

# Measuring High-Performance Computing with Real Applications

*The computer platforms the authors describe here performed both the best and the worst in a test of selected applications. “Best performance” significantly depended on the problem being solved, calling into question the value of computer rankings that use simplistic metrics.*

**B**enchmarking can help us quantify the ability of high-performance computing (HPC) systems to perform certain, known tasks. An obvious use of the results is to find a given architecture’s suitability for a selection of computer applications. Other equally important uses include the creation of yardsticks to assess the progress and requirements of future HPC technology. Such measurements must be based on real computer applications and a consistent benchmarking process. Although metrics that measure simple program kernel behavior have worked well in the past, they fail to answer questions that deal with the true state of the art of today’s HPC technology and the directions future HPC development should take. These answers have a direct impact on competitiveness.

To quantify this belief, we compared kernel benchmarks with real application benchmarks. We used high-performance Linpack (HPL) to represent kernel benchmarks; researchers often

use HPL in highly visible projects, such as those ranked in the top 500 supercomputers ([www.top500.org](http://www.top500.org)). For application benchmarks, we selected four computational codes represented in the Standard Performance Evaluation Corporation’s SPEC HPC2002 and SPEC MPI2007 suites ([www.spec.org/hpg](http://www.spec.org/hpg)) and in the US National Science Foundation (NSF) HPC system acquisitions process.

Although we advocate the use of real application benchmarks, it’s important to note that kernel benchmarks have an essential role—a small code fragment that can time a single message exchange, for example, is most appropriate for measuring a message-passing function’s latency. We don’t argue here that small benchmarks are unimportant, but that people often use them to answer the wrong questions. Kernels are appropriate for measuring system components, but observations of real application behavior are essential for assessing HPC technology.

## Key Benchmarking Questions

Table 1 lists some important questions and indicates the ability of either kernel or full application benchmarks to provide answers. The first question considers the time required to solve today’s important problems. We might learn that a biology application takes a full year to fold a specific

1521-9615/08/\$25.00 © 2008 IEEE  
Copublished by the IEEE CS and the AIP

MOHAMED SAYEED, HANSANG BAE, YILI ZHENG,  
BRIAN ARMSTRONG, RUDOLF EIGENMANN, AND FAISAL SAIED  
*Purdue University*

protein on a present-day HPC platform or that forecasting the weather with a certain resolution and accuracy takes 10 hours. For obvious reasons, kernel benchmarks don't address such questions: we can't infer solution times for real problems from kernel execution times. In contrast, real application benchmarks with data sets that represent actual problems give direct answers.

The second question deals with relative performance. Although both types of benchmark provide answers, a system's overall ranking can hinge on the type of application involved. The same machine can perform the best and the worst, so relying on simple numbers not only provides inaccurate answers, it can lead to the wrong conclusions about machine suitability and HPC technology in general.

The strength of specialized kernel benchmarks is their ability to measure individual machine features by focusing on a particular system aspect. However, the results don't answer the question of relative component importance—breaking down a real application's execution time into computation, communication, and I/O, for example, shows each component's true relative importance. We can't get this type of information by combining each component's individual kernel benchmarks. However, kernel metrics *can* answer the question of what percentage of peak performance the floating-point operations can achieve: such metrics are idealized bounds under a given code pattern. But these results don't represent the percentage of peak performance a real application can achieve.

The sixth question deals with an important group of issues that benchmarking and performance evaluation must address: the characteristics of computational applications. Kernel benchmarks can't help here—they're typically created by identifying real application characteristics and then focusing on one of them.

Benchmarking can help predict the application behavior of data sets and machines that are much larger than what's feasible today. As the last question illustrates, the value of kernel versus real application benchmarks for such prediction is still controversial.

### Challenges of Measuring Performance with Real Applications

Communities ranging from HPC customers to computer manufacturers to research funding agencies to scientific teams have increasingly called for better yardsticks. Let's look at the stark contrast between this need and conventional wisdom.

**Table 1. Comparison of kernel versus full application metrics.\***

Benchmarking question	Ability to answer	
	Kernels	Full applications
1. What time is required to solve important computational problems on today's high-performance computing platforms?	n/a	+
2. What's the overall platform performance?	-	+
3. How do system components perform?	+	-
4. What's the importance of system components relative to each other?	n/a	+
5. What's the importance of system components relative to upper bounds?	-	+
6. What are the characteristics of important computational problems?	-	+
7. What are the characteristics of important future problems?	-	+

\* "+" is a good answer, "-" is a limited answer, and "n/a" is no answer.

### Simple Benchmarks Are Overly Easy to Run

One of the greatest challenges of evaluating performance with real applications is the simplicity with which we can run kernel benchmarks. With the investment of minutes to a few hours, we can generate a benchmark report that seemingly puts our machine "on the map." The simplicity of kernel benchmarks can overcome portability issues (no changes to the source code are necessary to port it to a new machine) as well as software environment issues (the small code is unlikely to engage unproven compiler optimizations or to break programming tools in a beta release). This simplicity might even overcome hardware issues—the kernel is unlikely to approach numerical instability, to which a new processor's floating-point unit might be susceptible.

However, these are the very features that we want in a true HPC evaluation. We don't want a machine to show up in the "Best 100" if it takes major code restructuring to port a real application, if porting the application requires major additional debugging, if the tools and compilers aren't yet mature, or if the hardware is still unstable.

Specialized benchmarks exist for a large number of metrics, including message counting in message-passing interface (MPI) applications, measuring the memory bandwidth of symmetric multiprocessor (SMP) platforms, gauging fork-join overheads in OpenMP implementations, studying scheduling characteristics,<sup>1</sup> and many more. The SPEC benchmarking organization ([www.spec.org](http://www.spec.org)) alone distributes 12 major bench-

mark suites. Diverse suites play an essential role because they can help us understand a specific aspect of a system in depth, but when it comes to understanding the behavior of a system as a whole, these detailed measures don't suffice. Even if a specific benchmark could measure each and every aspect of a system, no formula exists to help us combine the different numbers into an overall system metric.

