Empirical study design in the area of High-Performance Computing (HPC)

Forrest Shull[1], Jeffrey Carver[2], Lorin Hochstein[3], Victor Basili[1,3]

[1]*Fraunhofer Center Maryland*
{fshull, basili}@fc-md.umd.edu

[2]*Mississippi State University*
carver@cse.msstate.edu

[3]*Univ. of Maryland, College Park*
{lorin, basili}@cs.umd.edu

Abstract

The development of High-Performance Computing (HPC) programs is crucial to progress in many fields of scientific endeavor. We have run initial studies of the productivity of HPC developers and of techniques for improving that productivity, which have not previously been the subject of significant study. Because of key differences between development for HPC and for more conventional software engineering applications, this work has required the tailoring of experimental designs and protocols.

A major contribution of our work is to begin to quantify the code development process in a specialized area that has previously not been extensively studied. Specifically, we present an analysis of the domain of High-Performance Computing for the aspects that would impact experimental design; show how those aspects are reflected in experimental design for this specific area; and demonstrate how we are using such experimental designs to build up a body of knowledge specific to the domain. Results to date build confidence in our approach by showing that there are no significant differences across studies comparing subjects with similar experience tackling similar problems, while there are significant differences in performance and effort among the different parallel models applied.

Keywords: Developer productivity, High-Performance Computing, parallel programming, empirical study

1. Introduction

Within the field of software engineering, optimizing developer productivity has long been recognized as one of the primary motivating goals of research and practice. A significant portion of the literature consists of descriptions or evaluations of software development methods, practices, tools, etc., that are aimed at reducing the effort required to develop software, removing or avoiding defects and the rework they cause, quickly developing systems that respond better to customer requirements – in short, that are aimed at somehow improving productivity.

Empirical study has been increasingly seen as a necessary approach for understanding software developer productivity and the factors that increase or reduce it. In a survey of software engineering publications, Zelkowitz and Wallace showed that the number of papers published with no empirical validation has been dropping from 36% in 1985 to 29% in 1990 to 19% in 1995 [16]. A number of different empirical approaches have been developed and applied, each suited to different research hypotheses and types of environments, ranging from observational studies to case studies to controlled experiments.

However, fields outside of software engineering represent a very different paradigm, and empirical studies of developer productivity for specialized computing systems have not been as common. In this paper, we present initial results from novel work in the application of empirical studies of development productivity in the area of High Performance Computing (HPC). The outputs of this work, presented in this paper, include:

1. A set of lessons learned describing the similarities and differences between "typical" software development and the creation of specialized HPC codes (Section 2).
2. A set of guidelines and a template for empirical studies in this area (Section 3).
3. An overview of key quantitative results produced to date (Section 4).

These results are indicative not only of the unique points of HPC but may also serve as a blueprint for adapting empirical study to other specialized areas of computer science.

2. Overview of HPC software development

An HPC program (usually referred to as a "code") is a program which is written to run on parallel computers. The objective is to decompose the computation among multiple processors so that independent parts of the solution can be computed at the same time, thus returning a final answer in a fraction of the time required if all computation had to be done sequentially on a single processor. An HPC code will contain instructions pertaining not only to logic and control flow, but also to managing communication among the various processors.

Different programming models are available which determine how this communication among processors is managed in the code. The most widely used programming model is known as message-passing, since it generally achieves the best performance on a wide range of HPC machines. Most message-passing programs are written using the MPI library (see section 3.2). However, since it is believed to require a significant amount of effort to implement programs in MPI, there is an interest in alternative models.

While software engineering studies have been applied to a wide variety of different development contexts, HPC development is a highly specialized niche with several important disparities to the usual range of assumptions concerning developer behavior.

Who are the developers? The development of HPC codes is crucial to progress in many fields of scientific endeavor (e.g. climate science, astrophysics, molecular biology). Since development of such codes first and foremost requires an expertise in the scientific domain in which a solution is being sought, those domain experts are likely to be novice HPC programmers, at least when they are beginning their careers.

Truly effective HPC programmers are rare because HPC code development requires individuals who are both experts in the domain and in the HPC architecture on which the code is being developed. These problems will only increase in the future as tougher problems are attacked and more powerful (yet likely more difficult to program) HPC machines are created. For this reason, we decided to emphasize in our initial studies the question of how novice programmers learn to effectively develop HPC codes. This research question would have important implications for expanding the base of effective HPC programmers.

