Experience and Practice Teaching an Undergraduate Course on Diverse Heterogeneous Architectures

Eitan Frachtenberg
Department of Computer Science
Reed College
Portland, Oregon, 97202
Email: eitan@reed.edu

***Abstract*—Heterogeneous computing is growing as an important hardware and software paradigm, both in high-performance computing and in application computing in general. Nevertheless, the topic is a relative newcomer to undergraduate curricula, and there is a dearth of guidance on suitable syllabi and lesson plans. The educational challenge of teaching this topic is exacerbated by the rapid pace of heterogeneous-hardware innovation and adoption, which can render parts of current textbooks obsolete.**

To help other educators facing these challenges, and to promote a conversation about a more standardized approach toward teaching heterogeneous computing, this paper presents a case study for one semester-long class on the topic. It describes the goals, structure, challenges, and lessons learned from the introduction of a diverse heterogeneous hardware and software environment to computer science majors at Reed College, a small liberal-arts school. This paper also includes suggestions and ideas for future adoption, adaptation, and expansion of this class.

***Index Terms*—Heterogeneous computing, High-performance computing, Accelerator architectures, Computer science education**

I. Introduction

Heterogeneous computing is no longer a futuristic research concept in computer architecture. Because of the slowing of Moore's Law, large performance gains can no longer be obtained from "more of the same," i.e., more cores and more cache memory per processor die [1]. Instead, significant power and performance gains are obtained by customizing entire processors or parts therein to specialized tasks, such as matrix multiplication, encryption, compression, and graphics.

Consequently, heterogeneous computing is rapidly growing mainstream across all segments of computing. On the high-end, high-performance computing (HPC) systems have been increasingly relying on accelerator architectures such as graphical processors (GPGPUs) for nearly a decade [2], [3]. In the most recent list of the fastest 500 supercomputers in the world, 149 systems relied on accelerator architectures, 140 of which used Nvidia graphical processor units (GPUs) [4]. At the same time on the low end, ubiquitous ARM-based smartphones have been taking advantage of the big.LITTLE heterogeneous architecture [5], [6]. Even mainstream computing has recently turned to heterogeneous processors to reap significant performance-per-Watt benefits, with Apple's introduction of the M1 processor [7] and Intel's upcoming 'Lakefield' architecture [8].

The upshot of this transition from homogeneous to heterogeneous architectures is that the established programming model of symmetric multi-threading and multi-processing is no longer adequate to exploit the full potential of modern computing architectures. Just like the transition from single-core to multi-core architectures necessitated new tools, training, and programming models to bring software developers up to speed [9], so does the current transition [1], [10]. And just like teaching parallel programming became a requisite part of many undergraduate curricula [11], [12], so will likely be heterogeneous computing, as we are starting to see in recent studies [13], [1].

This paper describes one such class, taught by the author at Reed College during the 2021 Spring semester. Reed College is a liberal-arts undergraduate school in Portland, Oregon. The computer science (CS) department at Reed College is only a few years old, and the CS curriculum is still undergoing active development and expansion. This class was the first formal offering with a syllabus centered on accelerator architectures. The class involved both theoretical and practical aspects, including individual projects employing parallel and distributed computing in a heterogeneous platform, representative of several HPC applications. The only prerequisite for this class was a 300-level computer systems class—mandatory for all CS majors—that advances the concepts and practice of computer architecture and systems programming. This junior-year class expands beyond the CS-2 class, which already introduces some systems and parallel programming, following recent trends in CS education [14].

The rest of this paper is organized as follows. We start by reviewing the class' unique experimental platform and its software environment in Sec. II. The next section (III) details the structure of this class, followed by a section (IV) discussing the learning goals, their evaluation and outcomes. Section V then enumerates and elaborates on the particular challenges that this class presented for teaching heterogeneous systems programming, and what lessons might be gleaned from these challenges and their solutions. Perhaps the largest challenge to adopting and expanding this course in other institutions may be scaling it to larger class sizes. The factors and mitigation suggestions for this scalability challenge are presented in Sec. VI. Finally, we conclude in Sec. VII.

II. Experimental platform

A. Hardware environment

An explicit goal of this class was to blend research aspects of diverse heterogeneous architectures with hands-on learning of the practical aspects of programming such systems. For the latter, access to a diverse experimental platform was key, so a custom-built computer was assembled from parts during the winter of 2020. The backbone of the computer comprised off-the-shelf components: an X570-based motherboard, 48GB of 3,000MHz DDR4 RAM, a 500GB NVME PCIe boot drive, a 3TB 5,400 RPM data hard drive, and an AMD Ryzen 3900 desktop CPU. This CPU runs at up to 4.2GHz, employing 12 cores and 24 threads in a 65W power envelope. Its architecture is not dissimilar to many of the modern HPC architectures based on AMD platforms [15].

