Helping Faculty Teach Software Performance Engineering

John D. Owens
Dept. of Electrical & Computer Engr.
University of California, Davis
Davis, California, USA
jowens@ece.ucdavis.edu

Bruce Hoppe
Connective Associates
Arlington, Massachusetts, USA
behoppe333@gmail.com

***Abstract*—Over the academic year 2022–23, we discussed the teaching of software performance engineering with more than a dozen faculty across North America and beyond. Our outreach was centered on research-focused faculty with an existing interest in this course material. These discussions revealed an enthusiasm for making software performance engineering a more prominent part of a curriculum for computer scientists and engineers. Here, we discuss how MIT's longstanding efforts in this area may serve as a launching point for community development of a software performance engineering curriculum, challenges in and solutions for providing the necessary infrastructure to universities, and future directions.**

***Index Terms*—Performance; distributed, parallel, and cluster computing; data structures and algorithms; software performance engineering.**

I. Introduction

Over the past year, we conducted focused interviews with more than a dozen faculty worldwide about software performance engineering (SPE): making software run fast or otherwise consume few resources such as time, storage, and energy. Since the demise of Moore's Law, SPE has grown increasingly important as one of the most promising ways to continue providing gains in application performance. We were fortunate to work as part of the Fastcode Team with the support of Charles Leiserson. He co-developed SPE as a course at MIT, and we shared the resources from this course as part of our outreach. We offered the resources as a seed for an open-source repository of SPE teaching materials and as a springboard to discussing how others would teach SPE and what obstacles (if any) prevent them from doing so. This paper summarizes what we learned, focusing on the following questions:

- What is SPE? What are the barriers to teaching it?
- If the barriers were overcome, how could a collaborative and/or open-source ecosystem help you teach SPE? What topics would you put in your class? What teaching materials, broadly construed, would be most useful?
- How would you manage the computing resources required for teaching SPE?
- What are the directions we should take SPE? How would you hope to contribute?

The second author works as an external consultant for MIT and leads user engagement for the Fastcode OSE. This research was supported in part by NSF Grant 2229704 and in part by the United States Air Force Research Laboratory under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

II. Software Performance Engineering

Although software performance engineering has been around since the early days of computing, it has gained attention recently due to the end of Moore's Law [1]. For decades, Moore's Law allowed programmers to improve application performance simply by waiting for faster hardware, but no more. Not only that, the capabilities of modern multicore chips are much harder to exploit than their predecessors. We believe that SPE techniques and tools are among the most promising for providing applications with continued gains in performance, and so in 2022 we joined the SPE efforts of the multi-institution Fastcode Team. Charles Leiserson, who created the SPE curriculum with Saman Amarasinghe at MIT in 2009, leads the team and helps us to leverage all the resources developed over more than fifteen years of teaching SPE at MIT. Also in 2022, the Fastcode Team won Phase I support from a brand-new NSF program, Pathways to Enable Open-Source Ecosystems (POSE), which positioned us as the "managing organization" responsible for scoping (and eventually building) the Fastcode Open-Source Ecosystem (OSE). The core open-source content of the Fastcode OSE is OpenCilk [2], [3], an integrated platform for task-parallel programming. To kick off our ecosystem for using this content, we launched the OpenCilk.org website with a featured page of resources for teaching SPE, including downloadable lecture slides from MIT and UC Davis [4]. The work reported here is intended to complement OpenCilk and expand beyond it by exploring how we can make it easier for faculty to teach SPE—with or without OpenCilk. In short, the Fastcode OSE aims to catalyze an integrated community of software developers, researchers, and educators, who are equipped with software, learning materials, research foundations, and skill-

building environments to advance SPE as a rigorous and principled scientific field.

We believe that a foundational part of our effort to create a community around SPE is teaching students. We hope that we can train students ready to meet the significant need of the computing community by developing a high-quality curriculum in SPE, and ensuring that curriculum can be easily adopted by faculty at both research- and teaching-focused schools. To that end, we held focused one-on-one conversations with faculty, largely about obstacles they face in teaching SPE and how our nascent ecosystem might help. The majority of our outreach was to North American faculty at research-focused universities, but we were also able to talk to faculty outside of North America and to faculty at teaching-focused institutions. Because we talked to faculty who had already expressed interest in the topic, and/or had been recommended as having an interest by other faculty, we had a reliably positive reception. Here's what we learned:

