

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

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

Performance Modeling and Characterization Lab
San Diego Supercomputer Center
9500 Gilman Drive, La Jolla, CA
{mtikir, michael, 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 valuable information about the implications of constantly growing supercomputers on application scaling, and provides a means to explore the 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 or performance predictions are calculable expressions that describe the interaction of an application with the computer hardware. These 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 [1]. A recent trend in High Performance Computing (HPC) is the increase in the core count in supercomputers, which in turn has resulted in HPC applications to scale to tens or even hundreds of thousands of cores in recent years [2,3,4]. Moreover, performance models have been effectively used to enable applications to scale to these larger processor counts [2,4]. As this trend continues, it becomes even more important to accurately and efficiently model the performance of large HPC applications in order to gain a better understanding of application/machine interaction and their bottlenecks to higher scalability.

The performance of an HPC application is a complex function of many factors such as algorithm, implementation, compiler, underlying processor architecture and communication (interconnect) technology. As applications scale to larger processor counts, the communication technology becomes a more dominant factor in their performance, especially in Message Passing Interface (MPI) applications. These large scale applications combined with constantly changing network architectures of state-of-the-art large-scale systems presents new challenges to modeling and predicting application performance.

In this paper, we introduce an efficient, accurate and flexible trace-driven performance modeling tool, PSINS, for MPI applications. PSINS includes two major components, one for collecting event traces during application runs (*PSINS Tracer*), and the other for the simulation of event traces (*PSINS Simulator*) for the modeling of target HPC systems. Similar to previous event trace tools such as the tool in [5], MPIDtrace [6], TAU [8] and VAMPIR [9], PSINS Tracer produces an event trace that consists of a detailed description of each communication event and a record of the CPU bursts between these communication events during an application run. PSINS Simulator takes as input an event trace for an application and a set of parameters that describe a target system to model, and then

replays the event trace using the specified performance model in order to yield a performance prediction for that target system.

A key design goal for PSINS is its usability for applications at large processor counts. PSINS generates compact and tractable event traces for fast and efficient simulation while producing accurate performance predictions. In addition to its built-in trace format and communication models, PSINS provides the means to easily plug-in different event trace formats and communication models through the use of virtual functions. PSINS hides most of the complexity of this process by providing most of the common infrastructure that will be used by all parsers and models, only asking for the implementation of a few virtual methods.

This paper is organized as follows: Section 2 describes the PSINS architecture for its tracer and simulator. Section 3 describes the built-in communication models and shows how a new communication model can be added to the simulator. Section 4 presents the experimental results for the sizes of event traces collected and the times to replay event traces, and the results on the accuracy of PSINS. Section 5 presents the related work and finally, Section 6 concludes.

2 PSINS Tracer and Simulator

PSINS is a trace-driven performance prediction tool for MPI applications. PSINS includes two major components: PSINS Tracer is used for collecting event traces on a *base* system during application runs and PSINS Simulator replays the collected traces and model parameters for the *target* system for a performance prediction. In addition, PSINS includes other components such as an input event trace parser, communication models, and trace converters. Figure 1 shows the high-level design of PSINS as well as the flow of information that occurs for performance prediction. This flow and the components of PSINS are described in the subsections below.

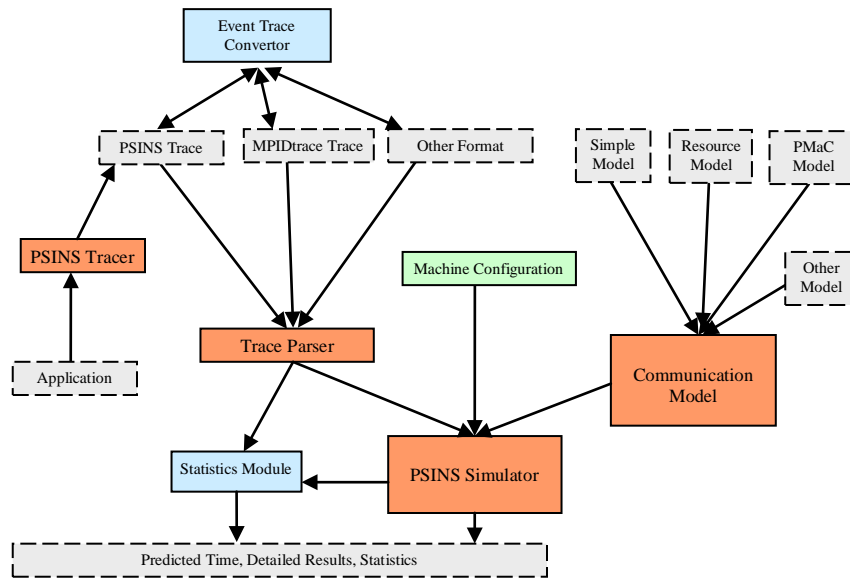


Figure 1 The high-level design of PSINS as well as the flow of information.

