Performance Technologies for Peta-Scale Systems: A White Paper Prepared by the Performance Evaluation Research Center and Collaborators

David H. Bailey (LBNL); Bronis de Supinski (LLNL), Jack Dongarra (U. Tenn.), Thomas Dunigan (ORNL), Guang Gao (U. Del.) , Adolfy Hoisie (LANL), Paul Hovland (ANL), Jeffrey Hollingsworth (U. Mar.), David Jefferson (LLNL), Chandrika Kamath (LLNL), Allen Malony (U. Oregon), Boyanna Norris (ANL), Daniel Quinlan (LLNL), Sally McKee (Cornell U.), Celso Mendes (U. Ill.), Shirley Moore (U. Tenn.), Daniel Reed (U. Ill.), Allan Snavely (SDSC), Erich Strohmaier (LBNL), Jeffrey Vetter (LLNL), Patrick Worley (ORNL)

Prepared in Response to the Invitation to Submit White Papers on High End Computing

May 14, 2003

Introduction

Future-looking high end computing initiatives will deploy powerful, large-scale computing platforms that leverage novel component technologies for superior node performance in advanced system architectures with tens or even hundreds of thousands of nodes. Recent advances in performance tools and modeling methodologies suggest that it is feasible to acquire such systems intelligently and achieve excellent performance, while also significantly reducing the user time required to attain high performance. These developments are relevant to several aspects of future HEC technology outlined in the recent HECRTF white paper request, in particular items 5.4, 5.5, 5.6, and 5.8. We envision the following specific capabilities:

1. *Performance modeling tools*, available to researchers and vendors, will extrapolate performance from prototype systems to full-scale systems, and even accurately predict performance behavior before systems are manufactured, thus enabling both improved designs and more intelligent selection of systems in procurements.
2. *System simulation facilities*, implemented on highly parallel platforms and available to researchers and vendors, will for instance realistically model the performance of a specific interprocessor network design running a specific scientific application code. As with item 1, these facilities can lead both to improved designs and procurement decisions that yield significantly greater sustained performance for targeted scientific applications.
3. *A program monitoring and analysis infrastructure*, scalable to 100,000 processors and beyond, will provide performance information at every level of system's memory hierarchy and network. This infrastructure will build upon knowledge discovery and data mining techniques to be significantly more scalable and easier to use than the current infrastructure, and a standard version will be incorporated in most high-end systems.
4. *Self-tuning software facilities*, now available only for a few specialized libraries and requiring separate test runs, will be integrated into a broad range of scientific application codes. Eventually these facilities will make use of the performance monitoring infrastructure mentioned above, and will extend to dynamic optimization at the subroutine level.

Each of these capabilities appears to be feasible, based on successes in current research projects. However, while the prototypes of many of these facilities are already in hand, significant additional research and development will be required to realize the full potential described above. In the following, we sketch some of this required research.

Performance Monitoring Infrastructure

Informal approaches to parallel performance monitoring and analysis may be acceptable at the present time, but such approaches will be woefully inadequate once systems are fielded with multiple levels of parallelism throughout the system's compute nodes, network, and memory hierarchy, and including tens or hundreds of thousands of compute nodes. It is also unlikely that we can effectively model and utilize novel architecture systems without the aid of an advanced monitoring infrastructure.

Advanced hardware performance monitoring facilities will be required to obtain performance data without significant perturbation. A key challenge beyond counting of events throughout the system is in gathering and interpreting the exploding quantity of data. Even now, collecting memory access pattern information, which is often crucial for understanding performance on deep-memory-hierarchy machines, implies a three orders-of-magnitude slowdown [17]. Yet many applications of interest run for hours or days, during which their performance behavior

changes frequently. Systems with tens or hundreds of thousands of processors will greatly compound this performance data analysis problem. Several alternatives are being explored, ranging from clever statistical sampling schemes to on-the-fly analysis of performance data that would reduce the amount of data involved. Meaningful analysis of this data will require advanced techniques such as multivariate statistical methods [1], knowledge discovery tools [19], time series analysis [21] and advanced visualization schemes [2] to distill important facts from these potentially massive data sets. This analysis can then be used to select the key features used in monitoring performance and to build predictive models of the performance of a single processor as well as the entire parallel system.

A unique opportunity exists for performance researchers to work with vendors to improve the selection of hardware performance data. Ideally, design of performance monitoring hardware should be driven by data input needs for application performance modeling and analysis, rather than modeling and analysis capabilities being limited by the available data. For example, one key item that current hardware monitors lack is information regarding memory addresses, such as data on gaps or patterns between successive addresses. This information would provide valuable insights into the memory behavior of a user program. Along this line, we observe that the counters currently available have been designed primarily to address the needs of vendor benchmark personnel. Hopefully in the future vendors will consider counters useful to application developers and performance tuners as well, for example by implementing the PAPI proposed standard metrics [3].

