A Performance Prediction Framework for Scientific Applications

Laura Carrington, Allan Snavely, Xiaofeng Gao, and Nicole Wolter

San Diego Supercomputer Center, University of California, USA
{lnett, allans,xgao,wolter@sdsc.edu}

Abstract. This work presents a performance modeling framework, developed by the Performance Modeling and Characterization (PMaC) Lab at the San Diego Supercomputer Center, that is faster than traditional cycle-accurate simulation, more sophisticated than performance estimation based on system peak-performance metrics, and is shown to be effective on the LINPACK benchmark and a synthetic version of an ocean modeling application (NLOM). The LINPACK benchmark is further used to investigate methods to reduce the time required to make accurate performance predictions with the framework. These methods are applied to the predictions of the synthetic NLOM application.

1 Introduction

In this work, we report our ongoing progress to develop a general performance prediction framework to predict and explain the performance of scientific applications on current and future HPC platforms. The framework is not designed for a specific application or architecture but is designed to work for an arbitrary application on an arbitrary machine. In previous work we introduced our convolution method [4-6] that is a computational mapping of an application's signature (a representation of an applications fundamental operations) onto a machine profile (a characterization of a machine's ability to perform fundamental operations) to arrive at a performance prediction. We introduced Memory Access Patter Signature (MAPS), a benchmark probe tool for collecting machine profiles. We introduced MetaSim Tracer, a tool for gathering application signatures. See www.sdsc.edu/PMaC for previous papers and access to these tools. Finally, we showed that the framework could model and improve understanding of the performance of small parallel scientific kernels and applications on several different HPC architectures. Here we provide an update on the ongoing work to make full applications modeling tractable via the convolution method.

2 The Convolution Method

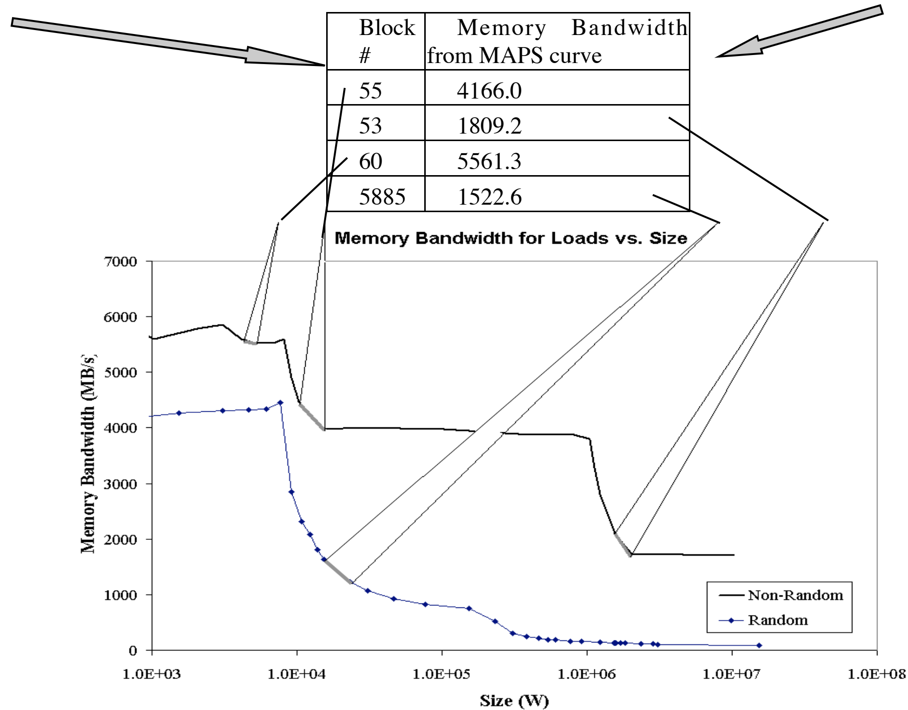
To create a model for the performance of a parallel application's serial sections (between communication events) we map the memory trace component of an application signature to the corresponding information in a machine profile in order to

model this (presumed to be dominant) factor in performance. Next, we map a communication trace to its corresponding information in a machine profile to get a model of the communication events. Then we combine the single-processor model (possibly supplemented with model terms for floating-point work, and other kinds of work), along with the communication model, to arrive at a performance model of a full parallel application.

