A Framework for Performance Modeling and Prediction

Allan Snavely, Laura Carrington, Nicole Wolter of The San Diego Supercomputer Center with
Jesus Labarta, Rosa Badia of The Technical University of Catalonia and
Avi Purkayastha of The Texas Advanced Computing Center

Abstract

Cycle-accurate simulation is far too slow for modeling the expected performance of full parallel applications on large HPC systems. And just running an application on a system and observing wallclock time tells you nothing about why the application performs as it does (and is anyway impossible on yet-to-be-built systems). Here we present a framework for performance modeling and prediction that is faster than cycle-accurate simulation, more informative than simple benchmarking, and is shown useful for performance investigations in several dimensions.

1 Introduction

The Performance Evaluation Research Center (PERC) is working to evolve practical frameworks for understanding the performance of HPC applications. The goal of PERC is to develop a science for understanding performance of scientific applications on high-end computer systems and develop engineering strategies for improving performance on these systems. PERC is a DoE SciDAC Center with several lab and university members and affiliates [1].

In this paper we present an instantiation of a PERC framework carried out by a team from The San Diego Supercomputer Center (www.sdsc.edu), The European Center for Parallelism of Barcelona (CEPBA) at the Technical University of Catalonia (UPC) (www.cepba.upc.es), and The Texas Advanced Computing Center (www.tacc.utexas.edu). This framework combines tools for gathering machine profiles and application signatures and provides automated convolutions. Machine profiles are measurements of the rates at which HPC platforms can carry out fundamental operations (for example, floating-point operations, memory accesses, message transfers). Application signatures are summaries of the operations to be carried out on behalf of an application to accomplish its computation. Convolution methods are techniques for mapping signatures to profiles in reasonable time complexity for predicting and understanding performance.

The framework is shown to be effective for practical problem solving in the domains of 1) meaningful machine comparison 2) performance evaluation of architectural upgrades 3) system tuning, and 4) applications scaling studies.

2 A Framework for Applications Performance Prediction and Understanding

The observed performance of a parallel application on a HPC machine is complicated; it is a function of (at least) algorithm, implementation, compiler, operating system, underlying processor architecture, and interconnect technology. Our approach is to disambiguate these many factors via principles of simplicity and abstraction. For example, we have a simplifying hypothesis that a parallel application's performance is often dominated by two major factors: 1) its single processor performance and 2) its use of the network. Clearly, there are other factors, but often processor and network performance are sufficient for accurate performance prediction. Therefore, our initial performance prediction framework consists of a single processor model combined with a network model.

Existing network simulators can do a good job of modeling an application's use of interconnect and capturing factors related to scalability [2]. In particular, rapid and accurate network performance estimates have been obtained with the simple L/B (latency/bandwidth) model for communication [3]. Thus, for a relatively complete model of an application's performance, we begin by understanding and modeling single-processor performance and then combine this information with a network simulator.

To model single-processor performance, we separate various performance factors by measuring each in isolation and then combine them for a model of overall performance. In our pursuit of rapid, useful and accurate performance modeling, we model only some of the features of modern, highly complex processors. For example, we have found that the performance of memory-bound codes (which are common in scientific applications) may be dominated by memory latencies that can largely obscure subtle performance effects pertaining to super scalar issues, out-of-order issues, speculations, and other advanced processors features. Therefore, we begin with simple models and few parameters, adding complexity only as needed to explain observed performance (Occam's razor [4]). Based on the idea that the per-processor performance of a memory-bound application is predominately a function of how it exercises the memory sub-system, our starting point for single-processor performance focuses on the local memory hierarchy. We then account for floating-point work and insert our models into a network simulator to model parallel applications at scale, completing the framework.
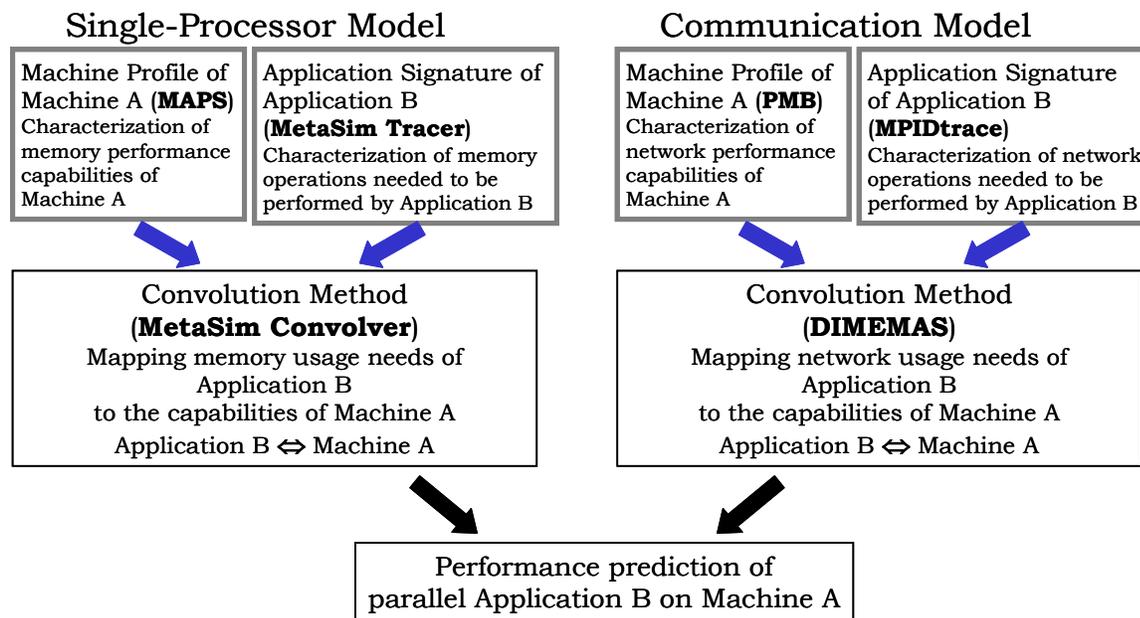
For the single-processor model we use a single-processor pseudo cycle-accurate simulator. In the communication model we use a network simulator. These are then combined in a framework for performance modeling that is faster than traditional cycle-accurate simulation, more sophisticated than "back-of-the-envelope" estimation, and is shown effective on a set of applications kernels run on several large-configuration HPC systems. The kernels are taken from the SciDAC ISIC application library PETSc, The Portable, Extensible Toolkit for Scientific Computation [5]. PETSc is chosen for sample applications because it has general capabilities for solving a variety of sparse-matrix problems common to many scientific problems. However, the techniques shown here apply to many other scientific applications as well. The method is based on the idea of convolving application signatures with machine profiles to yield performance predictions and, more importantly, insight into the factors affecting performance [6-7]. Our PERC framework is instantiated with:

- MAPS [8] and PMB [9] benchmark data for machine profiles
- MetaSim tracer [10] and MPIDtrace [11] for application signatures
- Dimemas [11] and MetaSim convolver to provide the automated convolving step

Other benchmarks, application tracers, and simulators could be installed in the framework to work in this coordinated way. We show how to gather and combine memory access-pattern trace information and MPI communications trace information from an application and then map the application, basic-block by basic-block, to its expected performance on a machine that has been characterized using MAPS and PMB benchmarks.

The components of the framework isolate the performance factors of the machine in Machine Profiles and those of an application in Application Signatures. For the single-processor model, we isolate only those performance factors that affect the performance of the processor and application between communication events. These are factors that focus on the local memory

hierarchy. In the communication model, we isolate only those performance factors that affect the performance of the network and an application's communication events. Figure 1 displays the various components of each model and how they are combined for the performance prediction of a parallel application.

Single-Processor Model   Communication Model

Machine Profile of Machine A (**MAPS**) Characterization of memory performance capabilities of Machine A	Application Signature of Application B (**MetaSim Tracer**) Characterization of memory operations needed to be performed by Application B

Machine Profile of Machine A (**PMB**) Characterization of network performance capabilities of Machine A	Application Signature of Application B (**MPIDtrace**) Characterization of network operations needed to be performed by Application B

Convolution Method
(**MetaSim Convolver**)
Mapping memory usage needs of
Application B
to the capabilities of Machine A
Application B ⇔ Machine A

Convolution Method
(**DIMEMAS**)
Mapping network usage needs of
Application B
to the capabilities of Machine A
Application B ⇔ Machine A

Performance prediction of
parallel Application B on Machine A

Figure 1: The Components of our PERC Framework

Figure 1 illustrates, the different components of the PERC framework and how they are used together to arrive at a performance prediction. The three main components are Machine Profiles, Application Signatures, and Convolution Methods. Next we describe each piece of the framework and then the role it plays in enabling application performance analysis, prediction, and insight.

2.1 Machine Profiles - The MAPS and PMB benchmarks

Machine profiles are tables of performance data gathered for existing machines via low-level benchmarks (sometimes called "probes") that measure simple performance attributes of machines. For machines that have yet to be built, machine profiles represent the engineer's expectations of the rates at which the machine will perform operations.

For this PERC framework we used two machine profiles in our performance prediction. The first is the machine profile used in the single-processor model of the framework that contains information about a machine's memory performance. The second is the profile used in the communication model, which contains information about a machine's interconnect performance.

To obtain a single-processor model's machine profile, the memory performance of a machine is collected with the MAPS (Machine Access Pattern Signature) benchmark probe. MAPS [8] is a benchmark derived from the STREAM benchmark [12] and is designed to measure platform specific bandwidths. These measurements include bandwidths of different levels of memory,

different size working sets, and access patterns. Practically speaking, applications performance is often bottlenecked by interaction with the memory hierarchy. Thus, knowing the rates at which the system can sustain load and stores, depending upon size and locality of the working set and memory access pattern, can be at least as important as knowing floating-point issue rates. MAPS gives detailed but neatly summarized information about the memory hierarchy. By way of example, Figure 2 shows a graph of MAPS data giving sustainable rates of memory loads depending on access pattern (stride 1 or random), and size of the working set (W) on a 667 MHz Alpha processor of the TCSini machine at Pittsburgh Supercomputing Center. The memory hierarchy (L1 and L2 cache) accounts for the stair-step shape of the curves. For every size and access pattern there is a corresponding L1 and L2 miss rate. MAPS is also capable of making the same measurements for the processors on a SMP node. For another example, Figure 3 shows the MAPS curves for 1 and 16 processors on 1 node of a Power 4 16-way node. This illustrates the performance hit in memory bandwidth when all processors are contending for the same memory bus on a SMP node.
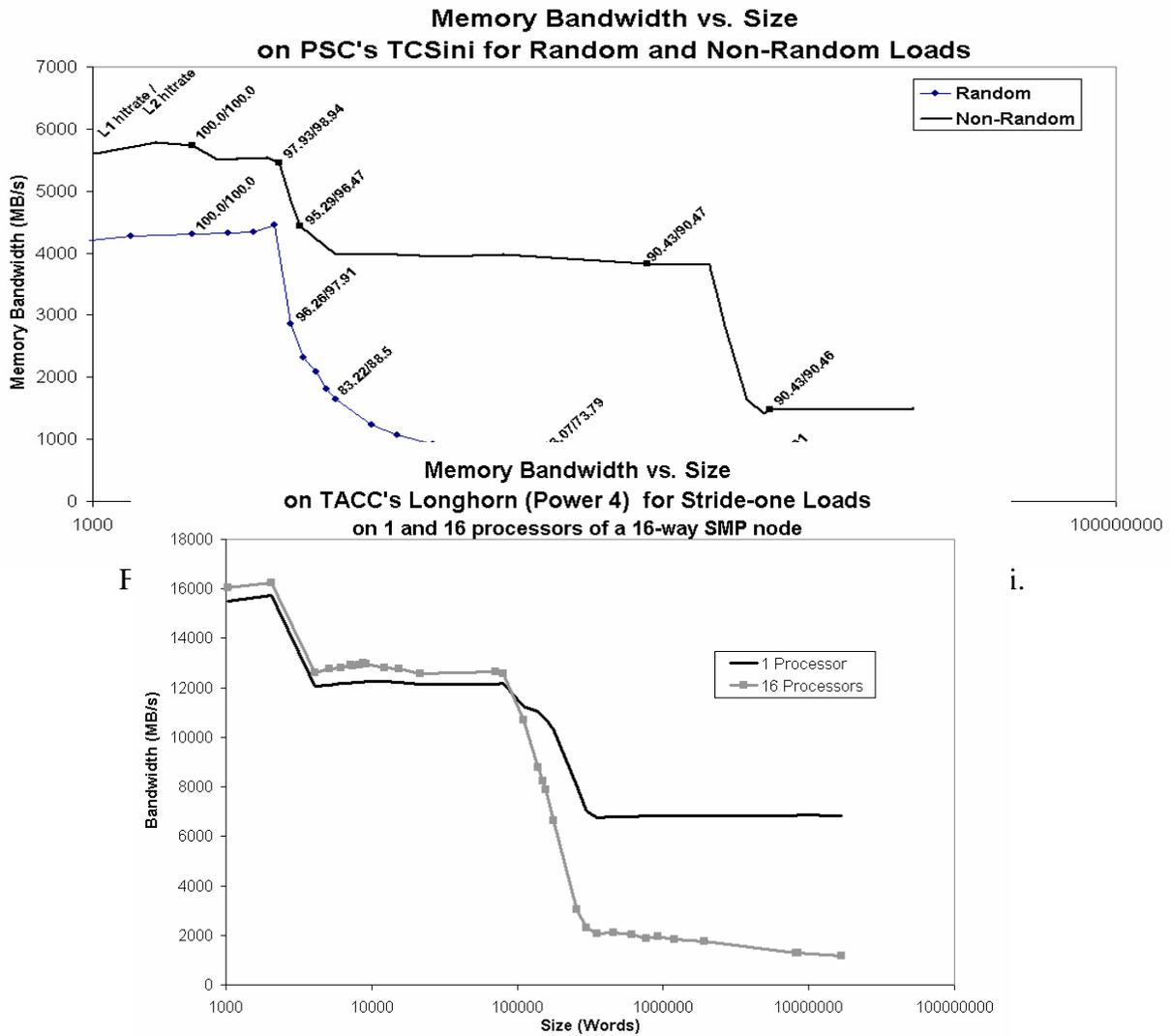