2.1 Tracer for Collecting Event Traces

The PSINS tracer collects an event trace for an application. 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 to each MPI call by sharing the same signature as the actual invocation.

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, the tracer library 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.

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 due to the significant I/O overhead. Like other efficient tracing tools [11], PSINS Tracer uses per-task local memory buffers to temporarily store event information and only dumps the events when the buffer for the task is full. Moreover, to eliminate any additional communication due to tracing, PSINS Tracer initially generates a separate event trace file for each MPI task.

To combine these separate trace files in to a single compact trace file, PSINS includes a trace consolidation utility, *mpi2psins*. This is done serially after the execution of the traced application. The *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. For instance, values less than 128 require 1 byte, values less than 16K require 2 bytes and so on. In the average, fewer bytes are used to represent each number because most numbers can be represented in fewer than 4 bytes. The final compact trace file produced by *mpi2psins* utility is in binary format and includes events for each task in the order of their ranks. In addition, for faster access and seek operations, it includes a header that holds file offset indices and number of events for each task.

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 event 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 in the underlying processors such as FLOP rate and 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 [13] and provides only a subset of information IPM provides.

PSINS Tracer library is ported and available for several HPC systems. Due to use of the PMPI interface and PAPI [14] for performance monitoring, porting PSINS Tracer to a new system requires only minimal modifications in the form of implementing fine-grained fast timers and defining the byte ordering (endianness) that the underlying processors use for data formatting.

2.2 Adding a New Input Trace Parser

In PSINS, the trace parser module is included as a separate module to allow the simulator use different input trace formats easily. This allows users to easily add another trace format such as TAU or the tool in [5] in addition to the already included parsers for PSINS and the MPIDtrace trace formats. An event trace consists of a sequence of events that occur for each task and to use another trace format, the parser for that format needs only to convert events 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 Figure 2). These virtual methods provide minimal functionality to access and consume the input trace. Hence, to add a new input trace format to PSINS, user needs to define a new class that extends the *Parser* class and implement its virtual functions accordingly. The constructor for this class takes the path to the trace file in addition to the number of tasks. The virtual function *getNextEvent* takes the task id and returns the event object corresponding to the next available event in the trace for that task.

```

class Parser {
protected:
    Parser(char*    traceFilePath,
           char*    application,
           char*    dataset,
           uint32_t taskCount);

public:
    .....
    virtual Event* getNextEvent(uint32_t taskId);
    virtual bool  isTraceConsumed(uint32_t taskId);
    virtual uint64_t getTotalEventCount();
};

```

Figure 2 Definition of Parser base class for input trace parsing.

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's built-in trace format requires only 384 lines of C++ code and the parser for MPIDtrace format requires 647 lines of C++ code.

2.3 Simulator for Performance Prediction

Execution time of an MPI application can be divided into two components, *computation* and *communication* time. The computation time is the time spent between MPI events whereas the communication time is the time spent during MPI events. To accurately simulate an MPI application on a target system, a prediction tool needs to model both computation and communication times for each task in the application.

PSINS Simulator takes an 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. Each step of this process is described in detail in the following subsections.

2.3.1 Target Machine Modeling Parameters

To simulate the execution of a target system the simulator needs details about the configuration/construction of the system. These details or modeling parameters consist of configurable components of a parallel HPC system.