On top of these components, the computer included three heterogeneous computational devices (accelerators):

1) An RTX 3090 GPU card with 24GB of RAM, the flagship consumer GPU card from Nvidia available at the time of this writing [16]. This card can be used as an accelerator using toolchains such as Nvidia's CUDA [17].
2) A Mellanox BlueField SmartNIC (P/N: MBF1M332A-ASCAT). This network interface card (NIC) supports two 25GbE ports over PCIe Gen 4 interface, with specialized hardware acceleration for cryptography. More importantly, it contains a fully-programmable processor with 16 ARM A72 cores at 1.3GHz (64-bit) and 16GB of RAM. It runs a fully functional operating system (OS), Ubuntu 20.04. It can be logged into via a PCI shim network interface from the host, and then programmed to execute arbitrary code [18]. Mellanox products are particularly interesting from an HPC perspective, because about a third of the top-500 supercomputers employ them, albeit not in their smartNIC capacity yet.
3) An NGD Systems "Newport" 8TB computational storage device (CSD). This device "looks and feels" like a normal PCIe gen 4 SSD drive to the host machine, except it additionally contains a 4-core/2GB ARM CPU in an 8W power envelope. It is also capable of running a complete OS (Ubuntu 16.04, later upgraded to 20.04), and can be logged into and programmed via a network interface over PCIe [19]. CSDs have also been recently evaluated as accelerators for HPC applications [20], as well as for applications such as artificial intelligence, machine learning, and big-data analytics [21], [19], [22].

In addition to this machine, we built a second machine for comparison and capacity purposes with an identical CPU, an entry-level GPU from AMD, and a Mellanox BlueField2 card (P/N: MBF2M332A-AEEOT). This second-generation "SmartNIC" uses a processor with 8 ARM A72 cores at 2.5GHz and faster hardware accelerators, but an otherwise identical software and development environment. We preferred the Bluefield1 card on our main evaluation machine because its cooling requirements were simpler. We did expect to compare performance across the two architectures, but ended up using only the first machine for the duration of the semester.

In terms of cost, we estimate that the components for each computer would retail for a total of approximately $7,500, although we were fortunate enough to receive accelerator parts at subsidized cost from Mellanox and NGD Systems.

Another device was considered, to round up accelerator computing, a field-programmable gate array (FPGA) card. Such an accelerator would represent reconfigurable computing, an important aspect of heterogeneous computing. But the difficulty of programming these devices for uninitiated CS students was deemed too high to accomplish in one semester. The added cost was also a limiting factor in this decision.

B. Software environment

The host ran the Ubuntu 20.04 operating system (OS), with a typical development toolchain (GCC, Clang, Rust, Python, and Java). The NIC also ran Ubuntu 20.04 with a similar toolchain, while the CSD ran Ubuntu 16.04 (later upgraded to 20.04 for compatibility). Students were given login credentials on all three (the GPU does not support a complete OS). They have not been given root access, however, to encourage them to build and install software dependencies from source, in their local directories. This constraint complicated the installation of software prerequisites, but provided valuable lessons in software portability, configuration management, package-management, and installation. Further, students' home directories from the host were mounted via network file system (NFS) on both the NIC and CSD, to avoid redundancy and to get around the extremely limited storage space on the NIC (16GB of slow flash).

III. Course structure

Seven students enrolled in this class, five seniors and two juniors.[1] The students had all previously taken CS389 "Computer Systems" as a prerequisite for this class. CS389 is also a requirement for the CS major, and introduces low-level programming and the interactions between software, hardware, and middleware.

The class met three times per week for hour-long sessions. The 14-week course proceeded in four distinct stages, each with its own format and student interaction.

A. Introductory lectures

The first week included three lectures, starting with the emerging role of heterogeneous architectures—in particular, GPUs—and their benefits and challenges. The second lecture described the software and hardware architecture of the smart-NIC and its potential use cases. Similarly, the third lecture focused on the architecture and applications of the CSD. Much of the material covered in these lectures came from the bibliography in this paper.

[1] This number likely affected by COVID-19 and the in-person requirements, but is not that unusual for upper-level classes at Reed College, where the students-to-faculty ratio is 9:1.

B. Seminar

The next three weeks were conducted as an informal research seminar, reading recent studies on accelerator architectures and use cases. First, the teacher lectured on two papers [19], [23] to exemplify the format and expectations for students who had not previously participated in a research seminar. Then, students gave two lectures in rotation on two different accelerator architectures, each on a recent paper of their choice. (All the chosen papers describe either general frameworks for application offloading or specific use cases, and appear in the bibliography [24], [25], [26], [27], [22], [28], [29], [30], [31], [32], [33], [34], [20], [35].)

