An Analytic Model for ATM Network Performance and its Application to BSP

E. Karimi and D.B. Skillicorn
{karimi,skill}@qucis.queensu.ca

December 1997
External Technical Report
ISSN-0836-0227-
97-414

Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada K7L 3N6

Document prepared December 5, 1997

Abstract

Bulk Synchronous Parallelism (BSP) as a parallel model enables accurate costs of parallel programs to be predicted from the program structure and two architectural parameters, g, the permeability of the network, and l, the time required for barrier synchronisation.

Networks such as ATM already play a role in parallel computers built as networks of workstations, and may become the standard mechanism for interconnecting processors at all scales. We present an analytic model for determining the BSP parameters of such architectures. Although the model is simple, there is substantial agreement with measured results where these are known. This represents the first time that these architectural parameters have been determined other than by benchmarking, and suggests that the approach may be serviceable for other wormhole routed networks.

Keywords: parallel computing, interconnection network, performance modelling, total exchange, bulk synchronous parallelism, latency, throughput.

1 Introduction

A successful model [26] for parallel computing should:

- be simple to understand,
- provide us with an easy way to program,
- be architecture-independent,
- and provide a method to predict performance.

Most parallel programming models do not have the above properties. There is a need for a standard architecture model of parallel computing, which would enable architecture-independent programming.

Bulk Synchronous Parallelism (BSP) [37] is a parallel computing model, which shows great promise [14, 27] and satisfies many of the properties above. BSP provides an easy way to model parallelism by abstracting the details of parallel execution.

The success of parallel computing also depends on the performance of communication among the processors, which is related to the physical network and its communication protocols. Implementing parallel computers atop Ethernet or other shared-physical media can result in large communication latencies, due to inefficient access to network resources. Switch-based media are a solution for this problem. The advantage of switch-based connections compared to shared media is that multiple packets can be passed simultaneously through switches. Switch-based communications are based on the store-and-forward switching technique. In this technique, each packet should completely arrive at the intermediate switches before it can be sent to the next destination along the path, which makes the latency of transferring packets dependent on the number of intermediate switches. Therefore, topology has a very strong effect on the latency of communication.

1.1 Contribution

We model ATM networks based on pipelining messages through the network because sensible runtime systems and hardware transmit messages in this way. The model is then used to determine optimum message sizes to achieve maximum throughput. There is strong agreement between the throughput and latancy predicted by the model, and measured results.

These results are then used to predict BSP architectural parameters. We use total-exchange (every processor sends/receives different messages to/from the others) and one-relation (each processor sends/receives at most one message to/from another processor) algorithms to predict the g parameter. Total exchange and hardware broadcast algorithms are modelled to determine the value of l.

The contributions of this paper are:

- Providing an analytic model to estimate the optimum message size that should be sent through the network, and the achieved latency. In this analytic model, we model the pipelining method for data transmission from the processor to the network.
- Demonstrating the use of the analytic model to analyse the BSP communication parameter, g, considering heavily-loaded, and lightly-loaded networks.
- Demonstrating the use of the analytic model to analyse the BSP synchronisation parameter, l, using two different implementation techniques, namely total-exchange and broadcast.

Section 2 is an overview of the parallel models, especially BSP, and the important concepts of ATM networks. In Section 3, we establish our analytic model to estimate the latency of communication considering both heavily-loaded and lightly-loaded networks. Data pipelining, which is used to transmit data through a network, is introduced in this section. Moreover, we make use of ATM hardware multi-point services to analyse the behaviour of collective operations (total-exchange, broadcast) on communication latency. These collective operations are later used in order to provide a framework to estimate the synchronisation cost of BSP programs. In Section 4, we expand our analytic model to cover the BSP synchronisation, and communication costs. We introduce a framework to estimate the BSP parameters, g and l, using this analytic model.

2 Background

In this section, we review the important concepts of ATM and the work that has been done on ATM APIs. We also give a brief presentation of the BSP parallel model and its properties.

2.1 Asynchronous Transfer Mode (ATM) Network

There are considerable technical and operational advantages to communication providers in integrating all of their services into a single network, which handles all geographical scales and delivers a variety of data rates. This integrated service is called *Broadband Integrated Services Digital Networks (B-ISDN)*, and offers video on demand, live television, CD-quality music, and many other services using the telephone line [34]. The technology that provides these integrated services is called *Asynchronous Transfer Mode (ATM)* [11]. ATM transmits data using small fixed-size packets called *cells*. The cells are 53 bytes in length, of which 5 bytes are devoted to the header and the rest are for payload.

The properties of ATM that make it attractive are :

- Using high-bandwidth optical fibres.
- Fast switching of small fixed-size cells. The advantage of switch-based connections over shared media is that they can pass multiple packets simultaneously.
- Connection-oriented communication.
- Scalability. It is applicable to both local area networks and wide area networks.
- In collective communications, in which more than two processors are involved, communication networks face the problems of network access, congestion, and latency. ATM networks are designed to provide high throughput, which should be a better way to meet the needs of collective communications [15, 25].

The intended speeds for ATM networks are 155 Mbps and 622 Mbps, with the possibility of gigabit speeds later [34].

ATM networks are connection-oriented, that is the route is established between source and destination before data transmission. This connection is called a Virtual Circuit (VC). ATM allows multiple VCs to be grouped into Virtual Paths (VP).

ATM's 53 byte cells have a 5 byte overhead, which contains:

- 4-bit Generic Flow Control (GFC), which is used for flow control.
- 8-bit Virtual Path Identifier (VPI).
- 16-bit Virtual Channel Identifier (VCI)
- 3-bit Payload Type Identifier (PTI). This field separates the user cells from the network management cells.
- 1-bit Cell Loss Priority (CLP). This field is used during network congestion. It helps to decide which cells to discard.
- 8-bit Header Error Correction (HEC), which is a checksum field on the first four bytes.

All the switches have a table of incoming and outgoing VCs, and they route cells accordingly. Because cells are small and of fixed length, switching operations are done very quickly.

In the ATM protocol stack (Figure 1), the ATM layer handles construction and verification of the cell headers, routing the cells, cells multiplexing and demultiplexing [11, 12, 31, 34]. However, to provide services to users, another layer is needed. The ATM Adaptation Layer (AAL), above the ATM layer, provides a variety of services for applications. There are four different AAL protocols:

- AAL 1, which supports real-time, constant bit rate, connection-oriented traffic;
- AAL 2, which supports variable bit rate, connection-oriented traffic;
- AAL 3/4, which supports variable bit rate, both connection-oriented and connectionless traffic; and
- AAL 5, which was designed to provide the AAL 3/4 services more efficiently.

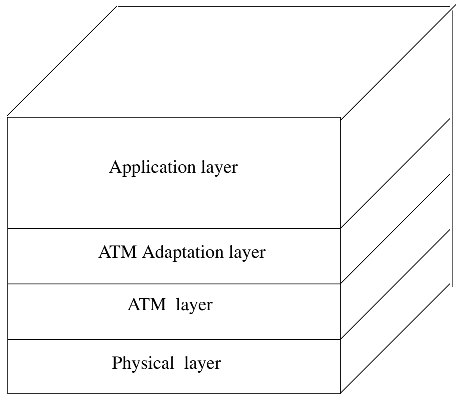

Figure 1: ATM protocol stack

2.2 ATM Application Programming Interface (API)

In order to communicate with the ATM layer, applications need to use Application Programming Interfaces (API). Several different implementations of APIs are available, for example: Fore Systems ATM API [8], the BSD socket programming interface [3], Sun's Remote Procedure Call (RPC) [3], and the Parallel Virtual Machine (PVM) [4] message passing library. Lin *et al.* [25] discuss the performance tradeoffs of different APIs in an ATM environment. Fore's API provides capabilities which are not available in other APIs because it provides direct access to the ATM layer for the application layer. Moreover, each API represents communication with a different protocol layer, and introduces different overheads. For example, in the socket interface, applications use TCP/IP protocols which introduce additional overhead. Sun's RPC uses External Data Representation (XDR) [3], which is also an extra overhead.

Fore's ATM API The ATM API routines provide connection-oriented communication. A connection has to be established between two ends before the data can be transferred. After that, the network makes a strong effort to send the ATM cells to their destination. Fore's library routines use a socket-like interface. This API provides lower communication overhead because it provides applications with direct access to the ATM Adaptation Layer. Figure 2 shows this capability.

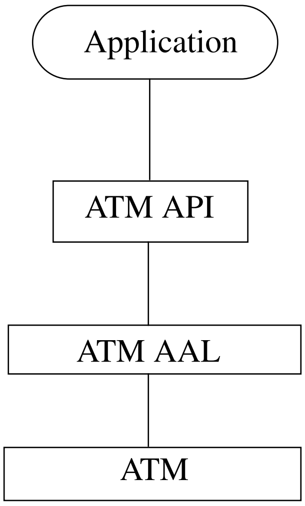

Figure 2: ATM API

Using Fore's API. In order to make a connection, the application first uses *atm_open()* to open a file descriptor. Then, it calls *atm_bind()* to bind a local Application Service Access Point (ASAP) to the file descriptor. Connection is set up using *atm_connect* on one end together with *atm_listen* and *atm_accept* on the other end. These routines allocate an ATM VPI and VCI to the connection. Before the connection is accepted, the bandwidth and QoS defined by the host is checked by the Connection Admission Control (CAC) algorithm [12]. If the communication resources are available for the requested QoS, then the connection will be set up.

Applications can specify the AAL layer from which they want to work. This is provided to the applications using an argument in the *atm_connect()* routine. After connection establishment, applications can use *atm_send()* and *atm_recv()* to send and receive messages. The maximum size of the message depends on the selected AAL and the device driver implementation. In our work, we want to automate the assigning of the maximum message size for a connection to minimise congestion and latency. Our model will be based on Fore's ATM API routines.

2.3 Collective communications

Collective communications, such as scatter, gather, broadcast, and reduce, are often used in parallel computing. Efficient implementation of such operations is critical to the performance of parallel applications. Collective operations can be classified as process control, data movement, and global compute operations. These operations involve more than two processors.

Category	Collective Operation	Definition
data movement	broadcast	one processor sends the same message to all processors
	scatter	one processor sends a different message to each processor
	gather	each processor sends a different message to one processor
	all-to-all broadcast	every processor performs a broadcast
	all-to-all scatter-gather total-exchange	every processor performs a scatter and gather
process control	barrier synchronisation	all processors reach the same point before continuing
global operations	reduction	processors perform a global operation on data
	scan	partial reduction

Table 1: Collective Operations

2.4 What is BSP?

One major area of research in practical parallel computing is the search for a standard architectural model for parallel computers. BSP [27, 29] is a promising start.

Bulk Synchronous Parallelism provides standardisation in two steps :

- A standard architectural model. It abstracts all parallel architectures into one simple machine that consists of a set of processor-memory pairs, a global communication network, and a mechanism for the barrier synchronisation of the processors. Both message passing and shared-memory programming styles can be represented by this architecture (Figure 3).
- A foundation for architecture-independent programming. A BSP program proceeds in one or more supersteps [14, 32, 36]. Each superstep consists of a set of local memory operations, a set of global communications, in which each

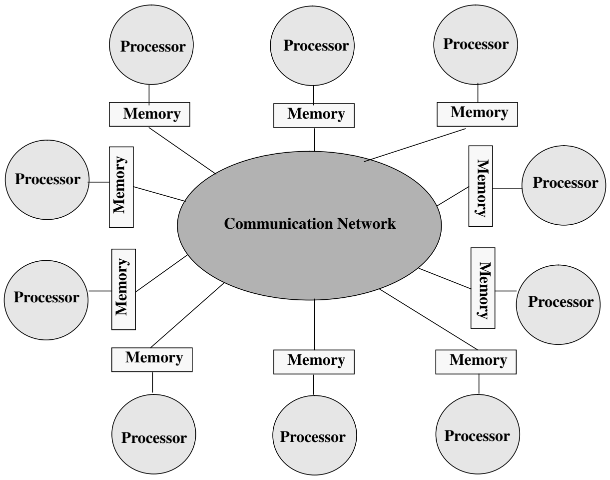

Figure 3: BSP architectural model

processor sends and receives a number of messages, and a barrier synchronisation, in which all the non-local data communications take effect (Figure 4).

