computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific Journal of Computing

A COMPARISON OF AGGREGATION/BROADCAST METHODS AND MULTICOMPUTER ARCHITECTURES, AND AN EXAMINATION OF THE COMMUNICATION OVERHEAD ON THE IBM PSERIES 655

Linda Markowsky

University of Maine, Orono, ME, USA
lmarkov@umcs.maine.edu

Abstract: *First, using a simulator, a detailed comparison of the butterfly and direct aggregation/broadcast methods on both hypercube and fully connected multicomputers, both with and without simulation of congestion, is made. Second, the communication overhead an IBM pSeries 655 is examined.*

Keywords: *aggregation/broadcast methods, multicomputer architecture, communication overhead, IBM pSeries 655*

1. INTRODUCTION

Parallel versions of Jacobi's Method using the message-passing paradigm were run both in a Pascal-based simulator [1] and on an IBM pSeries 655 (8 processors, 1.1GHz Power4, 16 GB memory, 36 GB scratch disk, 1Gbps Ethernet network) located at Boston University in Boston, MA, USA.

First, the results produced by the simulator are used to make a detailed comparison of the butterfly and direct aggregation/broadcast methods on both hypercube and fully-connected multicomputers, both with and without simulation of congestion.

Second, the results produced by the IBM pSeries 655 were used to make a detailed examination of the three phases of parallel computation—the distribution, computation, and communication phases—and to take a careful look at the communication overhead on the IBM pSeries 655.

The same algorithm—Jacobi's method, a well-known iterative algorithm used to solve diagonally dominant linear systems of equations—is used in both the simulator and pSeries portions of this study. For more on Jacobi's method and its implementation as a computer program, see, for example, [2] and [3].

2. SIMULATION: COMPARISON OF THE DIRECT AND BUTTERFLY AGGREGATION / BROADCAST METHODS ON A 5-DIMENSIONAL HYPERCUBE

Using a Pascal-based simulator [1], a system of 32 equations was solved on a 5-dimensional hypercube using both the direct and butterfly multiple broadcast/aggregation methods. In the direct method, each of the processors sends/receives a message to/from each of the others in each step of the algorithm, flooding the communication network with messages and causing congestion. The butterfly method, however, avoids congestion by restricting communication to direct links. The elapsed time and the speedup of the parallel region are depicted in Figs. 1a and 1b. In this simulation, the methods rank:

Fastest	Direct method – Congestion Off
↑	Butterfly Method – Congestion Off
↕	Butterfly Method – Congestion On
↓	Direct Method – Congestion On
Slowest	Sequential Program

When the simulator takes into account the cost of congestion, the butterfly method outperforms the direct method (because the butterfly method avoids congestion and the direct method does not). When the simulator ignores the cost of congestion, however, the simpler direct method outperforms the more complex butterfly method.