Both because of this focus on learning HPC programming, and to help debug our experimental protocols in a more controlled setting, we focused first on classroom studies. Still, as classes require access to specialized machines and are only taught at the graduate level, the pool of such courses and the number of subjects in each course were both much smaller than is typically expected of software engineering classes. This also had important implications for our study design (namely, results need to be abstracted from across multiple classes rather than from within a single class).

How important is hardware? In a "typical" software engineering study, the computing hardware is likely to merit only a brief mention along with the operating system (e.g. programming was done on a Unix workstation or a Windows PC), since modern compilers isolate programmers from the details of a particular machine architecture. Most software development studies place a higher importance on the programming language used, since this is assumed to have a much higher correlation with programmers' strategies for decomposing the problem and hence with overall development effectiveness.

In contrast, HPC machines vary considerably in their architectures [13]. Machine architectures have substantial effects on the effectiveness of different algorithms and programming models, since the way in which the processors are connected in the hardware needs to be reflected directly in the software which is written to take advantage of them, and may be more or less suitable to the ways in which various problems can be decomposed. To date, there are no HPC compilers that can shield the programmers from such details and still achieve acceptable performance. Thus, effective HPC development requires appropriate matches between the type of problem, the programming approach, and the underlying hardware architecture. In empirical study, then, reporting as much detail as possible about the hardware platform, and controlling for this source of variation, becomes extremely important.

What is optimized during development? As in other types of software development, the usual goal of developing HPC codes is to arrive at the solution of a problem with minimal effort and time. Thus, an important metric for evaluating various approaches to code development in HPC is "time to solution," encompassing both the effort required to understand and develop a solution as well as the amount of computer time it takes to execute that solution and arrive at an answer. Unlike most other software applications, the time to execute the implemented solution can be quite high. Hence the time to execute the code is a non-trivial part of the above equation.

The activities required to actually develop HPC code differ significantly from what would be expected on a development project outside of the HPC domain. While only a small subset of software development projects require significant effort to be spent on tweaking the code to improve performance, in HPC code development a large percentage of the effort is always expected to be spent on optimizing the code. This optimization is necessary to take advantage of the underlying hardware architecture and hence improve performance. Many HPC codes are designed for very specific problems. Those codes are typically run only once, with a significant running time. In this case, the time taken to do these optimizations is expected to pay off with a measurable decrease in execution time.

Within the HPC community, metrics and even predictive models have already been developed for measuring the final code performance, under various constraints (e.g. [12, 5]). However, little empirical work has been done to date to study the human effort required in the development of those solutions. There has been for example no investigation of what percentage of the total development effort is spent on optimizing code as opposed to developing it in the first place, and how much the effort spent on optimization improves the final execution time. However, development decisions are being made based on beliefs about how much improvement is likely to be obtained from effort spent on optimizing code performance.

How is quality assured? Although the issue of cost-effective software testing and quality assurance is an entire research area unto itself, for the vast majority of applications the general testing strategy is understood: a number of representative test cases are selected and the software product is executed under those conditions, so that the actual behavior can be compared to the expected. A discrepancy between the two points to a defect in the software.

The issue is much more complicated for many HPC codes, which are developed to help expand the understanding in a given field of science, usually by simulating complex physical phenomena (such as climate change) where experimental validation is impractical or impossible. That is, these codes are developed to tackle exactly those problems for which there is no “expected” answer. The code is developed to produce a result that would have been impossible to ascertain without it, not to automate an already well-understood task. For this reason, extensive checking of the code and domain models occurs during development.

Because of these differences, traditional approaches for measuring and improving productivity cannot be applied without modifications in the HPC domain. We therefore faced some difficulty when we were approached by DARPA to design and conduct studies of HPC development productivity. Despite years of running software productivity studies, we had little intuition about how to adapt our knowledge to run studies in this context.

To obtain the understanding necessary to run effective studies, we proceeded iteratively. We began with very informal, subjective data collection, the results of which we encoded as heuristics. These heuristics were used to suggest an initial set of feasible and testable hypotheses, which were then tested via an initial set of pilot studies. The results of the pilot studies were used both to evolve the hypotheses and further evolve our experimental protocols.

3. Implications for HPC study design

The above lessons point to a large number of variables that can affect the productivity of developing HPC solutions. This would imply the need for a large number of studies to compare and contrast them appropriately, especially when the number of problem domains that could be addressed are taken into consideration.

