Journal of Parallel and Distributed Computing 158 (2021) 56–66

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

A module-based introduction to heterogeneous computing in core courses

Apan Qasem [a,*], David P. Bunde [b], Philip Schielke [c]

[a] Texas State University, San Marcos, TX, United States of America
[b] Knox College, Galesburg, IL, United States of America
[c] Concordia University Texas, Austin, TX, United States of America

ARTICLE INFO

Article history:
Received 31 October 2020
Received in revised form 17 March 2021
Accepted 18 July 2021
Available online 4 August 2021

Keywords:
Heterogeneous computing
Module-based instruction
Pedagogy

ABSTRACT

Heterogeneous architectures have emerged as a dominant platform, not only in high-performance computing but also in mobile processing, cloud computing, and the Internet of Things (IoTs). Because the undergraduate computer science curriculum includes so many topics, adding a new course as a required part of the curriculum without increasing the number of hours to graduation is difficult. Integration of heterogeneous computing requires a module-driven approach in which coverage of the topics is broken down into smaller units and dispersed throughout the curriculum. The module-driven approach has been successfully implemented in introducing parallel and distributed computing concepts.

In this paper, we present a set of four teaching modules that introduce fundamental concepts in heterogeneous computing in lower-division computer science courses. The goal of these modules is not to teach students how to program heterogeneous systems but rather to expose them to this emerging trend and prepare them for material in future classes. Although concepts are covered at a high level, the modules emphasize active learning and include lab assignments that provide students with hands-on experience. We also present initial evaluation results for two of these modules based on their use in undergraduate courses at Texas State University. The results are quite encouraging both in terms of learning outcomes and student engagement and interest.

1. Introduction

The need to increase performance per watt coupled with the demands of processing diverse workloads has triggered a major industry shift towards heterogeneous computing systems. Integration of high-performance CPUs with energy-efficient GPUs is now common in all classes of HPC systems. Architectural heterogeneity has also permeated other computing domains such as mobile processing, cloud computing, and the Internet of Things (IoTs). Given this proliferation of heterogeneous architectures, arming computer science graduates with the requisite skills to program these complex systems is imperative.

Current undergraduate computer science (CS) curricula have yet to catch up with this emerging phenomenon and lack sufficient coverage of heterogeneous computing (HC) concepts. Heterogeneity is covered only as an upper-level elective and that too primarily at R1 institutions. Needless to say, including HC as a required part of the curriculum can be challenging. Given credit-hour restrictions and the typically large number of required courses in current CS/CE curricula, adding a new required course is generally not feasible without increasing the time to graduation. Furthermore, HC is important in diverse areas such as programming, algorithms, and architecture so it is better covered in multiple courses rather than a single course.

In this paper, we present four modules for teaching fundamentals of heterogeneous computing to computer science majors during the first two years of their undergraduate study. Table 1 lists the modules along with courses in which we recommend they be introduced. All four modules were developed under the NSF-funded ToUCH project for integrating heterogeneous computing concepts in the undergraduate curriculum [29,26]. The ToUCH initiative attempts to integrate HC content 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 indus-

* Corresponding author.
E-mail addresses: apan@txstate.edu (A. Qasem), dbunde@knox.edu (D.P. Bunde), Philip.Schielke@concordia.edu (P. Schielke).

https://doi.org/10.1016/j.jpdc.2021.07.011
0743-7315/

Table 1
Modules described in this paper. The module indices are derived from the area-based indexing scheme adopted in the ToUCH project [30]. In the remainder of the paper, we use the index and the title alternatively to refer to the modules.

Index	Title	Recommended Course
A1	Heterogeneous Computing: Elementary Notions	CS1
A2	Task Mapping on Soft Heterogeneous Systems	CS1, CS2
A3	Pollack's Rule as a Justification for Heterogeneous Computing	Computer Organization
C1	Heterogeneous ISA: ARM vs MIPS	Computer Architecture

Table 2
Bloom's Classification Levels [1,19] and depth of coverage of topics in the four modules.

Level	Description	Module
Knowledge [K]	recognizing or remembering facts, terms, basic concepts	A1, A2, C1
Comprehension [C]	demonstrating an understanding of facts and ideas by organizing and summarizing the main ideas	A1, A2, A3, C1
Application [A]	solving problems in new situations by applying acquired knowledge	A2, A3

try partners, and faculty training. The four modules presented in this paper focus specifically on fundamental concepts to be covered in freshman- and sophomore-level courses.

The goal of these modules is not to teach students how to program heterogeneous systems but rather expose them to this emerging trend and prepare them for material in future classes. Considering the introductory nature of lower-division courses, topics are covered primarily at the Knowledge level in Bloom's taxonomy. At the same time, the modules emphasize hands-on training and active learning, with appropriate supporting tools, so that the learning outcomes do not become merely an exercise in memorizing HC terminology. Table 2 lists the Bloom levels and the depth of coverage of various topics across the four modules.

The modules can be implemented standalone or as part of a broader plan of integrating HC topics into the curriculum such as those undertaken previously for parallel and distributed computing [4,7]. All supporting material for the modules including lecture slides, handouts, software and tools for the lab assignments, and pedagogical notes are made available on a public Git repository associated with the ToUCH project [30].

2. Design principles

The design of the modules conforms to the three core principles advocated in the early-and-often approach of curricular integration.

(i) **[Abstraction]** *Modules should introduce concepts at the appropriate level of abstraction:* Prior research shows that exposing students to CS principles at the right level of abstraction is critically important. Introducing a concept at an inappropriate level or exposing students to multiple levels at once can hinder the students' understanding of the concepts and reduce their ability to solve problems using the taught principles [14,17]. The modules described in this paper cover HC topics that can be introduced at a high level of abstraction without revealing the complexities of the underlying hardware. For example, Amdahl's Law and its implications for parallel performance are discussed in the module on *Elementary Notions*. The concept of scalability and the significance of Amdahl's Law can be taught without having the students learn how to code heterogeneous systems (or parallel systems for that matter).

(ii) **[Context]** *Modules should provide "heterogeneous context" to key topics in existing curricula:* Many theories and concepts covered throughout CS curricula can enhance a student's comprehension of HC principles. When context is exploited, a module does not need to introduce completely new ideas but rather build on topics already being covered in the course. For instance, when covering process scheduling in an Operating Systems course, the notion of a processor can be replaced by a heterogeneous processing element. The module on *Elementary Notions* introduces heterogeneous program execution when explaining the Von Neumann model, which is a topic commonly covered in most CS1 courses.