- **Faculty desire an integrated cross-stack approach.** Our vision of SPE incorporates material in many areas, including architecture, compilers, programming systems, algorithmic engineering, and algorithms. Faculty appreciated such a cross-stack approach to teaching SPE, complementing most courses in computer science and engineering that focus on only one level of the stack.
- **Faculty are highly time-constrained.** Preparing a new course is a lot of work; preparing it from scratch is considerably more; so to enable faculty to teach this material, we must prioritize their time.
- **Faculty need and appreciate high-quality teaching materials.** Lecture slides and videos teaching them were most commonly requested, but also homework assignments, tutorials, and projects. Several faculty, but not all, requested a textbook, while others did not expect to use or want one.
- **Faculty require help to configure teaching machines for performance engineering.** For reliable performance measurement, instructional computers are not suitable off-the-shelf without significant configuration. The typical professor has little experience with such configuration and will benefit from the experience of the Fastcode Team and the infrastructure we have developed within our ecosystem.
- **Insufficient instructional staff background.** Faculty typically have little experience using a variety of parallel-programming platforms (such as OpenCilk), and their TAs even less; providing high-quality, time-efficient training material that brings a teaching staff up to speed with the relevant platforms for their class is critical.

III. Curriculum and Teaching Materials

In the first part of our interview protocol, we asked faculty about obstacles preventing them from teaching SPE, despite our shared sense of its importance to modern computer science education. In the next part of the protocol, we asked faculty (i.e., recognized experts) how they would teach SPE if those obstacles were removed. We summarize their responses here, hoping to contribute to a collective conversation about what belongs in SPE. To give context to our interviewees and allow more pointed discussion about their preferences, we began the discussion with the 2018 MIT SPE curriculum [4]. This included lecture material on:

- Introduction & Matrix Multiplication
- Bentley Rules for Optimizing Work
- Bit Hacks
- Assembly Language and Computer Architecture
- C to Assembly
- Multicore Programming
- Races and Parallelism
- Analysis of Multithreaded Algorithms
- What Compilers Can and Cannot Do
- Measurement and Timing
- Storage Allocation
- Parallel Storage Allocation
- The Cilk Runtime System
- Caching and Cache-Efficient Algorithms
- Cache-Oblivious Algorithms
- Nondeterministic Parallel Programming
- Synchronization Without Locks
- Domain Specific Languages and Autotuning
- Speculative Parallelism
- Tuning a TSP Algorithm
- Graph Optimization
- High Performance in Dynamic Languages

In general, faculty agreed that the syllabus for the MIT course was broad and appropriate. Faculty appreciated the cross-cutting nature of the MIT course and contrasted it with the typical single-layer approach of the majority of their curricula. (Faculty feel that both single-layer and multi-layer courses in the curriculum are relevant and appropriate, but multi-layer courses are less common.)

In general faculty felt this course was best suited as an upper-division undergraduate course, although mezzanine (mixed advanced-undergrad and beginning-grad) and grad courses also received interest. One faculty member noted that the largest body of degree-holders that enter industry are BS degrees and thus we should focus on that cohort of students. Other faculty noted the growth of their (online) MS programs and that this material would be useful there. Some departments also offer industry-focused courses, and faculty saw that this material potentially fits that need.

Faculty had numerous suggestions for extending the existing repository of course material. The primary topics mentioned were distributed computing (multi-socket and multi-node, including frameworks like Spark), GPUs, and performance engineering of networks and I/O. Faculty also mentioned vectorization, managed languages, performance counters and how to effectively use them, how security intersects with SPE, and virtual-machine aspects of SPE, among many others.

The MIT curriculum focuses on using performance tools (e.g., profilers) to measure performance, identify bottlenecks, and address them. Such tools (broadly) don't modify source code but instead measure, and MIT teaches students how to control the effects of compile- and run-time optimizations before they take measurements. One of our faculty contacts

instead described his philosophy of *instrumenting* code to see "what's actually happening", rather than *profiling* it (e.g., as taught at MIT). Both approaches are valuable, and the Fastcode Team has since had discussions about the need to also teach instrumentation. This is certainly an interesting topic for an additional lecture.

In the opinion of the first author, who has no MIT connection, the MIT course (naturally) has a strong MIT flavor, in particular with respect to its focus on OpenCilk. Surveyed faculty generally agree: OpenCilk is beautiful, both theoretically and in practice, and is a joy to teach; however, the MIT course lacks a broader context for studying other multicore parallel programming environments. It would be useful to develop a taxonomy of classes of parallel computing problems and teach parallel computing through that lens, focusing on how and why different parallel computing environments are better or more poorly suited for particular classes of problems and why. This is especially important for students who may (soon!) work in companies where OpenCilk is not available, and who may need to choose other environments. We spoke with one team of faculty who are actively maintaining such a taxonomy as a foundational part of their online textbooks, *PDC for Beginners* and *Intermediate PDC* [5], [6]; related resources on patterns of parallel programming are listed in the bibliography [7]–[10].

