Potential Influence of Prior Experience in an Undergraduate-Graduate Level HPC Course

Chris Fietkiewicz
Electrical Engineering and Computer Science Department
Case Western Reserve University
Cleveland, OH 44106 USA
001-216-368-8829
chris.fietkiewicz@case.edu

ABSTRACT

A course on high performance computing (HPC) at Case Western Reserve University included students with a range of technical and academic experience. We consider these experiential differences with regard to student performance and perceptions. The course relied heavily on C programming and multithreading, but one third of the students had no prior experience with these techniques. Academic experience also varied, as the class included 3rd and 4th year undergraduates, master's students, PhD students, and a non-degree student. Results indicate that student performance did not depend on technical experience. However, average overall performance was slightly higher for graduate students. Additionally, we report on students' perceptions of the course and the assigned work.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education - Computer Science Education, Curriculum

General Terms

Education.

Keywords

Undergraduate, graduate, education, high performance computing.

1. INTRODUCTION

Graduate level courses at universities are typically open to undergraduate students with significantly less academic experience. Additionally, such courses can attract students from multiple disciplines and departments due to a shared interest in a particular topic. The potential for a high diversity in backgrounds and experience levels poses challenges for instructors. Previously, we investigated the potential influence of technical and academic experience levels for a single homework assignment in a class on high performance computing (HPC) at Case Western Reserve University in Cleveland, Ohio [1]. In that study, it was found that prior experience was not a significant predictor of a student's performance with regard to implementing a successful programming solution. In the present study, we look at the student outcomes for the course as a whole, and we consider how students' backgrounds may influence perceptions of the course.

© 2019 Journal of Computational Science Education
DOI: https://doi.org/10.22369/issn.2153-4136/10/1/15

2. METHODS

The class was taught during the Spring semester of 2018 at Case Western Reserve University in Cleveland, Ohio. The total enrollment was 23 students, including undergraduate, graduate, and non-degree students. The course had been offered twice before, and course evaluation statistics were available to prospective enrollees. At the beginning of the course, survey data was collected to determine whether students had prior experience with C programming and multithreading. Six main HPC techniques were covered in the course and are listed below:

- Batch job processing
- General optimization for sequential programming
- Parallel programming using spawned (forked) processes
- Parallel programming using OpenMP and multithreading
- Parallel programming using OpenACC and GPUs
- Parallel programming using message passing and MPI

All students were graded using the same criteria and rubrics. Assignments consisted of seven programming projects on required topics and a three-week course project that focused on an application of the student's choice. The seven programming assignments were designed to apply the above HPC techniques to four different applications. Assignments generally focused on either introducing an application or comparing different HPC techniques. The four applications covered in the programming assignments are listed below:

- Sorting algorithms (e.g. merge sort)
- Matrix multiplication (iterative and recursive)
- Prime number discovery
- Numerical integration of Laplace's equation

Assignments generally included 3 or more separate problems to be solved. Below is an example of a typical problem statement that requires parallel processes for the discovery of prime numbers:

Count the number of prime numbers up to two different maxima N_1 and N_2. Choose maxima such that the serial-version run time for N_1 is at least 5 seconds and for N_2 is at least 10 seconds. For parallel versions, using 2 and 4 processes respectively, each process should do an approximately equal amount of work (same approximate run time). For parallel versions, report the speedup as a ratio of the serial-version run time to the parallel-version run time. In your report, explain how you equalized the work, and briefly discuss how the speedup compares to the number of processes.

Prior to each assignment, lectures were provided on the requisite material, including discussion of all sample programs. For the example problem statement above, sample C programs were provided to demonstrate the use of the fork() instruction and a serial algorithm for discovering prime numbers (see APPENDIX for sample programs).

Assignments were designed to provide explicit instructions that would be understandable to typical undergraduates. The instructions had stringent reporting requirements that included a thorough explanation of methods, highly detailed timing results, and a careful discussion of results. The primary requirement of the discussion section of each report was a textual observance of any trends in the results and whether the student found the results to be as expected. Students were not required to accurately explain any anomalies.

3. RESULTS

Table 1 shows the distribution of students by level, including subcategories for undergraduate and graduate students. Graduate students include Doctoral and Master's. Undergraduates include juniors (3rd year) and seniors (4th year). Results also include one student who was of non-degree status but had bachelor's degrees in two related fields.

Table 1. Distribution of students by level.

Level	*Number*	*Portion*
Doctoral	4	17%
Master's	6	26%
Senior	10	44%
Junior	2	9%
Non-degree	1	4%
Total	23	100%

In the following analyses, we organized students into three categories: doctoral, master's + non-degree, and undergraduate. The non-degree student is included in the same group as the master's students because their academic backgrounds were equivalent. For the undergraduate category, we combined the seniors and juniors because there were only 2 juniors, and their performances fall within the bounds of the distribution for the seniors.