(iii) **[Adoption]** *Modules should be self-contained for easy adoption:* The modules are designed to be short (1-1.5 lecture hours) and self-contained. Instructor resources include lecture notes (including in-class demos and activities), lab assignments, sample exam questions and solutions, and pedagogical notes. When possible, the modules are language agnostic. Although they provide a textbook treatment of some material, the modules are not tied to any specific textbook. Each module provides a collection of reference materials for instructors unfamiliar with topics covered. The lab assignments are designed to provide students with first-hand experience in investigating performance issues in heterogeneous systems.

3. Related work

A survey of undergraduate CS curricula at institutions of varied orientation (e.g., R1, masters, liberal arts) shows a distinct lack of coverage of heterogeneous computing concepts. The lack of coverage is more pronounced in standalone computer science programs than combined computer science and engineering programs. Many computer engineering programs include an upper-level elective in which part of the course relates to heterogeneous architectures and related concepts. For example, the Introduction to Biophysical Systems, an upper-level elective in the ECE program at University of Texas at Austin has a section dedicated to System on Chip (SoC) design [16]. The graduate program in Computer Science and Engineering at Washington University in St. Louis includes a course (CSE566S [15]) in which architecturally diverse systems are covered.

Such courses are rare in computer science departments. Surveys of undergraduate computer science programs at Washington University in St Louis [15], Brown University [3], Rice University [10], Concordia University Texas [12], and Concordia University Wisconsin [5], for example, show no dedicated courses in HC. While Brown has extensive coverage of parallel computing at the undergraduate level, Washington University offers only one optional course at the sophomore level, and Rice University has one required junior level class. The aforementioned Concordias have no dedicated courses in parallel computing.

Coverage of HC is typically limited to an advanced course in parallel programming. One or two weeks in such a course will be devoted to GPU computing (e.g. CS4380 at Texas State [23]). Given the time constraints, these segments of the course serve more as a *tutorial* for CUDA programming and do not have the opportunity to investigate the nuances of task-offloading and load balancing in a more general CPU-GPU heterogeneous environment. Heterogeneity of processing cores within a single CPU and its exploitation via dynamic voltage and frequency scaling (DVFS) is covered, in a few instances, as part of advanced computer architecture and compiler courses (e.g. [9]).

The CSinParallel collection includes several modules that mention HC topics [11]. These modules are primarily designed to teach

students CUDA programming and do not emphasize or expose students to the underlying HC concepts. The Center for Parallel and Distributed Computing Curriculum Development and Educational Resources (CDER) boasts a large collection of teaching material for PDC [8]. Nonetheless, only two entries in the database cover the concepts of heterogeneity. One entry presents a computer organization course that integrates GPU CUDA programming for advanced CS students [6]. In the other entry, Gopalkrishnan [13] proposes that heterogeneous programming models should be included in advanced parallel computing courses.

4. Heterogeneous computing: elementary notions

Our first module introduces fundamental 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. The module also covers heterogeneity in workloads and data with examples from cloud computing and mobile applications. The module concludes with a discussion of available programming tools and performance challenges in heterogeneous computing.

4.1. Context

This module is primarily intended for CS1/CS2 students. Although the module introduces parallel computing concepts before moving on to processor heterogeneity, it is ideally suited for a course with some coverage of parallel computing material such as a CS1 course that incorporates a PDC module from [11], [8], or [22]. In the absence of PDC coverage, the length of this module will need to be increased or it will need to be combined with a PDC module.

4.2. Topics

The HC topics covered in this module are listed below. Bloom's classification for each is shown in brackets. (K = Knowledge and C = Comprehension)

- Concurrency and Parallelism [K]
- Multicore Processors [K]
- GPU Acceleration [K]
- System-on-Chip [K]
- Energy Efficiency [K]
- Amdahl's Law [C]

4.3. Learning outcomes

Having completed this module, students should be able to

1. describe the differences between a homogeneous and a heterogeneous computing system
2. describe and distinguish between different forms of heterogeneity
3. discuss the motivation behind the shift towards heterogeneous computing
4. recognize the importance of energy efficiency on current computing systems

4.4. Lecture

The module is designed to cover the concepts in 1 to 1.5 lecture hours. In this section, we describe the progression of the lecture with notes on pedagogy.

4.4.1. Review of von Neumann architecture

The lecture begins with a review of the basic von Neumann architecture. The main components of a von Neumann model and their role in computing are illustrated with an example of a desktop PC. Although perhaps not acquainted with the term itself, a typical CS1 student will be familiar with the fundamental organization of a computing system. A motherboard with a processor and memory module installed is passed around in the class to supplement this discussion. We then introduce students to different classes of computing devices such as mobile processors and IoT devices. The following two points are emphasized

1. We need different types of computers to perform different tasks.
2. Although there are many different types of computing devices in use today, the fundamental organization of these devices remains the same.

4.4.2. Parallel computing and its importance today

A set of lecture slides defines parallel computing and discusses its importance in today's world. A high-level definition of a parallel computer is presented. The discussion of the definition of a parallel computer is followed by some history of parallel computing. The point is made that parallel computing has been around for a long time, ever since the beginning of computing. Notwithstanding, it has only become mainstream in the last decade. Brief descriptions of mainframes, vector computers, and clusters are presented. These descriptions are followed by a discussion of today's multicore computers. The importance of energy efficiency and the role it has played in the evolution of computer chips and rise of multicore processors is discussed. The lecture slides emphasize the need for achieving higher performance at lower power consumption or at specified power budgets. The ubiquity of parallel computers is also discussed. Students are asked to guess/comment on the number of processing cores in their smartphones and tablets. Their guesses are then compared against actual numbers. A discussion follows on the need for more parallel processing cores.

