PSINS: An Open Source Event Tracer and Execution Simulator

Mustafa M. Tikir, Michael A. Laurenzano, Laura Carrington, Allan Snavely

Performance Modeling and Characterization Lab
San Diego Supercomputer Center
9500 Gilman Drive, La Jolla, CA
{mtikir, michaell, lcarring, allans}@sdsc.edu

As the size of today's supercomputers grow exponentially in numbers of processors, the applications that run on these systems scale to larger processor counts. The majority of these applications commonly use MPI; a trace of these MPI communication events is an important input to the tools that visualize, simulate for performance modeling, or enable tuning of parallel applications. We introduce an efficient, accurate and flexible trace-driven performance modeling and prediction tool, PMaC's Open Source Interconnect and Network Simulator (PSINS), for MPI applications. A principal feature of PSINS is its usability for applications that scale up to large processor counts. PSINS generates compact and tractable event traces for fast and efficient simulations while producing accurate performance predictions. PSINS was incorporated into PMaC's automated performance prediction framework and used to model three applications from the High Performance Computing Modernization Office's (HPCMO) Technology-Insertion 2009 (TI-09) application suite.

1 Introduction

Performance models are calculable expressions that describe the interaction of an application with the computer hardware. The performance models provide valuable information for tuning of both applications and systems [1]. An ongoing trend in High Performance Computing (HPC) is the increase in the total system core count; this in turn has enabled scaling to tens and even hundreds of thousands of cores in recent years enabled by performance models that are used to guide application tuning [2-4]. Application performance is a complex function of many factors such as algorithms, implementation, compilers, underlying processor architecture and communication (interconnect) technology and topology. However as applications scale to larger processor counts, the interconnect becomes a more prevalent factor in their performance. This requires improved tools to efficiently measure and accurately model the performance of applications.

We present an automated performance modeling framework that includes an efficient, accurate and flexible trace-driven performance modeling tool, PMaC's Open Source Interconnect and Network Simulator (PSINS), for MPI applications. PSINS includes two major components, one for collecting event traces during an application's run (*PSINS Tracer*), and the other for the replay and simulation of these event traces (*PSINS Simulator*) for the modeling of current and future HPC systems. The key design goals for PSINS are 1) scalability 2) speed 3) extensibility. To meet the first goal, PSINS Tracer runs with very low overhead to generate compact traces that do not use more bits than needed for a complete record of events. To meet the second goal, PSINS Simulator enables replay of events faster than real-time (a replay does not normally take as long as the original application run) while still producing accurate performance predictions. To meet the third goal, both PSINS Tracer and Simulator are provided freely as open-source and include its built-in trace formats, trace format conversion modules, communication models, and a graceful API for extending the communication models as well as input trace formats.

In this work, the addition of PSiNS Tracer and Simulator to PMaC's automated performance prediction framework is tested by predicting the performance of a set of applications from the Department

of Defense's (DoD) High Performance Computing Modernization Program's (HPCMP) Technology Insertion 2009 (TI-09) program. The remainder of the paper is laid out as follows. Section 2 describes the framework in details about how the PSiNS Tracer and Simulator fit in. Section 3 presents the experimental results. Section 4 describes the related work and Section 5 presents conclusions.

2 Automated Performance Prediction Framework

The PMaC framework for performance prediction has three primary components, 1) benchmarks for characterizing machines (collectively called Machine Profiles), 2) trace collection and simulation tools for gathering information about applications (collectively called Application Signatures), and 3) a tool for applying performance hypotheses on the second in light of the first (called the Convolver).

Figure 1 illustrates the layout of the automated performance prediction framework and its components. The framework is composed of two models, a single-processor model and a communications model. Each model is comprised of an application signature, a machine profile, and a convolution method. The application signature represents the work needed to be done by the application, or the fundamental operations required by the application independent of the machine. The application signature is collected on a base HPC system. The machine profile represents the capability of a target HPC system, or its measured performance in executing fundamental operations, something that is measured by low-level benchmarks such as *ICBench*, *NetBench* and *MultiMAPS*. The convolution method is a way of mapping the application's needed work to the machine's capabilities, specifically, mapping application trace data to measured benchmark data via simulation (i.e. a replay of the application running on the target system).