Figure 3. Per-processor MAPS curve of Power 4 SMP node for 1 and 16 processors.

For the communication model's machine profile, the interconnect performance of a machine is collected with PMB (Pallas MPI Benchmarks). The PMB benchmarks [9] provide a concise set of benchmarks for measuring performance of important MPI functions: point-to-point message passing, global data movement and computation routines, one-sided communications and file-I/O.

A machine profile combining MAPS and PMB data provides a relatively complete characterization of the ability of the machine to move data around from local or remote memory depending on access pattern and type of operation.

2.2 Application Signatures – MetaSim Tracer and MPIDtrace

An application signature is a summary of the operations required by an application to accomplish its computation. We have found that for some scientific codes, the majority of these operations are memory usage and communications. For these applications, a useful application signature must summarize patterns of memory usage and communications.

In order to determine an application's signature, we need tools that capture how an application exercises the local memory hierarchy and how it exercises the interconnect fabric. Capturing both the local memory hierarchy and interconnect data requires a two-part application signature. The first part is a trace of how each processor uses the memory sub-system. This memory trace is used to model the single-processor performance of an application between communication events. The second part of the application signature is an MPI trace, which gathers information on all communications occurring in the application. The MPI trace also gathers general information about the amount of CPU time spent between communications. The memory trace information then augments the CPU time in the MPI trace to give a reasonably complete Application Signature. While tools for tracing communications patterns are common, tools for gathering memory access patterns are less common and typically not platform independent. We use MPIDtrace developed by CEPBA [14] to gather the communication traces and our own MetaSim tracer to gather the memory traces.

The MetaSim tracer, developed at SDSC, is a prototype tool that generates detailed basic-block information about floating-point units and load/store unit usage for an application. The MetaSim tracer captures dynamic memory address information during an instrumented run of the (serial or parallel) code. The address stream is processed "on-the-fly" to determine memory access patterns (such as stride N or random). The MetaSim tracer is built upon the ATOM toolkit for accessing performance counters on Alpha processors (e.g. PSC's Lemieux). We are working to port the MetaSim tracer to Dyninst API [13], thus making it available on multiple processors and systems.

The MetaSim tracer accepts user input for machine memory parameters such as sizes and associativies of the different levels of cache, for a machine to be performance-predicted. The MetaSim tracer processes the address stream of an application with the user defined machine parameters to calculate the location of each address in the predicted machines memory sub-system. This information is gathered for each basic block. Table 1 shows general application information gathered from the MetaSim tracer without a user-defined machine. Table 2 shows the same MetaSim tracer run with a user-defined machine similar to that of the IBM Blue Horizon and Table 3 shows the same information with a user-defined machine similar to that of a Cray T3E. Note that the MetaSim tracer, running on an Alpha system, can model an arbitrary, user-defined system.

Table 1. General Application Signature information for NPB CG class B on 32

Basic Block #	Num. Inst. [1]	Num. Memory References [2]	% Total. Mem. Ref. [3]	Floating-Point Inst. [4]	%FP Inst. [5]	Random Ratio [6]	Ratio of FP ops/ Mem. ops [7]
373	2.06E+09	8.86E+08	0.22	8.15E+08	0.37	0.33	0.92
372	1.68E+09	8.57E+08	0.21	4.90E+08	0.22	0.37	0.57
371	1.30E+09	6.25E+08	0.15	3.57E+08	0.16	0.36	0.57
375	1.36E+09	4.96E+08	0.12	2.48E+08	0.11	0.35	0.50

[1]   Total number of instructions completed by that basic block
[2]   Total number of memory references (loads and stores) completed by that basic block
[3]   The percent of memory references of basic block to the total memory references by the application
[4]   Total number of floating-point instructions (add, multiply,…) completed by that basic block
[5]   The percent of floating-point instructions of basic block to the total floating-point instructions by the application
[6]   Ratio of random-stride loads to total loads for that basic block
[7]   Ratio of floating-point operations to memory operations for that basic block

Table 2. Application Signature for NPB CG class B on 32 CPUs with user supplied memory parameters for the IBM Blue Horizon.

