ToUCH: Teaching Undergrads Collaborative and Heterogeneous Computing

Apan Qasem
Texas State University
apan@txstate.edu

David P. Bunde
Knox College
dbunde@knox.edu

Philip Schielke
Concordia University Texas
philip.schielke@concordia.edu

ABSTRACT

To achieve aggressive performance and power goals, HPC systems are increasingly heterogeneous. This represents an educational challenge since few current curricula include much about heterogeneous computing except possibly in upper-division electives. This document presents plans for a module-based approach to teaching heterogeneous computing to undergraduates.

1. MOTIVATION

The need for increased performance/watt and the demands of processing diverse workloads have triggered an industry shift towards heterogeneous design of computing systems. Integration of high-performance CPUs with energy-efficient graphics processing units (GPUs) on a single compute node began with the breakdown of Dennard Scaling and has now become common in HPC systems. Fig. 1 shows the rapid increase of GPU-based systems in the Top 500 list. Summit, the recently-crowned fastest supercomputer, boasts a newly-designed heterogeneous architecture.

The use of heterogeneous compute resources is beyond the domain of traditional HPC. Processors on mobile devices are being designed as a System-on-Chip (SoC) which includes a CPU, a GPU, and other types of accelerators. Increased programmability of FPGAs has prompted industry vendors to attach reconfigurable units to conventional processors [7]. Cloud computing infrastructure and Cyber-Physical Systems exhibit a broader type of heterogeneity in which different processing units, such as compute servers, remote sensors, and edge devices [5] combine to complete user tasks with disparate resource requirements. Applications written for such platforms must seamlessly transition between different system layers, each built with a different instruction set architecture.

Heterogeneous Computing (HC) concepts are not yet part of standard undergraduate computer science curricula. The *Curriculum Guidelines for Undergraduate Degree Programs in Computer Science* (CS2013) emphasizes the need for incorporating heterogeneous computing into the curriculum [11], but makes very few recommendations about which specific topics to covered.

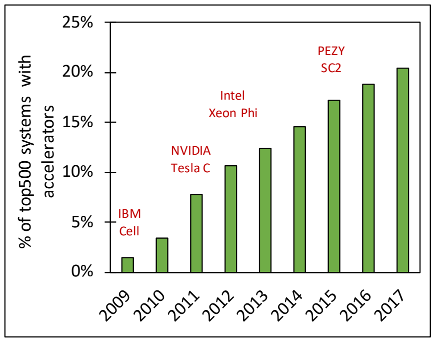

Figure 1: Accelerator-based systems in the Top 500 list [4]. Notable accelerator introductions in red.

A survey of current undergraduate computer science (CS) curricula shows a distinct lack of coverage of HC concepts, reminiscent of the treatment of parallel computing in the undergraduate curriculum in the pre-multicore era [6]. Nearly all coverage is in upper-level electives, mainly at R1 institutions. The sparse coverage of HC topics implies that most undergraduates joining the workforce will have little to no exposure to HC concepts. This is not acceptable, but rapid technological advancements have saturated the undergraduate computer science curriculum. Given this, it is a major challenge to integrate new content without increasing the number of hours to graduation. This ToUCH project was conceived to confront this challenge. This initiative attempts to integrate HC content into the curriculum using the early-and-often approach. The strategy is to develop modules that provide broad coverage of HC topics and inject them into several courses across the curriculum in a way that creates significant coverage of HC concepts along every path through the major. The enhanced curriculum is complemented with synergistic activities that include out-of-class training camps, collaborative design with industry partners, and faculty training.

2. PROPOSED MODULES

We have identified a core set of modules that can collectively provide the required HC coverage. These modules cover both hardware and software aspects and are divided

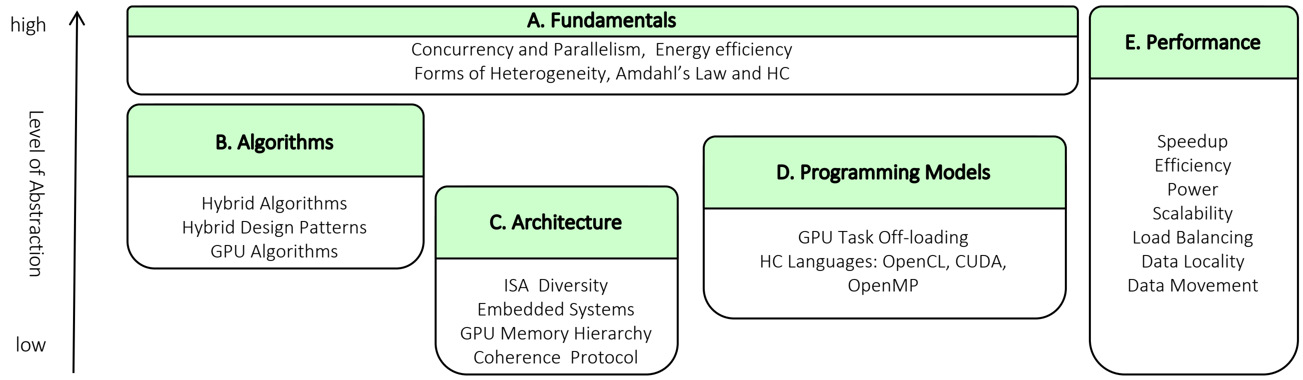

Figure 2: Classification of HC modules and their coverage at different abstraction levels

