

MASSIVE PARALLEL COMPUTATION OVER GAMMA

By

SMITHA.H

Reg No. 71203405017

Of

Kumaraguru College Of Technology, Coimbatore 641006

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

*In partial fulfillment of the requirements
for the award of the degree*

Of

MASTER OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

ANNA UNIVERSITY CHENNAI 600 025

June 2005



P-1544

BONAFIDE CERTIFICATE

Certified that this project report entitled "**MASSIVE PARALLEL COMPUTATION OVER GAMMA**" is the bonafide work of **Ms. SMITHA.H**, who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

S. Dhanalakshmi
GUIDE

S. J. Jayaram
HEAD OF THE DEPARTMENT

The candidate with University Register No. **71203405017** was examined by us in the project Viva-Voce exam held on 25.6.05

R. Dhanalakshmi
INTERNAL EXAMINER

C. K. Ramakrishna
EXTERNAL EXAMINER 25/6/05

ACKNOWLEDGEMENT

I express my hearty gratitude to our beloved Correspondent, **Professor Dr. K. Arumugam, B.E. (Hons), M.S. (USA), M.I.E.**, for giving me this great opportunity to pursue this course.

I thank, **Dr. K.K. Padmanabhan, B.Sc. (Engg), M.Tech., Ph.D.**, Principal, Kumaraguru College of Technology, Coimbatore, for being a constant source of inspiration and providing me with the necessary facilities to work on this project.

I would like to make a special acknowledgement and thanks to **Dr.S.Thangasamy, Ph.D.**, Professor and Head of Department of Computer Science and Engineering, for his support and encouragement throughout the project.

I tender my special thanks to **Mr. R. Dinesh, M.S. (Wisconsin)**, Assistant Professor and Project Coordinator, Department of Computer Science and Engineering for his valuable suggestions.

I express my deep sense of gratitude and gratefulness to my guide **Mrs.Devaki.P, M.S.** Assistant Professor, Department of Computer Science and Engineering, for her supervision, tremendous patience, active involvement and guidance.

I extend my sincere thanks to **Ms. L.S. Jayashree, M.E., Ph.D.**, Senior Lecturer, Department of Computer Science and Engineering for her valuable suggestions and guidance.

I would like to convey my honest thanks to all **members of staff** of the Department for their unlimited enthusiasm, friendship and experience from which I have greatly benefited.

I owe special gratitude to my husband, **S.Vinod kumar**, without whom this project work would not have the seen daylight.

I express my profound gratitude to my **parents** and **friends** for their moral support.

Above all I thank the **Creator** of this beautiful planet for his grace throughout my endeavors.

TABLE OF CONTENTS

CHAPTER	CONTENTS	Page No
	ABSTRACT	iii
1.	INTRODUCTION	1
	1.1. Problem Definition	1
	1.2. The Current Status Of The Problem Taken Up	1
	1.3. Relevance And Importance Of The Topic	2
2.	Details Of Literature Survey	2
	2.1 Available Implementation schemes	2
3.	LINE OF ATTACK	4
4.	DETAILS OF METHODOLOGY EMPLOYED	5
	4.1 Cyclic Reduction Method Of Tridiagonal Systems	5
	4.2 Serial Cyclic Reduction (SCR) For Tridiagonal Systems	6
	4.3 Parallel Cyclic Reduction(PCR) For Tridiagonal Systems	7
	4.4 Cyclic Reduction Communication Pattern For 7 - Processor Case	7
	4.5 Backsubstitution	8
	4.6 An Overview Of Gamma	9
5.	CLUSTERING USING FOLLOWING TOOLS	10
	5.1 LAM	10
	5.2 PVM	12
	5.3 GAMMA	13
6.	PERFORMANCE EVALUATION	17
	6.1. Measurements And Performance Comparison	17
		17

6.1.2.2.	Speedup obtained	27
6.1.2.3.	Efficiency obtained	28
6.1.2.4.	Efficacy obtained	29
7.	CONCLUSION AND FUTURE OUTLOOK	30
8.	REFERENCES	30

SNO	LIST OF FIGURES	PAGE NO.
1.	Cyclic Reduction Stages When 7 Processors Are Used	8
2.	Backward Solve Communication When 7- Processors Are Used	8

SNO	LIST OF TABLES	PAGE NO
1.	Computation Time Involved	26
2.	Speedup Obtained	27
3.	Efficiency And Efficacy Obtained	28

SNO	LIST OF CHARTS	PAGE NO
1.	Computation Time Involved	27
2.	Speedup Obtained	28
3.	Efficiency Obtained	29
4.	Efficacy Obtained	29

SNO	SNAPSHOTS OF SCREENS	PAGE NO
1.	Parallel Implementation on a LAM cluster of 3 nodes	18
2.	Screen – 1: Booting up machines on the LAM cluster	18
3.	Screen 2: Extraction of Triplets in a LAM cluster	19
4.	Screen - 3: Back substitution and solution of odd rows in a LAM cluster	20
5.	Parallel Implementation on PVM	21
6.	Screen – 4: Extraction of Triplets in a PVM cluster	21
7.	Screen - 5: Extraction of Triplets (Contd..)	22
8.	Screen - 6: Back substitution and solution of odd rows in a PVM cluster	23
9.	Parallel Implementation on a GAMMA cluster	24
10.	Screen - 7: Extraction of Triplets in a GAMMA cluster	24
11.	Screen - 8: Back substitution and solution of odd rows in a GAMMA cluster	25

LIST OF ABBREVIATIONS

1	MPI	Message – Passing Interface
2	NOWs	Network Of Workstations
3	MPPs	Massively Parallel Machines
4	LAM	Local Area Multi-Computer
5	GAMMA	Genoa Active Message Machine
6	LAN	Local Area Networks
7	PVM	Parallel Virtual Machine
8	IPC	Inter-Process Communication

ABSTRACT

This paper presents the implementation of a Fast Linear Solver for Tridiagonal systems using Cyclic Reduction Method on GAMMA clusters and provides with performance results with and without the enhancement through GAMMA. Tridiagonal systems often occur in the simulation of physical problems. The objective of the paper was to reduce the time of computation to less than $1/n$ the time needed for serial implementation when n number of processors are used

Among the direct and indirect methods available for solving a Tridiagonal system, Cyclic Reduction architecture is the most appropriate one for solving Tridiagonal systems.

Cyclic reduction is an algorithm for the direct solution of linear equations with tridiagonal property. Cyclic reduction algorithm is a modification of recursive doubling which avoids computational redundancy.

Basic idea of cyclic reduction is to halve the dimension of the equation system repeatedly until a single equation with a single unknown is left. This equation is solved and the previously eliminated unknowns are found by a backward substitution

The cost of high performance parallel platforms prevents parallel processing techniques from spreading in present applications. Indeed standard network protocols and mechanisms cannot deliver a satisfactory amount of communication performances of the raw hardware to applications. NOW parallel

architecture potentially enjoys several advantages. On the other hand, the cost of workstations and high-speed LAN devices is constantly decreasing, thus making this approach very appealing from the financial point of view. This advantage cannot be enjoyed by the commercial high-performance parallel platforms. The negative aspects of the NOW approach are related to the performance of the communication mechanisms usually made available at the application level. The use of traditional protocols for LANs yields extremely high latency and very low message throughput.

GAMMA is an attempt to overcome such a limitation by adopting a minimal communication protocol and Active Message communication paradigm. Gamma yields a communication latency time that is unbeaten by any NOW-based prototype leveraging comparable hardware, together with a good exploitation of the communication bandwidth of the raw hardware.

With the promising facilities provided by GAMMA, our implementation of a tridiagonal solver had outperformed the record produced in a NOW with the traditional protocols. Running GAMMA communications on a dedicated interconnect can boost performance of GAMMA parallel jobs, because separating GAMMA communications from the remaining network traffic allows a number of optimizations in the low-level messaging protocol

கருத்துச் சுருக்கம்

இவ்வாய்வில் மும்முலைவிட்டங்களுள்ள ஒழுங்கு முறைகளை GAMMA தொகுப்பின் மீதான சுழற்சி சுருக்க முறை மூலம் விரைவு வரிகளுக்குரிய வழிகளால் GAMMA மிகைப்படுத்துதலின்றி நல்ல செயல்முடிவுகள் கிடைக்குமாறு தீர்வு காணப்படுகிறது. மும்முலைவிட்ட ஒழுங்குமுறை போலியான வெளிப்புற இடையூறுகளில் அதிகம் நேரிடுகிறது. இவ்வாய்வின் நோக்கமானது கணக்கிடல்நேரத்தை, $1/n$ வரிசைக்கிரமமான நிறைவேற்றுதலின் நேரத்தைவிட குறைப்பதாகும் (n - செயல்படுத்திகளின் எண்ணிக்கை).

மும்முலைவிட்டங்களுள்ள ஒழுங்குமுறைகளைத் தீர்ப்பதற்காக உள்ள நேரடி மற்றும் மறைமுக வழிமுறைகளில் சுழற்சி சுருக்கமுறை கட்டமைப்பே மிகப் பொருத்தமானதாகும். சுழற்சி சுருக்கமுறை மும்முலைவிட்ட பண்புடைய ஒருபடிச் சமன்பாட்டிற்கான நேரடித் தீர்வு செய்நெறிப்படி ஆகும்.

சுழற்சி சுருக்கமுறை மிகுதியான கணக்கிடல் நேரத்தைத் தவிர்க்கும் இரட்டிப்பு திரும்ப நிகழ்வு முறையைத் தழுவியதாகும்.

சுழற்சி சுருக்க முறையின் அடிப்படை உள்கருத்தானது சமன்பாட்டின் பரிமாணத்தை தொடர்ந்து ஒருசமன்பாட்டில் ஒரு குணகம் மட்டும் அறியாதது வரை பாதியாகக் குறைத்துக் கொண்டே வருவதாகும். இச்சமன்பாடு தீர்க்கப்பட்டு முந்தைய தவிர்க்கப்பட்ட அறியாத குணகங்கள் பின்னோக்கு ஈடுமுறை மூலம் காணப்படுகிறது.

இணையொத்த தளங்கள் மிகையான செயல்பாட்டை வெளிப்படுத்தினாலும் அதற்கான அதிக செலவுத் தொகை நடைமுறையில் செயல்படுத்தப்படுவதற்கு தடையாக உள்ளது. வலைத் தொடர்புகளுக்கான விதிமுறை மற்றும் வழிமுறைகள் திருப்திகரமான தொடர்பு செயல்பாட்டை வன்பொருள் மூலம் செயல்படுத்த முடியாது. இணையொத்த கட்டமைப்பின் உள்ளியல்பு பல நற்பயன்களை அளிக்கிறது. அதிவேக LAN சாதனத்திற்கான செலவு குறைந்துகொண்டே வருவதால் இவ்வழிமுறை அதிகமாக செயல்படுத்தப்படுகிறது.

தளத்துடன் தொடர்புடையதாகும். மரபு விதிமுறைகளை LANகளுக்கு உபயோகிப்பதால் மிகவும் குறைந்த வெளியீடு கிடைக்கிறது.

GAMMA வழிமுறை இத்தகைய வரைமுறை குறைந்த தொடர்பு விதிமுறைகள் மற்றும் AMC (Active Message Communication) மூலம் மீறுவதற்கான ஒரு முயற்சியாகும். MPI , GAMMA உள்ளடக்கிய இணையொத்த செயல்பாட்டை MPI / GAMMA மூலம் ஆதரிக்கிறது. NOW அடிப்படையிலான விதிமுறைகளின் தொடர்பு தாமதிக்கும் நேரத்தைவிட சிறந்த தொடர்பு தாமதிக்கும் நேரத்தை GAMMA வெளிப்படுத்துகிறது. NOW வழிமுறையின் வெளியீட்டைவிட மும்முலைவிட்டங்களுக்கான GAMMA முறையின் செயல் முறை அதிகம் சிறந்ததாகிறது. GAMMA தொடர்பு முறை இணையொத்த GAMMA செயல்முறைகளின் செயல்படுமுறையை ஊக்குவிக்கிறது.

