SPEPCA: A Simulator for Performance Evaluation of Parallel Communication Algorithms.

P.Srinivas Vishal Mehra
K.Sivaramakrishna
School of Computer & Information Sciences
Syracuse University.

May 17, 1995

1 Introduction

The development of software for sequential and parallel applications is intrinsically difficult and time consuming. Moreover, the design and development of software for parallel computer is even more tedious due to an increased complexity incurred in handling various tasks like scheduling, communication, and performance. So, the development of various tools to make software development easier - particularly on parallel machines [1, 2, 3] - has been adopted from time to time.

Past experience is proof enough to show that the automation of the whole software development process for parallel computers is very difficult [4]; starting from the process of task partitioning, scheduling and fine-tuning performance, and ending up generating the code. However, the automation of some phases of the development life cycle has a two-fold advantage. Firstly, of attaining an increased programming productivity and secondly, of relieving the programmer from doing the tedious chores computers perform better than humans. Clearly, the choice of better algorithms for a specific task is one of those tedious tasks that the the developer encounters. So, we have developed a simulator which does the job of selecting the best communication algorithm on distributed memory parallel computers. The simulator works on a simple sequential computer and hence there is no need to run the code on a parallel computer which is pretty expensive.

The development of new algorithms and routing methods has obviated the consideration of link contention [5, 6, 7]. Moreover, for the CM-5 architecture, the data transfer rate appears to be bounded by the rate at which data can be sent or received by any arbitrary processor. Hence, the time will be lower bounded by the time necessary to eliminate the messages from the network - by the processor receiving the maximum amount of data - provided, a particular processor receives more than one message or has to send more than one message in a particular quantum. In this paper we do not consider link contention to be a bottleneck, as the routing in CM-5 is randomized. The simulator can also be used with architectures with link contention, with considerable risk of performance.

We start with the role of the simulator, notations, and communication properties used throughout in Section 2. Section 3 is devoted for the explanation of an overview of CM-5. Section 4 presents the Design of the tool. In Section 5 are presented the testing environment and the algorithms. Section 6 presents the experimantal results on a sequential machine as well as on a parallel machine, CM-5. Finally, conclusions are drawn in the Section 7. A list of references are provided in Section 8 for further understanding of the concepts put forth in this paper. And, following that are figures and tables explaining the results and concepts therein. In the Appendices are given the source code of the simulator, the sequential and parallel code used in testing the simulator etc,.

2 Preliminaries

2.1 Definitions

The communication matrix COM is a $n \times n$ matrix where n is the number of processors. $COM(i,j)$ is equal to a positive integer ${}^i_jm$ if processor P_i needs to send a message (of size ${}^i_jm$) to P_j, otherwise $COM(i,j) = -1, 0 \leq i,j < n$. Hence, row i of COM represents the sending vector, $send_i$, of processor P_i, which contains information about the destination node and the size of outgoing messages. Column i of COM represents the receiving vector, $recv_i$, of processor P_i, which contains information about the source node and the size of incoming messages. The entry $send_i^j(recv_i^j)$ represents the j^{th} entry in the vector $send_i(recv_i^j)$. Assuming $COM(i,j) = m$, $send_i^j = (recv_i^j) = m$. We use $send(recv)$ to represent each processor's sending(receiving) vector without ambiguity. $COM_{o/p}$ is the output of the communication algorithm.

COM can be decomposed into a set of communication quantas, $Q_k, 1 \leq k \leq l, l$ is a positive integer, such that,

$$COM(i.j) = m, \quad m > 0 \quad \Rightarrow \quad \exists! k, \quad 1 \leq k \leq l, \quad Q_k^i = j.$$

We define the k^{th} communication quanta as,

$$Q_j^i = j, \quad i = 0, 1, , ..., n-1, \quad \text{and} \quad 0 \leq j < n.$$

if processor P_i needs to send a message to a processor P_j at the k_{th} quanta, otherwise $Q_k^i = -1$.

