PARDISC: A Cost Effective Model for Parallel and Distributed Computing

ACHUTHA RAMAN R RAJKUMAR HARI PRAKASH G BALA KISHORE B
Operating System Group
Centre for Development of Advanced Computing
2/1, Ramanashree Plaza, Brunton Road
Bangalore - 560 025, Karnataka, India
email: raj@cdacb.ernet.in

Abstract

A homogeneous system of PCs, workstations, minicomputers etc., connected together via a local area network or wide area network represents a large pool of computational power. However, in a network of PCs and workstations, transparency is not provided and hence, users are aware of other machines. PARDISC is a parallel programming environment, which provides the needed transparency as a scalable OpenFrame Computing Model. PARDISC stands for **PAR**allel and **DIS**tributed **C**omputing on a homogeneous network. It supports three models of computing by providing the functionalities required to view any homogeneous network as a Loosely Coupled Parallel Computer, Processor Pool Architecture, or Cluster of Workstations.

PARDISC aims at providing a cost effective parallel and distributed programming environment to the academic and R & D institutions, since it employs the existing well established Local Area Network (LAN) and models it to support both the paradigms. This paper presents an overview of PARDISC along with its architecture and design. It discusses how PARDISC can be used to configure the network as a loosely coupled parallel machine, processor pool architecture, and distributed computing environment with Logical Network Connectivity. Software architecture discusses configuration servers, client processes, processor pool servers, and process communication interface of PARDISC. We end the paper with a description of some issues involved in its implementation on UNIX platform, and its suitability for parallel programming.

Keywords: PARDISC, Parallel, Distributed, Processor Pool, Logical Connectivity, Loosely Coupled Parallel Machine, Configuration, Process Communication Interface.

1. Introduction

High speed network and improved microprocessor performance are making networks of workstations an appealing vehicle for parallel computing [5]. By relying solely on commodity hardware and software, network of workstations can offer parallel processing at low cost. A network of workstations can be realized as a processor bank in which dedicated processors provide computing cycles, or it can consist of dynamically varying set of machines that perform long running computations during idle periods. In this case, the hardware cost is essentially zero, since many organizations already have extensive workstation networks.

In terms of performance, network of workstations approach or exceed supercomputing performance for some applications. The problem of locating and efficiently utilizing the resources in a network is an important factor [11]. PARDISC is designed to allow the users to utilize the computing power of the network for executing their tasks, thereby providing access to computational resources far beyond that provided by a standard system.

Several working systems [2, 4, 7] have been built which offer some type of distributed computation. In [9], a system has been described based on Butler system which allows the users to execute jobs on remote workstations. It uses a central idle machine registry to find a candidate idle machine. A distributed operating system, MACH-1 based on workstation model has been described in [3]. It is a microkernel based operating system in which user-programs can run remotely when a specific machine gets overloaded. Amoeba [13] is a distributed operating system based on the processor pool concept. It makes a collection of CPUs and I/O devices behave as a single computer. The PARAM 9000 Supercomputer supports both the Massively Parallel Processing (MPP) and Cluster Computing personalities [8]. In the MPP personality, *compute nodes* are loaded with PARAS microkernel and system servers, and *service nodes* are loaded with Solaris (Sun OS). Whereas, in the alternate personality all the nodes are configured as service nodes.

PARDISC provides a software model of integrated solution which is not found in a single unit in the earlier systems, without any changes in the underlying current setup including network and operating system.

2. PARDISC at a Glance

The main aim of this work is to build a transparent distributed parallel environment, which is flexible, cost effective, and simple. PARDISC is a software model which provides the best of both worlds, Parallel and Distributed. It uses the existing network of homogeneous computers to run parallel programs using logical connectivity method and to execute distributed programs utilizing the power of other idle computers and pool processors in the network. The basic idea is to provide the users with the illusion of a single powerful timesharing system, when, in fact the system is implemented on a collection of machines, potentially distributed across the network. The architectural model of the PARDISC is shown in Figure 1.