CHAPTER 1

INTRODUCTION

Solving tridiagonal systems is one of the key issues of numerical simulations in many scientific and engineering problems. Solving a linear system $Ax = b$ requires more computational time due to the communication overhead involved in Parallel computing systems. To reduce the communication overhead, GAMMA is used

A tridiagonal system is a linear system of equations

$$Ax = d$$

Where x and d are n -dimensional vectors and $a = [a_i, b_i, c_i]$, is a tridiagonal matrix with dimension n . Tridiagonal systems are of the form

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = F_i, i = 1, \dots, n$$

1.1. Problem definition:

Solving a Tridiagonal system on top of GAMMA to achieve massive parallelism

1.2. The current status of the problem taken up:

Parallel tridiagonal algorithms have been studied extensively and remain an active research area. Most parallel tridiagonal solvers trade computation with parallelism.

1.3. Relevance and Importance of the Topic

Large tridiagonal systems of linear equations appear in many numerical

simulations. The efficient line relaxations needed by robust multigrid

communication paradigm is being used to enhance the performance of a tridiagonal solver. For simplicity, we assume in this paper that each processor has roughly the same number of subsequent rows of the tridiagonal system, and the number of processors N_p is strictly less than the number of unknowns N .

CHAPTER 2

DETAILS OF LITERATURE SURVEY

2.1. Available Implementation schemes:

Tridiagonal systems can be implemented using:

1. Direct Methods:

- 1.1. LU decomposition
- 1.2. Successive Doubling
- 1.3. Recursive Doubling and
- 1.4. Parallel Cyclic Reduction methods

2. Iterative type Methods:

- 2.1. Jacobi method
- 2.2. Gauss-Seidel

Guassian elimination (or) Thomas algorithm:

On a serial computer, Guassian elimination without pivoting can be used to solve a tridiagonal system of linear equations in $O(N)$ steps. This algorithm is referred to as Thomas algorithm therefore, it is inherently serial in the sense that its communication has a complexity of $O(N_p)$. Unfortunately, the algorithm is not well suited for parallel computers. The Thomas algorithm is

performs the process of elimination from the beginning each time a new system has to be solved, also if the coefficient matrix remains constant. If the whole solution of a single system is considered, Thomas algorithm can't compete with methods, which perform a factorization of the coefficient matrix when the solution of a large number of linear systems with only different right-hand sides has to be performed.

Recursive Doubling:

Stone introduced his Recursive-doubling algorithm. Both Cyclic reduction and Recursive doubling was designed for fine-grained parallelism.

Partitioning algorithm:

Sun et al introduced the parallel partitioning LU algorithm that is very similar to the Bondeli's divide and conquer algorithm. For both partitioning algorithms and divide and conquer algorithms, a reduced tridiagonal system of interface equations must be solved. Each processor owns only a small number of rows in this reduced system. As an example, in Wang's partitioning algorithm each processor owns one row of the reduced system. Recursive doubling solves this reduced system. However, numerical experiments were performed only on very small number of processors.

Wang introduced a new partitioning algorithm. The basic idea is that a tridiagonal interface of N_p linear equations is generated without communication. Each processor owns one equation of this interface system. After solving the interface system of equations a back substitution step generates the solution.

Divide and Conquer algorithm

The divide and conquer algorithm introduced by Bondeli consists of inverting the tridiagonal systems that reside on each processor, by disregarding the connections to neighboring processors. Similar to the partitioning algorithms, this also yields a interface system of equations that must be solved in parallel, for example by cyclic reduction. The storage requirement for this divide and conquer algorithm is about twice the storage of a tridiagonal system.

Jacobi and Gauss-Seidel Algorithms

Iterations in both algorithms should be slightly faster because the algorithm loops through every elements in the matrix.

CHAPTER 3

LINE OF ATTACK

3.1. Actual Approach: Cyclic Reduction Method:

The first parallel algorithm for the solution of tridiagonal systems was developed by Hockney and Golub. It is now usually referred to as Cyclic Reduction.

We focus heavily on the Cyclic Reduction schemes because this method outperforms conventional techniques for parallel implementations, and is the most appropriate one for solving tridiagonal systems and from the point of view of integration in VLSI technology, is the one which uses the least amount of area and the smallest number of pins. Cyclic Reduction method has been the most successful method. Cyclic reduction requires $2 \cdot \log_2 Np$ steps of nearest neighbor communication. Additionally storage requirements can be held to a minimum by overwriting the original tridiagonal system with all

We assume that each processor has a unique index or rank.

CHAPTER 4

DETAILS OF METHODOLOGY EMPLOYED

4.1. CYCLIC REDUCTION METHOD OF TRIDIAGONAL SYSTEMS:

Cyclic Reduction (CR) was first used to solve tridiagonal equations, arising from the finite-difference approximation to Poisson's equation. This particular case was limited to problem size given by powers of 2. CR was chosen over other algorithms such as Gaussian elimination, because it deals with periodic boundary conditions in a much neater way, eliminating the need for the calculation of auxiliary vectors.

Linear equations are combined to eliminate the odd numbered unknowns $x_1, x_3, x_5, \dots, x_n$. Reorder the unknowns and repeat the process until we reach a single equation with one unknown. March backward to obtain rest of unknowns.

Considering the case of $n=7 = 2^3 - 1$ unknowns or which we have three triplets. Start by forming the first triplet from the first three equations. Multiply by parameters $\alpha_2, \beta_2,$ and γ_2 to get

$$\alpha_2 b_1 x_1 + \alpha_2 c_1 x_2 = \alpha_2 F_1$$

$$\beta_2 a_2 x_1 + \beta_2 b_2 x_2 + \beta_2 c_2 x_3 = \beta_2 F_2$$

$$\gamma_2 a_3 x_2 + \gamma_2 b_3 x_3 + \gamma_2 c_3 x_4 = \gamma_2 F_3$$

To eliminate x_1 and x_3 , we add the equations and choose

$$\beta_2 = 1$$

$$\alpha_2 b_1 + \beta_2 a_2 = 0$$

$$\beta_2 c_2 + \gamma_2 b_3 = 0$$

Resulting in $(\alpha_2 c_1 + \beta_2 b_2 + \gamma_2 a_1)x_2 + \gamma_2 c_3 x_4 = \alpha_2 F_1 + \beta_2 F_2 + \gamma_2 F_3$

Similarly, combining the third, fourth and fifth equations obtained from equation (1), we form the second triplet, from which we obtain

$$\underbrace{a_4 a_3 x_2}_{\Phi_4} + \underbrace{(\alpha_4 c_3 + \beta_4 b_4 + \alpha_4 a_5)}_{\omega_2} x_4 + \underbrace{\gamma_4 c_5}_{\sigma_2} x_6 = \underbrace{a_4 F_3 + \beta_4 F_4 + \gamma_4 F_5}_{\hat{a}_2}$$

And $\alpha_4, \beta_4, \gamma_4$ are determined from

$$\beta_4 = 1$$

$$\alpha_4 b_3 + \beta_4 a_4 = 0$$

$$\beta_4 c_4 + \gamma_4 b_5 = 0$$

Finally, for the third triplet we obtain, as before, the only surviving equation

$$\Phi_6 x_4 + \omega_6 x_6 = \hat{a}_6$$

Where the parameters $\alpha_6, \beta_6, \gamma_6$ involved in the definition of $\Phi_6, \omega_6, \hat{a}_6$ are determined by solving

$$\alpha_6 b_5 + \beta_6 a_6 = 0$$

$$\beta_6 c_6 + \gamma_6 b_7 = 0$$

We see that the three resulting equations also form a tridiagonal system, that is

$$\omega_2 x_2 + \sigma_2 x_4 = \hat{a}_2 \quad (2)$$

$$\Phi_4 x_2 + \omega_4 x_4 + \sigma_4 x_6 = \hat{a}_4 \quad (3)$$

$$\Phi_6 x_4 + \omega_6 x_6 = \hat{a}_6 \quad (4)$$

Repeat the same elimination process as before, i.e., Multiply those equations by α_4', β_4' and γ_4' respectively and choose

$$\alpha_4' \omega_2 + \beta_4' \Phi_4 = 0$$

$$\beta_4' \sigma_4 + \gamma_4' \omega_6 = 0$$

This leads to only one equation:

$$\alpha_4^* x_4 = F_4^*$$

Using Back Substitution, after we obtain x_4 from the preceding equation, we can compute x_2 from the reduced equation (2) and x_6 from equation (3). Finally, we use the original equations to obtain x_1, x_3, x_5 , and x_7 .

4.2. SERIAL CYCLIC REDUCTION (SCR) FOR TRIDIAGONAL SYSTEMS:

Serial Cyclic Reduction has higher computational complexity and

leading to a poor balance. Serial cyclic reduction for tridiagonal systems includes Memory allocation and generation of the tridiagonal system, cyclic reduction is done on the system and Back substitution is done to recover the solution. Serial cyclic reduction for tridiagonal systems is rarely used because it is (1) More expensive than standard LU decomposition (2) Complex indexing needed

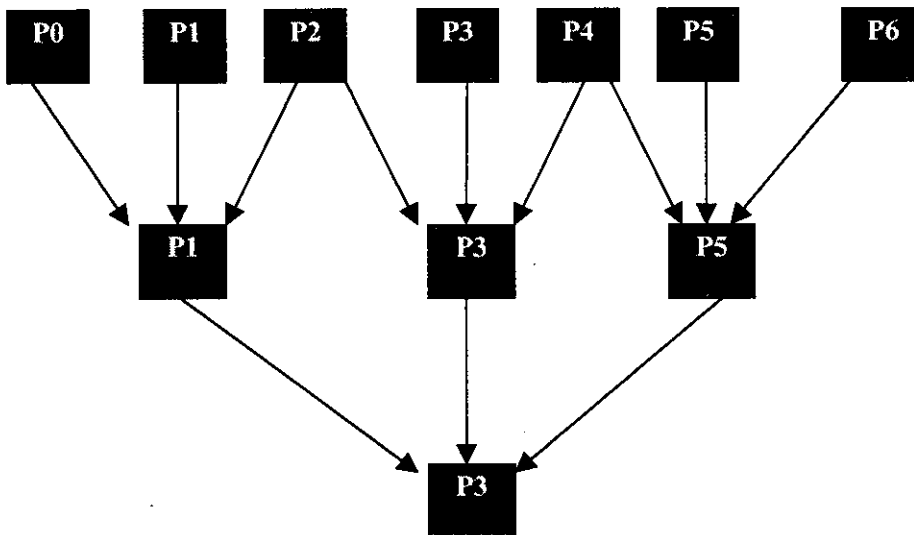
4.3. PARALLEL CYCLIC REDUCTION (PCR) FOR TRIDIAGONAL SYSTEMS

The idea relies on a novel data distribution scheme, which reduces the amount of serial computation involved per processors and minimizes the communication overhead. One of the major problems with the PCR algorithm is its inability to maintain maximum CPU utilization due to frequent waits for non-local data particularly for higher or lower terms in equation, delaying the overall computation times and hampering its scalability under strict parallel execution. Parallel implementation for tridiagonal system includes the MPI initialization, memory allocation and generation of the tridiagonal system. The process is followed by cyclic reduction of the tridiagonal system and Back substitution to obtain the solution which is done in parallel in the available machines in the cluster.

