Distributed Cluster Computing on High-Speed Switched LANs

by

Anindya Maiti

A Thesis submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Jointly with the departments of
Computer Science
Computer Engineering
Mechanical Engineering

of the

University of Manitoba, Winnipeg, Canada

Committee in charge:

Professor Peter C. J. Graham (Computer Science), Chair
Professor Robert W. McLeod (Computer Engineering),
Professor Robert W. Derksen (Mechanical Engineering).

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

DISTRIBUTED CLUSTER COMPUTING ON HIGH-SPEED SWITCHED LAN'S

BY

ANINDYA MAITI

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree

of

MASTER OF SCIENCE

Abstract

Distributed Cluster Computing on High-Speed Switched LANs

by

Anindya Maiti

©March 1999

In the area of high-performance computing, there is an ongoing technological convergence toward the use of distributed computing on networked workstation clusters. The emergence of high-performance workstations offering high-availability at relatively low-cost combined with recent advances in high-speed switched networks is motivating this change. As a result, a new computing paradigm centering around the use of distributed clusters of workstations (and/ or PCs) interconnected with low-latency, high-bandwidth networks (like ATM and switched Fast or Gigabit Ethernet) is becoming a commonplace high-performance computing infrastructure.

Parallel programming on these compute clusters using message-passing tools like PVM or MPI has many advantages including superior price-performance, scalability, and very large aggregate processing power and memory capacity. This computational paradigm has the potential to satisfy the computational demands of many large scientific and engineering applications which were historically achieved only with the use of traditional supercomputers or MPP systems.

In this thesis, the challenges that have to be met to bring the performance of cluster systems close to the traditional parallel machines are explored. In particular there is a need to benchmark the key metrics of network communication performance (bandwidth and latency) that are crucial for understanding the overall performance of distributed applications. This thesis provides a characterization and systematic analysis of the end-to-end and collective communication performance of three cluster interconnects,

viz. switched Ethernet, ATM and switched Fast Ethernet. Further two parallel applications that exhibit significantly different communication and computation patterns, viz. a matrix multiplication algorithm and a large fluid flow simulation problem, were implemented to serve as benchmarks for overall system performance evaluation. The results of the thesis experiments are reported and specific conclusions are drawn from them.

To my parents, Sabita and Binoy

and

my aunt, Sikha and uncle, Pradip

Contents

List of Figures

List of Tables

"No duty is more urgent than that of returning thanks."

- St. Ambrose.

Acknowledgments

Almost two years back when I wrote the proposal for this thesis, it was hard to justify that the work could be accomplished within the established boundaries of one department. This motivated me to pursue an Inter-disciplinary thesis, jointly with the departments of Computer Science, Computer Engineering and Mechanical Engineering. Since then I have been assisted and inspired by many individuals to whom I owe a great debt of gratitude.

First and foremost, I would like to thank the Chair of my Advisory Committee, Professor Peter Graham, for taking me on as a graduate student. He has been an ideal advisor to me in every respect. His valuable technical advice, and unstinting support has helped me see this thesis through its final completion.

I thank Professor Bob McLeod for serving on my thesis committee and for sharing his insights on ATM networking. I am also indebted to him for providing me access to the ATM testbeds at the VLSI lab of the University of Manitoba and at the Telecommunications Research Laboratory (TRLabs), Winnipeg. My appreciation to Dave Blight (Fujitsu Labs, Sunnyvale), Guy Jonatschick (VLSI Lab) and Dan Ericson (TRLabs) for helping me run the ATM testbeds.

I deeply appreciate the wonderful support of Professor Rob Derksen who also served on my thesis committee. His constant encouragement, professional advice, and technical inputs on numerical and CFD techniques were very valuable. I also acknowledge the generous support of Prof Ram Azad during the initial years and for

letting me use the Turbulence Lab at the University of Manitoba where I did most of my intellectual pondering!

My project collaborator, Ajay Pandya, had a great influence in shaping my thinking about networking and multi-threading (during our regular coffee-breaks at the University Center). A note of thanks, to all my collegues and friends at the University of Manitoba and the Indian Institute of Technology, Kharagpur for their support in various ways.

I sincerely thank Gilbert Detillieux and Tom Dubinski for their generous help on trouble-shooting problems on the switched Fast Ethernet testbed at the Computer Science Department of the University of Manitoba.

Research for this thesis was primarily supported through the Graduate Fellowship Awards from the University of Manitoba and TransCanada Pipelines Ltd., two Scholarship Awards from UMSU, and a grant (OGP-0194227) from NSERC. I gratefully acknowledge all of them.

I owe a special debt of gratitude to my aunt, Sikha, and uncle, Pradip, without whose co-operation and understanding I could not have done this thesis. Working on this thesis took time away from my two wonderful cousins, Sourabh and Soubhik. I deeply appreciate their patience and understanding during all those times.

Finally, my special thanks are due to my parents and my brother, Aniruddha, for their constant moral support from across the seas. They instilled in me the value of hard work and had given me the necessary education to get this far.

March, 1999.

"The last thing one discovers in writing a book is what to put first".

- Blaise Pascal.

Chapter 1

Introduction

1.1 Motivation

Grand challenge[1] computations today are becoming multi-disciplinary, combining computational techniques useful in analyzing a number of individual areas such as computational fluid dynamics (CFD), structural mechanics, electromagnetics, controls, acoustics and numerical optimization [1]. These computationally-intensive engineering applications have high requirements for computer processing speed and storage. For instance a state-of-the-art turbulent flow simulation of a Harrier jet fighter operating in-ground effect requires approximately 2.8 million computational points, 20 Mwords of run-time memory, and about 40 hours of CPU time on a Cray Y-MP running at a sustained speed of 160 MFLOPS [4].

The computational demands of such large scientific and engineering applications were historically achievable only with supercomputers, main-frames and more recently massively parallel computers. Unfortunately for many potential users of large-scale scientific computing, traditional parallel computers are often too expensive. Thus

[1] A grand challenge is a fundamental problem in science or engineering with broad applications whose solution might be enabled by future high-performance computing techniques.

their availability is not as prevalent as workstations or high-end personal computers. In the area of high-performance computing there is currently an ongoing technological convergence toward the use of clusters interconnected by high-speed switched local area networks (LANs) like Asynchronous Transfer Mode (ATM) and switched Gigabit Ethernet. At the same time the ever-increasing microprocessor speeds have led to the mass-market of inexpensive workstations of very high performance levels (see Figure 1.1). In the wake of this technological change the *network of workstations* (NOWs) model was developed [10]. Computing over such a fast networked cluster, using message-passing software tools like Parallel Virtual Machine (PVM) [5] or the Message Passing Interface (MPI) [6] offer advantages including superior price-performance, availability, scalability, and large aggregate processing power and memory capacity. For these reasons this computational paradigm is fast becoming an ubiquitous computing infrastructure for large-scale scientific applications.

Distributed cluster computing is the new wave in high-performance computing. From IBM to Microsoft, Compaq to Sun to Intel, virtually every large computer company is now gearing up to use clustering for high-availability and/or high-performance computing [2]. The emergence of clusters is making supercomputers and large massively parallel systems an extremely endangered class of machines. But what are *distributed clusters*? In the context of this thesis, clusters are defined to be a large set of standard desktop computers (Workstations, Personal Computers or even Symmetric Multiprocessors) connected through switch-based LANs, with software support forms a single large unified computing resource. However this definition of cluster is closely glued to the NOW concept (in contrast to the full-system cluster concept[2]). Rationalization for the factors that motivated the use of the cluster computing paradigm is given next.

[2]Clusters of Workstations, or COWs are typically distinguished from NOWs by the use of dedicated hardware and a single Operating System image across homogeneous processors.

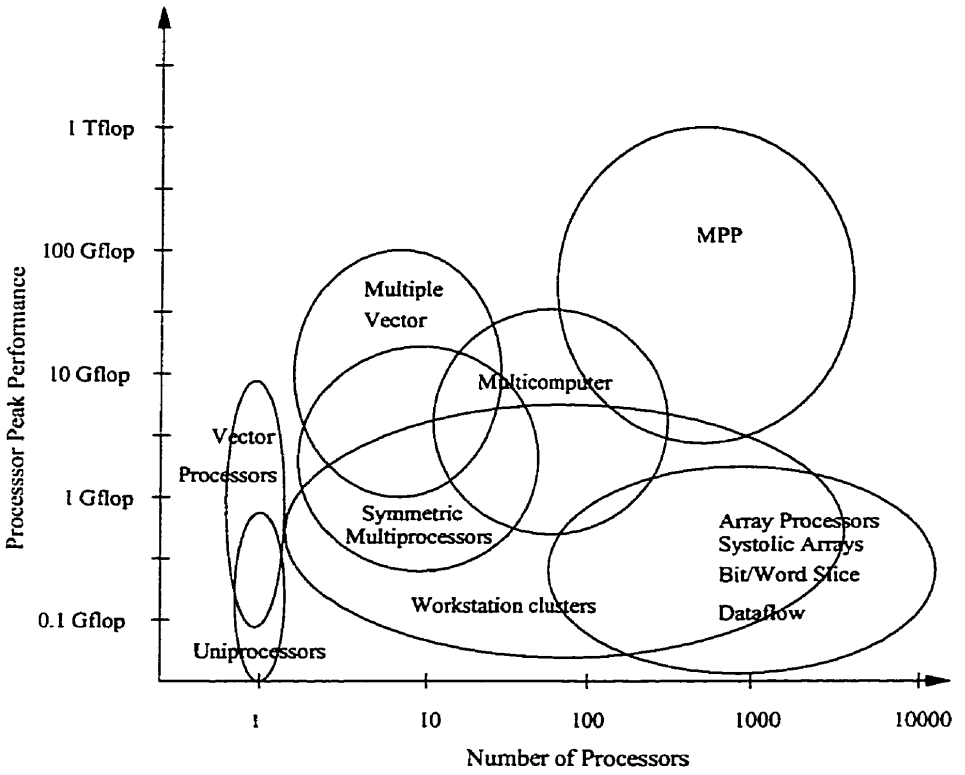


Figure 1.1: Taxonomy of modern parallel computers based on processor/ performance (Adopted from Lau, L. [3]).

Choice of Workstation Clusters

Rationale. A workstation cluster emulates a parallel machine by providing a shared pool of the collective resources of the machines including processors, memories, and disks. With workstations[3] becoming less expensive and more powerful in terms of processor performance and with significant network communication bandwidth improvements, clusters offer an attractive, low-cost alternative to traditional parallel machines for high performance computing [7]. High-end personal computers now include full-function operating systems and local-area networking and thus may participate effectively in clusters.

[3]The term workstation here refers to a generic, commodity computing resource typified by any sort of desktop machine.

Choice of Distributed Computing

Rationale. Distributed computing allows the solution of large computational problems by making collective use of the aggregate memory and computing power of several computers connected by a network. The most important factor favoring the choice of distributed computing is its low cost relative to the computational power available. In many organizations, workstations and the network often sit totally idle at night and on weekends and significantly idle even when in use (some estimates suggest as much as 80% overall idle time [8]). Distributed computing using the NOW concept exploits this existing hardware to provide a collective computational resource that may exceed the power of many high-performance computers.

Choice of PVM

Rationale. In distributed computer systems message passing constructs are necessary for inter-processor communications. Software tools and communication libraries like PVM or MPI provide the functions of message routing, data conversion, task synchronization and scheduling across a network of potentially heterogeneous computer architectures. In the distributed parallel computing community PVM is well established as a *de facto* standard message passing system. The availability of PVM in the public domain and its capability to run user programs over ATM networks by using fast message passing routines built on top of Fore Systems' ATM API, were the main motivations for using it as the parallel programming environment in this thesis.

PVM has also been developed to run on Win32 and Win NT platforms. Hence it can be used for clustering computers running Windows NT/95 as well as Unix variants. The other factors that favor PVM over MPI for using NOWs are PVM's better support for heterogeneity between different hosts and the provision of dynamic resource management and process control functions. MPI on the other hand is better suited to use on large collections of homogeneous processors and offers a somewhat higher level parallel programming environment.

Choice of Fast Switched Interconnects

Rationale. Parallel systems with their better interconnect hardware and high-speed switching techniques (e.g., worm-hole routing [12]) provide message passing bandwidths and latencies which are not achievable with NOWs constructed using traditional bus-based LANs where peak network capacity is limited to 10 Mbps (Ethernet) or 100 Mbps (FDDI). The bandwidth requirements can be improved to several gigabits/sec through the use of third generation [13] high-speed *switch-based* network architectures, such as the HIgh Performance Parallel Interface (HIPPI) [14], Fiber Channel Standard (FCS) [15], Fast Ethernet, ATM [16], [17], [18], the Scalable Coherent Interconnect (SCI) [29], and Gigabit Ethernet [46].

Switching technology (frame/cell switching) increases the efficiency and speed of networks by providing multiple, often dedicated, paths between processors instead of using a single shared medium. Recent advances in low-latency, high-bandwidth, cut-through switches like Myrinet [22] represents state-of-the-art and enable NOWs to communicate at full-duplex rates of 1.28+1.28 Gbps. Increasingly, most networks being deployed today employ switching technology.

Among the existing switch-based networks, ATM has become a well defined network standard. It is a connection-oriented service with high data transfer rates (beginning at OC-3, 155 Mbps) and is based on fast cell switching which is suitable for a wide range of applications. The ready availability of ATM local area switches and interface cards for most workstations, its wide-spread acceptance in the network community as a medium for high-speed local area networking [17], [19] and its significant performance advantages in PVM cluster computing [9], [20] are the major reasons for its choice as one of the high-speed network platforms in this thesis. For a comparative study, Fast Ethernet which also has a high data-transfer rate (100 Mbits/s) but utilizes the frame-based technology of Ethernet is also used as a cluster interconnect. For purposes of comparison with earlier work, experiments using traditional switched and unswitched (10 Mbps) Ethernet LANs were also conducted.

1.2 Problems and Challenges

Distributed cluster computing over a network of workstations may appear to be a viable platform for low-cost, high-performance computing in comparison to Massively Parallel Processors (MPPs). But to bring cluster computing performance closer to that of more tightly coupled parallel machines, three fundamental problems/challenges need to be addressed.

- **Challenge # 1: High communication Latency :** Reducing the communication latency between nodes has been the *Achilles' heel* of distributed cluster computing. It is primarily in the interconnection technology that MPP systems have a leading edge over NOW/COW based cluster architectures. The traditional hardware and software LAN technology was not developed for parallel processing and hence has inherently high communication overheads. For instance, the classical Ethernet-interconnected cluster typically achieves communication latencies of about 1000 μs and peak bandwidths of about 1.2 Mbps, whereas the corresponding values for a Cray T3D are 2 μs and 300 Mbps while for an IBM SP2 they are about 35 μs and 100 Mbps respectively.

  Previous work by Martin, *et al.* [27] on the effects of communication latency, overhead, and bandwidth in cluster architectures indicated that it is imperative to improve the performance of the communication system rather than invest in increasing machine performance. They also showed how improvement of communication performance improves the overall performance of most non-trivial parallel applications. Communication performance will be good only if the underlying cluster interconnect has low-latency and high-bandwidth. The requirements of scalable network bandwidth and low latency are addressed in this thesis by using ATM and switched Fast Ethernet networks.

- **Challenge # 2: Network hardware overhead :** High-speed cluster interconnects like ATM provide high link bandwidths but may *not* always result in

faster cluster communication. The communication performance relies on many factors including the overhead of using commodity network interfaces, the effects of message size, bandwidth, latency, communication patterns and network congestion. High-speed networks like ATM were not initially designed for use in a LAN environment. But when ATM is used as a cluster interconnect in LANs, there are high overheads at the network-host interface cards and the switches. Therefore the current high-speed network hardware technology has to be fine-tuned to overcome these overheads. Since improvement of the high-speed network hardware is beyond the scope of this thesis, no attempt was made to resolve this problem.

Despite this, there has been extensive research on the design of network interface hardware to achieve low-overhead communication. Some of these techniques include: designing high-performance network interfaces [30], integrating message transactions into the memory controller [31] or the cache controller [35], incorporating messaging deep into the processor [36], integrating the network interface on to the memory bus [37] rather than the I/O bus, providing dedicated message processors [22], providing various kinds of bulk transfer support [38], and supporting reflective memory operations [37].

- **Challenge # 3: Communication software overhead :** Using a low-latency, high-bandwidth cluster interconnect partially solves the communication problem. But delivering high performance to applications requires that the communication software be capable of matching the network performance. Most communication software tools (like PVM, MPI) were developed with the assumption that the underlying network is slow and unreliable. For these reasons and for portability, the BSD Socket programming interface was used as the interface between the network medium and the message passing system. The use of these communication protocols results in high overhead due to the high level of the protocols, and due to the use of operating system controlled access to

network interfaces and their device drivers.

For the presented ATM experiments, this problem was addressed by using an implementation of PVM that operates directly over the ATM API instead of using the BSD Socket interface. Since the ATM API resides at a lower layer in the protocol stack its overhead should be less. There have also been other efforts to reduce communication software overheads. These include the use of lean communication software layers such as Fast Messages [39], Active Messages [40], U-Net [44], VMMC [41]. Time constraints precluded the implementation of a new version of PVM incorporating any of these systems.

In addition to these problems there are other issues in distributed cluster computing that need mention here. The introduction of high-speed LAN technologies to cluster architectures, opens up other issues like system scalability, reliability, affordability, and software affordability that also need to be addressed. LAN environments are also progressively becoming *heterogeneous* due to user preferences and the commodity nature of workstations/personal computers. Cluster performance is also affected [45] by heterogeneity. This is reflected in terms of the varying speed and communication capabilities of the workstations and the co-existence of multiple heterogeneous network architectures.

1.3 Objectives and Structure of the Thesis

In this thesis, the primary objective is to develop a systematic understanding of three different distributed cluster systems connected by high-speed networks (switched Ethernet, ATM and switched Fast Ethernet). The emphasis is on studying the performance impact on two typical scientific problems of varying granularity, implemented on each cluster architecture. Finally, considering the problems posed by the above challenges two performance improvement techniques that can be applied to such systems are explored. The results of this study will help in designing parallel algorithms

suited to fully exploit the cluster architectures described.

The rest of the thesis is organized as follows. In Chapter 2, the various cluster architectures, interconnects and communication systems are reviewed and related research work on improving the performance of such distributed systems is discussed. In Chapter 3, the research problem is described and some of the parallel programming models and paradigms are reviewed. Two benchmark problems that were implemented and their parallelization techniques are then described.

Chapters 4 forms the core of the thesis and reports the experimental results. First the three different cluster test-beds viz. switched Ethernet (also unswitched Ethernet), ATM and switched Fast Ethernet that were used for the experiments are described. For each of the cluster systems, the performance in terms of communication, computation and cluster heterogeneity are then evaluated.

In Chapter 5, the conclusions of this study are presented and some performance enhancement techniques (with partial implementation results) are suggested. Finally, some directions for future research are outlined.

Appendices with details on some of the topics discussed are also included. These include the algorithmic details of the matrix multiplication and the Jacobi iteration technique, in Appendix I. The ATM Protocol architecture and the PVM system are described in Appendix II. The proposed PVM scheduling model and the related research in scheduling and fault-tolerance in PVM are discussed in Appendix III. The mathematical formulation of the numerical technique and the computational scheme used for implementing one of the benchmark problems (flow simulation) are described in Appendix IV along with the results.

"The whole past is the procession of the present."

- Thomas Carlyle.

Chapter 2

A Brief Review of Cluster Computing

Most research efforts on high-performance computing on distributed cluster architectures are focused on clusters of either high-performance workstations and high-end PCs or traditional, large multiprocessor systems. The purpose of this chapter is to introduce the reader to the different cluster architectures, cluster interconnects and cluster communication systems (both traditional and those that are currently state-of-the-art). Related research works are then discussed in the context of improving cluster performance through the design of better cluster interconnect hardware and communication software.

2.1 Cluster Variants

Pfister [2] defines *cluster* as a parallel or distributed system that consists of a collection of interconnected whole computers which is used as a single, unified computing resource. Cluster configuration provides a powerful bridge between computing on a single machine and computing on supercomputers. Cluster technology offers low-cost, high-performance, highly available resources for computer users.

2.1.1 Multiprocessor Clusters

Traditional parallel computers can be categorized into three broad architectural classes: Massively Parallel Processors (MPP), Symmetric Multiprocessors (SMP) and Scalable Parallel Processors (SPP).

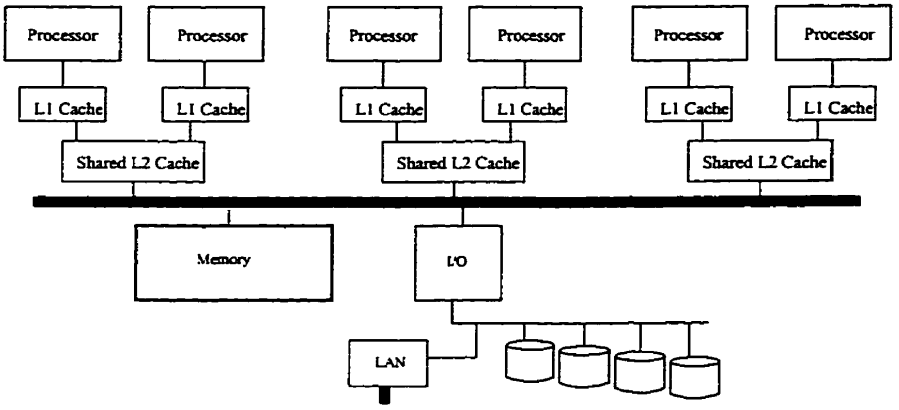


Figure 2.1: Diagram of a typical Symmetric Multiprocessor (SMP).

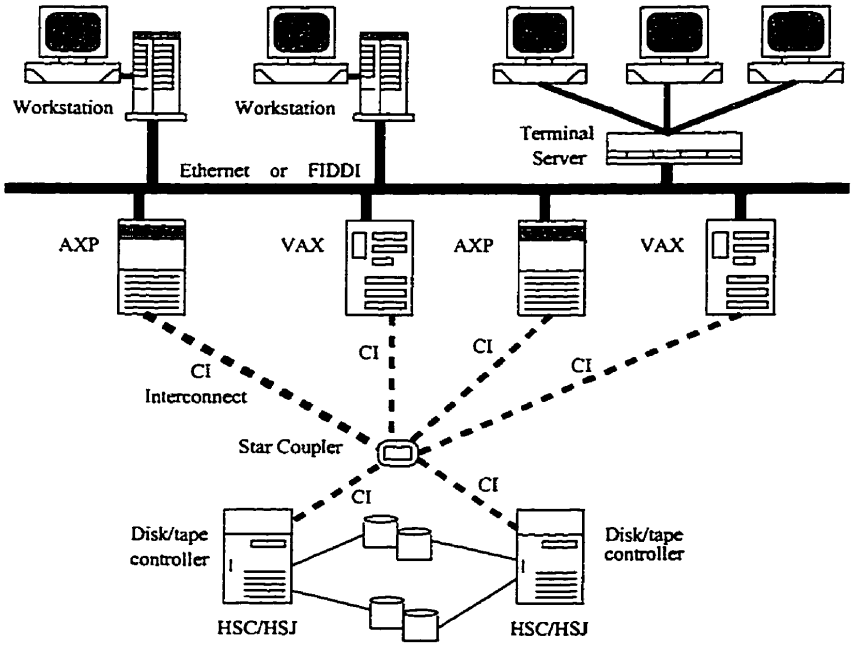


Figure 2.2: Digital Open VMS Cluster [2].

MPP systems are a set of loosely-coupled processing elements, each node having its

own resources (bus, memory, disks, I/O system) and hence represent a *share nothing* architecture. These systems usually have greater aggregate cpu-memory bandwidth, memory size, I/O bandwidth and interconnect bandwidth to overcome the inefficiencies of message-based communication thus allowing significantly higher scaling. They have the potential for very large parallelization (can theoretically use more than 10,000 cpu-nodes). Examples of MPP systems include the nCube, the Intel Paragon, the CM5, the Cray T3D and T3E, etc. There are some drawbacks to these systems. Each processor must use a message-passing scheme, analogous to network packets to access memory outside its own RAM. Programming is therefore more difficult, requiring embedding of message-passing constructs. Further the source code may be hardware dependent if standard software tools like PVM or MPI are not used.

In contrast to MPPs, Symmetric Multiprocessor (SMP) systems consists of a set of tightly-coupled multiprocessors, all of which share the same global resources (bus, memory, I/O systems). Hence, such machines are often defined as *share everything* architectures. All the processors of an SMP are symmetric (see Figure 2.1). That is to say they have exactly the same abilities. Examples of SMPs include: the Cray CS6400 Enterprise Server (up to 64 CPUs), the SGI Power Challenge (up to 18 CPUs) and the HP T-Class Servers (up to 14 CPUs). These systems are easy to program and by using a High Level Optimizer (HLO) can easily produce parallel code without message-passing instructions. Adding more processors, however, increases the memory traffic leading to rapid saturation of the system bus[1].

The third class of parallel architecture are the so-called Scalable Parallel Processing (SPP) systems. These systems use a two-tier memory hierarchy. The first memory tier consists of a hypernode, which is, essentially, an SMP system, complete with multiple processors and their globally shared memory. Large SPP systems are built by connecting several hypernodes (the second memory tier) via a high-speed

[1]Bus traffic may be reduced by adding large caches to every processor but this increases the cost per processor.

interconnect so that this tier appears logically as one global shared memory space to the nodes. Node addition does not severely increase memory traffic due to the high-speed interconnect and since only updates to keep memory coherent among nodes are required. Such systems thus have scalability similar to MPP systems. Examples of SPP systems are the Convex SPP1200 CD (up to 16 CPUs), the Convex SPP 1200 XA (up to 128 CPUs) and the SGI Origin-2000 (up to 256 CPUs). More processors may be added to these systems but there is still one unified memory space, even though the RAM is physically located in different machines.

2.1.2 Workstation Clusters

The incredible progress in microprocessor speeds together with the demand for workstations in the market has resulted in an enormous price/performance improvement for workstations. These advancements in computer technology have led to the concept of *workstation clusters* or *networks of workstations* (NOW[2]) [10] which as the name suggests are a cluster of general-purpose workstations connected by a high-speed network (like ATM or Myrinet). NOWs offer an attractive alternative to expensive supercomputers and specially designed multiprocessor computer systems. Using an installed-base of networked workstations there is the potential to scavenge unused cpu-cycles from the workstations for collective use in parallel processing.

The NOW project currently[3] aims to use desktop hardware and software and switch-based LANs as the building blocks for scalable cluster computing systems. However programming in a NOW environment presently requires algorithms that are extremely tolerant of load balancing problems and large communication latencies. These problems in NOW systems are currently being addressed through the development of certain key technologies. *Active Messages (AM II)* [40] is being developed to

[2]The NOW Project is conducted by the Computer Science Division at the University of California, Berkeley.

[3]The 1997 Project report of DARPA ITO Sponsored NOW Research at Berkeley is available at http://www.darpa.mil/ito/Summaries97/C137_0.html

provide fast, reliable communication, avoiding OS intervention through a technique known as network virtualization. The use of *Fast Sockets* [34], a user-level library, will improve the small message performance of high speed networks like Myrinet and ATM. NOWs can also use a global layer resource manager such as Global Layer Unix (*GLUnix* [10]) to coordinate access to the system's resources for both sequential and parallel jobs. The file systems project of NOW will provide Serverless File Service (*xFS* [10]) for co-operative caching by making use of the system disks for storage and memory. While network RAM [10] can make the system's DRAM globally available via a fast interface and thus increase the peak memory size available on a machine.

Incorporation of these key technologies will potentially make NOWs the *de facto* standard for low cost supercomputing that scales with demand and shows promise to replace the large traditional multiprocessor systems that are both expensive and difficult to upgrade.

2.1.3 Other Clusters and Related Research

Besides MPP and NOWs there are various other cluster architectures which merit discussion. One such system is the *Beowulf* cluster that emerged from the Beowulf project [43] at CESDIS (Center for Excellence in Space Data and Information Sciences, Maryland). In the taxonomy of parallel computers, Beowulf clusters fall somewhere between low-end MPPs and high-end NOWs. The Beowulf project was a NASA initiative to explore the potential of using a "pile-of-PCs" for parallel computing and to develop the necessary methodologies to apply these low cost system configurations to NASA's computational requirements in the Earth and Space Sciences.

The Paderborn Center for Parallel Computing has recently installed a massive SCI-based cluster named *PSC*[4] with distributed memory using 192 Pentium II PCs arranged as a 8×12 2D torus with distributed switches and a high-speed SCI communication network.

[4]PSC details can be found at the site http://www.uni-paderborn.de/pc2/systems/psc/

Cornell's U-Net architecture [44] provides a user-level network interface for clusters of workstations. It offers low-latency and high-bandwidth communication over ATM networked workstations and switched Fast Ethernet connected PCs (running Linux/ Windows NT).

Examples of several other full system clusters include: the IBM System/390 Parallel Sysplex and SP systems, Microsoft Cluster Services (Wolfpack), DEC OpenVMS Cluster (see Figure 2.2) and Memory Channel, Tandem ServerNet and Himalaya, Sun Microsystems's Fool Moon Cluster and Silicon Graphics's Cellular IRIX.

2.2 Cluster Interconnects

The cluster interconnects that are most used today include Ethernet [23], Fiber Distributed Data Interface (FDDI) [25], High Performance Parallel Interface (HIPPI) [14], Fiber Channel [15], Fast Ethernet [25] and Asynchronous Transfer Mode (ATM) [16]. The *pros* and *cons* of the use of each interconnect for high-performance distributed cluster computing and the related research are now discussed.

2.2.1 Ethernet

Ethernet [23] is a frame-based LAN technology that is capable of transmitting data at speeds of 10 Mbps. It is very widely used because it strikes a good balance between speed, cost and ease of installation. The first formal 10 Mbps Ethernet specification was published in 1980 by a consortium that included Xerox, DEC and Intel. This formed the basis for the IEEE 802.3 standard for CSMA/CD (Carrier Sense Multiple Access with Collision Detection) which is the most commonly used medium access control technique for bus/tree and star topologies.

Drawbacks: Although Ethernet has been the most popular interconnection network it is a bus-based shared medium architecture. The network capacity is shared among all the participating nodes and hence with increases in the number of nodes,

the network saturates [24] (i.e. there is no bandwidth guarantee). It is also considered less scalable and reliable than ATM. The introduction of switches has improved Ethernet's network performance by reducing media-sharing (i.e. separating the collision domains) and maintaining multiple simultaneous links between various ports. However, at a particular link from the switch the Ethernet segment may still be shared.

2.2.2 ATM

Asynchronous Transfer Mode (ATM) [16], [18] also known as cell relay, was developed originally as a Broadband ISDN (Integrated Services Digital Network) in 1986 to carry voice data. The first ATM specifications were released by the ATM Forum in 1992. It is a fast, virtual circuit-oriented, packet-switched network where data is fragmented into fixed size 53 octet packets called *cells*, which consist of 5 octet header and a 48 octet information field. Since ATM uses fixed-length packets, relay switches can process cells in parallel to achieve high data rates. The designated customer access rates in B-ISDN are OC-3 (155.52 Mbps) and OC-12 (622.08 Mbps). Again, since ATM is asynchronous, it can transfer cells as required by the application, making data transfer more efficient. This is in contrast to frame-based networking technologies, which transfer frames as called for by the transmission system. The details of the protocol architecture of ATM are discussed in Appendix II.

ATM offers a reliable and flexible transmission mode. It is flexible since its layered architecture lets the network efficiently carry data, voice and video (multimedia) simultaneously. ATM also allows very efficient utilization of network bandwidth, as it statistically multiplexes multiple logical connections (virtual channels) over a single physical interface thereby allowing the total bandwidth available to be dynamically distributed among a variety of user applications. It is also reliable since it is connection-oriented implying that an end-to-end connection is set up prior to transmission. This enables it to provide a way to guarantee delivery (called Quality of

Service, QoS) with a negotiated set of parameters.

Drawbacks: Since ATM is connection-oriented, it does not interoperate easily with the installed base of packet-based traditional Ethernet LANs. Thus, interaction or compatibility between an end system on an ATM network and an end system on a legacy LAN is achieved through the use of the ATM Forum's LAN Emulation (LANE) protocol. With all its advanced capabilities, ATM is a more complex and expensive technology than traditional Ethernet.

2.2.3 Fast Ethernet

The Fast Ethernet standard (IEEE 802.3u) raises the Ethernet speed limit from 10 Mbps to 100 Mbps with only minimal changes to the existing cable structure (twisted pair or optical fiber). Fast Ethernet (100 Base-T) uses the same CSMA/CD protocol as 10-base-T Ethernet and can provide full-duplex communication (200 Mbps aggregate bandwidth - 100 Mbps each way). It also uses *auto-negotiation* to automatically adopt the highest possible communication speed found at both ends of the cable. This facilitates easy migration from legacy Ethernet installations.

Drawbacks: Fast Ethernet has the same drawbacks as 10-base-T Ethernet including less scalability and reliability and lack of Quality of Service (QoS) when compared to ATM.

2.2.4 Other Interconnects and Related Research

Other prominent network technologies used in LAN environments include FDDI [25], the Scalable Coherent Interface (SCI) [29] and Gigabit Ethernet [46].

Fiber Distributed Data Interface (FDDI) is a mature backbone technology, based on a shared fibre optic medium architecture with data transfer rates of 100 Mbps. Unlike Ethernet, FDDI is fault tolerant (does self-management) and reliable (guarantees access by using a token passing access method). It is, however, comparatively expensive and significantly more complex to manage than Ethernet, and hence is not

widely deployed to the desktop.

Gigabit Ethernet is an emerging standard for network connectivity and is essentially an extension of the 10-BASE-T Ethernet and 100-BASE-T Fast Ethernet but has a data-rate of 1000 Mbps. It uses the same CSMA/CD protocol but adds carrier extension and packet bursting to increase the bandwidth efficiency. The high data-rates of Gigabit Ethernet makes it a competing technology with ATM. It is less expensive (by 50%) than OC-12 ATM, in part because of the volume of Ethernet-based products in the marketplace. In addition Gigabit Ethernet is finding wide acceptance because Ethernet is a simple and well understood technology.

However, Gigabit Ethernet is not as scalable and reliable as ATM. Further unlike ATM it does not have explicit and guaranteed QoS features. The IEEE and IETF are, however, currently working on RSVP (the resource reservation protocol) that will let Ethernet LAN users reserve bandwidth in advance.

Related Research: Research on performance improvement in distributed clusters on high-speed networks is actively being carried out at a number of institutions. The most notable of these is the Distributed Multimedia Research Centre (DMMC) of the University of Minnesota. Lin, *et al.* [20], from DMMC, have evaluated the performance of ATM local area networks for PVM communications. In their work they compared four different Application Programming Interfaces (APIs) such as Sun Microsystems's Remote Procedure Calls (RPC), the BSD socket programming interface, the PVM message passing library, and Fore System's ATM API. They concluded that on a NOW system the BSD sockets and Sun RPC/XDR were not good for implementing high performance computing applications. Even porting PVM over to an ATM LAN by using the BSD socket interface has high communication overhead compared to Fore Systems' API.

Chang, *et al.* [26] re-implemented the PVM message passing library using the Fore Systems' ATM API and studied the resulting PVM communication performance. Results from their experiments indicate that high-speed networks like ATM can yield

significant performance gains at the application level compared to conventional Ethernet networks. However the performance improvement is much less than the raw available bandwidth of the ATM network platform. Hsieh, *et al.* [32] studied the communication performance of HIPPI LANs for distributed applications. They reported enhanced PVM performance on HIPPI and Ethernet. In line with the work of Chang, *et al.* [26], they implemented PVM over HIPPI and Ethernet by using Hewlett Packard's Link Level Application (LLA) programming interface, an API similar to the Fore Systems' API for ATM networks. The overhead of PVM's default protocols, namely UDP (for daemon-daemon communication) and TCP (for task-task communication) was reduced significantly by the use of these lower layer protocols (LLA API).

Huang, *et al.* [21] investigated the performance of collective communications (e.g. multicast operations) across ATM networks by developing a thread-based software testbed. Comparison of PVM communication performance over ATM metropolitan area networks with Ethernet and FDDI LANs were made by Iannello, *et al.* [49]. Performance comparison of TCP/IP and MPI on FDDI, Fast Ethernet and Ethernet at different network loads were also studied by Nog, *et al.* [58]. Simon, *et al.* [42] presented the performance results of a multiprocessor cluster consisting of personal computers connected with PCI[5] communication cards based on the SCI[6] standard. They implemented PVM using SCI hardware on the Linux and Windows NT operating systems.

2.3 Cluster Communication Systems

Cluster software systems enable an interconnected collection of independent, possibly heterogeneous, computers to appear as a single *virtual* computational resource. To

[5]Peripheral Component Interconnect.

[6]Scalable Coherent Interface is based on IEEE standard 1596, 1992.

the application programmer this resource appears as a potentially large distributed-memory virtual computer. In the distributed/parallel programming community the most popular cluster communication systems are PVM [51], [55], [56] and MPI [57], [6].

2.3.1 MPI

The Message Passing Interface (MPI) [57] is the latest development in message passing systems. The main motivation for its development was to create a standard so that each MPP vendor would not create their own propriety message-passing API. Compared to PVM, MPI has a much richer set of point-to-point and collective communication functions. This is an important feature particularly for those algorithms which are dependent on the existence of special communication options or a logical communication topology. MPI introduced the concept of a *communicator*[7] to bind a communication context to a group of processes. This context is assigned by the operating environment (and not by the user) to provide support for the design of safe and concurrently inter-operable parallel software libraries.

Drawbacks: MPI does not support the concept of a *virtual machine* but instead provides a higher level of abstraction in terms of message-passing topology. MPI does not support inter-language communication and does not have robust fault-tolerance like PVM. On the other hand, PVM contains resource management and process control functions for creating portable applications that can run on heterogeneous clusters of workstations and MPPs. The University of Tennessee, Knoxville and Oak Ridge National Laboratory are currently trying to merge PVM and MPI under the PVMPI project [69]. Its objective is to utilize both the *virtual machine* features of PVM and the message passing features of MPI.

[7] A communicator is a *communication domain* that defines a set of processes that are allowed to communicate between themselves.

2.3.2 PVM

Parallel Virtual Machine (PVM) was developed [56] in 1990 at the Oak Ridge National Laboratory and is now a widely-used message-passing software system. It provides a unified computational framework for a network of heterogeneous computing resources. The cynosure of the design of PVM was the notion of a *virtual machine* which is a dynamic collection of *heterogeneous hosts* connected by a network that appears logically as a single large parallel computer. It is portable to a wide variety of different machine architectures and operating systems, including workstations, supercomputers, multiprocessors and personal computers. PVM is composed of a programming library (of interface routines) and manager processes (i.e. daemons) that transparently manage all message routing, data conversion, and task scheduling across a distributed cluster.

In PVM the user writes applications as a collection of cooperating *tasks*, each task written in a procedural host language (usually C or Fortran[8]) with embedded PVM primitives. The applications are then compiled for each architecture type in the network of hosts. PVM provides primitives for such operations as point-to-point data transfer, message broadcasting, mutual exclusion, process control, global sum, and barrier synchronization. It's message-passing primitives are oriented towards heterogeneous operation, involving strongly typed constructs for buffering and transmission.

The details of the PVM architecture, message-passing and the basic routines are described in Appendix II.