In-class presentation included approximately 20–30 minutes of reviewing the main ideas and results of the paper, followed by a lively discussion of their merits and applicability to our experimental platform. Each student was also asked to critique the paper they chose and suggest improvements to its methodology or analysis. The goal of this discussion was not only to expose students to the latest research on accelerators, but also to develop their critical thinking skills when reading systems literature, focusing especially on the various stated and unstated assumptions that come with the evaluation of a complex computer system.

C. Benchmarking

During the subsequent three weeks, the class shifted gears from the theoretical and research aspects of heterogeneous systems to their use and performance, by benchmarking the actual experimental platform.

The benchmark suites and tested architectures included:

- fio: I/O performance (CPU, CSD, NIC).[2]
- OpenBenchmarking: Scientific computing (CPU, GPU).[3]
- LLCBench: Low-level architectural characterization (CPU).[4]
- IOZone: filesystem benchmark (CPU, CSD).[5]
- Sysbench: Database benchmark (CPU, CSD).[6]
- RDMA-bench: RDMA performance (NIC).[7]
- Terasort: Hadoop I/O and sorting benchmark (CPU, CSD).[8]
- NAS Parallel Benchmarks: multithreaded scientific application benchmark (CPU).[9]

To be clear, the primary goal of this phase was not necessarily to produce the best performance numbers. There was no reason to spend extensive time tuning the benchmarks, and a raw performance comparison across accelerator architectures is not very meaningful to begin with, since they each have different strengths. Instead, the learning objectives included:

- Obtaining, configuring, building and installing prerequisite libraries across platforms, especially with no superuser access.
- Understanding the effects of workloads on performance and identifying appropriate workloads for each platform.
- Identifying and troubleshooting measurement anomalies and inconsistencies.
- Normalizing performance results by power usage to level the playing field across different accelerator architectures. Students measured the marginal power usage of their benchmarks (and therefore their energy consumption) using a WattsUp power meter.
- Communicating their performance comparisons effectively, using tables, figures, and prose.

All students posted their benchmark parameters and results on a cloud-shared spreadsheet that we reviewed in class. During classroom meetings, students engaged in open discussion, sharing questions, stumbling blocks, and results with each other. The interactive and informal team setting allowed everyone to absorb lessons learned from platforms and programming languages different than their own.

D. Individual projects

Throughout the second half of the semester, students had the opportunity to engage in hands-on development for accelerator architectures. During the seminar phase of the course, each student was also asked about the potential relevance of their research paper to our experimental platform and what projects could grow out of this work. Some of the students chose projects that were indeed inspired by these studies. Others picked projects based on their own ideas or the teacher's. The class continued to meet regularly with the same informal format as during the benchmarking phase. All students were expected to participate and report regularly on their progress.

At the end of the semester, in lieu of a final examination, students were asked to write a final report on their project. The report was not restricted to a particular format or length, but was expected to discuss the following aspects:

1) Background and bibliography on the problem.
2) Technical description of proposed solution on accelerator architecture(s).
3) Evaluation (correctness, performance, and energy).
4) Conclusion.

Selected projects: The complete list of projects that the students chose was as follows.

- A user-level filesystem on the NIC that mirrors a directory from the host's disk, using RDMA as the back-end communication protocol. Unlike NFS that was used to mount users' home directories on the NIC and CSD, this filesystem bypasses the TCP/IP stack (and the kernel), providing faster access to the host's files.[10]
- An implementation of the parallel prefix algorithm on the GPU. Although such implementations already exist (as

[2]https://github.com/axboe/fio
[3]https://openbenchmarking.org/
[4]https://icl.cs.utk.edu/llcbench/index.htm
[5]http://www.iozone.org/
[6]https://github.com/akopytov/sysbench
[7]https://github.com/efficient/rdma_bench
[8]https://www.systutorials.com/hadoop-terasort-benchmark/
[9]https://www.nas.nasa.gov/publications/npb.html

[10]The source code and final report for this project are available as a sample of the students' work at https://github.com/iwahbe/rdma-fs.