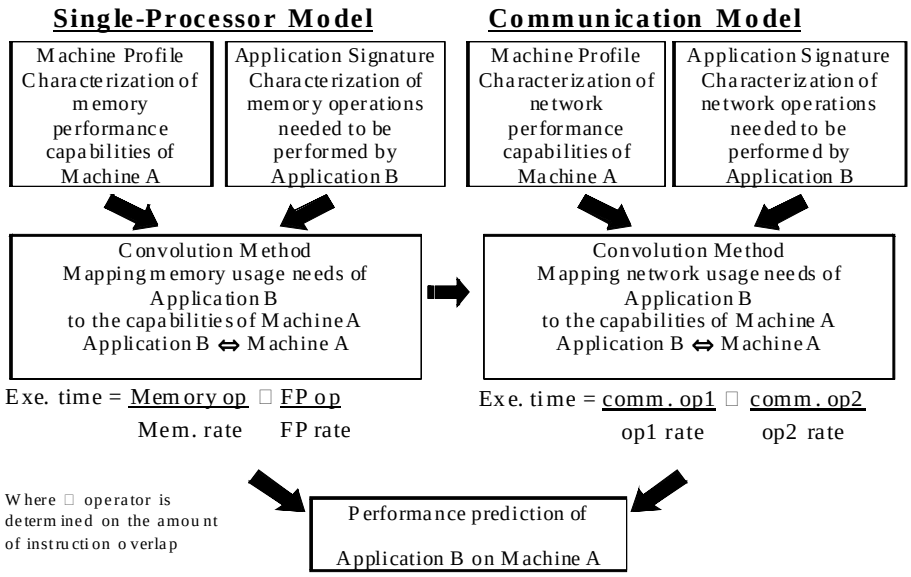


Figure 1. Automated performance prediction framework

In the following subsections, we describe the new components of the PSiNS Tracer (Section 2.1) to gather the communications model's application signature, and the PSiNS simulator (Section 2.2) as the overall convolution method for both the processor and communication model. For a more complete description of the other pieces of the framework, please see Snavely et al. [25], Carrington et al. [30] and Tikir et al. [31].

2.1 PSiNS Tracer for Collecting MPI Event Traces (Application Signature)

PSINS provides a tracer library based on MPI's profiling interface (PMPI) [10]. PMPI provides the means to replace MPI routines at link time allowing tool developers to include additional instrumentation code around the actual MPI calls. In addition, the PMPI interface enables gathering detailed information about the arguments that are passed to each MPI call by sharing the same function signature as the actual invocation.

The tracer library provides wrappers that serve as replacements for the MPI routines in the code (i.e. communication or synchronization events). For each MPI routine replacement, it uses additional code to gather detailed information about the called MPI function and its arguments. The tracer also gathers the time in between consecutive communication events, referred as the computation time and labeled as *CPUBurst*. This CPUBurst time is used later in the convolution to incorporate the processor model with the communication model. The PSiNS tracer includes a number of optimizations to reduce trace size and overhead of trace collection. These are discussed in detail in Tikir et al. [32].

Besides tracing functionality, PSINS tracer provides two additional libraries for performance measurement and analysis that can be included in the tracing run or collected independent from the trace. The first, called *PSINS Light*, is a library to measure overall execution time of the application and gather some event counts from the performance monitoring hardware (using PAPI [15]) in the underlying processors such as FLOP rate and overall cache miss counts. The second, called *PSINS Count*, is a library to measure the execution times and frequencies of each MPI function in the application in addition to those values collected by PSINS Light. PSINS Count is similar to IPM [14] and provides only a subset of information IPM provides. PSINS Tracer library is already ported for several HPC systems and is available at http://www.sdsc.edu/pmac/projects/psins.html.

2.2 Simulator for Performance Prediction

PSINS Simulator takes the communication event trace for an application and a set of modeling parameters for the target system and then replays the event trace for the target system, essentially simulating the execution of the parallel application on the target system. To simulate an MPI application on a target system, PSINS Simulator models both computation and communication times for each task in the application.

