A Gentle Introduction to Heterogeneous Computing for CS1 Students

Apan Qasem
Texas State University
San Marcos, TX
email: apan@txstate.edu

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 is over-crowded in its current state, it is difficult to include a new course as a required part of the curriculum without increasing the number of hours to graduation. Integration of heterogeneous computing content requires a module-based approach, such as those undertaken for introducing parallel and distributed computing.**

In this paper, we present a teaching module that introduces CS1 students to some of the fundamental concepts in heterogeneous computing. The goal of this module is not to teach students how to program heterogeneous systems but rather expose them to this emerging trend and prepare them for material they are expected to see in future classes. Although concepts are covered at a high-level, the module emphasizes active learning and includes a lab assignment that provides students with hands-on experience with respect to task mapping and performance evaluation of a heterogeneous system. The module was implemented at our home institution in Fall 2018. Initial evaluation results are quite encouraging both in terms of learning outcomes and student engagement and interest.

1. Introduction

The need for increased 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, it is imperative that computer science graduates are equipped with the requisite skills to program these complex systems of the future.

Current undergraduate computer science (CS) curricula is yet to catch up with this emerging phenomenon and lacks 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. Because the CS/CE curriculum is overcrowded in its current state, adding a new required course is generally not feasible without increasing the time to graduation. Furthermore, HC spans different areas such as programming, algorithms and architecture and as such they are better covered in multiple courses rather than a single course.

In this paper, we present a teaching module that introduces CS1 students to fundamental concepts in heterogeneous computing. The goal of this module is not to teach students how to program heterogeneous systems but rather expose them to this emerging trend and prepare them for material they are expected to see in future classes. Considering the introductory nature of the course, topics are covered primarily at the Bloom's Knowledge Level. At the same time, the module emphasizes hands-on training and active learning, with appropriate supporting tools, such that the learning outcomes do not become merely an exercise in memorizing HC terminology. The module can be implemented as 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 [**?**], [**?**]. All supporting material for the module including lecture slides, handouts, software and tools for the lab assignment, and pedagogical notes are made available on a public git repository associated with the ToUCH project [**?**].

2. Design Principles

The design of the module 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 has shown that it is critically important to expose students to CS principles at the right level of abstraction. 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 [**?**], [**?**]. The module described in this paper, covers 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 on parallel performance is discussed in this module. The notion 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 curriculum:* Many theories and concepts covered throughout the CS curriculum 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. In this module, we introduce heterogeneous program execution when explaining the Von Neumann model, a topic commonly covered in most CS1 courses.

(iii) **[Adoption]** *Modules should be self-contained for easy adoption:* The module is designed to be short (1-1.5 lecture hours) and self-contained. Instructor resources include lecture notes (including in-class demos and activities), a lab assignment, sample exam questions and solutions, and pedagogical notes. The module is language agnostic and although it provides a textbook treatment of some material, it is not tied to any specific textbook. A collection of reference material is included for instructors unfamiliar with topics covered in this module. The lab assignment is designed to provide students with first-hand experience in investigating performance issues in heterogeneous systems. To ensure that CS1 students are able to tackle this assignment, the authors developed `mapper`, a command-line tool for the Linux environment, which provides a simple interface to assign tasks to processing cores. `mapper` is packaged along with the rest of the instructor resources of this module. The module also includes a tutorial for instructors for simulating a heterogeneous platform on conventional multicore hardware.

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 SoC design [**?**]. The graduate program in Computer Science and Engineering at Washington University in St. Louis includes a course (CSE566S [**?**]) 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 [**?**], Brown University [**?**], Rice University [**?**] Concordia University Texas [**?**], and Concordia University Wisconsin [**?**], 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 [**?**]). 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. [**?**]).

The CSinParallel collection includes several modules that do touch on HC topics [**?**]. These modules are primarily designed to teach students CUDA programming and does 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 [**?**]. Nonetheless, only two entries in the database touch on the concepts of heterogeneity. One presents a computer organization course that integrates GPU CUDA programming for advanced CS students [**?**]. In the other, Gopalkrishnan [**?**] proposes that heterogeneous programming models should be included in advanced parallel computing courses.

4. Organization and Content

This 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 workload and data with examples from cloud computing and mobile applications. The module concludes with a discussion of programmability and performance challenges.

4.1. Context

This module is primarily intended for CS1/CS2 students. Although the module covers parallel computing concepts before moving on to processor heterogeneity, it is ideally suited for a course with some coverage of parallel computing material. For example, a CS1 course that incorporates a PDC module from [**?**], [**?**], or [**?**]. In the absence of

PDC coverage, the length of this module may need to be increased.