This simple notion of supersteps provides us with an easy way of achieving scalable portable parallel programming [9], because it hides the architecture-dependent features and abstracts a model of parallelism. It also makes it possible to predict the performance of software programs on a given architecture. BSP uses only two parameters to capture the properties of each architecture. These two architectural parameters are:

- g, which is the communication cost per one-word message, in the context of continuous randomly-addressed traffic; and
- l, which is the time required for a global synchronisation among the processors.

These parameters clearly depend on the number and computational speed of the processors, the speed and bandwidth of the communication networks, and the cost of barrier synchronisation. The speed of the processors is expressed in terms of number of basic unit operations (called *steps*) that they execute each second. Each step is often a single floating point operation (*flop*). The parameter g captures the effective ratio between the rate at which data can be moved between processors under heavy, but reasonable, load and the computational speed of the processors.

The cost of communication depends not on the total volume of traffic moving through the network but only on the maximum fan-out or fan-in of traffic at any processor. Call a communication pattern in which no processor sends or receives

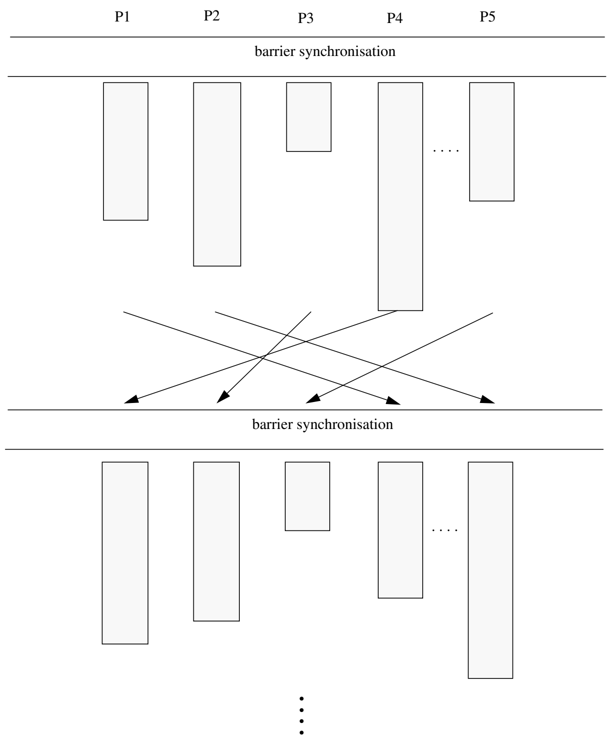

Figure 4: BSP programming phases

more than h words an h-relation. Then the BSP cost model asserts that the time taken for the global communication step will be hg. g is often expressed in units of words transferred per unit floating point instruction time.

Therefore, the cost of a superstep is the sum of three costs:

- the time of local computation in each process during the superstep,
- the time of delivery of the h-relation communication, and
- the cost of barrier synchronisation.

or

$$\text{Cost of a superstep} = \text{Max}(w_i) + \text{Max}(h_i g) + l$$

or simply as

$$\text{Cost of a superstep} = w + hg + l$$

where w_i is the local computation of process i, $h_i g$ is the cost of the h-relation communication and l is the cost of barrier synchronisation. The cost of a BSP program is the sum of the costs of its supersteps.

The current best implementation of BSP is the Oxford BSP Library [10]: It is a library callable from sequential languages such as Fortran and C. It uses the SPMD (Single Program Multiple Data) approach.

3 Analytic modelling of ATM performance

In order to observe the performance of a communication network, several approaches are available. The most common ones are:

- Implementation, which helps us judge the performance of a specific communication network with a certain environment and a particular parallel algorithm. Although implementation may give us precise information about parallel computing performance in a specific environment, it is not a good choice to use in order to observe the performance of a wide range of environments. Moreover, it is hard to provide certain experimental situations on a real environment.

- Simulation is another approach that may seem to be a good choice for performance measurement. It can be developed to provide the situations that are hard to achieve in a real environment, and therefore gives stronger results compared to an implementation model. However, simulation is usually very expensive, due to the level of detail that it uses.

- Queuing network modelling [2] models the performance of a communication network often using equations to describe the behaviour of each object in the system. Such models can be very complex.

- Analytic modeling is straightforward and cheap, due to the level of abstraction that it provides. However, it may miss details.

In this section issues in communication performance are discussed. We introduce an analytic model for network performance using a pipelining scheme considering the communication performance issues. The reason for choosing the pipelining scheme is that it represents what actually happens in processors, and the ATM hardware and switches.

3.1 Networks of workstations

The cost of supercomputers has encouraged researchers to find other cost-effective options in the area of parallel computing. One proposal is the use of a network of workstations [6, 7, 30, 38, 39]. The advantage of this approach is its lower cost. The drawback is that due to the limited speed and reliability of current medium-sharing LAN technologies such as Ethernet and token rings, the achieved performance is low. However, low-bandwidth networks (Ethernet, token ring) are being replaced by high-bandwidth switch-based networks (ATM). These improvements provide the potential for workstation clusters to provide performance comparable to that of supercomputers [5, 17, 39]. Figure 5 is an example of a cluster of workstations over an ATM network.

The appearance of high-bandwidth, flexible, switch-based network technology, such as ATM, promises to minimise the performance degradation inherent to networking. However, bandwidth alone is not the only factor needed to reduce communication time. Communication performance relies on many other factors [22, 23, 38] such as the overhead of the protocol stack, latency, congestion, and message size. In this section, we focus on optimising these factors to design a method of handling collective operations over an ATM communication network and model its cost.

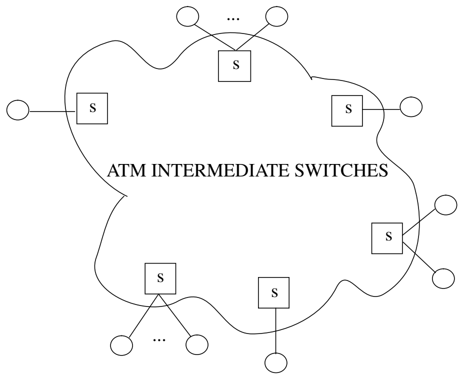

Figure 5: Network of workstations using ATM

3.2 Issues in Communication Performance

There are two components of communication overheads [21, 30, 40]: hardware and software overheads. Hardware overheads include: host interface overhead, the switch and signal propagation delay, and the architecture of the host computer. Software overheads include: interactions with the host operating system, device drivers, and protocols.

Overheads can also be broken into per-message and per-byte overheads. For example, interrupt handling, and context switching in the host operating system are per-message overheads. Data checksumming and data copying from user space to kernel space and vice versa are per-byte costs.

Performance can also be affected by network congestion [41]. The network is congested if its resources have been completely consumed. Congestion in the network may result in packet loss, and queueing delay in the switches.

There are different solutions for network congestion [11, 34]. ATM employs hardware-based flow control [23] to restrict each host to an agreed transfer rate based on the network capacity. This flow control reduces the network congestion. Throughput is limited by the rate at which the sender can send messages into the network.

3.3 Communication Overheads

The choice of API can have a significant effect on performance. For example, the BSD socket interface uses TCP as a transport layer with IP. This implementation degrades performance [1]. ATM networks provide an end-to-end Quality of Service (QoS) guarantee for each virtual circuit. IP loses this individual QoSs [20], because it multiplexes multiple transport connections into a single VC. TCP checksums a packet to detect problems. However, ATM Adaptation Layer 5 (AAL5) does checksumming, so TCP provides a redundant function which is costly. Moreover, TCP/IP increases the header overhead in the packets [11]. This overhead reduces the bandwidth available to the application layer. Some researchers [1, 5] have designed a transport layer that turns off data checksumming and other redundant functions of TCP/IP. They have written their own memory management, and task scheduling. In this case, the operating system is only responsible for handling the packet arrival interrupts, memory allocation, and calls to the task scheduler. The FORE Systems API provides relatively efficient communication.

3.4 Communication Latency

In communication networks, the time required to move data between nodes is critical to system performance, because it effectively determines what granularity levels of parallelism are possible in executing an application program. An important metric to evaluate a network is communication latency, which consists of three values :

- start-up latency, the time to handle the packet at both source and destination nodes.
- transmission time, the time after the head of the packet has entered the network at the source until the end of the packet exits from the network at the other end, without considering any blocking time.
- blocking time, all other delays that happen due to the use of shared resources, such as the delay of channel contention.

The start-up latency depends mainly on the design of protocol stack and the interface between the host and switch. In an ATM network, packets may traverse one or more intermediate nodes (switches) before arriving at the destination node. In this case, the topology of the network may have a major effect on the network latency. However, with the use of different switching techniques [33] this effect can be decreased considerably.

ATM switching techniques *Packet-switched* networks are based on the store and forward switching technique. ATM is also a kind of packet-switched network (packets are small size cells). In the store-and-forward method, when a packet reaches an intermediate node, the packet is stored. It is then forwarded to the next switch when the output channel is available. In this technique, the network latency is :

$$Latency_{network} = \left(\frac{PacketSize}{ChannelBandwidth}\right) * (NumberOfSwitches)$$

It is clear that the path length between the source and destination has a direct impact on the network latency in the store and forward technique. To decrease data transmission time, the virtual *cut-through* method has been introduced by Kermani and Kleinrock [24, 28]. In this method, the packet is stored in an intermediate switch only if the output channel is busy. In this technique, the header is analysed by the switch and the rest of packet follows the header in a pipeline fashion. Therefore the network latency is :

$$Latency_{network} = \left(\frac{HeaderLength}{ChannelBandwidth}\right) * NumberOfSwitches + \frac{PacketLength}{ChannelBandwidth}$$

When $PacketLength \gg HeaderLength$, the effect of the number of intermediate switches (*NumberOfSwitches*) on the network latency is reduced. Figure 6 shows these two techniques of packet switching.

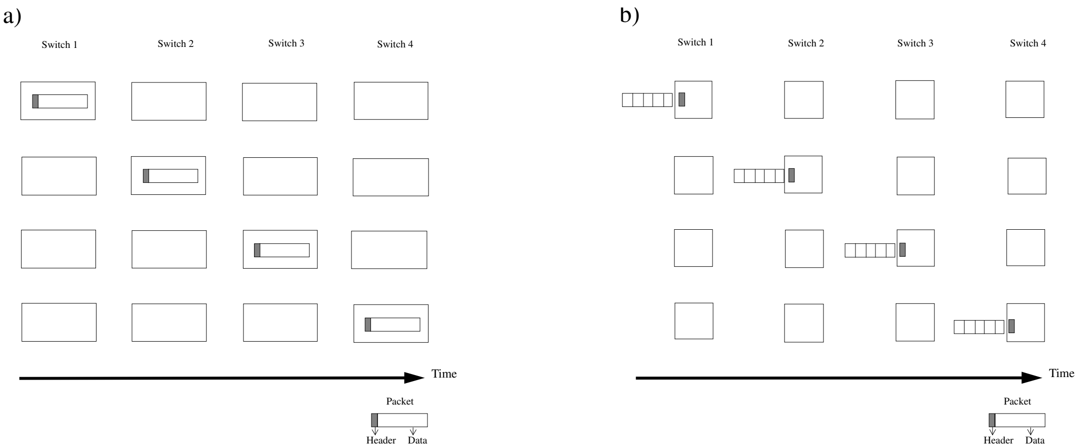

Figure 6: a) Store and forward packet switching. b) Cut-through packet switching.

Because ATM breaks a packet into a number of small cells, switching is performed very quickly. Furthermore, cells are pipelined over the virtual circuit [16, 18, 35]. These characteristics reduce the effect of the path length on the latency. We will use this approach (pipelining the packets through the virtual circuit) in our model. There are two reasons for this:

- pipelining packets provides us with continuous traffic schemes across the communication links. We want to have this continuous traffic to provide a good estimate of g.
- This approach reduces the effect of path length on message transfer time.

3.5 The maximum theoretically-available bandwidth

Another important factor for network performance is bandwidth. The maximum bandwidth available to the Application layer depends on protocol formats and the overheads which are involved in each layer. Let us assume that the physical layer is based on a 155 Mbit/sec SONET STS-3c/OC-3c Physical Layer Interface. Its frame format is shown in figure 7.