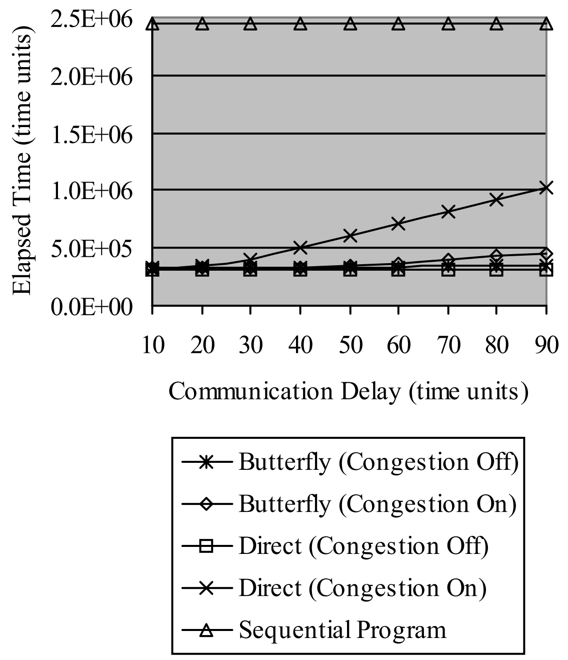

Fig. 1a – Elapsed Time vs. Communication Delay

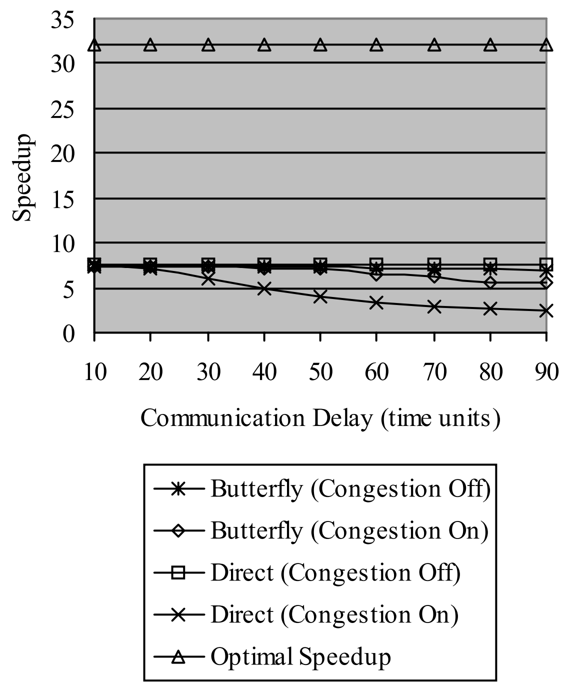

Fig. 1b – Speedup vs. Communication Delay

3. SIMULATION: COMPARISON OF THE DIRECT AND BUTTERFLY AGGREGATION/BROADCAST METHODS ON 2, 3, 4, AND 5-DIMENSIONAL HYPERCUBES

As in Lester [1], the direct and butterfly aggregation/broadcast methods (with congestion on) were compared for linear systems of size = 4, 8, 16, and 32 on hypercubes of dimension = 2, 3, 4, and 5, respectively.

On any size hypercube, the simpler direct method outperforms the butterfly method for small values of the delay, but as the delay increases, the performance of the direct method degrades more quickly than that of the butterfly method.

The crossover point—the value of the communication delay at which the butterfly method becomes superior—was found by graphing the performance of the direct and butterfly methods (Figs. 2a – 2d). These figures imply that for a small number of processors and a modest communication delay, the simpler direct method is superior, but that the more complex butterfly method becomes superior with either a larger number of processors or a longer communication delay. Furthermore, as the number of processors increased, the crossover point occurred at steadily smaller values of the communication delay:

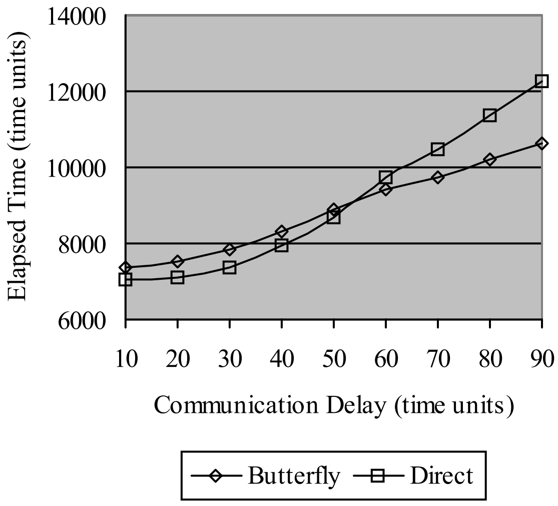

Fig. 2a – Comparison of Direct and Butterfly Methods on a 2-Dimensional Hypercube Congestion On

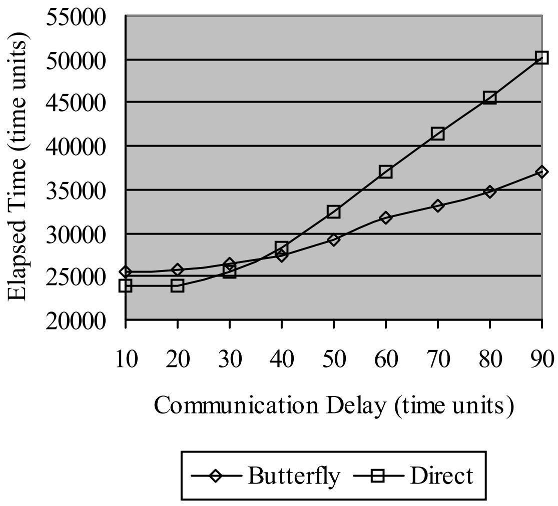

Fig. 2b – Comparison of Direct and Butterfly Methods on a 3-Dimensional Hypercube Congestion On