4.4. CYCLIC REDUCTION COMMUNICATION PATTERN FOR 7 - PROCESSOR CASE

When n is the number of processors used in the parallel processing, $2n-1$ is the number of rows of the tridiagonal system

Figure 1: Cyclic Reduction stages when 7 processors are used



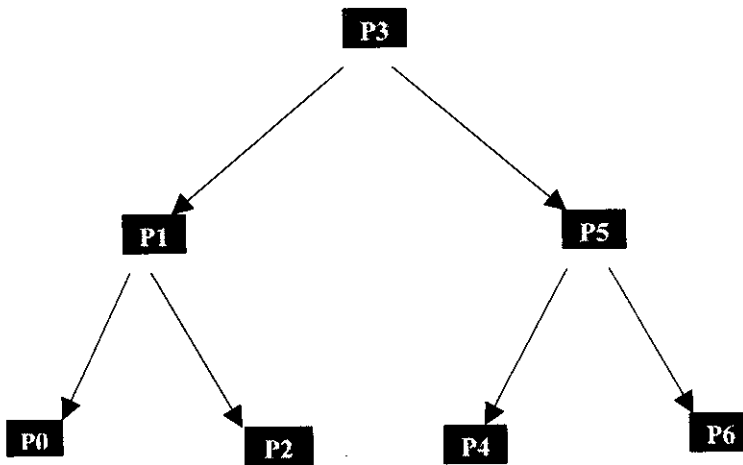
Processors P0, P1, P2 operate simultaneously on the first 3 rows of the matrix namely rows 1, 2, 3 and pass on the reduced row to processor P1. Processors P2, P3, P4 operate simultaneously on the next 3 rows of the matrix namely rows 3, 4, and 5 and pass on the reduced row to Processor P3. Processors P4, P5, P6 operate simultaneously on the next 3 rows of the matrix, namely rows 5, 6, 7 and pass on the reduced row to P5. Processors P1, P3, P5, process the three rows and the reduced row is passed on to the processor P3.

4.5. BACKSUBSTITUTION

Processor P3 processes the reduced row and obtains the value of the unknown variable and passes the result to P1 and P5. P1 and P5 would process the set of rows and give the result to P0, P2, P4 and P6

The result would show a clear advantage of available processors of this method, due to an alternate data distributions among processors that increases the overlap between communications and execution, increasing the

Figure 2: Backward Solve communication when 7- processors are used



4.6. AN OVERVIEW OF GAMMA:

A high performance Beowulf (PC cluster) machine installed with Linux OS and MPI for interprocessor communications has been constructed using Fast Ethernet and the communication software GAMMA. Each unit of the PC cluster machine, sometimes called a Beowulf cluster machine, has a scalar-type processing unit and its own memory. Thus, high performance computation comparable to a supercomputer is made possible at orders of magnitude lower cost. Scalability of application programs on the number of processors and reliability over a long period of time (days) has also been confirmed for the GAMMA communications.

The GAMMA system is based on the active message mechanism that enables direct communications between the application program and the network interface while bypassing the operating system. Nice features of the present method is that all the software including the Linux operating system and GAMMA software are either free or low priced, and that the hardware including processors, gigabit Ethernet cards and switching hubs are reasonably priced commercial products. On the other hand, for obtaining high

first network is a gigabit Ethernet for the GAMMA data transmissions, and the second one is a TCP/IP network for the NFS file system and general administration purposes.

The computational speed increases nearly linearly up to four processors, and improves gradually beyond that. The computation speed by GAMMA communications is about 1.5 times that by TCP/IP. It is important to use a low latency communication system for the computational speed to scale linearly with the number of processors ($P \ll 1$), since the communication overhead constitutes a non-parallelizable part.

Benefits of GAMMA include:

Supports both single and dual CPU processing nodes

Runs on Fast Ethernet and Gigabit Ethernet NICs

Compatible with standard network protocols and services (4) Good programmability due to fairly high abstraction level (5) Reliable due to mechanisms for retransmission of missing packets

Drawback of a standard NOW/cluster architecture is the poor performance of its support to inter-process communication over any LAN hardware.

Current implementations of industry-standard communication primitives (RPC), APIs (sockets), and protocols (TCP, UDP) usually show high communication latencies and low communication throughput.



CHAPTER 5

CLUSTERING WITH LAM, PVM, GAMMA

5.1. PC Clusters using LAM:

The most popular and cost-effective approach to parallel computing is cluster computing based on PCs running the Linux operating system. The effectiveness of this approach depends on the communication network connecting the PCs together, which may vary from fast Ethernet to Myrinet that can broadcast messages at a rate of several gigabits per second. (Gbps).

In addition to enhancements in the speed of individual processors, there have been several key developments that have enabled commodity supercomputing.

Cluster configuration involves changes in few configuration files and installing few applications, which allows running a network of workstations into a supercomputer cluster.

Following are the steps required for the configuration of the cluster:

Configuring `/etc/hosts` files

Configuring `/etc/hosts.equiv` files

Configuring rsh, login and other services

Configuring NFS server

Installing LAM/MPI

Configuring `/etc/hosts` files:

This file contains IP addresses and host names for local network as well as any other systems. Any network program on the system consults this file to determine the IP addresses that corresponds to a host name.

Configuring `/etc/hosts.equiv` files:

The `hosts.equiv` file lists the hosts and the users that are trusted by the local host when a connection is made using the `rshd` service. The `hosts.equiv` file is found in the `/etc` directory and lists the remote names that may connect to the local machine and the local user names those machines may connect as.

Each line of the file as follows:

Hostname [username]

Where, `hostname` may be given as a hostname, an address indicating that all the hosts are considered to be trusted.

Username (optional) specifies a username on a remote machine. When username has been specified only the users with entries on the specified hosts may log in the local machine. When username, is not specified then any user that has the same username on both the remote and local machine may log into the local network. This configuration is must for a cluster to boot.

Setting rsh, rlogin and other services

Remote Shell (rsh) executes command on the specified hostname. To do so, it must connect to the rshd service on the hostname.

Rsh usually sends two usernames to the rshd daemon-remote user and local user. Remote user is the username that is currently logged into the client machine. This user is the name that must appear in the /etc/hosts.equiv file. Local user is the name that the service or daemon uses to execute the command on the server. Rsh daemon is used by the application LAM/MPI for booting and checking the parallel environment making it the most important service for the cluster.

rlogin:

rlogin starts a terminal session on the remote host specified as host. The remote host must be running the rlogind daemon for rlogin to connect to. rlogin uses the /etc/hosts.equiv file for authorization. When no username is specified, rlogin connects as the user.

5.2. PC Clusters using PVM:

PVM does not require special privileges to be installed. PVM uses two environment variables when starting and running. Each PVM user needs to set these two variables to use PVM. The first variable is PVM_ROOT, which is set to the location of the installed pvm3 directroy. The second variable is PVM_ARCH, which tells PVM the architecture of this host and thus what executables to pickfrom the PVM_ROOT directory. These two variables can be set in the .cshrc file.

```
Setenv PVM_ROOT $HOME/pvm3
```

PVM_ARCH is set by concatenating to the file .cshrc the content of the \$PVM_ROOT/lib/cshrc.stub. The stub should be placed after PATH and PVM_ROOT are defined. This stub automatically determines the PVM_ARCH for this host and is particularly useful when the user shares a common file system across different architectures. Building for each architecture type is done automatically by logging on to a host, going into the PVM_ROOT directory,and typing make. The makefile will automatically determine which

\$PVM_ROOT/lib/PVM_ARCH, with the exception of the pvmgs which is placed in \$PVM_ROOT/bin/PVM_ARCH.

Set up summary:

Set PVM_ROOT and PVM_ARCH in the .cshrc file

Build PVM for each architecture type

Create a .rhosts file on each host listing all the hosts

Create a \$HOME/.xpvms_hosts file listing all the hosts

Starting PVM:

On the hosts where PVM is installed type

```
% pvm
```

So that the PVM is now running on the host.

Now hosts are added to the virtual machine by typing at the console

prompt

```
Pvm> add hostname
```

To see the present virtual machinetype

```
Pvm> conf
```

To see the pvm tasks that are running on the virtual machine,type

```
Pvm> ps -a
```

To finish the virtual machine, type

```
Pvm> halt
```

To run the program tridiag_solver on a cluster of 3 machine, type

spawn at the prompt

```
pvm> spawn -3 -> tridiag_solver
```

5.3. GAMMA CLUSTERING

GAMMA Installation:

Requirements:

User Requirements:

Hardware / Software Requirements:

A pool of personal computers (PCs)

3COM 3c905

A single crossover cable for a "back-to-back" connection

Linux kernel version 2.4.21

A good C compiler, and C libraries, gcc2.95.3 works successfully

Assumptions about file placement:

Pathname of the Linux source tree is `/usr/src/linux/`.

Configuration of GAMMA source code:

Gamma-7.----.tar.gz is used

Unpack the GAMMA source code:

Gamma.tar.gz is placed in a directory. Command `tar xvzf gamma.tar.gz` is invoked. This operation creates a subdirectory named `gamma`, containing the GAMMA software.

Configuring the GAMMA source code:

Entering the subdirectory `gamma/` and run the script named `configure`. The script creates a number of symbolic links to the appropriate files containing source code.

Enabling collision-safe transmission has a negative impact on communication performance but greatly decreases the rate of missing packets in case of congestion.

With "safe transmission" enabled, the GAMMA protocol will be forced to test for successful transmission of packets previously enqueued to the NIC transmission queue before attempting to enqueue a new packet. This makes GAMMA slightly less performing but much more stable.

For best performance we disable the IP traffic allowed on LAN feature. because pure polling leads to better performance, especially with MPI/GAMMA. GAMMA uses signals internally, to occasionally force a process to consume its pending messages.

Pathname of the Linux source tree for compilation in GAMMA enhanced Linux kernel is set as `/usr/src/linux/`.

Once compiled, the directory to install the GAMMA user library is set as `/usr/lib/`. The directory to install the GAMMA utilities for startup and recovery is set as `/usr/local/bin/`.

TUNING SOME GAMMA PARAMETERS:

NIC-specific settings:

All GAMMA parameters are defined as constants in file `gamma.nic.dependent.h`

Protocol-specific settings:

GAMMA parameters which affect the behaviour of GAMMA independent of the particular NIC in use are defined as constants in file `gamma_def.h`

Maximum cluster size:

Maximum allowed cluster size is defined by the constant `MAX_NUM_NODES` in file `lib/gamma_userlev.h`

CONFIGURATION OF LINUX KERNEL FOR USE WITH GAMMA:

Enter the directory `/usr/src/linux/`.

Type:

```
make config
```

A script starts which will prompt the following questions concerning Linux kernel configuration:

CCM -> Enter on each PC of your

Make config script asks the following question:

3c590/3c900 series (592/595/597) " Vortex / Boomerang" support
(CONFIG.VORTEX) [N/y/m/?]

Answer "N" is given to prevent the 3c59x.c Linux network driver from OS on your 3COM adapter, letting the GAMMA driver operate safely.

Building dependencies in the Linux source tree:

After the command make config has run to termination, run the command make dep to build dependencies in the Linux source tree. This operation may take quite a while.

COMPILING AND INSTALLING GAMMA:

After configuration and tuning, and after the Linux kernel has been configured correctly, the GAMMA source code is compiled and installed, i.e., integration of the GAMMA source code into the Linux kernel; Now Linux kernel after having installed GAMMA is recompiled. Entering into subdirectory gamma/, we compile GAMMA by invoking make. Now GAMMA is installed by invoking make install.

Setting up the environment

GAMMA configuration file

GAMMA needs a configuration file called /etc/gamma.conf, containing global information about the cluster. More precisely, /etc/gamma.conf must contain the mapping between each PC hostname and the MAC address of the NIC operating GAMMA communications. The same configuration file must be placed in all PCs of the cluster. The host names is excerpted from files /etc/HOSTNAME on each PC in the cluster

GAMMA startup and recovery utilities:

Enter the directory /usr/local/bin/ and copy all the GAMMA utilities on

CHAPTER 6

Performance Evaluation:

Performance Evaluation is a necessity in systems, which are too complex to test and validate and indeed build. Performance evaluation is an active area of interest especially within the parallel systems community. Performance evaluation strategies have been proposed and utilized to allow analysis to take place at various stages of application code development. A system's performance is parameterized in terms of the underlying operations. These can include a characterization of the disk accesses, the communication traffic on the interconnection network in a parallel system, and can describe the use of the functional units of a processor.

