1-1-1996

Distributed parallelism: impacts on GIS and collaborative spatial decision-making

Marc P Armstrong
University of Iowa

Richard J Marciano

DISTRIBUTED PARALLELISM: IMPACTS ON GIS AND COLLABORATIVE SPATIAL DECISION-MAKING

Marc P. Armstrong
Department of Geography and Program in Applied Mathematical and Computational Sciences
316 Jessup Hall
The University of Iowa
Iowa City, IA 52242
marc-armstrong@uiowa.edu

Richard J. Marciano
Enabling Technologies
San Diego Supercomputer Center
P.O. Box 85608
San Diego, CA 92186-5608
marciano@sdsc.edu

ABSTRACT

Because GIS software is used most effectively to address public policy problems if several alternatives, each representing the interests of different stake-holders, can be generated and evaluated during a meeting, the response time of GIS-based models can assume considerable practical importance. In this paper we present results from research that evaluates a new approach to the analysis of computationally-complex GIS-based models, with the aim of improving the performance of networked, collaborative decision-making environments. The first element of the paper centers on the use of a network of workstations (NOW) to support parallel interpolation for a large problem (1024 x 1024 cells with 10,000 control points). Computational experiments using this approach achieve a level of performance that compares favorably with a dedicated parallel supercomputer (Cray T3D). The experiments also show general promise for a large class of GIS algorithms that can be divided into large sub-problems. These results are especially significant since networked workstations present a ready source of computing power that can be accessed and used during the period of a typical group meeting.

1.0 INTRODUCTION

When groups of people are brought together to explore the different dimensions of complex public policy issues, meetings are normally scheduled to have a finite (and often short) duration. During such meetings GIS-based approaches can be used to support individual and collective deliberation about the geographical aspects of semi-structured public policy problems by enabling group members to visualize and better understand the effects of applying different sets of criteria and weights. If a GIS model can be executed in near-real-time (e.g., less than five seconds) then decision-making processes can proceed in a way that is unconstrained by computation. In contrast, if a substantial amount of time is required to create each prospective solution to a problem (see, for example, Densham and Armstrong, 1994), the number of alternatives that can be generated and evaluated during each meeting is greatly diminished. While it is not our intent to advance the simple argument that "more is better", when a variety of different aspects to a problem can be explored, and a wide range of criteria examined, it is *possible* that different interest groups will be able to have their views represented during a discussion. Consequently, performance is important since meeting time need not be devoted to a discussion of what small set of privileged alternatives will be generated and evaluated.

Please note that GIS is not a panacea that will cure all social ills. But, when applied to many types of problems, and when augmented with improvements in performance and functionality that introduce it into collaborative and participatory settings, GIS software can be used to embrace alternative views on the ways that public policy problems can be addressed and solved Armstrong, 1994). In so doing, GIS can throw off its deterministic shackles and be applied in ways that are consistent with democratic access to technology. The purpose of this paper, therefore, is to demonstrate how computationally-complex GIS-based analyses, especially as they relate to improving the performance of networked, collaborative decision-making environments, can be implemented using a network of workstations (NOW) approach (Anderson *et al.*, 1995). In particular, we show how a NOW environment is used for parallel interpolation of a large problem (1024 x 1024 cells with 10,000 control points) using an algorithm reported by Clarke (1990: 207; see also Armstrong and Marciano, 1994; 1996). We contrast the NOW results with those obtained using a dedicated parallel supercomputer (a Cray T3D).

2.0 BACKGROUND: MPI AS AN EMERGING PARALLEL STANDARD

Parallel processing has been plagued by a severe case of the "Tower of Babel Syndrome" because most parallel computer manufacturers have developed architecture-specific language extensions that have inhibited open exchange of code among different computer systems and architectures (c.f., Armstrong, Pavlik and Marciano, 1994a; 1994b; and Armstrong and Marciano, 1995). To make matters worse, many parallel computing firms (e.g., Kendall Square, Thinking Machines, MasPar) have "reorganized", leaving in their wake a smattering of "dead language" extensions. In some respects, these developments have "paralleled" those in computer graphics prior to the establishment of high-level device-independent graphics libraries such as GKS (recall long-suppressed memories about Calcomp and Tektronix plotting libraries at your own risk!).