part of NVidia's software-development kit, for example), the student pursued his own implementation as the means to learn about low-level details of CUDA programming and performance optimization.
- Two chess-related projects. One student implemented their own Minimax game-tree search algorithm, and ported it to the GPU, comparing its performance with the CPU. Another student ported the well-known open-source Stockfish chess engine [36] to all four computational platforms and then pitted them in tournament against each other. To keep things fair, each platform (processor) was given a play time budget that was normalized by its power consumption running Stockfish (which the student measured). Consequently, each platform was using up a similar amount of energy for the duration of the game, exposing the relative difference in processor performance on an energy-adjusted basis.
- Three differential privacy (DP) implementations for the CSD. The idea is that some sensitive data needs to be shared for statistical analysis while preserving the privacy of individual data points (persons). DP algorithms enable this functionality by adding noise to typical statistical queries such as mean, sum, and count, so that inferring individual data becomes statistically infeasible. DP implementations typically rely on a private server that only permits access to statistical queries over the network, but not to the raw data. The motivation behind these projects is that by putting this server on a CSD (fully supporting both host-sourced and networked queries), the data can be physically handed over on a device to its consumer, but they would still only be able to access it via protected queries. All the raw data would be cryptographically secured and available only to the CSD's processor and server. Each of the three projects chose a different existing DP library implementation (and programming language) to port to the CSD, but they all used the same workload and queries to compare performance across libraries.

Some of the other ideas for projects that were suggested or considered include:

1) A global lock-free task queue, shared across all four devices, for coordinated task scheduling and task stealing.
2) A NIC-based key-value server accessing data from either the host's RAM or the CSD's backing store, using RDMA.
3) Porting parts of the OpenCL[11] library to the NIC and CSD to streamline accelerator support.
4) Evaluation of the hardware encryption or compression engines of the NIC/CSD and rolling out an application atop this layer.
5) Regular-expression search acceleration, across the network (NIC) or disk (CSD).
6) A performance comparison of optimized deep-learning libraries on all four platforms.
7) Stateful, application-level packet filtering on the NIC.
8) Offloading I/O-heavy JOIN operations from a simple SQL database to the CSD.
9) Porting any MPI application to run concurrently on the CPU, CSD, and NIC.
10) Offloading various micro-services to the NIC, as suggested in several studies [37], [38], [30].

[11]https://www.khronos.org/opencl/

IV. Learning goals and outcomes

This course was designed with three main learning goals, to complement the foundational curriculum with a more advanced perspective on the heterogeneous specialty.

The first goal, addressed in the first two parts of the course, was to understand the challenges and potential applications of heterogeneous computing through exposure to a breadth of recent academic research. After reading, presenting, and learning about sixteen recent research efforts, the students expressed an appreciation of the importance of the problem domain, where it fits within the rest of computer science, and what the typical structure of a systems research project looks like.

The second goal, addressed in the third and fourth parts of the course, was to build confidence and independent practice with the tools of heterogeneous computing through the blending these research aspects with hands-on work. Given the limitations of a short semester and few prerequisites, this goal was only partially accomplished. Some students, particularly those with strong programming skills, were able to focus their attention and learning on the new systems aspects, while others were sidetracked by more mundane programming challenges such as debugging memory access. That said, even for these students, the first-time exposure to benchmarking, library compilation, program installation, and power measurements, appeared to have expanded their comprehension and independence with systems programming tools.

The final goal, addressed throughout all sections of the course, was to develop familiarity and appreciation with the typical workflow in systems research: problem statement, solution engineering, implementation and debugging, performance and power evaluation. Based on the final reports the students submitted, they have ostensibly understood and adopted this workflow in their own projects.

These three goals may have been ambitious for the time frame and given the course's modest prerequisites. In a graduate-level curriculum or in a school with a larger CS program, it may make more sense to split this course into two: a research seminar, and a lab-based class (optionally with higher prerequisites). However, this class attempted to blend both aspects of exposure to systems academic research and hands-on skill-building, which would not be otherwise available in a separate class to Reed students. I believe that the current structure represents a reasonable compromise between the two, and worked well for most students.

Overall, students appeared to concur. Although the results of the end-of-class student evaluation forms were not statistically

significant, owing to the small size of the class, all respondents agreed that the course encouraged them to think creatively, critically, and/or deeply about the subject. All respondents also strongly agreed that the conference lab discussions were facilitated effectively. Some of the positive feedback students offered at the conclusion of the class were:

> "I liked how we explored a lot of topics, both in the papers and in the projects, and learned a wide variety of things relating to systems."
> "Letting us have free reign was really nice, it let us explore a lot with our respective topics."
> "These [sic] was a topic class. It was mostly project driven, so I liked my project and thus the class. It would be great to do paper reading before 400 level classes, but I appreciated getting to it eventually."

On the critical side, one student felt that they "didn't get much out of my project as a result of not putting much in". A related suggestion was to "add a few more written assignments, small checkin ones, to prepare the students to know what to write for the final assignment". Another student was annoyed by the various technical glitches with the experimental platform, which are addressed in more detail in the next section.