Another area where the performance research and vendor design communities could work together is to extend inter-processor network hardware performance monitoring facilities to application performance analysis. Although network hardware often includes some performance monitoring facilities, the lack of support to associate performance data with a specific application code significantly hinders applying the data to application performance evaluation. The use of reconfigurable technology (such as FPGAs) might be of use to support performance-monitoring applications, for both hardware engineers and end users. Determining what events are most important to monitor, designing systems to support low-overhead monitoring that generates information useful to application developers, and designing software to utilize this information, are important topics of future research.

Any improvements that are made in the capabilities of performance tools must be matched by a corresponding improvement in ease of use, or otherwise they will have only limited impact in the overall goals of reducing time to solution and simplifying system acquisitions. In this regard, it is instructive to observe that while many professionals in the HEC field have produced web content, very few have taken formal training in HTML. Instead, most have merely copied and adapted a colleague's HTML or used higher-level tools. In a similar vein, we envision a set of standard templates for performance analysis that automatically engage a typical performance analysis scenario, using advanced tools. Monitoring should be as automatic as possible. For example, users should be able to specify data of interest at a higher level and in a standard manner across systems, without having to install the monitoring software themselves or write low-level library calls. High-levels tools could significantly increase the user base of performance facilities. These facilities would also apply existing tools for knowledge discovery to performance data. The application of techniques such as decision trees to performance data has been initially explored [19, 14], but clearly significant additional research in this area is needed.

Performance Modeling

Item 5.8 in the current call emphasizes the need to develop improved methodologies for procuring high-end computer systems. As systems become ten times or more larger in memory and computing power than those in operation at the time of the acquisition, both the challenge of making informed procurement choices, and the penalty for mistakes, will be correspondingly greater. Novel architectures, distinct in design and technology from any existing systems (such as those being explored in DARPA's HPCS program), will compound this challenge.

The emerging technology of performance modeling holds the key to meeting these challenges. For example, accurate performance models for several full applications from the ASCI workload [8, 11, 12] are routinely utilized for system design, optimization and maintenance. Moreover, a similar model has been used in the procurement process for the ASCI Purple system, predicting the performance of the code SAGE on several the systems in a recent competition [9]. Alternative modeling strategies have been used to model the NAS Parallel Benchmarks, several small PETSc applications, and the applications POP (Parallel Ocean Program), NLOM (Navy Layered Ocean Model), and Cobal60, across multiple compute platforms (IBM Power 3 and Power 4 systems, a Compaq Alpha server, and a Cray T3E-600) [4, 17]. These models are extremely accurate across a range of processors (from 2 to 128), with errors ranging from 1% to 16%.

These results suggest that it is possible to accurately predict the performance achieved by a future system (much larger in size and employing a distinct design from hardware currently in operation) [13], running a future scientific application (much larger in problem size than currently being run). We can even envision that a future call for proposals for a system procurement will specify that the vendor run some small loops or other simple test code on the vendor's system simulator (or even on prototype hardware) and report the results, thus providing the required input data for performance models of key applications. Decision makers would have at their disposal not only performance information but also the capability to pursue "what if" scenarios. Other uses include improved system configuration and system maintenance [4, 8, 9, 11, 17].

Executable analytical performance evaluation also shows promise [11]. These techniques can evaluate early stage architecture designs over a wide operating range, and are thus helpful in identifying advantageous architectural features, before instruction set architectures and other features are firmly established, and before system software (runtime systems or compilers) is available. The methodology here is to model program execution through a program graph that models thread-level parallelism in the application. The program graph is executed on the architecture model, while the resulting analytical model is solved using a queuing network tool enriched with synchronization. This approach has been applied to evaluate the impact of "percolation," which was first proposed for HTMT, and is now being studied under DARPA HPCS funding.

Performance models can even be used within a user code to control the execution dynamically for best performance. Along this line, some researchers are considering using simple performance models to improve load balancing in unstructured grid applications. All of this underscores the need for a variety of performance modeling methodologies, ranging from simple, curve-fitting approaches to sophisticated tools that perform a thorough inventory of all operations performed by the target application program on a particular system. However, much work is required to further automate and reduce the complexity, "craftiness" and cost of the modeling work. In addition, more work is needed define a better interface between "traditional tools" (such as profilers, timers and hardware performance monitors) and modeling tools.

