COMPETITIVE RESOURCE MANAGEMENT IN DISTRIBUTED COMPUTING ENVIRONMENTS WITH HECTILING*

Ioana Banicescu, Samuel H. Russ, Mark L. Bilderback, and Sheikh K. Ghafoor
Mississippi State University
Department of Computer Science, Department of Electrical and Computer Engineering,
NSF Engineering Research Center for Computational Field Simulation
Mailstop 9637, Mississippi State, MS 39762-9637
e-mail: {ioana, russ, bilderba, ghafoor}@erc.msstate.edu

KEYWORDS: Distributed Computing, Resource Management, Network of Workstations, Load Balancing, Hectiling

ABSTRACT

Resource management for scientific applications in distributed computing environments is a complex problem. Over time, various techniques to manage resources at either coarse or fine levels of granularity in a network of workstations (NOW) have been proposed. A technique which has proven to be highly effective at the finer level of granularity is Fractiling. It is a dynamic scheduling scheme, which after being integrated into a scientific application, could be easily combined with any of the various coarse-grained techniques to optimize resource management. Recently, Fractiling has been combined with Hector, a coarse-grained strategy that has proven to be efficient in NOW. This resulting combination, Hectiling, provides the benefits of both. In this paper, experimental results of using Hectiling to execute a computationally intensive scientific application are presented. The combination of Fractiling with other coarse-grained strategies, such as Condor, MIST, etc., is also possible. While this technique has been implemented in NOW environments, it could be expanded to the world-wide interconnected network. This will open the possibilities of efficiently using the world-wide interconnected network resources for scientific applications.

1. INTRODUCTION

Traditionally, computationally intensive scientific applications are executed on massively parallel computers. With the increase of the computational power of desktop workstations and advances in network technologies, network of workstations (NOW) has become a widespread platform for parallel computing. The advantages of such an environment are low cost, availability, and flexibility, compared to the high cost and scarcity of supercomputers. However, a disadvantage is a limited control over load imbalance. This may result in performance degradation and poor resource utilization. With the recent expansion of the world-wide interconnected network (Internet, Web), the execution of scientific applications on remotely connected networks has become more feasible. To efficiently execute such applications, these resources must be managed effectively. This is a highly complex problem.

An essential component of resource management is load balancing. In a NOW environment, load balancing can be performed at both fine and coarse levels of granularity. Presently, there are several static and dynamic techniques for each level. Dynamic fine-grained load balancing is usually achieved by data migration. A recently developed technique, Fractiling, based on a probabilistic analysis, has been proven to be effective in scientific applications. It is implemented within individual applications and balances processor loads by migrating the application's data from busy to idle tasks. Applications such as N-body simulations have successfully employed Fractiling to improve their performance (Banicescu 1996)(Banicescu and Lu 1998).

Dynamic coarse-grained load balancing is achieved by moving tasks from heavily to lightly loaded processors. In general, this approach is application independent. It is implemented at the operating system level, relieving the application programmer from this responsibility. In addition, the coarse-grained load balancing techniques consider all tasks from all applications on the system. Dynamic coarse-grained systems such as LSF (LSF 1997), DQS (Institute 1995), Condor (Tannenbaum and Litzkow 1995), MIST (Casas,

*This work was supported by the National Science Foundation Grant EEC-8907070.