Basic Block #	% Total. Mem. Ref.	Random Ratio	L1 Hit Rate [8]	L2 Hit Rate [9]	Data Set Location in Memory [10]
373	0.22	0.33	92.16	99.98	L1 Cache
372	0.21	0.37	90.14	99.07	L1/L2 Cache
371	0.15	0.36	88.93	98.67	L1/L2 Cache
375	0.12	0.35	93.02	99.99	L1 Cache

[8]   The calculated L1 cache hit rate based on memory addresses of the tracer and user supplied memory parameters for that basic block
[9]   The calculated L1 cache hit rate based on memory addresses of the tracer and user supplied memory parameters for that basic block
[10]   The calculated location of the data set for the basic block based on the cache hit rates

Table 3. Application Signature for NPB CG class B on 32 CPUs with user supplied memory parameters for the Cray T3E.

Basic Block #	% Total. Mem. Ref.	Random Ratio	L1 Hit Rate	L2 Hit Rate	Data Set Location in Memory
373	0.22	0.33	68.19	96.90	Main Memory
372	0.21	0.37	64.38	93.34	Main Memory
371	0.15	0.36	59.66	91.57	Main Memory
375	0.12	0.35	70.42	97.31	Main Memory

In Tables 2 and 3, the MetaSim tracer generated cache hit rates for each basic block based on the user-defined cache sizes and associativies. Limited T3E memory caused this application's working set to fall out of cache and into memory. The larger Blue Horizon cache however improved the application's performance by keeping the working set in cache.

The information shown in Tables 2 and 3 is used in conjunction with Machine Profiles to model an application's single-processor performance between communication events via a convolution method.

2.3 Convolution Methods – DIMEMAS and MetaSim Convolver

Our convolution method is the computational mapping of an application's signature (application A) onto a machine profile (machine B) to arrive at a performance prediction (performance of application A on machine B). We first map the memory trace component of the application signature to the corresponding information in the machine profile in order to model single-processor performance of an application between communication events. Next, we map the MPI trace component to its corresponding information in the machine profile to get a communication model. Then we take the single-processor performance model, along with the communication model, to arrive at a complete performance model for the application.

To model the single processor performance of basic blocks in an application between communication events, we map each basic block's expected location in memory (determined from the MetaSim tracer) onto the benchmark-probe curves from MAPS. The process for mapping is illustrated by Table 4 and Figure 4. Table 4 is the product of the MetaSim tracer on a PETSc application with the user supplied machine parameters of the PSC's TCSini machine. This table is similar to Tables 2 and 3, with the addition of the memory and weighted bandwidth information. The new bandwidth information is generated from the convolution of the MetaSim tracer information with the MAPS data. Convolutions such as these are generated automatically by the MetaSim Convolver.

Table 4. Application Signature example via MetaSim Tracer.

Block #	Procedure Name	% Mem. Ref.	Ratio Random	L1 hit Rate	L2 hit Rate	Data Set Location in Memory	Memory Bandwidth	Weighted Bandwidth
180155	dgemv_n	0.9198	0.07	93.47	93.48	L1 Cache	**4166.0**	3831.7
180153	dgemv_n	0.0271	0.00	90.33	90.39	Main Memory	**1809.2**	49.1
180160	dgemv_n	0.0232	0.00	94.81	99.89	L2 Cache	**5561.3**	129.3
5885	MatSetValues	0.0125	0.20	77.32	90.00	L1/L2 Cache	**1522.6**	19.0
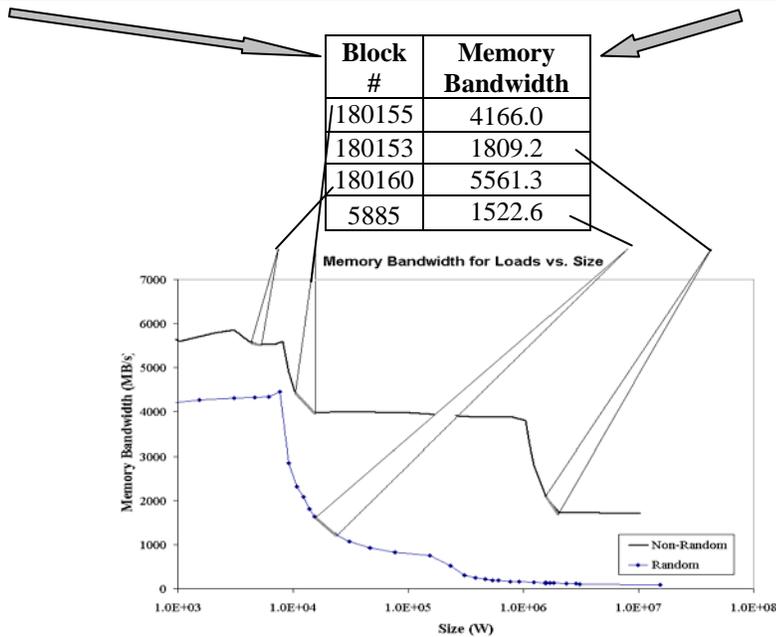
Figure 4. MAPS curves for PSC's TCSini for random and non-

7

Convolutions can be arbitrarily complex depending upon how many features of the application and the machine are being accounted for. The simple convolution represented between Figure 4 and Table 4 can we written

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

Equation 1 predicts that the Memory Execution Time for an application 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 MetaSim convolver. MemRate BB_i has subcomponent rates taken from the MAPS curves. This simple example shows only predictions involving memory operations but a full convolution can deal with other kinds of operations and the interactions and overlap between the operations. If an application is heavily memory bound, Memory Execution Time may be a large percentage of total execution time. Otherwise, additional model terms are added to account for cycles spent doing non-overlapped floating-point work, branches, file I/O, communications etc. Once these convolutions are complete the results are used by Dimemas (the network simulator) to predict the overall performance of an application.

Dimemas consumes MPIDtrace files to model the performance of an application's communication pattern on an arbitrary (user parameterized or PMB measured) network. MPIDtrace obtains the sequence of CPU demands and communication requests launched by the processor during an application's execution. The CPU demands from MPIDtrace are specified in terms of CPU time consumed in the machine where the trace was obtained. Dimemas uses a parameter (CPU ratio) to scale these CPU bursts for a machine under simulation. A naïve way of obtaining CPU ratio might be (for example) to use the ratio of clock speed or the ratio of peak floating-point issues between the processor where the trace was obtained and the processor of the machine under simulation. We improve upon this idea by focusing on the relative speeds of the memory subsystem in addition to speeds of the floating-point units; our CPU ratio is calculated from the single-processor model using MetaSim tracer and MAPS data (Equation 1) for the intervals that elapse on-processor between communications events. Thus, once an application has been characterized by memory access patterns (and possibly other operations like floating-point) by MetaSim tracer, and by communications patterns by MPIDtrace, we have a flexible framework for varying characteristics of memory subsystem and network to investigate performance via simulation.