Figure 7: SONET STS-3c/OC-3c

The size of the SONET STS-3c/OC-3c frame is:

$$\begin{aligned}\text{Oc3cframe} &= ((87 + 3) * 9) * 3 \\ &= 2430 \text{ bytes}\end{aligned}$$

The overhead consists of: path overhead (9 bytes), line overhead (54 bytes), and section overhead (27 bytes). The total overhead of the physical layer is 90 bytes, and the payload is (2430 - 90) bytes. We can see that the bandwidth left to the ATM Layer is :

$$
\begin{aligned}
AtmBandwidth &= \frac{PayloadLength}{TotalLength} * ChannelBandwidth \\
&= \frac{2430-90}{2430} * 155.52 \\
&= 149.760 \text{ Mbit/sec}
\end{aligned}
$$

The same computation can be performed for the other layers up to the Application layer. At the ATM layer, the data units are cells. Figure 8 shows an ATM cell. As shown in the figure, a cell has a 5 bytes header and a 48 byte payload. So,

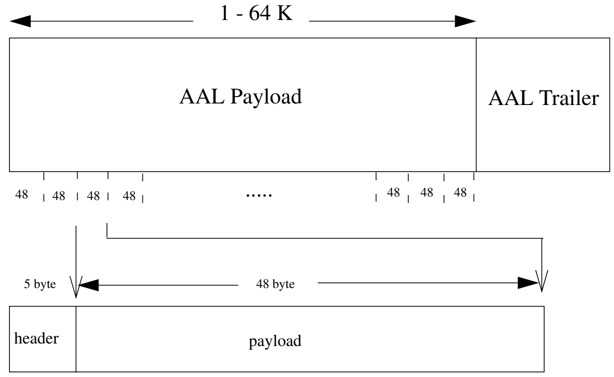

Figure 8: ATM cell

the bandwidth available to AAL5 layer is :

$$
\begin{aligned}
AalBandwidth &= \frac{PayloadLength}{TotalLength} * ChannelBandwidth \\
&= \frac{48}{53} * 149.760 = 135.632 \text{ Mbit/sec}
\end{aligned}
$$

The Fore's ATM API does not have any transport protocol layer over the AAL5 so 135.632 Mbit/sec is the theoretically-calculated bandwidth that is given to the application.

3.6 Pipelining and optimum message size

ATM cells are pipelined through the VCs, to reduce the communication latency. We use pipelining to model the data transmission in the upper layers of the protocol stack because it is used in runtime system, and the hardware works in pipelined manner. Obviously, pipelining messages reduces the start-up latency, and the effect of the transmission time on the end-to-end latency (See Figure 9).

An abstract view of the model Assume x to be the size of data to be sent through the VCs. Instead of sending x as a single message, we send it as n messages

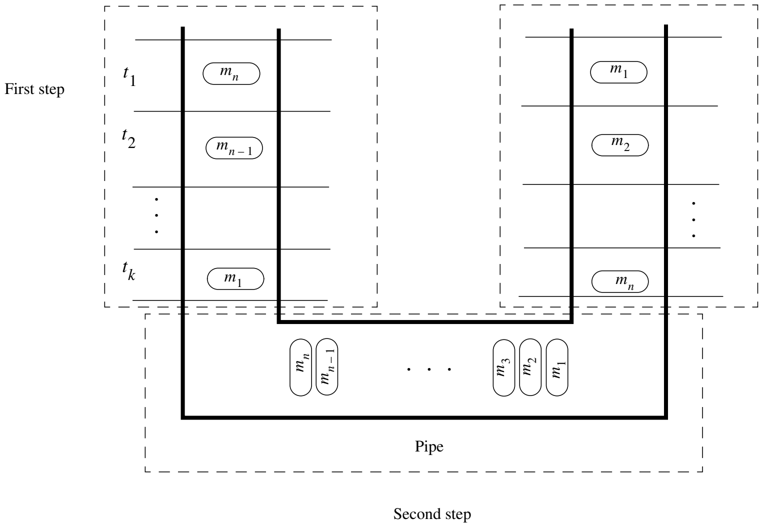

Figure 9: Pipelining messages from the application layer to the pipes or VCs at one end and receiving them at the other end

of size m (m_1 to m_n). Moreover, assume t_k, ($k = 1$ to i) to be the latencies of different layers in the protocol stack, as the messages are traversing the pipeline stages, and i to be the number of layers that the message traverses before it gets to the pipe (VC). Note that, t_k is a function of m. Therefore, pipelining messages has the following latency:

$$Latency_{protocol} = \sum_{k=1}^{i} t_k + (n-1)\,\mathrm{MAX}_{k=1}^{i}\,(t_k) \tag{1}$$

This is the latency at one end. The other end receives the messages in the same manner (pipelined), and therefore it has almost the same protocol latency. The second step of Figure 9 shows the latency of transferring data through a pipe. Buffering messages and packing them, before they are sent through the pipes (VCs), helps to better fill up the pipes, and therefore get a lower network latency.

Let us consider a pipe modelling the communication channel and data size of x. It does not matter if the data is sent in chunks of size m or $2m$ or larger. The time of transferring this data is $x/ChannelBandwidth$. In order to get a minimum start-up latency, Equation 1 should be minimised.

Analytic model Because of the limited physical bandwidth, and other latencies that were discussed in the previous sections, there is an optimum message size of

m bytes and the number of messages n that gives us the minimum latency. In this case we will have $x = mn$.

The application should pipeline the data x in the chunks of size m. The messages are sent to the ATM Adaptation Layer using Fore's API. AAL breaks down messages into 48 byte segments. Segments are sent to the ATM layer, and after adding a 5 byte header, they are given to the physical layer. On the destination node, cells are assembled and submitted to the application. The data flow from the source to the destination involves the steps (Figure 10) given below:

1. The user application calls the ATM API function *atm_send*, which makes a write system call. This, in turn, calls the *aal_send* aal procedure in the kernel, which hands the message to the ATM device driver's *drv_queue* routine that enqueues the message in the device transmission queue.

2. The card picks up the packet from the queue, and adds the AAL5 trailer and segments the AAL5 frame into ATM cells. Then, cells are transmitted to the destination node.

3. On the destination end, the card picks up the cells and enqueues them. Then, the AAL5 trailer is checked and the required functions for detecting errors are performed by the card. Furthermore, the packets are placed in a per VCI queue at the device driver.

4. Then, *aal_receive* picks up the packet from the per-VCI queue and after checking enqueues the packet. Finally, the application receives the message by calling *atm_receive*, which makes a read system call. This routine copies the data from kernel space to user space calling (calling *aal_receive*).

The latency of the data flow from the source to the destination can be divided into two major categories: per-byte latency, and per-message latency. We model the latency of the first step as:

$$t_1 = a_1 * m + b_1 \tag{2}$$

where a_1 is the per-byte latency, and b_1 is the per-message latency. Similarly, the latency of the second step can be modeled as:

$$t_2 = a_2 * m + b_2 \tag{3}$$

The third and fourth steps (receiving steps) have almost the same latency as the first and second steps.

Let us return to the pipelining schemes of our model. The latency of pipelining n messages of size m in the sending node, using Equation 1 will be :

$$Latency_{protocol} = t_1 + t_2 + (n-1) * \mathrm{MAX}(t_1, t_2)$$

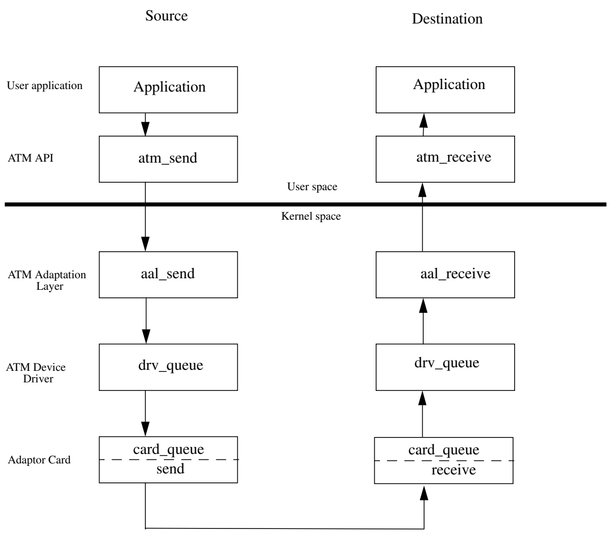

Figure 10: Data flow from source to destination.

To achieve a minimum protocol latency by message size, we should compute the result of: $d\ Latency_{protocol}/dm = 0$. Assuming that $t_2 > t_1$, we will have:

$$\frac{d\ Latency_{protocol}}{dm} = \frac{d\ \left(\sum_{k=1}^{2} (a_k * m + b_k) + (n-1) * (a_2 * m + b_2)\right)}{dm} \tag{4}$$

Substituting $n = x/m$ in the Equation 4, we get:

$$\begin{aligned} \frac{d\ Latency_{protocol}}{dm} &= \frac{d\ \left(\sum_{k=1}^{2}(a_k*m+b_k)+a_2*x-a_2*m+\frac{b_2*x}{m}-b_2\right)}{dm} \\ &= a_1 - \frac{(b_2*x)}{m^2} \\ &= 0 \end{aligned}$$

The minimum latency occurs when:

$$m = \sqrt{\frac{b_2 * x}{a_1}} \tag{5}$$

Equation 5 shows that the optimum message size is of $O(\sqrt{x})$. It is related to the square root of the ratio of the maximum per-message latency to the total per-byte latencies of the protocol software. Using this message size, the achieved protocol latency at the sending node is:

$$\begin{aligned} Latency_{protocol} &= a_1 * \sqrt{\frac{b_2*x}{a_1}} + b_1 + a_2 * \sqrt{\frac{b_2*x}{a_1}} + b_2 + a_2 * x - a_2 * \sqrt{\frac{b_2*x}{a_1}} + \frac{b_2*x}{\sqrt{\frac{b_2*x}{a_1}}} - b_2 \\ &= 2 * \sqrt{a_1 * b_2 * x} + a_2 * x + b_1 \end{aligned} \tag{6}$$

There is almost the same protocol latency at the receiving end. Therefore, the protocol software latency of sending and receiving data would be:

$$\begin{aligned} Latency_{sndrcv} &= 2 \times Latency_{protocol} \\ &= 2 * (2 * \sqrt{a_1 * b_2 * x} + a_2 * x + b_1) \end{aligned} \tag{7}$$

where, $Latency_{sndrcv}$ is the latency that the sending and receiving end-points are involved with. This latency is of $O(x)$.

Using the model There has been much work measuring the throughput and latency of ATM networks in different environments. Most of the experiments use the TCP/IP protocol. Keeton, Anderson, and Patterson [23] have performed experiments on different hardware and software environments. We will use one of their environments to validate our analytic model. Its parameters are shown in Table 2.

Host workstation	50 MHz Sun SparcStation 20s running Solaris 2.4
Network specification	ATM, using Fore SBA-200 Network Interface (NI) card, and ASX-200 Switch
Link bandwidth	155 Mbit/sec

Table 2: The analytic environment

Keeton, Anderson, and Patterson [23] have measured per-message latency, and per-byte latency for the protocol software and network interface (NI) environment in Table 2. These measurements are shown in Table 3.

	per-byte (μs)	per-message (μs)
protocol software and device driver	0.0375	151
NI	0.0425	200

Table 3: Per-byte and per-message latencies for Sun Sparc-20 workstations connected by ATM

Using this data together with the Equations 2 and 3, we have:

$$t_1 = 151 + 0.0375 * m$$

and

$$t_2 = 200 + 0.0425 * m$$

It is clear that $t2 > t1$ so, substituting these values in the Equation 1, we have:

$$Latency_{sndrvc} = 302 + 0.075m + 400 + 0.085m + (n-1) * (400 + 0.085m) \quad (8)$$

Using Equation 5, the optimal message size is:

$$\begin{aligned} m &= \sqrt{\tfrac{200*x}{0.0375}} \\ &\approx 73\sqrt{x} \end{aligned} \quad (9)$$

This gives the optimum message size (Equation 9) to minimise the latency of the protocol software and network interface costs (Equation 8).

The problem is not as simple as this computation. The link bandwidth is also an important factor that has to be considered. We cannot send a message faster than the available bandwidth.