### **We Can't Abstract**

#### **Realistic Benchmarks from Real Applications**

An inviting approach to benchmarking with real applications is to extract important excerpts from real codes, with the aim of preserving the relevant features and omitting the unnecessary ones. Unfortunately, this means we're making decisions about an application's less important parts that might be incorrect. For example, we might find a loop that executes 100 computationally near-identical iterations and reduce it to just a few, but this abstraction might render the code use-

*Large, real applications tend to contain legacy code with programming practices that don't reflect tomorrow's software engineering principles and algorithms.*

---

less for evaluating adaptive compiler techniques: repetitive behavior can be crucial for an adaptive algorithm to converge on the best optimization technique. Similarly, data downsizing techniques<sup>2</sup> might ensure that a smaller problem's cache behavior remains the same, but the code would become useless for determining the real problem's memory footprint.

The difficulty of defining scaled-down benchmarks is evident in the many criteria for benchmark selection suggested in past efforts: codes should contain a balance of various degrees of parallelism, be scalable, be simple yet reflect real-world problems, use a large memory footprint and a large working set, be executable on a large number of processors, exhibit a variety of cache-hit ratios, and be amenable to a variety of programming models for shared-memory, multicore, and cluster architectures. Last but not least, benchmark codes should represent a balanced set of computer applications from diverse fields.

Obviously, nothing could satisfy all of these demands—some of them are directly contradic-

tory. Furthermore, selecting yardsticks by such criteria would dismiss the fact that we want to learn about these properties from real applications. We want to learn how scalable a real application is, for example, not just select a scalable one. Similarly, if an application's real data sets don't have large memory footprints, then our evaluation has produced an important result: inflating input data parameters to fill some memory footprint benchmark selection criterion wouldn't be meaningful.

### **Today's Real Applications**

#### **Might Not Be Tomorrow's**

Large, real applications tend to contain legacy code with programming practices that don't reflect tomorrow's software engineering principles and algorithms.

This is perhaps the strongest argument against using today's real applications to determine future HPC needs, but when we ask what future applications should include, the answer isn't forthcoming. Should we use specific selection criteria? Are we sure that the best of today's algorithms and software engineering principles will find themselves in tomorrow's applications? If we choose a certain path and miss, we risk losing on two fronts: abandoning today's established practices and erring in what tomorrow's technology will be. Nevertheless, the best predictor of tomorrow could very well be today's established practice.

### **Benchmarking Isn't Eligible for Research Funding**

Performance evaluation and benchmarking projects are long-term efforts, so data must be gathered and kept for many years. (The 15-year-old [perfect-benchmarks@csrd.uiuc.edu](mailto:perfect-benchmarks@csrd.uiuc.edu) mailing list still receives occasional queries.) Unfortunately, this type of task doesn't easily include the advanced performance-modeling topics that interest scientists, so most efforts that focus on benchmarking infrastructure and its long-term support lack continuity in funding from science sponsors.

Programs that create such services at funding agencies could eventually appear—perhaps this article will motivate future initiatives. An alternative is a combined academic/industrial effort, such as SPEC's High-Performance Group (HPG), in which industrial benchmarking needs meet academic interests, resulting in suites of real, HPC applications.

### **Maintaining Benchmarking Efforts Is Costly**

A performance evaluation and benchmarking ef-

fort entails collecting test applications, ensuring portability, developing self-validation procedures, defining benchmark workloads, creating benchmark run and result submission rules, organizing result review committees, disseminating the suite, maintaining result publication sites, and even protecting evaluation metrics from misuse. Dealing with real, large-scale applications further necessitates assisting benchmark users and maintaining the involvement of domain experts in their respective application areas.

The high cost of these tasks is obvious. Many benchmarking efforts cover their costs through initial research grants or volunteer efforts, but this support typically pays for the first round of benchmark definitions, not subsequent steps or publication sites. To date, SPEC is the only organization that maintains full, continuously updated benchmarking efforts. Its funding comes primarily from industrial membership and a comparably small fee for the actual benchmark suites.

#### **Proprietary Application Benchmarks Can't Serve as Open Yardsticks**

So, should realistic benchmarks match the exact applications that will run on the target system of interest? Clearly, prospective customers think their own applications would be the best choice for testing a system's desired functionality. It might serve the customer best if multiple vendors ran applications in a way that allowed fair comparison, but because most applications are proprietary, neither the scientific community nor the public can verify or scrutinize the generated performance claims. Hence, the value of proprietary benchmarks is limited to simple metrics.

Public benchmarks might be of higher value. Fair benchmark results can be costly to generate, and vendors are under pressure to produce good results in a very short period, often competing internally for machine resources. Unless customers can closely supervise the benchmarking process with significant expertise, they might not get results that they can compare fairly. Without any deceptive intentions on the vendor side, the lack of a clear evaluation methodology often allows shortcuts and "optimizations" that can differ significantly among evaluation groups. In contrast, even though established, public, full-application benchmark results might not have computational patterns identical to a customer's codes, the performance numbers' consistency and availability might outweigh this drawback.

#### **Meeting the Challenge:**

##### **Toward a Benchmarking Methodology**

An advanced HPC benchmarking methodology

- creates performance yardsticks based on real applications and openly shared data sets,
- defines metrics that indicate overall problem solution time as well as the performance of important kernels that constitute the application,
- defines rules for running and reporting benchmarks, so that comparisons are fair and any relevant information is fully disclosed, and
- enables the creation of a repository that maintains benchmarking reports over a long time period.

Rules are only useful if they're enforceable, which emphasizes the need for a benchmark review process. The goal here is to facilitate objective efforts rather than self-evaluations by machine vendors and HPC platform owners. Full disclosure is also crucial: the applications, their data sets, the benchmarking process, and the software and hardware configuration with which the results were obtained must all be open to inspection.