Drawbacks: PVM unlike MPI does not support logical communication topologies which are important for certain parallel applications. It does not have as rich a set of communications functions as MPI, and hence applications can not exploit special communication modes. PVM is not as established a standard as MPI is. The default scheduler that is embedded in PVM is based on a simple Round-Robin allocation scheme where newly created tasks are assigned to available processors in a cyclic fashion. It employs only a static load balancing scheme with no support for check-

[8]Presently C++, Java (JavaPVM) and Perl (Perl-PVM) can also be used with PVM calls.

pointing or task migration, i.e., once a job is assigned, it runs on the assigned processor until completion. PVM's global scheduler does provide primitives for remote task creation and inter-process-communication but it does not have support for intelligent dynamic scheduling or resource management. Its scheduling decisions are not based on idle processor selection or any other form of processor load information.

2.3.3 Other Systems and Related Research

Besides PVM and MPI several message-passing systems have been built in the past. The p4 system[60] developed at the Argonne National Laboratory is a portable parallel toolkit which provides (C or Fortran) functions for explicitly programming shared memory machines (using *monitors*) or distributed memory machines (using message passing). ParaSoft's Express [64] is another collection of tools for programming distributed memory multiprocessors, both true multiprocessors and networked UNIX clusters. ISIS [61] is a message-passing system for workstation clusters that can replicate data and processes in order to tolerate machine failures, and can also configure a new virtual machine and restart a whole application after a large crash. The Portable Instrumented Communication Library (PICL) [65] is a portable message-passing library designed to standardize message functions available on machines such as the Intel iPSC/2, the iPSC/860 and the Ncube/3200. Parform [63] is a parallel programming system that runs on a network of UNIX workstations. Some of the other message-passing systems include PARMACS [52], Zipcode [53], Chameleon [54], TCGMSG [66], APPL [68], Network Linda[9] [62] and POSYBL [67].

Additionally, there are several proprietary message-passing libraries like IBM's MPL (message passing library) for the RS/6000 based SP-2 multiprocessor system. There are also various free implementation of the MPI standard such as MPICH from Argonne National Laboratories and Mississippi State University, LAM from the Ohio Supercomputing Centre and University of Notre Dame, CHIMP from the Edinburgh

[9]Linda is a product of Scientific Computing Associates, Inc.

Parallel Computing Centre, MPI-FM from the University of Illinois, Concurrent Systems Architecture Group, and UNIFY, a subset of MPI within a PVM environment, from Mississippi State University.

Related Research: There has been considerable research effort in enhancing the PVM/MPI communication performance in distributed cluster environments. Work in this direction has been done by Zhou, *et al.* [9] who used the lower layer Fore Systems' ATM API to developed a faster message passing route named *PvmRouteAtm* to exploit the high bandwidth of ATM networks. Their experiments showed that the AAL-based *PvmRouteAtm* achieved higher bandwidths for large data sizes but failed to gain any latency improvement over its counterpart, the TCP-based *PvmRouteDirect.* In other work, Zhou, *et al.* proposed a light-weight process based implementation of PVM called LPVM [11]. It was designed to improve performance by adopting a multithreaded message passing system in place of the standard process based system. LPVM however lacks portability since it is essentially a *modified* version of PVM with added features (like thread safety) and with a different user interface. Hence programs already written for standard PVM could not be transparently ported to the LPVM system. It was originally implemented on a single SMP machine although current research is being done (at the Oak Ridge National Laboratory, Tennessee) to extend it to a cluster of SMPs with disjoint address spaces. Prior to LPVM's development another message passing library was proposed by Ferrari, *et al.* called TPVM [33] which was implemented on SMP clusters. It is also a PVM derivative but unlike LPVM it was built on top of the standard PVM system, a design which provides portability at the cost of added overhead.

2.4 Summary

This chapter started with a description of the traditional multiprocessor clusters and highlighted the technological shift toward more practical, cost-effective compute clus-

ters. Section 2.2 provided background material on the high-speed, switched networks commonly used for cluster interconnection and enumerated current research in the design of cluster-network interface hardware. It was observed how increases in the network bandwidth can have a potential impact on cluster performance.

Section 2.3, introduced the two most popular cluster communication systems, viz. MPI and PVM and focused on related research into similar systems providing low (latency) overhead message-passing. This motivated the use of a version of the PVM system in this thesis that has low protocol overhead.

"An invasion of armies can be resisted.
But not an idea whose time has come".
- Victor Hugo.

Chapter 3

Research Problem and Methodology

In this chapter the research problem addressed in the thesis is described and a rationale for conducting the work is given. In Section 3.2 some of the parallel programming paradigms and models used in this work are introduced. Finally, Section 3.3 provides a detailed description of the benchmark problems.

3.1 Research Problem

As observed in the previous chapters the ever-increasing microprocessor speed and the mass-market of inexpensive, high-performance desktop computers has lead to the use of clusters in high-performance computing. This development has been further motivated by the availability of high-speed switched networks.

In view of this new computational paradigm this thesis seeks to explore the use of future *heterogeneous* distributed clusters which it anticipates will be similar to the one shown in Figure 3.1. Existing LAN systems of high-end desktops connected either with Ethernet, ATM or Fast Ethernet and located in different buildings, can

all be integrated to form a unified large-scale computing system for high-performance computing. The incorporation of recent advances including *Active Messages* (network interface hardware), network RAM (network-wide resource management), serverless (parallel) file systems and global resource management (distributed scheduling) to such systems will result in a collective system that will cost-effectively support future high-performance computing.

In the research in this thesis, the three LAN systems shown in Figure 3.1 are segregated and studied in isolation. The goal of the research is to develop a systematic understanding of each system in terms of its communication and computational performance and also evaluate the architectural requirements of using distributed clusters for computationally intensive scientific and engineering applications.

The rationale for conducting this research is to determine the factors that affect achieving low-latency and high-bandwidths, the two primary performance bottlenecks of existing cluster systems. This study will contribute to an existing base of knowledge that will help in designing parallel algorithms for large scale scientific and engineering computations using cluster architectures.

In brief the objectives of this research are:

1. Performance evaluation of three high-speed switched LANs (Ethernet, ATM and Fast Ethernet).

2. Implement and study the performance of two typical parallel algorithms that vary in granularity.

3. Explore certain performance improvement techniques that can be applied to such systems.

Network performance evaluation involves measuring communication performance in terms of metrics like start-up latency, t_0, maximum achievable throughput, r_{max} and half-performance length, $n_{1/2}$. Performance evaluation also involves measuring the computational performance of parallel programs in terms of speed-up, S_p, total

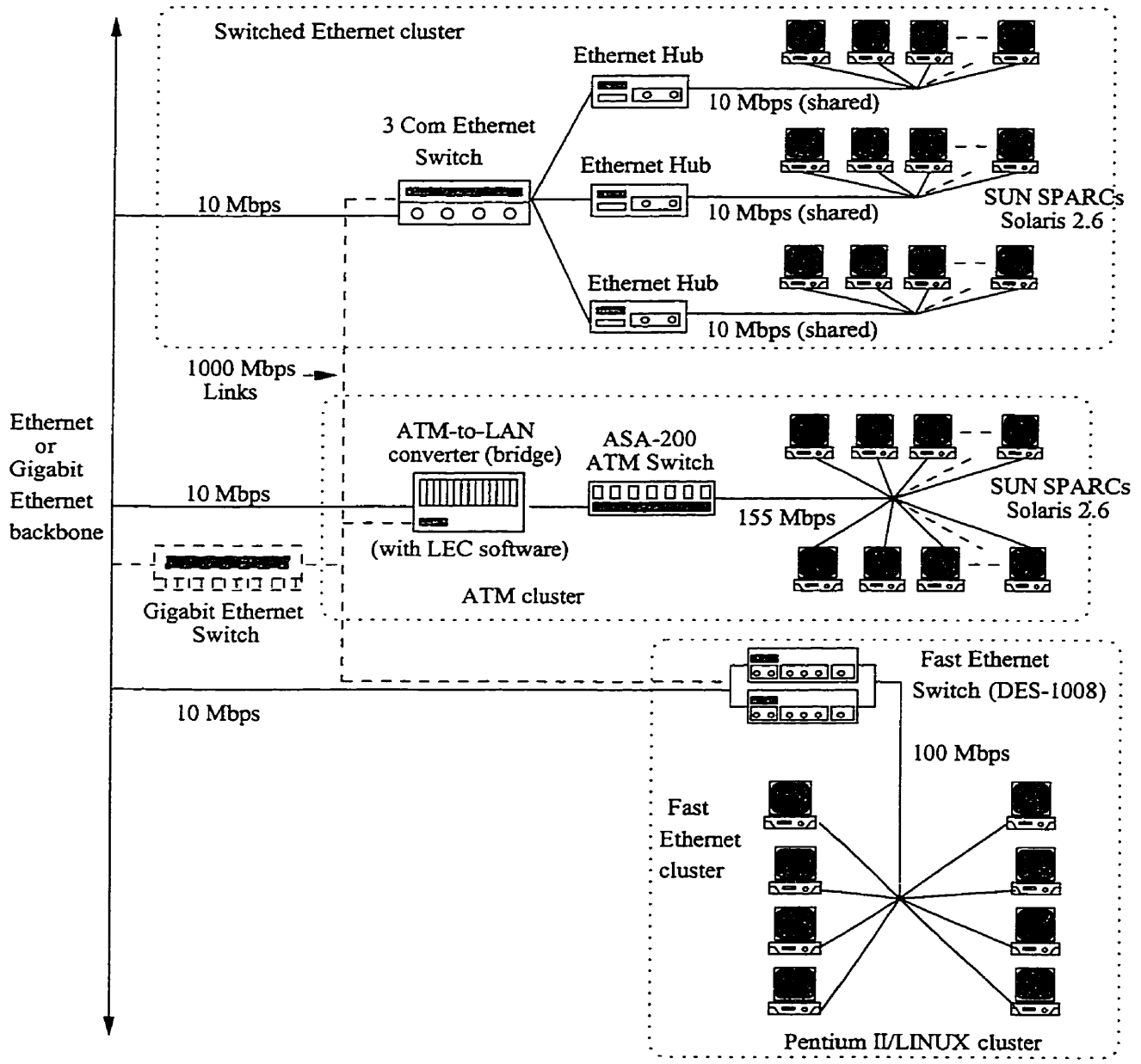


Figure 3.1: An example distributed cluster built from heterogeneous desktops.

efficiency, E^{tot} and algorithmic efficiency, E^{alg}. The effect of cluster heterogeneity is also studied.

The parallel algorithm implementations used in this work include a coarse grained problem, matrix-multiplication, and a fine grained problem, potential flow simulation over a cascade of airfoils.

Performance improvement techniques discussed include latency-hiding by overlapping computation with communication via multi-threading and load-balancing by incorporating dynamic scheduling into the PVM system.

3.2 Parallel Programming Paradigms

This section reviews the various parallel programming paradigms used for developing applications on traditional parallel architectures in general and distributed memory message-passing systems in particular.

Problem architectures

Based on the features of the problem structure most parallel applications can be classified into the following five problem architectures:

- *Synchronous.* In fully synchronous applications, all the processes start their operations at the same time in a lock-step manner (analogous to Single Instruction Multiple Data, SIMD) and are synchronized at regular points. Programming is easy as the partitioned subtasks are essentially identical.

- *Loosely synchronous.* These applications are similar to fully synchronous ones but the data elements are not identical. However parallelism is still achieved using macroscopic time synchronization.

- *Asynchronous.* Applications are asynchronous when they exploit functional (or data) parallelism that is irregular in space and time. Often used in loosely

coupled clusters and so need to have near optimal decompositions to minimize communications.

- *Embarrassingly parallel*[1]. These are ideal parallel applications that can be divided into completely independent parts which can be executed simultaneously. and require no inter-process communication during computation.

- *Metaproblems.* These applications be divided into an asynchronous collection of (loosely) synchronous components which can themselves be parallelized.

Programming paradigms

The natural structure of the parallelized algorithms and their communication topologies can be exploited by using algorithms that are based on any of the three fundamental parallel programming paradigms:

- *Crowd Computations.* This is the most common model where a collection of closely related processes typically executing the same code, perform computations on different portions of the workload and periodically exchange intermediate results. This paradigm is further sub-divided into two categories:

  - *Master-Slave.* In this model a separate control program called the *master* is responsible for process spawning, initialization, and collection and display of results. The *slave* programs perform the actual computations and their workloads are assigned either by themselves or their master (statically or dynamically). The matrix multiplication algorithm implemented in this thesis is based on this paradigm.

  - *Node-only*[2]. Multiple instances of a single program are executed in this model, with one process (typically the one initiated manually) taking on

[1] A term due to Geoffrey Fox; Wilson [28], 1995.

[2] This is a case of the Single Program Multiple Data (SPMD) model where a single source program is written and each processor will execute its personal copy of this program, although independently

the non-computational responsibilities in addition to contributing to the computation itself. The flow simulation algorithm implemented in this thesis uses this paradigm.

- *Tree Computations.* These computations spawn processes (usually dynamically as the computation progresses) in a tree-like manner and establishes a tree of parent-child relationship (as opposed to crowd computations where a star-like relationship exists). This paradigm is an extremely natural fit to applications where the total workload is not known *a priori*, for example, in recursive *divide-and-conquer* and alpha-beta search algorithms.

- *Hybrid Computations.* These are essentially a combination of the crowd and tree models. They possess an arbitrary spawning structure implying that at any point in time during application execution, the process relationship structure may resemble an arbitrary and changing graph.

Algorithmic Paradigms

Some of the commonly used parallel algorithmic paradigms are:

- *Domain decomposition.* This involves partitioning of the program data structure (given some constraints) and applying computational operations on the divided data structures concurrently. It is also called data partitioning and is an important strategy for scaling parallelism in many applications.

- *Functional decomposition.* In this paradigm the program is partitioned into independent functions that are executed concurrently (thus different tasks perform different operations). Parallelism is obtained by concurrent execution of

---

and not in synchronization. The source program is constructed so that parts of it will be executed by some nodes (say the master) and not others (say the slaves) depending on the identity of the node.

functions or by establishing a pipeline[3] (continuous or quantized) between them.

- *Divide-and-conquer.* This approach is characterized by dividing a problem into subproblems that are of the same form as the larger problem. Recursion is used for further division into still smaller subproblems (*M-ary* divide and conquer) until no further division is possible. Very simple tasks are performed on these subproblems and the results are combined.

Workload allocation

The traditional issues in mapping data partitions to processor topology graphs are:

- *Partitioning.* Depending on the problem domain, partitioning specifies how data should be divided between available processes.
- *Granularity.* When a computational task is partitioned into several subtasks, the size of each subtask (number of sequential instructions) defines its *granularity.* The computation/communication ratio is a granularity metric.
- *Mapping.* Assignment of parallel subtasks onto processors based on communication graphs (e.g. topology and exchanged data amounts).
- *Load balancing.* Distributing subtasks evenly across processors in order to achieve the highest possible execution speed by equalizing the amount of work performed by each processor. Load balancing is useful when the work load is not known prior to execution and when processors are heterogeneous.
- *Scheduling.* This is the process of allocating subtasks (based on the load balancing algorithm) to processors as and when they become free.

[3]Another form of functional parallelism is *pipelining* where repeated function executions are overlapped.

There are several other models[4] of computation that can be used to model either message passing systems or shared memory multiprocessor systems but they are primarily theoretical and not of direct relevance to this thesis.

Parallel Execution Time (t_p)

The parallel execution time, t_p is the sum of two parts, the communication time, t_{comm} and the computation time, t_{comp}. For workstation clusters, t_{comm} depends on many factors, including network structure, network contention, etc. For time complexity analysis this thesis uses the following expression as a first approximation,

$$t_{comm} = \alpha + n\beta \qquad\qquad t_{comp} = m$$

where α is the start-up time (assumed constant), β is the transmission time to send one data word (assumed constant), n is the number of data words sent, and m is the number of computational steps. For the analysis it is assumed that, (i) t_p is normalized in units of an arithmetic operation (which depends on the computer system), (ii) the system is homogeneous, every node is identical and operating at same speed, and (iii) all arithmetic operations are considered to require the same time.

3.3 Benchmark Suite

Two programs, a distributed matrix multiplication (MM) program and a distributed flow simulation (FS) program were implemented. These applications were selected because they generate two significantly different types of communication and computation patterns. The distributed MM is a typical example of a coarse-grained, *computation-intensive* application. It requires comparatively few communications but the messages communicated are large. The distributed FS problem on the other hand, is a fine-grained or *communication-intensive* application requiring many communications between the processing nodes where the messages communicated are

[4]Examples of these include the PRAM [87], the BSP, [90], the LogP, [27], and the CSP, [86].

usually small. The implementation of these two applications are now described.

3.3.1 A coarse grained problem

The parallel multiplication of two matrices, **A** and **B** to yield the product matrix **C** = **A**×**B** (where $A \in \mathcal{R}^{m \times n}$, $B \in \mathcal{R}^{n \times l}$ and $C \in \mathcal{R}^{m \times l}$) is a coarse-grained problem. The matrix multiplication algorithm used in this thesis is based on the SUMMA (Scalable Universal Matrix Multiplication Algorithm) [70] but with a slight modification. A number of other other algorithms [71], [72] are also available for multiplying two matrices on distributed-memory machines. The SUMMA algorithm[5] was used as a benchmark problem because it is a general, simple, and very efficient algorithm for execution on loosely coupled (message-passing based) distributed-memory systems.

Traditional matrix-multiplication algorithms include 1D-systolic [72], 2D-systolic [72], Cannon's algorithm [73], Broadcast-Multiply-Roll [74] and Fox's algorithm [75]. Most of these algorithms are based on generalizations of the broadcast-multiply-roll algorithm. SUMMA on the other hand uses a sequence of rank-one updates and is based on identical mappings of matrix blocks to the nodes in a logical 2-D processor mesh of arbitrary dimension. Two algorithms that are close variants of SUMMA are: PUMMA [76] which implements Fox's classic algorithm using block-cyclic data decompositions, and DIMMA [77] which incorporates SUMMA with two new schemes, a modified pipelined broadcast scheme to effectively overlap computation with communication and the LCM (least common multiple) block concept to obtain the maximum performance of the sequential BLAS[6] routines independent of the block size.

The modification made to SUMMA in this thesis is reflected at the end/slave nodes. In the slave nodes instead of performing a regular matrix multiplication of the sub-matrices, a block multiplication with a level-4 loop unrolling is used if the

[5]SUMMA is implemented as a sequence of rank-one updates in this thesis as opposed to rank-k_b updates, where k_b is the block size of the columns of A or rows of B

[6]Lawson, C., *et al.* proposed (ACM Trans. Math Software, 5:308-323, Sept 1979.) a number of basic linear algebra subroutines which are generally accepted as the so-called BLAS.

sub-matrices are square and simple block multiplication is used if the sub-matrices are rectangular.

Data Decomposition

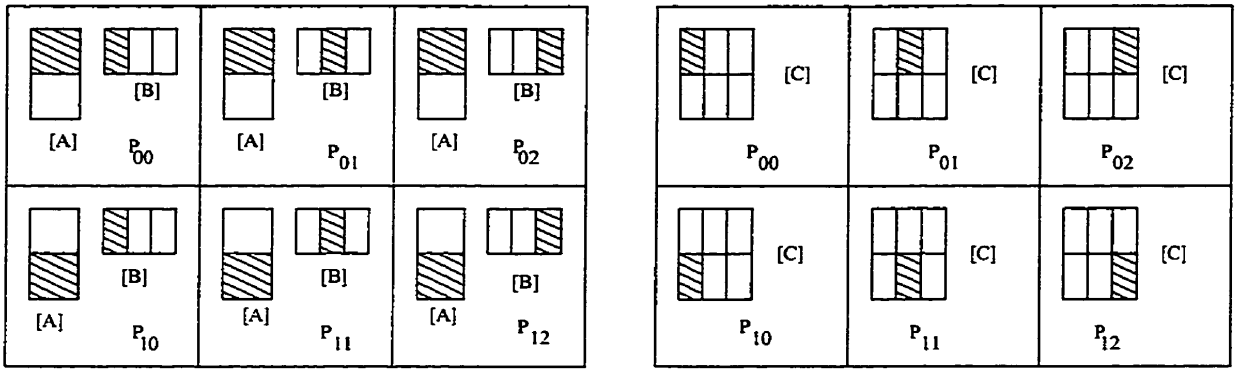
It is assumed that there are p nodes in the PVM virtual machine which form a logical $r \times c$ mesh, irrespective of the actual physical node configuration. The p nodes (where $p = rc$) are indexed by their row and column index and we use P_{ij} to denote the (i, j) node. Each matrix X is assumed to be of dimension $\alpha^X \times \beta^X$, where[7] $X \in \{A, B, C\}$. With an $r \times c$ logical mesh, the data is decomposed in two dimensions as follows:

$$X = \left(\begin{array}{c|c|c} X_{00} & \cdots & X_{0(c-1)} \\ \hline \vdots & & \vdots \\ \hline X_{(r-1)0} & \cdots & X_{(r-1)(c-1)} \end{array}\right)$$

and submatrix X_{ij} is assigned to node P_{ij}, where X_{ij} has dimensions $\alpha_i^X \times \beta_j^X$, with $m = \sum \alpha_i^X$ and $n = \sum \beta_j^X$. For the two matrices, **A** and **B**, $\alpha^A = m$, $\beta^A = n$ and $\alpha^B = n$, $\beta^B = l$. The two-dimensional data decomposition using a 2×3 logical processor mesh is done in two steps as shown in Figure 3.2 (where $r = 2$ and $c = 3$). In the first step, the matrix **A** is partitioned into r row-blocks and *broadcast* to those processing nodes in the same row. In the second step, the matrix **B** is partitioned into c column-blocks and *broadcast* to those nodes in the same column of the processor grid. After the distribution phase, each processing node computes a submatrix of **C** using the appropriate partitioned matrices of **A** and **B**. In the last phase the resultant submatrices of **C** are collected by the *master*, a designated processing node, from each of the other processing nodes.

The basic SUMMA algorithm and the modification to it used in this thesis are described in Appendix I.

[7]Here the notation α^X and β^X are used to denote respectively, the row and column dimensions of the matrix X.



(a) Submatrices of [A] and [B] after partition in slave nodes.

(b) Submatrices of [C] after computation in slave nodes

Figure 3.2: Data decomposition of matrices [A] and [B] on a 2×3 processor mesh.

Parallelization Technique

The modified SUMMA algorithm was implemented for a distributed cluster architecture using the *master-slave* model of computation. The rationale for using this model was the fact that domain decomposition of the SUMMA algorithm is very well-defined. Partitioning of the problem results in a set of subtasks each having identical computations (like SPMD) which can be executed simultaneously. In this master-slave model, the master task first spawns $p = r \times c$ slave tasks. It then partitions $[A]$ and multicasts the appropriate submatrices of $[A]$ to the r row nodes of the processor mesh. Similarly it partitions $[B]$ and multicasts the particular submatrices of $[B]$ to the c column nodes. Using this data (i.e. the submatrices of $[A]$ and $[B]$) the slave tasks compute the submatrices of $[C]$ using the block multiplication algorithm. These submatrices are then sent to the master task which assembles them to form the result product matrix, $[C]$ as shown in Figure 3.3.

Time Complexity

- *Computation.* Denoting $m_b = \lceil m/r \rceil$ and $l_b = \lceil l/c \rceil$, the computation time can be given by $t_{comp} = 2m_b n l_b$. For $m = n = l$ and $r = c = \sqrt{p}$, the time complexity can be expressed as $\Theta(\frac{n^3}{p})$.

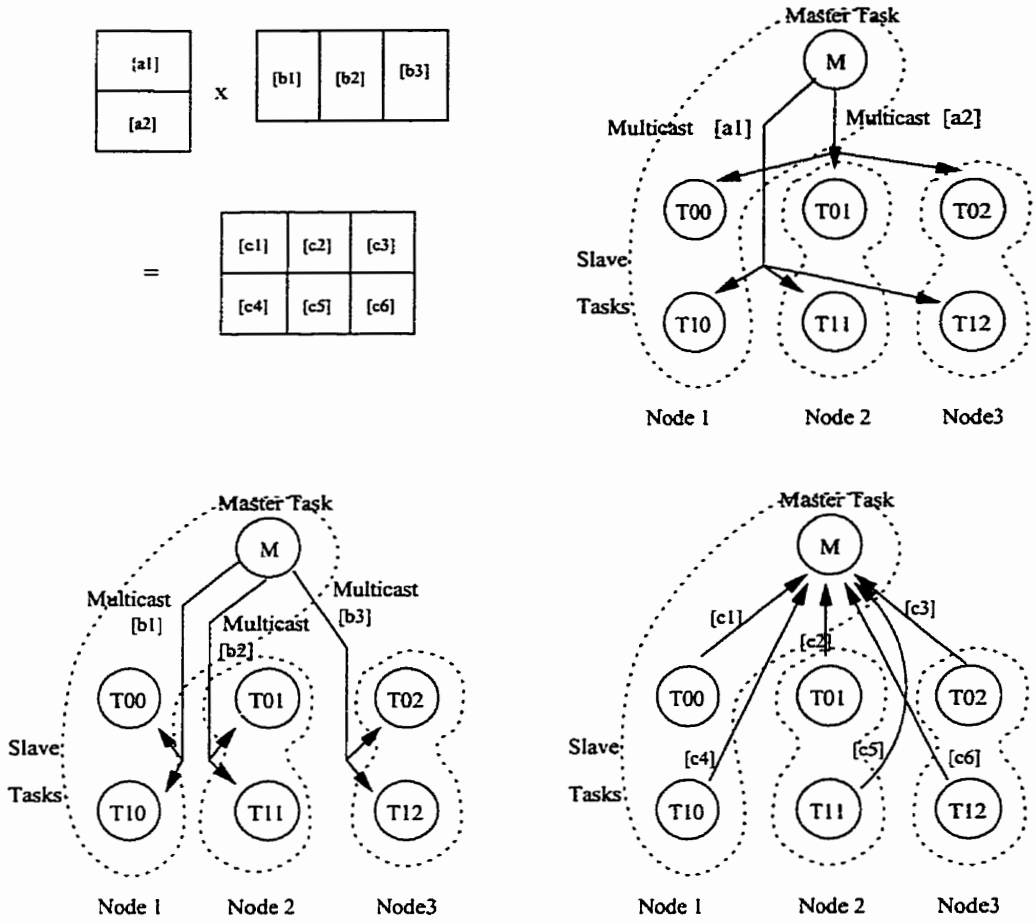


Figure 3.3: Computation graph for matrix multiplication on a 2×3 processor mesh.

- *Communication.* We observe that communication occurs in 3 stages,

  stage 1: $m_b \times n$ blocks of $[A]$ send to p nodes, $t_{comm1} = p(\alpha + m_b n \beta)$

  stage 2: $n \times l_b$ blocks of $[B]$ send to p nodes, $t_{comm2} = p(\alpha + n l_b \beta)$

  stage 3: $m_b \times l_b$ blocks of $[C]$ recv from p nodes, $t_{comm3} = p(\alpha + m_b l_b \beta)$

  With $m = n = l$ and $r = c = \sqrt{p}$ the total communication time becomes, $t_{comm} = 3p\alpha + 2n^2\beta\sqrt{p} + n^2\beta$, and for constant β this has a time complexity[8] of $\Theta(n^2\sqrt{p})$. If *scatter* and *reduce* routines[9] are used, then the $3p\alpha$ term can be replaced with 3α.

[8] Assuming that the start-up time, α is not significant compared to n.

[9] MPI has these but PVM version 3 only has reduce.

- *Overall.* The overall time complexity can be expressed as $\Theta(n^2\sqrt{p} + \frac{n^3}{p})$.

3.3.2 A fine grained problem

A communication intensive flow simulation problem was selected as the fine-grained problem for study. A two-dimensional, incompressible potential flow past a system of lifting NACA0012 aerofoils is analyzed using the surface vorticity boundary integral method [78]. This is an important problem in the field of aeronautics and engine aerodynamics. Such sub-sonic airfoil aerodynamic problems pose critical computational difficulties like aerodynamic interference resulting from the proximity of vorticity elements on opposite sides of the airfoil-profiles.

The *surface vorticity panel method* which is briefly described in Appendix IV was used. This is a popular method due to its flexibility and relative economy and has been used in the aircraft industry for a long time. Existing programs based on panel methods and widely used in the industry include PMARC [79], developed by the NASA Ames Research Center, PAN AIR [80] developed by Boeing, VSAERO [81] developed by Analytical Mechanics Inc., and QUADPAN [82] developed by Lockheed. However, the algorithm used in this thesis is based on that given by Lewis [83].

The flow solution over a *single* airfoil section immersed in a uniform flow-field involves the following steps. First, the airfoil surface is discretized into a finite number of line elements or *panels* that represents the numerical airfoil model (see Figure 3.4). Then vortex singularities of unknown strength are distributed over the surface vorticity panels and not at the pivot points. Integral equations are written and the Dirichlet boundary conditions of zero parallel surface velocity are applied. The resulting equation is of the form,

$$[K_{mn}]\{\gamma(s)\} = \{W(s)\} \tag{3.3.1}$$

where K_{mn} is the coupling co-efficient matrix (linking elements m and n), $\gamma(s)$ is

the unknown vortex distribution, and $W(s)$ corresponds to the free-stream velocity components parallel to the element surface, s.

The linear system of algebraic equations (as represented by equation 3.3.1) are solved to determine the unknown strengths of the singularities (vorticities $\gamma(s)$). The surface vorticity distribution ($\gamma(s)$) is then used to determine the surface pressure co-efficient, C_p, and the lift co-efficient, C_l, which are the important parameters required for aerodynamic analysis. The detailed computational scheme is described in Appendix IV.

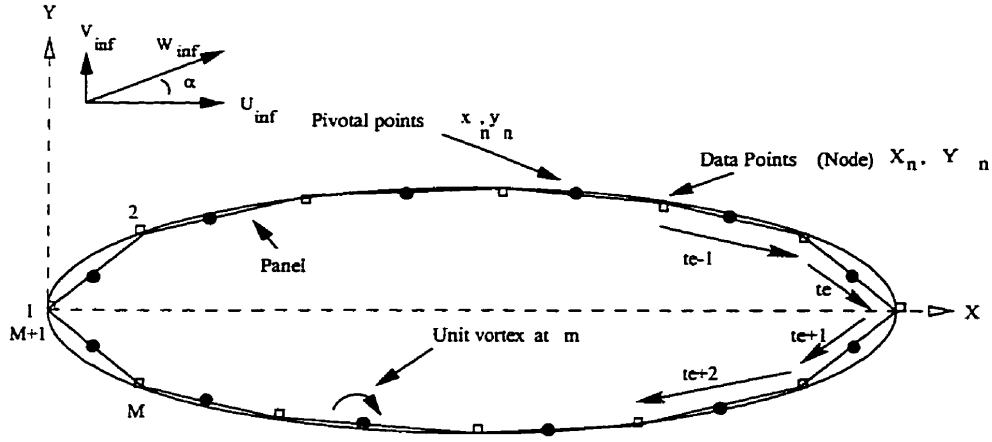


Figure 3.4: Airfoil representation by discrete line elements or panels.

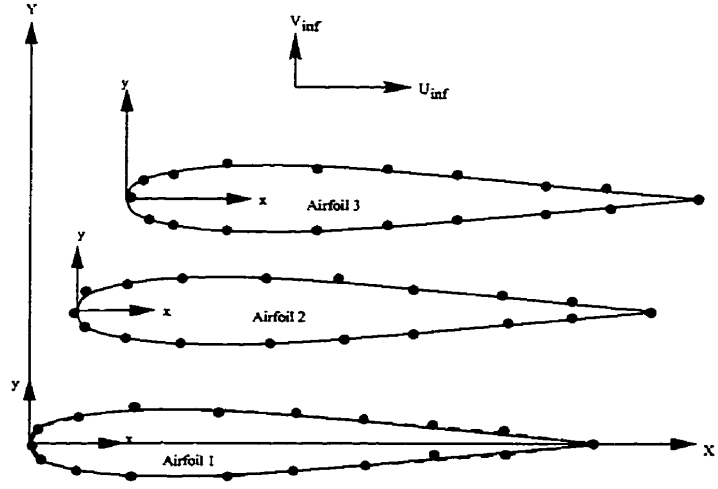


Figure 3.5: The panel discretization of 3 airfoils in proximity.

For the *multi-body* airfoil-assembly system, the computational scheme is similar and is shown in the flowchart in Figure 3.6. The integral equations used to represent the flow past an assembly of P mutually interacting airfoils is represented by a simple adaption of the isolated single element potential flow equation (equation 3.3.1). Similar Dirichlet boundary conditions are applied to the surface elements of the P airfoils and finally the vortex distributions over each of the P airfoils are found by solving the global linear equation set. The co-efficient of pressure distributions and the lift co-efficient are then calculated from the vortex distribution of each airfoil. The computational details of the multi-foil case are also described in Appendix IV.

The fine grained problem used in the experiments is the multi-body airfoil analysis just described. The specific problem used comprises an assembly of a maximum of 20 airfoils (a 3 airfoil system is shown in Figure 3.5) so positioned (1 chord length apart) that their mutual interference effect is significant. The detailed results and validation of the sequential single-element and the multi-element code can be found in Appendix IV.

Parallelization Technique

The algorithm for the flow simulation problem was parallelized using the *node-only* programming paradigm. The same code runs on all the participating nodes (forming a PVM group) with one node doing the additional non-computational job of assembling results, etc. Domain decomposition is achieved by partitioning the computation into a set of identical tasks that concurrently compute on each airfoil element. The coding of the program was done in C++ and one of the C++ classes used was an Airfoil class (*Tairfoil*) that was designed to compute the elemental[10] $\{w_j^{(e)}\}$, and elemental $k_j^{(e)}$ sub-matrices for both the diagonal and off-diagonal terms for the j^{th} airfoil element (refer to Figure 3.6). The remainder of the problem lies in solving the large linear systems of equations that are present as partitioned blocks in each of the participating

[10]The subscript e denotes elemental, i.e. concerning the particular airfoil.

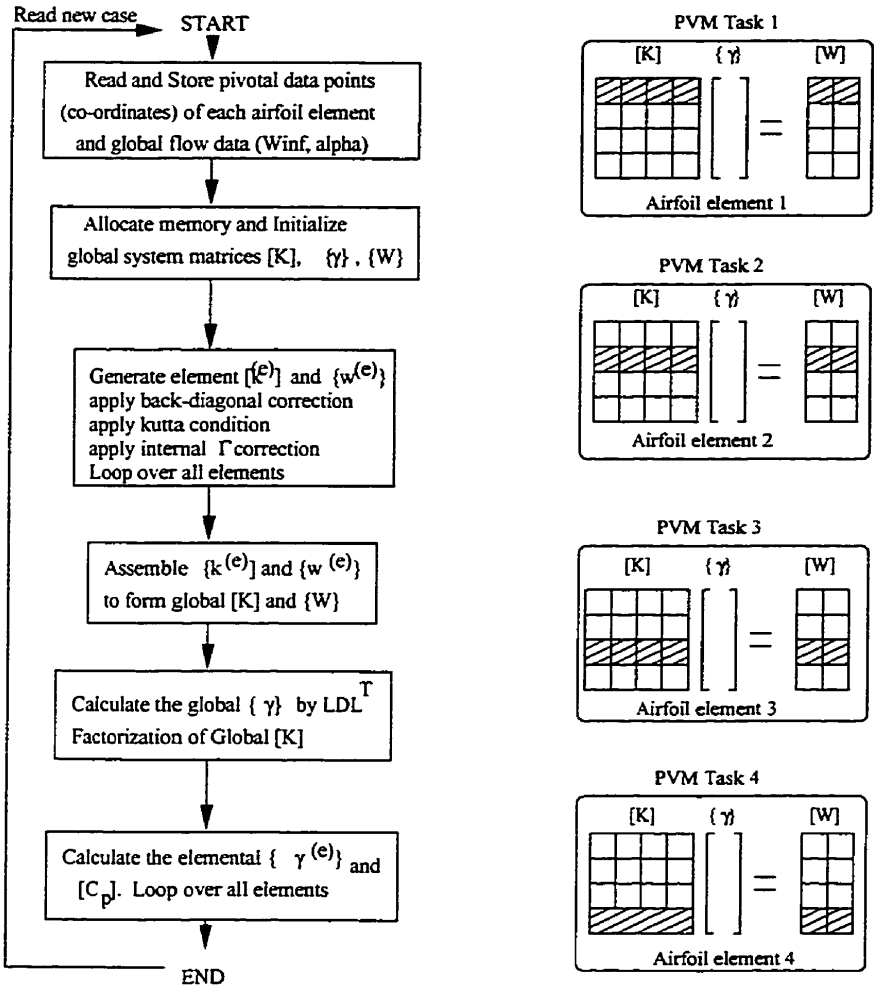


Figure 3.6: Sequential Flow chart and domain decomposition for the parallel Multi element flow code.

tasks. The parallel *Jacobi iteration* technique[11] with *block allocation* scheme [89] was used to solve this block-partitioned equation system.

The Jacobi iteration technique can be parallelized by using synchronous iterations with global *barriers*[12]. Unlike the sequential algorithm the large global $[K]$ matrix is not assembled in the master or any other single node. Instead each task has its own block of the $[K]$ matrix and $\{W\}$ (already strip partitioned as shown in Figure

[11]The sequential Jacobi iteration technique is explained in Appendix I.

[12]A barrier is a mechanism that prevents any task from continuing past a specific point until all the tasks reach that point.

3.6). On each iteration, k, the newly computed unknowns, $\{W_i^{(k)}\}$ in the task i are broadcast to all other tasks and subsequently all the tasks (except j) broadcast their new $\{W_j^{(k)}\}$ values to task i. At the end of each iteration a barrier is executed to ensure that other tasks have completed their iterations. These steps are continued until convergence[13] is reached. Figure 3.7 shows the broadcast and receive operations performed during the Jacobi iteration stage.

Once the solution has converged, the $\{W\}$ values in each task are used to calculate the C_p distribution for each airfoil. These are then sent to the task which acts as the *master* to report the final results.

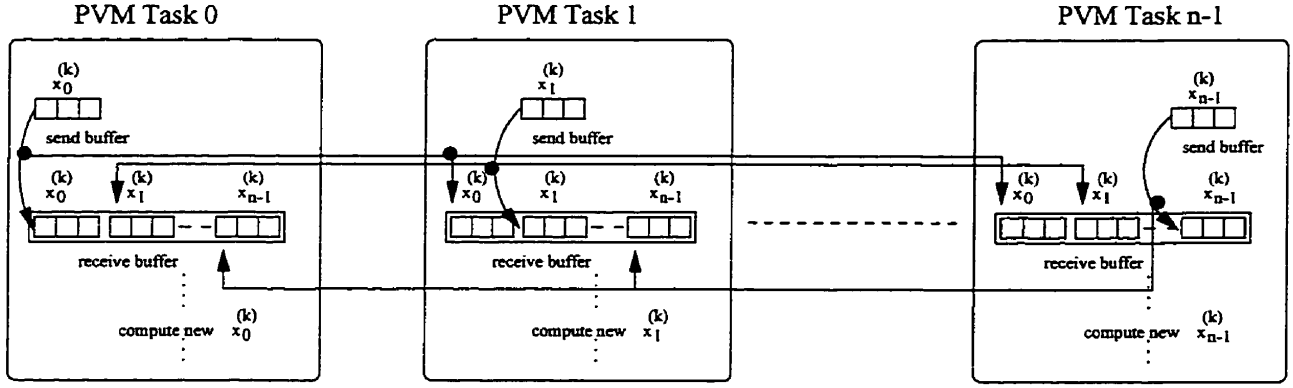


Figure 3.7: Schematic data communication for Jacobi iteration.

Time Complexity

Denoting p as the number of nodes, n as the size of $[K]$, and τ as the number of iterations[14] until convergence for the Jacobi iteration technique, the time complexity of the flow problem is derived as follows.

- *Computation.* Formation of the system matrices involves $\lambda n/p$ steps ($1 < \lambda <<$

[13]Refer to Appendix I for convergence criterion.

