CTWatch QUARTERLY

ISSN 1555-9874 VOLUME 2 NUMBER 4A **NOVEMBER 2006**

High Productivity Computing Systems and the Path Towards Usable Petascale Computing

Part A: User Productivity Challenges

GUEST EDITOR **JEREMY KEPNER**

Available on-line at http://www.ctwatch.org/quarterly/

CyberInfrastructure Partnership

Cyberinfrastructure Technology Watch

http://www.ctwatch.org/

COMING SOON

High Productivity Computing Systems and the Path Towards Usable Petascale Computing

Part B: System Productivity Technologies

GUEST EDITOR **JEREMY KEPNER**

AVAILABLE NOVEMBER 30 http://www.ctwatch.org/quarterly

Metrics for Ranking the Performance of Supercomputers
Tzu-Yi Chen, Meghan Gunn, Beth Simon, Laura Carrington, and Allan Snavely

Modelling Programmer Workflows with Timed Markov Models
Andrew Funk, John R. Gilbert, David Mizell, and Viral Shah

Large-Scale Computational Scientific and Engineering Project Development and Production Workflows
D. E. Post and R. P. Kendall

A Compiler-guided Instrumentation for Application Behavior Understanding
Pedro C. Diniz and Tejus Krishna

A System-wide Productivity Figure of Merit
Declan Murphy, Thomas Nash, Lawrence Votta, Jr., and Jeremy Kepner

Symbolic Performance Modeling of HPCS Applications
Sadaf R. Alam, Nikhil Bhatia, and Jeffrey S. Vetter

SLOPE - A Compiler Approach to Performance Prediction and Performance Sensitivity Analysis for Scientific Codes
Pedro C. Diniz and Jeremy Abramson

Designing Scalable Synthetic Compact Applications for Benchmarking High Productivity Computing Systems
David A. Bader, Kamesh Madduri, John R. Gilbert, Viral Shah, Jeremy Kepner, Theresa Meuse, and Ashok Krishnamurthy

Performance Complexity: An Execution Time Metric to Characterize the Transparency and Complexity of Performance
Erich Strohmaier

INTRODUCTION

High Productivity Computing Systems and the Path Towards Usable Petascale Computing

Jeremy Kepner
MIT Lincoln Laboratory

This work is sponsored by the Defense Advanced Research Projects Administration under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

High Performance Computing has seen extraordinary growth in peak performance from Megaflops to Teraflops in the past decades. This increase in performance has been accompanied by a large shift away from the original national security user base of the 1970s and 1980s to more commercially oriented applications (e.g., bioinformatics and entertainment). In addition, there has been a significant increase in the difficulty of using these systems, which is now the domain of highly specialized experts. In response to these trends the DARPA High Productivity Computing Systems (HPCS) program was established to produce a new generation of economically viable, high productivity computing systems for the national security and for the industrial user communities. The primary technical goals of the program are to produce petascale computers that can better run national security applications and are usable by a broader range of scientists and engineers. The HPCS program is fostering many technological innovations, and two of the more noteworthy include: the concept of a "flatter" memory hierarchy (to accelerate a whole new set of applications) and high level programming languages (to allow a new set of users to take advantage of petascale systems).

Beyond just producing petascale systems that are easier to use, the HPCS program has also funded a team of researchers to explore how to better measure productivity. This special double issue of CTWatch Quarterly features 17 articles on the topic written by folks who have devoted themselves to these questions for the past few years. The articles are broken down into two groups. The first group "User Productivity Challenges" consists of a series of broader and more easily accessible articles that seek to highlight the user experience with HPC Technologies and the current issues therein. The second group "System Productivity Technologies" is made up of more deeply technical articles that describe specific technologies for enhancing productivity. Although CTWQ is not a traditional technical journal and would not normally carry articles in the second group, we felt it was important to try to present a more complete and balanced picture of the HPCS Productivity effort and give it the kind of wide exposure that CTWQ's on-line publication model makes possible.

The articles deal with the question of productivity from a number of perspectives. Tools are then provided for folks who want to look at productivity from an organizational perspective, from an individual programmer perspective, or from the perspective of a hardware or software innovator. From the technology innovator's perspective I think we have provided two particularly useful tools. The *HPC Challenge benchmarks* developed under HPCS is a suite of benchmarks that allows hardware designers to get credit for innovating above and beyond the design space defined by the Top500 benchmark. The *HPC software development measurement system* developed under HPCS allows software designers to conduct detailed experiments with programmers and determine precisely where the users spend their time.

The productivity of HPC users intrinsically deals with some of the brightest people on the planet, solving very complex problems, using the most complex computers in the world. Anyone who truly wants to get insight into such a complex situation must be prepared to invest some time in the endeavor. However, I am confident that those who do will find the effort rewarding.

Making the Business Case for High Performance Computing: A Benefit-Cost Analysis Methodology

Albert Reuther
MIT Lincoln Laboratory
Suzy Tichenor
Council on Competitiveness

High performance computing (HPC), also known as supercomputing, makes enormous contributions not only to science and national security, but also to business innovation and competitiveness—yet senior executives often view HPC as a cost, rather than a value investment. This is largely due to the difficulty businesses and other organizations have had in determining the return on investment (ROI)[1] of HPC systems.

Traditionally, HPC systems have been valued according to how fully they are utilized (i.e., the aggregate percentage of time that each of the processors of the HPC system is busy); but this valuation method treats all problems equally and does not give adequate weight to the problems that are most important to the organization. With no ability to properly assess problems having the greatest potential for driving innovation and competitive advantage, organizations risk purchasing inadequate HPC systems or, in some cases, foregoing purchases altogether because they cannot be satisfactorily justified.

This stifles innovation within individual organizations and, in the aggregate, prevents the U.S. business sector from being as globally competitive as it could and should be. The groundbreaking July 2004 "Council on Competitiveness Study of U.S. Industrial HPC Users,"[2] sponsored by the Defense Advanced Research Projects Agency (DARPA) and conducted by market research firm IDC, found that 97% of the U.S. businesses surveyed could not exist, or could not compete effectively, without the use of HPC. Recent Council on Competitiveness studies reaffirmed that HPC typically is indispensable for companies that exploit it.[3]

It is increasingly true that to out-compete, companies need to out-compute. Without a more pragmatic method for determining the ROI of HPC hardware systems, however, U.S. companies already using HPC may lose ground in the global competitiveness pack. Equally important, companies that have never used HPC may continue to miss out on its benefits for driving innovation and competitiveness.

To help address this issue, we present an alternative to relying on system utilization as a measure of system valuation, namely, capturing the ROI by starting with a benefit-cost ratio (BCR) calculation. This calculation is already in use at the Massachusetts Institute of Technology, where it has proven effective in other contexts.

HPC and Business Competitiveness

Alongside theory and experimentation, modeling and simulation using HPC has become the third leg of science and industrial design engineering. Industrial and other business firms are driven by external competition in a never-ending race to be first to market with the best products. The 2004 Council on Competitiveness study mentioned earlier also found that in these battles for global market supremacy, more-capable HPC resources often translate into faster time-to-market (in some cases more than 50% faster), reduced costs, and superior product quality.

[1] ROI can be captured with a variety of corporate finance techniques, including benefit-costs ratio (BCR), net present value (NPV), and internal rate of return (IRR).

[2] http://www.compete.org/hpc/

[3] "Partnering for Prosperity: Harnessing Our HPC Assets for Competitive Strength." Two Council on Competitiveness studies, both completed in January 2006: "Industrial Partnerships through the National Science Foundation's Supercomputing Resources," and "Industrial Partnerships through the NNSA's Academic Strategic Alliance Program."
Available at: http://www.compete.org/hpc/

Businesses use HPC systems (supercomputers) to design the cars we drive and the aircraft we fly in, to find and help extract new energy sources, to forecast severe weather, to discover new life-saving medicines and to safeguard our national security.

These benefits of HPC are often substantial:

- In 1980, Boeing tested 77 wings for the 767. Thanks primarily to HPC simulation, Boeing needed to test only 11 wings for the 7E7 Dreamliner series.
- Entertainment leader DreamWorks Animation SKG has leveraged supercomputing capabilities to set a whole new standard for animated films. As a result, the U.S. animation industry leads the world in market share.
- At The Proctor & Gamble Company, HPC simulations are used for everything from testing the absorbency of the materials in Pampers, diapers, to engineering bleach containers that minimize weight and maximize resistance to breakage, to designing the right geometric shape for Pringles, potato chips that allow the chip to drop gently into a container rather than fly off the conveyor belt.
- Wal-Mart relies heavily on HPC for supply chain management, including daily data analysis to determine what to stock in every Wal-Mart store worldwide, as well as for ergonomics—right down to turning on the lights in all the stores.
- Recent advances in HPC technology were crucial for enabling Chevron and two of its partners to discover a new field in Gulf of Mexico deepwater that could yield 3-15 billion barrels of oil, boosting U.S. reserves by up to half.

The story doesn't end there. Going forward, America's technological visionaries foresee equally dramatic advances if massive improvements in HPC power can be made available:

- HPC could revolutionize medical procedures and devices as well as product safety for a variety of consumer products by creating virtual humans of all shapes and ages. For example, the current generation of crash test dummy, while significantly improved over earlier generations, is essentially composed of metal frames with sophisticated sensors, lacking the ability to assess the impact of collisions on muscle, bone, tendons and soft tissues.
- HPC could increase oil recovery by 50-75% with more accurate seismic modeling of oil reservoirs. Currently, the uncertainties in the seismic models can lead to errors in drilling that both decrease output and increase environmental impact.
- HPC could create designer catalysts that selectively interact with the molecules in crude oil, for example, allowing greater production of high-value products in the oil refining process at lower costs. Moreover, specialized catalysts would convert more of the "waste" in low-grade crude to usable products while simultaneously decreasing the impact on the environment of disposing of unusable product.
- HPC could be used to model the spread of epidemics, enabling public health officials to intervene appropriately to halt the expansion of life-threatening diseases.
- HPC offers the potential for real time analysis of data traffic flow through thousands of miles of communications links. Increased computing power could achieve real time monitoring of the Internet and sense and avert denial of service and other types of attacks.

Barriers to HPC Use in the Private Sector

Council on Competitiveness research has identified three principal barriers inhibiting more widespread use of HPC. Educational and training hurdles (shortage of computational scientists) and technical obstacles (e.g., legacy codes need updating, new code development lags, growing performance gap between fast processors and other system technologies) are largely external to any one organization, although all of them affect the cost and ease of HPC use by the private sector. Within corporations, however, business strategies and decision-making processes can be a more significant obstacle to acquiring or accessing HPC tools.

In the boardrooms of many American companies, HPC isn't seen as an innovation edge, but rather as a cost of doing business—an enormous "hole in the pocket." And when investment options are being considered, it is often easier to get management approval for one that will minimize or reduce costs in the short tem versus an investment that has the potential to generate revenue over the longer term. HPC often is seen as the latter. As a result, management often favors "cheaper" systems and will not invest in more productive systems and the requisite training for personnel. For example, a $10 million investment in a high-end HPC machine to enable entry to a market six months early could result in $500 million of additional revenue in some industries, whereas a $1 million system with higher latencies and lower bandwidth might not be able to complete the solutions. Yet management often selects the less expensive approach.

The Council's research also found that organizations are often more limited by their budgets than by the computers available in the market. The impact of this can be severe. While some companies can run their existing problems on cheaper machines with faster turnaround, they are often unable to run new problems that lead to the breakthroughs required to maintain global competitiveness.

Not surprisingly, the other key barrier mentioned by businesses was the difficulty in determining the ROI from their current or proposed HPC systems.

Making the Business Case for HPC

One reason that many senior executives view HPC as a cost rather than a value investment is due to the difficulty in determining the ROI to the organization of the system. Traditionally, HPC systems have been valued by the usage of the system, i.e., the aggregate percentage of time that each of the processors of the HPC system is busy. The rationale behind this valuation is that a large sum of money was spent to purchase and maintain the system, and the justification for spending such sums is that the demand for computing on the system keeps the system busy close to 100 percent of the time.

Such a valuation method encourages the HPC system owners to use a resource management queuing system that schedules a non-stop stream of smaller computational simulations by many users to keep the HPC system busy. While this encourages utilization, it may not actually accommodate the most important problems facing the organization. Thus, this valuation methodology may not capture the true value that the use of an HPC system would provide to the organization, and it may not optimize users' needs in that it does not measure whether the problems most pressing for an organization's long-term competitive edge are being addressed. Without considering these issues, the actual out-of-pocket cost for an HPC system may appear

unaffordable, leading an organization to forego needed hardware purchases, upgrades, or certainly "greenfield" investments in HPC systems.

An alternative to relying on system utilization as a measure of system valuation is to capture the ROI by starting with a benefit-cost ratio (BCR) calculation. The BCR is expressed as the profit or cost savings divided by the sum of the investment over a given time period. In this article we shall simplify the discussion by assuming that all of the evaluations are conducted for a one-year time period. When we assume a one-year time period, the BCR is related to the one-year internal rate of return (IRR) with the following formula: BCR = 1 + IRR, or IRR = BCR – 1. Also, a net present value (NPV) analysis can be constructed using the benefits and costs identified in this article.[4] Naturally, all of these analyses rely on the collection of accurate data.

When evaluating investments in HPC systems,[5] the denominator of the BCR can be easy to find though one must make sure that all of the costs involved have been identified. However, the profit or cost savings due to the investment may not be nearly as easy to determine. The DARPA High Productivity Computing Systems (HPCS) program[6] has been working on determining the factors that play into the numerator and denominator of a BCR evaluation. In an article from the Winter 2004 Special Edition on HPC Productivity of the *International Journal of High Performance Computing Applications*,[7] the HPCS research team used productivity as their measure and defined it in classic economic terms as utility divided by cost. This is very similar to the BCR equation:

$$\text{BCR} = \frac{\text{benefit}}{\text{cost}} \qquad \text{productivity} = \frac{\text{utility}}{\text{cost}}$$

To expand on the utility (benefit) and cost for an organization, Dr. Jeremy Kepner of MIT Lincoln Laboratory, a HPCS Productivity Team member, developed a high performance productivity framework and evaluation model. The HPCS productivity model looks past the traditional measures of high performance computing systems such as peak floating point operations per second (flops) and system demand, since they usually do not have much influence on productivity. The result is the (SK)3 formulation:

$$\underset{\text{(BCR)}}{\text{productivity}} = \frac{\text{(time saved by users on system)}}{\left(\begin{smallmatrix}\text{time to}\\ \text{parallelize}\end{smallmatrix}\right) + \left(\begin{smallmatrix}\text{time to}\\ \text{train}\end{smallmatrix}\right) + \left(\begin{smallmatrix}\text{time to}\\ \text{launch}\end{smallmatrix}\right) + \left(\begin{smallmatrix}\text{time to}\\ \text{administrate}\end{smallmatrix}\right) + \left(\begin{smallmatrix}\text{system}\\ \text{cost}\end{smallmatrix}\right)}$$

In other words, the productivity level of a high performance computing system is a function of the time saved by engineers and scientists in solving advanced problems, taking into account not only the cost of the system, but also the time required to train users on it, prepare the application code(s) for parallel processing, launch the application(s), and administer the system. This formulation is intended for a research-oriented institution like a university or national laboratory.

In an industry environment, where systems are used less for basic research and more for solving product design and development challenges (i.e., a "production" environment), the variables for determining BCR/productivity may very well be different. Rather than computing the time saved by users on the system, an industry user may be more concerned with the value of newly developed products, potential increases in market share, profits generated (or lost) using HPC systems, or the importance of the job to be completed (i.e., how much revenue or market

[4] For some background on using BCR, IRR, and NPV to evaluate projects, please refer to G. Tassey, "Method for Assessing the Economic Impacts of Government R&D," Planning Report #03-1, National Institute of Science and Technology, Sept. 2003. For a through treatment please refer to S. Ross, R. Westerfield, and J. Jaffe, *Corporate Finance*, McGraw-Hill Irwin: New York, 2004.

[5] The intent of these sections is not to teach a tutorial on corporate finance but rather to illuminate various ways in which the benefits and costs associated with evaluating investments in HPC assets can be viewed and analyzed. The formulas and methodologies suggested here are based on the experiences at MIT Lincoln Laboratory. Our intent is to encourage readers to think about how their organization values an HPC solution.

[6] DARPA HPCS - http://www.darpa.mil/ipto/programs/hpcs/

[7] Kepner, J. "HPC productivity model synthesis," *International Journal of High Performance Computing Applications*, Vol. 18, No. 4, November 2004.

share will the company be able to gain once this large, extremely important problem is solved). In the denominator, the "time to parallelize" is irrelevant because all of the parallel software running on the HPC system is purchased. Hence the "time to parallelize" is replaced by the "cost of software." Also the launch time becomes minute in comparison to the software execution time. Again, we start with the basic formula: productivity (BCR) equals utility/benefit divided by cost. With the above changes, the new expanded BCR formulation follows.

$$\text{productivity (BCR)} = \frac{\sum \left(\text{Profit gained or maintained by project}\right)}{\left(\text{Cost of software}\right) + \left(\text{Training cost}\right) + \left(\text{Admin cost}\right) + \left(\text{System cost}\right)}$$

In the next section, we use these two productivity (BCR) formulas with numerical examples to demonstrate their use.

BCR Case Study Examples

Research Laboratory Example

At MIT Lincoln Laboratory, a federally funded research and developement center (FFRDC) for the Department of Defense, the research-oriented formula was used to evaluate the financial efficacy of a 600-processor, enterprise grid cluster solution that would be used by 200 users across Lincoln. The numerator value and each of the five denominator values were related to an average, fully burdened salary of $200,000 per year.

- In terms of the time saved by users, a system evaluation yielded that approximately 36,000 hours of user time would be saved by the system.[8]
- The time to parallelize all of the 200 users' algorithm and simulation code would take approximately 6,200 hours.
- Getting users trained on using Lincoln's system takes about 4 hours. Therefore the time to train is a total of 800 hours.
- Given a 10 second job launch time and an estimated 10,000 parallel job launches per year, it amounts to 27.8 hours of launch delayed that would not be experienced by serial jobs executed on desktops.
- The HPC system would be administered by one system administrator or 2000 hours.
- Buying 200 CPUs (100 dual-processor server nodes) per year with an estimated $5,000 per node costs $500,000. That is equivalent to 5,000 staff hours.

Inserting these values into the BCR/productivity equation yields:

$$BCR = \frac{[Salary] \times 36000}{[Salary] \times (6200 + 800 + 27.8 + 2000 + 5000)} = \frac{36000}{14028} = 2.6,$$

$$IRR_{1year} = BCR - 1 = 1.6 = 160\%.$$

Then we can also determine the one-year IRR as 160%. When taking the full range of average programming rate and cost to parallelize into account they found that BCR = 2.6 to 4.6.

[8] The time saved by users was calculated in a very conservative manner: time saved = (time system is in use) * (average number of users) * (1 – 1 / (Average number of CPUs per job)). This formula assumes that all jobs are fine-grained, synchronous parallel jobs; often parallel jobs are less synchronous and more coarsely grained. The last term in that expression can be substituted with a less conservative term such as log2(Average number of CPUs per job) or just Average number of CPUs per job. Using the latter, we have calculated a BCR greater than 30.

The MIT Lincoln Laboratory High Performance Computing team has compiled many examples of how their users are more productive when they use their interactive, on-demand enterprise HPC system. For example, one of the technical staff members is designing and evaluating algorithms to improve the weather radars used across the United States. When he was running his algorithm simulations on his very powerful desktop computer, they would execute in about ten hours. He was able to make adjustments or run different data sets twice a day: once during the business day and once overnight. He was trained to use the HPC system in a single morning, and he had parallelized his simulation code by the time the day was finished. His simulation on the HPC system executed in an interactive, on-demand manner on eight to sixteen processors and usually executed in less than an hour. That allowed him to execute between ten and twelve simulations per day, thereby enabling him to deliver more accurate algorithms and raise the level of confidence in the effectiveness of the algorithms. This translates into delivering much better weather radar effectiveness for his project, its sponsor, and eventually our nation. However, it is not the usual way in which HPC systems are used.

Industry Example

Let us now use this formula in an industrial production example. As with the research example and for the sake of simplicity, we will assume a time period of one year (investment at the beginning of the year and benefits by the end of the year).

We assume that an automotive firm has three high priority projects that cannot go forward without the purchase of an HPC system. If the HPC system is purchased, the three projects are very likely to be successful and are expected to bring in profits of $5.25 million, $2.0 million, and $4.5 million. In terms of costs:

- The software licenses will cost $2.5 million.
- A forecasted 90 users will get 80 hours of training on the system and software at $120 per hour.
- Ten system administrators will service the HPC system with an average annual fully burdened salary of $200,000.
- The HPC system will cost $3.0 million.

Given these estimates, we insert the values into the formula:

$$BCR = \frac{\$5.25m + \$2.0m + \$4.5m}{\$2.5m \text{ x } (90 + 80hours + \$120/hour) + (10 \text{ x } \$200k) + \$3.0m} = \frac{\$11.75m}{\$8.364m} = 1.40$$

$$IRR_{1year} = BCR - 1 = 40\%$$

From this result, the management must determine whether a one-year IRR of 40% (BCR = 1.4) is acceptable and whether the purchase will go forward.

