Employing Student Retention Strategies for an Introductory GPU Programming Course

Julian Gutierrez
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
jgutierrez@ece.neu.edu

Fritz Previlon
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
previlon.f@husky.neu.edu

David Kaeli
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
kaeli@ece.neu.edu

***Abstract*—Graphics Processing Units (GPUs) have become a vital hardware resource for the industry and research community due to their high computing capabilities. Despite this, GPUs have not been introduced into the undergraduate curriculum of Computer Engineering and are barely covered in graduate courses.**

Bridging the gap between university curriculum and industry requirements for GPU expertise is ongoing, but this process takes time. Offering an immediate opportunity for students to learn GPU programming is key for their professional growth.

The Northeastern University Computer Architecture Research Lab offers a free GPU programming course to incentivize students from all disciplines to learn how to efficiently program a GPU. In this paper, we discuss the methods used to keep students engaged in a course with no academic obligations. After applying these strategies, we were able to retain more than 80% of the students who started the course. Moreover, the students gave positive feedback on these strategies.

I. Introduction

Parallelism is the future of computing [1]. Over the last decade, studies including the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing [2] have reported that incorporating parallel and distributed computing concepts across the undergraduate computer science program is essential. Furthermore, the recent industry move towards parallel computing hardware and software demonstrates these changes are needed today.

Graphics Processing Units (GPUs) have become essential in today's computing paradigm. The recent performance advances provided by major chip manufacturers such as NVIDIA and AMD have significantly impacted the scientific and commercial computing communities. By increasing number of cores per chip, increasing memory bandwidth and improving power efficiency, GPUs have become extremely popular for a wide range of applications. These algorithms span a diverse range of areas, from deep learning, video tracking, and speech recognition, to materials modeling, physics simulation and bioinformatic applications.

Given this move to parallel hardware and software, there is clearly a fracture in the academic curriculum at universities. Industry and the research community are demanding more experts capable of writing efficient parallel algorithms targeting GPUs, but our universities are not meeting this demand.

Students from a range of disciplines are showing growing interest in GPU programming. As figure 1 suggests, during Fall 2017, an online survey was sent to students from Northeastern University to evaluate their interest in a GPU programming class. Over 80 students covering 14 different disciplines showed interest in taking this course. This indicates that students from a wide range of academic backgrounds were interested in GPU programming.

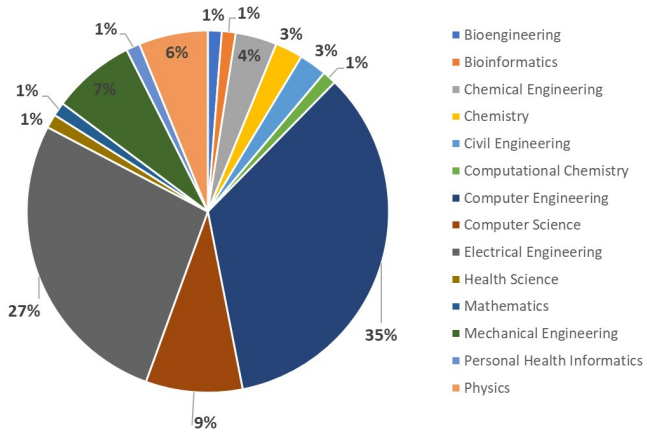

Figure 1. Distribution of disciplines from students interested in a GPU programming course at Northeastern University.

The Northeastern University Computer Research Laboratory graduate students have consistently offered a free GPU programming course every semester for more than 5 years. Despite student interest, only 5% of registered students who started the course remained to completion in previous semesters. Student retention had been a struggle because this course has no academic obligations. Through careful review, retention strategies were applied to specific segments of the course and proven to be effective for students through an anonymous evaluation, and obtaining +80% retention in the following semesters.

Our goals are summarized as follows:

- Foster an environment where students can start thinking in parallel.
- Increase the number of students that are competent in parallel programming with GPUs.
- Fill the gap between the skills required by the industry and academic offerings for students.
- Apply retention strategies to keep students interested and engaged when the class is not mandatory.
- Not compromise on the quality of the material taught to achieve these goals.

The remainder of this paper is organized as follows. In Section II, we discuss related work on parallel and GPU course work. In Section III, we discuss the methodology used for the course, including the retention strategies used. In Section IV, we review specific elements of the course in detail. In Section V, we discuss the qualitative results obtained from an anonymous survey done at the end of the course. And in Section VI, we conclude the paper and consider directions for future work.

II. Related Work

As the move toward parallelism continues to accelerate within industry, the curricula in computer-related fields at leading universities is still focused on using sequential programming. The NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing [2] promotes the adoption of a required parallel programming course in computing-based curricula across universities. Our main goal is to support this initiative, with an increased focus on GPU programming.

A number of approaches have been explored to incorporate parallelism concepts into already existing courses [3], [4]. Additionally, a number of unique classroom approaches have been explored to present the concepts of concurrency and parallelism [5], [6], [7], [8]. While we applaud these efforts, our specific focus is on GPU programming. We provide a half-semester course devoted to GPU programming that is accessible by all students, regardless of their technical backgrounds.

Advances in GPU performance have been achieved through new fabrication technologies and new architectural changes. A great example for this is the incorporation of tensor cores in the latest NVIDIA Volta architecture, offering the capability of achieving 120 TOPS with a single GPU. Due to these changes in GPU architecture, teaching material can easily become obsolete in 2 years. Material developed for classes across multiple universities needs to be constantly updated to incorporate these new features.