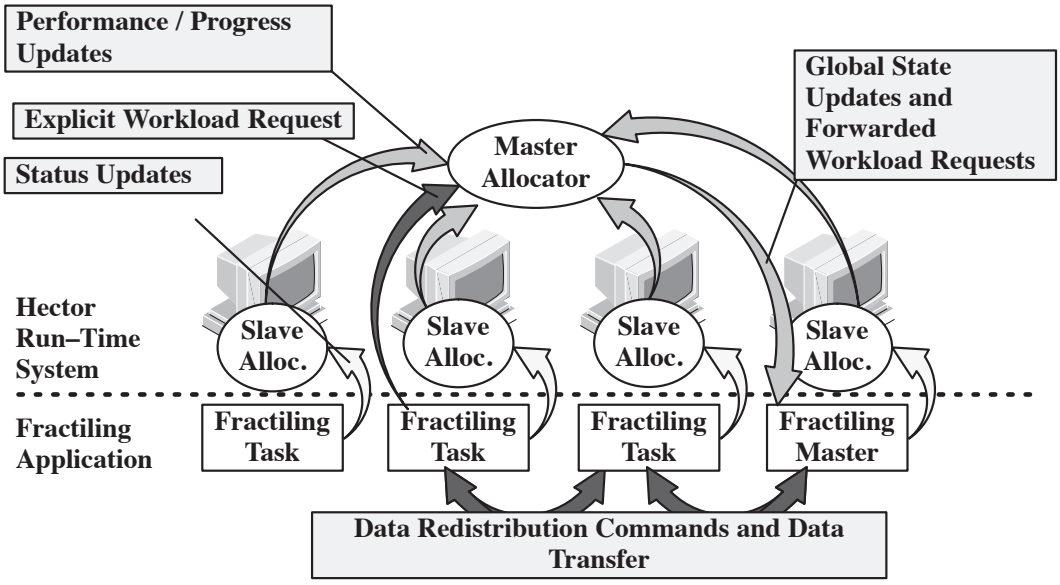

Fig. 1: Hectiling Interface (Fractiling/Hector)

Clark, Konuru, Otto, Prouty, and Walpole 1995), and Hector (Russ, Flachs, Robinson, and Heckel 1996) have been implemented in NOW environments. Hector, a distributed run-time environment developed at Mississippi State University, provides dynamic load balancing for MPI applications on Sun and SGI workstations.

Recently, Fractiling and Hector have been combined to create Hectiling (Russ, Banicescu, Ghafoor, Janapareddi, Robinson, and Lu 1998a), a system which provides resource management benefits of both. Hectiling, by managing resources at both levels of granularity, provides a more efficient utilization of resources than either technique could provide individually. In Hectiling, the communication structure of Hector is used by Fractiling to channel messages which control the exchange of data. This implementation results in efficient communication, therefore improvements in performance is achieved. This paper presents experimental results of running a computationally intensive scientific application, the Parallel Fast Multipole Algorithm (PFMA) for N-body simulations, within the Hectiling environment. These results show that an efficient resource utilization is achieved when an application is executed under Hectiling. Experiments of running N-body simulations with Fractiling under Hector have recently been reported (Russ, Banicescu, Ghafoor, Janapareddi, Robinson, and Lu 1998a)(Banicescu, Ghafoor, and Bilderback 1998)(Russ, Banicescu, Ghafoor, Janapareddi, Robinson, and Lu 1998b).

With the recent expansion of the world-wide interconnected network, the execution of scientific applications on remotely connected networks has become more feasible. Extending a Hectiling-like systems from NOW to the world-wide interconnected networks has for the first time been presented in (Banicescu, Ghafoor, and Bilderback 1998). The issues pertaining to this extension have been discussed in (Banicescu, Ghafoor, and Bilderback 1998).

The rest of this paper is organized as follows. Sections §2, §3, and §4, describe Hector, Fractiling, and Hectiling, respectively. Section §5 presents conclusions and proposed future work.

2. COARSE-GRAINED RESOURCE MANAGEMENT

In a distributed computing environment, such as a network of workstations, performance of scientific applications could be significantly improved by balancing the work load. A certain degree of load balancing can be achieved by moving tasks from highly loaded

to lightly loaded or idle nodes. Various parallel runtime systems and environments have been proposed to achieve such coarse-grained load balancing.

Coarse-grained load balancing via task migration from heavily to lightly loaded processors can be supported using two distinct methods. First, users can write their own state transfer routines which can be invoked by the runtime system to migrate or checkpoint a job. Systems such as LSF (LSF 1997) and DQS (Institute 1995) work in this fashion. The alternative is to provide systemic support for checkpointing and migration. Condor (Tannenbaum and Litzkow 1995), MIST (Casas, Clark, Konuru, Otto, Prouty, and Walpole 1995), and Hector work in this fashion. LSF is a widely used commercial package for controlling clusters. It launches utility tasks on each candidate host, which in turn monitors usage and provides remote launching capability. It runs parallel jobs and supports task migration through user level checkpointing. The Condor environment, developed at the University of Wisconsin, is a widely used public domain package. It groups workstations into "flocks", monitors their availability, and only runs parallel jobs written to tolerate variable numbers of hosts during execution. Research efforts in managing NOW (as the ones presented above) and a list of commercial products with their characteristics, are surveyed in detail in (Baker, Fox, and Yau 1995).