These two examples demonstrate how the productivity (BCR) formulation can be used in different organizations. The key is to identify and estimate the benefit(s) and costs for a particular organization.

Summary

In the past several decades, HPC has made a large impact in the growth of the American economy, has helped build and maintain American competitiveness in the world economy, and has enabled many of the products and capabilities that we have today. However, Council on Competitiveness research has revealed that despite the opportunities to use HPC to increase productivity and competitiveness, many executives believe that their firms are failing to apply this technology as aggressively as possible. Some of the hindrances have occurred because of lack of talent and certain technical barriers. Another hindrance is a that many boardrooms in American companies see HPC systems as a cost of doing business without realizing the benefits that such a system can bring to the organization and the bottom line. We have presented a variety of benefits and costs that may be realized in organizations that purchase and use HPC systems. While the Research and Industry equations and examples are presented as two distinct scenarios, many actual situations may prompt a melding of the two. Overall, the examples show that HPC assets are not just cost, but that they actually can contribute to healthy earnings reports as well as more productive and efficient staff.

Acknowledgments

The Council on Competitiveness thanks the Advanced Simulation and Computing program (ASC) at the Department of Energy's National Nuclear Security Administration and the Defense Advanced Research Projects Agency's High Productivity Computing Systems program for sponsoring Council on Competitiveness research that contributed to this article. At MIT Lincoln Laboratory, this work was sponsored by the Defense Advanced Research Projects Agency under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government. The authors also wish to thank Dimitri Kusnezov, director of the ASC program, for useful discussions.

The Council on Competitiveness is an organization of the top business, university and labor leaders in the United States, responsible for influencing the course of American competitiveness on regional, national and global scales. For additional information about its High Performance Computing project and copies of reports, surveys and case studies, see www.compete.org/hpc

What's Working in HPC: Investigating HPC User Behavior and Productivity

Nicole Wolter
San Diego Supercomputing Center
Michael O. McCracken
San Diego Supercomputing Center
Allan Snavely
San Diego Supercomputing Center
Lorin Hochstein
University of Nebraska, Lincoln
Taiga Nakamura
University of Maryland, College Park
Victor Basili
University of Maryland, College Park

Introduction

High performance computing, as a field, involves a great deal of interdisciplinary cooperation. Researchers in computer science work to push the boundaries of computational power, while computational scientists use those advances to achieve increasingly detailed and accurate simulations and analysis. Staff at shared resource centers enable broad access to cutting edge systems while maintaining high system utilization.

Attempts to evaluate the productivity of an HPC system require understanding of what productivity means to all its users. While each of the above groups use HPC resources, their differing needs and experiences affect their definition of productivity. This, in turn, affects decisions about research directions and policies. Because so much is at stake, measuring and comparing productivity is not to be taken lightly. There have been many attempts to define productivity quantitatively, for example, see Kuck[1] for a definition of user productivity and Kepner[2] for a definition of the productivity of a system.

Our approach avoids the problems involved in trying to quantify productivity and instead defines the productivity of a system in terms of how well that system fulfills its intended purpose. Certainly the intended purpose of an HPC system is not just to stay busy all the time, but instead to deliver scientific results. Working with the San Diego Supercomputer Center (SDSC) and its user community, we have analyzed data from a variety of sources, including SDSC support tickets, system logs, HPC developer interviews, and productivity surveys distributed to HPC users. In order to better understand exactly how HPC systems are being used, and where the best opportunities for productivity improvements are, we have compiled a list of conjectures about HPC system usage and productivity (each originally suggested by experienced researchers in HPC) and have compared these to the usage patterns and attitudes of actual users through four studies. The seven conjectures are as follows:

- *HPC users all have similar concerns and difficulties with productivity.*
- *Users with the largest allocations and the most expertise tend to be the most productive.*
- *Computational performance is usually the limiting factor for productivity on HPC systems.*
- *Lack of publicity and education is the main roadblock to adoption of performance and parallel debugging tools.*
- *HPC programmers would require dramatic performance improvements to consider making major structural changes to their code.*
- *A computer science background is crucial to success in performance optimization.*
- *Visualization is not on the critical path to productivity in HPC in most cases.*

In the discussion that follows, we evaluate each of the conjectures. After summarizing the data sources we used and how we collected them, we present our findings and try to clarify how well each of these beliefs actually stands up to the evidence.

[1] Kuck, D. "Productivity in High Performance Computing," *International Journal of High Performance Computing Applications*, 18: 2004. pp. 489-504.

[2] Kepner, J. "High Performance Computing Productivity Model Synthesis," *International Journal of High Performance Computing Applications*, 18: 2004. pp. 505-516.

Procedure

In this study we address the above conjectures by evaluating consult tickets and job logs and further verifying the preliminary assessments, based on the previously mentioned means, with user surveys and by personal interviews of developers using current TeraGrid Sites. More in depth explanation of the studies conducted can be found at SDSC's Performance Modeling and Characterization (PMaC) web site.[3]

[3] PMaC web site for productivity study materials. PMaC Publications. http://www.sdsc.edu/PMaC/HPCS/hpcs_ productivity.html. Aug. 2006.

The initial assessment, to quantify and qualify HPC productivity, was derived from evaluating the user-submitted help tickets. The ticket sampling included all help tickets submitted to the SDSC help desk from March 2004 to March 2006. These tickets span numerous architectures. The consulting tickets enabled the identification of possible HPC resources productivity bottlenecks.

Because only 307 of the 920 registered users submitted support tickets during the time span we investigated, it was clear that ticket analysis alone did not account for all users. Attempting to include a broader set of users, we looked at system job logs of the SDSC DataStar supercomputer,[4] a 2,368-processor IBM Power4 system. We evaluated high-level trends for all 59,030 jobs run on the DataStar P655 nodes at SDSC from January 2003 to April 2006. The jobs ranged in size from 1 to 128 eight-processor nodes.

[4] SDSC User Services. SDSC DataStar user guide: http://www.sdsc.edu/user_services/datastar/

To further address some of these questions raised in the previous two studies we embarked upon a survey and interviewing campaign. We developed an interview strategy based on available references[5] and examples[6] to avoid common pitfalls in design. The questions included a general background and experience section, a development section, and a development practices and process section. The full interview script is available on the SDSC PMaC web site.[3] This strategy was extended to create a user survey. The survey had two goals; first, to get answers from questions similar to the interview script from a larger sample size, and second, to find potential future interview subjects for further investigation. The full survey is also available on the PMaC web site.[3]

[5] Robson, C. "Tactics -the Methods of Data Collection" in *Real World Research*, Blackwell Publishing, 2002.

[6] Cook, C., Pancake, C. "What users need in parallel tool support: Survey results and analysis," *IEEE Computer Society Press*, pages 40– 47, May 1994.

Analysis

The studies above identified some system and usage trends. This section draws conclusions about the conjectures made in the introduction section of this paper, and discusses how the studies influenced our understanding of, and the implications for, HPC centers and users.

Conjecture 1: *HPC users all have similar concerns and difficulties with productivity.*

While it is clear that users have a wide range of system demands, as seen in the variety of allocation and job sizes, it is not uncommon to assume that as long as users share a system, they share the same challenges staying productive. However, as details of individual usage patterns emerged, we found that not all users experienced the same problems with the same systems and tools, and some problems affected some users more than others.

Three distinct classes of users emerged based on HPC resources utilization, project scale, and the problems encountered. While these classes are clearly related to allocation size, they are defined by characteristics of their system usage.

Marquee Users: Marquee users run at very large scale, often using the full system and stressing the site policies and system resources. For these reasons they will be the most affected by node failures and are generally unable to use the interactive nodes to debug or benefit from backfill opportunities to the queue. We have named this group "Marquee" users because such projects are often used to publicize HPC centers.

Marquee users often have a consultant, or are the consultant, working on the application to improve the performance, to port to a new system or scale to larger numbers of processors. The marquee users are generally represented in our study through the job logs and personal interviews.

Normal Users: The largest class, "Normal" users, tend to run jobs using between 128 and 512 processors. Their problem size is not necessarily limited by the available resources of the systems they use. This greater flexibility in their resource usage, allows them the ability to run on smaller systems that may not be as heavily loaded. Normal users are less likely to have been forced to tune for performance optimization, or to have used performance tools. Normal users are generally represented in our study in the form of user surveys and job logs.

Small Users: Small users have a minor impact on system resources, do not tend to use large allocations, and generally run jobs with fewer than 16 processors, which are often started quickly by backfilling schedulers. In general, small users are learning parallel programming, and their productivity challenges are more likely due to unfamiliarity with the concepts of HPC computing. "Small" users are generally represented in our study in the form of help tickets and job logs.

Clear examples of the differences between the normal and marquee users can be seen in the different responses from the verbal interviews and the survey results. In some cases the interviewees and survey respondents were using the same systems and the same software, but due to different processor counts and memory requirements, their opinions of the productivity impact of system traits were very different.

The interviews with SDSC performance consultants showed that memory or I/O was the primary bottleneck for their codes. However, 8 out of 12 of our survey respondents believed that processor floating-point performance was their top bottleneck. Since some survey respondents were using the same code as our interview subjects, we attribute this discrepancy to the scale at which the marquee users run. In many cases performance bottlenecks will only become apparent when the system resources are stressed. The survey respondents, representing our normal users, report standard production runs requesting 256-512 processors, while interview subjects often use more than 1,024 processors.

The interactive running capabilities of a system provide another opportunity to distinguish user classes. Most survey respondents were able to successfully debug their codes on 1-8 processors running for less than one hour, which is feasible on current interactive nodes. Marquee users, on the other hand, must submit to the batch queue and are subject to long wait times to run very short jobs to reproduce bugs and test fixes. The marquee users expressed their frustration with the lack of on-demand computing for a large number of processors, specifically for debugging purposes.

This implies that different system policies are appropriate for different users. For example, in various phases of a project, resource demands may vary. When major development and

tuning has ceased and production has begun, a policy that would allow the entire system to be reserved could be a major improvement to productivity. One marquee user gave the example that it would take a week running around the clock to get one simulation completed and that repeatedly waiting in the queue is wasteful and can extend this process by at least factor of four. In contrast, another researcher was uncomfortable with having dedicated time due to the risk of wasting allocation while fixing problems, preferring a high-priority queue reservation policy instead. System design and site policies should reflect the different types of users and stages of development.

Conjecture 2: ***Users with the largest allocations and most experience are the most productive.***

A common policy of HPC centers is to give preference in queue priority to jobs with higher node requirements, which encourages users to make full use of rare high capacity systems. Large users may also receive more personal attention, even to the point of having a dedicated consultant working on their programs. Does this conclusively imply that larger users are the most productive?

Through our interviews it emerged that productivity, in terms of generating scientific results, is just as difficult to achieve for large users, if not more so, than for smaller users. Queue wait time, reliability and porting issues all cause major problems for large users. Large-scale programs often push the limits of systems and therefore run into problems not often seen at lower scale, such as system and code performance degradation and system reliability problems.

Evaluating the queue based on job logs can be complicated. There are a number of factors that could affect queue priority. Some of the most influential variables are length of runtime, processor count requested, the size of a user's unspent allocation, as well as site administrators' ability to change the priority of individual jobs directly.

Processor Count	Average	Median	Maximum	Minimum
8	8.87	0.06	858.93	0.00
64	12.64	0.45	792.33	0.00
128	24.12	2.56	752.67	0.00
256	21.13	3.51	415.69	0.00
512	19.79	3.81	266.44	0.00
1024	23.99	9.07	205.49	0.00

Table 1. Wait time of jobs 2003-06 on DataStar, in hours.

Processor Count	Average	Median	Maximum	Minimum
8	2.43	0.52	624.89	0.00
64	2.49	0.69	49.42	0.00
128	3.84	0.63	105.01	0.00
256	3.33	0.46	100.06	0.00
512	3.13	0.40	19.44	0.02
1024	2.60	0.31	18.01	0.03

Table 2. Run time of jobs 2003-06 on DataStar, in hours.

Processor Count	Average	Median	Maximum	Minimum
8	8.02	1.15	22666.83	1.00
64	19.22	1.66	11826	1.00
128	68.71	3.31	5137.17	1.00
256	44.42	5.35	3809.40	1.00
512	53.53	6.00	3924.50	1.00
1024	90.69	11.68	2800.39	1.00

Table 3. Expansion factor for jobs 2003-06 on DataStar.

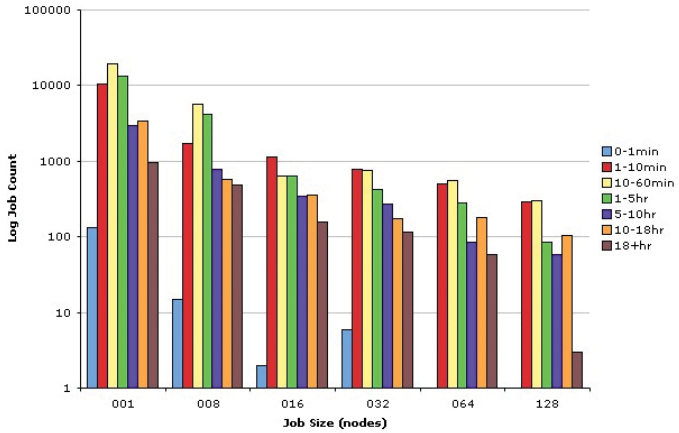

Figure 1. Run timedistribution, grouped by job size.

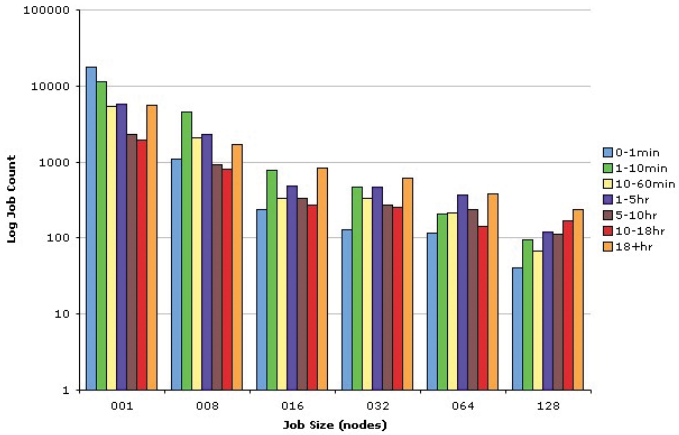

Figure 2. Wait time distribution, grouped by job size.

In support of our initial statement, Table 1 displays some trends that are not surprising. On average small jobs have shorter wait times. The average wait time for jobs requesting only one node was nine hours while the average wait time for 128 nodes (1024 processors) was approximately 24 hours. In contrast, the inverse trend is evident in maximum wait times; as the number of nodes increased, the maximum wait time decreased. The maximum wait time for a 128 node job was 17 days and the maximum wait time for one node was 71 days. The median tells us a similar story to the average. The trends in run time and wait time are also visible in the

histograms shown in Figure 1 and Figure 2, grouped by job size. Table 2 shows statistics for the job run time.

The expansion factor ((waittime + runtime) / runtime) data in Table 3 reinforces the trends established by the wait times and also displays how the queue favors large allocations. The average expansion factor increases as the number of nodes increases up to 256 processors, where there is a dip in the expansion factor curve. This appears to show a preference for large jobs in the scheduler beginning at 256 processors. On closer inspection, the trend was influenced by a single high priority account with a large allocation, running many jobs on 256 processors. The actual priority increase for "large" jobs begins at 512 processors or more. Also note the high expansion factor for 1024 processor runs. While it would seem to show that users running on large portions of the system are not very productive, we see that this effect is due to a large number of such jobs with very short run time. 632 of the 839 jobs that ran on 128 nodes (1024 processors) ran for less than two hours, and most of those jobs were part of a benchmarking effort. The average expansion factor for jobs that were not benchmarking was 9.58, and the median was 10.05, which more clearly shows the scheduler favoring large jobs as intended.

Although the analysis above does show that marquee users receive priority for using large portions of the system and are therefore not disproportionately hurt by queue wait time, they do face a serious productivity bottle-neck in the inability to get a large-enough allocation on one system, or on the right system for their task. Due to site policies, single-site allocations often have a limit. In some cases, users were forced to port codes to several systems due to allocation limits, resource constraints, or system and support software compatibility issues. One developer told us of having to port one program to four different systems during the period of one project. Another told us of having to port to at least two systems for an individual run to be able to capitalize on the different features supplied on different systems.

Porting itself can be time consuming and difficult. We heard about the time required to complete a port, ranging from less than an hour to many days, and in some cases the successful port was never achieved. Once the code is ported, these multi-platform scenarios tend to require tedious manual effort from the user to move data files around, convert formats, and contend with the problems of working on many sites with different policies, support, and capabilities.

Conjecture 3: *Time to solution is the limiting factor for productivity on HPC systems.*

While the initial migration of a project to HPC systems is certainly due to expanding resource requirements, we found across all four studies that the HPC users represented in our samples treat performance as a constraint rather than a goal to be maximized. Code performance is very important to them only until it is "good enough" to sustain productivity with the allocation they have, and then it is no longer a priority. In economics literature, this is called satisficing,[7] and while it is not surprising in retrospect, it is important to keep this distinction in mind when thinking about what motivates HPC users. The in-depth system log evaluation showed most users not taking advantage of the parallel capacity available to them. Out of 2,681 jobs run in May 2004, 2,261 executed on one eight-processor node. This accounted for 41% of all the CPU hours utilized on the system. Furthermore, the job logs show that in between 2004 and 2006, 1,706 jobs out of 59,030 jobs on DataStar were removed from the batch nodes for exceeding the maximum job time limit of 18 hours. Of these jobs, 50% were running on fewer than eight processors. Given access to a resource with thousands of processors, the majority of users choose to reduce the priority of performance tuning as soon as possible, indicating that they have likely found a point at which they feel they can be productive.

[7] Simon, H. "Rational choice and the structure of the environment," *Psychological Review*, Volume 63: 2002. pp. 129–138.

The support ticket evaluation tells a similar story. Many users requested longer run times, and fewer than ten users requested anything related to performance. Furthermore, some of the users requesting longer run times did not have checkpoint restart capabilities in their code, and many were running serial jobs.

While performance is certainly the original reason for moving to an HPC platform, the attitudes and statements of our interview subjects reinforces the assertion that it is not a primary goal of theirs. When asked about productivity problems, performance was generally taken for granted while other issues such as reliability, file system capacity and storage policies, as well as queue policies and congestion, were discussed in depth. Performance problems are just one of many barriers to productivity, and focus on performance at the expense of other improvements should be avoided.

Conjecture 4: ***Lack of publicity is the main roadblock to adoption of performance and parallel debugging tools.***

While it is true that many users are unfamiliar with the breadth of performance tools available, a significant portion of users simply prefer not to use them. Reasons given by interviewees included problems scaling tools to large number of processors, unfamiliar and inefficient GUI interfaces, steep learning curves, or the overwhelming detail provided in the displayed results.

Unsurprisingly, the performance optimization consultants were the most likely to use tools. However even they often opted for the seemingly more tedious path of inserting print statements with timing calls and recompiling rather than learning to use performance tools.

The situation for parallel debuggers was no more encouraging. Only one interviewee used readily available parallel debugging tools. However, other developers indicated that if the debugging tools were consistent and easy to use at scales of up to hundreds of processors, they would have been used. In their current state, parallel debugging tools are considered difficult or impossible to use by the interviewees who had tried them.

It is clear that there is a long way to go before common HPC programming practice embraces the powerful tools that the research community has built. Certainly there is a lack of motivation on the part of many HPC programmers to learn new tools that will help them with the task of performance optimization, which is not always their main priority. Aside from the obvious issue of acceptable performance at large scale, it seems that continuing to strive for tools that are easier to learn and use is important for improved adoption. As discussed by Pancake,[8] it is important to design tools from a user's perspective and with early user involvement in the design process for the design to be effective.

[8] Pancake, C. "Can Users Play an Effective Role in Parallel Tool Research?" in *Tools and Environments for Parallel Scientific Computing*, SIAM, 1996.

Conjecture 5: ***HPC programmers would demand dramatic performance improvements to consider major structural changes to their code.***

Programmers we surveyed showed a surprising indifference to the risks involved in rewriting often very large code bases in a new language or changing a communication model. Although many successful projects last for tens of years without making significant changes, eight of the 12 Summer Institute attendees responded that they were willing to make major code changes for surprisingly small system performance improvements or policy changes.

Many of the codes discussed were larger than 100,000 lines of code (LOC). Though all eight codes over 10,000 LOC had checkpoint restart capabilities, three users were willing to rewrite code for the ability to run a single job longer, two requesting only a factor of two job runtime limit extension. The respondents were likely unaware that most sites will allow such small exceptions on a temporary basis. Of the three respondents with more than 100,000 lines of code, only one considered such a major change, requesting in return a factor of ten improvement in either processor speed or job time limits. The results imply that although performance, judged by time to solution, is not always the main goal of HPC users, users would be very receptive to work for guaranteed system and policy changes.

While we see some conflicting responses, it might be possible for HPC centers to capitalize on these attitudes to get users to use profiling tools and spend some effort to improve individual code performance, and ultimately queue wait times, by giving minor compensation to cooperative users. Also, in some cases, it would seem that there is a gap between what users want and what they think they can get. This gap could be bridged with improved communication with users regarding site policies and resource options.

