Parallel Optimization for Robotics: An Undergraduate Introduction to GPU Parallel Programming and Numerical Optimization Research

Brian Plancher[1]

***Abstract*— While parallel programming, particularly on graphics processing units (GPUs), and numerical optimization hold immense potential to tackle real-world computational challenges across disciplines, their inherent complexity and technical demands often act as daunting barriers to entry. This, unfortunately, limits accessibility and diversity within these crucial areas of computer science. To combat this challenge and ignite excitement among undergraduate learners, we developed an application-driven course, harnessing robotics as a lens to demystify the intricacies of these topics making them tangible and engaging. Our course's prerequisites are limited to the required undergraduate introductory core curriculum, opening doors for a wider range of students. Our course also features a large final-project component to connect theoretical learning to applied practice. In our first offering of the course we attracted 27 students without prior experience in these topics and found that an overwhelming majority of the students felt that they learned both technical and soft skills such that they felt prepared for future study in these fields.**

I. Introduction and Motivation

The fields of parallel programming, particularly on graphics processing units (GPUs), and numerical optimization hold immense potential for tackling complex computational challenges across diverse disciplines. Recently, our research has shown that integrating these approaches can lead to significant performance improvements for robotic planning and control [1], [2], [3], [4]. However, these areas currently suffer from high barriers to entry due to their perceived complexity and technical demands, limiting accessibility and diversity within these areas of computing. In fact, the required topics for study in these fields, low-level systems programming and vector calculus, are colloquially referred to as "weed-out" topics, and there is a well documented under representation of women in research in both computer systems (a Female Author Ratio of only 11%), as well as and theory and algorithms (8%) [5].

At the same time, while concepts of parallelism are increasingly found in undergraduate curricula, both through dedicated courses and by weaving the concepts into core classes, and GPUs are now a recommend topic in the updated NSF/IEEE-TCPP Curriculum Guidelines [6], many studies on undergraduate parallel programming do not address the underutilized potential of GPUs [7], [8], [9], [10]. Furthermore, numerical optimization is rarely taught below the graduate level, beyond coverage in standard calculus and linear algebra courses, or a brief module in interdisciplinary engineering courses [11], [12]. These gaps are concerning given the increasing prominence of GPUs in real-world computational applications and the fact that numerical optimization underlies much of modern algorithms that students aim to explore (e.g., training neural networks).

That said, the few studies that do focus on GPU programming do find a handful of common pitfalls and opportunities for increasing student learning [13], [14], [15], [16], [17], [18], [19]. In particular, many students lack background in low-level programming languages and tools. Second, access to GPU hardware is often challenging and most courses leverage the use of remote access to a computer lab, supercomputing cluster, or other cloud-based computing resource. Third, most successful GPU programming courses include project-based learning, both through laboratory and problem-set assignments, as well as large final projects.

In response to these challenges, and building on best practices in the literature, we have developed a course aimed at demystifying these topics through the lens of state-of-the-art robotics applications. Importantly, this course's prerequisites are only courses drawn from the required undergraduate introductory core curriculum[1]. The course is structured into four distinct modules, first introducing foundational topics, before moving to integrated applications of these topics in robotics research, and then ending with a final project.

Through our robotics framing, and low prerequisite approach, we hope that this course attracts a broad range of students and helps address the problematic diversity gaps in these fields. Excitingly, in our first offering of the course we attracted 27 students (one third identifying as women) without prior experience in these topics, and found that an overwhelming majority of the students felt that they learning both technical and soft skills such that they felt prepared for future study in these fields.

II. Course Design

This course introduces undergraduate students to the fundamental concepts of (GPU) parallel programming and numerical optimization through the applied context of robotics. Through a structured, four-module curriculum, students gain

This material is based upon work supported by the National Science Foundation (under Award 2246022). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the funding organizations.

This pseudonymous and anonymous student survey study was approved as exempt by the Institutional Review Board (IRB) of Barnard College on January 18, 2023 (Approval 2223-0505-031E).

