Courses in high-performance computing for scientists and engineers

Richard Vuduc, Kenneth Czechowski, Aparna Chandramowlishwaran, Jee Whan Choi
Georgia Institute of Technology
School of Computational Science and Engineering
Atlanta, Georgia, USA
Email: {richie,kentcz,aparna,jee}@gatech.edu

***Abstract*—This paper reports our experiences in reimplementing an entry-level graduate course in high-performance parallel computing aimed at physical scientists and engineers. These experiences have directly informed a significant redesign of a junior/senior undergraduate course, *Introduction to High-Performance Computing* (CS 4225 at Georgia Tech), which we are implementing for the current Spring 2012 semester. Based on feedback from the graduate version, the redesign of the undergraduate course emphasizes peer instruction and hands-on activities during the traditional lecture periods, as well as significant time for end-to-end projects. This paper summarizes our anecdotal findings from the graduate version's exit surveys and briefly outlines our plans for the undergraduate course.**

***Keywords*-parallel computing; computational science and engineering; education; peer instruction**

I. Introduction and Brief History

The demand for high-performance computing (HPC) among the general science and engineering population at Georgia Tech is on the rise. Figure 1 summarizes this trend, showing how enrollments in just the core HPC classes[1] have steadily increased since the 2006-7 academic year (AY).[2] Where 62 students took two graduate-level courses in AY 2006-7, 146 students are taking four courses in AY 2011-12, including both graduate and undergraduate course offerings.[3] As we explain below, the audiences for these courses have diverse backgrounds and needs. The major question with which this paper is concerned is how to implement the courses in a way that can best meet those needs.

As is true elsewhere, computer-based modeling and simulation plays a prominent role in the science and engineering research and education at Georgia Tech. As such, the new HPC offerings came about in part because of the creation of a new interdisciplinary academic department, the School of Computational Science and Engineering (CSE),[4] which at present offers its own masters and Ph.D. degree programs and faculty lines. This department is distinct from those

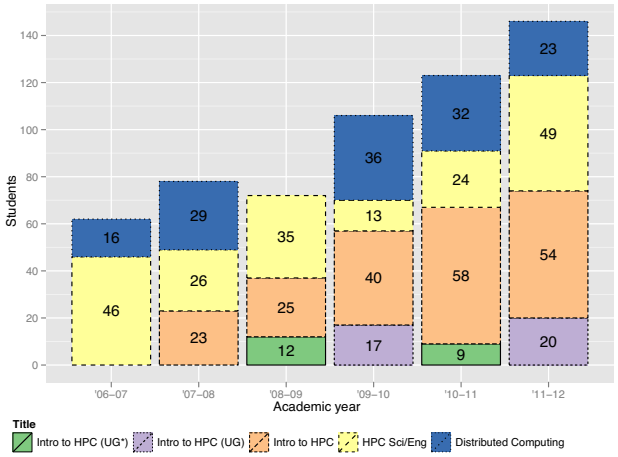

Figure 1. HPC enrollment trends at Georgia Tech. All courses are graduate-level, except *Intro to HPC (UG*)*, an undergraduate section of the corresponding graduate course; and *Intro to HPC (UG)*, a completely separate undergraduates-only course.

in computer science, mathematics, and the physical and biological science and engineering disciplines. Thus, where one graduate HPC course (labeled *HPC Sci./Eng.* in Fig. 1) used to serve all students interested in HPC, there are now two such courses: the same course[5] targeting the broad cross-section of scientists and engineers on campus; and a new *Intro to HPC*,[6] which satisfies a core breadth requirement in the CSE program and whose syllabus has a stronger parallel algorithms theory component. Separate from these courses, the faculty is developing additional transition courses, both to help non-computing specialists learn enough computer science to take HPC and to help computer scientists learn enough parallel numerical and combinatorial methods to apply HPC to disciplinary science.

The overall demand for an HPC skillset is trickling down to the *undergraduate* level. In response, the faculty created in AY 2008-9 a new *Intro to HPC* for undergraduates.[7] This course was a special section of the graduate course

[1] "Core HPC" *excludes* the usual courses in computer architecture, compilers, programming languages, numerical algorithms. It also excludes specialty courses in parallelism, such as a multicore and GPU video game course and a variety of graduate-level topics-driven seminars.