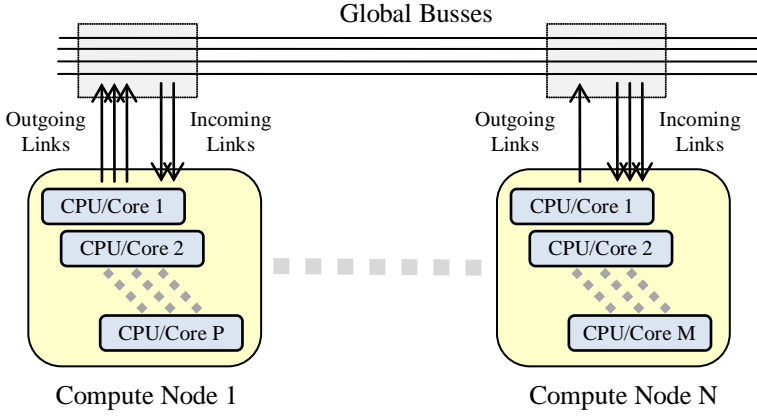


Figure 3 Target architecture for simulation

To begin with, 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 3). Each computation node contains configurable number of processing units (processors or cores) and incoming and outgoing links to the global busses. PSINS assumes the global busses as well as incoming and outgoing links for the compute nodes all have homogenous properties. However, 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 relaxing the restrictions on the target system architecture, PSINS provides the capability to simulate varying types of systems ranging from computational grids to shared memory multiprocessor systems.

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 busses, the best achievable local bandwidth and latency within the node, a mapping of MPI tasks to processors, and a CPU ratio which describes relative speeds (ratios) for the computational work of the target with respect to the base system on which the trace was collected.

This CPU ratio is used by PSINS to model the computation time. This is done by simply projecting the time spent for each CPUBurst event to the target system using the multiplicative factor of how much faster or slower the processing unit in the target system (the CPU ratio) is relative to the base system. This approach has been shown to be effective in previous research [1,6].

By separating the parameters for the target system from the communication model used for the simulation (as shown in Figure 1), PSINS allows even more flexibility toward investigating the impact of different communication models. The PSINS built-in communication models are described in detail in Section 3.1 but the process of simulating the events is described next.

2.3.2 Execution of Trace Events

PSINS Simulator consumes events from the input trace in the order of their execution within the simulator rather than consuming per-task basis. 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. That is, during simulation, an event may be re-inserted in the event queue multiple times and 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, and the next event from its task is inserted in to the queue.

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 CPU burst event, it is completed by calculation of its time on the target system using the CPU ratio described above. Point-to-point communications are executed by calculating achievable latency and bandwidth for the message based on the state of the system and communication model used and then the message is scheduled for sending or receiving. 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 global communication, communication model is asked to calculate the achievable bandwidth and latency at the time of its execution and the event is executed. For simplicity, PSINS assumes that for a global communication to execute, all participating tasks first synchronize at the time of posting by the last arriving task and then the message(s) are sent according to that event's model. Each event can have a different model based on network or system configuration. These models are described in more detail in Section 3.

2.3.3 Output Information from the Simulator

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 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 the message sizes and on the ranges of bandwidths calculated by the communication model for the communication events.

Such information provides valuable feedback to the 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, such 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 Communication Models

PSINS isolates the modeling parameters and communication models from the simulator to enable users to easily plug-in new communication models. From the perspective of the PSINS Simulator, the communication model is a black box. When a communication or synchronization event is about to be executed by the simulator, the communication model is asked to calculate how long a communication or synchronization among tasks will take for the event given the state of the system.

The communication model takes an event and can use the parameters from the configuration file and the current state of the simulated system (such as busses or incoming and outgoing links) to calculate the sustained latency and bandwidth for remote (inter-node) or local (intra-node) 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 can then calculate 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).

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 systems state, the number of processors involved in the event, and the scalability of the event on the network. PSINS simulator allows the easy inclusion of a user-defined communication model but also includes built-in communication models. The built-in models and the process of adding a new model are described in the subsections below.

3.1 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 [15] indicates that these models can accurately be used to model application performance for a majority of 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 (global busses, incoming and outgoing links from compute nodes) are available and calculates the time for 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 type (such as Gather, Allreduce or Alltoall) that indicates whether that communication is implemented 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-system interaction.

As an extension to the simple model, PSINS provides three resource contention models based on the number of global busses, incoming, and outgoing links from compute nodes. These are called *bus-only*, *incoming-link-only*, and *outgoing-link-only* models. Unlike the simple model, 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 at the earliest time a resource is available along the message's path through the resources. 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. It is in this manner that studying these types of predictions can aid in our understanding of the communication patterns that occur during an application run and how this pattern interacts with the systems hardware.