Details of the convolution method for serial sections involve mapping each basic-block's expected dataset location onto the benchmark-probe curves from MAPS. The process is illustrated in Table 1 and Figure 1. Table 1 is the output of MetaSim Tracer on a Portable, Extensible Toolkit for Scientific Computation (PETSc) [15] application convolved with machine memory-hierarchy parameters from Pittsburgh Supercomputer Center's TCSini Compaq machine.

Table 1. Application Signature Example via MetaSim Tracer.

Block #	% Mem. Ref.	Ratio Rand	L1 hit Rate	L2 hit Rate	Data Set Location	Memory Bandwidth
55	0.9198	0.07	93.47	93.48	L1 Cache	4166.0
53	0.0271	0.00	90.33	90.39	Main Memory	1809.2
60	0.0232	0.00	94.81	99.89	L2 Cache	5561.3
5885	0.0125	0.20	77.32	90.00	L1/L2 Cache	1522.6



Block #	Memory Bandwidth from MAPS curve
55	4166.0
53	1809.2
60	5561.3
5885	1522.6

The convolution represented between Figure 1 and Table 1 is carried out automatically by the MetaSim Convolver [16] and can be written as:

$$1)\quad \text{Memory Execution Time} = \sum_{i=1}^{n} \left(\text{MemOps } BB_i \,/\, \text{MemRate } BB_i \right)$$

Equation 1 predicts that the Memory Execution Time for an application between communication events is the sum, over all the basic-blocks in the application, of the expected time required to carry out the loads and stores in each basic-block. The expected execution time depends on the rates at which the machine can carry out loads and stores based on instruction type, access pattern, and where the references fall in the memory hierarchy. MemOps BB_i is the total number of dynamic memory references in basic block i. MemRate BB_i is the rate at which the machine can sustain these operations. MemOps BB_i subcomponents (random loads to main memory, stride 1 accesses to L2 cache etc.) are determined by the convolver. MemRate BB_i has subcomponent rates taken from the MAPS curves. This simple example illustrates only predictions involving memory operations, but more complex convolutions can deal with other kinds of operations, and the interactions and overlap between those operations. If an application is heavily memory bound, Memory Execution Time may anyway be a large percentage of total execution time. Otherwise, additional model terms are added to the convolution to account for cycles spent doing non-overlapped floating-point work, branches, file I/O, communications etc. Once the convolution for the single-processor model is complete it is used in conjunction with the communication model for the performance prediction of parallel applications.

Similar to the single-processor convolutions, for the communication model we map communication event trace to their corresponding information in the machine profile (speeds of communications operations) to model the communication events. For the communication event trace of an application we use MPIDtrace, which contains the sequence of CPU demands and communication requests launched by the CPU during an application's execution. For the convolution step we use the network simulator, Dimemas [14], which using this trace can model the communication requests of the application on an arbitrary machine provided a user inputted parameterized network (machine profile) of that machine. So a modeler inputs the performance parameters of an arbitrary machine they desire to model along with the communications trace. Dimemas will then calculate the expected performance of the application's communication events on that machine.

Dimemas not only models the communication part of the application but can also model the CPU demands between communication events of the application. The MPIDtrace of the application contain CPU demand information about the application specified in terms of the CPU time consumed on the machine where the trace was obtained. Dimemas uses a parameter (CPU ratio) to scale the CPU bursts of the trace for predicting the performance of a machine other than the machine where the trace was obtained. The ratio being the ratio of processor speeds between a target machine to predict and the machine where the MPIDtrace was collected. A naïve way of calculating CPU ratio might be to use the ratio of clock speed or the ratio of peak floating-point issues between the two processors. Our framework improves upon this

idea by calculating the ratio from the single-processor models we develop using MetaSim Tracer and MAPS data. This new ratio will be the expected single-processor performance of the target machine being predicted (taking into account especially the performance of its memory hierarchy) to the single-processor performance of the machine where the MPIDtrace was collected. So given the single-processor model of an application along with the application signature and machine profile parts for the communication model, Dimemas can predict the performance of a parallel application.