Despite the advances achieved at an architecture level, most universities do not have access to this hardware. Additionally, many of the principles involved in parallel programming have not changed with the architecture of the GPUs. Emphasizing the main principles of parallel programming, and providing hands-on examples that expose the architecture of the GPU to the student, has proven to be an effective way for teaching parallel programming [9], [1].

Previous studies have described effective teaching methods and focus on presenting proper gpu programming techniques [10], [11], [12]. While some of this prior work may be considered outdated, many of the concepts and lessons are still applicable in today's technology. Furthermore, NVIDIA provide a significant amount of material for teaching GPU computing through their *NVIDIA's Academic Programs* and their *CUDA Education and Training* program. Many of the examples covered in this course are inspired by their work and are developed considering the recommendations and guidelines found in the CUDA programming guide [13].

III. Methodology

This course is taught by graduate students from the Northeastern University Computer Architecture Research Lab. It is offered at no cost to the students, in an effort to expand and allow a wide variety of students to familiarize themselves with GPU and parallel programming. The course is centered around the CUDA API framework on C/C++ offered by NVIDIA. CUDA offers many advantages over other API's such as OpenCL. CUDA offers extensive online resources, and comes with the support from experts at NVIDIA and the broader community. Additionally, thanks to the simplicity of the CUDA API, students can focus on how to expose parallelism in their applications using CUDA, versus spending a large amount of time struggling to understand the API.

A. Learning Outcomes

In the class, the students learn how to interact with GPUs. We want them to learn how to write and design algorithms to develop efficient and high-performance programs that expose a GPU's full performance. At the end of the sessions, students should understand all the necessary steps that are required to translate an algorithm to GPU computing, as well as how to approach an optimization problem and use the appropriate tools to discover which are the best ways to optimize the application.

B. Syllabus

The course consists of twelve two-hour long sessions, covering all the basic concepts including GPU architecture, the CUDA programming model, the CUDA memory model and advanced topics. The topic presentation order has been designed to reinforce concepts learned in previous sessions with the new topics discussed in later classes. Significant effort has gone into aligning the hands-on labs to support the theory presented from the in-class sessions, in order for the students to retain all of the concepts covered in those sessions. Table I illustrates the schedule for the class, including the final project. Each of these sessions are discussed in detail in the following sections.

As a retention strategy, students are offered a certification of completion for the course if they are able to complete the following requirements:

Table I. Course content per session.

Session	Topics
1	Introduction Discovery Cluster Basics
2	Cuda Execution Model Architecture and NVVP and NVPROF
3	Lab 1
4	Cuda Memory Model
5	Optimizing Memory Performance and Synchronization
6	Lab 2
7	Advanced Memory Topics, Concurrency and Dynamic Parallelism
8	Lab 3
9	Image Processing
10	Lab 4
11	Applications
12	Additional Topics and Final Project Presentation

- Participate in all four lab sessions and demonstrate results on the challenges stated in them.
- Successfully develop an implementation of the final project application and present it to the class.

Awarding certificates of completion have proven to be motivationing factor for students, as demonstrated by many technical summer schools and seminars [14], [15]. Certifications offer benefits, including the following:

- An objective students can work toward during the course.
- A tangible recognition they can take home, feeling accomplished.
- They can be added on their resumes, recognizing the knowledge obtained on the subject.

In addition to awarding a certificate, the students completing the best project are awarded an NVIDIA GPU. This helped retain students in the final weeks of the class. As noted by Small et al. [16], providing external motivation can improve student motivation, but needs to be done through an instructor who can generate interest in students by creating challenging learning situations. We have emphasized this environment during our hands-on in-class programming sessions. This extra carrot has helped increase the number of students completing a final project.

C. Schedule

This class is taken by students from various disciplines and different academic levels. Providing a schedule that works for all students is key for retaining them across the duration of the course. Additionally, the schedule for the class significantly affects the number of students who can participate due to conflicts with other classes.

The schedule of the class is always offered in the evenings to reduce the number of potential conflicts with other classes. As a retention strategy, the class is scheduled with two sessions per week versus just once per week. As a result, the class finishes in a shorter amount of time, and avoids conflicts with other academic activities. Employing this strategy demonstrated positive results given it avoids students from dropping at the end of the semester due to heavy workloads from other classes, including final exams and final projects. Reducing the schedule in half allows the instructor to complete all of the sessions in approximately 6 weeks, making sure that students complete the course before midterm exams.

These measures were applied for the first time in Fall 2016. An increase in the number of students at the end of the course was observed. From a drop of 94% (1 student remaining out of an initial 16 for Spring 2016) to 12% (7 students remaining out of an initial 8 for Fall 2016).

D. Communication

Piazza [17] is used as the main source of communication with students. This collaborative learning tool enables easy resource access for students, and provides a platform for students to ask questions and help each other.

