HPC Needs a Tool Strategy

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

D.E. Post
Los Alamos National Laboratory
P.O. Box 1663, MS E526
Los Alamos, NM 87544
1-505-665-7680
post@ieee.org

Mary E. Zosel
Lawrence Livermore National Laboratory
P.O. Box 808
Livermore, CA 94551
1-925-422-4002
mzosel@llnl.gov

ABSTRACT

The High Productivity Computing Systems (HPCS) program seeks a tenfold productivity increase in High Performance Computing (HPC). A change of this magnitude in software development and maintenance demands a transformation similar to other great leaps in industrial productivity. By analogy, this requires a dramatic change to the "infrastructure" and to the way software developers use it. Software tools such as compilers, libraries, debuggers and analyzers constitute an essential part of the HPC infrastructure, without which codes cannot be efficiently developed nor production runs accomplished.

The underappreciated "HPC software infrastructure" is not up to the task and is becoming less so in the face of increasing scale, complexity, and mission importance. Infrastructure dependencies are seen as significant risks to success, and significant productivity gains remain unrealized. Support models for this infrastructure are not aligned with its strategic value.

To achieve the potential of the software infrastructure, both for stability and for productivity breakthroughs, a dedicated, long-term, client-focused support structure must be established. Goals for tools in the infrastructure would include ubiquity, portability, and longevity commensurate with the projects they support, typically decades. The strategic value of such an infrastructure necessarily transcends individual projects, laboratories, and organizations.

Categories and Subject Descriptors

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

Keywords

High Performance Computing, Software Development Tools, Software Productivity.

SE-HPCS'05, May 15, 2005, St. Louis, Missouri, USA.

1. INTRODUCTION

The computing power of high performance computing (HPC) has continued to grow exponentially. A major portion of this growth has been achieved through massive parallelization with the result that machine architectures have become highly complex. Memory access speed has not kept pace with processor speed, and memory has been fragmented so that much of the memory access is across complex and slow networks. The difficulty of developing programs for high performance computers has grown with computer power. To take full advantage of these more powerful computers, applications developers are constructing larger and more complex codes that include increasingly more effects. The level of difficulty for code developers has thus increased both because of the scale of the applications and the complexity of the platforms. The computational milieu is daunting and is becoming more so.

The result is that the productivity of software developers in the HPC community has not kept pace with the performance of their computing systems, nor is it up to the increasing demands being placed on the HPC community to develop and apply increasingly powerful applications to problems of strategic importance. This reduced relative productivity, and the growing necessity for verification and validation represents a "looming crisis in computational science" [11].

The sheer scale of current supercomputers, combined with the extraordinary parallelism needed to exploit them fully, is outpacing development skills and tools. The lifetime of many HPC codes, for which extensive optimization is considered essential, typically spans several decades and many machine generations [10]. Developing and optimizing applications for a specific platform is not feasible. Recognizing this problem, the Defense Advanced Research Projects Agency (DARPA) is funding the High *Productivity* Computing Systems (HPCS) program to "create new generations of high end programming environments, software tools, architectures, and hardware components" viewed from the perspective of overall productivity rather than unrealistic benchmarks [4].

The role of the software infrastructure in this, including compilers, languages, libraries, debuggers, analyzers and any other software that supports developers, is often underappreciated. Every code depends vitally, throughout its lifetime, on some of this infrastructure. Furthermore, significant increases in developer productivity will involve fundamental transformations in this infrastructure.

The current outlook for tools meeting the challenge is bleak. Although there are brilliant and highly productive exceptions, "the low state of the art in software tool development has been effected by the lack of cross platform tools, by the understandable self-interests of hardware manufacturers, and the lack of a broad enough market to attract and sustain independent developers who can keep pace with changes in technology" [6].

Meeting the emerging HPC challenge, to sustain as well as dramatically improve productivity, demands a more effective software infrastructure than is present today. Building that infrastructure requires a new paradigm for its funding, development, and support.

Section 2 of this paper shows how productivity depends on software infrastructure much more than is appreciated, and that significant productivity increases must be found here. Section 3 contrasts this observation with the current state of affairs in the HPC community, where some of the trends are negative, and tools are all too often seen more as a risk than as a lever. A vision for what software might make the difference is presented in Section 4. Section 5 reviews the discouraging prospects of such software arising out of the current funding and support models in HPC. Section 6 discusses prospects for a new model, and Section 7 follows with a discussion of the impact this would have.

