

PARALLEL SIMULATION OF LARGE-SCALE PARALLEL APPLICATIONS

Rajive Bagrodia
Ewa Deelman
Thomas Phan

COMPUTER SCIENCE DEPARTMENT, UNIVERSITY OF
CALIFORNIA, LOS ANGELES

Summary

Accurate and efficient simulation of large parallel applications can be facilitated with the use of direct execution and parallel discrete-event simulation. This paper describes MPI-SIM, a direct execution-driven parallel simulator designed to predict the performance of existing MPI and MPI-IO application. MPI-SIM can be used to predict the performance of these programs as a function of architectural characteristics, including number of processors, message communication latencies, caching algorithms, and alternative implementations of collective I/O operations. Results are presented, which show the use of MPI-SIM in performing a scalability study of real-world applications. The benchmarks chosen for the study include Sweep3D, one of the ASCI benchmarks, and BTIO, an I/O-intensive benchmark from the NAS Parallel Benchmark suite. MPI-SIM is shown to accurately and efficiently predict the performance of Sweep3D running on an Origin 2000. It is also used to demonstrate the impact of the number of I/O nodes on BTIO's performance.

Address reprint requests to Ewa Deelman, UCLA Computer Science Department, 3532 C Boelter Hall, Los Angeles, CA 90095-1596, U.S.A.; e-mail: deelman@cs.ucla.edu.

The International Journal of High Performance Computing Applications,
Volume 15, No. 1, Spring 2001, pp. 3-12
© 2001 Sage Publications, Inc.

1 Introduction

Accurate and efficient performance prediction of large-scale parallel applications on multiple-target architectures is a challenging problem. Two primary classes of simulators have been developed to study parallel program behavior: architecture and program simulators. The former typically use a detailed model of one or more components of the parallel architecture (e.g., the memory subsystem or interconnection network) to identify their impact on application behavior. Program simulators often use abstract models of the architectural components to evaluate program scalability or algorithmic alternatives. Many of the early architecture simulators were designed for sequential execution (Brewer et al., 1991; Davis, Goldschmidt, and Hennessey, 1991; Covington et al., 1988). To reduce the large execution times for such models, the simulators used direct execution, where sequential code fragments were executed rather than simulated. However, even with the use of abstract models and direct execution, sequential program simulators tended to be slow, with slowdown factors ranging from 2 to 35 for each process in the simulated program (Brewer et al., 1991). Several recent efforts have explored the use of parallel model execution (Legedza and Weihl, 1996; Dickens, Heidelberger, and Nicol, 1996; Mukherjee et al., 1997; Prakash and Bagrodia, 1995; Reinhardt et al., 1993) to reduce the simulation execution times, with varying degrees of success.

At UCLA, we have developed a suite of tools for the parallel simulation of parallel applications. The tools are being used for detailed program simulations within the POEMS project (Deelman et al., 1998). POEMS (Performance-Oriented End-to-End Modeling System) is a collaborative, multi-institute project whose goal is to create and experimentally evaluate a problem-solving environment for end-to-end performance modeling of complex parallel and distributed systems. The proposed POEMS environment includes the following:

- a language for the specification and composition of models for different domains;
- a set of multiparadigm (e.g., analytic, simulation, or mixed analytic and simulation) models at multiple resolution of granularity (e.g., use of direct execution to simulate sequential processor and memories or detailed simulation models of processor architectures) models at each of the application, system software, and hardware levels;

- a generalized task graph representation to provide workload information at multiple levels of abstraction to drive the system models;
- a set of representative applications that require the large computation, communication, and I/O resources provided by the massively parallel processor (MPP) systems;
- a database of models, results, and analysis capabilities.

Several of the preceding components are described in related papers in the previous issue of this journal. In particular, the specification and compositional modeling framework is described in Browne, Berger, and Dube (2000), and a relevant large-scale application, the Sweep3D kernel, is described in Hoisie, Lubeck, and Wasserman (2000). The use of analytical models in predicting performance of the Sweep3D application on very large machine configurations is in Sundaram-Stukel and Vernon (1999); the task graph notation and its use in modeling large-scale applications such as Sweep3D are described in Adve and Sakellariou (2000). In this paper, we give an overview of the parallel simulation capability for detailed performance prediction of large-scale scientific applications and illustrate its use by presenting two case studies—one showing the simulation of a CPU-bound application, the other dealing with an I/O-intensive kernel.