E. Hardware Resources

Students have available the Northeastern University Discovery Cluster, a high performance computing system used to supply state-of-the-art HPC resources to faculty and students for research and education. This system is located at the Massachusetts Green High-Performance Computing Center (MGHPCC) [18] in Holoke, MA. All students are granted access to this resource in the first week of the class. Presently the cluster provides 30 nodes with E5-2650 Xeon processors with 32 logical cores each and a NVIDIA TESLA K20m GPU. It also includes 15 nodes with E5-2690 Xeon processors with 48 logical cores and a NVIDIA TESLA K40m GPU. The code and hands-on labs developed for this class have been tailored to work with this environment and hardware. Advanced codes are delivered using makefiles that are configured for this cluster, but can be easily customized for other systems.

The authors acknowledge Information Technology Services and Research Computing at Northeastern University for providing high performance computing and storage, related software, visualization, and consulting resources that have contributed to the development of the work described in this paper [19].

F. Class Sessions

We build upon publically available material to develop the slides used in the theory-based lectures in the class. This material [20], [21], [13], [22], [14] was chosen based on ease of use and if the material could be customized to fit the structure of the course.

Significant work was done to improve the hands-on labs and the final project. These sessions were expanded to provide more depth to the theory behind GPU computing, and provides more detailed guidance for students, based on feedback received from students in the class. We have introduced a set of questions into the guided labs, challenging the students to try additional experiments, and even suggesting them to break the code to help the development of stronger debugging skillsets.

We became increasingly concerned with the lack of effort put into homeworks in previous years. This was usually a result of the competition with coursework in their other classes. While research suggests homework helps students

learn more [23], under the context of a free course, we observed that removing this requirement was received positively by students.

Our earlier experience found that students did not complete them and discouraged them to continue in the class if they could not turn them in. The same effect was observed with quizzes, which again, impacted retention. PowerPoint slides were used during the theory sessions, allowing the students to have the material available before class, and reducing the need to take detailed notes during class.

G. Laboratories

By removing homework and quizzes from the course, we needed a way for students to practice the material that was presented in class. We added laboratories to reinforce key concepts, enabling students to apply what they had learned. Previous work has demonstrated that hands-on labs are a great method for students to master the concepts that are discussed in class [24]. Labs consist of a set of guidelines students have to follow in order to implement a particular algorithm, including multiple questions that need to be answered in order to develop a deeper understanding of parallel programming concepts. If students are unable to complete a lab class, they can continue working on them at home.

H. Final Project

Most textbooks rely on mathematical examples to illustrate best practices in parallel programming. Example programs include matrix multiplication and vector add, which are commonly used methods/operations, and lend themselves to parallel execution. These examples are first presented in serial fashion, and later followed with parallel implementations.

Despite their common use in class, these examples lack sophistication of a complete application, and are difficult to use in a final project. We have found that students are most motivated to work hard when their work demonstrates practical results. One motivating class of applications is imaging processing algorithms, where the student can view the before and after image, visually inspecting the results of their processing. Data visualization has long been used to improve data and algorithm analysis in scientific computing [25], [26]. Exposing students to these tools allows them to have a better understanding of their code, while creating a motivating environment for students. Utilizing image processing algorithms is an effective way to explore parallel frameworks, while allowing the students to visualize the results of running their parallelized code. They can both check for correctness, while also focusing on application performance, which is generally the main focus of the assignment.

Recognizing the value of a visual application, the final project consists on improving the performance of the *Histogram Equalization* algorithm applied to a corpus of images, as shown in figure 2.

The project can be developed in groups of (at most) 3 students. The students need to study the application to understand how the algorithm works. They develop an optimized GPU code using CUDA, providing the same functionality of the histogram equalization found in the OpenCV package, but improving the performance of the processing considerably. All students work on the same project. Students are encouraged to explore their own unique algorithm. We spend the last class session discussing their implementations, sharing different ways of optimizing their resulting CUDA code. Finally, the best performing project receives a prize (an NVIDIA GPU) for their hard work.

As observed by Shindler et al. [27], creating an in-class competition can be highly beneficial to students, but with many caveats to consider. We avoid any potential drawbacks from competitions by making the competition an optional target for students. This project encourages students to apply the lessons learned from the course to be able to produce an efficient implementation. This way, the competition becomes an additional form of encouragement for those students who want to pursue a reward for achieving **the best** performing implementation. This competition results in a fun extracurricular activity for students, with no penalties for losing. Most importantly, in the last session, students get to share their optimization experiences with the rest of the class, creating a healthy competition environment.

Students are given a baseline code that works with a CPU-based OpenCV interface, and a simple CUDA kernel which reads the input image and provides methods to update the image. This structure allows the students to focus on the algorithm implementation and performances, without worrying about getting the initial code structure working right. Additionally, evaluating the final code from students becomes easier if they all follow the same coding structure. We received positive feedback from our anonymous survey on our model. One typical comment was: *"It was amazing! specially the last session I learned a lot from others as well."*

IV. Sessions

The class is organized around five main subjects, described in greater detail in the sub-sections that follow:

A. Introduction to Parallel Processing and The CUDA Execution Model
B. The CUDA Memory Model
C. Advanced Topics in CUDA
D. Image Processing
E. Parallel Applications and Additional Topics

Approximately half of the last session is devoted to the final project presentations by the students. This allows students to share their experience and perspectives on improving the performance of the assigned algorithm.

A. Introduction to Parallel Processing and the CUDA Execution Model

Two technical sessions and one lab session are used to cover the topics in this section. The sessions cover an introduction to the course, including the syllabus and how to request access to the Discovery Cluster. This includes an introduction to parallel

Figure 2. Example input image and the output of the histogram equalization algorithm.

programming and the importance of parallel programming and GPUs in today's society. Furthermore, examples, such as the jigzaw puzzle by Neeman [28], are used to help students reason about parallelism.

