1-1-1998

A Network of Workstations (NOW) Approach to Spatial Data Analysis: The Case of Distributed Parallel Interpolation

Marc P. Armstrong
University of Iowa

Richard J. Marciano
University of California at San Diego

A Network of Workstations (NOW) Approach to Spatial Data Analysis: The Case of Distributed Parallel Interpolation

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 Group
San Diego Supercomputer Center
University of California at San Diego
10100 John Jay Hopkins Drive
La Jolla, CA 92093-0505
marciano@sdsc.edu

Abstract

Parallel processing has been widely demonstrated to be of significant value in reducing computation times for a range of geographical problems. Most papers have reported on the use of exotic and difficult-to-access dedicated parallel computing systems. However, researchers have begun to develop ways of exploiting collections of readily available workstations to implement off-the-shelf parallel computer systems. In this paper we report on the use of a Network of Workstations (NOW) environment that is applied to a typical spatial analysis problem– inverse distance weighted interpolation. Our results indicate that while a collection of networked workstations can be used to reduce run times from minutes to a few seconds, the implementation that we explored was not able to make fully efficient use of the available computing resources.

1.0 Introduction

GIS software has matured during the past decade. Methods of spatial analysis continue to increase in number within many widely available commercial software products, and these methods have also begun to be compartmentalized into modular toolboxes by software vendors. As the range of spatial analytical capabilities expands, it is clear that many of the new methods are also increasing in computational complexity. Some methods have now ceased to rely on assumptions of normality and employ problem-specific distributions, computed on the fly, that are used to support inference. Other methods, some employing optimization procedures that are computationally complex (the traveling salesperson problem is, after all, a network spatial analysis problem) and large amounts of interpoint-distance calculations, are also being applied with increasing frequency. Exacerbating these developments in computational complexity is a commensurate increase in the availability of disaggregated data. As geographical databases continue to grow in size, and as these databases are subjected to new analytical methods

that require considerable amounts of computation, the problem of providing response times to users that are acceptable to them also continues to grow. The purpose of this paper is to demonstrate how a network of conventional workstations can be used to implement a high performance parallel computing environment that can compute results to spatial analytical problems in a time-frame that is acceptable to most users. In particular, we use a testbed of workstations (called NOW for **n**etwork **o**f **w**orkstations) that is located at the University of California at Berkeley to implement an inverse-distance weighted interpolation algorithm and to evaluate its performance across a range of processors (maximum of 55). We also compare the results obtained with the NOW environment to results obtained in previous research.

2.0 Previous Applications of Parallel Processing to Spatial Problems

Parallel processing has begun to attract considerable attention in the GIS research community as a way to combat response time latencies inherent in spatial data analysis. Early work focused, among other things, on type placement problems, overlay analysis and interpolation. In some senses, a recent book (Healey *et al.*, 1997) has established a "datum" on which future work might build. The bulk of this previous work, however, has been performed using high-end (and extremely expensive and scarce) computer systems. Though this previous research has demonstrated that parallelism is broadly successful when applied to geographical problems (but c.f. Armstrong and Marciano, 1997), the use of exotic parallel systems precluded routine use: accounts needed to be applied for and access was limited. Recent work has begun to demonstrate that networks of "off-the-shelf" commodity-class computers are able to approach the level of performance achieved by state-of-the-art systems of a few years ago. Anderson *et al.*, (1995) make a convincing case for the network of workstations (NOW) approach (URL 1) and other authors have advanced similar arguments (e.g., Sterling, 1996). This approach has begun to gain widespread acceptance and is the cornerstone of the high performance computing environments that are under development as part of NCSA (Kennedy *et al.*, 1997). Consequently, it is important to begin to explore the use of such environments in spatial data handling applications.

In this paper we examine a problem that we have explored in previous work, using a variety of parallel architectures. Consequently, we have accumulated a considerable amount of information about the performance of the problem we are investigating. In particular, we use as our computational problem a parallel inverse-distance weighted interpolation algorithm published originally by Clarke (1990) as an educational demonstration program. We have implemented this code in a SIMD environment (up to 16K processors on a MasPar; Armstrong and Marciano, 1997), in various MIMD architectures including a conventional shared memory machine (Encore; Armstrong and Marciano, 1994), several distributed memory architectures (Cray T3D and DEC Alpha cluster of workstations (Armstrong

and Marciano, 1996a), and a type of distributed-shared memory (DSM) architecture (the KSR-1 system which uses a hierarchical cache-based memory system; Armstrong and Marciano, 1996b). Though our efforts were not particularly well-rewarded in the case of the SIMD architecture, we were able to extract considerable improvements in performance from the other architectures. In this paper we explore the use of a less formal collection of workstations that exists in a configuration that roughly approximates in functionality that which is available in many multiple-workstation GIS shops.