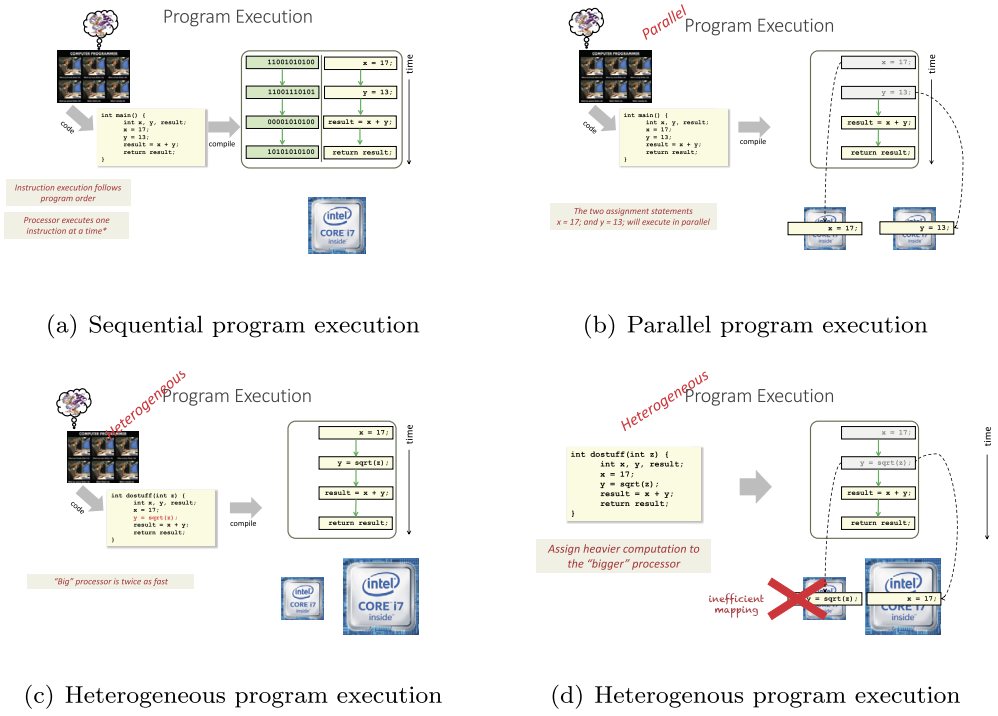
4.4.3. Heterogeneous system design

The need for heterogeneous processors is then motivated using the mobile phone as a running example. Students are polled on the typical usage of their phones and tablets. This discussion is used to introduce the notion of a workload and how different programs within a workload may have different characteristics and different demands for resources. A block diagram of a mobile processor is presented to illustrate how the demands of a diverse workload are handled on such a system with the deployment of a collection of heterogeneous processors. High-performance CPUs, low-power CPUs, and GPUs are illustrated on block diagrams and contrasted with the block diagram of a homogeneous desktop computer.

This example serves as a lead-in to the discussion of different forms of heterogeneity present in today's computing systems. GPUs and their role in processing graphics and general-purpose workloads are discussed, followed by a description of CPU-GPU heterogeneous compute nodes. The notion of soft heterogeneity, processing cores of varying operational frequency, is then introduced.

4.4.4. Sequential, parallel and heterogeneous program execution

A major portion of the module is spent introducing the students to the fundamental difference between sequential, parallel, and heterogeneous program execution. A walk-through example is used for this purpose. Fig. 1 shows a subset of the slides that are used to explain this topic. The slides are accompanied by a set of examples written in SimPar [27]. SimPar is a simple macro language that uses an intuitive pragma-based syntax. Two examples of SimPar programs are shown in Fig. 2. Since students are generally not



(a) Sequential program execution

(b) Parallel program execution

(c) Heterogeneous program execution

(d) Heterogenous program execution

Fig. 1. Excerpts from lecture slides illustrating the differences in serial, parallel and heterogeneous program execution. Animation is used for the different blocks in the slideshow.

```
int add() {
  int x, y, result;
  #PARALLEL {
    x = 17;
    y = 13;
  }
  result = x + y;
  return result;
}
```

(a) Simple parallelization

```
int add() {
  int x, y, result;
  #PARALLEL {
    x = 17;
    y = 13;
    result = x + y;
  }
  return result;
}
```

(b) Incorrectly parallelized code

Fig. 2. SimPar code samples.

expected to be familiar with any parallel programming language in CS1, SimPar is an effective tool to discuss parallelism with real examples without getting bogged down in syntax minutiae. Supplementary materials for this module include a SimPar parser that can be used to create other simple examples. The instructor should be aware that SimPar is not a realistic parallel language and is very limited in ability. Thus, it should not be used for creating extended examples beyond CS1.

During the walk-through of the example, students are asked to list the order in which the statements will execute on the processor. A parallel directive is then inserted for the two assignment statements and the meaning is explained to the students. The program is then extended to include array assignments instead of just simple assignments. This program is compiled and executed and the result examined in class. Students are then asked to comment on what other statements could be parallelized. The instructor leads them to an example where the result statement is put in the `PARALLEL` block along with the two assignment statements. This program is run, potentially several times, and the error demonstrated to the students. The students are then asked to describe the problem in the code. This is followed by a discussion of data dependence and the challenges with parallel programming.

The code example is then extended to illustrate execution of a parallel program on a hypothetical heterogeneous system with a big.LITTLE configuration. The simple assignment statement is replaced with a more computation heavy statement (e.g., `sqrt()`). The parallel execution of the program on the big.LITTLE system is simulated with the computation-heavy statement mapped to the more power-efficient (i.e., *LITTLE*) core. Students are then asked about the performance implications of such a mapping. The instructor then leads them to the correct mapping in the ensuing in-class discussion.

4.4.5. Programming tools

Students are told that SimPar is not a real language. The syntax of real languages are more complex and so are the programming models. Some of the currently available parallel languages and APIs, including OpenMP, Pthreads, and MPI are presented. CUDA and OpenCL are singled out as languages/APIs that support programming in heterogeneous systems.

4.5. Performance challenges

Time permitting, this module can include a section on the performance challenges in parallel and heterogeneous systems. In particular, the notion of sequential and parallel speedup and efficiency can be introduced via examples, followed by a discussion on Amdahl's Law and its performance implications on parallel computing systems. Note, in the author's opinion, Amdahl's Law as it pertains to heterogeneous processors, although important, is too advanced a topic and is not recommended for inclusion when using this module for CS1 students.

5. Task mapping on soft heterogeneous systems

Our second module covers the concepts of task mapping and scheduling on single-ISA systems that embody soft heterogeneity. On such systems, differences in core compute capabilities are the result of differences in operating core frequency. The first part of

the module begins with a brief review of the notions of concurrency, parallelism, and energy efficiency, which serves as a lead-in to the discussion on the motivation behind the shift towards heterogeneous processing. Different forms of heterogeneity are introduced including CPU-GPU heterogeneous execution and heterogeneity that arises in single-ISA systems.

The second part of the module begins with a discussion on workload heterogeneity with examples from mobile computing. The concepts of task mapping and scheduling on sequential and parallel systems are then reviewed to set the stage for the task mapping problem on soft heterogeneous systems. An in-class interactive demo on a real system illustrates the performance and energy efficiency challenges of task mapping which factors in operating frequency. The module is accompanied by a hands-on lab that reinforces these ideas.

5.1. Context

This module is primarily intended for CS2 students. Although the module introduces parallel computing concepts before moving on to processor heterogeneity, it is ideally suited for a course with some coverage of parallel computing material such as a course that incorporates a PDC module from [11], [8], or [22]. In the absence of PDC coverage, the length of this module will need to be increased or it will need to be combined with a PDC module.

This module can be combined with [Module A1] *Heterogeneous Computing: Elementary Notions*. In this case, the module can potentially be used in a CS1 class such as a CS1 class designed for Honors students or one which provides a breadth-first introduction to computer science.

5.2. Topics

The HC topics covered in this module are listed below. Bloom's classification is shown in brackets.

