

Scaling to New Heights Workshop, PSC, May 2002: A Summary

Motivation

The goal of supercomputers dedicated to science is to enable the solution of critical scientific and engineering problems which would otherwise be intractable. In a previous era, supercomputers were characterized by special purpose processors, whose performance greatly exceeded that of commodity processors. Today, supercomputers are largely built on commodity processors, and their strength comes from aggregating a large number of processors, and linking them with high-bandwidth, low-latency networks. We understand well how these computers can be used as capacity engines. The issue we wished to investigate was their use as capability engines- what can be achieved by having these resources co-located, and closely coupled as opposed to loosely coupled? Exploiting this capability means developing codes that scale efficiently to very large processor count. Today, the leading supercomputers have thousands of processors. New and alternative architectures are emerging such as “processor-in-memory” and “embedded computing” architectures. Industry is making large investments in the adaptation of these and other new technologies to high performance scientific computing. These new computers are shortly expected to have 10,000 to 100,000 processors.

The goal of this workshop was to examine practical mechanisms for scaling applications for closely coupled high-end distributed-memory architectures. For example, in this context we were not interested in parameter search problems, which can easily be distributed over many loosely coupled machines.

Sample issues

Workshop organizers identified a set of sample issues that they invited participants to address. However, participants were encouraged to identify and discuss other issues as well. The sample issues included:

- What are the characteristics of applications that scale well, in computation, data, and I/O? It is permissible to scale the problem size as the number of processors increases.
- What are the resource needs of emerging applications (e.g., what is the balance of computation to I/O, say for data base access or data mining) relative to “traditional” floating point intensive applications.
- What are indicators of bad scaling (e.g. writing to single files, collective operations, all-to-all communications)?
- What tools exist to help users achieve better scaling (e.g. Vampir)? Note, we are not interested here in single processor optimization, except insofar as that may impact scaling.
- What tricks exist to achieve better scaling (e.g. issue receives before sends, asynchronous communications, writing one file per PE, prefetching)?
- What is the status for parallel math and I/O libraries that already exist or are being developed? What is expected to happen with these in the coming months and years?
- What are the prospects for automating better scaling, as one automated vectorization, and some parallelization?

- What types of fault tolerance and checkpoint/restart strategies have been implemented in applications codes?
- From the viewpoint of the users/researchers, what are the systems software requirements for large systems?

Workshop Participants

Sponsored by the NSF and DOE's Office of Science, and organized by their leading-edge centers (NCSA, PSC, SDSC, ORNL and NERSC), the workshop included invited and contributed talks, a poster session, and a panel. Input was sought from tool developers, people with enlightening case histories, and others with insights into techniques which will make scaling easier and more effective. Ninety-one participants from universities, research centers, and corporations around the country attended the workshop, May 20 and 21, at PSC.

Invited speakers were selected on the basis of their known achievements in this field. Some effort was made to balance presentations by universities, industry, and national laboratories. There were 6 invited speakers and 7 submitted papers selected for presentation during the meeting (see below).

The Program

The invited speakers and their topics were:

- Laxmikant Kale, University of Illinois: *An Object Based Approach to Developing Scalable Applications: Challenges and Techniques*
- William Shelton, Oak Ridge National Laboratory: *Scaling of First Principles Electronic Structure Methods on Future Architectures*
- Jim Demmel, UC Berkeley: *Scaling in Numerical Linear Algebra*
- Paul Woodward, University of Minnesota: *Scaling to the TeraGrid by Latency Tolerant Application Design*
- Manish Gupta, IBM T.J.Watson Research Center: *Challenges in Developing Scalable Software for BlueGene/L*
- David E. Keyes, Old Dominion University: *Understanding the Parallel Scalability of An Implicit Unstructured Mesh CFD Code*

The accepted submitted papers were:

- Susanne M. Balle, Compaq Computer Corporation: *A New Approach to Parallel Debugger Architecture*
- Laura C. Carrington, San Diego Supercomputer Center: *A Framework for Application Performance Prediction to Enable Scalability Understanding*
- Alan Heirich, Compaq Computer Corporation: *Scalability in Large-Data Scientific Visualization*
- Fabrizio Petrini, Los Alamos National Laboratory: *Scaling to Thousands of Processors with Buffered Coscheduling*
- James C. Phillips, University of Illinois at Urbana-Champaign: *NAMD: Biomolecular Simulation on Thousands of Processors*

- Henry Tufo, Argonne National Laboratory: *High Performance Spectral Element Methods for Simulation of Transition in Vascular Flows*
- Theresa Windus, Pacific Northwest National Laboratory: *NWChem: Computational Chemistry for Large Numbers of Processors*

Furthermore, the following contributions were presented as Posters.

- *The Advanced Computational Testing and Simulation Toolkit (ACTS): What can ACTS do for you?* Leroy A. Drummond and Osni A. Marques, Lawrence Berkeley National Laboratory
- *Highly Scalable Inversion:* Omar Ghattas, Carnegie Mellon University and John Urbanic, Pittsburgh Supercomputing Center
- *Performance Monitoring Tools on the TCS:* Roberto O. Gomez and Raghu Reddy, Pittsburgh Supercomputing Center
- *The NEOSIM Neural Simulation Kernel:* Greg Hood, Pittsburgh Supercomputing Center and Nigel Goddard and Fred Howell, University of Edinburgh
- *Scalability Issues for Software Tools,* Padmanabhan Iyer, Etnus
- *Architecture, Algorithms and Applications: NNSA's Research Agenda for BlueGene/L:* Robert K. Yates, Jeffrey S. Vetter, Mark K. Seager and Lynn Kissel, Lawrence Livermore National Laboratory
- *Hidden Cost of Memory Management in Asynchronous Communication:* Joel M. Malard and R. D. Stewart, Pacific Northwest National Laboratory
- *PAPI: Performance Application Programming Interface:* Shirley V. Moore and Dan Terpstra, Innovative Computing Laboratory, University of Tennessee
- *Strategies for Scaling Parallel I/O on GPFS:* David Skinner, NERSC/Berkeley Lab
- *Scalability of FETI/Salinas on ASCI Machines:* K. H. Pierson, M. K. Bhardwaj, G. M. Reese, D. M. Day, and T. F. Walsh, Sandia National Laboratories
- *Fluid Flow on a Computer: Scaling in High-Reynolds-Number Turbulence:* P. K. Yeung, Georgia Institute of Technology and K. R. Sreenivasan, University of Maryland
- *Scalable Computing at the Cellular Level:* Jung-Hsing Lin, Nathan A. Baker, and J. Andrew McCammon, San Diego Supercomputer Center

Finally, a plenary Panel Session led by Bill Camp of Sandia National Lab wrapped up the meeting. The panelists were Laxmikant Kale (UIUC), Sergiu Sanielevici (PSC), John Towns (NCSA) and Paul Woodward (University of Minnesota).

Approaches Discussed

The talks probed a wide variety of issues for scientific application developers, utility software developers, system architects and system operators.

Dr. Kale presented an object-based methodology of seeking the optimal division of labor between the programmer and a standard library of reusable parallel components. The programmer is responsible for dividing the computation into a large number of pieces, independent of the number of processors and typically *larger* than the number of processors. The standard library *charm++*, developed by Dr. Kale's group, will then automatically map the computational tasks onto the processors, using techniques such as measurement based load balancing. This exploits such ideas as prioritizing remaining tasks, so that one executes those most likely to be needed (avoiding those not likely to be

needed in speculative execution; overlapping communication and computation; aggregating small messages to reduce latencies. The successes and challenges of this approach were illustrated by a variety of applications to molecular dynamics, rocket simulation, finite element modeling, and particle collision detection codes. Performance on existing machines such as the TCS was discussed, as well as the emulation of future machines such as IBM Blue Gene/G.

