,IE)
CONTINUE
18700
19000 *PAREND
19000 IF (NSTEP.EQ.12) GO TO 21000
21000 CALL SHUF1
21000 CONTINUE
21000 C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
230 C
231 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
232 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
233 C #DOALL 30000
234 C CALL TIMOUT(ITIME)
235 30000 *PAREND
236 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
237 C [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
238 C
239 C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL
240 C PROCEDURE.
241 C
242 40000 FORMAT(6,40000) ITIME
243 40000 FORMAT(6,40000) ITIME
244 40000 FORMAT(6,40000) ITIME
245 50000 FORMAT(6,50000) (X(1), I=1,1,IT)
246 50000 FORMAT(6,50000) (X(1), I=1,1,IT)
247 50000 FORMAT(6,50000) (X(1), I=1,1,IT)
248 60000 *STOP
249 C
250 C
251 C ***************
252 C * Phase One *
253 C ***************
254 C
255 C SUBROUTINE INITIAL
256 C
257 C THIS SUBROUTINE PERFORMS INITIALIZATION.
258 C
259 C DIMENSION EA(256),EB(256),EC(256),INDEX(256)
260 C DIMENSION DA(256),DB(256),DC(256),INDEX(256)
261 C DIMENSION RHE(256),RHO(256)
262 C DIMENSION ME(256),WO(256),X(256)
263 C DIMENSION ITIME(100),A(256)
264 C INTEGER A
265 C $SHARED EA,EB,EC.INDX.OA.OB,OC,INDXX,
266 C RHSE,RHPO,N,E,W,X,I,IT,IZ,NPROC,
267 C ITIME,NSTEP,IK,IM,IPRINT,AKK
268 C READ (5,1100) IPRINT
269 C WRITE (6,1100) IZ
270 C READ (3,1120) IZ
271 C WRITE (6,1120) IZ
272 1120 FORMAT (I2)
273 1120 FORMAT (I2)
274 1120 FORMAT (I2)
275 1120 FORMAT (I2)
276 1150 CONTINUE
277 1150 CONTINUE
278 1150 CONTINUE
279 1200 READ (5,1200) EA(J),EB(J),EC(J)
280 1200 READ (5,1200) EA(J),EB(J),EC(J)
281 1200 READ (5,1200) EA(J),EB(J),EC(J)
282 1200 READ (5,1200) EA(J),EB(J),EC(J)
283 1200 INDEX(J) = J
284 1200 INDEX(J) = J
285 1300 CONTINUE
286 1300 DO 1500 J = 1, IT
287 1300 CONTINUE
288 1400 READ (5,1400) RHE(J)
289 1400 READ (5,1400) RHE(J)
290 1400 READ (5,1400) RHE(J)
291 1500 CONTINUE
292 1500 CONTINUE
293 C
294 C ***************
295 C * Phase Two *
296 C ***************
297 C
298 C SUBROUTINE SHUF1
299 C
300 C THIS SUBROUTINE Performs THE INTERCHANGING OF THE MODIFIED
301 C ELEMENTS AND THE SHUFFLING AND COPYING OF THE USED INDICES
302 C IN THE APPROPRIATE ARRAYS.
303 C
304 C DIMENSION EA(256),EB(256),EC(256),INDEX(256)
305 C DIMENSION DA(256),DB(256),DC(256),INDEX(256)
306 C DIMENSION RHE(256),RHO(256)
307 C DIMENSION ME(256),WO(256),X(256)
308 C DIMENSION ITIME(100),A(256)
309 C INTEGER A
310 C $SHARED EA,EB,EC,INDX.OA.OB,OC,INDXX,
311 C RHSE,RHPO,N,E,W,X,I,IT,IZ,NPROC,
312 C ITIME,NSTEP,IK,IM,IPRINT,AKK
313 C IF (NSTEP.EQ.11) GO TO 19200
314 C
315 C COPY THE COMPUTED ELEMENTS FROM THE 'EVEN' AND 'ODD' LINES
316 C TO THE CORRESPONDING POSITIONS IN THE OPPOSITE ARRAYS.
317 C
318 C DO 19100 J = 2, IT, 2
319 C OAJ(INDEX(J)) = EA(INDEX(J))
320 C OB(INDEX(J)) = EB(INDEX(J))
321 C OC(INDEX(J)) = EC(INDEX(J))
322 C EA(INDEX(J-1)) = OA(INDEX(J-1))
323 C EB(INDEX(J-1)) = OB(INDEX(J-1))
324 C EC(INDEX(J-1)) = OC(INDEX(J-1))
325 C
C STREAM ELEMENTS OF SUBROUTINE ACCOMODATING EACH 
C GAUSS ELIMINATION METHOD.