MPI (message passing interface) is a high-level standard that attempts to accomplish some degree of independence between parallel application code and the machine on which it is implemented. Developed with support from a consortium of private companies and public agencies (including the U.S. National Science Foundation and the European Commission), MPI enables users to write parallel code in FORTRAN 77 or C for a wide range of parallel systems, including dedicated parallel computers as well as networked workstations (see Dongarra *et al.*, 1996; Snir *et al.*, 1996). This portability serves to overcome the system-specific approach to parallel code development that has, in part, hindered the widespread adoption of parallel programming paradigms. A key organizing principle of MPI is the specification of a range of flexible communication strategies across heterogeneous system elements, including point-to-point messages between individual processors as well as message "broadcasts" to all, or groups of, processes. MPI also provides an interface between parallel code and run-time performance monitoring and error handling tools available in different parallel computing environments. This feature helps users to debug and assess the performance characteristics of their parallel programs.

A major feature of MPI's point-to-point communication capability is its bundling together of data with typing information (Dongarra *et al.*, 1996). When typing is supported in this way, data can be passed among heterogeneous elements in different architectures in ways that ensure their correct conversion in a program. MPI also supports a set of send and receive semantics. For example, some systems have a greater buffering capacity than others, and, thus, the interaction between sending a message and its receipt at a destination could vary among architectures. MPI provides a set of blocking facilities that clarify communication ambiguities.

3.0 AN MPI VERSION OF THE CLARKE INTERPOLATION ALGORITHM

In the computational experiments reported in this paper a self-scheduling or *manager-worker* task-scheduling algorithm was employed. In this approach a single process is designated as a manager that maintains an inventory of the pool of tasks that need to be completed by the algorithm and it distributes tasks when a request is sent from an idle worker process. The set of tasks was created by applying the interpolation algorithm to a subsection of the entire grid, in this case a vertical tile, that was specified to be of size 1024 x 8 (eight grid columns). This makes for an initial pool of 128 tasks. Because it is often difficult to predict the amount of work that is required to complete the interpolation computations for each tile, due, for example, to variation in control point density, we chose to create a large pool of such tiles, so that when a processor completes its task it could immediately be put back to work. Processors essentially compete for available tasks, and are assigned the next available one.

This particular domain decomposition strategy can be difficult to implement efficiently because even though the Clarke algorithm is applied locally, global information (information outside of a local tile) may be required to complete some computations. We addressed this issue by increasing the amount of data that was sent to each processor. To reduce the amount of interprocessor communication required, we replicate the entire initialized grid over all processors. The Clarke algorithm can then be applied locally without having to interrogate neighboring processors, because all required information is now stored locally. The results from each completed tile are sent back to the manager for final logging.

This is illustrated in the following pseudo-code version of the Clarke algorithm that uses MPI functions ***send***, ***receive***, and ***broadcast*** (actually called *MPI_SEND*, *MPI_RECV*, and *MPI_BCAST*). In this pseudo-code, *NUM_PROCS* refers to the total number of processors used at run time. In the implementation of this algorithm we actually use two arrays: one for the grid, and a second Boolean array of the same size. This Boolean array is used as a binary mask that maintains control over completed (interpolated) tiles. Thus, the total amount of memory used in our implementation is 2 x 1024*1024 integer values. Also, the manager uses a special tag field in MPI to signal that all jobs have been completed. This optional tag value is checked in the worker code, so that all workers can gracefully exit when the task pool has been depleted. Finally, by default, we use blocking receives in the manager and worker codes.

```
manager node 0:
        read grid data from disk
        read number of k-neighbors for interpolation
        NUM_JOBS = 128
        broadcast entire grid
        broadcast k-neighbors value
        broadcast NUM_JOBS value
        NEXT_JOB = 1
        for i=1 .. NUM_PROCS - 1
                proc_num = i
                send NEXT_JOB to processor proc_num
                NEXT_JOB = NEXT_JOB + 1
        endfor
        for i=1 .. NUM_JOBS
                receive from any of the worker nodes its completed vertical tile
                send new NEXT_JOB value to that now idle worker node
                NEXT_JOB = NEXT_JOB + 1
        endfor

worker node N:
        receive NEXT_JOB value from manager node
        do interpolation for a tile of grid using the Clarke algorithm
        send that interpolated tile back to the manager node
```

4.0 DESCRIPTION OF THE COMPUTER SYSTEMS

Two computer systems were used to compute the results of the MPI implementation of the Clarke algorithm. The first is a workstation cluster located at the San Diego Supercomputer Center (SDSC). A cluster is a group of systems that are closely coupled into a single entity in order to provide fast, uninterrupted computing service. Instead of custom components, clusters combine off-the-shelf components into cooperative teams, that are able to approach or surpass the capabilities of mainframes, supercomputers, and fault tolerant systems, but at much lower costs. The second system used in the computation of results is a dedicated parallel machine, a Cray T3D system, that is also located at SDSC.