An open repository of fully disclosed benchmark results is essential to fair, consistent HPC evaluation. Such an effort benefits prospective customers of HPC systems as well as researchers attempting to advance HPC technology. It's even more important in the acquisition and evaluation of HPC systems with public funds.

#### **Benchmarking Efforts with Real Applications**

Performance evaluation efforts with real applications began to emerge in 1988 with the Perfect Benchmarks.<sup>3,4</sup> This set of scientific and engineering applications was intended to replace kernel-type benchmarks and the commonly used peak-performance numbers, and represented a significant step in the direction of application-level benchmarking. Although they continue to circulate in the research community, the original Perfect Benchmarks are small compared to today's real applications (the largest included some 20,000 lines of FORTRAN 77), and their data sets execute in the order of seconds on today's machines. No results Web site is available for the Perfect Benchmarks.

Similarly, the ParkBench (for PARallel Kernels and BENCHmarks)<sup>5</sup> effort emerged in 1992 with some research funding but didn't update its initial suites in response to newer generations of HPC

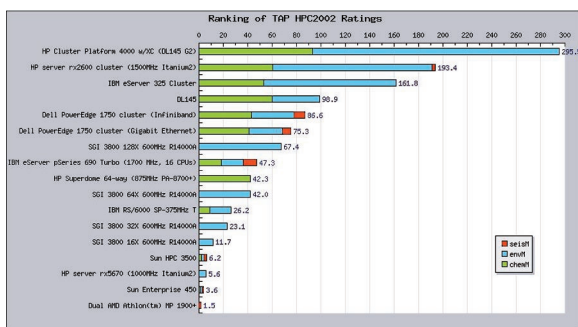


Figure 1. Top application performers. This aggregate view combines the SPEC HPC2002 suite's results into an overall rank, with the bars subdivided into each individual benchmark application's contributions to the rank. The TAP lists also show rankings based on other application suites.

systems. The effort was very ambitious in its goal of delivering a set of benchmarks that range from kernels to full applications. The largest, full-application suite was never created, though.

### SPEC HPG

Like the Perfect Benchmarks, SPEC also debuted in 1988. Although it's largely vendor based, the organization includes a range of academic affiliates. Initially, SPEC focused on benchmarking uniprocessor machines and, in this area, became the leader in providing performance numbers to workstation customers. SPEC suites are also increasingly used by the research community to evaluate the performance of new architecture concepts and software prototypes. Today, SPEC offers a large number of benchmarks that evaluate workstations, graphics capabilities, and high-performance systems. Most notably for HPC evaluation, the SPEC HPG formed in 1994 out of an initiative to merge the Perfect Benchmarks effort's expertise in HPC evaluation with SPEC's capability to sustain such efforts long term. Since its foundation, this group has produced several HPC benchmarks, including the HPC suite,<sup>6-8</sup> the OMP suite (for OpenMP applications),<sup>9</sup> and the MPI suite.

SPEC's HPG suites are based on widely used computational applications that can be openly distributed to the community. The codes are implemented with the MPI and OpenMP standards for parallel processing, and SPEC provides a result submission and review process, a repository at [www.spec.org/hpg](http://www.spec.org/hpg), and a continuous benchmark update process. Intended consumers include end users, system vendors, software vendors, and researchers.

### Other HPC Benchmarking Efforts

Other attempts to provide HPC benchmarks have emerged over the past decade as well. Notable examples include the Euroben effort in Europe ([www.euroben.nl](http://www.euroben.nl)) and NASA's Parallel Benchmarks (NPB).<sup>10</sup>

The US Department of Defense's High Performance Computing Modernization Program (HPCMP; [www.hpcmo.hpc.mil](http://www.hpcmo.hpc.mil)) developed its own suites that include synthetic benchmarks and real applications to support its yearly acquisition activities. A related effort is the benchmarking project that's part of Darpa's High-Productivity Computing Systems (HPCS) program ([www.highproductivity.org](http://www.highproductivity.org)). Currently, this effort has suites of kernel benchmarks and several synthetic compact applications. Another important effort is the benchmarking process defined by the NSF for its acquisition of a national HPC system infrastructure. The NSF benchmarks include both a set of kernel benchmarks and real applications for system evaluation. However, neither the HPCS nor the NSF benchmarking efforts have an associated project to maintain a result repository that the public can view.

### Metrics for HPC Evaluation

The general consensus in the benchmarking community is that overall performance must be evaluated via wall-clock time measurements, but an open and often controversial issue is how to combine measurements from multiple benchmarks into one final number. SPEC's approach is to leave the decision up to the benchmark report reader—that is, each code is reported separately. Other suites, such as SPEC CPU, SPEC OMP, and SPEC MPI, report the geometric mean of individual program performance results. The NSF benchmarking effort includes both kernel and applications metrics, which are reported independently. Kernel benchmarks evaluate a wide variety of system components, so the metrics also vary widely, and to our knowledge, no current benchmarking effort can relate kernel and application benchmarks by, for example, measuring important constituent kernels as part of a real application benchmark run.

An increasingly important class of metrics characterizes the program properties of computational applications (examples include the use and frequency of algorithms, program patterns, and compiler analysis results). Understanding such benchmarking metrics is key to improving the scalability of computational applications.