V. Challenges and lessons learned

This section reviews some of the idiosyncratic challenges faced by the students or teacher with this class format, in no particular order. Where applicable, it also presents alternatives and suggestions for the next iteration of the class.

a) NFS mounts: Both the NIC and CSD accelerators had their own OS and local storage, which meant that students could copy over any files to the accelerator to write and measure their code. But for convenience and for practical concerns, primarily low storage space, all student home directories were mounted on the accelerators using the NFS protocol from the host [39]. This mechanism meant that students only had to work on their files on one platform (typically the host), and have them automatically available on the accelerators. This file sharing worked well for most instances. But because files were mounted over a relatively slow file system, atop a relatively slow shim network, I/O performance was significantly slower on the accelerators for home directories. The delays were not prohibitive for development work, but were clearly not representative for benchmarking work. Once the students identified this cause for slow performance, they performed all subsequent I/O tests on the local file system on each accelerator.

b) Superuser access: The decision to withhold superuser privileges ("root") from students was meant to encourage students to download, configure, and if necessary, port and install local copies of software dependencies. Students are rarely expected to learn these skills in undergraduate classes, even though they can be essential in the real-world environment of security-sensitive deployments. The main disadvantage of this decision, however, is that some dependencies simply do not work without root privileges, and others are clunky.

For example, vendor-specific dependencies such as Nvidia's CUDA[12] and OFED[13] require system-wide installation to work well. Some students also needed to configure the machine's network interfaces, which requires root privileges.

Another interesting example was Python libraries with binary components, such as numpy.[14] On the one hand, these libraries are designed to work well in local installations (using tools such as Python's virtual environment, *venv*). On the other hand, because of the NFS sharing of home directories, these local installations shared files across hosts. For those libraries that incorporate a compiled binary component, the binary is then shared across incompatible architectures (x86-64 / AArch64), which inevitably does not work.

One simple solution to this problem is to avoid file sharing across the host and accelerator and duplicate all the files across architectures. This leads to space waste, redundancy, confusion, and errors. Worse still, it may simply not be practical on accelerators with extremely low storage, such as the SmartNIC. Another solution, which we implemented in class, is to install only the binary dependencies at the system level (using Ubuntu's *apt* tool). This compromise did require the use of root privileges by the teacher, but did not affect the students' learning goals of porting and installing source-based dependencies. In retrospect, it would have made more sense to preinstall common system-level libraries before the course began and eliminate this needless stumbling block.

c) Software compatibility and upgrade woes: As described in Sec. II, the host CPU and the SmartNIC were running Ubuntu 20.04, while the CSD was running Ubuntu 16.04. This mismatch became a problem when one of the projects required a recent version of Python, which simply would not install cleanly on Ubuntu 16.04. NGD Systems responded quickly by providing an experimental OS image with Ubuntu 20.04, and by assisting through several iterations of configuration, installation, and troubleshooting.

Each one of these iterations required erasing and rewriting the main partition on the CSD's drive. This presented only a minor hassle to students, since their home directories resided on the host's drive, mounted over NFS, and they could continue development on the similar architecture of the NIC. But in one of these iterations, the host's main drive was mistakenly selected for erasure instead of the CSD's. Fortunately, a data-loss disaster was averted, because the *'/home'* partition was safely separated from the *'/'* partition on the host. This was a valuable, albeit inadvertent, lesson on the configuration of such a host machine. This mistake nevertheless caused delays, because the host's OS had to be reconfigured, which took several hours.

During this process, it was very useful to follow and update a detailed set of notes on the installation and configuration of the host OS (and each accelerator's). Another obvious takeaway is to keep frequent and remote backups. In our case, most

[12] https://developer.nvidia.com/cuda-toolkit
[13] https://developer.nvidia.com/networking/infiniband-software
[14] https://numpy.org/

students kept repositories of their code on github.com, with fairly recent changes pushed. But a more systematic backup solution may be advisable and likely simple to implement.

d) Hardware compatibility: The fact that three of the platforms provided the same OS did simplify code portability across these platforms. In most cases, code simply had to be recompiled and then ran as-is. One compatibility issue arose in the FUSE/RDMA project, when raw binary data structures (using C's `union` structure) was transferred in-memory back and forth between ARM and x86, requiring some debugging because of potential differences in endiannes.

Another hardware issue arose from the early development status of the CSD product. Its hardware support for cryptographic acceleration, which could have leveraged the differential-privacy projects, had still not been exposed to programmers via application-programmer interfaces or libraries, and remained unusable for the duration of the semester.

Finally, the three GPGPU programmers required slightly different CUDA versions each, which caused some minor incompatibility issues.

e) CUDA complexity: Those three students who chose to port or write code for the GPGPU were expecting their project to make rapid progress, since GPGPUs are ostensibly the most mainstream accelerator, and therefore presumably the most established one to develop for. But the lack of a standard OS on the GPU and the requirement for specialized programming idioms and tools presented a steep learning curve for these students with no prior relevant experience. Ab-initio CUDA training proved to be too big a task for half a semester, and none of these students fully met their initial GPGPU goals.