Conjecture 6: ***A computer science background is crucial to success in performance optimization.***

It may seem straightforward to say that if you want to improve your code performance you should take it to a computer scientist or software engineer. However, of the developers on successful projects interviewed, only one had a formal computer science background. In fact, many of these successful projects are operating at very large scale without any personnel with a formal computer science or software engineering background.

There was a general consensus among interviewees that without competence in the domain science, changes to the project are destined for failure. Two subjects actually contracted out serious code bugs and made use of library abstractions to avoid in depth knowledge of parallel programming. One such library was written by a computer scientist, and once it matured, users were able to achieve high performance using it without a parallel computing background.

HPC centers and software developers should keep in mind that their target audience is not computer scientists, but rather physical scientists with a primary focus on scientific results.

Conjecture 7: ***Visualization is not on the critical path to productivity in HPC.***

Every interview subject and all but three survey respondents used visualization regularly to validate the results of runs. For those projects, any problems and productivity bottlenecks that affect visualization have as much impact on overall productivity as poor computational performance during a run.

Such bottlenecks can include problems with available visualization and image conversion software and with finding capable resources for post-processing and visualization rendering. As evidence, one interviewee stated a desire for a large head node with a large amount of RAM to handle rendering. This implies that users' productivity could be greatly enhanced by a dedicated visualization node that shared a file system with computation nodes.

Conclusion

In performance tuning, it is common to hear advice to profile before attempting an optimization, because often the true bottleneck is a surprise. In this paper, we have attempted to find where the true bottlenecks in HPC productivity are by investigating general trends in user behavior and by asking users directly.

Our analysis shows that, in general, HPC user needs are heterogeneous with respect to HPC resource usage patterns, requirements, problems experienced and general background knowledge. These factors combined dictate how individual productivity is viewed and clarifies the motivations for focusing on performance. Our research also gave us insight into the tools and techniques currently available and to what extent they have been embraced in the community.

Future research will evaluate the effectiveness of some of the techniques discussed by users. Based on our findings, we will continue to evaluate HPC community feedback to build a general consensus on the problems that affect productivity and where research time and system money can best be spent to allow these systems to live up to their promise and continue to support leading-edge research.

Acknowledgments

The authors thank the subjects of our interviews and surveys for their cooperation. We also thank the staff of SDSC user services, Douglass Post, and Nick Wright for help coordinating various parts of the study and for their insight.

Design and Implementation of the HPC Challenge Benchmark Suite

Piotr Luszczek
University of Tennessee
Jack Dongarra
University of Tennessee
Oak Ridge National Laboratory
Jeremy Kepner
MIT Lincoln Lab

1. Introduction

The HPCC benchmark suite was initially developed for the DARPA's HPCS program[1] to provide a set of standardized hardware probes based on commonly occurring computational software kernels. The HPCS program has initiated a fundamental reassessment of how we define and measure performance, programmability, portability, robustness and, ultimately, productivity in the high-end domain. Consequently, the suite was aimed to both provide conceptual expression of the underlying computation as well as be applicable to a broad spectrum of computational science fields. Clearly, a number of compromises must have lead to the current form of the suite given such a broad scope of design requirements. HPCC was designed to approximately bound computations of high and low spatial and temporal locality (see Figure 1 which gives the conceptual design space for the HPCC component tests). In addition, because the HPCC tests consist of simple mathematical operations, this provides a unique opportunity to look at language and parallel programming model issues. As such, the benchmark is to serve both the system user and designer communities.[2]

[1] Kepner, J. "HPC productivity: An overarching view," International Journal of High Performance Computing Applications, 18(4), November 2004.

[2] Kahan, W. "The baleful effect of computer benchmarks upon applied mathematics, physics and chemistry," The John von Neumann Lecture at the 45th Annual Meeting of SIAM, Stanford University, 1997.

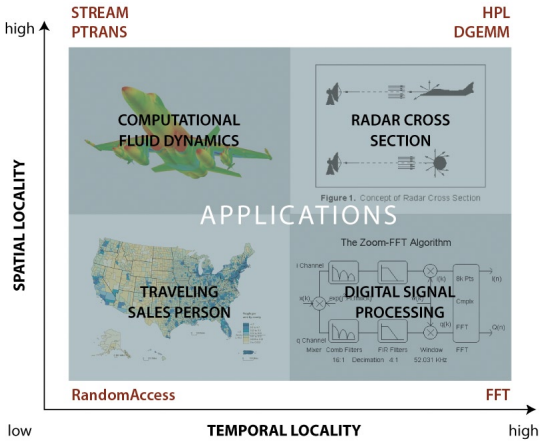

Figure 1. The application areas targeted by the HPCS Program are bound by the HPCC tests in the memory access locality space.

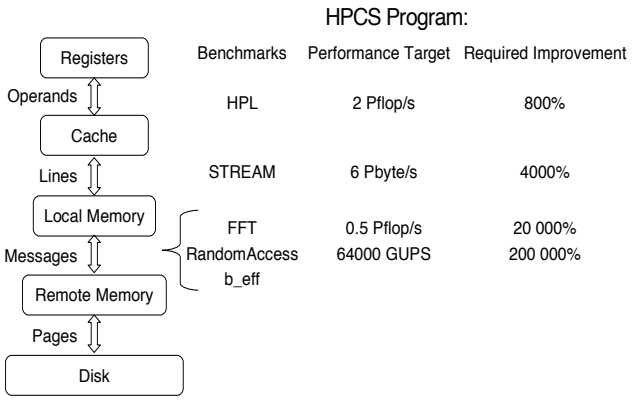

Figure 2. HPCS program benchmarks and performance targets.

Finally, Figure 2 shows a generic memory subsystem and how each level of the hierarchy is tested by the HPCC software and what are the design goals of the future HPCS system - these are the projected target performance numbers that are to come out of the winning HPCS vendor designs.

2. The TOP500 Influence

Rank	Name	Rmax	HPL	PTRANS	STREAM	FFT	RANDA	Latency	Bandwidth
1	BlueGene/L	280.6	259.2	4665.9	160	2311	35.47	5.92	0.16
2	BlueGene W	91.3	83.9	171.5	50	1235	21.61	4.70	0.16
3	ASC Purple	75.8	57.9	553.0	44	842	1.03	5.11	3.22
4	Columbia	51.9	46.8	91.3	21	230	0.25	4.23	1.39
9	Red Storm	36.2	33.0	1813.1	44	1118	1.02	7.97	1.15

Table 1. All of the top-10 entries of the 27TH TOP500 list that have results in the HPCC database.

The most commonly known ranking of supercomputer installations around the world is the TOP500 list.[3] It uses the equally famous LINPACK Benchmark[4] as a single figure of merit to rank 500 of the world's most powerful supercomputers. The often raised issue of the relation between TOP500 and HPCC can simply be addressed by recognizing all the positive aspects of the former. In particular, the longevity of TOP500 gives an unprecedented view of the high-end arena across the turbulent times of Moore's law[5] rule and the process of emerging of today's prevalent computing paradigms. The predictive power of TOP500 will have a lasting influence in the future as it did in the past. While building on the legacy information, HPCC extends the context of the HPCS goals and can serve as a valuable tool for performance analysis. Table 1 shows an example of how the data from the HPCC database can augment the TOP500 results.

[3] Meuer, H. W., Strohmaier, E., Dongarra, J. J., Simon, H. D. *TOP500 Supercomputer Sites*, 28th edition, November 2006. (The report can be downloaded from http://www.netlib.org/benchmark/top500.html).

[4] Dongarra, J. J., Luszczek, P., Petitet, A. "The LINPACK benchmark: Past, present, and future," *Concurrency and Computation: Practice and Experience*, 15:1-8, 2003.

[5] Moore, G. E. "Cramming more components onto integrated circuits," *Electronics*, 38(8), April 19 1965.

3. Short History of the Benchmark

The first reference implementation of the code was released to the public in 2003. The first optimized submission came in April 2004 from Cray using their recent X1 installation at Oak Ridge National Lab. Every since then Cray has championed the list of optimized submissions. By the time of the first HPCC birds-of-feather at the 2004 Supercomputing conference in Pittsburgh, the public database of results already featured major supercomputer makers - a sign that vendors noticed the benchmark. At the same time, a bit behind the scenes, the code was also tried by government and private institutions for procurement and marketing purposes. The highlight of 2005 was the announcement of a contest: the HPCC Awards. The two complementary categories of the competition emphasized performance and productivity - the very goals of the sponsoring HPCS program. The performance-emphasizing Class 1 award drew attention to the biggest players in the supercomputing industry, which resulted in populating the HPCC database with most of the top-10 entries of TOP500 (some of which even exceeding performance reported in the TOP500 - a tribute to HPCC's continuous results' update policy). The contestants competed to achieve highest raw performance in one of the four tests: HPL, STREAM, RANDA, and FFT. The Class 2 award, by solely focusing on productivity, introduced subjectivity factor to the judging but also to the submitter criteria of what is appropriate for the contest. As a result, a wide range of solutions were submitted spanning various programming languages (interpreted and compiled) and paradigms (with explicit and implicit parallelism). It featured openly available as well as proprietary technologies, some of which were arguably confined to niche markets and some that are widely used. The financial incentives for entering turned out to be all that was needed, as the HPCC seemed to have enjoyed enough recognition among the high-end community. Nevertheless, HPCwire kindly provided both press coverage as well as cash rewards for four winning contestants of Class 1 and the winner of Class 2. At the

HPCC's second birds-of-feather session during the SC|05 conference in Seattle, the former class was dominated by IBM's BlueGene/L from Lawrence Livermore National Lab while the latter was split among MTA pragma-decorated C and UPC codes from Cray and IBM, respectively.

4. The Benchmark Tests' Details

Extensive discussion and various implementations of the HPCC tests are given elsewhere.[6][7][8] However, for the sake of completeness, this section lists the most important facts pertaining to the HPCC tests' definitions.

All calculations use *double precision* floating-point numbers as described by the IEEE 754 standard[9] and no mixed precision calculations[10] are allowed. All the tests are designed so that they will run on an arbitrary number of processors (usually denoted as p). Figure 3 shows a more detailed definition of each of the seven tests included in HPCC. In addition, it is possible to run the tests in one of three testing scenarios to stress various hardware components of the system. The scenarios are shown in Figure 4.

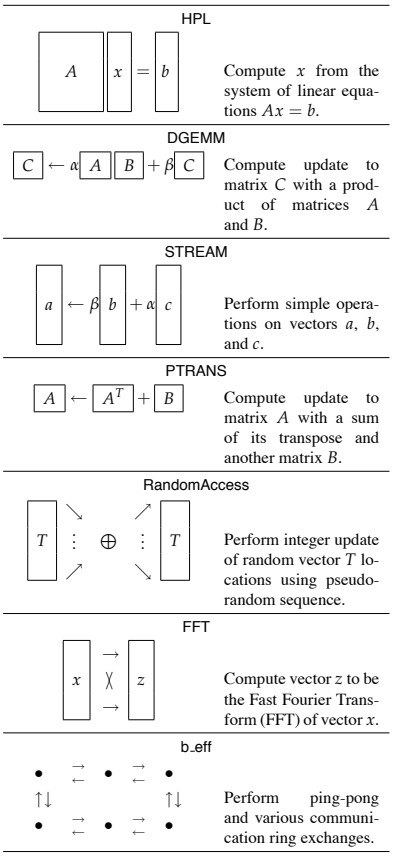

Figure 3. Detail description of the HPCC component tests (*A, B, C* - matrices, *a, b, c, x, z* – vectors, α, β - scalars, *T* - array of 64-bit integers).

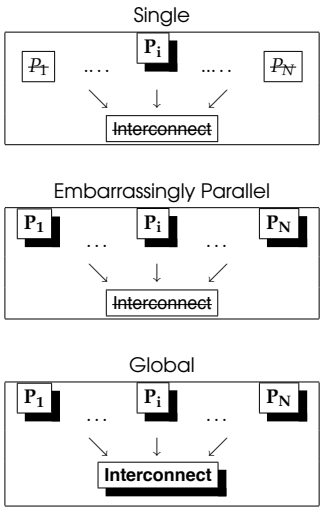

Figure 4. Testing scenarios of the HPCC components.

[6] Dongarra, J., Luszczek, P. "Introduction to the HPC Challenge benchmark suite," *Technical Report UT-CS-05-544*, University of Tennessee, 2005.

[7] Luszczek, P., Dongarra, J. "High performance development for high end computing with Python Language Wrapper (PLW)," *International Journal of High Performance Computing Applications*, 2006. Accepted to Special Issue on High Productivity Languages and Models.

[8] Travinin, N., Kepner, J. "pMatlab parallel Matlab library," *International Journal of High Performance Computing Applications*, 2006. Submitted to Special Issue on High Productivity Languages and Models.

[9] ANSI/IEEE Standard 754-1985. "Standard for binary floating point arithmetic," *Technical report*, Institute of Electrical and Electronics Engineers, 1985.

[10] Langou, J., Langou, J., Luszczek, P., Kurzak, J., Buttari, A., Dongarra, J. "Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy," *In Proceedings of SC06*, Tampa, Florida, Nomveber 11-17 2006.
See http://icl.cs.utk.edu/iter-ref.

5. Benchmark Submission Procedures and Results

The reference implementation of the benchmark may be obtained free of charge at the benchmark's web site.[11] The reference implementation should be used for the base run: it is written in portable subset of ANSI C[12] using hybrid programming model that mixes OpenMP[13 14] threading with MPI[15 16 17] messaging. The installation of the software requires creating a script file for Unix's `make(1)` utility. The distribution archive comes with script files for many common computer architectures. Usually, few changes to one of these files will produce the script file for a given platform. The HPCC rules allow only standard system compilers and libraries to be used through their supported and documented interface and the build procedure should be described at submission time. This ensures repeatability of the results and serves as an educational tool for end users that wish to use the similar build process for their applications.

After, a successful compilation the benchmark is ready to run. However, it is recommended that changes be made to the benchmark's input file that describes the sizes of data to use during the run. The sizes should reflect the available memory on the system and the number of processors available for computations.

There must be one baseline run submitted for each computer system entered in the archive. There may also exist an optimized run for each computer system. The baseline run should use the reference implementation of HPCC and, in a sense, it represents the scenario when an application requires use of legacy code - a code that cannot be changed. The optimized run allows developers to perform more aggressive optimizations and use system-specific programming techniques (e.g., languages, messaging libraries, etc.) but at the same time still gives the verification process enjoyed by the base run.

[11] HPCC - http://icl.cs.utk.edu/hpcc/

[12] Kernighan, B. W., Ritchie, D. M.. *The C Programming Language*. Prentice-Hall, 1978.

[13] OpenMP: Simple, portable, scalable SMP programming. http://www.openmp.org/.

[14] Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., Menon, R. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2001.

[15] Message Passing Interface Forum. "MPI: A Message-Passing Interface Standard," *The International Journal of Supercomputer Applications and High Performance Computing*, 8, 1994.

[16] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard (version 1.1), 1995. Available at: http://www.mpi-forum.org/.

[17] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface, 18 July 1997. Available at http://www.mpi-forum.org/docs/mpi-20.ps.

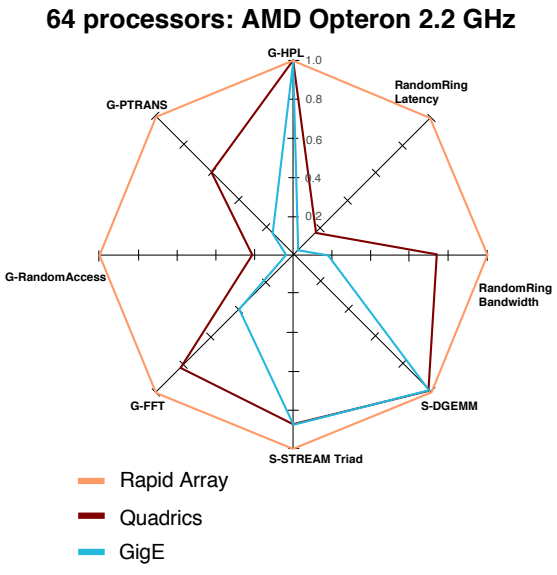

Figure 5. Sample kiviat diagram of results for three different interconnects that connect the same processors.

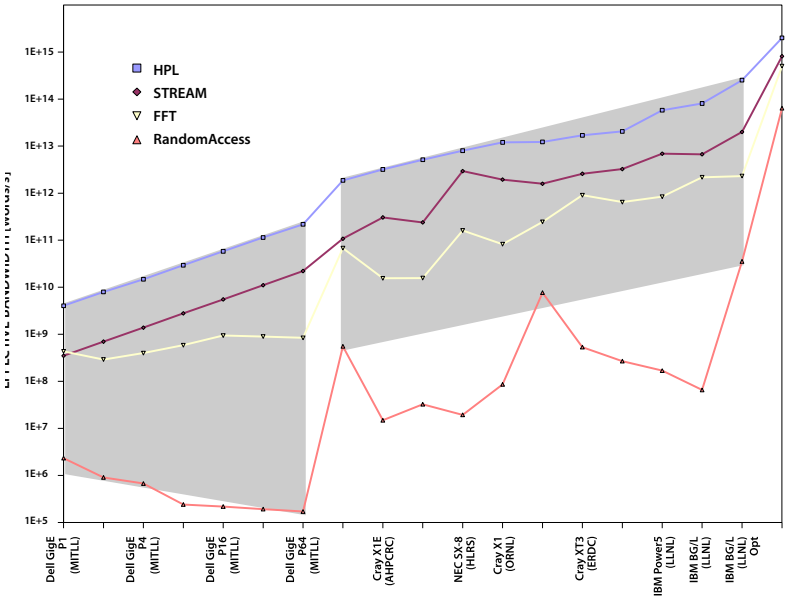

Figure 6. Sample interpretation of the HPCC results.

All of the submitted results are publicly available after they have been confirmed by email. In addition to the various displays of results and raw data export the HPCC website also offers a kiviat chart display to visually compare systems using multiple performance numbers at once. A sample chart that uses actual HPCC results' data is shown in Figure 5.

Figure 6 show performance results of currently operating clusters and supercomputer installations. Most of the results come from the HPCC public database.

6. Scalability Considerations

Name	Generation	Computation	Communication	Verification	Per-processor data
HPL	n^2	n^3	n^2	n^2	p^{-1}
DGEMM	n^2	n^3	n^2	1	p^{-1}
STREAM	m	m	1	m	p^{-1}
PTRANS	n^2	n^2	n^2	n^2	p^{-1}
RandomAccess	m	m	m	m	p^{-1}
FFT	m	$m \log_2 m$	m	$m \log_2 m$	p^{-1}
b_eff	1	1	p^2	1	1

Table 2. Time complexity formulas for various phases of the HPCC tests (m and n correspond to the appropriate vector and matrix sizes, p is the number of processors.

There are a number of issues to be considered for benchmarks such as HPCC that have scalable input data to allow for an arbitrary sized system to be properly stressed by the benchmark run. Time to run the entire suite is a major concern for institutions with limited resource allocation budgets. Each component of HPCC has been analyzed from the scalability standpoint and Table 2 shows the major time complexity results. In following, it is assumed that:

- M is the total size of memory,
- m is the size of the test vector,
- n is the size of the test matrix,
- p is the number of processors,
- t is the time to run the test.

Clearly any complexity formula that grows faster than linearly with respect to any of the system sizes is a cause of potential problem time scalability issue. Consequently, the following tests have to be addressed:

- HPL because it has computational complexity $O(n^3)$.
- DGEMM because it has computational complexity $O(n^3)$.
- b_eff because it has communication complexity $O(p^2)$.

The computational complexity of HPL of order $O(n^3)$ may cause excessive running time because the time will grow proportionately to a high power of total memory size:

$$t_{\mathrm{HPL}} \sim n^3 = \left(n^2\right)^{3/2} \sim M^{3/2} = \sqrt{M^3} \qquad (1)$$

To resolve this problem, we have turned to the past TOP500 data and analyzed the ratio of *Rpeak* to the number of bytes for the factorized matrix for the first entry on all the lists. It turns

out that there are on average 6±3 Gflop/s for each matrix byte. We can thus conclude that the performance rate of HPL remains constant over time ($r_{HPL} \sim M$), which leads to:

$$t_{HPL} \sim \frac{n^3}{r_{HPL}} \sim \frac{\sqrt{M^3}}{M} = \sqrt{M} \qquad (2)$$

which is much better than (1).

There seems to be a similar problem with the DGEMM as it has the same computational complexity as HPL but fortunately, the n in the formula related to a single process memory size rather than the global one and thus there is no scaling problem.

Lastly, the b_eff test has a different type of problem: its communication complexity is $O(p^2)$ which is already prohibitive today as the number of processes of the largest system in the HPCC database is 131072. This complexity comes from the ping-pong component of b_eff that attempts to find the weakest link between all nodes and thus, theoretically, needs to look at the possible process pairs. The problem was remedied in the reference implementation by adapting the runtime of the test to the size of the system tested.