Hector (Russ, Flachs, Robinson, and Heckel 1996), a distributed runtime environment developed at Mississippi State University, provides dynamic load balancing by task migration. It supports MPI applications and is currently implemented for the Sun and SGI workstations. Hector attempts to balance the load by migrating task(s) during execution, from heavily loaded to idle or lightly loaded workstations. It has the facility to create backup copies of running programs, thereby providing fault tolerance. Hector is designed to use a master-slave hierarchy. A single task, called "Master Allocator" (MA), runs on one workstation performing all decision making functions. Instead of directly controlling MPI programs, the MA communicates with tasks called "Slave Allocators" (SA). There is only one SA for each physical machine. Each slave allocator launches and controls all MPI tasks running on its machine. In addition, it monitors performance characteristics of the MPI tasks. The SA periodically reports performance information to the MA. The MA then makes decisions regarding allocation and migration. When a decision has been made to migrate a task, the MA sends migration commands to the SA which had previously started the task. Comparisons between Hector features and those of similar systems can be found in (Russ, Gleeson, Meyers, Rajagopalan, and Tan tion).

The Hector runtime environment ensures dynamic load balancing at a coarser granularity level. In the next section a dynamic scheduling technique that provides load balancing at a finer granularity level is presented.

3. RESOURCE MANAGEMENT FOR DATA-PARALLEL APPLICATIONS

Problems in scientific computing are in general data-parallel, irregular, and computationally intensive. Many algorithms used in solving these problems are amenable to parallel execution. Load imbalance is one of the main performance degradation factors in parallel scientific applications. Imbalances could be caused by algorithmic factors, such as irregular data distribution and different processing requirements of data, as well as systemic factors, such as data access latency and external load due to other applications.

Previously, various methods have been used to balance processor loads and to exploit locality. Static partitioning and repetitive static partitioning heuristics have been the only methodology used to overcome dynamic load imbalance. Most of these methods use profiling by gathering information on the work load from a previous time step in the execution of the algorithm, in order to estimate the optimal work load distribution at the present time step. The cost of these methods increases with the number of processors and problem size. These methods are not robust and do not address load imbalances due to systemic factors.

Adapting to system induced load imbalances requires dynamic work assignment. Dynamic scheduling schemes attempt to maintain balanced loads by assigning work to idle processors at runtime. Thus, they adjust to systemic as well as algorithmic variances. In general, there is a tension between exploiting data locality and dynamic load balancing as the re-assignment of work may necessitate access to remote data. The cost of dynamic schemes is loss of locality, which translates in increased overhead.

Load balancing techniques have been extensively applied to scientific applications by using information about data distribution to guide the static assignment of data to processors (Singh, Holt, Totsuka, et al.

Particle Distribution	Number of Proc.	PFMA	Fractiling	Hectiling	Fractiling % Improvement	Hectiling % Improvement
Uniform	4	0.00719	0.00025	0.000529	96.00	92.64
	8	0.00886	0.00058	0.001992	93.28	77.52
	16	0.11009	0.00972	0.027990	91.17	74.58
	32	0.13298	0.03197	0.053198	75.96	60.00
Gaussian	4	0.01218	0.00095	0.007894	92.23	35.19
	8	0.01860	0.00365	0.010819	80.89	41.83
	16	0.65480	0.24522	0.173953	62.55	73.43
	32	0.93900	0.42940	0.248252	54.27	73.56
Corner	4	0.56758	0.23270	0.219075	59.00	61.09
	8	0.66997	0.14415	0.125606	78.48	81.25
	16	0.66776	0.12640	0.087870	81.07	86.84
	32	0.68977	0.10402	0.033675	84.82	95.12

Table 1: C.O.V. of processors finishing times

1993)(Warren and Salmon 1993)(Salmon and Warren 1997)(Board, Hakura, Elliot, et al. 1995). Some of these techniques include the orthogonal recursive bisection (ORB), the Costzones method, and a hashed oct-tree (HOT) method which employs the Morton order (Singh, Holt, Totsuka, et al. 1993)(Warren and Salmon 1993). Some experimentation with new scheduling schemes applied to scientific problems have been presented in (Hummel, Schonberg, and Flynn 1992)(Banicescu 1996)(Banicescu and Lu 1998)(Lu 1997). These schemes combine static techniques that exploit data locality with dynamic techniques that improve load balancing. In these schemes, work units and their associated data are initially placed on the same processor. Each processor executes its units in decreasing size chunks to preserve load balancing and to minimize overhead. After exhausting its local work, each processor acquires decreasing size chunks of work from other processors. These decreasing size chunks are represented by multidimensional subtiles of the same shape selected to maximize data reuse. The sizes of these subtiles are chosen so that they have a high probability of finishing before the optimal time. The subtiles are combined using the Morton order in larger subtiles, thus preserving the self-similarity property of fractals (Banicescu and Hummel 1995)(Banicescu and Lu 1998).