Using detailed single-processor models with a special emphasis on the memory hierarchy and the network simulator Dimemas, the framework was shown in [4-6] to model the performance of NPB kernels, a PETSc kernel, and PETSc small applications with an error range of 1% to 16%. These performance predictions were consistent across a range of compute platforms (SDSC Power3 system Blue Horizon, PSC Compaq Lemiuex, a Cray T3E-600, TACC Power4 Longhorn), and across a range of processors from 2 to 128 for both weak and strong scaling. Some of the predictions used more complicated convolutions than Equation 1 and took into account (in addition to memory work) floating-point work and instruction level parallelism with overlap of memory and floating-point work.

The collection of an application signature via the MetaSim Tracer is needed only once per application-prediction series, allowing the convolution to run multiple times with the same application signature. Unfortunately MetaSim Tracer can require orders of magnitude slowdown of the instrumented code.

3 Speeding up Tracing

In a quest to supply modelers with a general performance prediction framework that could work in a relatively short amount of time even on long-running applications, several methods of reducing trace time were investigated. The LINPACK Benchmark and the synthetic version of the Navy Land Ocean Model (synNLOM) [8] application were used in the investigation of trace time reduction and the results of predictions with these new methods is discussed in section 4. The application synNLOM was of particular interest because it exhibited many traits of a scientific application plus, the run time of the application took over an hour, making the reduction of trace time critical in its prediction.

The idea of trace sampling is to turn the trace collection of an application on and off at certain intervals while running the application. MetaSim Tracer processes all of the memory addresses of an application on-the-fly. Sampling would allow the tracer to process only a percentage of the addresses in an attempt to reduce the trace time. In our implementation the user can specify the size of the interval and the number of sequential addresses to process per interval (sample size). If interval size is set for 1,000,000 and sample size is set for 10,000 then as the application runs, every 1,000,000 addresses the first 10,000 will be processed. This would result in a 10% sampling of the full trace. The idea is that as each basic-block is traversed and processed the cache-hit rates may not change significantly. So calculating the hit rates based on traversing and processing the basic-block only 10% of the time may yield

similar hit rates to those based on traversing the basic-block 100% of the time. It is the hit rate values that are used in the convolution to determine the bandwidth for that basic-block. This bandwidth is then used in the convolution Equation 1 to determine the Memory Execution Time.

Another issue with trace sampling is that the total number of memory references estimated for a basic-block may differ depending on the percent sampled, this is the MemOps BB value from Equation 1. This can be remedied by collecting two traces. The first, uses no sampling but only counts (no processing of) memory references, instructions, and floating-point operations, thus it collects the correct value for MemOps BB for each basic-block. The second is the detailed trace collection for memory accesses with sampling. The first trace, because it does no processing, only takes a minimal amount of time, but is significant in that the data collected in this trace used in conjunction with the second trace ensures correct values for number of memory references and floating point operations for each basic-block. The accuracy of predictions using different sampling sizes and their respective trace times are discussed in section 4.

Another way to reduce the trace time is to put an upper limit on the number of times a basic-block is traversed and processed. This means that if the user sets an upper limit as for example 1,000 then after a basic-block is traversed more than 1,000 times, the tracer no longer processes that information. The reasoning is that each time a basic-block is traversed it behaves similarly so the hit rates for a basic-block averaged over 1,000,000 traverses is going to be similar to the hit rates for a basic-block averaged over 1,000 traverses. This basic-block trace limit along with sampling can be used together to reduce the total trace time. Results of predictions for a series of traces using different basic-block limits are discussed in section 4.