7. Conclusions

No single test can accurately compare the performance of any of today's high-end systems let alone any of those envisioned by the HPCS program in the future. Thus, the HPCC suite stresses not only the processors, but the memory system and the interconnect. It is a better indicator of how a supercomputing system will perform across a spectrum of real-world applications. Now that the more comprehensive HPCC suite is available, it could be used in preference to comparisons and rankings based on single tests. The real utility of the HPCC benchmarks are that architectures can be described with a wider range of metrics than just flop/s from HPL. When looking only at HPL performance and the TOP500 list, inexpensive build-your-own clusters appear to be much more cost effective than more sophisticated parallel architectures. But the tests indicate that even a small percentage of random memory accesses in real applications can significantly affect the overall performance of that application on architectures not designed to minimize or hide memory latency. The HPCC tests provide users with additional information to justify policy and purchasing decisions. We expect to expand and perhaps remove some existing benchmark components as we learn more about the collection.

This work was supported in part by the DARPA, NSF, and DOE through the DARPA HPCS program under grant FA8750-04-1-0219 and SCI-0527260.

Experiments to Understand HPC Time to Development

Lorin Hochstein
University of Nebraska
Taiga Nakamura
University of Maryland, College Park
Victor R. Basili
University of Maryland, College Park
Fraunhofer Center, Maryland
Sima Asgari
University of Maryland, College Park
Marvin V. Zelkowitz
University of Maryland, College Park
Fraunhofer Center, Maryland
Jeffrey K. Hollingsworth
University of Maryland, College Park
Forrest Shull
Fraunhofer Center, Maryland
Jeffrey Carver
Mississippi State University
Martin Voelp
University of Maryland, College Park
Nico Zazworka
University of Maryland, College Park
Philip Johnson
University of Hawaii

1. Introduction

Much of the literature in the Software Engineering community concerning programmer productivity was developed with assumptions that do not necessarily hold in the High Performance Computing (HPC) community:

1. In scientific computation insights culled from results of one program version often drives the needs for the next. The software itself is helping to push the frontiers of understanding rather than the software being used to automate well-understood tasks.
2. The requirements often include conformance to sophisticated mathematical models. Indeed, requirements may often take the form of an executable model in a system such as Mathematica, and the implementation involves porting this model to HPC systems.
3. "Usability" in the context of an HPC application development may revolve around optimization to the machine architecture so that computations complete in a reasonable amount of time. The effort and resources involved in such optimization may exceed initial development of the algorithm.

Due to these unique requirements, traditional software engineering approaches for improving productivity may not be directly applicable to the HPC environment.

As a way to understand these differences, we are developing a set of tools and protocols to study programmer productivity in the HPC community. Our initial efforts have been to understand the effort involved and defects made in developing such programs. We also want to develop models of workflows that accurately explain the process that HPC programmers use to build their codes. Issues such as time involved in developing serial and parallel versions of a program, testing and debugging of the code, optimizing the code for a specific parallelization model (e.g., MPI, OpenMP) and tuning for a specific machine architecture are all topics of study. If we have those models, we can then work on the more crucial problems of what tools and techniques better optimize a programmer's performance to produce quality code more efficiently.

Since 2004 we have been conducting human-subject experiments at various universities across the U.S. in graduate level HPC courses (Figure 1). Graduate students in a HPC class are fairly typical of a large class of novice HPC programmers who may have years of experience in their application domain but very little in HPC-style programming. Multiple students are routinely given the same assignment to perform, and we conduct experiments to control for the skills of specific programmers (e.g., experimental meta-analysis) in different environments. Due to the relatively low costs, student studies are an excellent environment to debug protocols that might be later used on practicing HPC programmers.

Limitations of student studies include the relatively short programming assignments due to the limited time in a semester and the fact these assignments must be picked for the educational value to the students as well as their investigative value to the research team.

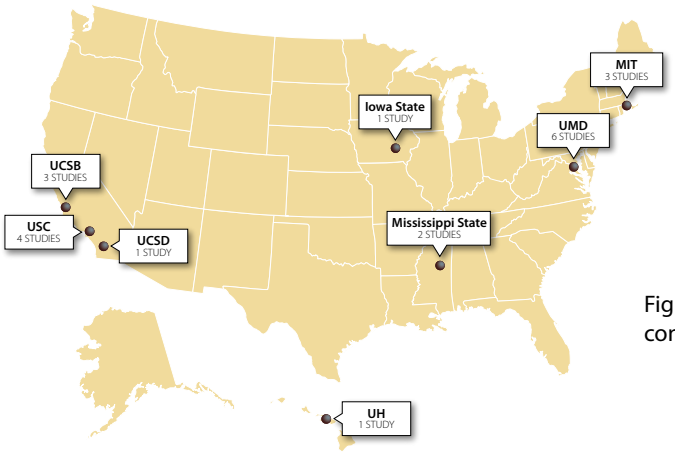

Figure 1. Classroom studies conducted.

In this article, we present both the methodology we have developed to investigate programmer productivity issues in the HPC domain (Section 2), some initial results of studying productivity of novice HPC programmers (Section 3), and current plans for improving the process in the future (Section 4).

2. Experiment methodology

In each class, we obtained consent from students to be part of our study. There is a requirement at every U.S. institution that studies involving human subjects must be approved by that university's Institutional Review Board (IRB). The nature of the assignments was left to the individual instructors for each class since instructors had individual goals for their courses and the courses themselves had different syllabi. However, based on previous discussions as part of this project, many of the instructors used the same assignments (Table 1), and we have been collecting a database of project descriptions as part of our Experiment Manager website (See Section 4). To ensure that the data from the study would not impact students' grades (and a requirement of almost every IRB), our protocol quarantined the data collected in a class from professors and teaching assistants for that class until final grades had been assigned.

Embarrassingly parallel:	Buffon-Laplace needle problem, Dense matrix-vector multiply
Nearest neighbor:	Game of life, Sharks & fishes, Grid of resistors, Laplace's equation, Quantum dynamics
All-to-all:	Sparse matrix-vector multiply, Sparse conjugate gradient, Matrix power via prefix
Shared memory:	LU decomposition, Shallow water model, Randomized selection, Breadth-first search
Other:	Sorting

Table 1. Sample programming assignments

We need to measure the time students spend working on programming assignments with the task that they are working on at that time (e.g., serial coding, parallelization, debugging, tuning). We used three distinct methods: (1) explicit recording by subject in diaries (either paper or web-based); (2) implicit recording by instrumenting the development environment; and (3) sampling by an operating system installed tool (e.g., Hackystat[1]). Each of these approaches has strengths and limitations. But significantly, they all give different answers. After conducting a series of tests using variations on these techniques, we settled on a hybrid approach that combines diaries with an instrumented programming environment that captures a time-stamped record of all

[1] Johnson, P. M., Kou, H., Agustin, J. M., Zhang, Q., Kagawa, A., Yamashita, T. "Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from Hackystat-UH," *Proceedings of the 2004 International Symposium on Empirical Software Engineering*, Los Angeles, California, August, 2004.

compiler invocations (including capture of source code), all programs invoked by the subject as a shell command, and interactions with supported editors. Elsewhere,[2] we describe the details of how we gather this information and convert it into a record of programmer effort.

After students completed an assignment, the data was transmitted to the University of Maryland, where it was added to our Experiment Manager database. Looking at the database allows post-project analysis to be conducted to study the various hypotheses we have collected via our folklore collection process.

For example, given workflow data from a set of students, the following hypotheses that are the subjective opinion of many in the HPCS community, collected via surveys at several HPCS meetings, can be tested:[3]

Hyp 1: *The average time to fix a defect due to race conditions will be longer in a shared memory program compared to a message-passing program.* To test this hypothesis we can measure the time to fix defects due to race conditions.

Hyp. 2: *On average, shared memory programs will require less effort than message passing, but the shared memory outliers will be greater than the message passing outliers.* To test this hypothesis we measure the total development time.

Hyp. 3: *There will be more students who submit incorrect shared memory programs compared to message-passing programs.* To test this hypothesis we can measure the number of students who submit incorrect solutions.

Hyp. 4: *An MPI implementation will require more code than an OpenMP implementation.* To test this hypothesis we can measure the size of code for each implementation.

The classroom studies are the first part of a larger series of studies we are conducting (Figure 2). We first run pilot studies with students. We next conduct classroom studies, then move onto controlled studies with experienced programmers, and finally conduct experiments in situ with development teams. Each of these steps contributes to our testing of hypotheses by exploiting the unique aspects of each environment (i.e., replicated experiments in classroom studies and multi-person development with in situ teams). We can also compare our results with recent studies of existing HPC codes.[4]

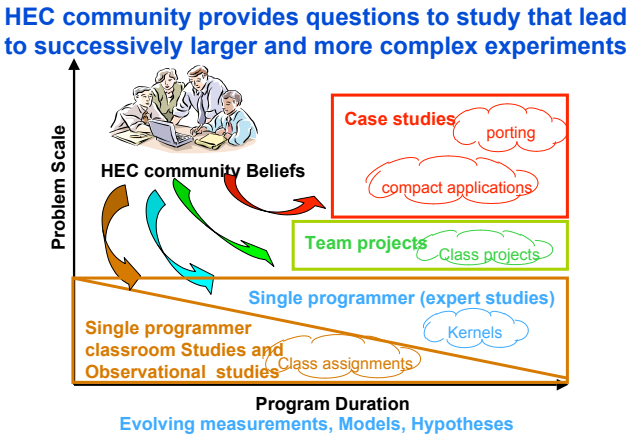

Figure 2. Research Plan.

[2] Hochstein, L., Basili, V., Zelkowitz, M., Hollingsworth, J., Carver, J. "Combining self-reported and automatic data to improve effort measurement," *Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2005)*, Lisbon, Portugal, September 2005, 356-365.

[3] Asgari, S., Hochstein, L., Basili, V., Zelkowitz, M., Hollingsworth, J., Carver, J., Shull, F. "Generating Testable Hypotheses from Tacit Knowledge for High Productivity Computing," *2nd International Workshop on Software Engineering for High Performance Computing System Applications*, (May, 2005) St. Louis, MO, 17-21.

[4] Post, D., Kendall, R.P., Whitney, E. "Case study of the Falcon Project," *Second International Workshop on Software Engineering for High Performance Computing Systems Applications*, St. Louis, MO, 2005.

Defect studies

As part of our effort to understand development issues, our classroom experiments have moved beyond effort analysis and have started to look at the impact of defects (e.g., incorrect or excessive synchronization, incorrect data decomposition) on the development process. By understanding how, when, and the kind of defects that appear in HPC codes, tools and techniques can be developed to mitigate these risks to improve the overall workflow. As we have shown,[2] automatically determining workflow is not precise, so we are working on a mixture of process activity (e.g., coding, compiling, executing) with source code analysis techniques. The process of defect analysis we are building consists of the following main activities:

Analysis:

1. Analyze successive versions of the developing code looking for patterns of changes represented by successive code versions (e.g., defect discovery, defect repair, addition of new functionality).
2. Record the identified changes.
3. Develop a classification scheme and hypotheses.

For example, a small increase in source code, following a failed execution and following a large code insertion, could represent the pattern of the programming adding new functionality, followed by a test and then defect correction. Syntactic tools that find specific defects can be used to aid the human-based heuristic search for defects.

Verification:

We then need to analyze these results at various levels. Verification consists of the following steps, among others:

1. If we can somehow obtain the "true" defect sets, we can directly compare our analysis results with them to evaluate the analysis results quantitatively.
2. Multiple analysts can independently analyze the source code and record identified defects.
3. Examine individual instances of defects to check if each defect is correctly captured and documented.
4. Provide defect instances and classify them into one of the given defect types. This can be used to check the consistency of the classification scheme.

Nakamura, et al. explore this defect methodology in greater detail.[5]

[5] Nakamura, T., Hochstein, L., Basili, V. R. "Identifying Domain-Specific Defect Classes: Using Inspections and Change History," *International Symposium on Empirical Software Engineering, (ISESE)*, Rio de Janeiro, September, 2006.

3. Results

Early results needed to validate our process were to verify that students could indeed produce good HPC codes and that we could measure their increased performance. Table 2 is one set of data that shows that students achieved speedups of approximately three to seven on an 8-processor HPC machine. CxAy means class number x, assignment number y. This coding was used to preserve anonymity of the student population.

Data set	Programming Model	Speedup on 8 processors
Speedup measured relative to serial version:		
C1A1	MPI	mean 4.74, sd 1.97, n=2
C3A3	MPI	mean 2.8, sd 1.9, n=3
C3A3	OpenMP	mean 6.7, sd 9.1, n=2
Speedup measured relative to parallel version run on 1 processor:		
C0A1	MPI	mean 5.0, sd 2.1, n=13
C1A1	MPI	mean 4.8, sd 2.0, n=3
C3A3	MPI	mean 5.6, sd 2.5, n=5
C3A3	OpenMP	mean 5.7, sd 3.0, n=4

Table 2. Mean, standard deviation, and number of subjects for computing speedup on Game of Life program.

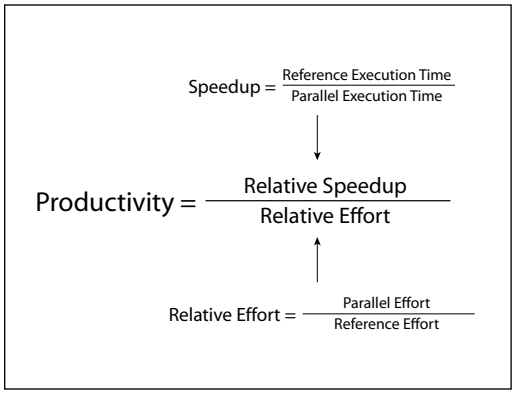

Figure 3. HPC productivity.

Additional classroom results include the following:

1. Measuring productivity in the HPC domain is part of understanding HPC workflows. However, what does productivity mean in this domain?[6] Figure 3 is one model that we can derive from the fact that the critical component of HPC programs is the speedup achieved by using a multiprocessor HPC machine over a single processor.[7] Productivity is defined as the relative speedup of a program using an HPC machine compared to a single processor divided by the relative effort to produce the HPC version of the program compared to a single processor version.

[6] *The International Journal of High Performance Computing Applications*, (18)4, Winter 2004.

[7] Zelkowitz, M., Basili, V., Asgari, S., Hochstein, L., Hollingsworth, J., Nakamura, T. "Measuring productivity on high performance computers," *IEEE Symp. on Software Metrics*, Como, Italy, (September 2005).

Program →	1	2*	3	4	5
Serial effort (hrs)	3	7	5	15	
Total effort (hrs)	16	29	10	34.5	22
Serial Exec (sec)	123.2	75.2	101.5	80.1	31.1
Parallel Exec (sec)	47.7	15.8	12.8	11.2	8.5
Speedup	1.58	4.76	5.87	6.71	8.90
Relative Effort	2.29	4.14	1.43	4.93	3.14
Productivity	0.69	1.15	4.11	1.36	2.83
*- Reference serial implementation					

Table 3. Productivity experiment: Game of Life.

Table 3 shows the results for one group of students programming the Game of Life (a simple nearest neighbor cellular automaton problem where the next generation of "life" depends upon surrounding cells in a grid and a popular first parallel program for HPC classes).[8] The data shows that our definition of productivity had a negative correlation compared to both total effort and HPC execution time, and a positive correlation compared to relative speedup. While the sample size is too small for a test of significance, the relationships all indicate that productivity does behave as we would want a productivity measure to behave for HPC programs, i.e., good productivity means lower total effort, lower HPC execution time and higher speedup.

[8] Gardner, M. "Mathematical games," *Scientific American*, October, 1970.

	Serial	MPI	OpenMP	Co-Array Fortran	StarP	XMT
Nearest-Neighbor Type Problems						
Game of Life	C3A3	C3A3 C0A1 C1A1	C3A3			
Grid of Resistors	C2A2	C2A2	C2A2		C2A2	
Sharks & Fishes		C6A2	C6A2	C6A2		
Laplace's Eq.		C2A3			C2A3	
SWIM			C0A2			
Broadcast Type Problems						
LU Decomposition			C4A1			
Parallel Mat-vec					C3A4	
Quantum Dynamics		C7A1				
Embarrassingly Parallel Type Problems						
Buffon-Laplace Needle		C2A1 C3A1	C2A1 C3A1		C2A1 C3A1	
Other						
Parallel Sorting		C3A2	C3A2		C3A2	
Array Compaction						C5A1
Randomized Selection						C5A2

Table 4. Some of the early classroom experiments on specific architectures.

Table 4 shows the distribution of programming assignments across different programming models for the first seven classes (using the same CxAy coding used in Table 2). Multiple instances of the same programming assignment lend the results to meta-analysis to be able to consider larger populations of students.[2]

[2] Hochstein, L., Basili, V., Zelkowitz, M., Hollingsworth, J., Carver, J. "Combining self-reported and automatic data to improve effort measurement," *Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2005)*, Lisbon, Portugal, September 2005, 356-365.

Dataset	Programming Model	Application	Lines of Code
C3A3	Serial	**Game of Life**	mean 175, sd 88, n=10
	MPI		mean 433, sd 486, n=13
	OpenMP		mean 292, sd 383, n=14
C2A2	Serial	**Resistors**	42 (given)
	MPI		mean 174, sd 75, n=9
	OpenMP		mean 49, sd 3.2, n=10

Table 5. MPI program size compared to OpenMP program size.

For example, we can use this data to partially answer an earlier stated hypothesis (**Hyp. 4:** *An MPI implementation will require more code than an OpenMP implementation*). Table 5 shows the relevant data giving credibility to this hypothesis (but this early data is not statistically significant yet).

2. An alternative parallel programming model is the PRAM model, which supports fine-grained parallelism and has a substantial history of algorithmic theory.[9] XMT-C is an extension of the C language that supports parallel directives to provide a PRAM-like

[9] Vishkin, U., Dascal, S., Berkovich, E., Nuzman, J. "Explicit Multi-Threading (XMT) Bridging Models for Instruction Parallelism," *10th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1998.

model to the programmer. A prototype compiler exists that generates code that runs on a simulator for an XMT architecture. We conducted a feasibility study in a class to compare the effort required to solve a particular problem. After comparing XMT-C development to MPI, on average, students required less effort to solve the problem using XMT-C compared to MPI. The reduction in mean effort was approximately 50%, which was statistically significant at the level of $p<.05$ using a t-test.[10]

[10] Hochstein, L., Basili, V. R. "An Empirical Study to Compare Two Parallel Programming Models," *18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '06)*, July 2006, Cambridge, MA.

3. While OpenMP generally required less effort to complete (Figure 4), the comparison of defects between MPI and OpenMP, however, did not yield statistically significant results, which contradicted a common belief that shared memory programs are harder to debug. However, our defect data collection was based upon programmer-supplied effort forms, which we know are not very accurate. This led to the defect analysis mentioned previously,[5] where we intend to do a more thorough analysis of defects made.

[5] Nakamura, T., Hochstein, L., Basili, V. R. "Identifying Domain-Specific Defect Classes: Using Inspections and Change History," *International Symposium on Empirical Software Engineering, (ISESE)*, Rio de Janeiro, September, 2006.

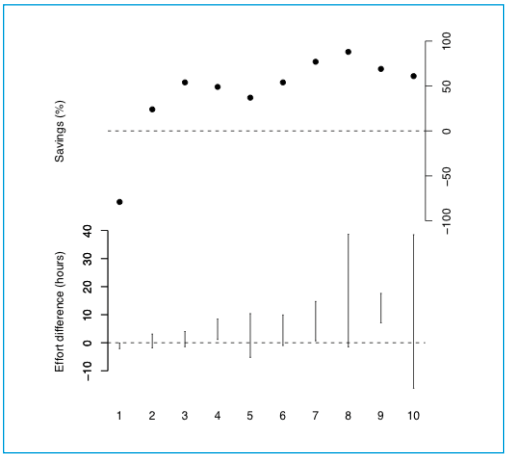

Figure 4. Time saved using OpenMP over MPI for 10 programs. (MPI used less time only in case 1 above).

4. We are collecting low-level behavioral data from developers in order to understand the "workflows" that exist during HPC software development. A useful representation of HPC workflow could both help characterize the bottlenecks that occur during development and support a comparative analysis of the impact of different tools and technologies upon workflow. One hypothesis we are studying is that the workflow can be divided into one of five states; serial coding, parallel coding, testing, debugging, and optimization.

In a pilot study at the University of Hawaii in Spring of 2006, students worked on the Gauss-Seidel iteration problem using C and PThreads in a development environment that included automated collection of editing, testing, and command line data using Hackystat. We were able to automatically infer the "serial coding" workflow state as the editing of a file not containing any parallel constructs (such as MPI, OpenMP, or PThread calls), and the "parallel coding" workflow state as the editing of a file containing these constructs. We were also able to automatically infer the "testing" state as the occurrence of unit test invocation using the CUTest tool. In our pilot study, we were not able to automatically infer the debugging or optimization workflow states, as students were not provided with tools to support either of these activities that we could instrument.

