

# A Framework For Application Performance Prediction to Enable Scalability Understanding

Laura Carrington  
*San Diego Supercomputer  
Center*  
*lnett@sdsc.edu*

Nicole Wolter  
*San Diego Supercomputer  
Center*  
*wolter@sdsc.edu*

Allan Snavely  
*San Diego Supercomputer  
Center*  
*allans@sdsc.edu*

## Abstract

*This paper presents a performance modeling methodology 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 a class of High Performance Computing PETSc kernels. The method yields insight into the factors that affect performance and scalability on parallel computers.*

## 1. Introduction

As high performance computers become larger and more complex, the task of deciphering code optimizations to enable performance and unlock scaling potential becomes increasingly difficult. Determining and *understanding* the factors that affect scientific application performance is necessary for identifying and ameliorating bottlenecks that limit scalability, as well as for guiding resource selection, assisting machine specific application tuning, and helping to meaningfully compare machines for their performance on the application.

We wish to enable tools and methods for gaining insight into the factors that affect performance. We want to be able to predict the performance of an application on an arbitrary machine. This capability can enable scientists to investigate the factors that are affecting performance and scalability. Given tools that predict the performance of an application, the computational scientist or programmer should be able to gain insight into the performance of the application, and also play with “what if” scenarios for upgrading the machine’s hardware. For example, one would like to know the effect of increasing network bandwidth and decreasing latency. Predicting how this would affect the performance of an application on a large number of processors can give insight into the factors limiting the application’s scalability. If slight performance gains are predicted with an improved network, then the scientist can infer that there may be limitations in the algorithm or implementation that are resulting in unsatisfactory scaling. If, on the other hand, the prediction shows that performance improves, the scientist can recognize that running the code on different hardware may improve the time to solution.

We present a framework for performance modeling, prediction and understanding. The Performance Modeling and Characterization (PMAc, see [www.sdsc.edu/PMAc](http://www.sdsc.edu/PMAc)) lab at the San Diego Supercomputer Center (SDSC) is focused on developing methods and tools for understanding and predicting the performance of scientific applications on HPC platforms. We anticipate that these tools will aid computational scientists in understanding the performance of their scientific applications, and help them to better

understand the bottlenecks that affect their application's scalability. Our methods are in the early stages of development, but already show promise for enabling insight into scaling and other performance issues. In this paper we describe our methodology, progress to date, and results.

In order to simplify the task of understanding and modeling performance, we began by developing methods and tools for kernel versions of scientific applications run on current parallel platforms. Initial work on the performance prediction of two kernels from the NAS Parallel Benchmarks (NPBs) [1] showed promising results [2]. For this study we focused on kernels built from the Portable, Extensible Toolkit for Scientific Computation (PETSc) [3]. PETSc was chosen because it has sets of routines designed for the parallel solution of partial differential equations that are commonly used in scientific models. PETSc routines represent tasks that are performed in a wide variety of scientific applications. By starting with the simpler PETSc routines, we are able to start small where modeling is tractable and add more routines until the kernels become complex enough that they are comparable to full scientific applications.

What distinguishes our work from prior research is our framework for providing useful, accurate performance modeling and performance understanding that is tractable for a wide variety of machines and applications. Previous work either developed very detailed models for performance [4-8], concentrated on tool development [9-10], was very specific to a given application domain [11-13], or focused on integrating compilation with scalability analysis [14]. Additionally, previous work by Worley [15] evaluated specific machines via benchmarking.

## 2. Tools and Techniques for Performance Prediction

The observed performance of a parallel HPC application is complicated; it is a function of (at least) algorithm, implementation, compiler, operating system, underlying processor architecture, and interconnect technology. The approach adopted here is to proceed via principles of simplicity and abstraction.

We have a working assumption that a parallel application's performance is based on two major factors: 1. single processor performance and 2. use of the network. Clearly, there are other factors, but often these dominate. Existing network simulators do a good job of modeling an application's use of the interconnect and capture factors related to scalability [16]. In particular, very reasonable performance estimates have been obtained with the simple L/B (latency/bandwidth) model for communication [17].

For a reasonably complete model of an application's performance, we must also understand and model single processor performance and combine this information with a network simulator. In modeling single processor performance, we separate various performance factors by measuring each in isolation and then combining them to construct a model for performance prediction. These performance factors are captured in *Machine Profiles* and *Application Signatures*.

A Machine Profile is a characterization of the rates at which a machine can (or is projected to) carry out fundamental operations abstract from the particular application. The Memory Access Pattern Signature (MAPS) Benchmark used to collect or project Machine Profiles is described in more detail in section 2.1.

An Application Signature is a detailed summary of the fundamental operations to be carried out by the application, independent of any particular machine. We used MetaSim to collect Application Signatures, see section 2.2 for more detail.

To arrive at the single processor performance of an application, the Machine Profile and the Application Signature are combined using a Convolution Method. A Convolution Method is an algebraic mapping of the Application Signature onto the Machine Profile; this is explained in more detail in section 2.3.

In section 2.4, we show how to combine single-processor performance from a Convolution Method with the network simulator, DIMEMAS [16], to predict performance of parallel applications.

For each piece of the model, we begin with simple models and few parameters then add complexity only as needed to explain observed performance (Occam's razor). Based on the idea that the per-processor performance of an application is predominately a function of how it exercises the memory sub-system, our starting point for the Application Signature and Machine Profile is focused on the memory hierarchy.

## 2.1 Machine Profiles via MAPS

MAPS is a benchmark probe used to measure the rate at which a single processor can sustain rates of loads and stores depending on the size of the problem and the access pattern. MAPS has been ported to many HPC platforms including IBM Power3, Compaq Alpha, Intel Itanium, Cray T3E, T90, and SV1, NEC SX-4 and SX-5, Sun HPC 10K, and Fujitsu VPP700 and VPP5000. Derived from the STREAMS benchmark [18], MAPS extends STREAMS to various strides and random access patterns. A substantial amount of MAPS data for various machines is available at [www.sdsc.edu/PMaC](http://www.sdsc.edu/PMaC). We are also supporting and distributing this benchmark.

Figure 1 below is the MAPS curve for stride-one loads on the Power3 NightHawk II processor, enhanced with some information about the hardware. It can be seen that the MAPS curve reveals attributes of the machine, and the likely performance implications at a glance.