Instead of using sampling to reduce trace time, tracing only certain sections of the application is also an option. The idea is that, in a lot of applications, there are only a small number of basic-blocks that account for most of the wall-clock time in the application. The reduction in trace time comes from the fact that instead of tracing all 100,000 basic-block in the application you only trace 100. The trick is determining which basic-blocks to trace in order to capture most of the applications performance attributes. This method also requires two traces. The first uses only counts (no processing of) memory references, instructions, and floating-point operations for the entire application. Using this trace, one can determine those basic-blocks that are contributing to the majority of memory references of the application. In the second trace, only those basic-blocks determined from the first trace are traced and processed. This results in only a fraction of the total number of basic-blocks from the application being traced. Section 4 discusses the results of tracing different numbers of basic-blocks.

Another approach enabled of this tracing method is that the trace of an application can be collected in phases, where a phase represents a certain number of basic-blocks. For example, an application containing 100 basic-blocks could be traced in two runs. The first run would trace basic-blocks 1-50 and the second run would trace basic-blocks 51-100. The has the advantage of being able to reduce trace time of a phase in order to fit into the queuing limits of the machine used to collect the traces. Also, this allows each phase to be collected simultaneously (multiple jobs in the queue). So although this method does not effect the cumulative trace collection time, it makes

trace collection more parallel and flexible. Results of using this phase collection are discussed in section 4.

The methods of trace time reduction using sampling, basic-block trace limits and tracing only a fraction of the basic-blocks can be combined to further reduce tracing time and enable a more flexible trace collection process. The results of both prediction accuracy and the trace time reductions of all methods, including combinations of them are discussed next in section 4.

4 Results and Discussion

The framework was first used to confirm the accuracy of using it in the prediction of the performance of the LINPACK benchmark. The LINPACK benchmark was predicted on four different machines at different number of processors. The size of the problem solved by the code was scaled with the number of processors (i.e. weak scaling). The results of these predictions are shown in Tables 2-5. Since Blue Horizon was used to collect the MPIDtraces, eliminating the single-processor ratio, predictions are for that machine are viewed as an accuracy check of the network simulator rather than validation of the entire framework. Tables 2-5 have the real run time for each machine-processor pair, the predicted run time by the framework, and the relative error. The results show that for varying numbers of processors and different machines, the framework is accurate in its predictions.

Tables 2-5. Real and predicted time for LINPACK benchmark.

Number of Processors	**PSC Lemieux**[1]		
	Real Time (s)	Predicted Time (s)	% Error[5]
4	9.3	8.9	4.3
16	21.9	20.9	4.6
64	23.9	22.8	4.6
256	22.2	21.4	3.6

1 Pittsburgh Supercomputer Center Compaq Alpha-server ES45 with 1-GHz processors and Quadrics interconnect.

Number of Processors	**TACC Longhorn**[2]		
	Real Time (s)	Predicted Time (s)	% Error[5]
4	8.5	8.8	-3.5
16	21.1	20.4	3.3
64	25.4	22.8	10.2
256	NA	21.1	NA

2 Texas Advanced Computing Center IBM Regatta-HPC with an IBM high-speed switch (SP Switch2).

Number of Processors	NERSC Seaborg[3]		
	Real Time (s)	Predicted Time (s)	% Error[5]
4	18.8	17.7	5.9
16	41.7	39.0	6.5
64	45.4	44.9	1.1
256	51.7	45.2	12.6

3 National Energy Research Scientific Computing Center IBM SP RS/6000

Number of Processors	SDSC Blue Horizon[4]		
	Real Time (s)	Predicted Time (s)	% Error[5]
4	18.8	19.2	-2.1
16	41.4	40.9	1.2
64	45.4	43.1	5.1
256	43.0	41.3	4.0

4 San Diego Supercomputer Center IBM SP RS/6000
5 Percent relative error: (Real Time – Predicted Time)/(Real Time) *100.

As Tables 2-5 confirm, the framework is an accurate predictor of the LINPACK benchmark, therefore this code was then used to investigate the viability of using the trace time reduction methods discussed in section 2.4. The LINPACK Benchmark run on 64 processors was used to compare the accuracy of using each of the different trace time reduction methods as well as their overall trace time reduction. Then the methods were combined to predict the synNLOM application run on 28 processors (the usual size for the Navy's production runs). Trace time reduction method was essential to predict the synNLOM application due to the long runtime of the application and the queuing limits on the machine used to collect the traces.