Thus, *node contention* can be defined as,

$$\exists k, \quad 1 \leq k \leq l, \quad Q_k^{i_1} = j_1 \quad \text{and} \quad Q_k^{i_2} = j_2 \Rightarrow i_1 \neq i_2 \quad \text{and} \quad j_1 = j_2 \neq -1,$$

$$\text{where} \quad i_1, i_2 = 0, 1, ..., n-1 \quad \text{and} \quad 0 \leq j_1, j_2 < n.$$

A partial permutation M_k is a communication quanta such that

$$M_k^{i_1} = j_1 \quad \text{and} \quad M_k^{i_2} = j_2, \quad i_1, i_2 = 0, 1, ..., n-1 \quad \text{and} \quad 0 \leq j_1, j_2 < n,$$

$$i_1 = i_2 \quad \Leftrightarrow \quad j_1 = j_2;$$

M_k^i = -1 if P_i does not send a message at this permutation.

As permutation has the property that every processor does the job of both sending and receiving at most one message, it wouldn't cause any node contention. This paper uses permutation algorithms for the testing of the simulator.

2.2 Role of the Simulator in software design

Here we explain the role of a simulator in the deisgn life cycle of software development. Figure 1 gives a an idea of, where exactly the simulator fits in. The input to the communication algorithm is the communication matrix

COM. COM is as shown in the Figure 2. The notation is as explained above. The algorithm then appropriately schedules the messages which need to be communicated among the processors, represented in the figure by its size $^{i}_{j}m$. The algorithm schedules these messages according to a rule which reduces or accomplishes the minimum node contention, while optimising total amount of communication. The schedule that is obtained is $COM_{o/p}$, and this is as shown in the Figure 3. The Scheduler takes this $COM_{o/p}$ as its input and does the simulation, to produce a performance measure of the algorithm under test. Then the efficiency of the communication algorithm can be estimated. The simulator is not a programming environment, but a pre-implementation design tool. This tool can, to some extent, perform "what if" analysis about a parallel program design and to provide information on what can be done to fine-tune a design before it actually runs on a parallel machine. This tool provides the developer with the ability to try different design alternatives like uniformity of messages, density of communication matrix, static scheduling, runtime scheduling etc,.

2.3 Notations & Assumptions

The categorization of the scheduling algorithms have been done in the following categories:

- *Uniformity of message* – Whether all the messages are of equal size or not. We assume the most general case of messages of non-uniform size. Algorithms having same size have considerably smaller scheduling overhead [8].

- *Density of communication matrix* – Whether the communication matrix is nearly dense or is sparse. In the former case, every processor sends(receives) messages to(from) every other processor. In the latter case, a subset of processors send(receive) messages to(from) a subset of all the processors.

- *Static or runtime scheduling* – Whether communication scheduling is performed statically or dyanamically.

For reasons mentioned earlier, the scheduler does not account for link contention. Also the following assumptions are made in the paper:

- In case, communication is sparse, all nodes send(receive) an approximately equal number of messages. Let the density be represented by d. So to say that d represents the number of messages sent(received) to(by) every processor.
- We assume that each processor can send only one message and receive only one message at a given time. If the density is d, then atleast d permutations are essential to send or receive all messages.

3 CM-5 System

3.1 Overview

This section gives a brief overview of the CM-5 system for which the simulator has been designed, and used to conduct our experiments. CM-5 is available in configuration of 32 to 1024 processing nodes, each node being a SPARC microprocessor (μp) with 32MB of memory and optional vector units. The node operates at 33MHz and is rated at 22MIPS and 5MFLOPS. When equipped with vector units, each node of the machine is rated at 128MIPS (max) and 128MFLOPS (max).

The CM-5 has two components in its internal network, a data network and a control network. It also is equipped with a separate diagnostics network to detect and isolate errors throughout the system.

High-performance data communications among all the system components are provided by the data network. The network operates at a peak bandwidth of 5MB/s for node-to-node communication. However, if the destination is within the same cluster of 4 or 16, it can deliver a peak bandwidth of 20MB/s and 10 MB.s respectively [9].