Let us consider the available bandwidth which was calculated in Subsection 3.5. The *unit-time* (time to transmit one unit of data) of protocol software cannot be less than the unit-time of physical link available to the application layer, which is given below:

$$\begin{aligned} UnitTime_{link} &= \tfrac{8}{135.632}\ \mu\text{sec/byte} \\ &\approx 0.06\ \mu\text{sec/byte} \end{aligned}$$

Moreover we have :

$$\begin{aligned} UnitTime_{protocol} &= \tfrac{Latency_{protocol}}{x} \\ &= \tfrac{2*\sqrt{a_1*b_2*x}+a_2*x+b_1}{x} \end{aligned} \quad (10)$$

This cannot be less than 0.06; otherwise it causes link overflow. Let us again consider the latencies shown in Table 3. We have, $a_1 = 0.0375\ \mu sec$, $b_1 = 151\ \mu sec$, $a_2 = 0.0425\ \mu sec$, and $b_2 = 200\ \mu sec$. Figure 11 shows the unit-time values (Equation 10) for the range of data sizes from 1,000 bytes to 1,000,000 bytes. For the small size of messages, the unit-time is effected mostly by the per-message cost. As the size of messages becomes larger, the per-byte cost becomes a more important factor for the cost of unit-time.

This graph crosses the 0.06 line at a data size of almost 114,600 bytes. Considering this data size, the achieved unit-time would be minimised. The required message size for this size of data, using Equation 5, would be:

$$m = \sqrt{\frac{b_2 * x}{a_1}} = 24,700 \text{ bytes}$$

Figure 12 shows the lowest latency of 6,870 μsec, which is achieved with the message size of 24,700 bytes. This value is the theoretical value for the message size without considering any other traffic, so obviously this value is an overestimation of the message size.

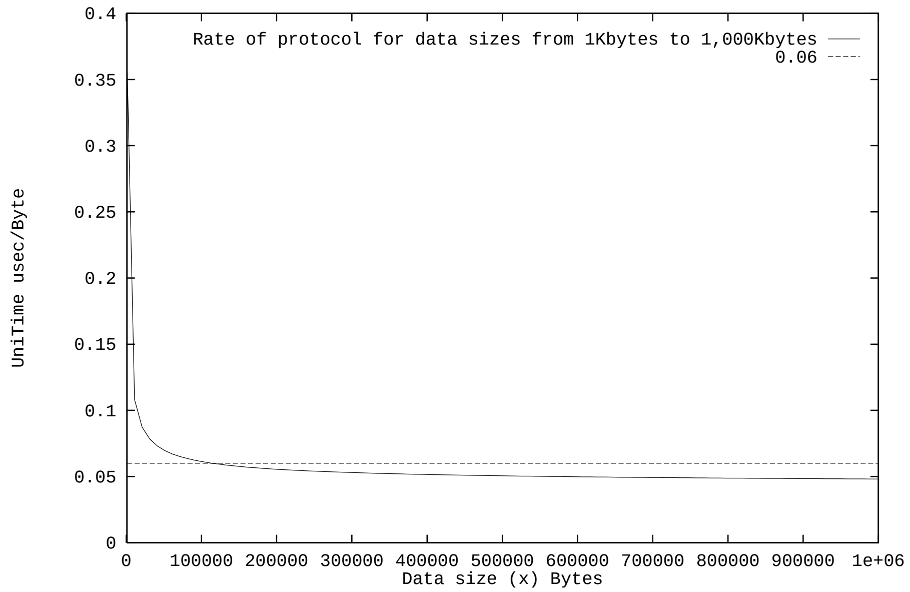

Figure 11: The $UnitTime_{protocol}$ for different data sizes from 1 Kbytes to $1,000$ Kbytes

3.7 Latency considering network congestion

The estimated latency in theory is different from the experimental results because of other traffic in the network. So far, we have estimated the latency of communication between two hosts without any intermediate switching nodes or any other traffic, but what is the latency and optimal message size when a message has to pass through several switches with other traffic present in the network at the same time?

Pipelining the cells through virtual circuits reduces the effect of path length in the cost of communication. But, there is another issue that increases the latency, which is congestion. Congestion happens whenever the input rate is greater than the available link capacity. It means that :

$$\sum(InputRate) > Available\ link\ capacity$$

Many congestion control methods exist. One of the most common methods to divide the bandwidth fairly among several sources is *max-min* allocation [19]. This method is defined as follows: suppose there are n sources, and the ith source gets a bandwidth of x_i. The allocation vector $x_1, x_2, ..., x_n$ is feasible if all link load levels are less than or equal to 100%. The total number of these vectors is infinite, but for each of them the source that is getting the least allocation is the unhappiest source.

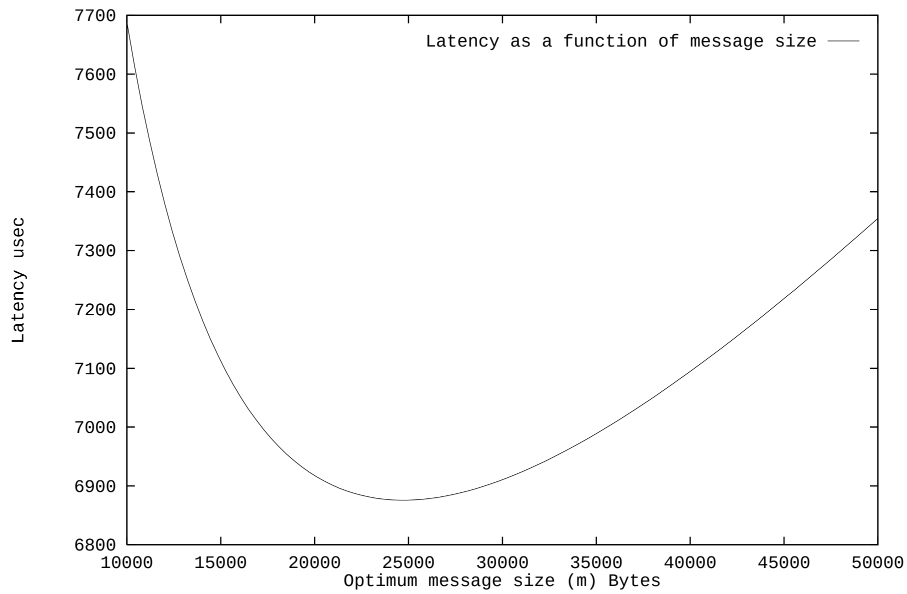

Figure 12: Minimum latency based on a range of message sizes and the data size of 114,600 bytes

Among these allocation vectors, we want to find the one which gives each source the maximum possible bandwidth. In the environment that we are using, the physical bandwidth is 155 Mbit/sec from which we get a bandwidth of 135.632 Mbit/sec available for the application layer. Based on the number of the processors which use this bandwidth, and taking advantage of max-min algorithm, we can assign a fair allocation vector.

3.8 ATM switches and their effect on latency

In this section, we consider intermediate switches and their effect on our model. We consider an ATM network with any reasonable topology. Moreover, we consider *total-exchange* communication, because it is often used for synchronisation operations. In a total-exchange, every processor sends/receives different messages to/from all the others.

Consider p processors connected to a cluster of switches. Furthermore, assume a total exchange in which each communication pair has its own VC. In this case there are $p * p$ VCs for sending data through switches.

The total throughput is limited by the busiest switches and links. The peak cell rate should be chosen based on the rates available through these bottlenecks.

The busiest part of the network in total exchange is shown in Figure 13. If we cut the network into two subnetworks with equal nodes on each side, and with the minimum possible number of links are placed on the cut-line, then these links are the bottlenecks. The reason is that these links have to tolerate $p/2 * p/2$ bidirectional communication, which is the worst case scenario. The number of physical links that cross the cut-line is the *bisection-width* of the network.

Most of today's ATM switch interconnection topologies with N switches have $bisection\text{-}width = \sqrt{N}$ [11]. If we assign $bisection\text{-}width = \sqrt{N}$, then we have :

$$VcBandwidth_{cutline} = \frac{\sqrt{N} * C}{\frac{p^2}{4}} \tag{11}$$

for $VcBandwidth_{cutline}$, the assigned bandwidth per VC.

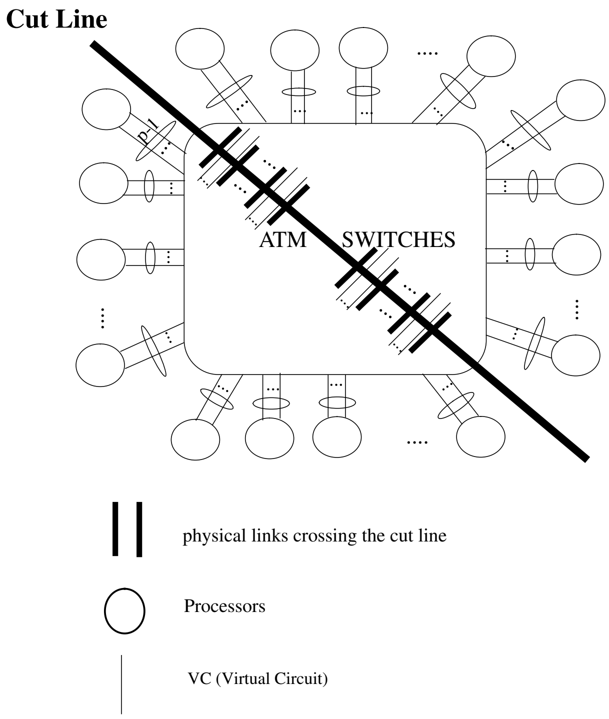

Figure 13: The location of bottlenecks in total exchange

So far, we have considered the network-network bottlenecks. We now consider the bottleneck at the host interface to the network links. Each host sends $p-1$ messages to the other processors and receives $p-1$ messages from them, so we will have $2(p-1)$ VCs and $2L$ physical links from host interface to the network. L is the number of physical links for each direction. From these $2L$ links, L links are used for sending data to the network, and the rest are for receiving data from the network. The reason for separating sending links from receiving ones, is that fibre optics, which are used in ATM, cannot be bidirectional. Usually, $L = 1$ [20, 22, 25, 38]. We also consider $L = 1$ for our analytic model, but it can be replaced with the desired number of links based on the available environment. If each of these $2L$ links has a bandwidth of C' Mbps, then the bandwidth available for e

$$VCBandwidth_{interface} = \frac{L * C'}{(p-1)} \tag{12}$$

Using the *max-min* algorithm we will assign each VC a bandwidth of:

$$VCBandwidth = \text{Min}\left(\frac{\sqrt{N} * C}{\frac{p}{2} * \frac{p}{2}}, \frac{C'}{(p-1)}\right) \tag{13}$$

We have to compute the optimum message size (m bytes) that can be sent in order to get the minimum latency and optimum use of the available bandwidth.

3.9 Optimum message size in total-exchange

The latency of the protocol software was computed in Subsection 3.6. We now compute the protocol *unit-time*, in which a unit of data (1 byte) can be sent to the VCs. This unit-time would be:

$$\begin{aligned} UnitTime_{protocol} &= \frac{Latency_{protocol}}{x} \\ &= \frac{a_2 * x + 2 * \sqrt{a_1 * b_2 * x} + b_1}{x} \end{aligned}$$

This unit-time cannot be smaller than the $UnitTime_{vc} = 1/VCBandwidth$, which was analysed in Subsection 3.5, otherwise the communication link will be faced with data overflow and message loss problems. Therefore, we have :

$$a_2 * x + 2 * \sqrt{a_1 * b_2 * x} + b_1 \geq UnitTime_{vc} * x \tag{14}$$

Clearly, the best situation is when $UnitTime_{protocol} = UnitTime_{vc}$. For the environment that we are analysing, with the latencies that are shown in Tables 2 and 3, the $UnitTime_{vc} \geq a_2$. Thus, we can say:

$$(UnitTime_{vc} - a_2) * x - 2\sqrt{a_1 * b_2 * x} - b_1 \leq 0 \tag{15}$$

For simplicity, we write $UnitTime_{vc}$ as T in the rest of this discussion. Solving this inequality, we have:

$$\sqrt{x} \leq \frac{\sqrt{a_1 b_2} + \sqrt{a_1 b_2 + b_1 (T - a_2)}}{T - a_2} \tag{16}$$

where x is the largest size of data that the end point can transmit in order to get the highest throughput without causing a congestion problem in the network. This equation has one condition, namely $x > \frac{b_2}{a_1}$, in order to allow pipelining. If this condition is not satisfied, then message size and data size would be equal. From previous discussions we have:

$$T = \mathrm{MAX}\left((p-1) * UnitTime_{link}, \frac{p^2}{4\sqrt{N}} * UnitTime_{link}\right) \tag{17}$$

In order to compare our results with other researchers' experiments [18], let us assume that the $bisection\_width = 2$. Based on the variety number of switches and processors, the available unit-time for each VC would be different.