Performance Evaluation is usually performed through three general methodologies. These are performed at different stages of the planning, the design, the construction of the system, and the operation of the finished product. The three techniques are: measurements, simulation and analytical modeling.

6.1. MEASUREMENTS AND PERFORMANCE COMPARISON:

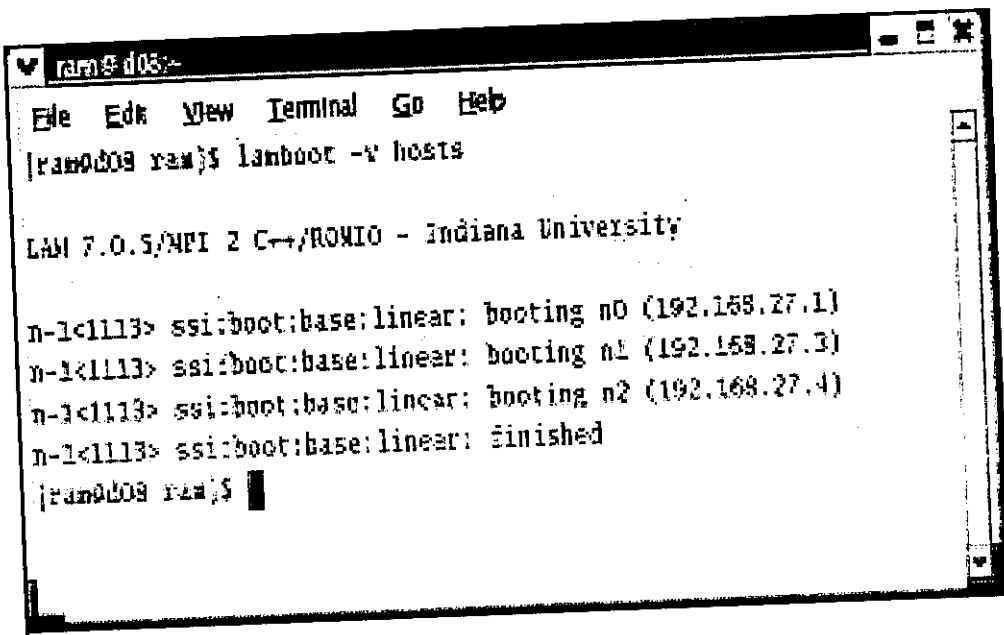
With the promising facilities provided by GAMMA, our implementation of a tridiagonal solver had outperformed the record produced in a NOW with the traditional protocols. We provide some preliminary results obtained from an experimental implementation on NOW.

6.1.1. The Measurement Test bed:

To get the figures for GAMMA and TCP and PVM, dedicated environments were created and the implementation of the tridiagonal solver is done on all the three environments.

SCREEN SNAPSHOTS OF IMPLEMENTATION

Parallel Implementation on a LAM cluster of 3 nodes



```
ram@d03:~  
File Edit View Terminal Go Help  
[ram@d03 ram]$ lambboot -v hosts  
  
LAM 7.0.5/NPI 2 C++/ROMIO - Indiana University  
  
n-1<1113> ssi:boot:base:linear: booting n0 (192.168.27.1)  
n-1<1113> ssi:boot:base:linear: booting n1 (192.168.27.3)  
n-1<1113> ssi:boot:base:linear: booting n2 (192.168.27.4)  
n-1<1113> ssi:boot:base:linear: finished  
[ram@d03 ram]$
```

Screen – 1: Booting up machines on the LAM cluster


```

ran@03:~$ ./smitha/ctest1 3-2-05
File Edit View Terminal Go Help

-----
Backsubstitution proceeds .....

-----
Solving for odd rows proceeds .....

-----
Tri-diagonal matrix using Cyclic Reduction Method
-----
s = 0      0  0  0  0  0  1  2  3
s = 1      0  0  0  0  0  1  2  3
s = 2      0  0  0  0  0  1  2  3
s = 3      0  0  0  0  0  1  2  3
s = 4      0  0  0  0  0  1  2  3
s = 5      0  0  0  0  0  1  2  3
s = 6      0  0  0  0  0  1  2  3
-----
r values
-----
r(0) = 7
r(1) = -3
r(2) = -20
r(3) = -31
r(4) = -26
r(5) = -22
r(6) = -34

*** Elapsed time: 5 secs ***

[ran@03 3-2-05]$

```

Screen - 3: Back substitution and solution of odd rows in a LAM cluster

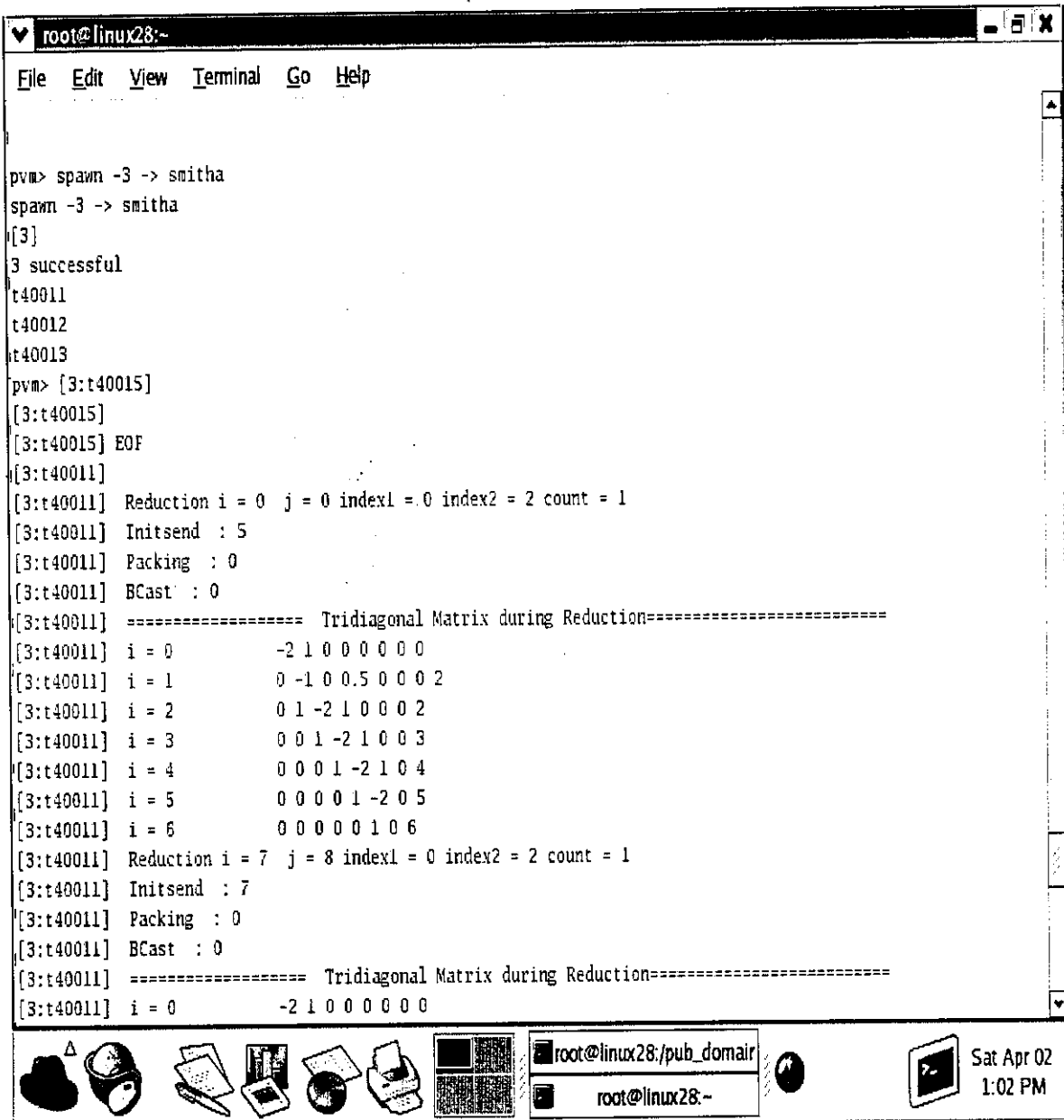
Parallel Implementation on PVM

```

root@linux28:~
File Edit View Terminal Go Help

pvm> spawn -3 -> smitha
spawn -3 -> smitha
[3]
3 successful
t40011
t40012
t40013
pvm> [3:t40015]
[3:t40015]
[3:t40015] EOF
[3:t40011]
[3:t40011] Reduction i = 0 j = 0 index1 = 0 index2 = 2 count = 1
[3:t40011] Initsend : 5
[3:t40011] Packing : 0
[3:t40011] BCast : 0
[3:t40011] ===== Tridiagonal Matrix during Reduction=====
[3:t40011] i = 0      -2 1 0 0 0 0 0
[3:t40011] i = 1      0 -1 0 0.5 0 0 0 2
[3:t40011] i = 2      0 1 -2 1 0 0 0 2
[3:t40011] i = 3      0 0 1 -2 1 0 0 3
[3:t40011] i = 4      0 0 0 1 -2 1 0 4
[3:t40011] i = 5      0 0 0 0 1 -2 0 5
[3:t40011] i = 6      0 0 0 0 0 1 0 6
[3:t40011] Reduction i = 7 j = 8 index1 = 0 index2 = 2 count = 1
[3:t40011] Initsend : 7
[3:t40011] Packing : 0
[3:t40011] BCast : 0
[3:t40011] ===== Tridiagonal Matrix during Reduction=====
[3:t40011] i = 0      -2 1 0 0 0 0 0

```



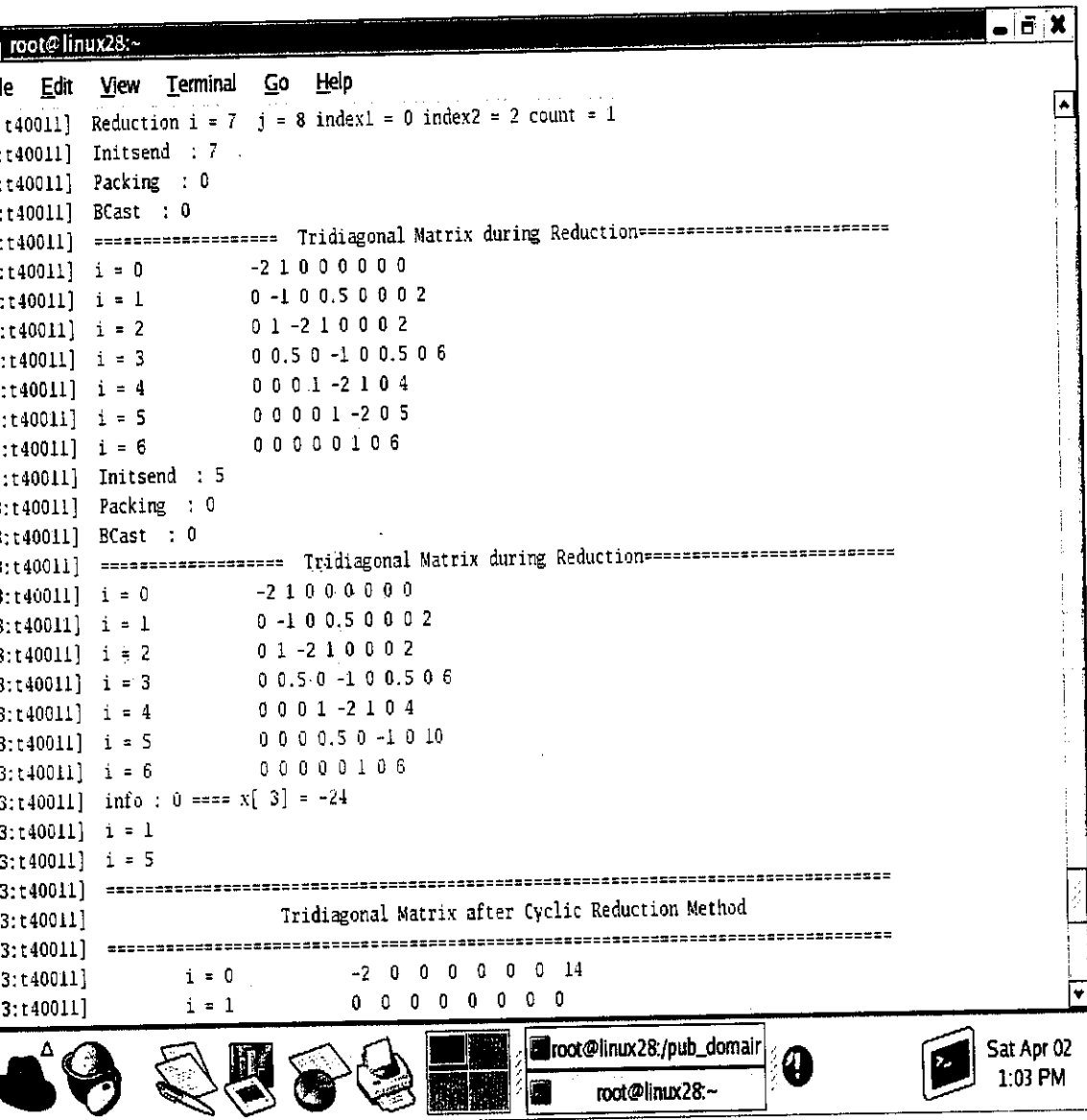
The terminal window displays the execution of a PVM cluster. It shows the spawning of three processes (t40011, t40012, t40013) and the execution of a reduction operation. The output includes the reduction parameters (i, j, index1, index2, count) and the resulting tridiagonal matrix for each process. The matrices are shown for i=0 and i=7, with the first row of each matrix being: [-2, 1, 0, 0, 0, 0, 0].