[1]Brian Plancher is with Barnard College, Columbia University, New York, NY. bplancher.barnard.edu

[1]Our course prerequisites include 2-3 required courses for the Comptuer Science major, namely: multivariable calculus, linear algebra, and one of Advanced Programming (taught in C), Fundamentals of Computer Systems, or prior experience in C(++) programming.

a comprehensive understanding of both theoretical foundations and practical applications of these rapidly evolving fields. Hands-on coding assignments, in-depth problem sets, and an integrative final project empower students to develop essential skills and apply their knowledge to real-world robotics challenges while also ensuring the development of softer skills such as teamwork and scientific writing. A high level timeline of the course can be found in Figure 1. Detailed descriptions of the course design, and topics covered, can be found in the following sections.

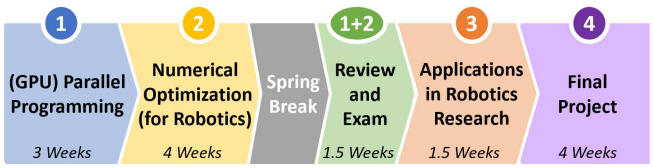

Fig. 1: High-Level Course Overview

A. Guiding Principles and Learning Goals

As our course's guiding motivation is to increase the accessibility of these topics, we designed the course to scaffold topics and assignments through combinations of formative and summative assessment. We also focused on teaching both foundational technical skills, as well as soft-skills, to ensure that students learn skills they can leverage in both future courses and careers. This was done via the choices of topics covered, implementation of problem sets and office hours, as well as the design of the final project.

Accessible Foundational and Applied Topics: As the fields of parallel programming, numerical optimization, and optimization for robotics are all large fields in their own right, we cannot cover all topics, even at a high level. Instead we focus on deeply exploring key fundamental topics and then connecting them to select more complex application driven examples. By initially separating the core topics of programming and mathematics, the course allows students to immerse themselves in each subject and solidifying their understanding and mastery of key skills before moving on. This structure caters to diverse learning styles, offering programming-inclined and mathematically-inclined students a chance to shine in separate modules. Furthermore, by later linking and integrating these concepts through active areas of robotics research, students were exposed to examples of how these fundamentals can be leveraged for real-world applications. Finally, to give all students free access to GPUs both during and beyond the course, we leveraged the Google Colaboratory Programming Environment [20].

Formative Problem Sets and Structured Availability of Office Hours: To make sure students had opportunities for formative learning and could attend office hours, for each problem set, a full class day is turned into a "lab session" to work through the assignment with the professor and TAs. This not only ensures that any confusion with assignment directions can be clarified quickly, but more importantly ensures that all students can access office hours style feedback even if their particular life schedules and demands preclude them from attending formal professor or TA office hour times. Similarly, where possible, coding assignments leverage autograders with a policy that students can submit infinite times until the assignment deadline. This ensures that typos and syntactical errors have a minimal impact on student grades and instead allows students to focus on learning key algorithmic concepts.

Research Integration and Project-Based Learning: The course enables the integration of cutting-edge research into the academic curriculum and offers undergraduates the opportunity to explore such topics through the large final project component. This experience is invaluable, especially for undergraduates, as it provides a safe environment to explore and contribute to ongoing research, fostering a sense of ownership and independence in their learning journey, and exposing them to the opportunities of a career in research.

We begin this phase of the course with a module providing examples of such active, integrative research and the techniques used, and challenges faced by, the researchers in executing these projects. Then, during the project phase, instead of traditional (guest) lectures, we transform 4 weeks of class time into dedicated "lab sessions," to give students ample time for informal feedback and implementation. We also scaffold the project with multiple opportunities for formal feedback including: a proposal, mid-project update, and final report. To further reduce the risks of taking on a more risky, but exciting and potentially impactful, project, we explicitly note in our project instructions that evaluation is not done on the success of the project, but on how the project demonstrates comprehension of the concepts, techniques, and tradeoffs explored in this course. Finally, by requiring students to submit multiple written documents and meet with the course staff multiple times, we aim to foster the development of scientific writing and communication skills.

