POENDOM: A Parallel Operating Environment on a Network of DOS Machines

GUNNESWARARAO M and RAJKUMAR
Operating Systems Group
Centre for Development of Advanced Computing
2/1, Ramanashree Plaza, Brunton Road,
Bangalore - 560 025, India
e-mail: {raomg, raj}@cdacb.ernet.in

G MOHAN
Department of Computer Science & Engineering
Regional Engineering College, Tiruchirappalli - 620 015.

Abstract

Parallel computing on a cluster of workstations and personal computers has very high potential since it leverages the existing hardware and software. Parallel programming environment offers the user a convenient way to express parallel computation and communication. A lot of work has been done in synthesizing distributed memory parallel machines from a cluster of workstations. However, such a parallel environment is not supported on a network of DOS machines.

POENDOM, **P**arallel **O**perating **E**nvironment on a **N**etwork of **DOS** **M**achines — provides a master process for job mapping and configuration management and several slave processes for job spawning on client nodes in coordination with the master process. The MPL (**M**essage **P**assing **L**ibrary) provided by POENDOM can be used to develop coarse grained parallel applications which can be executed in POENDOM environment. MPL supports message passing interfaces for point-to-point communication, collective communication, and for synchronization among different nodes working on the same problem. MDF (**M**ulti-**D**imensional-**F**iles) library provides interfaces which deal with special files called multi-dimensional-files, suitable for writing parallel programs using the data parallelism paradigm. The emulated parallel machine of POENDOM consists of a master (host) node and several slave (remote) nodes. The master node accesses a user specified configuration file and distributes the tasks to slave nodes. On task mapping, master node and slave nodes will collectively solve the problem. This paper presents the design and implementation of POENDOM which includes master and slave processes. It also discusses MPL and MDF libraries.

1 Introduction

A number of distributed memory parallel machines [1, 2] have been developed for high performance computation. However, they are not easily accessible due to their high costs. But with the reduction in processor costs and the availability of high speed LANs, a significant work is seen in the literature to synthesize parallel machines from a cluster of workstations [3]. The distributed operating system AMOEBA [4] offers a new language called Orca for parallel programming. More recently PVM [5] is being used in developing parallel programs on a cluster of workstations. Even a standard interface, MPI [6] is also proposed for writing parallel programs on message passing parallel machines. Similar work had been reported in [7] in which, a distributed programming language was developed as an extension of C. The disadvantage of such an approach is that a novice user can not use it conveniently. Instead, a different

approach has been taken in our work by providing the interface to the POENDOM parallel environment in the form of library calls.

POENDOM offers a cost effective solution in synthesizing parallel machine from a LAN. The existing parallel machines IBM SP2 [2], PARAM [10] etc. follow the message passing paradigm for scalability. We also followed the same approach in developing the support for parallel programming.

The remainder of the paper is organized as follows: Section 2 discusses the methodology and design. Section 3 discusses the implementation overview. Section 4 discusses the performance evaluation and finally and Section 5 gives a conclusion of the work.

2 Methodology and Design

POENDOM consists of three components: *Master and Slave* model, *MPL*, and *MDF* libraries. The first component is an underlying runtime system which supports execution of the parallel programs. The second and third components are MPL and MDF libraries which offer primitives for developing parallel programs in POENDOM environment.

2.1 Architecture of POENDOM

Supervisor - Worker model [8] was followed with relevant modifications in the design of the underlying system. Emulation of a parallel machine mainly consists of two components: a Host node and other Remote nodes. The architecture of the POENDOM is shown in Figure 1. The runtime system must be established before submitting a job for parallel execution. First, all the nodes which act as slaves must be loaded with slave processes. The master process assigns slave processes, name of the task to be spawned as specified in the configuration file.

