Speedcode: Software Performance Engineering Education via the Coding of Didactic Exercises

Tim Kaler
MIT CSAIL
tfk@mit.edu

Xuhao Chen
MIT CSAIL
xchen@csail.mit.edu

Brian Wheatman
Johns Hopkins University
wheatman@cs.jhu.edu

Dorothy Curtis
MIT CSAIL
dcurtis@csail.mit.edu

Bruce Hoppe
Connective Associates
behoppe333@gmail.com

Tao B. Schardl
MIT CSAIL
neboat@mit.edu

Charles E. Leiserson
MIT CSAIL
cel@mit.edu

***Abstract*—This paper introduces Speedcode, an online programming platform that aims to improve the accessibility of software performance-engineering education. At its core, Speedcode provides a platform that lets users gain hands-on experience in software performance engineering and parallel programming by completing short programming exercises.**

Speedcode challenges users to develop fast multicore solutions for short programming problems and evaluates their code's performance and scalability in a quiesced cloud environment. Speedcode supports parallel programming using OpenCilk, a task-parallel computing platform that is open-source and easy to program, teach and use for research.

Speedcode aims to reduce barriers to learning and teaching software performance engineering. It allows users to run and evaluate their code on modern multicore machines from their own computer without installing any software. This provides users an easy introduction to the topic, and enables teachers to more easily incorporate lessons on software performance engineering into their courses without incurring the onerous overhead of needing to setup computing environments for their students.

I. Introduction

Modern software performance engineering is becoming an increasingly vital skill as Moore's Law and the days of free performance improvements come to an end. Many of the innovations that have been improving the capabilities of hardware over the past decade, such as larger numbers of cores and large vector units, add significant complexity to the traditional programming model and are consequently difficult to program by non-experts. These more complex programming models place a growing demand on educators to prepare students with the skills required to understand and optimize for software performance in the modern computing landscape.

Despite the growing importance of software performance engineering, there are few opportunities for students to pickup these skills. Software performance engineering is rarely taught in undergraduate computer science programs, and there are few informal avenues for novice programmers to pickup these skills outside of the classroom. This situation is in stark contrast with other aspects of the computer science curriculum like algorithms, data structures, and machine learning where gaining hands-on experience is relatively accessible.

Online programming platforms, the most popular of which is LeetCode [1], have become widely-used resources for individuals wishing to improve their proficiency with algorithms, data structures, and programming. Sites like LeetCode are popular with undergraduates in computer science programs, and even popular among more experienced software engineers wishing to refresh or sharpen their skills[1].

Software performance engineering, however, is not presently easy to learn (or teach) via casual practice. One reason for this difficulty is that the simple task of collecting reliable measurements of a code's performance can be difficult when operating in noisy computing environments (e.g., one's personal computer or on a shared remote machine in the cloud). Another major difficulty is the large overhead required to setup and learn how to use various "tools of the trade" that are needed to analyze software performance in a principled fashion. Existing online platforms for teaching parallel and distributed computing, such as OnRamp [14], have done much to alleviate this burden in classroom settings, but are typically not broadly accessible for informal learning.

This paper reports on our ongoing efforts to improve the accessibility of software performance-engineering education through the development of an online programming platform called ***Speedcode***[2] that supports both parallelism and quiesced performance analysis. Speedcode allows users to develop fast multicore solutions to short programming problems and evaluate the performance and scalability of their solutions in a quiesced cloud environment, with no local installation required. Speedcode supports parallel programming using OpenCilk, a task-parallel computing platform that is open-source [2] and easy to program, teach and use for research [24].

The remainder of the paper is organized as follows. Section II describes the pedagogy and technologies used by Speedcode to teach parallel programming. Section III describes the design of the Speedcode programming platform and illustrates the tools it provides to make it easier to write and analyze parallel codes. Section IV describes our

[1]A search for "leetcode" on TeamBlind, an online professional community, returned over 28,000 topics on the use of Leetcode for interview preparation.

[2]http://speedcode.org

preliminary experiences using the Speedcode platform in classrooms, tutorials, and other small-group settings to teach concepts in parallel computing. Section V discusses our plans for developing well-organized curriculums to teach software performance engineering through the Speedcode platform.

