Hands-on PDC in Undergraduate Computing Education*

Hala ElAarag and Anas Gamal Aly
Mathematics & Computer Science
Stetson University
DeLand, FL 32723
`{helaarag, agamal}@stetson.edu`

Abstract

Parallel and Distributed Computing (PDC) is a critical yet conceptually challenging area of the undergraduate computer science curriculum. While students often encounter these concepts in theory, few gain exposure to experience in real high-performance computing (HPC) environments. Research shows that when students are engaged in project-based learning they retain knowledge more effectively. They also develop a deeper understanding of concepts taught in the classroom. This paper presents a practical assignment in which students engage directly with the University of Florida's HiPerGator supercomputer to implement and benchmark matrix multiplication using Python and C (via POSIX threads and OpenMP). Students navigate batch scheduling, core allocation, and performance tuning, experiences that are rarely accessible at the undergraduate level. We describe the assignment in detail and provide a three-year evaluation across multiple course offerings, highlighting how structured access to real HPC infrastructure can deepen student understanding of parallelism and multithreading.

1 Introduction

Undergraduate courses in Parallel and Distributed Computing (PDC) rarely grant students hands-on access to high-performance computing (HPC) re-

*Copyright ©2025 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

sources. To bridge this gap, we developed an assignment in which students implement and benchmark multithreaded matrix-multiplication code on the University of Florida's HiPerGator supercomputer [13, 5]. Matrix multiplication is a $\mathcal{O}(n^3)$ operation [12] that underlies many advanced systems—most prominently the Transformer architecture that powers modern large-language models (LLMs) [14]. Although recent research seeks to eliminate computationally-intensive matrix multiplication operations in deep learning [15], conventional LLM pipelines as of 2025 remain dominated by matrix multiplication [9].

CS pedagogy studies show that hands-on, project-based activities increase engagement and long-term retention, especially for abstract topics such as concurrency and parallelism [10, 3]. When students tackle open-ended problems on real hardware—whether breadboards, Raspberry Pis, or HPC clusters — they report greater satisfaction and achieve deeper conceptual understanding [6, 7, 11]. Discovery learning is therefore potentially useful for PDC [1].

Previous efforts have helped students visualize multithreaded behaviour [2]. We extend this work by moving beyond simulation: students write POSIX-threaded and OpenMP implementations, craft batch scripts, submit jobs to HiPerGator, and analyse scalability across languages, thread counts, and cores. The assignment situates algorithmic theory within a real-life HPC workflow, giving students first-hand experience with scheduling policies, queue limits, and performance bottlenecks.

This paper details the structure of the assignment, presents an evaluation of three course offerings (Spring 2022, 2024, 2025), and discusses its pedagogical impact.

2 Related Work

Several works have also explored matrix multiplication as a vehicle for teaching parallel programming. Fietkiewicz demonstrated the educational value of recursive matrix multiplication in a parallel setting and suggested requiring students to perform more detailed efficiency analyses as part of the learning process [8]. More recently, Bober and Bylina investigated teaching parallel programming on students' personal computers using matrix multiplication with MKL, OpenMP, and SYCL libraries, emphasizing performance comparisons across multiple frameworks [4].

Our work extends these efforts in two key ways. First, rather than focusing primarily on framework-level differences, we situate matrix multiplication in a production-grade HPC workflow using the HiPerGator supercomputer. This exposes students to job scheduling, queue management, and resource allocation, experiences that are rarely available in undergraduate curricula. Second, by requiring students to benchmark implementations in both Python and C across multiple threading paradigms, we highlight the contrast between language-level concurrency models and performance bottlenecks. The novelty of our approach lies in combining algorithmic simplicity with the central learning goal of HPC workflow proficiency, thereby deepening student understanding of Parallel and Distributed Computing (PDC) and bridging the gap between this conceptually challenging area of the undergraduate curriculum and real-world applications.

3 Assignment Design

During the assignment design, we intentionally chose matrix multiplication because it is a well-known common algorithm that is computationally intensive [12]. Its algorithmic simplicity is a key advantage, as it allows students to focus on the primary learning objectives: navigating a real High-Performance Computing (HPC) environment to gain a better understanding of parallel programming and multithreading.

