SIMULATION OF SEMICONDUCTOR DEVICES AND PROCESSES Vol. 4
Edited by W. Fichtner, D. Aemmer - Zurich (Switzerland) September 12-14, 1991 - Hartung-Gorre

Parallelization of Monte Carlo Analysis on Hypercube Multiprocessors and on a Networked EWS System

Satoshi Sugino*, Chiang-Sheng Yao and Robert W. Dutton
Integrated Circuits Laboratory
AEL 203, Stanford University, Stanford, CA 94305, U. S. A.
(TEL)415-725-0458 (FAX)415-725-7298

Abstract

Comparison of Monte Carlo simulation on two different parallel computer architectures has been made. The speed-up and communication times are summarized in analytical form. The advantage of using parallel processing on the Intel hypercube architecture is shown compared with the networked EWS system, especially when more than 32 nodes are used. The performance of the 32 nodes on the hypercube machine is demonstrated to be about 1.7 times faster than that of a 32-EWS system. Moreover, because of the high data transmission rate and hypercube connection, we have shown that the hypercube multiprocessor should not exhibit performance saturation up to 128 nodes. For the networked EWS system the performance falls off dramatically beyond the peak at 32 processors.

Using the Monte Carlo (MC) method for semiconductor device analysis is becoming a realistic tool for device engineers[1][2](See Figure 1). The huge computational time, however, is one of the main drawbacks for MC simulation. Although both the windowed method and coupled Drift-Diffusion(DD) method have already been used for the Monte Carlo analysis, it still takes many hours even for a high-end workstation(See Table 1). Because of the intrinsic parallelizable feature of the MC method, parallel computation offers the possibility of driving down not only the computational time but also the cost.

Parallelized Monte Carlo simulation for semiconductor device analysis has been investigated both on hypercube multiprocessor machines and networked Engineering WorkStation (EWS) systems. The communication issues between many CPU's and message passing timing are of critical concern in the parallelization process. In the present work, a parallel algorithm for the MC simulation has been implemented into a self-consistent 2D simulator *BEBOP*[3] and developed on the Intel iPSC/860 hypercube as well as a networked EWS system. The main feature of these systems are summarized in Table 2 . The performance of the parallelized MC simulation on both systems are compared. The speed-up and communication time are measured experimentally and generalized into analytical forms based on the parameters of the systems used.

1 Parallel MC algorithms

1.1 Communications

In our MC experiment, when more than 10,000 particles are used, we can show that more than 98% of the time is spent on the part that can be parallelized easily, which corresponds to the

*Visiting scholar from Matsushita Electric Works, 1048 Kadoma, Osaka 571, Japan.

Machine	Elapsed Time
sun sparc 1+	669min/bias
sun4/490	401min/bias

Table 1: Monte Carlo benchmark.

Intel i860		
processor	i860(64bit)x32	
Memory	16Mbyte/node	
Sun Sparc station		
	sparc 1+	server 4/490
processor	15.8MIPS (25MHz)	22MIPS 3.8MFLOPS (33MHz)
FPU	sparc fpp (25MHz)	TI 8847 (33MHz)

Table 2: SUN SPARC computer & Intel iPSC/860.