Our analysis of these results leads us to conclude that workflow inference may be possible in an HPC context. We hypothesize that it may actually be easier to infer these kinds of workflow states in a professional setting, since more sophisticated tool support is often available that can help support inferencing regarding the intent of a development activity. Our analyses also cause us to question whether the five states that we initially selected are appropriate for all HPC development contexts. It may be that there is no "one size fits all" set of workflow states, and that we will need to define a custom set of states for different HPC organizations in order to achieve our goals. Additional early classroom results are given in Hochstein, et al.[11]

[11] Hochstein, L., Carver, J., Shull, F., Asgari, S., Basili, V., Hollingsworth, J. K., Zelkowitz, M. "HPC Programmer Productivity: A Case Study of Novice HPC Programmers," *Supercomputing 2005*, Seattle, WA, November 2005.

4. Current Developments

As stated earlier, we have collected effort data from student developments and begun to collect data from professional HPC programmers in three ways; manually from the participants, automatically from timestamps at each system command, and automatically via the Hackystat tool, sampling the active task at regular intervals. All three methods provide different values for "effort," and we developed models to integrate and filter each method to provide an accurate picture of effort.

Our collection methods evolved one at a time. To simplify the process of students (and other HPC professionals) providing needed information, we developed an experiment management package (Experiment Manager) to more easily collect and analyze this data during the development process. It includes effort, defect and workflow data, as well as copies of every source program during development. Tracking effort and defects should provide a good data set for building models of productivity and reliability of HEC codes.

The Experiment Manager (pictured in Figure 5) has three components:

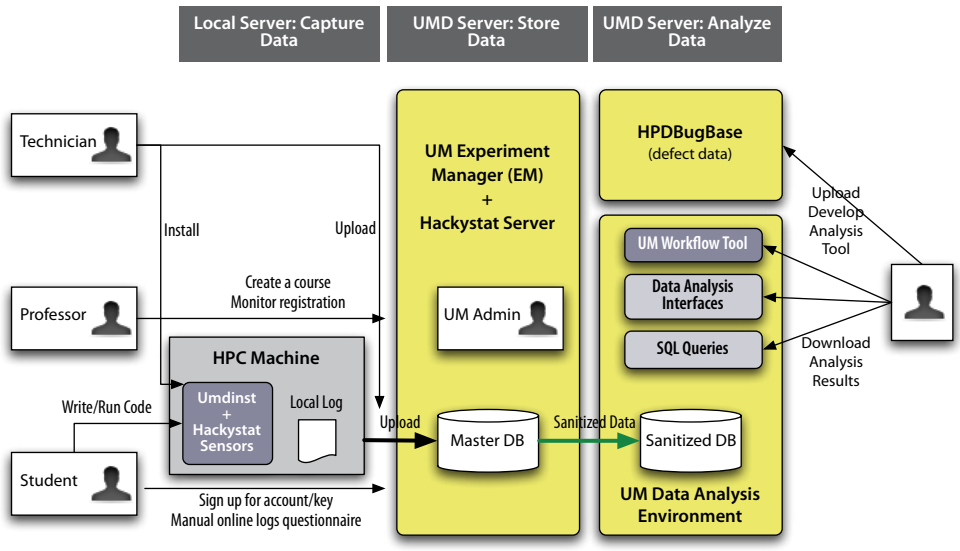

Figure 5. Experiment Manager Structure.

1. *UMD server*: This web server is the entry portal to the Experiment Manager for students, faculty and analysts and contains the repository of collected data.

2. *Local server*: A local server is established on the user machine (e.g., the one used by students at a university) that is used to capture experimental data before transmission to the University of Maryland.
3. *UMD analysis server*: A server stores sanitized data available to the HPCS community for access to our collected data. This server avoids many of the legal hurdles implicit with using human subject data (e.g., keeping student identities private).

For the near future, our efforts will focus on the following tasks:

- Evolve the interface to the Experiment Manager web-based tool to simplify use by the various stakeholders (i.e., roles).
- Continue to develop our tool base, such as the defect data base and workflow models.
- Build our analysis data base including details of the various hypotheses we have studied in the past.
- Evolve our experience bases to generate performance measures for each program submitted in order to have a consistent performance and speedup measure for use in our workflow and time to solution studies.

5. Conclusions

Over the past three years we have been developing a methodology for running HPC experiments in a classroom setting and obtaining results we believe are applicable to HPC programming in general. We are starting to look at larger developments and at large university and government HPC projects in order to increase the confidence on the early results we have obtained with students.

Our development of the Experiment Manager system allows us to more easily expand our capabilities in this area. This allows many others to run such experiments on their own in a way that allows for the appropriate controls of the experiment so that results across classes and organization at geographically diverse locations can be compared in order to get a thorough understanding of the HPC development model.

Acknowledgments
This research was supported in part by Department of Energy contract DE-FG02-04ER25633 and Air Force grant FA8750-05-1-0100 to the University of Maryland.

Observations about Software Development for High End Computing

Jeffrey C. Carver
Mississippi State University
Lorin M. Hochstein
University of Nebraska, Lincoln
Richard P. Kendall
Information Sciences Institute
University of Southern California
Taiga Nakamura
University of Maryland, College Park
Marvin V. Zelkowitz
University of Maryland, College Park
Fraunhofer Center for Experimental Software Engineering
Victor R. Basili
University of Maryland, College Park
Fraunhofer Center for Experimental Software Engineering
Douglass E. Post
DoD High Performance Computing Modernization Office

1. Introduction

Computational scientists and engineers face many challenges when writing codes for high-end computing (HEC) systems. With the movement towards peta-scale machines and the increasing focus on improving development productivity, writing a good code is even more challenging. The DARPA High Productivity Computing Systems (HPCS)[1] project is developing new machine architectures, programming languages, and software tools to improve the productivity of scientists and engineers.[2] Although the existence of these new technologies is important for improving productivity, they will not achieve their stated goals if individual scientists and engineers are not able to effectively use them to solve their problems. A necessary first step in determining the usefulness of new architectures, languages and tools is to gain a better understanding of what the scientists and engineers do, how they do it, and what problems they face in the current HEC development environment. Because the HEC community is very diverse,[3] it is necessary to sample different application domains to be able to draw any meaningful conclusions about the commonalties and trends in software development in this community.

This article discusses some of the similarities and differences found in 10 projects taken from different domains as understood by researchers in the HPCS project. We have undertaken a series of case studies to gain deeper insight into the nature of software development for scientific and engineering software. These studies have focused on two different types of projects, (characterized in Table 1):

- ASC-Alliance projects, DOE-sponsored computational science centers based at University of Illinois Urbana-Champaign, California Institute of Technology, University of Utah, Stanford University, and University of Chicago. In the rest of this paper, the codes are referred to as *ASC Codes*.
- Codes from DARPA HPCS mission partners (organizations that have a vested interest in the outcome of and often financial sponsorship of the project), some of which are classified and would therefore be inaccessible to other researchers. Due to the sensitive nature of many of these projects, they must remain anonymous in this paper. In the rest of this paper, these codes are referred to as *MP Codes*.

	ASC Codes	MP Codes
# of projects	5	5
Environment	Academia (ASC-Alliance projects)	Mission Partners (DoD, DOE, NASA)
Classified	No	Some
Code size	200-600 KLOC	80-760 KLOC
Type	Coupled multi-physics applications	Single physics to coupled multi-physics and engineering

Table 1. Types of projects examined.

[1] DARPA HPCS - http://www.highproductivity.org/

[2] Hochstein, L., Carver, J., Shull, F., Asgari, S., Basili, V., Hollingsworth, J.K., and Zelkowitz, M. "HPC Programmer Productivity: A Case Study of Novice HPC Programmers". In *Proceedings of SuperComputing*, 2005, p. 35.

[3] Post, D.E., Kendall, R.P., and Lucas, R.F., "The Opportunities, Challenges, and Risks of High-Performance Computing in Computational Science and Engineering", in *Advances in Computers*, p. 239-301.

While these case studies are still ongoing, the results to date allow for some cross-project analysis to provide a deeper understanding of the state of the practice.[4 5 6 7] In this paper, we discuss some observations gained from this analysis.

2. Goals & methodologies

While the ultimate goal for the case studies of the ASC codes and the MP codes is the same (to improve the productivity of computational scientists and engineers), we have used a different approach for each type of code. The object of study in the ASC codes has been the individual programmer (e.g., "Where does the computational scientist spend her time?"), while the object of study in the MP studies has been the project (e.g., "Which factors determine project success?"). Table 2 shows a comparison of the goals of the two types of case studies.

ASC Codes	MP Codes
• Characterize which scientific programming activities are time-consuming and problematic • Characterize the impact of technologies on developer effort. • Characterize the common problems encountered by programmers.	• Identify ways that successful projects manage risk • Identify project success factors • Identify productivity barriers that should be addressed by vendors • Develop a reference body of case studies

Table 2. Case study goals.

Table 3 provides an overview of the methodology used for each type of case study. In general, the approach for the MP codes was more comprehensive (longer questionnaire, on-site interviews, multiple subjects interviewed independently), and the approach for the ASC codes was more lightweight, which permitted quicker turnaround time when running the studies. Furthermore, each type of study collected different types of information. The focus in the ASC was lower level (i.e., more details about fewer things), while the focus in the MP codes was higher level (i.e., less details about more things).

	ASC Codes	MP Codes
Type	Ongoing	Retrospective
Interviewees	Technical leads	Projects leads, project staff
Overview	1. Pre-interview questionnaire 2. Telephone interview 3. Generate summary document 4. Send summary document for approval/comments 5. Generate synthesis report across all projects 6. Send synthesis report to all centers for approval/ comments	1. Identify project and sponsors 2. Negotiate case study participation 3. Pre-interview questionnaire 4. On-site interview 5. Initial list of findings 6. Follow-up 7. Write report
Focus	• Product: attributes, machine target, history • Project organization: structure, staff, configuration management • Development activities: adding new features, testing, tuning, debugging, porting, effort distribution, bottlenecks, achieving performance • Programming models and productivity: choice of model, adoption of language, productivity measures	• Goals, requirements, deliverables • Project characteristics, structure, organization and risks • Code Characteristics • Staffing • Workflow Management • V&V, Testing • Success Measures • Lessons Learned

Table 3. Methodology.

[4] Kendall, R.P., Carver, J., Mark, A., Post, D., Squires, S., and Shaffer, D. "Case Study of the Hawk Code Project," *Technical Report, LA-UR-05-9011*, Los Alamos National Laboratories, 2005.

[5] Kendall, R.P., Mark, A., Post, D., Squires, S., and Halverson, C. "Case Study of the Condor Code Project," *Technical Report, LA-UR-05-9291*, Los Alamos National Laboratories, 2005.

[6] Kendall, R.P., Post, D., Squires, S., and Carver, J. "Case Study of the Eagle Code Projec," *Technical Report, LA-UR-06-1092*, Los Alamos National Laboratories, 2006.

[7] Post, D., Kendall, R.P., and Whitney, E. "Case study of the Falcon Project". In *Proceedings of Second International Workshop on Software Engineering for High Performance Computing Systems Applications (Held at ICSE 2005)*, St. Louis, USA, 2005, p. 22-26.

3. Observations

Using the different methodologies and the different foci described in Section 2, we performed a cross-study analysis between the ASC codes and the MP codes to look for similar and different observations that can lead to deeper insight into the software development process. In this section, we present those observations along with the support (or lack of support) provided by each type of code (ASC and MP) during independent case studies. To facilitate the discussion of the observations, we have grouped them into high-level categories (each found in a subsection below) with a list of specific observations.

3.1 Goals and Drivers of Code Development

Code performance is not the driving force for developers or users; the science and portability are of primary concern: All of the projects studied were parallel programming projects, therefore code performance is clearly an important goal. But the primary interest of the users and developers is the science, not fast, scalable code (except where fast, scalable code is required to meet the scientific objective). As long as the overall project performance goals were met, the science and portability are of greater concern than the last 10-20% of speedup. For example, a member of one ASC Code team stated that metrics, like scalability, are recorded because the sponsor requests them, while "scientifically useful results per calendar time" would be a more appropriate productivity metric. Furthermore, the developers of the MP Codes indicated that because the codes are often used for decades, they focus more effort on portability than on speed and scalability for the current hardware platform. In fact, to the extent that an increase in performance can be achieved through new hardware, these developers believe that the portability of the software is far more important than the efficiency of the software.

Code success depends on customer satisfaction: For the MP Codes, success or failure depends on whether the code developers keep their customers (not always their sponsor) satisfied. Conversely, in the ASC Codes, the developers were their own customers, so they had no external customers to please.

3.2 Actions and Characteristics of Code Developers

Most developers are domain scientists or engineers, not computer scientists: The majority of the developers for the ASC Codes did not have any formal training in computer science or software engineering. On some (but not all) ASC Codes, the chief software architect had a background in computer science. The developers of the MP Codes found that it is easier to teach domain scientists and engineers how to write code than it is for computer scientists to comprehend the deep scientific or engineering phenomena being captured by the code, especially at the research level.

The distinction between developer and user is blurry: In the ASC Codes, there are no "external" customers whose needs must be met. The primary users of the code are the developers themselves, who were adding functionality to advance their own research. In some cases, there are external users of the code. But, because these codes may require additions or modifications to be useful, an external "user" may still need to do a certain degree of programming. The MP Codes have more external users than the ASC Codes, but the developers still constitute an important portion of the user-base.

There is high turnover in the development team: Because the ASC Codes are developed in academic environments, many project members (postdocs and grad students) are involved for only a few years (e.g., in one project, the two technical leads had been involved for less than four years). Most of the ASC Codes evolved from earlier codes that were written by scientists who had long since left the organization. Conversely, most of the MP Codes had a core set of developers that remained with the project for the duration, often for a decade or more.

3.3 Software Engineering Process and Development Workflow

There is minimal but consistent use of software engineering practices: All ASC Codes exhibited the use of a subset of standard software engineering practices (i.e., the use of version control systems, regression tests, and software architecture). However, the developers of these codes did little defect tracking or documentation beyond user guides. Conversely, the MP Codes did not show this type of consistency in the use of software engineering practices across teams. Each team made use of a few recognized software engineering practices, but there was no uniformity across projects.

Development is evolutionary at multiple levels: In both the ASC and MP Codes, the developers are working on "new science." As a result, they do not always know the correct output in advance, providing numerous challenges for verifying the code relative to the phenomenon being simulated. The teams all use an iterative, agile approach, where new algorithms are implemented and then evaluated to determine whether they are of sufficient quality for current simulations. Even though the more rigid CMM approach may not be appropriate for many of the MP Codes, many projects claim to follow this approach, possibly as a result of pressure from sponsors.

Tuning for a specific system architecture is rarely done, if ever: None of the ASC Codes optimize/tune their code for particular platforms. For example, they assume that they are developing for a "flat MPI" system, and so do not optimize for machines with SMP nodes. One project member made a comment that was representative of all projects:

> The amount of time it takes to tune to a particular architecture to get the last bit of juice, is considerably higher than the time it takes to develop a new algorithm that improves performance across all platforms. Our goal is to develop algorithms that will last across many lifetimes. We are not really interested in getting that last little bit of performance.

The MP Codes did perform some tuning, but it was in conflict with their larger goal of portability (see Observation 1 in Section 3.1).

There is little reuse of MPI frameworks: Several of the ASC and MP Codes built their own frameworks to abstract away the low-level details of the MPI parallelism. However, these frameworks are built from scratch each time, rather than being reused from another system. One developer from an ASC Code explained this lack of reuse:

> We have encountered other projects where people have said, 'We'll use class library X that will hide array operations and other things', but all sorts of issues arose. These frameworks make assumptions about how the work will be done, and to go against the assumptions of the framework requires getting into really deep details, which defeats the purpose of using such a framework.

Most development effort is focused on implementation rather than maintenance: In both the ASC and the MP Codes, most of the effort was expended during implementation, rather than maintenance. This distribution indicates that rather than being released and maintained like traditional IT projects, these projects are under constant development.

3.4 Programming Languages

Once selected, the primary language does not change: Languages adopted by the MP Codes do not change once they have been selected. The majority of the older codes were written in FORTRAN77. Some newer codes and many of the ASC Codes are in FORTRAN90, C and C++. C seems popular for handling I/O issues. Some of the ASC and MP Codes have also adopted Python to drive the application. However, on one project the Python-based driver was later abandoned because it increased debugging complexity and reduced the portability of the code.

Higher level languages (e.g., Matlab) are not widely adopted for the core of applications: The MP teams have not adopted these higher level languages for use in their codes. The ASC teams have restricted the use of these high level languages to prototyping numerical algorithms.

3.5 Verification and Validation

Verification and Validation are very difficult in this domain: While V&V is difficult in all software domains, it was seen as especially difficult for both the MP and ASC Codes due to some unique characteristics. It is difficult for developers to judge the correctness of code because they often do not know the "right" answer. In addition, there are at least three places where defects can enter the code, making it difficult to ultimately identify the source of the problem: 1) the underlying science could be incorrect; 2) the translation of the domain model to an algorithm could be incorrect; and 3) the translation of that algorithm into code could be incorrect.

Visualization is the most common tool for validation: All of the ASC Codes provide visualization to allow the users to view the large amounts of outputs produced. For the experienced user, this visualization provides a sanity check that the code is behaving reasonably.

3.6 Use of Support Tools during Code Development

Overall, tool use is lower than in other software development domains: Both the ASC and the MP Codes tended to view integrated development environments (IDEs) as being too restrictive. Instead, they liked the flexibility of UNIX-style command-line tools, such as make and shell scripts, to use for building applications. A complicating factor for wide tool use is that the target machines do not support the development tools that are available to programmers in other domains. Conversely, developers from both ASC and MP Codes did make some use of HPC-specific tools such as performance monitors and parallel debuggers.

Third party (externally developed) software and tools are viewed as a major risk factor: Because the code developed by the MP Codes tend to take years to develop, they are not willing to put their development at risk by relying on a software package or tool that may not be supported in the future. Furthermore, the ASC teams were concerned with the portability of libraries, which decreased their likelihood of use.

4. Conclusions

In this paper, we have summarized our findings from a series of case studies conducted with ten ASC and MP Codes as a series of observations. Due to different environments in which each code is developed, some of the observations are consistent across code teams while others vary across code teams. Overall, we found high consistency among the ASC Codes and the MP Codes. Due to the different environments and foci of these projects, this result is both surprising and positive. In addition, despite the fact that a large majority of the developers on these teams have little or no formal training in software engineering, they have been able to make use of some basic software engineering principles. Further education and motivation could increase the use of these principles and further increase the quality of scientific and engineering software that has already demonstrated its value.

Based on the positive results thus far, we have plans to conduct additional case studies to gather more data in support of or in contradiction to the observations presented in this paper. In future case studies, we will strive to investigate codes from additional domains, thereby allowing broader, more inclusive conclusions to be drawn.

Acknowledgments
This research was supported in part by Department of Energy contract DE-FG02-04ER25633 and Air Force grant FA8750-05-1-0100 to the University of Maryland.

Application Software for High Performance Computers: A Soft Spot for U.S. Business Competitiveness

Suzy Tichenor
Council on Competitiveness

Recent Council on Competitiveness[1] studies have found that nearly all businesses that exploit high performance computing (HPC) consider it indispensable for rapid innovation, competitive advantage and corporate survival. But Council on Competitiveness research has also identified barriers to the broader use of HPC in the business sector. The companion article on high performance computing (HPC) in this issue of *CTWatch*[2] examined one of these barriers—the difficulty of determining the return-on-investment for HPC hardware systems (also known as supercomputers)—and proposed a methodology for addressing this issue. Another important barrier preventing greater HPC use is the scarcity of application software capable of fully exploiting current and planned HPC hardware systems.

U.S. businesses rely on a diverse range of commercially available software from independent software vendors (ISVs). At the same time, experienced HPC business users want to exploit the problem-solving power of contemporary HPC hardware systems with hundreds, thousands or (soon) tens of thousands of processors to boost innovation and competitive advantage. Yet few ISV applications today can exploit ("scale to") even 100 processors, and many of the most-popular applications scale to only a few processors in practice.

Market forces and technical challenges in recent years have caused the ISVs to pull away from creating new and innovative HPC applications, and no other source has arisen to satisfy this market need. For business reasons, ISVs focus primarily on the desktop computing markets, which are much larger and therefore promise a better return on R&D investments. ISVs can sometimes afford to make modest enhancements to their application software so that it can run faster on HPC systems, but substantially revising existing applications or creating new ones does not pay off. As a result, the software that is available for HPC systems is often outdated and incapable of scaling to the level needed to meet industry's needs for boosting problem-solving performance. In some cases, the applications that companies want simply do not exist.

This need for production-quality application software and middleware has become a soft spot in the U.S. competitiveness armor; a pacing item in the private sector's ability to harness the full potential of HPC. Without the necessary application software, American companies are losing their ability to aggressively use HPC to solve their most challenging problems and risk ceding leadership in the global marketplace.

To better understand why U.S businesses are having difficulty obtaining the high-performance, production-quality application software they need, the Council on Competitiveness convened an HPC Users Conference, followed by a day-long software workshop held in collaboration with the Ohio Supercomputer Center (OSC). Participants[3] discussed the effectiveness of current business models for HPC application software, as well as technical and resource barriers preventing the development of more effective software. They produced a roadmap of actions ISVs, universities and government research establishments could follow, individually and collectively, to address these barriers and ensure that our country has the application software needed to solve our most important competitive problems. The major discussion themes and recommendations are highlighted in the following sections.