For these reasons, our solution has been to define the parameters of a family of related experiments that would help us to understand how to best leverage the results of the studies that we can collect. Such a family also addresses the need discussed in Section 2 to combine subjects from multiple classroom environments to build up a sufficiently large sample size. Applying an over-arching framework to the problem space also allows us to collect data opportunistically (depending on the specific circumstances and interests of the educators with whom we are working) while still providing overall guidance and direction (e.g. we can see for what combinations of variables we have so far achieved little coverage).

Since the studies within the family have to be comparable, in order for each to contribute to the overall body of knowledge, we first define a common set of dependent and independent variables, then show experimental designs that permit the capture of the required information. The overall family of studies is summarized in Table 1.

3.1. Dependent variables

We define the following as the variables of interest for describing the *outcome* of an HPC code development:

- A primary measure of the quality of the HPC solution is the *speedup* achieved, that is, the relative execution time of a program running on a multi-processor system compared to a uniprocessor system. The speedup measured for a given HPC implementation can be assessed in comparison to the number of processors on which the code is run: Running on 8 processors, a hypothetical ideal HPC version of a functionally equivalent serial program can achieve a speedup of up to 8 times. In practice, HPC implementations are not expected to exhibit this idealized behavior, but it does provide a useful way of understanding the level of improvement that is achieved. (In this paper, all values reported for speedup were measured when the application was run on 8 parallel processors, as this was the largest number of processors that was feasible for use in our classroom environments.)

- A primary measure of cost is the amount of *effort* required to develop the final solution, for a given problem and given approach. The effort undertaken to develop a serial solution includes the following activities: thinking/planning the solution, coding/debugging, and testing. In comparison, the effort undertaken to develop a parallel solution includes all of the above as well as tuning the parallel code (i.e. improving performance through optimizing the parallel instructions). It is necessary to break effort measures down at this level of detail in order to understand the tradeoffs with the code performance achieved; it is not sufficient to know which model required more effort, but rather what degree of better performance was achieved for the extra effort. Our mechanisms (both automated and manual) for measuring subjects' effort are described in much more detail in a separate paper [1]. (In this paper, HPC development was always done with subjects producing a serial version first and then developing a parallel version from that. Whether this is an effective way for parallel development to be done, or whether developers should start from scratch and think from the beginning in a parallel computing paradigm, is a question of current debate in the HPC community. But, measures of effort thus have to include the effort for both versions.)
- In comparing various parallel programming models, an important description of the solution is captured by the *code expansion factor* of the final code. In order to take full advantage of the parallel processors, HPC codes can be expected to include many more lines of code (LOC) than serial solutions. For example, certain parallel programming models require a significant amount of code to deal with communication across different nodes. The expansion factor is the ratio of LOC in an HPC solution to LOC in a baseline serial solution of the same problem.
- Another metric that assists in comparing the effects of different HPC programming approaches is the *cost per LOC* of the final codes. This value is another measure (in person-hours) of the relative cost of producing code in each of the HPC approaches. It is relevant for an investigation of HPC development since LOC has been used by the HPC community in the past as a proxy for effort [e.g. 3, 14]. It is important for our empirical work to validate whether LOC does indeed meaningfully correlate to effort across various models.

3.2. Independent variables

We identify the following variables which can influence the outcome (costs and benefits) of an HPC code development, and hence which needed to be controlled or monitored in productivity studies:

Problem type: Because the scientific applications implemented on HPC machines vary so widely, it is important to investigate whether the results observed vary from one problem to the next or, more interestingly, from one class of problem to another.

As examples, the assignments used in our studies to date can be grouped into four distinct problem types. Problems categorized as "nearest neighbor" are those in which the problem space can be subdivided into regions, with each processor assigned to a region. Two processors only need to communicate with each other if their regions are adjacent in order to obtain a solution [12]. In contrast, "embarrassingly parallel" problems rely on computations that can easily be broken into mostly independent components, requiring almost no communication among nodes. The analysis discussed in this paper makes use of data concerning both of those problem types.

There are other problem types possible; for example, "broadcast" problems, which require that one processor communicates with all others, or "all-to-all" problems, in which each node must communicate with all others. As we accumulate more data, we will be able to investigate whether there are characteristic patterns observable within and among the various problem types.