Let us consider $p = 8$, and the physical bandwidth is 155 Mbps, which means that $C = 155$ Mbps and $C' = 155$ Mbps. Then, each VC unit-time is:

$$\begin{aligned} T &= \frac{p^2}{4\sqrt{N}} * UnitTime_{link} \\ &= \frac{64}{8} * 0.06 = 0.48\ \mu\text{sec/byte} \end{aligned}$$

Figure 14 shows a data size of 670 bytes from which we get a unit-time of 0.48 $\mu sec/byte$. The optimal message size is shown in Figure 15, which is 670 bytes, considering data size of 670 bytes. The reason that we chose a message with size equal to the data size is that the optimal data size that each processor can send is smaller than $\sqrt{b_2/a_1}$. Therefore, the message size is the same as the data size. The theoretical latency with this message size would be $Latency_{protocol} = 404\ \mu$sec for one end point. Total-exchange consists of $2(p-1)$ protocol latencies, because each source-end sends/receives $(p-1)$ messages to/from the others. Therefore, the latency of total-exchange would be:

$$\begin{aligned} Latency_{tot\_exchange} &= 2(p-1)Latency_{protocol} \\ &= 2 * 7 * 404 \\ &= 5,656\ \mu\text{sec} \end{aligned}$$

This result is in line with the results of Hung, Kasten, and McKinley in their experiments over the topology that is shown in Figure 16 [17, 18]. Table 4 shows the results of our analytic model, which are computed using Equation 1 and the above discussion, compared to experimental results with a variety number of processors and data sizes.

3.10 Multicast VCs using API

So far we have used a separate addressing approach for total-exchange. With the appearance of ATM switches that can handle multiple point services in hardware,

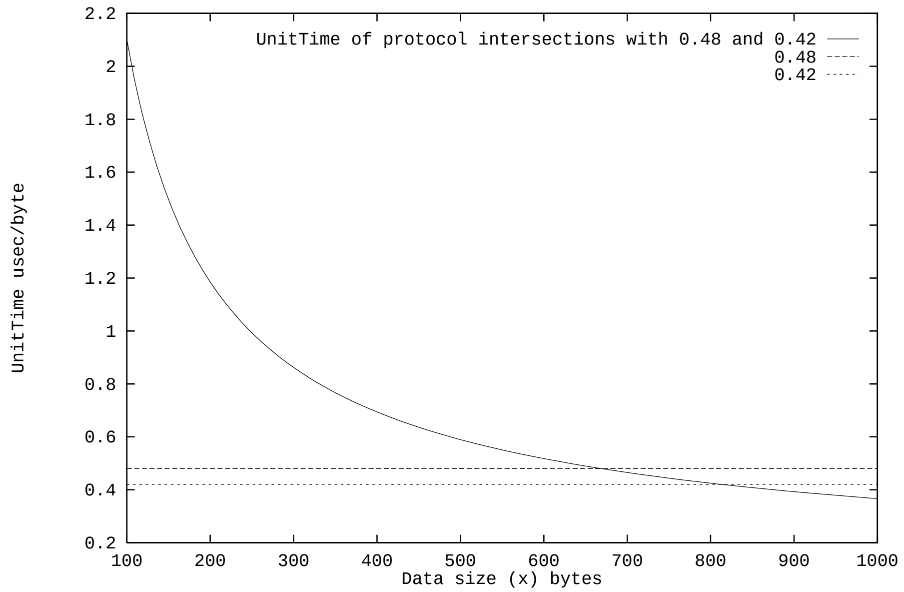

Figure 14: The $UnitTime_{protocol}$ with 8 processors sending a variety of data sizes up to 1,000 Kbytes through the VCs.

the latency of collective operations can potentially be decreased. The ATM User Network Interface (UNI) standard supports VCs with multiple destinations, or multicast VCs. The API system calls provide us with a way of using this hardware implementation of multicast. In this case, there is no need to establish different VCs to send n copies of data to n multiple end-points. The application can establish multicast VCs and send one copy of data to the switch. Data is then replicated in the switch and sent through multiple outputs to traverse the path to the destination nodes. This approach improves performance for collective operations.

Let us take a look at the way that API creates multicast VCs and how the switch handles multiple end-points. The Fore Systems' user-level ATM library routines (API) communicate with the ATM device driver. End systems and switches are identified by service access points. An application is referred to as an ATM end-point and has a unique address called application service access point (ASAP). Each ATM end-point is attached to a particular ATM switch port. We refer to a switch-id and port number as network service access point (NSAP). An NSAP is used with API functions. When an application starts, it uses *atm_open()* system call to open the specified device. Using this system call, the device driver assigns a file descriptor to the specified ASAP and NSAP. The *atm_open()* system call returns a file descriptor to be used for establishing the connections in the later request. After

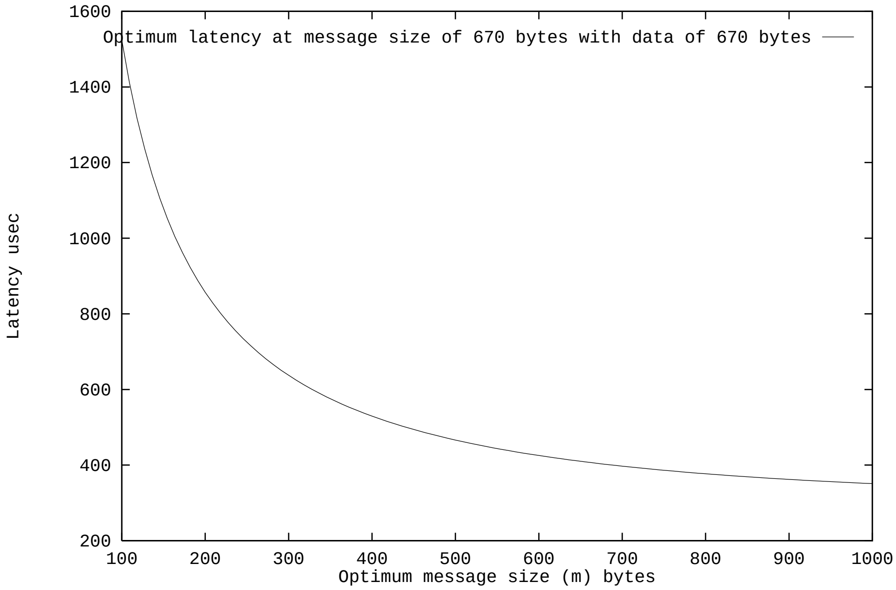

Figure 15: Minimum latency achieved with 8 processors and data size of 670 bytes

opening the file descriptor, an application asks the device driver to bind its ASAP and NSAP to the file descriptor.

In the case of multicasting, the device driver associates the VPIs and VCIs with the source fd during the connect and accept system calls. This multicast data flow provides the facility of sending one copy of a piece of data to multiple destinations instead of sending several copies to each endpoint. Replication of data is done in the ATM switches using hardware.

In the previous section, we used separate addressing in which the application has to make several copies of data to send to different destinations. Separate addressing increases overheads and therefore causes higher latency. Figure 17 shows the difference between these two approaches and the way that ATM switches can handle the multicast communication. In Figure 17 (a), $L1$ physical link bandwidth should be divided into 4 VCs, but in the Figure 17 (b) all the physical bandwidth is assigned to one VC.

3.11 Latency using the multicast VCs

Again, we consider a network of interconnected switches with any topology. If we cut this network into two subnetworks with an equal number of end-points while the smallest number of links are crossing the cut-line, we will have $|bisection\text{-}width|$

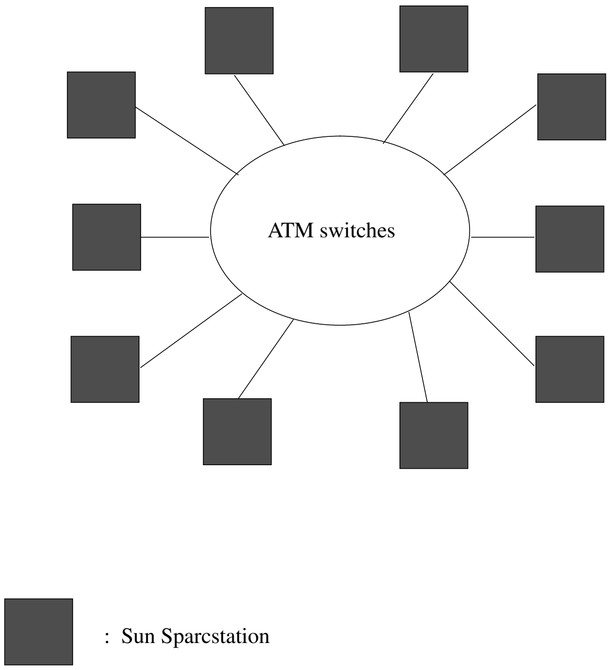

Figure 16: Cluster of Sun Sparc-20 Stations over an ATM network.

physical links crossing the cut-line. Most ATM networks have $|bisection\text{-}width| = \lceil\sqrt{N}\rceil$ in their interconnection topologies. p/p multicasting (p processes send/receive messages to/from all of the others) over these types of networks using the ATM hardware facility, provides us with a tree structure of replicating data in each switch in the path to various destinations. As before, the worst point of communication is on the cut-line because $p/2$ of end-points send data to the other $p/2$ end-points. In this case (hardware multicasting) the user network interface links need to be divided into p VCs, one for sending data to $(p-1)$ d

$$VCBandwidth_{send} = \frac{C}{1} \tag{18}$$

$$VCBandwidth_{receive} = \frac{C}{p-1} \tag{19}$$

Equations 18 and 19 show the per-VC bandwidth in the links between the end-points and the switches. Moreover, the busiest switch-switch links are those on the cut-line. These physical links have to tolerate the traffic of $\frac{p}{2}\sqrt{N}$, because there are $\sqrt{N}$ edges crossing the cut-line. In the worst case scenario, each switch that is placed across the cutline should replicate $p/2$ outgoing messages. Therefore, we will

Latency (μsec)						
Data size (Kbyte)	4 Processors		6 Processors		8 Processors	
	Analytic	Experiment	Analytic	Experiment	Analytic	Experiment
1	3,800	5,000	6,320	7,000	8,848	9,000
4	9,260	12,500	15,440	16,000	21,616	22,000
16	33,890	34,000	56,480	57,000	79,070	80,000
32	66,720	68,000	111,120	112,000	155,000	160,000

Table 4: Comparison of the achieved latencies of total-exchange between the analytic model and the experimental ([HMK94]) results of McKinley, Huang, and Kasten on the configuration shown in Figure 16, using the separate addressing approach

have a per VC bandwidth of:

$$VCBandwidth_{cutline} = \frac{\sqrt{N}C'}{\frac{p}{2}\sqrt{N}} \tag{20}$$

Again we have to choose the minimum value of the three Equations 18, 19, and 20 to assign to the VCs. Therefore, we have:

$$VCBandwidth = \text{Min}\left(C, \frac{C}{p-1}, \frac{2C'}{p}\right) \tag{21}$$

Now, let us take a look at the latency of this model. In our environment Table 2, Equation 21 becomes:

$$VCBandwidth = \frac{C}{p-1} Mbit/sec$$

Assuming $p = 8$, the optimal data size and message size for this communication according to the previous values of interface latencies and system calls will be almost: $data\_size = 814$ bytes, and $message\_size = 814$ bytes. This is shown in Figure 14. The latency of one VC would be 416 μsec for an end-point. Moreover, there are p VCs on each end-point, so the total latency will be:

$$\begin{aligned} Latency_{tot\_exchange} &= p * Latency_{protocol} \\ &= 8 * 416 \\ &= 3,328 \mu sec \end{aligned}$$

This latency is in line with the results that Hung, Kasten, and McKinley have given. Table 5 shows a comparison of our results with experimental results. Our results are computed using Equation 1.