3 PERC Framework Applied to Comparing HPC Platforms

One application of the framework is to explain the observed performance of applications on existing HPC platforms. If the performance of an application is mostly explained by its memory access and communications patterns mapped to benchmarked memory subsystem and network speeds of machines then we know several very useful things: 1) the relative performance of machines on this application is due to these attributes and we can quantify how much of the performance differences are due to which memory and/or network attributes, 2) the application's

8

expected performance by this simple model matches reality implying the implementation and tuning of the architecture is approximately correct, 3) the only way to improve the performance of the application is to change its underlying algorithm, tune the application, or upgrade the target machine in these dimensions. We neglect issues such as O/S interference, influence of other jobs, network daemons, and other possible secondary causes of poor performance because they are manifestly insignificant factors in wall-clock time.

We modeled example applications from the PETSc library on Blue Horizon—the Teraflops IBM system at SDSC, TCSini—the prototype Compaq/Quadrics system at Pittsburgh Supercomputing Center (PSC), Lemieux—the production Compaq/Quadrics system at PSC, and a node of the IBM Power 4 based system Longhorn at Texas Advanced Computing Center (TACC). We modeled and predicted performance using a convolution method that consumed MAPS and PMB benchmark data for the machine profiles and MetaSim tracer and MPIDtrace data for the application signatures. The convolution was carried out by the MetaSim convolver for on-processor work and by Dimemas for communications work. We modeled weak scaling (where the problem size was doubled for twice the number of processors) and strong scaling (where the problem size was held fixed). We used a slightly more complicated convolution than Equation 1 above for a rough-estimate of the time required to execute floating-point work:

$$(2) \qquad \text{Serial Execution Time} = \sum_{i=1}^{n} \left(\left(\text{FloatOps BB}_i / \text{FloatRate} \right) + \left(\text{MemOps BB}_i / \text{MemRate BB}_i \right) \right)$$

Where FloatOps BB_i is the number of dynamic floating-point operations in basic block i and FloatRate is the peak floating-point issue rate of the processor according to the manufacturer. This is a crude estimate of the time to execute floating-point instructions excluding, as it does, issues of dependency and overlap of memory and floating-point work. The question is, "how well can one predict performance with such a simple statistical model?"

Next we report the difference between predicted performance and actual performance of various applications. When the prediction is slower than actual performance a negative error is reported.

Table 5 gives representative results for our framework applied to a kernel from PETSc called Matrix.F that does a matrix-vector multiply. We report prediction and real runtime results for four machines: Blue Horizon (BH), Lemieux, TCSini, and a node of a Power4 Regatta system (Longhorn). The network between nodes on Longhorn was still under construction at the time of these experiments thus limiting our runs to a single node. We ran on various numbers of processors and various problems sizes. Table 5 details results for weak scaling for different number of processors on a size of problem (MM) that falls mostly out of cache on these systems. Error % is defined as (real runtime – predicted runtime) / (real runtime * 100).

Table 5: Real and Predicted Runtimes of Matrix.F with weak scaling.

Matrixf.F size MM weak scaling predictions for Blue Horizon			
CPU	Real time (s)	Prediction (s)	% Error
2	31.78	31.82	0.13
4	29.07	31.27	7.57
8	36.13	33.72	6.67
64	44.91	43.91	2.23
96	48.87	47.15	3.52
128	52.88	52.46	0.79

Matrixf.F size MM weak scaling predictions for TCSini			
CPU	Real time (s)	Prediction (s)	% Error
2	26.71	27.40	-2.58
4	27.63	26.54	3.94
8	27.97	28.65	-2.43
64	40.15	38.56	3.97
96	43.77	38.82	11.31
128	49.78	44.37	10.86

Matrixf.F size MM weak scaling predictions for Lemieux			
CPU	Real time (s)	Prediction (s)	% Error
2	19.79	22.63	14.33
4	20.36	21.07	3.47
8	20.93	23.66	13.01
64	30.54	31.58	3.38
96	31.84	32.93	3.42
128	34.58	36.81	6.44

Matrixf.F size MM weak scaling predictions for Longhorn			
CPU	Real time (s)	Prediction (s)	% Error
2	14.95	14.16	-7.03
4	14.45	11.27	11.23
8	17.01	14.98	11.10

Real time is shown as the average of several runs. Blue Horizon results verify the network simulator and are modeled with a gratifying level of accuracy that is actually within the observed variability of runtimes of that machine; however since the MPIDtraces were actually taken on BH a fairer test of the power of the method is to examine what it predicts for machines different from where the trace was taken. Predictions across Lemieux, TCSini, and Longhorn are on average a little less than 7% error, with only 15% maximum error for the runs. It is reasonable to ask whether such an error rate is "good" or "bad". In this case we are pleased that a simple model that accounts only for predicted interactions between the application and local-memory, floating-point units, and interconnect can explain at least 85% of observed performance. This level of accuracy is sufficient for answering useful questions. For example, we predicted TCSini would be about 1.17 times faster than BH on this problem due to its faster processors and interconnect. In fact it was about 1.14 times faster. We predicted Lemieux would be about 1.43 times faster, in fact it averaged 1.53 times faster. We predicted the PW4 system would be 2.23 times faster and in fact it is about 2.08 times faster. It is fair to point out that Lemieux and Longhorn have just recently come on line and their performance may improve from system tuning (see section 6). Using this framework we ranked machines correctly for this application and, more usefully, explained those performance factors affecting this difference.

Table 6 contains similar results for performance predictions of the Matrix.F kernel using strong scaling (keeping problem size fixed). For the small CPU runs (2-8) a different problem size was used than for the large CPU (64-128) runs. With these problem sizes we can see the effects of different size caches on the machines. For instance, on the Power 4 system, the scaling from 2 to 4 CPUs is super-linear resulting from the problem size per-processor decreasing as the number of processors increases. The prediction framework is able to capture subtle effects of the memory hierarchy as the local problem size moves from main memory into cache as the number of processors increase.