Screen – 4: Extraction of Triplets in a PVM cluster


```

root@linux28:~
le Edit View Terminal Go Help
:t40011] Reduction i = 7 j = 8 index1 = 0 index2 = 2 count = 1
:t40011] Initsend : 7
:t40011] Packing : 0
:t40011] BCast : 0
:t40011] ===== Tridiagonal Matrix during Reduction=====
:t40011] i = 0      -2 1 0 0 0 0 0
:t40011] i = 1      0 -1 0 0.5 0 0 0 2
:t40011] i = 2      0 1 -2 1 0 0 0 2
:t40011] i = 3      0 0.5 0 -1 0 0.5 0 6
:t40011] i = 4      0 0 0 1 -2 1 0 4
:t40011] i = 5      0 0 0 0 1 -2 0 5
:t40011] i = 6      0 0 0 0 0 1 0 6
:t40011] Initsend : 5
:t40011] Packing : 0
:t40011] BCast : 0
:t40011] ===== Tridiagonal Matrix during Reduction=====
:t40011] i = 0      -2 1 0 0 0 0 0
:t40011] i = 1      0 -1 0 0.5 0 0 0 2
:t40011] i = 2      0 1 -2 1 0 0 0 2
:t40011] i = 3      0 0.5 0 -1 0 0.5 0 6
:t40011] i = 4      0 0 0 1 -2 1 0 4
:t40011] i = 5      0 0 0 0.5 0 -1 0 10
:t40011] i = 6      0 0 0 0 0 1 0 6
:t40011] info : 0 ==== x[ 3] = -24
:t40011] i = 1
:t40011] i = 5
:t40011] =====
:t40011] Tridiagonal Matrix after Cyclic Reduction Method
:t40011] =====
:t40011] i = 0      -2 0 0 0 0 0 0 14
:t40011] i = 1      0 0 0 0 0 0 0 0

```



Screen - 5: Extraction of Triplets (Contd..)

```

root@linux28:~
File Edit View Terminal Go Help
[2:t4000b] =====
[2:t4000b]                      Tridiagonal Matrix after Cyclic Reduction Method
[2:t4000b] =====
[2:t4000b]      i = 0          0 0 0 0 0 0 0 0
[2:t4000b]      i = 1          0 0 0 0 0 0 0 0
[2:t4000b]      i = 2          0 0 0 0 0 0 0 0
[2:t4000b]      i = 3          0 0 0 0 0 0 0 0
[2:t4000b]      i = 4          0 0 0 0 0 0 0 0
[2:t4000b]      i = 5          0 0 0 0 0 0 0 0
[2:t4000b]      i = 6          0 0 0 0 0 0 0 28
[2:t4000b] =====
[2:t4000b]                      X Values
[2:t4000b] =====
[2:t4000b]                      x[0] = -7
[2:t4000b]                      x[1] = -14
[2:t4000b]                      x[2] = -20
[2:t4000b]                      x[3] = -24
[2:t4000b]                      x[4] = -25
[2:t4000b]                      x[5] = -22
[2:t4000b]                      x[6] = 22
[2:t4000b]                      Elapsed time : 3 secs
[2:t4000b] EOF

```

Screen - 6: Back substitution and solution of odd rows in a PVM cluster

PARALLEL IMPLEMENTATION ON GAMMA CLUSTER

```

gamma@netix gamma
File Edit View Terminal Go Help

[gamma@netix gamma]$ mpicc c2mc7.c -o c2mc7 -lm
[gamma@netix gamma]$ mpirun N ./c2mc7

gamma_init_argv(): argc 6: /home/gamma/c2mc7| |NI| |GAMMANP| |1|
gamma_init_argv(): first call, 3 instances
gamma_init(): created virtual GAMMA #3 with nodes:
0:1->netix
0:1->mtech17
0:1->mtech11

=====
Tridiagonal matrix before manipulation
=====
i = 0      -2  1  0  0  0  0  0  0
i = 1       1 -2  1  0  0  0  0  1
i = 2       0  1 -2  1  0  0  0  2
i = 3       0  0  1 -2  1  0  0  3
i = 4       0  0  0  1 -2  1  0  4
i = 5       0  0  0  0  1 -2  1  5
i = 6       0  0  0  0  0  1 -2  6
=====

Initial set of triplets
=====
i = 0      -2 X0 + 1 X1 = 0
i = 1      + 1 X0 -2 X1 + 1 X2 = 1
i = 2      + 1 X1 -2 X2 + 1 X3 = 2
i = 3      + 1 X2 -2 X3 + 1 X4 = 3
i = 4      + 1 X3 -2 X4 + 1 X5 = 4
i = 5      + 1 X4 -2 X5 + 1 X6 = 5
i = 6      + 1 X5 -2 X6 = 6
=====

```

Screen - 7: Extraction of Triplets in a GAMMA cluster

```

gamma@netix gamma
File Edit View Terminal Go Help
=====
Backsubstitution proceeds .....
=====
Solving for odd rows proceeds .....
=====
Tridiagonal matrix after Cyclic Reduction Method
=====
i = 0      0  0  0  0  0  0  0  0
i = 1      0  0  0  0  0  0  0  0
i = 2      0  0  0  0  0  0  0  0
i = 3      0  0  0  0  0  0  0  0
i = 4      0  0  0  0  0  0  0  0
i = 5      0  0  0  0  0  0  0  0
i = 6      0  0  0  0  0  0  0  0
=====
X values
=====
x[0] = -7
x[1] = -14
x[2] = -20
x[3] = -24
x[4] = -25
x[5] = -22
x[6] = -14

**** Elapsed time : 0.153886 secs****

[gamma@netix gamma]$

```

Screen - 8: Back substitution and solution of odd rows in a GAMMA cluster

The focus lies on measuring speedup, efficiency and efficacy. For all measurements, three machines were used. For GAMMA, interrupts and TCP/IP communication were disabled.

6.1.2. Performance Results:

6.1.2.1. Computation Time Needed For Processing A Tridiagonal Solver

As given in the graph, the computation time for the sequential execution of the tridiagonal solver comes upto 9secs, the parallel implementation of the solver on a LAM cluster of three machines, consumes about 5.25secs. This time consumption is due to the communication overhead involved in the parallel computation. The communication overhead can be fully avoided by the usage of GAMMA. The graph shows the computation time of 0.1559sec for the solver when executed on a GAMMA cluster of three machines. This proves the Active Message Paradigm.

Table 1: Computational Time involved

Execution on	Computation time
Single machine	9
LAM cluster (3 machines)	5.25
PVM cluster (3 nodes)	3
GAMMA cluster (3 machines)	0.155986

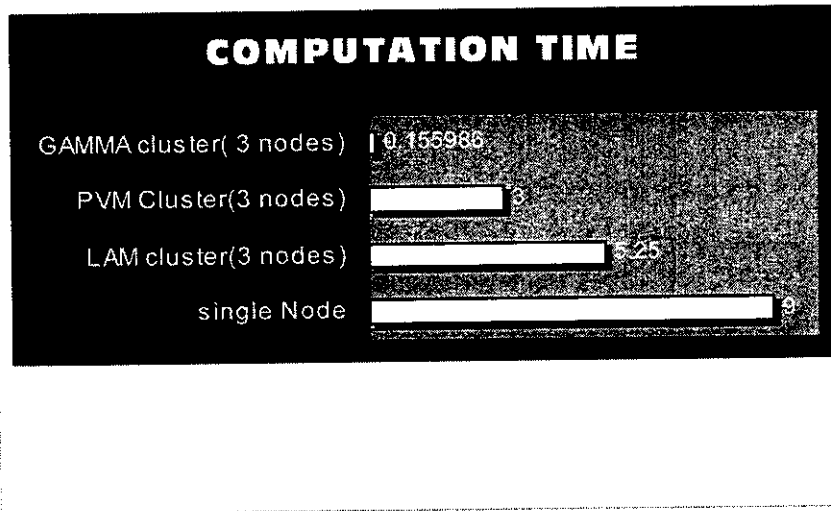


Chart 1: Computation speed

6.1.2.2. Speedup Obtained:

Speedup is a measure of how much faster the programs run

Execution on	Speedup
LAM cluster (3 nodes)	1.71
PVM cluster (3 nodes)	3
GAMMA cluster (3 nodes)	57.6974

Table 2: Speedup obtained

Speedup Obtained

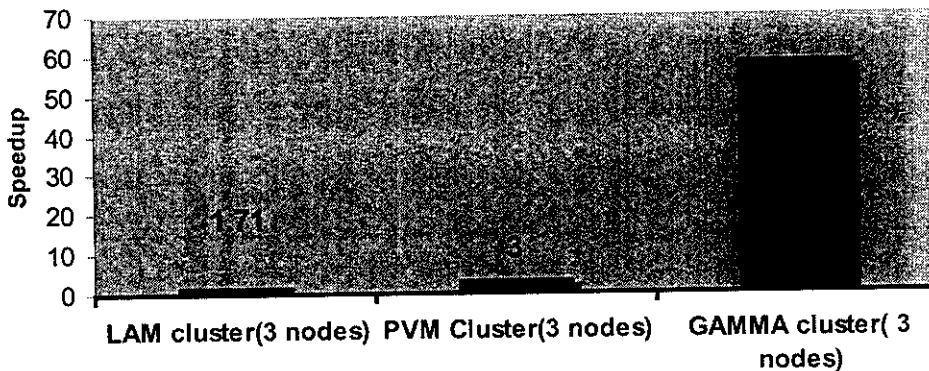


Chart 2: Speedup obtained

on a parallel machine than it does on a serial machine.

Clusters with higher computation time will produce lower speedups. A Poor speedup results due to the decomposition, inefficiently parallelized section of the code and computation overhead.

6.1.2.3. Efficiency:

Efficiency gives the average contribution of the processors towards the global computation. With GAMMA cluster, the efficiency comes upto 19.232, which is a major contribution GAMMA can offer to the world of parallel computation.