2. TOOLS AND PRODUCTIVITY

This section looks at the central role tools play in the productivity of software development.

Tools are the infrastructure. For the purpose of this paper, "tools" means any software that supports software developers. This includes tools in standard development environments: languages and compilers, libraries, linkers, loaders, editors, debuggers, test frameworks, code repositories, configuration management tools, and more. HPC environments are characterized by tools dedicated to performance and parallelism, including parallelized versions of the above, as well as:

- Problem set-up tools;
- Tools and libraries to support massive parallel data storage and retrieval;
- Compilers and parallel programming models;
- Domain- and task-specific languages and frameworks;
- Libraries for mathematics, I/O, and visualization;
- Parallel debuggers, performance and memory analyzers and profilers;
- Tools for resource management and runtime scheduling;
- Tools for visualization, data mining, data analysis and comparison (V&V), test coverage analysis; and
- Tools for production run scheduling and runtime configuration and logs.

Tools are vital. A software project depends on the software infrastructure, without which code development and production runs cease to be viable. The platforms are more complex and the applications are more ambitious than in the past. The long term support, evolution, and porting of a code necessarily requires the support, evolution, and porting of the tools as well.

Tools are the pathway to productivity. It is not merely important for tools in the HPC community to get better; it is essential. There is fundamentally no other way to achieve the huge productivity improvement that is the mission of the HPCS program. We can come at this observation from two directions.

In the first commonsense approach we can say that a huge increase in developer productivity only happens by eliminating some of their work. Developers are already highly skilled, and the fundamental nature of the job is unlikely to change. Eliminating work means automating some things they do now, and for that we look to tools.

The same observation can be viewed through economics. Productivity comes from the division of labor (Adam Smith); in a more modern view it comes from the division of knowledge (across time and space). In this perspective, the essential mechanism for any kind of increased productivity is the transfer of human skills and knowledge into the "capital infrastructure," which in HPC corresponds to the tools described above. Each embodies knowledge that was originally developed by human practitioners. When knowledge is embedded in a tool, it frees the practitioner from the need to master that particular knowledge, it changes the skills needed, and it opens the way for development of new knowledge. This transfer of knowledge is the engine of productivity growth [2].

3. TOOLS IN HPC TODAY

Although the field of HPC can claim credit for many outstanding achievements, the community sees tools as a "problem" [1][6][14]. The tools aren't keeping up with the challenges posed by the growth in the complexity of platform architectures and the complexity and scale of the applications. There is common agreement that tools are important and that it is important that they get better. Some tools improve, but by many assessments the overall picture is not improving and may be getting worse. Software developers aren't as productive as they know they could be, and there appears to be no substantial change on the horizon. The "tools problem" has many facets.

Tools are hard to learn. The complexity of software tools specialized for HPC is a natural consequence of the complexity of the job. This essential fact is exacerbated by the nature of the tools market, in which many tools originate as research products or the results of Open Source projects staffed by donated time. The maturity that is characteristic of tools in the larger commercial market is seldom present. Proficiency with such tools demands a significant investment in time and effort. Indeed, just getting them to work at all is often challenging.

Tools don't scale. Implementation techniques for tools often fail when systems grow into clusters of thousands of processors. Operations taking a fraction of a second expand to minutes or hours, memory requirements expand, and visual layouts fail. On some systems, lightweight kernels successfully minimize the overhead from the OS software, but the reduced system functionality often removes features needed by the runtime tools.

Tools differ across platforms. Tool, especially those provided by hardware vendors, often differ dramatically from one another. Otherwise portable codes are often confounded by nonessential differences in libraries, configurations, and job management environments; such differences contribute significantly to the cost of migrating codes and people across platforms.

Tools are slow to appear on new platforms. A traditional engine of progress in HPC has been the advancement of fundamental

technologies, combined with new, often experimental architectures designed to exploit the technologies. In practice, however, effective tools—other than compilers—often lag the arrival of a new machine by several years, a circumstance that significantly diminishes the value of both.

Tool support is inadequate. In a field driven by innovation it is not surprising that some of the best HPC tools appear at the bleeding edge of research, precisely where support is unlikely to meet the standards required for mission critical software. Even platform vendors sometimes curtail support for tools on previous generations of machines.

