Parallelism with FPGAs in a Computer Systems & Organization Course

Kevin A. Walsh

Department of Mathematics and Computer Science
College of the Holy Cross, Worcester, MA
kwalsh@cs.holycross.edu

***Abstract*—In an effort to improve student understanding and retention of parallel and distributed computing concepts, we incorporated elements of the NSF/TCPP curriculum into the fall 2012 offering of Computer Systems and Organization at College of the Holy Cross. The most significant change was the introduction of a small hardware design project in which students first design a simple, special-purpose cell processor datapath using a software circuit simulator, and then later instantiate their designs using hardware FPGA prototype boards. This paper describes this design project, its impact on other aspects of the course, and the context in which they were introduced, and a preliminary evaluation.**

Keywords—parallelism; multiprocessors; computer organization; computer systems.

I. INTRODUCTION

A well prepared student of computer science must have full comprehension of parallel computing concepts and be able to apply these concepts in their work. This need is widely recognized even by students: they increasingly rely on cloud computing and other distributed systems, they see the widespread adoption of multi-core and multi-processor architectures, and they hear of large data sets and computational challenges that are best approached using parallel and distributed programs.

As is typical for a small liberal arts college, the College of the Holy Cross follows a curriculum in which students of computer science are exposed to parallel concepts primarily in *Operating Systems* and other upper-level elective courses. In fall 2012 we introduced elements of the NSF/TCPP curriculum[1] into *Computer Systems and Organization* (CSCI 226), a course that students take early in their progression towards a degree. This course is offered each fall semester and is required of all computer science majors and minors. It is typically taken directly after two introductory courses, one on programming and the other on data structures. By emphasizing parallel concepts early, it is hoped that students will be better able to apply these concepts later in their studies.

The revised curriculum for CSCI 226 uses concurrency as a theme throughout the course. Compared to previous offerings, we covered topics that emphasize "parallel thinking" more thoroughly and in greater depth: pipelining, super-scalar and multi-core architectures, co-processors, and hardware primitives to support concurrency. As part of this re-orientation of the curriculum, a number of topics were covered with less emphasis than in previous semesters: mechanical hard disks, the assembly language layer, assemblers and linkers, and CPU micro-code translation. The revised curriculum, along with course materials, assignments, and project notes, has been made available online.[2]

II. FPGA DESIGN PROJECT

Among the curriculum changes introduced in CS 226, the most significant was the introduction of a 1.5 week project in which students design a simple single-purpose "cell processor" comprising a simple datapath and state machine. The theme of the project was based loosely on cryptography—the project specification called for students to design a processor that performs a naïve search for factors of a target integer, i.e. as might be derived from the public half of a key pair—but other themes would likely work equally well. Initially, students used Logisim[3], a software simulator to design and test both single- and multi-cell configurations of their processor, and they used hand calculations to compare the performance of both designs. Later, students worked in groups to realize their designs using Altera's Quartus software and Altera DE0 FPGA prototype boards.

The project incorporates several parallel concepts directly: concurrent execution of different datapath elements, in this case an adder and a divider; concurrent execution of independent cell processors in the multi-cell configuration; arbitration for access to shared resources, in this case control and data buses; and workload partitioning across multiple cells. In this way, students gained hands-on experience with concurrency at three levels: across elements of a datapath, across processors, and across tasks.

Moreover, the project was chosen because it provides natural motivating example for a variety of parallel concepts, and it was designed specifically so it could be offered very early in the semester and subsequently used as a running example in later topics. Once students had covered standard pipelining and CPUs, we revisited the cell processor to drive

[1] Funding was provided by the NSF/TCPP Curriculum Initiative Early Adopters Fall 12 program, which we gratefully acknowledge.

[2] Available at: http://mathcs.holycross.edu/~csci226/

[3] Available at: http://ozark.hendrix.edu/~burch/logisim/

a discussion of instruction-level parallelism, super-scalar processors, multi-core, and multi-processors. For example, a unit on multi-processor architectures was introduced by simply replacing the cells with a MIPS CPU we had designed in class and adding a shared memory to the system. Similarly, SIMD was introduced by replacing the independent state machine control logic for the cells with a single shared state machine to control multiple cell datapaths. A discussion of how the multi-cell configuration of the project would be driven by a master CPU in practice led naturally to a discussion of GPUs, workload offloading, and vector coprocessors.

During the hardware portion of the project, students controlled their cell datapaths manually using a series of toggle switches and buttons connected directly to the shared communications bus. This experience led naturally to a discussion of bus mastering. During a later unit on memory technologies and system busses, the concepts of address-based chip select signals and shared data busses were already familiar to students from their earlier work with the cell processor.

III. Discussion

Using industry design tools is difficult for a beginning student, especially since we do not teach the types of hardware description languages that are well-supported by industry software. Instead, students in CSCI 226 rely purely on graphical design entry. In fact, our prior experience—confirmed by initial testing by a course teaching assistant—has been that graphical design entry, testing, and debugging with industry tools is inconvenient and frustrating for students new to hardware. This was the motivation for us to provide a student-friendly circuit simulator like Logisim for the initial processor design, test, and debug phases. Only after students had a working design did they duplicate their finished design within a more traditional industry-standard design software package. This approach was surprisingly successful—most student groups completed the entire hardware portion of the project in less than two hours, with only minimal help from course staff.

Based on student feedback, the project was successful in increasing student understanding of parallel computing concepts. After building a single-cell machine, students were very quick to realize the performance benefits of using a multi-cell design because of the obvious parallelism inherent in the original computation. This led to a discussion of which types of problems have parallelism that is easily exposed. In the software circuit simulator, the need for a control and data bus is not obvious because adding additional control wires is trivial. However, when students built FPGAs-based versions of the cell processor, it was immediately clear that a shared bus to connect the processors would reduce wire counts and simplify the design of the interconnect.

Evaluation of the curriculum changes described in this paper is ongoing. In particular, we look forward to exploring the impact that earlier exposure to parallel computing concepts may have on student achievement in subsequent courses. Early anecdotal results from a spring 2012 operating systems course indicates that those students having taken CSCI 226 with the revised curriculum are better prepared to study concurrency and synchronization issues at the operating system and higher levels of the software stack.