The inclusion of GPGPUs in this class, critical because of their importance in heterogeneous computing and HPC, likely means that adaptations of this class should modify its prerequisites or expectations. One such prerequisite could be a parallel programming class, which is offered in many undergraduate CS programs, including occasionally at Reed College. An even better prerequisite might be a specialized class in GPGPU programming, although such a class may be rarer and overlap too much with this class.

Admittedly, these are not very realistic prerequisites for a small undergraduate-only department. They may make more sense when adopting this class for a graduate program. For future iterations of this course at Reed College, it may make more sense to simply limit students' projects such that any project that uses the GPGPU only tailors and leverages existing libraries, unless the student has relevant prior experience.

f) COVID-19: Perhaps the most unusual challenge was teaching and learning during the height of a global pandemic. Naturally, the class faced the same challenges shared by all other undergraduate CS classes: remote students, communication breakdowns, increased overhead, and reduced effectiveness. In addition, the need to have physical access to the experimental platform (to measure its power consumption) posed additional obstacles for the students, who had to schedule exclusive time slots in the machine room. This constraint could have severely limited the scalability and effectiveness of the class, but the small number of students meant that in practice, scheduling conflicts were rare, and local students often volunteered to carry out power measurement tasks for remote students.

VI. Scaling the class size

Responding to these challenges was helped by the small size of the inaugural class, but this is obviously not a generalizable solution for larger schools. Some of these challenges are alleviated simply with time and experience, such as the initial setup complexities. For other the challenges, the following suggestions could help scale this class to larger sizes.

a) Equipment cost: One major limitation to scaling is the necessary lab equipment. A single computer or accelerator can only serve so many students (probably ten or fewer), so more equipment needs to be acquired for larger class sizes. As mentioned earlier, two computers were set up for this class, but the small number of students was adequately served with just one.

This course used bleeding-edge equipment that is not necessary for achieving its learning goals. GPU programming is not substantially different between a $2,000 card and a $400 one. Nor is there a meaningful difference when using entry-level CSD and SmartNIC. Acquiring previous-generation or lower-end equipment may be a cost-effective way to scale the equipment for this class. The performance obtained may not be as impressive or representative, but maximizing performance numbers is not one of the stated learning goals of the course. The same argument can be made for allowing the platform to age for two-to-four years before budgeting for replacement.

b) Technical support: A significant portion of the students' time, as well as the instructor, was spent on configuring, debugging, and patching the experimental platform. The time spent on this task is not productive towards the learning outcomes of the class, and likely grows with the number of students. On the other hand, much of this effort should not recur from semester to semester, and may subside substantially after the teething troubles of the initial course run. The setup and administration effort may also be reduced by acquiring more mature accelerators instead of cutting-edge pre-production parts, as suggested for lowering costs as well.

c) Attention to individuals: For modestly larger class sizes (say, up to 20–30 students), much of the original class structure should scale successfully with little change. The second part of the class (research seminar) can be modified such that each student only presents a single paper, or even half a paper (in pairs). It can also be overlapped with the third part (benchmarking), which tends to require more offline work than online presentation, and thus be extended through the middle of the course. The fourth part, projects, rarely required the full 50 minutes of lecture time to review all seven projects, especially since not all students could produce reportable progress three times per week. It is therefore expected that moderately increasing the number of students could work without altering the format of this part.

For much larger classes, this course will probably not scale as is. Much of the classroom interaction relies on the familiarity of every student (and instructor) with everyone's benchmarks and projects. There are also frequent technical stumbling blocks that require instructor assistance, which would represent a bottleneck beyond a certain class size. This is not dissimilar from the difficulties in scaling other lab-intensive science classes [40], and may require the same solutions and class structures that work for such classes, including splitting students into smaller sections and employing advanced students and lab assistants.

VII. Conclusion

This paper described a new course at Reed College's Computer Science department that introduced undergraduate students to the theoretical and practical foundations of heterogeneous programming. The class integrated traditional lectures on heterogeneous computer systems, student-driven reading and discussion of recent research papers, actual application of performance evaluation techniques and analysis, and hands-on development on diverse accelerator architectures.

Despite the numerous technical challenges posed by the experimental hardware, and perhaps because of them, students remained highly motivated and engaged. Year-end feedback from students included an appreciation of the wide choice of topics students had at every phase of the course and the inclusion of nonacademic practical skills in programming, benchmarking, and system administration.

This paper includes bibliography and linked resources to reproduce the material and curriculum for this class. Hopefully, these materials can serve to develop and evolve similar classes on this emerging and increasingly important aspect of high-performance computing.

