PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications

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

Abstract. The size of supercomputers in numbers of processors is growing exponentially. Today’s largest supercomputers have upwards of a hundred thousand processors and tomorrow’s may have on the order one million. The applications that run on these systems commonly coordinate their parallel activities via MPI; a trace of these MPI communication events is an important input for tools that visualize, simulate, 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. It also allows researchers to easily plug in different event trace formats and communication models, allowing it to interface gracefully with other tools. This provides a flexible framework for collaboratively exploring the implications of constantly growing supercomputers on application scaling, in the context of network architectures and topologies of state-of-the-art and future planned large-scale systems.

Keywords: High Performance Computing, Message Passing Applications, Performance Prediction, Trace-Driven Simulation, and Supercomputers.

1 Introduction

Performance models are calculable expressions that describe the interaction of an application with the computer hardware providing 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 permitted application 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 CPU architecture and communication (interconnect) technology. However as applications scale to larger CPU counts, the interconnect becomes a more prevalent factor in their performance requiring improved tools to measure and model them.

We present 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 have, in addition to its built-in trace formats, format conversion modules, and communication models, a graceful API designed such that anyone can easily extend these tools via plug-in virtual functions. PSINS interacts gracefully with other popular tracers and modeling and visualization tools such as that presented by Ratn et al. [5], MPIDtrace [6], Dimemas [7], TAU [8] and VAMPIR [9]. Figure 1 shows the high-level design of PSINS as well as the flow of information that occurs for performance prediction.

1.1 Tracer for Collecting Event Traces

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. The PSINS 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 individual communication events or the computation time, labeled as *CPUBurst*. To gather CPUBurst events, the library uses timers at the end and the beginning of each MPI routine replacement so that when an MPI function is called, the time spent since the end of the last MPI call to the current call is recorded in the trace.

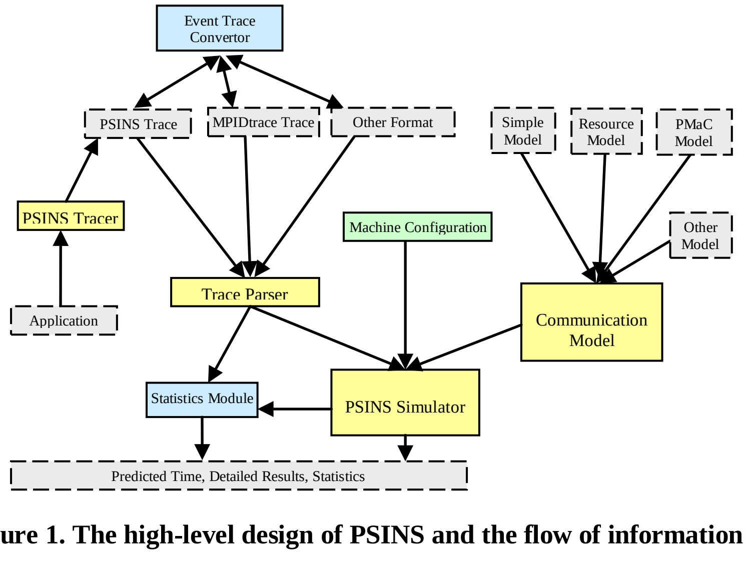


Figure 1. The high-level design of PSINS and the flow of information within.

Since HPC applications typically run for long duration and tend to execute millions of MPI function calls, recording each event to a trace file as it occurs is not practical since this would introduce many small, latency-bound file I/O accesses. Like other efficient tracing tools [11,28-29], PSINS Tracer uses per-task local memory buffers to temporarily store event information and only dumps the events in a task's buffer when that buffer is full. Moreover, to eliminate the need for any synchronization or any additional communication among tasks during tracing, initially PSINS generates a separate event trace file for each MPI task.