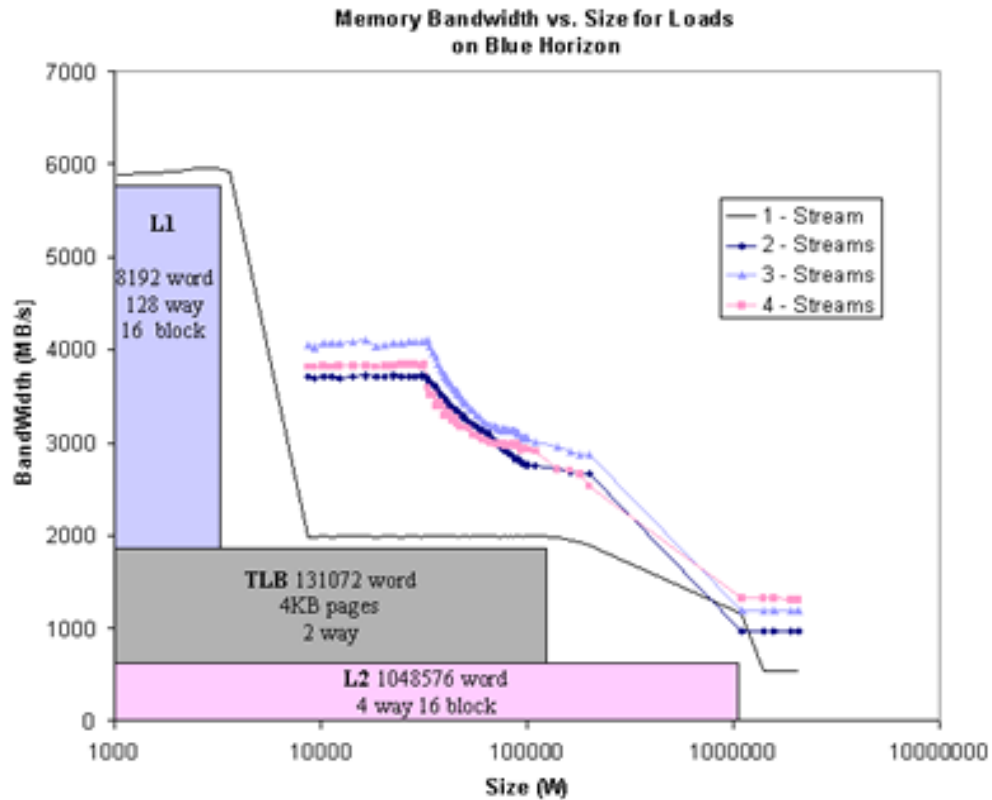


Figure 1. A MAPS generated signature for Blue Horizon (Power3 NightHawk II).

The Power3, like many modern processors with large L2 cache, does not fully map the L2 cache in the Translation Lookaside Buffer (TLB). Thus, we get a "three stair step" function for memory access patterns, one that is especially pronounced when pre-fetching is enabled.

Figure 2 below shows a typical MAPS generated Machine Profile for stride-one and random loads on Pittsburgh Supercomputer Center's (PSC) TCSini. It shows that the sustainable rate of memory loads depends on the size of the problem and the memory access pattern. A large load-bound problem with a random memory access pattern may run 10 or even 100 times slower than a small problem that fits in L1 cache and accesses memory sequentially, or better yet has a small working set yielding significant cache reuse.

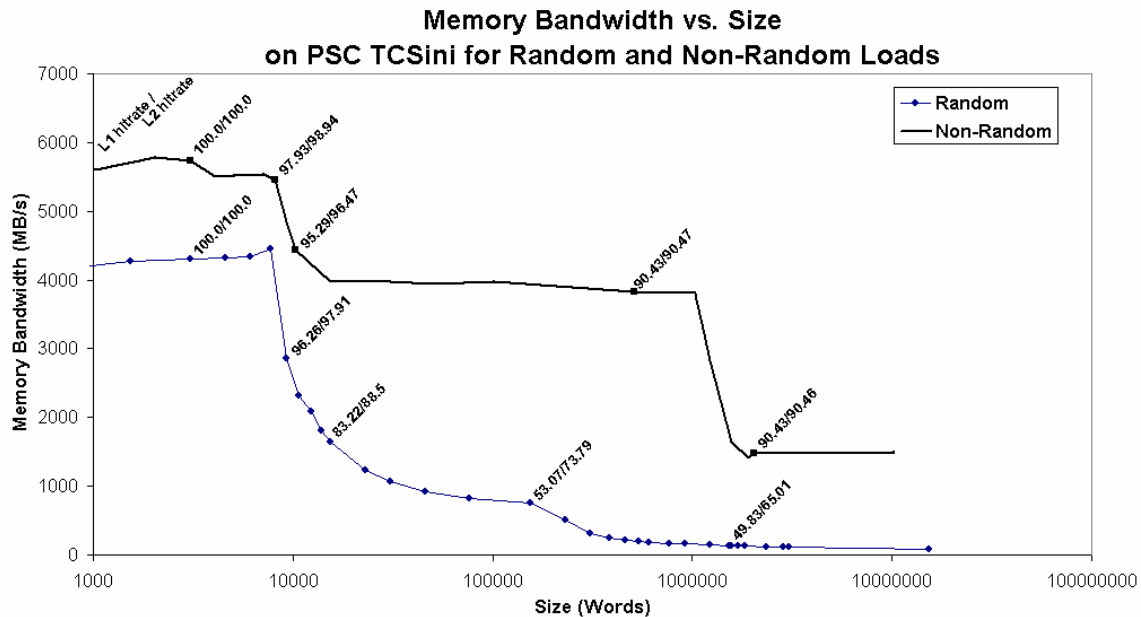


Figure 2. A MAPS generated profile for the PSC TCSini. Random and stride-one curves are marked with L1 and L2 cache hit rates.

Figure 2 also illustrates that a high hit rate in cache is required on TCSini for the cache to be useful as a latency hiding tool. A miss rate of 83% in L2 is sufficient to reduce the speed of random loads to nearly the rate expected for main-memory loads. Similar results apply to many cache-based architectures we have measured, including the Nighthawk II processors on Blue Horizon. When modeling the performance of an application, it is important to map its loops and subroutines to their expected performance on the MAPS curve. Statistics for this mapping can be gathered with MetaSim.

## 2.2 Application Signatures via MetaSim

MetaSim is a tool developed by PMAc to generate an Application Signature. MetaSim is similar to a semi-cycle accurate simulator, and is capable of gathering an abundance of information about an application. While MetaSim gathers multiple types of data, our Application Signatures are constructed primarily with the data that describes how an application uses a processor's memory hierarchy. Since an accurate performance model for the microprocessor often results by modeling the memory hierarchy, we may greatly simplify or even omit modeling functional unit queues, out-of-order issue, branch prediction, and a number of other microprocessor subtleties that we *could* model but decline to in these cases because they do not lie in the critical path for performance.