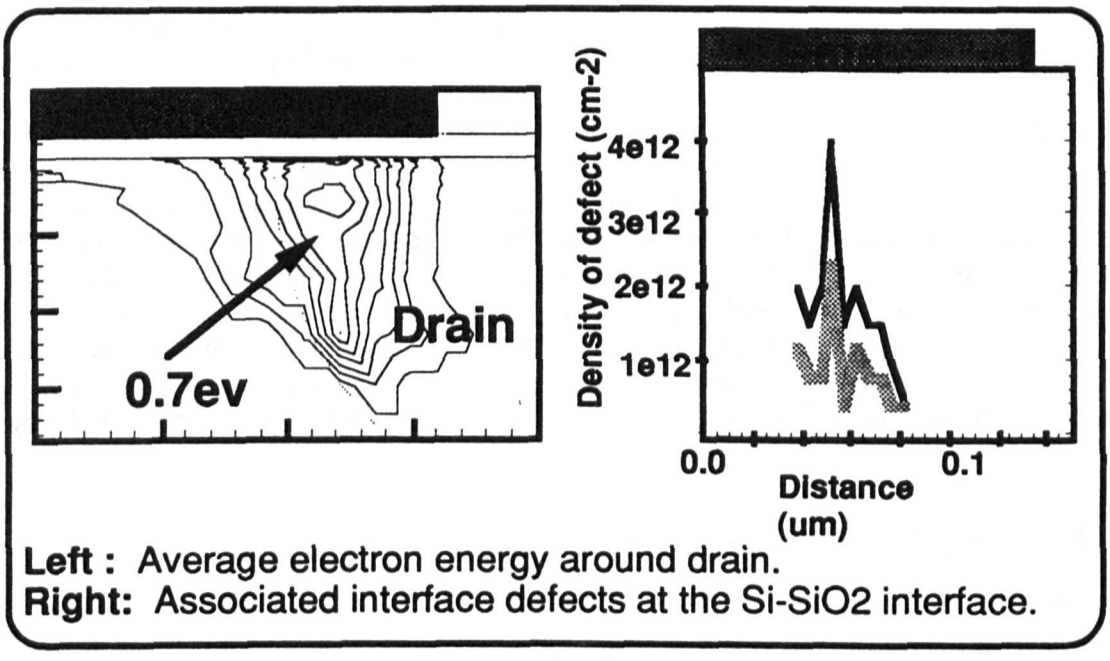

Figure 1: Relation of average electron energy and oxide interface defects.

calculation of movement and scattering of particles(hatched part of Figure 2). The remaining parts, for example, the Poisson solver and gathering the statistics results, are not considered to be parallelized in this work. Instead of using the host computer with an i386 CPU, one of the 32 nodes in the hypercube machine is used to do the nonparallelized parts of the MC simulation. The host node(node 0 in our case) makes 4 different kinds of communication with the others. One communication is to send global data which contains electric field and scattering table data to other nodes. Another is to send back the MC results(such as the number of events for each scattering mechanism) from other nodes to the host node. These two message passings are done every time when the Poisson solver loop is called.

Inside the Poisson loop, the communication is made by sending and gathering particle-associated data(position, momentum etc.)just before getting into and getting out of the parallel MC procedure. The data size of these messages are 1.03 megabytes for global data, 100 kilobytes for statistics data and 56 bytes per particle for the particle-associated data. The global data size depends on the device structure, but the data size for each particle is fixed.

1.2 Load balancing

The other important issue in developing parallel algorithms is how to make the load on each node balanced so that each node can finish its job at the same time. For MC simulation , if particles of the same amount are allocated in each node , it seems likely that the load balancing can be achieved easily. In reality ,however, load balancing can't be achieved unless both of the following conditions are satisfied. First, each node must have the same amount of particles. Second, the initial spatial distribution of the particles inside the device must be the same for each node. Figure 3 shows the NMOSFET structure we uses in this work. Figure 4 shows a bad example of particle distributions. Under the condition of Figure 4, the number of particles are equal for each node, but particles are allocated to the nodes with an order of the MOS channel coordinates. Consequently it makes about a 5 times difference for the elapsed time between the fastest node and the slowest one. In our MC algorithm, since the host node has to wait for all nodes to finish their jobs in order to get back the new data, the computational efficiency of the parallel part have been seriously compensated by the slowest node. Figure 5 shows the results for the modified, correct method. Only 5% of the elapsed time difference between the fastest node and the slowest node is observed. The reason of these phenomena comes from the fact that the elapsed time is strongly related to the magnitude of the local electric field, which is used in the most time consuming part of the MC simulation so called "trajectories calculation of particles"; more than 65% of the elapsed time is spent here. This method should be considered seriously since the electric field may vary by several orders of magnitude inside the real device.