The data sets described in this paper were generated for two specific applications: The "game of life" posits a two-dimensional grid where every cell can be either on or off; over a series of turns the grid evolves with the behavior of each cell determined according to a set of rules about the state of the cells surrounding it. The Buffon-Laplace needle problem posits a grid of evenly spaced parallel lines, where each square on the grid is of size a by b. If a needle of length l is dropped onto this grid, it will land on at least one grid line with probability $(2*l*(a+b)-l^2)/(\text{pi}*a*b)$. Running Monte Carlo simulations of various numbers of pin drops is then used to approximate pi.

The names of the other specific applications are included just to give a flavor of the range of problems being addressed. However, a full treatment of these problem types is outside the scope of this paper.

HPC parallel programming model: A second key factor that will affect the outcome of a development effort is the *parallel programming model*, which

	Serial	MPI	OpenMP	StarP	XMT
Nearest-Neighbor Type Problems					
GoL (Game of Life)	C3A3	C3A3 C0A1; C1A1	C3A3		
GoR (Grid of Resistors)	C2A2	C2A2	C2A2	C2A2	
LE (Laplace's Eq.)		C2A3		C2A3	
SWIM			C0A2		
Sharks & Fishes	C6A2	C6A2	C6A2		
Broadcast Type Problems					
LU Decomposition			C4A1		
Parallel Matvec				C3A4	
Embarrassingly Parallel Type Problems					
Buffon-Laplace Needle		C2A1; C3A1	C2A1; C3A1	C2A1; C3A1	
(Miscellaneous Problem Types)					
Parallel Sorting		C3A2	C3A2	C3A2	
Array Compaction					C5A1
Randomized Selection					C5A2
Matrix power computation		C8A1	C8A1		
Parallel sums	C6A1	C6A1	C6A1		
Sparse matrix multiply					C7A1
Dense matrix multiply	C6A1	C6A1	C6A1		

Table 1: Matrix describing the problem space of HPC studies being run. Each study is indicated with a label CxAy, identifying the participating class (C) and the assignment (A). Results discussed in this paper are grey-shaded.

describes how components of the code running on different processors communicate with one another.

The two instances of parallel programming models about which we have collected the most data are MPI and OpenMP. These approaches are mature and are used in industry. MPI [10] is a portable, scalable programming approach that can be used on both distributed-memory multicomputers and shared-memory multiprocessors. All communication is explicitly managed by the programmer, who must call "send" and "receive" functions to communicate messages between processes. MPI is implemented as a library and requires no special compiler support [5].

OpenMP [11] is a shared-memory programming model. OpenMP takes advantage of the ability to directly access shared memory throughout the system along with fast shared-memory locks to improve on the complexity of the MPI approach. OpenMP is meant to be useful for quickly parallelizing existing code and for developing a broad set of new applications. It is commonly used to achieve "loop-level parallelism": the programmer annotates loops with special compiler directives, and the compiler distributes the iterations of the loop across multiple processors. OpenMP is implemented as an extension of C and Fortran and requires support from the compiler [4].

Some of the researchers whose classes we observed are also using new HPC approaches which they have developed. These include StarP, a parallel extension to the Matlab environment [8], and Explicit Multi-Threading (XMT), a conceptual framework with language extensions to C that implement parallel random-access algorithms [15].

We also use the value "serial" to represent the decision to use no parallel programming model at all (i.e. to create a non-HPC baseline).

Developer experience: In the studies reported here, the majority of subjects were novices in the area of HPC development (not surprisingly, as the studies were run in a classroom environment). Such a student population is highly relevant to our work, since one of our key research areas is how people *learn* to program HPC codes effectively. Also, as mentioned in Section 2, many of the likely users of HPC computers in government and industrial contexts, who are researchers and domain experts in other, unrelated fields, are likely to be classified as novice users of many HPC approaches themselves.

3.3. A framework for a family of studies

Our current framework is illustrated in Table 1. It helps organize studies according to two major variables: The problem being solved and the parallel programming model that was applied to create the solution. These make natural axes for this matrix since the problem type is of primary importance for our work: We are hoping to uncover basic phenomena in HPC studies by understanding which types of problems seem to respond best to similar solutions, and why. Since the programming model is one of the basic choice points for a code developer, we hope to be able to provide useful decision support on this issue by understanding under what contexts the various approaches lead to the most effective solutions.

By labeling each dataset according to the class and assignment, we show which data came from the same subjects and hence where dependencies are. The assignment number conveys some information about the order in which assignments were done; since our subjects were in a learning environment, it may be the case that they were more effective at doing later assignments than earlier ones.

