Parallel Computing in the Computer Science Curriculum via the Computer Architecture Course

Mark Fienup
Computer Science Department
University of Northern Iowa
Cedar Falls, Iowa 50614-0507
fienup@cs.uni.edu

Abstract

This paper describes the author's experience with incorporating parallel computing topics into the Computer Science curriculum at the University of Northern Iowa via the Computer Architecture course. It details the curricular and pedagogical issues including course goals, parallel programming tools, and techniques for teaching parallel programming topics.

1 Introduction

The University of Northern Iowa (UNI) is a comprehensive region university that offers BA and BS Computer Science degrees, but has no engineering programs. Historically, the only Parallel and Distributed Computing (PDC) topics in the Computer Science curriculum was an introduction to concurrent programming as part of the elective Operating Systems course. With the advent of widely available multi-core CPUs and many-core GPUs, traditional sequential programming skills are clearly not enough for undergraduate Computer Science students. This paper describes the author's experience with incorporating parallel computing skills into the Computer Science curriculum via the Computer Architecture course.

Ideally, PDC topics would be integrated throughout the Computer Science curriculum, but that would require a major redesign of the curriculum with consensus and cooperation from all faculty members. Even adding a new, dedicated PDC course to the already full CS curriculum was not an option. As the primary instructor for the existing Computer Architecture (CS 2420) course this author was able to substantially revise its content so that the latter two-thirds of the course was dedicated to parallel computing while reinforcing Computer Architecture concepts introduced in the first third of the course.

2 Curricular Issues

The Computer Architecture course at UNI has two prerequisite courses: (1) Introduction to Computing (CS 1510) and (2) Computer Organization (CS 1410). The Introduction to Computing course is a four-hour beginning programming course with a weekly lab that is offered in either Python or Ada. However, most Computer Architecture students will have had the Data Structures course and our Intermediate Computing course which focuses on object-oriented design using Java. The Computer Organization course content is roughly split into thirds: (1) data representation and digital logic including common computer circuits (i.e., decoder, MUX, adders, flip-flops, register file, RAM memory, control unit), (2) assembly language programming, and (3) hardware support for the operating system.

The course description of the traditional Computer Architecture course is as follows:

CS 2420. Computer Architecture -- 3 hrs.

Basic concepts of computer architecture with special focus on principles underlying contemporary uniprocessor design. Interaction of hardware and software, and consideration of efficient use of hardware to achieve high performance. Topics include instruction set design, processor design, pipelining, the memory hierarchy, design trade-offs, I/O systems, performance measurement, and multiprocessors.

The main philosophy of the course has always been that knowing how the hardware works enables the programmer to write more efficiency software. However, it was traditionally taught with a computer engineering focus.

The revised Computer Architecture course condenses the computer engineering topics to the first third of the semester with the remaining two-thirds focused on hands-on parallel programming using widely available tools: (1) pthreads on multi-core computers, (2) MPI on a cluster, and (3) CUDA programming on GPUs. The parallel programming examples helped to drive home the concept that "knowing how the hardware works enables the programmer to write more efficiency software." The weekly schedule of topics during the first offering in Fall 2013 is shown in Table 1.

Week	Major Topics Covered
1	Review of von Neumann architecture Hardware support for the operating system
2	Memory hierarchy – cache
3	Virtual memory – paging and segmentation
4	Pipelining, superscalar, and multi-core architectures
5	Flynn's taxonomy, shared vs. distributed memory machines Interconnection networks
6	Review of above and Test 1
7	C and Linux overview Parallel programming design – task vs. data parallelism
8	Pthread introduction – sum 1D array of floats Common synchronization patterns using mutexes and condition variables
9	Concurrent shared data structures and deadlock Static vs. dynamic allocation of work
10	Pthread textbook examples: n-body and Traveling Sales Person (TSP) problems MPI introduction – sum 1D array of floats
11	Common synchronization patterns using MPI group communication functions MPI textbook examples: n-body and TSP problems
12	Review of pthreads and MPI and Test 2
13	CUDA Architecture and count 3s in 1D array of integers and sum 1D array of floats
14	2D Successive Over-Relaxation (SOR) pthread and MPI homework assignment comments 2D SOR on GPU design and implementation discussion
15	n-body and TSP GPU implementation discussion Review for Final Examination

Table 1: Computer Architecture Course Weekly Schedule.

3 Pedagogical Techniques Utilized in the Course

The pedagogical techniques described below are not especially novel, but the author has found them extremely useful to aid student learning.

3.1 Active-Learning in the Classroom

First, class-time was not spent lecturing. Instead, active-learning techniques were utilized as described in [1]. Basically, "mini"-lectures of about 10 minutes were followed by small groups of students trying to answer questions using the mini-lecture material. Whole-class discussion followed each group activity to discuss answers and make sure all students understood the material correctly. This has been shown to reset the students' attention span, so the next mini-lecture can be effective [2]. The questions discussed are available at the course website: http://www.cs.uni.edu/~fienup/cs2420f13/.

3.2 Common Examples Across Each Parallel Programming Environment

The second pedagogical technique used was introducing the three parallel programming environments (pthreads, MPI, and CUDA for GPUs) using the same simple problem: summing a one-dimensional array of floats. This technique has the advantages of:

- allowing students to focus on aspects of the parallel programming environment without the difficult of also understanding a complex problem, and
- allowing students to understanding the similarities and differences of each parallel program environment.