2 Parallel MC Systems

2.1 Intel iPSC/860

In the present work, an Intel hypercube machine with up to 32 nodes is used for experimentation. All the nodes in the iPSC system are fully connected and the message goes directly to the receiving node without disturbing any of the other nodes through the Direct-Connect Module. It also offers a special method, the so called "global operation", to do the vector-like operation between the nodes. Because of the hypercube connection, the communication time will be reduced to a function of only the dimension of the cube size when the message is passed from one node to all the others. For example, if we have 32 nodes in the cube, only 5 communication

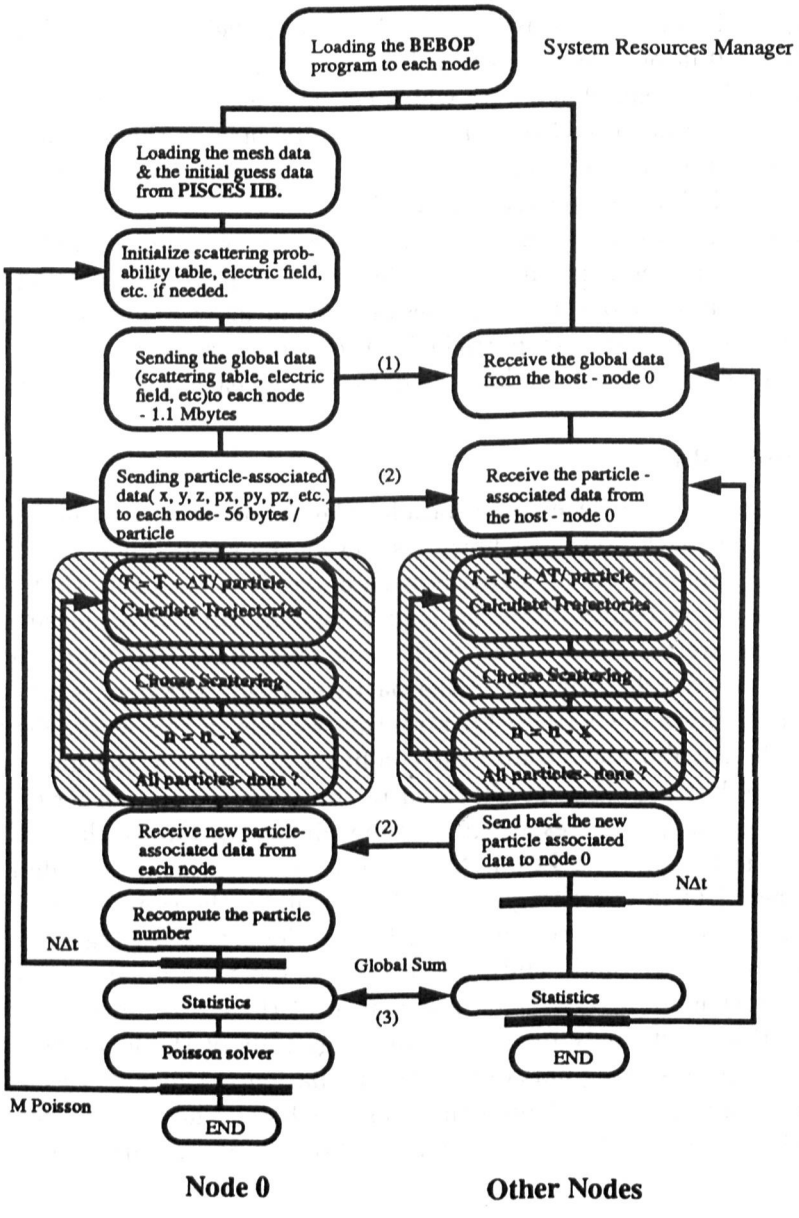

Figure 2: The flowchart of the Parallelized MC method.