Independent of the topics covered, we asked faculty about the importance of lecture slides, homeworks, projects, and other teaching materials. What would be most helpful as part of our open-source ecosystem? Faculty agreed that the combination of lecture slides and videos of those lectures being delivered are the single most important item that we can provide, and we are fortunate to have permission to share all of MIT's (superb!) lecture slides and videos, which have been developed over many years. A library of existing projects that are interesting and educational for students is also highly desirable. Faculty acknowledge the challenge in developing workable, properly targeted projects, especially those that may use a programming environment where the teacher may have little or no expertise. MIT's philosophy in writing projects is to give students working code that fulfills the project description but is slow (e.g., unoptimized or unparallelized). Students are then responsible to improve project performance. We agree with this philosophy, but it does require identifying projects where the performance gap between initial code and final code is both large and pedagogically interesting.

Faculty expressed the desire for a *modular* SPE curriculum where different pieces of material could be easily integrated into an existing course. We return to this in Section V, along with our desire to provide a roadmap that helps faculty to approach SPE from different perspectives.

IV. Computing Infrastructure

Teaching SPE requires accurately measuring performance. The MIT course stresses reproducible, consistent measurements on *quiesced* servers. We believe quiesced servers are critical for teaching SPE, but this means that a performance engineering class can almost certainly not use generic department-supplied computing environments without custom effort. (For example, the typical department computing resource is a pool of Unix workstations that allow local and remote logins; it is very difficult to get reliable performance measurements when many students can be running jobs on the same machine.)

Over the course of our outreach, we discussed two tentative paths toward providing faculty with quiesced servers for teaching SPE: using a cloud provider and configuring local machines. The Fastcode Team hopes to build an ecosystem for helping potential SPE instructors as they consider these options, and we welcome inquiries from interested faculty. (See also Section V for other ways we hope to help.) Briefly, here's what we discussed so far.

The benefit of a cloud provider is that it "only" costs money but does not require internal department resources. (At UC Davis, the cost was $155 per enrolled student over a 10-week academic quarter.) Setup involves billing as well as configuring a cloud computer to provide reliable performance measurements. This option is perhaps best for departments that have a little more cash than time or computing resources. It is also probably more straightforward for departments that want to offer an experimental (one-time) course before committing to adding it to their curriculum. Both MIT and UC Davis have used AWS.

Configuring a local machine is perhaps best for departments that have a server or servers that can be dedicated to the course for the entire academic term. UC Davis chose this option in its most recent course offering, with managed access to a dedicated server in the college's compute cluster. The server was only available to students in the course, was configured for reliable performance measurements at the start of the quarter, was accessed through the Slurm job submission system, and appeared to provide enough compute to satisfy 30 students in the course, even at peak times.

In addition to the above two options, one of our faculty contacts noted his success with the NSF ACCESS program for student assignments. We have not yet investigated the suitability of ACCESS for quiesced servers or for a typical student computing load in course on SPE, but we believe it is a promising approach.

V. Future Directions

Our outreach was successful in determining several next steps that would help advance our curricular goals.

A. Collaborating with Related Projects

Currently, the Fastcode Team sees SPE research as scattered across traditional areas of computer science, and we seek to catalyze a more integrated SPE community for researchers and educators. This includes integrating our own work with other projects. During our outreach we talked with leaders of three projects that we think are especially promising allies in our work to advance SPE education.

The CSinParallel project has been devoted since 2010 to providing freely available educational materials in the form

of teaching modules and hands-on book chapters that enable professors to introduce parallel and distributed computing (PDC) at all levels of the undergraduate CS curriculum using small units that they can fit into existing courses with minimal disruption [11], [12]. The participants and contributors to this effort come largely from more teaching-focused institutions, and complement our research-focused outreach to date.

The CS-Materials website aims to help instructors navigate the process of integrating PDC into their courses [13]. The system allows faculty to view and curate teaching materials and course descriptions through the lens of NSF/IEEE-TCPP PDC curriculum guidelines. The research behind this system reflects a comprehensive landscape of efforts to organize and implement the topics of PDC education [14].

The Rogues Gallery project was initiated by Georgia Tech's Center for Research into Novel Computing Hierarchies [15]. They approach SPE education by focusing on next-generation hardware and uncommon technologies, which complements our focus on "commodity" CPUs.