The next session introduces the basic CUDA concepts and the general architecture of a GPU. They are presented together to explain the general flow of execution of a GPU program. The main CUDA API functions are presented, including the functions required to copy data between the CPU and GPU, and launch kernels on the GPU. Further exploration includes approaches to optimize execution on the GPU. We review warp execution and show how the GPU handles vectorization through warps. This is presented in a practical manner by following the execution on the GPU step-by-step.

The last topic is a brief introduction to profiling tools, which are used for measuring and understanding performance of the GPU. These tools will be used extensively during the labs. We include the measurement of time with CUDA events with a number of profiling tools, including nvprof, nvvp, and cuda-memcheck.

Hands-on Lab

This lab is essential for students to fully understand how the CUDA Execution Model works. The first stage of the lab consists of ensuring students have access to the cluster and understand how to request a node with a GPU. This includes installing and using ssh and scp tools, such as PuTTY and FileZilla, respectively for Windows users.

The second part of the lab consists of writing their first GPU program. A well documented program is given to students as a baseline to implement a vector add on the GPU. This code includes commented sections to guide the students where they need to insert the necessary API function calls. We introduce additional questions in the guidelines with the objective of ensuring the students understand what is going on in their first program.

The third and fourth parts of the lab consist of using profiling tools to gain a better understanding of the performance of the application. This experience serves as a stepping-stone for more advanced usage in later labs. The tools used include cuda-memcheck - to confirm memory accesses are well-behaved, and nvprof - to profile the application and analyze performance metrics such as gst_efficiency, gld_efficiency. Students are encouraged to run the application with different block sizes to see the effects on the performance metrics. We try to instill intrinsic motivation [16] in the students, by providing questions that require additional investigation.

B. The CUDA Memory Model

Two technical sessions and one lab session are dedicated to the CUDA Memory Model. We discuss in depth the importance of understanding the Memory Model for GPUs. To be able to obtain high performance, students need to understand the memory model. This includes the importance of locality and the different memory types available in the CUDA Memory Model, including: registers, local memory, shared memory, constant memory, texture memory, global memory and caches.

In the next session, we focus on the importance of synchronization across all levels of the memory hierarchy, and a deeper dive into how each memory level works in order to understand how to efficiently exploit that level. This includes shared memory and the importance of avoiding bank conflicts to expose the highest bandwidth from the memory [13]. Global memory is covered in depth as well, including the importance of exposing memory coalescing to reduce the number of transactions and its impact on memory bandwidth from RAM memory [13]. We finish by covering some simple examples on how to use constant and texture memories.

Hands-on Lab

The main objectives of this lab are two-fold:

- Test the impact of coalesced reads in global memory for a vector add application.
- Test the advantages of using shared memory for a 1-D stencil algorithm.

This session is focused on the importance of understanding the CUDA memory model and how to achieve efficiency and performance. This session is fairly complex, and we recommend having an extended session due to the complexity of the examples.

The first part of the session is focused on testing global memory performance by using coalesced reads/writes for a vector add application. This is achieved by implementing the algorithm using a structure of arrays (SOA) and an array of structures (AOS), as shown in figures 4 and 3, respectively.

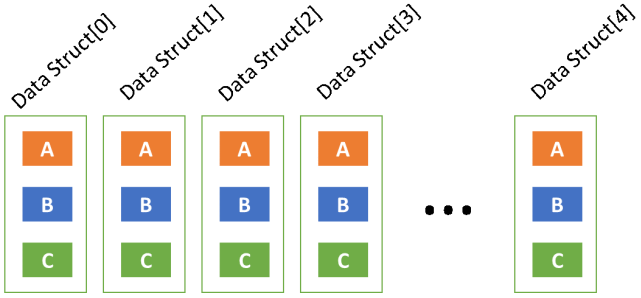

Figure 3. Example of an array of structures (AOS).

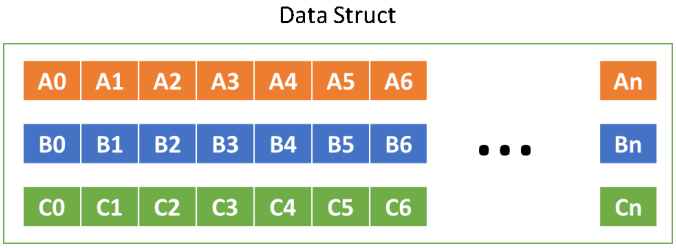

Figure 4. Example of a structure of arrays (SOA).

The baseline code provided implements the vector add algorithm using an array of structures (AOS). Students are guided to measure the performance of this implementation, and look at metrics provided by nvprof to quantify the performance/bandwidth achieved when using large vectors.

Once the students have familiarized themselves with the baseline, a challenge is provided to implement the same algorithm using structure of arrays (SOA). Finally, when students have been able to implement a correct algorithm with SOA, they can compare the performance of both implementations. Results will show that the SOA version will perform faster due to the coalesced reads. Metrics collected with nvprof are used to confirm this behavior.

The second part of the lab focuses on using shared memory, and the impact it can have on performance when used appropriately. The algorithm used in this example is a 1-D convolution filter, where a mask of size *2*RADIUS+1* filled with 1's is applied to an input array as shown in figure 5.