Tools are hard to test. Access to at-scale target platforms is often inadequate for tool development, testing, and training, even for platform vendors, who lose access when systems are installed into secure sites.

Tool availability is uncertain. Research-based tools disappear due to events beyond the control of clients, for example funding interruptions and graduations. Tools produced by independent software vendors are vulnerable to business failure or acquisition by companies with different goals. Such disruptions can have large, unplanned impacts on the schedule and costs of projects that depend on them. In general, the HPC community has—except possibly for the DOE ASCI program—been reluctant to devote significant resources to develop and support these tools.

Tools are often too expensive for universities. Given the relatively small market for HPC tools, vendors must charge significant licensing fees to stay in business. These fees are often too large for widespread use in universities, so the new generation of HPC application developers often has no chance to become skilled in the use or to contribute to the development of tools**.**

Tools are seen as a risk to project success. The unfortunate consequence of these issues is that project planners, faced with crucial dependence on software tools as well as long project lifetimes, often see tools more as a risk to project success than as an essential advantage. This is a hidden cost that impacts budgets and mission success prospects. For an application that has a twenty-year life cycle, application developers are reluctant to invest in the use of tools that will likely have a much shorter lifetime than the code or in tools that will be viable for just a few platforms.

4. A PRODUCTIVITY INFRASTRUCTURE

What kind of software infrastructure would make the difference in HPC, not only sustaining current development models but also creating a pathway for extraordinary productivity improvement?

A common tool set. What's missing is a common set of tools built with the intention of supporting the kind of scalable, stable, highly parallel programming required in the HPC community.

Functionally complete. The tools must include those now considered standard in the general programming community, for example compilers, debuggers, code editors, application frameworks, source code repositories, refactoring, testing frameworks, code coverage, etc. It must also include extensions and adaptations of the standard tools that address scalability and massive parallelism. Finally, it must include those tools, traditionally developed within the HPC community, that are specialized for the domain, for example specialized libraries, solvers, parallel tracing, visualization, and performance analysis tools.

Multiplatform. The tools must run on every platform of interest and must appear and function as uniformly as possible.

Specialized. The tool set must be designed around the need to "plug in" pieces specialized for particular platforms. Ideally these would be adapters that make standard functionality available, but some might be uniquely specialized for a single platform.

Widely available. There must be no significant barriers to the use of the tools everywhere in the community, including educational institutions.

Enduring. Planners must be able to rely on tools for the lifetime of projects lasting decades, which means that the tools must evolve along with applications in the face of ongoing change: new systems, new system software releases, new programming models, higher scale, new customer requirements, and new customer usage patterns. Programmers, particularly students, must have confidence that skills will remain valuable.

Inviting of research. The tools must encourage research in tool technologies with an open architecture, a wealth of common functionality, and a community of interested users.

Financially viable. Tool development must be financially sustainable. It must be recognized as part of the cost of using HPC to address problems of strategic importance. It has the same importance as platforms and applications code development and production. "A chain is only as strong as its weakest link" [15]. Yet, as noted above, the tools must be widely available.

5. WHO WILL BUILD IT?

The HPC community has many participants with some stake in tools, and yet no sustainable and successful productivity infrastructure has arisen. This section looks at how existing motivations and business models make such an outcome extremely unlikely.

Hardware vendors focus on bleeding edge hardware technologies and architectures. They often provide platform-specific tools, especially those needed to exploit unique architectures, but they seldom see it in their interests to produce cross platform tools. Indeed, they often see exotic tools as "differentiators" that offer competitive advantage. Furthermore, their business models don't always include a commitment to their platforms for the very long haul.

SMP Platforms are increasingly being constructed from commodity components by local institutions (e.g. Beowulf clusters). They consist of commodity processors and memory linked together with commodity networks. The operating system is typically open source LINUX that is locally modified and maintained. Compilers, debuggers, etc. are either locally developed or obtained from third party systems. No one outside the local institution (and sometimes inside the institution) is committed to support the platform and its software infrastructure. The result is often that no one is responsible for ensuring that the machine and the software infrastructure work smoothly. Conflicts between different sub-systems often lead to "finger-pointing" and long delays in getting hardware and software infrastructure problems resolved. This produces a less expensive machine, but with hidden costs. It places increased burdens on the applications developers, and overall productivity can be substantially reduced compared to a well-supported machine and software infrastructure.

