Applying an Automated Framework to Produce Accurate Blind Performance Predictions of Full-Scale HPC Applications

Laura Carrington, Nicole Wolter, Allan Snavely, and Cynthia Bailey Lee

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

Abstract: This work builds on an existing performance modeling framework that has been proven effective on a variety of HPC systems. This paper will illustrate the framework's power by creating blind predictions for three systems as well as establishing sensitivity studies to advance understanding of observed and anticipated performance of both architecture and application. The predictions are termed blind because the results were completed without any knowledge of the real runtime of the applications; the real performance was then ascertained independently by a third-party. Two applications, Cobalt60 and HYCOM, were predicted to illustrate the frameworks accuracy and functionalities.

Keywords: Performance Modeling; Performance Evaluation; High Performance Computing

1. Introduction

Performance of a parallel application on a High Performance Computing (HPC) machine is resultant from at least factors of algorithm, implementation, the compiler, operating system, underlying processor architecture, and interconnect technologies. Therefore one might conclude that performance models for scientific applications on these complex systems must account for all of the above system and application attributes. This work shows that a framework based on simplicity, including only the major factors in performance, can predict an application's performance.

This framework is designed to have tools that combine simulation and analytical modeling to automate the entire performance prediction process for an application. The design implements easy to use tools that create an accurate model in a reasonable amount of time for users and centers. In previous work [25-27], this framework was described and validated to accurately model and improve understanding of the performance for small parallel scientific kernels and applications on different HPC architectures. One possible criticism to the work was unlimited access and familiarity with the target systems. In this research we challenged the general methodology to predict performance of scientific applications on current HPC platforms, to which access was limited to the running of two small benchmark "probes" to capture low-level performance attributes of the machines. The probes used were MAPS [30], a probe to measure the memory bandwidth of a processor or SMP node, and maps_ping,[31], a ping pong benchmark to measure bandwidth and latency across two processors or SMP nodes. The applications performance was predicted with only limited benchmark data from the target machines. Predictions of the applications were completed without any knowledge of the real runtime (blind predictions). The results were evaluated using sensitivity studies, to further explain the observed performance of the application.

The paper will briefly review the different pieces of the framework in section 2. Section 3 applies the framework to the "real world" challenge, showing blind performance predictions for two different large scientific applications. Section 4 will illustrate processor and network investigations enabled by the framework on those

applications. Section 5 describes the background and related work, some of which this is based on.

2. A Performance Modeling Framework

In the pursuit of rapid, useful, and accurate performance models, that account for complexities of the memory hierarchy and work with all arbitrary applications on all arbitrary machines, the performance modeling framework's design is based on the principles of isolation and simplicity. Measuring the various performance factors in isolation enables independent performance investigations of each system feature; exhibited in the sensitivity studies of Section 4. This design feature allows for a dynamic framework that, when coupled with the simplicity feature which dictates the framework be based on as few parameters as possible, retains the ability to easily add and remove significant factors as needed to sufficiently depict a given application or system. A detailed description of the framework can be found in Snavely et al [26].

Based on the hypothesis that a parallel application's performance is often dominated by two major factors: 1) single processor performance and 2) use of the network, the framework was developed to model these factors along with only some of the features of modern, highly complex processor. Starting simple and only adding complexity when needed to account for observed performance, the framework consists of a single processor model, combined with a communication model (see Figure 1). Clearly, there are other factors that can affect performance, but often processor and network performance are sufficient for accurate performance prediction (~10% error) while adding more factors only increases the complexity of the model with nominal gains (~1-2%) in accuracy [26].

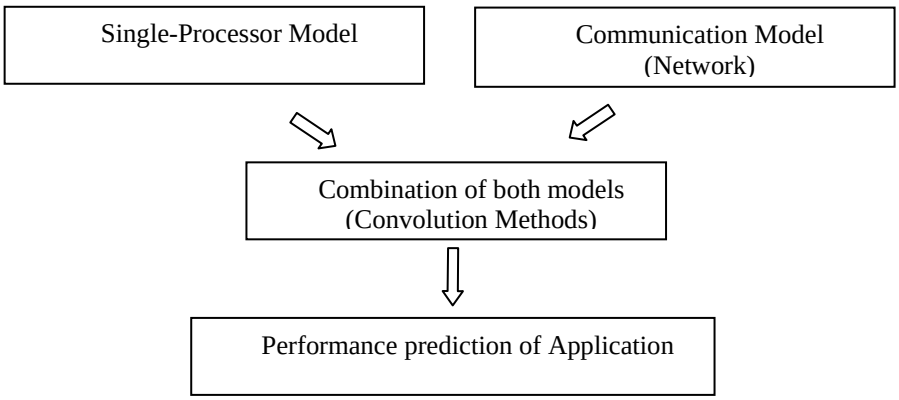

Figure 1. Performance prediction framework for a parallel application.

The single-processor and network models both use independent *Application Signatures* and *Machine Profiles*, which are combined using *Convolution Methods*. An Application Signature is a summary of the operations to be carried out by an application, including memory and communication access patterns, independent of any particular machine. Application Signatures are collected via traces. For the single-processor model these are memory traces collected via the MetaSim Tracer. For the communication model these are MPI traces collected by MPIDtrace.

A Machine Profile is measurements of the rates at which a machine can perform basic operations, including message passing, memory loads and stores, and floating-point operations, independent of any particular application. This data is collected via low level benchmarks or probes. To arrive at a performance prediction for an application, its Application Signature is mapped to the corresponding performance in the Machine Profile of the machine on which the application is being predicted, by the Convolution

Methods. These mappings are automated by the MetaSim Convolver for the single-processor model and Dimemas for the communications model. The convolutions of the Application Signature and Machine Profile result in a runtime, which the application should achieve on the target machine. The framework is composed of tools to automate each of the components and steps in the performance prediction of an application. This allows anyone to feed an application through the framework and arrive at a runtime prediction on any HPC system.