System Simulation

System simulation is another mechanism that could provide greater understanding of performance phenomena. At the recent High-Speed Computing Conference in Oregon, one speaker noted that although computational scientists have become highly skilled in simulating physical phenomenon, as yet they have not exploited this technology to understand the performance behavior of their applications. This indicates a "last mile" disconnect: a few system simulators are available in the research community [16], and vendors often develop cycle-accurate or near-cycle-accurate simulators as part of their product development, but computational scientists nonetheless rarely use such tools to understand or predict the performance of their applications.

Several challenges must be overcome for these simulators to be useful to application performance understanding. Perhaps most importantly, simulator execution times required to analyze performance for even a small loop are very large; the analysis of a full-length application code has been out of the question. Another common weakness of these simulators is that they typically target only single processor systems, or at best shared memory multiprocessor systems.

But with the emergence of highly parallel computing platforms, we can consider highly detailed parallel simulations of scalable systems. Low-level processor-memory behavior can be mostly decoupled from the analysis of inter-processor network phenomena. Then, once the communication behavior of an application has been profiled, one can simulate its inter-processor network behavior by generating a sequence of communication operations on each node, mimicking the statistics of frequency and message length typical of the program's phases.

Ideally, we envision an open-source architectural simulation framework and API that enables plug-and-play between separately-developed simulators for different architectural features (e.g., PIM, polymorphic multithreaded processor, and network), and would also enable zoom-out and zoom-in between statistically-based and cycle-accurate simulation techniques. This framework will, however, require significant advances in simulation methodologies in order to support concurrent use of modules running at different time-scales and based on different simulation techniques. For example, current architectural simulation engines tend to be time-stepped; but realistic models of scalable hardware and software are much too dynamic, asynchronous, and temporally sparse for that kind of synchronization. Instead, we anticipate that the simulations will be decomposed into logical processes, and will be synchronized by either conservative or optimistic methods (or both), as developed in the parallel discrete event simulation (PDES) community [7, 10]. With that approach, the high degree of real parallelism these systems exhibit will tend to translate to a similarly high degree of computational parallelism in the simulation as well.

Libraries, Compilers and Self-Tuning Software

It is not sufficient to merely study the performance of large future systems – facilities for automatic and/or semi-automatic performance tuning must also be improved. One approach here is to expand the scope of optimized scientific libraries for high performance computing. Three canonical examples are the ScaLAPACK, PETSc, and the NWChem libraries. Some related efforts include the emergence of the Community Climate Code (CCM) in the climate modeling community and similar efforts to unify fusion and accelerator modeling computations.

One of the more promising developments in this arena is the recent emergence of "self-tuning" library software. Examples include the FFTW library [6] and versions of ATLAS, ScaLAPACK, and LFC library routines [5]. The approach is to run, in an initialization step, a program that

tests a number of different computational strategies (such as different parameters for array padding or cache blocking). The tuning program then selects the option that demonstrates the best performance in the test run for future production runs. This general approach can be extended to almost any large-scale software library. However, the process of devising tests, determining optimal parameters and using the resulting parameters in the production code must be simplified if this general scheme is to be implemented widely. One possibility here is to combine rapid, on-the-fly performance modeling with such self-adaptive, self-tuning codes to narrow the parameter space for trying different computational strategies.

Eventually these self-tuning facilities can be incorporated directly into conventional user code. In other words, we foresee the time when self-tuning facilities will be understood well enough that they can be inserted by a preprocessor (and eventually perhaps by a compiler) directly into a user code at the start of the main program, or even at the subroutine level. Parallel processing can be utilized in a novel way here. The first iteration can be performed using different low-level data layout options on each processor. Then after the first iteration, the program uses the best performing choice on all nodes. This may seem futuristic, but in reality the basic facilities have already been demonstrated in current research (mostly in the PERC project), including self-tuning library software, performance assertions, compiler enhancements and semiautomatic code modifications [15]. In addition, the Active Harmony system (another PERC activity) has demonstrated the ability to automatically improve the performance of some large scientific application programs, including the POP ocean model code [18].

In this regard, it is instructive to recall the history of vector computing. Initially, compilers offered little or no assistance – it was necessary for programmers to explicitly vectorize loops. Then semi-automatic vectorizing compilers became available, which eventually were quite successful. The final step was run-time vectorization, with compilers generating both scalar and vector code, and then deciding at run time if the vector code is safe. We see a similar long-term potential for self-tuning code that makes use of performance monitoring. Other ideas for compiler technology that show promise include dynamic compilation and compile-time searching for optimal run-time alternatives, including array blocking, loop fusion and fission, flexible data layout and array padding. Since these changes in several cases go beyond the limits of what is permissible according to existing language standard definitions, this points to the need to work with language standard committees in tandem with this research.

