Using Performance Modeling to Design Large-Scale Systems

Article *in* Computer · November 2009

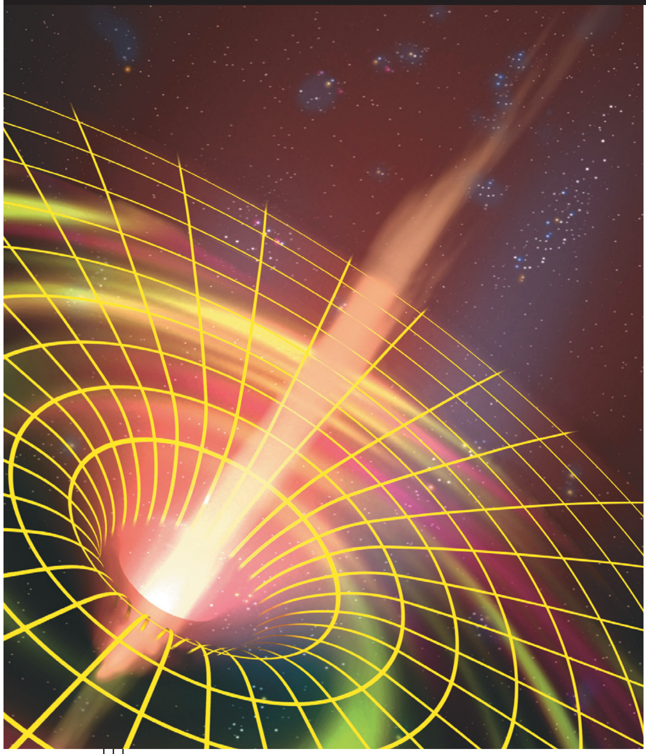
DOI: 10.1109/MC.2009.372 · Source: DBLP

Kei Davis
Los Alamos National Laboratory
95 PUBLICATIONS **1,085** CITATIONS
SEE PROFILE

Michael Lang
Los Alamos National Laboratory
72 PUBLICATIONS **852** CITATIONS
SEE PROFILE

Scott Pakin
Los Alamos National Laboratory
94 PUBLICATIONS **2,509** CITATIONS
SEE PROFILE

José Carlos Sancho
Barcelona Supercomputing Center
35 PUBLICATIONS **738** CITATIONS
SEE PROFILE



USING PERFORMANCE MODELING TO DESIGN LARGE-SCALE SYSTEMS

Kevin J. Barker, Kei Davis, Adolfy Hoisie, Darren J. Kerbyson, Michael Lang, Scott Pakin, and José Carlos Sancho, *Los Alamos National Laboratory, New Mexico*

A methodology for accurately modeling large applications explores the performance of ultrascale systems at different stages in their life cycle, from early design through production use.

Increasingly, scientific discovery has relied on computer simulation, fueling an insatiable demand for ever-faster supercomputers. Such computers can deliver results sooner, often with higher fidelity. However, as supercomputers get faster, they also get more complex. While a high-end desktop computer might contain 16 processor cores, a single memory address space, and a simple, internal, all-to-all network connecting the cores, today's fastest supercomputers—the Roadrunner system at Los Alamos National Laboratory,[1] the Jaguar system at Oak Ridge National Laboratory, and Jugene at Forschungszentrum Juelich—contain from tens to hundreds of thousands of cores, as many separate address spaces, and multiple interconnection networks with different features and performance characteristics.

All of these components interact in complex ways because of their hierarchical organization and contention for limited system resources. Thus, adjacent processor cores might coordinate their activities faster than distant cores, but distant cores need not compete with each other for access to memory. Roadrunner, Jaguar, and Jugene are all petascale systems, which can process 1×10^{15} floating-point operations per second (1 petaflop/s). Already, the high-performance computing community is investigating the challenges of exascale systems, which, while possibly only six years away, will have 1,000 times the peak performance of today's systems (1 exaflop per second) and a corresponding increase in complexity.

PERFORMANCE MODELING

Given the complexity of supercomputing architectures, it is hard to predict how fast an application running on today's petascale (or smaller) supercomputers will run on an exascale supercomputer. A thousandfold increase in peak performance rarely translates to an identical increase in application speed. Managing complexity necessarily exacts a performance toll. Accurately extrapolating performance from one system to another merely by gut feeling and guesswork is virtually impossible. However, being able to predict performance is an important capability because it helps system architects determine how to address the various design tradeoffs they face. Furthermore, it helps computational scientists decide which supercomputer to port their applications to, as porting to a state-of-the-art supercomputer can be a time-consuming process.