- Single-ISA Heterogeneity [K]
- System-on-Chip [K]
- DVFS and Soft Heterogeneity [K]
- Energy Efficiency [K]
- Heterogeneous Tasks and Workloads [K]
- Task Mapping and Scheduling [C]
- Tools for thread affinity and CPU frequency scaling [A]

5.3. Learning outcomes

Having completed this module, students should be able to

- discuss the motivation behind the design of heterogeneous computing systems
- recognize the importance of energy efficiency in current computing systems
- explain how tasks in a workload have different demands for compute and memory resources
- describe the notion of task mapping as performed by an operating system
- analyze the performance and energy effects of task mapping on a heterogeneous system

5.4. Lab

To reinforce student understanding of the concepts covered in the module, we designed a lab assignment that provides students with hands-on experience on a real heterogeneous system. In this lab, students conduct experiments with different classes of workloads on a heterogeneous multicore machine and investigate the power and energy implications of task mapping on such systems. The multicore system is configured such that each core operates at a different clock frequency (i.e., soft heterogeneity). The configuration is done using commonly available Linux tools, `cpufrequtils` and `cpupower`. Detailed instructions for the configuration are made available as part of the instructor resources that accompany this module. The required hardware must be available at the instructor's institution. However, all that is needed is a Linux-based, ssh-enabled multicore server with at least 4 cores. The pre-programmed workloads are also distributed as part of the instructor resources. The source code is written in C/C++ and can be built in a Linux environment with gcc 5.4.0 or above. The lab requires the students to have some basic familiarity with a Linux environment. A description of the lab handout is included in Appendix A.

6. Pollack's rule as a justification for heterogeneous computing

Our third module uses high-level performance modeling to explain the motivation of multicore systems to include a heterogeneous collection of cores, as is common on small systems (e.g. cell phones) and on very high-end systems (e.g. the Xeon Phi and all the HPC systems with GPUs). The module was inspired by an architecture paper by Morad et al. [21] that advocated for a diversity of core sizes to maximize performance under a power budget.

The module's performance modeling relies on two rules. The first is Pollack's Rule [25,2], which states that the performance of a core is proportional to the square root of its area. This is an empirically-observed relationship similar to Moore's "Law". An intuitive justification is that increasing core area allows performance-improving features such caching or more complicated logic (branch prediction, deeper pipelines, etc). The natural implication of Pollack's Rule is that increasing the number of cores improves performance; splitting a single core into k smaller cores using $1/k$ of the area means that each of them provides only $1/\sqrt{k}$ as much performance but that the peak performance of the system increases by a factor of $k/\sqrt{k} = \sqrt{k}$. By this logic, processors should consist of as many tiny cores as possible.

Pushing back against this conclusion is the well-known Amdahl's Law, which the module uses to calculate the performance of hypothetical workloads where a fraction of the work is arbitrarily-parallelizable and the rest must be run serially. These calculations show that having cores with heterogeneous size (and hence heterogeneous performance) allows a processor to capture much of the benefit of using many tiny cores while also providing a large core to protect the performance on the workload's serial component.

The module fits into a single lecture, requiring 30–40 minutes to cover. It has a theoretical feel because the performance model is fairly abstract, with the only parameters being the core sizes and the percent of the program that is parallelizable. Followup homework and exam problems mirror those of lecture, asking students to compute the performance for different parameter values. To help tie the module to students' own experiences, we recently added a part where the results of a cell phone profiling tool are presented. The tool shows that the professor's phone is multicore, with different speeds possible on each. (While this is a different kind of heterogeneity than that modeled in the module, power management exhibits a similar balancing act to the size issue we see in the module; single core performance is maximized by a single fast core, but total peak performance is maximized by having as many slow cores as possible.)

6.1. Context

This module was developed as part of an introductory computer systems course at Knox College. The course covers low-level programming in assembly and C, caching/locality, concurrency and

parallelism, and security. The prerequisite course is CS 2, but most students take the course as sophomores, after 3–4 prior CS courses. Some of the students are more experienced; a couple each year are seniors in their second to last term.

The module comes about two thirds of the way through the course, following coverage of homogeneous parallel programming. In the homogenenous parallel programming part of the course, students use pthreads and OpenMP to parallelize simple programs, resolving races in the pthreads version with mutexes. Prior lectures cover classic synchronization problems such as producer-consumer and readers and writers, as well as deadlock and some of the ways to avoid or resolve it. The module introduces and motivates heterogeneity in order to transition to a brief introduction to CUDA and the material on the Thumb mode of module C1 (presented in the next section). Thus, the module appears as part of a fairly robust discussion of parallelism and concurrency (roughly one third of the course).

Although we use the module as part of a large unit, we believe it is also suitable in different contexts. Students need familiarity with the idea of parallel computing and the speedup metric. As written, the module also assumes prior exposure to Amdahl's law, though that could be provided with the module as well. The module would fit well with any course teaching about multicore systems, including Computer Organization. It could also be used to explain the use of heterogeneous elements in mobile or HPC systems. Because it does not require programming, in principle the module could be used in CS1 after a module introducing to parallel computing or to expand on the coverage of module A1.

When used at Knox College, this module is taught using Peer Instruction [24,20], a pedagogy in which lecture is punctuated with multiple choice questions that students answer individually and then again after a discussion in small groups before the professor leads a whole-class discussion. To facilitate use of the module by a variety of instructors, two versions of slides for the module are provided, one with embedded questions for Peer Instruction and one that solves these questions for use in a more traditional lecture-based delivery.

6.2. Topics

The topics covered in this module are listed below. The Bloom's classification for each is shown in brackets. (C=Comprehension, A = Application)

- Rationale for heterogeneity in parallel systems [C]
- Performance modeling for parallel systems [C]
- Amdahl's Law [A]
- Pollack's Rule [A]

6.3. Learning outcomes

After completing this module, students should be able to

- justify the use of heterogeneous cores in multicore systems
- use Pollack's Rule to estimate the peak performance of a multicore system relative to another system (serial or multicore) of the same size
- use Amdahl's Law to compute the performance of a parallel system on jobs with serial sections of different size

6.4. Followup assignments

To create homework and exam questions based on this module, we have simply given the students a novel set of core sizes, had them compute the relative peak performance of this system to a single core system of the same size, and then compute the speedup achievable on this system for a job with a given level of parallelizabilty. These problems are exactly the same kind as solved in lecture. In our experience, students tend to do well on them and the problems are fairly easy to grade.

7. Heterogeneous ISA: ARM vs. MIPS

The primary goal of our fourth module is to expose students to an architecture with which they are unfamiliar. Computer Science students typically receive minimal exposure to hardware architectures in one or perhaps two classes. This module can be added to an existing course (which focuses on an architecture other than ARM) giving students experience with an alternate architecture in a minimal amount of time.