Results obtained on	LAM cluster	PVM cluster	GAMMA cluster
Efficiency	0.57	1	19.232
Efficacy	0.9747	3	1109.66

Table 3: Efficiency and Efficacy obtained

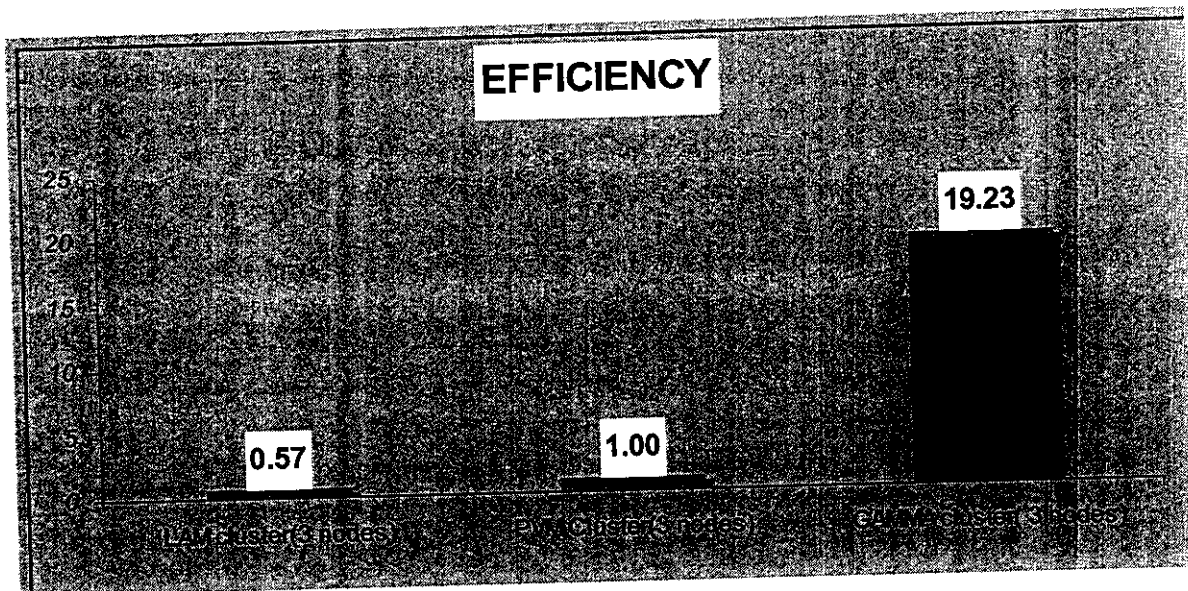
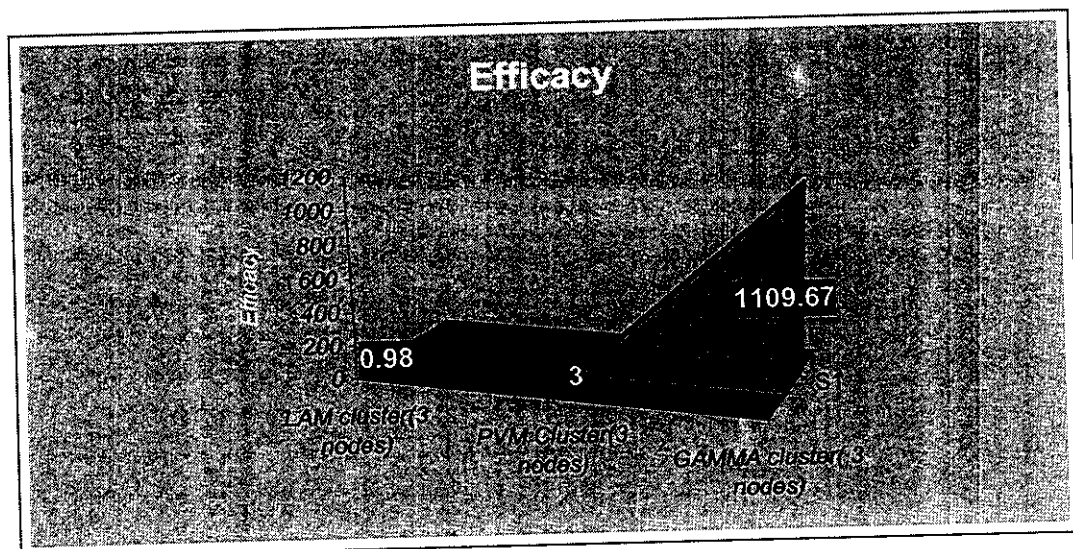


Chart 3: Efficiency obtained

6.1.2.4. Efficacy:

An important property of the Efficacy curve when plotted as a function of number of processors is that its first maximum corresponds to an optimal system operating point.



When a processor can be added to the computation with a resulting increase in efficacy, then the gain of the extra processor out measures the cost of adding it. When on contrary, the efficacy diminishes, then the cost of the addition outweighs the potential performs. In the efficacy graph, the GAMMA cluster with three nodes shows the maximum efficacy.

This type of analysis is very important for algorithms that will run on distributed memory machines, where locality and communication costs will play a major role in efficiency

7. CONCLUSION:

Many discretization methods for the solution of partial differential equations for the modeling of physical phenomena result in large sparse systems of equations. Cyclic reduction can be applied to solve these systems. The method has a similar sequential execution time as other discrete methods, but provides enough parallelism for an efficient parallel execution. Method can be efficiently implemented for message-passing programming models by organizing the computations and communications such that only neighbor communications are necessary. Message passing leads to better reliable runtime behavior. And a significant improvement in performance is given by parallel model when executed over GAMMA.

REFERENCES:

- [1] Giuseppe Ciaccio ., How to install GAMMA: the Genoa Active Message Machine.,
- [2] G.Chiola, G.Ciaccio., Implementing a low cost, low latency parallel platform.
- [3] G.Chiola,Giuseppe Ciaccio ., Porting and Measuring the Linpack Benchmark on GAMMA.,

- [4] X.-H.Sun, H.Zhang and L.Ni, Efficient tridiagonal solvers on multicomputers, IEEE Trans.Comput, 41(3)(1992), 286-296
- [5] H.Wang, A Parallel method for tridiagonal equations, ACM Trans, Math Software, 7(1981), 170-153G.Chiola and G.Ciaccio. GAMMA home page, <http://www.disi.unige.it/project/gamma/>.
- [6] G.Chiola and G.Ciaccio. Efficient Parallel processing on Low-cost clusters with GAMMA Active ports, Parallel Computing, (26): 333-354, 2000
- [7] Giuseppe Ciaccio. Messaging on Gigabit Ethernet: Some Experiments with GAMMA and other systems. International Parallel and Distributed Processing Symposium (IPDPS '01), 1530-2075, 2001.
- [8] Cyclic reduction on Distributed Shared memory., Sebastian Allman, Thomas Rauber, Gudula Runger, IEEE. Ninth Euromicro Workshop on Parallel and Distributed Processing(EUROPDP'01) 1066 – 6192/01 2001

BOOKS

- [1] George Em Karniadakis., Robert M.Kirby II., Parallel scientific computing in C++ and MPI – A Seamless approach to parallel algorithms & their implementations
- [2] Peter S.Pacheco ., Parallel Programming with MPI.

WEBSITE FOR THE SOFTWARE

- [1] <http://www.netlib.org/pvm3/index.html/pvm3.3.9.tar.gz>
- [2] www.disi.unige.it/project/gamma/gamma-04-08-13.tar.gz
- [3] www.mpi-softech.com/mpich-1.1.2.tar.gz

Serial Implementation of the Tridiagonal Solver

```
#include <iostream.h>
#include <time.h>
#include <conio.h>
class test
{
    public: double n,p;
           int start, end, trip_no, no_triplets, add_count, no_of_term,
           copy_n,no_nodes;
           int no_unknowns,count_of_triplets,count,found,
           a[3],exponent;
           double x[20],LHS;
           struct node
           {
               double coef;
               int exp;
               int flag;
               node *next;
           } *cpos, *cpos1, *poly[10], *head[10], *curr[10],
           *mul_head[5][10], *head1[10], *curr1[10], *temp1, *curr_pos,
           *added_triplet[30], *cur[10][10],*copy_triplet[20],*copy[20];
           void get_no_proc();
           void calc_n();
           void disp_n();
           void init_poly();
           void init_copy_triplet();
           void create_poly();
           void copy_poly();
           void disp_poly();
           void disp_copy_poly();
           void form_triplet();
           void disp_mult_head();
```

```

void add_triplet();
void disp_added_triplet();
void reduce_added_triplet();
void disp_reduced_triplet();
void rearrange_added_triplet();
void eliminate_zero();
void eliminate_last();
void eliminate_odd_entry();
int check_odd(int);
void back_solve();
void back_solve1();
void arrange_solved();
void displ();
void copy_reduce_added(int);
};
void test::get_no_proc()
{
    cout<<"Enter the number of processors : ";
    cin>>p;
}
void test::calc_n()
{
    n = pow(2,p) - 1;
    no_unknowns = n;
    copy_n = n;
}
void test::disp_n()
{
    cout<<"n = "<<n<<endl;
    {
        poly[i] = new node();
        head[i] = new node();
        head1[i] = new node();
        copy[i] = new node();
    }
}

```

```

        mul_head[1][i] = new node();
        mul_head[2][i] = new node();
        mul_head[3][i] = new node();
        curr[i] = new node();
        curr1[i] = new node();
        poly[i]->next = NULL;
        added_triplet[i] = new node();
        x[i] = -9999;
    }
}

void test::init_copy_triplet()
{
    for(int i=0;i<=20;i++)
    {
        copy_triplet[i] = new node();
        copy_triplet[i]->next = NULL;
    }
}

void test::create_poly()
{
    int c,e=0;
    node *temp1,*temp2;
    for(int i=1;i<=n;i++)
    {
        head[i] = poly[i];
        curr[i] = head[i];
        copy_triplet[count_of_triplets] = poly[i];
        count_of_triplets++;
        for(int j=1;j<=3;j++)
        {
            cout<<"Enter the coefficient : ";
            cin>>c;
            temp1 = new node();
            temp1->coef = c;

```

```

        temp1->exp = e;
        temp1->flag = 0;
        temp1->next = NULL;
        curr[i]->next = temp1;
        curr[i] = curr[i]->next;
        e++;
    }
    temp2 = new node();
    cout<<"Enter value of right hand side : ";
    cin>>temp2->coef;
    cout<<exp = 0;
    temp2->flag = 1;
    temp2->next = NULL;
    curr[i]->next = temp2;
    e = e-2;
}
}
void test::disp_poly()
{
    for(int i=1;i<=n;i++)
    {
        curr[i] = head[i]->next;
        while(curr[i]->next != NULL)
        {
            cout<<coef<<"x"<<exp<<" + ";
            curr[i] = curr[i]->next;
        }
        cout<<" = ";
        cout<<coef;
        cout<<=end;i++)
    {
        curr[i] = head[i]->next;
        curr1[i] = mul_head[trip_no][i];
        cout<<"Enter the value of a["<<i<<"]";
    }
}

```

```

cin>>a[i];
while(curr[i]->flag != 1)
{
    curr1[i]->coef = curr[i]->coef * a[i];
    curr1[i]->exp = curr[i]->exp;
    curr1[i]->flag = curr[i]->flag;
    curr[i] = curr[i]->next;
    temp1 = new node();
    temp1->coef = curr[i]->coef;
    temp1->exp = curr[i]->exp;
    temp1->flag = 0;
    temp1->next = NULL;
    curr1[i]->next = temp1;
    curr1[i] = curr1[i]->next;
}
curr1[i]->coef = curr1[i]->coef * a[i];
curr1[i]->exp = 0;
curr1[i]->flag = 1;
curr1[i]->next = NULL;
}
}

```

```
void test::add_triplet()
```

```

{
    node *temp3;
    add_count=1;
    int s,last;
    cpos = added_triplet[trip_no];
    for(int i=start;i<=end;i++)
    {
        cpos1 = mul_head[trip_no][i];
        while(cpos1 != NULL)
        {
            temp3 = new node();
            temp3->coef = cpos1->coef;