2 Simulator

The goal of the simulator is to enable the simulation of large-scale parallel applications written using existing parallel languages such as UC (Bagrodia, Chandy, and Dhagat, 1995) or in conventional languages such as Fortran or C with the aid of communication libraries such as message-passing interface (MPI) (MPI_Forum, 1993) on a variety of high performance architectures. The application program to be simulated is referred to as the *target program*, and the architecture on which its performance is to be predicted is referred to as the *target architecture*. The machine on which the simulator is to be executed is referred to as the *host machine*, which may be sequential or parallel.

The simulation environment is composed of several distinct, yet tightly coupled components (see Figure 1):

- the simulation kernel,
- the MPI communication library simulator (MPI-SIM),
- the data-parallel language simulator (UC-SIM),
- the parallel I/O simulator (PIO-Sim), and
- the parallel file system simulator (PFS-Sim).

“The application program to be simulated is referred to as the target program, and the architecture on which its performance is to be predicted is referred to as the target architecture. The machine on which the simulator is to be executed is referred to as the host machine, which may be sequential or parallel.”

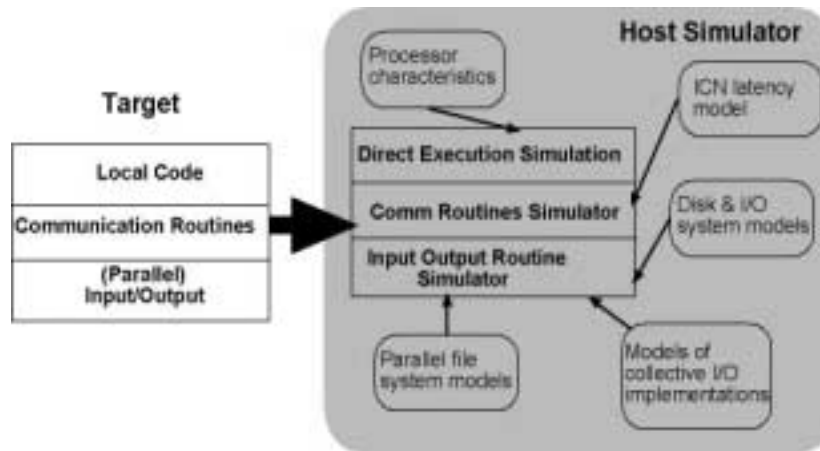


Fig. 1 UCLA's simulation environment

Each successive component builds on and extends the capabilities of previous components, expanding the breadth and the depth of the performance issues, which may be investigated with the simulator. The simulation kernel provides the framework; it implements the simulation protocols and provides support for the scheduling and execution of threads. MPI-SIM provides the capability to simulate individual and collective MPI communication routines (Bagrodia and Prakash, 1998). Similarly, UC-SIM simulates the execution of data-parallel programs that are written in a data-parallel language called UC (Bagrodia, Chandy, and Dhagat, 1995). PIO-Sim extends MPI-SIM's capabilities to include I/O routines as well as providing several implementations of collective I/O. It has the ability to handle user-defined data types that are needed to support complex I/O operations and a simple I/O service time model. PFS-Sim completes the simulation environment by providing detailed simulation of the parallel file system and of multiple-caching algorithms (Bagrodia, Docy, and Kahn, 1997). The simulator itself is portable and runs on a variety of parallel platforms—the IBM SP, the Origin 2000, and the Intel Paragon.