[14]Here τ is taken as equal to n to avoid the time involved in computing the complex termination condition.

n/p) and the Jacobi iteration involves $\frac{n}{p}(2n+4)\tau$ steps resulting in a computational complexity of $\Theta(\frac{n^2}{p})$.

- *Communication.* Communication again involves 3 stages,

stage 1: $2\frac{n}{p}$ data send to p nodes,	$t_{comm1} = p(\alpha + 2n\beta)$
stage 2: $\frac{n}{p}$ data send to p nodes for τ iterations,	$t_{comm2} = p(\alpha + \frac{n}{p}\beta)\tau$
stage 3: $2\frac{n}{p}$ data recv from p nodes,	$t_{comm3} = p(\alpha + 2n\beta)$

  This results in $t_{comm} = \tau n\beta + p(2\alpha + \alpha\tau + 4n\beta)$. Thus for constant τ and a given fixed value of n, t_{comm} is an increasing function of p.

- *Overall.* When α becomes non-negligible, the overall parallel execution time is a composite function of a decreasing function of p (t_{comp}) and an increasing function of p (t_{comm}). It thus has a minimum value.

3.4 Summary

This chapter discussed the research problem and the reasons for undertaking this study. It reviewed some of the common parallel programming paradigms used in the context of this thesis. Finally a detailed description of the two parallel algorithms that were implemented were discussed.

"I never did anything by accident, nor did any of my inventions come by accident; they came by work".

- Thomas A. Edison.

Chapter 4

Cluster Performance Evaluation

This Chapter first describes the testbeds for the experiments. Section 4.2, then evaluates the communication performance of the four clusters and present a prediction model. This is followed by the evaluation of overall computational performance in Section 4.3. Finally in Section 4.4 the impact on cluster performance due to heterogeneity is evaluated.

4.1 Experimental Setup

Four different test-beds were used for the experiments. The benchmark programs were implemented on clusters connected either by Ethernet (10-BASE-T), ATM (OC-3) or Fast Ethernet (100-BASE-T) available at the University of Manitoba[1]. For latency and bandwidth tests in each of the network environments, two SUN SPARC 5 workstations running Sun-OS 5.5.1 were used except in the case of switched Fast Ethernet where all the nodes were Pentium II, 333's running Linux 2.0.35. PVM version 3.3.11 was used on Ethernet (both switched and unswitched) and on switched

[1]The ATM test-bed of Telecommunications Research Labs (TR*Labs*), Winnipeg, was used to test some of the distributed algorithms.

Fast Ethernet. The network protocol hierarchy in this case is shown in Figure 4.1. When ATM was used as the cluster inter-connect, the *pvm-atm*[2] package developed by the Distributed Multimedia Research Centre (DMRC) at the University of Minnesota, was used. In this PVM package, the PVM system was re-implemented directly on top of the Fore Systems ATM API instead of the BSD socket interface. The protocol hierarchy of this implementation is shown Figure 4.2.

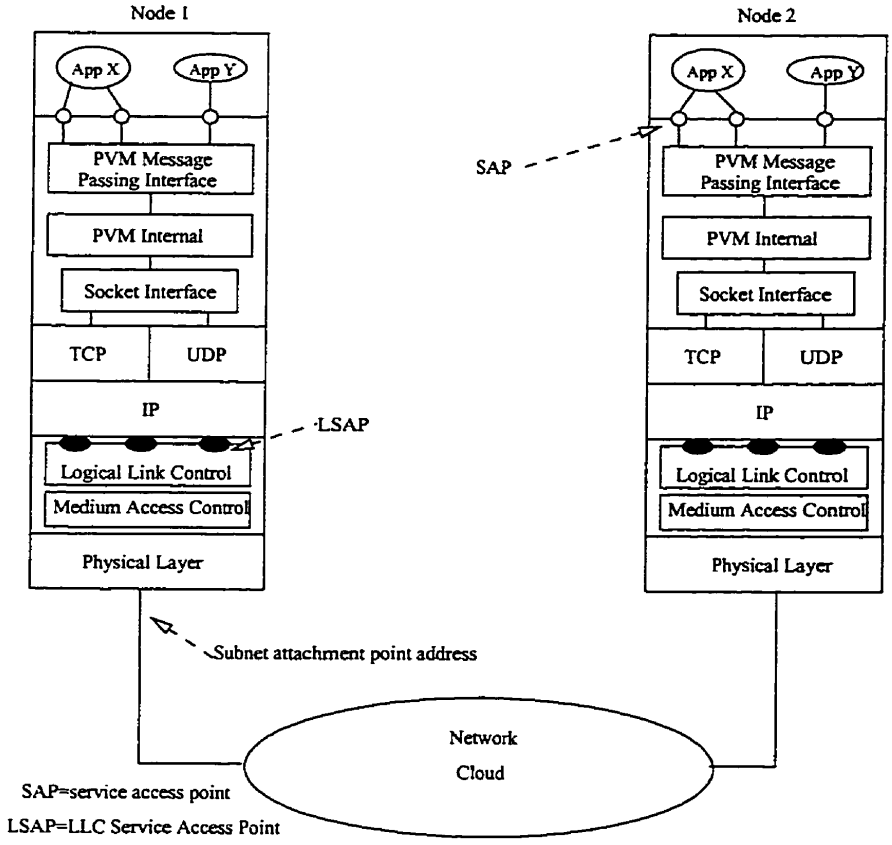


Figure 4.1: Network Protocol hierarchy for Ethernet-based LANs

4.1.1 Ethernet Test-bed

Two different Ethernet set-ups were used, one switched and the other unswitched.

[2]PVM over ATM is available via ftp://ftp.cs.umn.edu/users/du/pvm-atm/www.html

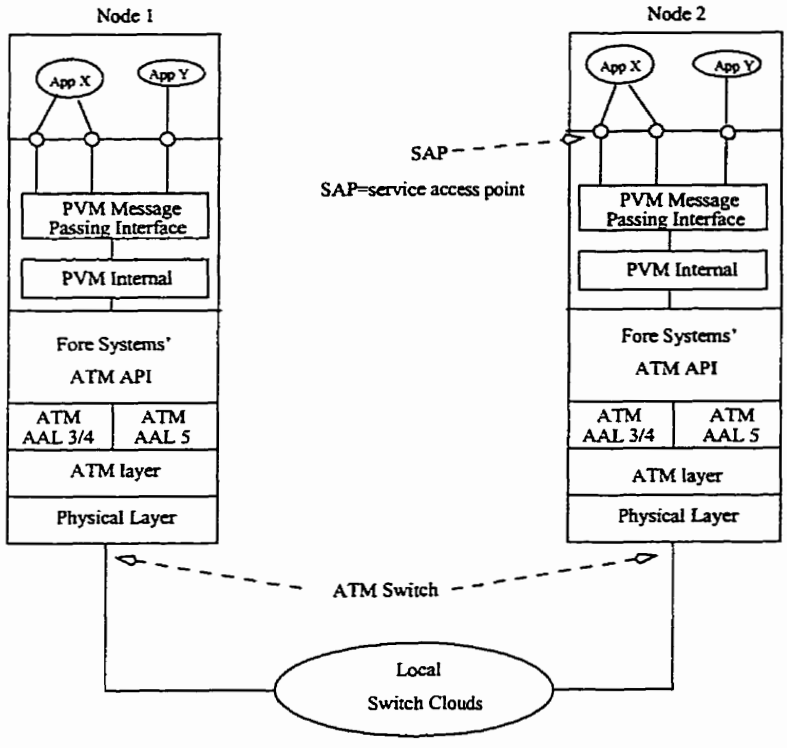


Figure 4.2: Network Protocol hierarchy for ATM LANs

Unswitched Ethernet

The experiments on the unswitched Ethernet LAN were conducted using the network of the University of Manitoba Computing Centre's UNIX Lab in the Engineering building. The network topology of this unswitched Ethernet LAN is shown in Figure 4.3. It is comprised of a typical bus architecture with different segments of the LAN forming subnets that are connected by routers to the main login servers.

Switched Ethernet

The 10-BASE-T switched Ethernet LAN used was that of the Electrical & Computer Engineering department. The network configuration of this LAN is shown in Figure 4.4. Latency analysis and bandwidth measurements on this network were conducted using two SUN Sparc 5 workstations having nearly the same configuration. The benchmark problems were tested using a 4 node configuration (see Figure 4.4)

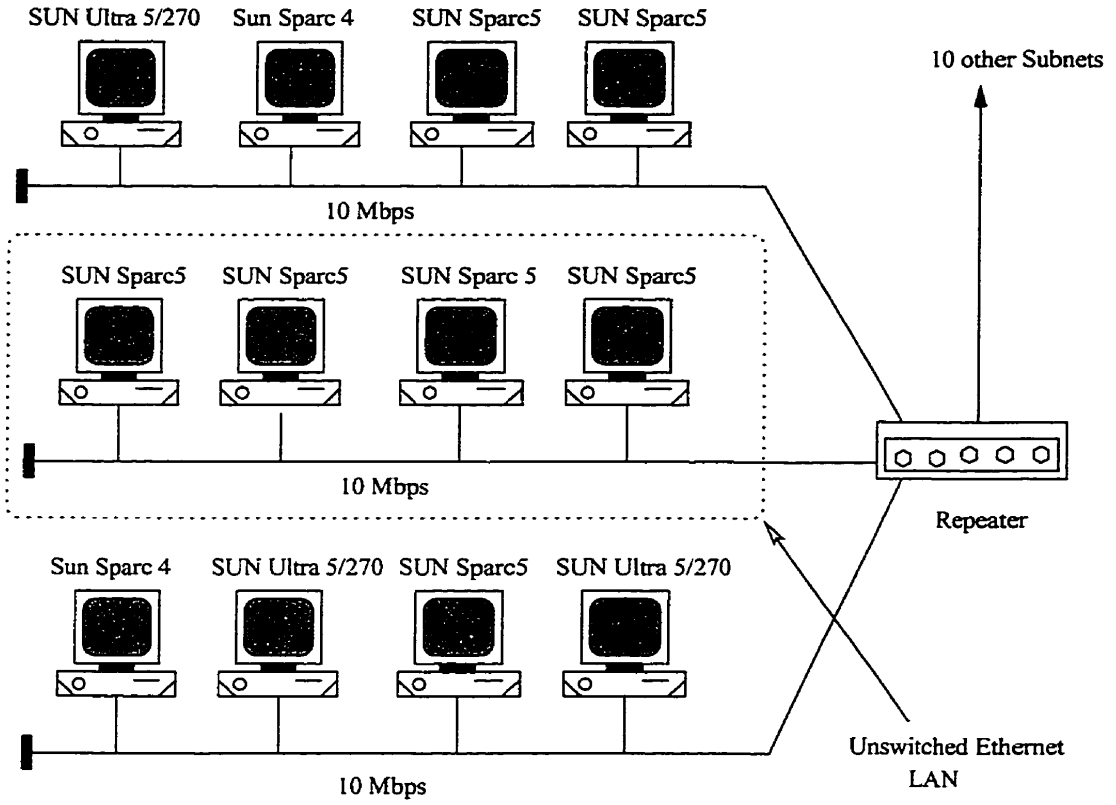


Figure 4.3: The Unswitched Ethernet LAN set-up.

which included a SUN Ultra 1/30 (Model 170), a Sparc 10 and two Sparc 5 workstations, all running Solaris 2.6 and connected to a 3Com Ethernet LAN switch. Two nodes were on one hub and the other two on a second hub, both the hubs in turn were connected to the 3Com switch (this formed a partially switched configuration).

4.1.2 ATM Test-bed

The ATM network testbed used for the experiments consisted of the same 4 nodes that were used in the Switched Ethernet network configuration (existing in the Electrical & Computer Engineering department). Each of the 4 workstations were equipped with an SBA-200 (Fore's second generation) SBus adaptor card and were directly connected to the Fore Systems *ForeRunner*^{TM} ASX-200 local ATM switch using 155 Mbps fibre optic links (OC-3c/STM-1 SONET/SDH). The ATM LAN test set-up is

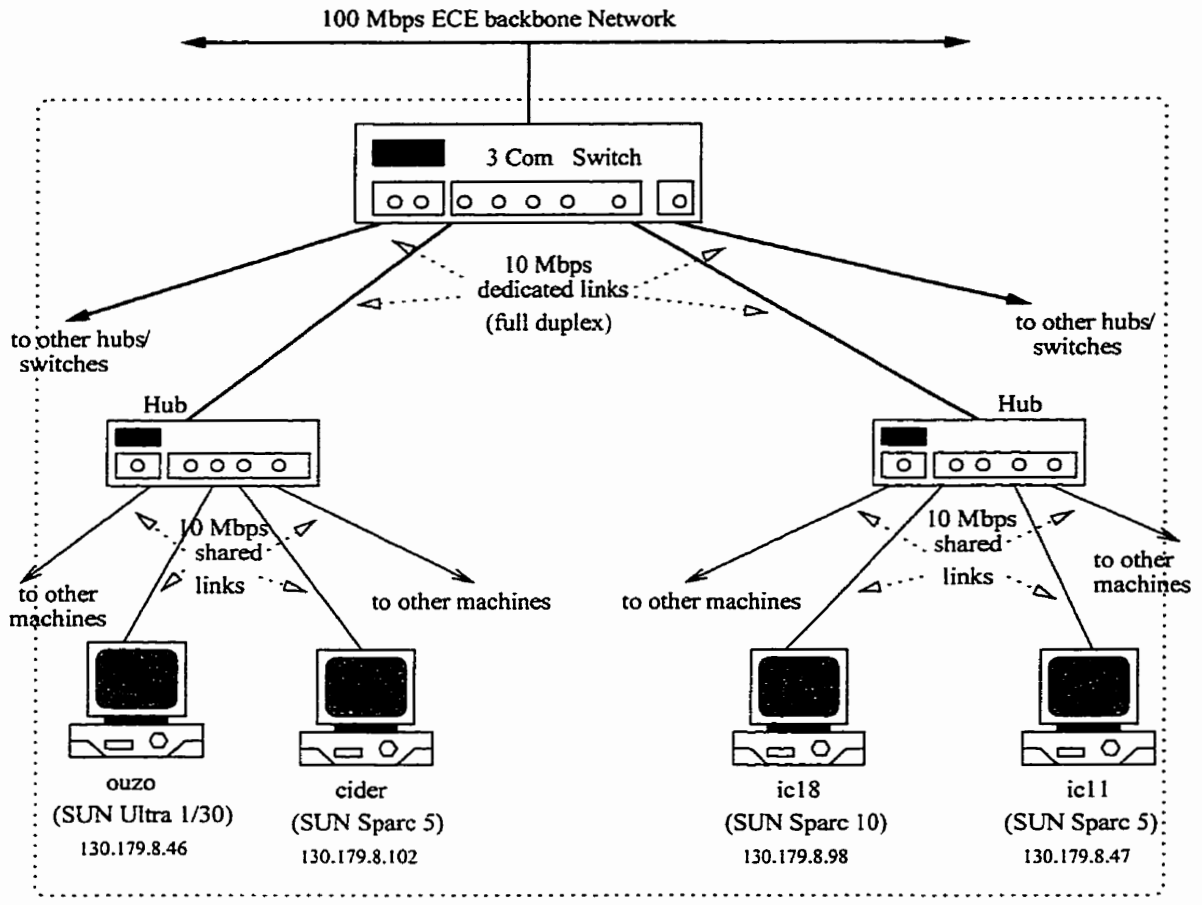


Figure 4.4: The Switched Ethernet LAN set-up.

shown in Figure 4.5.

The SBA-200 adapter card contains an onboard 25 mHz Intel i960 RISC processor that does the SAR (segmentation and reassembly) functions of AAL 3/4 and AAL 5 and cell multiplexing. The adapter interface feeds the incoming and outgoing packets into and out of the buffer of the i960 so that the adapter memory (256 kbytes on-board RAM) is never used for cell storage. The i960 uses DMA (Direct Memory Access) to move cells out of and into the host's local memory.

The ASX-200 switch has a non-blocking switching capacity of 2.5 Gbps and provides up to 16 full duplex ports of connectivity, each running at 155 Mbps or up to 4 ports running at 622 Mbps (OC-12/STM-4c). It consists of a SPARC RISC switch control processor (SCP) which provides ATM switching functions including

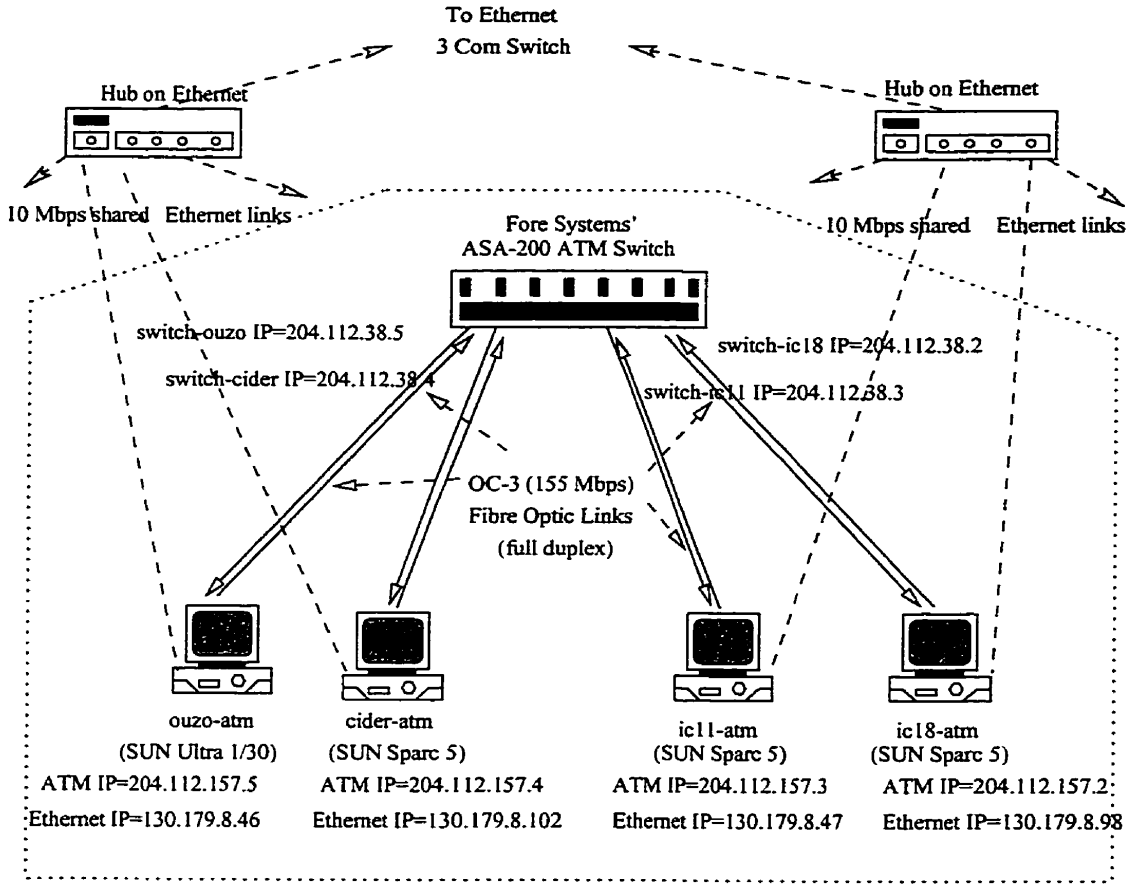


Figure 4.5: The ATM LAN set-up.

distributed connection set-up management and traffic control. The switch supports Fore's SPANS (Simple Protocol for ATM Network Signaling) protocol to establish either Switched Virtual Circuits (SVCs) or Permanent Virtual Circuits (PVCs). All the experiments were based on PVC ATM connections and the circuit set-up times were ignored .

4.1.3 Switched Fast Ethernet Test-bed

The 100-BASE-T switched Fast Ethernet network set-up of the Computer Science department was used. In this network configuration (see Figure 4.6), all 10 hosts were Pentium II/333's running Linux 2.0.35 and connected to a cascade of 2 Fast Ethernet (D-Link DES-1008) switches (5 hosts on each switch) in a *star* topology using CAT

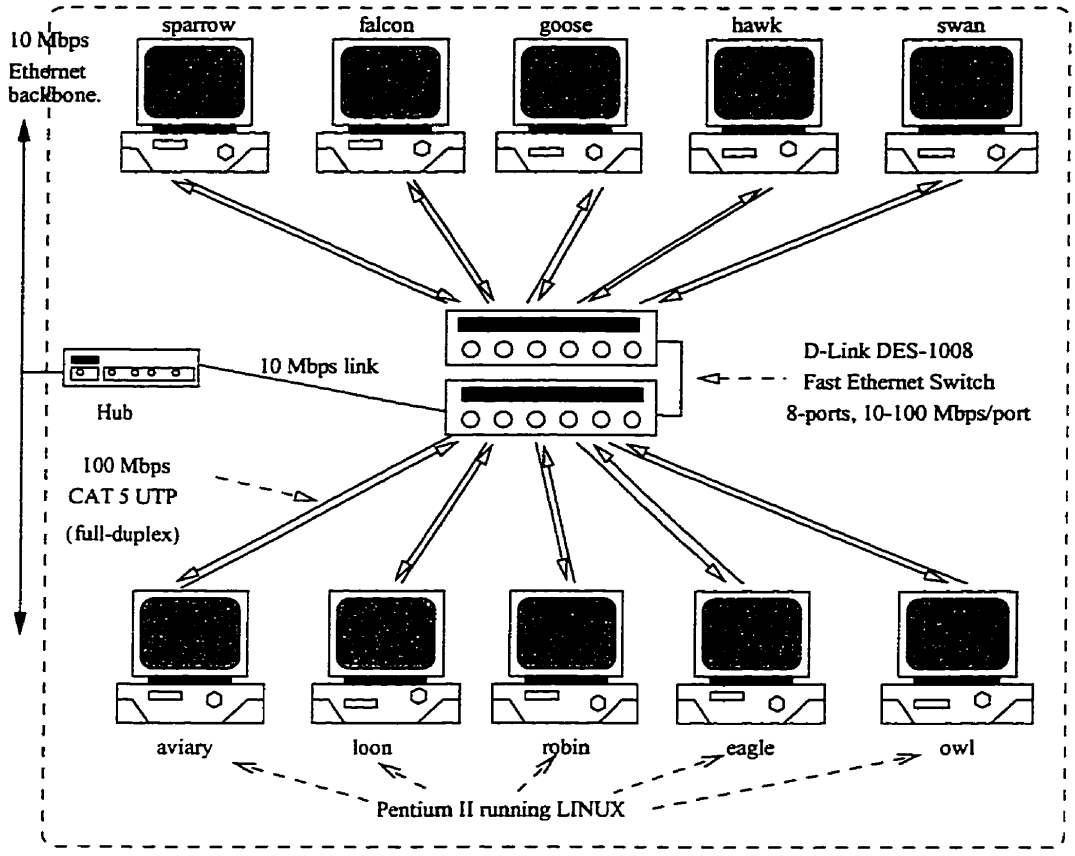


Figure 4.6: The Fast Ethernet LAN set-up.

5 UTP cables. The DES-1008 switch is based on IEEE 802.3u 100-base-TX Fast Ethernet Standard and each switch had 8 ports running at 100 Mbps supporting a maximum data transfer rate of 200 Mbps in full duplex mode.

The DES-1008 switch uses the same CSMA/CD protocol as standard 10-BASE-T Ethernet but achieves the high-bandwidths by combining dynamic buffer allocation (at each port which has 8 Mb RAM) with a store-and-forward switching scheme to support rate adaption and ensure data integrity. It uses n-way auto-negotiation for any port which allows for auto-sensing of speed (10/100 Mbps) and auto-detection of mode (full or half duplex) thus providing automatic and flexible network connections. It has a packet filtering or forwarding rate of 148,800 pps per port at 100 Mbps wire speed.

4.2 Communication Performance

The end-to end communication performance can be characterized by a set of well known network performance metrics. In the following sections these metrics are defined and evaluated for the different interconnection networks.

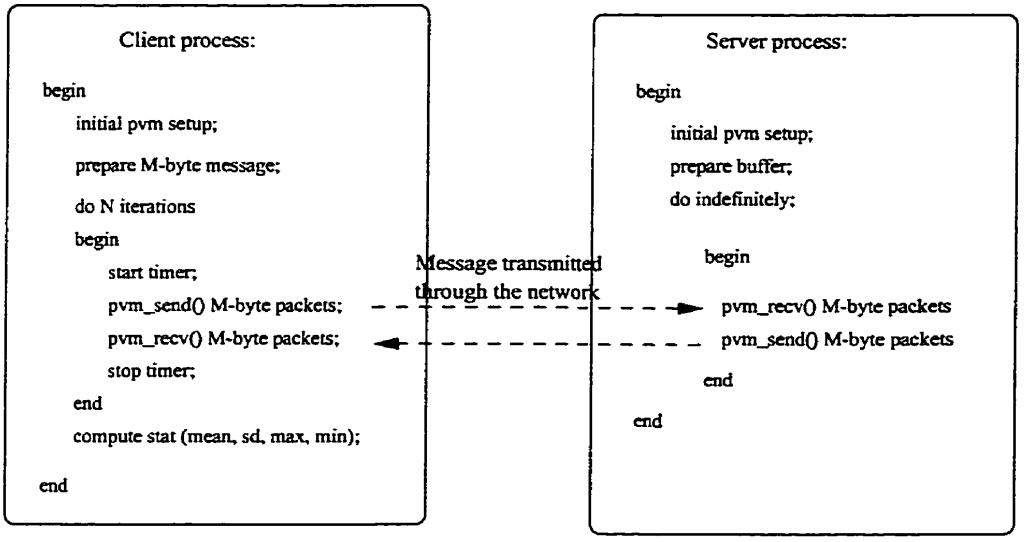


Figure 4.7: Pseudo-code for the *echo* program

To measure these network parameters a *client/server echo* program was used. The pseudo-code for this echo program is shown in Figure 4.7. The program simply repeats a Unix *"ping"* type operation (sending and receiving an M-byte message) between any two nodes in the cluster a number of times (N) and measures the average round-trip time (RTT) over all the iterations at the sender node. The communication time between sending and receiving the M-byte message is measured only at the sender node in order to avoid the problem of *clock synchronization* [47] between two different nodes. The experimental latency measurements also include an additional time which is spent in the network interfaces in addition to the actual transit time through the network, since this is indistinguishable to the node. All the measurements for latency and bandwidth on the different networks were done at night when the network was *virtually quiescent*. The two communicating nodes in any network were identical

workstations (SUN Sparc 5/170's or Pentium II/333's for the switched Fast Ethernet).

Latency Analysis

Communication latency is defined to be the complete communication delay while transferring a message. A variation of RTTs with the message size (M) for the different networks using TCP and UDP are shown in Figures 4.8, 4.9, 4.10 and 4.11. Here the message size (M) is varied as $M = 4 + n \times (2 \times payload\_size)$ bytes where $n = 0, 1, 2, 3, \ldots$ up to M = 8 Kbytes. The *payload* size for ATM cells is fixed at 48 bytes. Since Ethernet and Fast Ethernet use variable length frames, the *payload* size used was the minimum, 64 bytes, frame size. For each message length (M), 100 samples were taken and the *mean*, *minimum* and *mean + standard deviation* of the RTTs were plotted against the message lengths.

Discussion of Results: It can observed from the latency analysis (Figures 4.8, 4.9, 4.10 and 4.11) that the end-to-end communication latency is significantly less in all four networks when PVM uses TCP/IP instead of UDP/IP. This is due to the fact that for the two communicating tasks to communicate using the direct TCP route (*PvmRouteDirect*) requires establishing only a single TCP socket connection versus two Unix Domain socket connections and two UDP connections when the default UDP route (*PvmAllowDirect*) is used. TCP performs reliability checks not implemented by UDP, which is an unreliable connectionless protocol. Thus the observed performance increase of more than two-fold is in conformance with existing results [20]. When PVM uses the TCP based, direct link, the RTT is a linear function of the message size but when UDP is used there is a timing abnormality (a vertical shift in RTT values)that is observed for all networks at a message size of 4064 bytes.

4.2.1 Start-up Latency t_0

The *start-up latency* t_0 is defined to be the minimum time required to send a zero (or very short) length message. It is the delay incurred in setting up the communication

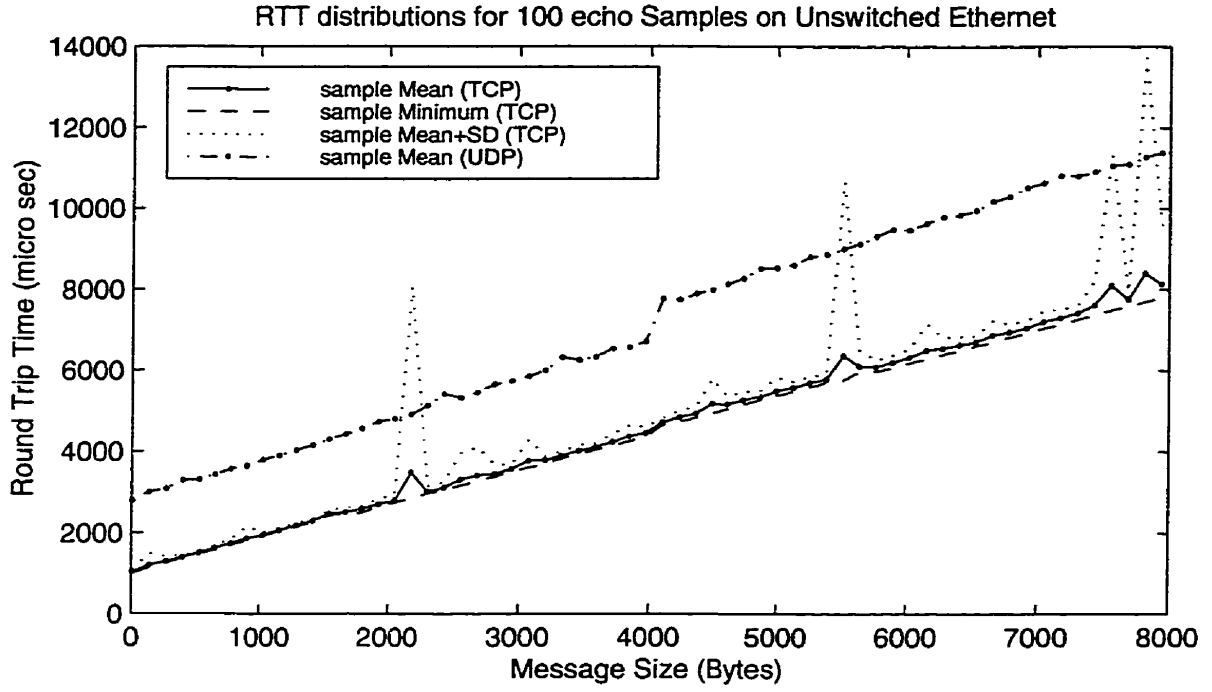


Figure 4.8: Communication Latency for PVM clusters over Unswitched Ethernet

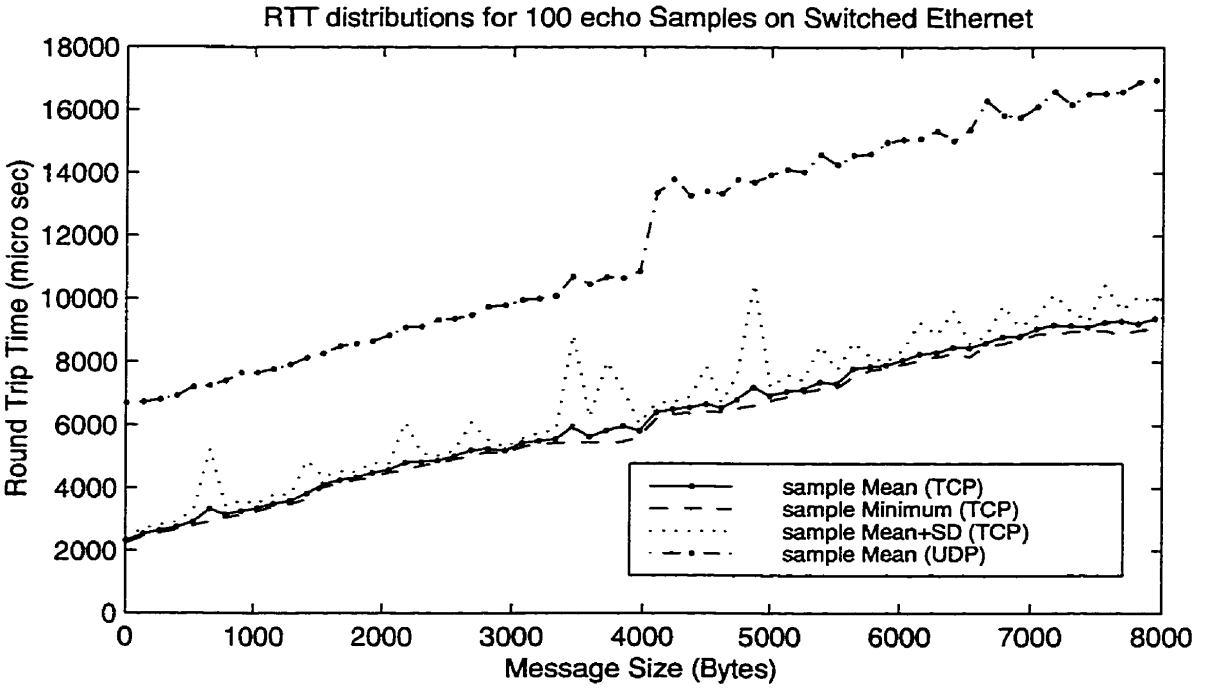


Figure 4.9: Communication Latency for PVM clusters over Switched Ethernet

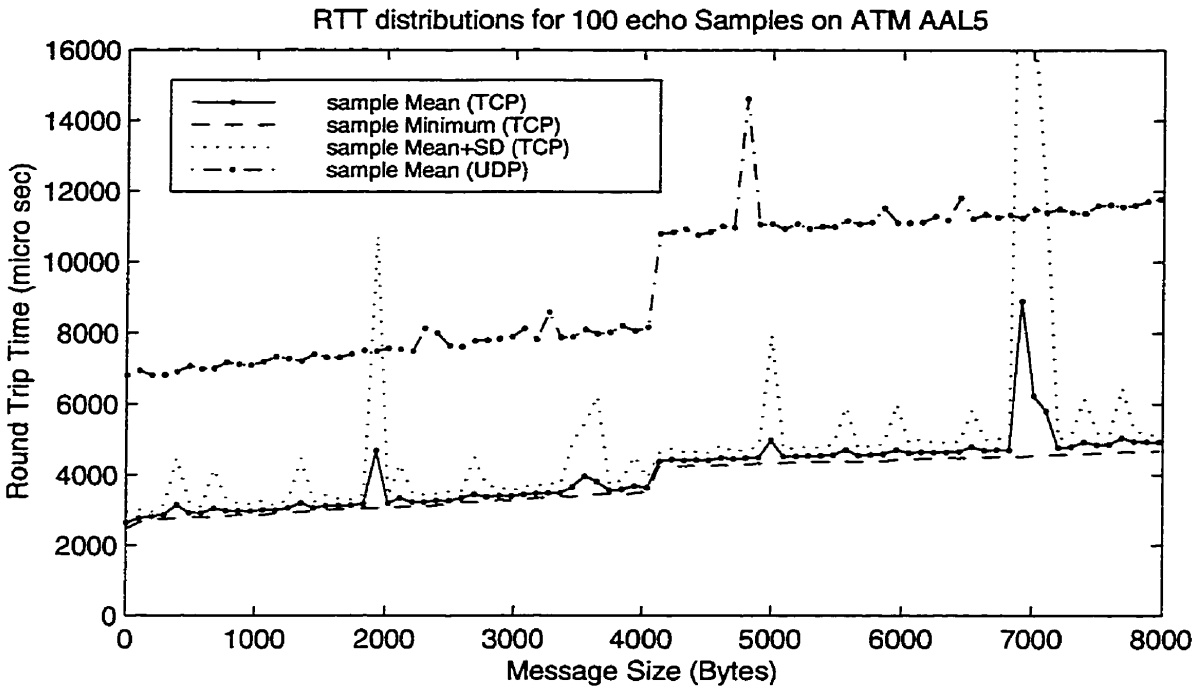


Figure 4.10: Communication Latency for PVM clusters over ATM AAL5

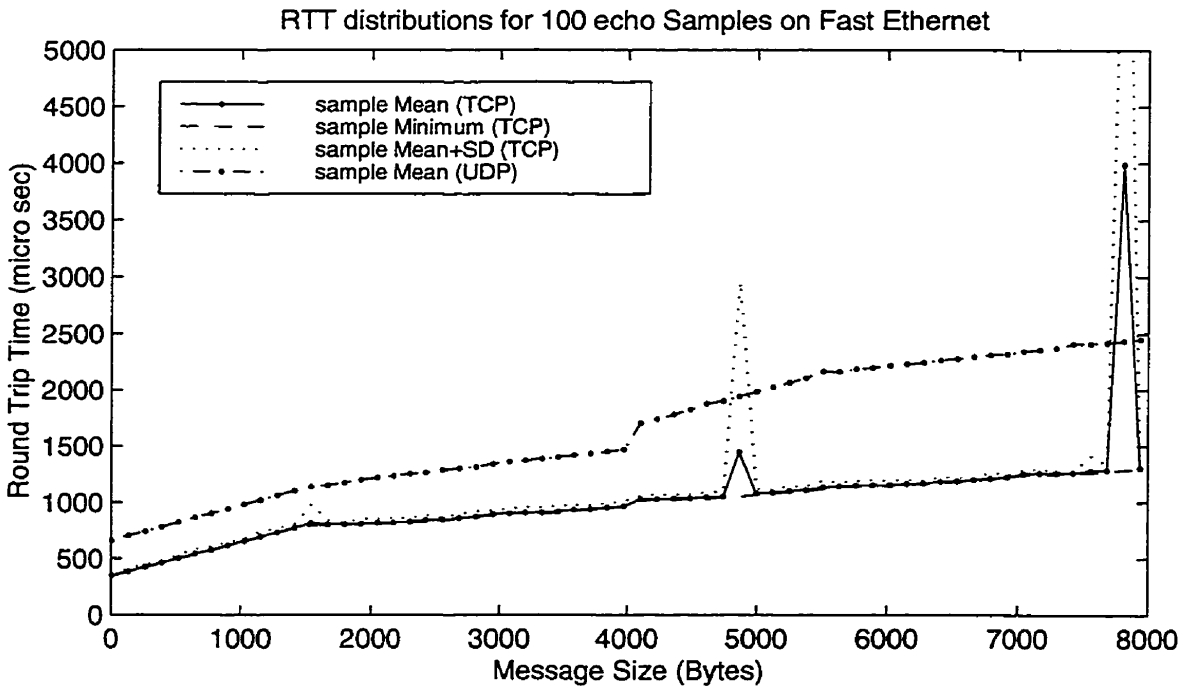


Figure 4.11: Communication Latency for PVM clusters over switched Fast Ethernet

Table 4.1: Communication performance metrics using PVM over different networks. The figures in bracket are the Standard Deviations (SD).

LAN Environment PVM over	*Protocol*	t_0 μ sec (SD)	r_{max} Mbits/sec	$n_{1/2}$ Bytes
Ethernet (unswitched)	TCP	802 (59.67)	8.99	1855
Ethernet (switched)	TCP	1506 (89.34)	9.24	2016
ATM (ASA-200)	AAL 5	2601 (98.31)	30.43	12879
Fast Ethernet (switched)	TCP	174 (1.88)	85.95	6628

between the source and target node and is independent of the message size. In the experiments this was calculated using the echo program by finding half of the RTT required to send a 4 byte message. Start-up latency is a metric for characterizing the communication capability of a node for very short messages. Table 4.1 gives the typical average values along with the *standard deviations* obtained from the experiments on two nodes averaged over a collection of 100 timing samples.