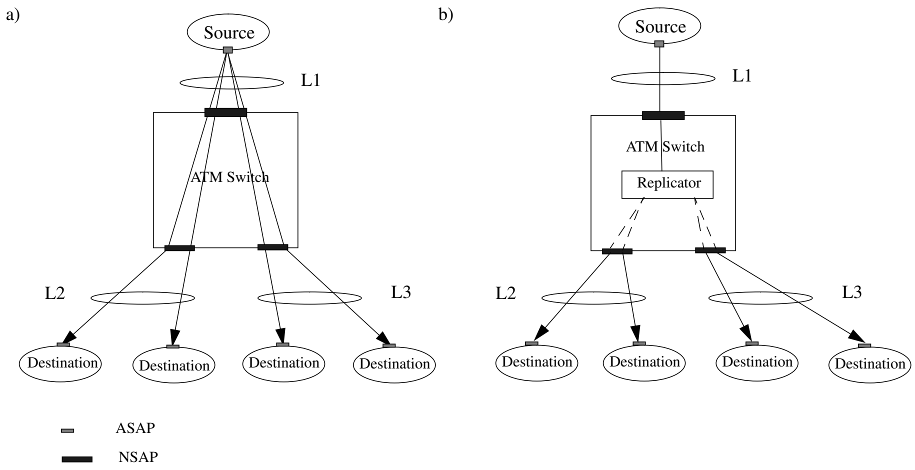

Figure 17: a) Multicast using separate addressing in the application end-point b) Multicast using ATM switch hardware multi-point services.

Data size (Kbyte)	8 Processors	
	Analytic	Experiment
1	5,208	14,000
4	10,848	16,000
16	39,048	41,000
32	76,648	77,000

Table 5: Comparison of the achieved latencies of total-exchange between the analytic model and the experimental ([HMK94]) results of McKinley, Huang, and Kasten on the configuration shown in Figure 16, using the hardware-multicast approach

When comparing the results of two different approaches for multicasting (Tables 4 and 5), we see that the latency of hardware approach is about half of that of separate addressing.

4 Analysing BSP Parameters over ATM Networks

The main purpose of this section is to provide an analytic model for predicting g and l parameters over ATM networks. The g parameter, is based on two algorithms. These are: *total-exchange*, and *one-relation* communications. We predict the value of l using broadcast and total-exchange algorithms. We use this analysis to estimate g and l for a shared-memory Sun Sparc-20 multiprocessor and an IBM SP2.

4.1 BSP communication and synchronisation routines

For global communication, processors copy the data that should be sent, first to library space (which may be in the kernel space of the same end point), and then through the network (Figure 18).

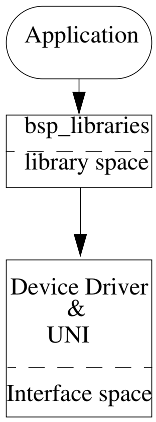

Figure 18: Steps that are involved in the *BSP* model for networks of workstations

This process takes effect by first copying data to library space, which can be in user space or kernel space (using *bsp_put()*, a BSP library routine to put local data into the memory of target processor). Then, during synchronisation (*bsp_sync*(), a BSP library routine for barrier synchronisation), the data is copied to device driver space from library space. During synchronisation, *atm_send()* is called in order to send the data to the other end and the other end retrieves it by calling the *atm_receive()* routine. Therefore, there are two copying processes at each end-point, one using library routines to copy from user space to library space, and the other utilising library routines to copy from library space to device driver space. If we want to trace the latency of the process, we see that the application uses the *bsp_put()* routine to copy the data to the library space. This involves the cost of a system call and a data copy. If we show the cost of a system call as b' ($b' = 40$ μsec for a shared-memory Sun Sparc-20 [7]) and the cost of data copy as a' ($a' = 0.05$ μsec for a Sun Sparc-20 [7]), we will have: cost of bsp_put(m) $= a'm + b'$. As Hill and Skillicorn [14] show, collecting data and sending it to the network in large pieces increases efficiency.

Therefore, assume that there are k messages to be transmitted, followed by a synchronisation. In this case, we will have:
$bsp\_put(m_1) = a'm_1 + b'$
$bsp\_put(m_2) = a'm_2 + b'$
.

.

.

$bsp\_put(m_k) = a'm_k + b'$

$- -$

The total cost of this part is $a' \sum_{i=1}^{k} m_i + kb'$
where $\sum_{i=1}^{k} m_i = x$. Therefore,

$$The\ total\ cost\ of\ communication\ before\ synchronisation = a'x + kb'$$

where x is the total data size m_1 to m_k are the size of the messages that are sent, and k is the number of *bsp_put* calls.

Barrier synchronisation takes place by calling $bsp\_sync()$. In this routine, $atm\_send()$ is called to send messages through VCs. Let us take a look at the cost of *bsp_sync*. Assume m is the optimal message size to be sent across ATM networks, n is the number of messages to be sent through VCs, and $x = mn$. The cost of synchronisation is as follows:

$cost\ of\ bsp\_sync()\ call = b'$

$pipelining\ n\ messages\ of\ size\ m$

$- -$

$atm\_send(m) = a_1m + b_1 + a_2m + b_2$

$atm\_send(m) = a_1m + b_1 + a_2m + b_2$

.

.

.

$atm\_send(m) = a_1m + b_1 + a_2m + b_2$

$- -$

The total cost of this part $= \sum_{i=1}^{2} (a_im + b_i) + (n-1)\, \mathrm{MAX}_{\mathrm{i}=1}^{2} (a_im + b_i) + b'$.

The other end receives data using *bsp_get()* and *bsp_sync()* routines. Therefore, we will have the same cost at the receiving node as at the sending node. Now, let us compute the values of g and l using the discussion of the previous section regarding optimal message size.

4.2 Analysing g over ATM networks

The optimal data size and message size that were computed in the previous section provide us with a minimum latency communication. We want to compute the values of g for our BSP model over ATM. Using the information from the previous section, we have Equation 8 which gives the protocol latency of sending data through a VC and receiving it from the VC at the other end. At present, there is no clear framework for the costs that are involved in evaluating g. We have assumed the g

value to be the sum of three latencies, the protocol latency to send a unit of data, $Latency_{sndrcv}$; and the latency of $bsp\_put()$ routines for a data unit.

g is usually computed using a heavily-loaded communication network. Therefore, the time of transferring the cells through the VCs is short compared to protocol processing, and most probably is overlapped with it. Therefore we have:

$$\begin{aligned} g &= \left(Latency_{sndrcv} + Cost\ of\ bsp\_put() + Cost\ of\ bsp\_get()\right)/x \\ &= \left(2\left(a'x + b'k + a_1 m + b_1 + a_2 m + b_2 + (n-1)\left(a_2 m + b_2\right)\right)\right)/x\ \mu\text{sec/word} \end{aligned} \tag{22}$$

Substituting the optimum value of m, computed in the previous section, into this equation, we have :

$$g_{optimum} = \left(2\left(\left(a' + a_2\right)x + 2\sqrt{a_1 b_2 x} + kb' + b_1\right)\right)/x \tag{23}$$

We consider two algorithms, which give us measurements of lower-bound and upper-bound values of the g parameter. The first algorithm is a *1-relation* communication as shown in the Figure 19(a), and the second one is a *p-relation* communication provided by *total_exchange* (Figure 19(b)). We consider a network topology

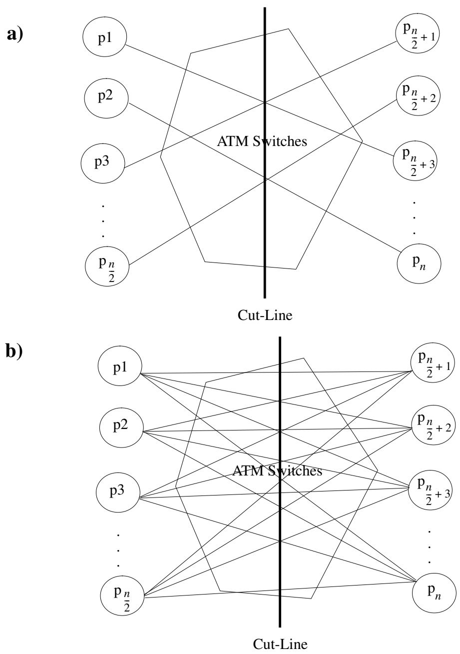

Figure 19: a) A 1-relation communication through cut-line switches. b)A p-relation communication using total-exchange through cut-line switches.

of 4 switches and 8 processors, as shown in Figure 20, and compare our results with the experimental results for an IBM SP2 computer with the same topology network and Sun Sparc-20 shared-memory architecture [13] with 8 processors.

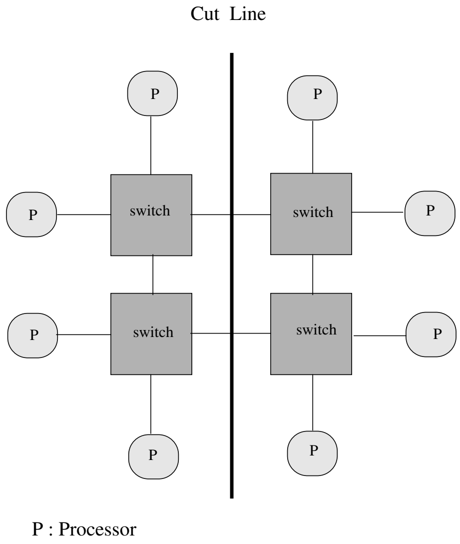

Figure 20: Cluster of 8 processors with 4 switches connected to each other

4.3 Analysing the value of g using a 1-relation communication

Let us consider the first algorithm and compute the value of g. As before, we assume the link capacity of C and C' in switch-switch and host-network links. The latency of the protocol stack and operating system, before submitting data to the VCs and handling it at the other end, is $2\left(\sum_{i=1}^{2}(a_i m + b_i) + (n-1)\mathrm{MAX}_{i=1}^{2}(a_i m + b_i)\right)$. The number of VCs crossing the cut-line is $p/2$ in each direction, as shown in Figure 19.a. Depending on the bisection_width, the $VCBandwidth$ is:

$$VCBandwidth = \mathrm{Min}\left(\frac{bisection\_width * C}{p/2}, \frac{C'}{1}\right)$$

Using the environment that we are analysing and shown in Figure 20, we have: $VCBandwidth = \frac{2*C}{8/2}$ Mbps for our network topology. Therefore, the unit-time of each VC would be:

$$\begin{aligned} T &= \tfrac{p}{2*bisection\_width} * UnitTime_{link} \\ &= 2 * 0.06 \\ &= 0.12\ \mu\text{sec/byte} \end{aligned}$$

Therefore we have:

$$\frac{Latency_{protocol}}{x} \geq 0.12\ \mu\text{sec/byte} \quad (24)$$

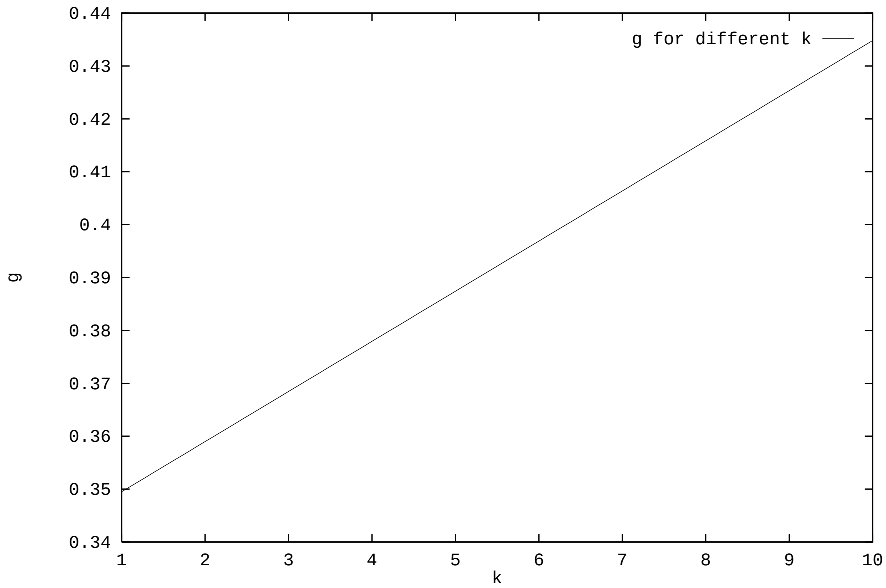

Figure 21: Corresponding g values for different number of *bsp_put()* system calls in a 1-relation communication with 8 processors

The optimal data size for this unit-time is $8,440$ bytes, and the message size is $6,700$ bytes.