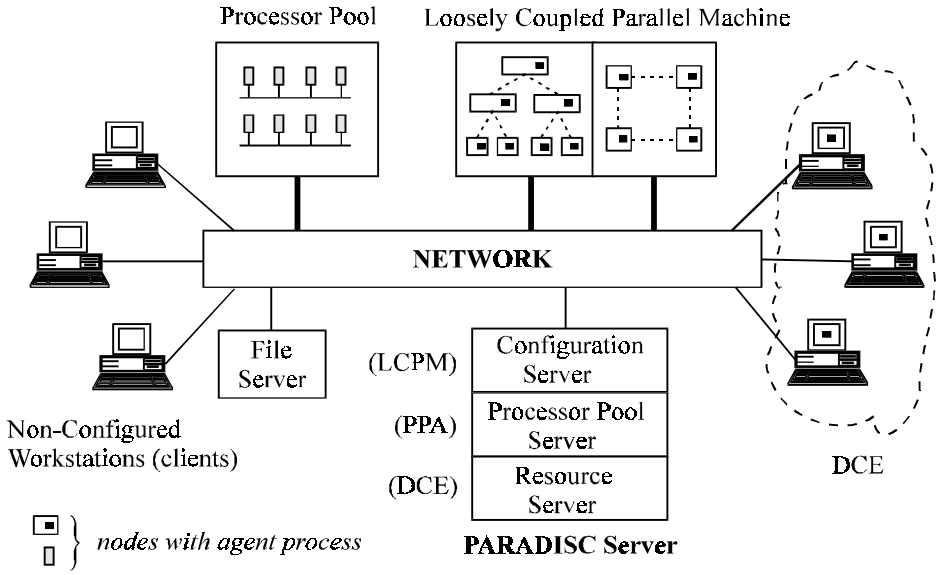

Figure 1: Architectural Model of PARDISC

PARDISC is typified by specialized components such as PARDISC server, which includes configuration server, processor pool server, and resource server. In each of the three configurations supported by PARDISC viz. Loosely Coupled Parallel Machine (LCPM) [6], Processor Pool Architecture (PPA) [1], and Distributed Computing Environment (DCE), it has a central server running on one of the designated nodes (computers with main processor and memory), coordinating with loader servers running on each of the other nodes specific to the configuration.

PARDISC server configures a group of computers logically to the required topology and it allocates idle machines and pool processors for the users on demand. As a LCPM, the network can be configured to standard topologies such as tree, mesh, cube etc., or user defined topologies suitable to the problem domain. These topologies are logically connected through configuration server, provided by the PARDISC. As a PPA, user can make use of dedicated nodes for parallel and distributed applications. As a DCE, PARDISC provides a rich set of commands such as PARCPL (Parallel Compile), PARRUN (Parallel Run), PARMAKE (Parallel Make) etc., to distribute user tasks across the network transparently.

The network can be configured into any of the two modes, parallel or distributed or both, with the usage of two different configuration files. Each node maintains a per-process node connectivity status. It enables the identification of neighborhood nodes. The configuration server maintains the global per-process connectivity information. The interprocess communication between processes in and across the processors is provided by process communication interface.

Process Communication Interface (PCI) provides two modes of communication: point-to-point and collective communication (broadcast and multicast), which are layered over the basic primitives such as `send()` and `receive()`.

3. Software Architecture

The network can be configured in three aforementioned architectures by using different system configuration files. It contains the following information about the network configuration:

- number of nodes in the network
- number of LCPM/Processor Pool/DCE nodes
- maximum number of tasks that can be loaded per node
- resources available on each node
- default specification, etc.

The PARDISC server provides three different functional services depending on the architecture. It works as Configu-

ration Server in case of LCPM, Processor Pool Server in case of PPA, and as a Resource Server in case of DCE architecture. All other nodes, except PARDISC server node, register their network addresses and ports with the PARDISC server at boot time.

Loosely Coupled Parallel Machine (LCPM)

The components of LCPM model are configuration server (topology server), agent process, and PARDISC loader. (See Figure 2.) A configuration server, which runs on a single dedicated machine, is responsible for establishing the logical connection among the other nodes according to the defined topology (tree, mesh, cube, etc., or user defined topologies). An agent process runs on each of the other nodes (except configuration sever node) and is responsible for (loading and) spawning of tasks. PARDISC loader is the front end user interface, which loads the user tasks on remote nodes, assigned by the configuration server.

