SANDIA REPORT
SAND2016-3730
Unlimited Release
Printed April 2016

Performance, Efficiency, and Effectiveness of Supercomputers

Robert W. Leland, Mahesh Rajan, Mike A. Heroux, and Doug Doerfler

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: http://www.ntis.gov/search

SAND2016-3730
Unlimited Release
Printed April 2016

Performance, Efficiency, and Effectiveness of Supercomputers

Robert W. Leland[1], Mahesh Rajan[2], Mike A. Heroux[3], and Doug Doerfler[4]
R&D S&E, Computing Science
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0351

Abstract

Our first purpose here is to offer to a general technical and policy audience a perspective on whether the supercomputing community should focus on improving the efficiency of supercomputing systems and their use rather than on building larger and ostensibly more capable systems that are used at low efficiency. After first summarizing our content and defining some necessary terms, we give a concise answer to this question. We then set this in context by characterizing performance of current supercomputing systems on a variety of benchmark problems and actual problems drawn from workloads in the national security, industrial, and scientific context. Along the way we answer some related questions, identify some important technological trends, and offer a perspective on the significance of these trends.

Our second purpose is to give a reasonably broad and transparent overview of the related issue space and thereby to better equip the reader to evaluate commentary and controversy concerning supercomputing performance. For example, questions repeatedly arise concerning the Linpack benchmark and its predictive power, so we consider this in moderate depth as an example. We also characterize benchmark and application performance for scientific and engineering use of supercomputers and offer some guidance on how to think about these.

Examples here are drawn from traditional scientific computing. Other problem domains, for example, data analytics, have different performance characteristics that are better captured by different benchmark problems or applications, but the story in those domains is similar in character and leads to similar conclusions with regard to the motivating question. For more on this topic, see Large-Scale Data Analytics and Its Relationship to Simulation.

---

[1] Director, Computing Research Center, Sandia National Laboratories
[2] Distinguished Member of the Technical Staff, Sandia National Laboratories
[3] Distinguished Member of the Technical Staff, Sandia National Laboratories
[4] Distinguished Member of the Technical Staff , Sandia National Laboratories

CONTENTS

NOMENCLATURE

HPC	High Performance Computer
LU	LU factorization
CG	Conjugate Gradient
MG	Multigrid methods
HPCG	High Performance Conjugate Gradient
FFT	Fast Fourier Transform
NSF	National Science Foundation
NCSA	National Center for Supercomputing Applications
LANL	Los Alamos National Laboratory
SNL	Sandia National Laboratories

INTRODUCTION

Our first purpose here is to offer to a general technical and policy audience a perspective on whether the supercomputing community should focus on improving the efficiency of supercomputing[1] systems and their use rather than on building larger and ostensibly more capable systems that are used at low efficiency. After first summarizing our content and defining some necessary terms, we give a concise answer to this question. We then set this in context by characterizing performance of current supercomputing systems on a variety of benchmark problems and actual problems drawn from workloads in the national security, industrial, and scientific context. Along the way we answer some related questions, identify some important technological trends, and offer a perspective on the significance of these trends.

Our second purpose is to give a reasonably broad and transparent overview of the related issue space and thereby to better equip the reader to evaluate commentary and controversy concerning supercomputing performance. For example, questions repeatedly arise concerning the Linpack benchmark and its predictive power, so we consider this in moderate depth as an example. We also characterize benchmark and application performance for scientific and engineering use of supercomputers and offer some guidance on how to think about these.

Examples here are drawn from traditional scientific computing. Other problem domains, for example, data analytics, have different performance characteristics that are better captured by different benchmark problems or applications, but the story in those domains is similar in character and leads to similar conclusions with regard to the motivating question. For more on this topic, see *Large-Scale Data Analytics and Its Relationship to Simulation*[2].

SUMMARY

There are various ways to measure and characterize computing system performance. The impression left varies substantially, depending on the measurement approach used, the characteristics of the machine, the scale at which the problem is run, and the inherent computational difficulty of the problem.

Much of the confusion regarding supercomputing performance derives from confusing efficiency and effectiveness. An *efficient* algorithm exercises the computational hardware of a system more fully than an inefficient one. An *effective* algorithm uses less time to solve the problem than an ineffective one, generally because it requires less work[3]. Often a less efficient algorithm is substantially more effective. In our view, the right goal is to implement the

[1] We use *supercomputer* here rather than the more general *term High Performance Computer (HPC)* to connote systems at the leading edge in scale and capability because that is the context in which we considered question.
[2] R. Leland, R. Murphy, B. Hendrickson, K. Yelick, J. Johnson, and J. Berry, *Large-Scale Data Analytics and Its Relationship to Simulation*, March 2014.
[3] The rigorous definition of *effectiveness* is more general – see R. Leland, *The Effectiveness of Parallel Iterative Algorithms for Solution of Large Sparse Linear Systems*, D. Phil. Thesis, Oxford University, 1989. The definition of effectiveness given here actually subsumes efficiency and may consider objectives other than minimizing computational time, but the distinction made in the text of our current paper is the essential one in this context.

most effective algorithm as efficiently as possible, even if this results in a lower efficiency of use than alternatives.