The assignment does not focus on the implementation of the algorithm itself. The main goal is for students to understand the impact of multithreading on algorithm performance. This goal is achieved through exposing students to the process of designing experiments, managing resources on a real-world supercomputer via a job scheduler, and analyzing performance variations across different parallelization strategies.

This approach situates theoretical knowledge within a practical, project-based framework, moving students from abstract concepts to tangible skills grounded in the principles of discovery learning [1].

3.1 Learning Objectives

Upon completing this assignment, students are expected to be able to:

- **Implement and Compare** different parallel programming methods (POSIX threads, OpenMP) as well as understand their performance implications.

- **Manage HPC Workflows** by writing SLURM batch scripts, submitting jobs to a scheduler, and monitoring their execution on a remote HiperGator cluster.

- **Analyze Performance Trade-offs** by designing an experiment to measure how execution time is affected by matrix dimensions, programming language choice, thread count, and core allocation.

3.2 Structure

We structured the assignment as an open-ended investigation. Rather than providing a rigid set of instructions, we task students with an open-ended objective to investigate and report on the factors affecting the performance of matrix multiplication. This structure encourages a discovery-based approach where students must formulate their own hypotheses and design experiments to test them.

Students completed the project individually, ensuring that each student gained hands-on experience with both implementation and HPC workflows. However, we encouraged informal peer discussion during lectures, particularly when troubleshooting job scripts or interpreting unexpected performance results.

To help students remain focused on the long-duration aspects of the project, we divided the work in the assignment instructions into discrete milestones. Each stage (implementation, HPC workflow, data analysis and final report) functioned as a mini-project with its own deliverables.

We structured the process as follows:

1. **Baseline Implementation:** Students first implement standard, single-threaded matrix multiplication in Python and C. This establishes a performance baseline.

2. **Parallel Implementation:** Students then develop two parallel versions in C (using POSIX threads and OpenMP) and one in Python (using its threading library).

3. **HPC Immersion:** Students are given access to HiPerGator and its documentation. They learn to write shell scripts to automate their experiments and submit jobs to the SLURM scheduler. This is a critical task that mirrors the typical workflow of researchers and software engineers. They must specify job parameters such as core count, memory allocation, directly engaging with concepts of resource management discussed in lectures.

4. **Data Collection and Analysis:** Students run their code with varying matrix sizes, thread counts, and core allocations. They record results and are required to produce visualizations to support their analysis.

5. **Final Report:** The project culminates in a technical report where students present their findings, interpret their graphs, and explain the observed performance behaviors.

4 Evaluation and Discussion

We evaluated the assignment's effectiveness using two primary methods: direct analysis of student work products (final reports and code) and indirect assessment via anonymous surveys measuring student perceptions. As shown in Table 1, data is aggregated from three semesters: Spring 2022, 2024, and 2025.

4.1 Analysis of Student Submissions

Students submitted final reports that provide clear evidence of student learning. By tasking students to explain their results, we can assess their conceptual understanding. The figures below, taken from a representative student report, are presented here with the kind of analysis we expect students to produce, which demonstrates the learning that occurred.

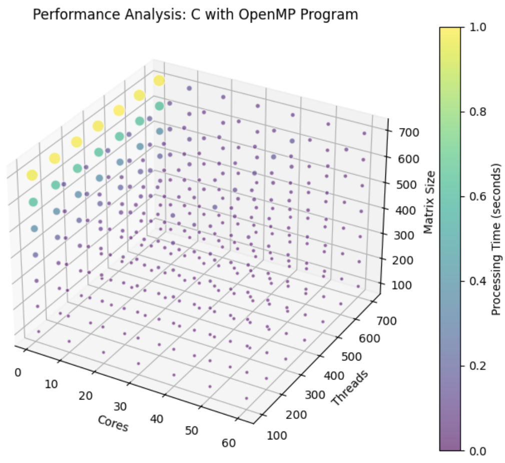

Figure 1: Performance of C with OpenMP.

Figures 1 and 2 show a student's comparison of C-based implementations. In their reports, students identified the C implementations as significantly faster than Python — as expected. They noted that the OpenMP version (Figure 1) offered the best performance with the least programming effort, providing a concrete example of the value of libraries that are optimized for parallel execution.