Acknowledgments

Both NGD Systems and Mellanox provided detailed and responsive technical support for their products. This course would have likely failed without the tireless and patient support of Alexey Senin from Mellanox and Ashok Savdharia from NGD Systems. I would like to specifically thank Michael Kagan of Mellanox and Mike Yousef of NGD Systems for their commitment and prompt assistance to procure these early products, despite the small scale of the class. I am also grateful for Scott Meyers' feedback on an early draft of this paper.

References

[1] L. P. Carloni, E. G. Cota, G. D. Guglielmo, D. Giri, J. Kwon, P. Mantovani, L. Piccolboni, and M. Petracca, "Teaching heterogeneous computing with system-level design methods," in *Proceedings of the Workshop on Computer Architecture Education*, Jun. 2019. doi: 10.1145/3338698.3338893 pp. 1–8.

[2] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational solutions to large-scale data management and analysis," *Nature reviews genetics*, vol. 11, no. 9, pp. 647–657, 2010. doi: 10.1038/nrg2857

[3] "Highlights from the top-500 list," 11 2011, retrieved May 22, 2021. [Online]. Available: https://www.top500.org/lists/top500/2011/11/highlights/

[4] "Top500 expands exaflops capacity amidst low turnover," 11 2020, retrieved May 22, 2021. [Online]. Available: https://top500.org/news/top500-expands-exaflops-capacity-amidst-low-turnover/

[5] H. Chung, M. Kang, and H.-D. Cho, "Heterogeneous multi-processing solution of exynos 5 octa with arm big. little technology," *Samsung White Paper*, 2012.

[6] K. Yu, D. Han, C. Youn, S. Hwang, and J. Lee, "Power-aware task scheduling for big. little mobile processor," in *2013 International SoC Design Conference (ISOCC)*. IEEE, 2013. doi: 10.1109/ISOCC.2013.6864009 pp. 208–212.

[7] E. Engheim, "Why is apple's m1 chip so fast?" 2020, retrieved May 22, 2021. [Online]. Available: https://debugger.medium.com/why-is-apples-m1-chip-so-fast-3262b158cba2

[8] W. Gomes, S. Khushu, D. B. Ingerly, P. N. Stover, N. I. Chowdhury, F. O'Mahony, A. Balankutty, N. Dolev, M. G. Dixon, L. Jiang, S. Prekke, B. Patra, P. V. Rott, and R. Kumar, "8.1 lakefield and mobility compute: A 3d stacked 10nm and 22ffl hybrid processor system in 12×12mm¡sup¿2¡/sup¿, 1mm package-on-package," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020. doi: 10.1109/ISSCC19947.2020.9062957 pp. 144–146.

[9] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's journal*, vol. 30, no. 3, pp. 202–210, 2005. [Online]. Available: http://www.gotw.ca/publications/concurrency-ddj.htm

[10] A. Dubey, K. M. Riley, and D. E. Bernholdt, "Teaching software sustainability for high performance computing at atpesc," in *2020 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*. IEEE, 2020. doi: 10.1109/EduHPC51895.2020.00008 pp. 19–24.

[11] C. W. Kessler, "Teaching parallel programming early," in *Proceedings of the Workshop on Developing Computer Science Education*, 2006.

[12] A. Marowka, "Think parallel: Teaching parallel programming today," *IEEE Distributed Systems Online*, vol. 9, no. 8, p. 1, 2008. doi: 10.1109/MDSO.2008.24

[13] D. P. Bunde, A. Qasem, and P. Schielke, "Teaching about heterogeneous computing," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021. doi: 10.1145/3408877.3432506 pp. 1355–1355.

[14] J. C. Adams, "Injecting parallel computing into CS2," in *Proceedings of the 45th ACM technical symposium on Computer science education*, 2014. doi: 10.1145/2538862.2538883 pp. 277–282.

[15] P. Rojas, C. J. Barrios-Hernandez, and L. A. Steffenel, "Low energy consumption on post-moore platforms for HPC research," in *Advanced and High Performance Computing Trends in Latin America Workshop*, 2020.

[16] A. Watson, "Deep learning techniques for super-resolution in video games," *arXiv preprint arXiv:2012.09810*, 2020.

[17] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[18] D. Kim, S. Lee, and K. Park, "A case for smartnic-accelerated private communication," in *4th Asia-Pacific Workshop on Networking*, 2020. doi: 10.1145/3411029.3411034 pp. 30–35.

[19] J. Do, V. C. Ferreira, H. Bobarshad, M. Torabzadehkashi, S. Rezaei, A. Heydarigorji, D. Souza, B. F. Goldstein, L. Santiago, M. S. Kim *et al.*, "Cost-effective, energy-efficient, and scalable storage computing for large-scale ai applications," *ACM Transactions on Storage (TOS)*, vol. 16, no. 4, pp. 1–37, 2020.