Table 6: Real and Predicted Runtimes of Matrix.F with strong scaling.

CPU	Real time (s)	Prediction (s)	% Error
Matrixf.F size MM strong scaling predictions for Blue Horizon			
2	99.19	99.53	0.34
4	57.44	55.04	4.18
8	35.70	35.41	0.81
64	114.96	113.60	1.18
96	64.57	64.74	0.26
128	58.50	54.63	4.91

CPU	Real time (s)	Prediction (s)	% Error
Matrixf.F size MM strong scaling predictions for TCSini			
2	103.27	111.88	-8.34
4	53.15	44.84	15.63
8	27.40	29.44	-7.46
64	96.92	90.93	6.18
96	58.97	52.21	11.45
128	56.69	47.01	17.07

CPU	Real time (s)	Prediction (s)	% Error
Matrixf.F size MM strong scaling predictions for Lemieux			
2	76.36	78.60	-2.92
4	39.91	31.83	20.24
8	20.09	20.91	-4.10
64	66.70	56.81	14.82
96	43.42	32.78	24.51
128	39.22	29.52	24.72

CPU	Real time (s)	Prediction (s)	% Error
Matrixf.F size MM strong scaling predictions for Longhorn			
2	46.7	47.17	-1.01
4	21.5	19.34	10.05
8	11.7	13.86	-18.46

The results of the prediction for the Matrix.F kernel showed relatively good results; accuracy for strong scaling is somewhat reduced. We think modeling strong scaling is a generally harder problem than modeling weak scaling.

We also made predictions with a mini-application that more closely approaches the complexity of a real application. This application is built on top of PETSc and comes from TOPS [15] and uses a nonlinear solver in a 2D driven cavity code with a velocity-vorticity formulation and a finite difference discretization on a structured grid. Table 7 shows the results of these predictions for the same four machines using strong scaling. These predictions had an average 7% error (18% maximum) showing we can predict reasonably well on more complicated codes.

Table 7: Real and Predicted Runtimes of EX19 with strong scaling.

CPU	Real time (s)	Prediction (s)	% Error
EX19 size MM strong scaling predictions for Blue Horizon			
2	66.54	66.53	0.03
4	46.44	46.64	0.55
8	32.40	33.16	2.34

CPU	Real time (s)	Prediction (s)	% Error
EX19 size MM-strong scaling predictions for TCSini			
2	45.00	50.10	-11.34
4	35.93	35.12	2.28
8	32.58	28.82	11.55

CPU	Real time (s)	Prediction (s)	% Error
EX19 size MM strong scaling predictions for Lemieux			
2	30.75	32.05	-4.23
4	25.18	22.51	10.61
8	20.83	18.51	11.16

CPU	Real time (s)	Prediction (s)	% Error
EX19 size MM strong scaling predictions for Longhorn			
2	23.83	24.56	-3.07
4	18.90	16.78	11.22
8	16.19	13.24	18.22

We explored different convolutions from Equation 2 above, trying predictions based on memory operations alone (assuming floating-point operations are "in the noise" for performance). We tried a flavor of Equation 2 that uses the maximum value instead of adding the two for the combining operation between floating-point and memory work (this would say that work in these two categories can overlap on a modern super-scalar processor). Neither was as accurate as Equation 2 for these problems although the maximum value convolution shows some promise. Table 8 shows the results of 3 predictions of the Matrix.F kernel on TCSini with weak scaling. The first prediction is using the convolution involving just memory operations. The second prediction is a convolution based on Equation 2 and the third convolution is using the maximum value of contributions from floating-point and memory operations. The predictions based on Equation 2, Prediction 2, are the most accurate of the three predictions.

Table 8. Predictions using three different convolutions.

# CPUs	Real time (s)	Prediction 1 (s)	Prediction 2 (s)	Prediction 3 (s)
64	40.15	35.92	38.56	35.84
96	43.77	37.22	38.82	35.43
128	49.78	42.37	44.37	40.63

4 Verifying Framework components – Paraver and Dimemas

While the prediction results in Tables 5-7 verify the accuracy of the entire framework, it is useful to be able to verify the individual components of the framework. Verification of the single-processor model built with MetaSim tracer and MAPS is done using Paraver. Paraver, developed by CEPBA, is a visualization tool and a set of instrumentation mechanisms targeted at the analysis of message-passing applications via hardware counters, system activity and Dimemas predictions. In this framework, Paraver can either be used to visualize Paraver traces (.prv) of Dimemas simulations or to visualize traces obtained from real executions. Paraver's analysis features enable it to validate performance predictions obtained with the convolution of the MetaSim tracer and MAPS data. This is achievable as Paraver, given a program trace file, is able to perform a detailed quantitative analysis of the program's performance.

For example Paraver can display performance indices such as L1 miss ratios, L2 miss ratios, CPU Bandwidth or Main memory bandwidth as a function of time. The analysis modules of Paraver can compute the average value of one such index as a function of other indices. Therefore, it is possible to obtain the signature of load/store bandwidths versus L2 miss ratio for a real run of the program. This can then be compared to the data reported by MetaSim tracer for validation of simulated results.

Validation of the network simulator Dimemas can be done using the predictions from Blue Horizon (see Tables 5-7). Blue Horizon results verify the network simulator because the MPI traces are collected on Blue Horizon and the CPU ratio into the simulator is 1 (we model the same machine as the one where the trace was collected). Thus the only likely source of error is the simulator itself. What these predictions say is that given an accurate processor ratio the simulator is capable of prediction with an average error of 4%. This is actually within the observed variability of runtimes of that machine.