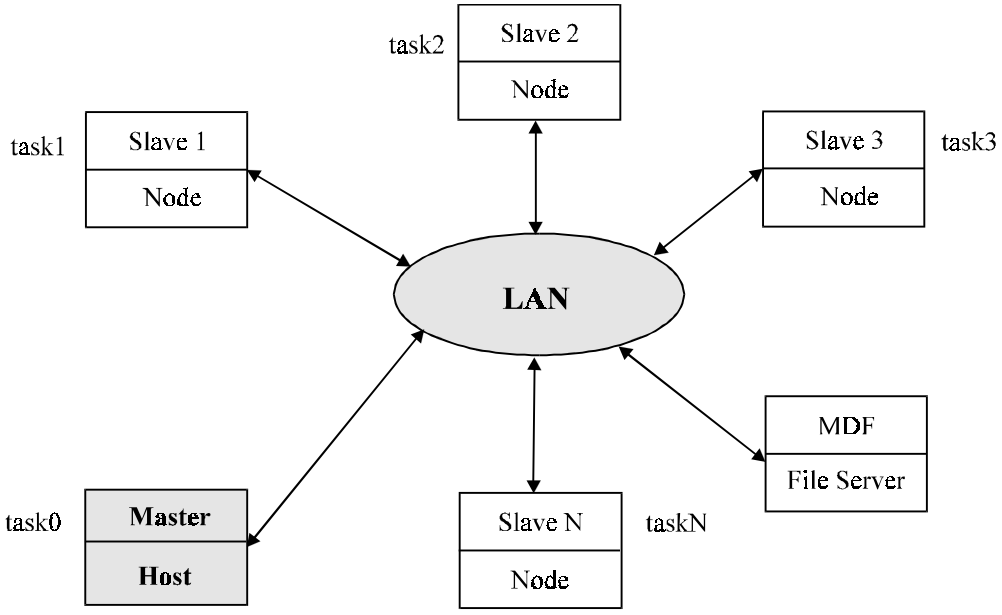

Figure 1: Architecture Model of POENDOM

The master node will perform certain supervisory work. It accesses the user specified configuration file and assigns processes to the slave nodes. The slave nodes will execute the assigned processes cooperating each other. POENDOM also allows the master node to execute a task parallely in synchronization with the slave nodes to enhance the performance. A logical node_id is attached with each participating node. The node with the node_id zero is the master node and the other nodes are the slave nodes. The computed results are gathered by the master node at the host end.

2.2 MPL Library

A parallel program is a collection of subprograms which execute parallely on different nodes but working on the same problem. The two popular models for parallelism, process parallelism and data parallelism can be realized using MPL calls. MPL supports the following categories of calls:

- Port Operations
- Point to Point Communication
- Collective Communication
- Synchronization of Tasks

A process has to open a communication channel (port) before initiating communication. A port is a repository of messages. This concept is similar to TCP/IP sockets. The primitives `openport()` and `closeport()` are provided for opening and closing of ports.

The primitive way of communication is between two nodes via a port. This can be either synchronous or asynchronous. In the former case, the sender waits until receiver acknowledges the message sent. The MPL's `sync_send()` and `block_receive()` support this. In the later case, the sender is handed over the control immediately after copying the message into the sender agent buffer. The `async_send()` and `no_block_receive()` are the primitives for this mode of communication.

Collective communication routines are provided for coordinated communication among a group of processes [6]. Initially, `create_mgroup()` can be used to create a multi_cast group with a group-name and specified number of members. Other nodes can register themselves to this group using `attach()`. The nodes have to specify on which port they want to receive messages. The function `multi_cast()` allows to send a message to the members of a specified multi_cast group. The receiving node can use either `block_receive()` or `no_block_receive()` calls. The broadcast facility of the LAN is used in sending a message using `mcast()` to the members of a multi-cast group which are listening using `mcast_receive()`. Internally such members are listening on the same port which is transparent to the user.

When a number of nodes work on the same problem parallely there should be some mechanism to keep synchronization among all the participating nodes. A node may have to wait till some event has happened at some other node or a node may have to send the completion signal to a node waiting on this node. This is can be achieved by using `start()` and `wait()` primitives. The `start()` sends a start message to all the nodes that are waiting on it. The `wait()` allows to wait until it receives start message.

2.3 MDF Library

The data parallelism requires partitioning of data into multiple chunks and each chunk being processed independently. POENDOM's MDF library allows data distribution and processing using multi-dimensional files. MDF supports the following categories of calls:

- Functions to create MDFs and open MDFs.
- Functions to operate on MDFs.
- Functions to maintain synchronization.

The data to be processed by a parallel program will often be in the form of multi_dimensional arrays on secondary storage. Mapping these arrays onto a one dimensional byte stream in the form of a file will not only be inconvenient but also inefficient and cumbersome to users. So special files called *multi dimensional files* (MDFs) which have a structure identical to the data a node is processing are used.