Comparing a predicted run time with the actual runtime is the method we use for validating the model for that application [18]. Validation of models for two different scientific applications is covered in section 3.

3. HPC Application Prediction and Model Verification

In sections 3.1-3.2, we apply the framework to two scientific applications to predict their performance on multiple HPC architectures. Cobalt60 and HYCOM predictions are all "blind" predictions. "Blind" predictions imply that the predictions were created before the modelers had full access to the machine and the real runtimes of the applications were not collected by the modelers and were not shown to the modelers until after the predictions were complete. Only small benchmarks were run on the target machines to collect a machine profile. These benchmarks only used a few CPUs of the target machine but were used in predicting performance of an application running on hundreds of CPUs. The advantage of this is that typically in building large (>1000 CPUs) HPC machines a small prototype will be available long before the full system can be built. The benchmarks can be run on the prototype system and predict the full system before it is built.

MetaSim and MPI (mpidtrace) traces were collected on independent machines, PWR3 system Blue Horizon at the San Diego Super Computer Center (SDSC) and an SC45 based system, Lemieux, located at Pittsburgh Supercomputing Center (PSC). Scientists local to the target systems were responsible for collecting the real time runs and both predicted times and real runtimes were given to a third party to evaluate. The verdict as to the accuracy of the predictions was handed down at a meeting [31] by the funding agency.

3.1 HYCOM (HYbrid Coordinate Ocean Model)

HYCOM is a primitive equation general ocean circulation model using density, pressure, and sigma coordinates in the vertical. It evolved from the Miami Isopycnic-Coordinate Ocean Model (MICOM) HYCOM was developed to address known shortcoming of the MICOM vertical coordinate scheme.

The data sets run for these predictions were a 26-layer 1/12-degree fully global data set. The smaller data set executes HYCOM for 1 model day and requires about 0.75 GB of memory per processor and approximately 4 GB of globally accessible scratch disk. The larger data set executes HYCOM for 0.3125 model days and requires about 0.9 GB of memory per processor and about 23 GB of globally accessible scratch disk.

HYCOM was modeled on three systems for three different processor counts. The standard input set was used on 59 and 96 processor counts and the large input was used for 234-processor count. The results are seen in table 1. An error of around 20% is considered acceptable to our user/funding agency for the purpose of getting a general idea of the application's performance on the target machine. Table 1 show that predictions for the PWR3 system are well below the acceptable limit. The predictions for the SC45 have 2 of the 3 prediction well below the acceptable limit and the third prediction only slightly above the limit. For the PWR4 system 2 of the 3 predictions are within the limit with the third prediction being off the limit. The large error from the third prediction can be partially attributed to the current framework's inability to

accurately predict I/O performance. The PWR4 system has two different file systems, NSF and GPFS with very different performances. This combined with the fact that at the 234 CPU run I/O can contribute to nearly 10% of the run may account for the large error for that prediction.

Table 1. Blind Performance prediction of HYCOM application on three machines.

# of CPUs	Habu (IBM PWR3)			Emerald (Compaq SC45)		
	Real Time (s)	Predicted Time (s)	% Error*	Real Time (s)	Predicted Time(s)	% Error
59	4558	4359	4.6	2382	1960	21.5
96	2605	2624	-0.7	1432	1378	3.9
234	5612	5311	5.7	3043	3300	-7.8

# of CPUs	Marcellus (IBM PWR4)		
	Real Time (s)	Predicted Time(s)	% Error
59	2081	1799	15.7
96	1205	1210	-0.4
234	2692	2062	30.6

*% Error = (Real time – Predicted Time)/ (Real Time)*100

3.2 Cobalt60

Cobalt60 is an unstructured Euler/Navier-Stokes flow solver that is routinely used to provide quick, accurate aerodynamic solutions to complex CFD problems. Cobalt60 handles arbitrary cell types as well as hybrid grids that give the user added flexibility in their design environment. It is a robust HPC application that solves the compressible Navier-Stokes equations using an unstructured Navier-Stokes solver. It uses Detached-Eddy Simulation (DES), which is a combination of Reynolds-averaged Navier-Stokes (RANS) models, and Large Eddy Simulation (LES).

The predictions were for the large input data set from Department of Defense's (DoD) TI-04 procurement set. The data set simulated a 3-D model of turbulent viscous flow over the geometrics of a wind tunnel with an airplane wing with a flap and endplates that has 7,287,723 cells.

Cobalt60 was modeled for three systems on 64 processors. The results for modeled, real time and percent error for a large input are exhibited in table 2. They show that all three predictions are under the acceptable limit of error, which in turn validates the accuracy of the model.

Table 2. Blind Performance prediction of Cobalt60 application on large input data.

# of CPUs	Habu (IBM PWR3)			Emerald (Compaq SC45)		
	Real Time (s)	Predicted Time (s)	% Error	Real Time (s)	Predicted Time(s)	% Error
128	3672.07	4092.90	-11.5	2352	1956	20.2

# of CPUs	Marcellus (IBM PWR4)		
	Real Time (s)	Predicted Time(s)	% Error
64	1631	1865	-12.5

Based on the results above, the majority of predictions for the two applications were within acceptable range of error. This helps in the validation of the methodology applied to full-scale HPC applications. We are currently investigating how to improve accuracy. Having verified a performance model's accuracy, one can proceed to investigating independent performance factors of the hardware to establish how they effect an application's overall performance. The sensitivity studies that follow were established for both applications as they were described in section 3.

