Yes, There Is an "Expertise Gap" In HPC Applications Development

Susan Squires
Sun Microsystems, Inc.
15 Network Circle UMPMK15-204
Menlo Park, CA 94025
1-650-786-3441
susan.squires@sun.com

Michael L. Van De Vanter
Sun Microsystems, Inc.
16 Network Circle UMPMK16-304
Menlo Park, CA 94025
1-650-786-8864
michael.vandevanter@sun.com

Lawrence G. Votta
Sun Microsystems, Inc.
16 Network Circle UMPMK18-216
Menlo Park, CA 94025
1-650-786-7514
lawrence.votta@sun.com

ABSTRACT

The High Productivity Computing Systems (HPCS) program seeks a tenfold productivity increase in High Performance Computing (HPC), where productivity is understood to be a composite of system performance, system robustness, programmability, portability, and administrative concerns. Of these, programmability is the least well understood and perceived to be the most problematic. It has been suggested that an "expertise gap" is at the heart of the problem in HPC application development. Preliminary results from research conducted by Sun Microsystems and other participants in the HPCS program confirm that such an "expertise gap" does exist and does exert a significant confounding influence on HPC application development. Further, the nature of the "expertise gap" appears not to be amenable to previously proposed solutions such as "more education" and "more people." A productivity improvement of the scale sought by the HPCS program will require fundamental transformations in the way HPC applications are developed and maintained.

Categories and Subject Descriptors
D.2.0 [**Software Engineering**]. D.1.3 [**Programming Techniques**]: Concurrent Programming –*parallel programming*.

Keywords
High Performance Computing, Software Productivity.

1. INTRODUCTION

The development of application software in the High Performance Computing (HPC) domain is extraordinarily difficult. Indeed the Defense Advanced Research Project Agency (DARPA) has called out programmability as one of the key goals of the High Productivity Computing Systems (HPCS) program. DARPA understands productivity to be a composite of system properties and has set HPCS program goals in each [2]:

- **Performance** (time-to-solution): speedup critical national security applications by a factor of 10X to 40X
- **Programmability** (idea-to-first-solution): reduce cost and time of developing application solutions (~10X)
- **Portability** (transparency): insulate research and operational application software from system
- **Robustness** (reliability): apply all known techniques to protect against outside attacks, hardware faults, & programming errors
- A fifth property, **Systems Administration**, was added at the suggestion of Sun Microsystems, one of the HPCS vendors.

Acknowledging that productivity is fundamentally not well understood, DARPA has also funded a broad research program on HPC productivity that engages universities, research labs, the program vendors, and the HPCS "Mission Partners," including the Department of Defense, Department of Energy and federally funded sites such as Sandia National Laboratory and Los Alamos National Laboratory.

Of the system properties that make up productivity, programmability is the least well understood and perceived to be the most problematic. It has been suggested that an "expertise gap" is a significant barrier to increased software productivity: "Programming today's HEC[1] systems requires a high level of expertise, but the current trend of available skills is increasingly one of few HEC expert programmers …" [12].

A goal of the HPCS Program is to create a base of empirical data describing all aspects of productivity. Although the program is far from complete, preliminary analysis of data concerning programmability confirms that there is in fact an HPC "expertise gap" and that it is

Conference PPHEC'06, February 12, 2006, Austin, Texas, United States.

[1] High-End Computing, another term for High Performance Computing (HPC)

fundamental to the ways software is developed in many areas of the HPC domain.

This paper presents those preliminary analyses, beginning in section 2 with a discussion of the environment in which HPC software is developed, maintained, and evolves. Many of these insights about the environment have themselves been produced by HPCS program research. Section 3 describes the interdisciplinary research methodology used by the Sun productivity program, drawing on a variety of social science and empirical software engineering techniques. Selected preliminary findings are presented in section 4, with primary emphasis on suggestive case studies. Section 5 discusses some of the implications of these findings for the HPCS program goals.

2. BACKGROUND AND RESEARCH GOALS

The HPC environment is characterized by very large problems (measured by both data size and computation) that require very large, sometimes exotic computing systems and the exploitation of very high degrees of parallelism. The focus of the HPCS program has been on "Mission Partners" in the United States: the Department of Defense, the Department of Energy, and the intelligence community. HPC also takes place in commercial environments and those have been the subject of some studies as well.

Case studies published to date describe an environment that has much in common with other kinds of software development, but which differs markedly in a few respects [4][5][9][10]. The following characteristics are important for the purposes of this discussion:

1. HPC codes typically take years to develop.
2. Once developed, HPC codes have a maintenance and evolution lifetime that can be counted in decades.
3. Because HPC codes have such a long use life, they may be ported to new machines several times during their lifetime, possibly every 3-4 years.
4. HPC codes embody very complex science and numerical methods.
5. As a result of their age, legacy codes are typically written in Fortran77 and MPI. This sets them apart from other kinds of programming.
6. New code is often being written in C++.
7. HPC codes are often supported by tools that are specifically written for them.