To achieve this goal, it is important to pursue both hardware and algorithmic advances contemporaneously and in proper balance. The two coevolve, and undue emphasis on one at the expense of the other will lead to stagnation. In some important cases, balanced progress has been achieved over decades such that net improvement in computational power has been approximately the *square* of Moore's Law. Expectations are geared to this remarkable result, and the focus should therefore not be on improving efficiency alone, as that would actually work against continued progress at historical rates. Hence the answer to our motivating question is "no."

The popular Linpack benchmark, which solves a dense set of linear equations, can be implemented quite efficiently on modern supercomputers — 80% to 90% of peak floating-point performance is often observed. Unfortunately Linpack is a poor predictor of performance on modern applications, which in most cases use more effective solution algorithms with different character.

Iterative solvers using sparse matrix storage schemes make up one such modern category of methods for solving linear equations. The High Performance Conjugate Gradient (HPCG) benchmark captures the key computational characteristics of most iterative solvers. In our tests HPCG ran at about 2% of peak floating-point performance. Similarly the Fast Fourier Transform (FFT) is an important kernel algorithm that also runs at low efficiency as a fraction of peak (typically less than 1%) but is highly effective for signal processing and in other relevant domains.

More generally, system performance at scale on computational problems that arise in mission, commercial, or scientific work is below 20% of peak floating-point performance. For a set of well-tuned scientific codes, we found results that ranged between 5% and 20% and averaged nearly 15% of peak. Engineering codes are typically dominated by kernel operations like HPCG or FFT, hence efficiency for even well-tuned codes are often as low as a few percentage points on more challenging problems at scale. It may be just a fraction of a percentage point in some critically important cases, particularly if multiple scales and multiple physical regimes must be addressed[4].

It is important to keep in mind, however, that the proper basis of comparison is not some unrealizable peak rate, but rather a significant advance with respect to the previous state of the art. In this context the history is positive. Many government procurements of supercomputers are explicitly structured to achieve specific increases of speed over previous generation systems, and typically increases by factors of approximately 10 are achieved.

Such improvements are possible because the efficiencies reported have held roughly constant over the relevant time period. There is substantial and well-founded concern

[4] Note that performance may depend on many factors other than floating-point performance. For example, memory or communication performance dominate for some important problems. As the system scales up in size, these performance characteristics generally scale with it and provide benefit independently of floating-point performance.

that this trend is now at risk due to the erosion of Moore's Law. This erosion is driving the emergence of a new architectural and algorithmic design space with many unresolved challenges. This topic is treated in depth in *Computing Beyond the End of Moore's Law: Alternatives for Sustaining Supercomputing Performance Improvements Despite Approaching Limits in Semiconductor Microelectronic Technology*. [5]

WHAT WE MEAN BY PERFORMANCE, EFFICIENCY,EFFECTIVENESS, AND RELATED TERMS

To measure *performance* of a computational system, we must specify an objective function. This could be cost, energy usage, or programming time, for example, but in the supercomputing context the objective function chosen is most commonly *time-to-solution* for some reference problem, and that is the sense we use here. Performance can be referred to as *peak* (some maximal rate that may be achieved only briefly or theoretically) or *sustained* (an average rate over some suitable time period intended to be reflective of the performance that can be expected in practice). *Floating-point* performance refers to the rate at which a computational system can perform arithmetic on real numbers, typically represented as 64-bit words. Depending on the problem, communication and memory performance can be much more important to overall performance.

The performance of application codes on large systems is a function of the size of the problem run and the fraction of the system used. We say *at scale* to capture the notion of running a problem using all or most of a system. Performance is also a function of the demands a computational approach makes on a given system's architecture. By *challenging problem* we mean a problem that stresses the weak points of a given system and therefore executes with poor performance relative to the typical case.

To measure *efficiency*, we must specify some basis of comparison against which to normalize performance. In assessing the efficiency of large supercomputers consisting of multiple processors, the general practice is to use the *parallel efficiency* measure, which compares the performance of the full system or a specific subset of the processors to the performance of one processor[6]. Large parallel supercomputers can also be assessed by their performance relative to their peak floating-point performance, and we start with this interpretation because that is how our main question is framed.

It is important to recognize that a computational approach may be efficient without being *effective*[7]. That is, an approach may execute operations at or near some peak rate but remain captive to a poor computational approach. Another approach with a better algorithm might solve the same physical or logical problem with operations that execute less efficiently on a given system but with drastically fewer of them and so come out far ahead in time-to-solution.

---

[5] J. Shalf, R. Leland, *Computing Beyond the End of Moore's Law: Alternatives for Sustaining Supercomputing Performance Improvements Despite Approaching Limits in Semiconductor Microelectronic Technology*, March 2014.[SAND2015-8039J]

[6] Ideally this division of labor across processors should result in a proportional decrease in execution time.

[7] R. Leland, The Effectiveness of Parallel Iterative Algorithms for Solution of Large Sparse Linear Systems, D. Phil. Thesis, Oxford University, 1989.

This distinction is at the root of much of the confusion over measurement of performance and efficiency of supercomputers. The confusion can be considerable when considering a variety of parallel architectures running challenging problems at scale.

SHOULD THE FOCUS BE ON IMPROVING THE EFFICIENCY OF COMPUTING SYSTEMS RATHER THAN SCALING THEM UP?