4. Performance Sensitivity Studies

The usability of a model, that can accurately predict application performance based on properties of the code and the machine, can be extended to carry out precise modeling experiments. Such research can be used to quantify the performance impact of different machine hardware components. This information, for example, would be valuable in machine procurement for a workload comprised of these applications.

The following exercises can be applied to any size problem and for any application modeled via the framework. As an example, the models created and verified in the previous section, were used to explain and quantify observed performance of the processor and network. The study observed the performance effects of the two different applications for permutations of a Power3 system architecture via machine-independent performance sensitivity studies.

The performance model for Cobalt60 on 64 processors was used to investigate the general performance effects of a machine's network and processor on the application. The starting point for the investigation was the performance of the application on Habu, which is a 4-way IBM Power3 system with a Colony switch located at Naval Oceanographic Office (NAVO). The model was used to investigate the effects of improving different hardware components by a factor of four. The data was normalized to the original performance on Habu, referred to as the "base" case. The cases were broken down as follows: Case 1 represents the performance of Cobalt60 run on the "base" machine configuration. Case 2 shows performance of improving peak network bandwidth by a factor of 4 and case 3 illustrates performance effect when the network latency was reduced by a factor of four. Case 4 depicts Habu with a complete network upgrade, four-fold improvements to the network latency and peak bandwidth. Case 5 illustrates the performance effects Cobalt60 would see if the Habu "processors" were upgraded four-fold, by increasing the floating-point rate of the CPU and memory subsystem. Case 5 was included to get a general idea of how processor performance affects the application's performance. It is understood that given the gap between memory and float-point performance on a processor that increasing both these components by an even factor of four is not realistic. But the results of Case 5 would show if processor performance was a significant factor or not and if so further studies could be done to split the individual components of memory and floating-point performance.

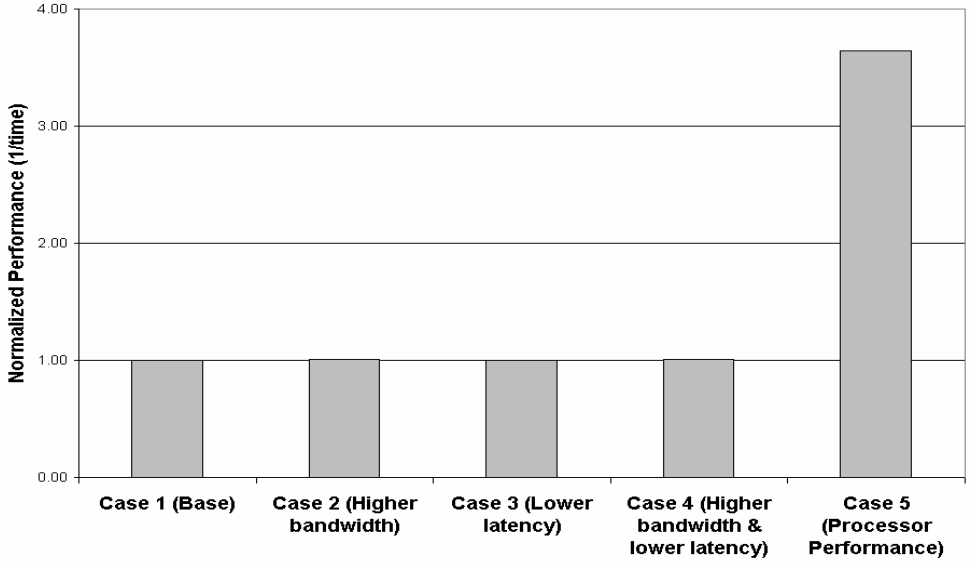

Figure 2. Sensitivity study for Cobalt60 for 64 processors.

At this size, Cobalt60 demonstrates sensitivity to the "processor" upgrade, (faster processor and memory subsystem) as seen in the case 5, but not the network enhancements. The high-level implication of these sensitivity studies directs upgrades for future machines, with respect to Cobalt60 performance, to focus on improving the processor. Having identified the principle performance attribute for Cobalt60 run on 64 CPUs we can dissect the "processor" influences further via a more focused sensitivity study.

Figure 3 "zooms in" to study the effects of independent processor attributes on performance for Cobalt60. In the above results for Cobalt60, the processor improvements show that modeled execution time would decrease by having a four-times better "processor" (case 5 figure 2) "Improving the processor" implies not only an improvement in rate with respect to four-fold floating-point issue but also quadrupling the bandwidth and quartering the latency to all levels of the memory hierarchy. Figure 3 breaks down how a better processor would perform relative to the Power 3 processor for Cobalt60 if it had these performance improvements. Case 2 represents a four-fold improvement in the floating-point rate of the processor. Case 3 shows a two times faster L1 cache bandwidth, while case 4 shows a two times faster L1 and L2 cache bandwidth. Case 5 represents a two times faster L1 and L2 cache bandwidth as well as a two times faster main memory bandwidth.