3.0 The NOW Environment

The Berkeley NOW project aims at obtaining the levels of low latency and high bandwidth communication traditionally associated with tightly-integrated MPP systems, by using of-the-shelf processors and high-speed communication switches. The Berkeley NOW hardware environment is constructed from 105 Sun Ultra workstations that each operate at 167 MHz, with 512 KB level-2 cache, 128 MB of RAM, and 4.6 GB of local disk space (see Culler *et al.*, 1997). Each workstation also has a Myricon network interface card (NIC) (with its own 37.5 MHz processor and 256 KB of SRAM) on the SBus (25 MHz). The NOW network uses multiple stages of Myricon switches in a variant of a fat-tree topology: five hosts are connected at the leaf level by a switch; seven of these five-machine clusters are linked at the next level to form 35 workstation sub-clusters, and three of the sub-clusters are wired together to form the NOW.

Each node or workstation runs a complete independent Solaris Unix operating system. NOW extends Solaris at each node to support global operations through a software layer called GLUnix (Global Layer Unix). GLUnix provides process management at the cluster level allowing services such as load balancing at job startup: the single least loaded node is chosen from the node pool for a sequential execution and the set of most available nodes is selected for a parallel execution. A set of Unix-like commands allows flexible process management control, such as:

- *glurun*: run sequential or parallel applications on the cluster, for example, glurun -4 a.out
- *glukill*: sends a signal to a GLUnix job
- *glups*: queries currently running jobs
- *glustat*: queries current status of all NOW nodes
- *glupart*: manages naming partitions and helps with the dedicated reservation of such a named sub-cluster

Active Messages are the basic communication primitives on NOW (Culler *et al.*, 1997). They correspond to a basically simplified remote procedure call model. The newest implementation AM-II, is integrated with POSIX threads. Three classes of AMs are implemented. Short messages send

eight 32-bit arguments that are executed by a remote handler on a destination node. Medium messages point to a data buffer (size range 128 bytes to 8KB) for the handler to use. Bulk messages use a memory-to-memory copy before the handler is invoked.

4.0 MPI

MPI (message passing interface) is an attempt to implement a standard that supports independence between parallel application code and a machine implementation. It enables users to implement parallel code in FORTRAN 77 or C for parallel computers as well as for networked workstations (see Dongarra *et al.*, 1996; Snir *et al.*, 1996). In user-developed code, MPI supports the specification of different communication strategies across heterogeneous system elements, including point-to-point messages between individual processors as well as "broadcasts" to all, or a subset of, processors. When point-to-point communication is used, MPI bundles data with typing information (Dongarra *et al.*, 1996). Consequently, data can be passed among PEs in different architectures in ways that ensure their correct conversion in a program.

MPICH, a freely available implementation of the MPI standard developed by Argonne National Laboratory and Mississippi State University), was ported to the NOW. MPICH runs on networks of workstations, but suffers the higher communication costs associated with TCP/IP. The NOW port, however, was built on top of the Active Message layer, allowing applications to make full use of the underlying network hardware. This allows MPI-AM to compare with the performance delivered by most MPPs, in that it achieves a latency of 17 us (micro seconds) end-to-end, and a maximum bandwidth of 36.8 MB's.

5.0 Description of the Parallel Clarke Algorithm

The algorithm used is available in Clarke (1990: 207-219). It reads control points and assigns them in a "spatial structuring" step to their location in the grid that will be interpolated (thus small positioning errors are possibly introduced). Once the grid is initialized, it is not necessary to compute inter-control point distances and then sort them to determine which are the closest ones that should be used for interpolation. Instead, row and column indices can be incremented and decremented to effect a neighborhood search for control point information to those cells around each grid element. Thus, far fewer computations are required. A more complete analysis of this algorithm is provided in Cramer and Armstrong (1998).

We employed a self-scheduling or *manager-worker* task-scheduling algorithm to implement the parallel computational experiments reported in this paper. A single process is designated as a manager that maintains an inventory of the set of tasks that must be completed by the algorithm. The manager then distributes tasks when a request for work is sent from an idle

process. The initial pool of tasks was created by partitioning the entire grid into a set of vertical tiles of size 1024 x 8 (eight grid columns). This makes for an initial pool of 128 tasks. Variation in control point density normally makes it difficult to predict the amount of work required to complete the interpolation computations for each tile. Thus, we created a large pool of tiles, so that when a processor completes its task it may request a new task from the pool and be put back to work immediately.