As a percentage of peak floating-point performance, system performance at scale on computational problems that arise in mission, commercial, or scientific work is generally below 20%. It is often as low as a few percentage points on more challenging problems at scale, and may be just a fraction of a percentage point in some critically important cases. Should the community therefore focus on improving efficiency on these key problems rather than focusing on building bigger, ostensibly more capable systems that run at very low efficiency?

Our basic answer is that it should not. The question presents a false dichotomy, and it is important to pursue both objectives in proper balance. Improvements in mathematical algorithms reduce the work required to achieve a given accuracy in the solution, and in most cases these advances can be implemented to advantage on subsequent architectures as well[8]. In some cases, algorithmic advances are specifically tuned to a given architecture and are not as transferable to future systems. These can provide quite important advances for specific needs, but tend to provide a narrow and unsustainable benefit. In other cases, algorithms have been provably optimized or nearly so, and the only real path to greater performance is better hardware[9]. The right goal is to implement the most effective algorithm for a given hardware system as efficiently as possible, even if this results in a lower efficiency of hardware use than alternatives might yield.

Our more nuanced answer has to do with the manner in which computational science proceeds. Typically when an advance in architecture occurs, capable teams across the technical community think hard about how to exploit this by developing more effective algorithms. Eventually these algorithms mature and are packaged and shared across the technical enterprise. Researchers striving to improve on the status quo eventually innovate with respect to the accepted algorithms. As these new methods evolve, they inspire advances in next-generation architecture. By this process architecture and algorithms co-evolve in the manner music and dance often co-evolve. If we were to rely solely on one or the other, we expect things would stagnate. Instead, we have experienced in some important cases roughly the same rate of progress in hardware and software capability over many decades. As a result, the rate of improvement in computational performance in many relevant areas has actually been

[8] The reduction in work tends to be more important in practice than the efficiency of the implementation. For example, in the early days of massively parallel computing, algorithms were at times proposed that made more efficient use of the hardware, but made less real progress toward the solution. Eventually the view took hold that, in most cases, the best parallel algorithm was a good parallel implementation of the best serial algorithm.

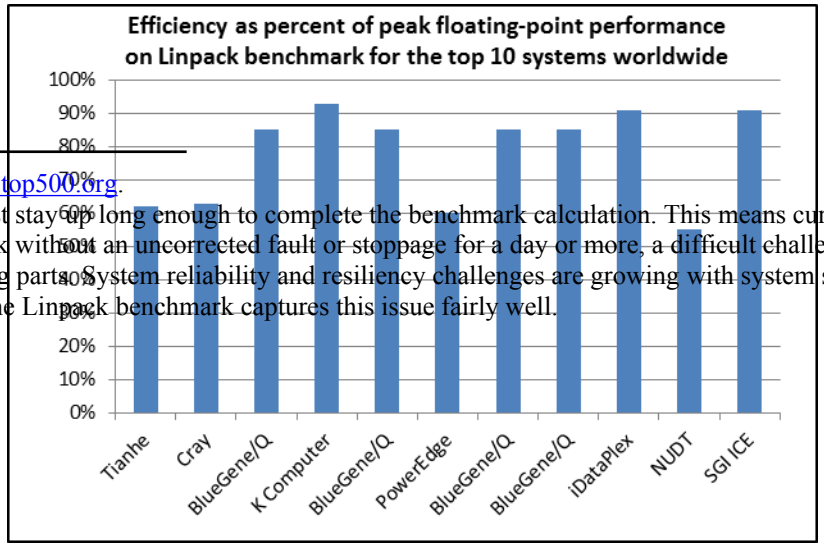
[9] This is arguably the case for linear solvers, where solvers that have a computational complexity that is linear in the problem size are known for a relatively broad class of problems. More work is needed to make these general and robust in all cases, but dramatic gains in algorithmic efficiency cannot be expected in these cases because we are already within a constant factor of an optimal solution.

approximately the *square* of Moore's Law. Expectations are geared to this remarkable result, and a focus on efficiency alone would actually work against continued progress at historical rates.

WHAT IS THE TYPICAL PERFORMANCE LEVEL ACHIEVED ON LINPACK AND WHY IS IT RELEVANT?

The Linpack benchmark requires the computer to solve a set of linear equations corresponding to a random, dense matrix using a method based on Gaussian Elimination. As shown in Figure 1, performance at scale on a well-tuned implementation of Linpack is often in the range of 80%–90% of peak floating-point performance. This high fraction of peak floating-point performance is achievable for several reasons: (1) the dense structure of the data causes the necessary memory references to occur in a very predictable manner which can be optimized, (2) the high computational intensity of the algorithm allows for a high degree of re-use of data once they are brought in from main memory and also masks the cost of communicating data between processors, and (3) computer architecture has evolved over many years to deliver high performance on this class of problems.

The Linpack benchmark is important in part because it is the basis for the Top500 ranking of supercomputers. The Top 500[10] list is the most widely quoted performance rating system for supercomputers, and the results posted there tend to dominate popular discourse concerning supercomputing. It is also important today because it serves as a fairly severe test of reliability for large systems[11].



[10] See http://www.top500.org.

[11] The system must stay up long enough to complete the benchmark calculation. This means current generation large systems must work without an uncorrected fault or stoppage for a day or more, a difficult challenge given their large number of working parts. System reliability and resiliency challenges are growing with system scale and complexity, and the Linpack benchmark captures this issue fairly well.