C THIS SUBROUTINE SOLVES THE MATRIX EQUATION: A.X = V, WHERE 'Q'.
C IS A GENERAL TRIDIAGONAL MATRIX, SEQUENTIALLY, BY APPLYING THE
C GAUSS ELIMINATION METHOD.
C THE ADDITIONALLY DECLARED ARRAYS FORM THE WORKSPACE IN THE
C SUBROUTINE ACCOMODATING EACH TIME THE CORRESPONDING MODIFIED
C ELEMENTS OF THE SYSTEM, IN ACCORDANCE WITH THE COMPUTATIONAL
C STREAM THE SUBROUTINE IS CALLED FROM.

C SPECFICATION OF THE ADDITIONAL ARRAYS

C ***************************************************************

C AD : IT HOLDS THE DIAGONAL ELEMENTS OF MATRIX 'Q'.
C SUBD : IT HOLDS THE SUPER-DIAGONAL ELEMENTS OF MATRIX 'Q'.
C SUPD : IT HOLDS THE SUB-DIAGONAL ELEMENTS OF MATRIX 'Q'.
C V : IT HOLDS THE R.H.S. ELEMENTS OF THE SYSTEM.
C INDEXO : IT HOLDS THE INDICES OF EITHER OF 'INDEX' OR 'INDOX'.
C C ARRAY DEPENDING ON THE STREAM OF THE CYCLIC ODD-EVEN
C REDUCTION PROCEDURE CALLING THE SUBROUTINE.

C THE SIZE OF MATRIX 'Q'.
C IF(IFLAG.EQ.1) GO TO 10520
C L=0
C DO 10510 J=IS,IE
C L=L+1
C INDEXO(J)=INDEXO(J)
C SUBD(INDEXO(J))=E(INDEXO(J))
C AD(INDEXO(J))=EB(INDEXO(J))
C SUPD(INDEXO(J))=EC(INDEXO(J))
C V(INDEXO(J))=RHSO(INDEXO(J))

C THE ELIMINATION PROCESS STARTS.
C DO 10550 K=1,INN
C FACT=SUBD(INDEXO(K+1))/AD(INDEXO(K))
C AD(INDEXO(K+1))=AD(INDEXO(K+1))-FACT*SUPD(INDEXO(K))
C V(INDEXO(K+1))=V(INDEXO(K+1))-FACT*V(INDEXO(K))

C THE SOLUTION PART [Back-substitution].
C X(INDEXO(LN))=V(INDEXO(LN))/AD(INDEXO(LN))
C DO 10560 L=LN-1,1
C FACT=V(INDEXO(L))/AD(INDEXO(L))
C AD(INDEXO(L))=AD(INDEXO(L))-FACT*SUPD(INDEXO(L))
C V(INDEXO(L))=V(INDEXO(L))-FACT*V(INDEXO(L))

C RETURN
C END
C THIS PROGRAM IMPLEMENTS THE GAUSS ELIMINATION METHOD TO SOLVE
C THE MATRIX EQUATION: QX = V SEQUENTIALLY, WHERE 'Q' IS A GENERAL
C TRIDIAGONAL MATRIX.
C SET THE REQUIRED ARRAYS.
C
C DIMENSION AD(256), SUPD(256), SUBD(256), V(256), X(256)
C
C SPECIFICATION OF THE ARRAYS
C
C AD : IT HOLDS THE DIAGONAL ELEMENTS OF MATRIX 'Q'.
C SUPD : IT HOLDS THE SUPER-DIAGONAL ELEMENTS OF MATRIX 'Q'.
C SUBD : IT HOLDS THE SUB-DIAGONAL ELEMENTS OF MATRIX 'Q'.
C V : IT HOLDS THE R.H.S. ELEMENTS OF THE SYSTEM.
C X : IT HOLDS THE SOLUTION OF THE SYSTEM.
C ITIME : IT HOLDS THE TIMING INFORMATION.
C
USEPAR
READ(5,9990) IPRINT,M
WRITE(6,9990) IPRINT,M
C THE SIZE OF MATRIX 'Q'.
N=2*M
DO 30 I=1,N
READ(5,9991) SUBD(I), AD(I), SUPD(I)
30 CONTINUE
DO 50 I=1,N
READ(5,9992) V(I)
50 CONTINUE
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C START TIMING THE COMPUTATIONAL PROCEDURE.
C
C
C THE ELIMINATION PROCESS STARTS.