In a post-trace phase, to combine these separate trace files in to a single compact trace file, PSINS includes a trace consolidation utility, *mpi2psins*. The combining and compacting step is done serially after the execution of the traced application. This mpi2psins utility uses an encoding mechanism similar to general UTF encodings [12] in order to reduce the size of the final trace. It uses the most significant bit in each byte to determine the number of bytes that will be used to represent a number and the other seven bits to store the actual value. Using this technique it is possible to represent 2^{7n} possible values with n bytes. An event trace is made up mostly of small integers that represent processor IDs, larger integers that represent message sizes, and real numbers that represent times. On average our encoding saves 60% of the size that would be required if these values were kept as normal 4 byte or 8 byte values. The trace thus serves as a minimal complete representation of events to which further compression techniques such as those that detect and encode regular expressions can be applied [26]. More importantly, when carrying out strong scaling studies, the size of communication traces encoded by this method grows linearly as function of processor count *even though the global communications may grow exponentially* [27]. This is because the time becomes shorter (at least for scalable codes) and the message sizes tend to decrease, and thus the UTF encodings become smaller with increasing processor count even though the total number of communications may go up.

Besides tracing functionality, PSINS tracer provides two additional libraries for performance measurement and analysis that can be included in the event trace 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.

1.2 Adding a New Input Trace Parser

In PSINS, the trace parser module is included as a separate module to allow the simulator to use different input trace formats easily. This allows users to easily add another trace format such as TAU in addition to the already included parsers for PSINS and the MPIDtrace trace formats. A trace consists of a sequence of events that occur for each task and to use another trace format, the new parser needs only to convert events in the trace file to the PSINS internal representation of trace events.

In PSINS a new trace parser is added via use of virtual C++ functions. PSINS provides a base class, *Parser*, with a few virtual methods (see technical report [13] for more detail), which provide minimal functionality to access and consume the trace. Even though adding new parsers to PSINS requires some coding knowledge, PSINS hides most of the complexity of this process by providing most of the common infrastructure that is used by all parsers, requiring only the implementation of a few virtual methods. For example the parser for PSINS built-in trace format requires only 384 lines and the parser for MPIDtrace format requires 647 lines of C++ code.

1.3 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 models both computation and communication times for each task in the application. To simulate an execution on a target system, the simulator needs details about the configuration and construction of the system. These modeling parameters consist of configurable components of a parallel HPC system.

PSINS assumes that the target architecture is a parallel computer composed of multiple computation nodes connected via configurable number of global busses (as shown in Figure 2). Each computation node contains a configurable number of processing units (processors or cores) and incoming and outgoing links to the global busses. It provides the flexibility for each compute node to have different numbers of incoming and outgoing links to the global busses and different number of processing units in the node. In addition, the processing units within a compute node can be specified to have different speeds. By using a flexible description of the target system architecture, PSINS provides the capability to simulate varying types of systems ranging from computational grids to shared memory multiprocessor systems.

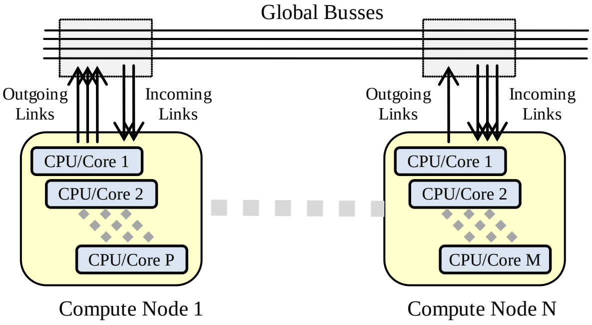


Figure 2. Target architecture for simulation

All of these configurable modeling parameters are given to simulator in a small ASCII configuration file. The configuration file contains parameters for the system as a whole, for each compute node and for the MPI task-to-processor mapping. For the system, required parameters include the number of compute nodes, the best achievable bandwidth and latency for each bus for two nodes to communicate, and the number of busses. For each compute node, required parameters include the number of processing units, the number of incoming and outgoing links from/to the busses, the best achievable local bandwidth and latency within the node, a mapping of MPI tasks to processors, and CPU ratios which describe relative speeds (ratios) for the computational work of the target with respect to the base system. This CPU ratio is used by PSINS to model the computation time. Modeling done by simply projecting the time spent for each CPUBurst event to the target system using this ratio of how much faster or slower the processing unit in the target system is relative to the base system. This approach has been shown to be effective in previous research [1,6].