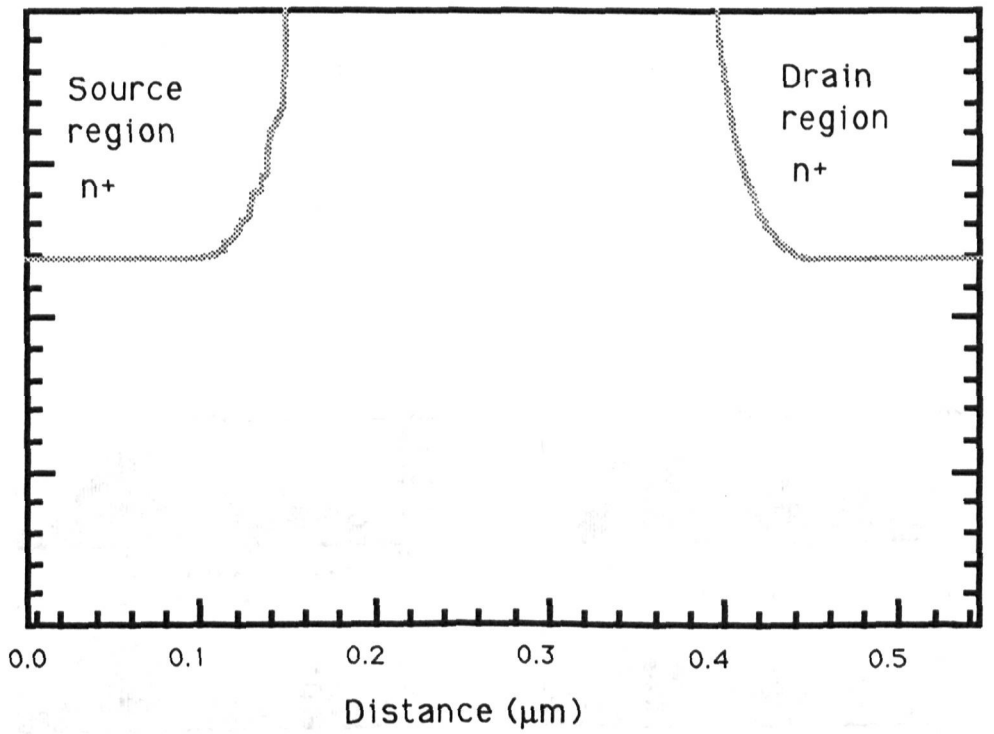

Figure 3: The NMOSFET structure.

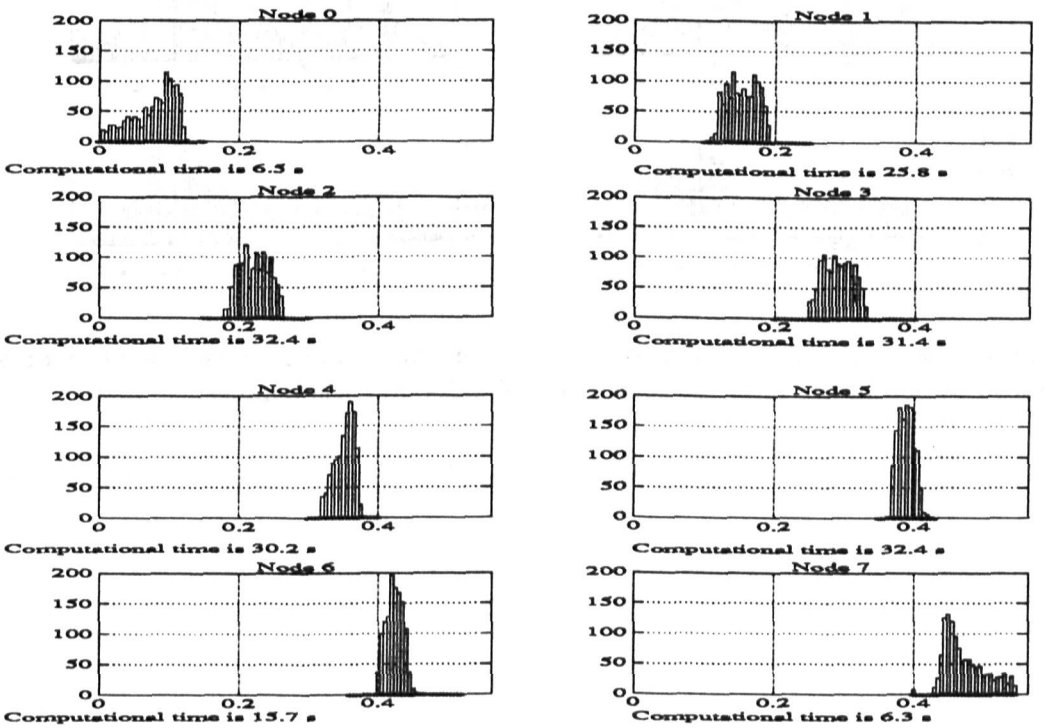

Figure 4: The spatial distribution of particles along the channel in each node: bad example. X axis—Position in μm from the source end. Y axis—Particle number.