MetaSim is implemented on top of the ATOM toolkit available only on Alphas. We have an ongoing effort to implement it on top of DyninstAPI [19], a portable instrumentation library. While MetaSim only runs on Alphas, it can emulate an arbitrary machine with less than 100-fold slowdown, so it is much faster than traditional cycle-accurate simulators. Intuitively, it may seem that a fast, simple simulator such as

MetaSim is an inaccurate simulator. One might reasonably presume that if we modeled more attributes of the microprocessor, our simulations would be more accurate (though slower). However, Gibson et. al.[5] showed that simulators that model many architectural features have more possible sources for error, resulting in complex simulators that produce greater than 50% error, while simple simulators are sometimes more accurate than complex ones. We will show some supporting evidence for this in section 3.

MetaSim works by gathering statistics on expected cache hit rates of routines and loops in an application. It accomplishes this by snooping the memory bus to catch addresses as they are generated by load and store operations, then calculates memory access patterns based on the addresses. This calculation is done by statistically binning the addresses into stride buckets based on comparison with the last N addresses that have been encountered. It works in a way analogous to the hardware pre-fetching mechanism on some modern processors where memory access patterns of the program are dynamically discovered under instrumented execution.

MetaSim, using this memory access pattern information combined with some user-supplied information, generates the Application Signature. The information supplied by the user describes the memory hierarchy of a machine (real or hypothetical) for which the user wishes to predict the performance of the application. MetaSim generates an Application Signature for the particular machine by processing the application's address stream against the memory subsystem of the user-supplied parameterization of the (hypothetical or real) machine.

MetaSim gathers information, at the basic block level, on how an application might use the predicted machine's memory sub-system. It does so by gathering memory access pattern information and dynamic counts of each instruction type, and associates these with the basic block that generated them. What emerges is a detailed signature of basic blocks sorted by their contribution to the dynamic instruction count, and further profiled by instruction mix, ratio of arithmetic operations (as distinct from control flow operations) to memory operations, and memory access pattern. Such a signature is orders of magnitude smaller than an address trace file, yet contains an abundance of information suitable for careful performance analysis. Table 1 shows an example of the information generated in a MetaSim simulation of SDSC's Blue Horizon for a PETSc Matrix-Vector multiply kernel.

Table 1. MetaSim output for Matrix-vector multiply kernel simulation Blue Horizon.

<b>Basic Block #</b>	<b>% Cumulative Memory Ref.</b>	<b>% Memory Ref.</b>	<b>FLOPS/MOPS</b>	<b>Fraction Random</b>	<b>L1 Hit Rate</b>	<b>L2 Hit Rate</b>
919	0.246	0.246	0.485	0.0	100.0	100.0
25	0.443	0.197	0.960	0.0	95.3	95.3
921	0.591	0.148	0.942	0.0	100.0	100.0

As can be seen, three basic blocks account for 60% of the total dynamic memory references (and total execution time). The summary report lists the basic blocks in the order of contribution to total dynamic memory references. It provides for each basic block its percentage of dynamic memory references, its ratio of floating-point operations to memory operations, the fraction of its memory accesses that are stride-random (the tool can generate a more detailed analysis of stride patterns), and the hit rates in L1 and

L2 cache. For this kernel, Table 1 shows that the top three basic blocks account for a large portion of the memory references. However, other kernels may have twenty or thirty basic blocks that contribute significantly to performance, and real applications may have several hundred significant basic blocks. Automated gathering and summary of program properties is therefore essential for performance analysis.

MetaSim does not directly measure cache hit rates on the machine where it runs, but it can calculate the cache hit rates that would result if the application were run on a processor of arbitrary (user supplied) configuration with respect to cache sizes, line lengths, and associativities (i.e. it calculates hit rates for a different machine). In the next section, we show how to use this information to map each basic block to its expected performance on the MAPS curve, and weight each basic block appropriately to predict how a program will perform on the arbitrary processor.

### 2.3 Combining MAPS and MetaSim via the Convolution Method

The Convolution Method is a way of mapping a Machine Profile to an Application Signature to arrive at a per-processor or single processor performance prediction.

We determine the predicted sustainable rate of loads and stores by mapping the MetaSim profile block-by-block onto the MAPS curve points that match profiled cache hit rates and memory access patterns. As mentioned above, the per-processor performance of memory bound codes is often determined by the rate at which the machine can do loads and stores and other architectural and application features may be “in the noise” with respect to performance prediction.

Figure 3 below shows a Machine Profile (MAPS plot), and an enclosed table (2) of the MetaSim data. The data generated from MetaSim is convolved with the data measured by the MAPS curves. Table 2 is an example of such data for those basic blocks with random-loads from memory.

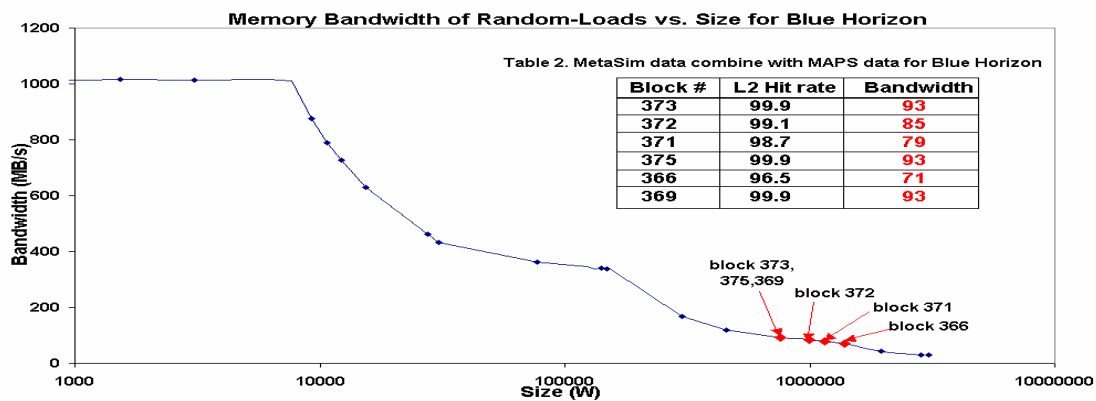


Figure 3. Blue Horizon Machine Profile with MetaSim data.