The second goal of the module is to provide instructors an easy way to expose students to the intrinsic heterogeneity [18] of the ARM architecture. Hardware features and co-processors available on various versions of the ARM architecture require various design trade-offs to be considered by the developer. For example, use of ARM's Thumb mode may reduce code-size and power usage while sacrificing performance. ARM's Neon and VFP subsystem co-processors provide SIMD features which require increased developer design time and possibly higher power consumption.

7.1. Context

This module is intended for a Computer Organization course or a Computer Architecture course after students have become proficient in writing simple programs in assembly language for the ISA targeted by the course. Typically, such courses would occur in a student's sophomore or junior year.

7.2. Topics

The topics covered in this module are listed below. The Bloom's classification for each is shown in brackets. (K = Knowledge, C=Comprehension, A = Application)

- Linux command line [A]
- ARM architectural overview [K]
- Code-size/performance/power trade-offs [C]
- Co-processor architectures [C]
- Elementary SIMD concepts [K]

7.3. Learning outcomes

After having completed this module, students should be able to

- understand simple assembly programs written in the ARM instruction set
- describe the differences and similarities between MIPS and ARM
- (Optional) discuss the trade-offs encountered when writing code in Thumb mode versus ARM mode
- (Optional) describe the benefits of using ARM's SIMD engine Neon

7.4. Lectures

There are three lectures provided. The first lecture provides a general introduction to the ARM architecture, specifically contrasted with MIPS. (Courses not using MIPS can simply ignore those slides.) It should be used for students who are new to ARM. The slides cover the ARM instruction set, addressing modes, status register, predicated execution, and function call implementation in assembly. It can be completed in one lecture hour.

The other two lectures are optional and independent of the first. The second lecture describes ARM's Thumb mode, and exposes students to the trade-offs of run-time performance, code-size, and power which are available with Thumb. Several examples of the Thumb instruction set are presented. The third lecture introduces students to the concept of co-processors, and specifically describes ARM's SIMD engine, Neon. Courses where ARM is the primary architectural focus can omit the first lecture while still utilizing either or both of the other two. Both of these lectures are targeted at a fairly high level of abstraction and can be completed in one lecture hour total. Several links to a more detailed description of Neon are provided.

7.5. Labs

Three labs are provided with this module. In the first, students write a few simple programs using ARM assembly language. The second lab follows the second (Thumb) lecture and gives students an opportunity to compare the code-size and run-time performance of C code when compiled for ARM, Thumb-1, and Thumb-2 targets. The final lab follows the third (NEON) lecture, and gives students an opportunity to see the speed-ups possible when parallelizing a simple sequential program using the NEON SIMD co-processor. All labs are designed to be run on a Raspberry Pi running the Raspbian operating system. The GNU tool-chain must be installed.

7.5.1. Lab1 intro to ARM

In this lab, students will complete a simple ARM assembly language program that computes factorial to familiarize them with ARM assembly and the tools used to build ARM programs. Students should be given the code in lab1.s, a shell assembly program. This starter code prompts the user to enter a value and reads that value, n, from stdin by making a library call to C's `scanf()`. This value is placed in ARM register R0. The student should then write a simple loop to compute $n!$ (n factorial), and put the result of that computation into $R2$. The provided code will then print that value to stdout by making a libary call to `printf()`. Detailed instructions are provided in lab1.md and a solution for the instructor is provided in lab1_sol.s.

7.5.2. Lab2 comparing ARM and Thumb

The instructions for this lab can be found in the markdown file lab2.md. Students will download a zipfile with two simple benchmarks written in C, a matrix multiply function and a function that finds Prime numbers using the sieve of Eratosthenes. A makefile is provided that will compile the code by targeting ARM (32-bit instructions), Thumb-1 (16-bit instruction only), and the version of Thumb-2 (mixed 16-bit and 32-bit instructions) found on the particular chip utilized by the Raspberry Pi. (Note that compiling for Thumb-1 is not directly supported since the Raspberry Pi's Broadcom chip uses a version of Thumb-2. The makefile forces a cross-compilation targeting a previous architectural revision to generate Thumb-1 code and then modifies the generated assembly to allow the code to link. Note that any Thumb-2 processor is completely capable of running Thumb-1 code. The difficulty lies solely in the GNU toolchain.)

Students should be familiar with a Unix (Linux) command line, and will use a variety of command line tools to measure and inspect the generated code. If the class is unfamiliar with command line usage, an additional tutorial or instruction may be required.

7.5.3. Lab3 ARM NEON

In this lab, students will compare three implementations of code to add the elements of one array to the elements of another array, storing the result in a third array. Simple element-by-element implementations of this computation are provided in C and assembly. Students will take the sequential assembly code and rewrite it to make use of the ARM SIMD (Neon) instructions. Students will then compare the run-time performance of the three implementations. The Neon version of the code can be further improved by making use of the 128 bit registers, which will enable the code to perform 16 parallel additions of eight-bit values using one instruction.

8. Evaluation

We now present preliminary evaluation for modules A1 (§ 4) and A2 (§ 5). The other modules have been used in classes as part of their development, but the evaluation done was informal and intended only to guide module development.

Modules A1 and A2 have been used multiple times in undergraduates courses at Texas State University. The A1 module was introduced in the CS1 course in Fall 2018 and Fall 2019. The A2 module was first introduced alongside the A1 module in the CS1 course in Fall 2018. Based on student and reviewer feedback, A2 was split from A1 and the second instance was offered independently in a section of CS2 in Fall 2019.

The CS1 course at Texas State introduces programming using C++ and provides some coverage of computer science breadth topics. The particular section of CS1 in which the modules were taught was designated as an Honors section. The enrollment in the Honors section is selective and only students with a strong academic background are allowed to enroll. The class was capped at 20 and 21 in Fall 2018 and Fall 2019, respectively. The class comprises of both majors and non-majors. In both Fall 2018 and Fall 2019, only half of the enrolled students were declared CS majors. Enrollment in the non-Honors sections of CS1 at Texas State can reach up to 350 and the sections are co-taught by multiple faculty members. For this reason, we deemed the Honors section of CS1 to be a good venue for a pilot implementation.

The CS2 course at Texas State also uses C++ as the main programming language. Average enrollment for a section is around 50. In Fall 2019, when the A2 module was introduced, 52 students were registered for the course.

8.1. Methodology

8.1.1. Summative evaluation of learning outcomes

We designed a set of exam questions to assess the four learning outcomes associated with A1 (§4.3). One question was selected from this pool for inclusion in the final exam. The question was mandatory and constituted 5% of the final exam grade. Since heterogeneous computing is not a required part of these courses, the questions did not carry significant weight.