The goal of our research is to collect empirical data and build understanding of HPC application development, with a particular emphasis on characterizing the key "bottlenecks" that appear to have negative impact on application development effectiveness. From this data we intend to develop a model for reasoning about programmability and for predicting how bottlenecks might be ameliorated.

3. METHODOLOGY

To gather more detailed information about HPC software development, we adopted a case study approach; this approach provides a data collection framework that is flexible and allows deep exploration of one or more cases [4][5][9][10]. We used multiple methods from case study research [14] that would allow us to gather information from professionals and teams of professionals who are writing code for highly parallel machines. These methods included qualitative data collection including:

- Semi-structured interviews with individual HPC programmers.
- Structured group sessions (surveys and interviews) with existing Mission Partner code teams.

We also used quantitative methods that allowed us to validate case study findings across a larger sample, including a survey of Mission Partner teams, using a multiple choice and open-ended questions format.

The use of mixed methods, measuring, observing and interviewing programmers, has a number of advantages over reliance on one type of data collection alone. For example, journal entries allowed us to collect "real-time" accounts of code development from a programmer perspective. Individual and group interviews help us understand long-term team and context issues. Survey results provide a structured overview and validated observations.

Of course the raw data alone did not provide us with the insights we seek. By weaving together the qualitative data, we were able to compare journal entries, survey responses, and interviews in order to fill out our understandings and isolate inconsistencies. From the combined data we then began to identify patterns across individuals and teams, plot bottlenecks and create models of HPC programmers based on empirical data.

4. FINDINGS

The data collected so far shed light on the suggested "expertise gap" that is widely reported in anecdotal form, for example as noted by Sarkar et. al. [12]. Analysis can be carried out from more than one perspective. This paper examines insights gained by identifying "bottlenecks:" barriers to the accomplishment of development goals. Analysis from a workflow perspective will be reported separately.

4.1 "Hot Spot" interviews

The Sun Microsystems productivity team began to investigate this evidence with a systematic analysis of data collected from interviews with five Mission Partners, conducted through DARPA's productivity research effort. There were three components to the data [3]:

1. A set of proposed workflow diagrams, describing idealized work processes in different aspects of HPC

software development; respondents were invited to review the diagrams and mark those workflow nodes they perceived to be "hot spots" causing particular difficulty or expense.

2. Free-form comments from interview respondents.
3. An oral debriefing of the interviewer, during which additional respondents' comments, not otherwise recorded, were reported.

No clear consensus emerged from the respondents' node selections, but additional analysis that included both sets of comments suggested that respondents had experienced some difficulty mapping the proposed workflows into their own work practices. A revised analysis at a coarser granularity, taking full account of the comments, revealed a clear consensus, as represented in the following chart.

	MP1	MP2	MP3	MP4	MP5
Develop HPC Code	x	x	n/a		x
Debug, Test, V&V	x	x	x	x	x
Optimize Code	x	x	x	x	x
Schedule/ Run Code	x		x		x
Math Libraries	x	x	n/a		x

Chart of Mission Partner Bottlenecks

The chart depicts the areas where data revealed each Mission Partner to be experiencing difficulty, i.e. where HPC code developers spend the greatest time and effort. Five areas were reported by a majority of Mission Partners:

1) Developing HPC Code,
2) Debugging, Testing and Verification & Validating,
3) Optimizing Code,
4) Scheduling Code Runs, and
5) Creating, Selecting, and Using Math Libraries

Areas identified by fewer Mission Partners do not appear in the chart. Mission Partner 3 does not develop code.

Importantly, of the five areas that consume time and effort, Mission Partners unanimously reported that optimizing, debugging, testing, and Verification and Validation of code were bottleneck areas.

We concluded that scaling code for HPC systems was the most likely area to become a bottleneck, which supports anecdotal reports that specialized "expertise" is crucial for scaling code for High Performance Computing (HPC). A lack or "gap" in the availability of that expertise might be an important factor blocking increased HPC productivity.

4.2 Classroom "defect" studies

Recent quantitative research from a study conducted at the University of Maryland has confirmed similar bottleneck areas while studying novice developers [7]. In this study, students were monitored while developing solutions to small HPC programming problems; all of the students' code runs were examined and the "defects" in faulty codes categorized. Those findings match the first two bottleneck areas identified in Sun's analysis: Coding for HPC, and Optimizing Code. The novices being studied do not engage in validation, scheduling or significant library use.