Once the bandwidths have been collected from the appropriate MAPS curve for all basic blocks, Equation 1 (a simple convolution) can be used to sum the weighted bandwidths of each basic block to calculate its MetaSim number, which is the

application's expected performance on that machine. The ratio of the predicted machine's MetaSim number to the traced machine's MetaSim number is used as a *processor speed ratio* supplied to the DIMEMAS network simulator.

$$(1) \quad \text{MetaSim Number} = \sum_{i=1}^n (\text{Wt. BB}_i * \text{Rate BB}_i)$$

In Equation 1,  $\text{BB}_i$  is a basic block,  $\text{Wt. BB}_i$  is the percentage of total dynamic memory references in the program contributed by this basic block, and  $\text{Rate BB}_i$  is the MAPS measured rate at which MetaSim reports the basic block can sustain loads and stores based on the size of its working set and its access pattern.

We do not expect, however, that such simple convolutions will yield accurate performance predictions in all cases. We report the degree of accuracy yielded by such simple models, and add additional complexity only when required to explain observed phenomena.

## 2.4 DIMEMAS, the Network Simulator

DIMEMAS, developed by the European Center for Parallelism of Barcelona [16], is a network simulator that consumes MPI and/or OpenMP traces and allows modeling of an interconnect of arbitrary topology and bandwidth-latency characteristics. It does not attempt to simulate the execution of an application on a number of processors different from the number of processors the trace was gathered on. However, one can simulate the execution of an application on a machine (real or hypothetical) other than that on which the trace was gathered, by specifying a *processor ratio* for the speed of the simulated machine relative to the machine where the trace was gathered, as well as network topology and network bandwidth/latency characteristics of the machine to be simulated. DIMEMAS does a good job of modeling the network, but provides little guidance for choosing processor speed ratios. Indeed, the notion of relative processor speed is (we believe) overly simplistic. The relative speed of two processors depends on the problem, but is often a function of the rate at which the processors can sustain loads and stores depending on access patterns and the size of the working set. Furthermore, the relative speed may vary throughout the program. In this study we address this issue by using MAPS and MetaSim to model single-processor performance (Convolution Method). Thus, improving the accuracy and resolution of the processor ratio for DIMEMAS.

## 3. Results and Conclusions

The tools and methods described in the previous section were used in multiple studies. The first validated single processor predictions using MAPs and MetaSim by predicting the performance of serial kernels. The second study predicted parallel PETSc kernels on two current architectures and a proposed future architecture. The final study used performance prediction to understand the scalability limitations of the kernels. For both single processor and parallel processor studies, a PETSc kernel was built using the PETSc routines required to build matrices and vectors across multiple CPUs and multiply



them. This routine does a Matrix-Vector multiplication in parallel or serial, an operation common in scientific applications.

### 3.1 Single processor predictions of serial PETSc.

Figure 4 below shows the results of applying the convolution in Equation 1 to the serial version of the PETSc kernel for two problem sizes. As can be seen, the method is accurate for these simple kernels. In this case, the kernels were profiled on the Compaq Alpha cluster at Pittsburgh Supercomputer Center using MetaSim, and their performance was predicted on SDSC's Blue Horizon. Predicted results are compared to actual observed performance. The kernels are predicted well within the accuracy usually associated with cycle-accurate simulation, this at the cost of less than a sixty-fold slowdown for profiling with MetaSim, and a roughly equal amount of time spent in gathering Blue Horizon's MAPS profile. In fact, the performance predictions are within the variability of observed runtimes on Blue Horizon. This suggests that the high cost of a cycle-accurate simulation (frequently as high as a 6 orders of magnitude slowdown) would be overkill for these memory-bound kernels. Furthermore, this shows that it is properties of the memory hierarchy that determine the performance of these kernels on these processors.

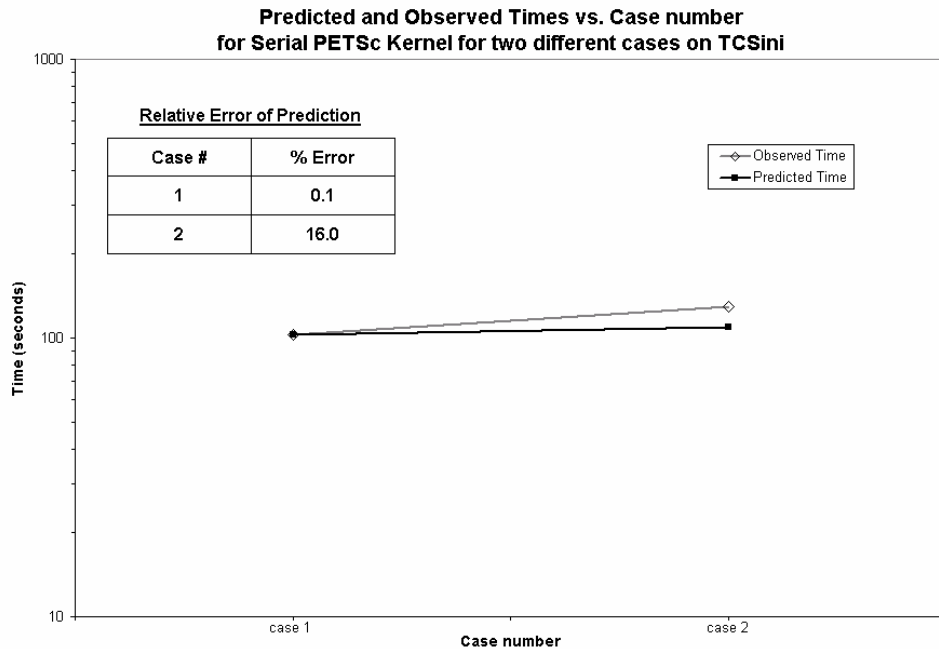


Figure 4. Single Processor Kernel Predictions vs. Observed Timings for Blue Horizon with percentage error.

The results of the single processor prediction also help validate the use of MAPS and MetaSim for single processor performance used in the parallel predictions studies.

### 3.2 Predictions of PETSc on current and future platforms

Once the single processor predictions were validated, the next study involved predicting a parallel version of the kernel on different platforms at different sizes and scaling. All sizes were built so that the local matrix on each processor no longer fit into cache and was forced to employ the main memory of the processor.

Both ASCI and non-ASCI scaling were investigated. In ASCI scaling, the size of the local matrix is kept constant as the number of processors increase. In non-ASCI scaling, the size of the global matrix is kept constant while the number of processors increase. Along with the different sizes and scaling, two groupings of processor sizes were also studied. The kernel was predicted on small number of processors (2, 4, and 8), and also on larger number of processors (64, 96, and 128).

Figures 5 and 6 show the prediction results for the kernel using ASCI scaling for PSC's TCSini and SDSC's Blue Horizon on 2, 4, and 8 processors for MM size. Similar results are shown in Figures 7 and 8 for non-ASCI scaling. For all the predictions, the traces used for the network simulator were collected on SDSC's Blue Horizon.