For the application to be simulated, the code has to be slightly modified. The basic steps in using the simulator to simulate an application are illustrated in Figure 2. In general, the number of processors in the host machine will be less than in the target architecture being simulated, so the

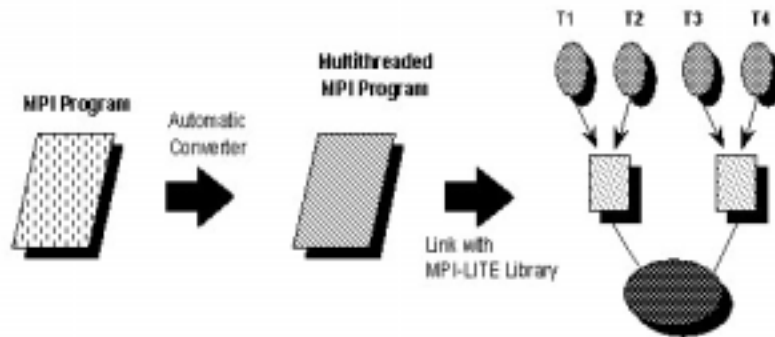


Fig. 2 Process of modifying the program to be simulated

simulator must support multithreading. For this purpose, we have developed *MPI-LITE*, a portable library to support multithreaded MPI programs. Executing an existing MPI program as a multithreaded program requires some modifications. The primary one deals with transforming the permanent variables (i.e., global variables and static variables within functions). If the unmodified MPI program is executed as a multithreaded program, all threads on a given host process will access a single copy of each permanent variable. To prevent this, it is necessary to *privatize* the permanent variable such that each thread has a local copy. Each permanent variable is redeclared with an additional dimension whose size is equal to the maximum number of threads on a host processor. Each reference to the permanent variable is also modified such that each thread uses its unique ID to access its own copy of the permanent variable. This process of adding a dimension to the permanent variables is referred to as *privatization*.

A preprocessor is provided with MPI-SIM that automatically privatizes permanent variables, converts each MPI call to the corresponding MPI-SIM or PIO-Sim call, and implements miscellaneous transformations needed to link the program with the simulation kernel. In MPI-SIM, the routines for interthread communication are syntactically identical to those for interprocess communication, except for the use of a special prefix to distinguish between the two. The kernel on each processor schedules the threads and ensures that events on all processors are executed in their correct timestamp order. A target thread is simulated as follows: the local code (code that does not require communications) is simulated by direct execution (Covington et al., 1988), and all communication and I/O

commands are trapped by the simulator. MPI-SIM uses an appropriate model to predict the execution time for the corresponding activity on the target architecture. For instance, consider a collective communication command such as the MPI_Bcast, which causes a message to be broadcast from a “root” process to all other processes in the program. To simulate the execution of this operation on a target architecture, the simulator must first model the implementation of the broadcast operation on the target machine. Such a model, together with a model of the interconnection network, is used to compute the communication latency for a message sent between any two processors on the target architecture. It allows the simulator to predict the delay for the execution of the broadcast operation by the target program on the target architecture. The corresponding communication operation is also executed on the host architecture to ensure that the results produced by the simulation model and target program are identical (to be able to supply the actual data in the operation to the threads). Similarly, if the target program contains an I/O operation, the time required to complete the operation is predicted by the model. Depending on the level of detail desired in the model, the I/O routine simulator may invoke an abstract model that simply estimates, at a gross level, the transfer time for a given I/O operation, or it might use a detailed model that implements various forms of caching to make a more precise prediction. Once again, the corresponding I/O commands are also executed for consistency with the target program. In some cases, the actual operation may be skipped, if the program does not subsequently use the results of that operation.

The use of direct execution for the simulation of local code requires that the processors of the host and target machines be similar. However, the interconnection network, parallel I/O system, and file systems on the two architectures may be very different. The simulator supports most of the commonly used MPI communication routines, such as point-to-point and collective communications. In the simulator, all collective communication functions are implemented in terms of point-to-point communication functions, and all point-to-point communication functions are implemented using a set of core nonblocking MPI functions (Bagrodia and Prakash, 1998). The interconnection network model currently ignores contention in the network. More detailed models are being developed, but given the excellent validation obtained with the simpler model for a variety of benchmarks both here and in previous work (Bagrodia et al., 1999), this was not considered to be a serious limitation.

The parallel I/O component of the simulator simulates the individual and collective I/O constructs provided by MPI-IO. These constructs include creating, opening, closing, and deleting a file; most data access (read/write) operations; and a local data-type constructor introduced as part of the MPI-IO specification. The file system component simulates the parallel file system used to service I/O requests generated by the MPI-IO programs. This component is self-contained and may be replaced by a simple disk access model to speed up simulation whenever a detailed system model is not required. However, using the detailed model allows the study of a wide variety of parallel file system configurations. The basic structure and functionality of the file system component are taken from the Vesta parallel file system, a highly scalable, experimental file system developed by IBM (Corbett and Feitelson, 1996). The behavior of the physical disks is simulated by a set of disk models. We have included simple models based on seek time, rotational latency, and data transfer rate as well as a highly detailed model developed at Dartmouth (Nieuwejaar and Kotz, 1996).