All operations that require the cooperation of many or all processors, are performed by the control network. It accelerates cooperative operations like the integer reduction operation, broadcast, or system management operations such as error handling.

3.2 Node contention aspects of CM-5

The impact of node contention on CM-5 is illustrated in Figure 4. In these experiments processor P_{31} is the receiving node, and processors P_i, $0 \leq i < d$

are sending nodes. In each quantum, each sending node sends an equal amount of data (256 bytes or 4096 bytes) to P_{31} simulataneously. We record the time (in milliseconds) taken for P_{31} to ocmplete receiving all incoming data, and the maximum, minimum, and average time taken among sending nodes to complete sending data.

The results support the fact that that when the number of messages sent to the same node (simultaneously) increases, the average time each sending node needs to complete sending its message increases (as do the maximum and minimum times). Hence two processors sending a message to the same processor simultaneously, should be avoided. Also, some nodes may take a greater time (than the maximum time) to complete sending messages. The only exception to the increase in time is when all 31 nodes send messages to processor P_{31}, nodes P_{28}, P_{29}, P_{30}, P_{31} are in the same 4-node cluster. This provides higher communication bandwidth, so the minimum time taken during this stage is drastically decreased on comparision with the previous (16-node) stage.

These results show that the node contention factor becomes a severe bottle-neck which can degrade the overall performance, due to communication delays. Hence, node contention is a serious candidate for consideration when scheduling is involved.

4 Design of the Simulator

4.1 Basics of design

As illustrated in Figure 2, the Simulator takes its input from the output of the Communication Algorithm. The input is as shown in Figure 3. The $COM_{o/p}$ table has as its entries, the message length ${}^{i}_{j}m$, the start time, $T_{start(i,j)}$, and the destination i. The table as $COM_{o/p}$ is present in each of the processors. So, for the processor P_i, we have $COM^{i}_{o/p}$ at its disposal. The start time, $T_{start(i,j)}$ is defined as the time at which the message is to be sent. At a particular time (or even at start time), there might be more than one message waiting to be sent to a destination. This is known as the problem of Node Contention. This is exactly what we are trying to address in the designing of the Simulator. Before we go any further into the discussion on the design of the simulator, we like to re-state the parameters we are trying to optimize. These are the Total Execution time (TET), and the Turn-around Time (TAT), taking node contention into consideration. This is done by prioritizing the messages according to a certain criteria and sending the message with the highest Priority (P), first. What is done here is at the sending end, we have a send queue, which is a queue of messages waiting to be sent. First, the Start time $T_{start(i,j)}$ of $COM_{o/p}$ table is sorted in increasing order. Now, we have all the messages with their lengths to be sent to various destinations in the increasing order of Start times. The first thing to do now is to see if any messages are there, which need to be sent at time 0 (There will be because we assume, that the communication starts at time 0). The next thing to check is to see if any messages are contending to send a message at this time. If yes, then calculate the Priority (P_s), and send the one with the highest priority. If no messages are contending, then we send the first packet of the message during the Time-slice (T_{slice}) alloted to it. And, after the message is sent, the Length of the message to be sent is reduced by the amount already sent. This takes care of the scheduling process at the sender's side. Upto now we have addressed the issues at the sender's end. We note that the receiver's side also has similar concerns.

Many messages might be contending to be received at the same time. So, even here we address the same problem of Node Contention, but in a slightly different way. The Priority (P_r) calculated here is different. The calculation of priorities are addressed below.

4.2 Parameters their definitions and determination

- $Priority$ - Priority is defined is defined as the difference of weighted T_w and weighted L_{rem}.

- $Wait-time$ - Wait-time T_w is defined as the amount of time a message has been waiting in a queue.

- $Remaining-length$ - Remaining-length L_{rem} is defined as the length of message which is waiting to be sent/received.