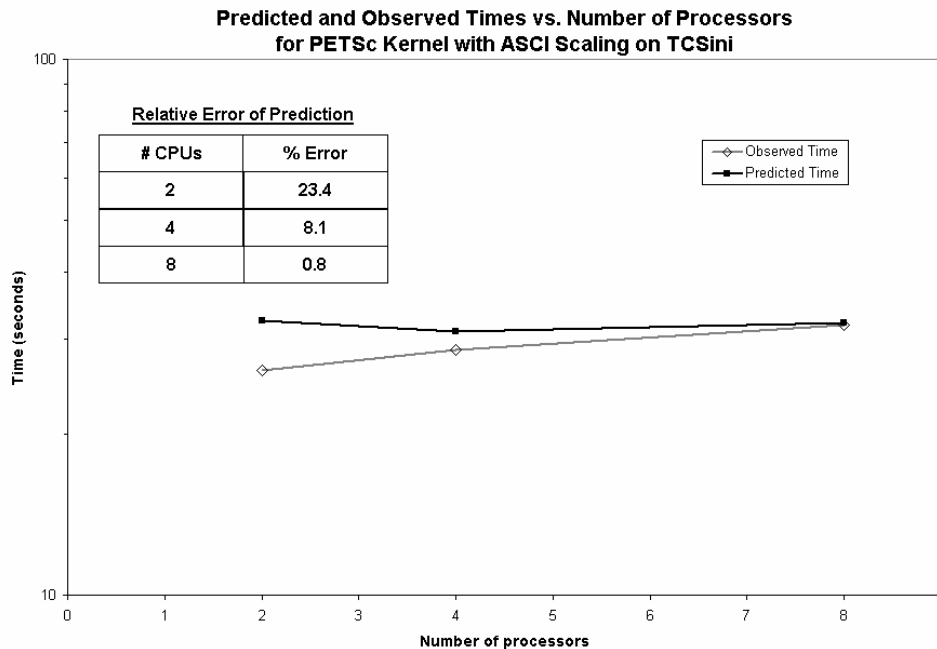


Figure 5. Predicted and Observed time vs. number of processors for ASCI scaling on TCSini.

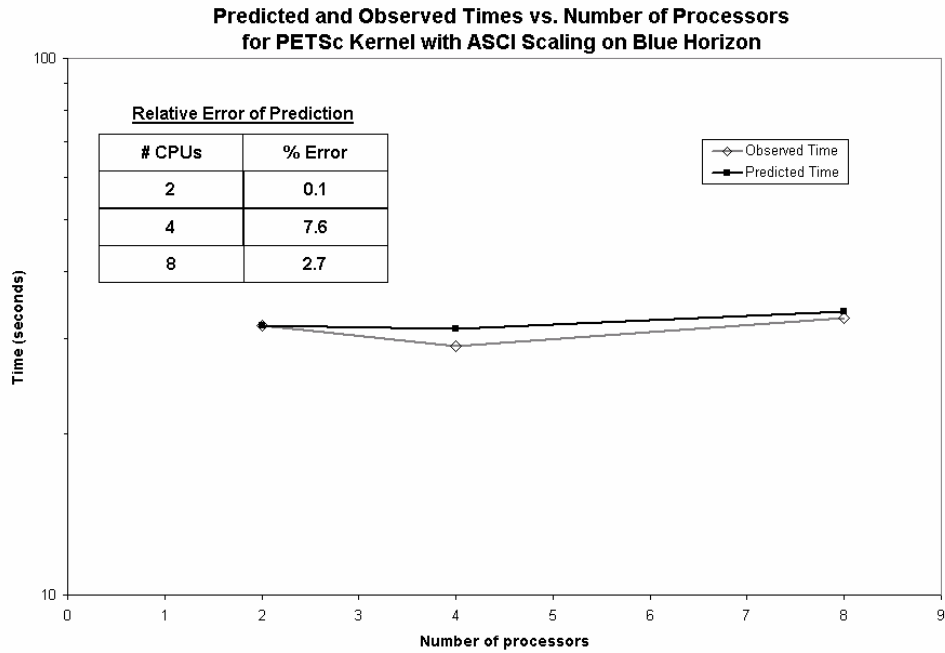


Figure 6. Predicted and Observed time vs. number of processors for ASCI scaling on Blue Horizon.

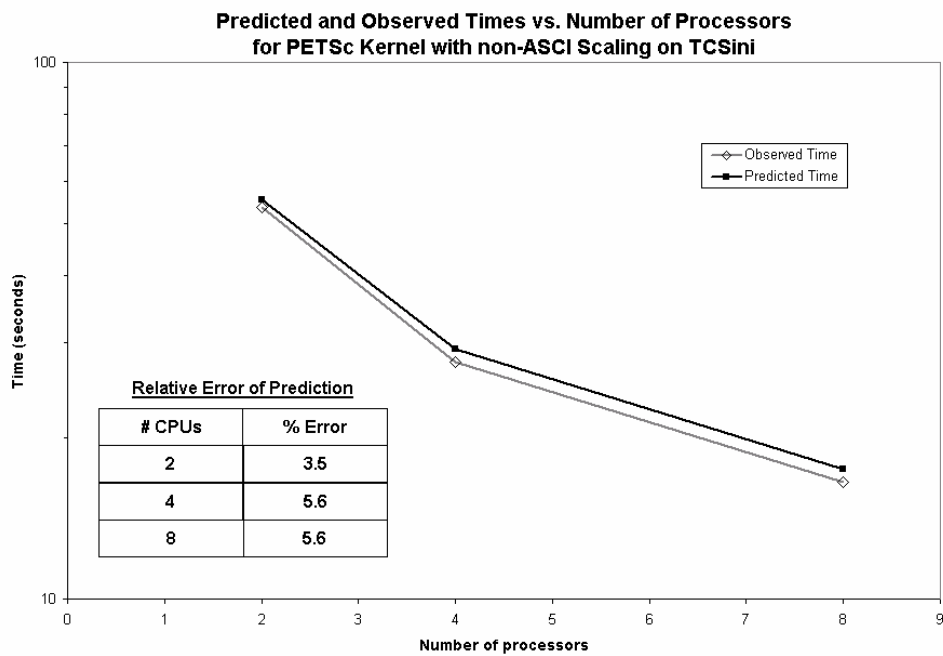


Figure 7. Predicted and Observed time vs. number of processors for non-ASCI scaling on TCSini.