into five broad categories as shown in Fig. 2. Example modules in each category are also shown in Fig. 2. ***(A) Fundamentals:*** Modules in this category covers elementary concepts in heterogeneous computing. Notions of concurrency, parallelism, and energy efficiency are discussed to explain the motivation behind the move towards heterogeneous processing. Different forms of heterogeneity are introduced including soft heterogeneity (i.e., difference in core compute capabilities within a multicore system), CPU-GPU heterogeneous execution and System-on-Chip (SoC) design. Modules in this class also cover heterogeneity in workload and data with examples from cloud computing and Big Data applications. The modules also include a discussion of programmability and performance challenges, including the implications of Amdahl's Law for HC.

These modules are primarily intended for CS1/CS2 students. They are ideally suited for a course with some coverage of parallel computing material, such as those that introduce PDC modules from [2], [1], or [3].

(B) Hybrid Algorithms: These modules will cover the fundamentals of algorithmic design for heterogeneous systems. Students will learn to develop efficient algorithms while considering the exposed heterogeneity of the underlying processing elements. A typical algorithms from a Data Structures and Algorithms course will be selected and a step-by-step method for its hybridization will be presented. For instance, a breadth-first search (BFS) graph traversal can be used as the running example. BFS can be hybridized on the basis of task granularity. For higher efficiency, nodes with larger degrees should be processed on processing elements with more compute power.

Hands-on exercises will be developed for this module so that these concepts can be illustrated without actual implementation. Nonetheless, if parallelization techniques have already been covered in a prior course then the module can be enhanced by requiring students to come up with a heterogeneous implementation of the selected algorithm.

(C) Heterogeneous Architectures: Under current CS curricula, a typical undergraduate is exposed to only one type of instruction set architecture (ISA). The objective of this module is to familiarize students with a different ISA to emphasize the architectural heterogeneity in today's computing systems. ARM processors are ubiquitous on IoT and mobile devices. Yet it is not the first choice for ISA in many undergraduate courses [9, 8]. For this reason, we propose to use ARM in this module. The module will cover basics of ISA design including instruction encoding, register usage, and support for addressing modes. Students will learn to write simple programs in ARM assembly on a Raspberry PI, a particularly affordable resource.

The module is intended to be embedded in a Computer Organization/Architecture course. The new ISA will be introduced using a compare-and-contrast approach. Instruction semantics will be explained by comparing them to their equivalent counterparts in the ISA that is being covered in the course. For institutions where ARM is the primary ISA (e.g., Concordia), an alternate MIPS module will be developed.

(D) Programming Models: A set of modules will be developed to acquaint students with the many different heterogeneous programming models in use today. These modules will expose students to both low-level (e.g., OpenCL and CUDA) and high-level languages (e.g., OpenACC, C++ AMP, and Chapel [10]). Each module in the set will focus on one particular model and the language associated with it. Key characteristics of the programming model will be presented followed by a discussion of the language constructs that support the model. The module will include a hands-on tutorial for writing simple programs (slightly more advanced than the canonical Hello World) in the selected language. Programming exercises will be developed at the same level of difficulty.

Introduction of this module will be curriculum-dependent. OpenACC or C++ AMP may be more suited for a curricula with little to no coverage of parallel programming. On the other hand, if a course on parallel programming exists, then an OpenCL/CUDA module can be integrated with little difficulty. The choice of the model and language may also depend on other factors such as availability of hardware resources. This was a motivating factor behind our plan to develop a set of modules on different programming models.

(E) Performance: These modules will introduce fundamental performance issues arising in heterogeneous systems, specifically on systems with CPU-GPU processing units. Elementary concepts of heterogeneity and parallel computing, such as concurrency, task off-loading and task decomposition will be reviewed. Students will be given sample codes that are parameterized to expose key performance bottlenecks (e.g., thread block size of a GPU kernel). They will run experiments by adjusting these control knobs and then