Learning outcomes in A2 (§ 5.3) were evaluated based on student performance on the lab assignment. Students worked in pairs on this lab, as they do for several other programming projects in CS1 and CS2 classes at Texas State. Rubrics were constructed for each learning outcome such that student performance in the lab assignment could be used as a measure of student learning.

8.2. Surveys to evaluate student engagement

We conducted an end-of-the semester survey to gauge student interest in the topic and assess student perception of the learning experience. Questions were selected from the Student Assessment of Learning Gains (SALG) survey [28]. Students were asked to rate the module in the following categories (note the verbiage is a little different from the actual survey administered)

(1) *Class activities:* Were the class activities (i.e., lecture, in-class activity, live-demo) associated with the module helpful and engaging?

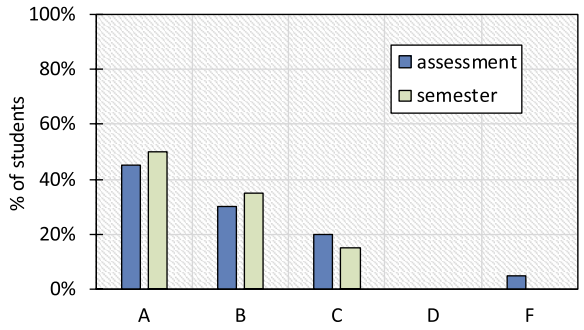


Fig. 3. Semester grades and assessment question grade distribution for A1 module, Fall 2018.

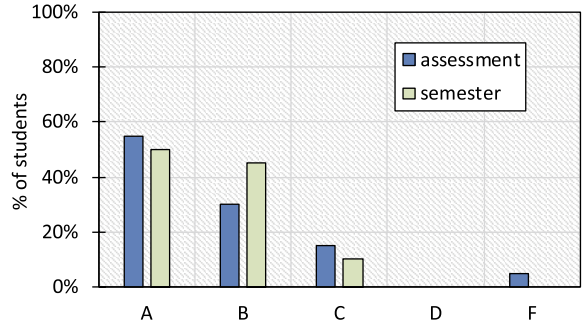


Fig. 4. Semester grades and assessment question grade distribution for A1 module, Fall 2019.

(2) *Learning support:* Did the instructor provide enough support (e.g., further reading, tutoring) outside the classroom for learning the material taught in this module?
(3) *Learning experience:* Overall, how would you rate your learning experience in this module compared to the rest of the course; how would you rate it compared to other courses?
(4) *Confidence and interest gains:* Has this module increased your interest in pursuing a CS degree or taking more CS courses?

Students answered each question on a scale of 0-4 (e.g., strongly disagree, disagree, neutral, agree and strongly agree).

8.3. Evaluation of module A1

Fig. 3 shows the student grade distribution on the final exam question for module A1 in Fall 2018. 90% of the students received a passing grade with almost half of them receiving full credit. One student received a failing grade. This student did not show up the day the module was introduced and opted to not answer the question in the final. As noted before, the students in the Honors section in general are high achievers. Thus, the outcome results should be considered in context. The class grade distribution is shown in Fig. 3. As we can see, the distribution does not follow a normal curve and is skewed to the left. However, the cumulative grades closely match the grades on the module exam question, indicating that the students did not find the material associated with the module significantly more difficult to understand than the rest of the material covered in the class.

Results from the second installment of the A1 module are shown in Fig. 4. A different question was used to evaluate student learning outcome in Fall 2019. The results are very similar to the results from the prior year. There was one student who did not write a response to the assessment question and received a failing grade. For the rest, the semester grades matched closely with grades received in the assessment question.

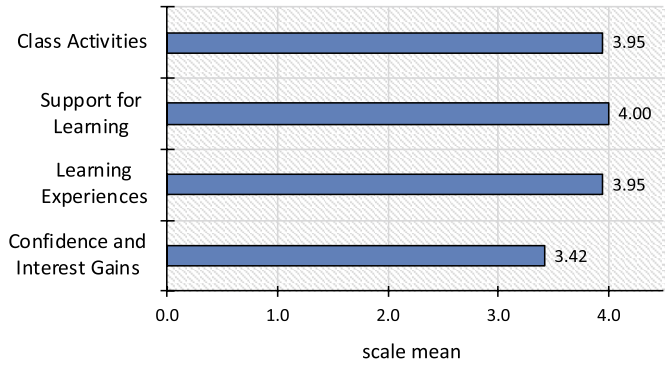


Fig. 5. Student learning experience, confidence and interest gains in A1 and A2 modules in Fall 2018. Survey respondents: 19/20.

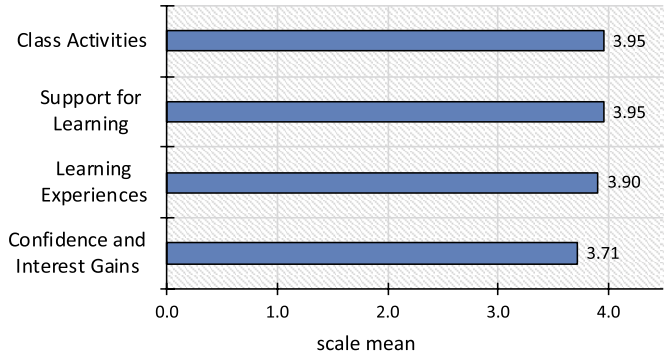


Fig. 6. Student learning experience, confidence and interest gains in A1 module, Fall 2019. Survey respondents: 21/21.

The results of the survey conducted in the CS1 class in Fall 2018 are shown in Fig. 5. Note, in Fall 2018 both A1 and A2 were introduced in CS1 and therefore the survey response relates to both modules. Overall, the students rated the learning experience and instructional environment very positively. All but one respondent, rated the class activities as “very helpful” and “very engaging”. In the comments section of the survey, several students singled-out the in-class demo as being particularly helpful. All students said that there was sufficient help outside the classroom. This is a reflection of the (i) quality of help provided by the TA (a graduate student working in the area of HC) and (ii) helpfulness of Linux tools developed to allow students to complete the lab assignment. Overall, all but one student rated their learning experience in the module as very positive. In terms of interest gains in CS, most students (12 out of 19) had a positive impression. However, these responses are not as overwhelmingly positive as the other categories. This is not unexpected. The data for incoming freshman at Texas State suggests that most of them choose CS as a major because they want to pursue a career as a programmer or coder. Since the HC material is a little removed from programming, the material failed to create as strong an impression to these budding computer scientists.