Published by the IEEE Computer Society 0018-9162/09/$26.00 © 2009 IEEE

A common approach to predicting performance is to use a machine simulator, a program that runs on an existing system but mimics a target system. While simulation works well for quick-running applications on modest numbers of processors, this approach struggles with predicting performance of long-running applications on extreme-scale systems. In this case a simulator needs either to run on a system of a size comparable to the target system, run for an unacceptable length of time, or forgo simulation accuracy.

We present an alternative to system simulation: *performance modeling*. Just as a model of a physical process provides a set of analytical formulas that describe the salient features of a complex natural phenomenon, a performance model offers a set of analytical formulas that describe the salient performance characteristics of a complex artificial phenomenon: a scientific application running on a current or future supercomputer. Furthermore, like its natural-world counterpart, a performance model supports not only prediction but also insight and interpretation.

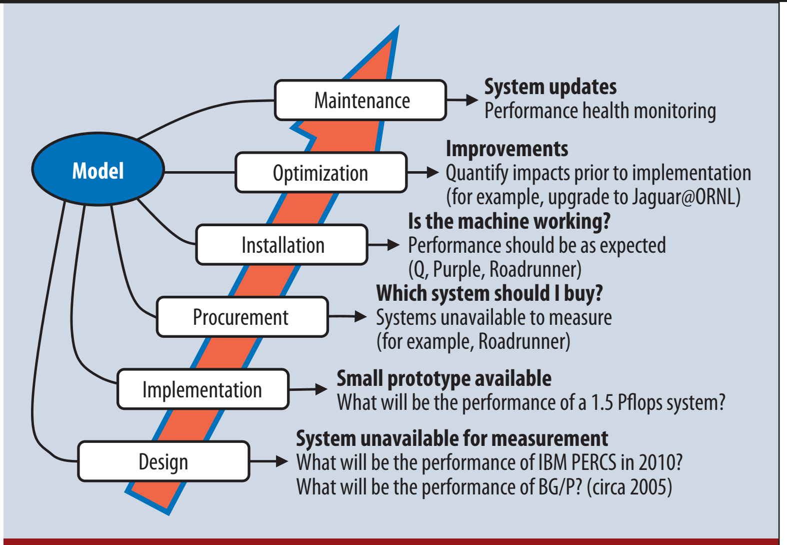

Model
Maintenance
System updates
Performance health monitoring
Optimization
Improvements
Quantify impacts prior to implementation
(for example, upgrade to Jaguar@ORNL)
Installation
Is the machine working?
Performance should be as expected
(Q, Purple, Roadrunner)
Procurement
Which system should I buy?
Systems unavailable to measure
(for example, Roadrunner)
Implementation
Small prototype available
What will be the performance of a 1.5 Pflops system?
Design
System unavailable for measurement
What will be the performance of IBM PERCS in 2010?
What will be the performance of BG/P? (circa 2005)

Figure 1. Widely applicable throughout a system's life cycle, performance modeling offers invaluable tools for analyzing performance.

At Los Alamos we have been using performance modeling almost daily over the past decade to explore the performance of numerous systems at different stages in their life cycle, from early design through production use. Figure 1 shows how widely applicable performance modeling is and how performance models are invaluable tools for performance analysis.

Early system design. Many models have been used to explore the performance of machines still on the drawing board. For example, we utilized our performance models to guide the design of the innovative rich-interconnection network that will be used in the forthcoming IBM Blue Waters system at the National Center for Supercomputing Applications. In addition, we also used performance models to examine hybrid accelerated systems prior to Roadrunner's procurement.[1]

Implementation. When a small, perhaps prototype, system becomes available, we can use modeling to predict expected performance for larger-scale systems. Often, only a single node is available during early access, as was the case with modeling the full-scale Roadrunner and 270 compute-node Tri-Lab Computing Clusters now deployed at Los Alamos, Sandia, and Lawrence Livermore National Laboratories.

Procurement. We also have used performance modeling to compare competing vendor systems during procurement. This was first done in 2003 for the procurement of the ASC workhorse system Purple, deployed at Lawrence Livermore. Modeling was subsequently used in the procurement of Roadrunner and several other laboratory HPC systems. Measuring performance is typically not possible, either because the system scale is larger than anything currently available, or the system uses yet-to-become-available next-generation components.

Installation. Performance modeling helps to predict an expected level of performance and thereby verify that a system is correctly installed and configured. We first applied this process to the Accelerated Strategic Computing Initiative (ASCI) "Q" system, installed at Los Alamos in 2002. We used performance models to correctly identify a performance issue with the initial installation[2]: A factor of two performance loss attributed to significant operating system noise (or jitter) that, after optimization, resulted in achieved application performance being very close to the initial model predictions. We have subsequently applied this process to all recent systems at Los Alamos, and to ASC Purple at Lawrence Livermore.