[2] *AY* begins with a Fall semester an ends with a Spring semester.

[3] There is typically a very small overlap of students taking more than one of these four courses.

[4] See: http://www.cse.gatech.edu. The School of CSE was created during Richard A. DeMillo's tenure as dean of the College of Computing [1].

[5] Now known as CSE 6230; see: http://bit.ly/gtcse6230 [2].

[6] Known as CS 6220 and CSE 6220; see: http://www.cc.gatech.edu/~gbiros/teaching/6220-s10.html.

[7] Known as CS 4225; see: http://j.mp/gtcs4225 [3].

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.169

IEEE computer society

in AY 2008-9 and AY 2010-11, and an entirely *separate* course in AY 2009-10 and AY 2011-12. (Faculty availability drives this schedule.) In the current separate offering for Spring 2012, exactly half of the twenty students enrolled are computer science majors; the other half come from physics, materials, chemical engineering, electrical, and aerospace engineering. Notably and perhaps unsurprisingly, enrollment is higher in the separate undergraduate course than the undergraduate section of the graduate offering (in Fig. 1, 17 and 20, versus 12 and 9).

The first three authors used the Fall 2011 graduate *HPC Sci./Eng.* course as an experiment to prototype a variety of educational activities. These activities now form the heart of the current separate Spring 2012 undergraduate HPC course, taught by the first and last authors. This paper summarizes our Fall 2011 experience and how it informed our Spring 2012 plan. In particular, the Spring 2012 course will try to engage undergraduates with intensive hands-on projects, labs, and lectures using peer activities [4], [5], based on the exit-survey results of the Fall 2011 prototype.

Importantly, this paper is only an experience report, rather than any sort of controlled study that proves or disproves a particular parallel computing teaching methodology as superior to some other. Furthermore, we report on a graduate course, from which we have designed the undergraduate course being implemented this semester. We hope that, if invited to participate in the EduPar workshop, we can interact with others on such issues as well as report on the undergraduate version of the course being offered now.

II. Graduate HPC for Scientists and Engineers

The structure of the Fall 2011 graduate course, *HPC: Tools and Applications* (CSE 6230 [2]), departed from the largely traditional lecture + homework + project + exam-based format of previous instances. This course is designed to be accessible to a broad audience of scientists and engineers; as such, it de-emphasizes theoretical aspects covered in the core CSE *Intro to HPC* class in favor of practical topics, such as programming and analysis and tuning of parallel programs. This section reviews the similarities and differences, and analyzes the exit survey given at the end of the course. These results were used to re-design the undergraduate course that we describe in Sec. III.

Regarding demographics of the course, there were a total of 49 students enrolled, about two-thirds of whom were masters degree students. About 60% of these students were from computer science, with the remaining from math, biophysics; electrical, mechanical, aerospace, and nuclear engineering; and operations research. We also gave a "calibration quiz" on the first day of class to get a sense of the students understanding of basic sequential programming, algorithmic complexity, and microprocessor architectures. Although all the students did well on a programming exercise (writing a function in any language), only 20% knew that $n^{1.1}$ grows faster asymptotically than $n \log_2 n$.[8] In addition, less than a third of students were able to answer the question about microprocessor pipelines. These results implied that our course needed to review algorithm analysis and give a crash course on the fundamentals of sequential computer architecture.

A. Similarities to prior offerings

As in the past, the three-credit course met twice per week for 80 minutes each session over a total of 15 weeks. Each student took a final exam (with no other exams) and spent the last third of the course working on an independent project of their own choosing and design, possibly with a partner. The textbook was Levesque's [6], though it served primarily as a reference and supplemental reading material since it does not cover many of the preceding topics. In terms of breadth and depth of topics, the course's scope is largely the same as earlier versions, with the notable exception of omitting parallel I/O and debugging, other than a few cursory mentions.