DO 60 K=1,IN
FACT=SUBD(K+1)/AD(K)
AD(K+1)=AD(K+1)-FACT*SUPD(K)
V(K+1)=V(K+1)-FACT*V(K)
60 CONTINUE
C TERMINATE TIMING THE COMPUTATIONAL PROCEDURE.
C
C OUTPUT THE RESULTS OBTAINED FROM THE TIMED COMPUTATIONAL
C PROCEDURE.
WRITE(6,9994) X(I), I=1,N
C TERMINATE PROGRAM.
STOP
END
MATHEMATICAL BACKGROUND
AND EXEMPLARY
NUMERICAL EXAMPLES
We have assumed that the \((nxn)\) matrix \(A\) has the property that its LU-decomposition can be done by Gaussian elimination without pivoting, which is true if \(A\) is a symmetric positive-definite, or an irreducible, diagonally dominant matrix.

Gaussian elimination can simply be described as a series of transformations that takes the matrix \(A\) into a triangular form given by

\[
N_{n-1}N_{n-2}\ldots N_1A = U ,
\]

where \(N_1\) is the transformation that eliminates the elements \(a_{jl}, 1 < j < n\).

We may introduce a series of similar transformations \(N'_1\) (see Hatzopoulos [HATZ74]), which alternately with the \(N_1\) transformations eliminate the elements \(a_{jl}, 1 < j < n\). Thus, the folding algorithmic concept can be visualized as a Gaussian forward elimination process carried out from the top of the matrix and a Gaussian backward elimination process performed from the bottom of the matrix.

The transformation matrices \(N_1\) and \(N'_1\) are given by

\[
N_1 = (n_{jk}) = \begin{cases} 
-a_{jk} & \text{for } k=1, k < j < n \\
-a_{kk} & \text{for } j=k \\
1 & \text{elsewhere} \\
0 & \text{elsewhere} 
\end{cases} \quad (A.C-VI:2)
\]

and

\[
N'_1 = (n'_{jk}) = \begin{cases} 
-a_{jk} & \text{for } k=1, 1 < j < n \\
-a_{kk} & \text{for } j=k \\
1 & \text{elsewhere} \\
0 & \text{elsewhere} 
\end{cases} \quad (A.C-VI:3)
\]

Let us suppose that \(n\) is odd (i.e., \(n=2m+1\)), then the elimination processes yield the following matrix identity:

\[
N'_{m+2}N'_{m+1}\ldots N'_{n-2}N_{n-1}N_1A = M , \quad (A.C-VI:4)
\]
where $M$ has the form:

\[
\begin{bmatrix}
\ddots \\
0 \\
0 \\
\end{bmatrix}
\]

middle row $\text{(A.C-VI:5)}$

In a similar manner, if $n$ is even (i.e., $n=2m$), the elimination processes when carried out produce the matrix identity:

\[
N_{m+2}N_{m-1} \cdots N_{n-1}N_nA = M', \quad (A.C-VI:6)
\]

where $M'$ has the form:

\[
\begin{bmatrix}
\ddots \\
0 \\
0 \\
\end{bmatrix}
\]

middle rows $\text{(A.C-VI:7)}$

For the solution of the transformed system with a coefficient matrix as in $\text{(A.C-VI:5)}$, a backward and a forward substitution process commencing at the $(n+1)/2$ diagonal element is required; whereas, for the system with a coefficient matrix as in $\text{(A.C-VI:7)}$, we first have to solve the resulting $(2\times2)$ central subsystem, before carrying out the necessary backward and forward substitution processes from the $n/2+1$ and $n/2$ diagonal elements, respectively.

Let us now consider a characteristic numerical example for the
case that \( n \) is even, since it literally covers the other instance.

**Example:**

Consider the \((4\times4)\) linear system:

\[
\begin{bmatrix}
2 & 3 & 4 & 1 \\
3 & 1 & 2 & 1 \\
2 & 1 & 3 & 1 \\
4 & 1 & 2 & 3
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix}
= 
\begin{bmatrix}
14 \\
9 \\
9 \\
14
\end{bmatrix},
\]

(A.C-VI:8)

the solution vector of which has been pre-arranged to be \( x=(1,2,1,2)^T \).