II. Pedagogy for teaching parallel programming

Speedcode's approach follows established principles from the parallel computing community for developing parallel codes that are easy to write, debug, and analyze.

A. Deterministic parallelism.

Researchers over multiple decades have advocated for the use of "deterministic parallelism" [9], [13], [17], [22], [25] to reduce the difficulty of parallel programming. With a deterministic parallel program, the programmer observes no logical ***concurrency***, that is, no nondeterminacy in the behavior of the program due to the relative and nondeterministic timing of communicating processes such as when one process arrives at a lock before another. The semantics of a deterministic parallel program are therefore serial, and reasoning about such a program's correctness, at least in theory, is no harder than reasoning about the correctness of a serial program. Testing, debugging, and formal verification are simplified, because there is no need to consider all possible relative timings (interleavings) of operations on shared mutable data.

B. Fork-join parallelism.

Programming models such as ***fork-join parallelism*** have made significant progress towards easing the difficulty of writing correct and deterministic parallel codes. Fork-join parallelism is usually implemented using ***work-stealing*** [8], where worker threads in the runtime system coordinate to load-balance the computation, as in the various Cilk dialects [15], [20], [24], Fortress [3], Habanero [5], HotSLAW [21], Java Fork/Join Framework [18], OpenMP [4], Task Parallel Library [19], and Threading Building Blocks (TBB) [23]. In this model, subroutines can be spawned in parallel, generating a series-parallel execution dag in which the synchronization of subtasks is managed by the runtime system. Constructs such as `parallel_for` can be implemented as syntactic sugar on top of the fork-join model. As long as the parallel program contains no ***determinacy races*** [13], the program is deterministic. Moreover, efficient tools exist that are guaranteed to detect determinacy races or validate their absence [13], [24].

C. Work-span analysis

Work-span analysis [11, Ch. 27] is a technique for analyzing the theoretical parallelism in a parallel program and predict its wall-clock execution time on P-processors. Given an execution of a parallel program, one defines the ***work*** T_1 to be the total number of instructions executed and the ***span*** (or ***depth***) T_∞ to be the total number of instructions executed on the program's critical path. Conceptually, the work of a program is its sequential runtime and its span is the program's ideal runtime on an infinite number of processors. The parallelism of a program can be computed as the ratio T_1/T_∞ of the work and span. Greedy schedulers [10], [12], [16] can execute a program on P processors in time T_P satisfying $max\{T_1/P, T_\infty\} \leq T_P \leq T_1/P + T_\infty$ on an ideal parallel computer. Similar bounds can be achieved when using randomized work-stealing schedulers [7], [8].

D. Fork-join parallel programming using OpenCilk

Speedcode employs technologies developed by the open-source OpenCilk project [24]. OpenCilk provides a compiler, programming model, and program analysis tools that facilitate the development of deterministic fork-join parallel codes that are fast in both theory and practice. Importantly, the Cilk programming model has serial-semantics for race-free parallel programs, which makes it easier for those new to parallel computing to understand and debug their code's correctness.

Notably, OpenCilk provides two high-quality tools that make it easier to teach parallel programming. The first is the ***Cilksan*** race-detector which reports determinacy races that exist when a code is run on a given input, and verifies the absence of races when the code is race-free. The second is the ***Cilkscale*** scalability analyzer which measures the work T_1 and span T_∞ of a program execution to compute the parallelism T_1/T_∞ of a submitted solution.

III. Speedcode online learning platform

The current prototype of Speedcode provides an online development environment that is integrated with OpenCilk (and related tools) to support fork-join parallel programming, scalability analysis, and race detection. The remainder of this section explains how these tools work within Speedcode.

Speedcode consists of a collection of short programming exercises[3] that are written and evaluated via a front-end web interface. Each problem has a short description or prompt that includes details about the problem, the expected inputs, and (in some cases) suggestions on how to optimize the reference solution. Correct starter code is given for each problem.

User interface. Users solve Speedcode problems using a web front-end interface. Figure 1 provides an illustration of the main user-interface for submitting and evaluating solutions to Speedcode problems. Presently, Speedcode displays reports that include details on correctness, benchmarks, parallel scalability, and race detection. As additional tools are integrated into Speedcode, we envision a modular user interface where the user or problem developer can choose which tools and reports are generated and displayed.