Discussion of Results: The start-up latency is observed (Table4.1) to be the lowest for switched Fast Ethernet and maximum for ATM AAL5. This large difference (3.24 times) in t_0 values of ATM and switched Fast Ethernet (as well as Ethernet) is attributed to the overhead of the ATM interface device drivers. It is believed that the better communication latency in Ethernet (100BASE-T and 10BASE-T) is due to its firmware code being fine-tuned [20].

4.2.2 Maximum Achievable Throughput r_{max}

The maximum achievable throughput (r_{max}) or *bandwidth* is the speed at which messages can be transmitted between two remote user processes. It is a measure of the

communication capability of a node interconnect and is important for applications which require large volumes of data transmissions. r_{max} is calculated as the ratio of message size (2× M) by the transfer time (RTT), as in this case the latency becomes negligible (for large M). The experimental values of M were increased from 256 bytes to a value (1 MB) for which r_{max} becomes almost constant. Again, for each message size M, a set of 100 samples were taken and the *mean*, *minimum* and *maximum* bandwidth of these samples are plotted as a function of the message sizes in Figures 4.12, 4.13, 4.14 and 4.15. r_{max} is calculated as the value for which the sample *maximum* becomes a constant and the typical values for different networks are shown in Table 4.1.

Discussion of Results: The maximum throughput was achieved with switched Fast Ethernet as compared to ATM which yielded only r_{max} = 30.43 Mbits/sec although its raw available bandwidth is 155 Mbits/sec. Switched Ethernet (92.4%), Unswitched Ethernet (89.9%), switched Fast Ethernet (85.9%) and ATM AAL5 (19.6%) showed decreasing values of network bandwidth utilization, respectively. The poor performance of ATM for maximum achievable throughput is due to the overhead that occurs at two levels: the end system and the ATM switch interface (both software and hardware) and the overhead of the PVM-ATM system as well. Hardware overheads included the host interface board, signal propagation delay, bus architecture of the host and the network adapter card, all of which depend on the particular end-system components. On the other-hand the software overheads may include device driver overhead, execution of higher layer protocols and the interactions with the host system's operating system. Hence the ATM network subsystem and I/O subsystem of the host has to be fine-tuned to achieve better communication bandwidths.

The actual performance gain on using ATM AAL5 is 3.38 times and that for switched Fast Ethernet is 9.56 times when compared to conventional unswitched Ethernet (10BASE-T). This indicates that considerable performance gain can still be achieved using a fast network inter-connect like either ATM or switched Fast Ethernet.

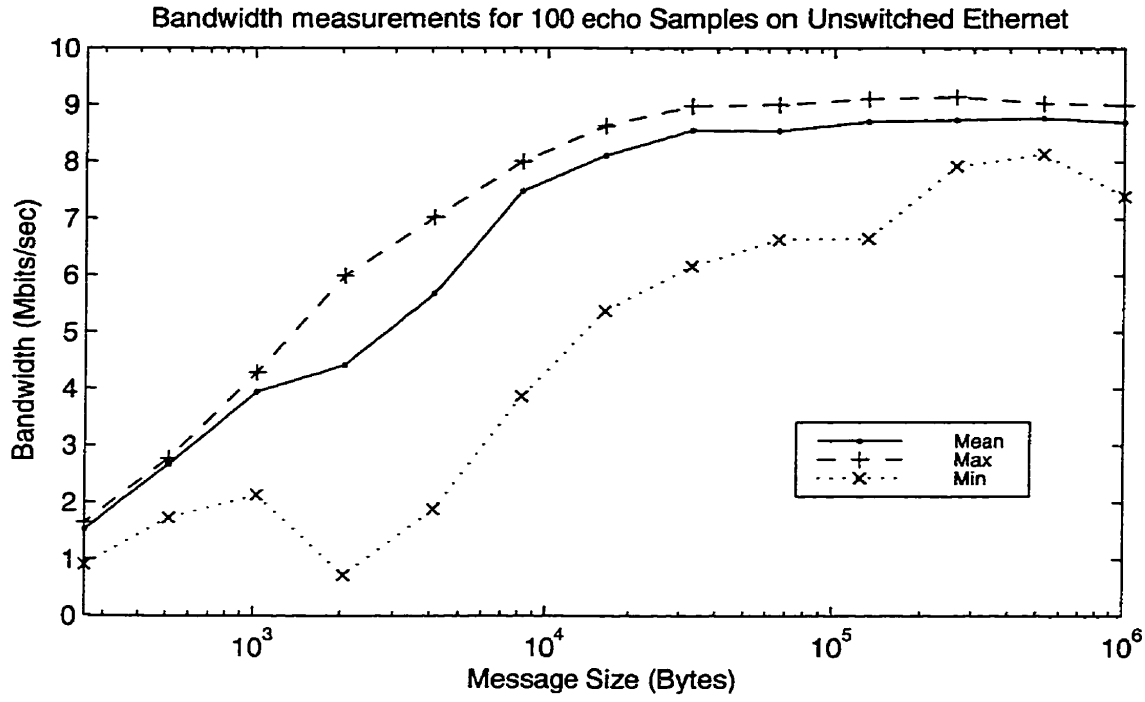


Figure 4.12: Communication Bandwidth for PVM clusters over Unswitched Ethernet

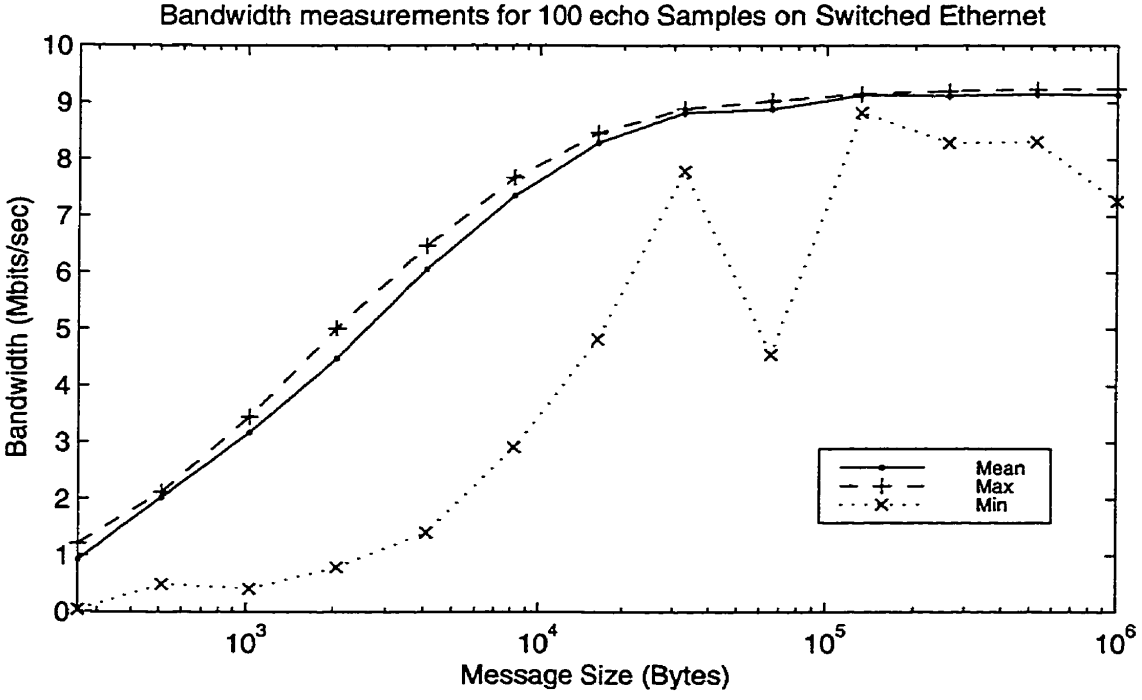


Figure 4.13: Communication Bandwidth for PVM clusters over Switched Ethernet

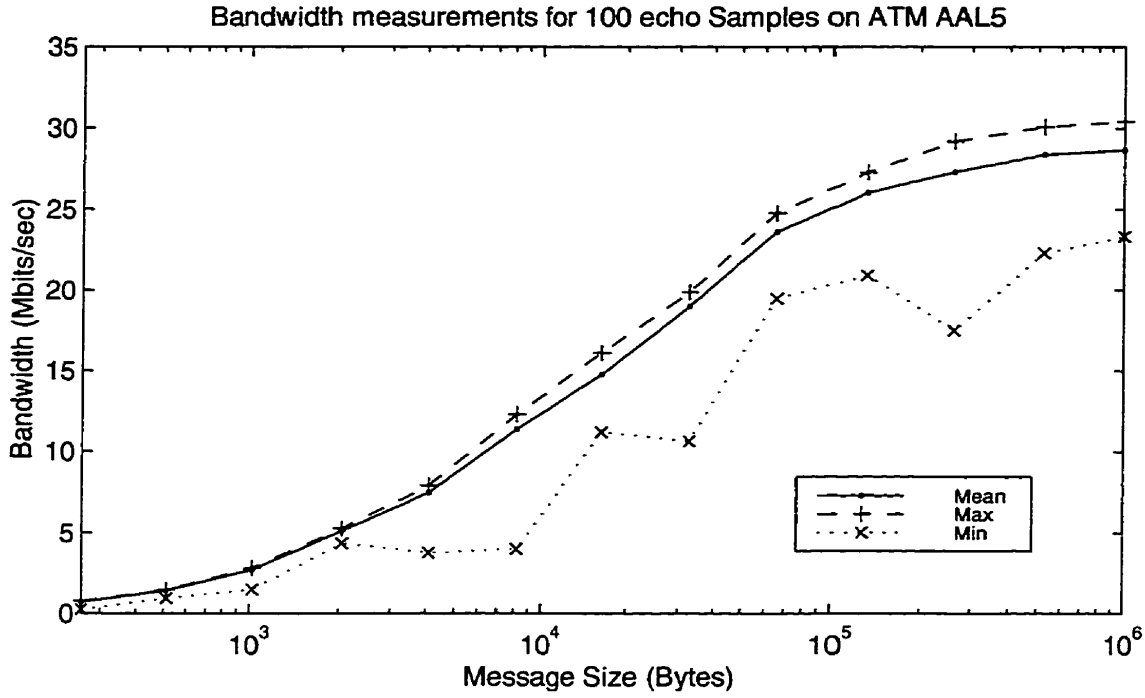


Figure 4.14: Communication Bandwidth for PVM clusters over ATM using AAL5

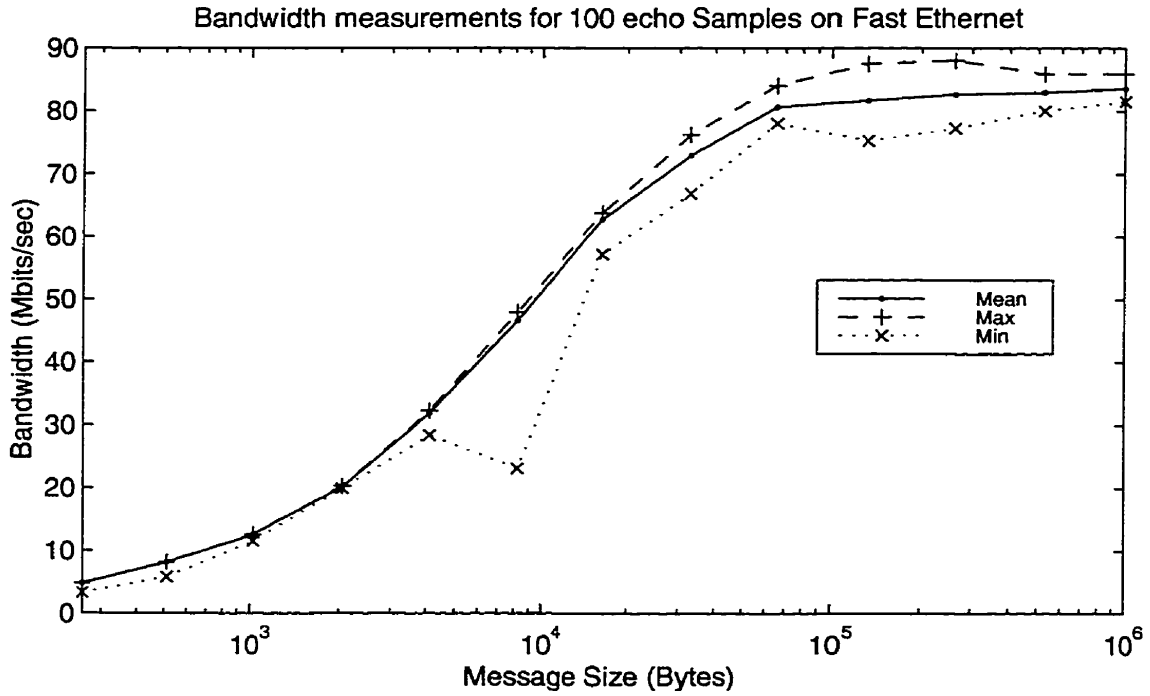


Figure 4.15: Communication Bandwidth for PVM clusters over switched Fast Ethernet

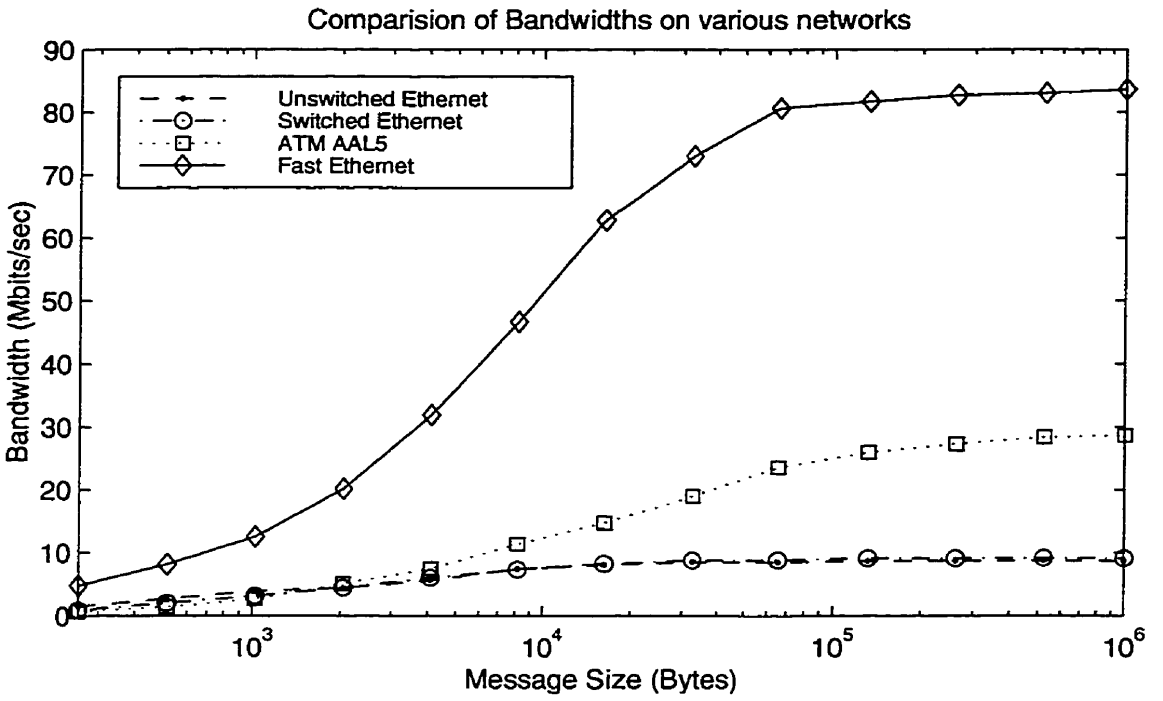


Figure 4.16: Communication Bandwidths for PVM clusters over various networks.

4.2.3 Half Performance length $n_{1/2}$

Half performance length ($n_{1/2}$) is the message size needed to achieve half the maximum achievable throughput. For a given software and hardware configuration, the smaller the $n_{1/2}$ value, the better the performance for medium-size messages. Thus $n_{1/2}$ provides a measure of how efficient a communication system is in delivering messages of intermediate length. The half performance length $n_{1/2}$ is calculated using the expression:

$$n_{1/2} = \frac{\alpha}{2/r_{max} - \beta}$$

where the constants α (start-up time) and β (transmission time per byte) are obtained by interpolating the data from the *echo* program and using the following expression:

$$RTT = 2(\alpha + \beta \times M)$$

where RTT is the round-trip time in seconds and M is the message size in bytes. This

calculation of $n_{1/2}$ is based on the assumption that all hardware and software overheads are included in a linear model of the process-to-process message transmission time [48]. The typical $n_{1/2}$ values for different networks are shown in Table 4.1.

Discussion of Results: As expected half Performance length ($n_{1/2}$) is found to be maximum for ATM followed by switched Fast Ethernet, Ethernet (switched) and Ethernet (unswitched) respectively. Thus, for medium-sized messages the clusters that were connected by Ethernet(unswitched) perform better and diminishing trend in performance was observed with Ethernet(switched), switched Fast Ethernet and ATM. At this point not much insightful explanation can be given to justify this behavior however since the maximum achievable throughput is a function of the particular systems' hardware and software configurations, $n_{1/2}$ metric only serves as a reference point indicating the message size needed to reach half of the maximum achievable throughput.

4.2.4 Communication Model

A communication model given by Clement [59] was used to predict the PVM communication time over Ethernet, ATM and Fast Ethernet. For a given program execution, the communication time T_{comm} as predicted by this model can be expressed as

$$T_{comm} = \sum_{i=1}^{N_c} \left(\xi + \frac{\gamma B_i}{\eta} \right) \tag{4.2.1}$$

where N_c is the total number of communications per processor, ξ is the message latency, η is the network bandwidth, B_i is the size of message i, and γ is the contention factor. The contention factor γ is 1 for ATM, switched Fast Ethernet and Ethernet (switched) and for unswitched Ethernet it is equal to p, the number of nodes, assuming that all the p nodes are communicating simultaneously.

The communication times predicted by this model are compared with the actual communication times in Figures 4.17, 4.18 and 4.19 for different message sizes and network media.

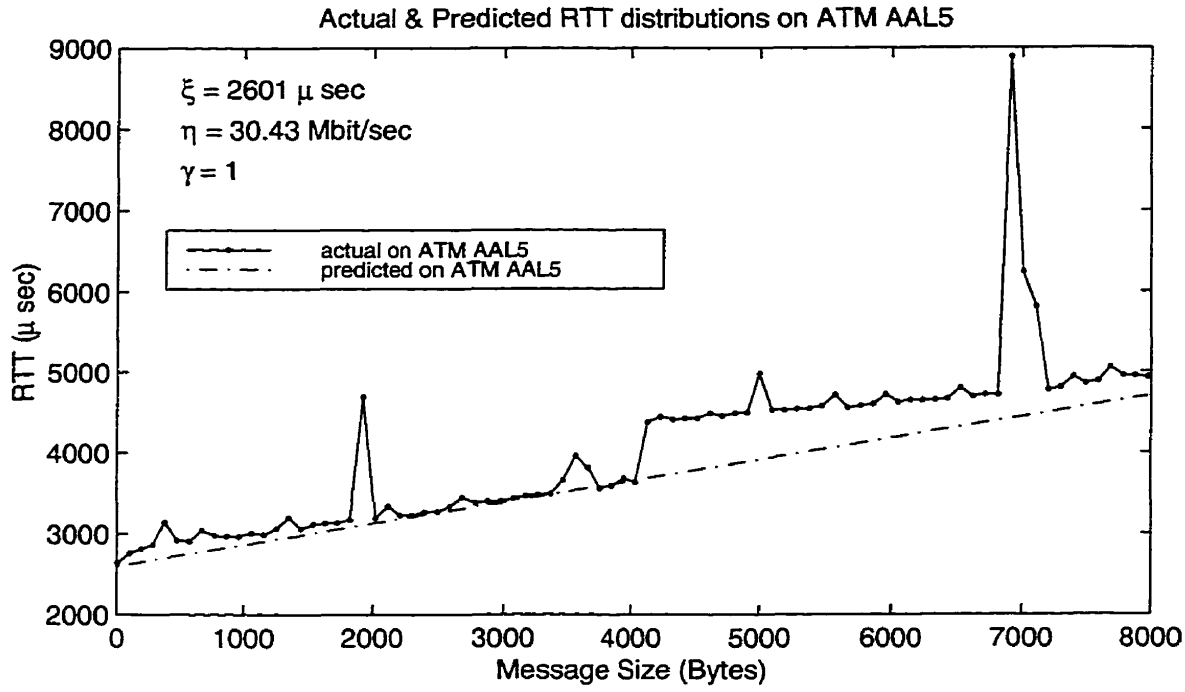


Figure 4.17: Predicted and Actual Communication times on ATM AAL5

*Discussion of Results:*The estimated round-trip times calculated using equation 4.2.1 were observed to be very close to the actual values found from the experiments. The standard deviation values were $260\mu sec$ for Ethernet(unswitched), $678\mu sec$ for ATM (using AAL5) and $363\mu sec$ for switched Fast Ethernet. Thus it can be inferred that the communication prediction model described by equation 4.2.1 is reasonably accurate for predicting the communication times in the above networks using the given values of γ, the contention factor.

4.3 Computation Performance

Computation performance was measured by using the two benchmark problems.

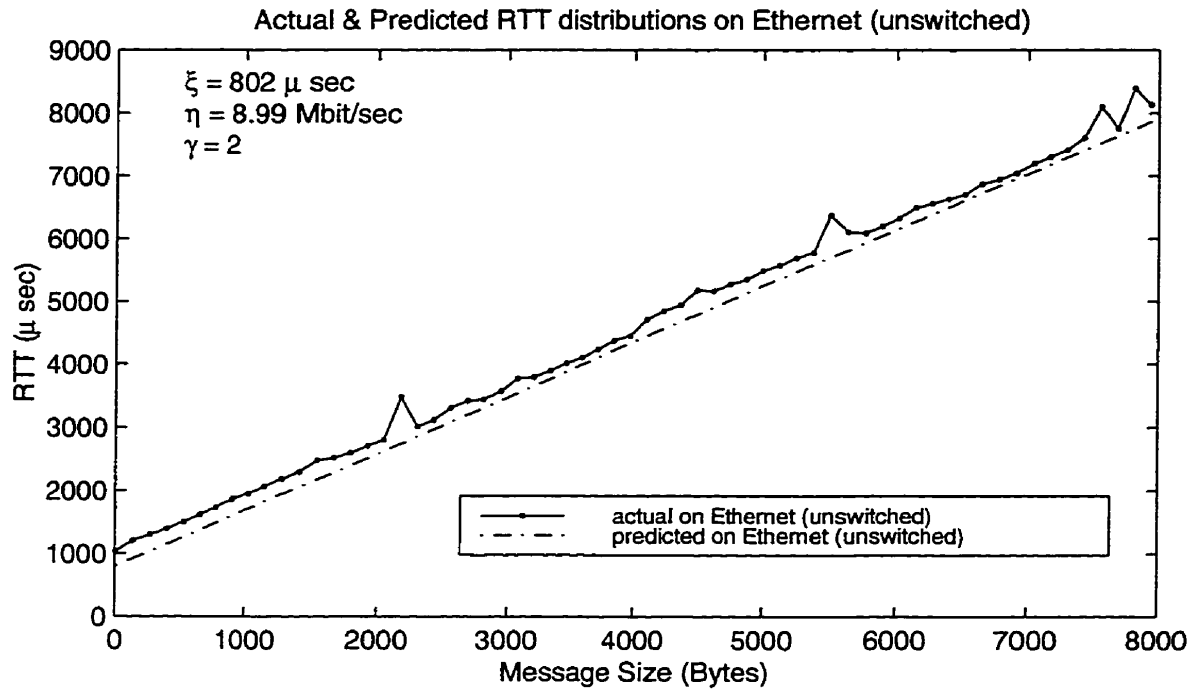


Figure 4.18: Predicted and Actual Communication times on Ethernet (unswitched)

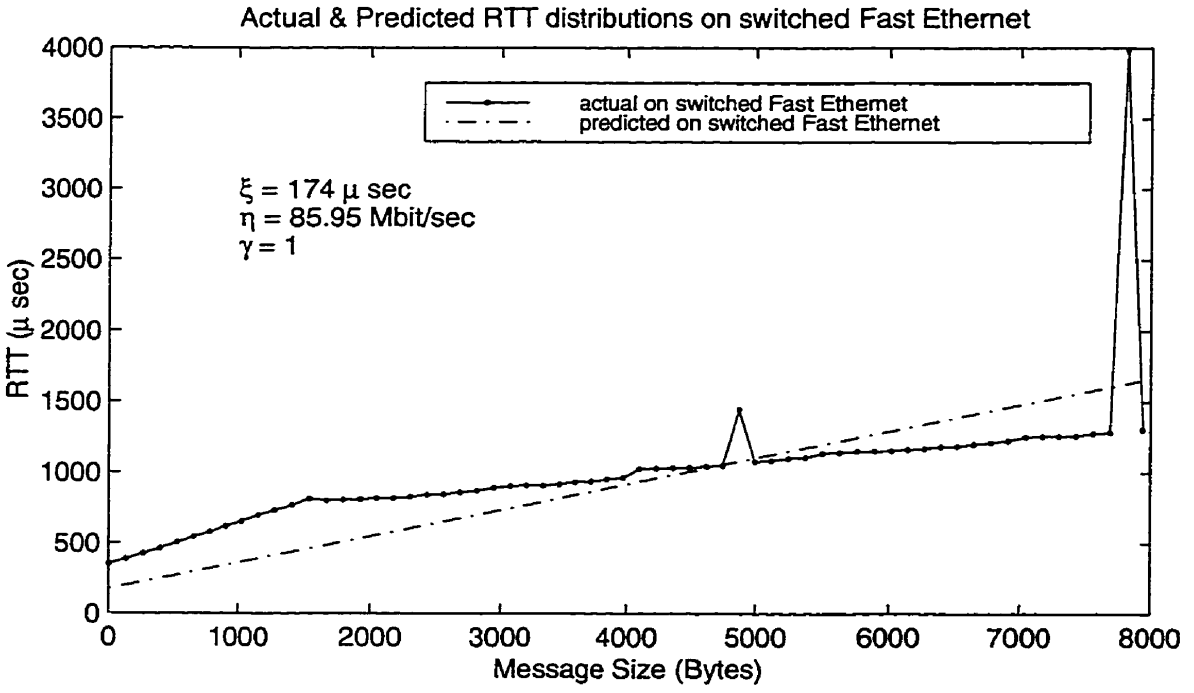


Figure 4.19: Predicted and Actual Communication times on switched Fast Ethernet

4.3.1 Speed-up and Efficiency

The computation efficiency of the parallel programs were measured using the performance metrics: *speed-up* factor, S_p, and *total efficiency*, E_p^{tot}. These are defined by:

$$S_p = \frac{T_s}{T_p}, \qquad E_p^{tot} = \frac{S_p}{p}$$

where T_s is the execution time for the best serial algorithm on a single node and T_p is the execution time for the parallelized algorithm using p nodes. For both the algorithms $T_s \neq T_1$. Another metric is the *algorithmic efficiency*, which is defined as:

$$E_p^{alg} = \frac{T_{p=1}}{pT_p}$$

where $T_{p=1}$ is the execution time for the parallel algorithm on a single node.

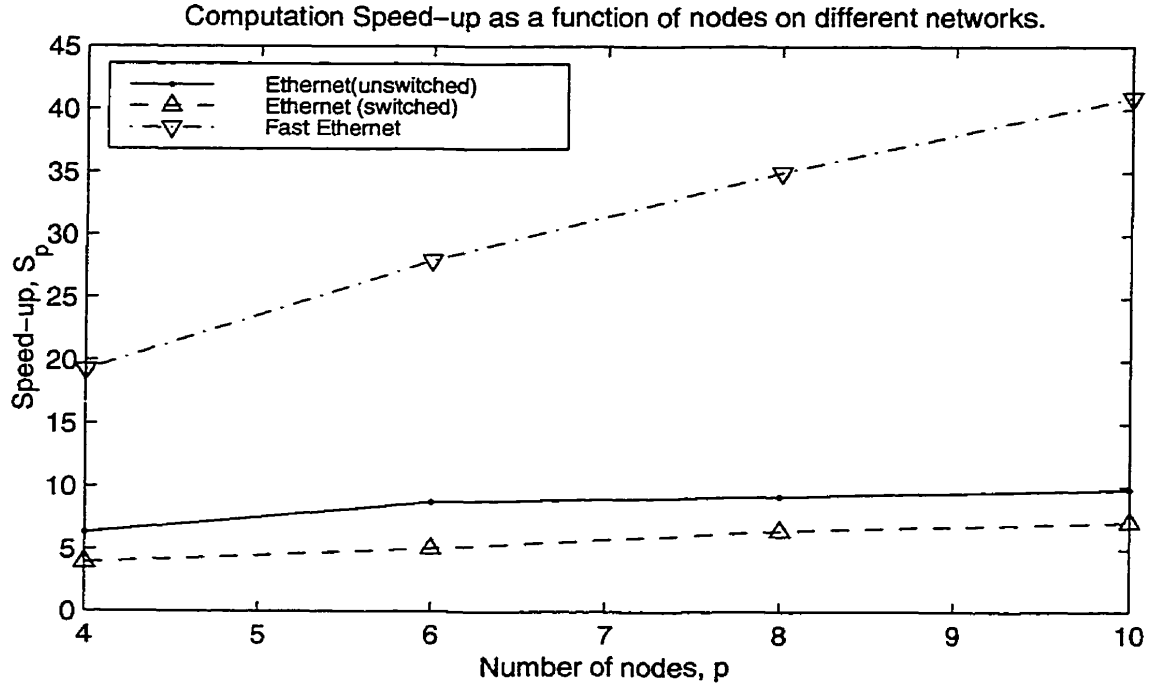


Figure 4.20: Matrix computation Speed-up on various networks.

Table 4.2: Matrix Computation (SUMMA) time (sec) on different networks.

Network Media (p = 4 nodes, 2 × 3 mesh)	Matrix size (n) 100	200	400	800	1600
Best Sequential (T_s)	1.33	11.32	93.94	756.73	1807.35
Parallel $T_{p=1}$ (1 node)	4.04	24.22	186.67	1534.51	2514.48
Ethernet (unswitched)	0.84	2.13	11.97	106.51	1247.97
Ethernet (switched)	0.23	0.95	7.08	57.68	657.19
ATM (AAL 3/4)	0.18	0.95	7.01	53.47	601.93
ATM (AAL 5)	0.17	0.96	6.99	50.25	532.28
Fast Ethernet (switched)	0.23	0.89	5.79	49.94	402.69

Table 4.3: S_p and E_p^{tot} of Matrix computation (SUMMA with n = 1600).

Network Media (p = 4 nodes, 2 × 3 mesh)	T_p seconds	S_p seconds	E_p^{tot} %	E_p^{alg} %	λ_p (calc)	λ_p (pred)
Ethernet (unswitched)	1247.97	1.45	36.25	50.37	1.72	0.36
Ethernet (switched)	657.19	2.75	68.75	95.65	-	-
ATM on AAL 3/4	601.93	3.00	75.00	104.43	-	-
ATM on AAL 5	532.28	3.39	84.75	118.16	0.77	0.11
Fast Ethernet (switched)	402.69	4.49	112.25	156.10	0.03	0.07

Table 4.4: Parallel Flow Simulation time (sec) on different networks.

Network Media (p = 4 nodes)	Number of Airfoils, N (each with 276 *dof*)				
	4	8	12	16	20
Best Sequential (T_s)	309	3078	8308	12823	+
Parallel $T_{p=1}$ (1 node)	351	6320	+	+	+
Ethernet (unswitched)	287	2978	+	+	+
ATM (AAL 5)	186	2243	3584	5898	9826
Fast Ethernet (switched)	88	816	2252	3537	6184
+ code exceeded the virtual memory. *dof* = degrees of freedom					

Table 4.5: S_p and E_p^{tot} of Parallel Flow Simulation (with 8 Airfoils).

Network Media (p = 4 nodes)	T_p seconds	S_p seconds	E_p^{tot} %	E_p^{alg} %
Ethernet (unswitched)	2978	1.03	25.75	53.05
ATM on AAL 5	2243	1.37	34.25	70.44
Fast Ethernet (switched)	816	3.77	94.25	193.62

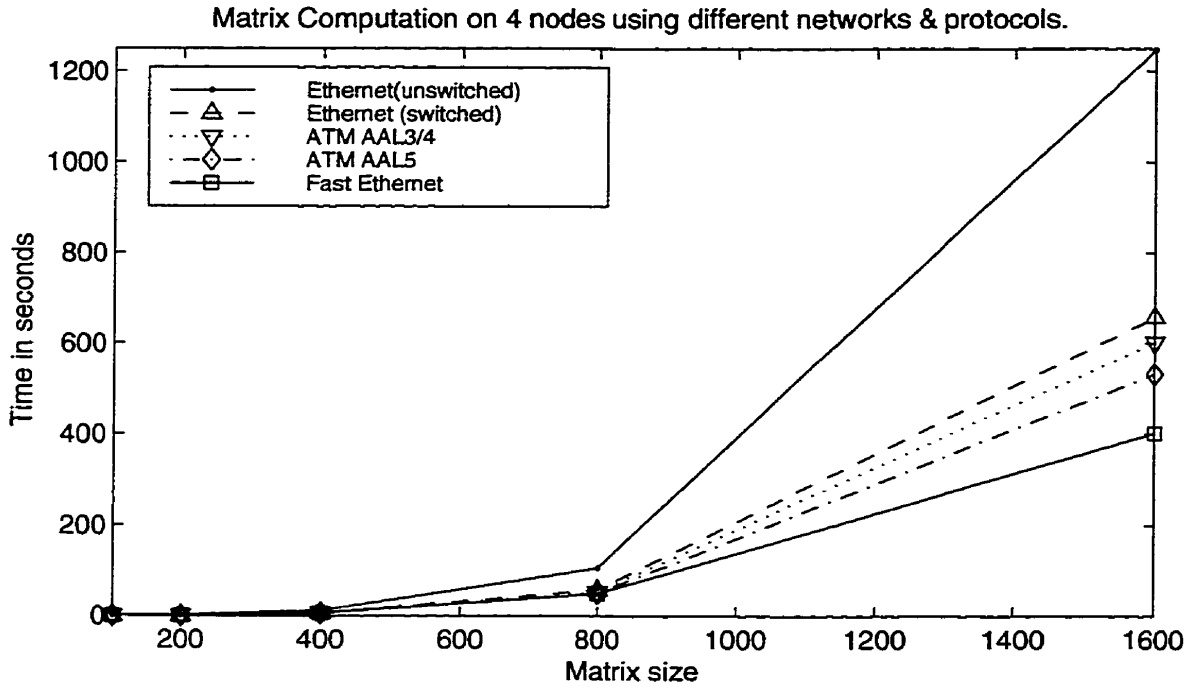


Figure 4.21: Matrix computation using PVM clusters over various networks.

4.3.2 Computation Model

The computation time, T_s, can be estimated as:

$$T_s = N_s^{flop} \tau$$

where N_s^{flop} is the total number of floating point operations and τ is the time per floating point operation ($\approx$ 0.22 sec/Mflop on a SUN SPARC 5). For executing a parallel algorithm on p nodes, the total execution time, T_p is the sum of computing T_p^{calc} and communication T_p^{comm} time:

$$T_p = T_p^{calc} + T_p^{comm} = N_p^{flop} \tau + T_p^{comm} \qquad \lambda_p = T_p^{comm} / T_p^{calc}$$

where N_p^{flop} is the total number of floating point operations in all the p nodes and T_p^{comm} is calculated from the Communication Model (Equation 4.2.1). Hence

$$E_p^{tot} = \frac{T_s}{pT_p} = \frac{N_s^{flop} \tau}{p\left(N_p^{flop} \tau + T_p^{comm}\right)} = \left[\frac{N_s^{flop}}{pN_p^{flop}}\right]\left[\frac{1}{1+\lambda_p}\right]$$

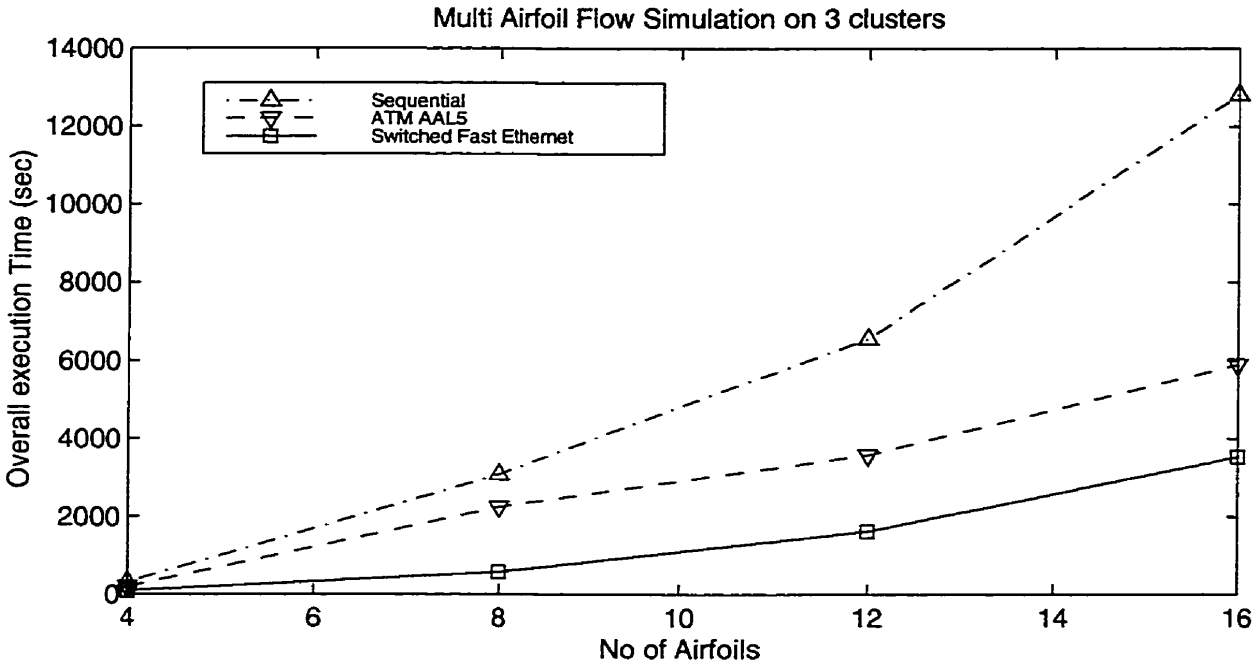


Figure 4.22: Performance of the flow code over the 3 different clusters of 4 nodes.

$$E_p^{tot} = E_p^{calc} E_p^{par}$$

Thus the major factors that affect the *total efficiency*, E_p^{tot} are the *computational efficiency*, E_p^{calc} and the *parallel efficiency*, E_p^{par}. E_p^{calc} accounts for the affect of some nodes being idle due to uneven load while E_p^{calc} accounts for the time spent in communication during which computation cannot take place.

Discussion of Results of parallel matrix computation: The results of the matrix multiplication algorithm tabulated in Table 4.2 indicate that the best sequential algorithm performs better than the parallel algorithm computed on one host. This is not surprising and is also reflected in the high algorithmic efficiency E_p^{alg} shown in Table 4.3. Although the $T_{p=1}$ times are machine dependent, their consistent increase with matrix size shows the process management overhead of the parallel algorithm on a single node compared to a single-process in the sequential algorithm.