3.1.1 PMaC Model

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 given in the configuration file to determine the usable bandwidth for this communication.

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 via measuring the bandwidth for each collective communication routine for an increasing size of messages on a target system. Then using a curve-fitting algorithm the measured bandwidths are fit to a continuous function. The fitted function is later used by the model to calculate the bandwidth for a given message size for a given collective communication event.

To measure the bandwidths for each collective communication routine, PSINS package includes a synthetic benchmark, *PSINSBench*. For each collective routine, this benchmark executes the routine over the network repeatedly for each message size and records the time spent to execute the communications in addition to the total number of bytes transferred. Figure 4 presents the bandwidths measured by PSINSBench on an IBM Cluster 1600 Power 5 system using a dedicated high-speed-network of which the bandwidth is measured to be 3.4GB/s using simple ping-pong test. The figure illustrates that for all the collectives, the measured bandwidth is a fraction of that measured by a point-to-point event.

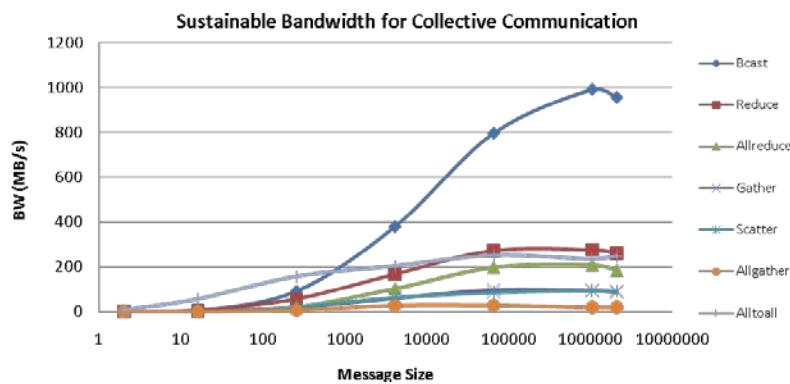


Figure 4 Sustainable bandwidths for some of the MPI collective routines on an IBM Cluster 1600 Power 5 system with a high-speed network.

3.2 Adding a New Model

In addition to the built-in models, PSINS allows users to easily plug-in new models. Like trace parsers, new communication models are added via use of virtual C++ functions. PSINS provides a base class, *Model*, with some virtual methods (shown in Figure 5). 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 user needs to define a class that extends the *Model* class and implement its virtual functions. Constructor for the extended model class is given the flexibility to take additional information that may be passed to the model to be used in bandwidth and latency calculations.

```
class Model {
public:
    virtual double    calcModelTime (Event* event,
                                   LinkedList<Event*>* others);
    virtual uint32_t schedule (P2PCommEvent* event,
                              double taskTime, double* scheduledTimes);
protected:
    virtual void      initResources ();
    virtual double    reserveResources (double afterThisTime,
                                       uint64_t messageSize,
                                       uint32_t from, uint32_t to,
                                       Reservation* reservation);
    virtual void      updateResources (Reservation* reservation);
};
```

Figure 5 Definition of Model class for adding new communication Models.

The two major virtual methods that need to be implemented for a new communication model are the *calcModelTime* and *schedule* methods. The *calcModelTime* method calculates the time to execute a given event, broken down into wait time and transmission time, based on the state of the simulated system at the time of its execution. It is given an event as well as a list of other events that the given event is related to, such as the events that comprise a collective communication from all of the other tasks participating in that communication or the matching receive event that goes with the given send event. The *schedule* method takes a point to point communication event and the current timer for the source task. It is responsible for determining when each of the messages in that event will arrive at its destination. In addition to these two major methods, extended Model classes that use some type of resource model should also provide internal methods to manage these resources.

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.

4 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 [16], HYCOM [17] and ICEPIC [18] from the HPCMPO TI-09 Benchmark Suite[19]. The traces were then simulated for a set of target HPC systems and the results of these simulations were compared to the actual measurements gathered on the target systems.

All the traces were collected on the base system, which was NAVO'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. 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 with Intel Woodcrest, called *Jaws* (5120 cores connected with Cisco Infiniband) and ERDC's Cray XT3 system, called *Sapphire* (8320 cores connected with dedicated Cray SeaStar communications 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 results of these experiments in terms of event trace sizes, simulation times, and prediction/simulation accuracy.