4.1 Alpha Cluster

The Alpha Cluster (or informally "alpha farm") consists of eight DEC Alpha processors that are connected by a GIGAswitch/FDDI network. Each node in the cluster is a DEC 3000 AXP Model 400/400S system with a DECchip 21064 Alpha AXP microprocessor running at 133.33 MHz. With the maximum floating point speed of each processor at 133Mflops, a total aggregate capability of 1.07 Gflops is theoretically possible. Each processor in the system is configured with 128 MB of memory. A key element in the performance of the system is the speed at which messages can be passed among the processors. The cluster's GIGAswitch/FDDI network uses high-performance crossbar switching technology. Whereas FDDI networks are limited to the 100 Mb/s of a single ring, GIGAswitch/FDDI breaks this barrier by creating a switched FDDI network where the aggregate bandwidth is much greater than the individual link bandwidth. The network supports a point-to-point maximum bandwidth of 12.5 Mbytes/sec with maximum aggregate bandwidth of 450 Mbytes/sec (3.6Gb/s).

4.2 Cray T3D

The Cray T3D has 128 processing elements (PEs), each of which includes a DEC Alpha EV4 (21064) processor (150 mflops) and 64 megabytes of memory. The system acts as a computational server to a front-end machine, a Cray C90 in this case, that handles all system calls and network communications. T3D processors are organized into nodes consisting of two PEs each. The communication network is configured as a 3-D torus that connects each PE in a bi-directional manner to its six neighbors (north, south, east, west, up, and down). PEs on the edge of the mesh are connected to PEs on the opposite edge, creating the torus structure. Even though memory is physically distributed among processors, it is globally addressable (any PE can address any memory location in the system), making the T3D a "shared distributed

memory" system. This characteristic allows the user to choose from among several high level programming models, such as MPI, PVM, SHMEM, and CRAFT.

5.0 RESULTS

The same MPI code was used for both the Alpha Cluster and the Cray T3D. In all cases a 1024 x 1024 grid was interpolated from 10,000 randomly distributed control points and the number of *k*-nearest neighbors used for the interpolation of each grid cell was held constant at three (k=3). Note, however, that the number of processors available and used for the Alpha Cluster experiments ranged from one to eight, while the runs completed on the larger Cray T3D used up to 128 processors.

5.1 Alpha Cluster Results

The results obtained with the Alpha Cluster demonstrate that MPI can be used with networked workstations to decrease run times substantially when applied to problems of this type (Table 1). In fact, run times show a decline from nearly two minutes to 19 seconds when the entire suite of eight processors is used. Note, however, that the single processor result was obtained using code that was highly optimized to execute on a single Alpha workstation. It is clear from these results that this level of optimization led to a run time that is superior to the use of two processors when MPI was run on this system. When three processors are used, however, the run time is reduced to approximately half that obtained with the optimized code, and run times continue to drop monotonically as workstations are added to the computation of results (Figure 1).

Though these results provide support for the viability of this approach, it is important to consider that they were conducted at an "off peak" time (a Sunday morning) when we were granted "sole user access" privileges. When a reduced set of the same tests was conducted on a more heavily loaded system, performance suffered, with all parallel runs taking nearly three times as long to execute (see right-hand column of Table 1). In such cases the parallel processes are forced to compete for available cycles with other currently executing jobs. Though the *exact* times obtained are a function of the other jobs that were being run, these results provide an indication about the general impact of competition on results.

Table 1. Run times using the DEC alpha farm system. Run times are in seconds.