Optimization. We also have used performance modeling to quantify systems prior to performing optimizations or upgrades. For example, the Jaguar machine at Oak Ridge National Laboratory has been through several upgrade cycles, including migrating the processors from dual-core to quad-core and also from single-processor to dual-processor nodes. Modeling provided an expected level of application performance in advance, which was used as a verification mechanism for a successful system upgrade. In addition, we have used models to guide routing optimizations for InfiniBand networks.

Maintenance. Performance modeling can be incorporated into a system's everyday operation and maintenance. We are currently exploring a multitude of modeling applications to assist with identifying potential performance problems during normal production use. By using a per-

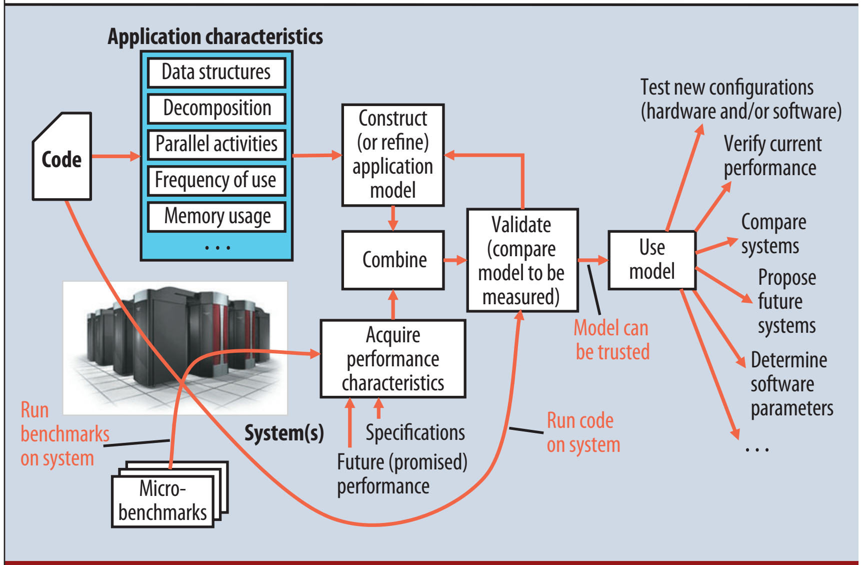

Application characteristics
Data structures
Decomposition
Parallel activities
Frequency of use
Memory usage
...
Code
Construct (or refine) application model
Combine
Validate (compare model to be measured)
Use model
Test new configurations (hardware and/or software)
Verify current performance
Compare systems
Propose future systems
Determine software parameters
...
Model can be trusted
Acquire performance characteristics
Run benchmarks on system
System(s)
Specifications
Future (promised) performance
Run code on system
Micro-benchmarks

Figure 2. Development of a performance model. An application-centric approach starts with the application code plus a representative input deck. Unlike other forms of performance prediction, this approach treats the application as a white box to be examined and understood.

formance model to determine expected performance, we can report in real time the system's "performance health" as its deviation from that expectation.

We have also used performance modeling to compare the performance of many systems, some in existence and some not, without the need for lengthy benchmarking activities. In the past, this has included a comparison of the Japanese Earth Simulator with other large-scale systems,[3] as well as comparing the use of different accelerator hardware in Roadrunner prior to procurement.

In addition, we have used performance modeling to compare code design alternatives—in terms of how to optimize a code's underlying data decomposition and mapping to a particular system to improve performance. Our design study uses performance modeling to explore the design space of future potential systems. Using validated performance models, we can quantify the impact on application performance given various improvements to system components.

IMPORTANT FACTORS IN PERFORMANCE MODELING

Figure 2 shows our approach to developing a performance model. Because our models are application-centric we begin with the application code plus a representative input deck. Unlike other forms of performance prediction, performance modeling treats the application as a white box to be examined and understood rather than a black box that simply spits out a runtime.

The first step identifies the application characteristics that contribute most to overall runtime. These typically include information about how the global data is distributed across processes, how much memory is accessed per data unit, how much communication is performed per data unit, and which processes communicate with which others. We are concerned only with first-order effects on performance; it does not matter what happens during initialization, for example.

A variety of techniques can be used to identify application characteristics. We start by profiling the code to see where the bulk of the processing time is spent and to understand the communication patterns, such as neighbor relationships, message counts, and message sizes. By comparing profiles across different problem sizes and process counts, we can form hypotheses about how communication and computation vary with program inputs and machine sizes.

Next, we manually examine the code to test our hypotheses and refine them accordingly. The goal is to produce a performance model that expresses runtime in terms of both application characteristics (for example, message sizes and counts as a function of the number of particles in a mesh) and system characteristics (such as the time needed to send a message as a function of message size).