To demonstrate the usability and efficiency of PSINS, we first present results for trace sizes of MPI event traces collected via PSINS Tracer as well as times to simulate these traces for a target system using PSINS Simulator. We also present results for the overhead introduced by PSINS Tracer during trace collection/consolidation.

4.1 PSINS Trace Sizes and Simulation Times

To demonstrate the usability and efficiency of PSINS, PSINS Tracer was used to collect event traces for AVUS, HYCOM and ICEPIC each at 4 different processor counts ranging from 256-1280. 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 data set (i.e. strong scaling).

The sizes of the traces for each application and processor count is given in Figure 6. This figure illustrates that the size of PSINS event traces grows linearly as the processor count grow and the sizes range from 4GB to 32GB. These sizes are over 4 times smaller than the event trace sizes generated by another MPI event tracer MPIDtrace, presented in [6]. More importantly, Figure 6 shows that for complex scientific applications running on 1280 processors for a long duration, PSINS generates event traces with manageable space requirements.

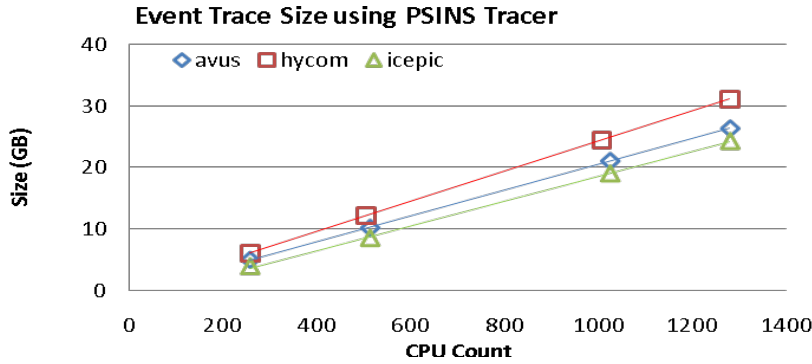


Figure 6 PSINS event trace size vs CPU count for 3 applications.

These collected event traces were then fed through the PSINS Simulator to measure the time it takes to simulate each trace for a target system. The results of these simulation times are presented in Figure 7. Figure 7 shows that PSINS Simulator is able to replay these collected traces for a target system in under an hour for all applications with all processor counts. These simulation times are 10 to 15 times less than another network simulator Dimemas presented in [6].

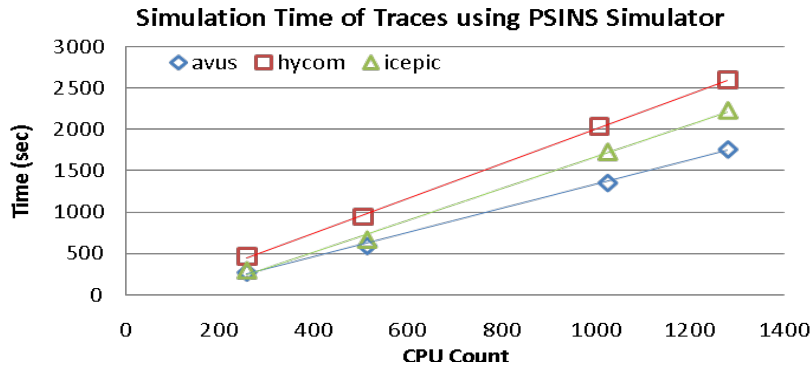


Figure 7 PSINS simulator simulation time vs. CPU count for 3 applications.

Overall, Figure 6 and Figure 7 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. 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 processor counts, PSINS is likely to continue to be usable and effective in modeling and predicting the performance of applications on large-scale HPC systems.

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. 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. The average overhead for all applications and processor counts is 5.9%. This shows that PSINS Tracer does not introduce significant overhead to collect an MPI event trace during an application run can be efficiently used for large processor counts.

4.2 Comparison of PSINS to MPIDtrace and Dimemas in terms of Trace Sizes and Simulation Times

This section presents the results of comparison of PSINS traces sizes to MPIDtrace trace sizes in addition to the PSINS simulation times to Dimemas simulation times for AVUS. We used two data input sets, standard and large, for a varying CPU counts. For a fair comparison, we collected MPI event traces using MPIDtrace on the same base system as the traces collected using PSINS and ran the Dimemas simulations on these traces on the same Linux system PSINS simulations are run.