Researchers innovate, build prototypes, and share them freely. Some of the best HPC software has come from universities, but researchers have little motivation to extend support for these tools over time and to any platforms not locally in use. Research funding for such activities is not available. Furthermore the increasing size and complexity of HPC programming raises the bar for effective research contributions under the best of circumstances. All too often, this research does not lead to a well-supported, long-lived product that can be used by others with confidence.

Independent tool companies start up, exploit research, aim for cross-platform support, and struggle for financial survival in the face of customers who often don't wish (or aren't funded) to purchase industrial strength tools. When companies fail, their technology can disappear through legal concerns and lack of support. When companies succeed, their technology can disappear through acquisition and lack of support. A recent example is the demise of the popular KAI C++ compiler.

Customers, especially those with long life cycle codes (10 to 30 years), consider developing their own tools to mitigate risk. This lowers productivity by diverting resources away from applications development, and risks the reinvention of wheels in place of ongoing innovation.

Voluntary collaborations are sometimes formed to create standards, initiate and coordinate efforts (typically Open Source), and support cooperation and sharing. These can be quite important and effective, but it is difficult to create or maintain critical mass when support and funding is essentially being diverted from customer projects. For example the influential Parallel Tools Consortium [8] ceased to function in 2003.

Heroes are individuals or small groups who provide service to the community that rises above and beyond the job and mission in which they are nominally engaged. Heroes can often be found behind important resources in the HPC community, but this is not a scalable model, especially when planning horizons span decades.

None of these stakeholders have the primary mission, resources, or longevity to produce what's needed: to create a complete productivity infrastructure for the HPC community.

6. A NEW APPROACH

HPC stakeholders understand that progress depends on coordinated, collaborative effort, and there have been numerous attempts made in this regard. The late Parallel Tools Consortium (pTools) [8] was an important example, as was the National Compiler Infrastructure (NCI) project [7] and many smaller collaborations. Some collaborations are voluntary (pTools, for example), some are encouraged by common funding sources such as the U.S. Department of Energy (for example Open Speedshop [13]), and some have been at least partially funded, for example the Common Component Architecture (CCA) Forum [3]. None of the past and current efforts, however, even the successful ones, approach the goals set forth in the previous sections.

A new model is needed, and a new structure to support it must be put in place. This is not a research problem, but a strategic effort that must deal with the pragmatic issues: funding, business models, intellectual property rights, and procurement possibilities. Existing models, for example the Internet Engineering Task Force [5], should be reviewed for relevance.

There are two prerequisites. First, there must be wide recognition that the software infrastructure is as crucial to mission success as platforms and applications. Second, there must be funding and direction given from the same level where strategic missions are defined and funded, a level where the pragmatic issues mentioned above can be addressed effectively.

It would be premature to describe the necessary support structure in detail, but it is possible to enumerate its essential characteristics.

- **Mission**: dedicated to creating and supporting the Software Productivity Infrastructure described in earlier sections, in collaboration with all major stakeholders. This includes languages, compilers, libraries, solvers, application frameworks, parallel tools, and any other software that supports HPC.
- **Longevity**: constituted and funded for the long haul, i.e. for a lifetime commensurate with 20 to 30-year strategic missions and beyond.
- **Pragmatism**: focused on industrial strength support for HPC software development, with an emphasis on broadening the audience through increased usability.
- **Research**: engaged in research that furthers the integrity and leverage of the infrastructure, for example by developing "common ground" architectures that permit "plug-in" adaptation to specific platforms. Efforts currently under way in this area, such as Open Speedshop [13], should be supported and sustainable business models put in place.
- **Flexibility**: equipped with development and support models appropriate to the full spectrum of stakeholders. For example, university research would be coordinated and supported; open source projects would be hosted; independent companies would be supported contingent on code escrow or other arrangements needed to address vital customer concerns; vendors would be encouraged (via standardized procurement clauses, perhaps starting with existing guidelines [9]) to port the common tool set onto new platforms in a timely manner.

These goals bear an interesting resemblance to the "software reuse" challenges that have been faced in other areas of computing, typically at the scale of individual organizations. For example, Rosenbaum and du Castel report that the most important factors for success in their environment were [12]:

1. Having a team dedicated to the infrastructure;
2. Extensive interaction between team and "customers;" and
3. Senior management that fully supports its existence and promotes its usage.