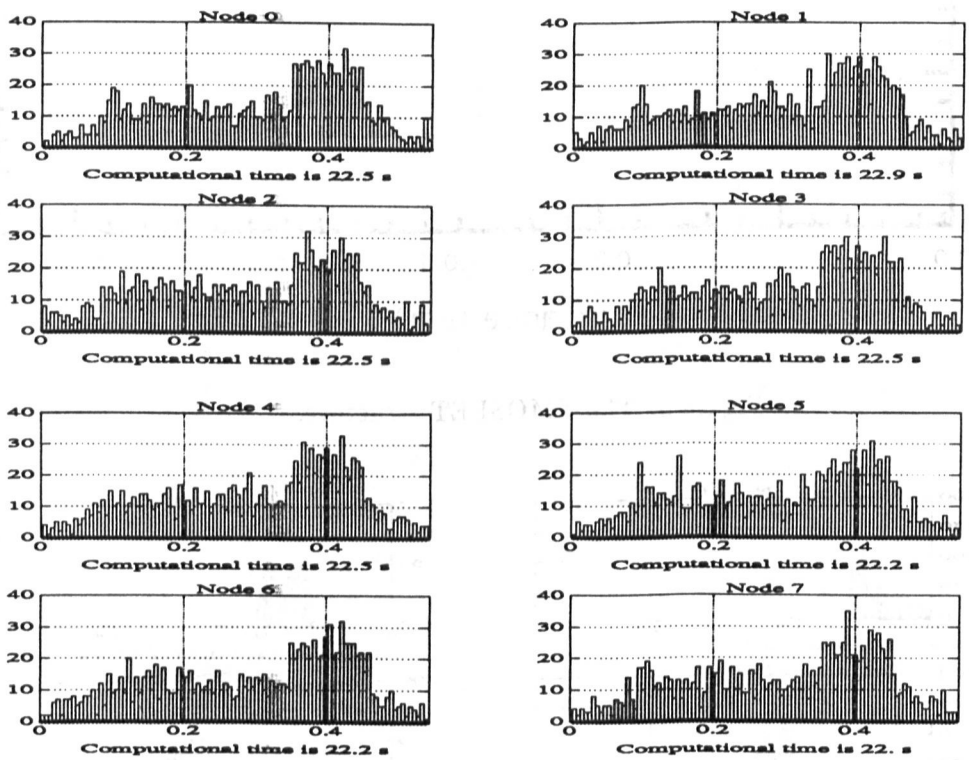

Figure 5: The spatial distribution of particles along the channel in each node: good example. X axis—Position in μm from the source end. Y axis—Particle number.

Experiment condition:
21 time steps
1 Poisson iteration
global data size:1.03 Mbytes
particle data size: 56 bytes / particle
10,000 particles
98.25% are parallelized

	SPARC1$^+$	SUN4/490	1 nodes	32 nodes
time	4042.3sec	2672.7sec	3956.5sec	201.42sec

Table 3: Comparison of the performance on iPSC/860 and SUN computers

messages are necessary instead of 31 times in an usual way. The global operations also use this kind of communication for their operations.

2.2 Networked EWS system

Network programming of the parallel version of MC simulation has been carried out using SUN-RPC(Remote Procedure Call) language and remote execution commands provided by 4.3 BSD UNIX[5]. In a remote procedure call, a process on the local system invokes a procedure on a remote system. RPC makes it appear to the programmer as if usual subroutine calls are taking place. The network is established by Ethernet with the TCP/IP protocol. Four SPARC 1$^+$ workstation with 24 megabytes memory and nine SPARC IPC with 12 megabytes memory have been used for this experiment. Here mainly the communication time between workstations and CPU time for MC on one CPU are measured in order to get an analytical form for the networked EWS system.