This algorithm contains a higher level of complexity as compared to previous examples. As a rule of thumb, shared memory is a complex mechanism to get it to work correctly in a small amount of time. Due to this complexity, students are guided through a partially complete code so they can fill in the correct indexes and other important features to use shared memory appropriately.

Once their implementation is working properly, students run a test sweep to compare the performance of the optimized GPU

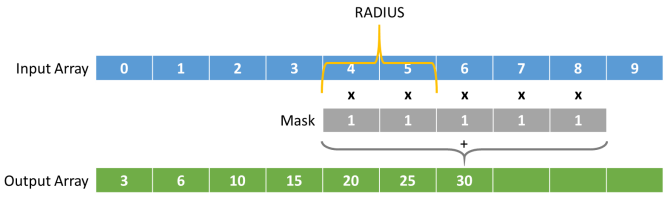

Figure 5. Example of a 1-D convolution filter (stencil) applied to an array.

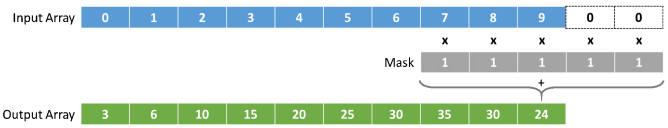

Figure 6. Example of how the boundary conditions will be handled in the 1-D convolution filter example.

version using shared memory, and compare it to a naive GPU version and a CPU version. These results will demonstrate that signifcant improvements in performance are achievable when using shared memory. Finally, students will explore the impact on speed-up when the radius of the mask is increased or decreased. A larger radius will share more data across threads in a block, which translates to better data reuse in shared memory. Running this experiment will demonstrate how exploiting shared memory is a great way to improve algorithmic performance when data reuse can be exposed.

C. Advanced topics in CUDA

We devote one technical session and a lab session to cover advanced topics. We elected to cover only a few advance approaches because we wanted students to be knowledgeable in some additional tools to improve the performance of their code. The main topics discussed in these sessions are the following:

- Improving host to device memory transfers by using pinned memory, zero copy and unified memory.
- Using concurrency (streams) and how to execute multiple GPU functions concurrently, including execution of memory copies while running CUDA kernels.
- Exploiting dynamic parallelism, showing the flexibility it offers for the development of advanced algorithms.

Hands-on Lab

The objective of this lab is three-fold:

- Demonstrate the benefits of pinned memory to optimize a simple algorithm.
- Demonstrate the simplicity of unified memory through a vector add application.
- Study the impact of dynamic parallelism across a multi-dimensional algorithm.

The same code used on the previous lab is used for the pinned memory example. The students will modify this code to use CUDA API function templates for pinned memory instead of standard allocations. The student will compare the

performance of this new version against the previous version. They will be able to observe that memory transfer times decrease, improving the performance of the application.

A simple vector add example is provided to the students as the baseline for the unified memory example. This code initializes the input arrays with a separate function and then implements the computation in a second function. The student is then guided to use this same code and implement it using unified memory API function calls to execute the computation on the GPU. Students will realize that the changes to the code are minimal, and most of the code remains the same, demonstrating that unified memory is a much easier way to exploit the GPU.

The final section of this lab consists of using dynamic parallelism and measuring the impact it can have on a specific example of multiple independent matrix multiplies. The student is encouraged to use multiple levels of nested parallelism and measure the performance of the different versions. The general observation is that the ratio between the outer loop and the inner loop will define whether the performance of the algorithm improves with dynamic parallelism. The rule for dynamic parallelism is that the outer loop should have a small number of iterations as compared to the inner loop. This translates to a smaller number of kernel launches as compared to the reverse case. Dynamic Parallelism also has the positive side-effect of making the code easier to read and understand.

D. Image Processing

One technical session is allocated for experimenting with image processing, and we include one lab session. In section III, we discussed the reasons for having an image processing algorithm, as this will nicely setup our final project for the course. To give students a deeper understanding of these algorithms, we dedicate two sessions to this topic.

Image processing is one of the most popular applications implemented on GPUs. These examples not only show huge performance improvements on the GPU (versus running on a CPU), but they are also interesting for students as they can see the impact their code has over the output of the algorithm. Basic image processing operations are covered. We also cover simple algorithms which will be covered in the following lab. The algorithms include a simple pixel brightness modifier, and a Sobel edge detector algorithm that uses a 2D convolution filter.

Hands-on Lab

Two examples of image processing algorithms are covered in this lab. The first part consists of implementing an algorithm with the GPU that changes the brightness of the pixels in a grey-scale image. Students are shown how to display ppm/pgm images on their personal computers. A baseline code is provided to students that includes reading an input image, allocating the data in a two-dimensional array, and modifying the pixel values with the values read from the command line. Students are guided to find the best block size configuration that will yield the best performance.

The second part of this lab involves analyzing different implementations of a Sobel algorithm. The student identify which implementation provides the best performance and they are asked to postulate and explain the results. The different versions include:

- Basic: A simple implementation of the sobel algorithm.
- Opt1: A simple implementation that uses shared memory as the optimization.
- Opt2: A simple implementation that reduces the total number of reads from global memory and simplifies the computation.
- Opt3: A complex implementation that uses shared memory and packaging to process multiple pixels per thread.
- Opt4: A fairly complex implementation that uses texture memory to speed up the memory reads from the input array.