Conclusion

Designing, deploying and programming the next generation of high-end computing platforms, which will feature tens or hundreds of thousands of processors, with new designs such as processor-in-memory or multi-threaded architectures, requires advanced tools (both hardware and software) to monitor, model and control performance. We believe that such facilities can be developed, although there are many questions that remain to be answered.

References:

[1] D. H. Ahn, J. S. Vetter, "Scalable Analysis Techniques for Microprocessor Performance Counter Metrics," *Proceedings of SC 2002*, IEEE, Nov. 2002.

[2] R. P. Bosch Jr., "Using Visualization to Understand the Behavior of Computer Systems," *Stanford University Ph.D. dissertation*, Aug. 2001.

[3] S. Browne, J. Dongarra, G. Ho, N. Garner, P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors", *International Journal of High Performance Computing Applications*, vol. (2000), pg. 189-204.

[4] L. Carrington, A. Snavely, N. Wolter, X. Gao, "A Performance Prediction Framework for Scientific Applications," *Workshop on Performance Modeling and Analysis - ICCS*, Melbourne, June 2003.

[5] J. Dongarra and V. Eijkhout, "Self Adapting Numerical Algorithm for Next Generation Applications", to appear in *International Journal of High Performance Applications and Supercomputing*, 2003.

[6] M. Frigo and S. Johnson, "FFTW: An Adaptive Software Architecture for the FFT", *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Seattle, WA, May 1998.

[7] R. Fujimoto, *Parallel and Distributed Simulation Systems*, Wiley Interscience, January, 2000.

[8] A. Hoisie, O. Lubeck, H. Wasserman, "Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Applications," *The International Journal of High Performance Computing Applications*, vol. 14, no. 4 (Winter 2000).

[9] A. Jacquet, V. Janot, R. Govindarajan, C. Leung, G. Gao, and T. Sterling, "An Executable Analytical Performance Evaluation Approach for Early Performance Prediction", *Proceedings of IPDPS'03*, 2003.

[10] D. Jefferson, B. Beckman, F. Wieland, L. Blume, M. DiLoreto, P. Hontalas, P. Laroche, K. Sturdevant, J. Tupman, V. Warren, J. Wedel, H. Younger, S. Bellenot, "Distributed Simulation and the Time Warp Operating System", *11th Symposium on Operating Systems Principles,* Austin, TX, Nov., 1987.

[11] D. J. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, M. Gittings, "Predictive Performance and Scalability Modeling of a Large-Scale Application", *Proceedings of SC2001*, IEEE, Nov. 2001.

[12] M. Mathis, D. Kerbyson, A. Hoisie, "A performance model of non-deterministic particle transport on large-scalesystems", *Workshop on Performance Modeling and Analysis - ICCS*, Melbourne, June 2003.

[13] D. J. Kerbyson, H. J. Wasserman, A. Hoisie, "Exploring Advanced Architectures using Performance Prediction," in *Innovative Architecture for Future Generation High-Performance Processors and Systems*, IEEE Computer Society Press, 2002, pg. 27-37.

[14] B. P. Miller, M. D. Callaghan, J. Cargille, J. K. Hollingsworth, R. B. Irbin, K. Karavanic, K. Kunchithapadam, T. Newhall, "The Paradyn Parallel Performance Measurement Tools," *IEEE Computer*, vol. 28 (1995), no. 11, pg. 37-46.

[15] D. Quinlan, M. Schordan, B. Philip and Kowarschik, M., "Parallel Object-Oriented Framework Optimization," to appear in *Special Issue of Concurrency: Practice and Experience*, 2003.

[16] M. Rosenblum, "SimOS," available at http://simos.stanford.edu.

[17] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia A. Purkayastha, "A Framework for Performance Modeling and Prediction," *Proceedings of SC2002*, IEEE, Nov. 2002.

[18] Tapus, C., I.-H. Chung, and J.K. Hollingsworth, "Active Harmony: Towards Automated Performance Tuning," in *Proceedings of SC2002*, IEEE, Nov. 2002.

[19] J. S. Vetter, "Performance Analysis of Distributed Applications using Automatic Classification of Communication Inefficiencies," Proceedings *of the ACM International Conference on Supercomputing (ICS)*, ACM Press, 2000.

[20] J. S. Vetter, P. Worley, "Asserting Performance Expectations," *Proceedings of SC2002*, IEEE, Nov. 2002.

[21] M. Wang, T. Madhyastha, N. H. Chan, S. Papadimitriou, C. Faloutsos, "Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffic, *International Conference on Data Engineering*, 2001.

Contact Information:

David H. Bailey
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
Email: dhbailey@lbl.gov
Tel: 510-495-2773