One piece of information missing from this view is the machine type/platform on which the solutions were developed. We do not have enough data yet across different platforms in order to contrast results under different circumstances. Eventually, we hope to have enough data within each cell of the matrix that we can evaluate whether there are differences among solutions developed using the same approach but on different platforms.

3.4. A design for HPC productivity studies

The following designs were created to be used in a number of different environments, with minimal further tailoring. Through our connection with HPC classroom environments we have discovered that there are two approaches to a graduate class curriculum in HPC development:

Introduction to multiple HPC approaches, one at a time. In this approach to structuring the class, multiple programming models are introduced one at a time and discussed in-depth. For the sake of simplicity, let us assume two different models are to be introduced and refer to these as mod1 and mod2. Subjects will be asked to apply each model to a given problem, in order to gain experience with its use. A study design that fits into such a class is described in Table 2. It requires randomly dividing the class into two groups. This design also requires two different problems, prob1 and prob2, which will be implemented by the groups in varying order.

Step	Group 1	Group 2
1	Instruction in HPC model1 (mod1)	
2	Treatment1 (optional): serial version of prob1	Treatment1 (optional): serial version of prob2
3	Treatment2: mod1 applied to prob1	Treatment2: mod1 applied to prob2
4	Instruction in HPC model 2 (mod2)	
5	Treatment3 (optional): serial version of prob2	Treatment3 (optional): serial version of prob1
6	Treatment4: mod2 applied to prob2	Treatment4: mod2 applied to prob1

Table 2: Design variant 1 for HPC classroom studies.

By switching documents between the two groups, we allow the techniques to be taught in sequence but allow the interaction between the problems and the HPC approaches to be understood. If desired, this design can accommodate each group implementing the given problems in serial code, or can allow developers to work only in HPC versions. This design does however still suffer from the threat to validity concerning maturation, i.e. results may be biased in favor of mod2 since it is not taught until later in the semester, and some skill development in general HPC programming may have occurred.

The design can of course be easily extended to include additional HPC models that might be taught over the course of the semester, if the class is divided into an additional group, so that the same algorithm for varying documents could be followed.

Introduction to multiple HPC models at once, with a large assignment to compare and contrast. An alternate approach to the class is to introduce students to multiple HPC models at the same time. A single problem to be solved is then taught and students are expected to solve the problem using all of the HPC models taught in order to compare and contrast them.

As in the previous case, this design allows the use of multiple HPC models, generically referred to as mod1 and mod2, applied to different problems to be solved, prob1 and prob2. This design is illustrated in Table 3.

Step	Group 1	Group 2
1	Treatment1 (optional): serial version of prob1	
2	Treatment2: mod1 applied to prob1	Treatment2: mod2 applied to prob1
3	Treatment3: mod2 applied to prob1	Treatment3: mod1 applied to prob1
4	Treatment4 (optional): serial version of prob2	
5	Treatment5: mod2 applied to prob2	Treatment5: mod1 applied to prob2
6	Treatment6: mod1 applied to prob2	Treatment6: mod2 applied to prob2

Table 3: Design variant 2 for HPC classroom studies.

This design removes the threat of the maturation effect; all HPC models are applied both early and late in the semester. It does require students to conform to an order in which they apply the various HPC models to the given problem; we are solving this by giving out each portion of the assignment individually and setting a due date for each treatment. Once we have finished piloting this design in the current semester's classes we will have some feedback as to whether this way of imposing an ordering on the assignment is comfortable for student subjects.

4. Example results to date

To verify that our framework is a meaningful way of combining families of studies, we need to show that data collected *within* a cell is more highly correlated than data *across* cells. Said another way, we are concerned with verifying that our identified variables of the computing problem and the HPC approach are in fact important factors for predicting the outcome of a development effort.

To carry out this verification, and as examples of the kinds of analyses that our framework was intended to support, we compare data for the same problem but different models, and for the same model but different problems. For the statistical tests run in this paper, we used an alpha-value of 0.05 in all cases. (It should be noted that, since our experimental design has been evolving over time, some of these studies were conducted using an experimental design that is similar to the one in Table 2 but with only one group of subjects, that is, with all subjects applying the same activities at the same time. New data is being collected using the design as it is found in Table 2.)