Detailed system simulations are slow. Parallel simulators can potentially reduce execution time of the model and provide greater amounts of memory, another necessity for large, detailed simulations. The simulator kernel provides support for sequential and parallel execution of the simulator; parallel execution is supported via a set of conservative parallel simulation protocols (Bagrodia et al., 1999). When combined with the kernel’s built-in multithreading capabilities, this allows the simulator to effectively use however many host processors as are available without limiting the size and type of experiments that may be run. The simulator also supports a number of optimizations that are based on an analysis of the behavior of the parallel application. Among the optimizations made available by program behavior analysis is a technique that allows the simulation protocols described above to actually be turned off, eliminating the costly overhead of global synchronization (Prakash, Deelman, and Bagrodia, 2000).

Parallel simulation has been shown to improve the performance of a number of applications, including parallel programs (Dickens, Heidelberg, and Nicol, 1996; Mukherjee et al., 1997; Bagrodia and Prakash, 1998). Although a number of parallel discrete-event simulation (PDES) protocols have been defined, only the conservative protocols have been shown to be effective in the simulation of parallel programs. The simulation kernel used in our simulator supports multiple conservative PDES protocols that include the null message, conditional event,

“Parallel simulation has been shown to improve the performance of a number of applications, including parallel programs.”

“Complex parallel simulation protocols are necessary because simulated processes may receive messages from different processes in an order different than they would have been received in a physical system.”

and the accelerated null message protocol. Special optimizations are made possible by an understanding of parallel application behavior. Each protocol provides benefits under different simulation circumstances. The null message protocol (Chandy and Misra, 1979) provides fast progress when event lookahead (*lookahead* refers to the ability of a process to predict its future behavior) is good, and the conditional event protocol (Jha and Bagrodia, 1993) is more appropriate when the lookahead is poor. The accelerated null message protocol combines features of both protocols such that synchronization overheads can be reduced for a range of simulation conditions.

The simulator also supports a number of optimizations that are based on an analysis of the behavior of the parallel application. Among the optimizations made available by program behavior analysis is a technique that allows the simulation protocols described above to actually be turned off, eliminating the costly overhead of global synchronization. It was discovered that the communication in a great number of applications is deterministic, where processes explicitly name the source of a message within their receive statement. Complex parallel simulation protocols are necessary because simulated processes may receive messages from different processes in an order different than they would have been received in a physical system. Messages from the same process, however, are always received in the proper order. So, when communication is deterministic, the simulation protocols are no longer needed, and performance of the simulator can be greatly enhanced, with performance being limited only by the parallelism inherent in the target program. Also, the implementation of many collective MPI-IO functions, in particular, is deterministic, and consequently their parallel simulation can be made significantly more efficient.

3 Case Studies

We tested our simulation package using a number of benchmarks to evaluate its prediction accuracy and performance scalability. Next, we detail our findings for two categories of experiments—namely, the simulation of an MPI program and of a parallel I/O program.

MPI PROGRAM SIMULATION

The core benchmark in our experimental set is Sweep3D, a scientific application that solves a three-dimensional, time-independent neutron particle transport problem. Sweep3D is a part of the Accelerated Strategic Computing Initiative Blue Benchmark suite of programs (http://www.llnl.gov/asci_benchmarks) and uses the MPI message-

passing library during the course of its execution. The computation portion of Sweep3D calculates the flux of particles through a given discretized region of three-dimensional space as a function of the flux through adjacent spatial cells. The program progresses in a wavefront manner from all the eight spatial octants, with each octant containing six independent angles. The program uses a 2-D decomposition onto a 2-D array of processors, and in this composition, the sweeplike progression continues as a processor computes the flux through the column of cells, sending the outgoing flux information to its two neighboring processors. Parallelism is used by way of having the third dimension and the angles divided into blocks, allowing a processor to calculate only a portion of its values in that dimension and only a few angles before sending its partial results to its neighbors.