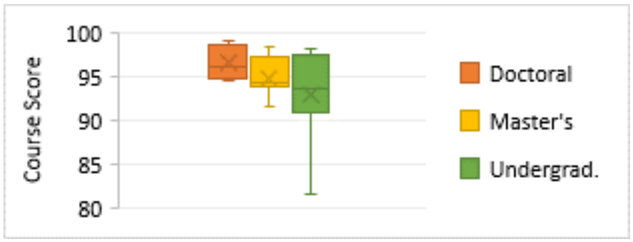

Figure 1. Course scores by academic experience. Left: Doctoral. Middle: Master's + non-degree. Right: undergraduate. In this box-and-whisker plot, horizontal bars indicate quartiles, and the X indicates the mean.

We used the final score for the course (maximum of 100) and compared the three categories of students. The scores for the three categories are analyzed in Figure 1. The mean scores are 96.5 for Doctoral students, 94.9 for Master's students, and 93.7 for undergraduates. It can be seen in Figure 1 that the mean score decreases as the level of academic experience decreases.

All students had significant programming experience, but 35% (*n* = 8) reported having no significant experience with C programming or multithreading. We analyzed the course scores (maximum of 100) based on whether or not students had this prior technical experience. The results are shown in Figure 2.

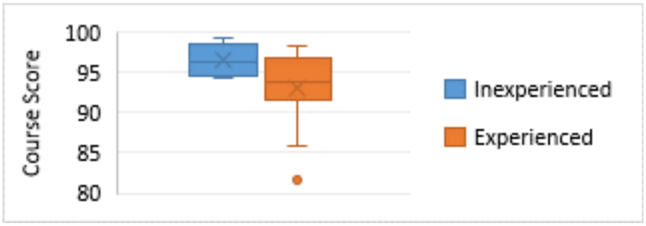

Figure 2. Course scores by technical experience. Left: No prior experience with C programming or multithreading. Right : Prior experience. In this box-and-whisker plot, horizontal bars indicate quartiles, X indicates the mean, and the circle indicates an outlier.

The mean scores are 96.5 for technically inexperienced students and 93.0 for technically experienced students. It can be seen in Figure 2 that the mean score for the inexperienced students was higher than that of the experienced students. These results can be understood by looking at the levels of academic experience within these groups. The graduate students were more likely to lack technical experience, having come from other programs at other institutions. In fact, the inexperienced students were comprised of 87.5% graduate students, while the experienced students were comprised of only 33.3% graduate students. Because graduate students generally had higher scores (see Figure 1), this accounts for the negative correlation with technical experience, indicating that academic experience is more important in predicting success in the course.

We also considered students' perceptions of the course in an effort to characterize the appropriateness of graded work. Anonymous course evaluations were submitted by 11 students. As the evaluations were entirely anonymous, it is not possible to separate them according to academic experience. Overall, students gave the course a rating of 4.09 on a scale of 1 – 5. Students were asked to provide anonymous comments on the assigned work, and all comments were positive in this regard. We provide only one example below that was similar to the other student comments:

"The assignments he gave really helped me understand the content of this course and help me to understand how to implement it to any other algorithm out there. He also tells you what he expects to see in the report for each assignment."

All comments regarding graded work indicated that the problems were relevant and instructions were clear.

4. DISCUSSION

We have presented a course comprised of both graduate and undergraduate students. Because the course required a high degree of technical competence, we expected that technical experience might be an advantage to students and be reflected in student performance. To the contrary, however, we found that academic experience was correlated to performance, and technical experience may have no correlation at all, assuming adequate coverage in class is provided.

Different reasons are possible for the correlation between performance and academic experience. In the most general sense, graduate students may simply be more capable of working with larger projects and report writing, as compared to undergraduates.

Though we did not track requests for help from the instructor, we did perceive that graduate students appeared more likely to seek help and request clarifications regarding the instructions.

In the future, we will consider two changes to our course design to improve relative performances of graduate and undergraduate students. First, we will consider requiring graduate students to do additional project work and reporting, as compared to undergraduates. This is a well known practice, and it is clearly appropriate in our course. A second consideration in the future will be to administer post-assignment surveys that allow students to reflect on their performance and possible influences. Survey results could be used to identify challenges common to undergraduates.

5. ACKNOWLEDGMENTS

This work made use of the High Performance Computing Resource in the Core Facility for Advanced Research Computing at Case Western Reserve University.

6. REFERENCES

[1] Fietkiewicz, C. (*in press*) Student Outcomes in Parallelizing Recursive Matrix Multiply. *Journal of Computational Science Education.*

7. APPENDIX: Sample Code

Figure 3: C program that demonstrates fork instruction:

```
pid_t pid;
/* fork a child process */
pid = fork();
if (pid == 0) { /* child process */
  printf("Child pid = %d\n", pid);
}
else { /* parent process */
  printf("Parent pid = %d\n", pid);
  /* wait for the child to complete */
  pid = wait(NULL);
  printf("Child %d is done.\n", pid);
}
```

Figure 4: C program for discovering prime numbers:

```
int nMax = 100; // Upper limit
int n, d, isPrime;
for (n = 2; n <= nMax; n++) {
  isPrime = 1;
  for (d = 2; d < n; d++){
    if (n % d == 0){
         isPrime = 0;
         break;
    }
  }
  // Print each prime number
  if (isPrime == 1)
    printf("%d ", n);
  }
}
```