The matrix identity, given by \((A.C-VI:8)^\dagger\) produces:

\[\text{But, until the } \ldots N_m^{m+1} \text{ transformation stage. In this case the backward and forward substitution processes start from the } n/2 \text{ and } n/2+1 \text{ diagonal elements, respectively.}\]
\[
\begin{align*}
\begin{bmatrix}
\frac{N'_3}{a_{13}} & 0 & 0 & 0 \\
0 & \frac{a_{23}}{a_{33}} & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\frac{N'_2}{a_{14}} & 0 & 0 & 0 \\
0 & \frac{a_{24}}{a_{44}} & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\frac{N'_4}{a_{11}} & 1 & 0 & 0 \\
0 & \frac{a_{31}}{a_{11}} & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\frac{N_1}{a_{21}} & 1 & 0 & 0 \\
0 & \frac{a_{32}}{a_{22}} & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
2 & 3 & 4 & 1
\end{bmatrix}
&= 
\begin{bmatrix}
1 & 0 & 0 & 0 \\
-3/2 & 1 & 0 & 0 \\
-1 & 0 & 1 & 0 \\
-2 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & -1 \\
0 & 1 & 0 & 1/2 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
2 & 3 & 4 & 1 \\
0 & -7/2 & -4 & -1/2 \\
0 & -2 & -1 & 0 \\
0 & -5 & -6 & 1
\end{bmatrix}
&= 
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & -1/3 & 1 & 0 \\
0 & -5/6 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
2 & 8 & 10 & 0 \\
0 & 1 & 0 & 0 \\
0 & -1/3 & 1 & 0 \\
0 & -5/6 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
2 & 8 & 10 & 0 \\
0 & -6 & -7 & 0 \\
0 & -4/3 & 0 & 0 \\
0 & -1/6 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 & -15/2 & 0 \\
0 & 1 & 21/4 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
= 
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 8 & 0 & 0 \\
0 & 0 & 4/3 & 0 \\
0 & 0 & -1/6 & 1
\end{bmatrix}
\end{align*}
\]
whereas, for the r.h.s. vector, it correspondingly produces:

\[
\begin{bmatrix}
14 \\
9 \\
9 \\
14
\end{bmatrix} + \begin{bmatrix}
N_1 x b^{(1)}
\end{bmatrix} + \begin{bmatrix}
N_4 x b^{(1)}
\end{bmatrix} + \begin{bmatrix}
N_2 x b^{(2)}
\end{bmatrix} + \begin{bmatrix}
N_3 x b^{(3)}
\end{bmatrix} = \begin{bmatrix}
b^{(4)}
\end{bmatrix}
\]

Hence, the solution obtained from the final system:

\[
\begin{bmatrix}
2 & 8 & 0 & 0 \\
0 & -6 & 0 & 0 \\
0 & 0 & 4/3 & 0 \\
0 & 0 & -1/6 & 1
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix} = \begin{bmatrix}
18 \\
-12 \\
4/3 \\
11/6
\end{bmatrix}
\]

is

\[
-6x_2 = -12 \iff x_2 = 2
\]
\[
4/3x_3 = 4/3 \iff x_3 = 1
\]
\[
2x_1 + 8x_2 = 18 \iff x_1 = 1
\]
\[
-1/6x_3 + x_4 = 11/6 \iff x_4 = 2
\]

Finally, it is obvious that for a full general matrix the folding method requires more eliminations (i.e., more numerical operations) compared with the standard unidirectional Gaussian elimination. It is also interesting to note that the solution process for the folding method is effected from the center values of the matrix outwards, thus reducing
the growth of the rounding error in the forward and backward substitution processes.

In conclusion, the folding method is very suitable for pipelining on systolic arrays of banded matrices in particular, taking into consideration, however, the increase in the degree of difficulty to handle the opposite factorizing procedures in the center of the matrix.
Again, we shall distinguish two cases depending upon the value of \( n \). For both cases, however, because the matrix \( A \) has semi-bandwidths \( p=q=2 \), we must consider the following relations:

\[
\begin{align*}
\text{(k)} & \quad a_{i+1,j+1} = a_{i,j+1} \\
\text{(k)} & \quad a_{i,j+1} = a_{i+1,j+1} \\
\text{(k)} & \quad a_{i+1,j+1} = a_{i+1,j+1} + \text{ } \text{for } 1 \leq k \leq 1, \ i \geq 2 , \ (1^\text{st stream}) \quad \text{(A.C-VI:10)} \\
\end{align*}
\]

and

\[
\begin{align*}
\text{(k)} & \quad a_{i-1,j} = a_{i-1,j} \\
\text{(k)} & \quad a_{i,j-1} = a_{i,j-1} \\
\text{(k)} & \quad a_{i,j-1} = a_{i,j-1} + \text{ } \text{for } 1 \leq k \leq (n-1), \ (2^\text{nd stream}) \quad \text{(A.C-VI:11)} \\
\end{align*}
\]

Similarly, note that, \( k \) indicates computational steps and rows.