The first investigation was into the use of different sampling percentages in tracing. Sampling percentages of 100% (no sampling), 10%, 5%, and 1% are shown in tables 6 and 7. Table 6 shows the trace time slowdown for each sampling size on the application. Remember that a cycle-accurate simulation typically results in a 1,000,000 times slow down so tracing, while slow, is not *that* bad. Table 6 shows that initially the full trace slowed down the application by a factor of 859 times, whereas the sampling trace can reduce this time by a factor of 8. Table 7 shows the results of predictions using the different sampling size traces. It illustrates that sampling not only reduces the trace collection time to one ten thousandths of that of a typical cycle-accurate simulation, but it is able to predict the performance of the application with only a maximum error of 8.4%.

Table 6. MetaSim Trace collection time comparison using trace sampling.

Sampling size[1]	Slowdown factor[2]
NO Sampling	859
10%	152
5%	141
1%	132

1 The sampling size used for the trace, where 100% means no sampling
2 The slowdown factor is the number of times longer the trace takes to collect than the application

Table 7. Predictions for PSC's Lemieux results for different sampling sizes.

Sampling size	Predicted Time (s)	Real Time (s)	% Error[3]
NO Sampling	22.8	23.9	4.6
10%	24.8	23.9	-3.8
5%	25.0	23.9	-4.6
1%	25.9	23.9	-8.4

3 The percent error is calculated: (Real time – Predicted time)/(Real time)*100

The second investigation was into reducing trace time by using the basic-block trace limit discussed in section 3 with no sampling. Three different limits were compared to a case with no limit. Table 8 displays the trace slowdown factors for each case. This table shows that there is a trace time-reduction benefit depending on the basic-block limit but that the accuracy of the prediction is sacrificed a bit for this method of trace time reduction.

Table 8. MetaSim Trace collection time using basic-block trace limits.

Basic-block limit[1]	**Slowdown factor**
NO Sampling/limit	859
10,000	134
1,000	130
100	130

1 The basic-block limit is the limit of the number of times a basic-block is traced.

Table 9. Predictions for PSC's Lemieux for different basic-block trace limits.

Basic-block limit	Predicted Time (s)	Real Time (s)	% Error[3]
All BB	22.8	23.9	4.6
10,000	28.7	23.9	-20.1
1,000	29.2	23.9	-22.2
100	29.5	23.9	-23.4

The third investigation was into the trace reduction time by tracing only a small number of basic-blocks as discussed in section 3. Three different basic-block numbers were compared both in reduction of trace time and accuracy of prediction. Table 10 displays the trace slowdown factors for each case. This table shows that this method does have trace reduction benefits and Table 11 confirms that the accuracy is still relatively good.

Table 10. MetaSim Trace collection time tracing only certain basic-blocks.

Num. Basic-block	**Slowdown factor**
All	859
20	154
10	151

Table 11. Predictions for PSC's Lemieux results for different basic-block groups.