B. More Modular Materials

In addition to helping faculty teach courses on SPE, we also want to help them integrate SPE topics into the crowded curricula of existing classes. To that end, our goals are similar to those of CSinParallel and CS-Materials for teaching PDC. Our ideal SPE curriculum would have a broader set of materials that can be presented at multiple levels of detail: for example, 20 minutes of lecture, one full lecture, or three lectures. We also hope to identify topics that are well suited for self-study or TA-led recitations. Finally, we would like to help faculty mix and match modules with confidence, regardless of where they're starting from (e.g., approaching SPE from a theory perspective or from a parallel-programming perspective). We therefore hope to provide a roadmap of dependencies describing what must be covered before presenting any particular module in our SPE ecosystem.

C. Real-World Case Studies

We believe the SPE community would benefit greatly from acquiring a set of *case studies*. Each case would feature existing code from a real-world application that is slow, for some reason, where students must profile and/or instrument this code to find where it is inefficient and fix it. One benefit of case studies is to equip students for success after school. We spoke to numerous companies during our outreach, and they all told us how hungry they are for people who have learned how to optimize existing code written by others. Another benefit of case studies, compared to traditional student projects that are often wholly constructed by the instructor, is to help faculty identify the key principles and connections that bind and differentiate SPE with other areas of computer science. We believe SPE will be much stronger as a field if faculty can construct student projects from real-world problems or applications that are bottlenecked by performance, and the Fastcode Team hopes for an SPE ecosystem that helps to identify those real-world problems and develop them into case studies.

D. Distributed Content Development

In our outreach, we found that that faculty acknowledge that at the outset they would expect to mostly be consumers of course material, but they are also enthusiastic about contributing their material back to a common repository, and perhaps taking a leadership role on particular topics they know best. We look forward to cultivating distributed content development.

VI. Conclusion

We are grateful for the time of, and valuable discussion with, the faculty from our year of outreach. The most interesting points above were nearly all developed as a direct result of these discussions. We strongly believe that SPE is a significant opportunity for the computer science and engineering community and hope that our efforts will help students bring its lessons and techniques into industry, government, and academia.

References

[1] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's plenty of room at the Top: What will drive computer performance after Moore's law?" *Science*, vol. 368, no. 6495, Jun. 2020. [Online]. Available: https://doi.org/10.1126/science.aam9744

[2] T. B. Schardl and I.-T. A. Lee, "OpenCilk: A modular and extensible software infrastructure for fast task-parallel code," in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '23. ACM, Feb. 2023, pp. 189–203. [Online]. Available: https://doi.org/10.1145/3572848.3577509

[3] The OpenCilk organization, "OpenCilk," Available at https://github.com/OpenCilk, 2023.

[4] ——, "Teach Performance," Available at https://www.opencilk.org/community/teach-performance/, 2023.

[5] J. C. Adams, R. Brown, S. J. Matthews, and E. Shoop, "PDC for Beginners," Available at https://dx.doi.org/10.55682/VXWY1300, 2023.

[6] E. Shoop, "Intermediate PDC," Available at https://www.learnpdc.org/IntermediatePDC/, 2023.

[7] T. G. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*, 1st ed. Addison-Wesley Professional, Sep. 2004.

[8] "Our Pattern Language," Available at https://patterns.eecs.berkeley.edu/, 2024.

[9] M. McCool, J. Reinders, and A. Robison, *Structured Parallel Programming: Patterns for Efficient Computation*, 1st ed. Amsterdam: Morgan Kaufmann, Jul. 2012.

[10] P. Balaji, *Programming Models for Parallel Computing*. MIT Press, Nov. 2015, google-Books-ID: L6kCCwAAQBAJ.

[11] "CSinParallel," Available at https://csinparallel.org/, 2024.

[12] "Learn PDC," Available at https://learnpdc.org/, 2024.

[13] E. Saule, K. Subramanian, and J. Payton, "CS Materials," Available at http://cs-materials.herokuapp.com/, 2020.

[14] A. Goncharow, M. Mcquaigue, E. Saule, K. Subramanian, P. Goolkasian, and J. Payton, "CS-Materials: A system for classifying and analyzing pedagogical materials to improve adoption of parallel and distributed computing topics in early CS courses," *Journal of Parallel and Distributed Computing*, vol. 157, pp. 316–330, Nov. 2021. [Online]. Available: https://doi.org/10.1016/j.jpdc.2021.05.014

[15] Georgia Tech Center for Research into Novel Computing Hierarchies, "The rogues gallery," Available at https://crnch-rg.cc.gatech.edu/, 2024.