The course textbook [3] further leveraged this second advantage by discussing solutions to two larger problems (n-body problem and traveling sales-person problem) using several parallel programming environments including pthreads and MPI. The textbook had no coverage of GPU programming, but possible solutions to these problems were outline during week 15 of the course.

The parallel programming projects over the three parallel programming environments was also a common task: 2D Successive Over-Relaxation (SOR) (often used in 3D form to solve differential equations such as Navier-Stokes equations for fluid flow). Detailed descriptions are available at: http://www.cs.uni.edu/~fienup/cs2420f13/homework/.

4 Parallel Programming Tools

In the not too distance past, introducing parallel programming into the curriculum would have required specialized hardware or remote access to a national Super Computer center. Students might have viewed parallel programming as a specialized skill rarely used in the

"real" world. Now with the advent of widely available multi-core CPUs and many-core GPUs, local hardware can be used with freely available parallel programming tools. Clearly, students are more motivated if they view parallel programming as a relevant, real-world programming skill.

The pthread library was chosen for its wide availability and its shared-memory parallel programming paradigm. Students in the course had a choice of remotely accessing a multi-core student server for pthread programming or using a lab computer with an Intel i7 processor which could run eight threads (four cores with dual SMTs) in parallel. The standard g++ compiler with the pthread library were utilized.

The MPI library was chosen for its wide availability and its message-passing distributed-memory parallel programming paradigm. Students in the course remotely accessing a multi-core Linux server running a collection of virtual machines configured as a MPICH cluster. Ideally, a network of workstations might have been a more realistic MPI environment, but this configuration was easy to setup and manage.

Finally, CUDA programming for GPUs was chosen for its wide availability and its SIMD parallel programming paradigm on the many-core GPU combined with heterogeneous programming of the host processor. Students in the course remotely accessed a Linux server containing a collection of GPU devices (several Tesla C2070 cards and a Tesla C1060 card). This configuration eliminated the need of dedicated computers each with programmable GPU cards.

While programming with multiple of these environments simultaneously is possible, and even desirable, time constraints limited the course to considering each parallel programming environment separately.

5 Computer Architecture Topics Reinforced via Parallel Programming Examples

The traditional Computer Architecture course utilized predominantly pencil-and-paper homework exercises to illustrate concepts. The "new" version allowed for more concrete examples from the parallel programming portion of the course to illustrate Computer Architecture concepts. Some examples are listed in Table 2.

Computer Architecture Topic	Reinforcing Parallel Programming Example
Cache hit rate	Block vs. cyclic data allocation of 1D arrays
False-sharing in cache line	Pthread id i sum into threadSums[i]
Virtual memory	Pinning an array on the host to speed cudaMemcpyDeviceToHost
Performace Analysis	Parallel speedup and Amdalh's law

Table 2. Computer Architecture Topics Reinforced via Parallel Programming Examples

6 Course Evaluation

Evaluation of the modified Computer Architecture course against the traditional Computer Architecture course proved difficult since the modified Computer Architecture course condenses the traditional course's content to the first third of the modified course. The latter two-thirds of the modified course teaches parallel programming which was not covered in the tradition course. Thus, we are comparing "apples" and "oranges". The best metric between the two versions of the Computer Architecture course is arguably the DWF (D-grade, Withdrawal, or F-grade) rates which correlate with mastery of "related" content, but also hints at student motivation to master the content. Table 3 summaries the DWF rate of both versions of the course.

Computer Architecture Version	**Enrollment**	**DWF Rate (#Ds, #Ws,#Fs)**
Tradition Course without Parallel Programming (Fall 2008 to Spring 2012)	91	23.1% (1 D, 6 Ws, 14 Fs)
Modified Course with Parallel Programming (Fall 2013)	37	2.7% (0 Ds, 0 Ws, 1 F)

Table 3. DWF rates for both versions of the Computer Architecture Course

As Table 3 shows the DWF rate for the modified Computer Architecture course with parallel programming was substantially better, but the sample size it too small to draw any statistically significant conclusions.

7 Conclusions and Future Improvements

Overall the modified Computer Architecture course with parallel programming content was a success from the author's perspective. Previously no parallel programming content was being taught in the Computer Science curriculum at the University of Northern Iowa, so devoting two-thirds of the Computer Architecture course to parallel programming is a substantial improvement. The trade-off of a condensed coverage of the traditional Computer Architecture topics is more than offset by parallel program skills on widely available multi-core CPUs and many-core GPUs.

Acknowledgment

The author would like to thank his colleague Dr. Paul Gray for setting up and administering the MPI and GPU servers used during the Fall 2013 Computer Architecture course detailed in this paper.

References

[1] M. Fienup, "Active and group learning in the computer architecture classroom," Proceedings of the 2000 Workshop on Computer Architecture Education (WCAE '00), ACM, 2000, article no. 12, doi: 10.1145/1275240.1275256.

[2] M. Fienup and J. P. East, "Improving computer architecture education through the use of questioning," Proceedings of the 2002 Workshop on Computer Architecture Education (WCAE '00), ACM, 2002, article no. 7, doi: 10.1145/1275462.1275473.

[3] P. S. Pacheco, An Introduction to Parallel Programming. Morgan Kaufmann Publishers, Burlington, MA, 2011.

[4] Computer Architecture course Fall 2013 URL: http://www.cs.uni.edu/~fienup/cs2420f13/