### Tools for Gathering Metrics

Obtaining overall timing metrics is relatively

**Table 2. Wall-clock execution times and problems solved by application benchmarks on the IBM P690 platform, using 32 processors.**

Application	Execution time	Problem description for a medium data set
Seismic (SPECseis)	625 sec.	Seismic is a suite of codes typical of the seismic processing applications used to find oil and gas. The data set processes seismic traces of $512 \times 48 \times 128 \times 128$ . (Samples per trace $\times$ traces per group $\times$ groups per line $\times$ number of lines, where a trace corresponds to a sensor that has a sampling frequency; the sensors are strung out on multiple cables behind a ship.) The total data set size in Phase 1 of SPECseis is 1.5 Gbytes and reduces to 70 Mbytes in Phase 2.
GAMESS (SPECchem)	3,849 sec.	GAMESS is a general ab initio quantum chemistry package. The data set computes self-consistent field wavefunctions (RHF type) for thymine (C <sub>5</sub> H <sub>7</sub> N <sub>3</sub> O – 16 atoms), one of the bases found in nucleic acids.
WRF (SPECenv)	742 sec.	WRF is a weather research and forecasting modeling system for the mesoscale (meters to thousands of kilometers). The data set simulates the weather over the continental US for a 24-hour period starting from Saturday, 3 November 2001, at 12:00 a.m. The grid is $260 \times 164 \times 35$ with a 22-km resolution.
MILC	1,497 sec.	MILC is used for large-scale numerical simulation of quantum chromodynamics to calculate the masses and other basic properties of strongly interacting particles (quarks and gluons). It simulates quantum chromodynamics with improved staggered quarks of two masses and performs a computation over a 4D lattice with 32 points, involving approximately 34 million variables for the integral evaluation.

straightforward, but tools for gathering detailed execution characteristics are often platform-specific, so it can be difficult to obtain the metrics of interest on a given platform. It's even more difficult to conduct comparative evaluations that gather a certain metric across several platforms. Among the tools we've used in our projects are mpiP, hpmcount, and strace:

- mpiP is a lightweight profiling library for MPI applications that reports the percentage of time spent in the MPI. It also includes the times each MPI function uses.
- IBM's Hardware Performance Monitor suite includes a simplified interface, hpmcount, which summarizes data from selected hardware counters and computes some useful derived metrics, such as instructions and flops per cycle.
- We can measure I/O behavior (by recording I/O system call activities) with the strace command; from this output, we can extract statistics for file I/O using a script.

These tools are but a small sample of a large set of instruments available on myriad platforms today, thus an important goal is uniformity. Benchmarkers need tools and interfaces to gather relevant performance data consistently across the range of available platforms. Ideally, these tools won't just report volumes of performance counter results—they'll also abstract these volumes, thereby creating the end metrics of interest.

### Ranking HPC Systems

Benchmark reports for real application suites, such as the SPEC HPC codes, can help us create more realistic rankings of HPC platforms ([www.purdue.edu/TAPlist](http://www.purdue.edu/TAPlist)). Figure 1 shows the top application performers (TAP) list as of January 2006; this list defines an aggregate metric with which we can combine the results of our three benchmark applications into an overall rank. The metric weights individual application performance results according to their average runtime across different platforms. The list also allows a single benchmark to be used for ranking. The TAP list contains links to the original SPEC benchmark reports.

### Performance Results

We used four application benchmarks and tested them on three different parallel architectures to evaluate performance; we discuss the performance results in this section in terms of the key benchmarking questions discussed earlier. Table 2 gives a brief description of the four applications along with the problem being solved. The codes GAMESS and WRF are members of the SPEC HPC and NSF benchmark suites (see [www.nsf.gov/pubs/2006/nsf0605/nsf0605.jsp](http://www.nsf.gov/pubs/2006/nsf0605/nsf0605.jsp)); WRF and MILC are also part of SPEC MPI2007. Here, we've measured the SPEC HPC versions of GAMESS and WRF and the MILC version of the NSF benchmarks; the Seismic application is part of the SPEC HPC suite. Where appropriate for comparison, we've shown results obtained from