Fractiling, a dynamic scheduling technique based on a probabilistic analysis that incorporates the above features, balances processor loads and maintains locality by exploiting self-similarity properties of fractals (Hummel, Schonberg, and Flynn 1992)(Banicescu 1996). It accommodates load imbalances caused by predictable events (such as irregular data) and unpredictable events (such as data access latency and operating system interference). The technique has been applied to N-body simulations using the parallel implementation of the Greengard's 3-d Fast Multipole Algorithm (Greengard and Rokhlin 1987) on both a distributed memory shared-address space and a message passing environment. The distributed memory shared-address space implementation was run on a KSR-1 at the Cornell Theory Center (Banicescu 1996)(Banicescu and Hummel 1995) and the message passing environment implementation was run on an IBM SP2 at the Maui High Performance Computing Center (Banicescu and Lu 1998)(Lu 1997). Implementations of a parallel and a fractiled N-body simulation on uniform and nonuniform distributions of particles of various sizes and using up to 64 processors were compared. Experimental results confirmed that fractiled N-body simulations consistently resulted in improved performance (Banicescu 1996)(Lu 1997).

The next section presents an integration of Fractiling into a task parallel load balancing strategy to improve the overall resource utilization in a distributed computing environment.

4. INTEGRATED STRATEGY

Hector achieves better resource utilization by migrating tasks from highly loaded workstations to idle or lightly loaded workstations. Since the sizes of tasks are unequal, this coarse-grained load balancing strat-

Particle Distribution	Number of Proc.	PFMA	Fractiling	Hectiling	Fractiling % Improvement	Hectiling % Improvement
Uniform	4	1160	998	779	13.97	32.84
	8	1058	1050	1004	0.76	5.10
	16	1304	1182	1429	9.36	-9.59
	32	1591	1471	3293	7.54	-106.97
Gaussian	4	2177	1721	1606	20.95	26.23
	8	2342	2324	2030	0.77	13.32
	16	4047	3620	3419	10.55	15.52
	32	7252	6512	6692	10.20	7.72
Corner	4	2288	1721	1294	24.78	43.44
	8	3286	1781	1563	45.80	52.43
	16	4011	2106	1988	47.49	50.44
	32	4663	3123	3919	33.03	15.96

Table 2: Cost of PFMA, Fractiling and Hectiling (in seconds)

egy continues to suffer from load imbalance. On the other hand, fine-grained data parallel load balancing strategies, such as Fractiling, will ensure a high degree of load balancing by migrating data from one task to another. In a distributed computing environment a fractiled application may suffer from poor resource utilization, since Fractiling does not support task migration.

Consider a scenario in which a fractiled scientific application is running in a distributed environment. While fractiled tasks are running, one or more workstations become overloaded due to some additional external load. The Fractiling algorithm will now balance the load by migrating data from tasks running on overloaded workstations to lightly loaded workstations. Suppose some workstations become idle; if Fractiling had the capability of task migration in a Hector-like fashion, the fractiled tasks from the overloaded workstation could have been migrated to idle workstations. In this way, better resource utilization would have been achieved because idle resources would be utilized.

To take advantage of the benefits offered by Hector and Fractiling, a new system integrating both has been designed and implemented. Fractiling-like data parallel applications require communication to control the exchange of data between tasks. This process requires knowledge of both busy and idle tasks. Detection of idle tasks is simple, since idle tasks can always send requests for additional work. However, the detection of busy tasks is more difficult, since it requires ongoing knowledge of the status of all tasks. Hector has the capability of acquiring this knowledge through its information gathering infrastructure (Russ, Meyers, Gleeson, Robinson, Rajagopalan, Tan, and Heckel 1997).

An architecture for running a fractiled application under Hector is described in (Russ, Banicescu, Ghafoor, Janapareddi, Robinson, and Lu 1998a). Recently, an actual interface between fractiling tasks, fractiling master, and MA of Hector has been implemented (see Fig. 1). This implementation has been named Hectiling. The goal of this implementation is to route the "fractile-ask messages" (in Fractiling, fractiling tasks ask the fractile master for additional workload once they finish their current work assignment) from fractiling tasks to the fractiling master via the MA. This interface requires several modifications to the MA, SA, fractile master, and fractile task. Implementation details of Hectiling can be found in (Russ, Banicescu, Ghafoor, Janapareddi, Robinson, and Lu 1998b).