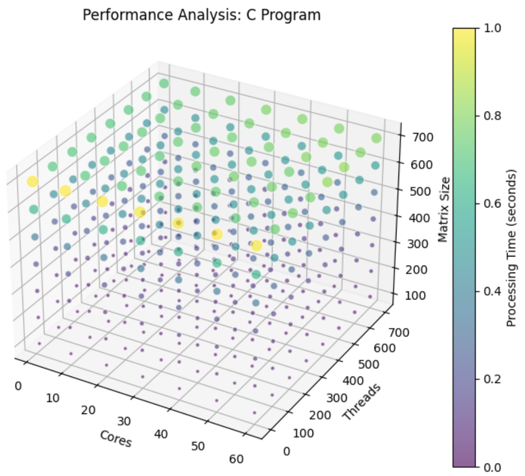

Figure 2: Performance of C with POSIX threads.

The slight performance differences and diminishing returns at higher thread counts led students to independently research and discuss concepts like thread creation overhead and the limits of parallelization.

A key insight for many students was the unexpected performance behavior of multithreaded Python. As shown in Figure 3, increasing the number of threads did not lead to performance gains; in fact, some students observed slight slowdowns. This prompted students to re-examine their initial intuition that "more threads provide more speed", fostering a deeper understanding of the nuances of concurrency.

4.2 Student Perceptions and Self-Reported Learning

We collected Anonymous survey data at the end of each semester. Survey data analysis confirms that students found the assignment engaging, useful, and effective. As shown in Table 1, student reception has been overwhelmingly positive across all three course offerings.

The consistently high ratings for usefulness and willingness to recommend the project indicate that students recognize its value.

[0]The course was not offered in Spring 2023 due to instructor sabbatical. Y/M/N represents the percentage of Yes/Maybe/No responses for each survey question.

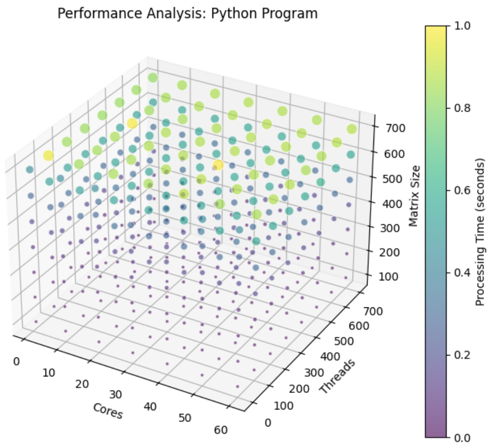

Figure 3: Performance of multithreaded Python.

Table 1: Survey Results by Year

Year	**Interest Y/M/N (%)**	**Understanding Y/M/N (%)**	**Useful Y/M/N (%)**	**Recommend Y/M/N (%)**	**Difficulty (1–5)**
2022	91/9/0	73/27/0	100/0/0	91/9/0	3.27
2024	75/25/0	88/0/12	63/37/0	88/12/0	3.50
2025	64/29/7	71/22/7	79/14/7	79/14/7	3.79

Qualitative feedback reveals three consistent themes across all cohorts. When asked **"What did you like most about the project?"**, students responded:

Access to Professional Computing Resources: Students consistently valued the opportunity to work with production HPC infrastructure:

- "Being able to use a supercomputer to run extremely large computations with a large amount of memory" (2022)
- "Using a super computer is something that I can put on my resume" (2022)
- "Getting to use UF's HiPerGator was an experience that not a lot of undergraduates get to use" (2025)

Hands-on Performance Observation: Students appreciated seeing theoretical concepts manifest in measurable performance differences:

- "I loved seeing the results across languages. It's fascinating to see just how slow python is for stuff like this" (2024)
- "I liked that the program made it very clear how threading effects time needed for execution of programs" (2022)
- "I enjoyed being able to see the difference in my home laptop vs what a supercomputer is capable of" (2025)

Applied Learning Experience: Students valued the practical application of classroom theory:

- "I liked the hands on approach that the project afforded me" (2022)
- "Actually writing code to see how the discussed topics work in real life scenarios" (2024)

This feedback, combined with the quantitative data, supports the effectiveness of providing students with hands-on HPC experiences for learning PDC concepts.