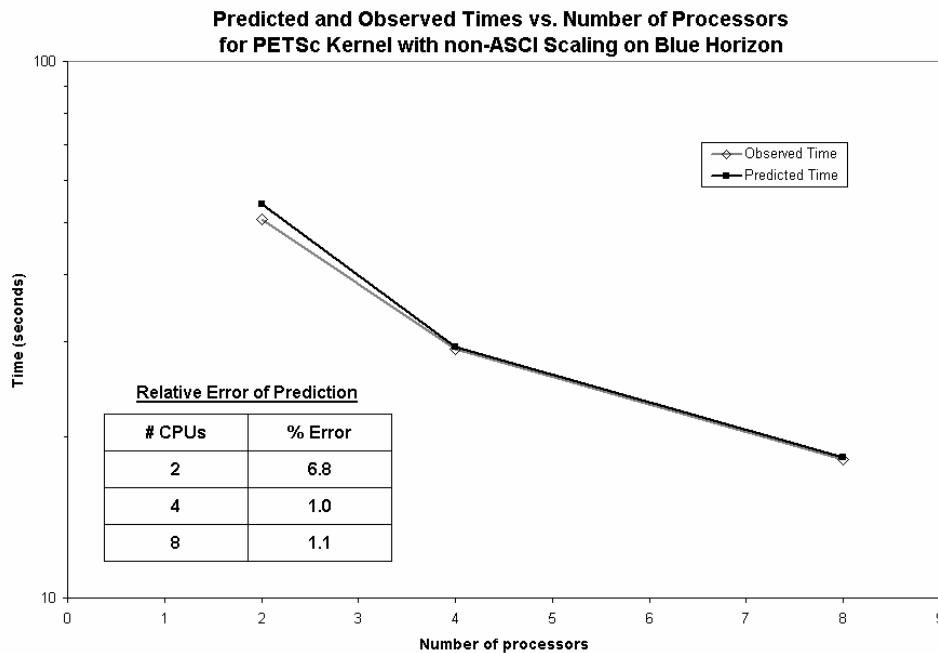


Figure 8. Predicted and Observed time vs. number of processors for non-ASCI scaling on Blue Horizon.

The prediction errors range from 1-22%, which we consider good for a simple model. Figures 6 and 8 show the predictions for Blue Horizon. Since traces for all the predictions were collected on Blue Horizon, we do not use these predictions to validate our prediction techniques, but rather to confirm that the network simulator, given the right value for the processor ratio, can simulate an HPC architecture's network with reasonable accuracy.

These techniques were also used to predict the kernel at much larger size runs in terms of processors. The results of these predictions can be seen in Figures 9 -12. These figures show that for both ASCII and non-ASCII scaling the predictions are within a reasonable amount of accuracy, especially given that at these size runs, variability in runtime can be larger than the error of the predictions. This is presumably due to factors that neither our methods, nor other modeling methods, capture – the impact of other user's jobs on shared resources including the interconnect.

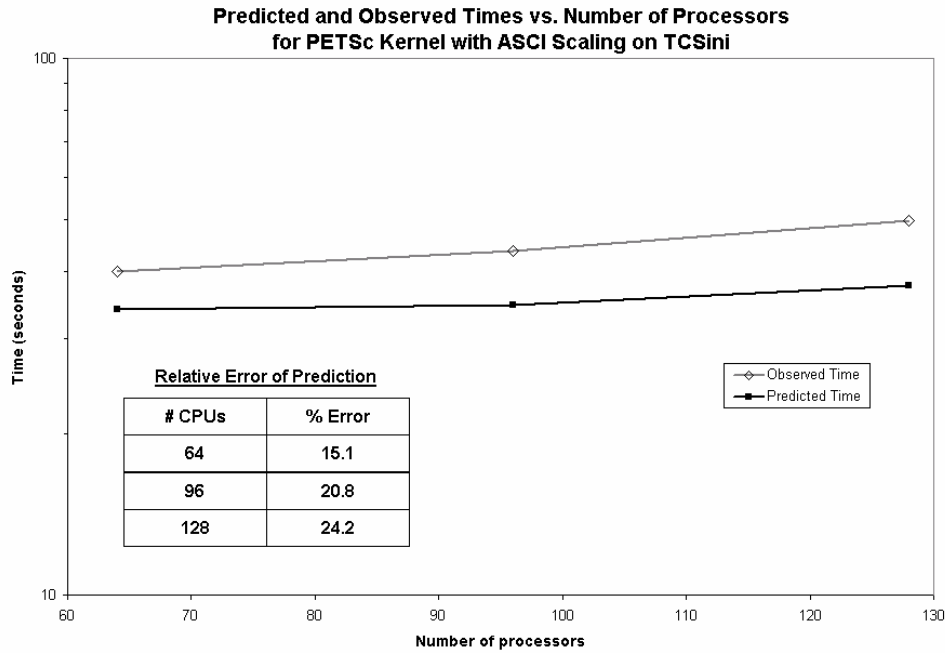


Figure 9. Predicted and Observed time vs. number of processors for ASCI scaling on TCSini.

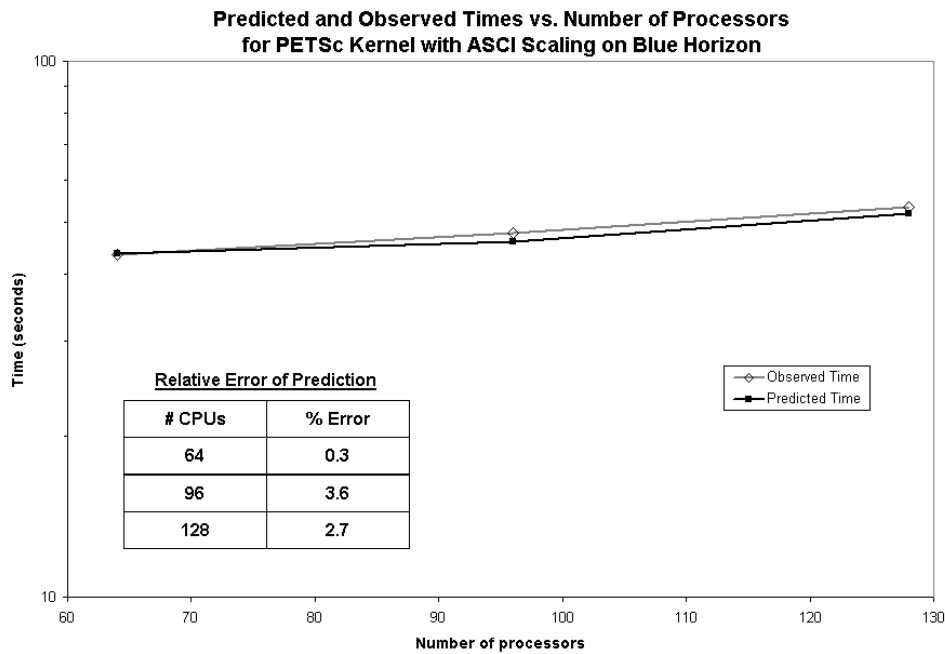


Figure 10. Predicted and Observed time vs. number of processors for ASCI scaling on Blue Horizon.

