Syllabus for Parallel Algorithms and GPU Computing, Spring 2019

People

- Instructor: Prof. José Bento. **Tip:** To maximize your chances of reaching me use class time, CANVAS messaging, and office hours, where I can meet you in person, individually or in groups. I receive many emails, so using email is a bad method to grab my attention.
- TA: Shikun (Jason) Wang (wangapb@bc.edu).
- Extra support: **Tip**: The Connors Family Learning Center offers great tutoring services on a variety of CS related topics. I encourage you to use them if necessary. Link

Office hours

- Prof. Bento: By appointment, Saint Mary's Hall South, S250.
- TA: Shikun (Jason) Wang, Tue 2:00pm-3:00pm; Wed 2:00pm-3:00pm; Fulton 160

Class schedule

Tuesdays and Thursdays, from 9:00am to 10:15am, Lyons 202. During lecture time **no cellphones/tablets/players/etc, and no food, are allowed**. You are, of course, welcome to **use your computer to follow and code algorithms being discussed in class.** You are very welcome to step outside for a snack, phone call, or a break, and them come back. Remember that you do not have to be in the lecture room, but if you willingly want to be there, you need to respect these minimal set of rules.

Prerequisites

Being fluid in

- C Language
- Calculus (to the level of Calculus I)
- Probability (to the level of Randomness and Computation)
- Algorithms

Unfortunately, having taken and forgotten the content of these courses does not fulfill the prerequisites. I strongly recommend you to refresh your memory. I try to make all of my lectures as self-contained as possible. **Tip**: You are encouraged to ask questions about prerequisites during lectures, or in office hours. You can also use the TA, or the tutoring services aforementioned, to refresh those prerequisites. However, this class is not a class on calculus, probability, or introduction to algorithms, which I assume you have already mastered. Unfortunately, I cannot devote most of my time to teach you these topics from scratch, at the expense of not teaching parallel algorithms and GPU computing to the students that do have the required background, and have the right to see the class topics covered in class. To help each student self-assess if he is ready to take this class, I will give a prerequisite quiz in the first lecture. Your score in this quiz **will not** count towards your class grade. Your score will help you self-assess if you are ready to take this class this year. There is nothing wrong in delaying one year to take this class.

If you understand everything that is reviewed in class sessions and you can apply all of those skills to work out the problems discussed in class by yourself, you will probably do well in the class. However, you might be asked to extrapolate the main techniques and ideas learnt in class to examples or problems that you have not seen before, just like when learning e.g., English, one learns some specific words and reads some specific texts in class, but then combine these "atoms" to produce new texts.

Reading

Books are expensive, and not everyone can afford them. There is no mandatory book for the class. I try to make all of my lectures as self-contained as possible (prerequisites assumed). However, this class is based on C CUDA, and this is a complex and fast changing language. Furthermore, the hardware supporting this language is also fast evolving. I therefore strongly suggest students to use as their main support to learn CUDA the online *Cuda C Programming Guide*, as well as the online *Cuda C Best Practice Guide*, which are constantly being updated.

There are many books on CUDA. Below I list a few. Because of the fast evolution pace of this field, some of the code covered in these books might be outdated, their relative merits/limitations might have changed, or they might have been designed and tested for hardware that is outdated. **Tip:** I strongly suggest that you test all the code found in any book by yourself, to see if it is behaving as claimed.

- *CUDA by Example: An Introduction to General-Purpose GPU Programming* by Edward Kandrot and Jason Sanders, 2010, ISBN 978-0131387683. Link
- *Professional CUDA C Programming*, John Cheng, Max Grossman and Ty McKercher, 2014, ISBN 978-1118739327. Link

There are many books on parallel algorithms. A good one, that I often use to suggest project ideas to students is *Introduction to Parallel Algorithms* by Joseph JaJa, 1992, ISBN 978-0201548563. Link

Course topics

This course is divided into two parts. **In the first half of the course**, we learn about the fundamentals of programming in CUDA, and some classical techniques to parallelize algorithms. In the first half of the course, in each lecture, I first introduce some new concepts, and then I exemplify their use by coding different CUDA examples on the fly with the help of the students. Although coding, debugging, and performance testing as a group slows the pace of the lectures, it allows for students to be fully engaged, and guarantees that they take something practical and useful with them after each lecture. Students are expected to finish and extend the examples covered in class by themselves at home. **In the second part of the class**, each student chooses a specific parallel algorithm, and implements it on a modern GPU using CUDA. The purpose of this exercise is for students to experience, first hand, the difference between (i) an algorithm that, in theory, performs well, or optimally, but whose actual implementation on a GPU might lead to poor performance, and (ii) an algorithm that, although suboptimal in theory, performs really well when implemented on a GPU. It is also an opportunity for students to experience that the same conceptual algorithm can be implemented in CUDA in different ways, and that not all of these implementations lead to the same performance.