Discussion of Results of flow simulation: Three flow codes (*Algo 1*, *Algo 2* and

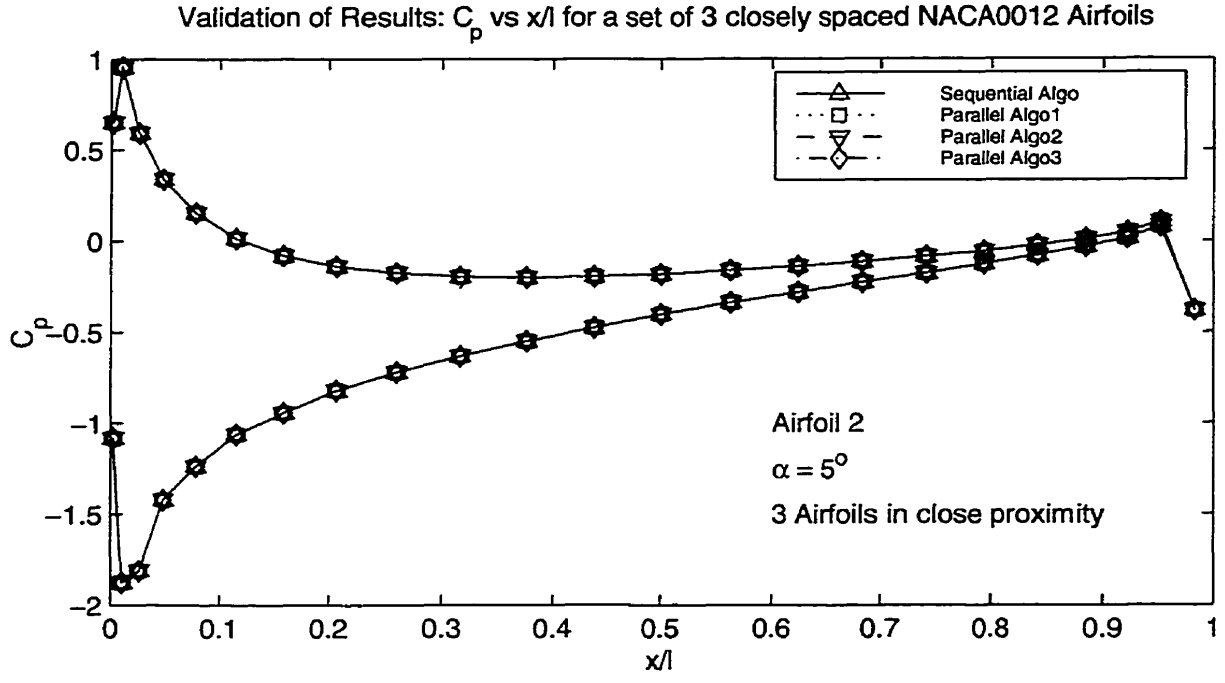


Figure 4.23: Validation of the 3 parallel algorithms.

Algo 3) were developed using 3 different partitioning techniques. The partitioning techniques differed in the number of communications and the size of the maximum block of data communicated as shown in Table 4.6. Overall execution time is affected by the partitioning technique used and this is reflected in the data of Table 4.7. It is clear from Table 4.7 that *Algo 3* has the minimum number of computations and the minimum block size of data communicated between nodes. Hence *Algo 3* was used for determining the overall execution time. The flow results obtained from the three codes were exactly same (see Figure 4.23).

It can be observed from Table 4.4 and Table 4.5 that the overall execution time T_p is the best on the switched Fast Ethernet cluster followed by the ATM (AAL5) cluster and then the Ethernet (switched) cluster. Figure 4.22 also shows the variation of the computation time with the best sequential algorithm (not the parallel algorithm running on one node). Comparing the *Speed-up*, S_p from Table 4.3 and Table 4.5, it can be observed that coarse granular problems perform better than fine granular

Table 4.6: Domain decompositions of the Parallel Flow Simulation code (n is the no. of airfoils used, m is the dof of each airfoil)

Algorithms (p = 4)	No. of communications	Max. size of matrix sent	No. of tasks forked
Algorithm 1	$4n$	m^2n^2	n
Algorithm 2	$2n^2 + 2n$	m^2	n^2
Algorithm 3	$4n$	m^2n	n

problems on all the three clusters.

4.4 Performance on Heterogeneous Clusters

Most LAN environments today are heterogeneous due to the commodity nature of workstations and the phased incorporation of new generation network equipment. Thus, heterogeneity may be reflected in terms of varying architecture, data formats, computational speed and machine or network loads. In a workstation cluster environment, heterogeneity can also be due to the communication capability of workstations, coexistence of multiple network architectures, and the availability of specialized support for collective communication and synchronization. Application performance can be significantly impacted by all such forms of heterogeneity in the cluster environment.

In this section, the node-heterogeneity of a workstation cluster on unswitched Ethernet is characterized. This characterization is done in terms of the overhead of point-to-point communication (e.g. start-up latency) and collective communications like broadcast and multicast latencies.

The testbed used in this set of experiments consisted of a 22 node SUN workstation cluster connected using unswitched 10BASE-T Ethernet. There were 8 fast

Table 4.7: Effect of domain decomposition on overall computation time on the ATM cluster.

No of Airfoils n	*Algo 1* T_p	*Algo 2* T_p	*Algo 3* T_p	*Seq. Code* T_s
1	1.69	2.14	1.48	0.52
2	9.32	5.01	2.95	9.76
3	19.6	11.34	4.76	25.6
4	52.29	20.29	11.13	71.24
++ With no matrix inversion and with $dof = 42$				

nodes (SUN Ultra 5/270s) and 14 slow nodes (7 SUN Sparc 5/170s, 4 SUN Sparc 5/70s and 3 SUN Sparc 5/85s). Measurements were taken on three different 16-node configurations with 12.5%, 25% and 50% fast nodes, respectively.

4.4.1 Broadcast Latency

Broadcast latency is defined as the elapsed time between the source node initiating a message and the last recipient receiving it. It was experimentally measured in the following way. In a P node system, $P-1$ broadcasts were performed by the source node and after each broadcast operation the source node received $P-1$ acknowledgments. At the source node, the time between the initiation of the broadcast operation and the receipt of the last acknowledgment was taken to be the maximum RTT. This maximum RTT is the sum of the broadcast latency and half of the round trip latency between the source and the last recipient node. The RTT latency between the source node and the last recipient node was measured by sending a message of the same length to the last recipient node from the source node. The broadcast latency[3] (on

[3]Broadcast latency on shared media can be measured using *snoop* or the *Berkeley Snoop Protocol* (refer http://www.cs.berkeley.edu/~hari/thesis/).

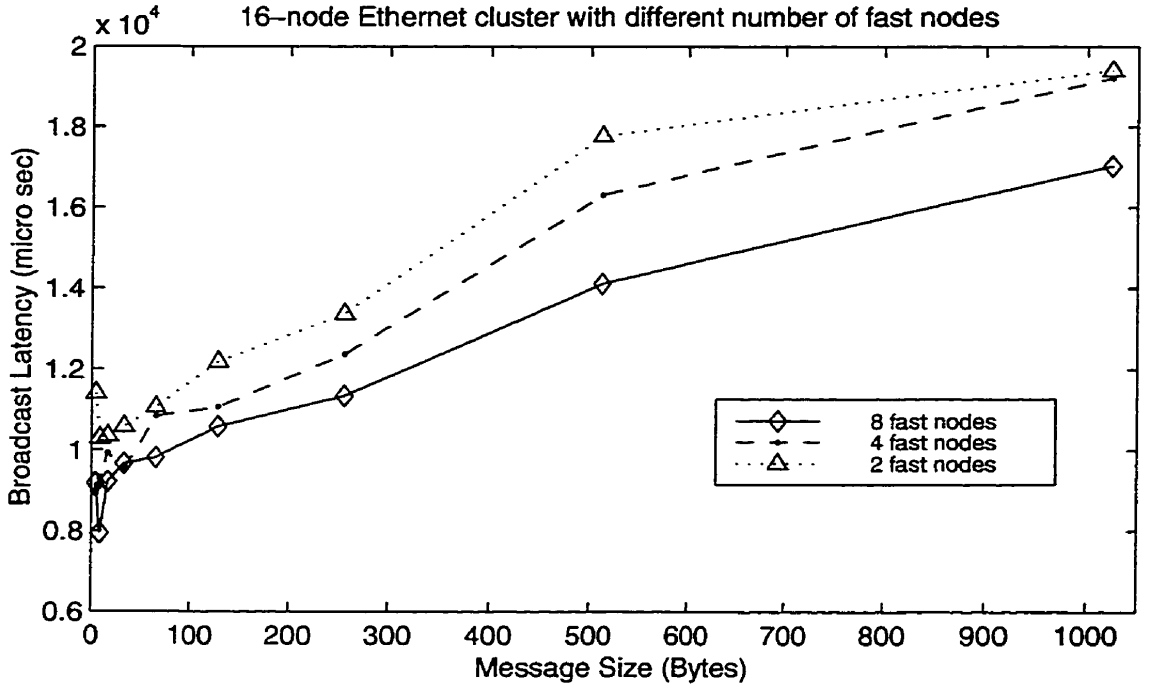


Figure 4.24: Broadcast Latency measurements using PVM/TCP over Ethernet.

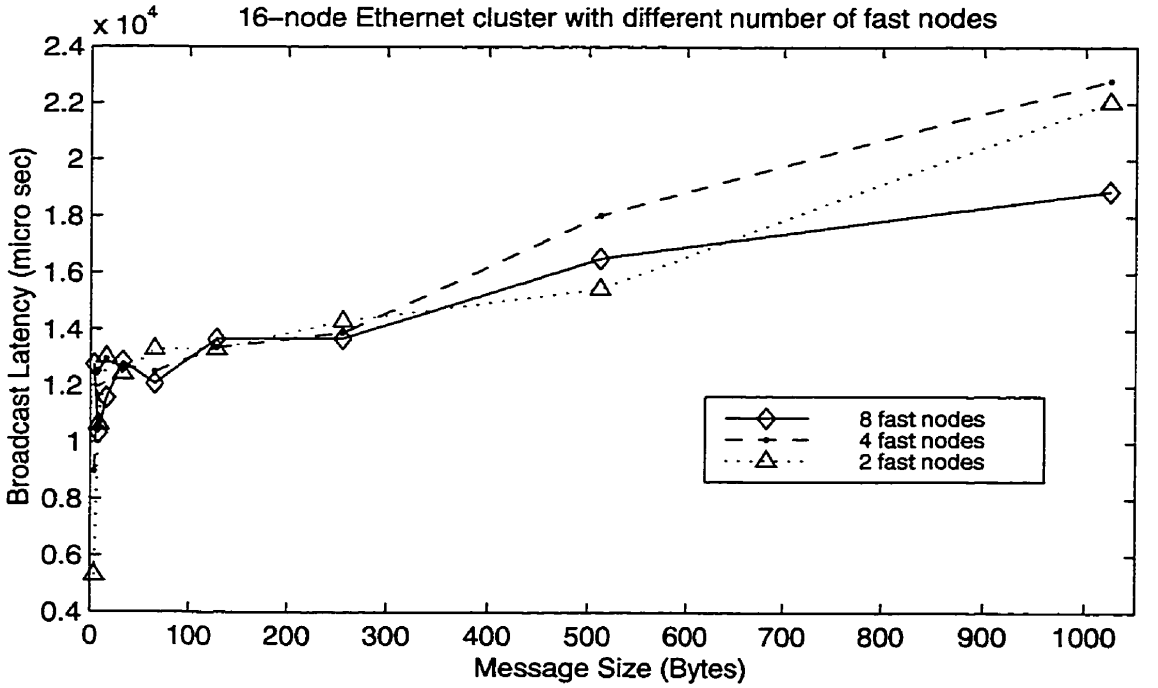


Figure 4.25: Broadcast Latency measurements using PVM/UDP over Ethernet.

a shared media like Ethernet) was obtained by subtracting half of the RTT from the maximum RTT reading. For each message size, 100 sample readings were taken and the minimum broadcast latency reported. Each experiment was repeated by varying the message size from 4 bytes to 1024 bytes. The graphs for broadcast latencies vs. message size with different fast and slow node configuration is slown in Figure 4.24 using TCP and in Figure 4.25 using UDP.

Discussion of Results: It can be observed from Figures 4.24 and 4.25 that the broadcast latencies are appreciably less when the TCP-based direct route is used for communication than the default route through UDP. The performance improvement is clearly marked when the set of nodes contains more faster nodes. This improvement in performance diminishes after a certain message size due to the fact that with increasing message size the transmission time dominates over the start-up latency. The effect of node heterogeneity could not be clearly observed due to network congestion.

4.4.2 Multicast Latency

The latency of sending a message from a source node to a set of recipients which is a subset of the set of nodes is termed the multicast latency. The same procedure for broadcast operations was repeated $100 \times (r-1)$ times, where r is the number of recipients. Here the *pvm_mcast()* function was used for multicasting. In Figure 4.26 and Figure 4.27 the multicast latencies using TCP and UDP are respectively shown.

Discussion of Results :Figures 4.26 and 4.27 show the results for multicast latencies. Ideally the performance should improve with an increase in the number of nodes in the cluster. Again as the message size increases the transmission time dominates and the performance decreases. It can be observed from Figures 4.26 and 4.27 that improvement is distinct when TCP based route is used instead of UDP. Some of the effects of node heterogeneity are, again, lost due to the network congestion.

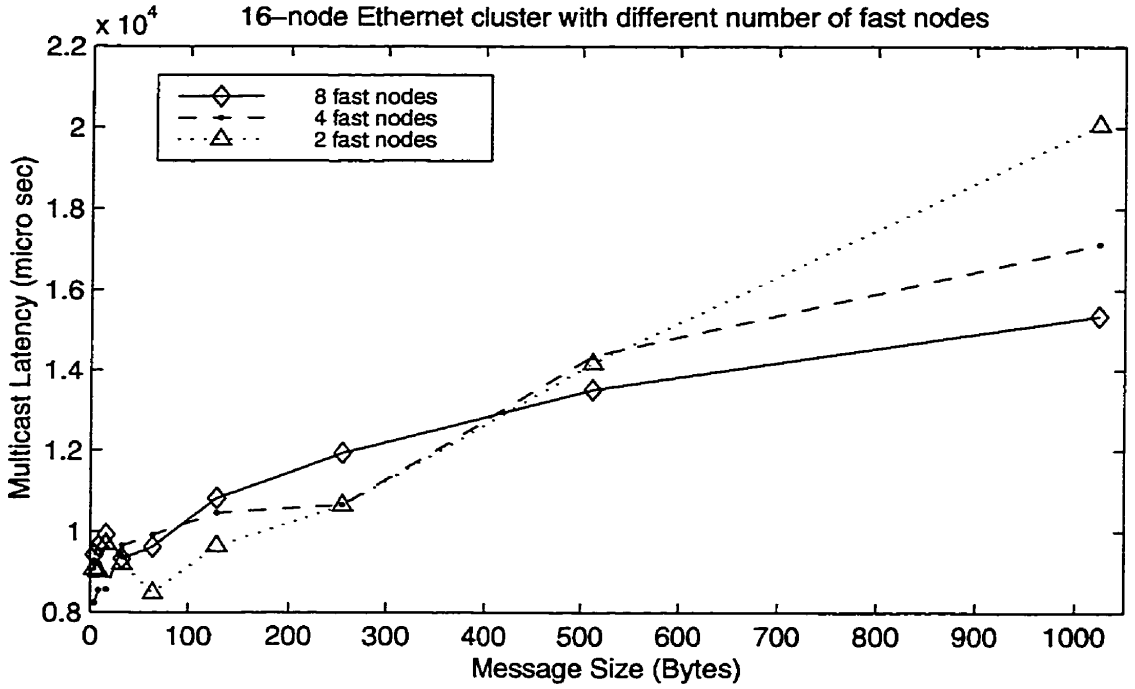


Figure 4.26: Multicast Latency measurements using PVM/TCP over Ethernet.

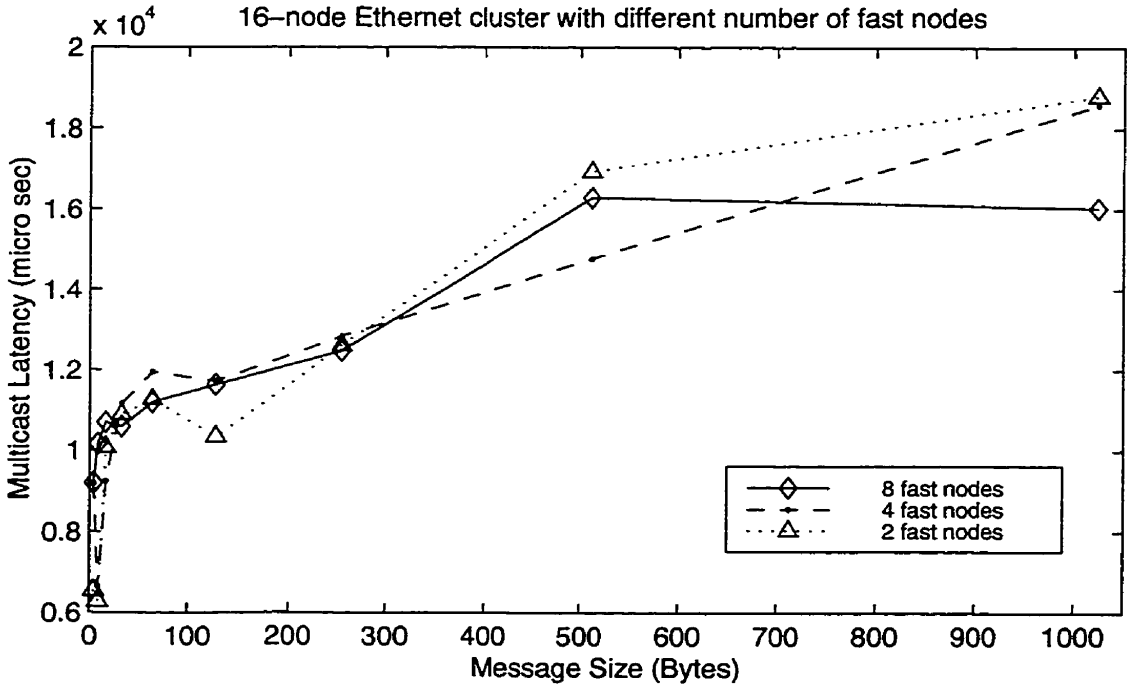


Figure 4.27: Multicast Latency measurements using PVM/UDP over Ethernet.

Table 4.8: Start-up latency (in μsec) on Ethernet for heterogeneous nodes.

	Ultra 5/270	Ultra 2/2300	Sparc 5/170	Sparc 2/100	IBM RS 6000/590
Ultra 5/270 (UP)	656	1374	1954	5868	2266
Ultra 2/2300 (SMP)		764	928	1975	1597
Sparc 5/170 (UP)			1041	2119	1748
Sparc 2/100 (UP)				3266	2736
IBM RS6000/590 (UP)					-

4.4.3 Point-to-Point Communication

The round trip latency between five different pairs of workstations with varying architectures, speeds and operating systems were measured. The workstations were connected through 10-BASE-T Ethernet and were running PVM version 3.3.11 for point-to-point communication. A 4 byte message was sent between the sender and receiver nodes and the averages of 100 such RTTs are shown in Table 4.8.

Discussion of Results: It can be observed from Table 4.8 that point-to-point communication is much faster between homogeneous nodes than between heterogeneous nodes. For instance the SUN Ultra 5/270 takes 8.95 times longer to communicate with a SUN Sparc 2/100 than with another similar Ultra 5/270 node. It can also be observed that the communication start-up time in the SUN Ultra 2/100 is around 5 times that of a SUN Ultra 5/270. Hence it can be inferred that heterogeneity of the nodes may have a significant effect on the communication start-up times.

4.5 Summary

This chapter concludes with a summary of the experimental results. Latency analysis showed performance gains in all the 4 clusters when TCP/IP was used instead of UDP/IP as the communication protocol. Switched Fast Ethernet had the best start-up latency, the worst being on ATM. This is attributed primarily to the overhead of ATM device drivers. The high NIC and PVM-ATM system overhead resulted in low application bandwidth (and half performance lengths) of ATM compared to switched Fast Ethernet which had the maximum throughput although the raw bandwidth of ATM was higher. The network model used predicted fairly close RTT times compared to the actual times for all the clusters. The overall execution times for the matrix multiplication problem and the flow simulation problem was the least in Fast Ethernet followed by ATM, although it showed some speed-up anomalies (superlinear speed-ups). The *Speed-up,* S_p, factor was found to be more for the coarse grained problem (matrix multiplication), as expected, than the fine grained problem (flow simulation). Point-to-point communication was found to be faster (as expected) between two homogeneous nodes than heterogeneous ones. In a heterogeneous cluster, as the number of faster nodes was increased, multicast and broadcast performance improved initially but as the message size increased it diminished since transmission time became significant.

"When you have eliminated all which is impossible,
then whatever remains, however improbable, must be the truth".
- Sherlock Holmes by Sir Arthur Conan Doyle.

Chapter 5

Conclusions and Future Work

This Thesis concludes with a summary of the inferences drawn from the work and an enumeration of possible directions for future work.

5.1 Summary

The observations drawn from this study of the three distributed clusters on high-speed networks can be summarized as follows:

- *Unswitched Ethernet:* Unswitched Ethernet was found to have a small start-up latency (802μs) compared to switched Ethernet, since its route involved a single-hop instead of 4 hops in the latter. However, the maximum achievable throughput (r_{max}) was higher in switched Ethernet, although both the networks had a raw bandwidth of 10 Mbps. This reflects the impact of having a switch for internode message transfers. The coarse grained problem showed the highest overall execution time on this network although it did achieve a speed-up, S_p, of 1.45 on 4 processors. From these results it can be concluded that unswitched Ethernet is suitable only for those distributed applications which involve very

little message passing (due to low $n_{1/2}$) or where minimizing the total execution time would be an alternate goal.

- *Switched Ethernet:* Switched Ethernet had a start-up time (1506μs) which was lower than ATM (due to less NIC overhead) but much higher than switched Fast Ethernet (due to the relatively slower data link). Switched Ethernet however had the highest network bandwidth utilization[1] (92.4%) compared to unswitched Ethernet and other networks. Again the impact of the switch can be seen, reducing the packet collision domain through switching. The execution time of the coarse grained problem was significantly less than on the unswitched Ethernet. These results were for virtually silent network load conditions but with high-network load such performance improvement may not be noticeable. From these results the same general conclusions as for the unswitched Ethernet can be drawn.

- *ATM:* ATM showed the highest start-up latency (2601μs), for reasons discussed in Section 4.2.1. It also resulted in the highest half-performance length and the lowest network bandwidth utilization (19.6%). In Section 4.2.2 and 4.2.3, the reasons for this poor performance of ATM for data communication in LANs was discussed. Compared to Ethernet (both switched and unswitched) a performance gain for coarse granular problems on ATM was observed. This gain was observed when either of the protocols ATM AAL 3/4 or AAL 5 was used, the maximum gain being with AAL5. It can be concluded that although ATM has a very high link bandwidth (155 Mbps), in its present state, ATM is not suitable for distributed applications involving frequent, small or even medium size message transfers because of latency issues. It is however a very good technology for backbone networks, WANs or other applications including multi-media communications. Thus to effectively use ATM as a cluster interconnect, further

[1]Network bandwidth utilization is the ratio of the actual bandwidth obtained to the raw available bandwidth.

research is needed to fine-tune its performance for LANs.

- *Switched Fast Ethernet:* The best results on both communication and computation performance were obtained by using the switched Fast Ethernet. A start-up latency of 174 μs, r_{max} of 85.95 Mbps, with 85.9% network bandwidth utilization and an $n_{1/2}$ of 6628 bytes was observed. The coarse grained problem performed the best on this network, with S_p of 4.49 and E_p^{tot} of 112.25%. These results clearly indicate the suitability of switched Fast Ethernet as a cluster interconnect for both coarse and fine granular distributed applications.

- In the context of distributed cluster computing as opposed to traditional parallel computing on MPP systems, it can be concluded, as would be expected, that the coarse granular problems performed better than the fine granular problems. On the communication side, TCP/IP based routing was found to be give faster communication than UDP/IP routing both for homogeneous and heterogeneous distributed clusters. On the computation side, we observe that algorithmic domain decomposition can have a significant impact on the overall execution time. Since decomposing data onto processors is a difficult (NP-complete) optimization problem there is need for a focus on good heuristics.

5.2 Future Directions

Based on the lessons learned from this study several future directions for research can be suggested. Some of these can be pursued as a logical extension of this work, while others are problems of a more general nature.

- *Reducing the spin-waiting latencies of parallel subtasks:* In this thesis the PVM system used had a default scheduler that scheduled atomic jobs over a predefined number of nodes using a round-robin scheme. It's scheduling policy was not based on the current workload of the nodes. A more dynamic scheduler could

be developed that would make more informed scheduling decisions based on the runtime load information of the nodes (like user/system CPU utilization, average number of jobs in the run queue, etc.). When spawning a task, the PVM resource manager could select the most lightly-loaded host from the host pool in order to attempt to balance the compute load. Ideally such a scheduling scheme would facilitate efficient use of existing computational resources by reducing the communication latency and response time (due to reduced spin-waiting periods and context switching latencies) of PVM jobs comprising of parallel subtasks. In Appendix III the design of one such scheduler is described.

- *Latency-hiding using time parallelism in the algorithms:* It was noted by analyzing the experiments that in distributed clusters, depending on the cluster interconnect, the *start-up* times can be as high as the computation times for some applications. This results in large communication latencies. There are two *latency hiding* techniques that can be used to ameliorate this problem. One technique is overlapping communication with computation by using non-blocking routines (send, receive). Thus while a process is waiting for a communication to be completed it can do some useful computation and thus be kept busy. The other technique is mapping multiple processes onto a single processor and using a time-sharing facility to efficiently context-switch from one process to another. To achieve this *parallel slackness*[2], *threads*[3] can be used to minimize the context switching overhead.

- *Faster PVM communication on ATM via direct use of ATM API:* In this thesis the PVM implementation over the Fore Systems' ATM API was used. However, the Fore Systems' ATM API library routines provide a connection-oriented

[2]When an m-process algorithm is implemented on an n-processor machine, the algorithm is said to have a *parallel slackness* of m/n for that machine.

[3]Thread is a *light-weigh* process which share the same address space and global variables with other threads unlike processes.

client and server model. It provides a socket-like portable interface to the ATM data link layer with support from the underlying device driver. Applications could be developed using these API routines which would result in better communication performance due to reduction of the protocol overhead [20].

- *Incorporating Fault-tolerance in PVM:* The PVM system used in this thesis is not rigorously fault tolerance. It does not provide any primitives for task check-pointing, migration, suspension, resumption or a combination of these to support adaptive execution. A discussion of some of the current work in this area is briefly mentioned in Appendix III.

- *Load balancing in heterogeneous clusters:* Distributed clusters may be heterogeneous, varying in their processor speeds, interconnects, communication capability, available memory, etc. In this research only the collective performance of heterogeneous clusters were evaluated. However, minimizing execution times by partitioning data effectively among a set of nodes, based on their processor speeds and communication cost is an important problem area that can be explored ([109], [110]).

- *Algorithmic improvement with parallel out-of-core LU decomposition:* The flow simulation problem of this thesis involved the solution of a large linear system of equations. A parallel Jacobi iteration technique was used, which does not always converge[4] or can have exceedingly slow[5] convergence rate. When the matrices (which are dense) become larger they may not fit entirely in physical memory. This calls for new solution techniques. One approach would be to use the parallel out-of-core factorization routines of ScaLAPACK [108].

- *Distributed computing on hybrid clusters:* With clusters becoming heteroge-

[4]It converges only if the matrix is strictly diagonally dominant and the convergence rate depends on the largest eigen-value in modulus of iteration matrix $-D^{-1}(L+U)$

[5]It converges in $\log(n)$ steps, the parallel time complexity with n nodes is $\Theta(\log(n))$

neous in terms of machine architectures (workstations or PCs) that run a variety of operating systems (Windows 98/NT, Mac, LINUX etc.), similar research could be done using a combination of such diverse clusters. The recent version of PVM (Version 3.4) has the ability to provide message-passing support in such a hybrid cluster architecture. Assessing the effects of this form of heterogeneity would be interesting.

"If I have seen further it is by standing on the shoulders of Giants".

- Sir Isaac Newton

Bibliography

[1] Simon, H. D., *et al.* (1992): Parallel CFD: Current Status and Future Requirements, Parallel CFD : implementations and results, *The MIT Press*, 1-29.

[2] Pfister, G. F., (1998): In Search of Clusters, *Prentice Hall*, PTR, Upper Saddle River, NJ.

[3] Lau, L., (1996): Implementation of Scientific Applications on Heterogeneous Parallel Architectures, PhD Thesis, Dept of Mathematics, University of Queensland, Australia.

[4] Smith, M. H., *et al.* (1991): Numerical Simulation of a Complete STOVL Aircraft in Ground Effect, *AIAA* Paper 91-3293.

[5] Geist, G. A., *et al.* (1994): PVM : Parallel Virtual Machine - A User's Guide and Tutorial for Networked Computing, MIT Press.

[6] Gropp, W., *et al.* (1994): Using MPI : Portable Parallel Programming with the Message-Passing Interface, MIT Press.

[7] Bell, G. (1992): Ultra-computer: A Teraflop before its Time. *Comm. ACM*, 35(8), 26-47.

[8] Mukta, M., *et al.* (1991): The available capacity of a privately owned Workstation Environment, Performance Evaluation, Vol. 12, No. 4, 269-284.

[9] Zhou, H., *et al.* (1995): Faster Message Passing in PVM, In Proceedings of the First International Workshop on High Speed Network Computing, *IEEE Computer Society Press*, 67-72.

[10] Anderson, T. E. *et al.* (1995): A case of NOW (Network of Workstations), *IEEE Micro*, 15(2), 54-64.

[11] Zhou, H., *et al.* (1995): LPVM: A step towards multithreaded PVM, *J. of parallel and Distributed Computing*, July, 1995.

[12] Ni, L. M. *et al.* (1993): A Survey of Wormhole Routing Techniques in direct networks, *IEEE Computer*, Vol. 26, No. 2, 62-76.

[13] Minoli, D. (1994): 1st, 2nd & Next Generation LANs, McGraw-Hill Inc.

[14] Renwick, J. (1992): Building a practical HIPPI LAN, Proceedings of 17th IEEE Conference on Local Computer Networks.

[15] Andersen, T. M., *et al.* (1992): High Performance Switching with Fibre Channel, IEEE Proceedings of Comp Con, 261-264.

[16] Vetter, R. J. (1995): ATM Concepts, Architectures, and Protocols, *Communications of the ACM*, Vol. 38, No. 2.

[17] Lyles, J., *et al.* (1992): The Emerging Gigabit Environment and the Role of Local ATM, *IEEE Communications Magazine.*

[18] Boudec, J. (1992): The Asynchronous Transfer Mode : A Tutorial, *Computer Networks and ISDN Systems*, Vol. 24, 279-309.

[19] Biagioni, E., *et al.* (1993): Designing a Practical ATM LAN, *IEEE Network*, 32-39.

[20] Lin, M., *et al.* (1995): Distributed Network Computing over Local ATM Networks, *IEEE Journal on Selected areas in Communications*, 13(4), 733-748.

[21] Huang, C., *et al.* (1995): A thread-based Interface for Collective Communication on ATM Networks, *Proceedings of the International Conference on Distributed Computing Systems*, Vancouver, BC, 254-261.

[22] Boden, N., *et al.* (1995): Myrinet - A Gigabit-per-Second Local Area Network, *IEEE Micro*, 15(1), 29-36.

[23] Stallings, W. (1997): Local and Metropolitan Area Networks, Fifth edition, Upper Saddle River, NJ, *Prentice Hall.*

[24] Schoch, J., *et al.* (1980): Measured Performance of an Ethernet Local Network, *Communications of ACM.*

[25] Black, D. (1998): Managing Switched Local Area Networks - A Practical Guide, *Addison-Wesley.*

[26] Chang, S. L., *et al.* (1995): Enhanced PVM Communications over a High-Speed Local Area Network, *Proceedings of the First International Workshop on High-Speed Network Computing*, Santa Barbara, California.

[27] Martin, R. P., *et al.* (1997): Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture, *Proc. of the* 24^{th} *Annual International Symposium on Computer Architecture*, 85-97.

[28] Wilson, G. V., (1995): Practical Parallel Programming, MIT Press, Cambridge, Massachusetts.

[29] Gustavson, D., (1992): The Scalable Coherent Interface and related standard projects, *IEEE Micro*, 12(1), February 1992, 10-12.

[30] Gillett, R., *et al.* (1997): Using the Memory Channel Network, *IEEE Micro*, Vol. 17, No. 1, 19-25.

[31] Scott, S. L., (1996): Synchronization and Communication in the T3E Multiprocessor. In *Proc. of 7th Inter. Conf. on Architectural Support for Prog. Languages & Operating Systems.*

[32] Hsieh, J., *et al.* (1996): Enhanced PVM Communications over HIPPI Networks, *Proceedings of the Second International Workshop on High-Speed Network Computing (HiNet' 96)*, Honolulu.

[33] Ferrari, A., *et al.* (1995): TPVM: Distributed Concurrent Computing with Lightweight Processes, *Proceedings of the 4th High-Performance Distributed Computing Symposium*, Washington, DC, 211-218.

[34] Rodrigues, S., *et al.* (1997): High-Performance Local Area Communication with Fast Sockets, *Proceedings of USENIX' 97*, Anaheim, California, Jan 1997.

[35] Agarwal, A., *et al.* (1995): The MIT Alewife Machine: Architecture and Performance, In *Proc. of the 22nd Inter. Symp. on Computer Architecture*, 2-13.

[36] Chiou, D., *et al.* (1995): StarT-NG: Delivering seamless Parallel Computing. in *EURO-PAR'95 Conference.*

[37] Blumrich, M. A., *et al.* (1994): Virtual Memory mapped Network Interface for the SHRIMP Multicomputer, In *Proc. of the 21st Inter. Symp. on Computer Architecture.*

[38] Arpaci, R., *et al.* (1994): Empirical Evaluation of the CRAY-T3D: A Compiler Perspective, In *Proc. of the 22nd Inter. Symp. on Computer Architecture.*

[39] Pakin, S., *et al.* (1997): Fast Messages: efficient, portable communication for workstation clusters and MPPs, *IEEE Concurrency*, 5(2), 60-73.

[40] Eicken, T. v., *et al.* (1992): Active Messages: a mechanism for Integrated Communication and Computation, In *Proc. of the International Symposium on Computer Architecture*

[41] Dubnicki, C., *et al.* (1997): VMMC-2: efficient support for reliable, connection-oriented communication, In *Proc. of Hot Interconnects V. IEEE.*

[42] Simon, J., *et al.* (1997): SCI multiprocessor PC cluster in a Windows NT environment, *Supercomputer Journal*, XIII-2.

[43] Ridge, D., *et al.* (1997): Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs, *Proceedings, IEEE Aerospace.*

[44] Eicken, T. v., *et al.* (1995): U-Net: A user-level network interface for parallel and distributed computing, In *Proc. of the* 15^{th} *ACM Symp. on Operating System Principles*, 40-53.

[45] Benikazemi, M., *et al.* (1998): Efficient collective communication on heterogeneous network of workstations, *Tech. Rep. OSU-CISRC-03/98-TR07*, The Ohio State University.

[46] Roberts, E., (1997): Gigabit Ethernet: Fat Pipe or Pipe Bomb?, In *Data Communications*, May.

[47] Lamport, L., (1978): Time, clocks, and the ordering of events in a distributed system. *Comm. ACM*, 21(7), 558-565.

[48] Schmidt, B. K., Sunderam, V. S. (1994): Empirical analysis of overheads in cluster environments, *Concurrency: Practice and Experience*, 6(1).

[49] Iannello, G., *et al.* (1997): PVM communication performance over an ATM MAN, *J. of Systems Architecture*, 43, 167-173.

[50] Manchek, R. J., (1994): Design and Implementation of PVM Version 3, *Master of Science Thesis*, University of Tennessee, Knoxville.

[51] Geist, A., *et al.* (1994): PVM: Parallel Virtual Machine, *MIT Press.*

[52] Robin, C., *et al.* (1994): Portable programming with the PARAMACS message-passing library, *Parallel Computing*, 20(4):615-632.

[53] Skjellum, A., *et al.* (1992): The Zipcode message-passing system, *Technical Report, Lawrence Livermore National Laboratory*, September, 1992.

[54] Gropp, W. D., *et al.* (1993): Chameleon parallel programming tools users manual, *Technical Report ANL-93/23*, Argonne National Laboratory.

[55] Geist, A., *et al.* (1992): Network-based concurrent computing on the PVM system, *Concurrency: Practice & Experience*, 4(4), 293-311.

[56] Sunderam, V. S., (1990): PVM: A framework for parallel distributed computing, *Concurrency: Practice & Experience*, 2(4).

[57] MPI Forum, (1994): MPI: A message-passing interface standard, *International J. of Supercomputer Applications*, 8(3/4), 165-416.

[58] Nog, S., *et al.* (1996): A Performance comparison of TCP/IP and MPI on FDDI, Fast Ethernet, and Ethernet, *Technical Report PCS-TR95-273*, Department of Computer Science, Dartmouth College, NH.

[59] Clement, M. J., *et al.* (1996): Network Performance Modeling for PVM Clusters, *SuperComputing 1996.*

[60] Butler, R., *et al.* (1993): User's guide to the p4 parallel programming system, Ver 1.3, *Technical report ANL-92/17*, Argonne National laboratory, Argonne, IL.

[61] Birman, K., *et al.* (1989): ISIS and the Meta Project, *Sun Technology*, Summer 1989.

[62] Carriero, N., *et al.* (1989): LINDA in context, *Communications of the ACM*, 32(4), 444-458.

[63] Cap, C. H., *et al.* (1992): The PARFORM - a High Performance platform for Parallel Computing in a distributed workstation environment, *Technical report 92.07*, University of Zurich Information Institute.

[64] Flower, J., *et al.* (1991): The Express way to distributed processing, *Supercomputing Review*, 54-55.

[65] Geist, G. A., *et al.* (1991): A User's guide to PICL: A portable instrumented Communication Library, *Technical Report ORNL/TM-11616*, Oak ridge National Laboratory, Oak Ridge, TN.

[66] Harrison, R., (1991): Portable tools and applications for Parallel Computers, *Intern. J. Quantum Chem.*, 40, 847-863.

[67] Levesque, J., *et al.* (1989): A Guide book to Fortran on Supercomputers, *Academic Press*, San Diego, CA.

[68] Quealy, A., *et al.* (1993): Portable programming on parallel/networked computers using the Application Portable Parallel Library (APPL), *Technical Report NASA/TM-106238*, NASA Lewis Research Center, Cleveland, OH.

[69] Fagg, G., *et al.* (1996): PVMPI: An Integration of PVM and MPI Systems, *Calculateurs Paralleles*, Volume 8, Number 2, 151-166.

[70] Geijn, Robert A. van de, Watts, J. (1995): SUMMA: Scalable Universal Matrix Multiplication Algorithm. *LAPACK Working Note 99*, Technical Report CS-95-286, University of Tennessee.

[71] Kumar, V. *et al.* (1994): Introduction to Parallel Computing. *The Benjamin Cummings Publishing Company, Inc.*, Redwood City, CA.

[72] Golub, G. H., Loan, C. V. Van (1994): Matrix Computations. *The John Hopkins University Press*, Baltimore, MD, Third Edition.

[73] Cannon, L. E. (1969): A Cellular Computer to Implement the Kalman Filter Algorithm. *Ph.D. Thesis*, Montana State University.