To simulate the execution of a target system, the simulator needs details about the configuration and construction of the system. These modeling parameters consist of the configurable components of a parallel HPC system and are described by the machine profile of the target machine.

PSINS Simulator consumes events from the input trace (PSiNS trace of application) in the order of their execution within the simulator rather than consuming them on a per-task basis. The simulator uses an event queue based on priority queues to replay the input trace.

The execution of an event during simulation depends on the type of the event and the state of the system at each event execution. The state of a system at any given time is a combination of the best achievable bandwidths and latencies, the bus load, contention, traffic in the network and the underlying network topology. If it is a CPU burst event, it is completed by calculation of its time on the target system using the data from the processor model. For blocking communication events, it is kept in the queue until its mate is posted. If the event is a global communication, it is kept in the queue until all participating tasks post the same event. When all participating tasks post the event for that communication, the communication model is asked to calculate the bandwidth and latency at the time of its execution and the event is executed. Each event type can have a different model based on network or system configuration. For more complete details of the simulator and its communication models refer to Tikir et al [32].

PSINS Simulator includes a statistics module to collect detailed information about the simulation of an event trace on the target system. The statistics module collects information about the event execution

frequencies, computation and communication times for each task as well as the execution time for each event type on the target system. It also collects the waiting time for each event type to provide information on load balancing during the execution. Moreover, it generates histograms on message sizes and on the ranges of bandwidths calculated by the communication model for the communication events.

Such information provides valuable feedback to users and developers to help them understand the interaction of applications with the target system, and can be valuable to guiding optimization and porting efforts for the application. More importantly, this information is useful for verifying simulation accuracy by comparing it to the same information measured during an actual run on the target system using performance monitoring tools such as IPM, TAU or PSINS Count.

3 Experimental Results

To demonstrate the usability, efficiency and accuracy of using the PSINS Tracer and Simulator in the performance prediction framework described in section 2 we have conducted several experiments. We used PSINS Tracer to collect MPI event traces for three scientific applications: AVUS [17], HYCOM [18] and ICEPIC [19] from the TI-09 Benchmark Suite [20]. The traces were then used in the framework to simulate a set of target HPC systems and the predictions resulting from these simulations were compared to the actual measurements gathered on the target systems.

All the traces were collected on a base system, Naval Oceanographic Office's IBM Cluster 1600 (3072 cores connected with IBM's High Performance Switch), called *Babbage*. We ran the scientific applications with two input data sets, namely *standard* and *large*, and processor counts ranging from 59 to 1280. The actual runtimes for the applications range from 0.5 to 2.5 hours where each application runs for around half an hour at the highest processor count and was scaled to that count using the same input data set (i.e. strong scaling). For replay and simulation of the collected traces, we ran the simulator on a Linux box with two dual-core processors. In addition to predicting the base system Babbage, we also predicted the Maui High Performance Computing Center's (MHPCC) Dell Cluster, called *Jaws* (5120 cores connected with Infiniband) and Engineer Research and Development Center's (ERDC) Cray XT3 system, called *Sapphire* (8320 cores connected with Cray SeaStar engine). In Table 2 we present the detailed results of these experiments in terms of prediction accuracy.

	Jaws	**Sapphire**	**Babbage**
Absolute Average Error Per Machine	8.2 %	13.0 %	12.1 %
	HYCOM	**AVUS**	**ICEPIC**
Absolute Average Error Per Application	14.0 %	9.0 %	10.1 %

Table 1. Errors per machine over all applications and per application over all machines.

Table 1 summarizes the average absolute error for each machine over all applications and CPU counts and the average absolute error for each application over all machines and CPU counts. Table 1 shows that overall average absolute prediction error for each HPC system is under 13% and overall error for each application is under 15%. We believe that the error for HYCOM over all machines is slightly higher compared to the error for other applications due to the imbalance in computation times among its tasks (as also shown in Figure 2).