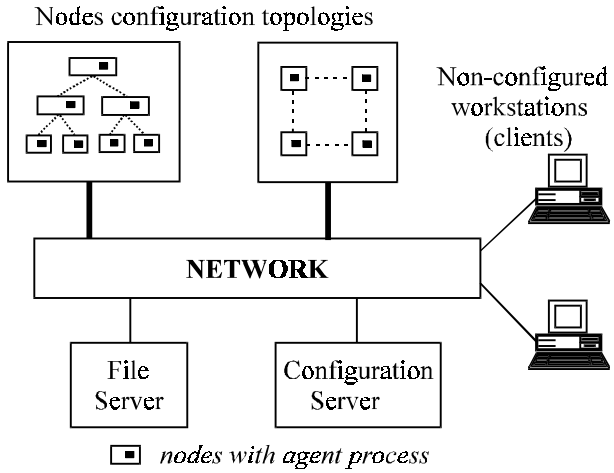

Figure 2: Loosely Coupled Parallel Machine

PARDISC loader sends the configuration information to the configuration server, which allocates requested number of nodes by the client by accessing its internal resource table and creates the topology table. The requests from the loader are sent to the configuration server in the form of a tuple:

(N, T) where { N = number of nodes,
T = topology (LINEAR, TREE, MESH, CUBE, USERDEFINED, NULL)
}

The configuration server identifies the N agent processes and forms the logical neighbourhood of each agent process by executing the topology configuration algorithm, and thus topology T is formed. Typically, the topology table contains the following information:

- topology name
- role of each node in the topology (e.g., PARENT, CHILD, etc.).
- immediate neighbors and their relation to one another.
- group id and node ids of all the nodes in the topology.
- originator id (root node id)

Configuration server sends the ALLOCATE_REQ message to all the agent processes running on the allocated nodes for the current application. In response to this message all the agent processes create communication ports to load the tasks and register the same with the configuration server. Configuration server updates its (current group) topology table with the communication ports, and sends this table to all the agent processes of the current application and to the PARDISC loader. PARDISC loader downloads the user tasks to the agent process communication ports of the configured nodes. Once the tasks are loaded on the respective nodes, they start executing and the tasks can communicate with other tasks of the same topology using PCI calls. Since every task has the topology table (shares with agent process), any one of them can communicate with its related neighbor within their topology. On complete execution of the application, PARDISC loader informs the configuration server to release the resources allocated for that application.

Processor Pool Architecture (PPA)

In PPA, all the computing power of a network is located in one processor pool, which consists of multiple CPUs, each with its local memory and network connections. Users can be assigned as many CPUs as they need for short periods by the Processor Pool (PP) server after which they are returned to the pool, so that other users can make use of them. There is no concept of ownership here; all the processors belong equally to everyone. Many parallel and concurrent applications' performance may not be satisfactory, if they are scheduled by traditional timesharing systems. They may need dedicated processors for improved performance. Processor pool model is best suited for these kind of applications. The PPA of PARDISC is shown in Figure 3.

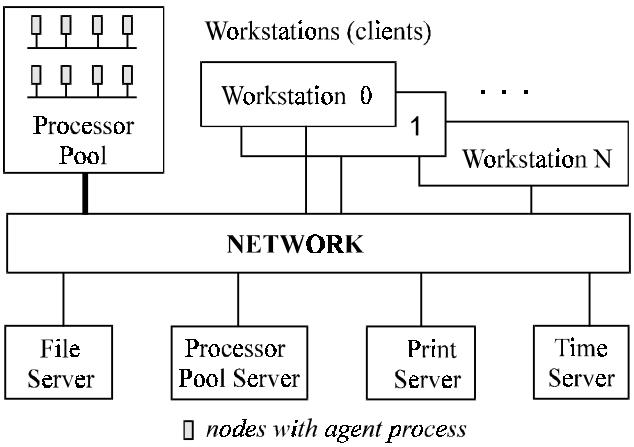

Figure 3: Processor Pool Architecture

In a LAN, each node has a processor and local memory. A group of such nodes collectively form the Processor Pool. The nodes which are not part of the pool act as client machines. User logs into the system only through the client machines, which are used as dummy terminals. Processor Pool server runs on a designated node to manage the pool processors. User submits tasks to the client machine through the PARDISC loader, which in turn requests the PP server to allocate required number of nodes. PP server allocates the requested number of processors from the pool and returns the corresponding node-ids and port-ids of the agent processes to the client. The client, after acquiring the node-ids, starts downloading the user tasks to the appropriate nodes. Agent process (residing on the allocated node) spawns the task on its node and redirects the results to the client's machine. Finally, on completion of the tasks, client requests the PP server to release the allocated processors and they are returned to the pool. The PP nodes can also be configured as LCPM.