During the second half of the class, during lecture time, students mostly work by themselves, each on their own project, while I go around chatting with each student one-on-one, checking on their project status, and helping them. Occasionally, when an issue comes up that is of interest to most of the class, I will ask students to pause their work, and I will explain the issue and how to address it to the whole class. The reason for doing this is that students that are learning CUDA and parallel algorithms for the first time, often require lots of periodic assistance. Past experience shows that not working on the project during class time, leads to students delaying working on the project until the end of the semester, when it is too late for me to provide such periodic assistance to all.

The level of detail with which I will cover the different class topics will vary with students' interests and time available. For each topic listed below, I include a reference to the relevant pages in either of the books *CUDA by Example* (CBE), or *Professional CUDA C Programming* (PCCP), aforementioned for those that want to do extra reading. Note that these books might not cover everything that I teach in class, or the same way that I cover it. They often go beyond what is covered in class. To guarantee that I cover everything in the syllabus, sometimes I might record a lecture and put it online for student viewing. **Tip**: Use the list of objectives in the table below self-assess your learning. Note: these objectives are guidelines, and are the minimum that each student should grasp for each topic. There is more to each topic than what can be covered in a syllabus table. A high-achieving student should feel comfortable in (i) establishing connections between the different topics, not see them in isolation, (ii) applying the main ideas in each topic beyond the examples covered in class, (iii) solving the same problem in several different ways (when possible).

Week	Topics	Pages	Dates	Objectives
1	• Course overview and syllabus; • Self-assessment quiz; • Server access • CUDA software and hardware view • Nvcc compiling • "Hello Word" example	CBE 21-35 PCCP 1-21	15th Jan 17th Jan	Understanding the main elements of NVIDIA hardware, CUDA software view, and their relation. Writing, compiling, and running a simple CUDA program.
2	• CPU vs. GPU core speed • Memory and execution model • Simple example of memory management and multithread kernels • Memory cache, mem. synchronization, mem. alignment, mem. coalescence	CBE 37-57 PCCP 23-47 135-176	22nd Jan 24th Jan	Copying data from the host to the device, and using different memory types (registers, local, global). Explaining what affects the performance of global mem. transactions.

3	• 1D, 2D, 3D kernels • Thread synchronization • Share memory • Bank conflicts • Matrix transposition example • Performance bounds of different parallel sum algorithms	CBE 59-94 PCCP 48-58 67-112 204-253	29th Jan 31st Jan	Launching kernels with different dimensions. Using thread cooperation via synchronization. Understanding what affects the performance of shared mem. transactions. Explain parallel reduction.
4	• Parallel sum implementation • Parallel algorithms from recursive algorithms • Loop unroll	CBE 59-94 PCCP 114-120	5th Fev 7th Fev	Implementing and comparing different serial and parallel sum algorithms, including using shared memory, and loop unrolls, and atomic expressions.
5	• Performance bounds of parallel cumulative sum • Atomic expressions • Thread fencing vs thread synchronism	CBE 163-183 304 214-216	12th Fev 14th Fev	Implementing and comparing different parallel cumulative sum algorithms, including using shared memory, and loop unrolls, and atomic expressions.
6	• Texture memory, 1D, 2D, 3D • Binding textures to global mem and arrays • Surfaces • Game of life example	CBE 115-137	19th Fev 21st Fev	Implement a parallel version of the Game of Life using different types of texture memory accesses.
7	• Game of life example (finish) • Dynamic parallelism • “Hello World” dynamic parallelism example	CBE 115-137 CPPC 122-132	26th Fev 28st Fev	See above. Implement a simple “hello world” using dynamic parallelism. Parallelize an array sum using dynamic parallelism.

	• Recursive sum using dynamic parallelism example			
Spring Break: No class on the 5th Mar and 7th Mar				
9	• Introduction to the different possible projects		12th Mar 14th Mar	
10	• Project time		19th Mar 21st Mar	
11	• Project time		26th Mar 28th Mar	
12	• Project time		2nd Apr 4th Apr	
13	• Project time		9th Apr 11th Apr	
Easter Break: No class on the 18th April				
14	• Project time		16th Apr 18th Apr	
15	• Project time • Project presentations		23rd Apr 25th Apr	
16	• Project presentations		30th Apr 2nd May	

Grading

- 30% mandatory class attendance
    - Class attendance is mandatory. This is a hands-on class, where I will spend most of the time pair-coding with the students. Any student that attends class will be required to do some in class coding at some point in the first part of the semester, and for sure in the second part of the semester, where each student will be working on their own projects. This is the reason why attendance gives you points, because you actually get work done in class. I take presences/absences at the beginning of each lecture, so arrive early. Your grade in this component is the fraction of lectures that you have attended.

- 30% finishing class examples
    - We code many examples together, as an entire class, during lecture time. These examples often have many possible variants that we do not have time to explore during lecture. Other times, we might code the main part of an algorithm in class, but leave some implementation details unfinished.