Studies included in this analysis were:

- **C0A1.** This data was collected in Fall 2003, from a graduate-level course with 16 students. Subjects were asked to implement the Game of Life program in C on a cluster of PCs, first using a serial solution and then parallelizing the solution with MPI. In this class, 56% of subjects had no experience in HPC development; 33% had previous class experience; and 11% had some industrial experience.
- **C1A1.** This data was from a replication of the C0A1 assignment in a different graduate-level course at the same university in Spring 2004. 10 subjects participated. In this class, 63% of subjects had no experience in HPC development; and 37% had previous class experience.
- **C2A1.** This data was collected in Spring 2004, from a graduate-level course with 26 students. Subjects were asked to implement the Game of Life and Buffon-Laplace problems in C on a cluster of PCs, first as a serial and then as an MPI and OpenMP version. 100% of the students had no previous experience with HPC development.
- **C3A{1,3}.** This data was collected in Spring 2004, from a graduate-level course with 20 students. Subjects were asked to implement the Game of Life and Buffon-Laplace problems in C on a cluster of PCs, first as a serial and then as an MPI and OpenMP version. In this class, 50% of subjects had no experience in HPC development while 50% had previous class experience.

4.1. Analyzing differences among problem types

One research question of interest is whether different problems have different, characteristic levels of improvement that can be achieved by using an HPC approach to implementing the solution. To validate our research framework, we would also like to test the related question of whether, regardless of the class implementing the solution, the codes developed for the same problem and using the same programming model exhibit similar behaviors.

To address this, we analyzed the four independent datasets summarized in Table 4.

Data set	Problem	Speedup
C0A1	Game of life	mean 4.86, sd 2.4, n=14
C1A1	Game of life	mean 4.81, sd 1.7, n=5
C2A1	Buff.-Lap.	mean 2.01, sd 1.0, n=8
C3A1	Buff.-Lap.	mean 3.73, sd 2.1, n=8

Table 4: Mean, standard deviation, and number of subjects for computing speedup achieved on different problems. All values were recorded for solutions using the MPI parallel programming model in C.

We first test for similarities within the same problem type. We used a t-test to test the hypothesis that the mean speedup achieved by C0A1 for the game of life is different from the mean achieved by C1A1. With a p-value of 0.96, we cannot conclude that the results for the two classes were different. Similarly, a test for differences between C2A1 and C3A1 on the Buffon-Laplace needle problem yields a p-value of 0.07, so we cannot conclude that these two classes were different. Thus there are no significant differences in the performance achieved for classes for either of our two problems.

Comparing across problems, we therefore combine datasets and run a t-test on the hypothesis that the mean speedup achieved for the game of life (regardless of class) is significantly different than the speedup achieved for Buffon-Laplace (regardless of class). The resulting p-value, 0.006, shows that there is in fact a statistically significant difference in the speedup achieved using the MPI model for these two problems.

4.2. Analyzing differences among parallel programming models

A second question of importance to HPC research is, what are the strengths and weaknesses of the various parallel programming models? That is, what are the tradeoffs between cost and benefits of the available parallel programming models?

Two datasets had sufficient number of subjects to enable a meaningful within-subjects comparison of effects: C3A3, applying the MPI and OpenMP models to the Game of Life problem, and C3A1, applying the same two models to the Buffon-Laplace needle problem. We investigated whether the two models required a different amount of effort for implementing the solution and thus a different final cost. (Unfortunately, since students were instructed to execute their Game of Life solution on 8 processors and their Buffon-Laplace solution on 2 processors, we could not meaningfully compare the speedup achieved on the two models.) Recall that the HPC effort here represents the effort needed to produce first a serial version and from that develop a parallel version. The effort/LOC metric is thus computed as total effort over total LOC for both serial and parallel versions.

Since the same individuals implemented versions of the same problem using different models, we rely on the *paired* t-test to test the differences in mean values for statistical significance.

Regarding the *effort* required to implement a solution using the two parallel programming models, the data show an interesting pattern, as shown in Table 5. For the Buffon-Laplace problem, OpenMP required significantly greater effort than MPI (p=0.02). However, for the Game of Life problem the relationship was also significant (p=0.01) but in the opposite direction. When the total implementation effort is normalized by the number of LOC written, as in Table 6, the mean value of this metric is also significantly different from one approach to the other (p=0.01 for the Buffon-Laplace problem and p=0.02 for the game of life).