PSINS Simulator consumes events from the input trace in the order of their occurrence. The simulator uses an event queue based on priority queues to replay the input trace. When an event is read, it is tagged with the earliest time it will be ready for execution as its priority. This time is the value of the per-task timer at the time of insertion for the task that event belongs to. If an event is not ready for execution such as a blocking receive, or global communication, it is re-inserted into the event queue for later processing with its priority reduced. An event is deleted from queue when its execution is over. When an event is executed, it is marked with its execution time as well as its wait time. The wait time is a record of time the event had to wait for its execution as in imbalanced parallel applications with blocking communications or barriers. After its execution, the execution timer for its task is incremented accordingly and global timer is updated for synchronous simulation.

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 CPU ratio described above. 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 the communication, communication model is asked to calculate the bandwidth and latency at the time of its execution and the event is executed.

PSINS Simulator includes a statistics module to collect detailed information about the simulation of an event trace on the target system, similar to IPM. 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. It also collects the waiting time for each event type to provide information on load balancing. 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 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.

2 Communication Models

PSINS isolates the modeling parameters and communication models from the simulator (as shown in Figure 1) to enable users to easily investigate new communication models. From the perspective of the PSINS Simulator, the communication model is a black box. By separating the parameters for the target system from the communication model, PSINS allows even more flexibility toward investigating the impact of different communication models.

The communication model takes an event, the parameters from the configuration file, and the current state of the simulated system to calculate the sustained latency and bandwidth for the messages that are associated with that event. The model is responsible for determining when an event will be executed, which might be at some point in the future due to the unavailability of resources or some other measure of contention. The model also determines which resources it will require and for how long the resources are required, which in turn can change the state of the simulated system based on the needs of the event. The communication model then calculates the time to complete the event including the time to transmit a message as well as the time that the message must wait for resources (wait time). Moreover, each event can have its own model. These models can be simple (i.e. based on bandwidth and latency) or more complex functions of the system's state, the number of processors involved in the event, and the scalability of the event on the network.

2.1 Built-in Models

PSINS includes several built-in communication models that can be used to investigate a target system. These models are the *simple* model, the *resource contention* models, and the *PMaC* model. Our experience [16] indicates that these models can accurately be used to model application performance for a majority of today’s HPC systems.

The simple model uses the best sustainable bandwidth and latency from the configuration file and assumes the resources available to the system are infinite. That is, when a message is ready to be sent, it assumes that resources along the path of the message are available and calculates the time to send the message as a simple addition of latency to the time spent to transfer the message body. For collective communications, this model uses a simple description for each communication event that indicates whether that event scales in *linear*, *logarithmic* or *constant* time with respect to the number of participating tasks. The simple model is designed to model the lower bound for the communication time for an application.

PSINS provides three resource contention models based on the number of global busses, incoming, and outgoing links from compute nodes, called *bus-only*, *incoming-link-only,* and *outgoing-link-only* models. These models assume that the number of a certain type of resource that is available for communication is limited and use a scheduling algorithm to schedule each message based on resource availability. These models are designed to investigate the impact of resource contention on the performance of an application. For instance, by predicting the performance of an application for an increasing number of busses, users can get a feel for how sensitive the application's performance is to number of busses available, which in turn can identify whether the application posts multiple messages at around the same time.

In addition to simplistic models, PSINS also includes a more complex communication model, called the *PMaC* model. This model is more complex than the previous models in order to increase the accuracy of the simulations. For point-to-point communications, this model takes the number of outstanding messages at the time of a message delivery and, based on the current load on the busses and input and output links, scales the maximum bandwidth accordingly.

For collective communications, alternative to using simple description of each MPI collective communication routine, the PMaC model also provides the means to use a more complex and realistic bandwidth calculations based on message sizes. This is done by measuring the bandwidth for each collective communication routine for an increasing size of messages using the synthetic benchmark, *PSINSBench*, that is included in PSINS package (see technical report [13] for details). Then using a curve-fitting algorithm the measured bandwidths are fit to a continuous function, which is later used by the model to calculate the bandwidth for a given message size.

2.2 Adding a New Model

In addition to the built-in models, PSINS allows users to easily plug-in new communication models. Like trace parsers, new communication models are added with virtual C++ functions. PSINS provides a base class, *Model*, with some virtual methods (see [13] for the list of virtual functions). These virtual methods provide the functionality to schedule events on resources as well as to calculate the time it takes to execute an event. Then, to create a new communication model, the developer needs only to define a class that extends the Model class and implement its virtual functions.