In Hectiling the SA sends the host name and port of the MA to all fractile tasks as part of the job-launch process. The fractile master registers with the MA, so that the MA knows which of the fractile tasks is the fractile master. Once they complete their current work assignment, the individual fractile tasks send a fractile-ask message for additional workload to the MA, which in turn forwards it to the fractile master. The fractile master then sends commands to fractile tasks to transfer data from relatively busy tasks to idle ones. For testing purposes, three implementations of the N-body simulations based on the Parallel Fast Multipole Algorithm by Greengard (Greengard and Rokhlin 1987)

have been used: one without Fractiling (PFMA), one with Fractiling and one with Hectiling. These implementations written in C and using MPI were run on a dataset of 100k particles. These experiments were conducted on a cluster of 8 quad processor, 90 MHz SPARCstation 10's. Three different data distribution were used: a uniform distribution of particles ("Uniform"), a Gaussian distribution of particles centered on a grid space ("Gaussian"), and a Gaussian distribution of particles shifted towards one corner of the computation space ("Corner"). All cases were run on 4, 8, 16 and 32 processors. During these executions the system was exclusively used for these experiments to exclude the effects of any external load. As a measure of resource utilization, the cost (number of processor multiplied by the run time) and the coefficients of variation (c.o.v.) of processors finishing times have been computed.

These results show that Hectiling significantly improves resource utilization. From these results it can be seen that there is a significant c.o.v. improvement of Fractiling and Hectiling compared to the PFMA. This improvement is due to Fractiling which balances processor loads by migrating data from busy tasks to idle ones. There is a similar improvement in cost. In general, the cost of Hectiling is lower than the cost of other implementations, especially when the number of processors is low. This is due to the fact that the routing of fractile-ask messages through the MA is faster than the MPI based direct routing from fractiling tasks to MA.

5. CONCLUSIONS

This paper has discussed two techniques which dynamically balance workload at fine and coarse levels of granularity in a NOW environment: Fractiling and Hector. These techniques have been combined into a single system, Hectiling, which provides better performance than either individually. This paper has also presented experimental results of N-body simulations using this system. These results show that Hectiling is an efficient resource management system for a NOW environment.

While the results for this paper have shown the benefits of Hectiling, the experiments conducted did not take into consideration the effect of external loads. Future experiments which will include external loads are expected to show an even larger degree of improvement over the results presented here.

A future modification to Hectiling will include prefetching. Since the Hector MA periodically gathers information about the state of the system, it is therefore in a position to determine in an efficient manner the point at which a fractiling task will require additional work. This allows the required data to be prefetched by a fractiling task, thereby reducing its idle time.

The idea of Hectiling can be extended to the world-wide interconnected network. This would allow the resources of the web to be effectively used for running scientific applications. Implementing Hectiling on the web will have to address several concerns (Banicescu, Ghafoor, and Bilderback 1998) such as: (i) where the MA will reside; (ii) how SAs will be started; (iii) how the MA will acquire a SA; (iv) how many MAs can a SA serve; (v) the communication protocol for MAs and SAs; (vi) and authorization and security issues.

Hectiling, being the combination of highly successful fine and coarse-grained techniques for resource management, is one of the most competitive system which could be developed at this time. However, as technology advances and more competitive fine and coarse-grained techniques become available, combining them will result into more effective Hectiling-like systems. These systems are and will continue to be highly desired by the scientific computing community, whose goal is to obtain low cost high performance solutions to their problems.

REFERENCES

Baker, M., G. Fox, and H. Yau (1995). Cluster Computing
Review. *Northeast Parallel Architecture Center, Syracuse University www.npac.syr.edu/techreports/hypertext/sccs-0748 /cluster-review.html.*

Banicescu, I. (1996, January). *Load Balancing and Data Locality in the Parallelization of the Fast Multipole Algorithm.* Ph. D. thesis, Polytechnic University.

Banicescu, I., S. Ghafoor, and M. Bilderback (1998). Efficient Resource Management for Scientific Applications in Distributed Computing Environment. In *Proceedings of the Workshop on Distributed Computing on the Web (DCW'98)*, pp. 45–54.

Banicescu, I. and S. F. Hummel (1995). Balancing Processor Loads and Exploiting Data Locality in N-Body Simulations. In *Proceedings of Supercomputing'95 Conference.*

Banicescu, I. and R. Lu (1998). Experiences with Fractiling in N-Body Simulations. In *Proceedings of High Performance Computing'98 Symposium*, pp. 121–126.