Through the use of these guiding principles, by the end of the course, our learning goals for students are that they:

- Understand the fundamental concepts of (GPU) parallel programming and numerical optimization including their computational and theoretical tradeoffs.
- Understand how to design and implement (efficient) parallel algorithms for GPUs using CUDA C++.
- Understand the opportunities and challenges of numerical optimization for robotics problems.
- Work effectively in teams to develop a final project.
- Communicate technical concepts clearly and concisely through presentations and reports.

B. Module 1: Parallel Computing (on GPUs)

Module 1 equips students with the theoretical and practical skills for efficient (GPU) parallel computing. This module includes both written and coding based assignments, with a focus on programming skills. Topics include:

- **Basic Parallelism**: Review of computer systems and memory hierarchy, introduction to threads, atomics, and synchronization primitives.
- **GPU Architecture and Programming**: Programming in CUDA C++, the GPU computational model (blocks, threads, warps, SMs) and memory hierarchy (shared,

global, local, host, unified), host-device communication and synchronization.
- **Advanced (GPU) Topics**: Exploring recently released features (e.g., Cooperative Groups).

C. Module 2: Numerical Optimization (for Robotics)

Module 2 equips students with the theoretical and computational tools for numerical optimization with a focus on applications to robotic motion planning and control. This module includes both written and coding based assignments, with a focus on mathematics. At the end of Module 2 there is a midterm exam covering both Modules 1 and 2 to ensure all students have learned all of the foundational knowledge needed to excel in their final projects. Topics include:

- **Theoretical Foundations**: Understanding convexity, global vs. local optima, gradient descent, and the application of vector calculus in optimization.
- **Nonlinear Optimization Techniques**: Utilizing Taylor expansions for function approximation, implementing line search and trust region methods.
- **Constrained Optimization**: (Non-)holonomic constraints, the use of penalty methods, augmented and regular Lagrangians, active sets, and their KKT systems.
- **Trajectory Optimization**: Direct methods, dynamic programming based methods, and their connections.

D. Module 3/4: Putting it All Together and Final Project

In Module 3, students explore how recent research integrates parallel programming to accelerate numerical optimization algorithms for improved robot performance. Course content focuses on how leveraging both instruction-level and algorithm-level parallelism, in combination with code generation techniques, can enable the development of portable and performant solutions.

Module 4, the capstone of the course, is the integrative final project. Students work in teams to tackle real-world robotics challenges. This intensive project engages the full spectrum of acquired knowledge and skills both technical and non-technical in nature. Students are required to: formulate a relevant (robotics) problem that can be solved through GPU parallel programming and/or numerical optimization (ideally both); design, implement, and optimize their solution and measure its performance; and communicate their findings. Most importantly, they must do all of this while collaborating on a team. Through this rigorous project, students gain valuable experience in integrating and applying their knowledge to solve complex engineering problems, preparing them for research and development in their future careers.

III. Preliminary Evaluation

This course was run for the first time in the Spring 2023 semester. 27 students (one third identifying as women) enrolled in the course from across Barnard College, Columbia College, and the Columbia School of Engineering and Applied Science. The instructor was supported by a masters student TA who had previously done research with the instructor on related topics. Throughout the course, students were polled pseudonymously to both understand student background, as well as get feedback on the workload, clarity of topics, and pace of the course. These polls were pseudonymous and not anonymous in order to correlate pre- and post-unit knowledge of the students through the inclusion of a handful of quiz questions on the surveys. Unfortunately, as these polls were optional, and many students forgot their pseudonyms, we did not collect paired responses from all students. Similarly, the final official course evaluations at Barnard are not mandatory and as such again we did not collect responses from all students. And, due to the different systems being used, the questions are not identical. However, the responses we did receive provide a positive indication that we attracted students to the course that had no prior background in these topics, that students learned both technical and soft skills during the course, and that the course increased students' interest in these fields.

