Joint UIUC/UMD Parallel Algorithms/Programming Course (Extended Abstract)

David Padua
Department of Computer Science
University of Illinois at Urbana-Champaign
padua@illinois.edu

Uzi Vishkin
The University of Maryland Institute for Advanced Computer Studies (UMIACS)
vishkin@umd.edu

Jeffrey C. Carver
Department of Computer Science
University of Alabama
carver@cs.ua.edu

***Abstract*—This extended abstract reviews an education experiment conducted through shared teleconferencing sessions between a University of Illinois course on Parallel Programming for Science and Engineering Majors and a University of Maryland on Parallel Algorithms in fall 2010, as well as shared programming assignments. The students were given the opportunity to compare OpenMP programming on an 8-processor SMP machine with PRAM-like programming using a 64-processor XMT machine.**

Keywords: parallel computing education, OpenMP, PRAM, XMT, XMTC.

I. INTRODUCTION

Power constraints forced the computing industry to bet future performance growth on parallelism, though it remains unclear how commodity parallel general-purpose computers of the future will be built and programmed for performance. Programmers of today's parallel machines must overcome several 'productivity busters' beyond just identifying operations that can be executed in parallel: (i) impose the Culler-Singh [2] 4-step programming-for-locality recipe: decomposition, assignment, orchestration, and mapping, which is often difficult; (ii) reason about concurrency, including race conditions, in threads; (iii) for machines such as GPU, that fall behind on serial (or low parallelism) code, whole programs must be highly parallel. Moreover, according to [1] (as well as [3]), only hero programmers succeed exploiting the vast parallelism in today's machines, leading to a quest for new computing stacks. An education agenda needs to recognize and adapt to this reality.

Parallel computing exists for providing speedups over serial computing. Its democratization mandates that the general body of computer science students and graduates will be capable of achieving good speedups. If a general-purpose computer could be programmed effectively by too few programmers, or requires excessive learning, application software development would be prohibitively expensive, greatly weakening the market potential of such a computer. Motivated by the observation that education is a facilitator, a testbed as well as a benchmark for such capability, we devoted nearly half of a fall 2010 semester for joint teleconferencing sessions teaching OpenMP and XMTC, and comparing them. OpenMP is a standard programming platform for several current parallel machines. PRAM is a parallel algorithms theory developed mostly during the 1980s. This rich theory has been criticized as presenting an overly simplistic abstraction of parallel architectures. Explicit Multi-Threading (XMT^i) is a many-core architecture designed to provide efficient hardware support for XMTC, a PRAM-like programming language. It is worth mentioning that architectural support for parallel programming has long been a topic of interest for workshops such as HIPS. ISCA'11 includes for the first time a workshop devoted to such future support. Yet, both require greater clarity on which parallel programming approaches are actually desired.

The UIUC course was titled Parallel Programming for Science and Engineering Majors and most of its students were non-CS majors, with a big variance of backgrounds. The UMD course was titled Parallel Algorithms. The students who took it for credit were mostly entering Electrical and Computer Engineering graduate students with a mix of backgrounds.

David Padua (DP) taught OpenMP programming. His teaching provided parallel architecture knowledge needed for OpenMP programming. Uzi Vishkin (UV) taught parallel (PRAM) algorithms with about 20 minutes devoted to XMTC programming. Most of the remaining sessions at UIUC, not shared by UMD students, were devoted to MPI. UIUC students also submitted more OpenMP programming assignments. The remaining sessions at UMD were devoted to more parallel algorithms. UMD students did a significant amount of dry homework related to the design and analysis of parallel algorithms and submitted a more demanding XMTC programming assignment.