4.2. Topics

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

- Concurrency and Parallelism [K]
- Multicore Processors [K]
- GPGPU [K]
- System-on-Chip (e.g., mobile processors) [K]
- Energy Efficiency [K]
- Tasks and Workloads [K]
- Task Mapping and Scheduling [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 heterogeneous computing system
2) describe and distinguish between different forms of heterogeneity
3) understand the motivation behind the design of heterogeneous computing systems
4) recognize the importance of energy efficiency on current computing systems
5) understand that tasks in a workload have different demands for compute and memory resources
6) understand the notion of task mapping as performed by an operating system
7) analyze the performance and energy effects of task mapping on a heterogeneous system

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

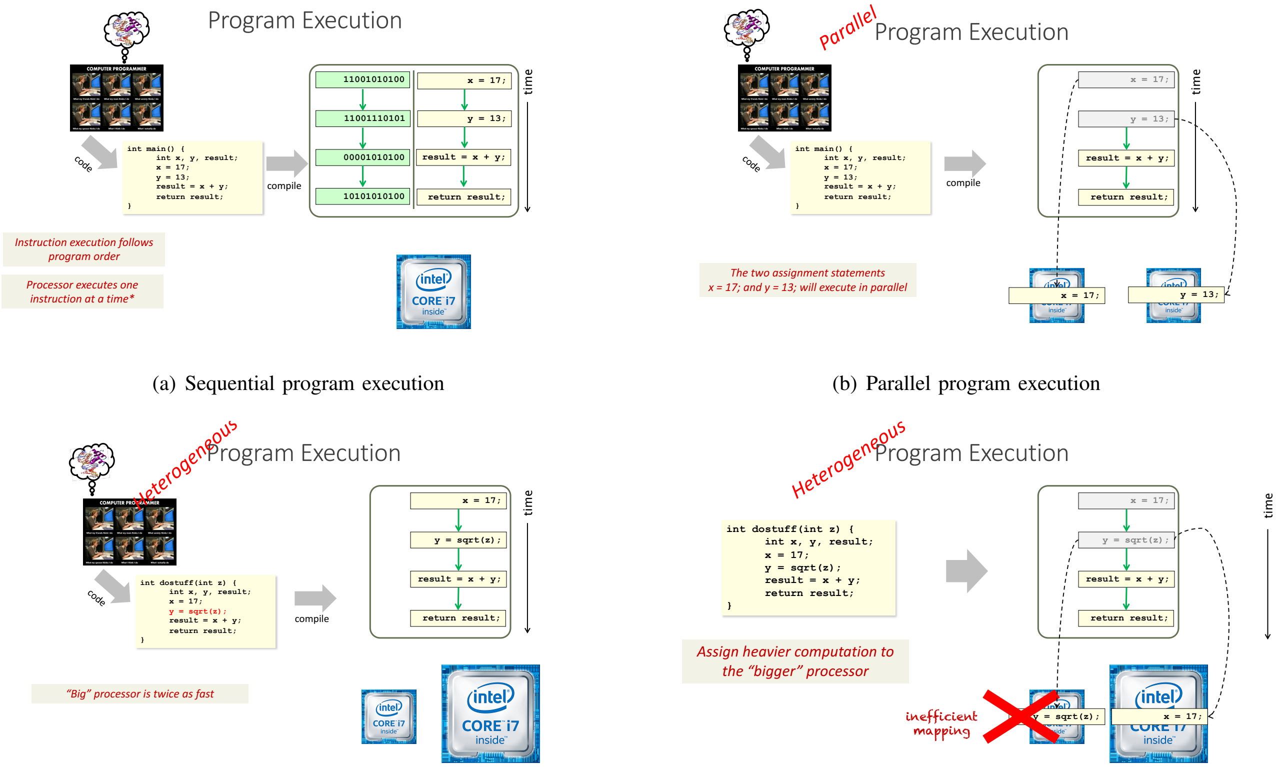
1) We need different types of computers to perform different tasks.
2) Although there are many different types of computing devices in use today, the fundamental organization of these devices remains the same.