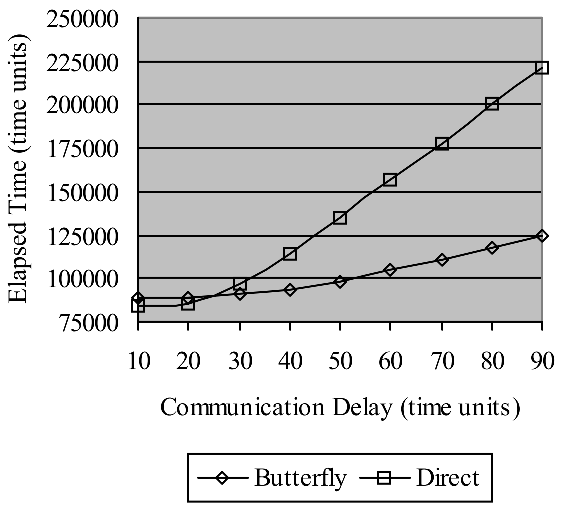

Fig. 2c – Comparison of Direct and Butterfly Methods on a 4-Dimensional Hypercube Congestion On

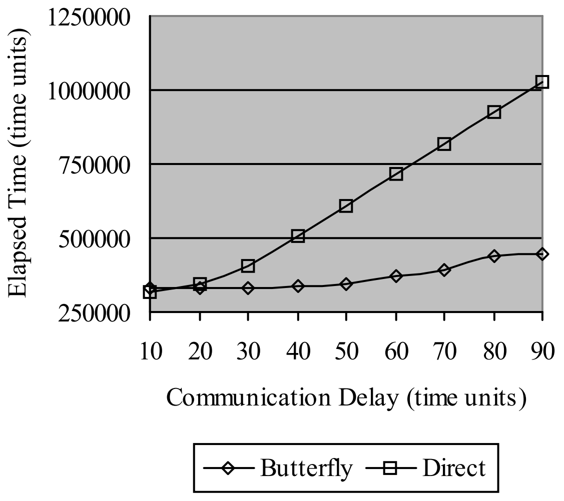

Fig. 2d – Comparison of Direct and Butterfly Methods on a 5-Dimensional Hypercube Congestion On

4. SIMULATION: COMPARISON OF THE HYPERCUBE AND FULLY CONNECTED MULTICOMPUTER ARCHITECTURES

In order to avoid the degradation resulting from congestion, we would like every multicomputer to have a direct link between each pair of processors, but unfortunately, the excessive cost of such systems forces us to settle for a compromise. One of the most common multicomputer architectures is the hypercube, which performs well at a reasonable cost. Using the simulator [1], a fully connected multicomputer (the ideal architecture) is compared to a hypercube (an affordable alternative).

Figs. 3a and 3b compare the running times of Jacobi's method on a 5-dimensional hypercube and a fully connected multicomputer. In Fig. 3a, the effect of congestion is ignored by the simulator, while in Fig. 3b, the effect of congestion is included.

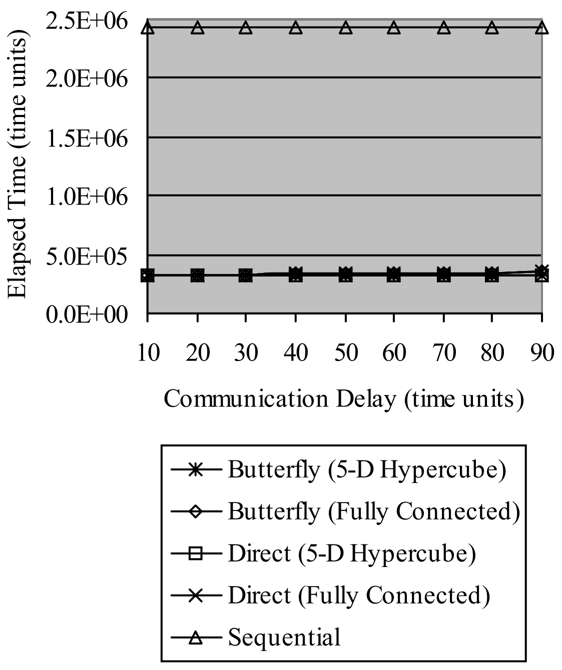

Fig. 3a – Elapsed Time vs. Communication Delay Size of Linear System = 32 Congestion Off