***Figure 1**: Performance as a percentage of peak floating-point performance for the Top 10 systems listed on the current (November 2013) Top 500 list. Systems are evaluated by running the Linpack benchmark at scale. Linpack measures time to complete solution of a set of dense linear equations with random entries. Note that well-tuned implementations often achieve 80% to 90% of maximally achievable performance on Linpack.*

WHY USE A METHOD THAT HAS VERY LOW EFFICIENCY?

Staying with our linear systems example, we note that computational methods have advanced considerably since Linpack was introduced in the late 1970s. Most problems of interest today are solved using *iterative* methods (which guess an answer and improve it based on feedback) applied to *sparse* representations (which encode the essential interactions with far less data). These sparse iterative methods do not execute as efficiently as Linpack but have much lower time-to-solution because they are more effective algorithms. To illustrate this, we note that Linpack uses an algebraic form of Gaussian elimination called *LU factorization* (LU) that has a run time proportional to n^3 where n is the number of unknowns in the system of equations. In contrast, the iterative *Conjugate Gradient* (CG) method typically has a computational complexity of $n^{(5/4)}$ with appropriate up-front *pre-conditioning* work. Members of a special class of iterative solvers called *Multigrid methods* (MG) iterate on multiple different representations of the problem with different resolution and cleverly combine candidate solutions at different levels to achieve a run time of order n. If n is large, say a million, the difference between n^3 and n corresponds to a reduction in work by a factor of a trillion. So if you are able to run MG on such a problem at 1% efficiency rather than LU at 90% efficiency, you are wise to do so.

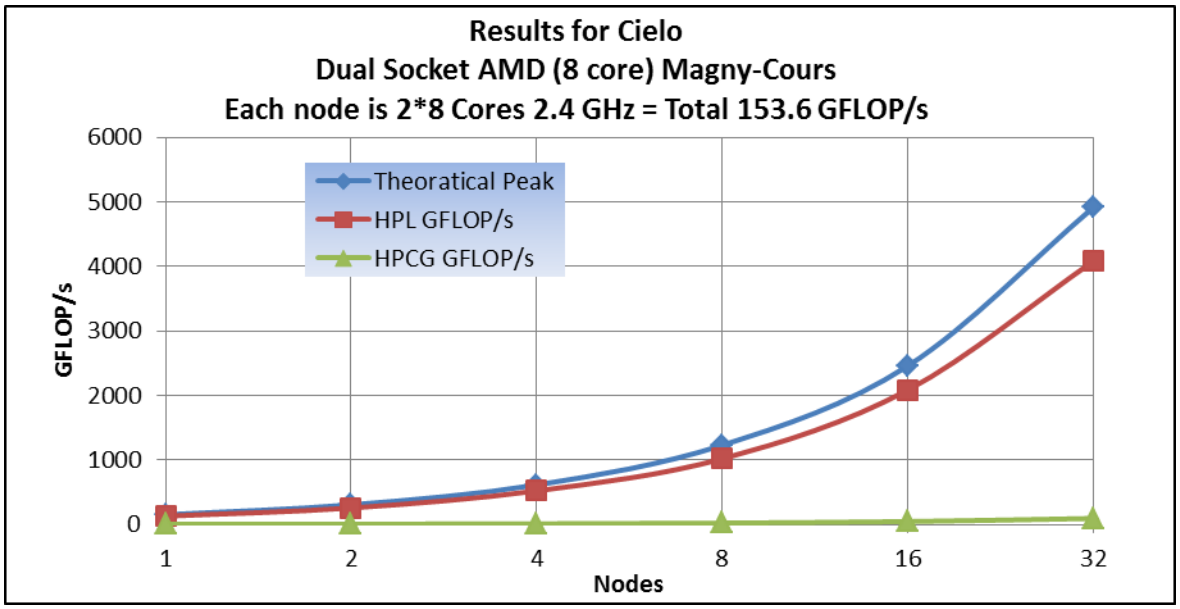


***Figure 2**: Here peak floating-point performance (in GFLOP/s) is shown by the blue line as a function of the number of computational nodes. Problem size scales with the number of nodes. The red line shows the performance of High Performance Linpack (HPL), a reference version of the Linpack benchmark. The green line shows the performance of High Performance Conjugate Gradient (HPCG) a reference version of the iterative Conjugate Gradient method. HPCG is currently proposed as a replacement for (or adjunct to) Linpack in the Top500 list. Note the performance as a percentage of peak for HPCG is poor by comparison — approximately 2%. Yet at scale it will solve a sparse problem to the same accuracy dramatically faster than HPL. Cielo, the machine used for these tests, is the current generation supercomputing platform deployed by Los Alamos and Sandia National Laboratories to support their stockpile computing. Cielo has a peak performance of approximately 1.4PF.*

Figure 2 illustrates this situation with run-time data obtained from a current generation supercomputing system. The High Performance Conjugate Gradient (HPCG) benchmark implements a Conjugate Gradient (CG) iterative solver. Because CG is a very effective algorithm, it is widely used in scientific and engineering simulations and is therefore a much better general predictor than Linpack of the performance to be expected from a portfolio of scientific and engineering codes running on a new platform. Note that the efficiency as a percentage of peak floating-point performance here is less than 2%, but due to the power of the algorithm it is still a much more effective approach where applicable.