Much of the burden of the model developer then resides in the areas that are almost completely model-specific, which leaves only a few virtual functions for the developer to implement. Among the built-in models in PSINS, the simplest model requires 228 lines of C++ code. A collection of resource contention models requires 158 lines of C++ code and the most complex model requires 433 lines of C++ code.

3 Experimental Results

To demonstrate the usability, efficiency and accuracy of PSINS Tracer and Simulator, we have conducted several experiments where 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].

All of the PSiNS traces were collected on a base system, NAVO's IBM Cluster 1600 (3072 cores connected with IBM's High Performance Switch), called *Babbage*. We ran each scientific application with two of their input data sets, namely *standard* and *large*, with 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 simulation of the collected traces, we ran the simulator on a Linux box with two dual-core processors. In addition to simulating the base system Babbage, we also simulated the MHPCC's Dell Cluster, called *Jaws* (5120 cores connected with Infiniband) and ERDC's Cray XT3 system, called *Sapphire* (8320 cores connected with Cray SeaStar engine). To compare PSINS to a state-of-art simulation tool, we also collected MPI event traces using MPIDtrace[6] and simulated them using Dimemas[6] for each application and processor count. We present the results as event trace sizes, simulation times, and prediction accuracy.

3.1 PSINS Trace Sizes and Simulation Times

The sizes of PSINS traces collected for each application and processor count is given in Figure 3. The figure illustrates that the size of PSINS event traces grows linearly as the processor count grows. The sizes range from 4GB to 32GB and are at least 4 times smaller than the event trace sizes generated by a similar state-of-the-art MPI event tracer, MPIDtrace [6] (the sizes of traces from MPIDtrace are presented in [13]).

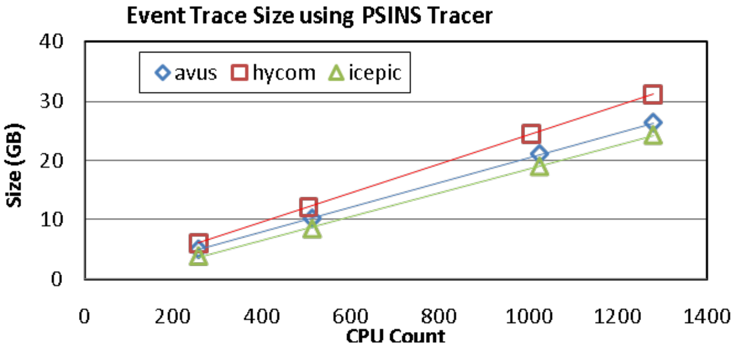


Figure 3. PSINS event trace size vs. CPU count for 3 applications.

These results suggest that one could practically store uncompressed traces for 10 thousand processors in about 300GB and for 100 thousand processor jobs in about 3TB using PSINS Tracer. Some compression techniques such as those used in [26] would be useful at large scale, though we note that some research groups already devote terabytes to storing the memory traces of strategic applications [25], so this same amount of storage devoted to communications traces is not out of the question.

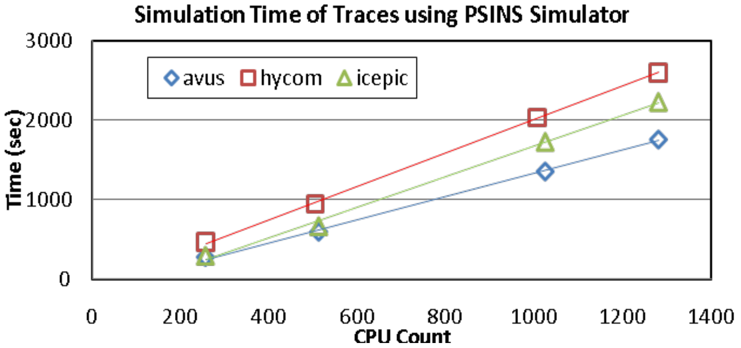


Figure 4. PSINS Simulator simulation time vs. CPU count for 3 applications.

These collected event traces were then fed through the PSINS Simulator, the simulation times are presented in Figure 4. The figure shows that PSINS Simulator is able to replay these collected traces for a target system in under 1 hour for all applications. On average the replay takes 7 times less time than running the program, however the replay time also grows linearly with processor count suggesting that in the future the replay procedure should itself be parallelized using the natural synchronization points that occur at global communications. However, these simulation times are already an order of magnitude faster than a similar network simulator Dimemas [6] (for a detailed comparison to Dimemas see [13]).