Let us now compute the corresponding elements of the resulting triangular matrices of Paradigms \([\text{VI.A.6:} \pi_1, \pi_2]\), respectively, using the recurrences \((\text{VI.B.1:3})\) in conjunction with the above relations.

**Paradigm \([\text{VI.A.6:} \pi_1]\)**

\( n \)-odd

1\(^{st}\) Stream

\[
\begin{align*}
\text{(k=1)} & \quad u_{1j} = a_{1j}^{(1)} \Rightarrow u_{11} = a_{11}^{(1)}, \ u_{12} = a_{12}^{(1)} \\
\text{(k=2)} & \quad u_{2j} = a_{2j}^{(2)} \Rightarrow u_{22} = a_{22}^{(2)}, \ u_{23} = a_{23}^{(2)} \\
\text{(k=3)} & \quad u_{3j} = a_{3j}^{(3)} \Rightarrow u_{33} = a_{33}^{(3)}, \ u_{34} = a_{34}^{(3)} \\
\text{(k=4)} & \quad u_{4j} = a_{4j}^{(4)} \Rightarrow u_{44} = a_{44}^{(4)}, \ u_{45} = a_{45}^{(4)} \\
\text{(k=5)} & \quad u_{5j} = a_{5j}^{(5)} \Rightarrow u_{55} = a_{55}^{(5)} \\
\end{align*}
\]

\((U_1)\)
(k=1) \rightarrow \ell_{11} = a_{11}^{(1)} u_{11}^{-1} \Rightarrow \ell_{21} = a_{21}^{(1)} u_{11}^{-1}

(k=2) \rightarrow \ell_{12} = a_{12}^{(2)} u_{22}^{-1} \Rightarrow \ell_{32} = a_{32}^{(2)} u_{22}^{-1}

(k=3) \rightarrow \ell_{13} = a_{13}^{(3)} u_{33}^{-1} \Rightarrow \ell_{43} = a_{43}^{(3)} u_{33}^{-1}

(k=4) \rightarrow \ell_{14} = a_{14}^{(4)} u_{44}^{-1} \Rightarrow \ell_{54} = a_{54}^{(4)} u_{44}^{-1}

\text{(L_1)}

\begin{align*}
\text{Since } p=q=2 \Rightarrow i=j \Rightarrow a_{ii}^{(k+1)} &= a_{ii}^{(k)} + \ell_{ii}^{(k+1)}(-u_{ii}) \text{, hence:} \\
\end{align*}

(2nd Stream)

(k=1) \rightarrow t=9 \rightarrow u_{9j} = a_{9j}^{(1)} \Rightarrow u_{99} = a_{99}^{(1)} , u_{88} = a_{88}^{(1)}

(k=2) \rightarrow t=8 \rightarrow u_{8j} = a_{8j}^{(2)} \Rightarrow u_{88} = a_{88}^{(2)} , u_{87} = a_{87}^{(2)}

(k=3) \rightarrow t=7 \rightarrow u_{7j} = a_{7j}^{(3)} \Rightarrow u_{77} = a_{77}^{(3)} , u_{76} = a_{76}^{(3)}

(k=4) \rightarrow t=6 \rightarrow u_{6j} = a_{6j}^{(4)} \Rightarrow u_{66} = a_{66}^{(4)} , u_{65} = a_{65}^{(4)}

(k=5) \rightarrow t=5 \rightarrow u_{5j} = a_{5j}^{(5)} \Rightarrow u_{55} = a_{55}^{(5)}

\text{(U_2)}

\begin{align*}
\end{align*}

(3rd Stream)

(k=1) \rightarrow t=9 \rightarrow \ell_{19} = a_{19}^{(1)} u_{99}^{-1} \Rightarrow \ell_{89} = a_{89}^{(1)} u_{99}^{-1}

(k=2) \rightarrow t=8 \rightarrow \ell_{18} = a_{18}^{(2)} u_{88}^{-1} \Rightarrow \ell_{78} = a_{78}^{(2)} u_{88}^{-1}

(k=3) \rightarrow t=7 \rightarrow \ell_{17} = a_{17}^{(3)} u_{77}^{-1} \Rightarrow \ell_{67} = a_{67}^{(3)} u_{77}^{-1}