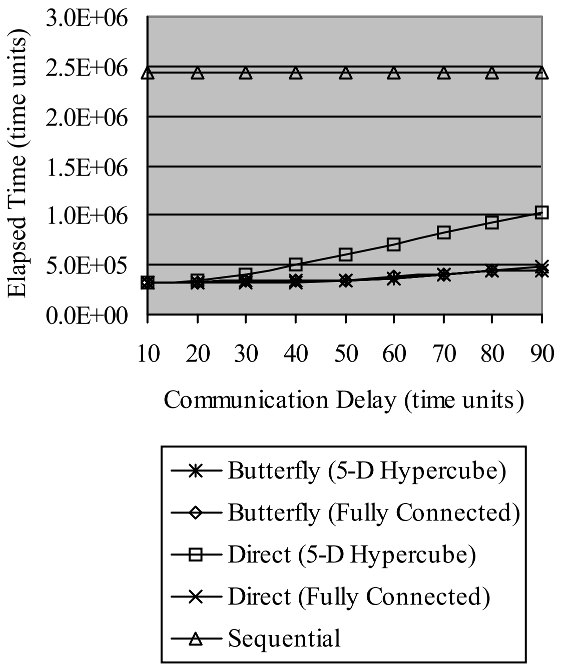

Fig. 3b – Elapsed Time vs. Communication Delay Size of Linear System = 32 Congestion On

Fig. 3a implies that with if congestion is ignored, the direct and butterfly methods perform comparably on each architecture. Although the results for the two architectures differ little, the fully connected architecture performed slightly better than the hypercube for both the butterfly and the direct methods, and the direct method performed better than the butterfly method, reflecting the relative complexity of the two aggregation and broadcast methods.

Comparing Figs. 3a and 3b illustrates the expected result: congestion degrades the performance of the direct aggregation/broadcast method running on a hypercube far more than the remaining method/architecture combinations. In the butterfly version running on a hypercube and in any program running on a fully connected architecture, communication is limited to neighboring processors. Thus, of the combinations tested, only the direct method running on the hypercube requires messages to be forwarded, incurring additional delays.

5. THE IBM PSERIES 655: ELAPSED TIME, SPEEDUP, AND PROCESSOR EFFICIENCY

A parallel version of Jacobi's Method using MPI was run on an IBM pSeries 655. The running time, speedup, and processor efficiency found for linear systems of sizes 32 – 1024 are summarized in Figs. 4a – 4c.

For small systems of equations, the communication overhead overwhelms the small granularity of the computation, degrading performance. For example, for N = 32, the parallel program slows down as the number of processors increases; for N = 64, the parallel program runs slower on 8 processors than it does on 4 processors; and for N = 128, the parallel program fails to speed up appreciably when the number of processors is increased from 4 to 8.

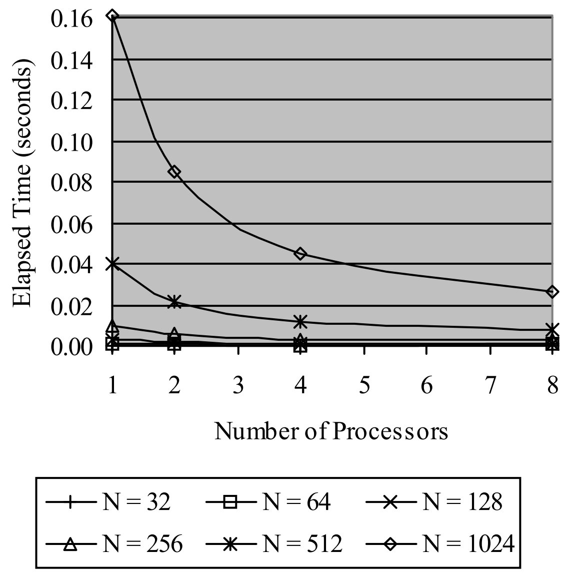

Fig. 4a – Elapsed Time vs. Number of Processors Size of Linear System = 32, 64, 128, 256, 512, 1024

As the granularity of the computation becomes large enough to warrant the cost of communication, the results improve: for N = 256, 512, and 1024, the speedup increases with increasing numbers of processors.