Dr. Shelton discussed how a new algorithm for first-principles electron level simulation of complex materials was designed from the ground up (physics approach, algorithm, implementation) to achieve linear scalability while relying on linear algebra routines that are always well optimized by system vendors. The resulting LSMS (locally self-consistent multiple scattering) code was the first full application code to sustain more than 1 Teraflop (1998 Gordon Bell prize winner) and now sustains 4.6 Teraflops on 3,000 processors of TCS.

Dr. Demmel focused on the fundamental building blocks of numerical scientific application codes: linear algebra routines. He discussed fundamental research in numerical mathematics and in computer science, aimed at making these routines scale efficiently to very large processor counts. Specific topics included ScaLAPACK, a parallel distributed dense linear algebra library; sparse direct and iterative solvers; automatic performance tuning of numerical kernels using table lookup on previously run examples to decide on memory access patterns and blocking sizes; and multigrid methods on irregular meshes.

Dr. Woodward addressed the issues of programming applications that can be made latency tolerant, efficiently utilizing the Teragrid as a capability system with thousands of processors. He observed that most current algorithms depend on low latency memory and networks, which would not run well on Grids and Clusters of inexpensive machines. The new approach he proposes, called *SHMOD/SHMON (shared memory on disk/network)*, eliminates latency dependence by decomposing the problem domain into *Grid Bricks*, each with its own data context. The computation is thus decomposed into independent Grid Brick update tasks, each of which can execute to completion as soon as its data context is ready. Tasks are ordered so that there are no barriers and a new task is always ready to fire. Like Kale, he emphasized having many more tasks than processors, and doing speculative execution. Applications to 1024^3 and 2048^3 PPM turbulence runs with real-time visualization were presented.

Dr. Gupta presented the system architecture of the future IBM Blue Gene/L system, which will be capable of delivering 360 Teraflops using 65,536 processors. Several programming models for the dual-CPU compute nodes will be supported: in *communication coprocessor mode* one CPU will be responsible for computing and the other for communications, which is good for communication-intensive applications. In *virtual node mode*, compute-intensive applications can use both CPUs to run MPI processes. The communication infrastructure is layered, supporting MPI as well as the development of new application-level communication libraries. The compiler designed for BG/L is capable of such feats as finding parallel operations that match SIMD

instructions in the user's Fortran, C or C++ code. System self-optimization and self-healing features were presented, utilizing a dynamic database of correctable errors as indicators of problematic components, along with support for very high performance I/O and application emulator performance results. Having a reduced kernel for the many compute nodes will minimize problems associated with unwanted O/S activity.

Dr. Keyes discussed the case history of the PETSc-FUN3D toolkit for solving partial differential equations (Gordon Bell Special Category winner, 1999). This effort applied principles such as the "owner compute" rule under the dual constraints of minimizing the number of messages and overlapping communications with computation, creating gather/scatter operations based on runtime connectivity patterns, and using profiling data to optimize performance in communication and memory hierarchy. Keyes stressed the importance of understanding one's algorithm in terms of many parameters such as memory bandwidth, internode latency, and network diameter. While processor scalability may not be a problem in principle (assuming the interprocessor network is scalable in hardware and protocol) synchronization is always a bottleneck. Memory latency is not a problem, provided enough bandwidth is available to cover it; but memory *bandwidth* is a major bottleneck. Load-store units may be a bottleneck if applications are not properly load balanced, and low frequency of floating point instructions is a bottleneck intrinsic to unstructured problems. Dr. Keyes went on to discuss the application of these insights to his current DOE SciDAC project, *Terascale Optimal PDE Simulations (TOPS)*.

The contributed talks presented case studies in application engineering (NAMD molecular dynamics, NWChem quantum chemistry, spectral CFD, large-scale distributed computer graphics) and tool development (large-scale parallel debugger, application performance prediction framework). The Posters illustrated the issues and topics summarized above based on a wide range of efforts and projects.

Lessons Learned

The process of extracting the ideas most likely to guide the computational science and engineering community's effort to achieve scalability began with the workshop's concluding Panel session, and continued when PSC, NCSA and SDSC organized a Birds of a Feather session on this topic at SC2002 in Baltimore (November 2002).