Helping to design the experiment, Jeffrey Carver (JC) also administered an anonymous questionnaire filled by the students. The questionnaire was accessed by DP and UV only after all grades were posted, per IRB guidelines. The questionnaire provided the following feedback:

- All responding students, but one, wrote that XMTC comes ahead of OpenMP for achieving speedups. The actual speedup results support this XMTC advantage. For example, *none* of the 42 students in the joint course got any speedups using OpenMP programming on a simple irregular problem (breadth-first search on graphs) using an 8-processor SMP. However, these students got speedups in the range *7X-25X* on a 64-processor XMT machine built using field-programmable gate-arrays (FPGA) technology. This comparison makes sense since the silicon area requirement of the XMT design is under that of 2 SMP processors.
- An interesting split between the UIUC and UMD students was around how they judged the help of PRAM algorithms for XMT programming. UMD students felt strongly that PRAM algorithms helped considerably with XMT programming, while most Illinois students felt otherwise. Recall that the exposure of UIUC students to PRAM algorithms and XMT programming was much more limited, and their understanding of this material was not challenged through either analytic homework, or exams. When faced with the same programming challenges, the performance of UIUC and UMD students was similar. Pedagogically, this may demonstrate that students must be exposed to a minimal amount of parallel algorithms and their programming, and be properly challenged on their analytic understanding, in order to internalize their merit. If this conclusion is valid, it creates tension with the pressure on instructors of parallel computing courses to cover several programming paradigms along with their required architecture background.

II. Other impressision and lessons for repeating the joint experiment

1. Due to its higher level of abstraction, and its focus on algorithms but not their programming, the PRAM part of the joint course was able to convey algorithms for more advanced problems than the other parts, though, as explained above, the understanding of this part was only tested on the UMD students. 2. When we started planning the experiment we were already assigned to teach our respective courses. The experiment confirmed that it would be better to do such experiments on more homogenous populations, starting with a CS graduate course. 3. Due to the IRB approval process, and the requirement to submit all questionnaires prior to the beginning of the course, it is important to complete the full planning of the course and all its homework, prior to finalizing the questionnaire.

III. Conclusion

Parallel computing platforms seek to succeed the serial platform, but so far with limited success and promise [1, 3]. Our joint effort aims to develop ways for whetting the appetite of students for learning about parallelism through their first experience. This experience got to be both meaningful and positive. Moreover, we believe that success that students have with achieving good performance, especially if it does not involve excessive effort on programming and debugging, could make a lasting impression in their mind and attract them to seek doing more of it after they graduate. On the other hand, a traumatic experience with parallel programming and very limited speedups, or none at all, are likely to have the opposite effect. We also hope that as we and others continue to develop these concepts, new stacks that embrace these concepts will emerge and, in due course, vendors will buy into them.

Course homepages

https://agora.cs.illinois.edu/display/cs420fa10/Home and http://www.umiacs.umd.edu/users/vishkin/TEACHING/ence459p-f10.html

For a summary of the education aspects of the PRAM/XMT approach, see [4]; references therein include teaching experience extending from middle school to graduate courses, course material including class notes and programming assignments, video presentations of a full-day tutorial and a full-semester graduate course, a software platform (comprising a compiler and cycle-accurate simulator) available for free download, and the XMT hardware.

References

[1] S.H. Fuller and L.I. Millett. "The Future of Computing Performance: Game Over or Next Level? The Computer Science and Telecommunications Board's (CSTB) of the U.S. National Academies, December 2010. This report contrasts the opportunities of parallel computing becoming mainstream with today's reality where only hero programmers manage to exploit the vast parallelism of current commodity hardware.

[2] D. Culler and J. Singh. Parallel Computer Architecture: A Hardware/Software Approach. Morgan-Kaufmann, 1999.

[3] Patterson, D. The trouble with multi-core: Chipmakers are busy designing microprocessors that most programmers can't handle. *IEEE Spectrum* (July 2010).

[4] U. Vishkin. Algorithms-based extension of serial computing education to parallelism, extended abstract. Workshop on Integrating Parallelism Throughout the Undergraduate Computing Curriculum, in Conjunction with PPoPP 2011, San Antonio, Texas, February 12, 2011 http://www.umiacs.umd.edu/users/vishkin/XMT/PPOPP-CPATH2011.pdf

[i] Not to be confused with the Cray XMT