[74] Fox, G. C. *et al.* (1988): Solving Problems on Concurrent Processors, Vol. 1, Prentice Hall, Englewood Cliffs, N. J.

[75] Fox, G. C. *et al.* (1987): Matrix algorithm on a hypercube I: Matrix Multiplication, *Parallel Computing*, 3, 17-31.

[76] Choi, J., Dongarra, J. J., *et al.* (1994): PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers. *Concurrency: Practice and Experience*, 6, 543-570.

[77] Choi, J. (1997): A New Parallel Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers,
http://www.netlib.org/lapack/lawns/lawn129.ps

[78] Lewis, R. I. (1984): An Introduction to Martensen' method for plane two-dimensional potential flows. Department of Mechanical Engineering, University of Newcastle upon Tyne, Internal Report No. Tb. 63.

[79] Ashby, D. L. *et al.* (1990): Development and Validation of an Advanced Low-Order Panel Method. NASA TM 102851.

[80] Magnus, A. E. *et al.* (1980): PAN AIR - A computer program for predicting subsonic or supersonic linear potential flows about arbitrary Configurations using a higher-order Panel Method, Vol 1, NASA CR 3251.

[81] Maskew, B. (1981): Prediction of sub-sonic aerodynamic characteristics: a case for lower order Panel Methods, AIAA Paper No. 81-0252.

[82] Coopersmith, R. M. *et al.* (1981): Quadrilateral element Panel Method (QUAD-PAN), Lockheed-California LR 29671.

[83] Lewis, R. I. (1991): Vortex Element Methods for fluid dynamic analysis of engineering systems, Cambridge University Press, First edition.

[84] Erickson, L. L. (1990): Panel Methods - An Introduction, NASA TP-2995.

[85] Abbott, I. H. *et al.* (1959): Theory of Wing Sections. Dover Publications, Inc., New York.

[86] Hoare, C. A. R., (1978): Communicating Sequential Processes, *Communications of the ACM*, 21(8), 666-677.

[87] Harris, T. J., (1994): A Survey of PRAM Simulation Techniques, *ACM Computing Surveys*, Vol 26, No. 2 (June), 187-206.

[88] Pacheco, P., (1997): Parallel Programming with MPI, *Morgan Kaufmann*, San Francisco, California.

[89] Wilkinson, B., *et al.* (1999): Parallel Programming - Techniques and Applications, Using Networked Workstations and Parallel Computers, *Prentice-Hall, Inc.*.

[90] Skillicorn, D., *et al.* (1997): Questions and Answers about BSP, *Scientific Programming*, Vol. 6, No. 3, 249-274.

[91] Al-saqabi, K., *et al.* (1997): Dynamic Load Distribution in MIST, *In Proc. of the International Conference on Parallel & Distributed Processing Techniques and Applications (PDPTA '97)*, Las Vegas, Nevada, June 1997.

[92] Sobalvarro, P. G., *et al.* (1998): Dynamic Coscheduling on Workstation Clusters, *Proceedings of the International Parallel Processing Symposium (IPPS' 98)*, March 30 - April 3.

[93] Arpaci-Dusseau, A. C., *et al.* (1997): Extending Proportional-Share Scheduling to a Network of Workstations, *In International Conference on Parallel &*

Distributed Processing Techniques and Applications (PDPTA '97), Las Vegas, Nevada, June 1997.

[94] Menden, J., *et al.* (1996): Providing Properties of PVM applications - a case study with CoCheck, *In Parallel Virtual Machine-EuroPVM '96*, Lecture notes in Computer Science, Vol. 949, D. G. Feitelson and L. Rudolph, editors, Springer-Verlag, 134-141.

[95] Barak, A., *et al.* (1993): The MOSIX distributed operating system - Load balancing for Unix, *Lecture notes in Computer Science*, Springer-Verlag.

[96] Artsy, Y., *et al.* (1989): Designing a process migration facility - the Charlotte experience, *Computer*, 22(9), 47-56, September 1989.

[97] Theimer, M. M., *et al.* (1985): Pre-emptable remote execution facility for the V-System, *In Proceedings of the 10th ACM Symposium on Operating Systems Principles*, 2-12, Washington, December 1-4 1985.

[98] Douglis, F., *et al.* (1987): Process Migration in the Sprite operating system, *In Proceedings of the 7th IEEE International Conference on Distributed Computing Systems*, 18-25, Berlin, West Germany, September 1987.

[99] Milojicic, D. S., *et al.* (1993): Task migration on the top of the Mach microkernel, *In MACH III Symposium Proceedings*, 273-289, Santa Fe, New Mexico, April 19-21 1993.

[100] Michael, J., *et al.* (1998): Condor - A hunter of idle workstations, *In Proceedings of the 8th IEEE International Conference on Distributed Computing Systems*, 104-111, June 1998.

[101] Jeremy, C., *et al.* (1995): MPVM: A Migration Transparent Version of PVM, *Technical Report CSE-95-002*, February 1995.

[102] Clifford, B. N., *et al.* (1994): The Prospero Resource Manager: A scalable framework for processor allocation in distributed systems, Concurrency: Practice and Experience, 6(4):339-355, June 1994.

[103] Konuru, R., *et al.* (1994): A user-level process package for PVM. In 1994 Scalable High Performance Computing Conference, 48-55, IEEE Computer Society Press, May 1994.

[104] Green, T., *et al.* (1993): DQS, a distributed queuing system, *Technical report*, Supercomputer Computations Research Institute, Florida State University, April 1993.

[105] Juan, L., *et al.* (1993): Fail-safe PVM: A portable package for distributed programming with transparent recovery, *Technical Report CMU-CS-93-124*, Carnegie Mellon University, February 1993.

[106] Bequelin, A., *et al.* (1994): Dome: Distributed object migration environment, *Technical Report CMU-CS-94-153*, Carnegie Mellon University, May 1994.

[107] Dikken, L., *et al.* (1994): DynamicPVM: dynamic load balancing on parallel systems, *High-Performance Computing and Networking*, Proceedings 1994, Vol 797, 273-277.

[108] Azavedo, E. F., *et al.* (1997): The Design and Implementation of the Parallel Out-of-core ScaLAPACK LU, QR and Cholesky Factorization Routines, *LAPACK Working Note 118.*

[109] DeSouza-Batista, J. C., *et al.* (1996): A sub-optimal assignment of application tasks onto heterogeneous systems, *Proceedings of the Heterogeneous Computing Workshop '94*, 9-16.

[110] Jacob, J. C., (1996): Task Spreading and Shrinking on a network of workstations with various edge classes, *Proc. 1996 Int. Conf. Par. Proc.*, Part III, 174-181.

"Deep in the fundamental heart of mind and Universe,
there is a reason".
- Slartibartfast.

.1 Appendix I: Matrix Algorithms

.1.1 Basic SUMMA Algorithm

In the SUMMA algorithm, the elements of **C** are given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

where a_{ij}, b_{ij} and c_{ij} denote the (i,j) element of the matrices, respectively. Here the rows of **C** are computed from rows of **A** and columns of **C** are computed from columns of **B**. Hence, $\alpha_i^C = \alpha_i^A$ and $\beta_j^C = \beta_j^B$. Here the notation α^X and β^X are used to denote respectively, the row and column dimensions of the matrix X. The blocks of C_{ij} are formed as:

$$C_{ij} = \overbrace{\left(A_{i0} \,\middle|\, A_{i1} \,\middle|\, \cdots \,\middle|\, A_{i(c-1)} \right)}^{\tilde{A}_i} \left\{ \begin{pmatrix} B_{0j} \\ \hline B_{1j} \\ \hline \vdots \\ \hline B_{(r-1)j} \end{pmatrix} \right\} \tilde{B}^j$$

It can be observed that $\tilde{A}_i$ is entirely assigned to node row i, while $\tilde{B}_j$ is entirely assigned to node column j, hence $\tilde{A}_i$ and $\tilde{B}_j$ can be written as:

$$\tilde{A}_i = \left(\tilde{a}_i^0 \,\middle|\, \tilde{a}_i^1 \,\middle|\, \cdots \,\middle|\, \tilde{a}_i^{n-1} \right) \text{ and } \tilde{B}^j = \left(\begin{array}{c} \tilde{b}_0^{j^T} \\ \hline \tilde{b}_1^{j^T} \\ \hline \vdots \\ \hline \tilde{b}_{n-1}^{j^T} \end{array} \right)$$

and thus the product matrix can be computed using:

$$C_{ij} = \sum_{k=0}^{n-1} \tilde{a}_i^k \tilde{b}_k^{j^T}$$

Hence the SUMMA algorithm is a sequence of rank-1 updates ($A \leftarrow A + \alpha x y^T$). Using this scheme of data-decomposition and matrix-matrix multiplication, the modification that was adopted for this thesis concerns the local matrix-matrix multiplication at each computing node. If the partitioned matrices are square matrices then a block-matrix multiplication with loop unrolling (to a depth of 4) is used and otherwise a regular matrix multiplication is used. The idea being that the *block algorithms* have the potential for larger amounts of reusable data and improve the computation performance by affecting the data flow between the functional units and the lower levels of memory. The algorithm for this local n-by-n matrix multiplication ($C = A \times B$) with N blocks (block size $= n/N$) is given as:

for $i = 1$ **to** N
  for $j = 1$ **to** N
    load $C^{i,j}$ into fast memory (cache)
    for $k = 1$ **to** N, steps of 4 (loop unroll)
      load $A^{i,k}$ into fast memory (cache)
      load $B^{k,j}$ into fast memory (cache)
      $C^{i,j} = C^{i,j} + A^{i,k} \cdot B^{k,j}$
      $C^{i,j+1} = C^{i,j+1} + A^{i,k} \cdot B^{k,j+1}$
      $C^{i+1,j} = C^{i+1,j} + A^{i+1,k} \cdot B^{k,j}$
      $C^{i+1,j+1} = C^{i+1,j+1} + A^{i+1,k} \cdot B^{k,j+1}$

```
            end for
            write C^{i,j} back to slow (main) memory
        end for
    end for
```

This algorithm uses $O(n^2)$ memory references (communications) and $O(n^3)$ flops yielding $Q = \frac{\#memref}{\#flops} = \sqrt{fast\ memory\ size/3}$. Thus, the larger the *fast memory size*, the greater Q and the better the memory utilization.

.1.2 Jacobi Iteration Technique

For a linear system given by, $[A]\{x\} = \{b\}$ where the matrix $[A]$ is diagonally dominant[6], the values of $\{x\}$ can be written as:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

or if $[A]$ is written as $[A] = [L] + [D] + [U]$, where $[L]$, $[U]$, $[D]$ are respectively strictly lower, upper and diagonal components of $[A]$. Then the Jacobi iteration can be written as:

$$x^{(k+1)} = -D^{-1}(L+U)\,x^{(k)} + D^{-1}b$$

The condition that $[A]$ be diagonally dominant is a sufficient condition and not a necessary condition for convergence. The Jacobi iterations require a test of convergence of the successive iterates. Two commonly used tests (with ϵ denoting the error tolerance) are

$$\|\mathbf{x^{(k+1)}} - \mathbf{x^{(k)}}\| \leq \epsilon$$

$$\|\mathbf{x^{(k+1)}} - \mathbf{x^{(k)}}\| \leq \epsilon \|\mathbf{x^{(k)}}\|$$

[6]That is, $\sum_{j \neq i} |a_{ij}| < |a_{ii}|\ \forall i \in [1, n]$. A matrix may be made diagonally dominant by permuting its rows and columns, if it is not already diagonally dominant.

Other termination techniques are: the vector termination technique suggested by Pacheco [88]:

$$\sqrt{\sum_{i=0}^{n-1}\left(x_i^{(k+1)} - x_i^{(k)}\right)^2} < \epsilon$$

and the Bertsekas and Tsitsiklis' (1989) termination condition given by:

$$\| \sum_{j=0}^{n-1} a_{ij} x_i^{(k)} - b_i \| < \epsilon$$

.2 Appendix II: ATM and PVM Architectures

.2.1 ATM Protocols

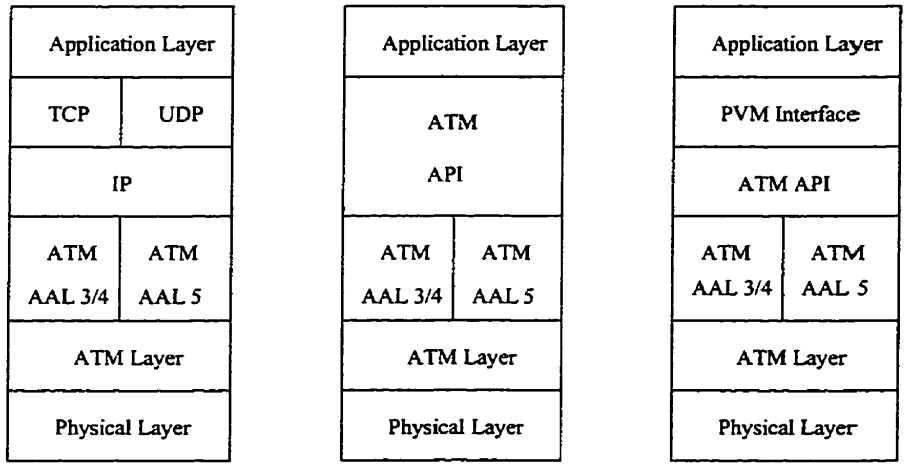


Figure 1: Network Protocol hierarchy for ATM LANs

This section briefly reviews the ATM protocol architecture. The layers of the protocol stack of ATM are shown in Figure 1. The ATM model consists of three layers: the physical layer, the ATM layer and the ATM Adaption layer (AAL). The physical layer encodes or decodes the data into suitable electrical/optical waveforms for transmission and reception on the communication medium. The ATM layer is responsible for cell relay between ATM-layer entities, cell multiplexing and demultiplexing, cell rate decoupling, flow control access, etc. The AAL provides a link between the services required by higher network layers and generic ATM cells used by the ATM layer. ITU-T[7] has defined four service classes based on three parameters: time relation between the source and the destination, constant or variable bit rate, and connection mode. The classes are,

[7]International Telecommunication Union Telecommunication Standard Sector.

Class A. Uses AAL 1 protocol, supports constant bit rate services (such as traditional voice transmission), time relation exists.

Class B. Uses AAL 2 protocol, supports variable bit rate video and audio information, time relation exits.

Class C. Uses either AAL 3/4 or AAL 5 protocol, supports connection oriented data service and signaling, variable bit rate, no time relation.

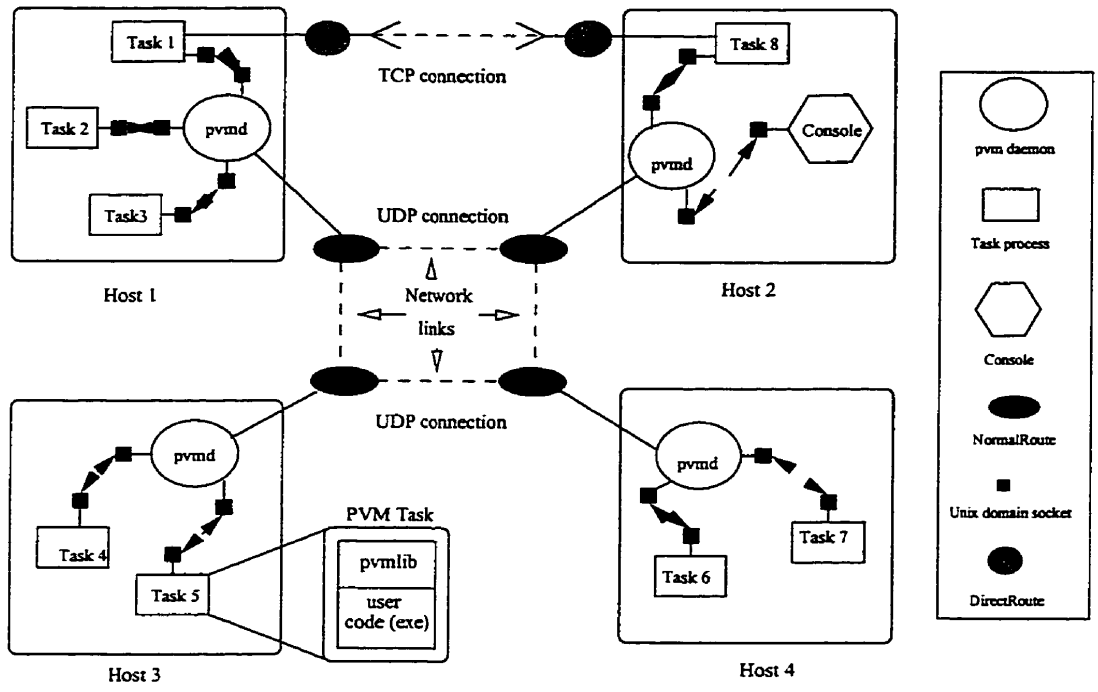
Class D. Uses either AAL 3/4 or AAL 5 protocol, supports connectionless service, variable bit rate, no time relation.

In this thesis work, the AAL 3/4 and AAL 5 protocols were used.

.2.2 The PVM System

The PVM architecture is shown in Figure 2. The PVM daemon process, known as *pvmd*, executes on each host. Each *pvmd* serves as a message router and a controller. It is also responsible for all application component *tasks* executing on their host, exchange of network configuration information and dynamic memory allocation of packets traveling between distributed tasks. Each host usually has a number of tasks and a local *pvmd.*

The communication between tasks in different hosts may occur directly as a *task-task* interaction or as a *task-pvmd-pvmd-task* interaction as illustrated in Figure 2. Direct task-to-task links in PVM are established using the *PvmRouteDirect* option which enables the source task to communicate with the remote task through a TCP based socket connection. The indirect task-pvmd-pvmd-task links are established using the *PvmAllowDirect* option where the source task first communicates with its local *pvmd* through a Unix domain socket. The local *pvmd* then uses a UDP socket to communicate with the remote *pvmd* which finally communicates with its local task through a Unix domain socket (see Figure 2). The indirect routing (which is the default routing) guarantees scalability, since it uses connectionless UDP sockets which, unlike TCP, does not consume file descriptors [50] and hence can communicate



The PVM configuration

Figure 2: The PVM architecture and its communication routes.

with any number of tasks (remote or local). But it is not efficient as it requires 2 Unix Domain socket connections and a UDP connection for messages to reach their destinations.

In PVM a console program is used to perform tasks such as configuring the virtual machine, starting and killing processes, and collecting and checking process status information. PVM supports dynamic reconfigurability by allowing hosts to enter or exit the host pool at any point in the execution of a concurrent application. It also supports the notion of dynamic process groups implying that processes can belong to multiple named groups and at any time during computation the groups can be changed. Although automatic recovery procedures are not built into the PVM system it can withstand some degree of host and network failures since operations may continue when any host, aside from the master host fails.

In version 3.3.4 and higher versions of PVM, it is possible to designate a special task as the resource manager or global scheduler (GS). This resource manager can

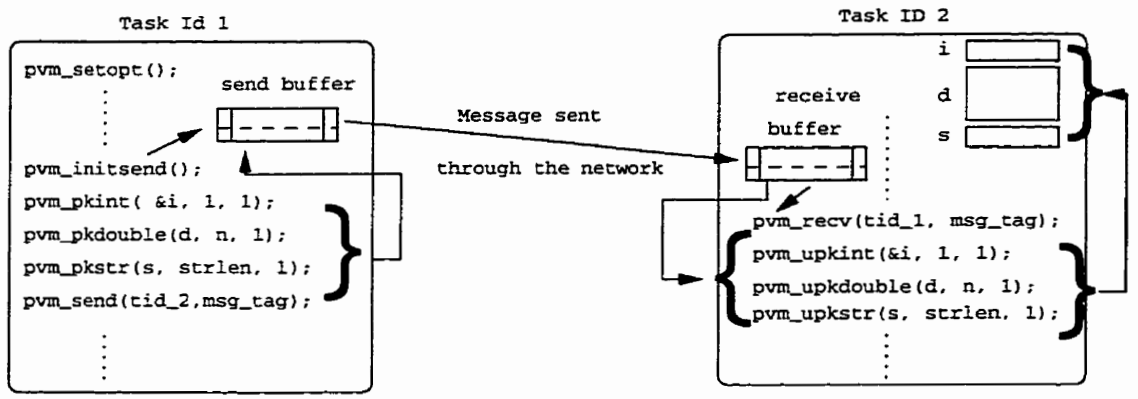


Figure 3: The mechanism of message packing, sending and unpacking in PVM.

make policies on task-to-processor allocation for efficient scheduling or load balancing of multiple parallel applications. Using such a global scheduler makes it convenient to experiment with different scheduling policies to improve system performance.

PVM programs are usually organized in a master-slave model where the single master task is first executed and all others are spawned from this master task. Execution of one or more identical processes (slave tasks) are started using the *pvm_spawn()* routine. The task IDs of the slave processes started on the specified hosts are returned in an array specified in *pvm_spawn()*. Message passing in PVM is done using the routines *pvm_send()* and *pvm_recv()*. Calls to these routines are embedded into the user code prior to compilation. All PVM send routines are nonblocking (asynchronous) while the receive routines can be either blocking (synchronous, *pvm_recv()*) or nonblocking (*pvm_nrecv()*). Before a send operation is done the default send buffer is initialized by the source process with *pvm_initsend()*. If the data to be sent is composed of various data-types then the data has to be packed into the PVM send buffer prior to sending the data. The basic packing routines are *pvm_pkint()* for integers, *pvm_pkdouble()* for doubles, etc. The other routines that are used for a group of processes include *pvm_bcast()*, *pvm_scatter()*, and *pvm_reduce()* for broadcast, scatter and reduce operations respectively. The PVM message passing mechanism is illustrated in Figure 3.

.3 Appendix III: Scheduling in PVM

.3.1 An Experimental PVM Scheduling Model

The abstract goal of scheduling PVM tasks can be stated as follows:

For a given a set of tasks comprising a computation and a set of computers on which these tasks can be executed, the objective is to map the tasks to processors based on the processor's load information at runtime so as to minimize the overall execution time.

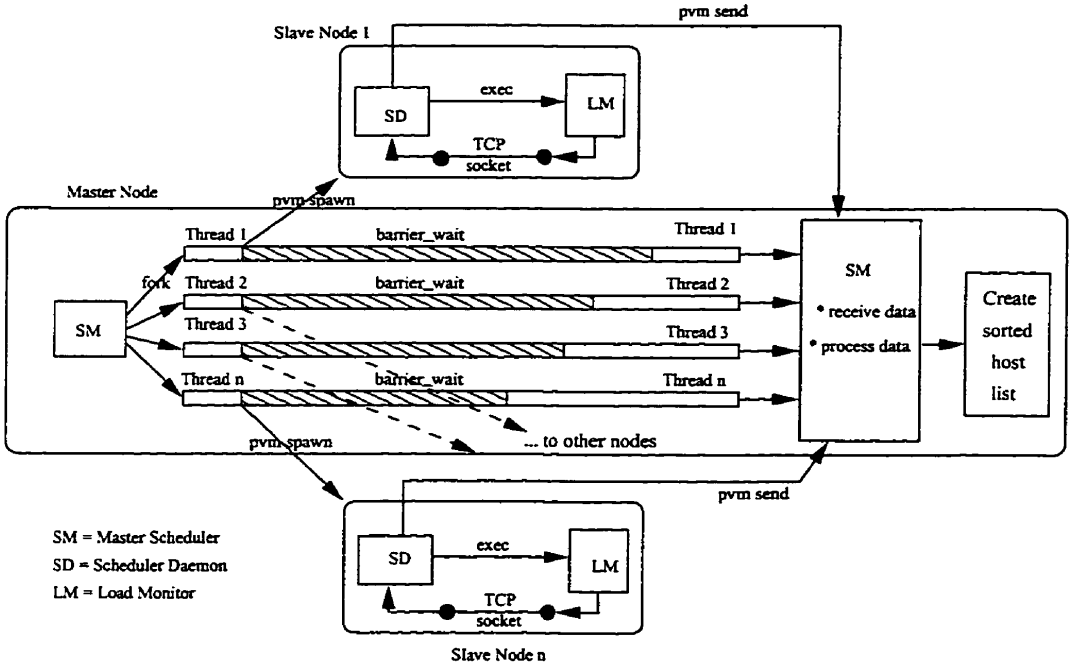


Figure 4: Use of multithreading in Scheduler master to determine load statistics.

In the PVM system this can be achieved by using the PVM resource manager (*pvm_reg_rm()* function) to do the scheduling of slave hosts based on the current load information of the nodes. Using PVM's resource manager with a customized scheduling policy[8] involves writing an entire scheduler similar to that of PVM's default

[8]PVM version 3.4 has given example routines like *tasker* and *hoster* for customized scheduling.

round-robin scheduler. This would be a significant undertaking. An easy and less involved alternative would be to *explicitly* spawn PVM tasks based on the nodes' current load information rather than doing a default cyclic allocation. This is the central idea behind the development of the experimental PVM task scheduler.

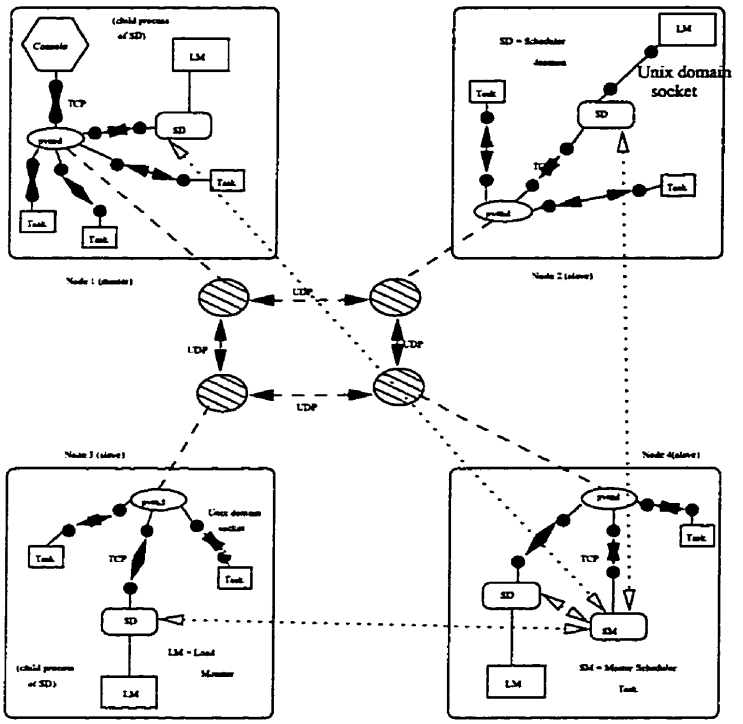


Figure 5: Scheduling in the PVM configuration. Scheduler tasks and subtasks run along with the actual problem tasks on each host.

The basic scheduling model consists of a master scheduler (SM), a scheduler slave or daemon (SD) and a load monitor (LM) running on each node as shown in Figure 5. The master scheduler forks a number of threads equal to the number of available nodes in the PVM system. Each thread in turn spawns a PVM task (SD task) in each of the nodes and executes a barrier. The SD in each node, after a specified polling interval, invokes the load monitor (LM) to collect (via a socket connection) the processor load statistics[9] every 30 seconds, parses them and sends the information to the master

[9]The Unix *top* utility was modified and run by the LM to determine the relevant processor load data including CPU idling percentage, number of processes in the run queue, etc.

Table 1: Machines (ic11, cider, ic18, ouzo) sorted in descending order of CPU load by the load monitor.

Machine	Time	idle-cpu	user-cpu	load-avg1	load-avg2
ic11-atm	[11:25:12]	90.90%	1.80%	0.00	1.00
cider-atm	[11:25:12]	0.00%	91.80%	1.06	0.00
ic18-atm	[11:25:12]	0.00%	91.80%	1.06	1.04
ouzo-atm	[11:25:12]	0.00%	97.10%	1.00	1.04

Machine	Time	idle-cpu	user-cpu	load-avg1	load-avg2
ic18-atm	[11:25:19]	91.70%	0.90%	0.02	0.01
ic11-atm	[11:25:19]	91.70%	0.90%	0.02	0.01
cider-atm	[11:25:19]	91.70%	0.90%	0.02	1.00
ouzo-atm	[11:25:19]	0.00%	98.00%	1.00	0.01

Machine	Time	idle-cpu	user-cpu	load-avg1	load-avg2
ic11-atm	[11:25:26]	90.80%	0.00%	0.01	1.01
cider-atm	[11:25:26]	0.00%	92.20%	1.09	1.05
ic18-atm	[11:25:26]	0.00%	94.00%	1.11	1.04
ouzo-atm	[11:25:25]	0.00%	97.10%	1.02	0.01

before it suspends itself for a specified sleep interval. The master scheduler awakes the threads to asynchronously receive the load statistics from the slave tasks and uses a sorting algorithm to order the nodes in a sorted host list. The PVM tasks comprising the actual computation are then self-scheduled (spawned) based on this sorted host list, i.e. it maps new tasks to nodes, favoring the idle or least loaded processor in the host list.

A snap shot of the sorted nodes and their user and system load statistics which were obtained from the load monitors in each node is shown in Table 1.

.3.2 Related Work on PVM Scheduling and Fault-tolerance

The resource manager interface of PVM is used to separate its scheduling mechanisms from its scheduling policy. This interface can be conveniently used to test various scheduling algorithms for use with the PVM system. The job of a resource manager in PVM is similar to that of a global scheduler. It embodies decision making policies including such things as task-processor allocation, task spawning and termination, etc. for efficient scheduling of multiple parallel applications.

Task or process migration implementations, in general, can be at the user-level or at the operating system-level, the latter involves modification of the OS-kernel. Global scheduling systems that have been implemented at the user-level include Condor [100], MPVM [101] (Migratable PVM), UPVM [103], PRM [102], DQS [104], Fail-Save PVM [105], DynamicPVM [107], and DOME [106]. Notable examples of system-level implementations are MOSIX [95], Charlotte [96], V [97], Sprite [98], and Mach [99].

The most popular of the user-level process migration implementations is Condor [100] which runs on a workstation cluster to harness wasted CPU cycles. Condor's scheduling mechanism have been integrated with PVM (through its CARMI interface [100]) to provide support for process suspension/resumption and process migration of sequential jobs (via the *remote system call* facility) but presently PVM/CARMI does not support PVM task migration. Transparent, asynchronous task migration between homogeneous nodes is supported by MPVM [101] which has portability and source code compatibility with PVM. To achieve migration, Condor uses a checkpoint/roll-back mechanism (offered by its CoCheck [94] interface) which ensures its fault-tolerance. MPVM is not fault-tolerant but achieves better task migration speed by transferring the process state of its migrating tasks directly through the network (using a TCP socket connection). Transparent task migration and restart is also implemented in Fail-Safe PVM [105] which uses synchronous check-pointing and explicit message flushing at checkpoint time to independently checkpoint applications.

The Prospero Resource Manager (PRM) [102] is another software environment

that supports PVM applications through an interface library which translates PVM API calls to their library's API. It's task migration scheme is similar to Condor's and is based on check-pointing involving the creation of core dumps and checkpoint files. UPVM [103] can also transparently migrate PVM tasks but unlike MPVM, tasks in UPVM are implemented as User-Level Processes (ULPs) which are thread-like entities. Unlike threads, however, each ULP has its own data, stack and heap space. Though ULPs have the potential for achieving faster migration speeds and better load balance they are restricted to SPMD programs only. Similar to MPVM is DynamicPVM [107] which supports process migration by adding checkpoint-restart primitives to the PVM runtime support system. The Distributed Queuing System (DQS) [104] also supports PVM jobs in addition to supporting load balancing of batch applications across a heterogeneous network. It creates a new set of PVM daemons at the start of a PVM job and terminates the daemons at job completion. This overhead at PVM system initiation, however, adds to the cost of running the job. DOME [106] is another computing environment that supports application-level check-pointing (to handle heterogeneity) and dynamic load balancing. It is implemented as a library of C++ classes and uses PVM for its process control and communication. Most of the above systems use various techniques for static and dynamic scheduling in such distributed-memory environments. The most common scheduling policies are based on either gang scheduling [91], dynamic co-scheduling [92] or proportional-share scheduling [93].

Support for adaptive execution of PVM tasks on a shared workstation cluster can also be provided by systems implemented at the operating system level. Although such systems can handle most of the problems that are associated with user-level implementations (e.g., total migration transparency, efficient task check-pointing, and migration) they are not portable and are hardware/OS dependent compared to the software systems implemented at the user-level.

.4 Appendix IV: Flow Simulation Problem

This appendix describes the theory behind the surface vorticity boundary integral method and describes the detailed computational scheme adopted in the solution of the multi-body airfoil problem.

.4.1 Surface Vorticity Analysis: Mathematical Formulation

The flow past a two-dimensional body in the (x, y) plane (see Figure 6), subjected to a uniform stream W_{inf} inclined at an angle α_{inf} to the x axis is considered. The potential flow past this body can be modeled by replacing the body surface with a vortex sheet $\gamma(s)$ of appropriate strength (Figure 6). The potential flow assumption is valid for such aerodynamic calculations because of the fact that the viscous effects are small in the flow-field and the flow-field is sub-sonic everywhere. The surface vortex sheets provide the necessary discontinuity in velocity from zero on the body to the potential flow velocity at the edge of the flow domain. These vorticity sheets created under the influence of the surface pressure gradient freely convect and diffuse downstream. Hence the surface vorticity model is a direct simulation of an ideal inviscid flow.

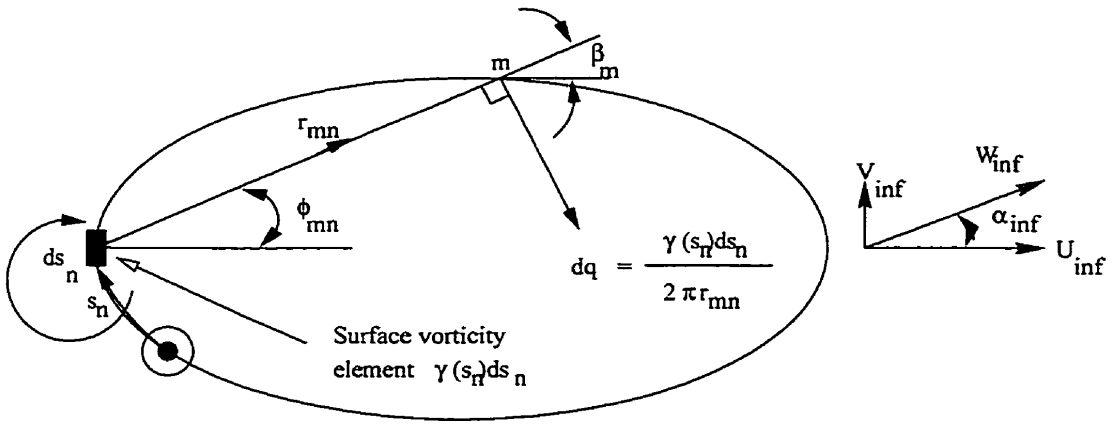


Figure 6: Surface vorticity model for a 2-D geometry.

The Biot-Savart law can be used to represent the velocity dq_{mn} induced at s_m (as shown in Figure 6) due to a small rectilinear vorticity element $\gamma(s)ds_n$ located at s_n on the body, by the following equation:

$$dq_{mn} = \frac{\gamma(s_n)ds_n}{2\pi r_{mn}}$$

The (x, y) components of dq_{mn} can be expressed as a function of its geometric co-ordinates as:

$$dU_{mn} = \frac{\gamma(s_n)ds_n}{2\pi r_{mn}} \sin\phi_{mn} = \left(\frac{y_m - y_n}{2\pi r_{mn}^2}\right)\gamma(s_n)ds_n$$

$$dV_{mn} = \frac{\gamma(s_n)ds_n}{2\pi r_{mn}} \cos\phi_{mn} = -\left(\frac{x_m - x_n}{2\pi r_{mn}^2}\right)\gamma(s_n)ds_n$$

Using the profile slope β_m, the velocity dq_{mn} resolved in the direction parallel to the body surface s_m is given by,

$$dv_{smn} = \frac{1}{2}\left\{\frac{(y_m - y_n)\cos\beta_m - (x_m - x_n)\sin\beta_m}{(x_m - x_n)^2 + (y_m - y_n)^2}\right\}\gamma(s_n)ds_n = k(s_m, s_n)\gamma(s_n)ds_n$$

where $k(s_m, s_n)$ is the coupling co-efficient matrix. For two-dimensional flows the Dirichlet boundary condition of zero velocity on (and parallel to) the body surface at s_m can be expressed as:

$$\oint k(s_m, s_n)\,\gamma(s_n)\,ds_n - \frac{1}{2}\gamma(s_m) + (W_\infty \cos\alpha_\infty)\cos\beta_m + (W_\infty \sin\alpha_\infty)\sin\beta_m = 0 \tag{.4.1}$$

where the last term is the component of W_∞ resolved parallel to s_m. Equation .4.1 is a *Fredholm integral equation* of the *second kind* and represents the classical *Martensen's boundary integral equation* for two-dimensional flows.

The numerical modeling of equation .4.1 is done by selecting a finite number M of *pivotal points* representative of the surface (see Figure 3.4). When the body surface is so discretized by M line elements of length $\triangle s_n$ and the equations for the M pivotal

points with M unknown values of surface vorticity, $\gamma(s_i), i = 1, M$ can be expressed as a linear system of the form:

$$\begin{aligned} \sum_{n=1}^{M} K(s_m, s_n)\gamma(s_n) &= W(s_m) \\ W(s_m) &= -U_\infty \cos\beta_m - V_\infty \sin\beta_m \end{aligned} \tag{.4.2}$$

where U_∞ and V_∞ are components of W_∞ parallel to the x and y axes and $K(s_m, s_n)$ are the coupling co-efficients linking elements m and n, which are given by:

$$K(s_m, s_n) = k(s_m, s_n)\triangle s_n$$

In equation .4.2 the term $\frac{1}{2}\gamma(s_m)$ from equation .4.1 is implicitly absorbed in $K(s_m, s_n)$. It can be observed that $K(s_m, s_n)$ is finite but is indeterminate when $m = n$. Hence, for the case when $m = n$, $K(s_m, s_n)$ is given by:

$$K(s_m, s_n) = -\frac{1}{2} + K'_{mm}$$

where K'_{mm} is the self-inducing coupling co-efficient given by

$$K'_{mm} = \frac{\triangle s_m}{2\pi} \lim_{s_m \to s_n} \left\{ \frac{(y_m - y_n)\cos\beta_m - (x_m - x_n)\sin\beta_m}{(x_m - x_n)^2 + (y_m - y_n)^2} \right\}$$

For linear elements K'_{mm} is assumed to be zero. The expression for the coupling co-efficients linking the panel m of body p with the panel n of body q in a P body assembly is given by:

$$\sum_{q=1}^{P} \sum_{n=1}^{M_q} K_{mn}^{pq}(s_{qn}) = -U_\infty \cos\beta_{pm} - V_\infty \sin\beta_{pm}$$

where $p \in [1, P]$ and $m \in [1, M_p]$. The left hand side of the above equation describes the contributions from all panels, $n \in [1, M_q]$ on each of the bodies, $(q \in [1, P])$. Each of the P bodies may have a different number of panels M_q that are required to represent its geometry.

.4.2 Computational Scheme

This section describes the computational scheme and the steps involved in the solution of the system of linear algebraic equations obtained by the above method for multi-body elements.