(k=4) \rightarrow t=6 \rightarrow \ell_{16} = a_{16}^{(4)} u_{66}^{-1} \Rightarrow \ell_{56} = a_{56}^{(4)} u_{66}^{-1}

\text{(L_2)}

\begin{align*}
\end{align*}

\text{Since } p=q=2 \Rightarrow i=j \Rightarrow a_{ii}^{(k+1)} &= a_{ii}^{(k)} + \ell_{ii}^{(k)}(-u_{ii}) \text{, hence:}
\[(k=1) \Rightarrow t=9 \quad a_{11}^{(2)} = a_{11}^{(1)} + \ell_{19}(-u_{91}) \Rightarrow a_{88}^{(2)} = a_{88}^{(1)} + \ell_{89}(-u_{98})\]

\[(k=2) \Rightarrow t=8 \quad a_{11}^{(3)} = a_{11}^{(2)} + \ell_{18}(-u_{81}) \Rightarrow a_{77}^{(3)} = a_{77}^{(2)} + \ell_{78}(-u_{87})\]

\[(k=3) \Rightarrow t=7 \quad a_{11}^{(4)} = a_{11}^{(3)} + \ell_{17}(-u_{71}) \Rightarrow a_{66}^{(4)} = a_{66}^{(3)} + \ell_{67}(-u_{76})\]

\[(k=4) \Rightarrow t=6 \quad a_{11}^{(5)} = a_{11}^{(4)} + \ell_{16}(-u_{61}) \Rightarrow a_{55}^{(5)} = a_{55}^{(4)} + \ell_{56}(-u_{65})\]

Paradigm \[VI.A.\theta : \nu_{2}\]

\[n \text{-even} \]

\[1^\text{st} \text{Stream}\]

\[(k=1) \quad u_{11} = a_{11}^{(1)} \Rightarrow u_{11} = a_{11}^{(1)} , u_{12} = a_{12}^{(1)} \quad \{U_1\}\]

\[(k=2) \quad u_{22} = a_{22}^{(2)} \Rightarrow u_{22} = a_{22}^{(2)} , u_{23} = a_{23}^{(2)} \quad \{U_1\}\]

\[(k=3) \quad u_{33} = a_{33}^{(3)} \Rightarrow u_{33} = a_{33}^{(3)} \quad \{U_1\}\]

Again, since \(p=q=2 \Rightarrow i=j \Rightarrow a_{ik}^{(k+1)} = a_{ik}^{(k)} + \ell_{ik}(-u_{1k})\), hence:

\[(k=1) \quad a_{11}^{(2)} = a_{11}^{(1)} + \ell_{11}(-u_{11}) \Rightarrow a_{22}^{(2)} = a_{22}^{(1)} + \ell_{21}(-u_{12})\]

\[(k=2) \quad a_{11}^{(3)} = a_{11}^{(2)} + \ell_{12}(-u_{21}) \Rightarrow a_{33}^{(3)} = a_{33}^{(2)} + \ell_{32}(-u_{23})\]

\[2^\text{nd} \text{Stream}\]

\[(k=1) \Rightarrow t=4 \quad u_{44} = a_{44}^{(1)} \Rightarrow u_{44} = a_{44}^{(1)} , u_{43} = a_{43}^{(1)} \quad \{U_2\}\]

\[(k=2) \Rightarrow t=3 \quad u_{33} = a_{33}^{(2)} \Rightarrow u_{33} = a_{33}^{(2)} \quad \{U_2\}\]

\[(k=1) \Rightarrow t=4 \quad u_{44} = a_{44}^{(1)} + \ell_{44}(-u_{44}) \Rightarrow u_{44} = a_{44}^{(1)} + \ell_{44}(-u_{44}) \quad \{L_2\}\]

Since \(p=q=2 \Rightarrow i=j \Rightarrow a_{ik}^{(k+1)} = a_{ik}^{(k)} + \ell_{ik}(-u_{1i})\), hence:
Comment: When \( n \) is odd the center (twice modified) element, i.e., the element \( a_{55} \) in Paradigm [VI.A.6:p.1], remains in the same \( k \)th step after its first modification by the \( 1^{st} \) stream, since it will be modified by the \( 2^{nd} \) stream as for the first time, and so the \( k \) steps should agree with those given by the formulae for the \( l_{ij} \)'s and \( u_{ij} \)'s. In the case when \( n \) is even there is no such problem. \( \Box \)
NOTES - NOTES - NOTES