```



```

        temp3->exp = cpos1->exp;
        temp3->flag = cpos1->flag;
        temp3->next = NULL;
        cpos->next = temp3;
        cpos1 = cpos1->next;
        cpos = cpos->next;
        add_count++;
    }
}
cpos->next = NULL;
}
void test::disp_added_triplet()
{
    cout<<endl<<"Added Triplet is "<<=trip_no-1;i++)
    {
        cout<<"\t\t"<next;
        while(cpos != NULL)
        {
            cout<coef<<"x"<exp<<" + ";
            cpos = cpos->next;
        }
    }
    cout<<=trip_no-1;i++)
    {
        cpos = added_triplet[i];
        int tot = 0;
        while(cpos->next != NULL)
        {
            if(cpos->next->flag == 1)
            {
                tot = tot + cpos->next->coef;
                cpos->next = cpos->next->next;
                add_count--;
            }
        }
    }
}

```

```

    }
    if(cpos->next == NULL)
    {
        temp = new node();
        temp->coef = tot;
        temp->exp = 0;
        temp->flag = 1;
        temp->next = NULL;
        cpos->next = temp;
    }
    cpos = cpos->next;
}
}
}
void test::disp_reduced_triplet()
{
    cout<<endl<<"Reduced Triplet is "<<trip_no-1;i++)
    {
        cout<<next;
        copy_reduce_added(i);
        count_of_triplets++;
        while(cpos->next != NULL)
        {
            cout<<coef<<"x"<<exp<<" + ";
            cpos = cpos->next;
        }

        cout<<" = "<<coef;
        cpos->next = NULL;
    }
}
}
void test::copy_reduce_added(int i)
{
    node *temp,*curr,*curr1;

```

```

int check_coef,check_exp;
curr = added_triplet[i]->next;
curr1 = copy_triplet[count_of_triplets];
while(curr != NULL)
{
    temp = new node();
    temp->coef = curr->coef;
    check_coef = temp->coef;
    temp->exp = curr->exp;
    check_exp = temp->exp;
    temp->next = NULL;
    curr1->next = temp;
    curr1 = curr1->next;
    curr = curr->next;
}
}

```



```

void test::reduce_added_triplet()

```

```

{
    int tot_coef;
    for(int i=1;i<= trip_no-1;i++)
    {
        cpos = added_triplet[i];
        while(cpos->next->next->next != NULL)
        {
            if(cpos->next->exp == cpos->next->next->next->exp)
            {
                cpos1 = cpos->next->next;
                tot_coef = cpos->next->coef + cpos1->next->coef;
                cpos->next->coef = tot_coef;
                cpos1->next = cpos1->next->next;
            }
            if(cpos->next->exp == cpos->next->next->exp)
            {
                cpos1 = cpos->next;

```

```

        tot_coef = cpos->next->coef + cpos1->next->coef;
        cpos1->coef = tot_coef;
        cpos1->next = cpos1->next->next;
    }
    cpos = cpos->next;
}
}
for(int m=1;m<=trip_no-1;m++)
{
    cpos = added_triplet[m];
    for(int n=1;n<=3;n++)
    {
        if(cpos->next->exp == cpos->next->next->next->exp)
        {
            cpos1 = cpos->next->next;
            tot_coef = cpos->next->coef + cpos1->next->coef;
            cpos->next->coef = tot_coef;
            cpos1->next = cpos1->next->next;
        }
        cpos = cpos->next;
    }
}
}

```

```

void test::eliminate_last()

```

```

{
    node *temp;
    for(int i=1;i<=n;i++)
    {
        cpos = poly[i];
        for(int j=1;j<=4;j++)
        {
            if(cpos->next->exp == (n+1))
            {
                temp = cpos->next;
            }
        }
    }
}

```

```

        temp = cpos->next->next;
        temp= cpos->next;
        cpos->next = cpos->next->next;
        delete temp;
    }
    cpos = cpos->next;
}
}
}
void test::eliminate_zero()
{
    cpos = added_triplet[1];
    if(cpos->next->exp == 0)
        cpos->next = cpos->next->next;
}
void test::eliminate_odd_entry()
{
    cpos = added_triplet[1];
    head[1] = added_triplet[1];
    no_of_term = 1;
    while((cpos->next->next != NULL)&&(cpos->next != NULL))
    {
        if(check_odd(no_of_term) == 1)
        {
            if(cpos->next->flag != 1)
            {
                cpos->next= cpos->next->next;
                no_of_term++;
            }
        }
        no_of_term++;
        cpos = cpos->next;
    }
    copy_reduce_added(1);
}

```

```

        count_of_triplets++;

for(int i=2;i<=trip_no-1;i++)
{
    cpos = added_triplet[i];
    head[i] = added_triplet[i];
    no_of_term = 0;
    while((cpos->next->next != NULL)&&(cpos->next != NULL))
    {
        if(check_odd(no_of_term) == 1)
        {
            if(cpos->next->flag != 1)
            {
                cpos->next= cpos->next->next;
                no_of_term++;
            }
        }
        no_of_term++;
        cpos = cpos->next;
    }
    copy_reduce_added(i);
    count_of_triplets++;
}
}

int test::check_odd(int term)
{
    if(term%2 == 1)
        return 1;
    else
        return 0;
}

void test::back_solve()
{
    cpos = copy_triplet[count_of_triplets];

```

```

    exponent = cpos->next->exp;
    x[exponent] = cpos->next->next->coef / cpos->next->coef;
    found++;
}
void test::back_solve1()
{
    cpos = copy_triplet[count_of_triplets];
    exponent = cpos->next->exp;
    x[exponent] = cpos->next->next->coef / cpos->next->coef;
    copy_triplet[count_of_triplets] = NULL;
    found++;
}
void test::arrange_solved()
{
    while(found < copy_n)
    {
        int i = 1;
        while(i <= copy_n)
        {
            count_of_triplets = count_of_triplets-1;

            if(count_of_triplets <= 0)
            {
                count_of_triplets = copy_n+1;
                exponent = 0;
            }

            cpos = copy_triplet[count_of_triplets];
            if(copy_triplet[count_of_triplets] != NULL)
            {
                for(int j=1;j<=copy_n;j++)
                {
                    if(x[j] > -9999)
                        exponent = j;
                }
            }
        }
    }
}

```

```

        cpos = copy_triplet[count_of_triplets];
        if(exponent != 0)
        {
while((cpos->next->exp != exponent)&&(cpos != NULL))
            {
                cpos = cpos-
>next;
            }
            if(cpos !=NULL)
            {
cpos->next->coef = cpos->next->coef * x[exponent];
                LHS = cpos->next->coef;
                cpos->next = cpos->next->next;
do{
                cpos = cpos->next;
}while(cpos->next != NULL);
if(cpos->next == NULL)
{
                cpos->coef = cpos->coef - LHS;
                i = copy_n;
            }
        }
    }
    }
    i++;

```

```

cpos = copy_triplet[count_of_triplets];
int count = 0;
do{
        cpos = cpos->next;
        count++;
}while(cpos->next !=NULL);
if(count == 1)
        cpos = NULL;
if(count == 2) back_solve1();

```



```

        }
    }
}
cpos = copy_triplet[count_of_triplets];

}
void test::displ()
{
    clrscr();
    int i=1;
    do
    {
        cout<<"\t"<<endl<<" i = "<<i<<" ";
        cpos = copy_triplet[i];
        do{
            cout<next->coef<<"x"<next->exp<<"+";
            cpos = cpos->next;
        }while(cpos->next != NULL);
        cout<<endl<<= count_of_triplets);
    }
}
int main()
{
    time_t first, second;
    test t1;
    int no_iterat,odd;
    clrscr();
    t1.count_of_triplets=1;
    t1.get_no_proc();
    t1.calc_n();
    t1.init_poly();
    t1.init_copy_triplet();
    t1.create_poly();
}

```

```

first = time(NULL); /* Gets system time */
t1.no_nodes = 4;
t1.eliminate_last();
cout<<endl<<"Tridiagonal system"<<=t1.no_triplets;count++)
    {
        t1.form_triplet();
        t1.add_triplet();
        t1.trip_no++;
    }
t1.disp_added_triplet();
t1.rearrange_added_triplet();
t1.reduce_added_triplet();
t1.eliminate_zero();
t1.disp_reduced_triplet();
t1.no_of_term = 1;
t1.eliminate_odd_entry();
t1.n = t1.no_triplets;
t1.no_nodes--;
}while(t1.no_triplets > 1);
t1.found = 0;
t1.count_of_triplets--;
t1.back_solve();
cout<<endl<<"Display copy of triplets"<< t1.no_unknowns);
cout<<endl<<"Result"<<=t1.found;i++)
    cout<<endl<<"x"<<i<<" = "<<<<endl<<"The difference is:
"<<difftime(second,first)<<"seconds";
}

```

MPI Program to solve a Tridiagonal system on a LAM cluster

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <math.h>
#include <time.h>
#define size1 2000
int main(int argc, char *argv[])
{
    int no_processes;
                                // Number of Processes involved...
    int my_rank;
                                // Process Rank...
    int size;
                                // Size of the Tridiagonal Matrix...
    int numrows;
    int i,j,k,index , index1 , index2;
    int numactivep, activep[20];
                                // Number of Active Processes...
    double alpha,gamma;
    time_t first,second;
    first = time(NULL);
                                // Assigns the time to the variable first...
    MPI_Init(&argc,&argv);
                                //Initializes for the MPI environment
    MPI_Comm_size(MPI_COMM_WORLD,&no_processes);
                                // Returns the number of processes...
    MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
                                // Returns the process number(rank)...
    MPI_Status status;
```



```

for(a=0;a<=size1;a++)
    count = (index1+index2)/2;
    printf("\n Triplet : ");
    for( k=0 ; k<=size ; k++)
    {
        A[count][k] -= alpha*A[index1][k]+
        gamma*A[index2][k];
        // Reducing the Triplets ...
        if(k!=size)
        {
            if(A[count][k] > 0)
            {
                printf(" + %gX%d
                ",A[count][k] , k);
            }
            else if(A[count][k] < 0)
            {
                printf(" %gX%d ",A[count][k] , k);
            }
        }
        if(k == size)
        printf(" = %gX%d ",A[count][k] , k);
    }
}

MPI_Bcast(&A,100,MPI_DOUBLE ,j , MPI_COMM_WORLD);
    // sends the updated matrix to all other processors
    within the communicator...

MPI_Barrier(MPI_COMM_WORLD);
    // blocks the calling process until all group
    processes have called the function...