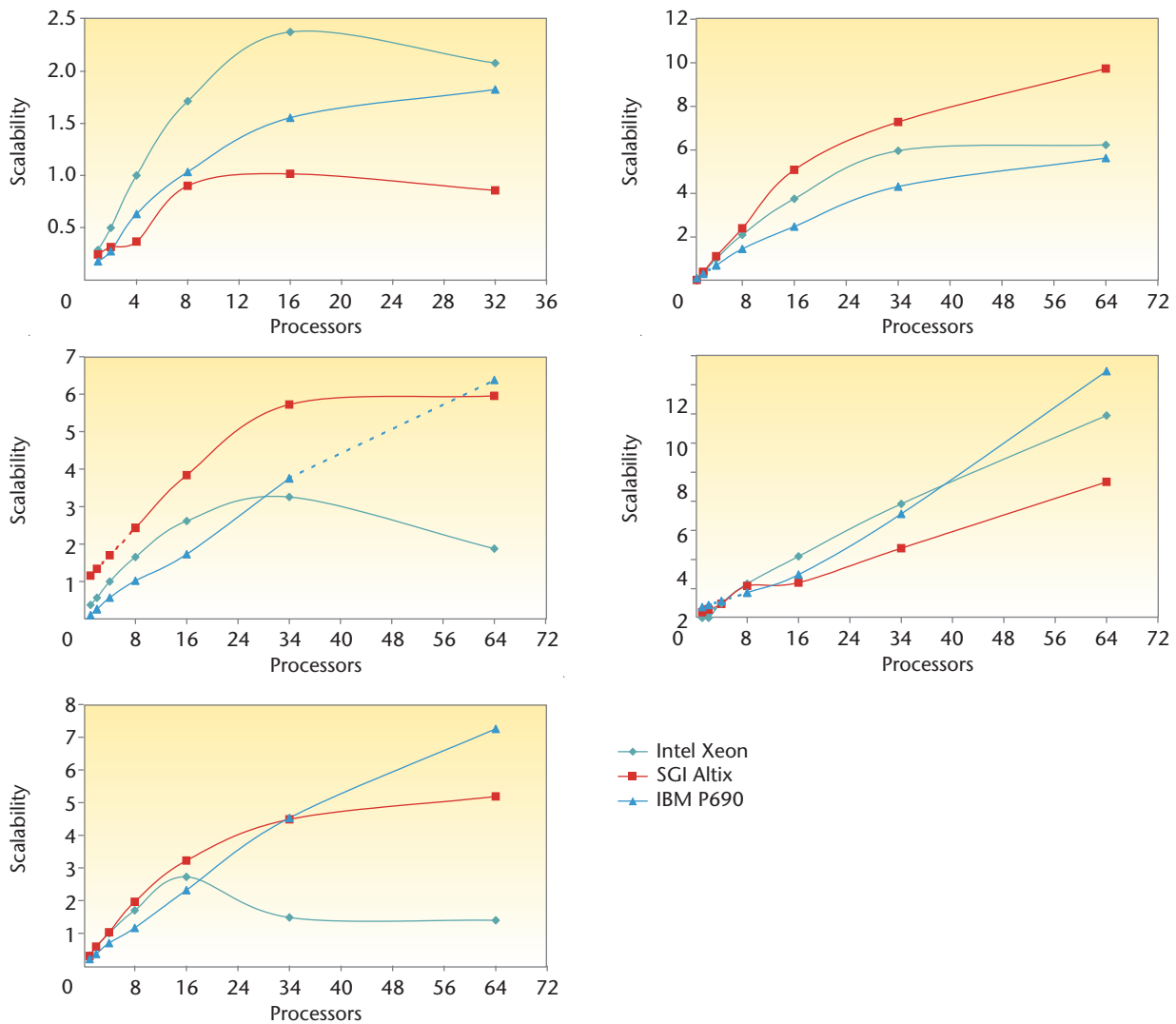


Figure 2. Relative performance of application benchmarks on three platforms. All performance numbers are relative to the execution speed on an Intel Xeon with four processors; we used the medium data set in all executions. For comparison, we show the execution times of the same machines using the HPL kernel benchmarks (with  $N = 9,900$ ). Note that every machine performed the best on one application and the worst on another. Hence, the relative ranking of these systems critically depends on the problem being solved.

the HPL kernel benchmarks (the HPL codes are also part of the NSF benchmark suite).

### Overall Performance

Absolute application performance is the total runtime for solving an overall application problem. The machines we used included an IBM P690 ([www.ccs.ornl.gov/Cheetah/Cheetah.html](http://www.ccs.ornl.gov/Cheetah/Cheetah.html)), an SGI Altix ([www.ccs.ornl.gov/Ram/Ram.html](http://www.ccs.ornl.gov/Ram/Ram.html)), and an Intel Xeon cluster ([www.itap.purdue.edu/rcac/news/news.cfm?NewsID=178](http://www.itap.purdue.edu/rcac/news/news.cfm?NewsID=178)). The runtimes for these medium data sets on a 32-processor IBM P690 platform ranged from approximately

10 minutes (for Seismic) to more than an hour (for GAMESS).

Figure 2 shows the relative performance of the individual benchmark applications on the three platforms as well as the HPL benchmark's performance. We took measurements on up to 64 processors, except for Seismic (up to 32; the 64-processor runs didn't validate).

The four applications had different rankings on the platforms. In terms of execution time on the highest measured processor count, each platform performed both the best and the worst across the different benchmarks. In terms of speedup be-

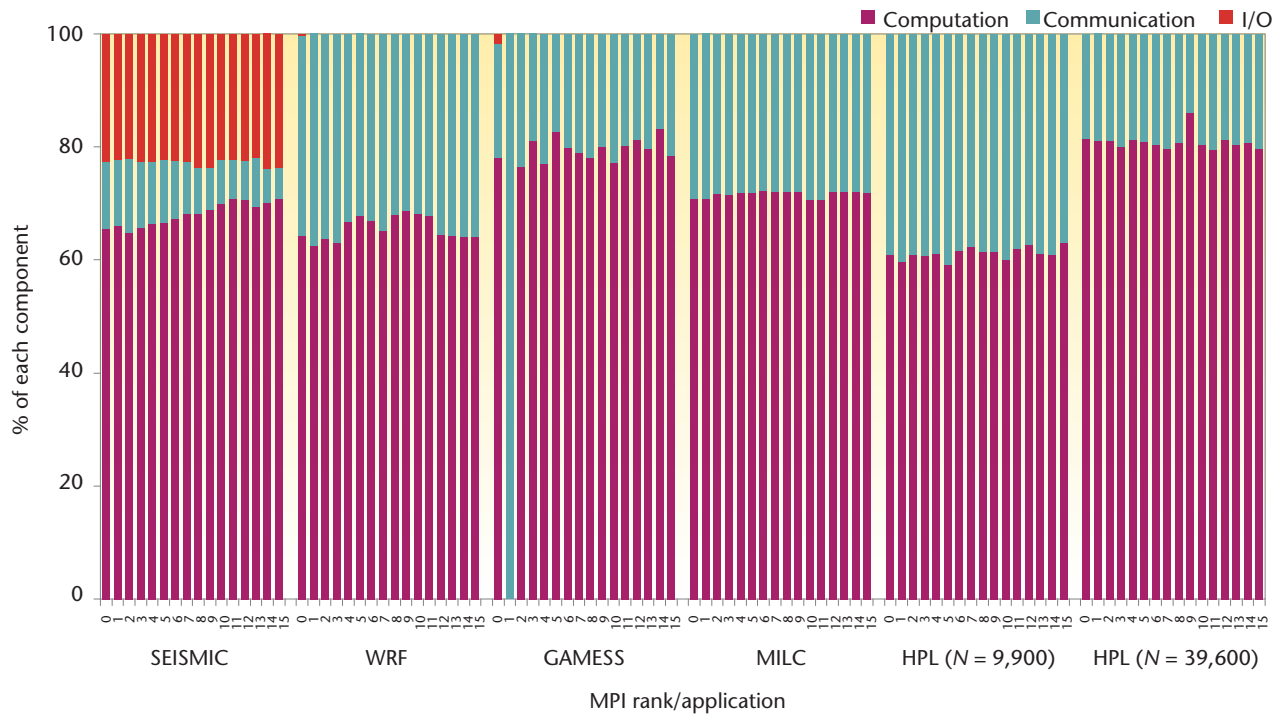


Figure 3. Components. When we look closer at the execution time breakdown of the benchmarks into computation, communication, and I/O, we see different application characteristics such as if the application is compute, communication, or I/O bound.