As shown in Figure 2, on a scale of 1 (No Background) to 5 (Expert), most students (n=25) gave themselves a 1 in course topics: parallel programming (62%), numerical optimization (65%), trajectory optimization (72%), robot motion planning and control (60%). Furthermore, no student gave themselves a 5 across any of those topics. Similarly, despite the course requiring some background in C(++), 80% of students gave themselves a 3 or lower for C(++) programming, while 84% gave themselves a 4 or higher in Python programming. As such, low-level programming was a perceived prior weakness for students.

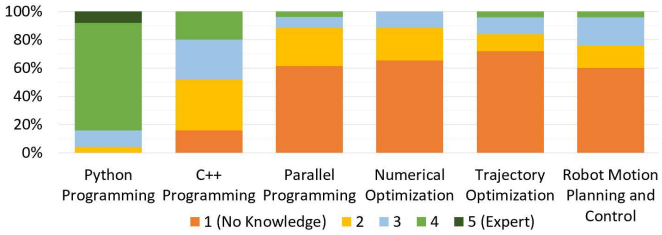

Fig. 2: Self-reported prior knowledge of students entering the course (n=25).

Figure 3 shows the results of the pseudonymous quizzes indicating that across all topics, responding students learned core concepts during the foundational units. Figure 4 then shows select summarized results from the official course feedback. Excitingly, over 90% of responding students perceived that they grew their quantitative, presentation, and writing skills, as well as their interest in and understanding of the field "quite a bit" or "very much" (the two highest options). Similarly, 90% of responding students "agreed" or "strongly agreed" (the two highest options) that they felt prepared for a future course in (GPU) parallel programming. As the course was taught in CUDA C++, this also indicates student comfort with low-level programming following the course. While, the results were less positive for (trajectory) optimization, the majority of the class, 60-70%, still "agreed" or "strongly agreed" that they would be prepared for a future course. Most importantly, a number of students have since pursued research and jobs leveraging these skills.

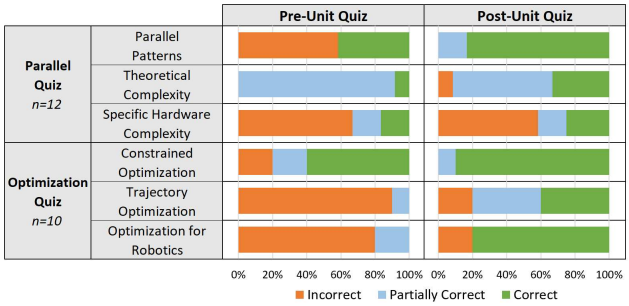

Fig. 3: Pseudonymous quiz results reveal that across all topics responding students learned core concepts.

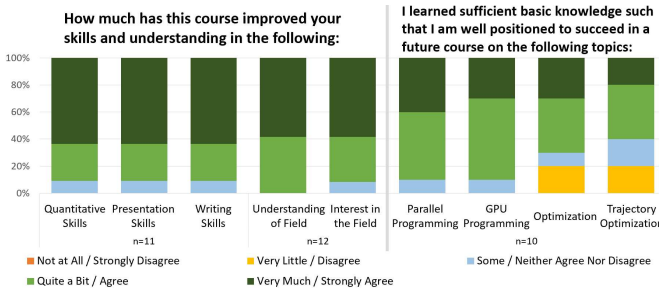

Fig. 4: Self-reported learning from students on the final course evaluations.

IV. Limitations and Future Work

While the pilot run of this course yielded promising results, one key limitation is the small sample size, both of enrolled students (27), of questions used to gauge their learning progress (one per quiz topic), as well as the response-rate on the surveys. Furthermore, the lack of consistently formatted and worded questions hinders our ability to objectively measure student understanding.

Despite these limitations, the pilot provided invaluable insights that we are using to improve the course this spring. First, we aim to scale enrollment from 27 to 75 students and incorporate dedicated in-class survey time. This should both allow us to collect better data for future evaluations, as well as offer expanded office hour times, which students noted would be helpful. Additionally, improving the standardization of assessment questions will enable more accurate measurement of student learning. We also hope to explore additional opportunities for hands-on lab style sessions by introducing select content in a flipped classroom style. Also, exploring alternative platforms beyond Google Colaboratory, which proved cumbersome for some use cases, will help improve student experience and foster more comprehensive projects. Finally, we hope to enable other educators to leverage our work by releasing our materials open-source.