To investigate the accuracy of the framework with the PSiNS tools, it was used to collect traces and predict the runtimes of AVUS, ICEPIC, and HYCOM on Jaws, Sapphire, and Babbage. Table 2 presents the comparison between the predicted runtime time, the measured runtime, and the relative error compared to an actual run of each application and the three HPC systems. Using the data from Table 2 we calculate the overall average absolute relative error for all the predictions to be 11.1% and the average

absolute relative error for each system to be below 13.1% with the majority of predictions resulting in an absolute relative error below 20%. These results show that the runtime predictions are quite accurate with the addition of PSiNS to the automated framework.

	# CPU	Jaws			Sapphire			Babbage		
		Runtime (s)		% Error	Runtime (s)		% Error	Runtime (s)		% Error
		Predicted	Real		Predicted	Real		Predicted	Real	
HYCOM LRG	**256**	5,973	5,800	**3.0**	5,756	5,956	**-3.4**	5,399	5,910	**-8.6**
	504	2,816	3,002	**-6.2**	2,764	3,752	**-26.3**	2,716	2,987	**-9.1**
	1006	1,483	1,751	**-15.3**	1,545	1,914	**-19.3**	1,524	1,918	**-20.5**
	1267	1,206	1,483	**-18.7**	1,284	1,524	**-15.7**	1,278	1,693	**-24.5**
HYCOM STD	**59**	7,286	7,563	**-3.7**	6,987	9,280	**-24.7**	7,875	7,312	**7.7**
	124	3,336	3,243	**2.9**	3,439	4,282	**-19.7**	3,941	3,336	**18.1**
	250	1,910	1,618	**18.1**	2,165	1,791	**20.9**	2,173	1,894	**14.7**
	501	1,113	1,042	**6.8**	1,309	1,128	**16.0**	1,372	1,217	**12.7**
AVUS LRG	**256**	6,042	7,201	**-16.1**	6,576	7,533	**-12.7**	6,568	8,053	**-18.4**
	512	3,394	3,768	**-9.9**	3,721	4,018	**-7.4**	3,702	4,366	**-15.2**
	1024	2,094	2,092	**0.1**	2,286	2,106	**8.5**	2,242	2,670	**-16.0**
	1280	1,850	1,769	**4.6**	2,026	1,773	**14.3**	1,972	2,340	**-15.7**
AVUS STD	**64**	7,062	7,835	**-9.9**	7,934	7,835	**1.3**	7,547	8,599	**-12.2**
	128	3,586	3,819	**-6.1**	4,115	4,102	**0.3**	3,906	4,244	**-8.0**
	256	1,923	1,918	**0.2**	2,243	2,036	**10.2**	2,119	2,180	**-2.8**
	384	1,366	1,293	**5.6**	1,619	1,371	**18.1**	1,524	1,486	**2.6**
ICEPIC LRG	**256**	3,746	3,497	**7.1**	4,812	4,796	**0.3**	6,016	6,371	**-5.6**
	512	2,251	2,563	**-12.2**	2,929	2,623	**11.7**	2,919	3,693	**-21.0**
	1024	2,245	2,271	**-1.1**	2,224	1,797	**23.8**	2,346	1,993	**17.7**
	1280	2,158	2,420	**-10.8**	2,143	1,772	**20.9**	1,936	2,245	**-13.8**
ICEPIC STD	**64**	4,284	4,185	**2.4**	5,419	5,082	**6.6**	6,791	7,321	**-7.2**
	128	2,469	2,970	**-16.9**	3,199	3,078	**3.9**	3,980	4,037	**-1.4**
	256	2,583	2,545	**1.5**	2,535	2,118	**19.7**	2,252	2,567	**-12.3**
	384	2,237	2,600	**-13.9**	2,212	2,086	**6.0**	1,902	2,000	**-4.9**

Table 2. Predicted and measured runtimes (in seconds) and prediction errors.

More importantly, the PSiNS Simulator enables users to investigate the accuracy of the model simulation at a finer granularity. Figure 2 illustrates a fine-grain comparison of the communication times simulated (predicted) to the measured times for each task of HYCOM for 124 processors on Babbage. The red vertical bars are used to represent the measured times whereas the green horizontal line is used to represent the simulated times. Figure 2 shows that despite the imbalance in communication times among tasks in HYCOM, the framework is quite accurate in predicting the communication time for the individual tasks.