As with any simulation, the accuracy of the results is very important. Here, we present the prediction accuracy of the simulator on one of our testbeds, the SGI Origin 2000. Figure 3 is a graph of the execution time of the real Sweep3D program compared to the execution time predicted by our simulator. Since the curves are a function of the number of physical processors used by the real Sweep3D program and the number of target processors simulated by MPI-SIM, the validation analysis is thus limited to the number of physical host processors at hand (we have 10 processors on our Origin). We constrained the number of Sweep3D executions to have a power-of-2 number of processors, resulting in runs of 1, 2, 4, and 8 processors. The problem size in this graph is a constant per processor size of $6 \times 6 \times 1000$ cells (this size is of interest to scientists at LANL) (Hoisie, Lubeck, and Wasserman, 2000). Because the simulator can provide a larger number of target processors than physical processors available, we averaged the predicted running time for a given number of target processors over the predicted times for all available combinations of host processors. For instance, for 8 target processors, the predicted running time was taken as the average running time with 1, 2, 4, and 8 host processors. From the graph, it is seen that the simulator is indeed accurate, correctly predicting the execution time of the benchmark to within 2% at all points. We also see that the simulator can predict the performance of the application running on a number of processors greater than are available on the host system. Here, we show the performance of Sweep3D as predicted by the simulator as the number of target processors is increased to 32 (see Figure 3).

Next, we present a number of results to demonstrate the performance scalability of the simulator attained from its parallel execution. Figure 4 shows the simulator's speedup

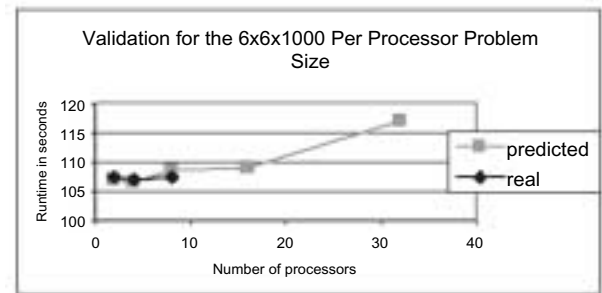


Fig. 3 Validation of the simulator running Sweep3D

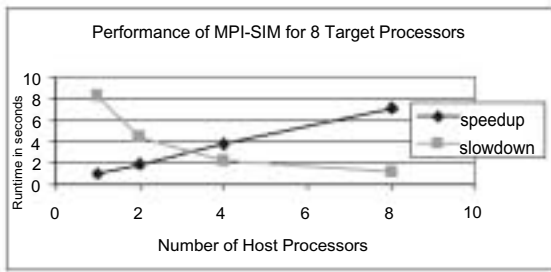


Fig. 4 Performance of the simulator running Sweep3D

while simulating Sweep3D for fixed per processor problem size (as above) on eight target processors. As seen from the figure, the simulator effectively uses additional physical processors, manifest by the near-linear speedup as the number of host processors is increased. A speedup of 7, relative to sequential execution of the simulator, is attained when eight host processors are used.

Another common metric to evaluate the performance of a simulator is the measurement of its slowdown relative to the target architecture. We define slowdown (H, T) as the time to simulate the application using H host processors divided by the time to execute the real application itself on T processors. Intuition tells us that slowdown will decrease as the ratio of target processors to host processors decreases. Figure 4 shows the slowdown for an execution with eight target processors, and we see that as more host processors are made available to the simulator, the slowdown decreases accordingly. When eight host processors are used ($H = T$), the slowdown is only 1.16; however, even if only four host processors are available ($H = T/2$), the slowdown is only 2.18. Note that other program simulators such as WWT (Mukherjee et al., 1997) and LAPSE (Dickens, Heidelberger, and Nicol, 1996) have reported slowdown factors as high as 100 for computationally intensive applications.

PARALLEL I/O PROGRAM

One of the few parallel benchmarks is BTIO. BTIO simulates the I/O required by a time-stepping flow solver that periodically writes its solution matrix. In the “full” benchmark, data are fully described using MPI noncontiguous, user-defined data types, and collective I/O is used. The benchmark exhibits numerous seek operations. One of the system characteristics that can be analyzed using our simulator is to determine the impact of the number of compute (cnode) and I/O (ionodes) nodes on the performance of the application.