Execution environment. Submitted problems are compiled and run on a dynamically scaled pool of quiesced multicore servers. Presently, Speedcode employs 8-core multicore machines with 2nd generation Intel Xeon Cascade Lake processors. User code is benchmarked, transparently to the user, via a micro-benchmarking library that allows for nanosecond resolution runtime measurements. The use of quiesced machines and this principled benchmarking methodology allows students to

[3]As of 1/28/2024 there are 18 programming exercises on Speedcode, most of which benefit from the use of task-paralellism.

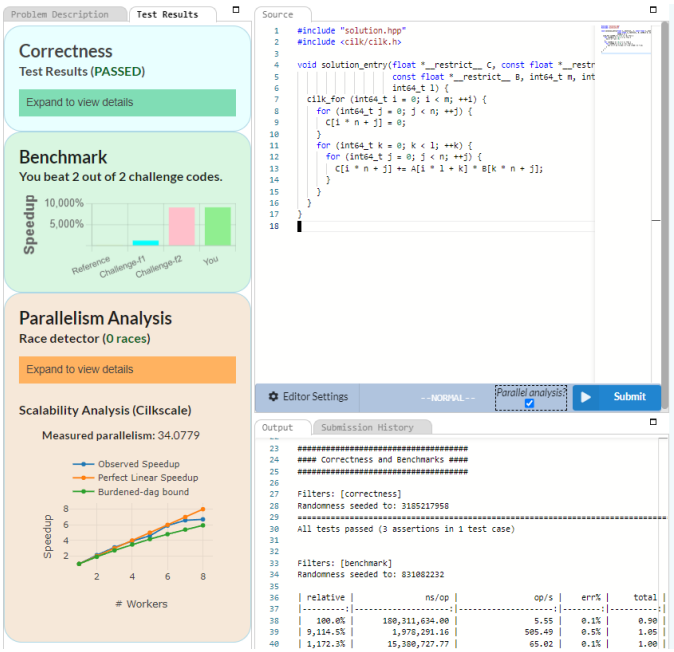

Fig. 1: User interface for a Speedcode problem. *Left*: the correctness, benchmark, and parallelism reports for a submission. *Right*: the code editor and program output.

focus on improving performance, as the measurement problem is taken care of by the Speedcode platform.

Benchmarks and challenge problems. The performance of the submitted code is measured using a set of test inputs provided by the problem developer. A user's goal, when solving a Speedcode problem, is to improve the performance of a correct reference implementation. To provide the user intermediate goals, the benchmark results of a user's submitted code are compared to a list of "challenge codes" which achieve varied levels of performance. These challenge codes are typically designed to be representative of the different kinds of optimizations users may try while trying to solve the problem. Although performance engineering tasks often do not have a well-defined notion of "completion", the task of improving performance until one has beaten all of the challenge codes is a tangible proxy objective.

Parallelism analysis. Speedcode runs and displays the reports generated by the Cilksan race-detection tool and the Cilkscale scalability analyzer. These tools assist students learning parallel programming by making it easier to debug, verify correctness, and understand the scalability of different parallel algorithms and implementation decisions.

IV. Experiences

Our prototype of Speedcode has only a little over 300 registered users, but our early experiences with the platform suggest that it substantially reduces the effort required to teach lessons in software performance engineering and parallel programming. Thus far, Speedcode has been used to run two tutorials, and three classroom assignments at Johns Hopkins University and U.C. Riverside. At MIT, it has been used as a training tool for undergraduate researchers starting projects related to parallel algorithms and performance engineering. The remainder of this section briefly summarizes some of our early experiences with the learning platform.

A. Classroom setting

Speedcode was used for two lectures on parallel computing for data science at Johns Hopkins University. The first lecture was on fork-join parallelism (using Cilk) and the second was on vectorization (SIMD parallelism). Students completed both in-class and take-home assignments using Speedcode. A mix of undergraduates and graduate students completed in-class and take-home assignments over the course of a week using Speedcode. Despite many of the students having little C/C++ development experience, students were able to engage with the exercises immediately using just their personal laptops and tablets without a cumbersome setup process.