In addition to comparing communication times for each task, the new framework enables an additional break down of the communication time into the time spent in each MPI routine. Using this feature, we measured the time spent for each type of MPI routine and compared the simulated times with the measured times. Figure 3 (a) presents the measured values for the percentages of time spent in each MPI routine over the total communication time whereas Figure 3 (b) presents the percentages for the PSINS simulation. Figure 3 shows that the percentage of time spent in MPI routines from the simulation closely matches the percentages from the actual run, indicating the accuracy of the framework at a finer granularity.

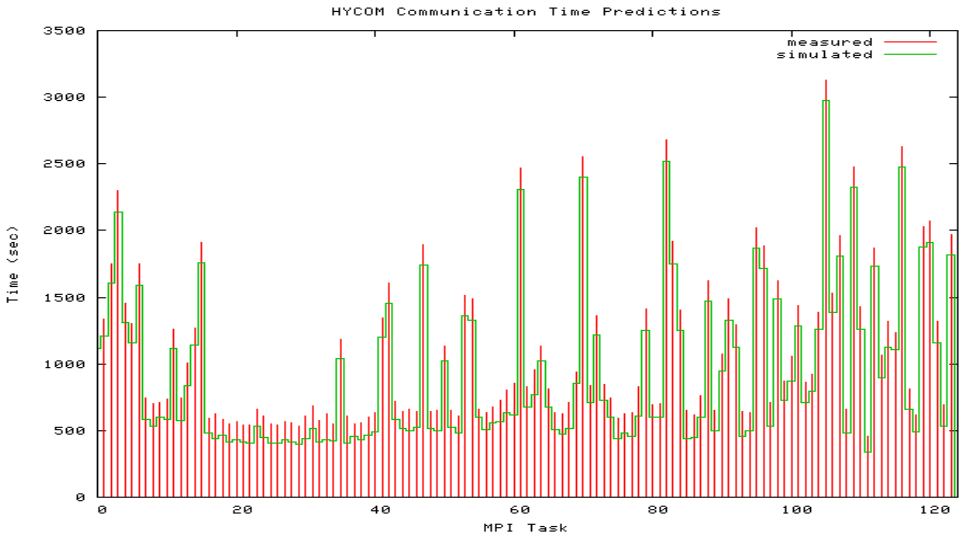
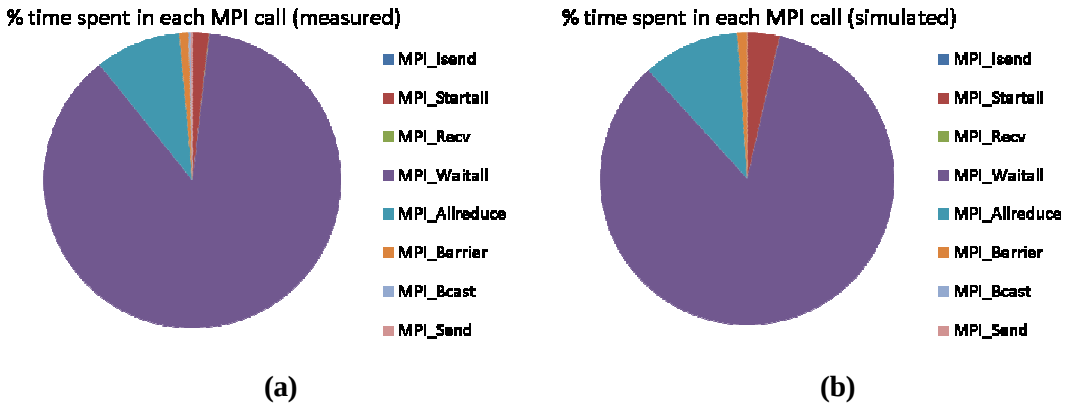


Figure 2. Measured and simulated communication times for all tasks.



(a) **(b)**

Figure 3. Communication time spent in MPI calls for HYCOM running with 124 tasks.