Factors 1 and 2 appear in the "mission" requirement above. By analogy, factor 3 reinforces the conclusion that crucial to the success will come from the agencies that fund the missions in this community, and that they must be actively engaged in supporting the new model and uses of its software.

7. IMPACT ON THE HPC COMMUNITY

The product of such an effort would be the eventual creation of a "capital infrastructure" of software development for HPC, which in turn would serve as the pathway to the tenfold productivity improvements chartered by the HPCS program [4]. The impact

would be as dramatic as the Internet and World Wide Web have been.

Predictability. The software infrastructure would support the long planning horizons that are uniquely important in a community where strategically crucial projects span decades and machine generations. Project planners will be able to depend on an infrastructure that will (at very least) not diminish during the lifetime of a project. Which is to say, tools will been seen purely as a productivity lever, not as a risk.

Ubiquity. A common infrastructure supported on every platform would eliminate effort now wasted dealing with *inessential* differences among platforms, both operationally (programming and job management becomes simpler) and with respect to training and skills (which become more portable and thus more valuable). These would make HPC more widely accessible which would in turn accrue further benefits [1].

Progress. Stewards of intellectual capital in the HPC domain (knowledge, skills, codes) would ensure monotonic progress, not only by preservation of past achievements, but also by providing a focus (as well as coordination, funding, and access to at-scale platforms) for ongoing innovation. Equally important is the emergence of a common infrastructure into which innovative pieces may be plugged in and evaluated.

8. CONCLUSIONS

The HPC community has undervalued software development tools, taken here to include languages, libraries, frameworks, solvers, and many other traditional tools. At the project level, planning for maintenance and evolution often neglects the crucial dependency on supporting tools. More broadly, the HPC community is in great need of a widely available, fully functional, portable tool set. This would be the "software productivity infrastructure:" a stabilizing force for the current state of affairs and the crucial lever for achieving dramatic productivity improvement.

Although there is some awareness of this, and despite a number of past efforts, the creation of a common software infrastructure for HPC programming has not been achieved. Meeting DARPA's challenge for a 10x increase in productivity will require that a structure be put in place that is dedicated to creating such an infrastructure, is committed for the long haul, and which meets the other success criteria proposed in this paper.

9. ACKNOWLEDGMENTS

We would like to thank all of our HPCS colleagues at Sun Microsystems, especially Lawrence Votta, Susan Squires, Rob Van der Wijngaart, Jan Strachovsky, Eugene Loh, and Michael Ball. Further, we would like to thank Brett Kettering and the LANL parallel tools group, Robert Ballance of Sandia National Labs, and many others in the HPC community for their helpful discussions and comments.

This material is based upon work supported by DARPA under Contract No.NBCH3039002. A portion of this work was performed under the auspices of the U.S. Department of Energy by by University of California Los Alamos National Laboratory under contract No. W-7405-ENG-36. LA-UR-05-1592. A portion of this work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. UCRL-CONF-210246.

10. REFERENCES

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

[2] Baetjer, H. Jr., *Software as Capital: An Economic Perspective On Software Engineering*. IEEE Computer Society, 1998.

[3] Common Component Architecture (CCA) Forum. <http://www.cca-forum.org/>

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

[5] Internet Engineering Task Force (IETF). <http://www.ietf.org/>.

[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] National Compiler Infrastructure Project (NCI). <http://suif.stanford.edu/suif/NCI/>.

[8] Parallel Tools Consortium (pTools). <http://www.ptools.org>.

[9] Pancake, C. M., and McDonald C., eds. Task Force on Requirements for HPC Software and Tools: Guidelines for Specifying HPC Software. Northwest Alliance for Computational Science and Engineering (March 31, 1999) <http://www.nacse.org/projects/HPCreqts/>.

[10] Post, D. E. Kendall, R. P., and Whitney, E. M., Case Study of the Falcon Code Project. Submitted to *Second International Workshop on Software Engineering for High Performance Computing System Applications*, 2005.

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

[12] Rosenbaum, S, and du Castel, B., Managing Software Reuse - An Experience Report. *Proceedings 1995 International Conference on Software Engineering*, 105-111

[13] Silicon Graphics, Inc. Open Speedshop. <http://www.sgi.com/company_info/newsroom/press_releases/2004/october/speedshop.html>.

[14] 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).

[15] Leslie Stephen in The Cornhill Magazine, 1868.