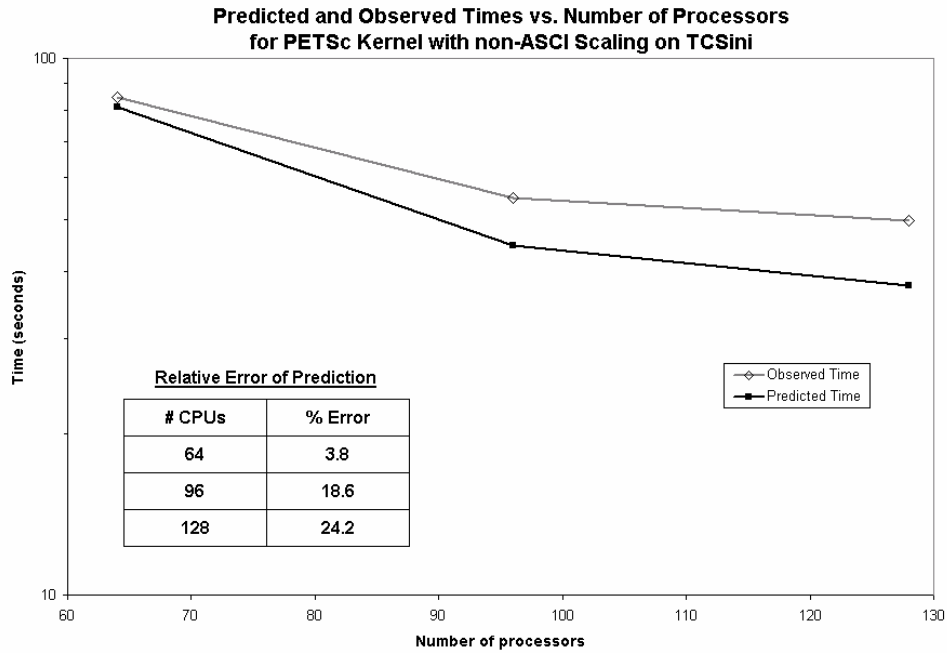


Figure 11. Predicted and Observed time vs. number of processors for non-ASCII scaling on TCSini.

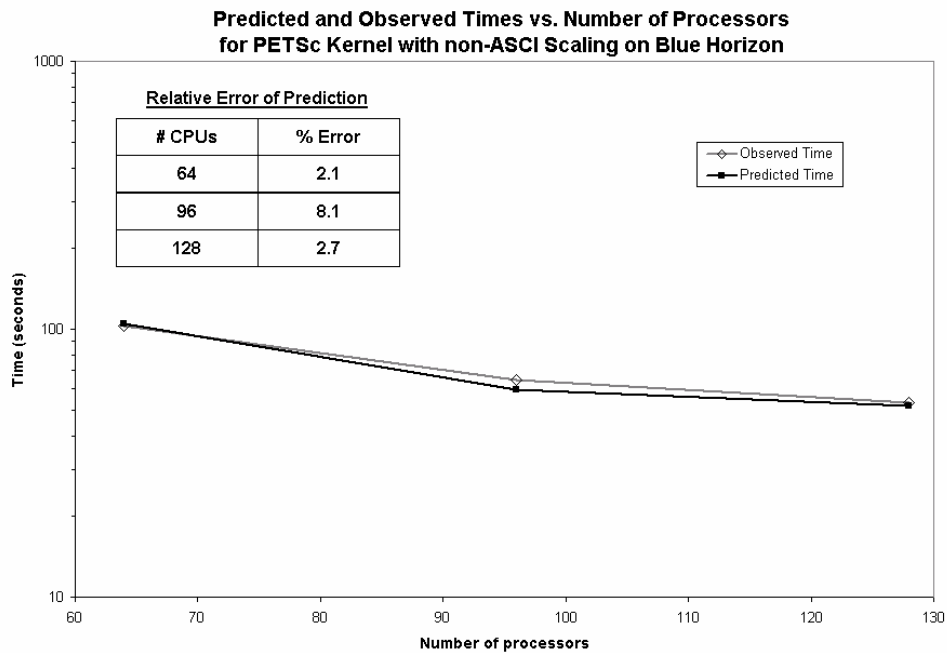


Figure 12. Predicted and Observed time vs. number of processors for non-ASCII scaling on Blue Horizon.

### 3.3 Predictions of Future Architectures

The same techniques used to predict applications on current architectures can be applied to theoretical architectures. In this study we consider a hypothetical machine, with three levels of cache, where the first and second level of cache have the same sustainable bandwidth. Since the machine does not exist the MAPS curve is built by estimating the curve shape based on the MAPS curve of Blue Horizon. Figure 13 shows the projected MAPS curve based on reasonably expected improvements in the memory bandwidth and latency. This along with MetaSim data were used to predict how the kernel might perform. While these results cannot be validated yet, previous predictions in figures 5-12 show that the tools and techniques can be quite accurate and give some legitimacy to this future projection. Figure 14 shows the predicted performance of the PETSc kernel on the hypothetical machine relative to the predicted performance of Blue Horizon. Figure 14 also shows how performance predictions can be valuable in the design and/or purchase of new hardware.