Num. Basic-block	Predicted Time (s)	Real Time (s)	% Error[3]
All	22.8	23.9	4.6
20	26.8	23.9	-12.1
10	26.2	23.9	-9.6

In the prediction of synNLOM a combination of all the trace reduction methods were used. This application, run on 28 processors of SDSC's Blue Horizon takes over 1 hour to complete. To simulate 1 hour of an application on a cycle-accurate simulator could take around 114 years of CPU time. Using no trace reduction method, the collection of the MetaSim trace would take around 850 hours. To reduce this to a more manageable time the trace was collected using 1% sampling, collecting the top 100 basic-blocks of the application, a basic-block limit of 200, and the trace collected in 10 phases. This allowed each phase to be collected in 2-6 hours, easily fitting into the queuing limits of most HPC machines. This also allowed 10 different jobs to be run (simultaneously) on the machine ranging from 2-6 hours, quite a reduction from the years required for cycle-accurate simulation. Table 12 shows the results of predicting the synNLOM on PSC's Lemieux, NERSC's Seaborg, TACC's Longhorn, and SDSC's Blue Horizon. The percent error of the prediction, >9%, shows that not only is the framework accurate in its prediction, but with trace reduction methods it is capable of predicting the entire scientific application's run for long periods of time and doing these predictions in a reasonable amount of time, something not feasible by cycle-accurate simulation.

Table 12. Prediction of synNLOM for different machines.

Machine	Real Time (s)	Predicted Time (s)	% Error
PSC's Lemieux	1818	1816	0.1
SDSC's Blue Horizon	4462	4594	-3.0
NERSC's Seaborg	4375	4756	-8.7
TACC's Longhorn	1944	1872	3.7

The results of performance predictions shown in Tables 5 through 12 illustrate the accuracy of using the performance prediction framework to predict scientific applications. Such predictions can be completed nearly 10,000 times faster than using a cycle-accurate simulator. In addition, the framework is flexible enough to be applied to many different architectures and applications.

This work was sponsored in part by the Department of Energy Office of Science through SciDAC award "High-End Computer System Performance: Science and Engineering". This work was sponsored in part by a grant from the Department of Defense High Performance Computing Modernization Program (HPCMP) and the National Security Agency. This research was supported in part by NSF cooperative agreement ACI-9619020 through computing resources provided by the National Partnership for Advanced Computational Infrastructure at the San Diego Supercomputer Center. Computer time was provided by the Pittsburgh

Supercomputer Center, the Texas Advanced Computing Center, and the National Energy Research Scientific Computing Center.

References:

1. D. J. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, M. Gittings, "Predictive Performance and Scalability Modeling of a Large-Scale Application", *Supercomputing 2001*.
2. J. Lo, S. Egger, J. Emer, H. Levy, R. Stamm, and D. Tullsen, "Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading", *ACM Transactions on Computer Systems*, August 1997.
3. J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich, "FLASH vs. (Simulated) FLASH: Closing the Simulation Loop", *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 49-58, November 2000.
4. A. Snavely, N. Wolter, L. Carrington, R. Badia, J. Labarta, A. Purkasthaya, "A Framework to Enable Performance Modeling and Prediction", *Supercomputing 2002*.
5. L. Carrington, N. Wolter, and A. Snavely, "A Framework for Application Performance Prediction to Enable Scalability Understanding", Scaling to New Heights Workshop, Pittsburgh, May 2002
6. A. Snavely, N. Wolter, and L. Carrington, "Modeling Application Performance by Convolving Machine Signatures with Application Profiles", *IEEE 4th Annual Workshop on Workload Characterization*, Austin, Dec. 2, 2001.
7. LINPACK
8. See http://www7320.nrlssc.navy.mil/html/lsm-home.html
9. J. Simon, J.-M. Wierum, "Accurate performance prediction for massively parallel systems and its applications", proceedings, *Proceedings of European Conference on Parallel Processing EURO-PAR '96*, Lyon, France, v2, pages 675-688, Aug. 26-29, 1996.
10. See http://www.cepba.upc.es/tools_i.html
11. See http://www.sdsc.edu/PMaC/MAPS/
12. See http://www.cs.virginia.edu/stream/
13. See http://www.sdsc.edu/PMaC/Benchmark/
14. See http://www.cepba.upc.es/
15. See http://www-fp.msc.anl.gov/petsc/
16. See http://www.sdsc.edu/PMaC/MetaSim/