[1] The Council on Competitiveness is an organization of the top business, university and labor leaders in the United States responsible for influencing the course of American competitiveness on regional, national and global scales. For additional information about its High Performance Computing project and copies of reports, surveys and case studies, see http://www.compete.org/hpc .

[2] Reuther, A., Tichenor, S. "Making the Business Case for High Performance Computing: A Benefit-Cost Analysis Methodology," *CTWatch Quarterly*, Volume 2, Number 4A, November 2006.

[3] The conference participants included executives representing independent software vendors, public and private sector HPC users, HPC hardware vendors, and public sector funders of HPC R&D.

Market and Resource Barriers

A variety of market and resource barriers are retarding development of more scalable, production-quality application software.

The Niche HPC Market Discourages Development of Third Party HPC Application Software

Despite the rise in global competition and HPC's proven ability to accelerate innovation and increase productivity, HPC remains a niche market with much smaller revenue opportunities than the larger commercial computing market. This lowers the research and development capital available to develop next-generation HPC application software and lessens the opportunity for ISVs to obtain investor funding for HPC R&D. It also heightens the risk that ISVs could be "bought out" by companies, which might restrict or eliminate access to their software by U.S. commercial and national security organizations that depend on it.

As for-profit companies, many of which are public, the ISVs often must pursue the broader commercial computing market with its larger revenue opportunity. In addition, according to the Council's 2005 Study of the ISVs *Serving the High Performance Computing Market, Part A: Current Market Dynamics*,[4] more than a third of the ISVs qualify as small businesses, earning less than $5 million annually. Even if these companies were to invest 10 percent of their revenue in research and development, this would be sufficient to add only a few new features to existing HPC applications. Fundamental rewrites of application software or the creation of new software for the HPC niche market require far more investment than this.

[4] http://www.compete.org/. See also Part B: End User Perspectives

The niche status of the HPC market also reduces the potential for outside investor funding for the ISVs that serve the HPC community, eliminating another possible source of research and development resources. Additionally, unlike software developers who serve the broad commercial computing market and can keep their prices low by amortizing their research and development costs across a large user base, the ISVs that serve the HPC market do not have a correspondingly large customer set against which they can amortize their expenses. As a result, the ISVs must shoulder the cost of a major software redesign or spread it across a small user community, significantly increasing the price of the HPC application software.

Traditional ISV Business Models are Perpetuating the Niche Market

In today's market, ISVs typically sell unlimited annual-use licenses, often on a per-user or per-processor basis. Users often find that they cannot afford the additional software licensing fees that apply when they want to run their problems over more processors. This discourages them from running larger, more complex problems on their current systems or adding additional processors. As a result, users don't exploit the applications to the desired and needed degree.

In addition, annual software license fees can be prohibitively expensive for the small and mid-sized companies that dominate the U.S business landscape – the same companies that represent the potential growth market for HPC. Most cannot afford HPC systems and software, nor do they want to deal with the complexity of this technology. Many only want access to HPC systems, software and expertise on a periodic basis. "The analogy," explained Paul Bemis, vice president for marketing at Fluent Inc., "is you're going to rent a car for the weekend. You go to Hertz and they say, sorry, all we offer are annual leases. But you only need the car for the

weekend. Sorry, they say again, all we offer are annual leases. That is the ISV community today. It needs to change. We need to be asking, 'How much do you need and how long do you need it?'"

An interesting service model has evolved in the oil and gas industry that may provide an analog. This industry is dominated by small, independent businesses that are the primary producers of oil and gas in this country. These small companies rely on service firms to perform their complex seismic computations and reservoir simulations. The service firms, populated with experienced engineers and scientists, have bundled their knowledge of the industry, their ability to create and use HPC software, and access to HPC systems. With these, they have created very profitable consulting businesses. This business model produces much higher margins than software development alone, because it delivers what the small oil and gas companies value most: access to industry expertise and advanced HPC tools in order to make decisions, without having to invest in the underlying technology. This permits even the smallest oil and gas companies to match the technical capabilities of larger competitors.

Many ISVs Cannot Afford to Acquire Needed Parallel Systems or Support Multiple HPC Platforms

The small size of many ISVs severely limits their ability to acquire even one large parallel system, much less the full variety of HPC systems from different vendors. Yet access to these systems is essential for developing ISV software applications that will run compatibly and fast on the range of hardware products available in the HPC market. Hardware vendors that once donated machines to ISVs for software development, testing and support can no longer afford to do so because of sharply reduced profit margins. Instead, ISVs must either set aside significant funds to acquire systems or, more typically, find a way to access HPC systems without acquiring them.

Limited funds also make it difficult for many ISVs to support multiple types of HPC computer platforms. They therefore tend to focus on one type of HPC system: the so-called "clusters" built from commodity components, because these lower-priced systems are the most pervasive computer products in the HPC market. This has alarmed business users whose problems are better suited to other types of HPC systems. These users believe their crucial, competitive problems are being ignored.

A Shortage of Skilled Talent Is Holding Back HPC Software Development

A close examination of the current education pipeline presents a troubling picture regarding the availability of next-generation scientists and engineers. American-born students are choosing to pursue computational science and engineering less frequently than in the past. Research institutions face an added challenge; the development schedule for HPC application software may exceed the time that graduate students spend at their facilities. "The people we need to architect these codes are unique individuals," explained John Morrison, division leader for computing, communications & networking, Los Alamos National Laboratory. "They have to understand the science domain, computational science, computer science, and then figure out how to make these big applications run on a parallel computer and to scale."

On a positive note, trends in desktop computing may bring new talent to bear on this problem. Multi-core processors – technology that allows multiple processors to work together on a single chip—already are finding their way to the desktop computers of many workers and

could stimulate development of more parallel software. As this technology proliferates in the future, every desktop computer is likely to become a parallel computer, and the broader commercial software development community will be forced to address the same challenge that the HPC community has struggled with for years: creating applications that work in a parallel environment. This should bring extraordinary energy, money and *talent* to the software development process, resulting in new tools and new solutions for HPC users, too.

Technical Barriers

In addition to market and resource barriers, HPC users and software developers also face technical barriers that are impeding the development of more advanced HPC application software and underlying middleware.

Current HPC Software Needs Better Interfaces So That It Can Be Integrated Into The Business Workflow

Businesses can increase dramatically the value of their HPC systems when usage is expanded beyond the Ph.D. researchers and integrated into the workflow of the organization. This means that HPC applications and tools must be distributed at the desktop to "non-experts," i.e., designers, engineers, analysts and others who are more directly involved in the production of an organization's product or service. The key to putting HPC applications and tools into the hands of the non-Ph.D.s at the desktop is better user interfaces that are easy to operate.

Integration of HPC into workflow practices becomes even more important as competitive pressures drive companies to shorten product cycles. As little as a six-month delay in product introduction can allow competitors to take market share and substantially cut into profit margins. Yet, because of the primitive user interfaces in use today, it can take longer to set up a problem than to run it, particularly as the number of processors increases in HPC systems. In this environment, even one day's delay in setting up and running an HPC problem can significantly impact the workflow. When users are under time-to-market pressures, they tend to run smaller problems whose solutions may not provide the needed competitive boost.

User Dependence On Legacy Systems And Their Integration Into Key Business Processes Can Slow The Development Of New Application Software

For many ISVs, the requirement to maintain and add features to "legacy codes" is tying up limited development dollars that could be spent creating new software with greater parallelism, or at a minimum substantially revising the legacy software. Given the limited R&D resources of most ISVs and the requirement that publicly traded companies must report quarterly earnings, ISVs invest first in projects customers will pay for today, such as modest revisions to existing legacy software. Only then can they consider higher risk, more costly efforts such as developing new solutions from scratch or undertaking the "hundred man-year investment" that is needed to correctly re-configure a complex software application like MSC.Software's NASTRAN to run on hundreds or thousands of processors.

Integrating HPC applications into the business workflow can increase productivity, but it can also slow development of new software. Industrial HPC users often are reluctant to make changes to hardware or critical software after it is integrated into key business processes. Re-coding legacy calculations to achieve greater parallelism can change the answers by several

percent. For many long-time HPC users, consistency in results can be even more important than improved performance.

Further, the business process reengineering required to make the switch to newer, more efficient applications frequently is too time consuming and risky when companies are under market pressures to deliver products to customers on time and within specified design and quality parameters. Dr. Donald Paul, vice president and chief technology officer for Chevron, explained that competitive advantage often comes from "how you integrate all of the applications that you use into major workflows, and how well those are integrated into and drive the way the business works. The business process structures that can be built around the technology are as important as the technology, and are in fact dependent on the technology." While rapidly changing software or systems might be acceptable or even desirable in a research environment, Dr. Paul explained that it is often intolerable in a business production environment, where a single application must be integrated into a complex stream of business processes.

Software from the Research Community Is Not Adequate for Industry's Needs

While national laboratories and universities spend hundreds of millions of dollars every year on fundamental HPC software research, this software tends to be difficult to use in a corporate production environment. Research software typically is developed by Ph.D.s for use by highly experienced scientists who are willing to put in extraordinary time and effort to make it work. But industry wants fast desktop tools that do not require a Ph.D. to use. The world of technical computing has evolved from one that was managed by a few highly skilled and educated engineers to one in which computing resources are spread throughout an enterprise to a wide variety of people. Research organizations do not develop their applications with these needs in mind.

Further compounding the problem is a shrinking federal budget that has reduced the amount of government funding available to support joint programs between national labs and industry. In this environment, it is unlikely that the laboratories will reallocate significant portions of their budgets to focus on joint endeavors with ISVs. The labs could be criticized for misspending funds, since there is an assumption that they have been given "just enough" to fulfill their missions.

Additionally, research software developed within the universities is often an abstraction of a real problem, and may be licensed as open source software. Therefore, it has not been subjected to the rigorous testing demanded by industry, and may not allow for proprietary extensions to the code that companies require. Consequently, the software developed by these research organizations, while providing an important foundation for commercial software, is often inadequate for immediate use by industry.

Some Open Source License Models Inhibit Commercialization of HPC Research Software

Most HPC research codes are produced by the national laboratories or universities and licensed as open source, i.e., the source code is freely available and may be copied or altered without permission or payment. However, the terms of some open source license models discourage software developers from improving and commercializing these codes, thereby hindering software innovation.

For example, the General Public License (GPL) requires that derivative works also bear the GPL designation, so they automatically become open source as well. This makes ISVs and other software developers reluctant to invest in development, because they cannot keep their improvements proprietary or derive competitive advantage from them. David Turek, vice president of Deep Computing at IBM, shared the frustration of many software developers: "It's very difficult when we make major investments to try to produce differentiated technology, and somebody says to me 'Really nice. Now can you put it on everybody else's platform in the world and make it cheap or free?' What we have to do, given this circumstance, is to work with other platform vendors on new business models that satisfy the needs of the end users while also making our own technology investments economically viable."

On the other hand, the Berkeley Software Distribution (BSD) license allows developers to incorporate the open source code into derivative works that are not open source and can become proprietary. Much successful HPC software that has transitioned from government and universities to industry bear BSD-type open source licenses.

A vibrant, growing HPC market is the best long-term guarantee that production quality HPC application software will be available. Participants in the Council-OSC software workshop also agreed that wider usage of HPC is crucial for future U.S. economic growth and the nation's ability to maintain a preeminent position in the global marketplace. Participants made recommendations for stimulating long-term HPC market growth through near-term accomplishments.

Recommendations for ISVs:

1. Consider pricing changes or moving to a service- based model in order to provide users with software, expertise and access to appropriate HPC hardware.
2. Partner with companies that use HPC to run demonstration projects that establish the business value of using HPC.
3. Develop easy-to-use interfaces that encourage HPC applications to be integrated into organizational workflows.

Recommendations for Universities:

1. Enhance their educational programs in computational science at the undergraduate and graduate levels to meet the needs for skilled technical workers.
2. Enhance their understanding of the ISV community's requirements so that they can better leverage their own software research and education agendas to help meet the ISVs' needs where appropriate.

Recommendations for Government:

1. Modify research support practices to provide sustained (multi-year) funding for research teams to develop mature research software and algorithms. Related to this, encourage commercialization of suitable software.
2. Where open source HPC research codes are being developed, terms of government grants and contracts should more seriously consider BSD model licenses, to enable ISVs to build

commercial products based on the codes without jeopardizing the ISVs' privately created intellectual property.

Conclusions

Implementing these recommendations entails considerable risk to the ISVs, the users, the research community, the government, and investors, but failure to take action may inhibit competitive advances by U.S. companies and place them in jeopardy should other countries or companies capitalize first on the potential for expanded use of HPC.

By leveraging HPC more fully to solve complex, crucial problems, America can unleash a new era of innovation-driven growth, creating new industries and markets, fueling wealth creation and profits, and generating high-value, higher-paying jobs that will raise the standard of living for all citizens.

Acknowledgments

The Council on Competitiveness thanks the Department of Energy's Office of Science and the Ohio Supercomputer Center for co-sponsoring the High Performance Computing Application Software Workshop that lead to this article. This work was supported by the U.S. Department of Energy, under grant DE-FG26-04NT42101.

Analysis of Parallel Software Development using the Relative Development Time Productivity Metric

Andrew Funk
MIT Lincoln Laboratory
Victor Basili
University of Maryland, College Park
Lorin Hochstein
University of Nebraska, Lincoln
Jeremy Kepner
MIT Lincoln Laboratory

Introduction

As the need for ever greater computing power begins to overtake the processor performance increases predicted by Moore's Law, massively parallel architectures are becoming the new standard for high-end computing. As a result, programmer effort is increasingly the dominant cost driver when developing high performance computing (HPC) systems. For the end user, whereas execution time was once the primary concern, development time is now a significant factor in overall time to solution.

In the DARPA High Productivity Computer Systems (HPCS) program[1] we have defined the overall system productivity, ψ, as utility over cost:

$$\psi = \frac{U(T)}{C_S + C_O + C_M}$$ [2],

where utility, $U(T)$, is a function of time. Generally speaking, the longer the time to solution, the lower the utility of that solution will be. The denominator of the formula is a sum of costs: software (C_S), operator (C_O), and machine (C_M). A higher utility and lower overall cost lead to a greater productivity for a given system.

Previously we defined a special case that we call relative development time productivity.[3] [4] For the programmer, system utility is measured in terms of application performance. From the end-user's perspective, the only relevant cost is that of software development. Thus, for the end-user developing small codes, we define productivity as:

$$\psi_{\text{small codes}} = \frac{\text{application performance}}{\text{programmer effort}}$$

For the case of developing parallel codes, we compare the parallel application performance to that of a baseline serial application (i.e., parallel speedup) and compare the effort required to develop the parallel application to the effort required for the serial version. Thus we define relative development time productivity (RDTP) as:

$$\psi_{\text{relative}} = \frac{\text{speedup}}{\text{relative effort}}$$

This metric has been used to analyze several parallel computing benchmark codes and graduate student programming assignments. For purposes of analysis in this paper, relative effort is defined as the parallel code size divided by the serial code size, where code size is measured in source lines of code (SLOC).[5]

[1] High Productivity Computer Systems - http://www.HighProductivity.org/

[2] Kepner, J. "HPC Productivity Model Synthesis." *IJHPCA Special Issue on HPC Productivity*, Vol. 18, No. 4, SAGE 2004.

[3] Funk, A., Basili, V., Hochstein, L., Kepner, J. "Application of a Development Time Productivity Metric to Parallel Software Development." *Second International Workshop on Software Engineering for High Performance Computing Systems Applications*. St. Louis, Missouri. May 15, 2005.

[4] Funk, A., Kepner, J., Basili, V., Hochstein, L. "A Relative Development Time Productivity Metric for HPC Systems." *Ninth Annual Workshop on High Performance Embedded Computing*, 20 – 22 September 2005, Lexington, MA.

[5] Wheeler, D. SLOCcount. http://www.dwheeler.com/sloccount/

Analysis

Classroom Experiments

In the HPCS program a series of classroom experiments was conducted in which students were asked to create serial and parallel programming solutions to classical problems (see "Experiments to Understand HPC Time to Development" in this issue). The students used C, Fortran, and Matlab for their serial codes, and created parallel versions using MPI, OpenMP, and Matlab*P ("StarP").[6] Using a combination of student effort logs and automated tools, a variety of information was collected, including development time, code size, and runtime performance.

[6] Choy, R., Edelman, A. *MATLAB*P 2.0: A unified parallel MATLAB*. MIT DSpace, Computer Science collection, Jan. 2003. http://hdl.handle.net/1721.1/3687

In Figure 1, the graph on the left depicts speedup vs. relative effort for seven classroom assignments. Each data point is labeled according to professor and assignment number, e.g., "P0A1." The graph on the right plots the median RDTP values for these same classroom assignments (P2A1 and P3A3 each involved two sub-assignments using different programming models). The speedup and relative effort were computed by comparing each student's parallel and serial implementations, and the median values for each assignment are plotted on the graph on the left. Error bars indicate one standard deviation from the median value. For purposes of comparison, all performance data was taken from runs using eight processors.

The MPI data points for the most part fall in the upper-right quadrant of the speedup vs. relative effort graph, indicating that students were generally able to achieve speedup using MPI, but at the cost of increased effort. The OpenMP data points indicate a higher achieved speedup compared to MPI, while at the same time requiring less effort. A plot of the RDTP metric (Figure 1, right) makes this relationship clear. Based on the results from these experiments, students were able to be the most productive when using OpenMP. While the StarP implementations required the least relative effort, most students were not able to achieve speedup. This is reflected in the lower RDTP value for StarP.

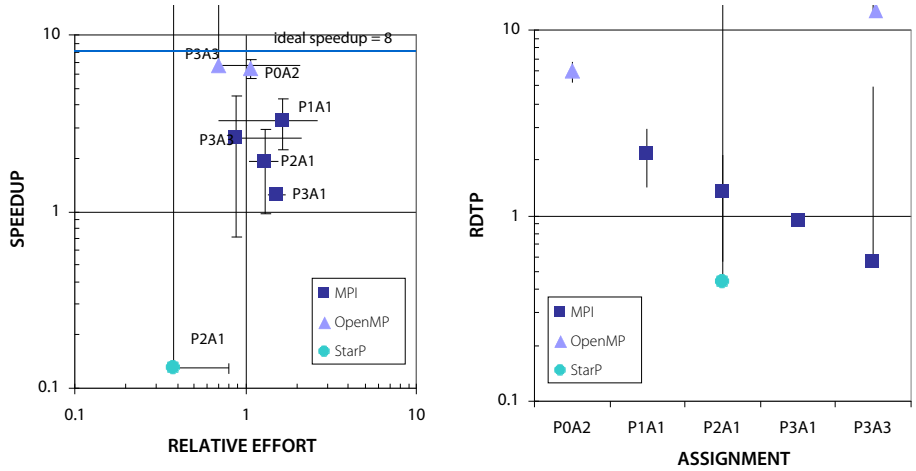

Figure 1. Speedup vs. Relative Effort and RDTP for the classroom experiments.

NAS Parallel Benchmarks

The NAS Parallel Benchmark (NPB)[7] suite consists of five kernel benchmarks and three pseudo-applications from the field of computational fluid dynamics. The NPB presents an excellent resource for this study, in that it provides multiple language implementations of each benchmark. The exact codes used were the C/Fortran (serial, OpenMP) and Java implementations from NPB-3.0, and the C/Fortran (MPI) implementations from NPB-2.4. In addition, a parallel ZPL[8] implementation and two serial Matlab implementations were also included in the study.

[7] NAS Parallel Benchmarks - http://www.nas.nasa.gov/Software/NPB/

[8] ZPL - http://www.cs.washington.edu/research/zpl/home/

In Figure 2, the chart on the left displays speedup vs. relative effort for the NPB. Each data point corresponds to one of the eight benchmarks included in the NPB suite, and the results are grouped by implementation. The chart on the right plots the RDTP values for the various implementations of each of the eight NPB benchmark codes. For purposes of comparison, each parallel code was run using four processors. The speedup and relative effort for each benchmark implementation are calculated with respect to a reference serial code implemented in Fortran or C.

As shown, the OpenMP implementations tend to yield parallel speedup comparable to MPI, while requiring less relative effort. This is reflected in the higher RDTP values for OpenMP. The lone ZPL implementation falls in the upper-left quadrant of the graph, delivering relatively high speedup while requiring less effort, as compared to the serial Fortran implementation. Accordingly, this ZPL implementation has a high RDTP value. Although a single data point does not constitute a trend, this result was included as an example of an implementation that falls in this region of the graph. The Matlab results provide an example of an implementation that falls in the lower-left quadrant of the graph, meaning that its runtime is slower than serial Fortran, but it requires less relative effort. By virtue of its extremely low relative effort, the serial Matlab manages to have a RDTP value comparable to parallel Java.