Then, we fill in values for the performance model's system parameters. We acquire primitive performance characteristics from the target system in one of two ways. If access to the system is possible, we run a suite of microbenchmarks to gather primitive performance information. If not, as in the case of a system that has not yet been built, we rely upon system specifications or reasonable extrapolations based on existing hardware. In either case, an exciting prospect is that a range of values can be used to explore the application's sensitivity to the various system parameters.

We now have enough information to map the performance model onto the target system. For example, if we know the size of an application's main data structure, how that data structure is distributed among parallel processes, and how pieces of the data structure are communicated among processes, we can calculate the amount of data transferred in each message and the total number of messages that must be transmitted. If we measure the time needed to transmit a single message on a target system, computing the total time the application will spend communicating on that system becomes straightforward.

Not unexpectedly, the next step validates the predicted times by running the application on the target system using as many problem sizes and process counts as feasible.

Inaccuracies can occur from oversimplifications in the model: Perhaps an application characteristic that seemed unlikely to impact performance and was therefore excluded from the model in fact must be included.

As the model is validated on increasing numbers of input parameters, system architectures, and process counts, greater levels of trust can be placed in the model's accuracy. A trusted performance model can be used for many purposes:

- testing hardware or software upgrades to ensure that they deliver the expected increase in performance,
- examining a new system's performance to verify that it runs as fast as possible,
- comparing proposed systems to determine which will run the application faster,
- exploring the architectural design space to specify an optimal system under a set of constraints, and
- determining application input parameters, data structures, or data-decomposition techniques that can be expected to improve performance.

In short, a validated performance model is useful for application developers, users, those in charge of procuring new systems, and system architects.

SIMPLE PERFORMANCE MODEL EXAMPLE

Performance modeling normally studies the workings of large, complex, scientific applications. We choose as our example a generic five-point stencil, as might be used for solving a partial differential equation. Although a five-point stencil is not an "application" in its own right, it could form a piece of a larger application. For example, it may be used in the heat-transfer step of a climate-modeling application that also includes routines for simulating salinity, evaporation, radiation, and cloud cover. A complete performance model would need to include several of these components. In a parallel implementation of a five-point stencil, each process manages a rectangular piece of a large array and exchanges its subarray's boundary rows and columns, called ghost cells, with its neighbor process. Then, for each element in the array, it performs some computation, generally expressed in the following form:

$$A(i, j) \leftarrow (A(i, j) + A(i-1, j) + A(i+1, j) + A(i, j-1) + A(i, j+1)) / 5$$

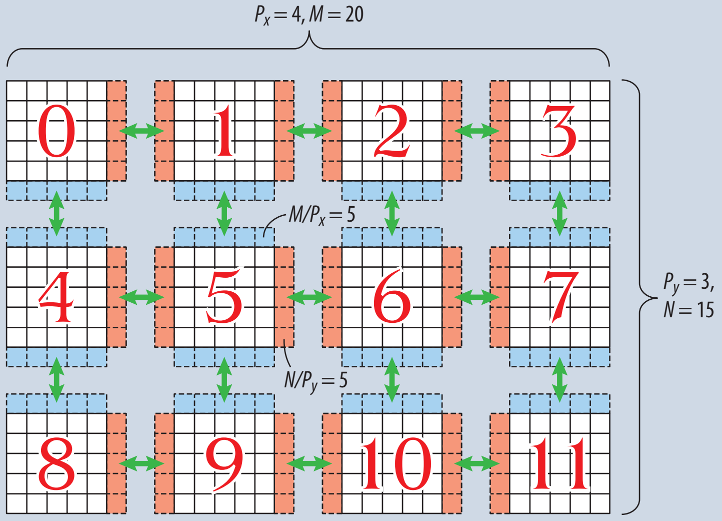


Figure 3. Generic five-point stencil application. Processes are numbered 0–11. White boxes represent the cells in a process's local fragment of the global array. Shaded boxes indicate ghost cells, and arrows indicate interprocess communication from one process's local cells to another process's ghost cells.

Assume the stencil runs on an $M \times N$ array of double-words (that is, 8-byte elements). Initially, we assume that blocking operations are used for all communication, with only one message sent or received at a time. Figure 3 shows a 20×15 array and the stencil communication and ghost zones used when this array is decomposed onto a 4×3 array of processes.

Given that problem specification, what do we need to know to predict how long our stencil application will run on a given system? Clearly, we need to know the values of M and N. We also need to know the number of parallel processes, P, and how the $M \times N$ array is distributed across those P processes. That is, given that the P processes will be laid out into a $P_x \times P_y$ arrangement, we need to know the values of P_x and P_y. On the system side, we need to know how long it takes to transmit a message of a given size ($T_{msg}(k)$) and how long it takes for the CPU to compute a value for a single array element, T_{elt}.