Data set	Prob.	Prog. model	Effort (person-hrs)
C3A1	BL	MPI	mean 1.4, sd 1.1, n=20
C3A1	BL	OpenMP	mean 2.5, sd 2.0, n=20
C3A3	GoL	MPI	mean 9.1, sd 4.3, n=15
C3A3	GoL	OpenMP	mean 4.3, sd 3.6, n=15

Table 5: Mean, standard deviation, and number of subjects for computing the effort required for implementing the solution. Data is from two different programming models applied to the Buffon-Laplace needle (BL) or the Game of Life (GoL) problems.

Data set	Prob.	Prog. model	Effort (person minutes/LOC)
C3A1	BL	MPI	mean 0.02, sd 0.02, n=15
C3A1	BL	OpenMP	mean 0.05, sd 0.04, n=15
C3A3	GoL	MPI	mean 0.04, sd 0.03, n=13
C3A3	GoL	OpenMP	mean 0.03, sd 0.01, n=13

Table 6: Mean, standard deviation, and number of subjects for computing effort per line of code. Data is from two different programming models applied to the Buffon-Laplace needle (BL) or the Game of Life (GoL) problems.

As a result of the above analysis, we conclude that the number of LOC in a code is not a good proxy for developer effort. This has important implications for attempts to build predictive models, or at least define easy-to-measure proxies for the phenomena of interest.

4.3. Threats to validity

The fact that these studies were run, not only in a classroom environment, but across several classroom environments, means that there are threats to the validity of our conclusions that should be kept in mind when interpreting the results. There is a threat to external validity in that our studies involved two-week assignments run on a small number of processors, while "real" HPC programs may take years of development and run on hundreds of thousands of processors.

One possible threat to internal validity resulting from the classroom environment is that we have to deal with problems of incomplete data, e.g. subjects not filling out their logs, or subjects creating implementing serial versions on non-instrumented machines or not submitting the serial version of their code. We have not observed any systematic bias in which subjects have omitted data. However, we do report the total number of students in each class and the size of the subset that was able to be included in each analysis.

We have already characterized the major context variables that may vary from one classroom environment to another, such as the programming language being used or the specific application assigned to the students. We argue that the experience level of subjects, as described in Section 4, is similar enough to be compared across studies: Nearly all of the students had either no experience or only classroom experience in parallel programming.

We used performance data that was reported by the subjects. This represents an internal threat to validity since the students may not have reported their data truthfully. They may have also made errors while recording execution time data (e.g. recording execution time when the machine is heavily loaded).

All effort data was collected through instrumenting the compiler and batch scheduler on the development machines. Our analyses show that there are some discrepancies between these measures and the self-reported logs that subjects also kept, although the results point to the instrumented data as being the more accurate source. Our analysis and process for reconciling these differences has been described in some detail elsewhere [7].

Finally, there is the possibility that students experienced a learning effect, i.e. that they might have become simply better HPC programmers over the course of multiple assignments, regardless of the HPC approach being used.

5. Future work

In order to facilitate the running of more studies that can contribute to this analysis framework, we have put effort into designing web-based lab packages that organize all the resources necessary for educators to implement the studies in their own courses. By making a library of predetermined choices available for each field of the template, we hope to show educators a range of choices that can meet their classroom conditions while maintaining the ability to compare between studies.

We have made available the instrumentation along with installation instructions for setting up automated data collection. Packets are available for batch processors and compilers on most HPC machines. We are currently working with HackyStat [9] and Eclipse [6] to create plugins for most common editors, which will eventually be available for downloading as well.

Our work has shown that the variables we have identified as of primary importance (namely, the type of problem implemented and the parallel programming model used) do have a measurable impact on the results of HPC development. As we collect more data and populate the matrix, we will build more sophisticated models of cause and effect, for example by investigating the role played by developer experience and whether its effect varies for different models. We will also explore additional research questions; for example, an important topic is whether a developer's *workflow* (the overall strategy used at the individual level in order to solve the problem) has an effect on development success. One of our long-term goals in this work is to investigate which types of developer behaviors have the best correlation with improved outcomes, e.g. when developing a parallel solution to a problem, is it always a good idea to create a serial version of the solution first, or is it better to begin programming directly on the parallel architecture? Such questions can be addressed once we have the solid baseline of data for addressing variations in problems and HPC programming models, which we are building in this work.