[20] M. Torabzadehkashi, S. Rezaei, A. HeydariGorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Computational storage: an efficient and scalable platform for big data and hpc applications," *Journal of Big Data*, vol. 6, no. 1, pp. 1–29, 2019. doi: 10.1186/s40537-019-0265-5

[21] K. Chapman, M. Nik, B. Robatmili, S. Mirkhani, and M. Lavasani, "Computational storage for big data analytics," in *Proceedings of 10th International Workshop on Accelerating Analytics and Data Management Systems (ADMS'19)*, 2019.

[22] A. HeydariGorji, M. Torabzadehkashi, S. Rezaei, H. Bobarshad, V. Alves, and P. H. Chou, "Stannis: low-power acceleration of dnn training using computational storage devices," in *57th ACM/IEEE Design Automation Conference (DAC)*, 2020. doi: 10.1109/DAC18072.2020.9218687 pp. 1–6.

[23] R. E. Grant, W. Schonbein, and S. Levy, "RaDD runtimes: Radical and different distributed runtimes with smartnics," in *2020 IEEE/ACM Fourth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware (IPDRM)*. IEEE, 2020. doi: 10.1109/IPDRM51949.2020.00007 pp. 17–24.

[24] A. Barbalace and J. Do, "Computational storage: Where are we today?" in *Conference on Innovative Data Systems Research*, 2021.

[25] W. Cao, Y. Liu, Z. Cheng, N. Zheng, W. Li, W. Wu, L. Ouyang, P. Wang, Y. Wang, R. Kuan *et al.*, "POLARDB meets computational storage: Efficiently support analytical workloads in cloud-native relational database," in *18th USENIX Conference on File and Storage Technologies (FAST'20)*, 2020, pp. 29–41.

[26] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*, 2014, pp. 401–414.

[27] Y. Go, M. A. Jamshed, Y. Moon, C. Hwang, and K. Park, "APUNet: Revitalizing GPU as packet processing accelerator," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)*, 2017, pp. 83–96.

[28] A. Kalia, D. Zhou, M. Kaminsky, and D. G. Andersen, "Raising the bar for using gpus in software packet processing," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*, 2015, pp. 409–423.

[29] K. Lee, M. B. Sullivan, S. K. S. Hari, T. Tsai, S. W. Keckler, and M. Erez, "GPU snapshot: checkpoint offloading for gpu-dense systems," in *Proceedings of the ACM International Conference on Supercomputing*, 2019. doi: 10.1145/3330345.3330361 pp. 171–183.

[30] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3: Energy-efficient microservices on smartnic-accelerated servers," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 363–378.

[31] S. A. Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography," in *2007 IEEE International Conference on Signal Processing and Communications*. IEEE, 2007, pp. 65–68.

[32] A. Mishra, L. Li, M. Kong, H. Finkel, and B. Chapman, "Benchmarking and evaluating unified memory for openmp gpu offloading," in *Proceedings of the Fourth Workshop on the LLVM Compiler Infrastructure in HPC*, 2017. doi: 10.1145/3148173.3148184 pp. 1–10.

[33] C. Mitchell, Y. Geng, and J. Li, "Using one-sided RDMA reads to build a fast, cpu-efficient key-value store," in *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, 2013, pp. 103–114.

[34] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc, "A performance analysis framework for identifying potential benefits in gpgpu applications," in *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, 2012. doi: 10.1145/2145816.2145819 pp. 11–22.

[35] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A smartnic-driven accelerator-centric architecture for network servers," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020. doi: 10.1145/3373376.3378528 pp. 117–131.

[36] M. Acher and F. Esnault, "Large-scale analysis of chess games with chess engines: A preliminary report," *arXiv preprint arXiv:1607.04186*, 2016. [Online]. Available: https://arxiv.org/pdf/1607.04186.pdf

[37] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum, "λ-nic: Interactive serverless compute on programmable smartnics," *arXiv preprint arXiv:1909.11958*, 2019. [Online]. Available: https://arxiv.org/pdf/1909.11958

[38] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. M. Swift, and T. Lakshman, "Uno: Uniflying host and smart nic offload for flexible packet processing," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017. doi: /10.1145/3127479.3132252 pp. 506–519.

[39] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, "NFS version 3: Design and implementation." in *USENIX Summer*. Boston, MA, 1994, pp. 137–152.

[40] J. D. Gaffney, E. Richards, M. B. Kustusch, L. Ding, and R. J. Beichner, "Scaling up education reform," *Journal of college science teaching*, vol. 37, no. 5, p. 48, 2008.