To combine those individual values into a performance model, we take a top-down approach, starting with the "fundamental equation of modeling":

$$T_{total} = T_{comp} + T_{comm} - T_{overlap}$$

That is, we separately model computation time, communication time, and the overlap between them, then combine these into an expression of total execution time. We model the time for a single iteration because the stencil application behaves identically with each iteration.

In our stencil example, T_{comp} is the time per array element multiplied by the number of array elements. The number of elements is determined by the size of a process's subarray, namely MN/P. The total time to process a subarray is therefore

$$T_{\text{comp}} = T_{\text{elt}}\, MN/P$$

per iteration. Because all processes compute in parallel, T_{comp} does not need to include an additional factor of P; the time for one process to compute is the time for all processes to compute. To model T_{comm} we need to know the maximum number of neighbors a process has. Because all processes communicate concurrently, the process performing the most communication determines T_{comm}.

The methodology for quasi-analytically modeling application performance applies just as well to actual multimillion-line applications.

Let's reason through some examples. In a sequential run, no process has any neighbors. If the processes are laid out in a horizontal line, the middle processes have the maximum number of neighbors, two: one to the left and one to the right. In an array of processes with at least three processes in each dimension, at least one process will have four neighbors, one each to the north, south, east, and west. By defining $N_{\text{max}}(p) = min\,(2, p - 1)$ we can more formally express the maximum number of neighbors in a 2D process array as $N_{\text{max}}(P_x)+N_{\text{max}}(P_y)$. Messages sent east or west are $8N/P_y$ bytes long (assuming 8-byte double words), while messages sent north or south are $8M/P_x$ bytes long. Because we assume blocking communication, all message times contribute to T_{comm}, implying that

$$T_{\text{comm}} = T_{\text{msg}}\,(8N/P_y)N_{\text{max}}\,(P_x) + T_{\text{msg}}\,(8M/P_x)N_{\text{max}}(P_y)$$

per iteration. Because there is no overlap of communication and computation in our stencil application,

$$T_{\text{overlap}} = 0.$$

We now have a complete performance model. Using only a couple of empirically or otherwise determined primitive-operation costs ($T_{\text{msg}}(k)$ and T_{elt}), and knowing a few application parameters (M, N, P_x, and P_y), we should in theory be able to predict the performance of our stencil code (T_{total}) all the way out to ultrascale systems. In practice, the model must be enhanced to take into account the additional complexity of a large-scale system. For example, T_{elt} might need to be made cognizant of vector or streaming operations (or even fused multiply-add instructions) and deep memory hierarchies; T_{msg} might need to take into account hierarchical communication costs—communication time within a processor socket, compute node, or network switch, and across network switches—as well as performance penalties due to network contention at each level of the communication hierarchy.

Nevertheless, we now have a straightforward methodology for quasi-analytically modeling application performance. This same methodology applies just as well to actual multimillion-line applications.

MODELING A LARGE-SCALE BLUE GENE SYSTEM

Blue Gene systems are some of the highest-performing machines available today, ranked in terms of peak performance on the Top 500 list of the world's fastest supercomputers (www.top500.org). Relative to other supercomputers, they provide as their salient characteristic enormous numbers of modest-speed processors. It is Blue Gene's massive parallelism that provides the potential for very high performance. We first consider the IBM Blue Gene architecture's latest incarnation: the Blue Gene/P.[4]

This system, used to validate our performance models, consists of 36,864 compute nodes. Sited at Lawrence Livermore National Laboratory, it arranges the compute nodes in a $72 \times 32 \times 16$ 3D torus. Each compute node contains four PowerPC 450 processor cores running at a clock rate of 850 MHz. Each core can issue four flops per cycle, for a peak performance of 3.4 gigaflops per second. The system's peak performance, totaled across all 147,456 processor cores, is a little more than 500 teraflops per second.

Several interconnection networks connect the nodes of a Blue Gene system. The main network is a 3D torus used for routine data communication among processors. This network has a peak bandwidth of 450 Mbytes in each of six directions, although software overheads limit the achievable performance to only 375 Mbytes per second. Communication latency between compute nodes depends on the number of hops a message must travel through the torus. We measured a minimum of 3 μs between adjacent nodes and a maximum of 6 μs between distant nodes. Additional networks support global synchronizations, global collective operations, and control functionality.[4]

Model validation

Our modeling approach is exemplified using three applications of interest to Los Alamos. These represent large-scale production applications used on many of the largest production supercomputers today.