This behavior is not specific to linear solvers. Figure 3, taken from the historic records of the HPC Challenge benchmark[12], shows a similar result for the distinct Fast Fourier Transform (FFT) algorithm, which is a critical kernel computation in signal processing and many other domains of science and engineering. The communication pattern of the FFT severely stresses the interconnect performance of large distributed systems and hence performance as a fraction of peak remains less than 1%. It is nevertheless a very effective algorithm which leads to dramatically lower time-to-solution for problems at scale.

[12] See http://icl.cs.utk.edu/hpcc.

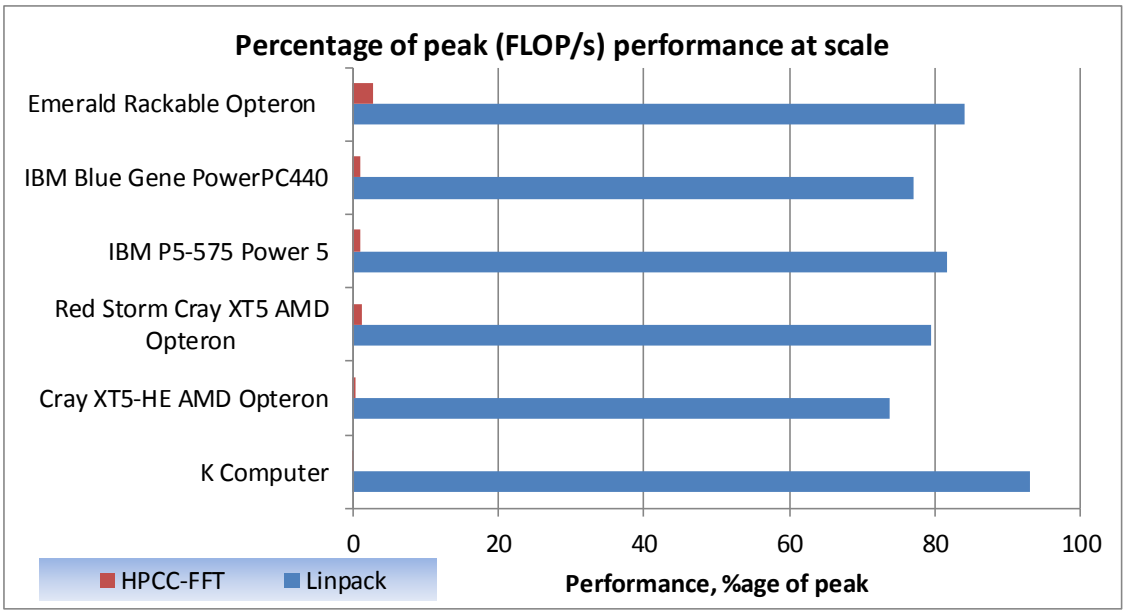


***Figure 3**: This chart shows the performance as a fraction of peak floating-point speed for various leading systems of their era on Linpack and the Fast Fourier Transform (FFT). The FFT algorithm provides a fundamental capability in signal processing and many other fields of science and engineering. Note that the FFT algorithm presents a challenging communication pattern to large, distributed systems and as a result performance is a small fraction of peak floating-point performance.*

WHAT SORT OF PERFORMANCE DO WE ACHIEVE ON ACTUAL APPLICATIONS?

Figure 4 shows performance as a percentage of peak floating-point performance for five different applications running on the National Science Foundation (NSF) Blue Waters system housed at The National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign. These scientific applications are running on a Cray system with a peak speed of 13 PetaFLOP/s (13E15 floating-point operations per second) and have been carefully tuned to achieve these performance levels, which are in the 5% to 20% of peak range and average nearly 15%. In another sample data set from the National Center for Atmospheric Sciences we note that their primary climate and weather codes (CESM, WRF, and POP)[13] measured less than 5% of the peak floating-point performance on production runs using as few as 8 nodes (256 cores).

Engineering applications typically have somewhat less regularity in geometry and typically employ less structured methods than scientific applications and so tend to perform less well. For reasons explained in the next section, we do not have a good data set to share regarding performance of these codes as measured by fraction of peak floating-point performance. We can say, however, that these applications tend to be dominated by a kernel operation, often HPCG or

[13] D. Del Vento, T. Engel, S. Ghosh D. Hart, R. Kelly, S. Liu, and R. Valent, "System-Level Monitoring of Floating-Point Performance to Improve Effective system Utilization," in 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC11).

FFT or something with similar performance. So the performance observed in these kernel operations (a few percent of peak floating-point performance) is likely reflective of performance of the over-all application. When the simulations require spanning multiple scales and multiple physical regimes, the performance may be lower still due to the added complexity and overhead of representing interactions across scale and regime.