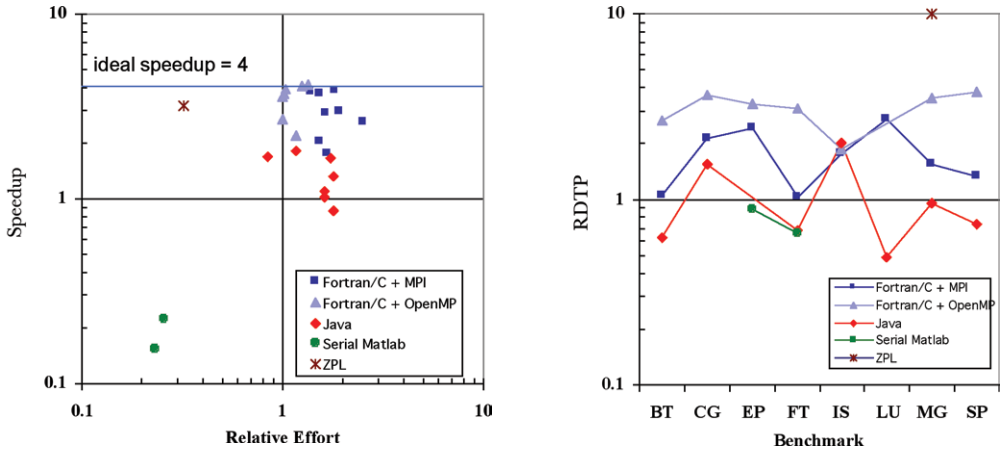

Figure 2: Speedup vs. Relative Effort and RDTP for the NPB.

HPC Challenge

The HPC Challenge suite consists of several activity-based benchmarks designed to test various aspects of a computing platform (see "Design and Implementation of the HPC Challenge Benchmark Suite" in this issue). The four benchmarks used in this study were FFT (v0.6a), High Performance Linpack (HPL, v0.6a), RandomAccess (v0.5b), and Stream (v0.6a). These codes were run on the Lincoln Laboratory Grid (LLGrid), a cluster of dual-processor nodes connected by Gigabit Ethernet.[9] The parallel codes were each run using 64 of these dual-processor nodes, for a total of 128 CPUs. The speedup for each parallel code was determined by dividing the runtime for a baseline serial C/Fortran code by the runtime for the parallel code (the serial Matlab code was treated the same as the parallel codes for purposes of comparison).

[9] Haney, R. et. al. "pMatlab Takes the HPC Challenge." Poster presented at High Performance Embedded Computing (HPEC) workshop, Lexington, MA. 28-30 Sept. 2004.

Figure 3 presents the results of RDTP analysis for the HPC Challenge benchmarks. With the exception of Random Access (the implementation of which does not scale well on distributed-memory computing clusters), the MPI implementations all fall into the upper-right quadrant of the graph, indicating that they deliver some level of parallel speedup, while requiring greater effort than the serial code. As expected, the serial Matlab implementations do not deliver any speedup, but all require less effort than the serial code. The pMatlab implementations (except Random Access) fall into the upper-left quadrant of the graph, delivering parallel speedup while at the same time requiring less effort.

The combination of parallel speedup and reduced effort means that the pMatlab implementations generally have higher RDTP values. On average the serial Matlab implementations come in second, due to their low relative effort. The MPI implementations, while delivering better speedup, also require more relative effort, leading to lower RDTP values.

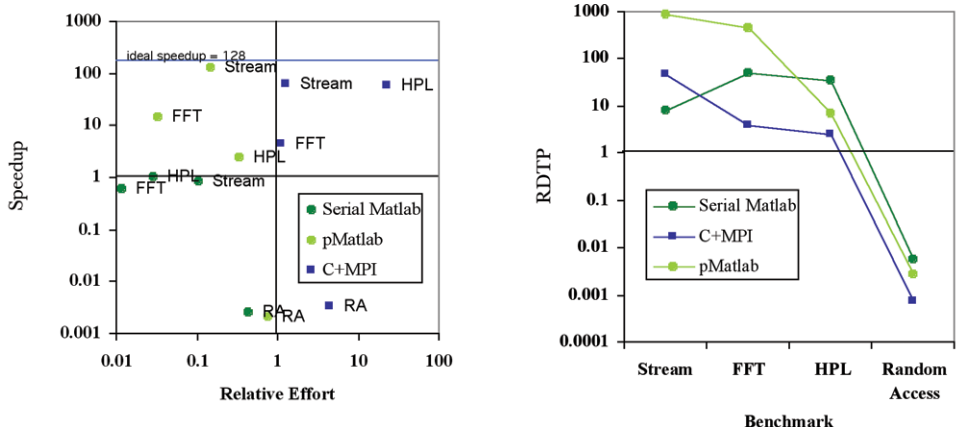

Figure 3. Speedup vs. Relative Effort and RDTP for HPC Challenge.

HPC Challenge Award Competition

In the HPC Challenge Award Competition, teams were invited to submit their own implementations of selected benchmarks from the HPC Challenge benchmark suite.[10] Evaluation of entries was weighted 50% for performance and 50% for code elegance, clarity, and size. The results of this competition are shown in Figure 4.

[10] HPC Challenge - http://icl.cs.utk.edu/hpcc/, http://www.hpcchallenge.org/

These graphs indicate the speedup and relative effort for each submitted implementation of the FFT, RandomAccess, HPL/Top500, and Stream benchmarks. Teams reported the performance of their implementation running on their own parallel computing platform, with no restriction on number of processors. This performance was compared to that of a baseline serial implementation to compute speedup. Relative effort is taken as the size of the parallel code relative to the baseline serial implementation. For each benchmark, the speedup and relative effort for the reference C/MPI implementation is indicated for comparison. The reference C/MPI implementation generally falls in the upper right quadrant of the graph, indicating that it achieved speedup at the cost of additional effort with respect to the serial version.

As shown in the graphs, 24 of the 30 submissions achieved speedup relative to the serial version, and 18 entries required less effort (i.e., smaller code size) relative to the serial version. All but two of the entries required less effort than the C/MPI reference implementation. It is also worth noting that half of the submissions, including all of the winning entries, fall in the upper left quadrant of the graph, indicating that they achieved speedup while requiring less development effort relative to the serial implementation. This result bolsters the argument for using a metric such as RDTP which takes into account both performance and developer effort.

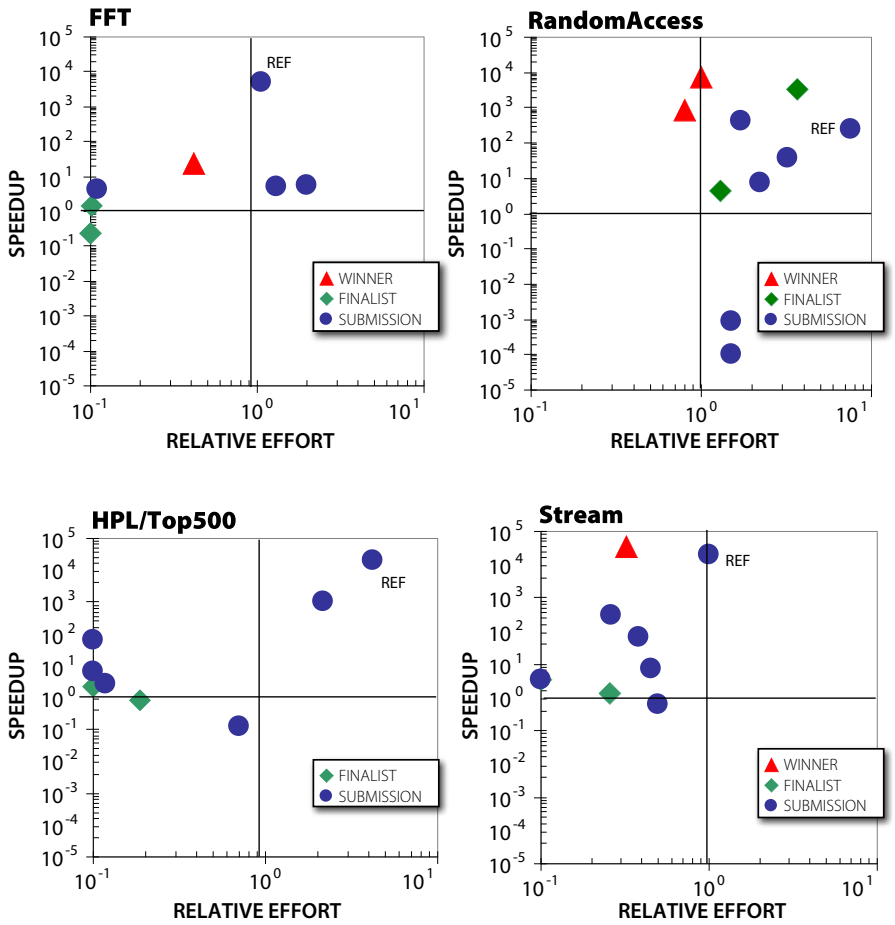

Figure 4. HPC Challenge Award competition results.

Conclusions and Future Work

We have previously introduced a common metric for measuring relative development time productivity of HPC software development. The RDTP metric has been applied to data from benchmark codes and classroom experiments, with consistent trends for various programming models.

In general the results support the theory that traditional HPC programming models such as MPI yield good speedup but require more relative effort than other implementations (Figure 5). OpenMP generally provides speedup comparable to MPI, but requires less effort. This leads to higher values of the RDTP metric. There are questions of scalability with regard to OpenMP that are not addressed by this study.

The pMatlab implementations of HPC Challenge provide an example of a language that can yield good speedup for some problems, while requiring less relative effort, again leading to higher values of the RDTP metric.

Further classroom experiments are planned, and as more data is collected it will be analyzed in the same manner, to see if other trends emerge. Our current work is focused on designing a standard framework for automating the collection, storage, and analysis of workflow data (see "Modeling HPC workflows with timed Markov models" in this issue).

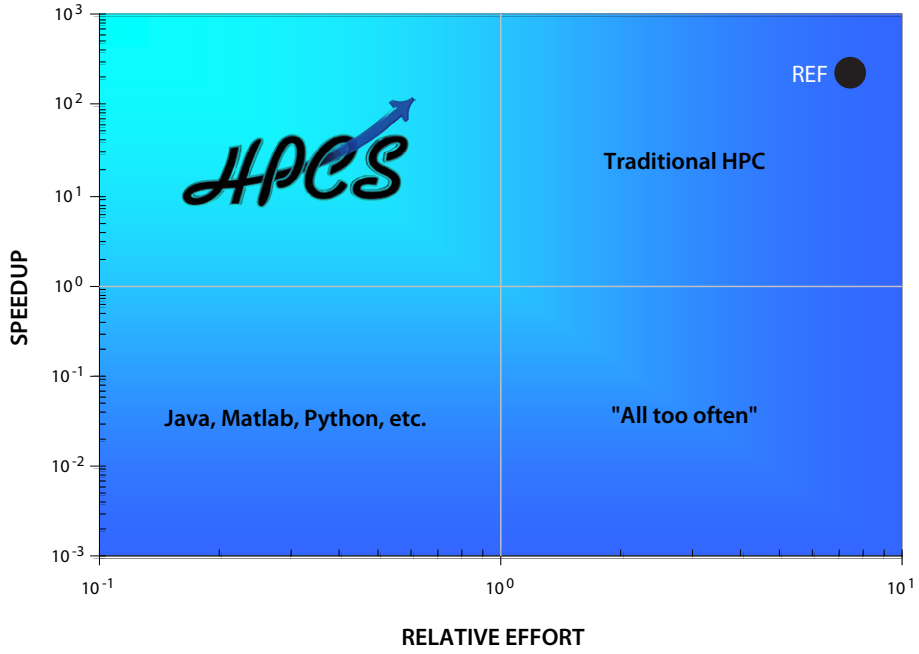

Figure 5. Speedup and Relative Effort are both important indicators of time to solution.

Acknowledgments

We wish to thank all of the professors whose students participated in this study, including Jeff Hollingsworth, Alan Sussman, and Uzi Vishkin of the University of Maryland, Alan Edelman of MIT, John Gilbert of UCSB, Mary Hall of USC, and Allan Snavely of UCSD.

This work is sponsored by Defense Advanced Research Projects Administration, under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Software Productivity Research In High Performance Computing

Susan Squires
Sun Microsystems Inc.
Michael L. Van De Vanter
Sun Microsystems Inc.
Lawrence G. Votta
Sun Microsystems Inc.

1. Introduction

The challenge of utilizing supercomputers effectively at ever increasing scale is not being met,[1] a phenomenon perceived within the high performance computing (HPC) community as a crisis of "productivity." Acknowledging that narrow focus on peak machine performance numbers has not served HPC goals well in the past, and acknowledging that the "productivity" of a computing system is not a well-understood phenomenon, the Defense Advanced Research Project Agency (DARPA) created the High Productivity Computing Systems (HPCS) program:[2]

- Industry vendors were challenged to develop a new generation of supercomputers that are dramatically (10 times!) more *productive*, not just faster; and
- A community of vendor teams and non-vendor research institutions were challenged to develop an understanding of supercomputer productivity that will serve to guide future supercomputer development and to support productivity-based evaluation of computing systems.

The HPCS Productivity Team at Sun Microsystems responded with two commitments:

1. Embrace the broadest possible view of productivity, including not only machine characteristics but also human tasks, skills, motivations, organizations, and culture, just to name a few; and
2. Put the investigation of these phenomena on the soundest scientific basis possible, drawing on well-established research methodologies from relevant fields, many of which are unfamiliar within the HPC community.

Team members brought expertise from multiple research fields with a specific focus on a sound working knowledge of concepts and methods appropriate to investigating human behavior. Socio-cultural concepts such as culture, ethnography, and social network analysis are not typically well understood in the computing community, sometimes leading to re-invention of methods already developed and validated in the social sciences. The first author is a social science professional with expertise in established practice. Other research-level expertise in the team included physics (both experimental and computational), software development (both technologies and human factors), and empirical software engineering. Given the breadth of the challenge and the small team size, a first principle was that every project demands careful – and quick – determination of appropriate outcomes, project constraints, and research methods. Conclusions must be founded in data and backed by justification for the design, execution, and application. (See also Kitchenham et. al. for general guidelines on conducting empirical research.)[3]

Social scientists have developed numerous methods that are both verifiable and reproducible in many contexts. However, the sheer number of methodological options makes it crucial that each project begin with clear research goals in order to identify the most effective combinations of concepts, research designs, information sources, and methods.

[1] Post, D.E., Votta, L.G. "Computational Science Requires a New Paradigm," *Physics Today*, 58(1):35-41.

[2] Defense Advanced Research Project Agency (DARPA) Information Processing Technology Office, High Productivity Computing Systems (HPCS) Program - http://www.darpa.mil/ipto/programs/hpcs/

[3] Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D. El Emam, K., Rosenberg, J. "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering*, 28:8, August 2002, pp. 721-734.

The research design presented in this paper is a three-stage framework, based on the scientific method, that allows the team to draw on multiple research disciplines appropriate to the phenomena under investigation. The framework is grounded in empirical data, validated by multiple approaches ("triangulation"), and applied to the practicing HPC professionals who actually perform the work being studied.

Research findings described in this paper must be understood in the context of the framework, as described in Section 2: definitions of the stages, methods used to collect and analyze information within each stage, and the relationships among the stages. Sections 3, 4, and 5 discuss how the framework was applied to studies of the HPC software development community. Although the productivity research program is still in progress, significant findings have already contributed to the community's understanding of HPC software development productivity.

2. A Three-stage Research Design

The research design is summarized in Figure 1. The stages are necessarily sequential; each provides a foundation for research methods in the next.

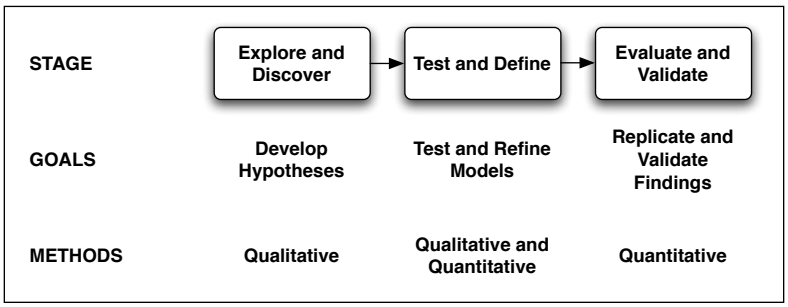

Figure 1. Research framework.

This framework is broadly analogous to realizations of the scientific method used in many disciplines, although the correspondence can be obscured somewhat by differences among the phenomena being studied and the methods appropriate to their study.

Stage 1: Explore & Discover. At the outset of an investigation researchers may not know the appropriate questions or issues to address, let alone have a coherent theory of the phenomena being studied. The first stage in the framework, based mainly on qualitative methods, is open-ended; it is designed to produce the insights necessary for hypothesis generation. Such insights can be considered an explicit "model" of the phenomena, analogous to the "paradigm" in which scientific inquiry is undertaken and through which experimental data are interpreted.

Stage 2: Test & Define. Rigor is added using methods that produce additional data surrounding theories generated in the first stage. This provides feedback on insights, supplies concrete data for model refinement, and leads to deeper understanding of what can be measured and how those measurements can be interpreted. This stage of "experimental design" is the necessary bridge between theory and experimentation.

Stage 3: Evaluate & Validate. Finally, more focused and mostly quantitative techniques are applied precisely in order to collect data, interpret the results, and validate the outcomes.

An important benefit of this framework is the reliable "roadmap" for moving from qualitative data collection and analysis, which are appropriate for discovery, to more quantitative data collection and analysis methods, which are appropriate to definitional and evaluative research.

The immediate challenge in a study of such breadth as software development is not so much in creating new methods, although researchers can be tempted to do so when investigating phenomena outside their area of training. A great many methods are available for studying human behaviors, the vast majority of which are known to graduate students who have taken class work in research design, advanced statistics, ethnographic interviewing, and content analysis.[4] The greater challenge is to determine exactly when, how, and why to use particular methods to draw out the implications of findings. This research framework makes it possible to select the appropriate qualitative and quantitative methods for data collection and analysis.

[4] Bernard, H. *Handbook of Methods in Cultural Anthropology*. Walnut Creek: Altamara Press. 1999.

Stage	Character	Methods
1. Explore & Discover	Open	Ethnography Case study Contextual observations Semi-structured Interviews Participation Document review Language patterning
2. Test & Define	Focused	Quasi-experimental studies Concept mapping Structured interviews Questionnaires Comparative studies Focus groups Semiotic analysis
3. Evaluate & Validate	Structured	Social network analysis Surveys Controlled experiments Product testing User-experience simulations Human testing Quality measurement

Table 1. Sample methods from the research toolbox.

Table 1 displays a sample of specific data collection and analysis methods appropriate to each stage, with the more qualitative methods appearing early and more quantitative methods appearing later. For example the case study with its open-ended qualitative observational and interview methods is most useful for early discovery research; on the other hand, surveys are appropriate in later stages when their design can be informed by data already collected. Data collection methods are just that: standardized techniques that enable researchers to gather valid and relevant information.

Qualitative and quantitative techniques are both useful, but only when used appropriately. For example, some methods are better to discern patterns, others to test specific ideas. Some methods are better to describe behavioral phenomena, others for opinions. Some methods excel at documenting how people interact with things, others at discerning how they interact amongst themselves. Some methods are better for discovering just how similar things or people are; others are better at teasing out causes from consequences. Qualitative data provide tremendous clues about people on a group level; examples of such "cultural" clues include descriptions of perceptions about how people use or classify objects, the nature of their personal interactions,

and opinions about the world around them. Quantitative data are also extremely useful, for example the number of lines of code (LOC) contained in an application or the number of full time personnel on a code project.

When techniques are combined appropriately, based on sound methodology, it becomes possible to create and distinguish among the most important human processes within and across groups of people.

3. Discovery Research, Stage 1: The Use of the Case Study Method

Discovery is the most open-ended and the most time consuming of the three research stages. The goal is to uncover and understand the socio-cultural system that frames human action; that understanding must be consistent with the way local people understand it and it must be expressed in terms of the local (emic) categories people use to describe and categorize their own reality. Researchers collect and analyze verbal, observational and contextual information to characterize what people say and do in their natural environment. Consistencies and, more frequently, inconsistencies help identify unarticulated or unrecognized needs, gaps and adaptations often called "work-arounds" and "disconnects." Translating disconnects into the frame of reference of socio-cultural systems allows the researcher to identify and neutralize well-established assumptions. Such assumptions would otherwise be taken as given, which leads to stereotypic treatment and precludes understanding of the important and difficult issues.

3.1 HPCS Stage 1 Methods

The key to a successful discovery in a context such as HPC software development is the combination of two approaches: case studies and rapid ethnographic methods.

The *case study approach* is a well understood method for gathering initial information about a situation, as defined by Robert K. Yin: "an in-depth look at one or more specific incidents or examples."[5] The case study typically employs a set of qualitative, open-ended methodologies to explore a topic or problem domain and develop hypotheses. Methods may include data collection techniques such as Document Reviews, Observation, Collection of Contextual Artifacts, Self-Reporting, and Interviews. The breadth of data that can be collected provides foundational knowledge for developing hypotheses. The case study approach has been used in software engineering[6][7][8][9] and has played a significant role in the HPCS productivity research program.[10][11][12][13][14]

The objective of *rapid ethnographic* assessment in discovery research is typically to construct a socio-cultural model of the local living system.[15][16] All rapid ethnographic approaches share three important characteristics "(1) a system perspective, (2) triangulated data collection, and (3) iterative data collection and analysis".[15][4] The value of the case study and rapid ethnographic assessment approaches is growing; they have been used by the National Center for Atmospheric Research,[17] Department of Energy,[18][19][20] NASA,[21] and for describing technical change at the Department of Defense.[22]

The application of these approaches stresses open-ended interviews, site tours (contextual observation), participant observation, literature reviews, cultural history, and semiotic (content) analysis. An example from the HPCS research will demonstrate how discovery research gen-

[5] Yin, R.K. *Case Study Research: Design and Methods. Sage Publications*, Second Edition, 1994.