The primitives `arcreat()` and `aropen()` allow to create an MDF and open it in a desired sharing mode. At the time of creation, name of the MDF should be specified. Once an MDF is created data can be written into it or can be read from it. Every MDF is divided into a number of chunks. The size of each chunk is to be specified at the time of creation of the MDF. The functions `readchunk()` and `writechunk()` can be used to read or write data out of or into an existing MDF. This helps the user to view the data as a fixed number of blocks. Functions are also provided to read (write) from (into) an MDF at any arbitrary position and of any arbitrary size. The primitives `readsubarray()` and `writesubarray()` can be used for this. To get the information about an existing MDF by any node other than the creating node, the function `get_attrib()` can be used. This returns the size of each chunk in the MDF, the size of each element in bytes etc.

To achieve synchronization among a group of nodes, primitives are provided to create synchronization groups and to maintain synchronization among the group members. The `create_sgroup()` allows to create a synchronization group with a specified number of group members. The function `g_sync()` allows to keep the members in synchronization.

Since MDFs are also one form of files the following procedure has been adapted in accessing the files to avoid deadlocks:

a. Check whether the portion is already locked by some other node in unsharable mode.
b. If locked wait till it is unlocked.
c. Lock the portion in the desired sharing mode.
d. Perform read/write operation.
e. Unlock the portion.

3 Implementation Issues

The software is designed and developed keeping in mind of the portability and scalability issues. The work is carried out on an ARCnet (2.5 Mbits/sec) LAN connecting x86 nodes. The network operating system used is Novell Netware and the nodes run MS-DOS operating system.

The IPX/SPX protocol is the underlying communication protocol that is used in message passing functions. Since IPX is unreliable, the MPL library explicitly addresses the issues related to reliability in communication.

On a file server, every file is owned by the node that has opened it. The ownership can be possessive or communal. The facilities provided by NetWare as synchronization services are used in providing shared access to the files. In the case of writing parallel programs using MDFs, since all the nodes can access the files simultaneously, record level physical locking is performed. There are four main steps to avoid deadlocks: logging, locking, releasing, and clearing.

To use the NetWare method of physical record locking, the protected region of the file is expressed in terms of a given number of bytes, starting at a given position in the file. A `LogPhysicalRecord` request is made first and then a `LockPhysicalRecordSet` request is made to lock a particular portion of the file. After the updates are completed, a `ReleasePhysicalRecordSet` and finally `ClearPhysicalRecordSet` requests are made to remove the affect of locking the file.

4 Performance Evaluation

The POENDOMs run-time system is tested by developing a number of parallel application programs. The programs used both message passing functions and functions that operate on MDFs.

The major observation is that the POENDOM better suits for loosely coupled parallel applications where the communication is less and computation can be performed more or less independently. Some of the problems that are tested successfully are: parallel sorting by enumeration, parallel computation of minimal spanning tree of a graph, parallel merge-split sorting, parallel implementation of median filtering of a given image, parallel matrix multiplication etc. Figure 2 shows the runtime performance improvement of a median-filtering algorithm [11] for image processing. In this case, performance is analyzed on fixed number of processors (p=3) with varying image sizes. Figure 3 shows runtime performance of matrix multiplication (90x90) and median-filtering (150x200) algorithms [11] on varying number of processors with fixed data sizes.