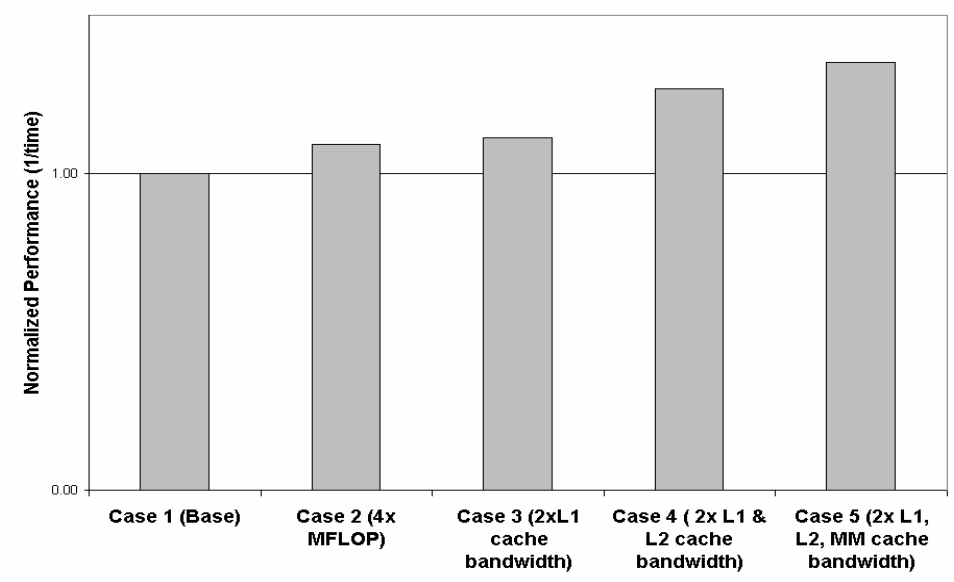

Figure 3. Cobalt60 Node Performance sensitivity study for 64 CPUs.

Case 2 in Figure 3 shows that Cobalt60 at this size is sensitive to the floating-point rate of the processor. However it would benefit more by improving bandwidth to secondary cache and main memory. Cases 4 and 5 imply that since improvements to secondary cache and main memory show the most performance benefits, that is probably where the application's data set is residing for most of the runtime. Such information may be of use to a code developer for code tuning.

Fashioned similarly to the sensitivity studies above, we observed the application behavior for a series of HYCOM studies for three CPU counts on two different size data sets. Figures 4, 5 and 6 mimic the study conducted in figure 2, in which four-fold improvements where made to the processor and network. Figure 4 represents the large data set for 234 processors, figure 5, the standard input set on 59 CPUs and figure 6 the standard data input set on 96 CPUs. Broken down: Case 1 represents the base performance of HYCOM on HABU. Case 2 the performance increase of four-fold network bandwidth enhancement. Case 3 illustrates the performance response to improved latency and case 4 performance attributed to both network bandwidth and latency improvements. Case 5 shows performance related to the processor upgrades.

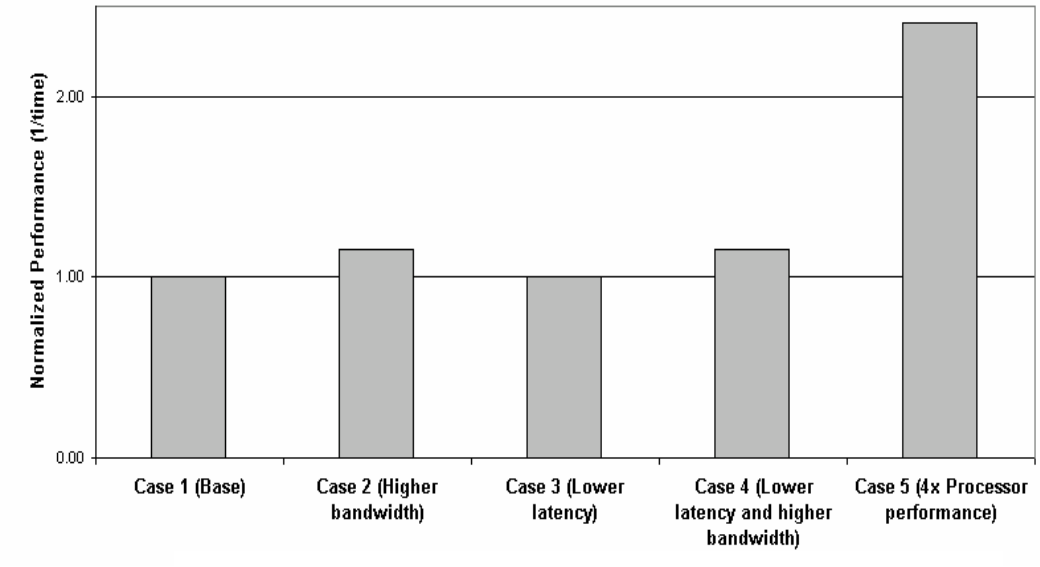

Figure 4. HYCOM sensitivity study for 234 CPUs.