The first 8-9 weeks focused on "core" topics: *analysis basics*, such as Amdahl's Law, Little's Law, and the notion of strong vs. weak scaling; *algorithm design* including PRAM-style work-span analysis, distributed latency-bandwidth communication model, the external memory model, and computational intensity analysis; *programming models*, including MPI and OpenMP; *parallel architectures*, including distributed memory, shared memory multicore, manycore (i.e., GPU), and multithreaded designs; *single-core architecture*, including pipelining, out-of-order superscalar execution, and cache design; and *low-level tuning techniques*, such as short-vector (SSE) programming. The remaining 6-7 weeks cover advanced topics, which in Fall 2011 consisted of a survey of advanced compiler topics (i.e., the polyhedral model) and experimental programming models, namely, UPC, Coarray Fortran, Chapel, and Concurrent Collections.

B. Differences from prior offerings

There were several key differences from prior instances of the course.

A highly structured "end-to-end" project: The course has always included an independent self-directed project, carried out during the second half of the term, with additional homework assignments during the first half. In this offering, we replaced the homework assignments with the following end-to-end project, carried out in teams of two. This first project, which we refer to as Project 1, asks students to implement a hybrid message-passing and shared-memory matrix multiply that is explicitly tuned for the memory hierarchy and x86 processors, including prefetching

[8] Most students plugged in what they thought was a large value of n. However, the cross-over point does not occur until $n \approx 10^{18}$, which suggests students lack imagination that something might reach *exascale*.

and SSE. (The self-directed independently chosen project became Project 2.) The goal of Project 1 was to give students practical experience in how to approach performance analysis and tuning from top-to-bottom. We chose matrix multiply because the computation is regular and the analysis results relatively well-established, thereby offering a clear way to connect theory and practice—well, in theory at least.

The students began by implementing a basic distributed memory code based on the SUMMA algorithm. The initial naïve distributed SUMMA implementations that they implemented initially ran at a mere 1% of peak. Through a series of staged checkpoints, the students gradually added more and more layers to their codes. By the end, the most successful students had implemented a complete hybrid MPI+OpenMP matrix multiply tuned for the memory hierarchy and underlying processor architecture so as to achieve over 75% of system peak. In addition, about half the class achieved 40-50% or more of peak.

Hands-on labs: The latter part of the class covered experimental programming models. As a matter of philosophy, we felt students would not really develop a concrete sense for the new models unless they were required to try them. However, since the students were supposed to spend their out-of-class time on their self-directed course projects, we felt adding more programming assignments would create too much work.

Instead, we devoted 1 week per programming model, using the first class meeting to give an overview of the model, and using the second class meeting to carry out a "hands-on" lab, in which students would be asked to implement and analyze some computation in the model. During the lab, the three instructors would circulate among the students to answer questions. The first of these labs took place during the second week of class, in order to get students up and running on the cluster, which required learning how to create and submit batch jobs. The remaining labs took place during the last part of the course, and included CUDA/GPU programming, as well as a range of models as mentioned above, which were developed as part of the DARPA HPCS and UHPC programs. Of course, it is hardly possible in two 80-minute sessions to become fluent in a model, especially with only one 80-minute session's worth of actual programming. However, since the students would have at this point already had relatively deep exposure to parallel programming from the end-to-end SUMMA project, we believed these sessions had the opportunity to be quite productive.

PeerWise Q&A: Since the course focused on practical aspects of parallel programming, the only formal and individualized evaluation was the final exam. Thus, we felt that we needed some additional activity to reinforce formal aspects of the course material. We chose to do so using PeerWise, an online system in which students create anonymous multiple choice test questions, and answer questions created by their peers [5]. The online system keeps track of who answers which questions and has a number of abstract awards (e.g., "Most Questions Correctly Answered") to encourage participation. There are also mechanisms that allow each student to identify challenging or interesting questions and to provide anonymous feedback to his or her peers. We asked students to create five such questions during the semester and made these a part of their grade. As an additional incentive to participate and use the PeerWise system to study, we told students of our intent to base about half of the final exam on these questions (suitably modified).