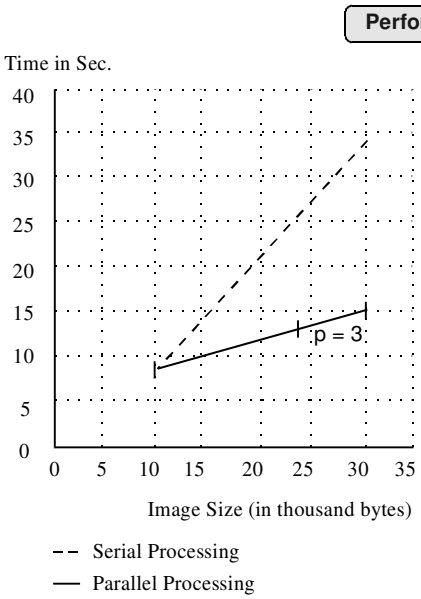

Figure 2: Serial Vs. Parallel

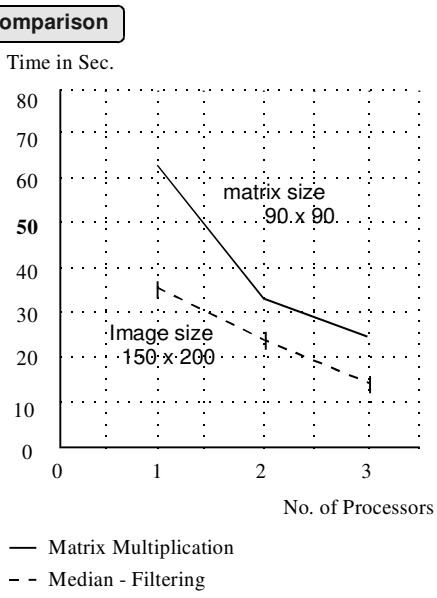

Figure 3: Number of PE's Vs. Time

A considerable increase in performance is observed for the problems: parallel implementation of median filtering of a given image and parallel matrix multiplication. For tightly coupled applications, because of the network delays and synchronization costs, it is observed that the developed parallel environment does not suit well.

5 Conclusions

The present work supports developing and running parallel programs on a LAN running NetWare. The work mainly concentrates on giving support to two parallel programming paradigms: process parallelism and data parallelism. The libraries are totally portable to work on different networks running on MSDOS operating system running over Novell NetWare network operating system.

POENDOM supports distributed processing on an existing network of computers. Therefore, it can be used as a cost effective model for developing and testing distributed processing in academic and re-

search institutions. Various loosely coupled applications developed and executed on POENDOM have shown improved performance. This has demonstrated the suitability and effectiveness of POENDOM for distributed computing.

References

1. Kai Hwang and Faye Briggs, *Computer Architecture and Parallel Processing*, McGrawHill Computer Science Series, 1985.
2. T.Agerwala, J.L.Martin,J.H. Mirza et. al., *SP2 System Architecture*, IBM Systems Journal 34, Number 2, 152 - 184 pp.
3. Thomas E. Anderson et al., *A Case for NOW (Networks of Workstations)*, IEEE Micro, 54 -64p, Feb '95.
4. Tanenbaum, A.S. et. al., *Experiences with the Amoeba Distributed Operating System*, Communications of ACM, Vol. 33, 1990,pp:46 -63.
5. G.A.Geisr, A.Beguelin, J.J. Dongna and V.S.Sunderam, *PVM :Parallel Virtual Machine* - A User's Guide & Tutorial for Networked Parallel Computing, The MIT Press, Cambridge.
6. Message Passing Interface Forum: *MPI : A Message-Passing Interface Standard*, International Journal of Supercomputer Applications 8,No. 3/4, pp: 165-414.
7. Gautam Barua, P.S.S.Rao, Anupam Sahai, *DC : A Distributed Programming Language*, CSI, Vol. 20, No. 2, 1990, pp: 1 - 13.
8. J.N. Magee and S.C.Cheng, *Parallel Algorithm Design for Workstation Clusters*, Software Practices and Experiences, Vol. 21(3), Mar'90,pp: 235 - 250.
9. Geetha S., Pankaj Kumar, Sandhya V., *Programming with PARAS*, Workshop on Parallel Processing, C-DAC at Hyderabad, April 1992.
10. C-DAC's Second Mission Team, *PARAM 9000 - Towards a National High-Performance Information Infrastructure*, Centre for Development of Advanced Computing, 1995.
11. Gunneswararao M, *Utilities for Parallel Programming on a LAN*, M.Tech.- Dissertation, Regional Engineering College, Tiruchirappalli.

About the Authors

Gunneswararao M is a Member of the Technical Staff, Operating System Group, C-DAC, Bangalore. He received the Bachelor of Engineering degree in Electronics and Communication Engineering from the University of Osmania in the year 1994 and the Master of Engineering degree in Computer Science from the Regional Engineering College, Tiruchirapalli in the year 1995. His research interests include Communication Protocols, Distributed Systems, and Signal Processing.

Rajkumar is a Member of the Technical Staff, Operating System Group, C-DAC, Bangalore. He received the Bachelor of Engineering degree in Computer Science from the University of Mysore in the year 1992 and the Master of Engineering degree in Computer Science from the University of Bangalore in the year 1995. He was awarded 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++. 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 the e-mail at raj@cdacb.ernet.in.

G. Mohan received the B.E. degree in Electronics and Communication Engineering from Regional Engineering College, Trichy, India, in 1986 and M.Tech. degree in Computers and Information Technology, from Indian Institute of Technology, Kharagpur, India in 1990. He is currently working as Lecturer in Department of Computer Science and Engineering, Regional Engineering College, Trichy, India and he is persuing the Ph.D. degree, specializing in parallel algorithms. His areas of interest include parallel algorithms and distributed computing.