Even though the Clarke algorithm is applied to a subset of the data (a tile), information outside of a local tile may be needed to complete some computations. To reduce the amount of communication required, the entire initialized grid was replicated over all processors using a broadcast type of communication construct. The Clarke algorithm can then be applied locally without interrogating neighboring processors; all required information is now stored locally. The results from each completed tile are returned to the manager task for final tabulation. Our implementation actually uses an array for the grid, and a second (Boolean) array of the same size that acts as a binary mask that is used to enumerate interpolated tiles. The total amount of memory used, therefore, 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.

6.0 Results

In all of the cases reported in this paper, 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 set at three (*k*=3). In a previous paper (Armstrong and Marciano, 1996) we reported results for this problem that were obtained with a cluster of workstations that were linked using a special-purpose high-performance network. That cluster, which consisted of DEC Alpha machines at the San Diego Supercomputer Center and was referred to as an "Alphafarm" can be considered a high-end benchmark against which the NOW environment can be compared (Table 1). The same MPI code was used for both the Alphafarm and the Berkeley NOW system. Note that Alphafarm results could only be obtained for a maximum of eight processors. At the time of these runs, 91 nodes were present on the Berkeley NOW system, but only 55 nodes were available for parallel processing configurations.

Several observations about Table 1 are relevant to an evaluation of the relative effectiveness of the results obtained using the NOW environment. For all of the parallel runs, optimization was disabled so as to eliminate uncontrolled variability in performance. When one processor was used to compute results, full optimization was enabled, however, and a single NOW processor computed the results in less than half the time required by an Alphafarm processor. The parallel results show smaller differences in

performance, with the NOW environment showing run times that are approximately 70% of the performance available with the Alphafarm up to its limit of 8 PEs. The results for the NOW environment, however, show a monotonic decline in run times. Moreover, when the full available configuration of 55 processors was used, the response times fall into a realm that is acceptable to many interactive users (approximately 5 seconds).

Table 1. Comparison of results obtained with NOW and Alphafarm.

Processors	Alphafarm (sec)	NOW (sec)	Alphafarm/NOW
1	112.7*	48.9**	2.30
2	125.4	180.3	0.70
3	62.8	92.4	0.68
4	42.9	61.1	0.70
5	31.7	45.8	0.69
6	25.9	37.4	0.69
7	21.7	31.1	0.70
8	18.9	26.9	0.70
9		23.5	
10		21.7	
15		14.5	
20		10.8	
25		9.1	
30		7.9	
35		7.0	
40		6.5	
45		6.2	
50		6.0	
55		5.8	

* Optimization level (f77 -O4 prog.f)

** Optimization level (f77 -O5) and glurun -1 -v a.out; this chooses one of the available nodes with the least load.

We also computed speedup and efficiency values for each of the runs. Speedup is a useful standard index of run time reduction for parallel implementations and is computed by forming a ratio of timings for two versions (parallel and non-parallel) of a program:

Speedup = run time for non-parallel / run time for parallel.

Efficiency is obtained by dividing speedup by the number of processors used and converting to a percentage value. The implicit standard to which parallel implementations are compared is a linear relation between speedup and number of processors; this obtains when efficiency is equal to 100%.

The efficiency measures show two important features of the NOW environment. First, the efficiencies observed are uniformly low since the comparison is made between a single processor using full compiler optimization and NOW processors that are executing unoptimized code. Efficiency hits a maximum value of only 23 across a range of configurations and a general decline is observed above 20 PEs. This decline continues until the maximum available configuration of 55 processors is used. In general, if the speedup values are examined, the problem scales using this architecture though the results are quite modest and a tapering off in performance is clearly observed (Figure 1).

Table 2. Speedup and efficiencies obtained with NOW.

Processors	Run time (sec)	Speedup	Efficiency %
1	48.9*	1.0	100
2	180.3	0.27	14
3	92.4	0.53	18
4	61.1	0.80	20
5	45.8	1.07	21
6	37.4	1.31	22
7	31.1	1.57	22
8	26.9	1.82	23
9	23.5	2.08	23
10	21.7	2.25	23
15	14.5	3.37	22
20	10.8	4.53	23
25	9.1	5.37	21
30	7.9	6.19	21
35	7.0	6.99	20
40	6.5	7.52	19
45	6.2	7.89	18
50	6.0	8.15	16
55	5.8	8.43	15

* Optimization level (f77 -O5) and glurun -1 -v a.out; available node with least load is chosen.