We realized that we would need to conduct additional research to explore and validate our initial findings as well as expand our knowledge of the expertise gap.

4.3 Scale up with more education?

When people are working hard but do not seem to be productive enough, it is tempting to think that more education might address any "expertise gap."

Will more or better education help solve the expertise gap? In the case study research with professional HPC developers, we included questions about education to help us gain a better understanding of the context of the developer's work and associated education challenges. The following example is representative of the information we collected.

Don[2] is typical of the HPC developers with whom we talked. His work history provides an example of the challenges associated with solving the expertise gap through education.

Don has a Ph.D. in Geophysics with a special interest in meteorite impact on planetary bodies. Because of his background in impact studies, he was recruited onto the Condor team at a Mission Partner Laboratory whose legacy code modeled impact [5].

The code that Don began to learn has existed for 20 years, is written in Fortran77 and contains well over 100,000 lines. Over the years this code has developed a large user base: more than 300 licenses with about 1000 users. Its users perceive this code as valuable, and Don was hired to maintain the code: fixing bugs reported by users and upgrading it when necessary.

Surprisingly, Don joined the code team with no knowledge of Fortran or MPI. He reports that it took him eight years to learn his job. Of course it did not take him eight years to learn Fortran; that took about eight months, which he called his "on-the-job training." The reason for his long learning curve is the complexity of the code itself. The code Don works on has had many authors over the years. It has also been ported to new machines several times during its lifetime use. Not all authors documented their work, and

[2] "Don" and other names that appear in case study descriptions are pseudonyms, used for confidentiality and security.

many legacy features in the code were left intact each time the code was ported.

Don eventually became the lead developer for this code. Although Don now feels confident with the code, he admitted that, after fifteen years working with the code, he still does not understand what all the code does.

Don's long learning period appears typical for HPC development. For example, Sarkar et. al. also reported a 10+ year hands-on learning curve to develop such expertise [12].

Formal education in languages such as Fortran or parallelizing tools such as MPI or OpenMP would not have helped in Don's case. He quickly mastered the skills in these areas. Instead the bulk of his learning was centered on acquiring experience with the legacy code. It was the complexity of this large code that demanded the extra time to master.

The learning curve requirements of the HPC developer working with complex code casts doubt on any long-term strategy of relying on education to solve the problem. Even if we could provide appropriate education, the timeframe for such a solution is unrealistic. The hope that the expertise bottleneck might be solved through education is not a long-term answer.

4.4 Scale up with more people?

If more education alone will not solve the expertise gap perhaps building teams of skilled people can cut down on the time to solution. In another example from our case studies we learned that the answer is "maybe."

The Hawk team filled out surveys and participated in a group interview as part of our case study research [4]. Asok was a founding member of the team, having been recruited as a graduate student. His expertise in fluid dynamics was important to a new project challenged to model a hi-tech plastic to be used in airplanes and armored vehicles, whose parts would be fabricated using large molds. However, it is difficult to fill the molds correctly, without air pockets. Asok was asked to model the mold filling as an alternative to expensive experimentation. Although Asok was knowledgeable about the science, he was not an HPC programmer, so he was teamed with a Fortran expert.

Asok understood the fluid dynamics, the programmer understood Fortran, but they did not understand each other. Four years later the Fortran code was unsuccessful, and the programmer quit.

A new programmer was assigned to work with Asok. Mitch was more fluent in C++ and suggested abandoning the existing code. It was a bold move, but the two decided to begin all over again with C++. They worked together to find a working solution to the modeling, this time creating a successful serial code. Unfortunately neither was expert in scaling. Their manager, Jean, was an expert in optimizing code; she stepped in to scale the code, working closely with Mitch.

Meanwhile the project sponsor started to become impatient, and a manager was still needed to handle expectations. So the project management was taken on by another person. His main role was to run interference for the team, keeping the sponsor happy. He also negotiated time to run the code on the large machine.

It took an additional three years to develop a successful parallel code in C++ but the team effort paid off. The new C++ code was delivered to the sponsor.

Asok attributed the success to two factors: his new programming partner and the coding knowledge he gained during the four years he spent attempting to write the code in Fortran. The C++ programmer, Mitch, attributed success to Asok's expertise in fluid dynamics and Asok's willingness to teach Mitch about it. Both agree that the project would not have succeeded if their manager had not stepped in to scale the code for a highly parallel machine.

This case study illustrates the potential for overcoming the expertise gap with appropriately configured teams. In less than seven years the Hawk team demonstrated that they could develop a working code in less time than a lone developer. But it is a cautionary tale. The team was successful only when they had the appropriate mix of knowledge represented in four areas:

- Science
- Programming
- Scaling / Optimizing
- Management

The success of a team strategy for overcoming the expertise gap relies on a conscious division of labor with a specific mix of domain knowledge. In the Hawk case, the first effort at assembling a team did not work. Throwing more people at the problem without consideration for the team skill mix led to four years of frustration. It was only when the mix of skills was balanced between science, programming, scaling, and management did the effort move forward.