Combined, Figure 3 and Figure 4 show that for each application there is a linear correlation between the input trace size and the time it takes to replay the trace for a target system in PSINS. They also demonstrate that PSINS Tracer collects MPI event traces of manageable and tractable sizes and PSINS Simulator replays these traces in a tractable time for a target system. This indicates that as applications scale to even larger processors counts, PSINS is likely to continue to be usable and effective.

In addition to trace size and simulation time, it is also important to quantify the overhead introduced by the PSINS Tracer itself during trace collection even though the cost is only born once. During our experiments, we observed that the overhead of PSINS Tracer ranges from 0.2% to 14.8% compared to the original execution times of the applications, which is very similar to the overhead of the state-of-art tool MPIDtrace. The average overhead for all applications and processor counts is 5.9% meaning it can be efficiently used for large processor counts, even in production runs.

3.2 Simulation Accuracy

Even though the usability of PSINS in terms of event trace sizes and simulation efficiency and tracing overhead is important, what matters most is the accuracy of the predictions produced by the models. To investigate accuracy at a finer granularity, we simulated an event trace collected using PSINS Tracer for HYCOM with 124 processors for the base system and compared the communication times simulated to the measured times for each task. For this experiment we used the built-in simple communication model.

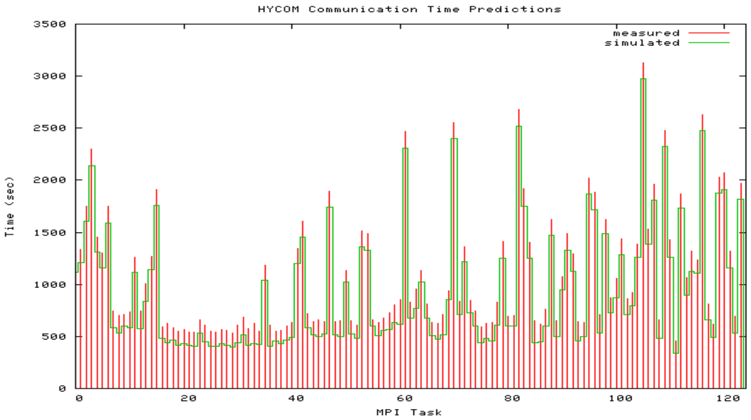
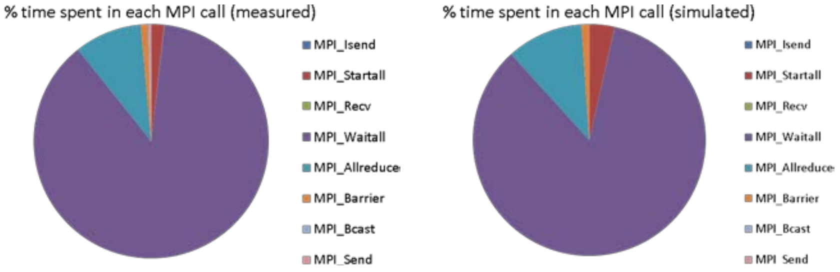


Figure 5. Measured and simulated communication times for all tasks.

Figure 5 presents the communication times measured and predicted for each task. The red vertical bars are used to represent the measured times whereas the green horizontal line is used to represent the simulated times. Figure 5 shows that PSINS Simulator is quite accurate in predicting the communication time for each task. The average absolute error in predicting the communication times for all tasks is 17% whereas the error in predicting the total communication time is 14%. More importantly, Figure 5 shows that despite the imbalance in communication times among tasks, the results of PSINS simulation closely match the observed behavior. Note again that the results in Figure 5 show simulation results using the built-in simple model, which tends to under predict the communication times.

In addition to comparing communication times for each task, we further broke down the communication time into the time spent in each MPI routine. Figure 6 (a) presents the measured values for the percentages of time spent in each MPI routine and Figure 6 (b) presents the percentages for the same MPI routines from the PSINS simulation. Figure 6 shows that the percentage of time spent in MPI routines from the simulation closely matches the percentages from the actual run, indicating that the simulation results match the measured results at an even finer granularity.



(a) **(b)**

Figure 6. Communication time spent in MPI calls for HYCOM.