Step 1. Input data

The $M+1$ geometric co-ordinates (X_i, Y_i) as shown in Figure 3.4 are specified clockwise around the profile starting from the leading edge such that point 1 and point $M+1$ are coincident points, ensuring the closure of the profile. For the experiments in this thesis, the NACA0012 airfoil data was taken from Abbott, I. H., *et al.* [85]. This data generates a crude airfoil profile which needs interpolation for finer discretization. The X_i co-ordinates of the NACA0012 airfoil are hence interpolated using a cosine spacing function given by:

$$\frac{x_i}{c} = \frac{1}{2}\left[1 - \cos\left\{\frac{(i-1)\pi}{(N-1)}\right\}\right] \quad i \in [1, N]$$

The idea behind using this type of distribution is to equally space the panel node points as shown in Figure 7 so that the panels are of equal length. The corresponding ordinates (Y_i) of the X_i values so determined are found by using a cubic spline interpolation based on the original profile data.

The spline functions of *MATLAB*[10] were used to generate the profile data. The entire interpolation was done for the top section of the airfoil. A point which should be noted is that at the leading edge a few points from the lower section were also used for interpolation in order to smoothen the

[10]MATLAB is a product of The Math Works, Inc. It is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language.

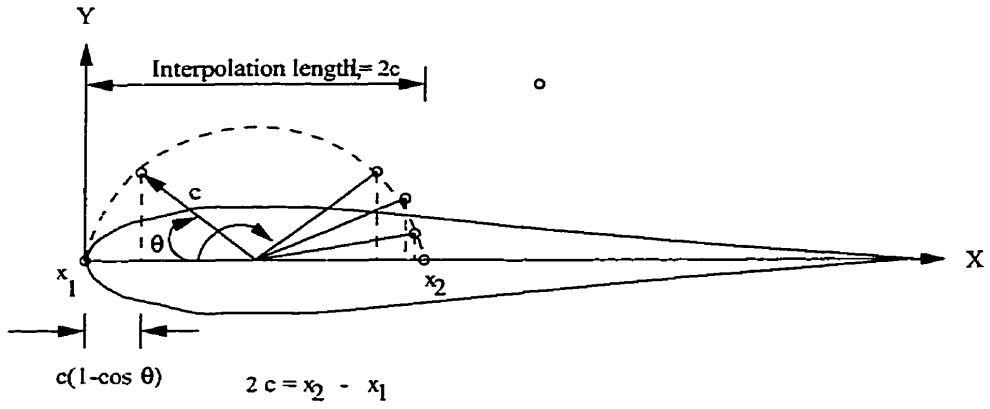


Figure 7: Interpolation of points using the cosine spacing function.

leading edge curvature. For the lower section the co-ordinates were formed by using the same X_i values and negating the Y_i values. The same data was used to generate the other airfoils except in those cases the (X_i, Y_i) co-ordinates were translated by a pre-determined amount.

Step 2. Data preparation

Line segments or panels were constructed by joining the successive data points (X_i, Y_i). Panel i represents the panel (Figure 8) between nodes i and $i+1$. The panel length $\triangle s_i$ is given by,

$$\triangle s_i = \sqrt{[(X_{i+1} - X_i)^2 + (Y_{i+1} - Y_i)^2]}$$

and the profile slopes in finite-difference form can be written as,

$$\cos\beta_i = \frac{X_{i+1} - X_i}{\triangle s_i} \qquad \sin\beta_i = \frac{Y_{i+1} - Y_i}{\triangle s_i}$$

These values can be used to calculate the profile slopes β_i, taking care of the correct geometric quadrant (from the sign of their cosine and sine values). The co-ordinates of the pivotal points (x_i, y_i) that are located at the mid-point of each panels, are calculated as:

$$x_i = \frac{1}{2}(X_{i+1} + X_i) \qquad y_i = \frac{1}{2}(Y_{i+1} + Y_i)$$

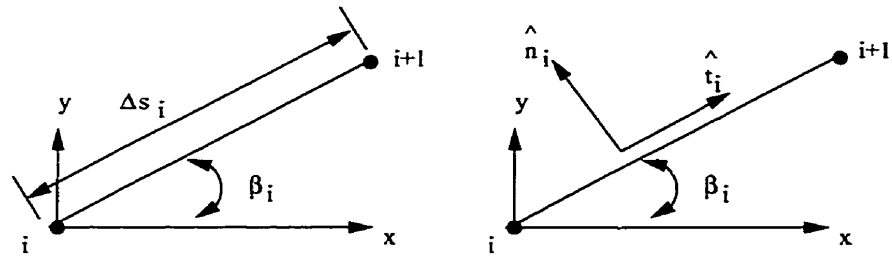


Figure 8: The local co-ordinate system for panel nomenclature.

Step 3. Coupling co-efficient matrix, $K(s_i, s_j)$

The coupling co-efficient matrix for a single isolated airfoil is calculated as follows:

$$K(s_i, s_j) = \begin{cases} \frac{\triangle s_j}{2\pi} \left\{ \frac{(y_i - y_j)\cos\beta_i - (x_i - x_j)\sin\beta_i}{(x_i - x_j)^2 + (y_i - y_j)^2} \right\} & \forall i \neq j \\ -\frac{1}{2} - \frac{1}{8\pi}\left(\beta_{i+1} - \beta_{i-1}\right) & \forall i = j \end{cases}$$

When there are P mutually interacting bodies, the coupling co-efficients representing the induced velocity at the pivotal point i of body p due to pivotal point j of body q are given by the following equation:

$$K^{pq}(s_i^p, s_j^q) = \frac{\triangle s_j^q}{2\pi} \left\{ \frac{\left(y_i^p - y_j^q\right)\cos\beta_i^p - \left(x_i^p - x_j^q\right)\sin\beta_i^p}{\left(x_i^p - x_j^q\right)^2 + \left(y_i^p - y_j^q\right)^2} \right\} \forall p \neq q$$

The global flow equation then takes the form:

$$\begin{bmatrix} \left[K_{ij}^{11}\right] & \cdots & \left[K_{ij}^{1P}\right] \\ \cdots & \cdots & \cdots \\ \left[K_{ij}^{P1}\right] & \cdots & \left[K_{ij}^{PP}\right] \end{bmatrix} \begin{Bmatrix} \{\gamma(s_i^1)\} \\ \vdots \\ \{\gamma(s_i^P)\} \end{Bmatrix} = \begin{Bmatrix} \{W(s_i^1)\} \\ \vdots \\ \{W(s_i^P)\} \end{Bmatrix}$$

Step 4. Right hand sides, W_i

For any body p, the right hand side vector is computed as,

$$W_i = -U_\infty \cos\beta_i^p - V_\infty \sin\beta_i^p$$

Step 5. Back diagonal correction

Kelvin's theorem states that the net circulation, $\triangle\Gamma_i$, around the profile interior induced by a surface vorticity element, $\gamma(s_i)\triangle s_i$, should be zero. When this condition is enforced on the coupling co-efficients K_{ij}, the elements of the column i get modified as follows:

$$K(s_k, s_i) = -\frac{1}{\triangle s_k}\sum_j^M K(s_j, s_i)\triangle s_j \quad \forall j \in [1, k) \cup (k, M]$$

where $k = M + 1 - i$. The back diagonal correction is required to ensure that the net circulation around the profile interior implied by the numerical model is explicitly made zero.

Step 6. Internal circulation correction

The internal circulation correction suggested by Wilkinson needs to be applied for bodies in close proximity. For airfoil systems where the gap between points pi and qj on adjacent bodies p and q are less than the local elemental lengths $\triangle s_i^p$ or $\triangle s_j^q$ numerical errors have to be eliminated. The circulation induced around the perimeter of body p due to a unit vortex placed at the center of element j of body q is given by the expression:

$$\begin{aligned}\triangle\Gamma &= \oint k_{ij}^{pq} ds_i^p \\ &= \frac{1}{\triangle s_j^q}\sum_{i=1}^{M_p} K_{ij}^{pq}\triangle s_i^p\end{aligned}$$

where M_p is the number of panels in body p. The net circulation $\triangle\Gamma$ is forced to zero by replacing the l^{th} coupling coefficient in column j by the value:

$$K_{lj}^{pq} = -\frac{1}{\triangle s_j^q}\sum_{i=1, i\neq l}^{M_p} K_{ij}^{pq}\triangle s_i^p \qquad (.4.3)$$

where l is the element row-index of body p that is in the closest proximity to element j of body q. The row-index l is found by determining the largest absolute value in column j of sub-matrix K^{pq}.

Step 7. Implementation of Wilkinson's Kutta condition

The *Kutta condition* states that the flow must leave the trailing edge smoothly which implies that the static pressures and therefore surface vorticity $\gamma(s)$ at the two trailing edge panels on the upper and lower surface of the model should be equal in magnitude. Mathematically, this condition can be expressed as,

$$\gamma(s_{te}) = -\gamma(s_{te+1}) \tag{.4.4}$$

The negative sign in equation .4.4 is because of the fact that smooth flows leaving the trailing edge $\gamma(s_{te})$ must be clockwise and those leaving $\gamma(s_{te+1})$ should be counter-clockwise. When the *Kutta condition* is applied to the Martensen's equation (the flow equation) by combining the columns te and $te+1$, the order of the equation set becomes $M-1$ (one row and column are eliminated).

Step 8. Matrix inversion

The coupling coefficient matrix $[k_{ij}]$ is a fully dense matrix of finite values and has a non-zero diagonal. Also since the matrix is non-singular it offers no difficulties for matrix inversion. For the sequential flow code an *LU decomposition* technique was used to invert the matrix efficiently while a parallel Jacobi iteration technique was used for the parallel code.

Step 9. Derivation of unit solutions, $U_\infty = 1.0$ and $V_\infty = 1.0$

The vorticity $\gamma(s_i)$ can be expressed in terms of the unit vorticities $\gamma_u(s_i)$ and $\gamma_v(s_i)$ as follows:

$$\begin{aligned}\gamma(s_i) &= U_\infty \gamma_u(s_i) + V_\infty \gamma_v(s_i) \\ &= W_\infty \left(\cos\alpha_\infty \gamma_u(s_i) + \sin\alpha_\infty \gamma_v(s_i)\right) \qquad (.4.5)\end{aligned}$$

Using the *principle of superposition of solutions* equation .4.2 can be written as:

$$\begin{aligned}\sum_{j=1}^{M} K(s_i, s_j)\gamma_u(s_j) &= -\cos\beta_i \\ \sum_{j=1}^{M} K(s_i, s_j)\gamma_v(s_j) &= -\sin\beta_i\end{aligned}$$

Solving for the values of $\gamma_u(s_j)$ and $\gamma_v(s_j)$ and using them in equation .4.5 we can determine $\gamma(s_i)$ for any set of W_∞ and α_∞ values.

Step 10. Calculation of C_p and C_L

The co-efficient of pressure, C_p which is a measure of the surface pressure distribution can be expressed as,

$$\begin{aligned}C_p &= \frac{p - p_\infty}{\frac{1}{2}\rho W_\infty^2} \\ &= 1 - \left\{\frac{\gamma(s)}{W_\infty}\right\}^2\end{aligned}$$

The lift co-efficient on, C_L on an airfoil is defined by:

$$C_L = \frac{L}{\frac{1}{2}\rho W_\infty^2 l}$$

where l is the chord length of the airfoil. Using *Magnus Law* the lift force, L generated on a body in the direction normal to W_∞ is given by:

$$L = \rho W_\infty \Gamma$$

Thus C_L can be expressed as

$$C_L = \frac{2\Gamma}{W_\infty l} = \frac{2}{l}\left\{\cos\alpha_\infty \sum_{j=1}^{M} \gamma_u(s_j)\triangle s_j + \sin\alpha_\infty \sum_{j=1}^{M} \gamma_v(s_j)\triangle s_j\right\}$$

.4.3 Results of the Flow Simulation

In this section the results of the multi-airfoil flow solution problem are presented from a fluid mechanist's point of view. For all the problems a symmetric airfoil, NACA0012 (whose profile data was taken from Abbott, I. H. *et al.* [85]) was used. This data was discretised using cubic spline interpolation and a total of 276 points were used to generate the geometry of one NACA0012 airfoil.

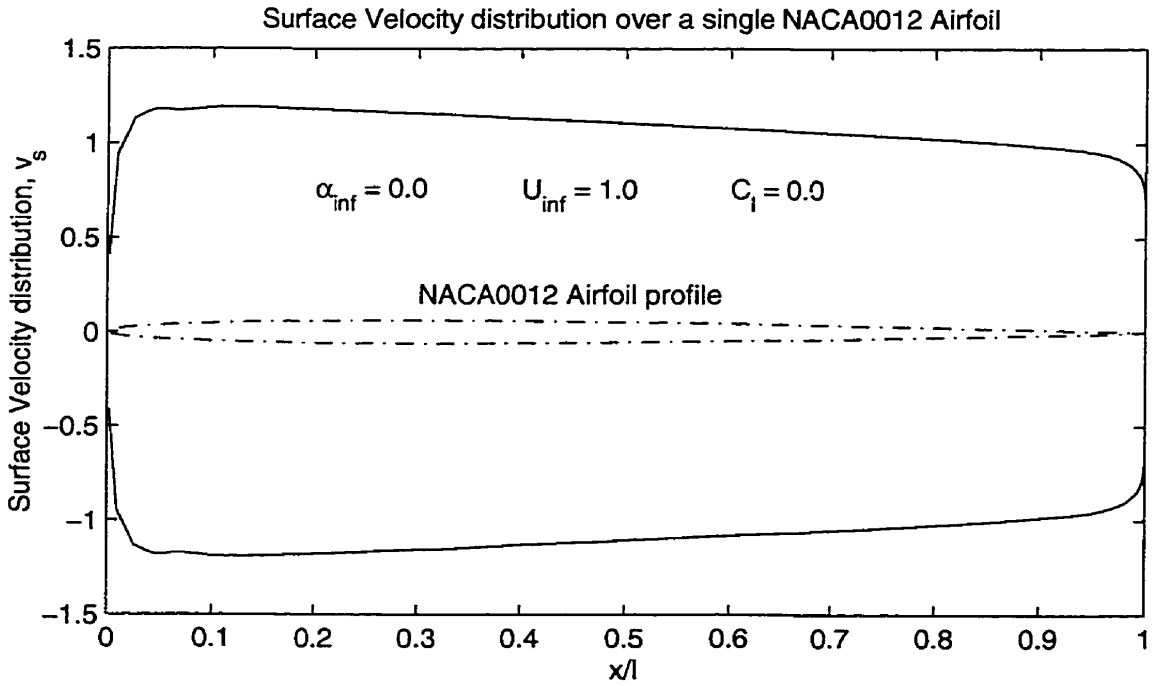


Figure 9: Velocity distributions over a NACA0012 using Surface vorticity method.

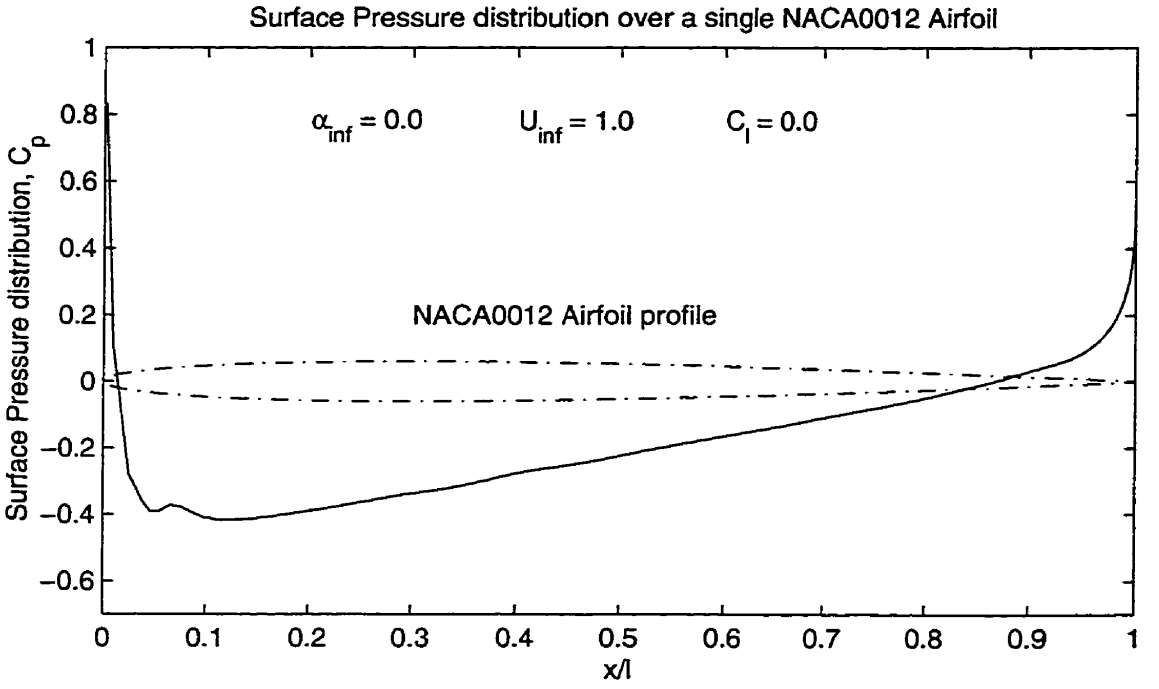


Figure 10: Surface Pressure distributions over a NACA0012 using Surface vorticity method.

Single airfoil system: Code Validation

The distributions of surface velocity, $\gamma(s)$ and surface pressure co-efficients, C_p over a single airfoil at zero angle of incidence were calculated using the sequential single element flow code. These results are presented in Figure 9 and Figure 10 respectively and are found to be in excellent agreement with those of the published literature ([83], pp. 429). A fewer number of points could be used to generate these plots but such a fine discretization (276 points) was used to correct the trailing edge anomaly. It can be clearly observed from both these plots that the $\gamma(s)$ and C_p distributions are very smooth with no kinks at the trailing edge. However, in both these plots a small kink at the leading edge can be observed. This is due to the fact that the initial data used to describe the airfoil had fewer points than the rate at which the curvature was changing.

A similar analysis was done using an angle of incidence, $\alpha = 5^o$ degrees. As

expected the $\gamma(s)$ and C_p distributions (Figure 11 and Figure 12) were in excellent conformation with those of the standard results ([83], pp 432). Smooth trailing edge distribution and a steep slope change at the leading edge can again be observed due to the reasons discussed above.

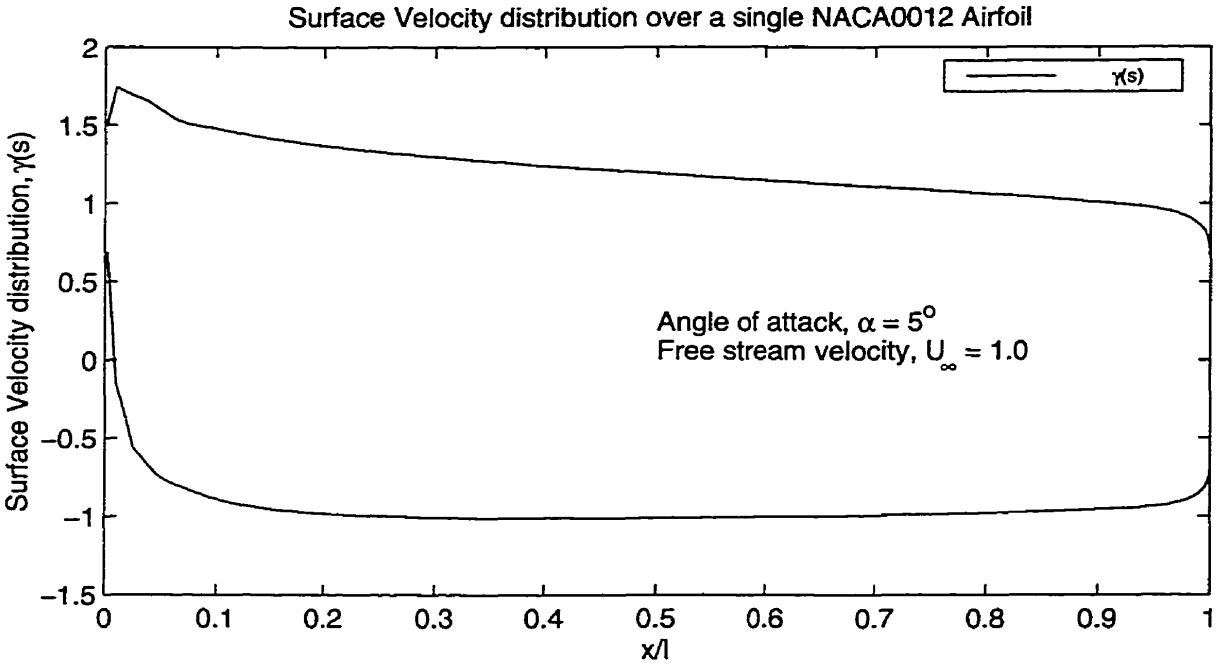


Figure 11: Velocity distributions over a NACA0012 at an angle of attack of 5°.

Multiple airfoil system: Code Validation

For the validation of the multi-element code, the surface pressure distribution, C_p was calculated for two multi-element systems, one with 3 NACA0012 airfoils and the other with 5 NACA012 airfoils. In both cases the airfoils were separated widely by 10 chord lengths both in the vertical and horizontal directions. Theoretically such airfoils would have no interaction with one another and the results of each airfoil would be the same as that of the concerned element analyzed is isolation. This fact is well validated from the plots shown in Figure 13 and Figure 14. A critical observation of these plots shows that for each of the two multi-element systems, the individual

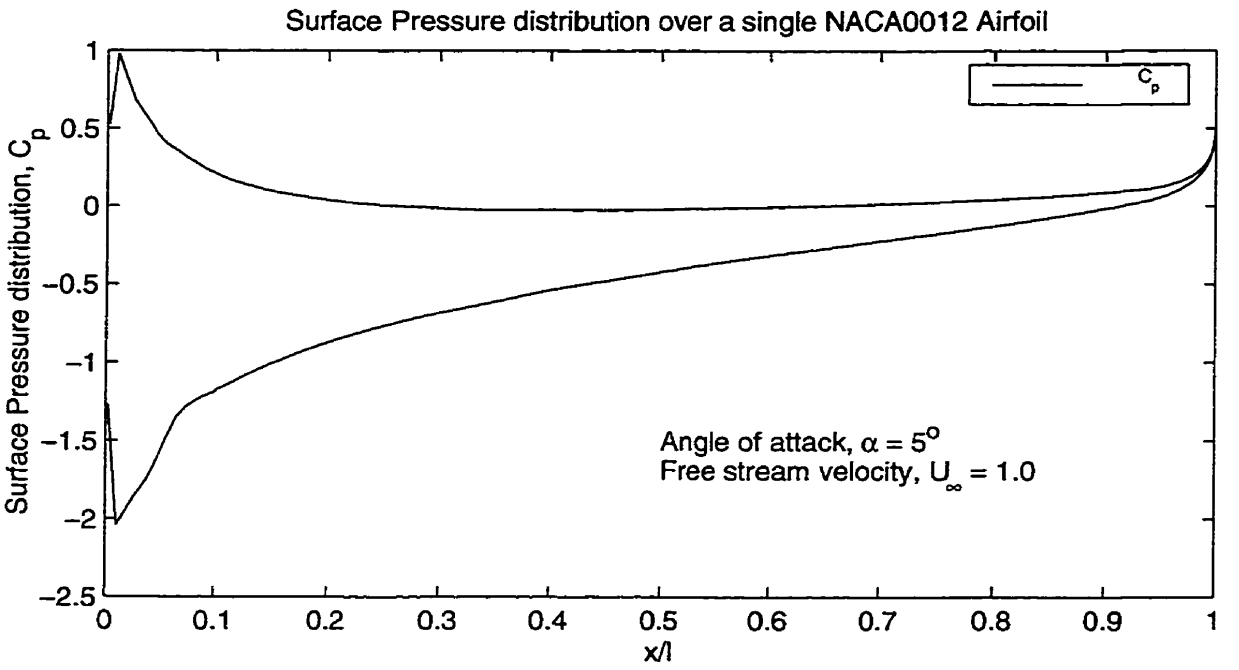


Figure 12: C_p distributions over a NACA0012 at an angle of attack of 5^o.

airfoil C_p distributions are virtually indistinguishable from others including that of the isolated one.

From these plots it can thus be inferred that the multi-element code can be expected to give results that are in excellent agreement with the theory.

The multi-element code was used to analyze the C_p distribution over each airfoil in two different systems, a 3-airfoil system and a 5-airfoil system and for two different angle of incidences, $\alpha = 0^o$ and $\alpha = 5^o$ degrees. The reason for selecting a 3-airfoil system as well as a 5-airfoil system was to show that for any arbitrary number of airfoil system the code gives results that are in conformance with other experimental results. In both the airfoil systems, the airfoils were in close proximity (1 chord length apart vertically) thus forming a cascade of airfoils, one over the other. In this system each airfoil is expected to be affected by the coupling effect of the adjacent airfoils. The C_p distributions of each airfoil are now discussed for the following cases:

1. A 3-element airfoil system in close proximity :

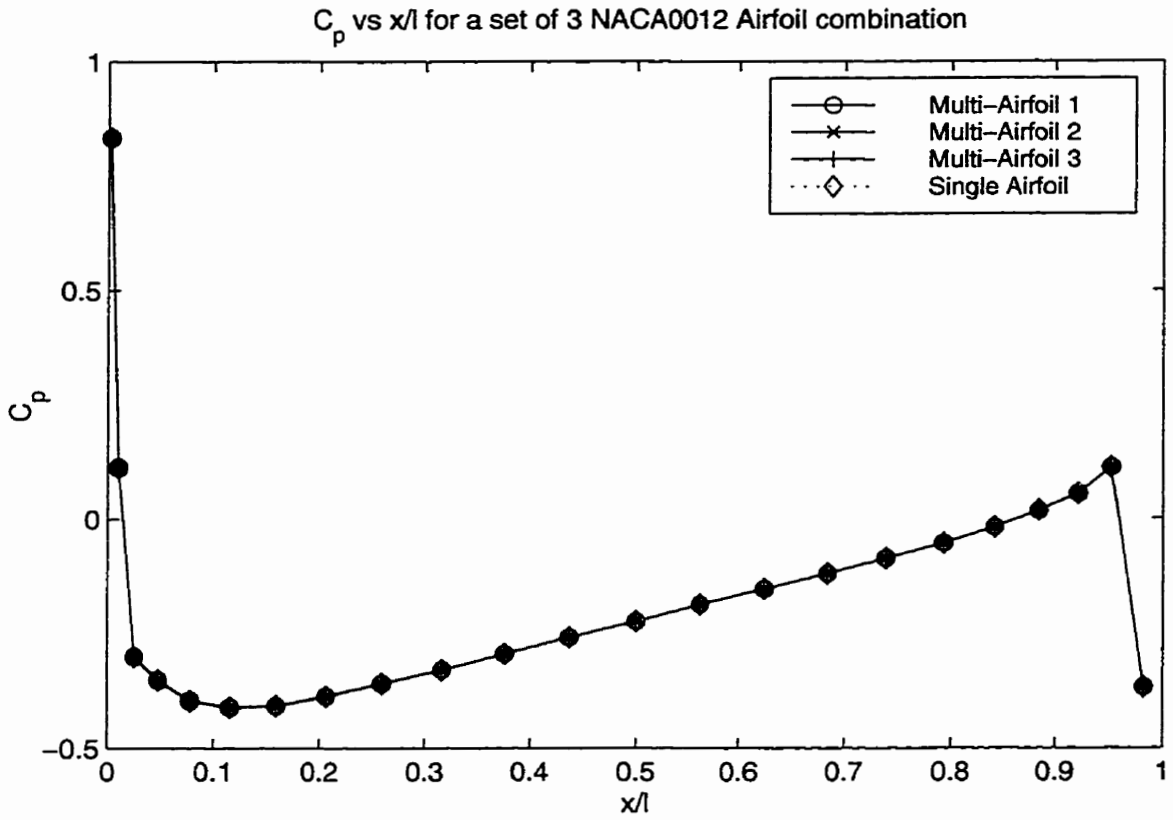


Figure 13: C_p distribution of each of the 3 widely spaced (x = 10 ch, y = 10 ch) airfoils superimposed on the C_p of a single airfoil in isolation.

(a) At $\alpha = 0^o$.

(b) At $\alpha = 5^o$.

2. A 5-element airfoil system in close proximity :

(a) At $\alpha = 0^o$.

(b) At $\alpha = 5^o$.

Case I. A 3-element airfoil system in close proximity: ($U_\infty = 1.0$, $m = 46 \times 3$, $x_i^{(p)} = 0.0$, $y_i^{(p)} = (p-1) \times 1$, $\forall p \in [1, 3]$).

(a) At zero angle of incidence: Referring to the graphs in Figure 15, Figure 16, and Figure 17 it can be observed that the C_p distribution of the upper

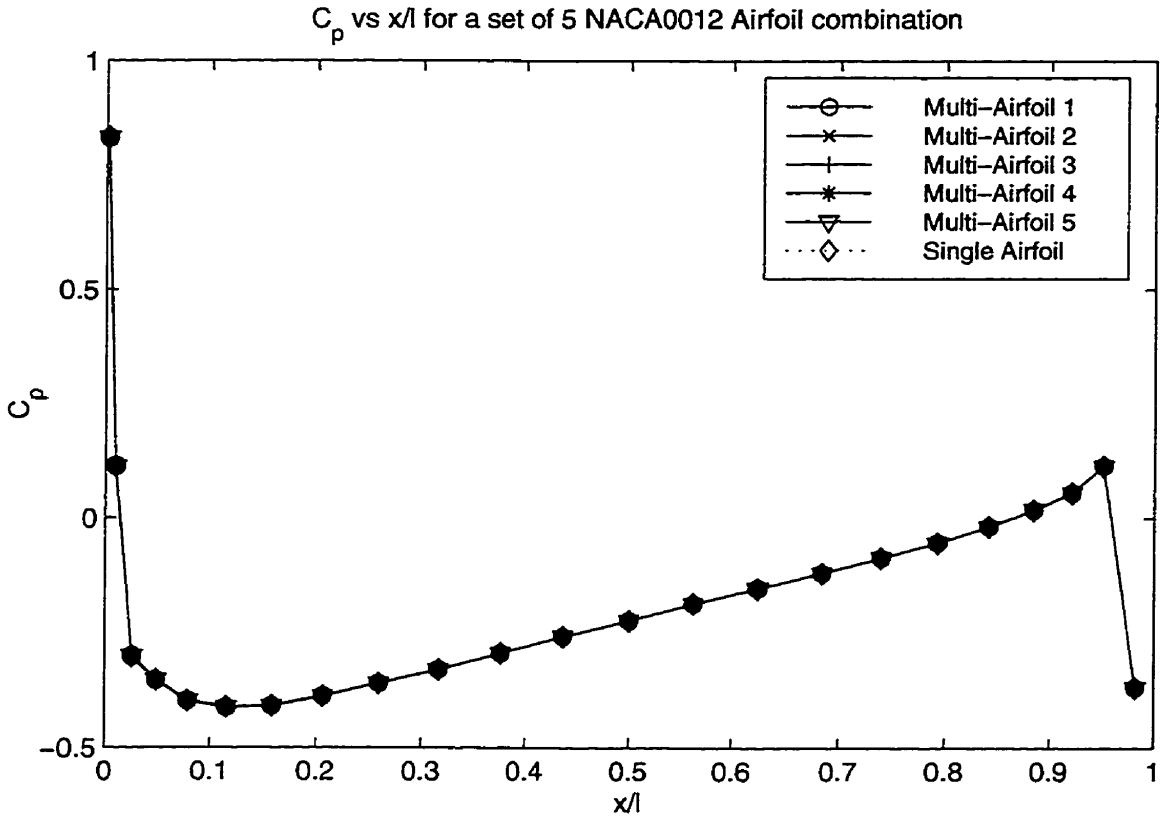


Figure 14: C_p distribution of each of the 5 widely spaced (x = 10 ch, y = 10 ch) airfoils superimposed on the C_p of a single airfoil in isolation.

and lower surface of all the airfoils are not the same as for the isolated single element. This implies that the flow between the airfoils is affected by the proximity of the adjacent airfoils, which is reflected in the numerical calculation by the elements of the cross-diagonal coupling co-efficient matrix being non-zero. It can also be observed that the C_p distribution of the lower surface of each airfoil progressively goes below the isolated airfoil C_p distribution. There seems to be a symmetry in the distributions of the upper surface of the bottom airfoil and the lower surface of the top airfoil, for any two adjacent pairs of airfoils. This is another fact in conformance with the experimental results. However in all 3 graphs it can be observed that at the trailing edge, the C_p distributions take negative values unlike

that of the single airfoil of Figure 10. This is a numerical error due the use of fewer points at the trailing edge. It is worth noting that Figure 10 shows the results with 276 points whereas each of the plots above use only 46 points for each airfoil (that was the minimum number of points necessary to generate a fairly good result).

(b) At an angle of incidence of 5^o: Referring to the graphs in Figure 18, Figure 19, and Figure 20 it can be observed that the lower surface C_p distribution progressively goes below the isolated C_p distribution as we proceed from airfoil 1 to 3. A similar observation can be made for the upper surface C_p distribution, which also goes below the isolated upper surface C_p. However, no symmetry is seen between the C_p distribution of the upper surface of the bottom airfoil and the lower surface of the top airfoil for any two adjacent airfoil pairs, unlike the system at zero angle of incidence. The same numerical error due to discretization at the trailing edge can be observed as in the other cases (compare with Figure 12).

Case II. A 5-element airfoil system in close proximity ($U_\infty = 1.0$, $m = 46 \times 5$, $x_i^{(p)} = 0.0$, $y_i^{(p)} = (p-1) \times 1$, $\forall p \in [1, 5]$).

(a) At zero angle of incidence: Referring to the graphs in Figure 21, Figure 22, Figure 23, Figure 24, and Figure 25, similar results to that of the 3-airfoil system at zero angle of incidence can be observed. Observe that the lower surface C_p distribution progressively goes below the isolated C_p distribution and the symmetry between the upper surface of the bottom airfoil and the lower surface of the top airfoil exists for any two adjacent pairs of airfoils. However the trailing edge anomaly is still observed due to the reasons discussed previously.

(b) At an angle of incidence of 5^o: Referring to the graphs in Figure 26, Figure 27, Figure 28, Figure 29, and Figure 30 it can be seen that the

results for the 5-element airfoil system are not much different from those of the 3-element airfoil system at the same angle of incidence. As the airfoils are displaced further away from the first one, similar observations like the gradual shift in the lower and the upper surface C_p distributions compared to that of the single airfoil in isolation can be made from these graphs. The same numerical error due to discretization can be observed for these graphs. Finally it can be inferred that the results of the 3-airfoil system and the 5-airfoil system are not drastically different.

Summary of Flow Results:

Based on the observations from the above graphs the following conclusions can be made:

- The single-element code is valid for both zero and any angle of incidence.
- The multi-element code is valid for both zero and any angle of incidence.
- The C_p and $\gamma(s)$ distributions for various combination of airfoils (in close or wide spacing and with different angle of incidences) are in conformation with the theoretical results as well as those available in the published literature.

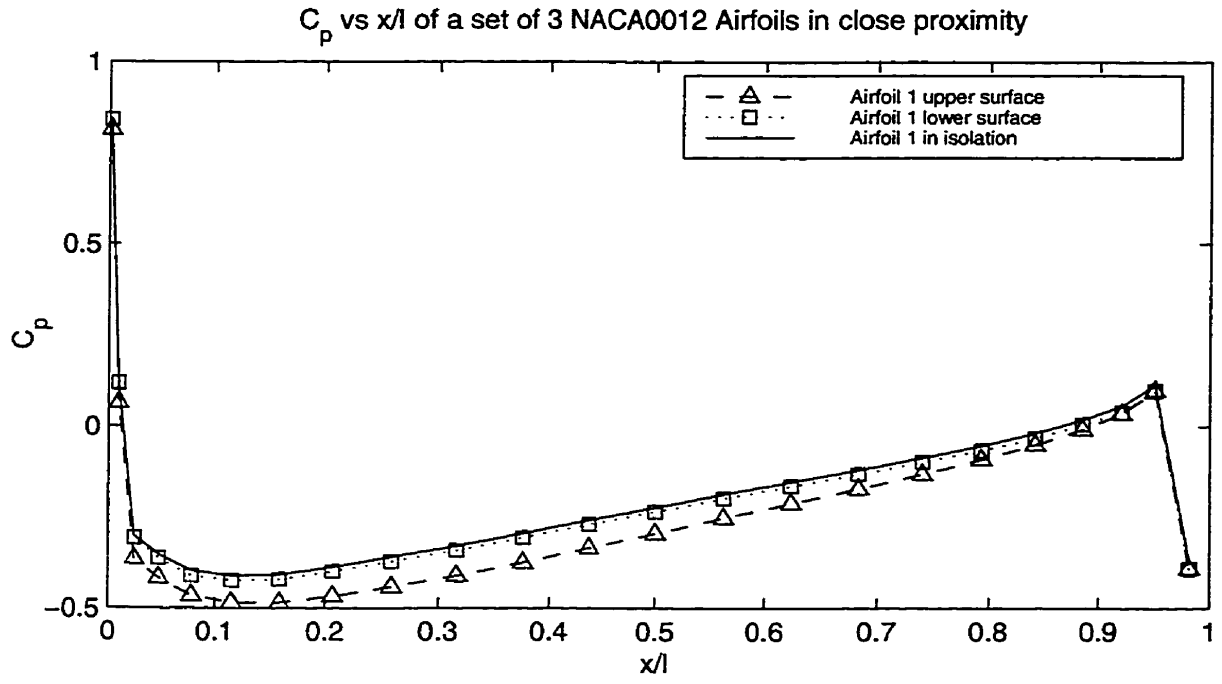


Figure 15: (a) C_p distribution of Airfoil 1 in a set of 3 airfoils in close proximity (x = 0, y = 1 chord length, $\alpha = 0$).

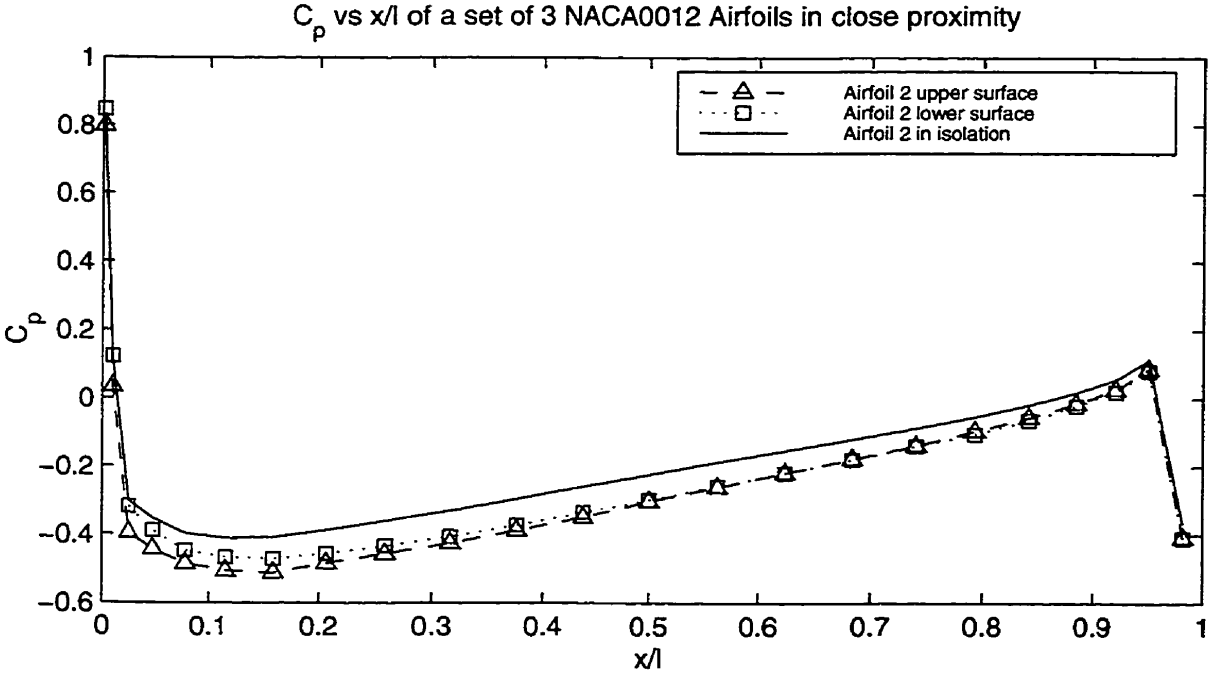


Figure 16: (b) C_p distribution of Airfoil 2 in a set of 3 airfoils in close proximity (x = 0, y = 1 chord length, $\alpha = 0$).