Board, J. A., Z. S. Hakura, W. D. Elliot, et al. (1995, February). Scalable Variants of Multipole-based Algorithms for Molecular Dynamics Applications. In *the Proceeding of Seventh SIAM Conference on Parallel Processing for Scientific Computing*, Philadelphia, pp. 295–300. SIAM.

Casas, J., D. Clark, R. Konuru, S. W. Otto, R. Prouty, and J. Walpole (1995). MPVM: A Migration Transparent Version of PVM. *Usenix Computing Systems Journal 8*(2), 171–216.

Greengard, L. and V. Rokhlin (1987, May). A Fast Algorithm for Particle Simulation. *Journal of Computational Physics 73*, 325–48.

Hummel, S. F., E. Schonberg, and L. E. Flynn (1992, August). A Practical and Robust Method for Scheduling Parallel Loops. *Communications of the ACM 35*(8), 90–101.

Institute, S. R. (1995). *DQS User Manual - DQS Version 3.1.2.3*. June: Florida State University.

LSF (1997). Product Reviews: Platform Computing Corp. Load Sharing Facility (LSF). *SunExpert 8*(8), 62–64.

Lu, R. (1997). Parallelization of the Fast Multipole Algorithm with Fractiling in Distributed Memory Architectures. Master's thesis, Mississippi State University.

Russ, S., I. Banicescu, S. Ghafoor, B. Janapareddi, J. Robinson, and R. Lu (1998a). Hectiling: An Integration of Fine and Coarse-Grained Load-Balancing Strategies. In *Proceeding of the IEEE International Symposium on High Performance Distributed Computing '98*, pp. 106–113.

Russ, S., I. Banicescu, S. Ghafoor, B. Janapareddi, J. Robinson, and R. Lu (1998b). Integrating Fine and Coarse-Grained Load-Balancing Strategies. *Submitted to Parallel and Distributed Computing Practices (PDCP) Special Issue on High Performance Computing on Clusters*.

Russ, S., B. Flachs, J. Robinson, and B. Heckel (1996). Hector: Automated Task Allocation for MPI. pp. 344–348.

Russ, S., M. Gleeson, B. Meyers, L. Rajagopalan, and C. Tan (Accepted for publication). Using Hector to run MPI Programs over Netwoked Workstation. *Concurrency: Practice and Experience*.

Russ, S., B. Meyers, M. Gleeson, J. Robinson, L. Rajagopalan, C. Tan, and B. Heckel (1997). User Transparent Run-Time Performance Optimization. In *The 2nd International Workshop on Embedded HPC and Applications at the 11th IEEE International Parallel Processing Symposium*.

Salmon, J. and M. S. Warren (1997). Parallel, Out-of-core Methods for N-Body Simulation. In *Proceeding of 8th SIAM Conference on Parallel Processing for Scientific Computing*. SIAM.

Singh, J., C. Holt, T. Totsuka, et al. (1993). A Parallel Adaptive Fast Multipole Algorithm. In *Proc. of Supercomputing'93*, pp. 54–65.

Tannenbaum, T. and M. Litzkow (1995). *The Condor Distributed Processing System*. Dr. Dobbs' Journal of Software Tools for Professional Programmer.

Warren, M. and J. Salmon (1993). A Parallel Hashed Oct Tree N-Body Algorithm. In *Proceeding of Supercomputing'93*, pp. 12–21. IEEE Computer Society.

Ioana Banicescu is an Assistant Professor in the Department of Computer Science at Mississippi State University, with research involvement at the National Science Foundation Engineering Research Center for Computational Field Simulation. Her research interests include parallel algorithms, scientific computing, scheduling theory and load balancing algorithms.

Samuel H. Russ is an assistant professor in the Department of Electrical and Computer Engineering at Mississippi State University and is a thrust leader at the National Science Foundation Engineering Research Center for Computational Field Simulation. His research interests include distributed run-time environments, parallel program performance optimization, and computer architecture. He received his BEE and PhD in Electrical Engineering from Georgia Tech in 1986 and 1991, respectively.

Mark L. Bilderback is a Ph.D. candidate in the Department of Computer Science at Mississippi State University and a Research Assistant at the National Science Foundation Engineering Research Center for Computational Field Simulation. His research interests include scheduling theory, load balancing algorithms and performance analysis.

Sheikh K. Ghafoor is a Master's candidate in the Department of Computer Science at Mississippi State University and a Research Assistant at the National Science Foundation Engineering Research Center for Computational Field Simulation. His research interests include run-time environments for networks of workstations and load balancing algorithms.