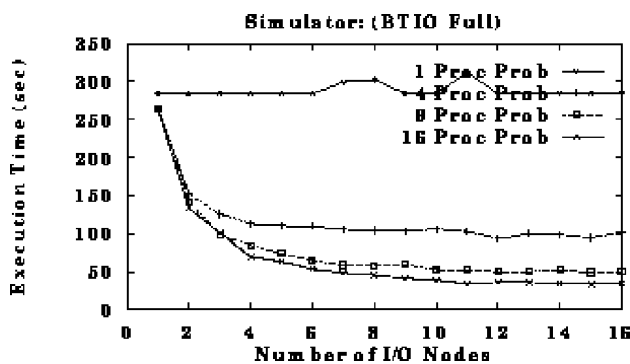


Fig. 5 Predicted execution time as a function of ionodes and cnodes

Figure 5 shows the performance of the NAS benchmarks as the number of ionodes in the system increases from 1 to 16. The one-processor problem does not benefit from the availability of more ionodes because the I/O requests performed by this benchmark are small enough to fit inside a single data block, allowing them to be serviced by a single ionode. When multiple processes are used to run the benchmark, more I/O requests are generated. This allows extra ionodes to relieve the congestion, which results when all target processes try to access a single ionode. It can also be seen that when a given number of processors are available in the system, performance is best when the

number of cnodes and ionodes is balanced. For example, with 17 processors, performance goes from worst to best when there are 1 cnode, 16 ionodes; 16 cnodes, 1 ionode; 4 cnodes, 13 ionodes; and 9 cnodes and 8 ionodes.

4 Conclusion

We have sketched the parallel application simulation tools developed at UCLA. The simulator can model the behavior of MPI and MPI-IO applications running on high performance architectures such as the IBM SP, the SGI Origin 2000, and the Intel Paragon. The simulator consists of modules capable of predicting the performance of applications as a function of communication latency, number of available processors on the machine of interest, different caching strategies for parallel I/O, parallel file system characteristics, and alternative implementations of collective communication and I/O commands. We have used case studies to illustrate the simulator's accuracy and good performance. In particular, the simulator was shown to be accurate to within 2% for the Sweep3D application kernel. Also, a very good speedup of 7 on eight processors was achieved for that application without suffering from undue slowdowns. We have also shown that the simulator can be used to analyze the behavior of I/O-intensive applications on a variety of machine configurations, which make use of a number of system resources. Tools, such as the UCLA simulator, can be used for performance studies of complex, large-scale applications on today's and tomorrow's high performance hardware.

ACKNOWLEDGMENTS

This work was supported by DARPA/ITO under contract N66001-97-C-8533, "End-to-End Performance Modeling of Large Heterogeneous Adaptive Parallel/Distributed Computer/Communication Systems."

BIOGRAPHIES

Rajive Bagrodia is a professor of computer science in the School of Engineering and Applied Science at UCLA. He obtained a bachelor of technology degree in electrical engineering from the Indian Institute of Technology, Bombay, in 1981. He obtained his M.A. and Ph.D. degrees in computer science from the University of Texas at Austin in 1983 and 1987, respectively. His research interests include nomadic computing, parallel simulation, and parallel languages and compilers. He has published more than 100 research papers on the preceding topics. The research has been funded by a variety of government and indus-

trial sponsors, including the National Science Foundation, Office of Naval Research, DARPA, Rome Laboratory, and Rockwell International. He is an associate editor of the ACM Transactions on Modeling and Computer Systems (TOMACS). He was selected as a 1991 Presidential Young Investigator by the National Science Foundation and won the TRW Outstanding Teacher award.

Ewa Deelman is a senior development engineer in the Computer Science Department at University of California, Los Angeles. She obtained her Ph.D. degree from Rensselaer Polytechnic Institute in 1997. Her thesis work concentrated on parallel discrete-event simulation of spatially explicit problems. To achieve good performance, new synchronization techniques and rollback mechanisms were formulated. Load-balancing techniques were also investigated. At UCLA, she is performing research in the area of performance prediction of message-passing applications on high performance machines. Her research interests are in parallel simulation and parallel programming, languages, compilers, and tools as well as application performance prediction.