  As noted, the main parameters involved in the design of the Simulator are the Priority at the sender's end (P_s), and that at the receiver's end (P_r). The priority at both ends depends on the (T_w), and the (L_{rem}) of the message. While, at the sender's end the remaining length decreases as with increasing time, the length at the receiver's end might increase or decrease with increasing time. This is the only difference between the two L_{rem}s. Now the priority increases with an increase in T_w, and decreases with L_{rem}. The more is the wait time, more is the priority , and, more is the remaining length, the less is the priority for that message. The formula used in calculating Priority (P) in general is,

$$P = (w_1 \times T_w) - (w_2 \times L_{rem})$$

where w_1 and w_2 are the weightages associated with T_w and L_{rem} respectively. The detemination of these weightages w_1 and w_2 forms the most important part of the design of the simulator.

The first thing that we did was to take an algorithm and run it on a sequential machine using the communication techniques that we were trying to implement in the simulator. Now, we had an Execution Analysis Table (EAT) which contained a set of values of weightages w_1, w_2, Time-slice T_{slice} and the associated Execuation times (T_{exec}). The same thing was repeated for two other algorithms. Next thing done is to run the same algorithms with the same set of COM matrices on the Parallel machine. So we had three sets of execution times on both sequential and parallel machines. Let us denote these execution times as T_{serial_exec1}, T_{serial_exec2}, T_{serial_exec3} for the sequential machines, and $T_{parallel\_exec1}$, $T_{parallel\_exec2}$, $T_{parallel\_exec3}$ for those on parallel machines. Now, if the simulator were a perfect one we would have the same values for the execution times for a particular algorithm on a sequential as well as a parallel machine. With this in mind, we calculate the ratios,

$$\frac{T_{serial\_exec1}}{T_{serial\_exec2}}, \qquad \frac{T_{serial\_exec2}}{T_{serial\_exec3}}, \qquad and \qquad \frac{T_{serial\_exec3}}{T_{serial\_exec1}}$$

for the sequential machine, and

$$\frac{T_{parallel\_exec1}}{T_{parallel\_exec2}}, \qquad \frac{T_{parallel\_exec2}}{T_{parallel\_exec3}}, \qquad and \qquad \frac{T_{parallel\_exec3}}{T_{parallel\_exec1}}\ ]$$

for the parallel machine. Now if the values of weightages w_1, w_2, Time-slice T_{slice} were chosen correctly,

$$\frac{T_{serial\_exec1}}{T_{serial\_exec2}} = k \times \frac{T_{parallel\_exec1}}{T_{parallel\_exec2}}$$

Where k is called the Figure of merit of the Simulator we designed. Under ideal conditions, i.e., for an Ideal Simulator, $k = 1$ (Refer Appendix A). But since all the algorithms involved in the parallel environment, have not been mapped exactly on to the simulator designed, we can only try to get a value of k nearly equal to 1. So, the parameters, w_1, w_2, and T_{slice} were chosen such that we would have the best value for k. At this value of k, we had,

$$w_1 = 3, \quad w_2 = 2, \quad and \;\; T_{slice} = 1.$$

Now, there is an important issue related to T_{slice} as far as overall Turn-around time (TAT) is concerned. The effect of T_{slice} on the TAT is shown in the graph. If the T_{slice} is very less, the TAT would be very high because none of the messages would get a fair piece of time and the overhead in scheduling the messages would be unncessarily be spent without much gain. And, if the T_{slice} is very high, then, it would unnecessarily waste the time spent in sending messages of small length, which need a very small time slice for optimality. So, the T_{slice} chosen should be such that, it neither is too high nor too low. This theory of determination of T_{slice} is in perfect conformation with the practical value that we have obtained.

Now the design is a robust one as far as the goal constraints are concerned. Mainly, the optimization of the TAT, and the consideration of node contention while scheduling the messages.