3 Comparison

3.1 One EWS vs. iPSC/860 Hypercube

The results are summarized in Table 3. The performance of MC are compared with the computational time per single CPU. The speed-up for 32 nodes is observed to be nearly 20 times faster than that of a single i860 CPU and 13 times faster than that of the SUN4/490 SPARC server and 20 times faster than that of one SPARC1$^+$.

3.2 Networked EWS system vs. iPSC/860 Hypercube

The performance difference between the networked EWS system and the hypercube machine becomes obvious as expected when more than 32 CPU's are used. For the 32 node case, the simulation time on the hypercube machine is about 1.5 times faster than that of the networked EWS system. Furthermore in the case of networked EWS system , we cannot expect to see more speed up even if more workstations are added to the network. These speed up performances are summarized into analytical form in the following section.

4 Analytical formula of Performance on both Systems

In order to compare the performances of these two systems, we have generated a simple analytical form of the elapsed time for both systems. Two assumptions are made here:

1. Each node spends the same computational time.
2. There is no data collision between nodes for message exchange.

Actually the second assumption is based on the first, as now explained.
The only place that data collision will occur is when the nodes send back the new particle-associated data to the host node. Because the original particle-associated data is sent to each node serially, the node that gets these data first should finish its computation and send back the data to the host first (Assumption 1). Based on these two assumptions and under our experiment condition, we get the following analytical form of the performance on the hypercube machine:

$$\begin{aligned} \tau = & \left(\frac{0.9825}{N} + 0.0175 \right) T_1 \\ & + 0.77 \times log_2 N \\ & + \frac{8.4}{N} \times (N-1) \\ & + 0.09 \times log_2 N \quad \text{(seconds)} \end{aligned} \tag{1}$$

In the above equation, N is the number of nodes, T_1 is the elapsed time spent by one CPU—the fist two terms are simply the elapsed time predicted by Amdahl's law[4]. The third term shows the communication time for sending the global data to the nodes, which is only a logarithmic function of the cube size. The fourth term is the message passing time for exchanging the particle-associated data between the host node and the others which is essentially a constant when $N \gg 1$. The last term is the communication time needed to do the "global operations".

In a similar way, we can also obtain the analytical form of the performance on the networked EWS system:

$$\begin{aligned} \tau = & \left(\frac{0.9825}{N} + 0.0175 \right) T_1 \\ & + 2.8 \times (N-1) \\ & + \frac{35.64}{N} \times (N-1) \\ & + 0.125 \times (N-1) \\ & + 1.2 \times (N-1) \quad \text{(seconds)} \end{aligned} \tag{2}$$

The first four terms have the same meaning as equation (1) except the larger communication time is due to the lower data transmission rate and the linear dependence on the size of the networked EWS system. Because there is no "global operations" in the networked EWS system, the fourth term of equation (2) is simply the message passing time from the remote system to the local system. The last term is the time needed to invoke the remote system procedure. Comparing these two equations we can find that the performance of the networked EWS system will soon be saturated because of the large communication time spent on transmitting the global data and statistics results.