Thomas Phan is a Ph.D. student in computer science at UCLA. His research interests include parallel and distributed computing, programming languages, and operating systems.

REFERENCES

- Adve, V., and R. Sakellariou. 2000. Application representation for multi-paradigm performance of large-scale parallel scientific codes. *International Journal of High Performance and Computing Applications* 14 (4): 304-316.
- Bagrodia, R., M. Chandy, and M. Dhagat. 1995. UC: A set-based data-parallel language. *Journal on Parallel and Distributed Computing* 28 (2): 186-201.
- Bagrodia R., E. Deelman, S. Docy, and T. Phan. 1999. Performance prediction of large parallel applications using parallel simulations. *SIGPLAN Notices* 34 (8): 151-162.
- Bagrodia, R., S. Docy, and A. Kahn. 1997. Parallel simulation of parallel file systems and I/O programs. In *SuperComputing '97*.
- Bagrodia, R., and S. Prakash. 1998. MPI-SIM: Using parallel simulation to evaluate MPI programs. In *Winter Simulation Conference*, Washington, DC.
- Brewer, E. A., et al. 1991. Proteus: A high-performance parallel architecture simulator. MIT Technical Report MIT/LCS/TR-516.
- Browne, J. C., E. Berger, and A. Dube. 2000. Compositional development of performance models in POEMS. *International Journal of High Performance and Computing Applications* 14 (4): 283-291.
- Chandy, K. M., and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering* (September): 440-452.

- Corbett, P. F., and D. G. Feitelson. 1996. The Vesta parallel file system. *ACM Transactions on Computer Systems* 14 (3): 225-264.
- Covington, R. G., et al. 1988. The Rice parallel processing testbed. In *Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*.
- Davis, H., S. R. Goldschmidt, and J. Hennessey. 1991. *Multiprocessor Simulation and Tracing Using Tango*. In *ICPP '91*.
- Deelman, E., et al. 1998. POEMS: End-to-end performance design of large parallel adaptive computational systems. In *Proceedings of the First International Workshop on Software and Performance '98—WOSP '98*, Santa Fe, NM.
- Dickens, P. M., P. Heidelberger, and D. M. Nicol. 1996. Parallel direct execution simulation of message-passing parallel programs. In *IEEE Transactions on Parallel and Distributed System*.
- Hoisie, A., O. Lubeck, and H. Wasserman. 2000. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. *International Journal of High Performance and Computing Applications* 14 (4): 330-346.
- Jha, V., and R. L. Bagrodia. 1993. Transparent implementation of conservative algorithms in parallel simulation languages. In *Proceedings of 1993 Winter Simulation Conference (WSC '93)*, Los Angeles.
- Legedza, U., and W. E. Wehl. 1996. Reducing synchronization overhead in parallel simulation. In *10th Workshop on Parallel and Distributed Simulation*.
- MPI_Forum. 1993. MPI: A message passing interface. In *Proceedings of the 1993 Supercomputing Conference*, Portland, OR.
- Mukherjee, S. S., et al. 1997. Wisconsin Wind Tunnel II: A fast and portable parallel architecture simulator. In *Workshop on Performance Analysis and Its Impact on Design (PAID)*.
- Nieuwejaar, N., and D. Kotz. 1996. The galley parallel file system. In *Proceedings of the 10th ACM International Conference on Supercomputing*.
- Prakash, S., and R. Bagrodia. 1995. Parallel simulation of data parallel programs. In *Proceedings of the 8th Workshop on Languages and Compilers for Parallel Computing*.
- Prakash, S., E. Deelman, and R. Bagrodia. 2000. Asynchronous parallel simulation of parallel programs. *IEEE Transactions on Software Engineering* 26 (5): 385-400.
- Reinhardt, S. K., et al. 1993. The Wisconsin Wind Tunnel: Virtual prototyping of parallel computers. In *Proceedings of the 1993 ACM SIGMETRICS Conference*.
- Sundaram-Stukel, D., and M. K. Vernon. 1999. Predictive analysis of a wavefront application using LogGP. *SIGPLAN Notices* 34 (8): 141-50.