5 Testing of the tool

5.1 Algorithms overview

The algorithms used for testing are all compliant with the assumptions and suit the environment under consideration [10]. In this section we present a brief overview of the algorithms used for testing the simulator. All the algorithms assume the availability of a global COM for their input. The algorithms themselves can be briefly clssified under the following titles:

– Asynchronous Communication
– Linear Permutation
– Priority Scheduling With Waiting

All algorithms are executable in a loosely synchronous fashion. And these algorithms do not use global synchronization to enforce synchronization between communication stages. Below is a brief discussion of all the algorithms, but for a detailed discussion on these algorithms we would refer you to [10].

5.2 Asynchronous Communication

The algorithm can be divided into three phases:

1. Each processor initially posts requests for incoming messages, and this operation would pre-allocate buffers for those messages.
2. Each processor sends all of its outgoing messages to other processors.
3. Each processor checks and confirms incoming messages which might have arrived at their receiving buffer(s) from other processors.

This algorithm is represented as below:

Asynchronous_Send_Receive()

For all processors P_i, $0 \leq i \leq n-1$, in parallel
do
    allocate buffers and post requests for incoming messages;
    send out all outgoing messages to other processors;
    check and confirm incoming messages from other processors;

enddo

Asynchronous Communication Algorithm

5.3 Linear Permutation

In this algorithm each processor P_i sends a message to processor $P_{i \oplus j}$[1] and receives a message from $P_{i \oplus j}$, where $0 < k < n$. When $COM(i,j) = -1$, processor P_i will not send a message to processor P_j, but will receive a message from P_j if $COM(j,i) > 0$. The entire communication uses pairwise exchange ($j = i \oplus k \Leftrightarrow i = j \oplus k$). This algorithm is represented as below:

Linear_Permutation()

For all processors P_i, $0 \leq i \leq n-1$, in parallel
do
    for $k = 1$ *to n-1 do*
        $j = i \oplus k$;
        if $COM(i,j) > 0$ *then* P_i *sends a message to* P_j;
        if $COM(j,i) > 0$ *then* P_i *receives a message from* P_j;
    endfor
enddo

Linear Permutation

5.4 Priority Scheduling With Waiting

This algorithm uses the maximum length first procedure to determine the priority for a particular communication. So, if there are n sources wanting to send messages to n destinations, the one wanting to send a message of maximum length among them has the highest priority. And, if there exists a processor which is sending a message of smaller than maximum length waits till the one with maximum length has finished, to start sending the next message.

In this algorithm the total time complexity is $O(n^3)$, since for all the processors, we calculate the permutation, and calculating permutation for each processor takes $O(n^2)$ time. During the communication scheduling, and priority scheduling with waiting, the worst case time complexity to access

[1] $\oplus$ represents bitwise exclusive OR operartor.

each entry of COM is $O(n^2)$. In order to reduce this overhead, there exists algorithms which compress the COM into an $n \times d$ matrix $CCOM$ by a simple compressing procedure [11]. This procedure will improve the worst case time to access each active element (of $CCOM$) to $O(dn)$ from $O(n^2)$. This can be incorporated into the current algorithm to improve the efficiency of the algorithm.

Priority_Scheduling_With_Waiting()

For all processors P_i, $0 \leq i \leq n-1$, in parallel
do
    for $i = 0$ *to n-1*
        $Calculate\_Permutation()$;
    $endfor$
    send the messages as dictated by the permutation.
    wait till the end of communication in the current permutation.
    make the COMM matrix entries 0, for all the communicating processors in the current permutation.
$enddo$

Calculate_Permutation()
do
    for $i = 0$ *to n-1*
        $max = 0$;
        for $j = 0$ *to n-1*
        if (COMM[i][j] ¿ max)
            $max = COMM[i][j]$;
            $destiation[i] = j$;
    $endfor$
$enddo$