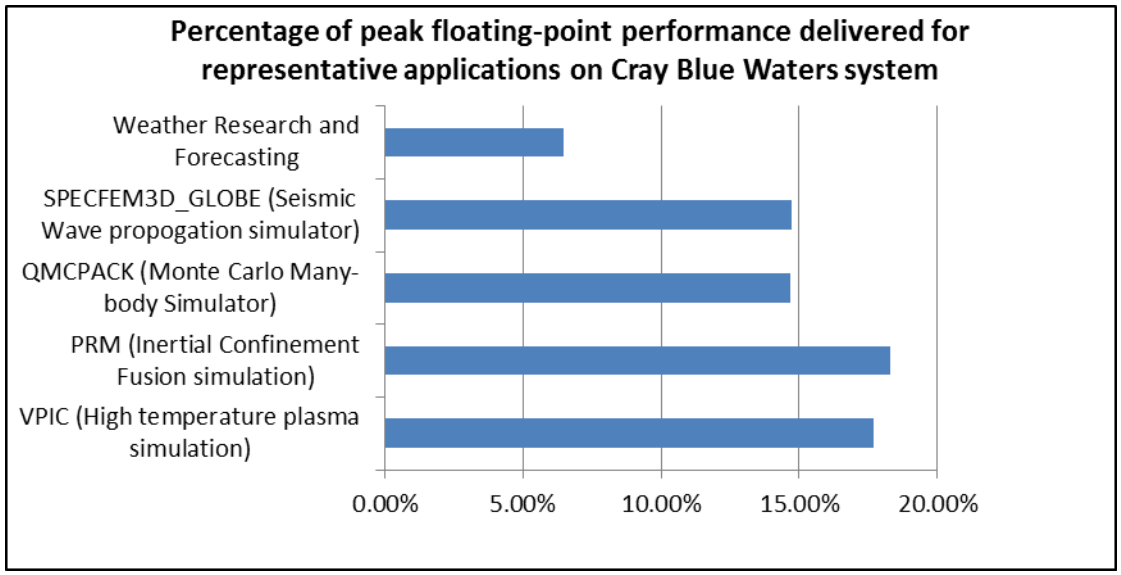


***Figure 4**: Percentage of peak floating-point performance delivered for representative applications run on the NSF Blue Waters system at the NCSA located at UIUC. Performance on these scientific applications is in the range of 5%–20% of peak floating-point performance and averages approximately 15%.*

HOW PERFORMANCE IS MORE TYPICALLY CHARACTERIZED IN THE SUPERCOMPUTING COMMUNITY

Except in the case of Linpack for the Top 500 List, researchers in the supercomputing field generally do not emphasize floating-point performance in reporting their results.[14] More typically they are concerned with how favorably the solution time scales and so report *speed-up* (in the simple case, the run time on one processor divided by the run time on multiple processors) or *parallel efficiency*, the speed-up normalized to p, the number of processors used.

A more typical plot characterizing supercomputing performance would look like that in Figure 5. Here the parallel efficiency of HPL and HPCG are shown, and it is clear they both scale well despite the major discrepancy in their performance as a fraction of peak floating-point performance. This approach distills out what is generally of greatest interest to supercomputing researchers, which is the scaling behavior rather than the performance of the single processor implementation[15].

[14] Some evidence for that is the difficulty we had in obtaining current data in this form for this paper.

[15] There is logic to this in that the individual processors are typically commodity parts (microprocessors) with designs that are determined by the broader market. It is the over-all system design (including the software stack) that has some flexibility in implementation.

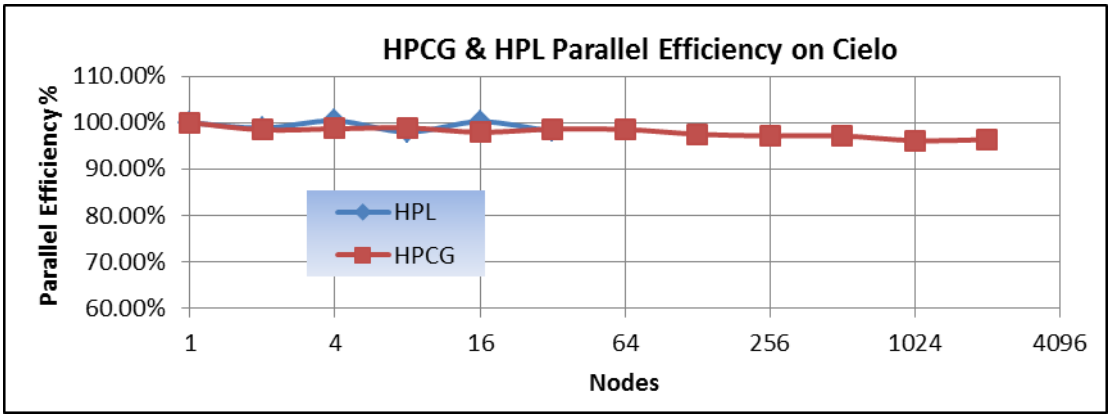


***Figure 5**: Plot showing the parallel efficiency of HPL (for solving the Linpack benchmark) as a function of system scale overlaid with a similar plot of efficiency data for HPCG. Both algorithms scale well (maintaining near perfect efficiency), although they have dramatically different performance as a percentage of peak.*

The other common approach is to show speed-up relative to some reference calculation. In the classical definition of speedup this reference basis is the single processor performance on that problem, but more recently it has been common to compare to the run time for the same problem instance on some previous generation supercomputer. This is typically in a procurement context, where performance criteria will be set to ensure the new supercomputer runs the work-load some factor faster than did the old system. An example from a procurement recently conducted jointly by Los Alamos National Laboratory (LANL) and Sandia National Laboratories (SNL) is shown in Figure 6. The average speed up over the previous platform is approximately a factor of 10. Cielo is a Cray system similar in design to the NSF Blue Waters system characterized in Figure 4, and we would expect similar scaling behavior.