4.4.2. Parallel computing and its importance today. A set of lectures 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 mainframe, vector computers and clusters are presented. This is followed by a discussion of multicore computers of today. 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 on their smartphones and tablets. Their guesses are then validated 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 is 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 the block diagram 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 is discussed, followed by a description of the CPU-GPU heterogeneous compute node. 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 student to the fundamental difference in sequential, parallel and heterogeneous program execution. A walk-through example is used for this purpose. Fig. **??** 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 [**?**]. Two such examples are shown in Figs. **??**-**??**. SimPar is a simple macro language that uses an intuitive pragma based syntax. Since students are generally not 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 chapter includes a SimPar parser that can be used to create other



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

```
11001010100    x = 17;
11001110101    y = 13;
00001010100    result = x + y;
10101010100    return result;
```

Instruction execution follows program order

Processor executes one instruction at a time*

(a) Sequential program execution

```
x = 17;
y = 13;
result = x + y;
return result;
```

The two assignment statements x = 17; and y = 13; will execute in parallel

(b) Parallel program execution

```
int dostuff(int z) {
      int x, y, result;
      x = 17;
      y = sqrt(z);
      result = x + y;
      return result;
}
```

"Big" processor is twice as fast

(c) Heterogeneous program execution

Assign heavier computation to the "bigger" processor

(d) Heterogenous program execution

Figure 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

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 *small* 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.

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

Figure 2. A simple parallel code written in SimPar

4.4.5. Programming tools. Students are told that SimPar is not a real language. The syntax for 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

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

Figure 3. Incorrectly parallelized code

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. This can be 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.

4.6. 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 is 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. Below we provide a description of the lab handout distributed to the students.

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

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

```
/usr/local/bin/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?

5. Evaluation

This module was implemented in a CS1 course at Texas State University in Fall 2018. 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 module was taught was designated as an Honors section. The enrollment in the Honors section is selective. The class is capped at 20 and only students with a strong academic background are allowed to enroll. The class comprises of both majors and non-majors. In Fall 2018, 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 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.

We designed a set of exam questions to assess student learning outcomes 1-5 (§ **??**). One of these questions was selected for the final exam in Fall 2018 to assess student comprehension of the covered material. Fig. **??** shows student grade distribution on this question. 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 taken into context. The class grade distribution is shown in Fig. **??**. As we can see, the distribution does not follow the normal curve and is skewed to the left. However, the cumulative grades closely match the grades on the module exam question. This indicates that the students did not find it significantly more difficult to understand the material associated with the module, as compared to the regular material covered in the class.

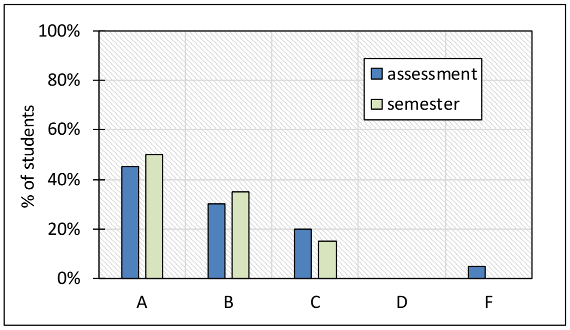


Figure 4. Semester grades and assessment question grade distribution

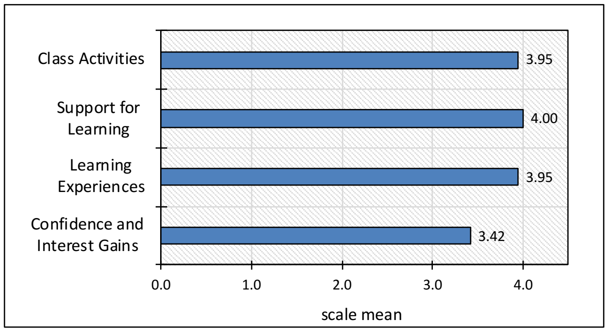


Figure 5. Student learning experience, confidence and interest gains

Learning outcomes 6 and 7 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 the class). All submissions received at least a B which is typical of lab assignments in that class.

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 [**?**]. 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?
(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). The results of the survey are shown in Fig. **??**. 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.

6. Conclusions and Future Work

This paper presented a teaching module for exposing CS1 students to heterogeneous computing. The module covers fundamental HC topics at a high level of abstraction but this coverage is complemented with a hands-on assignment that allows students to reinforce their understanding by conducting experiments on real hardware. Preliminary evaluation is promising both in terms of student learning outcome and engagement. Notwithstanding, the module has only been implemented in an Honors section of CS1 which generally boasts students with high aptitude. In future, we plan to calibrate the content such that it can be used in a regular section of CS1. We also plan to implement other HC modules focusing on algorithms and architectures that can be integrated into lower-division courses.

Acknowledgments

This work was supported by the National Science Foundation through awards CNS-1253292 and OAC-1829644. The author would like to thank David Bunde and Phil Schielke for reading an early draft and providing valuable feedback. The author also thanks the anonymous reviewers for their comments which helped improved the final version of the paper.