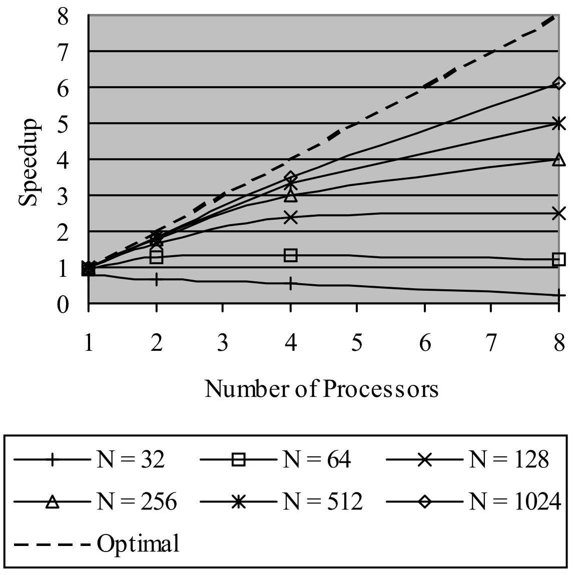

Fig. 4b – Speedup vs. Number of Processors Size of Linear System = 32, 64, 128, 256, 512, 1024

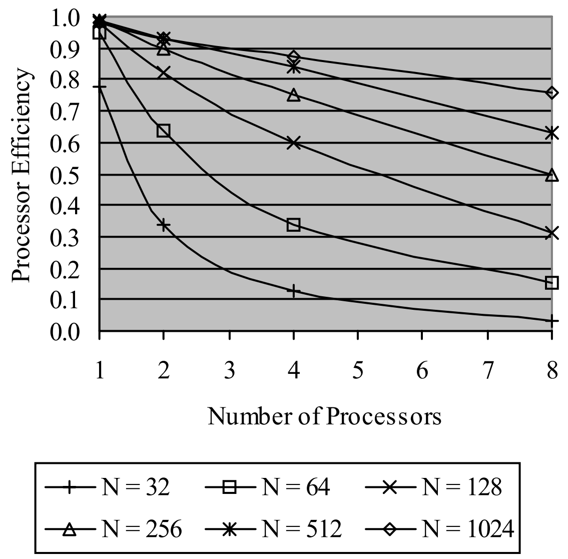

Fig. 4c – Processor Efficiency vs. Number of Processors Size of Linear System = 32, 64, 128, 256, 512, 1024

6. THE IBM PSERIES 655: THE DISTRIBUTION, COMMUNICATION, AND COMPUTATION PHASES

To further clarify the communication overhead, Figs. 5a–5c depict the time taken by the distribution, communication, and computation phases of the program for linear systems of size N = 64, 256, and 1024. The partition size is defined to be the number of unknowns computed by each processor.

In the distribution phase, the linear system's coefficient matrix and right-hand side are distributed to the processes using MPI_Scatter, and the initial guess of the solution vector is broadcast using MPI_Bcast. In the computation phase, the new values of the solution vector are calculated using the previous values. Finally, in the communication phase, the new values of the solution vector and the global Boolean value governing termination are collected and sent to all the processes using MPI_Allgather and MPI_Allreduce.

It is apparent from Figs. 5a – 5c that:

- The time taken by the computation phase is directly proportional to the size of the partition. This corresponds well with the algorithm: for a linear system of size N, the computation phase requires N multiplications and N – 1 additions for each of the k rows, where k = partition size = number of unknowns computed by each processor.