C. Survey responses

The university asks students to complete a standard survey at the end of the course. From our class, 42 of the 49 students completed this survey. On the overall question of, "Considering everything, this was an effective course," students gave the course an overall rating of 4.3 out of 5.0, where a 4.0 or a 5.0 indicate "agree" and "strongly agree," respectively, with no one selecting "disagree" or "strongly disagree."[9] However, this survey is also generic to all university courses and therefore does not indicate what students did or did not like about our course specifically.

Therefore, just prior to submission of Project 2 and the final exam, we also administered our own independent and anonymous survey. Thirty students responded. Of our survey's many questions, the one that best summarizes their overall reaction to the course was, "How much did you learn from ...?" Students answered by indicating positive ("a lot"), moderate ("a fair amount"), or negative ("very little") responses to each of the course's main teaching mechanisms: lectures, hands-on labs, the end-to-end project, the course textbook, and individual interaction with the instructors.

Figure 2 summarizes these responses, showing that the respondents felt overwhelmingly that they learned the most from the end-to-end Project 1. Answers to other questions (not shown) show that students felt Project 1 was by-and-large "very interesting," "highly relevant," and "well-designed," even though they also nearly universally agreed that it was "too much work."

However, Fig. 2 also shows that students perceived labs, lectures, individual interaction with instructors moderately, and by comparison to the project, deemed these mechanisms as decidedly less effective. The textbook was not assessed favorably as a learning tool, though we remind the reader that we used it mostly as a supplement to the course.

Regarding hands-on labs, the students' overall moderate reaction as suggested above may be more nuanced, as Fig. 3 indicates. The vast majority of respondents agreed that the labs were "beneficial," "interesting," "relevant," and "well-designed." However, these respondents also felt more

[9] Values for other courses that semester at the university were not available for comparison at the time of this writing.

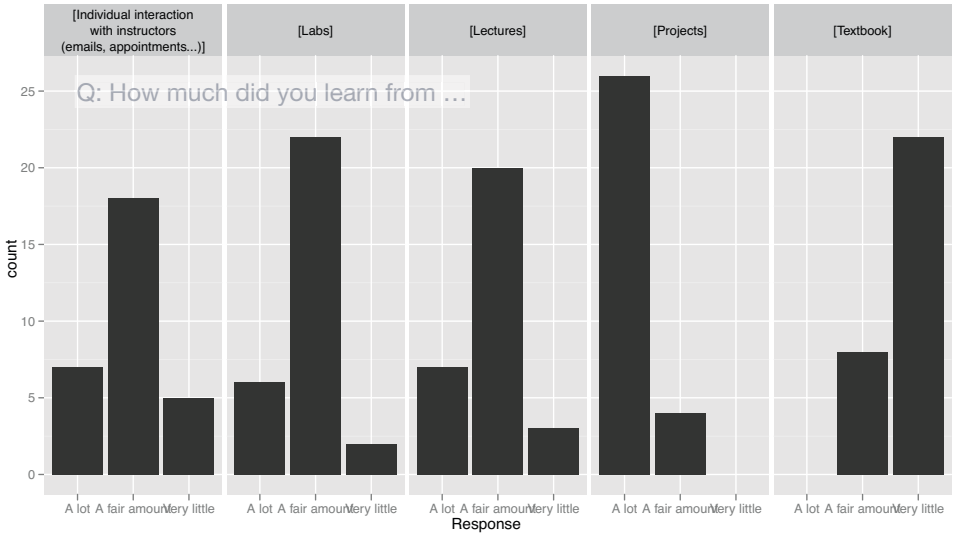

Figure 2. Survey results: *How much did you learn from* ..., on a 3-point scale of "A lot," "A fair amount," and "A little."