We concluded that the important principles are as follows:

1. **All** application components must scale
2. Control granularity; Virtualize
3. Incorporate latency tolerance
4. Reduce dependency on synchronization
5. Maintain per-process load; Facilitate balance
6. Use good algorithms

These, of course, have always been the principles of designing efficient parallel applications; their importance merely becomes much more important as we enter the domain of thousands of processors.

The work of participants such as Kale, Woodward and Keyes stresses the importance of controlling granularity. One should define a problem in terms of a large number of small objects independent of the processor count. This "virtualization" also facilitates static and dynamic load balancing.

In general it was felt that latency was not as limiting as bandwidth. While there are many latency hiding techniques (e.g. prefetching for regular algorithms), very few techniques for overcoming small bandwidth are available. This is also true for processor/memory bandwidth. To reduce the impact of network and memory latency, one should overlap communication and computation, and pipeline larger messages. As Woodward points out, the maxim "*Don't wait, speculate!*" applies to application design just as it does to high-performance processor design. On the other hand, many latency hiding techniques increase the amount of computation carried out (e.g. unnecessary speculative execution, or increased writes).

It can be proven that the cost of synchronization increases with processor count, so that truly scalable codes must be *asynchronous*. This is not easy to reconcile with the demands of regular communication patterns, but the negative effects of synchronization are especially apparent in heterogeneous systems such as those made available on a Grid. One can postpone the transition to asynchronous algorithms if one uses homogenous co-located clusters.

The programmer should not try to pre-specify the load balancing. Dynamic process management (load balancing) requires distributed monitoring to be provided by the system, along with mechanisms to feed load measurements back to the runtime application code. The run-time system should also be able to combine many messages into one (for example, if multiple objects on one processor are each sending messages to objects on another processors). Conversely, sometimes messages are too long, and

computation on the receiving processor could begin as soon as the first part of the message was received.

It is important that scaling efficiency never be obtained at the expense of per-processor performance. Optimization must begin with a strong effort to use each processor as efficiently as possible, exploiting its memory architecture and other features, while always using the efficient algorithms. Improvements in flop count associated with less efficient algorithms are seldom worthwhile. Improving data layout to increase locality can have significant impact. The computational problem should always scale with the processor count, so that each processor always has plenty of work to do. Camp pointed out that as algorithms scale, they become less well conditioned, requiring a larger number of iterations, which defeats scaling.

A number of tools are being developed to assist in these efforts, including automated algorithm selection and optimization (such as ATLAS from the Dongarra group and BeBoP from the Demmel group) and performance prediction frameworks such as PmaC from SDSC. On the other hand, it was generally perceived that performance monitoring and debugging tools were still not up to the task of dealing with codes using thousands of processors. Compaq/HP indicated some steps for improving scaling and reducing information overload presented to the user by systems originally targeted at a few processors, but much work remains. Moreover, we need to develop applications frameworks so that computing on such systems is feasible for a wide set of users, and not just the 'hero' programmers.

Regarding new machine architectures, several points were made. Processor/memory bandwidth has not kept up with increased processing speed. Next generation processors should focus on increasing memory bandwidth. Moreover, on systems with thousands of processors, a reduced operating system on compute processors would improve performance and predictability. This is being planned for Blue Gene/L. Since the probability of component failure grows with processor count, one will need more robust ways to design algorithms and detect errors. Standard checkpointing will not suffice. One will need internal consistency checks to trap undetectable two-bit errors; one will need proactive ways to anticipate component failure.

Dissemination

The Proceedings of the workshop were immediately made available online, at <http://www.psc.edu/training/scaling/index.html>. The presentation of lessons learned, given at the SC2002 BOF in November, is also available via this URL.

Organizers

Ralph Roskies, PSC
John Towns, Dan Reed NCSA
Nancy Wilkinson, Fran Berman SDSC
Horst Simon NERSC
Tom Zacharia ORNL