SAGE. SAIC's Adaptive Grid Eulerian (SAGE) code is a multidimensional, multimaterial, Eulerian hydrodynamics application that utilizes adaptive mesh refinement.[5] SAGE is used with a wide variety of scientific and engineering problems, including water shock, energy coupling, cratering and ground shock, explosively generated air blast, and hydrodynamic instability problems. Processing consists of three steps repeated many times in each iteration: one or more data gather operations to obtain a copy of remote neighbor boundary data, computation on each of the local cells, and a scatter operation to update boundary conditions on remote processors. The interprocessor communication volume can be large in SAGE and also requires a large number of collective communications.[5]

Sweep3D. This time-independent deterministic particle transport kernel represents the core of a widely used method for solving the Boltzmann transport equation. Each cell in the grid has a data dependency on three upstream cells—computation cannot proceed until the results of computation on these upstream cells completes. Results from processing the cell are passed to three downstream cells. Computation forms a set of wavefronts that originate at each corner of the three-dimensional grid and preserve the data dependencies. The wavefront operation is synonymous with a pipeline—which must be filled and emptied—and can lead to low parallel utilization at scale. Interprocessor communications are fine-grained with small payloads.[6]

VPIC. A relativistic three-dimensional particle-in-cell code self-consistently evolves a kinetic plasma. VPIC (Vector Particle-In-Cell) simulations of laser-plasma interactions have been conducted at extreme fidelity, modeling up to 10^{12} particles on 136 × 106 voxels.[7] While particle-in-cell codes typically require more data movement per computation unit than other methods—such as dense matrix calculations, N-body computations, and Monte Carlo simulations—VPIC implements a novel method to reduce the volume of interprocessor communication required during simulation. Because of this, performance for the VPIC code is typically compute-bound, meaning VPIC is rather insensitive to message-passing performance.

Each application is most often run in a weak-scaling mode: The problem per processor remains constant as the system size scales, hence larger systems achieve higher fidelity simulations. This is typical of memory capacity bound codes, in which all available memory is used in the simulations. Note also that the problem size per processor core is inversely proportional to the number of cores in a processor.

Figure 4 shows the measured and modeled performance of SAGE, Sweep3D, and VPIC on Blue Gene/P when using between 1 and 32,768 compute nodes. In all cases, the time per iteration for each application appears as a function of node count. The subgrid size per node is fixed, which implies that perfect scaling would appear as a horizontal line at the single-node iteration time. Such performance is seldom achievable in practice due to both application and system characteristics.

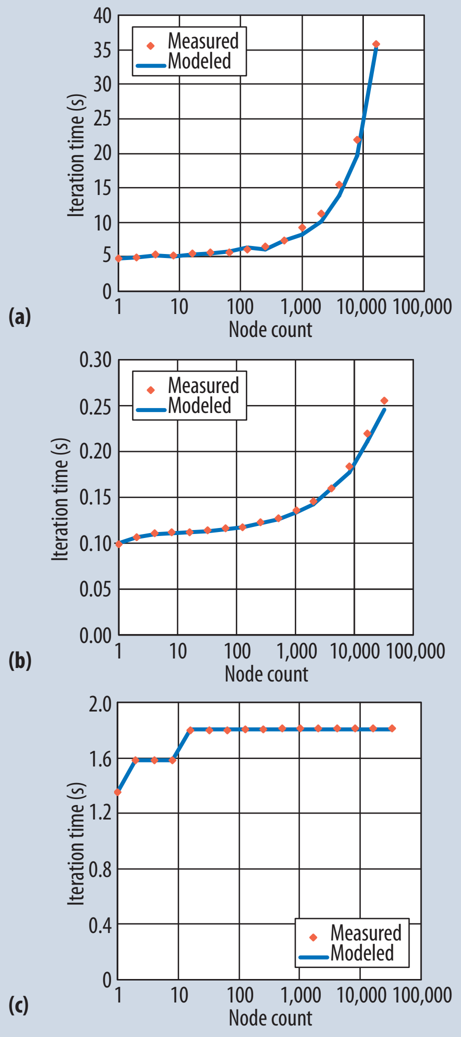


Figure 4. Validation of application performance models on a Blue Gene/P system. Graphs depict the measured and modeled performance of (a) SAGE, (b) Sweep3D, and (c) VPIC on Blue Gene/P when using between 1 and 32,768 compute nodes.

These graphs raise several points. First, our performance models predicted application performance extremely accurately out to all 32,768 nodes, even though the input to the models was based on measurements taken on only a few nodes. For SAGE, Sweep3D, and VPIC, the maximum prediction errors are 10 percent, 4 percent, and less than 1 percent, respectively, and the average prediction errors are 5 percent, 2 percent, and less than 1 percent.