It will be the students' responsibility to code all of the examples coded in class by themselves at home, including bits that were not completed in class, and to upload this code as a part of assignments that I will create after each lecture on CANVAS. Students might also have to explore variants of examples covered in class, or different solutions to the same problem. Each time there is there is something left for students to finish, or to explore, at home, I will say so in class, and I will state what is to be done in the respective CANVAS assignment. Students complete these assignments by submitting their working code in a zip file in CANVAS. Typical deadline for these assignments is 1 week, unless otherwise specified in CANVAS.

- 30% project
  - After the Spring break, class time will be devoted to working on students' projects. Each student is free to suggest his own topic for a project, which I might, or not approve. **Tip:** Although we only start working on the projects after the Spring break, I suggest that students choose their project, start working on it, and discuss it with me early on the semester.

    I include a few project topics that have been chosen in the past below, and, when applicable, references to the relevant pages in *Introduction to Parallel Algorithms by* Joseph JaJa.

    - Finding the distances of nodes to root in trees, pg. 52-56
    - List Ranking Algorithm, pg. 92-108
    - Range Minimum Query, pg. 131-136
    - String Matching, pg. 318-339, 456-457
    - Perfect Matching, pg. 460
    - Computing the Convex Hull, pg. 56-60, pg. 264-272
    - Finding Lower Envelopes, pg. 286-291, 304
    - Fast Fourier Transform, pg. 381-385
    - Parallel Merge Sort, pg. 158-160
    - Randomized Quick-sort, pg. 196, 465-473
    - Bitonic sorting, pg. 178-181
    - Parallel Minimum Spanning Trees algorithm, pg. 222-227
    - Solving triangular linear systems, pg. 375-379
    - Construction of suffix trees, pg. 339-352
    - Parallel search, pg. 146-148
    - Symmetry breaking and coloring, pg. 75-80
    - Parallel merging, pg. 61-65, 148-157
    - 2-3 tree parallel insertion, pg. 65-70

The algorithms described in the book *Introduction to Parallel Algorithms by* Joseph JaJa were not developed with modern GPUs in mind, which makes the project interesting and challenging. In particular, the goal of this project is for students to experience, first hand, the difference between (i) an algorithm that, in theory, performs well, or optimally, but whose actual implementation on a GPU might lead to poor performance, and (ii) an algorithm that, although suboptimal in theory, performs really well when implemented on a GPU. It is also an opportunity for students to experience that the same conceptual algorithm can be implemented in CUDA in different ways, and that not all of these implementations lead to the same performance.

Time permitting, all students are encouraged to develop, and compare, multiple possible CUDA implementations for a given algorithm's description. They should comment on their merits and limitations.

Code: Students will be required to submit working code associated with their project by the 10th of May, via a CANVAS assignment that I will create. I strongly recommend that you use a git repository to maintain your code throughout the semester.

Report: On the same date, they will have to submit a report associated with their project. The report is to be written using Overleaf and Latex. A Pdf version of their report will have to be submitted via CANVAS, also via a CANVAS assignment that I will create. Students will be able to see each other's reports on Overleaf. This will give you an opportunity to learn about other parallel algorithms, each your colleagues, and also gauge the quality of other reports. I strongly suggest students to start working on their report early. Use the following link to write your reports. I have created one folder and one .tex file for each student. Edit only your file/folder. I have also written one report myself, about pointer jumping techniques, that you can use to gauge the quality of the report that I am expecting.

Oral presentation: On the last three lectures, each student will give a 10 min presentation of their work to the rest of the class. You can use the board, or slides. Your presentation should succinctly answer the following questions: What is the problem? What is the main algorithm?

What is the main CUDA challenge? What are the performance gains obtained via CUDA?

- 10% class participation
    - Students can hear points in this component of their grade by actively participating in class. This includes volunteering to take the lead in coding class examples, coming up with smart alternative solutions, coming up with new problems, algorithms, and parallelization techniques, or unexplored CUDA features.

Computer access

In this class we will use NVIDIA GPUs and the CUDA programming language. The Computer Science Department at Boston College has a total of five GTX 1080 and twelve RTX 2080 Ti GPUs. These can be accessed remotely, and locally. I will explain how to do so in class. The use of these GPUs is exclusive for class use. Note that there are more than students that GPUs, so it is possible that two students try to use the same GPU at the same time. To avoid this problem, use the command *nvidia-smi* to determine which GPUs are being used in each machine. It is always on interesting exercise to test the performance of the same code on two different GPU architectures, so test your code on both the GTX 1080 and on the RTX 2080 Ti if possible.

Integrity

Do not copy your solutions for the homework from online sources or your colleagues. If you get caught, you will get into serious trouble. It is OK if you cannot solve a problem. Because of this, be careful with your collaborations. It is Ok to ask for help from a conceptual point of view. However, if you ask for very detailed help, there will be almost-equal homework solutions, and you will hurt yourself and your helping-friend. For exams and midterm, absolutely no information exchange is allowed.

Please read the following:
https://www.bc.edu/offices/stserv/academic/integrity.html