V. Conclusion

While GPU parallel programming and numerical optimization offer immense potential across disciplines, their complexity often provides a barrier-to-entry. To break down these barriers and excite undergraduates, we created a course on these topics framed through the lens of robotics that only required core CS courses as prerequisites. Through a focus on research applications and a large final project, students were given the opportunity to connect theory to practice. Our first offering of the course had 27 students who entered the course as novices and left feeling equipped with both technical and soft skills for further study in these fields.

References

[1] B. Plancher and S. Kuindersma, "A Performance Analysis of Parallel Differential Dynamic Programming on a GPU," in *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

[2] B. Plancher, *et al.*, "Accelerating robot dynamics gradients on a cpu, gpu, and fpga," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2335–2342, 2021.

[3] ——, "Grid: Gpu-accelerated rigid body dynamics with analytical gradients," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2022.

[4] E. Adabag, *et al.*, "Mpcgpu: Real-time nonlinear model predictive control through preconditioned conjugate gradient on the gpu," in *IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, Japan, May. 2024.

[5] E. Frachtenberg and R. D. Kaner, "Underrepresentation of women in computer systems research," *Plos one*, vol. 17, no. 4, 2022.

[6] S. K. Prasad, *et al.*, "Nsf/ieee-tcpp curriculum initiative on parallel and distributed computing: status report," in *ACM Technical Symposium on Computer Science Education*, 2018, pp. 134–135.

[7] S. Ghafoor, D. W. Brown, and M. Rogers, "Integrating parallel computing in introductory programming classes: an experience and lessons learned," in *Euro-Par 2017: Parallel Processing Workshops, Revised Selected Papers*. Springer, 2018, pp. 216–226.

[8] M. A. Kuhail, *et al.*, "Teaching parallel programming with active learning," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 369–376.

[9] D. J. Conte, *et al.*, "Teaching parallel programming for beginners in computer science," in *2020 IEEE Frontiers in Education Conference (FIE)*, 2020, pp. 1–9.

[10] A. A. Younis, *et al.*, "Developing parallel programming and soft skills: A project based learning approach," *Journal of Parallel and Distributed Computing*, vol. 158, pp. 151–163, 2021.

[11] S. M. Morse, *et al.*, "The impact of reducing numerical methods and programming courses on undergraduate performance," in *2014 ASEE Annual Conference & Exposition*, 2014, pp. 24–1223.

[12] A. Alpers and L. E. Trotter, "Teaching computational discrete optimization at the undergraduate level," *INFORMS Transactions on Education*, vol. 9, no. 2, pp. 63–69, 2009.

[13] M. Bailey and S. Cunningham, "A hands-on environment for teaching gpu programming," *ACM Sigcse Bulletin*, vol. 39, no. 1, pp. 254–258, 2007.

[14] D. Bunde, *et al.*, "Adding gpu computing to computer organization courses," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, 2013, pp. 1275–1282.

[15] R. Geist, J. A. Levine, and J. Westall, "A problem-based learning approach to gpu computing," in *Proceedings of the Workshop on Education for High-Performance Computing*, 2015, pp. 1–8.

[16] J. A. Shamsi, "A laboratory based course on gpu programming: Methods, practices, and lessons," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 367–374.

[17] J. Gutierrez, F. Previlon, and D. Kaeli, "Employing student retention strategies for an introductory gpu programming course," in *2018 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*, 2018, pp. 31–40.

[18] G. Fenwick and C. Norris, "Gpgpu programming for cs undergraduates: Which one is superman?" in *Proceedings of the 2020 ACM Southeast Conference*, 2020, pp. 2–9.

[19] M. Ohkawara, H. Saito, and I. Fujishiro, "Experiencing gpu path tracing in online courses," *Graphics and Visual Computing*, vol. 4, p. 200022, 2021.

[20] E. Bisong and E. Bisong, "Google colaboratory," *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pp. 59–64, 2019.