Second, each application exhibits a qualitatively different scaling curve because of the manner in which it utilizes the system resources. Each process in SAGE sends many messages to a large number and varying set of recipients, depending on node count. Therefore, SAGE's communication performance depends heavily on network bandwidth and the dimensions of the subset for the torus being used.

Table 1. Main characteristics of Blue Gene/P and possible future configurations

Characteristics	Current Blue Gene/P	Possible system		
		A	B	C
Computation				
Cores per processor	4			×4
Communication				
Near-neighbor latency (µs)	3.0		÷2	
Per-hop latency (ns)	50		÷2	
Near-neighbor bandwidth (Gbyte/s)	375	×4		

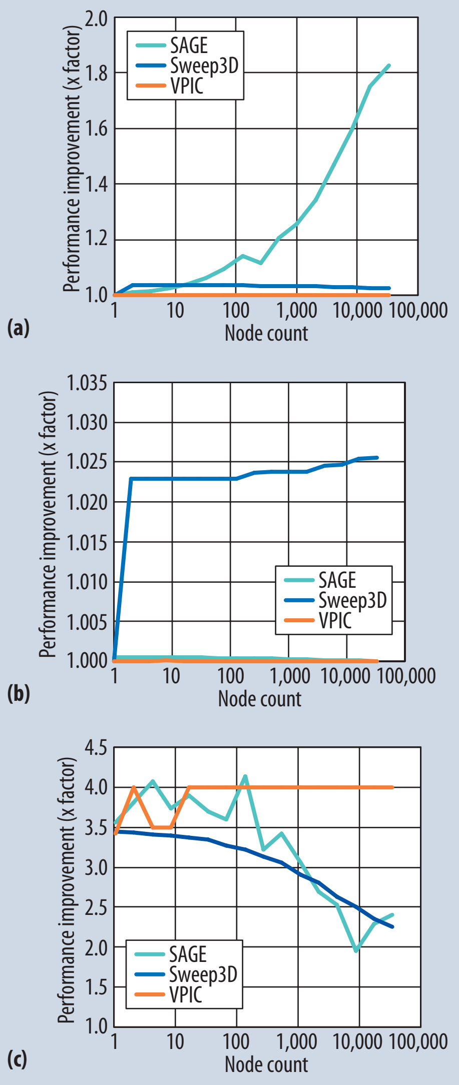


Figure 5. Expected improvements in application performance on a possible future Blue Gene/P type system.

Sweep3D's performance is compute-bound at small scale, but largely determined by the effect of the algorithmic pipeline at large scale. VPIC performance is determined almost exclusively by the speed at which particles are processed, varying only slightly when using 16 or more compute nodes. These differences reinforce the point that performance prediction based on simple extrapolation does not work: Consider measuring these applications on 100 nodes and predicting their performance on 10,000 nodes *without* the use of a performance model.

Exploring possible future system performance

In general, a model can be used to explore a parameter space that cannot be empirically measured. A performance model is no exception. In the context of exploring ultrascale system performance, investigating the performance of possible future systems offers an important use for modeling. At Los Alamos, we constantly focus on machines that do not yet exist but that could be deployed in 5 to 10 years. We illustrate this approach by modifying the baseline Blue Gene/P architecture's performance characteristics to reflect potential future hardware performance. In particular, we consider three separate possibilities: improving internode communication bandwidth, reducing internode communication latency, and increasing the number of cores per processor—all summarized in Table 1.

Figure 5 uses performance modeling to plot the performance improvement over the current Blue Gene/P system, given each of those possible system improvements. As in our validation experiments, qualitative differences in the performance behavior can be seen across the three applications. SAGE's performance is improved at large node counts when the communication bandwidth is quadrupled, as in Figure 5a, while Sweep3D's performance is barely affected, and VPIC's not at all.

Sweep3D's performance is, however, slightly affected by the reduction in communication latency shown in Figure 5b. As Figure 5c shows, the increase in core count per processor socket significantly improves the performance of all three applications at modest node counts, but this improvement diminishes with scale for SAGE and Sweep3D. In SAGE's case, the way its communication pattern increases in complexity with process count causes the diminishing improvement, which makes SAGE more sensitive to network performance on the 4× system than the baseline system. Sweep3D's diminishing performance improvement stems from the algorithmic pipeline effects, which are an inherent aspect of the code. VPIC is compute-bound even at 131,072 processes (32,768 nodes in the 4× system). Hence, its performance improvement at scale exactly matches the increase in core count.