4 Related Work

Early work on performance prediction of HPC applications was done in the Proteus simulator [21], an execution-driven simulator which met many of the design goals that have been laid out for PSINS at the time. Proteus was designed modularly so that it could be customized for the target system and tradeoffs could be made between accuracy and efficiency by using a different implementation of a certain simulation component. Unfortunately Proteus introduces a slowdown of 2-35x for each process in the target application, which renders it cumbersome for the purpose of simulating long-running large-scale applications at thousand of processors.

Later work, such as Parallel Proteus [21], LAPSE [22], MPI-SIM [23] and the Wisconsin Wind Tunnel [24] improved the efficiency of the simulation required to make predictions by executing simulations in parallel. Typically these tools are execution-driven and perform parallel discrete event simulation and tend to be full machine simulators that address many aspects of a target architecture other

than the network. This causes them to be slower, more complex and less modular than PSINS for the purpose of MPI scaling investigations.

The Dimemas project [7] uses the concept of largely divorcing network prediction from the prediction of serial computation portions of the code. Like PSINS, the user supplies Dimemas with a speedup ratio for a target system. Dimemas uses this speedup ratio along with the MPI event trace (in their case called an MPIDTrace) to perform a discrete event simulation of the application on a target system. Unlike PSINS, Dimemas is not open source, hence though useful it is not quite satisfactory as a medium for community development in this arena. Dimemas currently stores their MPI event traces as an ASCII text file resulting in event traces files that are unnecessarily large.

5 Conclusions

Performance models can provide valuable information in tuning of both applications and systems, enable application-driven architecture design and extrapolate the performance of applications on future systems. In the constantly changing and growing field of HPC, it is important to have a modeling tool that is flexible enough to adapt to architectural changes and is scalable enough to grow with the constantly increasing system sizes. PSINS adds this flexibility and scalability to the PMaC Performance Prediction Framework.

Our results demonstrate that using PSiNS within the prediction framework results in high accuracy when predicting the performance of three large scale HPC applications for a number of different input decks and processor counts for three HPC systems.

6 Acknowledgments

This work was supported in part by the DOD High Performance Computing Modernization Program and the DOE Office of Science through the SciDAC2 award entitled Performance Evaluation Research Center.

References

1. D.H. Bailey and A. Snavely, “Performance Modeling: Understanding the Present and Predicting the Future”, EuroPar, 2005.
2. J. Michalakes, J. Hacker, R. Loft, M. McCracken, A. Snavely, N. Wright, T. Spelce, B. Gorda and R. Walkup. "WRF Nature Run," Supercomputing, 2007.
3. G. Alvarez, M. Summers, D. Maxwell, M. Eisenbach, J. Meredith, J. Larkin, J. Levesque, T. Maier, P. Kent, E. D'Azevedo and T. Schulthess. "New algorithm to Enable 400+ TFlop/s Sustained Performance in Simulations of Disorder Effects in High-Tc Superconductors," Gordon Bell Prize Winner, Supercomputing, 2007.
4. L. Carrington, D. Komatitsch, M. Tikir, M. Laurenzano, A. Snavely, D. Michea, J. Tromp and N. Le Goff. "High-frequency Simulations of Global Seismic Wave Propogation Using SPECFEM3D_GLOBE on 62K Cores," Supercomputing, 2008.
5. P. Ratn, F. Mueller, B. de Supinski and M. Schulz. "Preserving Time in Large-scale Communication Traces," Supercomputing, 2008.
6. R. Badia, J. Labarta, J. Giménez and F. Escalé. "Dimemas: Predicting MPI Applications Behavior in Grid environments," Workshop on Grid Applications and Programming Tools, 2003.
7. S. Girona, J. Labarta and R. Badia. "Validation of Dimemas Communication Model for MPI Collective Operations," Proceedings of the European Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, 2000.
8. S. Shende and A. Maloney. "The TAU Parallel Performance System," International Journal of High Performance Computing Applications, 2006.