Distributed Computing Environment (DCE)

In DCE, each machine retains its own identity, i.e., user can log into any machine and can access any other machine in the network. Each node is a separate machine as in the case of LCPM, but distributed tasks can be executed transparently across the network on lightly loaded machines, which are managed by the Resource Server. (See Figure 4.) It gives the single system image to the user so that user tasks can be compiled, loaded, and executed any where on the network. The results of the task are redirected to the user owned machine.

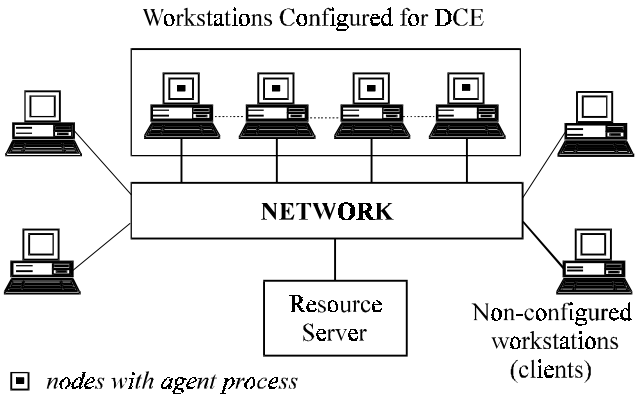

Figure 4: Distrubuted Computing Environment

In DCE, user views the network as a loosely coupled cluster of computers. A configuration file specifies which tasks should be loaded on which logical nodes. In general, user views DCE as a high performance virtual machine. Resource Server in this case is responsible for assigning the nodes for the user tasks uniformly across the network. It maintains processor-load table which keeps track of the most recent information of the load of all the DCE processors. This table is periodically updated by the agent processes running on the DCE machines, so that it can assign lightly loaded machines evenly when resources are requested.

The OSF/DCE software bundle is installed on all the machines and they are configured as a single *DCE cell*, so that all the DCE services can be used along with LCPM and PPA. DCE provides the distributed programming environment such as directory service, file service, time service and RPC. Users can write their own distributed programs by the tools and services provided by OSF/DCE. PARDISC DCE is layered over OSF/DCE and it is provides one more service known as *load balancing service*. This server uses the DCE API services, to collect the load info from each of the machines within the cell. This server is basically a distributed application by itself. It also provides services such as PARCPL, PARMAKE, PARRUN to compile, make and run the application on the DCE machines transparently.

Processor Allocation and Synchronization

PARDISC server is responsible for processor allocation in all the above discussed models. It implements *gang scheduling*, which guarantees simultaneous scheduling of parallel applications over multiple processors on the network. This implies that, processor allocation should be done for any parallel application only if all the tasks of a parallel application can be loaded on the allocated nodes. Before allocating the nodes, PARDISC server probes the nodes to detect the *health of the processor*. It selects only the live nodes. When all the loaded tasks complete execution, the agent process of the node requests the PARDISC server to deallocate the node.

The processor allocation is also based on priority. For example, interactive applications can be treated as high priority and the batch applications can be assigned low priorities. Priority assignment is configurable. For example, priorities can be classified based on user groups, type of applications, and other user specified parameters. Default processor allocation strategy is also supported in PPA and DCE models. If the number of processors or loading configuration information is not specified by the user, then PARDISC server allocates processors less than or equal to the default number of processors, which is specified in the system configuration file.

User Interface

PARDISC provides an user interface to load, control, and monitor user applications to give the required transparency.

In all the three models, pload (PARDISC loader) is provided to load the parallel and distributed applications on PARDISC environment. Pload takes the configuration file as the input. In LCPM, configuration file consists of two sections, *config* and *load sections*, whereas in the other two

models, the configuration file contains only the load section.

A configuration file allows the user to specify any topology in LCPM and it consists of configuration and loading sections. Configuration section specifies the topology and the relation of each node with its neighborhood nodes and loading section specifies the placement of tasks on nodes. The typical format of configuration file is as follows:

```
# CONFIG PART
TOPOLOGY: TREE
NUM NODES: 3
#optional fields

# LOADING PART
load parent.out on node1 #PARENT
load lchild.out on node2 #LCHILD
load rchild.out on node3 #RCHILD
```

Here `node1` refers to the root (can be the node from where the job has been submitted) whereas, the `node2`, .., `nodeN` are other nodes of the topology.