Our methodology for modeling the performance of large applications running on ultrascale systems comprises an analysis of the key contributors to application performance as an analytical expression that maps primitive system performance characteristics to a predicted execution time. Using three production applications—SAGE, Sweep3D, and VPIC—we demonstrated that our performance models can predict execution times out to 32,768 nodes/131,072 processor cores with a worst-case prediction error across all data points of only 10 percent. This is a significant accomplishment because these applications all exhibit nonlinear performance characteristics as system size increases. Hence, extrapolations based on intuition are unlikely to produce accurate predictions.

In our analysis example of a possible future Blue Gene system, we expect that some applications will run significantly faster given an improvement in communication bandwidth. Others will see no performance increase from bandwidth improvements, but will from an increase in the number of cores per processor socket.

We apply performance modeling almost daily to answer performance-related questions. The ability to examine different system configurations on both existing and possible future architectures is proving a valuable capability both at Los Alamos National Laboratory and elsewhere. Although our performance-modeling work has always been focused on large applications running on the world's fastest supercomputers, our methodology is equally applicable to smaller applications on more modest-sized systems. Anyone who is concerned about understanding, predicting, and possibly improving application performance should seriously consider utilizing the analytical performance-modeling methodology we have presented in this article.

Acknowledgments

We thank Steve Louis at Lawrence Livermore National Laboratory for dedicated access to the Dawn Blue Gene/P system. This work was funded in part by the Advanced Simulation and Computing (ASC) program and the Office of Science of the Department of Energy. Los Alamos National Laboratory is operated by Los Alamos National Security LLC for the US Department of Energy under contract DE-AC52-06NA25396.

References

1. K.J. Barker et al., "Entering the Petaflop Era: The Architecture and Performance of Roadrunner," *Proc. IEEE/ACM Supercomputing* (SC08), IEEE CS Press, 2008.
2. F. Petrini, D.J. Kerbyson, and S. Pakin. "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q," *Proc. IEEE/ACM Supercomputing* (SC 03), IEEE CS Press, 2003.
3. D.J. Kerbyson, A. Hoisie, and H.J. Wasserman. "A Performance Comparison between the Earth Simulator and other Top Terascale Systems on a Characteristic ASCI Workload," *Concurrency and Computation Practise and Experience*, vol. 17, no. 10, 2005, pp. 1219-1238.
4. G. Almasi et al., "Overview of the IBM Blue Gene/P Project," *IBM J. Research & Development*, vol. 52, no. 1/2, 2008, pp. 199-220.
5. D.J. Kerbyson et al., "Predictive Performance and Scalability Modeling of a Large-Scale Application," *Proc. IEEE/ACM Supercomputing* (SC 01), IEEE CS Press, 2001.
6. A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Applications," *Int'l J. High-Performance Computing Applications*, vol. 14, no. 4, 2000, pp. 330-346.
7. K.J. Bowers et al., "0.374 Pflop/s Trillion-Particle Particle-in-Cell Modeling of Laser Plasma Interactions on Roadrunner," Gordon Bell Finalist, *Proc. IEEE/ACM Supercomputing* (SC 08), IEEE CS Press, 2008.

***Kevin J. Barker** is with the Performance and Architecture Lab (PAL) at Los Alamos National Laboratory. Barker received a PhD in computer science from the College of William and Mary. Contact him at kjbarker@lanl.gov.*

***Kei Davis** is with PAL in the Computer Science for High Performance Computing Group (CCS-1) at Los Alamos National Laboratory. Davis received a PhD in computing science from the University of Glasgow and an MSc in computation from the University of Oxford. Contact him at kei@lanl.gov.*

***Adolfy Hoisie** is leader of computer sciences for the HPC Group and director of the Center of Advanced Architectures and Usable Supercomputing at Los Alamos National Laboratory. Hoisie won the Gordon Bell Award in 1996 and is a coauthor of the recently published SIAM monograph on performance optimization. Contact him at hoisie@lanl.gov.*

***Darren J. Kerbyson** is the team lead of the Performance and Architecture Lab (PAL) in the Computer Science for High-Performance Computing Group (CCS-1) at Los Alamos National Laboratory. He received a PhD in computer science from the University of Warwick, UK. Kerbyson is a member of the IEEE Computer Society. Contact him at djk@lanl.gov.*

***Michael Lang** is with PAL in the Computer and Computational Sciences Division at Los Alamos National Laboratory. Lang received an MS in electrical engineering from the University of New Mexico. Contact him at mlang@lanl.gov.*

***Scott Pakin** is with PAL at Los Alamos National Laboratory. Pakin received a PhD in computer science from the University of Illinois at Urbana-Champaign. Contact him at pakin@lanl.gov.*

***José Carlos Sancho** is with PAL at Los Alamos National Laboratory. He received a PhD in computer science from the Technical University of Valencia, Spain. Contact him at jcsancho@lanl.gov.*