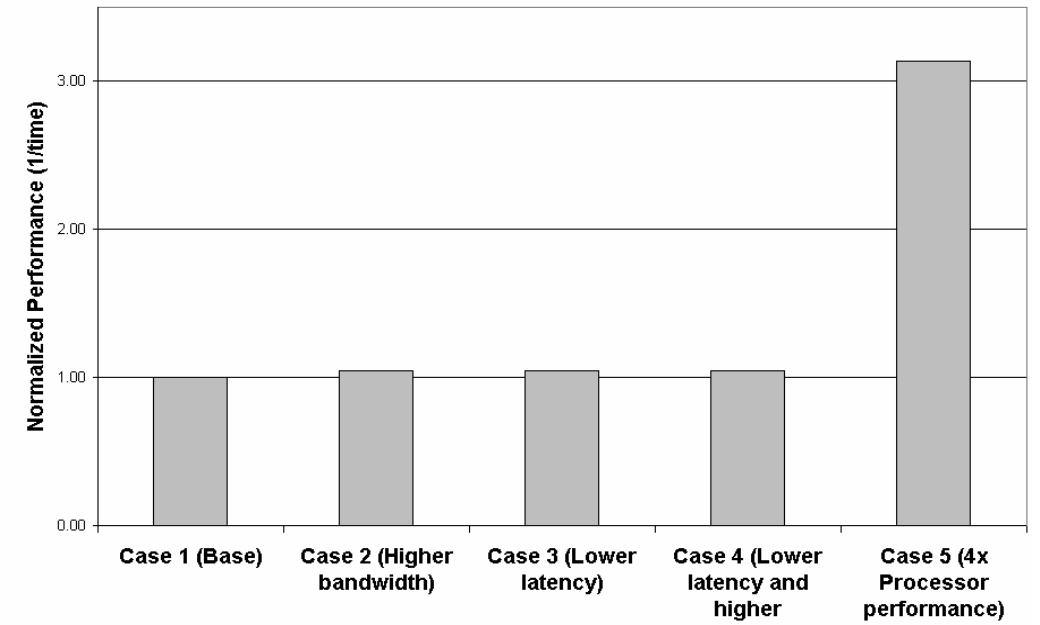

Figure 5. HYCOM sensitivity study for 59 CPUs.

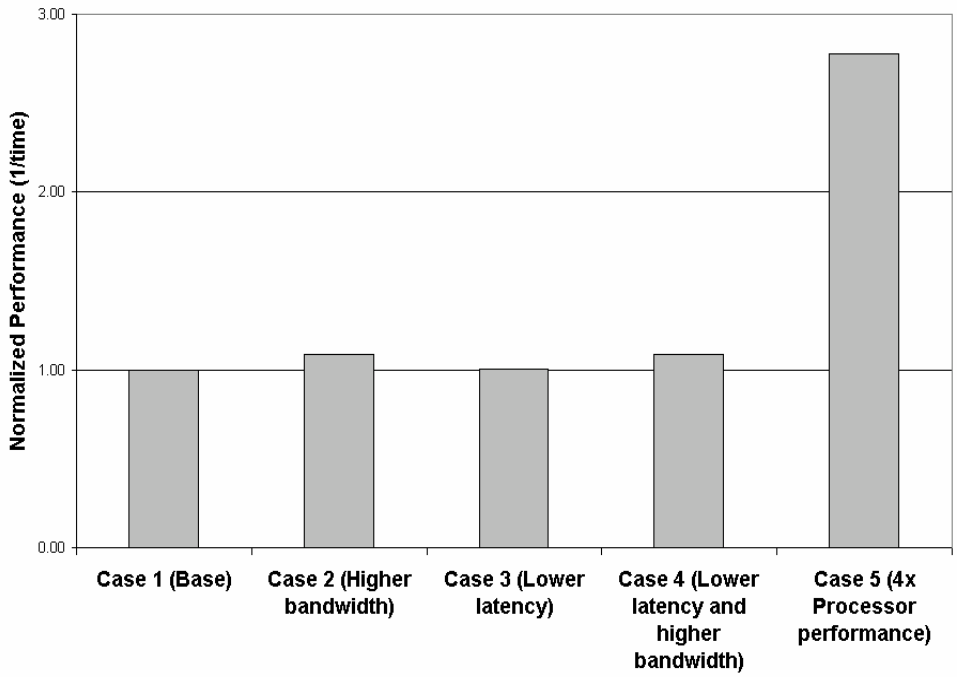

Figure 6. HYCOM sensitivity study for 96 CPUs.

In Figures 4-6, case 5 exemplifies HYCOMs sensitivity to the improvements in the processor performance; this behavior can be observed across all processor counts. It can also be deduced in cases 2 through 4 that there would be no substantial benefit to reducing the network latency or increasing bandwidth.

These sensitivity studies can be used to evaluate application price performance. As illustrated in the previous three figures, HYCOM performance is most influenced by improving the "processor". It should be noted that while all three models reveal performance boosts, HYCOM running on 96 processors, standard test case, saw the biggest performance enhancement with respect to its base performance. HYCOM on 59 CPUs standard test case depicted the second-most and 234 CPU large test case the least, but why? To investigate the individual features, studies were conducted similar to the sensitivity studies in figure 4. Using the HYCOM models at 234, 96 and 59 CPUs, the memory bandwidth and floating point operations were improved to exhibit their influence. In the following three figures the same set up as figure 3 is used. Case 1 is the base case of HABU. Case 2 shows the performance effects when the MFLOP rate of the processor is increased by a factor of four. Case 3 shows the performance effects when the L1 cache memory bandwidth is increased by a factor of two. Case 4 illustrates the effects when both L1 and L2 cache bandwidth are increased by a factor of two. Case 5 displays the effects when L1 and L2 cache bandwidths are increase as well as memory bandwidth by a factor of two.