Figure 8 shows the sizes of traces collected using PSINS Tracer and MPIDtrace for both input sets and varying CPU counts. It shows that the sizes of traces collected by the PSINS Tracer is significantly smaller compared to the sizes of traces collected using MPIDtrace for both input data sets.

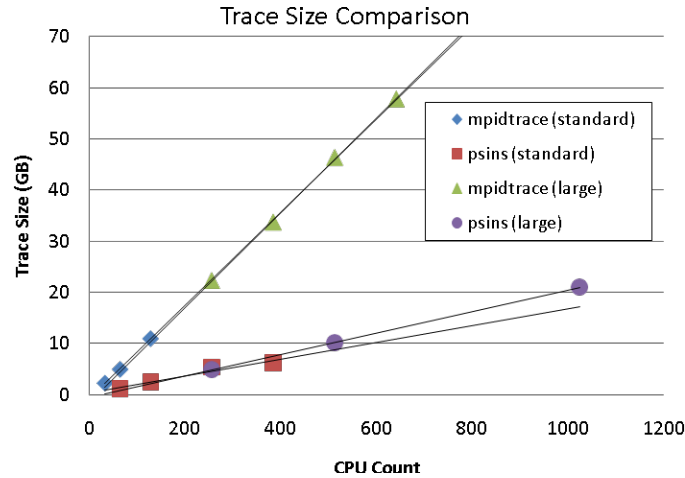


Figure 8 Comparison between PSINS and MPIDtrace in terms of trace sizes.

Figure 9 shows the simulation times of traces collected using PSINS Simulator and Dimemas for both input sets and varying CPU counts. It shows that the simulation times of traces by the PSINS Simulator is significantly smaller compared to the simulation times of traces using Dimemas for both input data sets.

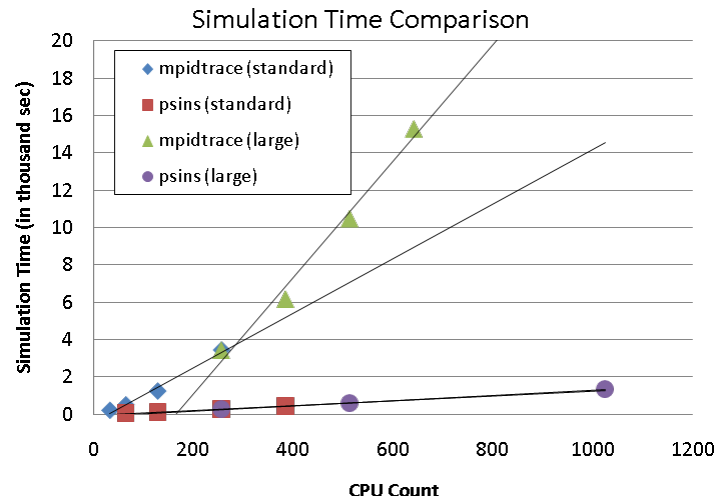


Figure 9 Comparison of PSINS and Dimemas in terms of simulation times.

4.3 Simulation Accuracy

Even though the usability of PSINS in terms of event trace sizes and simulation efficiency is important, what matters most is the accuracy of the predictions produced by the models. To investigate its accuracy at a finer granularity, we simulated an event trace collected using PSINS Tracer for HYCOM with 124 processors using the standard data input set for the base system and compared the communication times simulated to the measured times for each task. We further broke down the communication time per MPI event and compared those simulated times with the measured times. We used the built-in simple communication model in PSINS for the simulation of this application.

Figure 10 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 10 shows that PSINS Simulator is quite accurate in predicting the communication time for each task for HYCOM with 124 processors. 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 10 shows that despite the imbalance in communication times among tasks, the results of PSINS simulation closely match the observed behavior for communication times of each task

Figure 10 also shows that PSINS Simulator tends to slightly under predict the communication times for majority of the tasks compared to the actual communication times measured. This is due to the fact that we used the built-in simple model as the communication model and the simple model uses the best achievable bandwidth and latency values given in the configuration file and assumes no resource contention. If desired, for lower error in communication times, users can use one of the other built-in models. We used the simple model for this experiment to also be able to show the effectiveness of PSINS even with simplest communication model.