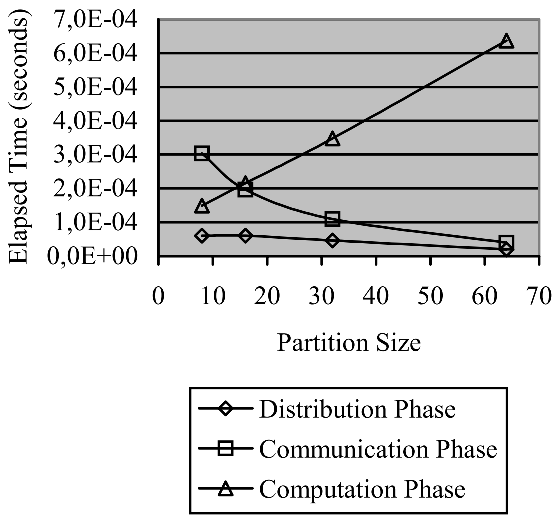

Fig. 5a – Size of Linear System = 64

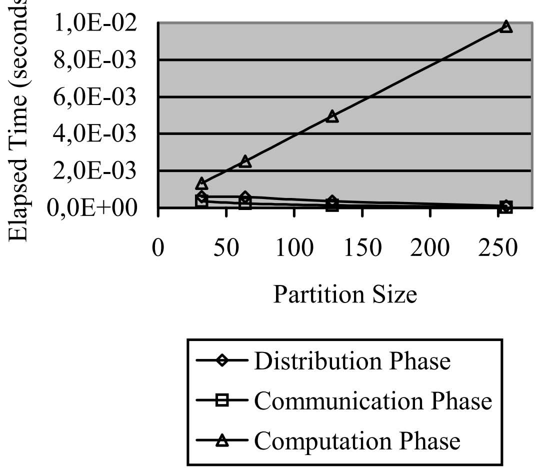

Fig. 5b – Size of Linear System = 256

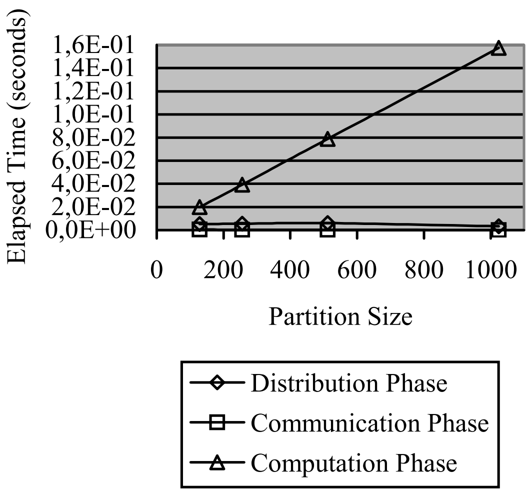

Fig. 5c – Size of Linear System = 1024

- The time taken by the distribution phase does not increase with the partition size. This reflects the data used to produce these graphs: N, the size of the system of equations, is constant for each graph, but the number of processors is not. Thus, for all partition sizes, the total number of items being scattered and broadcast in the distribution phase is constant: N * (N + 2).
- Both the time taken by the computation phase and the time taken by the distribution phase increase by an order of magnitude as N increases from 64 to 256, and again by an order of magnitude as N increases from 256 to 1024. Furthermore, the time taken by the distribution phase relative to the computation phase is nearly constant as N increases but plummets as the partition size increases.
- The communication phase consumes less time relative to the computation phase as both the partition size and N increase.

The time taken by the computation phase dominates the time taken by the distribution and communication phases, and this domination increases with both increasing N and increasing partition size. Consequently, as both N and the partition size increase, the communication overhead decreases, improving the performance of the program.

7. THE IBM PSERIES 655: A CLOSER LOOK AT THE COMMUNICATION PHASE AND COMMUNICATION OVERHEAD

The Communication Phase

Fig. 6a, which compares the communication phase for N = 64, 256, and 1024 (taken from Figs.

5a – 5c), implies that the time taken by the communication phase increases with increasing N, but by a factor less than N.

Fig. 6b depicts the same data as Fig. 6a, but with three changes:

- The horizontal axes differ.
- Smoothed lines connect the data points in Fig. 6a, but not in Fig. 6b.
- Logarithmic trend lines were added in Fig. 6b.

The close fit of the logarithmic trend lines in Fig. 6b strongly suggests that the most significant term in the cost of the communication phase is O(log(#processors)). That is, for this implementation of Jacobi's method running on the IBM pSeries 655, the cost of MPI_Allgather and MPI_Allreduce appears to be strongly related to the log of the number of processors involved in the global communication and less strongly related to the number of items contained in the messages.