Substituting this message size and data size into Equation 23, the value of g is:

$$\begin{aligned} g_{optimum} &= \left(2\left((0.05+0.0425)\,8440 + 2\sqrt{0.0375*200*8440} + 40k + 151\right)\right)/8440 \\ &\approx 0.35\ \mu\text{sec/byte} \end{aligned}$$

Figure 21 shows the value of g for a 1-relation communication. This value is 0.35 μsec/byte or $0.35 * 4 = 1.4$ μsec/word for Sun Sparc-20 over an ATM network. In order to compare our results with experimental results of Hill, we assume $k = 1$ in our discussion.

Table 6 shows the clock rates of Sun Sparc-20 and IBM SP2 [13]. The Sun

Machine	Clock rate (Mflop/s)
Sun Sparc-20	10
IBM SP2	26

Table 6: Clock rates of Sun Sparc-20 and IBM SP2

Sparc-20 has a clock rate of 10 Mflop/sec, and therefore g is:

$$g = 10 * 1.4 = 14 \text{ flop/word}$$

The IBM SP2, connected by the manufacturer's switch, has a clock rate 2.6 times faster than Sun Sparc [13]. Therefore, we can estimate a g value of $g = 14/2.6 = 5.38$ flop/word or $g = 5.3/26 = 0.20$ μsec/word for IBM SP2 over *ATM* using *BSP*. Now, let us compare our analytical results of g with the experimental results of Hill [13]. Hill has used Sun Sparc-20 shared-memory architecture in his experiments. Table 7 shows this comparison.

g values					
Machine	Analytic (over ATM)		Experiment (without ATM)		
	flop/word	μsec/word	flop/word		μsec/word
SM Sun Sparc-20	14	1.4	3.3		0.33
IBM SP2	5.3	0.2	6.9	switch	0.27
IBM SP2	-	-	1246	Ethernet	47.3

Table 7: Values of g for both analytic model and experiment, using a one-relation communication

As shown in this table, little improvement is observed for g using one-relation communication on an IBM SP2 over an ATM compared to IBM SP2 (switch). However, the results show a lot of improvement compared to IBM SP2 over Ethernet. Comparing the g values of shared-memory Sun Sparc-20 with an ATM interconnect shows a communication performance degradation compared to its performance using its internal bus. This is not surprising given the protocol overhead of ATM.

We should take into consideration that this value of g is the lower-bound limit. The upper-bound value of g is computed using the busiest communication algorithms such as *p/p broadcast* and *total-exchange* communication across the cut-line switches. In the next section we consider a total-exchange communication.

4.4 Analysing the value of g using a total-exchange communication

Now, let us compute the value of g for a p/p total_exchange on the same network topology as in Subsection 4.3 and compare our results with other researchers' experiments [13]. Using the discussion of Subsection 3.9, we have: T=0.48 μsec/byte. The optimal data size (x) for this unit-time (Figure 22) would be 670 bytes, but the message size (m) that we can assign to get a unit-time close to T (Figure 23) is also 670 bytes. Considering this value of the message size and the data size, the value of

g for the shared-memory Sun Sparc-20 (using Equation 22) would be:

$$g = 1.42 \ \mu\text{sec/byte} \tag{25}$$

If we compute the g value in units of μsec/word, we have:

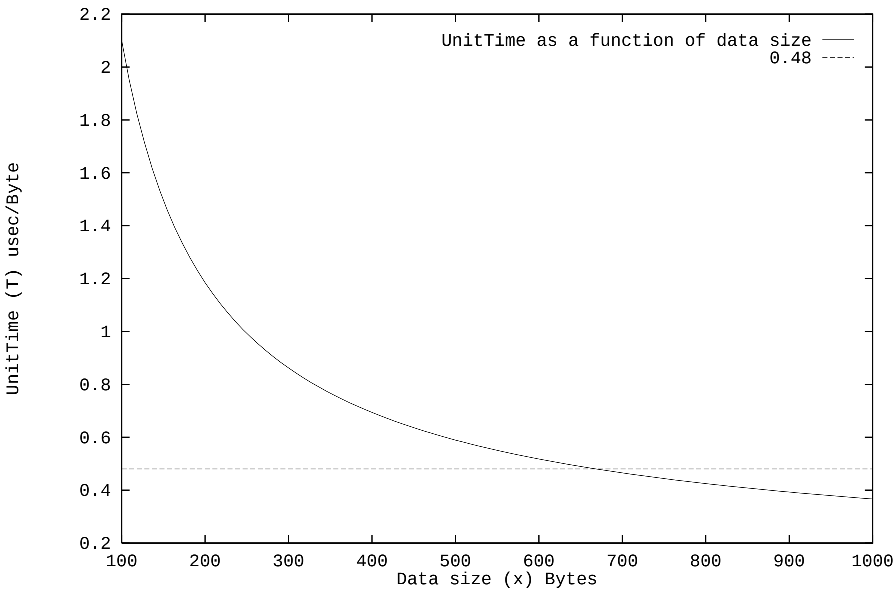

Figure 22: Corresponding data size for the unit-time of 0.48 μsec/byte in a 1-relation communication using *BSP* over *ATM*.

$$\begin{aligned} g &= 1.42 * 4 \\ &= 5.7 \ \mu\text{sec/word} \end{aligned} \tag{26}$$

Equation 26 shows the value of g for the Sun Sparc-20. In order to compare the results of our model with the experimental results of Hill, we consider the clock rates that Hill has measured for Sun Sparc-20 and IBM SP2 (Table 6), and compute the values of g for these two machines based on flop/word. Moreover, the g values that Hill achieves are the asymptotic communication cost for very large messages. In order to compare our results, we should compute this asymptotic values for our

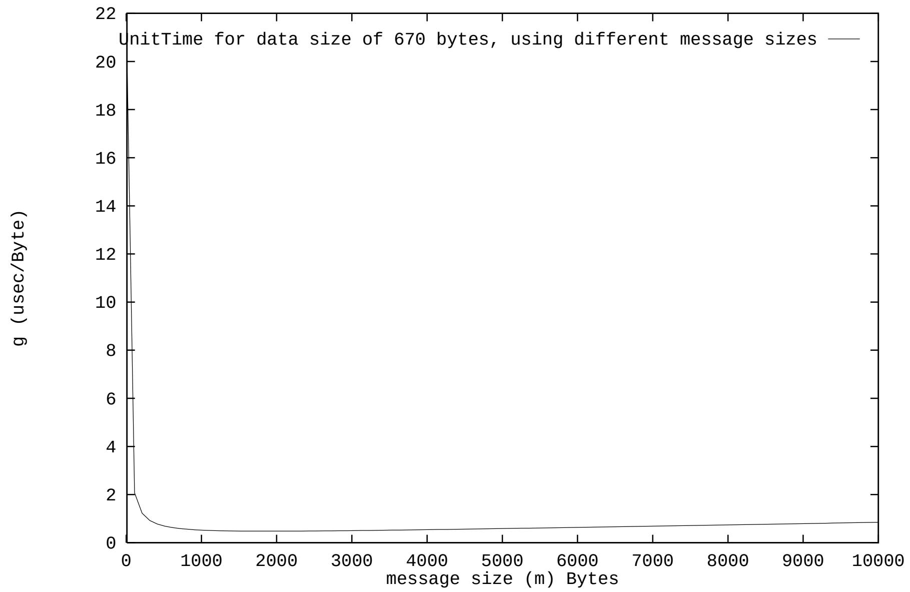

Figure 23: An optimal message size for ATM with data size of 670 bytes. g starts growing after this point, because per-byte cost will exceed the per-message cost

model. Therefore, we have:

$$\begin{aligned}
&\lim_{x\to\infty} g_{optimum} \\
&= \lim_{x\to\infty} 2\left(\left(a' + a_2\right)x + 2\sqrt{a_1 b_2 x} + kb' + b_1\right)/x \\
&= 2\left(a' + a_2\right) \\
&= 2\left(0.05 + 0.0425\right) \\
&= 0.18\ \mu\text{sec/byte} \\
&= 0.18 * 4\ \mu\text{sec/word} \\
&= 0.72\ \mu\text{sec/word} \\
&= 7.2\ \text{flop/word}
\end{aligned} \tag{27}$$

This value is for Sun Sparc-20. The clock rate of IBM SP2 is 2.6 times that for Sun Sparc-20. Therefore, the g value for IBM SP2 would be 2.77 flop/word or 0.11 μsec/word.

Comparing these g values with the experimental results of [13] for IBM SP2 (Table 8), shows an improvement of almost four times for g over an ATM network compared to one using the manufacturer's switch. This result, compared to IBM SP2 using Ethernet, shows an improvement of 440 times. No improvement is observed for Sun Sparc-20 over an ATM. This is not surprising since the Sun Sparc-20 used in this comparison is a shared-memory architecture.

g values					
Machine	Analytic (over ATM)		Experiment (without ATM)		
	flop/word	μsec/word	flop/word		μsec/word
SM Sun Sparc-20	7.2	0.72	4.1		0.41
IBM SP2	2.77	0.11	11.4	switch	0.43
IBM SP2	-	-	1224.1	Ethernet	47.1

Table 8: Values of g for both analytic model and experiment, using a total-exchange communication

4.5 Analysing l over ATM networks

The parameter l in the *BSP* model is the cost of synchronisation among p processors. This value depends mostly on the communication network, and the algorithm that is used for synchronisation. In this section, we will show two algorithms for synchronisation on the *BSP* model over an ATM network. The first algorithm uses *total_exchange* to synchronise processes at the end of a superstep, and the second algorithm uses a broadcast algorithm for this purpose. In the following discussion, we will explain these two algorithms, how they work, and the cost, l, of using each of them. Let us start with the *total_exchange* algorithm for barrier synchronisation.

4.6 Analysing the value of l using a total_exchange algorithm

In this algorithm, each process sends a separate message to others and informs them of the data size that it is going to send to each of them in that superstep. This information message is small in size (usually less than one hundred bytes). Total exchange for small messages is very expensive. Figure 24 shows this algorithm.

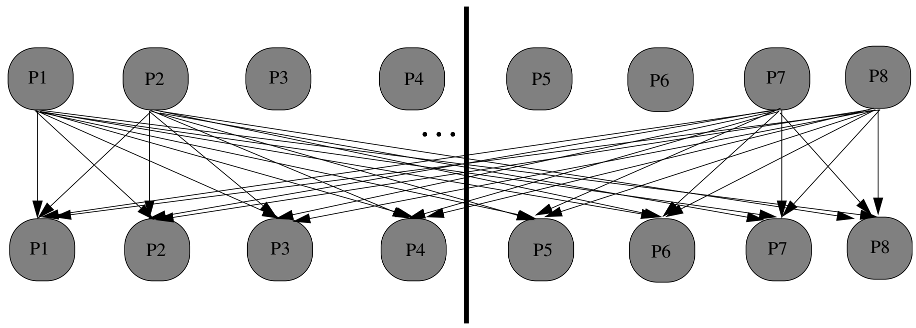

Figure 24: BSP model of total-exchange algorithm

Now let us consider our network configuration of p processors with N switches over ATM. The *VCBandwidth* assigned to each VC in a total-exchange communication is given by Equation 13. In order to compute the value of l for this algorithm, let us consider our previous network topology of 8 processors with 4 switches (Figure 20), and the environment of Table 2. We will compute the value of l for a BSP model over this network and compare it with an IBM SP2 (switch) with the same number of processors.