Figure 6 provides another dimension of the answer to the question that motivated our paper. The factor of 10 performance gain [16]seen in application run times between generations of systems here is a characteristic result that has held true for at least the last several machine generations. The efficiencies seen on those earlier systems were similar for similar methods solving similar problems. In the metric that we care most about in the supercomputing context — time to solution of real problems — next-generation hardware *does* provide a large increase in performance. Said another way, the proper baseline for comparison is not the peak speed of the current generation system, but rather the actual delivered performance of the previous generation system. In that metric progress remains good. There is, however, a well-founded concern that progress in this form is stalling out as a result of the erosion of Moore's Law. Resolution of this will not be a matter of focusing solely on efficiency, but rather will involve a complicated set of trends and dynamics like those described here.

[16] This is a *scaled* performance gain. See the next section for an explanation of the distinction between this and other possible measures of performance gain.

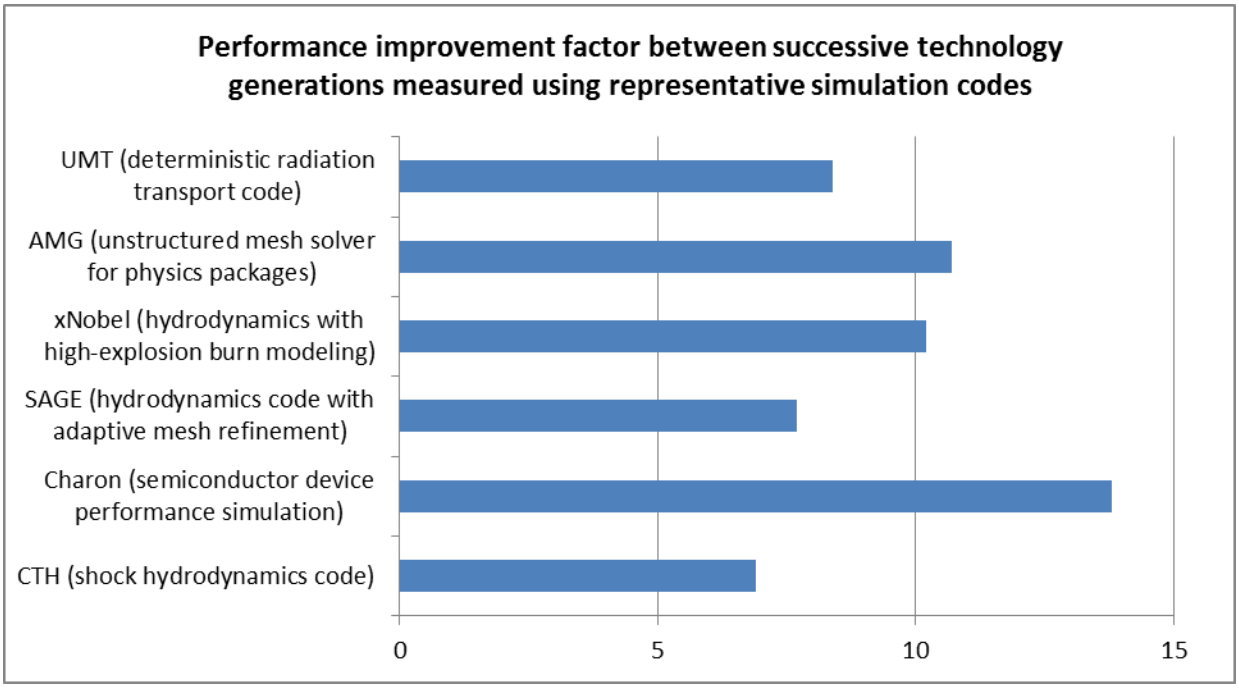


Figure 6*: Performance of various engineering codes from SNL and LANL running on a current system relative to their performance on a previous generation system. The systems are separated by approximately four years in age, and on average the applications run nearly 10 times faster. Results over the previous several generations have been similar, although there is concern within the community that the ratio of improvement will decline over the next decade due to trends in the semiconductor industry reflecting the erosion of Moore's Law.*

It is also true that on occasion a new, more effective method is discovered, but that may come at the *expense* of efficiency. Recall this was the case in the shift from the LU factorization method of Linpack to the Conjugate Gradient (CG) method of HPCG. It is also the case in the shift from CG to Multigrid. Through that progression we see a trend of *decreasing efficiency* but *increasing effectiveness*. A focus on efficiency alone here would be counter-productive. Of course much excellent work is devoted to optimizing efficiency of implementations and this is important as well, but it is far from the only consideration and often it is not the most important one.