havior, the IBM P690 platform scaled the best in Seismic, WRF, and MILC, but the worst in GAMESS. The good scaling behavior on MILC was consistent with a relatively high computation-to-communication ratio, which Figure 3 shows. On both the Intel Xeon and the SGI Altix, Seismic scaled only to 16 processors and WRF only to 32. We observed superlinear behavior for GAMESS on SGI Altix and Intel Xeon up to 32 processors and for MILC on 32 and 64 processors. The superlinear behavior might be due to data fitting nicely in L2 cache; we assumed ideal behavior for the four-processor case, as this was the smallest processor count the medium data sets could run on due to memory limitations.

The kernel benchmark results (HPL with  $N = 9,900$ ) were most similar to those of WRF. Evidently, because the different applications' performance behavior varied significantly, the kernel benchmark reflected only a small part of the application spectrum.

### Component Performance

As we mentioned earlier, system component measurements give insight into the behavior of individual

machine features: their relative performance shows a feature's contribution to a computational problem's overall solution. Furthermore, component performance relative to some upper bound shows us how efficiently the machine feature is exploited compared to theoretical limits. We used the performance analysis tools *mpiP*, *hpmcount*, and *strace* to measure specific application component characteristics such as communication, computation, and I/O. Figure 3 shows the time breakdown.

**Communication characteristics.** We measured the communication component overhead for all the application benchmark codes and the HPL benchmark with constant and scaled problem sizes, and found the communication cost to be 5 to 40 percent of the overall execution time. Seismic did the least amount of communication, followed by GAMESS, MILC, and WRF. We measured all four phases of Seismic separately and later aggregated them; ultimately, Seismic showed significant load imbalance. In GAMESS, a single processor communicated 100 percent of the time. WRF, MILC, and HPL were the most balanced, but the difference between the least



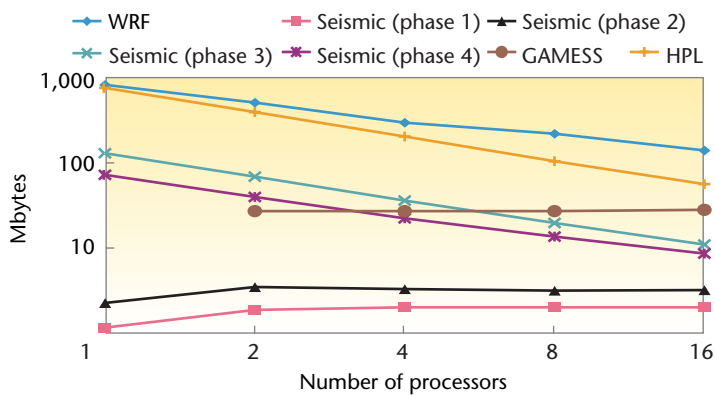


Figure 4. Benchmark memory footprints. The total memory size per processor for an Intel Xeon cluster varies widely depending on the benchmark used.

and most communicating processors was still 50 percent or more.

HPL communication depended on problem size and exhibited locality characteristics: with an increasing data set size, communication reduced significantly. This feature of the kernel benchmark led to generally good scaling behavior on very large machines, but it also contributed to the difference in rank lists of application versus kernel benchmarks. We observed a significant difference in communication cost overhead across architectures (IBM P690 and Intel Xeon Linux cluster) for the MILC and WRF benchmarks because they're communication intensive. The IBM P690 system provided higher communication bandwidth than the Intel Xeon system.

**I/O characteristics.** We measured I/O volume on all the benchmark codes and found that MILC and HPL didn't perform any I/O, whereas in both WRF and GAMESS, a single processor performed all the I/O. The I/O volumes of these codes (four Seismic phases measured separately, WRF, and GAMESS) ranged from 61 Mbytes to 5.5 Gbytes. The fraction of execution time taken by the I/O was small in both GAMESS and WRF but significant in Seismic, especially in phase 1. The I/O read and write volumes differed, yet they took the same amount of time due to differences in read and write speed.

**Memory footprints.** Figure 4 shows the benchmark's memory footprints as a function of the number of processors. Again, we split Seismic into its four execution phases, with the sizes ranging from 20 Mbytes per processor in Seismic's phase 1

to 1 Gbyte per processor in MILC (due to the large footprint, MILC couldn't run on less than four processors). MILC, WRF, and Seismic's phases 3 and 4 exhibited the commonly expected behavior: memory footprint decreased steadily with increasing processor numbers. But in Seismic's phases 1 and 2 and in GAMESS, the memory footprint was independent of the number of processors. This finding is important because it refutes the common assumption that larger systems will naturally accommodate larger data sets. This assumption is the basis for a benchmark methodology that lets data sets "scale" and thus reduces communication, leading to seemingly improved performance numbers on large systems. Our results show that this path to scalability might not be correct.

### Application Characteristics

We can characterize computational applications from many diverse angles—the physical problem being solved, the algorithms used, computer language and source code properties, compiler-applied optimization methods, generated instruction characteristics, and upper limit analyses, to name a few. A systematic methodology of application characterization<sup>11</sup> could guide the process of answering the relevant questions in this area. Table 3 shows a small set of such data, providing insight into the core algorithms and programming languages used to compose our test applications.