6. Conclusions

In this paper we have described a research program aimed at conducting empirical studies of a specialized type of software development. We have so far been successful at forging collaborations among researchers in software engineering as well as high-performance computing, and in adapting an empirical approach to the study of how HPC codes are engineered effectively. We have reported our high level approach for evolving our designs over time as we discover more about unique constraints in this area, which we hypothesize will be useful for similar efforts of tailoring empirical study for other specialized fields. We have reported what we have discovered about the particular constraints for empirical studies of HPC codes, and showed an experimental framework that takes the constraints into account.

Furthermore, we have begun collecting baseline data about how novices perform on various HPC applications, which can be useful for both researchers and educators who would like to replicate those experiences. The data help build confidence in our approach by showing that there are no significant differences across classes with similar experience tackling similar problems, while there are significant differences in performance and effort for the different parallel models applied.

Clearly, however, more work needs to be done to explore the influence of context variables and collect data from more widely disparate application development. Such work will be guided by the matrix that describes the problem space (Table 1) to show where we are lacking coverage. Future studies will be run to generate the large and diverse data sets, covering a majority of cells in the matrix, which will be required to address those questions in the most general case. We are also working with industrial and government HPC developers, to determine in what ways experience level affects development practices and results.

7. Acknowledgements

This research was supported in part by Department of Energy contracts DE-FG02-04ER25633 and DE-CFC02-01ER25489, and NASA Ames Research Center grant NNA04CD05G to the University of Maryland. We wish to acknowledge the contributions of the various faculty members and their students who have participated in the various experiments we have run over the past 2 years. This includes Alan Edelman at MIT, John Gilbert at the University of California Santa Barbara, Mary Hall at the University of Southern California, Jeffrey Hollingsworth at the University of Maryland, Alan Snavely at the University of California San Diego, Alan Sussman at the University of Maryland, and Uzi Vishkin at the University of Maryland.

8. References

[1] S. Asgari, V. R. Basili, J. Carver, L. Hochstein, J. K. Hollingsworth, F. Shull, and M. V. Zelkowitz, "Challenges in Measuring HPCS Learner Productivity in an Age of Ubiquitous Computing: The HPCS Program", In *Proc. ICSE Workshop on High Productivity Computing*, Edinburgh, Scotland, May 2004, pp. 27-31.

[2] S. Asgari, L. Hochstein, V. Basili, J. Carver, J. Hollingsworth, F. Shull, and M. V. Zelkowitz, “Generating Testable Hypotheses from Tacit Knowledge for High Productivity Computing,” In *Proc. ICSE Workshop on High Productivity Computing*, St. Louis, USA, May 2005.

[3] F. Cantonnet, Y. Yao, M. Zahran, and T. El-Ghazawi. Productivity Analysis of the UPC Language. IPDPS 2004 PMEO workshop. 2004.

[4] L. Dagum and R. Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming," *IEEE Comp. Science & Engineering*, 5(1), 1998, pp. 46-55.

[5] J. J. Dongarra, S. W. Otto, M. Snir, and D. Walker, "A message passing standard for MPP and workstations," *CACM*, 39(7), 1996, pp. 84-90.

[6] Eclipse.org. http://www.eclipse.org/

[7] L. Hochstein, V. R. Basili, M. Zelkowitz, J. Hollingsworth, and J. Carver. "Combining self-reported and automatic data to improve effort measurement." Accepted at the European Software Engineering Conference / ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2005 (ESEC-FSE'05).

[8] P. Husbands and C. Isbell, “MATLAB*P: A tool for interactive supercomputing,” *Proc. SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[9] P. M. Johnson. Hackystat system. http://csdl.ics.hawaii.edu/Research/Hackystat/.

[10] Message Passing Interface Forum, http://www-unix.mcs.anl.gov/mpi/mpi-standard/mpi-report-2.0-sf/mpi2-report.htm

[11] OpenMP C and C++ Application Interface, OpenMP Architecture Review Board Version 2.0, March 2002. http://www.openmp.org/drupal/mp-documents /cspec20.pdf

[12] J. T. Schwartz, “Ultracomputers,” *ACM Trans. Program. Lang. Syst.*, 2(4): 484-521, 1980.

[13] A.J. van der Steen, and J. J. Dongarra. Overview of Recent Supercomputers. http://www.top500.org/ORSC/2004.

[14] S. P. VanderWiel, D. Nathanson, and D. J. Lija. Complexity and performance in parallel programming languages. 2nd International Workshop on High Level Programming. 1997.

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

[16] M. V. Zelkowitz and D. Wallace, Experimental models for validating computer technology, *IEEE Computer* 31, 5 (May, 1998) 23-31.