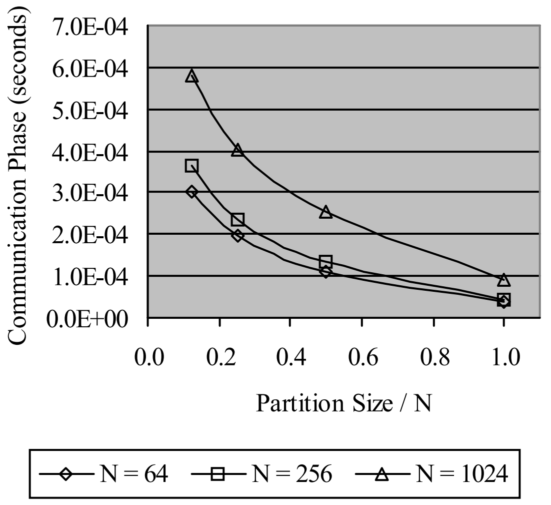

Fig. 6a – Comparison of the Communication Phase for Linear Systems of Size 64, 256, and 1024

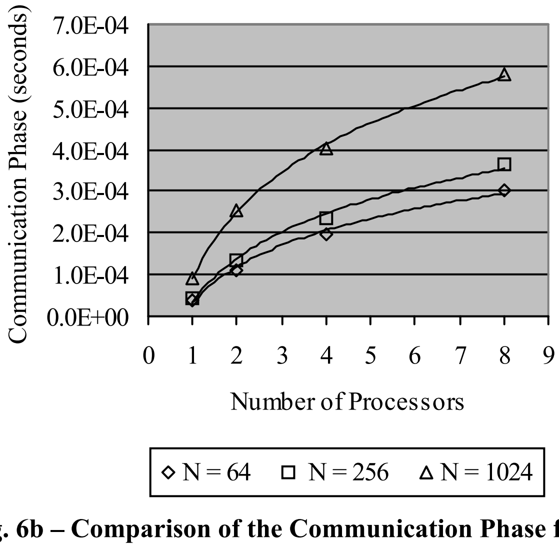

Fig. 6b – Comparison of the Communication Phase for Linear Systems of Size 64, 256, and 1024

Communication Overhead

Fig. 7a illustrates the relationship between the communication overhead and the number of processors.

Comparing Figs. 4c and 7a, it appears that for this program, especially for large N:

Processor Efficiency ≈ 1 – Communication Overhead

This implies that the time the time lost in synchronizing the processors following the computation phase of the algorithm is much less than the time spent in the communication phase—and that this difference increases with increasing N. Thus, for large systems of equations, the time spent synchronizing processes at the end of the computation phase is responsible for little of the lost efficiency, while the time spent sending and receiving messages is responsible for most of the lost efficiency.

Fig. 7a also illustrates that: (1) for a fixed number of processors, as N increases, the communication overhead decreases; and (2) for fixed N, as the number of processors increases, the communication overhead increases. Both of these observations are related to the granularity of the function responsible for computing the values of the unknowns. As the granularity of the function increases, the time spent in the computation phase increases relative to the time spent in the communication phase, decreasing the communication overhead.

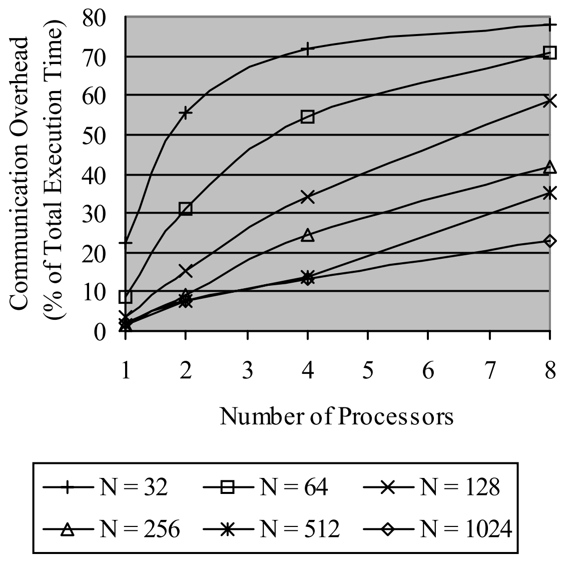

Fig. 7a
Communication Overhead vs. Number of Processors
Size of Linear System: 32, 64, 128, 256, 512, 1024

Fig. 7b illustrates the relationship of the communication overhead to the size of the linear system for various fixed partition sizes. The lines shown in Fig. 7b are logarithmic trend lines.