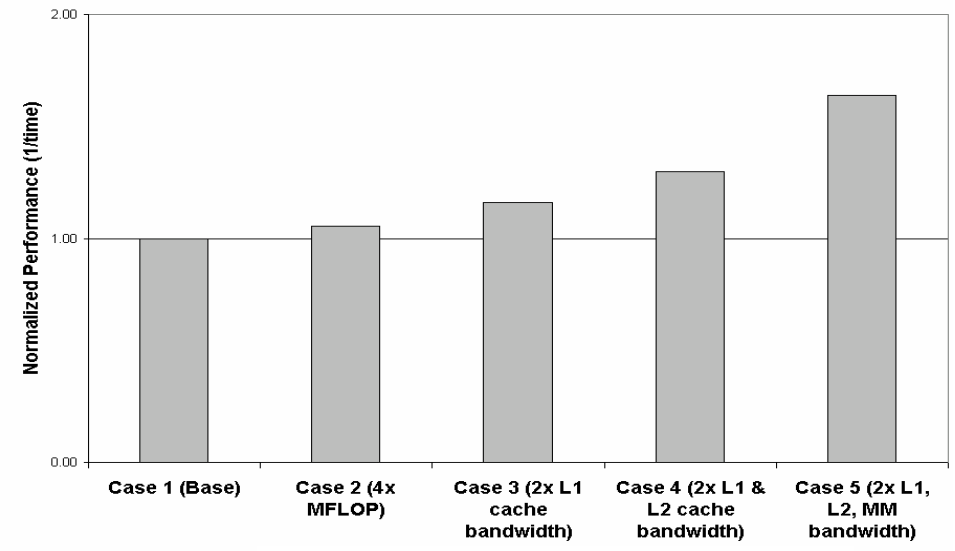

Figure 7. HYCOM processor study for 234.

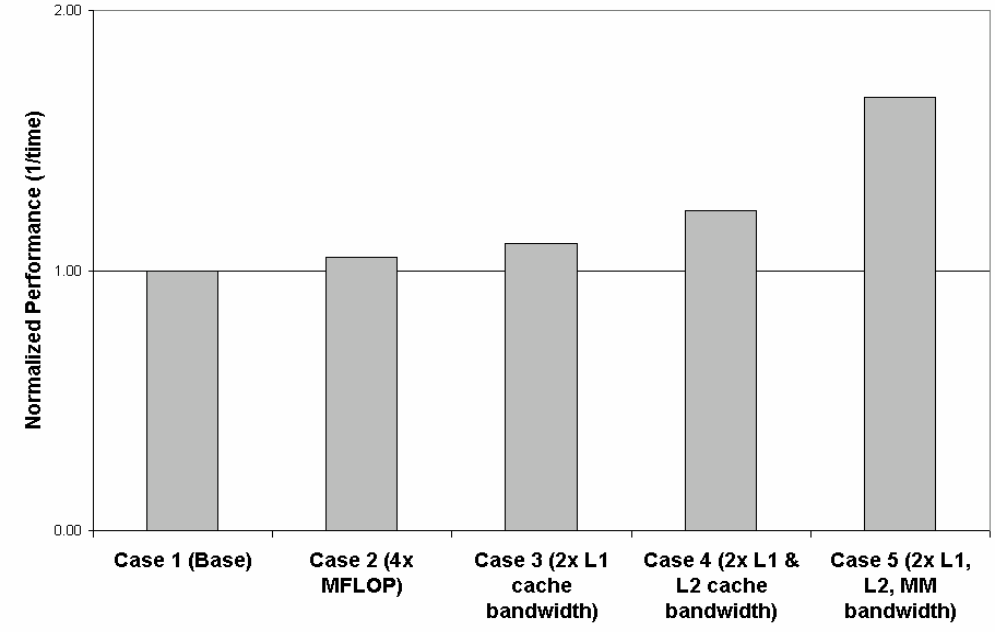

Figure 8. HYCOM processor study for 59 CPUs.

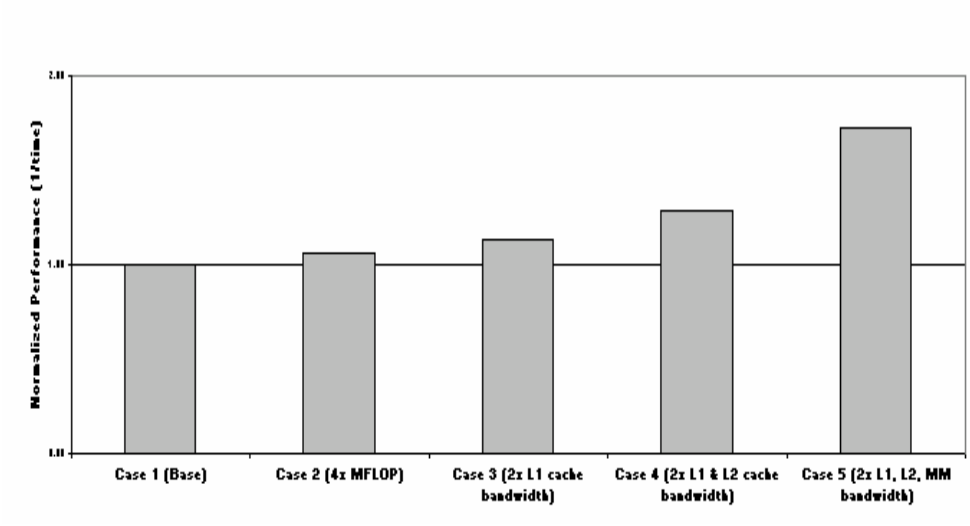

Figure 9. HYCOM processor study for 96 cpus.

By scrutinizing the previous three figures, it can be observed that, by improving the MFLOPs four fold, performance is improved minimally for all three sizes even on the different size data sets. More interesting however are the observations that can be made by modifying the memory structure. Case 5 shows, for all three models, that the overall performance is influenced most by increasing the performance of *main memory*. An interesting comparison between figures 8 and 9 shows that even with the same size data set and more processors, the 96 CPU run is still affected most by performance improvements to main memory. One might think that as the processor count increased from 59 to 96 CPUs that the data would move more into cache. Figure 9 illustrates that because performance improvements to main memory affect the over all performance the most, then the majority of the data is residing in main memory not in cache.