The survey results conducted in the CS1 class in Fall 2019 are shown in Fig. 6. There is about an 8% improvement in the area of Confidence and Interest gains while the responses in the rest of the categories are virtually identical to the previous year. We speculate that word-of-mouth may have contributed to the improved score in Confidence and Interest gains. Each year the Honors College at Texas State holds an open house for prospective students. In this event, students who have taken the class previously share their experiences. In the 2019 event several students discussed the HC module and how it helped them gain a better perspective of emerging trends in computer science. This background knowledge

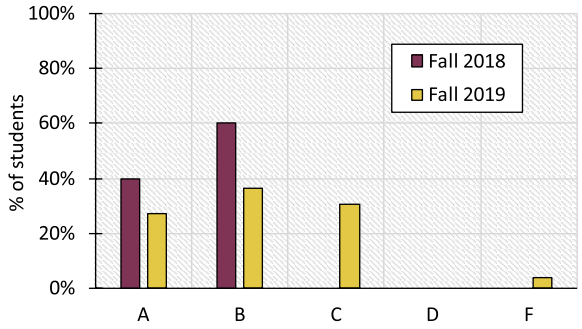


Fig. 7. Distribution of grades of HC lab assignment in A2 module, Fall 2018-2019.

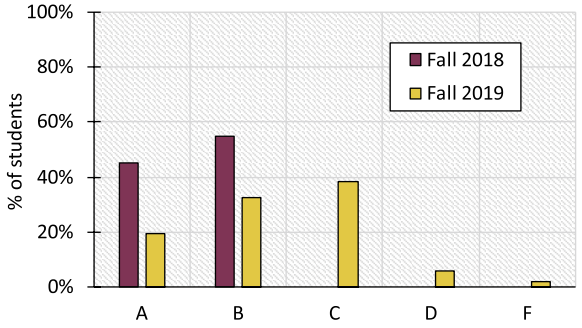


Fig. 8. Distribution of grades on non-HC lab assignments in A2 module, Fall 2018-2019.

may have influenced some students' response in that category, resulting in a higher score.

8.4. Evaluation of module A2

Learning outcomes in A2 were evaluated using student scores on the lab assignment. Fig. 7 shows the grade distribution in the assignment for both Fall 2018 and Fall 2019. When the module was offered the first time in the CS1 Honors class, no one received less than a B and 25% received an A. In the second installment in the CS2 course, 96% of the students received a passing score. Two of the 52 students received a failing grade because of incomplete submissions.

We believe the grades from Fall 2019 are more representative of actual student learning outcomes. As mentioned, the Honors section in consists of a select group of hard-working students with higher aptitude. These students will end up with a decent grade, even if the material itself is not particularly accessible. The grade distribution in the CS2 class is closer to *normal* and also reflects the overall grade distribution in the class. In Fig. 8, we present distributions of average grades for all non-HC lab assignments in Fall 2018 and Fall 2019. The distributions are slightly better for the CS1 Honors section and slightly worse for CS2. The data indicates that the students did not find the assignments on heterogeneous computing significantly more challenging. Notwithstanding, we cannot reach a definitive conclusion because our sample size is small.

Student engagement in the Fall 2018 offering of this module has already been discussed. Fig. 9 shows survey results from Fall 2019. The results are similar to that of the A1 module with a slight increase in student interest and confidence gains. We should note however, that we received survey responses from 43 of the 52 students enrolled in the class (83%). If students failed to respond when they did not find the module particularly helpful or interesting, a higher response rate could have yielded lower scores.

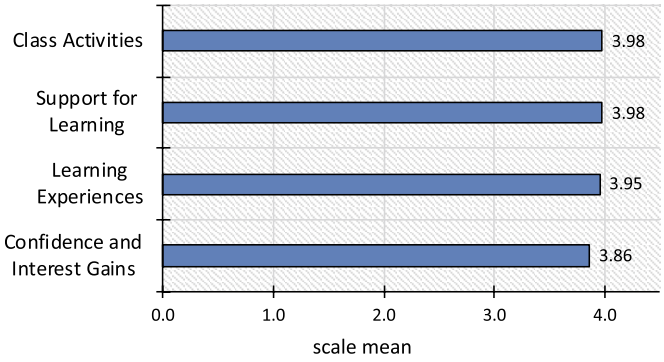


Fig. 9. Student learning experience, confidence and interest gains in A2 module, Fall 2019. Survey respondents: 43/52.

9. Conclusions and future work

This paper presents a set of four teaching modules for exposing introductory CS students to heterogeneous computing. The modules cover fundamental HC topics at a high level of abstraction but this coverage is complemented with hands-on assignments and in-class exercises that allow students to reinforce their understanding by conducting experiments on real hardware. Preliminary evaluation is promising both in terms of student learning outcomes and engagement.

We plan to continue our evaluation work. In addition to gathering data on them in our own courses, we will be presenting these modules to other faculty at several conferences and recruiting them to test the modules at their own institutions.

CRediT authorship contribution statement

Apan Qasem: Investigation, Methodology, Writing – original draft, Writing – review & editing. **David P. Bunde:** Investigation, Methodology, Writing – original draft, Writing – review & editing. **Philip Schielke:** Investigation, Methodology, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Science Foundation through awards CNS-1253292, OAC-1829644, and OAC-1829554. Equipment donations by IBM and NVIDIA helped in developing the labs associated with the modules. The authors thank the reviewers for their comments which helped improve this work.

Appendix A. Heterogeneous task mapping lab assignment

Objective

In this assignment, you will investigate performance and energy issues of a heterogeneous computing system. You will be given a set of four programs with different characteristics. Your goal is to determine the best mapping of these programs to the different processing cores via experimentation and analysis.

Environment

You will be running experiments on `megatron`, a heterogeneous multicore system. `megatron` has four processing cores and

each core has been configured to do a specific type of job. Although each core can do any type of computation it will perform certain tasks really well.

Tools

Familiarize yourself with the following tools. They are all installed in standard locations on `megatron`

- `mapper`: task mapping
- `perf`: performance evaluation via HW counters
- `likwid`: energy and power estimation
- `cpufrequtils`: CPU frequency scaling

Instructions

(1) Log in to megatron

`megatron` is a server behind the firewall. From within the school network, you can `ssh` into `megatron` as follows

```
ssh netid@megatron.cs.school.edu
```

From an off-campus network, you will first need to `ssh` into a gateway server (e.g., `gateway.cs.school.edu`) and then log in to `megatron`.

(2) Download code samples

Once you have logged into `megatron`, clone the following git repository into your home directory

```
git clone https://git.school.edu
```

Create a directory for the codes to reside and unzip the codes into that directory. You should see four executables and a README. The four executables are designed to perform the following tasks

- `p0`: numeric computation (e.g., excel)
- `p1`: graphics (e.g., game)
- `p2`: play music (e.g., music app)
- `p3`: communicate with the internet (e.g., web browser)

The README has more information about each application and their characteristics.