As we mentioned earlier, understanding the characteristics of today's applications could help us anticipate the behavior of tomorrow's applications. Most performance prediction techniques are based on application signatures and machine profiles, combined via convolution methods. In one approach,<sup>12</sup> the authors defined synthetic kernels that exhibit interesting code and machine properties and then extrapolated these kernels' measured behavior to larger data set sizes. Another approach<sup>13–15</sup> measured an application's key characteristics on a current platform and then forecasted the application behavior on scaled data and machine sizes, using predictive formulas. The authors derived the formulas with a least-squares fitting approach<sup>15</sup> or via the compiler from the source code, expressing the way input data affects the volume of computation, communication, and I/O.

**A** good benchmarking methodology can save a tremendous amount of resources in terms of human effort, machine cycles, and cost. Such a methodology

**Table 3. Benchmark composition in terms of number of files, subroutines, lines, programming languages, and core algorithms.**

	Seismic	WRF	GAMESS	MILC
Files	59	313	61	152
Subroutines	354	1139	835	382
Lines	24,000	168,000	122,000	20,000
Languages	56% Fortran, 44% C	85% Fortran, 15% C	99.5% Fortran, 0.5% C	100% C
Algorithms	3D FFT, finite difference, Toeplitz solver, Kirchhoff integral	CFD (Eulerian equation solver), split-explicit method	Eigenvalue solver, dense matrix	Conjugate gradient, matrix multiply

must consider the relevance and openness of the chosen codes, well-defined rules for executing and reporting the benchmarks, a review process to enforce the rules, and a public repository for the obtained information. For the methodology to be feasible, it must also be supported by adequate tools that enable the user to consistently execute the benchmarks and gather the requisite metrics.

At the very least, reliable benchmarking results can help people make decisions about HPC acquisitions and assist scientists and engineers in system advances. By saving resources and enabling balanced designs and configurations, realistic benchmarking ultimately leads to increased competitiveness in both industry and academia. 

## Acknowledgments

*This work was supported, in part, by the US National Science Foundation under grants 9974976-EIA, 0103582-EIA, and 0429535-CCF. The machines used in our experiments were made available by Purdue University's Rosen Center for Advanced Computing and the Oak Ridge National Laboratory.*

## References

1. A.T. Wong et al., "Esp: A System Utilization Benchmark," *Proc. IEEE/ACM SC2000 Conf.*, IEEE CS Press, 2000; <http://citeseer.ist.psu.edu/wong00esp.html>.
2. S.C. Woo et al., "The Splash2 Programs: Characterization and Methodological Considerations," *Proc. 22nd Int'l Symp. Computer Architecture*, ACM Press, 1995, pp. 24–36.
3. M. Berry et al., "The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers," *Int'l J. Super-computer Applications*, vol. 3, no. 3, 1989, pp. 5–40.
4. M. Berry, G. Cybenko, and J. Larson, "Scientific Benchmark Characterizations," *Parallel Computing*, vol. 17, Dec. 1991, pp. 1173–1194.
5. R.W. Hockney and M. Berry, "Parkbench Report: Public International Benchmarking for Parallel Computers," *Scientific Programming*, vol. 3, no. 2, 1994, pp. 101–146.
6. R. Eigenmann and S. Hassanzadeh, "Benchmarking with Real Industrial Applications: The SPEC High-Performance Group," *IEEE Computational Science & Eng.*, vol. 3, no. 1, 1996, pp. 18–23.
7. R. Eigenmann et al., "Performance Evaluation and Benchmarking with Realistic Applications," *SPEC HPG Benchmarks:*

*Performance Evaluation with Large-Scale Science and Engineering Applications*, MIT Press, 2001, pp. 40–48.

8. M.S. Mueller et al., "SPEC HPG Benchmarks for High Performance Systems," *Int'l J. High-Performance Computing and Networking*, vol. 2, no. 1, 2004; <http://citeseer.ist.psu.edu/672999.html>.
9. V. Aslot et al., "Speccomp: A New Benchmark Suite for Measuring Parallel Computer Performance," *Proc. Workshop OpenMP Applications and Tools*, LNCS 2104, Springer-Verlag, 2001, pp. 1–10.
10. D. Bailey et al., *The NAS Parallel Benchmarks 2.0*, tech. report NAS-95-020, NASA Ames Research Center, Dec. 1995.
11. B. Armstrong and R. Eigenmann, "Benchmarking and Performance Evaluation with Realistic Applications," *A Methodology for Scientific Benchmarking with Large-Scale Applications*, MIT Press, 2001, pp. 109–127.
12. D.H. Bailey and A. Snively, "Performance Modeling: Understanding the Past and Predicting the Future," *Proc. 11th Int'l Euro-Par Conf.*, LNCS 3648, Springer-Verlag, 2005, pp. 761–770.
13. B. Armstrong and R. Eigenmann, "Performance Forecasting: A Methodology for Characterizing Large Computational Applications," *Proc. Int'l Conf. Parallel Processing*, IEEE Press, 1998, pp. 518–526.
14. L. Carrington, A. Snively, and N. Wolter, "A Performance Prediction Framework for Scientific Applications," *Future Generation Computer Systems*, vol. 22, no. 3, 2006, pp. 336–346.
15. M. Mahinthakumar et al., "Performance Evaluation and Modeling of a Parallel Astrophysics Application," *Proc. High Performance Computing Symp.*, Soc. for Computer Simulation Int'l, 2004; [www.scs.org/docInfo.cfm?get=1681](http://www.scs.org/docInfo.cfm?get=1681).