5. Background and Related Work

Methods for performance evaluations can be broken down into two areas [1]: structural models and functional and analytical models. Structural models use descriptions of individual system components and their interactions, such as detailed simulation models. The second area, functional and analytical models, separates the performance factors of a system to create a mathematical model.

The use of detailed or cycle-accurate simulators in performance evaluation has been used by many researchers [2-6]. Detailed simulators are normally built by manufactures during the design stage of an architecture to aid in the design. For parallel machines, two simulators might be used, one for the processor and one for the network. These simulators have the advantage of automating performance prediction from the user's standpoint. The disadvantage is that these simulators are proprietary and often not available to HPC users and Centers. Also, because they capture all the behaviors of the processors, simulations generally take 1,000,000 times longer, than the real runtime of the application [7]. This means, to simulate 1 hour of an application it would take approximately 114 years of CPU time. Direct execution methods are commonly used to accelerate architectural simulations [8] but they still have large slowdowns. To avoid these large computational costs, cycle-accurate simulators are usually only used to simulate a few seconds of an application. This causes a modeling dilemma, for most scientific applications the complete behavior cannot be captured in a few seconds of a production run. Applications rarely spend all their time in one routine, and their behavior may change as the application progresses through its simulation (in some cases the actual physics of the problem being solved changes).

Cycle-accurate simulators are limited to only work in modeling the behavior of the processor for which they were developed, so they are not applicable to other architectures. In addition, the accuracy of cycle-accurate simulation can be questionable. Gibson et al [9] showed that simulators that model many architectural features have many possible sources for error, resulting in complex simulators that produce greater than 50% error. This work suggested that simple simulators are sometimes more accurate than complex ones.

In the second area of performance evaluation, functional and analytical models, the performance of an application on the target machine can be described by a complex mathematical equation. When the equation is fed with the proper input values to describe the target machine, the calculation yields a wall clock time for that application on the target machine. Various flavors of these methods for developing these models have been researched. Below is a brief summary of some of this work but due to space limitations it is not meant to be inclusive of all.

Saavedra [10-12] proposed applications modeling as a collection of independent Abstract FORTRAN Machine tasks. Each abstract task was measured on the target machine and then a linear model was used to predict execution time. In order to include the effects of memory system, they measured miss penalties and miss rates to include in the total overhead. These simple models worked well on the simpler processors and shallower memory-hierarchies of the mid 90's. The models now need to be improved to account for increases in the complexity of parallel architectures including processors, memory subsystems, and interconnects.

For parallel system predictions, Mendes [13-14] proposed a cross platform approach. Traces were used to record the explicit communications among nodes and to build a directed graph based on the trace. Sub-graph isomorphism was then used to study trace stability and to transform the trace for different machine specifications. This approach has merit and needs to be integrated into a full system for applications tracing and modeling of deep memory hierarchies in order to be practically useful today.

Simon [15] proposed to use a Concurrent Task Graph to model applications. A Concurrent Task Graph is a directed acyclic graph whose edges represent the dependence relationship between nodes. In order to predict the execution time, it was proposed to have different models to compute the communication overhead, (FCFS queue for SMP and Bandwidth Latency model for MPI) with models for performance between communications events. As above, these simple models worked better in the mid 1990's than today.

Crovella and LeBlanc [16] proposed complete, orthogonal and meaningful methods to classify all the possible overheads in parallel computation environments and to predict the algorithm performance based on the overhead analysis. Our work adopts their useful nomenclature.

Xu, Zhang, and Sun [17] proposed a semi-empirical multiprocessor performance prediction scheme. For a given application and machine specification, the application first is instantiated to thread graphs which reveal all the possible communications (implicit or explicit) during the computation. They then measured the delay of all the possible communication on the target machine to compute the elapsed time of communication in the thread graph. For the execution time, of each segment in the thread graph between communications, they use partial measurement and loop iteration estimation to predict the execution time. The general idea of prediction from partial measurement is adopted here.

Abandah and Davidson, [18] and Boyd et al [19] proposed hierarchical modeling methods for parallel machines that is kindred in spirit to our work, and was effective on machines in the early and mid 90's.

A group of expert performance modelers at Los Alamos have been perfecting the analytical model of two applications important to their workload for years [20-23]. These models are quite accurate in their predictions, although the methods for creating them are time consuming and not necessarily easily done by non-expert user [24]. Also, the models require input related to the applications data set that is not automated.

6. Conclusion

The simple framework was proven effective in a challenging scenario whereby a third party provided limited benchmark information in advance and then verified the accuracy of the performance predictions subsequently. It was shown effective for predicting the performance of machines on full applications when only low-level performance features of the machines were known in advance. The framework was rendered useful for price performance evaluations via sensitivity studies. The studies can determine beneficial architectural features for a workload to create reasonable procurement standards based on the computational demands of the target workload. It is reasonable to tune current systems and influence the implementation of near-future systems informed by the computational demands of the target workload with the performance information from application models. It is also reasonable to design future systems based on the quantified performance implications of hardware features for characterized workloads.

Acknowledgments

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 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 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, NAVO, and ERDC. Special thanks to Bob Alters of ERDC and Christine Cuicchi of NAVO for collecting real runtimes of the applications. We would also like to acknowledge the European Center for Parallelism of Barcelona, Technical University of Barcelona (CEPBA) for their continued support of their profiling and simulation tools.