We encourage students to change the programs and see the impact their changes have on the output. The idea is to demonstrate the benefits of typical optimizations, without having students implement each optimization. This will also provide a good baseline for the work they will have for the final project. Figure 7 shows the output for both applications.

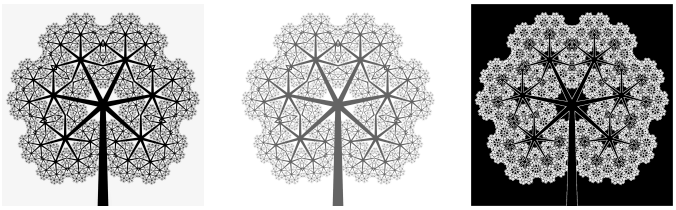

Figure 7. (a) Input image, (b) Output from the brightness application with a brightness increase of 100, (c) Output from the sobel application.

E. Parallel Applications and Additional Topics

Two technical session are dedicated to additional topics. Four common parallel algorithms are covered. Students consider the design of the algorithm as they try to implement them efficiently in parallel. The applications include: i) parallel reduction, ii) a parallel scan (prefix sum), iii) histogramming and iv) convolution. We show multiple examples to show how they can be implemented, and discuss the importance of these algorithms in many key applications. The final session covers a brief introduction to OpenACC and additional features added in Cuda 8 and CUDA 9, motivating students to continue learning.

V. SURVEY

An anonymous online survey was used to solicit feedback from students. This section summarizes responses from students for Spring and Fall semesters from 2017[1]. The figures show the combined results from both semester responses.

[1]Results for this survey were collected using Google forms and results are available upon request.

A. Education Level

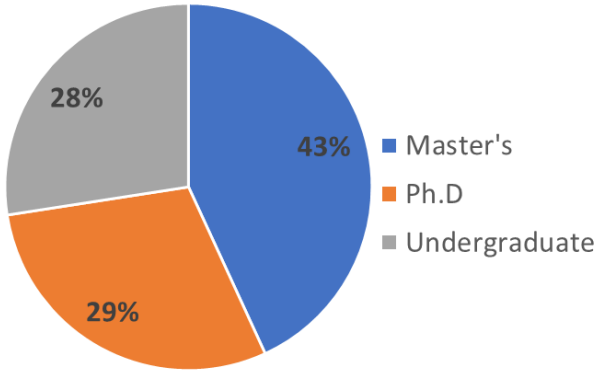

Figure 8. Level of education from students who attended the course.

As figure 8 shows, 72% of students who participated in the course were graduate students (Master's or PhD level). Graduate students at Northeastern University are only required to take two courses per semester, providing more flexibility to take this course compared to undergraduate students.

B. Attendance Rate

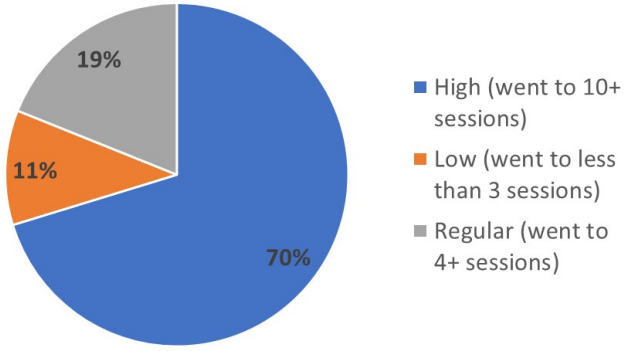

Figure 9. Attendance rate reported by respondents.

Figure 9 shows that 89% of students who responded to the survey considered their attendance rate as regular or high. Students who considered their attendance rate as low were asked the reasons why they were unable to attend most of the sessions. The two main reasons for self-reported low attendance were: 1) conflicts with timing with other classes, and 2) conflicts in terms of workload, including homeworks and projects.

C. Course Evaluation

In Fall 2017, we incorporated the laboratories as part of the course and this had a profound impact in terms of the overall outcome of students responses, as shown in figure 11. For example, 86% of students considered the contribution of the course to their skill/knowledge to be very good or excellent.

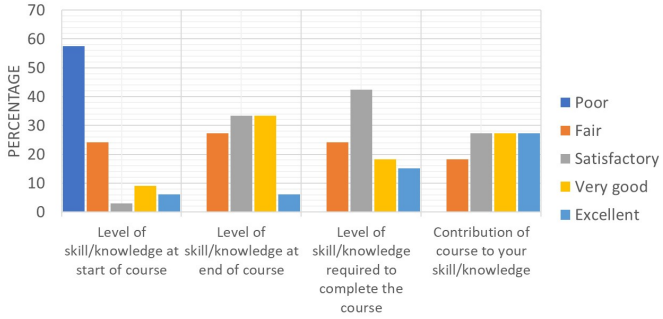

Figure 10. Contributions from the course measured by the students.

Comparing results between Spring and Fall of 2017, we saw a 30% improvement in this metric after the inclusion of these laboratories. Students also mentioned the labs were the most beneficial part of the course.