In case of user-defined topology configuration file contains the fields: node number, tasks that should be placed on the node, links connecting tasks, link identifiers. The typical format of user-defined configuration file (Config. section only) is as follows:

```
# CONFIG PART
#Node Tasks     Links                    Link-ID
N1 T1           0                        0
N2 T2,T3        T1-T2,T1-T3              1, 2
N3  T4,T5,T6,T7 T2-T4,T2-T5,T3-T6,T3-T7 3,4,5,6
```

The link-ids specified in the above configuration file can be used as logical communication-ids in PCI calls.

DCE and PPA support a rich set of commands such as PARCPL (Parallel Compile), PARRUN (Parallel Run), PARMAKE (Parallel Make), etc. to provide location transparent execution of the user jobs (set of tasks). PARCPL transfers independent source files to different nodes, which compiles and links the same on those remote nodes and transfers the executable to the owner machine. PARMAKE is used to build executable from the multiple source files of the project. It is performed by transferring each source file to a different node of DCE/PPA and compiles the same on those remote nodes and transfers the object files to the owner machine, which are in turn linked to produce an executable on owner machine. PARRUN distributes user tasks across the network for execution.

Job control commands are supported to display the task status, to suspend/resume/kill the tasks. Commands to collect the statistics of any application over the network and monitoring commands to trace the job activities, processor load and other information are also supported.

4. Process Communication Interface

Process Communication Interface (PCI) is a message passing interface of PARDISC supporting communication among tasks in the loosely coupled parallel machine configuration. The Application Program Interface (API) supported by PCI is consistent irrespective of device specific communication protocols and hence provide the portability feature to the PCI users. The functionality of the PCI is designed to provide simple and flexible syntax, and is based on current common practice, and is similar to that provided by widely used message passing systems such as PVM [12], MPI [14], etc. Unlike PVM and MPI interfaces, PCI interface calls are particularly designed for PARDISC environment, where the communication among the tasks are expressed using logical node connectivity. API calls supported by PCI fall into following three categories:

- Point-to-Point Communication,
- Group Communication, and
- Signal Passing.

All these interfaces are layered on top of the basic primitives `PCI_Send()` and `PCI_Receive()`. PCI supports both synchronous (`DELAY`) and asynchronous (`NO_DELAY`) modes of communication. The syntax of some of the important PCI calls is listed below:

```
int PCI_Init();
int PCI_Send( ReceiverId,MsgPtr,Length,Mode );
int PCI_Receive( MsgPtr, ExpectedLen, Mode );
int PCI_GroupSend( MsgPtr, Length, Mode );
int PCI_SendSignal( ReceiverId, SignalId );
```

The logical task-ids used with PCI calls are the same as those specified in the configuration file. The mapping of logical-ids to the physical addresses is performed by the PCI library with the help of per-group port and topology tables maintained by each task. `PCI_Init()` creates two ports, one for sending the message and the other for receiving the message, and registers them with root (master) process (which is assigned by the PARDISC loader) of its group, which then builds the group port table of that group, and multi-casts the table to the group members. With this, every task can communicate with every other task of that group independently.

To support asynchronous communication, PCI library maintains inbound queues to store the unposted receive messages. The group communication calls support communication among the tasks of the same group (group-cast). Signals can also be passed between tasks. Signals are sent to the agent process of the destination node and it is delivered to the target task, as a local signal from the agent process.

5. Implementation Issues

The implementation of PARDISC version 1.0 is underway on a network of computers running SVR4 Unix operating sys-

tem. It aims at source-level compatibility across SVR4 Unix systems. It uses PCI as the message passing library using TCP/IP sockets as low level device specific protocol. A multithreaded implementation of the PARDISC on multithreaded operating systems, such as Solaris and Mach, improves the performance.

6. Conclusions

Since PARDISC supports parallel and distributed computing with the existing network of computers, it can be used as a developing and testing environment for the parallel and distributed programs in academic and R & D institutions. In LCPM configuration, distributed parallel algorithms like distributed sorting etc., can be developed and tested. Using DCE configuration, distributed algorithms like clock synchronization algorithm, election algorithms, etc., can be developed and tested. In PPA, both parallel and distributed algorithms can be performed more efficiently using the power of dedicated processors.

In LCPM mode of PARDISC, the programmer must divide the computation among different tasks and use message passing facilities of PCI to control interaction among the concurrent tasks. It is suitable for both data and process parallelism [10]. Programmer need not worry about where the tasks are loaded physically and how to communicate with them.