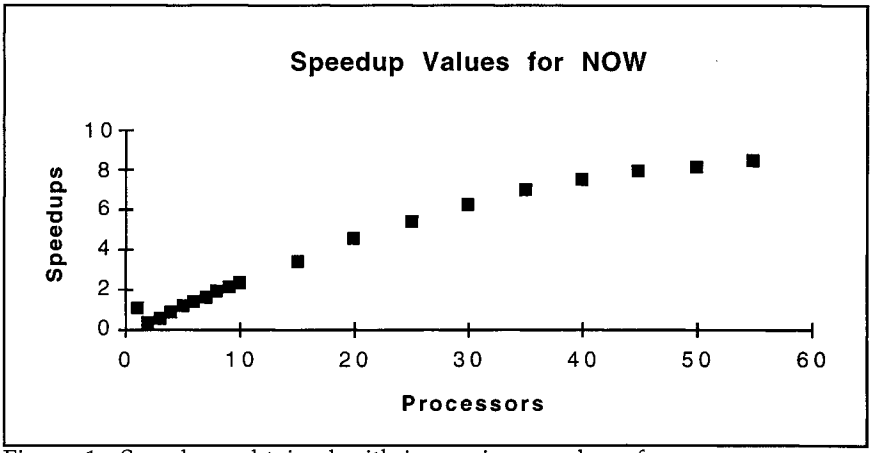

Figure 1. Speedups obtained with increasing number of processors.

There are several reasons for the level of performance that we have observed in this parallel environment. First, our MPI implementation (master-worker) uses a master process to assign tasks to workers, but it does not participate in the interpolation process itself. Further MPI code optimization would allocate tasks to the master. In addition, blocking sends and receives are used. This could also be relaxed so that more parallelism can occur with fewer barriers in place. Finally, we have not yet re-instrumented our MPI code to include performance monitoring features to track zones of processor idleness. Additional information will enable us to identify places in the program that can benefit from performance enhancements.

7.0 Conclusions

The Berkeley NOW system represents a new type of parallel processing environment that can be implemented to achieve relatively low cost parallelism. The system is built from commodity-class workstation components, though the crux of the architecture is high performance switches that route messages among processing nodes. In this high performance switch environment a readily decomposable inverse-distance weighted interpolation problem exhibited speedups across the range of processor configurations (minimum of 2 and maximum of 55) though processing resources were not exploited with high levels of efficiency (close to 20%). Such results indicate some promise for other types of spatial analysis methods that can be decomposed into discrete parts that can be allocated to available processors using a manager-worker model. However, further work is needed to extract high levels of performance on typical GIS-related problems from this architecture.

Acknowledgments

We thank Douglas Ghormley from UCBerkeley, for his help with GLUnix and Frederick Wong, also from UCBerkeley, for his assistance with the MPICH port on top of Active Messages.

URL List

1. http://now.cs.berkeley.edu/

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.J. 1996a. Distributed parallelism: impacts on GIS and collaborative spatial decision-making. *Proceedings of GIS/LIS '96*. Bethesda, MD: American Congress on Surveying and Mapping, pp. 527-539.

Armstrong, M.P. and Marciano, R. 1996b. 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.

Cramer, B. and Armstrong, M.P. 1998. An evaluation of domain decomposition strategies for parallel spatial interpolation of surfaces. Manuscript submitted for publication and available on request.

Culler, D.E., Arpaci-Dusseau, A., Arpaci-Dusseau, R., Chun, B., Lumetta, S., Mainwaring, A., Martin, R., Yoshikawa, C and Wong. F. 1997. Parallel computing on the Berkeley NOW. To appear in JSPP'97 (9th Joint Symposium on Parallel Processing) , Kobe, Japan. http://now.cs.berkeley.edu/Papers2/Postscript/jpps.ps

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.

Healey, R., Dowers, S., Gittings, B., and Mineter, M. 1998. *Parallel Processing Algorithms for GIS*. Bristol, PA: Taylor and Francis.

Kennedy, K., Bender, C.F., Connolly, J.W.D., Hennessy, J.L., Vernon, M.K. and Smarr, L. 1997. A nationwide parallel computing environment. *Communications of the Association for Computing Machinery*, **40** (11): 63-72.

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

Sterling, T. 1996. The scientific workstation of the future may be a pile of PCs. *Communications of the Association for Computing Machinery* **39** (9): 11-12.