Priority Scheduling With Waiting

6 Results

6.1 Explanation of the graph

When the algorithms described as above are tested on the simulator for its performance, a significantly good Figure of Merit k for the Simulator was acieved. This means that the assumptions made and also the design of the Simulator was nearly as it would have been, if the algorithms were run on a parallel machine. We have obtained in Figure.6, and Figure.7, the total turn around times for various $COMM$ matrices that we have considered. We have taken four 4×4 $COMM$ matrices to test the performance of the Simulator. In the figures, the algo1 refers to the performance of the Asynchronous Commuication algorithm, algo2 refers to the performance of the Linear Permutation algorithm, and algo3 refers to that of the Priority Scheduling With Waiting algorithms. The TAT is the Turnaround Time for a particular algorithm, when a particular $COMM$ matrix was input. The input for an algorithm is the $COMM$ matrix. The points are taken at the integer values of the $x - axis$, as these are the values of the $COMM$ matrix number. So, there is no need to worry about the TAT for fractional values of the $COMM$ matrix input.

6.2 Consolidation of the final result

In both the outputs (i.e,. the graphs obtained on a Sequential machine, and a Parallel machine), the shape of the contour obtained is to 'some extent' maintained. This feature, corroborates the fact that the Simulation has been done well. Specifically, when we take the differences of the TAT at the integer values of $COMM$ matrix, for both the graphs, the scaling factor remains approximately same. This scaling factor is what is called, $The\ Figure\ of\ merit$ of the simulator. For example, the differences (between TATs for the set of algorithms (1,2) and (1,3)) obtained on the parallel machine for $COMM$ matrix 2 are 10.4 and 3.6 respectively, and that for $COMM$ matrix 3 are 12.5 and 5.5 respectively. And, the same values on the sequential machine turn out to be 15 and 6 for $COMM$ matrix 2 and 39 and 9 for $COMM$ matrix 3 respectively. So, when the ratios of differences on parallel machine turn out to be 0.5769, and 1.2711, which is approximately two times the other. But potentially, this could have been in the range $(-\infty, \infty)$. Which shows that the k is good for this simulator we have designed.

Appendix

A. Proof for $k = 1$, for an Ideal Simulator

Consider two algorithms A and B. Let their execution times on a parallel machine be $T_{parallel_a}$ and $T_{parallel_b}$ repectively. Let the execution times of the same algorithms on a sequential machine be T_{serial_a} and T_{serial_b} repectively.

Since, we are trying to build a simulator which will evaluate the performance of any algorithm on a parallel machine without running it actually on the machine, the execution time on the sequential machine will be a constant times that on the parallel machine. So, for algorithm A, we have a constant k_1 such that,

$$T_{serial_a} = k_1 \times T_{parallel_a}$$

and since, this is true for any algorithm, we have another constant k_2 such that,

$$T_{serial_b} = k_2 \times T_{parallel_b}$$

Now, since the Simulator should predict the execution times exactly with the same precision for all the algorithms, we have,

$$\frac{T_{serial_a}}{T_{serial_b}} = \frac{T_{parallel_a}}{T_{parallel_b}}$$

$$\Rightarrow \frac{k_1 \times T_{parallel_a}}{k_2 \times T_{parallel_b}} = \frac{T_{parallel_a}}{T_{parallel_b}}$$

$$\Rightarrow \frac{k_1}{k_2} = 1$$

But since, we have that,

$$\frac{T_{serial_a}}{T_{serial_b}} = k \times \frac{T_{parallel_a}}{T_{parallel_b}}$$

where k is the Figure of merit of the Simulator, we have,

$$\frac{k_1}{k_2} = k = 1$$

Hence, for an ideal Simulator, $k = 1$.

B. Source code of SPEPCA

I. Sequential code listing

II. Parallel code listing

Acknowledgements

We are grateful to GOD, Who has helped in every aspect of the project.

References

[1] T. Lewis and H. El-Rewini, *Parallax: A tool for parallel program scheduling*, IEEE Parallel and Distributed Technology: Systems and Applications **1**, no.2, (1993), pp. 62-72.

[2] M. Wu and D. Gajski, *Hypertool: a programming aid for message-passing systems*, IEEE Transactions on Parallel and Distributed Systems **1**, no.3, (1990), pp. 101-119.