Speedcode allowed for new topics and exercises to be introduced into a class that typically operates within the Python programming ecosystem. In prior years, Cilk and fork-join parallelism had not been taught since the burden of setting up the necessary software and infrastructure was not justifiable for a single assignment. Vectorization was taught in prior years, but only theoretically and without any hands-on exercises.

At U.C. Riverside, Speedcode was used for an optional hands-on assignment for a theory-focused class on parallel algorithms. The course covered work-span analysis and algorithmic techniques in parallel programming. Speedcode was used for an optional assignment related to parallelizing a numeric integration algorithm. Although the class was focused on theory, many of the students had prior experience with programming and were familiar with LeetCode. Students were particularly enthralled by the parallel analysis tools provided by Speedcode that allowed for quick detection of race conditions and measurement of parallel scalability. Many mentioned that they had not previously seen such analysis, as much of their prior hands-on experience was with sequential programs.

B. Tutorials and presentations

The Speedcode platform has been used to run a tutorial at ACM SPAA, a parallel algorithms conference, and to present an interactive demo during a presentation to the DOE. The tutorial at SPAA focused on the general capabilities of Speedcode to teach parallel algorithms. The DOE presentation was not specifically focused on Speedcode itself, but rather on the importance of software performance engineering. Here Speedcode was used to provide a hands-on exercise to accompany a case study on the performance of matrix multiplication.

C. Screening and on-boarding

The Speedcode platform has been used to provide exercises to undergraduate students who are interested in doing parallel algorithms and systems research as an MIT UROP (undergraduate researcher). It is not uncommon for faculty and research scientists to assign didactic programming exercises to

prospective undergraduate researchers. Such exercises often serve the dual purpose of a screening tool to identify well-prepared students, and also an on-boarding tool to acquaint students with new research topics and methods.

We have used Speedcode as a screening/teaching tool for new projects related to graph algorithms and systems. Specifically, we use *triangle counting* (TC), a typical graph problem, for recruiting and evaluating potential UROP students. So far more than 30 MIT students have taken this exercise on Speedcode. The TC exercise involves several programming optimizations, which makes it a useful evaluation of a student's prior experience and interest in graph algorithms and systems.

V. Speedcode Curriculum

A major use case of Speedcode is online learning. Presently we are working with university and industry partners to build online SPE curriculum, and are looking to establish an inclusive and collaborative ecosystem for software performance engineering (SPE) education.

Our objective is to develop a Speedcode Curriculum (SCC) in collaboration with communities of educators to enable high-quality and highly-accessible online learning opportunities for topics related to software performance engineering and parallel programming. The SCC will be composed of an organized collection of programming exercises on Speedcode that illustrate critical SPE concepts such as Bentley's rules [6] for optimizing work, cache efficient algorithms, vectorization, data structures, and parallel programming patterns.

We welcome people in the SPE community to join and contribute to the curriculum.

VI. Conclusion

Speedcode aims to facilitate both student self-study of software performance engineering and instructors' introduction of software performance engineering to their students. The prototype of Speedcode is available at http://speedcode.org.

Acknowledgements

Research was sponsored by the United States Air Force Research Laboratory and the Department of the Air Force Artificial Intelligence Accelerator and accomplished 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 Department of the 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.

References

[1] [Online]. Available: https://leetcode.com/

[2] [Online]. Available: https://opencilk.org/

[3] E. Allen, D. Chase, J. Hallett, V. Luchangco, J.-W. Maessen, S. Ryu, G. L. Steele Jr., and S. Tobin-Hochstadt, *The Fortress Language Specification Version 1.0*, Sun Microsystems, Inc., Mar. 2008.

[4] E. Ayguade, N. Copty, A. Duran, J. Hoeflinger, Y. Lin, F. Massaioli, X. Teruel, P. Unnikrishnan, and G. Zhang, "The design of OpenMP tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, pp. 404–418, 2009.