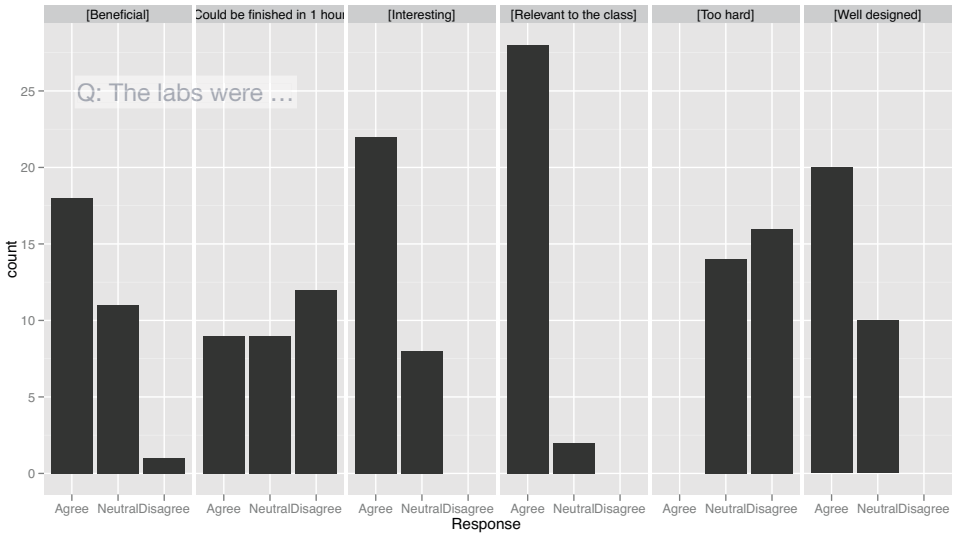

Figure 3. Survey results, overall lab evaluation: *The labs were* ..., on a 3-point scale of "Agree," "Neutral," and "Disagree."

neutral or even negative on whether the labs were either too hard or could be completed within the lab period. We also asked about specific lab activities (not shown). Students gave strongly positive responses to the labs on MPI and CUDA; slightly positive responses to labs on PolyOpt[10] and UPC; and slightly negative responses to labs on Chapel and Concurrent Collections. The latter two involved little or no programming; instead, students only performed some analysis (both reading of code and running benchmarks), which perhaps dampened these labs' appeal.

[10]A source-to-source parallelization and locality-enhancing compiler based on the polyhedral models. See: http://www.cse.ohio-state.edu/~pouchet/software/pocc/

D. Project vs. the final exam

The surveys indicate that the students appreciated the projects, especially the end-to-end Project 1. Indeed, the Project 1 implementations achieved very good performance as noted previously, and the grade assigned for Project 1 was based partly on the actual fraction of peak achieved. An interesting question is whether this experience with Project 1 is also reflected in the course's main individual assessment mechanism, namely, the final exam. The final exam questions emphasize conceptual and theoretical ideas, asking the students to generalize beyond the programming-oriented projects.

Figure 4 compares the grade received on Project 1 (x-axis)

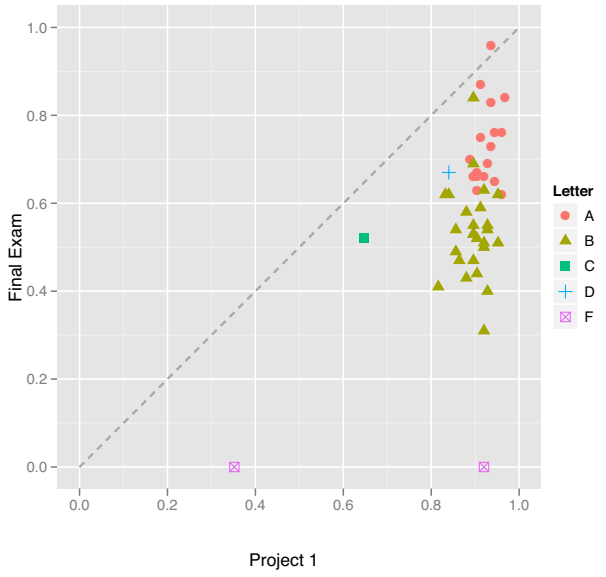

Figure 4. Comparison of Project 1 grades (x-axis) with those of the final exam (y-axis). Each point represents an individual student. The markers indicate the final letter grade assigned for the course. The 'F' grades actually represent incomplete grades due to extenuating circumstances at the time of the final.