[3] T. Yang and A. Gerasoulis, *PYRROS: static task scheduling and code generation for message passing multiprocessors*, Proceedings of the 6^{th} ACM International Conference on Supercomputing, (1992), pp. 428-443.

[4] Hesham El-Rewini, Theodore G. Lewis and Hesham H. Ali, *Task Scheduling in Parallel and Distributed Systems*, Prentice-Hall, (1994).

[5] Willian J. Dally and Chuck L. Seitz, *Deadlock-free message routing in multiprocessor interconnection networks*, IEEE Transactions on Computers **36**, no.5, (1987), pp. 547-553.

[6] Lionel M. Ni and Phillip K. McKinley, *A survey of wormhole routing techniques in direct networks*, IEEE Computer, **26**, no.2, (1993), pp. 62-67.

[7] Thinking Machines Corporation, Cambridge, MA, *The Connection Machine CM-5 Reference Manual*, (1992).

[8] Sanjay Ranka, Jhy-Chun Wang and Geoffrey C. Fox, *Static and runtime algorithms for all-to-many personilazed communications on permutation networks*, Proceedings of the 1992 International Conference on Parallel and Distributed Systems, HsinChu, Taiwan, (1992), pp. 211-218.

[9] Zeki Bozkus, Sanjay Ranka and Geoffrey C. Fox, *Benchmarking the CM-5 multicomputer*, Proceedings of the Frontiers of Massively Parallel Computation, (1992).

[10] Sanjay Ranka, Jhy-Chun Wang and Manoj Kumar, *Personalized Communication Avoiding Node Contention on Distributed Memory Systems*, International Conference on Parallel Processing, July (1993) vol.1, pp. 241-244.

[11] Sanjay Ranka and Jhy-Chun Wang, *Static and runtime scheduling of unstructured communication*, International Journal of Computing Systems in Engineering, (1993).

[12] E.Horowitz and Sahni, *Fundamentals of Data Structures in Pascal*, Computer Science Press, Maryland, (1987).

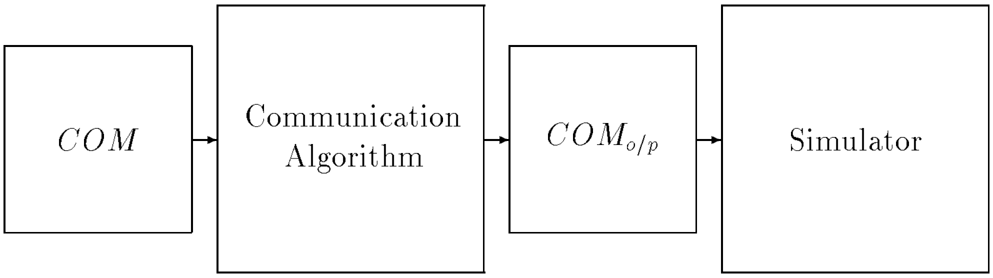

Figure 1. The flow of data for the Simulator.

	0	1	2	3	4	5	6	7
0	0	7	5	0	6	9	0	9
1	2	0	0	4	7	0	4	2
2	4	0	0	4	8	6	3	8
3	9	5	5	0	9	0	0	0
4	0	8	9	0	0	0	0	8
5	0	3	0	0	0	0	7	0
6	0	0	6	8	0	0	0	0
7	6	0	0	1	1	0	5	0

Figure 2. An 8×8 COM (A '0' denotes no communication.)

Length	*Start time*	*Destination*
100	0	2
200	2	3
220	1	2
200	2	3
200	2	3
200	2	1
200	2	1
200	2	0
200	2	0
200	2	4
200	2	4
200	2	6
200	2	7
200	2	7

Figure 3. A sample $COM_{o/p}$ table.

d	256 Bytes				4096 Bytes			
	recv	send			recv	send		
		max	min	avg		max	min	avg
1	0.089	0.131	0.050	0.061	0.516	0.504	0.485	0.488
2	0.125	0.150	0.070	0.081	1.083	1.048	1.023	1.038
4	0.205	0.199	0.098	0.116	2.189	2.124	2.085	2.097
8	0.375	0.298	0.173	0.210	4.693	4.844	4.353	4.502
16	0.731	0.575	0.302	0.394	9.865	10.065	9.155	9.476
31	1.396	1.279	0.151	0.815	19.485	19.544	2.847	15.550

Figure 4. Impact of node contention on the CM-5.

