Software expertise in high-performance computing

Pre-proposal
Grant Writers' Workshop

Lorin Hochstein
lorin@cse.unl.edu

Target agency: NSF

Overview and objectives

High-performance computers (HPC) are increasingly being used to solve complex scientific problems such as simulating climate change or oil reservoirs. Yet, these powerful supercomputers remain out of reach for many computational scientists who lack the expertise to write programs that can effectively use these machines. Large differences in programmer skill were identified in very early software engineering studies, which suggested that performance differences across programmers could be as large as 28:1. More recent studies have shown differences between experts and novices in how they solve isolated software-related tasks such as comprehending an unfamiliar program or locating software bugs. However, the methods by which expert HPC programmers acquire their skills and how they apply these skills in the context of computational science projects remain unexamined. Given that only a small number of scientists possess the skills necessary to use supercomputers, there is a critical need to understand how such skills are successfully acquired and applied. In the absence of such understanding, U.S. supercomputing resources will continue to be accessible to only a small fraction of the scientific community.

Our *long-term goal* is to contribute to improving the productivity of software developers through increased understanding of how developers behave and interact with software technologies. *Our objective in this application* is to identify how expert software developers in HPC acquire and apply their skills. Our central hypothesis is that skill acquisition follows the Dreyfus model and that information about skills can be captured by eliciting narratives about software development activities and by observing these activities in context. This hypothesis is based on the successful application of this research strategy in the study of expertise in clinical nursing. Our rationale is that understanding skill acquisition and application is a necessary step to increasing the pool of computational scientists that can effectively use HPC systems. We are well prepared to conduct this research because of the PI's published research into programmer productivity in HPC, and because we have established strong working relationships with computational scientists. We propose two primary objectives:

Objective #1: Delineate the practical knowledge embedded in expert HPC software development. Our *working hypothesis* is that practical knowledge breaks down into a discrete set of software-related tasks (e.g. algorithm design, tuning, debugging), and that programmer expertise will vary by task.

Objective #2: Describe how software skills are acquired in HPC. Our *working hypothesis* is that skill acquisition occurs through apprenticeship and follows the Dreyfus model.

This work is *innovative* because it relies on context-dependent narratives to capture expert behavior as opposed to context-free strategies. At the completion of this project, it is expected that we will have identified the different categories of skills that experts possess and a characterization of how those skills are applied, in the form of narratives that provide detailed, in-context descriptions. We will also expect to have identified how the HPC programmer progresses from novice to expert. We anticipate that these results will be transformed into educational materials that will be used to train novice HPC programmers, which will increase the productivity of U.S. scientific research by enlarging the talent pool. We also anticipate that existing HPC practitioners will be able to make direct use of the narratives to improve their own skills.

Expected Significance

Significance. Large variability in performance across programmers was discovered early on in software engineering (Sackman et al. 1968). While the original study was criticized for overestimating the reported 28:1 difference in performance (Dickey 1981; Prechelt 1999), the importance of highly skilled software developers is widely maintained in the practitioner community (Glass 2006; Davis 1995; McBreen 2002). The large variation in programmer ability is a particular problem for high-performance computing. Writing computational science software to run efficiently on high-performance machines remains a very difficult task (HECRTF 2004, Graham 2004, PITAC 2005), and there is a perceived “expertise gap” (Squires 2006) in the HPC community that restricts the number of scientists who can write computer models to run on high-performance systems. Outside of HPC, differences between novice and expert programmer behavior have been observed in laboratory settings for tasks such as requirements analysis (Batra & Davis 1992; Jeffries et al. 1981), software design (Adelson & Soloway 1985; Davies 1991) program comprehension (Adelson 1984; Koubek & Salvendy 1991), and testing/debugging (Law 1998, Weiser & Shertz 1983, Teasley et al. 1994, Krems 1995). However, researchers have hypothesized that results from studying isolated tasks may not generalize to larger software projects because of the differences in incentive on the part of the participant. (Thelin 2005). In addition, no such studies have focused on the HPC domain. Our contribution is expected to be a detailed understanding of the different skills of expert HPC programmers, how these skills are acquired and how they are applied in the context of developing scientific codes. *This contribution is significant, therefore, because it is expected to provide the knowledge needed to better train aspiring computational scientists to work productively on HPC systems*. Once we are able to transfer the skills of expert HPC programmers to others, we can increase the number of computational scientists who can take advantage of large-scale parallel systems. Thus, important advances in scientific disciplines that employ large-scale computer simulations could be expected. It is also expected that what is learned will apply equally to skill transfer in other software domains such as embedded systems or web-based applications. Furthermore, better understanding of tool requirements to better support HPC software development can be anticipated.