with the final exam score (y-axis). The mean on the final was about a 60%. Had the Project 1 and final exam grades been perfectly correlated, they would lie parallel to the diagonal line. However, we see only a weak relationship (correlation coefficient is 0.5). This lack of strong correlation owes partly to the fact that the Project 1 grades are compressed into a narrow range, mostly between 80% (an equivalent C+/B- letter grade) and 100% (A+). In any case, we do not have an explanation for the Project 1 vs. final exam grade discrepancy. However, we do note anecdotally that several students who came to look at their exam results commented that, since there were no other tests or quizzes during the semester, they felt a little surprised by the questions and at how relatively poorly they performed. Though we cannot explain this occurrence, it is clear that the perceived value of the hands-on experience simply did not translate directly into a mastery of the more abstract concepts.

III. Intro to HPC for Undergrads

The results summarized in Sections II-C and II-D implied a number of considerations for the undergraduate HPC course, which we are implementing in the current semester at the time of this writing (Spring 2012). (The syllabus is available at: http://j.mp/gtcs4225.)

- Students highly valued the experience from the end-to-end project. Thus, a strong hands-on component is desirable though care is needed to ensure it is not an excessive amount of work.
- However, hands-on experience did not translate directly into a mastery of abstract or higher-level concepts. Thus, more direct reinforcement of such concepts is necessary.
- The students were not very interested in the experimental programming models. Perhaps it is because we covered too many and none in any depth. Thus, focusing on just a few models but providing more experience and opportunity to use them might be a better approach.
- The students did not perceive above-average value from lectures and individual interaction with the instructors. Thus, mechanisms to increase this value are warranted.
- Students perceived little value in the textbook, which we emphasize we had used only in a peripheral way. If we require a textbook at all, it should be better and more directly integrated into the course.

From these observations, we have designed the undergraduate course to reuse the positive aspects of the graduate course while addressing its shortcomings.

The logistics of the course and background of its students as of this writing are as follows. The enrollment is capped at 20 students and is currently full. The course is worth three-credit hours and meets three days per week for 50 minutes each day. The class has only undergraduates, half of whom are computer science majors and the rest of whom come from other science and engineering disciplines. Based on an entry survey, only two have had any prior exposure to parallel computing.

Regarding the course design, we first are choosing to preserve the strong hands-on focus of the graduate course. However, we use *extended labs* to accomplish this goal. In this format, we devote one day of class time per week to a hands-on lab assignment, part of which is due at the end of class and the rest of which is completed as "homework." Thus, students are essentially forced to start assignments in-class and therefore keep better pace with the course material. Also, during the in-class time, we circulate and guide students through directed exercises. By devoting in-class time to this activity, we hope to increase the frequency and quality of face-to-face interaction time between the students and ourselves, as well as reduce the perception that the programming assignments require "too much work" outside of class.

Students of the graduate course cited the end-to-end experience of their Project 1 assignment as particularly valuable. Thus, we have sequenced the extended lab topics in a similar way, starting with message passing algorithms and codes to which we add shared memory, memory hierarchy optimizations, GPU acceleration, and low-level CPU and/or GPU tuning. We also spend several extended labs working with one programming model, to help increase the depth of experience in that model. Our programming model choices

this term are MPI, OpenMP, Cilk Plus, and CUDA, with about four weeks of instruction in each.

To increase the value of lectures and reinforce high-level abstract concepts, we use a variant of the peer instruction format [4]. In this format, we give mini-quizzes, or *checkpoints*, during certain points in each lecture. Students first submit an answer to a quiz question individually; they then have an opportunity to discuss their answer with their neighbor. We allow the students to resubmit their answer based on the peer discussion. These quizzes happen online in real-time, so that the instructors can monitor the responses and adjust the lecture accordingly to address confusing material. The checkpoints are also graded to create an incentive to pay attention during class and actually try to get the right answer.

To facilitate even more communication among the students as well as with the instructors *outside* of class, we are using Piazza[11] as our online discussion forum.