Number of Processors	Run Time (Dedicated)	Run Time (Loaded)
1*	112.7	116.1
2	125.4	524.5
3	62.8	
4	42.9	130.4
5	31.7	
6	25.9	82.0
7	21.7	
8	18.9	65.0

* Single processor result was run with maximum level of code optimization (f77 -O4)

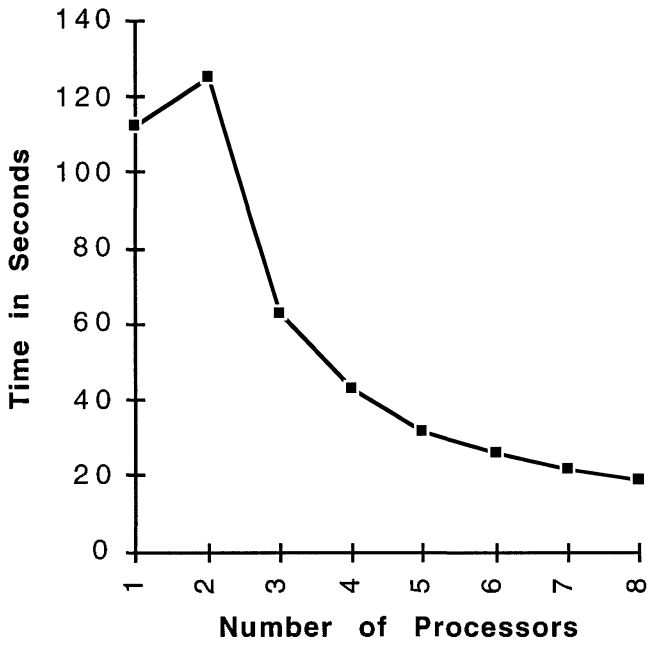

Figure 1. Run time in seconds for one to eight processors using the Alpha Cluster.

When the speedup and efficiency values obtained with the Alpha Cluster (Table 2) are examined it is evident that the parallel overhead is not recouped when two processors are used, as the speedup is less than one. However, after this initial dip, speedups continue to climb across the range of processors (Figure 2) and the efficiencies actually increase monotonically as the number of processors is increased. This somewhat unusual result is partly explained by the stringent sequential benchmark used to compute the speedups. In general, however, the parallel results demonstrate excellent scalability in this environment as borne out in the efficiency values for the largest configurations of processors. This increase in efficiency provides an initial indication that the use of additional processors would continue to enhance performance.

Table 2. Speedups and efficiencies for the results (dedicated access) reported in Table 1.

Number of Processors	Speedup	Efficiency %
1	1.0	100
2	0.9	45
3	1.8	60
4	2.6	66
5	3.6	71
6	4.4	73
7	5.2	74
8	6.0	75

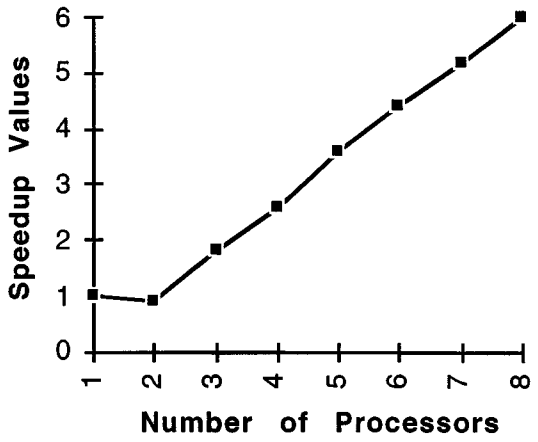

Figure 2. Speedup values for the results reported in Table 1.

5.2 Cray T3D Results

The results for the T3D runs also exhibit a monotonic decrease in run time as additional processors are added to the computation of results. While more than three minutes are required when one (C90) processor is used (see note, Table 3), and nearly three minutes are required when two T3D processors are used, the run time was reduced to 25 seconds when eight processors were applied to the problem. This can be contrasted with the time of 18.9 seconds obtained with the Alpha cluster. When the full complement of 128 processors was used, however, the run time was reduced to 3.1 seconds, thus satisfying our criterion of less than five seconds for near-real-time performance. Speedup and efficiency values indicate that the additional processors are being used effectively for up to 32 processors (Table 4). While the efficiency values exhibit a promising trend as they increase from 57% when two processors are used, to a maximum of 96% with 16 processors, when 64 and 128 processors are used, efficiencies begin to plummet (Figure 4) since the observed decrease in run time is small relative to the rapidly increasing (doubling) number of processors.

Table 3. Run times using the Cray T3D and the Cray C90. Run times are in seconds.

Number of Processors	Run Time
1	195.1 **
1	167.9 *
2	172.1
4	58.2
8	25.6
16	12.7
32	7.1
64	4.4
128	3.1

** Time (seconds) for a single CPU on a CRAY C90 (front end supercomputer for the T3D). Since the T3D is not designed to give optimal timing results for a single processor (there are two processors per board), parallel results are often compared to those obtained using a C90 processor.