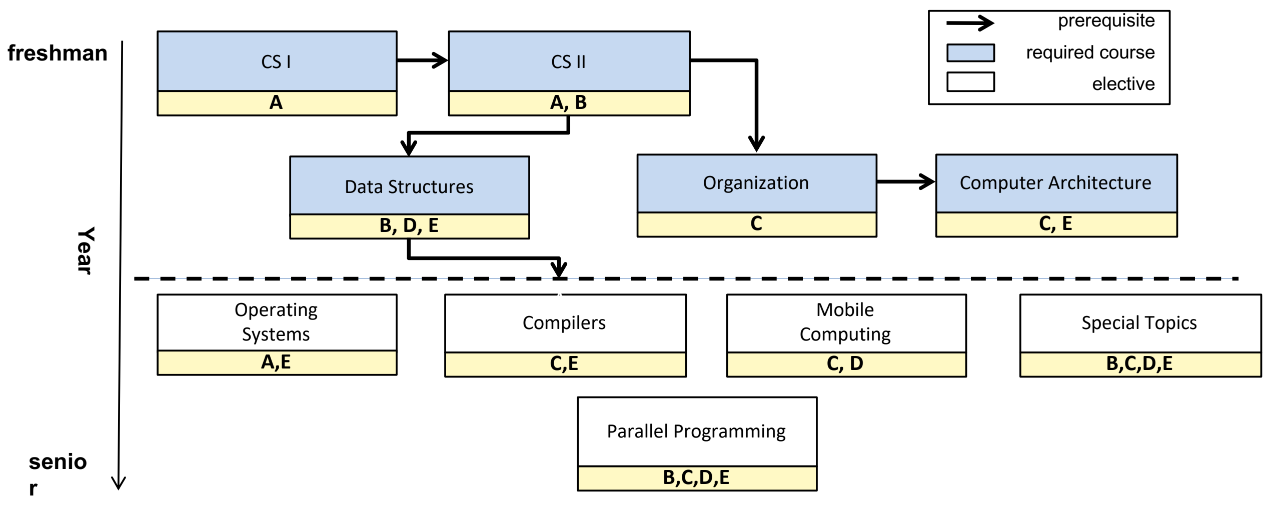

Figure 3: Mapping of HC modules into undergraduate CS courses. Example shows integration into Texas State undergraduate curriculum

report and analyze performance issues. Having completed these modules, students will be expected to have a good understanding of the importance of matching application characteristics to the capabilities of underlying processing units.

These modules can be injected into a variety of upper-level courses. The basic version of these modules will be constructed such that students are not required to write code. Nonetheless, the degree to which programming is introduced in this module can be adjusted based on student background and instructor expertise.

2.1 Design Principles

To facilitate easy integration and adoption, the design of each module will follow three basic principles.

(i) Abstraction: Module topics will be selected based on the level of abstraction at which they can be introduced. Topics that can be explained at a higher level of abstraction will be introduced early in the curriculum. For example, Amdahl's Law and its performance impact on heterogeneous systems could be introduced in a course module directed at CS 1 students.

(ii) Context: Modules take advantage of heterogeneous context. For instance, when covering process scheduling in an O/S course, issues related to *task offloading*, a key concept in the heterogeneous world, can be discussed. When context is exploited, modules do not need to introduce completely new ideas but rather extend topics already being covered. This will allow modules to be introduced with minimal disruption to the original course schedule and content.

(iii) Adoption: Modules will be short (1-3 contact hours) and self-contained. They will include lecture notes, assignments, exam questions, homework problems and solutions, and pedagogical notes to facilitate adoption.

Fig. 3 illustrates how the developed modules can be integrated into a typical undergraduate CS program.

3. REFERENCES

[1] Center for parallel and distributed computing curriculum development and educational resources (CDER). `http://www.cs.gsu.edu/~tcpp`.

[2] CSinParallel Project. `http://csinparallel.org/`.

[3] Parallel Computing in the Undergraduate Curriculum : the Early-and-Often Approach. `http://tues.cs.txstate.edu/`.

[4] Top 500 Supercomputer Sites. `http://www.top500.org`.

[5] P. Beckman, R. Sankaran, C. Catlett, N. Ferrier, R. Jacob, and M. Papka. Waggle: An open sensor platform for edge computing. In *SENSORS*, pages 1–3, 2016.

[6] M. Burtscher, W. Peng, A. Qasem, H. Shi, D. Tamir, and H. Thiry. A module-based approach to adopting the 2013 ACM curricular recommendations on parallel computing. In *SIGCSE*, 2015.

[7] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, et al. A cloud-scale acceleration architecture. In *MICRO*, pages 1–13. IEEE, 2016.

[8] B. U. Computer Science. CS0330: Introduction to Systems. `http://cs.brown.edu/courses/csci0330/`. accessed: 2018-02-11.

[9] T. A. U. Computer Science. CPSC 350: Computer Architecture. `http://courses.cs.tamu.edu/rabi/csce350/index.html`. accessed: 2018-02-11.

[10] A. Sidelnik, S. Maleki, B. L. Chamberlain, M. J. Garzarán, and D. Padua. Performance portability with the chapel language. In *Proc. 26th IEEE Intern. Parallel and Distributed Processing Symp. (IPDPS)*, 2012.

[11] The Joint Task Force on Computing Curricula ACM/IEEE Computer Society. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, 2013.