The three personalities of high performance computing, LCPM, PPA, and DCE are integrated into the PARDISC to realize a cost effective solution with zero extra hardware and no modification to the operating system.

Acknowledgments

The authors are grateful to all the members of C-DAC, Bangalore, and in particular all the members of Operating System Group. We owe a debt of gratitude to Prof. Vishwanathan Iyer K, Regional Engineering College, Tiruchirapalli and Prof. Venugopal K R, University Visvesvaraya College of Engineering, Bangalore University for their useful conversations and comments during the implementation and an earlier draft of the paper.

References

1. Achutha Raman R, *Remote Task Execution In Processor Pool Environment*, M. Tech. Thesis, Regional Engineering College, Tiruchirapalli, India, 1994.
2. Ashfaq A, Viktor K Prasana, Muhamad, and Cho-hi Wang, *Hetrogeneous Computing: Challenges and Opportunities*, IEEE Computer, June 1993.
3. Baron R, et al., *MACH-1: An operating environment for large scale multiprocessor applications*, IEEE Software, Vol. 2, 1985.
4. Cheriton D R, *The V Distributed System,* Communications of ACM, Vol 31, 1988.
5. Christiane Amza et. al., *ThreadMarks: Shared Memory Computing on Network of Workstations*, IEEE Computer, Feb. 1996.
6. Hari Prakash G, *Loosely Coupled Parallel Architecture on a LAN, M. Tech.Thesis, Regional Engineering College*, Tiruchirapalli, India, 1994.
7. Henry Clark and McMillan B, *DAWGS - A Distributed Computer Server Utilizing Idle Workstations*, Journal of Parallel and Distributed Computing, Vol. 14, 1992.
8. Mohan Ram N, Gangaprasad, Bala Kishore B, *The Operating Environment*, The PARAM 9000, (c) C-DAC, 1996.
9. Nichols D A, *Using Idle Workstations in a Shared Computing Environment*, Operating System Review, November 1987.
10. Rajkumar et al., *PEACE Threads Interface on Microkernel*, Proceedings of The Third Conference On Advanced Computing, Tata McGraw Hill, India, 1995.
11. Spector A Z, *Performing Remote Operations Efficiently on a LAN*, Communications of ACM, Vol 24(4), 1982.
12. Sunderam V S, *PVM: A Framework for Parallel and Distributed Computing*, Journal of Concurrency: Practice and Experience, Vol 2(4), 1990.
13. Tanenbaum A S, et al, *Amoeba: A Distributed Operating System for the 1990s*, IEEE Computers, Vol 23, 1990.
14. The MPI Forum, *MPI: A Message Passing Interface Standard*, May 1994.

About the Authors

Achutaraman R Consulting Software Engineer, HP-India Sofware Operations, Bangalore. He has completed Master of Engineering in Computer Science from the Bharathidasan University in the year 1994. His research interests include Distributed Computing, Networking, and Parallel Algorithms. He can be reached by e-mail at achu@india.hp.com.

Rajkumar Member of the Technical Staff, Operating Systems Group, C-DAC, Bangalore. He received Master of Engineering degree in Computer Science from the University of Bangalore in the year 1995. He was awarded the Dharma Ratnakara Memorial Trust *Gold Medal* for securing the I Rank in the year 1992. He has coauthored the books, *Microprocessor x86 Programming* and *Object Computing with C++*. Besides being a visiting faculty for the Bangalore University, he is conducting a popular course on the *Java by Email* which is circulated around the globe. His research papers in the area of High Performance Computing have appeared in National and International Conferences. His research interests include Graphics, Programming Paradigms, Multithreaded Computing, and Microkernel OS for MPP. He can be reached by e-mail at raj@cdacb.ernet.in.

Hari Prakash G Senior Software Engineer, HCL-HP R&D, Madras. He has completed Master of Engineering in Computer Science in the year 1994 from Barthidhasan University. His Research interest includes Distributed Computing, Parallel Algorithms, Parallel Architectures, Genetic Algorithms and Artificial life. He can be reached by e-mail at ghari@hclt.com.

Bala Kishore B Software Engineer, Silicon Automation Systems, Bangalore. He has completed Master of Technology in Computer Science from the Kakatiya University in the year 1995. His research interests include Real-Time Systems, Compilers, and Operating Systems. He can be reached by e-mail at bala@sas.soft.net.