9. W. Nagel, A. Arnold, M. Weber, H. Hoppe and K. Solchenbach. "VAMPIR: Visualization and Analysis of MPI Resources," Supercomputer 12(1):69–80, 1996.
10. MPI Profiling Interface, http://www.mpi-forum.org/docs/mpi-11-html/node152.html.
11. M Tikir, M Laurenzano, L Carrington and A Snavely. "PMaC Binary Instrumentation Library for PowerPC/AIX," Workshop on Binary Instrumentation and Applications, 2006.
12. Wikipedia contributors. UTF-8, http://en.wikipedia.org/wiki/UTF-8, accessed 2009.
13. PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications, Extended Version, http://www.sdsc.edu/pmac/projects/psins.html
14. Integrated Performance Monitoring, http://ipm-hpc.sourceforge.net/, 2008.
15. P. Mucci, S. Browne, C. Deane and G. Ho. "PAPI: A Portable Interface to Hardware Performance Counters," Department of Defense HPCMP Users Group Conference, 1999.
16. M. Tikir, L. Carrington, E. Strohmaier and A. Snavely. "A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations," Proceedings of the ACM/IEEE International Conference on Supercomputing, 2007.
17. W. Strang, R. Tomaro and M. Grismer. "The Defining Methods of Cobalt60: A Parallel Implicit, Unstructured Euler/Navier-Stokes Flow Solver," Institute of Aeronautics and Astronautics Paper 99-0786, 1999.
18. R. Bleck. "An Oceanic General Circulation Model Framed in Hybrid Isopycnic–Cartesian Coordinates," Ocean Modelling, 4, 55–88, 2002.
19. G. Sasser, J. Blahovec, L. Bowers, S. Colella, J. Luginsland and J. Watrous. "Current Emission, Resistive Losses, and Other Challenging Problems in the Simulation of High Power Microwave Components," Inst. of Aeronautics and Astronautics Paper, 1999.
20. Department of Defense, High Performance Computing Modernization Program. "Technology Insertion," http://www.hpcmo.hpc.mil/Htdocs/TI/, 2009.
21. E. Brewer, C. Dellarocas, A. Colbrook and W. Weihl. "Proteus: A High-Performance Parallel Architecture Simulator," MIT Technical Report MIT/LCS/TR-516, 1991.
22. P. Dickens, P. Heidelberger and D. Nicol. "A Distributed Memory LAPSE: Parallel Simulation of Message-Passing Programs," Proceedings of the 8th Workshop on Parallel and Distributed Simulation, 1994.
23. S. Prakash and R. Bagrodia. "MPI-SIM: Using Parallel Simulation to Evaluate MPI Programs," Proceedings of the Winter Simulation Conference, 1998.
24. S. Reinhardt, M. Hill, J. Larus, A. Lebeck, J. Lewis and D. Wood. "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers," Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1993.
25. A. Snavely, L. Carrington, N., Wolter, J. Labarta, R. Badia and A. Purkayastha. "A Framework for Performance Modeling and Prediction," Supercomputing, 2002.
26. M. Noetha, P. Ratna, F. Mueller, M. Schul, and B. R. de Supinski “ScalaTrace: Scalable compression and replay of communication traces for high-performance computing”, Journal of Parallel and Distributed Computing, 2008.
27. J. S. Kim, D. J. Lilja, “Characterization of Communication Patterns in Message-Passing Parallel Scientific Applications”, Proceedings of the Second International Workshop on Network-Based Parallel Computing, 1998.
28. X. Gao, B. Simon, A. Snavely, “ALITER: An Asynchronous Lightweight Instrumentation Tool for Event Recording”, Workshop on Binary Instrumentation and Applications (held in conjunction with PACT2005)
29. X. Gao, M. Laurenzano, B. Simon, A. Snavely, “Reducing Overheads for Acquiring Dynamic Traces”, International Symposium on Workload Characterization (ISWC05)
30. L. Carrington, A. Snavely, X. Gao, N. Wolter: A Performance Prediction Framework for Scientific Applications. International Conference on Computational Science 2003: 926-935
31. M. M. Tikir, L. Carrington, E. Strohmaier, A. Snavely: A genetic algorithms approach to modeling the performance of memory-bound computations. SC 2007: 47
32. M. M. Tikir, M. Laurenzano, L. Carrington, and A. Snavely, PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. Accepted for publication. Euro-Par 2009, August 25-28, Delft, the Netherlands