5 PERC Framework Applied to Understanding Scaling

One can also use this framework to investigate the scalability of an application. In Figure 6 we take the existing prediction for Blue Horizon on 64, 96, and 128 processors and modify the network parameters. If significantly improving the network of the machine has little to no effect on the performance, then it is clear that the limiting factor for scalability at these sizes is not the hardware, but something inherent in the application or some other aspects. This application (Matrix.F) already shows good scalability to these sizes and, as is seen in Figure 6, improving the processor's capability (but not the network) will benefit this application. When we improved the network without improving the processor, almost no performance gains resulted, confirming that scalability of this application is not limited by this network hardware.
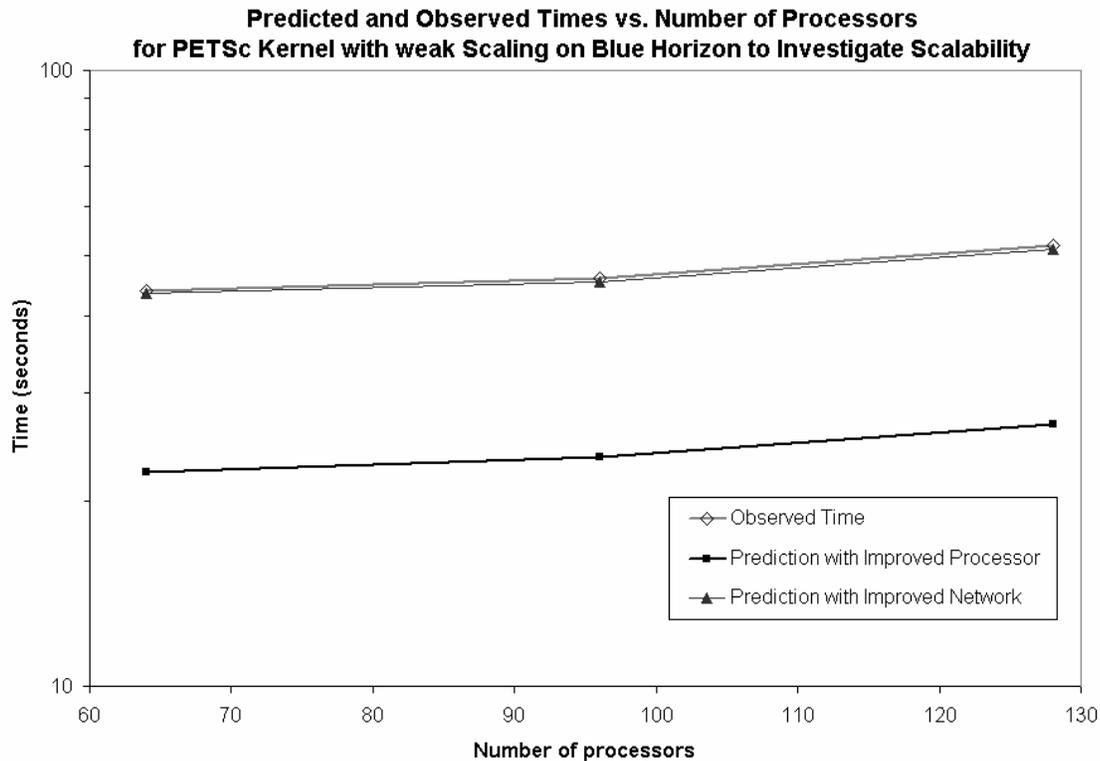
Figure 6. Predictions for Blue Horizon with network and processor improvements to investigate scalability.

The application in Figure 6 already shows good scalability, so the prediction framework is able to predict those hardware upgrades that will benefit that application the most. For those codes that do not scale as well, the framework is able to identify factors that are limiting the scalability. Figure 7 shows an application, run using weak scaling, that has poor scaling at 128 processors. The application is run through the framework with two predictions, one with improved network performance and one with improved processor performance. From the figure one can see that with an improved network the scalability of the application improves, confirming that the bottleneck for scalability is not the application but the hardware on which it is running. Conversely if the predictions with the improve network had not shown scalability improvements then one could conclude that the poor scalability was due to something inherent in the application, such as the algorithm or its implementation.
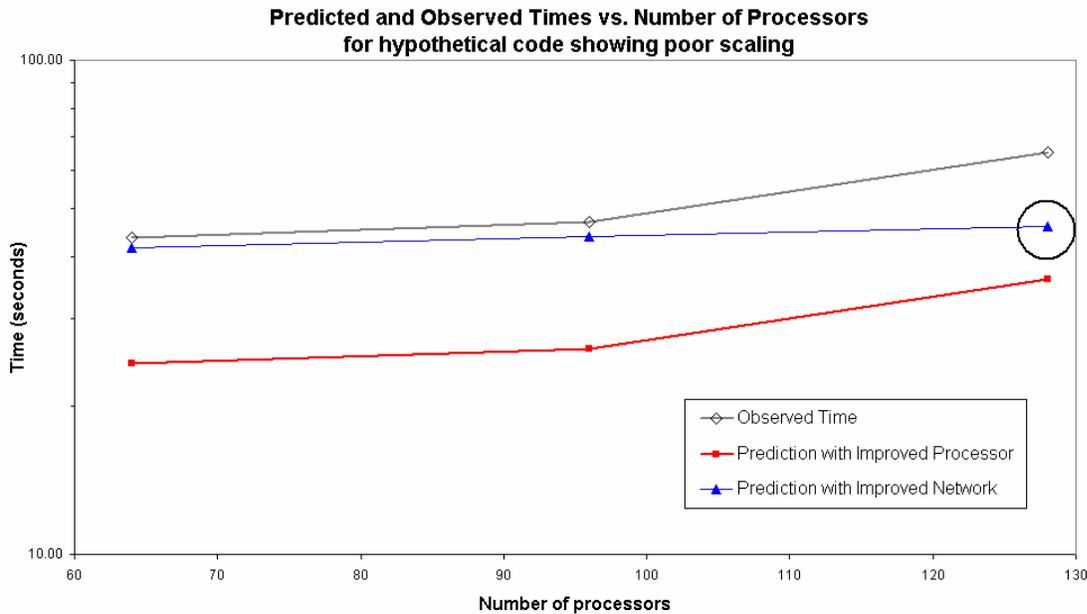
Figure 7. Poor scaling application with an investigation of its bottlenecks.

This sort of information can help scientists study the performance of their applications and determine the type of hardware that is best suited for their applications. Likewise, HPC centers can make better-informed decisions for hardware upgrades and new purchases based on user workload predictions using these tools and techniques.

6 PERC Framework Applied to Machine Tuning

When a performance prediction does not match reality, it may be the fault of the machine. We initially had dismal error rates on Blue Horizon whereby we predicted the applications should run 30% or more faster. By investigation we found the MP_INTERDELAY parameter was set incorrectly on BH. When this was corrected we saw very high agreement between prediction and observed runtime.