References:

1. Svobodova, L. Computer System Performance Measurement and Evaluation Methods: Analysis and Applications. Elsevier, N.Y., 1976.
2. Ballansc, R.S., Cocke, J.A., and Kolsky, H.G. "The Lookahead Unit", Planning a Computer System, McGraw-Hill, New York, 1962.
3. Boland, L.T., Granito, G.D., Marcotte, A.V., Messina, B.V. and Smith, J.W. "The IBM system 360/Model9:Storage System", *IBM J. Res. And Develop*., 11, pp.54-79 (1967).
4. Tjaden, G.S. and Flynn, M.J. "Detection and Parallel Execution of Independent Instructions", *IEEE Trans. Comptrs*., C-19, pp889-895 (1970).
5. Murphey, J.O. and Wade, R.M. "The IBM 360/195", *Datamation*, 16:4, pp. 72-79 (1970).
6. Burger, D., Austin, T.M., and Bennett, S. "Evaluating future microprocessors: The simplescalar tool set" *Tech. Rep. CS-TR-1996-1308*, University of Wisconsin-Madison, 1996.
7. Lo, J., Egger, S., Emer, J., Levy, H., Stamm, R., and Tullsen, D., "Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading", *ACM Transactions on Computer Systems*, August 1997.
8. Falsafi, B. and Wood, D.A "Modeling Cost/Performance of a Parallel Computer Simulator", *ACM Transactions on Modeling and Computer Simulation*, 7:1, pp. 104-130 (1997).
9. Gibson, J., Kunz, R., Ofelt, D., Horowitz, M., Hennessy, J., and Heinrich, M. "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.
10. Saavedra, R.H., Smith, A.J., "Measuring Cache and TLB Performance and Their Effect on Benchmark Run Times", *IEEE Transactions on Computers* 44(10) pp1223-1235 1995
11. Saavedra, R.H., Smith, A.J. "Analysis of Benchmark Characteristics and Benchmark Performance Prediction", *TOCS14*(4) pp344-384,1996
12. Saavedra, R.H., Smith, A.J. "Performance Characterization of Optimizing Compilers", *TSE21*(7) pp615-628,1995
13. Mendes, C.L., Reed, D.A. "Integrated Compilation and Scalability Analysis for Parallel Systems", *IEEE PACT* 1998
14. Mendes, C.L., Reed, D.A. "Performance Stability and Prediction", *IEEE / USP International Workshop on High Performance Computing*, 1994
15. Simon, J., Wierun, J. "Accurate Performance Prediction for Massively Parallel Systems and its Applications", *Euro-Par*, Vol II pp675-688, 1996
16. Crovella, M.E., LeBlanc, T.J. "Parallel Performance Prediction Using Lost Cycles Analysis", S*C 1994* pp600-609
17. Xu, Z., Zhang, X., Sun, L. "Semi-empirical Multiprocessor Performance Predictions", *JPDC* 39, pp 14-28, 1996
18. Abandah, G., Davidson, E.S. "Modeling the Communication Performance of the IBM SP2", *Proceedings Int'l Parallel Processing Symposium*, pp. 249-257, April 1996
19. Boyd, E.L., Azeem, W., Lee, H.H., Shih, T.P., Hung, S.H., and Davidson, E.S. "A Hierarchical Approach to Modeling and Improving the Performance of Scientific Applications on the KSR1", *Proceedings of the 1994 International Conference on Parallel Processing*, Vol. III, pp. 188-192, August 1994
20. Hosie, A., Olaf, L., Wasserman, H. "Performance Analysis of Wavefront Algorithms on Very-Large Scale Distributed Systems", *Springer's "Lecture Notes in Control and Information Sciences"*, 249, p. 171 (1999).
21. Hosie, A., Olaf, L., Wasserman, H. "Scalability Analysis of Multidimensional Wavefront Algorithms on Large-Scale SMP Clusters", *Proceedings of Frontiers of Massively Parallel Computing '99*, Annapolis, MD, February 1999.
22. Kerbyson, D.J., Hoisie, A., and Wasserman, H.J., "Modeling the Performance of Large-Scale Systems", *Keynote paper, UK Performance Engineering Workshop (UKPEW03)*, July 2003, and in IEE Software, Inst. Electrical Engineers, August 2003.
23. Yong, L., Olaf, L.M., Wasserman, H. "Development and Validation of a Hierarchical Memory Model Incorporating CPU- and Memory-Operation Overlap", *Proceedings of the First International Workshop on Software and Performance*, Santa Fe, NM, pp. 152-163 (1996).
24. Spooner, A. and Kerbyson, D. "Identification of Performance Characteristics from Multi-view Trace Analysis", Proc. Of Int. Conf. On Computational Science (ICCS) part 3, 2659, pp. 936-945 (2003).
25. Carrington, L., Wolter, N., and Snavely, A. "A Framework for Application Performance Prediction to Enable Scalability Understanding", *Scaling to New Heights Workshop*, Pittsburgh, May 2002.
26. Snavely, A., Wolter, N., Carrington, L., Badia, R., Labarta, J., Purkasthaya, A. "A Framework

to Enable Performance Modeling and Prediction", *Supercomputing 2002*.
27. Snavely, A., Wolter, N., and Carrington, L., "Modeling Application Performance by Convolving Machine Signatures with Application Profiles", *IEEE 4th Annual Workshop on Workload Characterization*, Austin, Dec. 2, 2001.
28. Davidson, E. and Kumar, B. "Computer System Design Using a Hierarchical Approach to Performance Evaluation", Communications of the ACM, 23:9, pp. 511-521 (1980).
29. http://www.sdsc.edu/PMaC/MAPs/
30. http://www.sdsc.edu/PMaC/Benchmark/maps_ping/
31. http://www.darpa.mil/ipto/programs/hpcs/