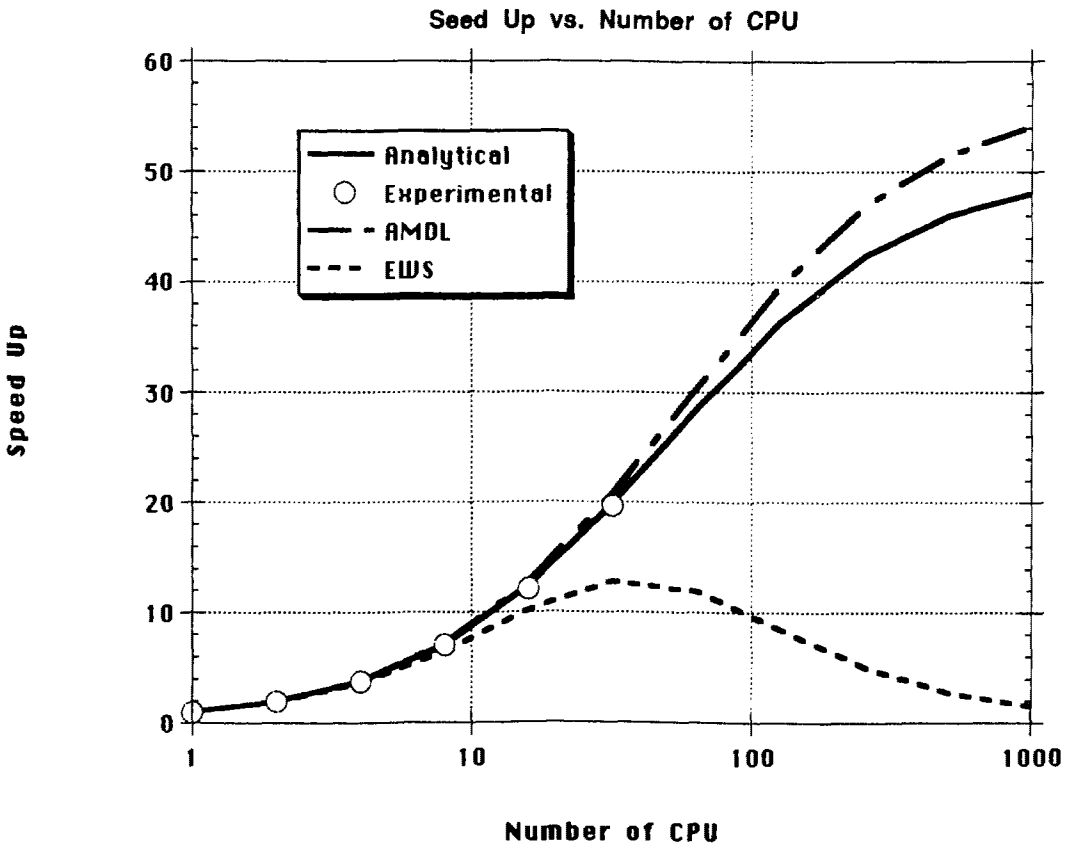

Figure 6: Comparison of the speed up for the ideal case and real cases.

5 Summary

Comparisons have been made between Intel iPSC/860 and the networked EWS system. The importance of communication time is clearly shown. Although one SPARC1$^+$ has almost the same performance as one i860 CPU for our MC simulation, the hypercube machine has better performance than the networked EWS system when more than 32 CPU's are used. For our MC simulation, the performance on the hypercube machine will not be saturated even using 128 nodes in contrast to the networked EWS system. The reason comes from the higher data transmission rate and the method of communication for the hypercube connection which is shown in equation (1). On the contrary, even though the networked EWS system has more than 32 workstations, its performance will be degraded by the communication time between the CPU's instead of the computational time. The comparison is shown in Figure 6. We can see that the maximum speed-up of the networked EWS system is for 32 CPU's and we can get 70% of this peak speed-up using 13 CPU's. To achieve 90% efficiency compared to the 32 node case, 24 CPU's must be used(apparently a high penalty compared to the 70% case). We can say that about 20 workstations in the networked EWS system is the optimized choice for this particular MC code and the present communication limits.

6 Acknowledgements

The authors would like to thank Samsung Electronics of Korea for the research support and Software Intelligence and System Technology Office(SISTO) of DARPA for providing the hardwares.

References

[1] F.Venturi et al. , *IEEE trans, on CAD*, vol. 8, p. 380, 1980.

[2] S. Sugino et al. , *IEDM Tech. Dig.*, p459, 1990.

[3] *"BEBOP"*—MONTE CARLO SIMULATOR (University of Bologna).

[4] John L. Hennessy and David A. Patterson, "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publishers, Inc., 1990.

[5] W. R. Stevens, "Unix Network Programming", Prentice Hall, 1990.