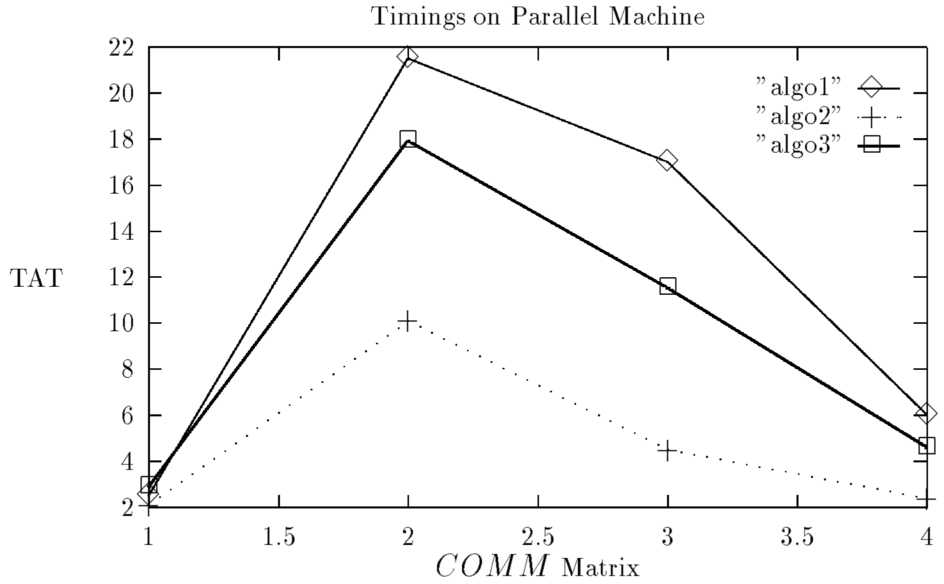

Figure 5. The Turnaround Time for the 3 algorithms on a Parallel Machine.

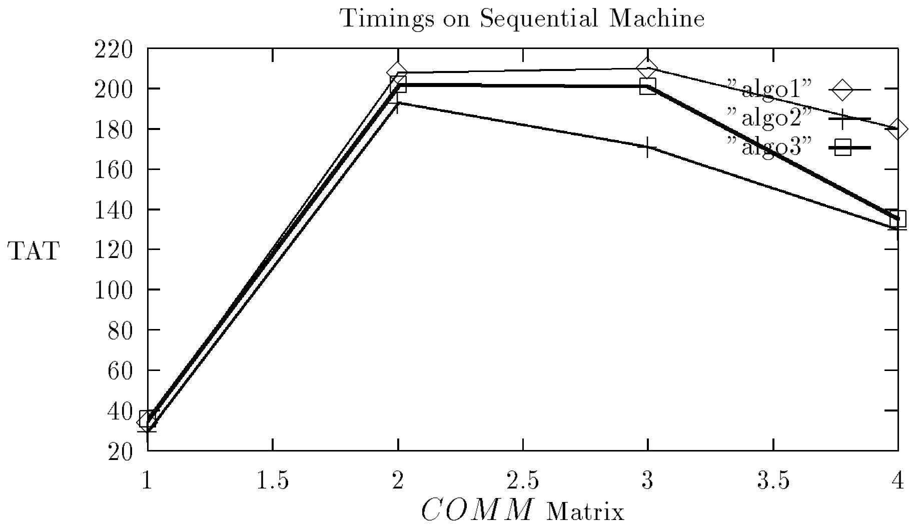

Figure 6. The Turnaround Time for the 3 algorithms on a Sequential Machine.