5 Conclusion

This paper presents a project-based learning assignment that leverages a production grade supercomputer to teach fundamental concepts in parallel and distributed computing (PDC). By framing a classic matrix multiplication problem within a discovery-learning framework, we successfully moved students beyond rote memorization of algorithms to a practical understanding of real-world HPC workflows, performance bottlenecks, and language-specific concurrency models.

The primary contribution of this work is a replicable pedagogical model that addresses the common challenge of teaching abstract PDC concepts. The analysis of student work demonstrates clear learning gains in understanding multithreaded behavior and HPC workflows. Furthermore, student self-reported data confirms the assignment is highly engaging and valuable. By situating theoretical knowledge within a practical, hands-on context, students not only learn the material more deeply but also acquire practical skills relevant to skills used in the industry. This approach can be replicated to provide an effective PDC education at the undergraduate level.

6 Acknowledgments

We thank Jonathan Lamoureux and Skyler Gipson for their contributions to earlier stages of this paper.

The authors also acknowledge UFIT Research Computing for providing computational resources and support that have contributed to the research results reported in this publication. URL: `http://www.rc.ufl.edu`

References

[1] Joel C Adams et al. "Teaching PDC in the time of covid: hands-on materials for remote learning". In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2021, pp. 342–349.

[2] Joel C. Adams et al. "Seeing Is Believing: Helping Students Visualize Multithreaded Behavior". In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. SIGCSE '16. Memphis, Tennessee, USA: Association for Computing Machinery, 2016, pp. 473–478. ISBN: 9781450336857. DOI: `10.1145/2839509.2844557`. URL: `https://doi.org/10.1145/2839509.2844557`.

[3] Doug Baldwin. "Discovery learning in computer science". In: *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*. 1996, pp. 222–226.

[4] Emilia Bober and Beata Bylina. "Teaching Parallel Programming on the CPU Based on Matrix Multiplication Using MKL, OpenMP and SYCL Libraries". In: *Proceedings of the 17th International Conference on Computer Supported Education - Volume 2: CSEDU*. INSTICC. SciTePress, 2025, pp. 713–720. ISBN: 978-989-758-746-7. DOI: `10.5220/0013279100003932`.

[5] Jeffrey Cook. "Supercomputers and supercomputing". In: *Visual Analytics and Interactive Technologies: Data, Text and Web Mining Applications*. IGI Global, 2011, pp. 282–294.

[6] Hala ElAarag. "A Hands-on Approach to Teaching Operating Systems through Building a Cluster Using Raspberry Pi's". In: *Journal of Computing Sciences in Colleges* 38.5 (2022), pp. 31–41.

[7] Hala ElAarag. "Deeper learning in computer science education using raspberry pi". In: *Journal of Computing Sciences in Colleges* 33.2 (2017), pp. 161–170.

[8] Chris Fietkiewicz. "Student Outcomes in Parallelizing Recursive Matrix Multiply". In: *Journal of Computational Science Education* (2019).

[9] Christoforos Kachris. “A survey on hardware accelerators for large language models”. In: *Applied Sciences* 15.2 (2025), p. 586.

[10] Gary Lewandowski, Elizabeth Johnson, and Michael Goldweber. “Fostering a creative interest in computer science”. In: *ACM SIGCSE Bulletin* 37.1 (2005), pp. 535–539.

[11] Robert Pucher and Martin Lehner. “Project based learning in computer science–a review of more than 500 projects”. In: *Procedia-Social and Behavioral Sciences* 29 (2011), pp. 1561–1566.

[12] Andrew James Stothers. “On the Complexity of Matrix Multiplication”. PhD thesis. University of Edinburgh, 2010.

[13] UFIT Research Computing, University of Florida. *HiPerGator: University of Florida Research Supercomputing Resources*. `http://www.rc.ufl.edu`.

[14] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[15] Rui-Jie Zhu et al. “Scalable matmul-free language modeling”. In: *arXiv preprint arXiv:2406.02528* (2024).