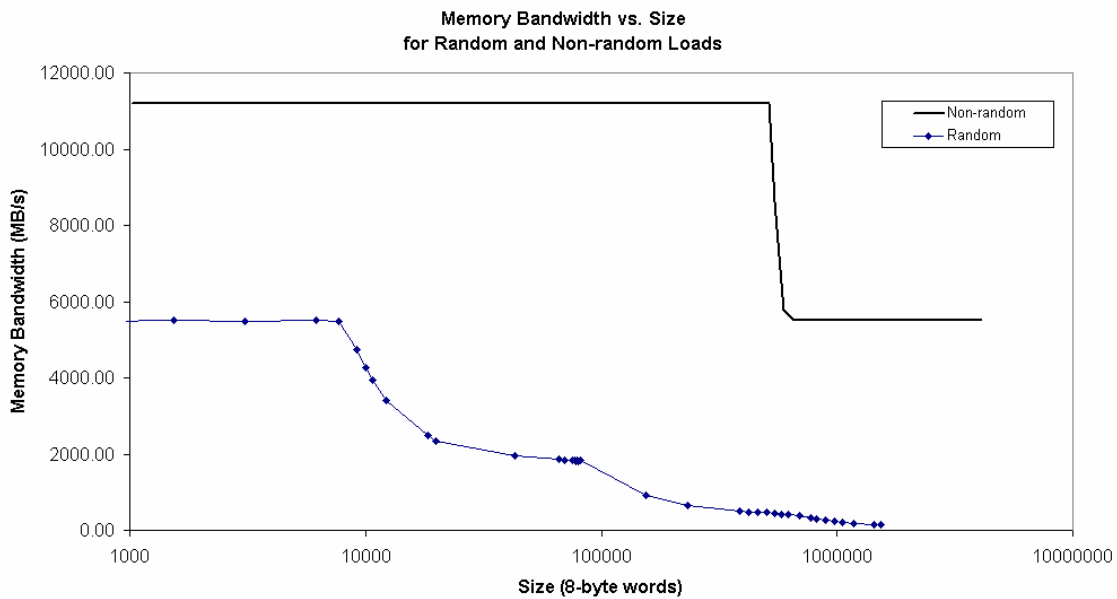


Figure 13. Estimated MAPs curve for Non-random and random loads on the hypothetical machine.

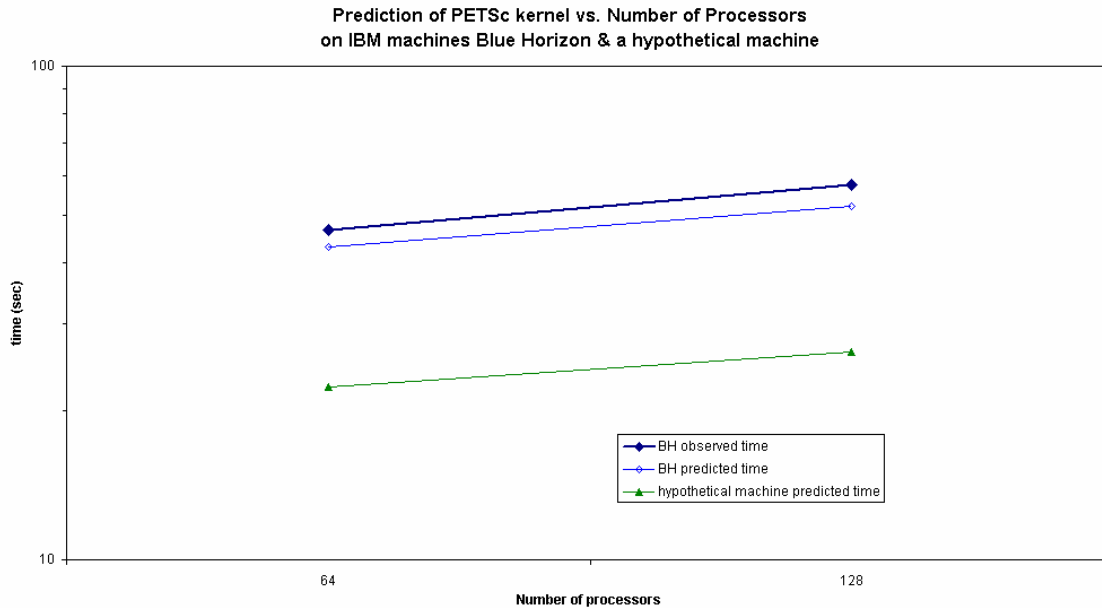


Figure 14. Predictions for Blue Horizon and the hypothetical machine for PETSc kernel ASCI scaling.

### 3.4 Using Performance Prediction to Investigate Scalability

One can also use these techniques to investigate the scalability of an application. In Figure 15 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 already shows good scalability to these sizes and, as seen in Figure 16, 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 is not limited by this network hardware.



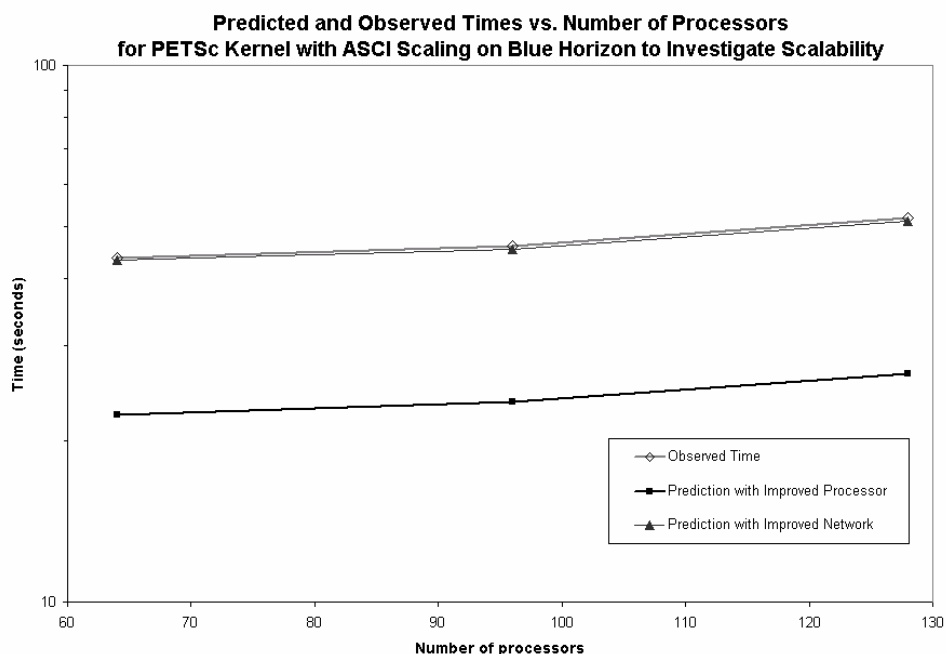


Figure 15. Predictions for Blue Horizon with network and processor improvements to investigate scalability.

It is this type of information that 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.

### Acknowledgements

This work was sponsored the Department of Energy Office of Science through SciDAC award “High-End Computer System Performance: Science and Engineering”. 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.

### References:

1. See <http://science.nas.nasa.gov/Software/NPB>.
2. A. Snively, N. Wolter, and L. Carrington, “Modeling Application Performance by Convoluting 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," 10<sup>th</sup> 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 Technology*, 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 Performance Bounds 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. See [http://www.cepba.upc.es/tools\\_i.html](http://www.cepba.upc.es/tools_i.html)
  17. 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

18. See <http://www.cs.virginia.edu/stream/>
19. B. Buck, J. Hollingsworth, "An API for Runtime Code Patching", *The International Journal of High Performance Computing Applications* , 2000