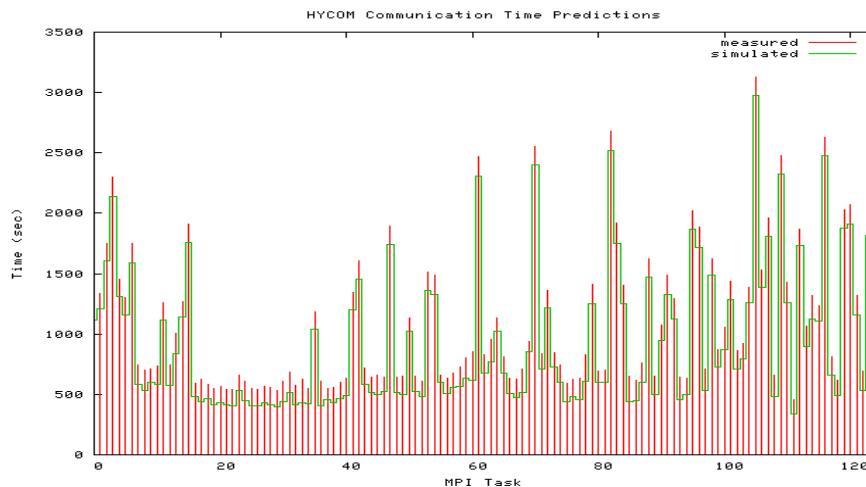


Figure 10 Measured and simulated times communication for all tasks for HYCOM with 124 processors.

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 11 (a) presents the measured values for the percentages of time spent in each MPI routine over the total communication time whereas Figure 11 (b) presents the percentages for the PSINS simulation. Figure 11 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 PSINS at a finer granularity.

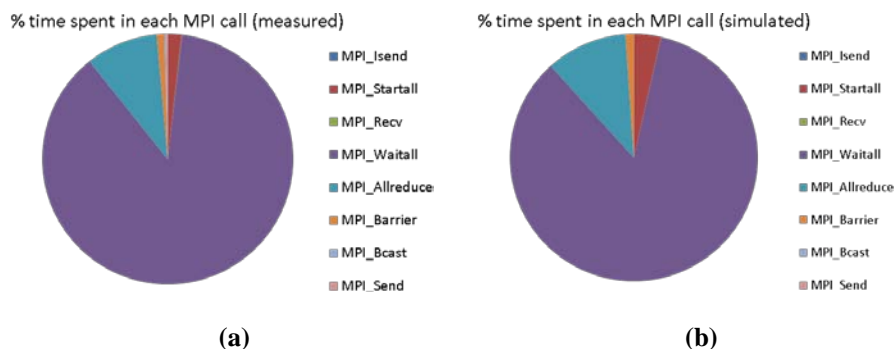


Figure 11 Percentage of communication time spent in MPI calls for HYCOM with 124 processors.

Table 1 presents the comparison between the total communication times measured during an actual run and times simulated by PSINS Simulator for two target HPC systems. We used the PMAc model for the simulations listed in this table. Table 1 shows that PSINS Simulator is able 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%. In AVUS with 64 processors on Sapphire, the communication time is only 7% of overall execution time and we believe the higher error percentage in Table 1 is introduced due to the synchronization of tasks (wait time) during communication due to the CPU ratio given to the simulator. Overall, Table 1 demonstrates

that PSINS is also effective in modeling and predicting the performance of applications for target HPC systems other than the base system on which the traces are collected.

	CPU Count	Jaws			Sapphire		
		Simul.	Meas.	% Error	Simul.	Meas.	% 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 (in seconds) spent in communication during simulation and during an actual run.

Similarly, Table 2 presents the comparison between the overall execution times measured during actual runs and runtime predictions by 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.

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%). Our investigation has shown that the over 15% error for Sapphire is mainly introduced by the error in relative speed calculation of its compute units. More importantly, 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)		%	Runtime (sec)		%
			Simu.	Meas.	Error	Simu.	Meas.	Error
HYCOM	STD	124	3,336	3,243	2.9	3,439	4,282	-19.7
		504	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).