* Time (seconds) for 1 processor on the T3D.

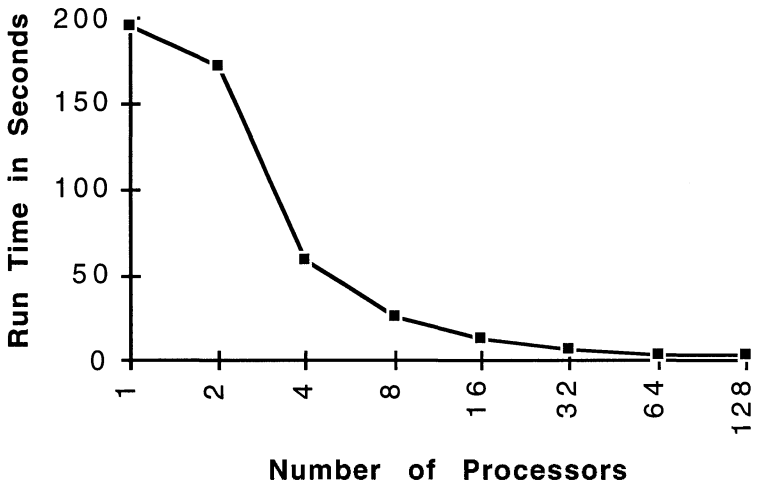

Figure 3. Run time in seconds for the Cray T3D. Note non-linear increase in the X axis.

Table 4. Speedups and efficiencies for the results reported in Table 3.

Number of Processors	Speedup	Efficiency
1	1.00	100
2	1.13	57
4	3.35	84
8	7.62	95
16	15.36	96
32	27.48	86
64	44.34	69
128	62.94	49

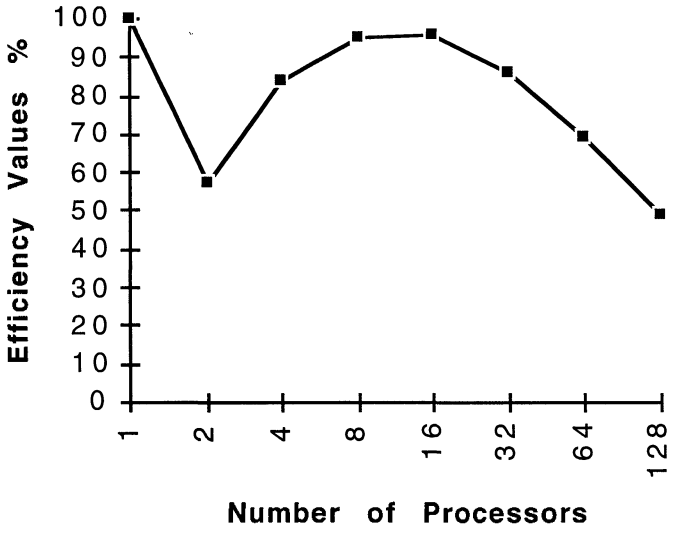

Figure 4. Efficiency values (%) for the Cray T3D results reported in Table 3. Note non-linear increase in the X axis.

6.0 DISCUSSION

It should be noted that the Alpha Cluster NOW technology is similar to the MPP T3D; in each system the PEs are off-the-shelf workstation processors. Table 5 shows a comparison of the performance of each system when applied to the interpolation problem. When two processors are used, the Alpha farm provides a 37% advantage in run time performance. This level of advantage is maintained when 4 and 8 processors are used as well. Because this level of performance is achieved with commodity-class technology, the price-performance of the Alpha cluster is quite high.

Table 5. Comparison of the actual and relative performance of the two systems. Time in seconds.

Number of PEs	Alpha Cluster	T3D	% Δ
2	125.4	172.1	37
4	42.9	58.2	36
8	18.9	25.6	35

The T3D results demonstrate that the MPI implementation scales beyond 8 processors. This reaffirms the indication that there is opportunity to reap performance benefits from larger clusters, and that such approaches to high performance computing are cost-effective. This makes the MPI approach worthwhile in terms of both performance and portability.