We initially found our predictions overestimated the PWR4 node by about 25%. By investigation we found the ESSLSMP library was spawning more threads than were requested thus adversely affecting performance. When this was corrected we got much better agreement between predicted and observed performance.

When the framework has been confirmed to work for a set of applications, then predictions for these applications can be useful tests for centers. Performance predictions can be useful to investigate the setup of new machines or machines after upgrades. When the predictions do not match the real time runs, centers will have an indication that performance of the machine is not as expected.

7 PERC Framework Applied to Projecting Impact of Architectural Upgrades

We are using the framework to explore the likely performance of future architectures. As a validating exercise we predicted Lemieux would be about 1.25 faster than TCSini on the

Matrix.F problem due to faster processors of the same kind and an improved (double-bandwidth) interconnect. In fact, once we got access to Lemieux, we found it to be about 1.35 faster.

We are exploring several options for building a Tflops system using PW4 and/or McKinley processors with MyraNet or another future switch fabric. We will report on these investigations (modulo non-disclosures) in the future. By way of a brief example we tried filling in the 96, and 128 processor points in Table 2 for Longhorn by an extrapolating simulation assuming the Blue Horizon interconnect. We predict runtimes on the Matrix.F problem of 25.24 seconds on 96 processors and 28.84 on the scaled up problem at 128 processors.

8 Conclusions

It is tempting to make performance models as complicated as possible to capture all of the features of modern processors, memory subsystems, and interconnects and the ways these can interact with a program. Taken to the extreme this approach yields cycle-accurate simulators that, while very useful for many kinds of investigations, are not very useful for modeling the performance of full applications at scale on large HPC systems due to time limitations. We prefer an approach that attempts to see how much of the factors that affect performance can be attributed to few parameters only adding complexity as needed to explain observed phenomena. We found this PERC framework has attributes of simplicity and abstraction that make it useful and enlightening for a range of performance investigations.

Acknowledgements

Cited References

1.   see perc.nersc.gov
2.   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, 26-29 Aug. 1996. p675-88 vol.2
3.   See http://www.cepba.upc.es/tools_i.html
4.   W. M. Thorburn, "Occam's razor," *Mind*, 24, pp. 287-288, 1915.
5.   see http://www-fp.mcs.anl.gov/petsc/
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. 2001.
7.   L. Carrington, N. Wolter, and A. Snavely, "A Framework to For Application Performance Prediction to Enable Scalability Understanding", *Scaling to New Heights Workshop*, Pittsburgh, May 2002.
8.   see http://www.sdsc.edu/PMaC/MAPS/
9.   see www.pallas.com/pages/pmb.htm

10.   see http://www.sdsc.edu/PMaC/MetaSim/
11.   see www.cepba.upc.es/tools_i.html
12.   see www.cs.virginia.edu/stream
13.   see www.dyninst.org
14.   see http://www.cepba.upc.es/
15.   see http://www.mcs.anl.gov/performance/TOPS.htm

General References in the area of Performance Modeling

1   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

2.   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.

3.   S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, L. C. McInnes, and B. F. Smith, "PETSc home page", http://www.mcs.anl.gov/petsc, 2001.

4.   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.

5.   J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich. "FLASH vs. (Simulated) FLASH: Closing the Simulation Loop", In Proceedings of *the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 49-58, November 2000.

6.   S. E. Chatterjee, E. Parker, P. J. Hanlon, and A. R. Lebeck., "Exact Analysis of Cache Misses in Nested Loops," *ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, June 20-22, 2001, Snowbird, Utah (to appear).

7.   S. Ghosh, M. Martonosi and S. Malik, "Caches Miss Equations: A Compiler Framework for Analyzing and Tuning Memory Behavior", *ACM Transactions on Programming Languages and Systems*, vol. 21, no. 4, pg. 703-746, July, 1999.

8.   D. A. B. Weikle, S.A. McKee, K. Skadron and W.A. Wulf, "Caches as Filters: A Framework for the Analysis of Caching Systems", *Third Grace Hopper Celebration of Women in Computing*, Sept. 14-16, 2000, Cape Cod, Massachusetts.

9.   L. DeRose, and D. A. Reed, "Pablo: A Multi-language, Architecture-Independent Performance Analysis System", *International Conference on Parallel Processing*, August 1999.

10.   L. DeRose, Y. Zhang, and D. A. Reed, "SvPablo: A Multi-Language Performance Analysis System," 10th International Conference on Computer Performance Evaluation – *Modeling Techniques and Tools – Performance Tools'98*, Palma de Mallorca, Spain, September 1998, pp. 352-355.

11.   I. T. Foster, B. Toonen and P. H. Worley, "Performance of Parallel Computers for Spectral Atmospheric Models", *Journal Atmospheric and Oceanic Techology*, vol. 13, no. 5, pg. 1031-1045, 1996.

12.   I. T. Foster and P. H. Worley, "Parallel Algorithms for the Spectral Transform Method", *SIAM Journal on Scientific and Statistical Computing*, vol. 18, no. 3, pg. 806-837, 1997.

13.   W. D. Gropp, D.K. Kaushik, D.E. Keyes and B.F. Smith, "Toward Realistic PerformanceBounds for Implicit CFD Codes", *Proceedings of Parallel CFD'99*, May 23-26, 1999, Williamsburg, Virginia.

14.   C. L. Mendes, and D. A. Reed, "Integrated Compilation and Scalability Analysis for Parallel Systems*", International Conference on Parallel Architectures and Compilation Techniques (PACT'98)*, Paris, France, October 1998, pp.385-392.

15.   P. H. Worley, "Performance Evaluation of the IBM SP and the Compaq AlphaServer SC", ACM *International Conference of Supercomputing 2000*, Santa Fe, New Mexico, May 8 - 11, 2000.

16.   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, 26-29 Aug. 1996. p675-88 vol.2

17.   B. Buck, J. Hollingsworth, "An API for Runtime Code Patching'', *The International Journal of High Performance Computing Applications* , 2000

18.   J. Gustafson, R. Todi; "Conventional Benchmarks as a Sample of the Performance Spectrum", *The Journal of Supercomputing*, 13, 321-342, 1999