(3) Conduct Performance Experiments

Launch the four programs, at the same time, with different thread mapping configurations. You can do this in one step using the mapper tool (installed in `/usr/local/bin/mapper`). For example,

```
mapper p0 p1 p2 p3 3 1 0 2
```

The above command will launch the four programs at the same time and map `p0`, `p1`, `p2`, `p3` to processing cores `3`, `1`, `0` and `2` respectively. The program arguments must be the fully qualified name of the executable and the processor arguments must be in the range 0-3. Type the following to see more options

```
mapper – help
```

For each configuration, record the performance of each individual core and the overall workload. You can use the `perf` tool for this purpose.

```
perf stat mapper p0 p1 p2 p3 3 1 0 2
```

`perf` will report a bunch of performance metrics. The ones that you want to pay particular attention to are *CPUs Utilized* and *instructions per cycle*. Instructions per cycle (IPC) is a throughput metric that normalizes performance across different workloads.

Repeat the experiments and measure the energy consumption. You can use `/usr/local/bin/likwid` to do this

```
likwid -c 0-3 -g ENERGY mapper <args>
```

(4) Analyze the data

Create charts showing performance (as measured using the metrics described above), power and energy for different configurations. Analyze the data and create a report answering the following questions

- Which processor is good at numeric computation?
- Which processor is good at graphics?
- Which processor is good at playing music?
- Which processor is good when there is a need to communicate over the network?
- Do the answers hold for power as well?
- What is the configuration that provides the best performance?
- What is the configuration that consumes least power?
- What is the configurations that is most energy efficient?

References

[1] B.S. Bloom, Taxonomy of Educational Objectives: The Classification of Educational Goals: By a Committee of College and University Examiners, David McKay, 1971.

[2] S. Borkar, Getting gigascale chips: challenges and opportunities in continuing Moore's law, ACM Queue 1 (7) (2003) 26–33.

[3] Brown cs:courses, https://cs.brown.edu/courses/. (Accessed 7 February 2018).

[4] R.A. Brown, E. Shoop, Modules in community: injecting more parallelism into computer science curricula, in: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE), 2011, pp. 447–452.

[5] Bs in computer science - cuw cs, http://www.cs.cuw.edu/cs-major/. (Accessed 7 February 2018).

[6] D. Bunde, K.L. Karavanic, J. Mache, C.T. Mitchell, Adding GPU computing to computer organization courses, in: Proc. 3rd NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar), 2013.

[7] M. Burtscher, W. Peng, A. Qasem, H. Shi, D. Tamir, H. Thiry, A module-based approach to adopting the 2013 ACM curricular recommendations on parallel computing, in: Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE), 2015.

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

[9] Code generation and optimization, NEEDCITATION. (Accessed 5 February 2018).

[10] Course catalog 2017-18, https://courses.rice.edu/admweb/!SWKSCAT.cat?p_action=CATALIST&p_acyr_code=2018&p_subj=COMP. (Accessed 10 February 2018).

[11] CSinParallel Project, http://csinparallel.org/.

[12] Degree requirements - computer science, http://www.concordia.edu/academics/school-of-natural-and-applied-sciences/computer-science/degree-requirements.html. (Accessed 7 February 2018).

[13] G. Gopalakrishnan, Formal methods for surviving the jungle of heterogeneous parallelism, in: 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), IEEE, 2012, pp. 1321–1324.

[14] O. Hazzan, Reducing abstraction level when learning computability theory concepts, in: Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE), 2002, pp. 156–160.

[15] High performance computer systems, http://bulletin.wustl.edu/undergrad/engineering/computerscience/#courses. (Accessed 7 February 2018).

[16] Introduction to cyberphysical systems, https://www.cs.utexas.edu/courses/378-introduction-cyberphysical-systems. (Accessed 5 February 2018).

[17] J. Kramer, Is abstraction the key to computing?, Commun. ACM 50 (4) (2007) 36–42.

[18] W. Lee, D. Sunwoo, C.D. Emmons, A. Gerstlauer, L.K. John, Exploring opportunities for heterogeneous-isa core architectures in high-performance mobile socs, Technical Report UT-CERC-17-01, The Computer Engineering Research Center, University of Texas at Austin, Mar. 2017.

[19] R.J. Marzano, J.S. Kendall, The New Taxonomy of Educational Objectives, Corwin Press, 2006.

[20] E. Mazur, Peer Instruction: A User's Manual, Prentice Hall, Upper Saddle River, New Jersey, 1997.

[21] T. Morad, U. Weiser, A. Kolodny, M. Valero, E. Ayguadé, Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors, IEEE Comput. Archit. Lett. 4 (1) (2005) 14–17.

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

[23] Parallel programming, https://cs.txstate.edu/academics/course_detail/CS/4380/. (Accessed 5 February 2018).

[24] Peer instruction for computer science, http://peerinstruction4cs.org/, October 2020.

[25] F. Pollack, New microarchitecture challenges in the coming generations of CMOS process technologies, keynote address at IEEE Intern. Symp. Microarchitecture (MICRO), http://hpc.ac.upc.edu/Talks/dir07/T000065/slides.pdf, 1999.

[26] A. Qasam, D. Bunde, P. Schielke, Touch: teaching undergrads collaborative and heterogeneous computing, in: Consortium for Computing Sciences in College: South Central Region 2019 Conference, University of Texas at Dallas, Richardson, TX, 2019, https://github.com/TeachingUndergradsCHC/CCSC19/blob/master/poster_48x36.pdf.
[27] A. Qasem, SimPar: a macro language for introducing parallel concepts to CS 1 students, https://github.com/TeachingUndergradsCHC/SimPar.git. (Accessed 11 March 2021).
[28] E. Seymour, D. Wiese, A. Hunter, S.M. Daffinrud, Creating a better mousetrap: on-line student assessment of their learning gains, in: National Meeting of the American Chemical Society, 2000.
[29] ToUCH: teaching undergrads collaborative and heterogeneous computing, https://touch.cs.txstate.edu.
[30] Touch: Teaching undergraduates collaborative and heterogeneous computing, https://github.com/TeachingUndergradsCHC/modules. (Accessed 23 September 2019).

Apan Qasem is an Associate Professor and the Associate Chair of the Department of Computer Science at Texas State University, where he conducts research in high-performance computing with an emphasis on code optimization for heterogeneous environments. From 2012-15, he led an NSF-funded project to integrate parallel computing concepts into the undergraduate curriculum.

David Bunde is the William & Marilyn Ingersoll Professor of Computer Science at Knox College, where he conducts research in high-performance computing. He currently leads the effort to promote Peachy Parallel Assignments, parallel computing assignments that are tested, easy for others to adopt, and motivational for students.

Philip Schielke is an Associate Professor of Computer Science at Concordia University Texas. He is active in curriculum development at all levels of Concordia's Computer Science programs. Schielke has revamped the University's Bachelor of Science in Computer Science, added a Bachelor of Arts in Computer Science, and rewritten about 75% of the CS course documents for the University.