Lastly, we have adopted the textbook by Hager and Wellein [7]. This textbook has more material directly of use in the course, based on the course's syllabus, and addresses some criticisms students expressed in the graduate course. Though the undergraduate class has not in years past had a textbook, students in the graduate class expressed a desire to have an organized offline reference.

IV. Relation to the NSF/TCPP Initiative?

The preceding discussion does not shed direct light on how to approach teaching of parallel computing concepts in the core computing curriculum, as outlined in the NSF/TCPP Curriculum Initiative (CI for short) [8].

That said, at least one aspect of our course is worth noting in relation to the CI: our course may actually "invert" several of the ordering and Bloom classifications of topics that the CI implies. In particular, we began with quick introductions to concepts of work and span for algorithm design, but immediately after that—starting in the second day of classes—we introduced distributed memory algorithms. Distributed memory exposes the need for *explicit* communication. This choice inverts the usual approach of focusing on shared memory parallelism, which hides communication and is therefore perceived as "easier" than distributed memory and (later) memory hierarchy-aware programming. However, it is our opinion that establishing strong foundations in explicit communication and locality is as important (if not more so) as parallelism itself. We are trying this approach in the undergraduate course as well.

Acknowledgment

We thank Richard M. Fujimoto, Chair of the School of CSE, for his overwhelming support and encouragement of innovative and experimental educational activities; as well as David A. Bader, George Biros, Edmond Chow, Jason Riedy, Karsten Schwan, Jeffrey Vetter, and Matthew Wolf, all of whom are an integral part of the core HPC research and education at Georgia Tech. Wendy Newstetter gave us a number of useful ideas from her problem-based learning activities, and Donna Llewellyn arranged access to innovative classroom spaces on campus to support peer instruction. Logan Moon, David Mercer, and Randy Carpenter all deserve additional thanks for the hard work of administering an experimental local CPU/GPU cluster resource for our classes. We also thank the anonymous reviewers for their constructive feedback. Lastly, this work was supported in part by the National Science Foundation (NSF) under NSF CAREER award number 0953100 and NSF TeraGrid/XSEDE allocations CCR-090024 and ASC-100019. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF.

[11] http://www.piazza.com

References

[1] R. A. DeMillo, *Apple to Abelard: The Fate of American Colleges and Universities*. MIT Press, 2011.

[2] R. Vuduc, K. Czechowski, and A. Chandramowlishwaran, "CSE 6230: High-performance computing tools and applications," Atlanta, GA, USA, Fall 2011. [Online]. Available: http://bit.ly/gtcse6230

[3] R. Vuduc and J. W. Choi, "CS 4225: Introduction to high-performance computing," Atlanta, GA, USA, Spring 2012. [Online]. Available: http://j.mp/gtcs4225

[4] C. H. Crouch and E. Mazur, "Peer Instruction: Ten years of experience and results," *American Journal of Physics*, vol. 69, no. 9, p. 970, 2001. [Online]. Available: http://link.aip.org/link/AJPIAS/v69/i9/p970/s1&Agg=doi

[5] P. Denny, J. Hamer, A. Luxton-Reilly, and H. Purchase, "PeerWise," in *Proceedings of the 8th International Conference on Computing Education Research - Koli '08*. New York, New York, USA: ACM Press, 2008, p. 109. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1595356.1595378

[6] J. Levesque and G. Wagenbreth, *High-performance computing: Programming and applications*. CRC Press, 2010.

[7] G. Hager and G. Wellein, *Introduction to high-performance computing for computational scientists and engineers*. CRC Press, 2010.

[8] A. Chtchelkanova, S. Das, C. Das, F. Dehne, M. Gouda, A. Gupta, J. JaJa, K. Kant, A. La Salle, R. LeBlanc, A. Lumsdaine, D. Padua, M. Parashar, S. Prasad, V. Prasanna, Y. Robert, A. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. Weems, and J. Wu, "NSF/TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates," NSF/TCPP, Tech. Rep., 2010. [Online]. Available: http://www.cs.gsu.edu//~tcpp/curriculum/http://www.cs.gsu.edu/~tcpp/curriculum/sites/default/files/NSF-TCPP-curriculum-Dec23.pdf