}

numactivep = 0;
for( j=activep[1] ; j< size+1 ; j++)
{

```

```

        printf(" %g ",A[i][j]);
    }
}
}
MPI_Barrier(MPI_COMM_WORLD);
    // blocks the calling process until all group
    processes have called the function...
/*----- BACK SUBSTITUTION -----*/
    printf("\nBacksubstitution\n");
    for(j=0 ; j< size+1 ; j++)
    {
        printf(" %g ",A[i][j]);
    }
}
}
MPI_Barrier(MPI_COMM_WORLD);
    // blocks the calling process until all group
    processes have called the function...
for(process_rank=0 ; process_rank <=no_processes ;
process_rank++)
{
    MPI_Barrier(MPI_COMM_WORLD);
        // blocks the calling process until all group
        processes have called the function...
    if(my_rank == process_rank)
    {
        for(i=0 ; ik ; found--)
        {
            if(x_value_found[found] == 1)proceed = 1;
                else proceed = 0;
        }
        if(proceed == 1)
        {
            x[k] = A[i][size] / A[i][k];

```

```

        x_value_found[k] = 1;
    }
}
}
for(i=0 ; i< size+1 ; j++)
{
    printf(" %g ",A[i][j]);
}
}
}
MPI_Barrier(MPI_COMM_WORLD);
    // blocks the calling process until all group
    // processes have called the function...
if(my_rank == 0)
{
    printf("\n\t\t\t\t\t=====");
    printf("\n\t\t\t\t\tX values");
    printf("\n\t\t\t\t\t=====");
    for(i=0 ; i<numrows ; i++)
        printf("\n\t\t\t\t\t x[%d] = %g",i,x[i]);
    // prints the unknowns...
}
MPI_Barrier(MPI_COMM_WORLD);
    // blocks the calling process until all group
    // processes have called the function...
MPI_Finalize();
    // Exits from the message-passing system
Printf("\n");
Second = time(NULL);
    // Assigns the time to variable second...
If(my_rank == 0)
printf("\n\t\t\t\t\t **** Elapsed time : %d secs **** \n", second - first);
// prints elapsed time in the system with rank 0 ... )

```


PVM Program to solve a Tridiagonal system on a PVM cluster

```
#include <stdio.h>
#include "/pub_domain/pvm3/include/pvm3.h"
int main(int argc, char *argv[])
{
    int i,j,count,k,activep[20],a;
    int my_rank,mygid;
    int index1,index2;
    float alpha,gamma;
    int no_processes,numactivep,size;
    int instance_no,process_rank;
    int info,msgtag,power_value;
    float tri_mat[20][20],x[20];
    int tid[10];
    int x_value_found[10];
    int proceed = 0;
    int found = 0;
    int mstat;
    time_t first,second;
    first = time(NULL);
    my_rank = pvm_mytid();
    printf("\n lam %x ",pvm_mytid());
                                     // returns the tid of this process...
    mstat = pvm_mstat("linux28");
    printf("\nStatus of host 90.0.2.78 : %d",mstat);
    mstat = pvm_mstat("linux30");
    printf("\nStatus of host 90.0.2.79 : %d",mstat);
    mygid = pvm_joiningroup("JoinSmitha");
```

*//enrolls the calling task in the group named
"JoinSmitha"and returns the instance number of this
task in this group...*

```
if(mygid == 0)
{
    pvm_spawn(argv[0],argv+1,0,"",2,tid);
    // starts up 2 copies of an executable file task on  
the virtual machine
}
printf("\n My Parent is : %x \n",pvm_parent());
printf("\nNumber of Members in the specified group  
:%d",pvm_gsize("JoinSmitha"));
instance_no = pvm_getinst("JoinSmitha",pvm_mytid());
printf("\nInstance number of the tid in the group  
:%d",instance_no);
if(instance_no == 0)
printf("\n Initialization.....");
size = 7;
for(i=0;i<size;i++)
{
    for(j=0;j<size;j++)
    {
        tri_mat[i][j] = 0.0;
    }
}
// Tridiagonal system formed...
for(i=0;i<size;i++)
{
    for(j=0;j<size;j++)
    {
        tri_mat[i][i] = -2.0;
        tri_mat[i][i-1] = 1.0;
        tri_mat[i][i+1] = 1.0;
    }
}
```

```

    tri_mat[i][size] = i;
}
for(i=0;i<10;i++) x[i] = 0;
if(instance_no == 0)
{
    printf("\n =====");
    printf("\n Tridiagonal Matrix before Manipulation");
    printf("\n =====");
    for(i=0;i<size;i++)
    {
        printf("\n i = %d      ",i);
        for(j=0;j<=size;j++)
        {
            printf(" %g",tri_mat[i][j]);
        }
    }
    printf("\n-----Triplets-----");
    for(i=0;i<size;i++)
    {
        printf("\n");
        for(j=0;j<=size;j++)
        {
            if(j != size)
            {
                if(j == i-1)
                {
                    if((tri_mat[i][j]>0)||
                    (tri_mat[i][j]<0) )
                    printf("%gX%d",tri_mat[i][j],j);
                }
                else
                {
                    if(tri_mat[i][j] > 0)
                    printf(" + %gX%d ",tri_mat[i][j],j);
                }
            }
        }
    }
}

```

```

        else if(tri_mat[i][j] < 0)
            printf(" %gX%d",tri_mat[i][j],j);
        }
    }
    if(j == size)
        printf(" = %g ",tri_mat[i][j],j);
    }
}

no_processes = 2;
numactivep = 2;
if(instance_no == 0)
    printf("\n-----Triplets-----\n");
info = pvm_barrier("JoinSmitha",2);
for(i=0;i<no_processes-1 ; i++)
{
    for(j=0 ; j<numactivep-1;j++)
    {
        index1 = (2*j) + i;
        index2 = (2*j) + (3*i) + 2;
        if(instance_no == j)
        {
            alpha = -0.5;
            gamma = -0.5;
            count = (index1+index2)/2;
            printf(" \n Count : %d\n \t\t ",count);
            for(k=0;k<=size;k++)
            {
                tri_mat[count][k] = tri_mat[count][k] -
(alpha * tri_mat[index1][k] + gamma * tri_mat[index2][k]);
                if(k!=size)
                {
                    if(tri_mat[count][k] > 0)
                    {

```

```

if(instance_no==0)printf("%gX%d+",tri_mat[count][k],k);
    }
    else if(tri_mat[count][k] <0)
    {
    if(instance_no == 0)printf(" %g X%d
",tri_mat[count][k],k);
    }
    }
    if(k == size)
    {
        printf(" = %g
X%d",tri_mat[count][k],k);
    }
    }
}
info = pvm_initsend(PvmDataRaw);
// clears the send buffer & creates a new one for packing a
new message into the buffer
info=pvm_pkfloat(&tri_mat[0][0],70,1);
//packs the active message buffer with
arrays of floating type...
msgtag = 5;
info = pvm_bcast("JoinSmitha",msgtag);
info = pvm_barrier("JoinSmitha",2);
}
numactivep = 0;
power_value = 1;
for(a = 0; a <= i+1;a++)
{
    power_value = power_value * 2;
}
for(j=activep[1] ; j < no_processes ; j=
j+power_value,first++)
{

```

```

        activep[numactivep++] = j;
    }
}
info = pvm_barrier("JoinSmitha",2);
for(i=0 ; i<size ; i++,first--)
{
    x[i] = 0.0;
    x_value_found[i] = 0;
}
info = pvm_barrier("JoinSmitha",2);
if(instance_no == 0)
printf("\n Back Substitution ..... \n");
if(instance_no == 0)
{
    x[(size-1)/2] = tri_mat[count][size] /
    tri_mat[count][(size-1)/2];
    printf(" \n x[ %d] = %g", (size-1)/2 , x[(size-1)/2]);
    for( i=0 ; i<size ; i++)
    {
        tri_mat[i][size] -= tri_mat[i][count] * x[(size-1)/2];
        tri_mat[i][count] = 0;
    }
}
info = pvm_barrier("JoinSmitha",2);
info = pvm_initsend(PvmDataRaw);
    // clears the send buffer & creates a new one for packing a
    // new message into the buffer
info = pvm_pkfloat(&tri_mat[0][0],70,1);
    //packs the active message buffer with
    // arrays of floating type...

msgtag = 5;
    info = pvm_bcast("JoinSmitha",msgtag);
    info = pvm_barrier("JoinSmitha",2);
info = pvm_initsend(PvmDataRaw);

```

```

        // clears the send buffer & creates a new one for packing a
        // new message into the buffer
info = pvm_pkint(&x_value_found[0],10,1);
        //packs the active message buffer with
        // arrays of integer type...

msgtag = 5;
info = pvm_bcast("JoinSmitha",msgtag);
    info = pvm_barrier("JoinSmitha",2);
    info = pvm_initsend(PvmDataRaw);
        // clears the send buffer & creates a new one for packing a
        // new message into the buffer
info = pvm_pkfloat(&x[0],10,1);
        //packs the active message buffer with
        // arrays of floating type...

msgtag = 5;
info = pvm_bcast("JoinSmitha",msgtag);
// Solving for Odd Rows.....
if(instance_no == 0)printf("\n Solving for odd Rows ..... \n");
process_rank = 0;
for(i=1 ; i<size; i=i+4)
{
    if(instance_no == process_rank)
    {
        printf("\n i = %d " , i);
        for(k=0 ; k<size ; k++)
        {
            if(tri_mat[i][k] != 0)
            {
                x[k] = tri_mat[i][size] / tri_mat[i][k];
                x_value_found[k] = 1;
            }
        }
    }
}
}

```

```

    info = pvm_barrier("JoinSmitha",2);
    info = pvm_initsend(PvmDataRow);
    // clears the send buffer & creates a new one for packing a
                                new message into the buffer
    info = pvm_pkfloat(&tri_mat[0][0],70,1);
                                //packs the active message buffer with
                                arrays of floating type...

    msgtag = 5;
    info = pvm_bcast("JoinSmitha",msgtag);
    info = pvm_barrier("JoinSmitha",2);
    info = pvm_initsend(PvmDataRow);
    // clears the send buffer & creates a new one for packing a
                                new message into the buffer
    info = pvm_pkint(&x_value_found[0],10,1);
    msgtag = 5;
    info = pvm_bcast("JoinSmitha",msgtag);
    info = pvm_barrier("JoinSmitha",2);
    info = pvm_initsend(PvmDataRow);
    // clears the send buffer & creates a new one for packing a
                                new message into the buffer
    info = pvm_pkfloat(&x[0],10,1);
    msgtag = 5;
    info = pvm_bcast("JoinSmitha",msgtag);
    process_rank++;
}   for(i=0 ; i<size ; i++)
{
    for(k=0 ; k<size ; k++)
    {
        if((tri_mat[i][k] != 0) && (x_value_found[k] == 1))
        {
            tri_mat[i][size] -= tri_mat[i][k] * x[k];
            tri_mat[i][k] = 0;
        }
    }
}

```



```

}
info = pvm_barrier("JoinSmitha",2);
if(instance_no == 0)
{
    printf("\n =====");
    printf("\nTridiagonal Matrix after Cyclic Reduction
    Method");
    printf("\n =====");
    for(i=0 ; i<size ; i++)
    {
        printf("\n\t\t\t i = %d \t\t",i);
        for(j=0 ; j<size+1 ; j++)
        {
            printf(" %g ",tri_mat[i][j]);
        }
    }
}
info = pvm_barrier("JoinSmitha",2);
for(process_rank = 0 ; process_rank <= no_processes ;
process_rank++)
{
    info = pvm_barrier("JoinSmitha",2);
    if(instance_no == process_rank)
    {
        for(i=0 ; i<size ; i=i+2)
        {
            for(k=0 ; k<size ; k++)
            {
                if(tri_mat[i][k] != 0)
                {
                    for(found = size-1 ; found > k ;
                    found--)
                    {

```

```

        if(x_value_found[found] == 1)
            proceed = 1;
        else proceed = 0;
    }
    if(proceed == 1)
    {
        x[k] = tri_mat[i][size] / tri_mat[i][k];
        x_value_found[k] = 1;
    }
}
}
for(i=0 ; i<size ; i++)
{
    for(k=0 ; k<size ; k++)
    {
        if((tri_mat[i][k] != 0) &&
            (x_value_found[k] == 1))
        {
            tri_mat[i][size] -= tri_mat[i][k] *
                x[k];
            tri_mat[i][k] = 0;
        }
    }
}
}
info = pvm_barrier("JoinSmitha",2);
}
info = pvm_barrier("JoinSmitha",2);
instance_no=pvm_getinst("JoinSmitha",pvm_myid());
if(instance_no == 0)
{
    printf("\n=====");
    printf("\nX Values ");

```

```
printf("\n=====");
    for(i=0 ; i<size ; i++)
        if(instance_no == 0)
            printf("\n x[%d] = %g",i,x[i]);
    }
info = pvm_barrier("JoinSmitha",2);
second = time(NULL);
                // Assigns the time to the variable second...
if(instance_no == 0)
printf("\n Elapsed time : %d",second-first);
                // Prints the elapsed time on the screen...
pvm_exit();
                // Process leaves PVM
}
```