[6] Card, D.N., Church, V.E., Agresti, W.W., "An Empirical Study of Software Design Practices," *IEEE Transactions on Software Engineering*, 1986. 12(2): 264-271.

[7] Müller, M.M. and Tichy, W.F. "Case Study: Extreme Programming in a University Environment," In *Proceedings of 23rd International Conference on Software Engineering*. May 12-19, 2001. pp. 537-544.

[8] Perry, D.E., Sim, S.E., and Easterbrook, S.M. "Case Studies for Software Engineers," In *Proceedings of 26th International Conference on Software Engineering*, ICSE 2004. pp. 736-738.

[9] Seaman, C.B. and Basili, V.R. "An Empirical Study of Communication in Code Inspections," In *Proceedings of 19th International Conference on Software Engineering*. Boston, MA. May 17-23, 1997. p. 96-106.

[10] Carver, J., Hochstein, L., Kendall, R., Nakamura, T., Zelkowitz, M., Basili, V., Post, D. "Observations about Software Development for High End Computing," *CTWatch Quarterly*, Volume 2, Number 4A, November 2006 – http://www.ctwatch.org/quarterly/

[11] Kendall, R., Carver, J., Mark, A., Post, D., Squires, S., Shaffer, D. "Case Study of the Hawk Code Project," *Los Alamos National Laboratory Report* LA-UR-05-9011, 2005.

[12] Kendall, R., Post, D., Squires, S., Halverson, C. "Case Study of the Condor Code Project," *Los Alamos National Laboratory Report* LA-UR-05-9291, 2005.

[13] Kendall, R., Post, D., Squires, S., Carver, J. "Case Study of the Eagle Code Project," *Los Alamos National Laboratory Report* LA-UR-06-1092.

[14] Post, D., Kendall, R., Whitney, "Case Study of the Falcon Code Project," *Proceedings Second International Workshop on Software Engineering for High Performance Computing System Applications*, St. Louis, 15 May 2005.

[15] Beebe, J. "Basic Concepts and Techniques of Rapid Appraisal," *Human Organization*, 54(1): 42-51. 1995.

[16] Trotter, R., Schensul, J. "Methods in Applied Anthropology," in *Handbook of Methods in Cultural Anthropology*, H. Russell Bernard (ed.), Walnut Creek: Altamara Press. 1999.

[17] Lahsen, M. "Seductive Simulations: Uncertainty Distributions around Climate Modeling," *Social Studies of Science* 36(6): 895-992. December 2005.

[18] Gusterson, H. *Nuclear Rites: A Weapons Laboratory at the End of the Cold War*, University of California Press: Berkeley. 1996.

[19] Gusterson, H. *People of the Bomb: Portraits of America's Nuclear Complex*, University of Minnesota Press: Minneapolis. 2004

[20] McNamara, L., Trucano, T. "So Why Do You Trust That Model? Some Thoughts on Modeling, Simulation, Social Science and Decision Making," *Advanced Concepts Group News and Views,* 8:2. Albuquerque, NM: Sandia National Laboratories. 2006.

[21] Shalin, V., Wales, R., "Shift Handovers in ISS Mission Control," in *Human Organizational Risk Management Workshop,* NASA-Ames April 25-27 2001.

[22] MacKenzie, D. "Missile Accuracy: A Case Study in the Social Processes of Technological Change," in *The Construction of Technological Systems*, Wiebe Bijker, Thomas Hughes and Trevor Pinch (Eds) MIT Press: Cambridge MA. 1987.

erates cultural insights; those cultural insights lead to better understanding and help develop hypotheses that can be tested in the subsequent research stage.

3.2 Example: The HPC "Expertise Gap"

Anecdotal evidence from DARPA and the HPCS Mission Partners suggested that an "expertise gap" lay at the heart of the crisis in HPC application development.[23] Case studies were conducted first to explore the expertise issue, starting with a detailed look at how professional HPC programmers and teams spend their time. Qualitative data collection methods included semi-structured interviews with individual HPC programmers, and contextual observations at sites in which HPC programmers work. Of course the raw data from these methods did not directly lead to the kind of insights that are the goal of this research stage. Combining and comparing the data, the team began to identify patterns across individuals and teams, plot bottlenecks and create models of HPC programmers, all based on information taken directly from the HPC professionals and the context of their work.

[23] Sarkar, V., Williams, C., Ebcioglu, K. "Application Development Productivity Challenges for High-End Computing," *First Workshop on Productivity and Performance in High-End Computing*, Madrid Spain, pp 14-18. February 2004.

From five case studies a pattern emerged in the area of expertise. In all cases at least one founding team member had been recruited for special knowledge of science, but in each case the scientist was not an HPC programmer and had little or no knowledge of FORTRAN or C++. The scientist's first required task was either to learn one of the programming languages or to build a working relationship with someone who did know it. In either case, the educational process took considerable time before the individual/pair could perform effectively. Project management was typically taken on by another person whose role was to "run interference" by keeping the sponsor happy and negotiating for time on a shared large machine. Teams in this context typically take about four to six years to get a working code. Success is commonly attributed to having the right mix of expertise. The team was successful only when they had the appropriate mix of knowledge, represented by four areas:

- Science
- Programming
- Scaling / Optimizing
- Management

Even having the range of knowledge is insufficient. Effective communication and collaboration among the experts can be very challenging and is crucial to project success.

Underlying the ethnographic case study approach is the understanding that all people belong to one or more networks of interlocking social relationships in which members share a common or core set of beliefs, values and behaviors. Anthropologists and other trained ethnographers use various methods to uncover the core sets of beliefs, such as:

- Gathering individual (emic) perspectives from members of these socio-cultural groups;
- Examining the collected information to identify patterns of shared beliefs, behaviors, values and rules;
- Constructing group "mental models" from identified patterns to understand the meaning at the core of the system; and
- Interpreting how the members of a socio-cultural network use their mental models to construct and express appropriate shared behaviors, beliefs, and values, to provide a contextual frame of meanings for products, and services.

The case studies of HPC software development led to recognizable patterns that might explain why HPC expertise is so scarce. A hypothesis was developed postulating that domain specific expertise in at least four different areas is needed to use highly parallel machines. As machines get bigger and more complex, the pool of experts narrows. Very, very few people have complete skill sets. Team approaches are the best strategy at the moment, but this by itself does not appear to represent a long-term solution. The next research step was to craft a more focused study to test the hypothesis and to understand in more detail when and how the various areas of expertise were used; this takes place in the second stage of the research framework.

4. Definition Research, Stage 2: Combining Qualitative with Quantitative Measurement

Definition Research helps test an idea or hypothesis focusing the research on more detailed use, use features, and meaning associated with activities and helps define work models and workflow. The methods used during Definition Research differ from Discovery because of what has already been learned during Discovery: parameters of the topic, the concepts, and something about the individuals. Definition Research usually starts with a series of questions, based on some grounded hypothesis generated during discovery. For example, researchers investigating programming techniques can compare existing codes in context to make inferences about how proposed changes might change both the programming work and the results.

Definition research concentrates on details, so any interviews conducted in this stage are more structured than during Discovery. These interviews follow a set of well-understood rules; for example the interviewer builds rapport in the first segment and subsequently seeks deeper information. Details are summarized at the conclusion of the interview in order to confirm the data with the respondent. Verbal and written statements are validated through observed actions. In order to fully understand the specifics of the context, researchers listen for native language: words, terms, and descriptions.

4.1 HPCS Stage 2 Methods

An advantage of both Discovery and Definition Research is that relatively few cases are needed to discern relevant cultural patterns and to learn about shared understandings and behaviors in a group. This approach has powerful implications for quantifying human behavior patterns in ways never imagined a few years ago. Notre Dame mathematician Albert-Laszlo Barabasi recently wrote in *Science* that our grouping ability to collect data about human actions combined "with the sophisticated tool of network theory, . . . (provides) a glimpse of an unprecedented opportunity to quantify human dynamics."[24] Social network analysis can take millions of bits of data and construct reliable and predictive human patterns. But such an approach can begin with small data sets as well.

[24] Barabasi, A.L. "Network Theory – the Emergence of the Creative Enterprise," *Science*, 308(29): 639-650. April 2005.

Mathematician Duncan Watts points out that "the world that we live in is not at all random. We are very much constrained by our socioeconomic status, our geographical location, our background, our education and our profession, our interests and hobbies. All these things make our circle of acquaintances highly nonrandom."[25] Watts and fellow mathematician Steven Strogatz are among a growing number of researchers who have been examining highly structured social networks in order to understand and use mathematical formulations to predict membership connectiveness. Their work has been inspired by, and extends the work of, the theoretical mathematician Paul Erdös, who has been indirectly responsible for popularizing

[25] Watts, D., Strogatz, S. Interview in *Discover.* 1998.

the idea of six degrees of separation: the idea that only six other individuals link all humankind through their social networks to almost everybody else in the world.

Although Watts and Strogatz are most interested in the interconnectiveness of social networks, many others such as Borgatti and Everett[26] (see also *Mathematical Social Sciences* and *Social Networks*) focus on the highly nonrandom nature of social networks. Their goal is to devise statistically reliable mathematical formulae that predict the number of individuals who need to be interviewed in order to capture the shared characteristics of social networks: shared beliefs, values and behaviors. The analysis of Handwerker and Wozniak suggests the surprising low number of seven,[27] although this number is reliable only when certain criteria are met:

[26] Borgatti, S., Everett, M.G. "Network Analysis of 2-mode Data," *Social Networks*, 19(3): 243-269. 1997.

[27] Handwerker, W.P., Wozniak, D. "Sampling Strategies for the Collection of Anthropological Data: An Extension of Boaz's Answer to Galton's Problem," *Current Anthropology*, 38(5): 869-875. 1997.

1. The information gathered is about shared or core understandings within the social network. This is not about group variation.
2. The cognitive domain of the social network is internally consistent and ordered.
3. Information is gathered from key members of the social network: cultural experts.
4. Informants are interviewed independently of others in their social network. Informants must not be allowed to confound information by checking or comparing notes with other members.
5. There are no known divisions within the social network: no sub-groupings that might have distinct sets of core knowledge and behaviors.[27]

One way to determine if a social network is bounded is to analyze patterns that emerge from the data. Highly redundant information about membership of a group and their perceptions of cultural norms are strong evidence that there is consensus about who is in the group and what they consider appropriate. However, if any of the criteria are not met, then another individual must be interviewed. Once a consensus is identified, then it is recorded as a discovery and the researcher moves on. In our Discovery research we were able to identify these shared work patterns with a high level of confidence, making it possible to identify potential participants for the following research stage.

One of the HPCS goals during the Definition research stage was to understand the workflow of HPC programmers and to identify how and where current HPC development paradigms limit code development. For example, a hypothesis from the Discovery stage was that one limiting factor is the level and range of expertise needed. Definition research was designed to validate the hypothesis and, if validated, shed additional light on the development process: where specific expertise was deployed and where programmers encounter significant time and effort bottlenecks.

4.2 Example: HPC Workflow

Two methods were chosen to collect data: one quantitative, one qualitative. Quantitative information on programmer activity (time on task details) was collected using HackyStat, an in-process software engineering measurement and analysis tool.[28] HackyStat recorded hours of event traces from development tools (for example "open file," and "build") while HPC professionals developed code. Additional (qualitative) data was collected in the form of real-time, time-stamped journals written by professional code developers who agreed to record a personal narrative of their work.

[28] Johnson, P., Paulding, M. "Understanding HPC Development through Automated Process and Product Measurement with HackyStat," *Second Workshop on Productivity and Performance in High-End Computing* (P-PHEC), San Francisco, Feb. 13, 2005.

By combining HackyStat telemetry data that measured activity with programmer journals, the team was able to corroborate, validate, and interpret results. For example, the significance of the expertise gaps and bottlenecks to the HPC productivity problem became apparent when studying individual professionals and their time usage; the journal entries represented real-time accounts of code development from the programmer's perspective. These patterns were used to define a typical HPC development workflow, identify where in the workflow the most effort is being expended, characterize the expertise profiles associated with workflow tasks, and draw conclusions about productivity bottlenecks and their root causes.

These results are summarized in Figure 2, which illustrates the typical workflow that developers go through in creating and optimizing HPC applications along with the skill sets required to perform each activity. A typical workflow includes understanding the problem that needs to be solved, formulating an initial computational solution, empirically evaluating the proposed solution through prototyping or experimentation, coding for sequential execution, evaluating the overall computational approach, then coding and optimizing the results for a parallel platform.

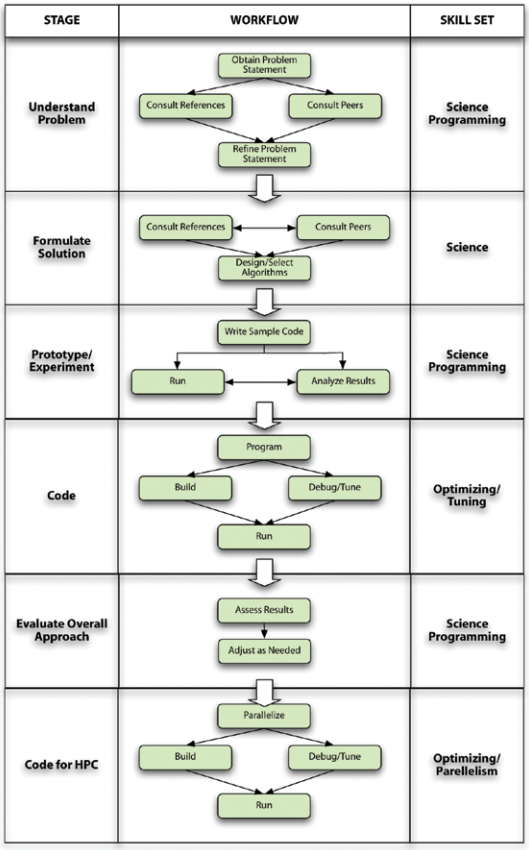

Figure 2. Work Model of HPC Programmers.

Activities that consume the greatest proportion of resources, effort, time, and expertise, within the overall programming effort were also identified:

- *Developing correct scientific programs:* activities associated with translating an understanding of the scientific problem that must be solved (e.g., a predictive weather model) into code.
- *Code optimization and tuning:* activities associated with refining a serial version of the code to ensure correctness and achieve desired levels of accuracy and efficiency.
- *Code parallelization and optimization:* activities associated with parallelizing the code and tuning to achieve high machine utilization and rapid execution.
- *Porting:* where a solution exists, this comprises the activities associated with translating the existing solution to a representation appropriate for a new computing platform.

Once the expertise problem was understood and the likely location that consumed the most time and effort for those experts was pinpointed, the question of how widely these findings could be applied was raised. Full validation required mapping the extent of the expertise gap, calling for a large quantitative survey of the sort suitable for the next stage of the research framework.

5. Evaluative Research, Stage 3: Developing Quantitative Models for validating HPCS Programmer Workflow

Like rapid ethnographic research, evaluative research has its own history that can be traced back to the middle of the 20th century. In 1967 Michael Scriven proposed that all evaluation could be broken down into two distinct types, formative and summative evaluation.[29] Formative evaluation validates and improves upon an idea or hypothesis. Summative evaluation answers the question, "to what extent?" Evaluative researchers use a toolbox of methods from many of the social sciences to validate a hypothesis or determine its extent in a population. Methods are typically quantitative. Again the HPCS research provides an example, although this phase of the program is in very early stages and the results are still preliminary.

[29] Scriven, M. "Beyond Formative and Summative Evaluation," In G. W. McLaughlin & D. C. Phillips (Eds.) *Evaluation and Education: At Quarter Century.* Chicago, IL: University of Chicago Press, pp. 19-64. 1991.

One of the patterns that emerged from case study research suggested that HPCS code teams were more concerned about programming correctness than performance. The workflow pinpointed the most likely places where a programmer would find the most difficulty. Up to this point the conventional wisdom had been accepted without question, namely that performance was the paramount concern of the programmer. To evaluate the extent of this pattern, which was uncovered in Stage 1 and 2, a survey was administered to HPC programmers at National Labs and in private institutions where large, highly parallel code is written. Quantitative statistical procedures were used to analyze the survey data. Table 3 provides the resulting response distribution when asked about the top issues facing the HPC programmer.

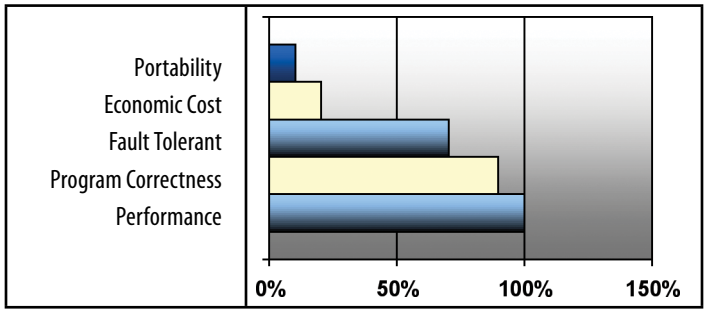

Table 2. Response Distribution of Top Issues.

The case studies in the first stage of this research had provided information about what programmers said and the empirical studies of programmers in the second stage supported the findings from the earlier research and validated the workflow of the programmer with the quantitative data. Not surprisingly, the survey data collected in the third stage of the research confirmed that performance is important in HPC code, but also confirmed that programming correctly is at least as important, as reported by 90% of those surveyed. The perceived value of performance is a strong shared value in the HPC community; however, the concern about correctness appears to be at least as strong although not verbalized as often.

This example highlights the need to follow the research stages from hypothesis development (Stage 1) and question generation (Stage 2) before attempts to quantify. Unfortunately there is a tendency to jump to quantitative research because of the supposed reliability of numbers. However, quantitative results are only as good as models of the phenomena that are the context for interpretation of the data. In the example, without having proceeded through the logical progression from hypothesis to validation, the significant concern for program correctness might have been overlooked. And, of course, the link between correctness and need for expertise is

clear. Such an oversight might have led to hardware and software design decisions that turn out to be counterproductive in the area of program correctness.

6. Conclusions

The three research stages and associated methods described in this paper have immense potential to increase understanding of software development, both in the HPC community and beyond. Research results to date include fundamental discoveries about productivity.[30] [31] [32] [33] These findings are grounded in empirically validated models that reflect the experience of practicing HPC professionals.

We are just beginning to understand how central to the efforts of HPCS research are the essential and intimate relationships among people, tools, and code: independent changes in each are unlikely to produce the dramatic 10x increase in software productivity that was envisioned by the founders of the DARPA HPCS program and which is desperately needed by the HPC community. Meeting that goal demands aligning those changes around a deep understanding of what makes software development productive: for machines, for individuals, for organizations, and for communities. As the technology historian, Kingery noted, "... No one denies the importance of things, but learning from them requires rather more attention than reading texts."[34]

Acknowledgments
We would like to thank our HPCS colleagues at Sun Microsystems and elsewhere in the HPC community for their helpful discussions and comments.

[30] Loh, E., Van De Vanter, M. L, Votta, L.G. "Can Software Engineering Solve the HPCS Problem?" in *Proceedings of Second International Workshop on Software Engineering for High Performance Computing System Applications*, St. Louis, 15 May 2005.

[31] Squires, S., Tichy, W.F., Votta, L.G. "What Do Programmers of Parallel Machines Need? A Survey," *Second Workshop on Productivity and Performance in High-End Computing* (P-PHEC), San Francisco, Feb. 13, 2005.

[32] Squires, S., Van De Vanter, M. L, Votta, L.G. "Yes, There Is an 'Expertise Gap' in HPC Application Development," *Third Workshop on Productivity and Performance in High-End Computing* (P-PHEC), Austin, Feb. 12, 2006.

[33] Van De Vanter, M. L, Post, D.E, Zosel, M. "HPC Needs a Tool Strategy," in *Proceedings of Second International Workshop on Software Engineering for High Performance Computing System Applications*, St. Louis, 15 May 2005.

[34] Kingery, W. (Ed), Editor's Preface, *Learning from Things: Method and Theory of Material Culture Studies*. Washington, D.C.: Smithsonian Institution Press. 1996.

CTWatch QUARTERLY

ISSN 1555-9874 VOLUME 2 NUMBER 4A **NOVEMBER 2006**

High Productivity Computing Systems and the Path Towards Usable Petascale Computing

GUEST EDITOR **JEREMY KEPNER**

AVAILABLE ON-LINE:
http://www.ctwatch.org/quarterly/

E-MAIL CTWatch QUARTERLY:
quarterly@ctwatch.org

CTWATCH IS A COLLABORATIVE EFFORT

http://icl.cs.utk.edu/

http://www.ncsa.uiuc.edu/

http://www.sdsc.edu/

CTWATCH IS A PUBLICATION OF THE CYBERINFRASTRUCTURE PARTNERSHIP

http://www.ci-partnership.org/

SPONSORED BY