It is also clear that Asok still had to acquire a working knowledge of programming in order to find a solution. He admits that the four years working with the Fortran programmer enabled him to work successfully with the C++ programmer with whom he was subsequently partnered. The programmer needed to gain a working knowledge in Asok's area of expertise as well.

This example suggests two "best practices" that can help a team to succeed.

1. The team needs the right mix of skills: science, programming, scaling/optimizing and management.
2. It is also important that each team member has a basic working knowledge of the other skill sets on the team so that they can communicate effectively.

But assembling a team of appropriate experts is not a complete solution. Of the four skill sets, scaling/optimizing remains a scarce resource, and if Jean had not possessed those skills the project might have stalled. By putting a team together that addresses the various expertise needs, the team approach can bridged the expertise gap. However this is temporary. As systems get even larger and more complex, this key expertise will become relatively more scarce, and teams will be unable to scale. For example adding more programmers to Asok's team would not have helped solve the problem any faster. In fact, more team members would have added a team / human complexity, without fundamentally addressing the "expertise gap."

5. DISCUSSION

A combination of qualitative and quantitative research methods allowed us to develop a more complete understanding of the issues faced in the HPC community than might have been possible with any single method. Using a case study approach allowed us to deeply explore the work of individuals and teams of HPC code developers. Quantitative method provided the breadth of data to validate our findings on a larger scale.

From our research we conclude that complexity is at the core of HPC bottlenecks, and it is a system level problem.

Previously proposed solutions such as "more education" or large team approaches are short-term interventions that will not be viable unless we address the root cause of the "expertise gap" – increasing complexity [6]. Very few individuals have the complete set of skills (science, programming, scaling and management) necessary to exploit fully these complex machines. Educating individual developers in all four skills requires both a gifted individual and many, many years.

Assembling a team of experts is an alternative that has short-term potential. However, these teams have a limited ability to scale. Doubling the number of scientists or programmers adds another layer of complexity to the team. As machines get bigger and more complex the pool of experts who have the ability to deal with the increasing level of complexity will continue to narrow. Unless we address the system level cause of the expertise gap, increasing complexity, we will not solve the problem.

6. CONCLUSIONS

Productivity improvement on the scale sought by the HPCS program must address scale and complexity by requiring fundamental transformations in the way HPC applications are developed and maintained [1][7][11][13]. In the end we believe that bottlenecks will be resolved with appropriate application of automation, abstraction and associated tools. Abstraction, including languages may solve the science domain expertise gap by reducing the programming complexity and allowing scientists to reason in the problem domain. Opportunity here now revolves around grounding practices in specific problem domains, automating that which can be automated, and abstracting away the most challenging aspects of the machine (parallelization).

7. ACKNOWLEDGMENTS

We are grateful to all of our HPCS program colleagues at Sun Microsystems, especially Eugene Loh, Michael Ball, and Victoria Livschitz. We also thank Doug Post, Richard Kendall, Jeremy Kepner, and many others in the HPCS community for their helpful discussions and comments.

This material is based upon work supported by DARPA under Contract No.NBCH3039002.

8. REFERENCES

[1] Ahalt, S. C, and Kelley, K. L., Blue-Collar Computing: HPC for the Rest of Us. *ClusterWorld, 2*, 11(Nov. 2004).

[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] Kepner, J, Personal Communication, 2005.

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

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

[6] Levesque, J., Have We Succeeded Because of Complex HPC Software or In Spite of It? Times N Systems, Inc., (August 17, 2001). <http://www.etnus.com/Company/press/press_release.php?file=hpc>.

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

[8] Nakamura, T., University of Maryland, Personal Communication, 2005.

[9] Post, D. E. and Kendall, R. P., Software Project Management and Quality Engineering Practices for Complex, Coupled Multi-Physics, Massively Parallel Computational Simulations: Lessons Learned from ASCI, *International Journal of High Performance Computing Applications: Special Issue on HPC Productivity*, J. Kepner (ed.), **18**(4), Winter 2004.

[10] Post, D. E. Kendall, R. P., and Whitney, E. M., 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.

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

[12] Sarkar, F., Williams, C, and Ebcioglu, K, Application Development Productivity Challenges for High-End Computing, *First Workshop on Productivity and Performance in High-End Computing* (P-PHEC), Madrid Spain. February 14, 2004.

[13] Squires, S, Tichy, W. G., and 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.

[14] Yu, R. K., *Case Study Research: Design and Methods* SAGE Publications, 2002.