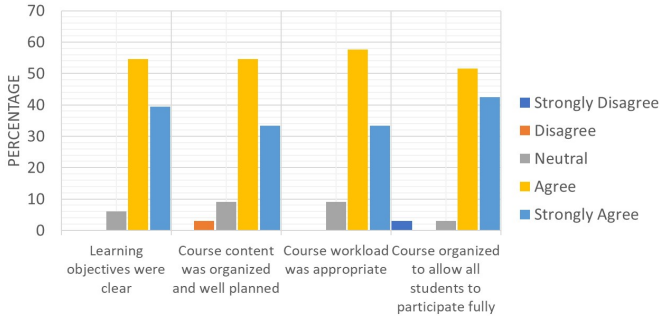

Figure 11. Course structure analysis from students.

In Figure 11, we can clearly see positive feedback observed across course structure analysis metrics. These metrics also improved in Fall 2017, which indicate the overall impressions of students in the course are positive. The survey data says that 93% of students recognized the improvements by rating with an "agree" or better.

Table II. Retention percentage for 2016 and 2017 semesters.

Semester	Total Students	Retention (percentage)
Spring 2016	16	6.25
Fall 2016	8	87.5
Spring 2017	23	82.61
Fall 2017	14	100

Table II shows the retention rates before (Spring 2016) and after the retention strategies were applied. We saw a jump in retention from 6% to +80% after applying our retention strategies. We note that the student demographics were similar before and after the introduction of the strategies. While many other factors may affect retention (namely, more experienced instructors, increased popularity of the course and others), based on the survey distributed at the end of the course, many students specifically cited our retention strategies as encouragement and reasons for them to complete the course.

Qualitative questions were part of the survey and requested individual responses from each student. Selected responses were chosen and displayed in the following sub-sections that summarize the general views from students.

1) What aspects of this course were most useful or valuable?:

- "Hands on lab"
- "labs and assignments and coding"
- "Hands on lab experiments and projects."
- "The lecture slides to refer back to after lectures"
- "Lab sessions"
- "The lab sessions that followed the coursework were very helpful. The slides that are shared were a good start to learn new concepts. Concise and clear contents in the Slides and PDF that were shared at the end of the class."

2) What did you think of the Final Project?:

- "It was pretty cool to apply knowledge to real world application"
- "Challenging, interesting, helped me apply the concepts I learnt in class"
- "Fantastic"
- "It was stimulating but I could not complete and present because of other coursework."
- "Really liked it. Allowed me to think about all kinds of different solutions and techniques I could apply. Learn about which methods would be useful and which ones won't be useful to the specific case through contemplation and experimentation."
- "Final project was optimal. Not too tough and not very easy."
- "It was amazing! specially the last session I learned a lot from others as well."

3) How would you improve this course?: Before Fall 2017, the majority of responses to this question involved having more practical examples in class, as well as suggesting hands-on labs or similar methods. Many of the following observations are already under consideration to improve the material for the following semesters. The remaining responses are summarized as follows:

- "Make it a credit course."
- "I would be sure to include more references so people less familiar with programming can try to follow along"
- "Maybe having an organized summary of notes for the course."
- "Maybe go a little slower over the material, as a person relatively new to coding I had difficulty following along."
- "Offer some specific readings of the CUDA docs for people with very little experience to make sure they don't get lost when discussing how to actually code for the GPU."
- "Cover more fundamentals for beginner and see more experiment codes."
- "Add more basic information about programming"
- "During the lecture sessions, instructor should demonstrate using more examples instead of just teaching in theory. Providing more example codes will help understand many concepts better. In that way, lectures and labs can be merged together and students can learn a lot more in lesser time. I went to the class with very little knowledge of gpu so I felt a bit of the gap in understanding concepts in lectures which was also a reason I skipped a few lectures as I learnt whatever I could from labs."
- "I would suggest having more than one session for the key concepts like CUDA memory model and lab sessions could be slightly longer (maybe 2 hours) than the regular class."
- "A follow along class would've been useful because I am not very well versed in coding and do not have a huge background"

4) Overall thoughts on the course:

- "Instructor put 110% of his efforts. A well designed course which can have two flavours such as one for industry people (having prior knowledge) and other for new students. Cheers!!"
- "I really appreciate the dedication and enthusiasm exhibited by the instructor. The instructor was very helpful and easy to approach."
- "Thank you for teaching this. It was actually a great learning experience even if I may not use it. The most useful part of it in the long run will be knowing how a processor works generally such as cache, memory, global memory, compute nodes etc."

VI. Conclusions

As industry continues to incorporate multi-core architectures in leading-edge systems, future computer scientists must be educated in this new parallel programming paradigm. This need is growing even faster given the growing popularity of using GPU parallelism. The computational performance of a GPU continues to increase, bridging the gap between single-threaded workloads and higher levels of parallelism. Despite the popularity of these devices in a growing number of application domains, there is a lack of new graduates with appropriate education in parallel computing principles and GPU programming.

This free GPU programming course contains all the necessary concepts needed to offer it as part of a core curriculum. The material presented in this class has already been used as part of graduate courses at Northeastern University, and has also been used as the introductory material taught by our lab members in a course customized for engineers at Philips and NASA Langley Research Center.

Initially, the class particition faced some retention issues. The following retention strategies were later implemented:

- Offer the class for free to all students.
- Provide a certification of completion.
- Adapt the schedule to best suit students.
- The incorporate hands-on labs instead of homeworks and quizzes.

- Assign a compelling final project that involves image processing.
- Award a prize for the best final project.