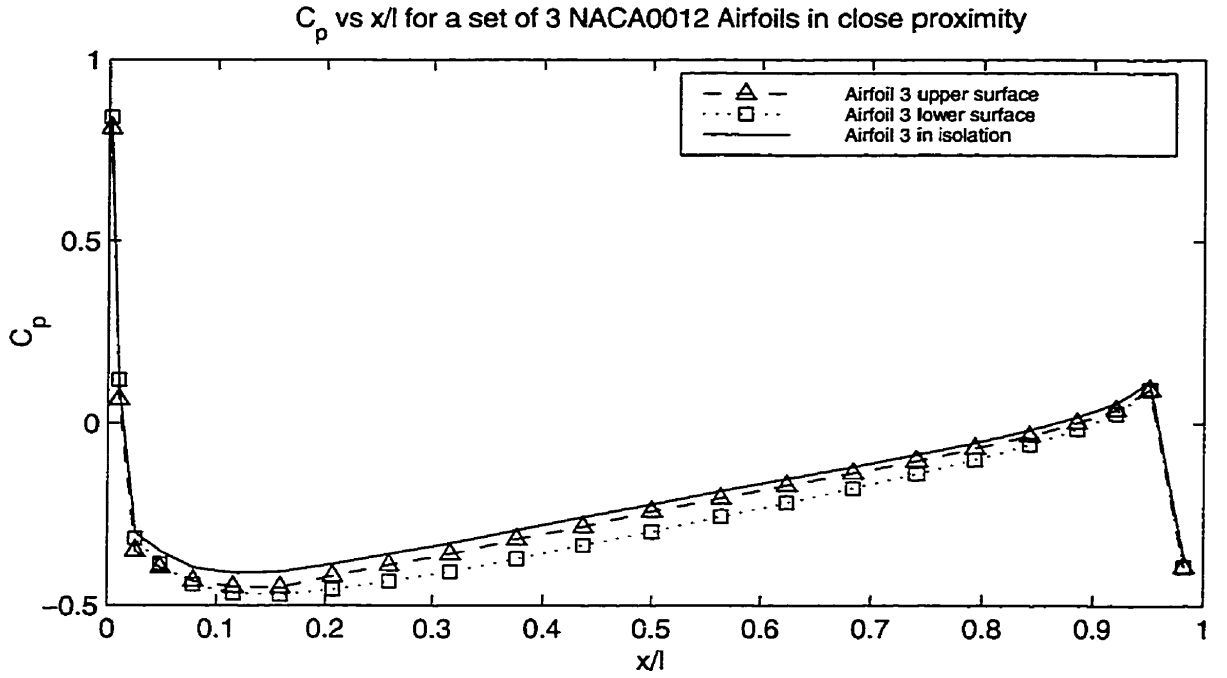


Figure 17: (c) C_p distribution of Airfoil 3 in a set of 3 airfoils in close proximity (x = 0, y = 1 chord length, $\alpha = 0$).

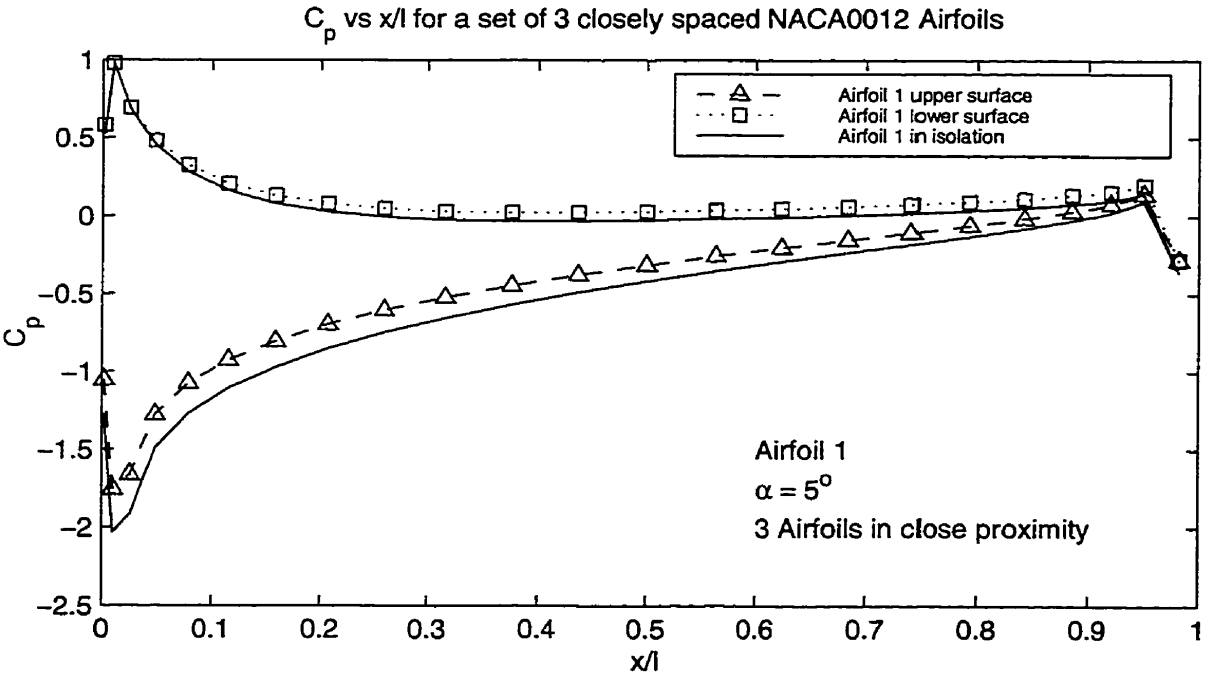


Figure 18: (a) C_p distribution of Airfoil 1 in a set of 3 airfoils in close proximity (x = 0, y = 1 chord length) at $\alpha = 5°$.

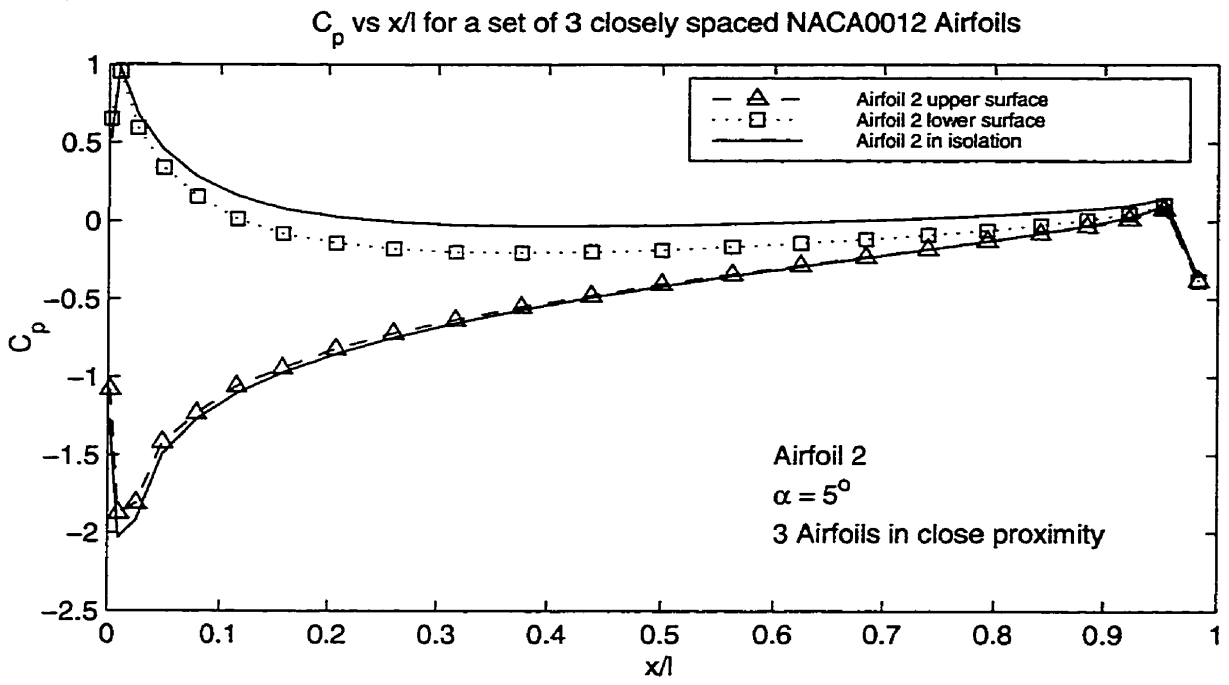


Figure 19: (b) C_p distribution of Airfoil 2 in a set of 3 airfoils in close proximity (x = 0, y = 1 chord length) at $\alpha = 5^o$.

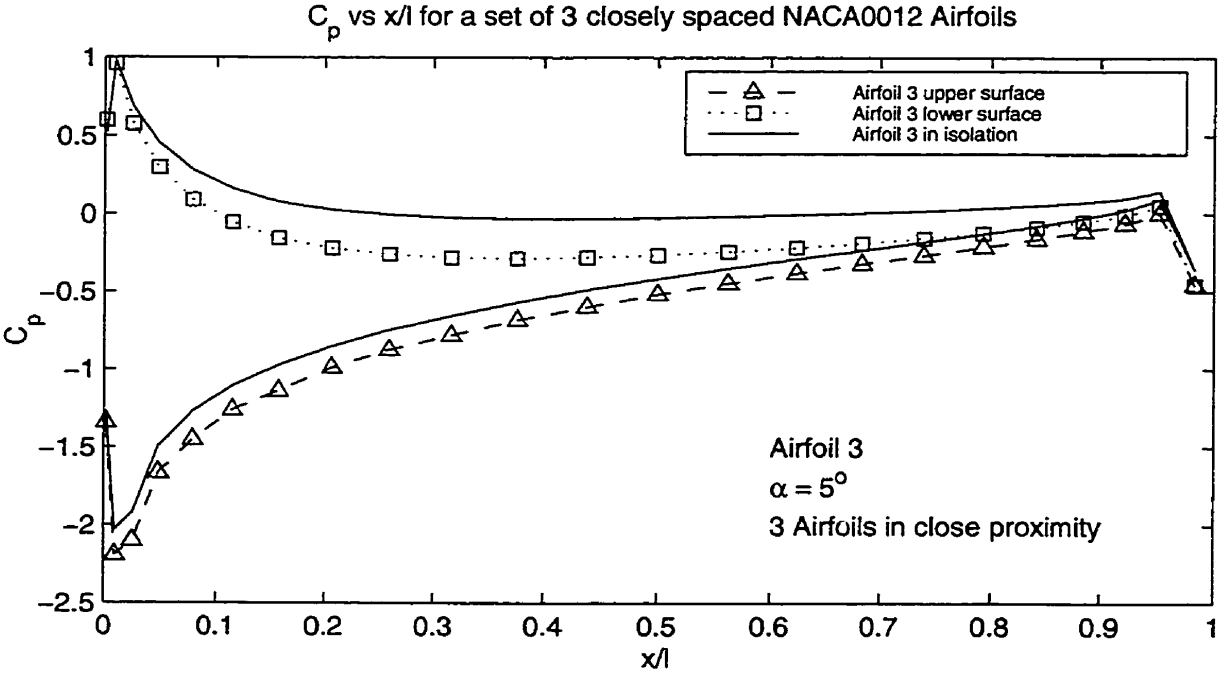


Figure 20: (c) C_p distribution of Airfoil 3 in a set of 3 airfoils in close proximity (x = 0, y = 1 chord length) at $\alpha = 5^o$.

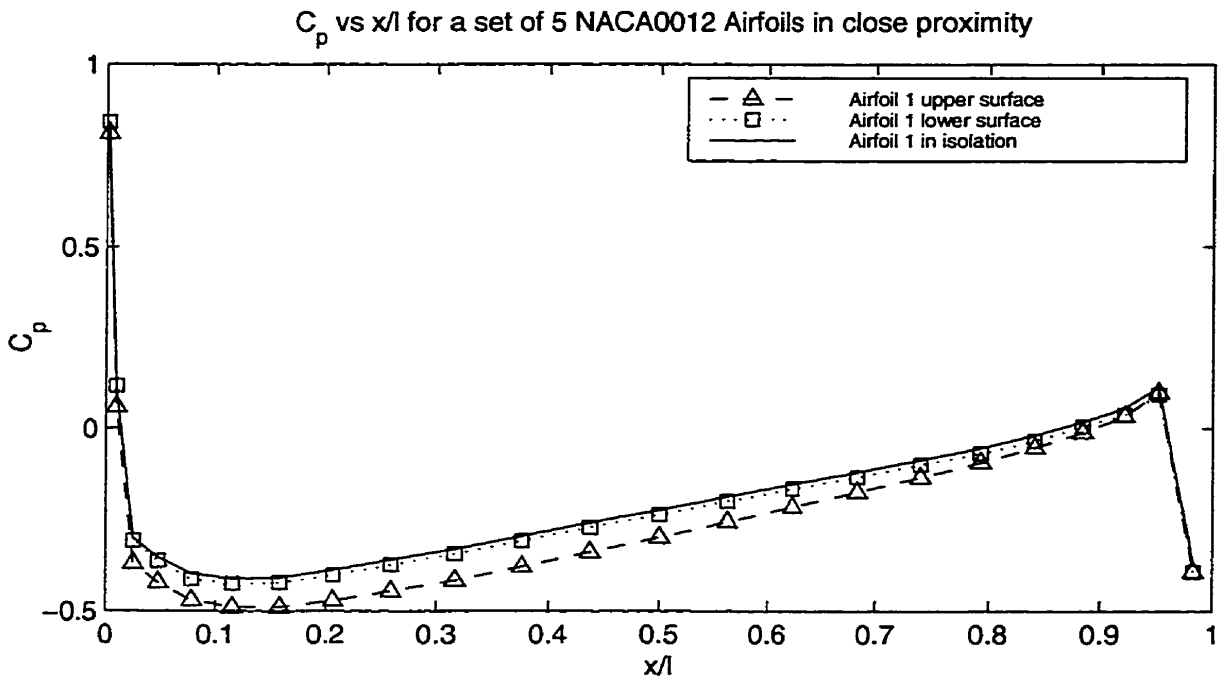


Figure 21: (a) C_p distribution of Airfoil 1 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length, $\alpha = 0$).

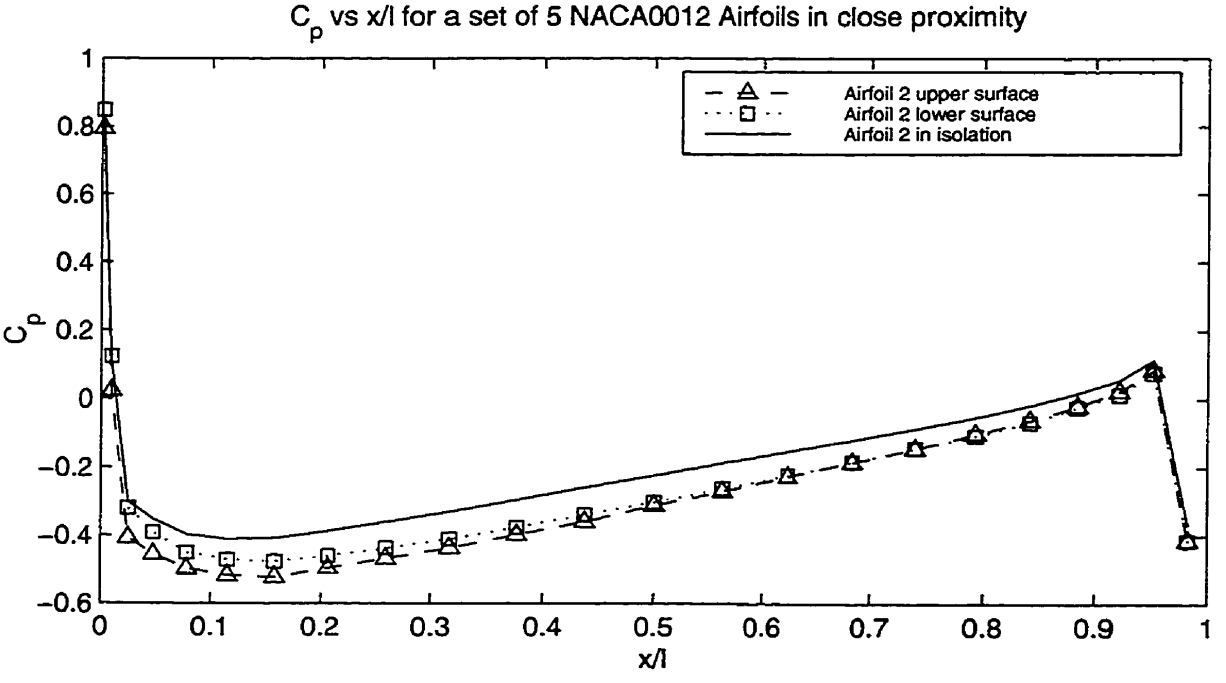


Figure 22: (b) C_p distribution of Airfoil 2 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length, $\alpha = 0$).

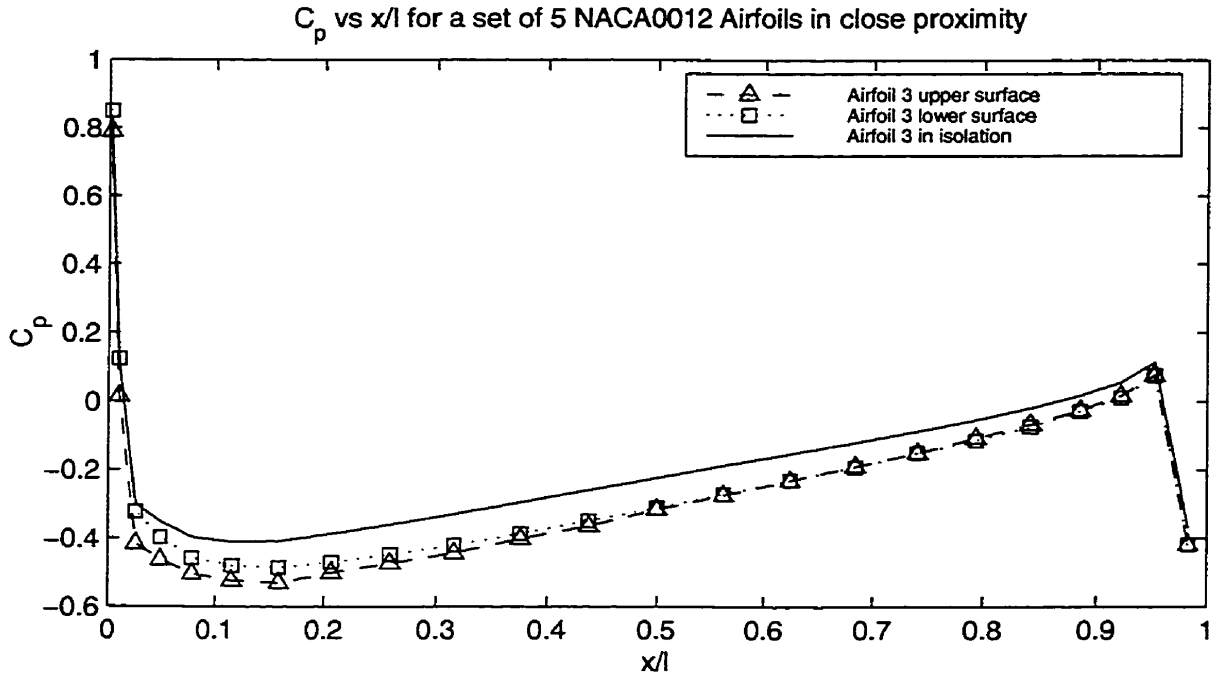


Figure 23: (c) C_p distribution of Airfoil 3 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length, $\alpha = 0$).

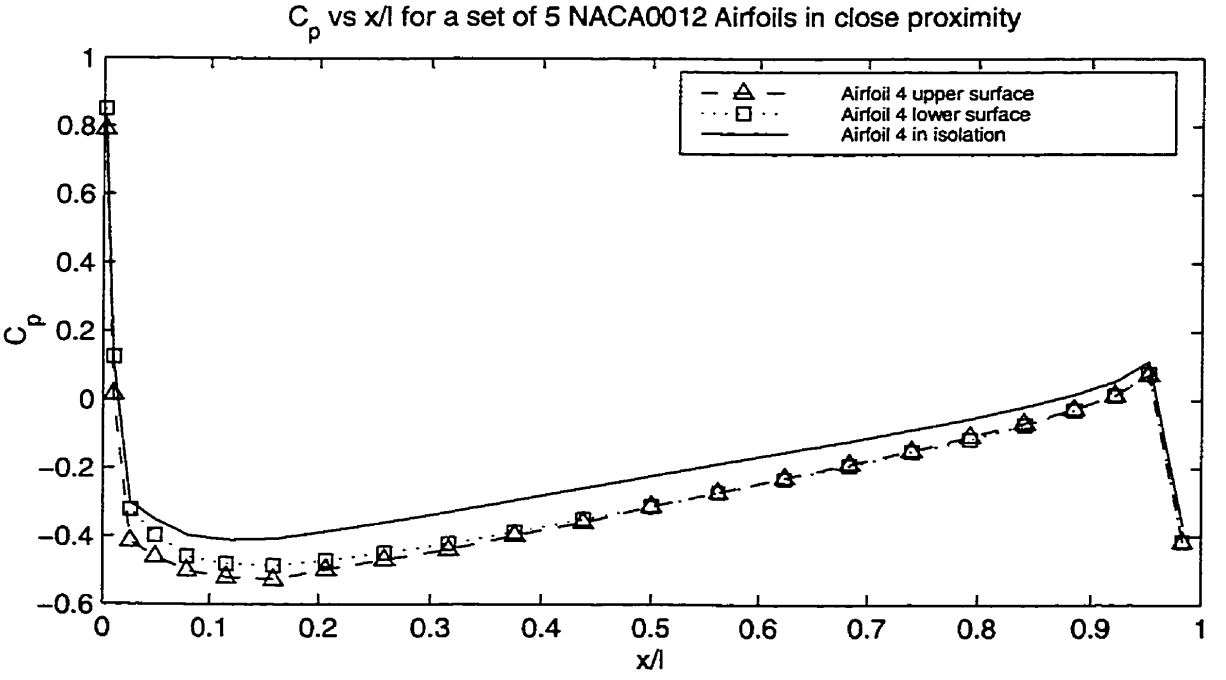


Figure 24: (d) C_p distribution of Airfoil 4 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length, $\alpha = 0$).

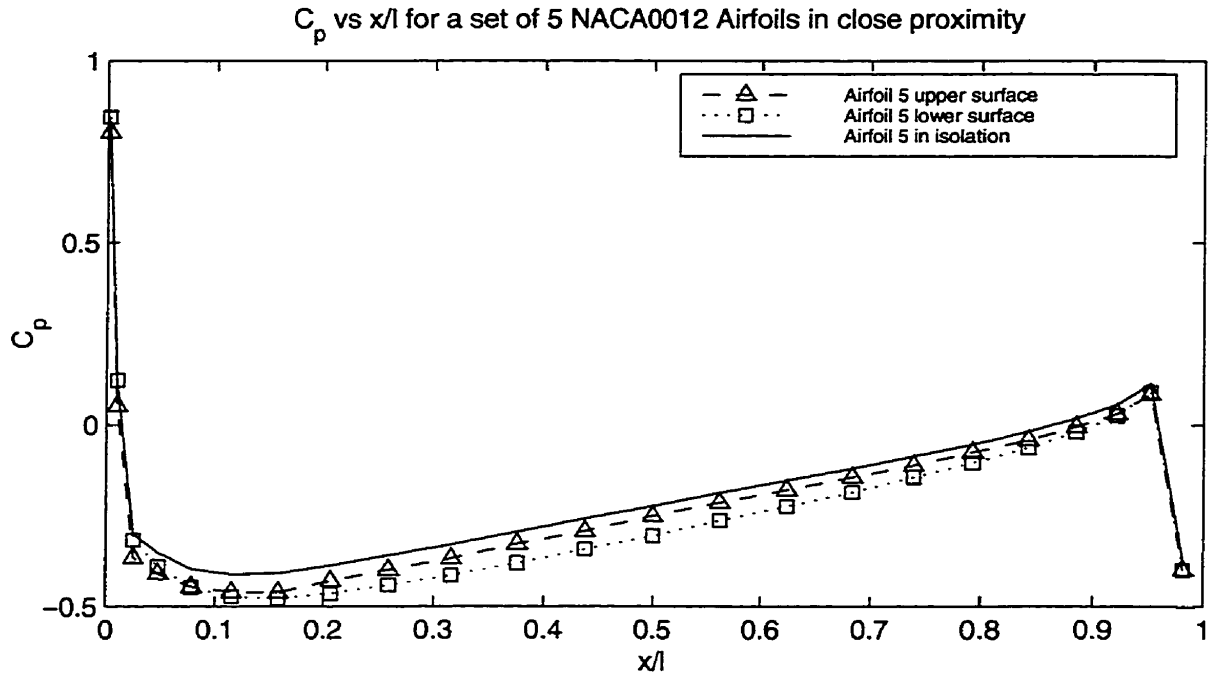


Figure 25: (e) C_p distribution of Airfoil 5 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length, $\alpha = 0$).

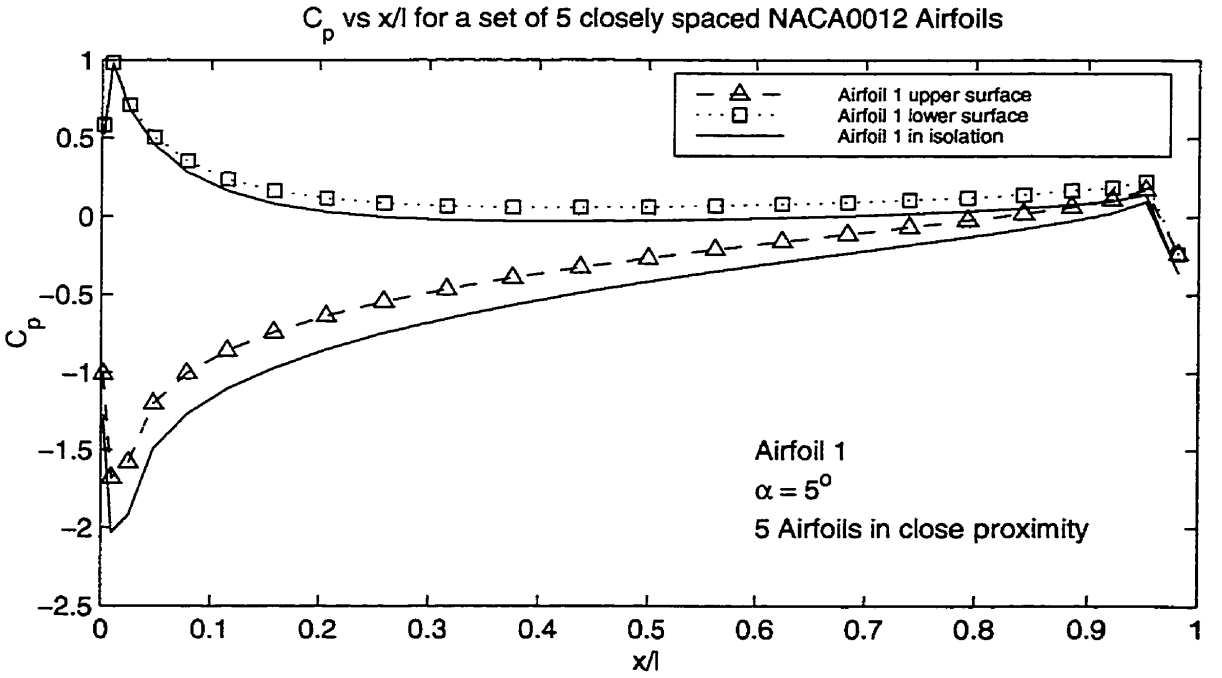


Figure 26: (a) C_p distribution of Airfoil 1 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length) at $\alpha = 5^o$.

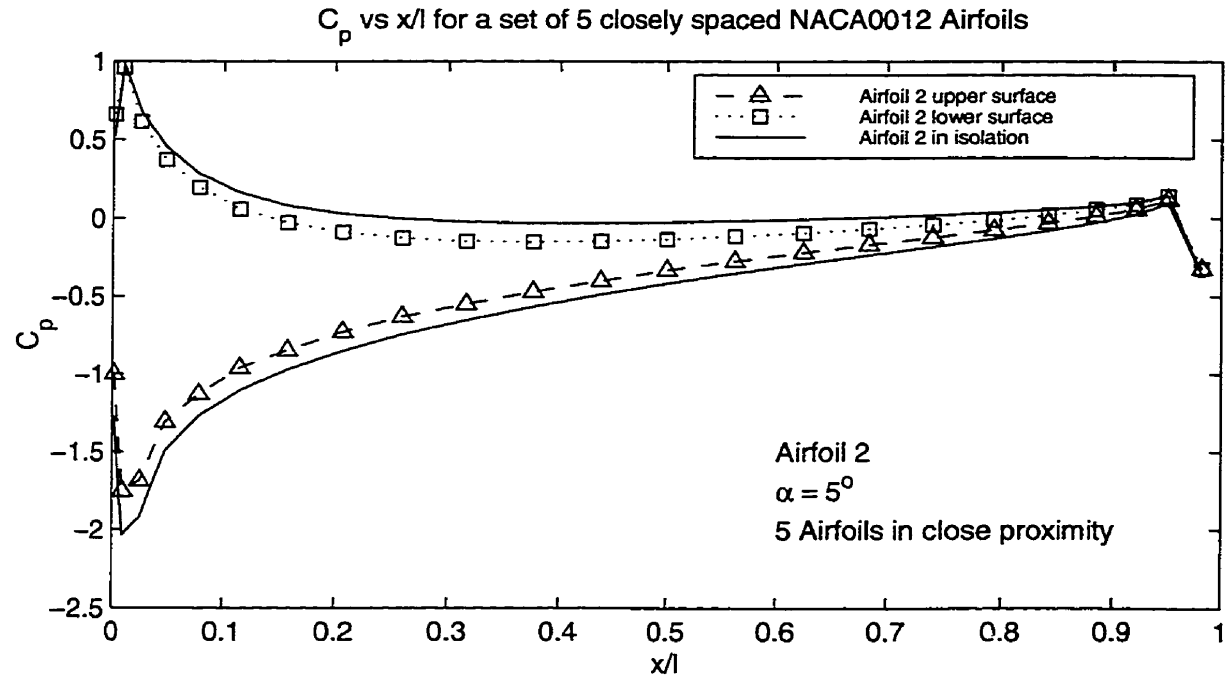


Figure 27: (b) C_p distribution of Airfoil 2 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length) at $\alpha = 5^o$.

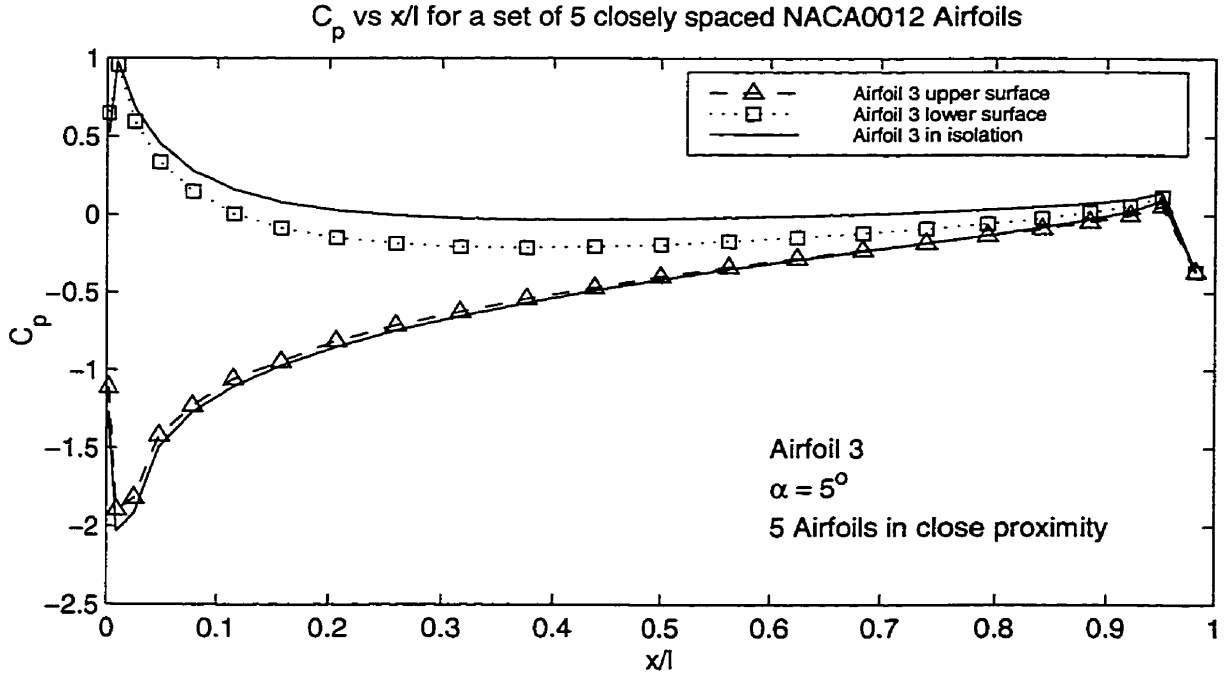


Figure 28: (c) C_p distribution of Airfoil 3 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length) at $\alpha = 5^o$.

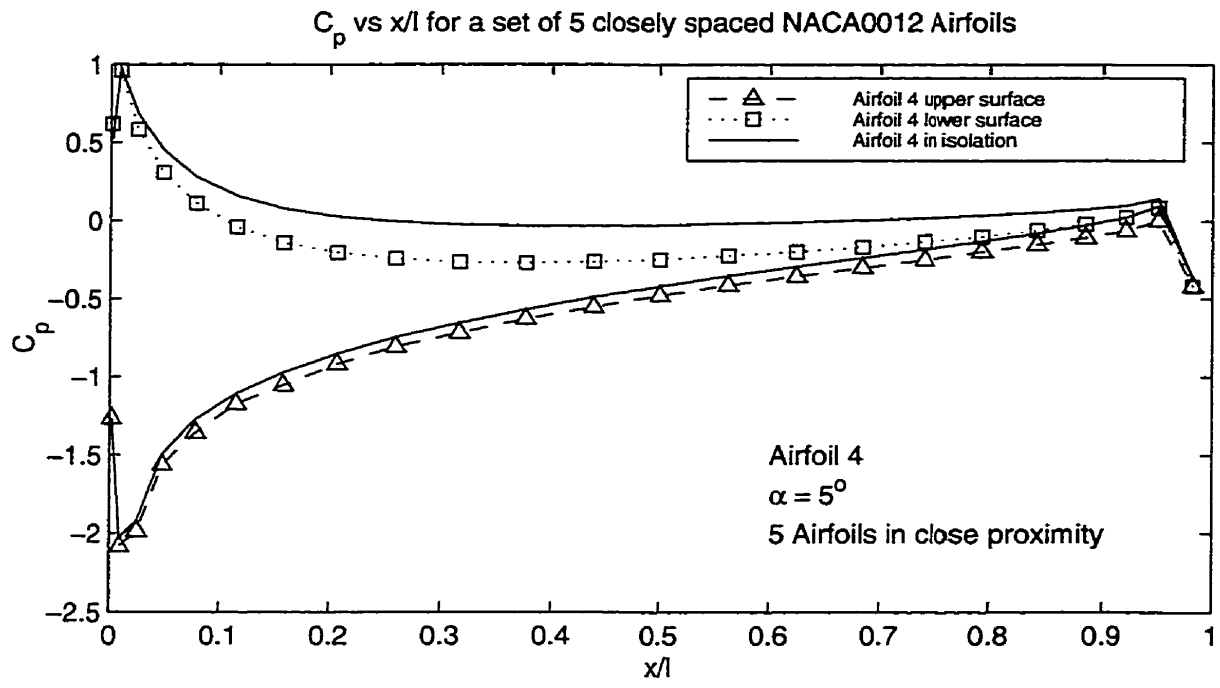


Figure 29: (d) C_p distribution of Airfoil 4 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length) at $\alpha = 5^o$.

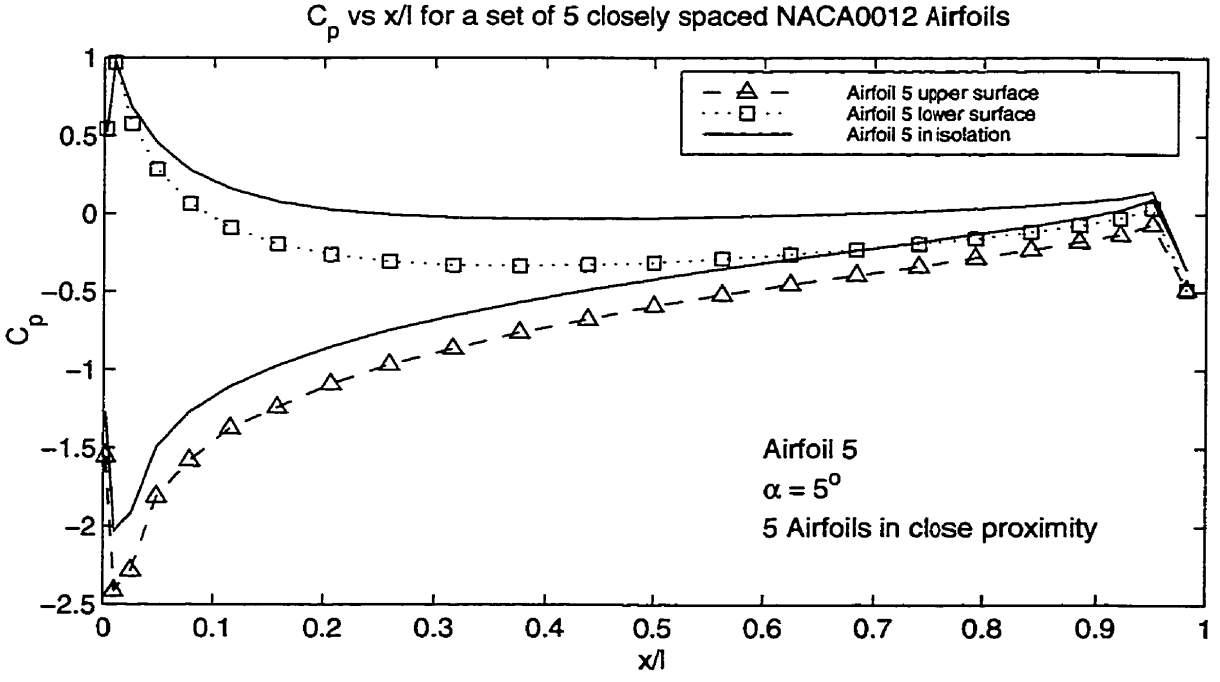


Figure 30: (e) C_p distribution of Airfoil 5 in a set of 5 airfoils in close proximity (x = 0, y = 1 chord length) at $\alpha = 5^o$.