*Mohamed Sayeed is a research scientist in the Computing Research Institute and Rosen Center for Advanced Computing at Purdue University. His research interests include numerical modeling, high-performance computing for scientific computing, and performance modeling. Sayeed has a PhD in civil engineering from North Carolina State University. Contact him at [msayeed@purdue.edu](mailto:msayeed@purdue.edu).*

*Hansang Bae is working toward a PhD in electrical and computer engineering at Purdue University. His research interests include optimizing compilers and performance analysis of high-performance applications and platforms. Contact him at [baeh@ecn.purdue.edu](mailto:baeh@ecn.purdue.edu).*

**PURPOSE:** The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.

**MEMBERSHIP:** Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

**COMPUTER SOCIETY WEB SITE:** [www.computer.org](http://www.computer.org)

**OMBUDSMAN:** Call the IEEE Member Services toll-free number, +1 800 678 4333 (US) or +1 732 981 0060 (international), or email [help@computer.org](mailto:help@computer.org).

## EXECUTIVE COMMITTEE

**President:** Rangachar Kasturi\*  
**President-Elect:** Susan K. (Kathy) Land, CSDP; \* **Past President:** Michael R. Williams; \* **VP, Electronic Products & Services:** George V. Cybenko (1ST VP); \* **Secretary:** Michel Israel (2ND VP); \* **VP, Chapters Activities:** Antonio Doria; † **VP, Educational Activities:** Stephen B. Seidman; † **VP, Publications:** Sorel Reisman; † **VP, Standards Activities:** John W. Walz; † **VP, Technical & Conference Activities:** Joseph R. Bumblis; † **Treasurer:** Donald F. Shafer; \* **2008–2009 IEEE Division V Director:** Deborah M. Cooper; † **2007–2008 IEEE Division VIII Director:** Thomas W. Williams; † **2008 IEEE Division VIII Director-Elect:** Stephen L. Diamond; † **Computer Editor in Chief:** Carl K. Chang†

\* voting member of the Board of Governors

† nonvoting member of the Board of Governors

## BOARD OF GOVERNORS

**Term Expiring 2008:** Richard H. Eckhouse; James D. Isaak; James Moore, CSDP; Gary McGraw; Robert H. Sloan; Makoto Takizawa; Stephanie M. White

**Term Expiring 2009:** Van L. Eden; Robert Dupuis; Frank E. Ferrante; Roger U. Fujii; Ann Q. Gates, CSDP; Juan E. Gilbert; Don F. Shafer

**Term Expiring 2010:** André Ivanov; Phillip A. Laplante; Itaru Mimura; Jon G. Rokne; Christina M. Schober; Ann E.K. Sobel; Jeffrey M. Voas

**Next Board Meeting:**  
**18 Nov. 2008, New Brunswick, NJ, USA**



revised 22 May 2008

## EXECUTIVE STAFF

**Executive Director:** Angela R. Burgess; **Director, Finance & Accounting:** John Miller; **Director, Governance, & Associate Executive Director:** Anne Marie Kelly; **Director, Information Technology & Services:** Neal Linson; **Director, Membership Development:** Violet S. Doan; **Director, Products & Services:** Evan Butterfield; **Director, Sales & Marketing:** Dick Price

## COMPUTER SOCIETY OFFICES

**Washington Office.** 1828 L St. N.W., Suite 1202, Washington, D.C. 20036-5104  
 Phone: +1 202 371 0101  
 Fax: +1 202 728 9614  
 Email: [hq.ofc@computer.org](mailto:hq.ofc@computer.org)

**Los Alamitos Office.** 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314  
 Phone: +1 714 821 8380

**Email:** [help@computer.org](mailto:help@computer.org)  
 Membership & Publication Orders:  
 Phone: +1 800 272 6657  
 Fax: +1 714 821 4641

**Email:** [help@computer.org](mailto:help@computer.org)

**Asia/Pacific Office.** Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan  
 Phone: +81 3 3408 3118  
 Fax: +81 3 3408 3553  
 Email: [tokyo.ofc@computer.org](mailto:tokyo.ofc@computer.org)

## IEEE OFFICERS

**President:** Lewis M. Terman; **President-Elect:** John R. Vig; **Past President:** Leah H. Jamieson; **Executive Director & COO:** Jeffrey W. Raynes; **Secretary:** Barry L. Shoop; **Treasurer:** David G. Green; **VP, Educational Activities:** Evangelia Micheli-Tzanakou; **VP, Publication Services & Products:** John Baillieul; **VP, Membership & Geographic Activities:** Joseph V. Lillie; **VP, Standards Association Board of Governors:** George W. Arnold; **VP, Technical Activities:** J. Roberto B. deMarca; **IEEE Division V Director:** Deborah M. Cooper; **IEEE Division VIII Director:** Thomas W. Williams; **President, IEEE-USA:** Russell J. Lefevre

*Yili Zheng is a PhD candidate in the School of Electrical and Computer Engineering at Purdue University. His research interests include cross-layer optimizations and end-to-end solutions for computation-intensive problems such as those in biomedical, financial, and energy applications. Contact him at [yzheng@purdue.edu](mailto:yzheng@purdue.edu).*

*Brian Armstrong is a PhD candidate in Purdue University's computational science and engineering program. His research interests are performance analysis of parallel and distributed applications and automatic parallelization of industrial-grade applications. Contact him at [barmstro@purdue.edu](mailto:barmstro@purdue.edu).*

*Rudolf Eigenmann is a professor in the School of Electrical and Computer Engineering and interim director of the Computing Research Institute at Purdue University. His research interests include compiler optimization, programming methodologies and tools, performance evaluation for high-performance computers and applications, and Internet sharing technology. Eigenmann has a PhD in electrical engineering and computer science from ETH Zurich. Contact him at [eigenman@purdue.edu](mailto:eigenman@purdue.edu).*

*Faisal Saied is a senior research scientist at Purdue University's Computing Research Institute and Rosen Center for Advanced Computing. His research interests include numerical analysis, parallel numerical algorithms, numerical methods for the Schrödinger equation, large-scale eigenvalue problems, and performance engineering for large-scale applications. Saied has a PhD in computer science from Yale University. Contact him at [fsaied@purdue.edu](mailto:fsaied@purdue.edu).*