5 Related Work

Predicting the performance of an existing HPC application through simulation is a challenging problem and there has been much work that addresses it. Early work on this was done in the Proteus simulator[20], an execution-driven simulator which meets many of the design goals that have been laid out for PSINS. Proteus is designed modularly so that it can be customized for the target system. They also use modularity with respect to the simulation of each component of the target system so that tradeoffs can be made between accuracy and efficiency by using a different implementation of a certain simulation component such as the memory subsystem or interconnect. Unfortunately Proteus introduces a slowdown of 2-35 for each process in the target application, which renders it unusable for the purpose of simulating applications that use hundreds or thousands of processes.

Later work, such as Parallel Proteus[20], LAPSE[21], MPI-SIM[22] and the Wisconsin Wind Tunnel[23] made attempts at improving 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, using a variety of techniques to reduce the synchronization overhead of parallel simulation. These tools 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 than they need to be for the purpose of predicting just the performance of the application on the target network. In contrast, our simulator framework divorces the tasks of network and serial computation simulation, allowing the network simulation portion of the framework to be very fast.

The Dimemas project [7] also uses this concept of largely divorcing network prediction from the prediction of serial computation portions of the code, which gives the benefit of allowing the user to use whatever technique they desire to determine the processor speedup ratio. Like PSINS, the user supplies Dimemas with a speedup ratio for a target system. This is the ratio of the speed of the target processor to the speed of the processor on which the trace was gathered for the serial computation portions of the application. 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 uses a fixed communication model that cannot be modified or examined by the user since it is not open source, hence it is not subject to the benefits gained by the modular and extensible design of PSINS. Dimemas also stores their MPI event traces as an ASCII text file instead of storing the trace as binary file, which is an inefficient use of space for storing the file and time for performing I/O on the file.

6 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 and application scaling. 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 for 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 for modeling large scale HPC applications.

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.

7 References

1. A. Snaveley, L. Carrington, N. Wolter, J. Labarta, R. Badia and A. Purkayastha. "A Framework for Performance Modeling and Prediction," Proceedings of the ACM/IEEE Conference on Supercomputing, 2002.
2. J. Michalakes, J. Hacker, R. Loft, M. McCracken, A. Snaveley, N. Wright, T. Spelce, B. Gorda and R. Walkup. "WRF Nature Run," Proceedings of the ACM/IEEE Conference on 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, Proceedings of the ACM/IEEE Conference on Supercomputing, 2007.
4. L. Carrington, D. Komatitsch, M. Tikir, M. Laurenzano, A. Snaveley, D. Michea, J. Tromp and N. Le Goff. "High-frequency Simulations of Global Seismic Wave Propagation Using SPECFEM3D_GLOBE on 62K Cores," Proceedings of the ACM/IEEE Conference on Supercomputing, 2008.
5. P. Ratn, F. Mueller, B. de Supinski and M. Schulz. "Preserving Time in Large-scale Communication Traces," Proceedings of the ACM/IEEE Conference on 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 7th European PVM/MPI 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 Snively. "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. Integrated Performance Monitoring, <http://ipm-hpc.sourceforge.net/>, 2008.
14. P. Mucci, S. Browne, C. Deane and G. Ho. "PAPI: A Portable Interface to Hardware Performance Counters," *Proceedings of the Department of Defense HPCMP Users Group Conference*, 1999.
15. M. Tikir, L. Carrington, E. Strohmaier and A. Snively. "A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations," *Proceedings of the ACM/IEEE International Conference on Supercomputing*, 2007.
16. 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.
17. R. Bleck. "An Oceanic General Circulation Model Framed in Hybrid Isopycnic–Cartesian Coordinates," *Ocean Modelling*, 4, 55–88, 2002.
18. 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," *Institute of Aeronautics and Astronautics Paper 99-3730*, 1999.
19. Department of Defense, High Performance Computing Modernization Program. "Technology Insertion," <http://www.hpcmo.hpc.mil/Htdocs/TI/>, 2009.
20. E. Brewer, C. Dellarocas, A. Colbrook and W. Wehl. "Proteus: A High-Performance Parallel Architecture Simulator," *MIT Technical Report MIT/LCS/TR-516*, 1991.
21. 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.
22. S. Prakash and R. Bagrodia. "MPI-SIM: Using Parallel Simulation to Evaluate MPI Programs," *Proceedings of the Winter Simulation Conference*, 1998.
23. 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.