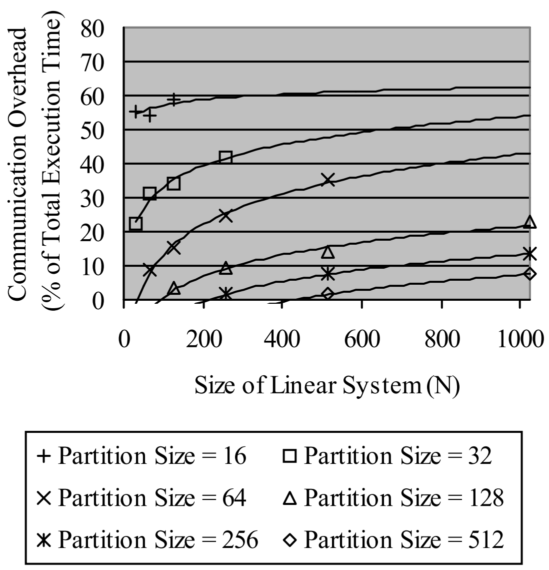

Fig. 7b
Communication Overhead vs. Size of Linear System for Fixed Partition Sizes

For a fixed value of N, the communication overhead decreases as the partition size increases, and although there are few data points, Fig. 7b suggests that for a fixed partition size, the communication overhead might be O(log(N)).

8. CONCLUSIONS

A well-known report [4] uses the LINPACK benchmark program to compare the performance of many computer systems by solving linear systems of equations. A second study [5] thoroughly benchmarks the IBM pSeries 690, and a third study [6] derives models for MPI primitives in order to facilitate estimating the communication overhead on the Fujitsu AP3000 using those models.

As in the references cited, this paper uses the results obtained by solving linear systems on a parallel computer, measures results on an IBM pSeries, and models MPI communication primitives. Unlike those studies, however, this paper focuses on three fundamental issues in parallel programming: aggregation/broadcast methods, multicomputer architectures, and communication overhead. Using a simulator and a software implementation of communication primitives, it was found that the simpler direct aggregation/broadcast method can outperform the more complex butterfly method on either a hypercube of small dimension or on a system with minimal communication delay, and on the IBM pSeries 655, the results suggest that:

- The cost of MPI_Allgather and MPI_Allreduce is O(log(#processors));
- Processor Efficiency ≈ 1 – Communication Overhead, i.e., the time lost in synchronizing the processors is far outweighed by the cost of communication; and
- The communication overhead when solving a linear system of size N using Jacobi's method is O(log(N)).

9. REFERENCES

[1] B. Lester. *The Art of Parallel Programming* (with accompanying Pascal-based simulator). Prentice-Hall. Englewood Cliffs, New Jersey, 1993. pp. 183-218.

[2] S. Leon. *Linear Algebra with Applications, Sixth Edition*. Prentice-Hall. Upper Saddle River, New Jersey, 2002. Supplemental Chaper 8, "Iterative Methods," www.prenhall.com/Leon.

[3] P. Lee. "Techniques for Compiling Programs on Distributed Memory Multicomputers," *Parallel Computing* 21 (12) (1995). pp. 1895-1923.

[4] J. Dongarra. "Performance of Various Computers Using Standard Linear Equations Software," www.netlib.org/benchmark/performance.ps, June, 2005.

[5] P. Worley. T. Dunigan, Jr. M. Fahey. J. White. A. Bland. "Early Evaluation of the IBM p690," *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Baltimore, Maryland, November 2002, pp. 1-21.

[6] J. Tourino. R. Doallo. "Modeling MPI Collective Communications on the AP3000 Multicomputer," *Recent Advances in Parallel Virtual Machine and Message Passing Interface: Proceedings (6th European PVM/MPI Users' Group Meeting)*, Barcelona, Spain, September 1999, pp. 133-140.

[7] J. Fastook. Personal communication.

***Linda Markowsky** holds a B.A. in Mathematics from the University of Maine, USA, where she is currently pursuing a Ph.D. in Computer Science. Her areas of interest include algorithms and the theory of computation.*