With our network, T = 0.48 μsec/byte (as computed before). Usually synchronisation messages are small in size (usually not more than 100 bytes). Let us compute the value of l for the message size of 100 bytes. Equation 28 shows the cost of l.

$$\begin{aligned} l &= Cost\ of\ bsp\_sync() + 2 * (p-1) * Latency_{protocol} \\ &+ Transfer\ time\ in\ the\ network \\ &= b' + 2\,(p-1)\,(a_1 m + b_1 + a_2 m + b_2) + \frac{x}{VCBandwidth} \end{aligned} \tag{28}$$

Each end-point sends $(p-1)$ separate messages and gets $(p-1)$ messages from the others. Based on the message size, we have:

$$l = 40 + 14 * (0.0375 * 100 + 151 + 0.0425 * 100 + 200) + 100 * 0.48 = 5114\mu\text{sec}$$

For the shared-memory Sun Sparc-20, with clock rate 10 Mflops, we have, $l = 51140$ flops. The IBM SP2 (switch) has a clock rate of 26 Mflop/sec or 2.6 times of that for a Sun Sparc-20. Therefore, we can see that the value of l for the IBM SP2 is $l = 19970$ flops. If we compare this value with the results of Hill [13], which is shown in Table 9, it shows that synchronisation over ATM takes 3.5 times longer than for an IBM SP2 (switch). The results show some improvement for the IBM SP2 using ATM compared to Ethernet (reduction of 4 times).

Now let us take a look at the second algorithm which uses broadcast communication for synchronisation.

l values					
Machine	Analytic (over ATM)		Experiment (without ATM)		
	flops	μsec	flops		μsec
SM Sun Sparc-20	51140	5114	118		11.7
IBM SP2	19970	768	5412	switch	208
IBM SP2	-	-	88795	Ethernet	3415

Table 9: Values of l for both analytic model and experiment using total-exchange

4.7 Analysing the value of l using a broadcast algorithm

Another way to synchronise is to send a list of all data sizes to all processes. Because the size of the synchronising message is small, combining all outgoing messages of a process in one list and using the hardware design of ATM to *broadcast* the combined message may help to reduce the latency of synchronisation operation.

Let us consider the broadcast algorithm and provide an analytic model for l. The *BSP* model of this algorithm is the same as Figure 24, but the cost model is different. In this algorithm, each processor sends a message of $m * p$ bytes instead of m bytes, because it combines p messages of size m bytes. Let us compute the value of l for the above equation.

As shown in Equation 21 the appropriate bandwidth to assign to a VC is $VCBandwidth = \frac{C}{p-1}$ Mbps for our network configuration. Therefore, the T value that is achieved with this bandwidth is $R = 0.42$ μsec/bytes. Therefore, the optimum message size to be sent through the network is $m = 814$ bytes. Moreover, each end-point needs to send one message and get $p-1$ messages from the others. Therefore, we have a latency of:

$$\begin{aligned} l &= \textit{Cost of bsp\_sync}() + 2 * (p-1) * Latency_{protocol} + \textit{Transfer time in the network} \\ &= b' + p\,(a_1 m + b_1 + a_2 m + b_2) + \frac{x}{VCBandwidth} \end{aligned} \tag{29}$$

Assuming the same configuration as before, we will have $l = 3,368$ μsec for a message size of 814 bytes. The messages that are used for synchronisation are small in size (a few hundred bytes). Therefore, for different message sizes from 100 bytes to 814 bytes, we will have different values for l.

In our network configuration, which consists of 8 processors and 4 switches, combining 8 messages of 100 bytes into one message, gives us a message of at most 800 bytes. The value of l for this message size is almost $3,000$ μsec (Equation 29). This value for a Sun Sparc with a clock rate of 10 Mflops/sec is $30,000$ flops, and for IBM SP2, is about 11500 flops. These values (Table 10) are much smaller than the ones computed using the total-exchange algorithm in which $l = 51140$ flops and $l = 19970$ flops. The reason is that setting up separate VCs for different destinations is costly, while sending one combined message and using ATM hardware replication to send the message to separate end-points saves a lot of time.

Implementing synchronisation using the broadcast algorithm has a lower synchronisation cost. However, comparing this l value with the one in [13], we will see a latency that is two times greater for IBM SP2 (switch), and 7 times less for IBM SP2 (Ethernet). It is clear that the value of l for IBM SP2 (switch) is still large.

l values					
Machine	Analytic (over ATM)		Experiment (without ATM)		
	flops	μsec	flops		μsec
SM Sun Sparc-20	30,000	3000	118	11.7	
IBM SP2	11500	442	5412	switch	208
IBM SP2	-	-	88795	Ethernet	3415

Table 10: Values of l for both analytic model and experiment using broadcast

5 Summary and Conclusions

The first contribution of this paper is to show that the performance of ATM networks can be modelled using simple analytic models. The agreement of the model presented here with measured results is impressive. This suggests that the performance of ATM networks behaves tractably, and highlights performance deficiences, primarily the dominance of operating system overheads.

The second contribution is to show, for the first time, how to obtain BSP parameters other than by benchmarking. This enables us to explore the likely effects of using difference implementation techniques for collective operations and barriers.

Overall the model predicts that g values for ATM-connected parallel computers are likely to be comparable with, or slightly better than, existing interconnection networks in terms of throughput, but rather worse in terms of latency.

References

[1] R. Ahuja, S. Keshav, and H. Saran. Design, implementation, and performance measurement of a native-mode ATM transport layer (extended version). *IEEE/ACM Transactions on Networking*, 4(4):502–515, Aug. 1996.

[2] E. Billard. Queueing networks. http://www-ci.u-aizu.ac.jp/Public-Lectures/English/billard.html, 1997.

[3] D. E. Comer and D. L. Stevens. *Internetworking with TCP/IP*, volume III. Prentice-Hall, Inc., 1993.

[4] E. A. Crawford. PVM: An introduction to parallel virtual machin. http://www.gatech.edu/hpc/HPC-Home/seminar/pvm.html , Oct. 1996.

[5] P. W. Dowd, T. M. Carrozzi, F. A. Pellegrino, and Chen A. X. Native ATM application programmer interface testbed for cluster-based computing. In *Proc. IPPS'96*, Apr. 1996. http://tebbit.eng.umd.edu/people/fap/fap-publication.html.

[6] P. W. Dowd, S. Srinidhi, and R. Claus. Issues in ATM support of high performance geographically distributed computing. In *Proc. IPPS'95 Workshop on High Speed Networks*, pages 352–358, Apr. 1995.

[7] T. V. Eicken, V. Avula, A. Basu, and V. Buch. Low-latency communication over ATM networks using active messages. *IEEE Micro*, Vol. 15(No. 1.):46–53, Feb. 1995.

[8] Fore Systems. *Fore system ATM API reference manual*, 1996.

[9] M. W. Goudreau, J. M. D. Hill, K. Lang, W. F McColl, S. D. Rao, D. C. Stefanescu, T. Suel, and T. Tsantilas. A proposal for a BSP worldwide standard. http://www.bsp-worldwide.org/, Apr. 1996.

[10] M.W. Goudreau, J.M.D. Hill, K. Lang, W.F. McColl, S.D. Rao, D.C. Stefanescu, T. Suel, and T. Tsantilas. A proposal for a BSP Worldwide standard. BSP Worldwide, `http://www.bsp-worldwide.org/`, April 1996.

[11] F. Halsall. *Data communication, computer networks and open systems.* Addison-Wesley, 1995.

[12] R. Handel and M. N. Huber. *Integrated broadband networks: An introduction to ATM-based networks.* Addison-Wesley, 1991.

[13] J. Hill. *BSP machine parameters*, Jun. 1997. http://www.bsp-worldwide.org/implmnts/oxtool.html.

[14] J. M. D. Hill and D. B. Skillicorn. Lessons learned from implementing BSP. In *High-Performance Computing and Networks*, Spring Lectures Notes in Computer Science Vol. 1225, pages 762–771, Apr. 1997. Also appears as Oxford University Computing Laboratory, Techical Report TR-96-21.

[15] C. Huang and P. K. McKinley. Communication issues in parallel computing across ATM networks. *IEEE Parallel and Distributed Technology*, Vol. 2(No. 4):73–86, 1994.

[16] C. Huang and P. K. McKinley. Design and implementation of global reduction operations across ATM networks. In *Proc. 3rd International Symposium on high-performance distributed computing*, pages 43–50, Aug. 1994.

[17] C. Huang and P. K. McKinley. Efficient collective operations with ATM network interface support. In *Proceedings of the 1996 International Conference on Parallel Processing*, volume I., pages 34–43, Aug. 1996.

[18] C. Huang, P. K. McKinley, and E. P. Kasten. Design and implementation of multicast operations for ATM-based high performance computing. In *Proc. Supercomputing'94*, pages 164–173, Nov. 1994.

[19] J. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, 1981.

[20] A. Jain and S. Keshav. Native-mode ATM in freeBSD: Experience and performance. In *Proc. NOSSDAV '96*, Apr. 1996. http://simon.cs.cornell.edu/Info/People/Skeshav/papers.

[21] H. A. Jamrozik, M. J. Feeley, G. M. Voelker, J. Evans, A. R. Karlin, H. M. Levy, and M. K. Vernon. Reducing network latency using subpages in a global memory environment. In *Proceedings of the 7th ACM Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[22] D. D. Kandlur, D. Saha, and M. Willebeek-Lemair. Protocol architecture for multimedia applications over ATM networks. *IEEE Journal on Selected Areas in Communications*, 14(7):1349–1365, Sept. 1996.

[23] K. Keeton, T. E. Anderson, and D. A. Patterson. LogP quantified: The case for low-overhead local area networks. In *Proc. Hot Interconnects III: A Symposium on High Performance Interconnects*, pages 10–12, Aug. 1995.

[24] P. Kermani and L. Kleinrock. A tradeoff study of switching systems in computer communication networks. *IEEE Transactions on Computers*, pages 1052–1060, Dec. 1980.

[25] M. Lin, J. Hsieh, D. H. C. Du, J. P. Thomas, and J. A. MacDonald. Distributed network computing over local ATM networks. Technical report, University of Minnesota, 1994. TR-94-17.

[26] W. F. McColl. General purpose parallel computing. In A. M. Gibbons and P. Spirakis, editors, *Lectures on Parallel Computation*, Cambridge International Series on Parallel Computation, pages 261–336. Cambridge University Press, 1993.

[27] W. F. McColl. The BSP approach to architecture independent parallel programming. Technical report, Oxford University Computing Laboratory, Dec. 1995. http://www.comlab.ox.ac.uk/oucl/users/bill.mccoll/ftpind.html.

[28] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26:67–76, Feb. 1993.

[29] Oxford Parallel Applications Centre. *Better ways to program parallel processors.* http://www.comlab.ox.ac.uk/oucl/oxpara/bsp/bspmodel.htm.

[30] G. B. Sabbatel. Hardware solution for efficient distributed computing on ATM networks. http://www.ens-lyon.fr/~btouranc/ETPSC-III/program/node18.html.

[31] M. Schwartz. *Broadband integrated networks.* Prentice Hall PTR, 1996.

[32] D. B. Skillicorn. Multiprogramming BSP programs. http://www.qucis.queensu.ca/home/skill/papers.html.

[33] W. Stallings. *Data and computer communications.* Collier Macmillan, 1985.

[34] A. S. Tanenbaum. *Computer networks.* Prentice-Hall, 1996.

[35] Y.-J. Tsai, Y. Huang, and P. K. McKinley. Performance evaluation of barrier synchronization in ATM networks. In *Proceedings of the 1996 IEEE International Conference on Computer Communications and Networks*, Oct. 1996. http://www.cs.msu.edu/~mckinley/Projects/publications.html.

[36] University of Oxford Parallel Application Centre. *BSP, The Bulk Synchronous Parallel model*, 1997.

[37] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, Aug. 1990.

[38] K. Vibahatavanij and C. Cheng-Hsing. Network of workstation. URL: http://www.ece.orst.edu/~vibhatke/p567.html, 1995.

[39] J. Vila-Sallent and J. Sole-Pareta. High performance distributed computing over ATM networks: A survey of strategies. In *Proceedings of the Second International Conference on Massively Parallel Computing Systems (MPCS '96)*, pages 153–160, 1996.

[40] A. Wolman, G. Voelker, and C. A. Thekkath. Latency analysis of TCP on an ATM network. Technical report, University of Washington, 1993. Technical Report 93-03-03.

[41] H. Zhu, L. A. Dasilva, J. B. Evans, and V. S. Frost. Performance evaluation of congestion control mechanisms in ATM networks. Technical report, Department of Electrical Engineering and Computing Science, The University of Kansas, 1995. http://www.ittc.ukans.edu/Projects/AAL/reports/.