Effective Software Engineering Approaches to Resolving Scientific Computing’s Productivity Gridlock

Stuart Faulk, Ph.D.
Computer and Information Science
University of Oregon
faulk@cs.uoregon.edu

Supercomputer designers traditionally focus on low-level hardware performance criteria such as CPU cycle speed, disk bandwidth, and memory latency. The High-Performance Computing (HPC) community has more recently begun to realize that escalating hardware performance is, by itself, contributing less and less to real productivity—the ability to develop and deploy high-performance scientific computing applications at acceptable time and cost.

In prior work we examined the source and nature of productivity problems in large-scale scientific programming ([1], [2], and [3]). The results suggest that the dominant barriers to productivity improvement are in the software processes. The scientific programming community has evolved its own characteristic software development approach that, for historical reasons, focuses on code efficiency, scientific worth, and machine utilization. Our workflow studies have shown that this approach creates bottlenecks that constrain developers’ ability to improve real (end-to-end) productivity. Further, these bottlenecks are inherent in the approach, particularly in its reliance on having multidisciplinary experts (the science, the code, and the hardware) hand-craft and hand-tune the code.

The software engineering community has developed a number of technologies, strategies, and methods that have the potential to remove these productivity bottlenecks. However, these technologies have typically not been adapted to the rigors and constraints of massively parallel scientific computing. Thus, it is currently unclear which aspects of software engineering should be used, exactly where they should be applied, or how the technologies will need to be adapted to address the problems specific to HPC. For example, the software community has developed a wide range of software development process models. However, the models that currently receive the most attention in the literature (i.e., agile methods like Scrum) are based on assumptions that are inconsistent with the developmental realities of the HPC community.

Nonetheless, the software engineering community has much to offer scientific computing. While software engineering has not yet addressed HPC’s specific bottlenecks, it has addressed a wide range of similar productivity problems in many problem domains, and across all aspects of development. Thus, while the current focus on agile processes may not help, there exists a broad capability in modeling software development processes, process measurement, process improvement, and other results that can be brought to bear to model, measure and improve processes
used in scientific computing. Some of the most promising areas of collaboration include [2]:

- **Automation** – increased automation of labor-intensive tasks, particularly parallelization, data allocation, and latency management.

- **Abstraction** – the expertise bottleneck arises directly from the need to maintain concurrent expertise in multiple distinct disciplines. This bottleneck can be reduced to the extent scientists are able to think and code in problem-level abstractions (as opposed to machine-specific codes). While appropriate abstractions are lacking, the software engineering community has a deep background in creating such abstractions in a variety of problem domains.

- **Measurement** – many of the traditional HPC metrics (e.g., machine utilization, throughput, latency, etc.) do not measure the qualities critical to end-to-end productivity. Process and other productivity metrics developed in the software engineering community, along with continuous process improvement, will be key to improving HPC processes.

My own work has more specifically focused on the development and reuse of critical problem abstractions in a way that allows the reuse of substantial parts of design, code, and other development artifacts. In non-HPC systems such contextual reuse has been shown to improve productivity (fairly routinely) by a factor of 3X to 8X. A wide range of processes, methods and models has been developed for the engineering of such software product lines (e.g., [4]). Our experience with other difficult problems (e.g., small, hard-real-time avionics) suggests that many of these results might be repurposed for application in the scientific programming context. However, significant work must be done to develop HPC-appropriate abstractions, demonstrate their efficacy on problems of real scientific value, and collaborate with the scientific computing community to achieve useful technology transference.

**References**