The retention strategies employed in this course have been judged successful, but refinement is still necessary to keep the course relevant. The wide broad background of the students taking the class implies the course has to adapt to the incoming students and provide all the necessary information for these students to learn. As educators, teaching parallel programming across multiple disciplines today is vital to provide our students the tools they will need for the next leading-edge systems.

Special thanks to Leiming Yu, Fanny Nina Paravecino, Xun Gong, and Xiang Gong for their help in developing the content used for this course. All of the material is openly available here: https://github.com/jgutierrezm113/gpu-class.

REFERENCES

[1] M. Bailey and S. Cunningham, "A hands-on environment for teaching gpu programming," in ACM SIGCSE Bulletin, vol. 39, no. 1. ACM, 2007, pp. 254–258.

[2] S. K. Prasad, A. Y. Chtchelkanova, S. K. Das, F. Dehne, M. G. Gouda, A. Gupta, J. Jaja, K. Kant, A. La Salle, R. LeBlanc et al., "Nsf/ieee-tcpp curriculum initiative on parallel and distributed computing: core topics for undergraduates," in SIGCSE, vol. 11, 2011, pp. 617–618.

[3] J. Zarestky and W. Bangerth, "Teaching high performance computing: Lessons from a flipped classroom, project-based course on finite element methods," in Proceedings of the Workshop on Education for High-Performance Computing. IEEE Press, 2014, pp. 34–41.

[4] P. Neumann, C. Kowitz, F. Schranner, and D. Azarnykh, "Interdisciplinary teamwork in hpc education: Challenges, concepts, and outcomes," Journal of Parallel and Distributed Computing, vol. 105, 2017, pp. 83–91.

[5] B. Neelima and J. Li, "Introducing high performance computing concepts into engineering undergraduate curriculum: A success story," in Proceedings of the Workshop on Education for High-Performance Computing, ser. EduHPC '15, 2015, pp. 6:1–6:8.

[6] C. Bederián and N. Wolovick, "A project-based hpc course for single-box computers," in Education for High-Performance Computing (EduHPC), 2016 Workshop on. IEEE, 2016, pp. 1–6.

[7] S. V. Moore and S. R. Dunlop, "A flipped classroom approach to teaching concurrency and parallelism," in Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International. IEEE, 2016, pp. 987–995.

[8] M. I. Capel, A. J. Tomeu, and A. G. Salguero, "Teaching concurrent and parallel programming by patterns: An interactive ict approach," Journal of Parallel and Distributed Computing, vol. 105, 2017, pp. 42–52.

[9] D. B. Kirk and W. H. Wen-Mei, Programming massively parallel processors: a hands-on approach. Morgan kaufmann, 2016.

[10] N. Wilt, The cuda handbook: A comprehensive guide to gpu programming. Pearson Education, 2013.

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

[12] H. Nguyen, Gpu gems 3. Addison-Wesley Professional, 2007.

[13] C. Nvidia, "Nvidia cuda c programming guide," Nvidia Corporation, vol. 120, no. 18, 2011, p. 8.

[14] "Pumps summer school," Jul 2016. [Online]. Available: http://bcw.ac.upc.edu/PUMPS2016/

[15] "Argonne training program on extreme-scale computing," Jul 2018. [Online]. Available: http://extremecomputingtraining.anl.gov

[16] R. V. Small, N. Zakaria, and H. El-Figuigui, "Motivational aspects of information literacy skills instruction in community college libraries," College & Research Libraries, vol. 65, no. 2, 2004, pp. 96–121.

[17] P. Technologies. Piazza. [Online]. Available: https://piazza.com (2016)

[18] "Massachussetts green high performance computing center," 2018. [Online]. Available: https://www.mghpcc.org/

[19] "Research computing." [Online]. Available: https://its.northeastern.edu/researchcomputing/

[20] M. Harris, "An even easier introduction to cuda," Nvidia blog post, accessed, vol. 11, 2017, p. 27.

[21] ——, "How to access global memory efficiently in cuda c/c++ kernels," NVIDIA, Jan, 2013.

[22] C. NVidia, "C best practices guide," NVIDIA, Santa Clara, CA, 2012.

[23] H. Cooper, J. C. Robinson, and E. A. Patall, "Does homework improve academic achievement? a synthesis of research, 1987–2003," Review of educational research, vol. 76, no. 1, 2006, pp. 1–62.

[24] M. Bailey and S. Cunningham, "A hands-on environment for teaching gpu programming," SIGCSE Bull., vol. 39, no. 1, Mar. 2007, pp. 254–258. [Online]. Available: http://doi.acm.org/10.1145/1227504.1227401

[25] W. J. Schroeder, B. Lorensen, and K. Martin, The visualization toolkit: an object-oriented approach to 3D graphics. Kitware, 2004.

[26] H. Thorvaldsdóttir, J. T. Robinson, and J. P. Mesirov, "Integrative genomics viewer (igv): high-performance genomics data visualization and exploration," Briefings in bioinformatics, vol. 14, no. 2, 2013, pp. 178–192.

[27] J. Shindler, "Examining the use of competition in the classroom," Transformative Classroom Management, 2009.

[28] H. Neeman, L. Lee, J. Mullen, and G. Newman, "Analogies for teaching parallel computing to inexperienced programmers," in ACM SIGCSE Bulletin, vol. 38, no. 4. ACM, 2006, pp. 64–67.