[5] R. Barik, Z. Budimlic, V. Cavè, S. Chatterjee, Y. Guo, D. Peixotto, R. Raman, J. Shirako, S. Taşirlar, Y. Yan, Y. Zhao, and V. Sarkar, "The Habanero multicore software research project," in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA '09. New York, NY, USA: ACM, 2009, pp. 735–736. [Online]. Available: http://doi.acm.org/10.1145/1639950.1639989

[6] J. L. Bentley, *Writing efficient programs*. Prentice-Hall, Inc., 1982.

[7] R. D. Blumofe and C. E. Leiserson, "Space-efficient scheduling of multithreaded computations," *SIAM J. Comput.*, vol. 27, no. 1, pp. 202–229, Feb. 1998. [Online]. Available: http://dx.doi.org/10.1137/S0097539793259471

[8] ——, "Scheduling multithreaded computations by work stealing," *J. ACM*, vol. 46, no. 5, pp. 720–748, Sep. 1999. [Online]. Available: http://doi.acm.org/10.1145/324133.324234

[9] R. L. Bocchino, Jr., V. S. Adve, S. V. Adve, and M. Snir, "Parallel programming must be deterministic by default," in *Proceedings of the First USENIX Conference on Hot Topics in Parallelism*, ser. HotPar'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855591.1855595

[10] R. P. Brent, "The parallel evaluation of general arithmetic expressions," *J. ACM*, vol. 21, no. 2, pp. 201–206, Apr. 1974. [Online]. Available: http://doi.acm.org/10.1145/321812.321815

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[12] D. L. Eager, J. Zahorjan, and E. D. Lazowska, "Speedup versus efficiency in parallel systems," *IEEE transactions on computers*, vol. 38, no. 3, pp. 408–423, 1989.

[13] M. Feng and C. E. Leiserson, "Efficient detection of determinacy races in Cilk programs," in *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, June 1997, pp. 1–11.

[14] S. S. Foley, D. Koepke, J. Ragatz, C. Brehm, J. Regina, and J. Hursey, "Onramp: A web-portal for teaching parallel and distributed computing," *Journal of Parallel and Distributed Computing*, vol. 105, pp. 138–149, 2017.

[15] M. Frigo, C. E. Leiserson, and K. H. Randall, "The implementation of the Cilk-5 multithreaded language," *SIGPLAN Not.*, vol. 33, no. 5, pp. 212–223, May 1998. [Online]. Available: http://doi.acm.org/10.1145/277652.277725

[16] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, pp. 1563–1581, 1966.

[17] T. Kaler, "Programming technologies for engineering quality multicore software," Ph.D. dissertation, Massachusetts Institute of Technology, 2020.

[18] D. Lea, "A Java fork/join framework," in *Proceedings of the ACM 2000 Conference on Java Grande*, ser. JAVA '00. New York, NY, USA: ACM, 2000, pp. 36–43. [Online]. Available: http://doi.acm.org/10.1145/337449.337465

[19] D. Leijen and J. Hall, "Optimize managed code for multi-core machines," *MSDN Magazine*.

[20] C. E. Leiserson, "The Cilk++ concurrency platform," *Journal of Supercomputing*, vol. 51, no. 3, pp. 244–257, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11227-010-0405-3

[21] S.-J. Min, C. Iancu, and K. Yelick, "Hierarchical work stealing on manycore clusters," in *Fifth Conference on Partitioned Global Address Space Programming Models (PGAS '11)*, Oct. 2011.

[22] S. S. Patil, "Record of the project MAC conference on concurrent systems and parallel computation," J. B. Dennis, Ed. New York, NY, USA: ACM, 1970, ch. Closure Properties of Interconnections of Determinate Systems, pp. 107–116. [Online]. Available: http://doi.acm.org/10.1145/1344551.1344561

[23] J. Reinders, *Intel Threading Building Blocks*, 1st ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2007.

[24] 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. New York, NY, USA: Association for Computing Machinery, 2023, pp. 189—-203. [Online]. Available: https://doi.org/10.1145/3572848.3577509

[25] G. L. Steele, Jr., "Making asynchronous parallelism safe for the world," in *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '90. New York, NY, USA: ACM, 1990, pp. 218–231. [Online]. Available: http://doi.acm.org/10.1145/96709.96731