Several possible strategies could be used to improve the results reported in this paper. First, it is possible to implement the MPI program such that the manager node would be self-assigned some amount of interpolation processing, rather than waiting (idly) for worker results to be returned, by using a non-blocking receive or by polling for completed worker tasks. In the current implementation, the manager node declares an additional receiving buffer to copy out the completed data tile, and this is a sub-optimal use of memory. Redesigning this portion of the program could yield performance improvements. Finally, it is possible, and perhaps desirable, to make the number of tasks a dynamic quantity, and to vary *num_jobs* and consequently, task (tile) size. Each of these changes would also possibly improve performance and we are investigating these effects in ongoing research.

7.0 CONCLUSIONS

Based on a division of an interpolation algorithm into a set of mutually independent processes, computational experiments using NOW achieves a level of performance that exceeds that of the current generation of workstations. In fact, the results compare favorably with those obtained when a dedicated parallel supercomputer (Cray T3D) was applied to the same problem. The experiments show general promise for a large class of (grid cell) GIS algorithms that can be divided into large sub-problems. These results are especially significant given that workstation-based computing now dominates GIS. These workstations, which are also the predominant technology used in collaborative decision-support environments present a ready source of spare computing power that can be accessed and used during the period of a typical group meeting. Future work should concentrate on testing and evaluating the NOW approach using conventionally-connected (ether, fast-ether, or ATM) heterogeneous assemblages of workstations to assess the effects on performance of different tile sizes and load-balancing strategies.

ACKNOWLEDGMENTS

This research was made possible by computer time provided by the San Diego Supercomputer Center. MPA would like to acknowledge support provided by the National Center for Geographic Information and Analysis; this paper is a contribution to Initiative 17 (Collaborative Spatial Decision-Making) of the NCGIA which is supported by a grant from the National Science Foundation SBR-88-10917. Thanks also to Bart Cramer for his assistance at The University of Iowa, to Cindy Zheng and Ken Stube at SDSC for scheduling the resources required to complete the work reported in this paper, and to Richard Frost (SDSC) who installed the latest version of MPI on the T3D, and in the process, located and fixed some nasty bugs.

REFERENCES

Anderson, T.E., Culler, D.E., Patterson, D.A. and the NOW team. 1995. A case for NOW (Networks of Workstations). *IEEE Micro*, February, 54-63.

Armstrong, M.P. 1994. Requirements for the development of GIS-based group decision support systems. *Journal of the American Society for Information Science*, **45** (9): 669-677.

Armstrong, M.P. and Marciano, R. 1994. Inverse distance weighted spatial interpolation using parallel supercomputers. *Photogrammetric Engineering and Remote Sensing*, **60** (9): 1097-1103.

Armstrong, M.P., Pavlik, C.E. and Marciano, R. 1994a. Parallel processing of spatial statistics. *Computers & Geosciences,* **20** (2): 91-104.

Armstrong, M.P., Pavlik, C.E. and Marciano, R. 1994b. Experiments in the measurement of spatial association using a parallel supercomputer. *Geographical Systems*, **1** (4): 267-288.

Armstrong, M.P. and Marciano, R. 1995. Massively parallel processing of spatial statistics. *International Journal of Geographical Information Systems*, **9** (2): 169-189.

Armstrong, M.P. and Marciano, R. 1996. Local interpolation using a distributed parallel supercomputer. *International Journal of Geographical Information Systems*, **10** (5): 713-729.

Clarke, K.C. 1990. *Analytical and Computer Cartography*. Englewood Cliffs, NJ: Prentice-Hall.

Densham, P.J. and Armstrong, M.P. 1994. A heterogeneous processing approach to spatial decision support systems. In T.C. Waugh and R.G. Healey, eds. *Advances in GIS Research: Proceedings of the Sixth International Symposium on Spatial Data Handling*, Volume 1. London, UK: Taylor and Francis Publishers, pp. 29-45.

Dongarra, J.J., Otto, S.W., Snir, M., and Walker, D. 1996. A message passing standard for MPP and workstations. *Communications of the Association for Computing Machinery*, **39** (7): 84-90.

Rokos, D. and Armstrong, M.P. 1996. Using Linda to compute spatial autocorrelation in parallel. *Computers & Geosciences*, **22** (5): 425-432.

Snir, M., Otto, S., Hess-Lederman, S., Walker, D., and Dongarra, J. 1996. *MPI: The Complete Reference*. Cambridge, MA: MIT Press.

Also at: http://www.netlib.org/utk/papers/mpi-book/mpi-book.html