THE CRUCIAL DISTINCTION BETWEEN SCALED AND FIXED PROBLEM PERFORMANCE MEASUREMENTS

There is an important subtlety to these measurements which merits understanding. As the system on which we are running the problem scales up, there is a choice to be made regarding the conditions of the measurement. We can either scale the problem proportionately or we can hold it constant in size. This is a seemingly innocuous point, but it is actually highly significant. In the first case we are measuring the *scaled parallel speedup/efficiency*, and in the second we are measuring the *fixed parallel speedup/efficiency*. In the scaled case the size of the local problem on each processor stays the same, and generally the overheads which cause inefficiency remain relatively small. For example, in most physical problems, we need to compute something associated with each region of the discretized volume assigned to a

processor and then communicate the state of the bounding surface of that volume to a neighboring processor. This volume to surface area effect is the key to good performance because the waiting time associated with communication is amortized by a calculation that is large in proportionate relationship, and these volume calculations can all proceed in parallel. As a result, complex problems can scale well to hundreds of thousands of processors, and perhaps beyond.

In the fixed case, as processors are added, the size of the local problem on each processor diminishes, and the proportion of the overhead climbs. In the physical example from above, the surface area effect quickly dominates, much as it would in the physical world. In practice, by the time a few hundred processors have been engaged a point of diminishing returns is reached and no further speedup is feasible. *Amdahl's Law*, the projected limit on speedup to be gained by a parallel approach, is a case of this fixed-problem scaling challenge. The concept of scaled speedup, which broke through this conceptual barrier, is one of the most important results in the field[17]. It also proved to be predictive of behavior — when given a larger machine, most scientists or engineers choose to solve a larger problem that better approximates the true solution of the real-world problem and hence use the scaled approach. There are contexts where fixed problem speedup or efficiency is the relevant measure, but generally the work done and the results shown reflect the scaled approach.

AN INTEGRATING EXAMPLE IN REVIEW

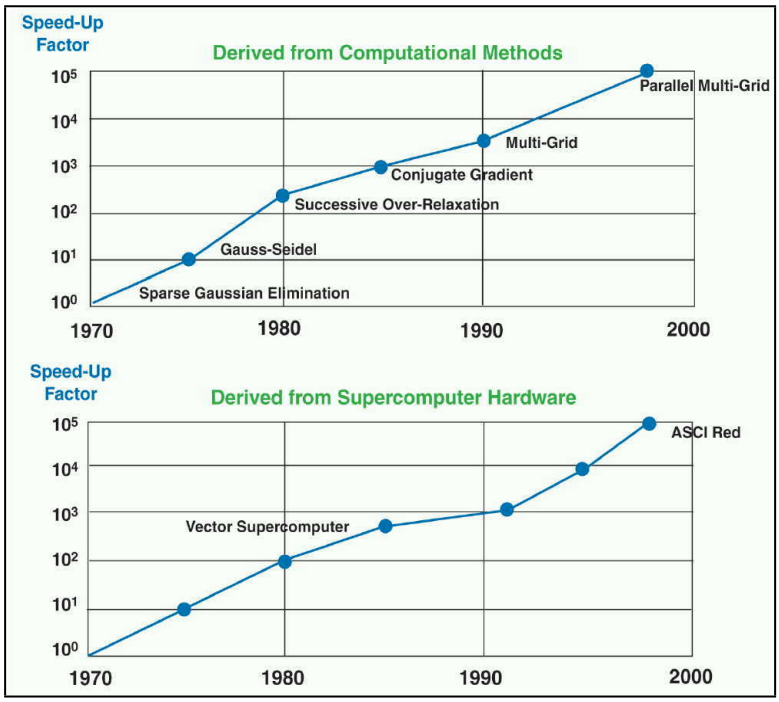
Consider as an integrating example the story reflected in the two plots shown in Figure 7. The top plot shows progress in methods for solving linear systems by indicating the relative speedup (on a suitable model problem) obtained with successive generations of new algorithms. The bottom plot shows a similar progression of speedups associated with successive generations of hardware platforms. From context we can conclude the scaled speedup approach was used, otherwise the problems would have been subdivided across larger and larger node counts to the point where further speed up would have been infeasible.

The algorithm progression starts with *Sparse Gaussian Elimination*, which exploits structure in the problem to provide a more effective algorithm than the LU factorization used in Linpack. Iterative methods then became more popular as they were well suited to the vector-based machines which became available in the late 1970s and early 1980s. The iterative methods were less efficient than LU factorization but more effective and dominated that era. Over time Multigrid methods were developed that were general enough and sufficiently robust to displace the older CG-based iterative methods in some contexts.

All of the methods were challenged by the advent of parallel machines in the late 1980s and early 1990s, but they were adapted successfully. Where applicable, the Multigrid methods proved the least efficient but the most effective. Looking back over this progression, it is clear that speedup derived from hardware and that from algorithms is similar, so that the over-all performance gain is roughly the square of Moore's Law. At this point, only constant factor

[17] J. Gustafson, G. Montry, and R. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube," *SIAM Journal of Scientific and Statistical Computing,* 1988.

improvements over Multigrid are possible, so progress depends mostly on hardware improvements in the cases where it can be used.



From Society for Industrial and Applied Mathematics Review, 2001.

***Figure** 7: These plots show algorithmic progress (speedup on a model problem) over many successive generations of algorithms for solving linear systems and compare this progress to the speedup seen in system hardware. Note the average improvement in algorithmic performance and hardware performance are comparable over the 25 year period shown.*

Distribution

4 Lawrence Livermore National Laboratory
Attn: N. Dunipace (1)
P.O. Box 808, MS L-795
Livermore, CA 94551-0808

1 MS0899 Technical Library 9536 (electronic copy)


Sandia National Laboratories