Table 1 presents a comparison between the total communication times measured during an actual run and times simulated by PSINS for two HPC systems. We used the PMaC model for the simulations listed in this table; it shows the ability to predict the communication times of applications within 15% error for all cases except AVUS running with 64 processors on Sapphire. The absolute average error among all cases is only 9.0%. Overall, Table 1 demonstrates that PSINS is effective in modeling and predicting the performance of applications for HPC systems. The largest error in communication time occurs in AVUS with 64 processors on Sapphire. However despite a large relative error, the communication time accounts for only 7% of the overall execution time and the runtime prediction error from Table 2 is only 2%.

	CPU Count	Jaws			Sapphire		
		Simulated	Measured	% Error	Simulated	Measured	% Error
HYCOM	124	121,476	128,285	-5%	161,055	167,620	-4%
HYCOM	504	449,646	519,335	-13%	573,365	621,793	-8%
AVUS	64	27,764	26,194	6%	30,680	22,561	36%
AVUS	1280	1,333,414	1,193,967	12%			
ICEPIC	64	72,144	71,073	2%	89,950	88,708	1%
ICEPIC	1280	1,178,914	1,142,970	3%			

Table 1. Total time (sec) spent in communication events (using PMaC model).

Similarly, Table 2 presents the comparison between the execution times measured during actual runs and runtime predictions from PSINS (recall that event traces are collected on a different system than these HPC systems). In PSINS simulations, the relative speed of compute units in each target system to the base system is calculated using the PMaC Prediction Framework [25].

Table 2 shows that the absolute prediction error using PSINS is under 10% for majority of the cases and except 4 cases for Sapphire, it is under 15%. The prediction error ranges from -9.9% to 6.8% for Jaws (average absolute error is 7.4%) and it ranges from -26.3% to 18.1% for Sapphire (average absolute error is 11.6%). Further investigation has shown that error in the relative speed calculation of its compute units is largely responsible for the higher error for the Sapphire predictions. Table 2 demonstrates that PSINS is effective in modeling and predicting the overall execution times of applications on HPC systems as well as the total communication times.

	Input Deck	**CPU Count**	**Jaws**			**Sapphire**		
			Runtime (sec)		**% Error**	**Runtime (sec)**		**% Error**
			Simu.	**Meas.**		**Simu.**	**Meas.**	
HYCOM	**STD**	124	3,336	3,243	2.9	3,439	4,282	-19.7
		501	1,113	1,042	6.8	1,309	1,128	16.0
	LRG	256	5,973	5,800	3.0	5,756	5,956	-3.4
		504	2,816	3,002	-6.2	2,764	3,752	-26.3
AVUS	**STD**	64	7,062	7,835	-9.9	7,934	7,835	1.3
		384	1,366	1,293	5.6	1,619	1,371	18.1
	LRG	512	3,394	3,768	-9.9	3,721	4,018	-7.4
		1280	1,850	1,769	4.6			
ICEPIC	**STD**	64	4,284	4,185	2.4	5,419	5,082	6.6
		384	2,237	2,600	-13.9	2,212	2,086	6.0
	LRG	512	2,251	2,563	-12.2	2,929	2,623	11.7
		1280	2,158	2,420	-10.8			

Table 2. Simulated (using PMaC model) and measured runtimes (sec).

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 and 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 large event traces files.

5 Conclusions

Performance models can provide valuable information in the 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 has this flexibility and scalability along with specific features that make it practical to use for model generation. PSINS tracer allows event traces to be captured with low overhead and recorded at manageable sizes even for large processor counts of MPI applications. PSINS simulator is capable of simulating different HPC networks with a high degree of accuracy in a reasonable amount of time. This makes PSINS is a multifunctional tool of which flexibility, scalability, and accuracy allow its utilization in collaborative studies involving modeling large scale HPC applications.

PSINS Tracer and Simulator is already ported for several HPC systems and is available at http://www.sdsc.edu/pmac/projects/psins.html.

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.

6 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. 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," Work. on Grid Applications and Prog. 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. 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. M. Tikir, L. Carrington, E. Strohmaier and A. Snavely. "A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations," SC 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," 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
29. X. Gao, M. Laurenzano, B. Simon, A. Snavely, “Reducing Overheads for Acquiring Dynamic Traces”, International Symposium on Workload Characterization (ISWC05)