COURSE SYLLABUS

CIS 4930 / CIS 5930
Parallel Programming on Multicore Processors / Programming Heterogeneous Multicore Processors

Spring 2010

Prerequisites:

COP 4610: Operating Systems. You should be comfortable programming in C/C++, and have good knowledge of undergraduate level algorithms, data structures, and computer architecture. No knowledge of parallel computing is required; however, you should have some experience with threads and some exposure to locks, atomic operations, and mutual exclusion, which are typically discussed in operating systems.

Class Schedule:

Activity	Day	Time	Location
Lecture	MW	2:00 pm - 3:15 pm	LOV 103

Contact information:

Instructor:	Ashok Srinivasan
Office hours:	I will decide on my office hours after finding times convenient to you. I am also usually available in my office, and you can feel free to meet me in the afternoons, except before class. Alternatively, you may schedule an appointment, either by email or by phone.
Office:	169, Love Building
Phone:	644-0559
Email:	asriniva AT cs.fsu.edu

Course material:

Required Material:

- There is no required text book.
- Online course web page: available through Blackboard (http://campus.fsu.edu).

Reference material:

- MPI: www-unix.mcs.anl.gov/mpi/
- OpenMP: www.openmp.org/
- Single core optimization: spiral.ece.cmu.edu:8080/pub-spiral/pubfile/paper_100.pdf
- CUDA: developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdfwww.openmp.org/
- Cell programming: www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1741C509C5F64B3300257460006FD68D/$file/CellBE_PXCell_Handbook_v1.11_12May08_pub.pdf
- I will provide other material that will be used in this class.

Online resources:

- Single Unix Specification: http://www.unix.org/single_unix_specification.
- FSU User Services Site Licenses: http://sl.us.fsu.edu/index.html

Computer accounts:

- You will need a computer account in the Computer Science department. Please follow the procedure outlined in http://www.cs.fsu.edu/sysinfo/newstudent.html to obtain an account, if you do not have one already.
- Class email will be sent to your FSU account (@fsu.edu). So please obtain an FSU account, if you do not have one already. If you use another email

account (such as yahoo), then you **must** forward your FSU email to that account. Instructions on obtaining an account and forwarding email are available at http://www.ucs.fsu.edu.

- The subject line in any email you send me should start with `CIS 5930`.
- You will need to use Blackboard (http://campus.fsu.edu). You will be able to access it only if you have an ACNS account.

Course rationale:

In your programming, algorithms, and data structures courses, you would have learned how to write efficient sequential programs, that is, programs running a single thread. However, all modern processors are multicore, and in order to make effective use of them, you need to run multiple threads. When accelerators such as GPUs or the Cell processor are available, then the number of threads needs to be even larger. In your operating systems course, you would have learned to write code that uses multiple threads, and how to avoid common errors such as race conditions. This course teaches you how to organize the computations of the threads so that they work together and perform the required computations efficiently, making good use of the available hardware resources.

Course description:

In a parallel computation, multiple cores work together to solve a given problem. These are exciting times in parallel computing, because, with multicore processors, all modern computers are parallel machines. In fact, the largest parallel machine has over two hundred thousand processors, and high end machines of the near future will require the order of a million threads if one wishes to make optimal use of them. Our goal is a little more modest; we will deal with computational resources requiring the order of ten to a few thousand threads, which is typical of computing resources available to the vast majority of computer users.

While modern computers provide enormous raw computational power, it is often not easy to make effective use of all this power. The problems encountered in making effective use of a large number of cores are similar to those encountered in making a group of people work together. (i) People may spend much of their time talking to each other, instead of doing useful work. Communication between cores is quite expensive, compared with the CPU speed. So we need to pay attention to minimizing the amount of communication (data movement); otherwise much of the time will be spent on communication, rather than on useful work. (ii) In a group, a few people may do much of the work, while the others relax. Similarly, in a computation, the work load on different cores may differ. In order to make effective use of the computer, we want to keep the work load balanced on all cores. (iii) It may be difficult to decompose a problem so that people can work on different parts simultaneously. For example, consider someone who want to have dinner cooked, eat it, and then have the dishes washed. It is not easy to speed up this process by hiring someone to cook, and another person to wash the dishes, because the three tasks are sequential; the food needs to be cooked before it is eaten, and the food needs to be eaten before the dishes are washed. Similar problems occur in parallel computations too, and sequential parts of the computation can reduce the effectiveness of parallelization substantially.

This course will describe different techniques used to solve the above problems, in order to develop efficient parallel algorithms for a variety of problems. We will also pay much attention to practical aspects of implementing parallel code that actually yields good performance on multicore processors. A major emphasis of this course will be on parallelism available on accelerators such as GPUs and the Cell processor (which is used on the Sony PS3). We will first learn how to program conventional multicore processor, GPUs, and the Cell processor. We will then learn how to make efficient use of a single core, because one cannot make efficient use of many cores until one learns efficient use of a single core. This often requires considerable effort. We will, therefore, learn how to profile the code and identify computational bottlenecks, so that we can concentrate our efforts on optimizing that portion of the computation. We will then learn how to optimize the performance. This will be accomplished through good algorithms, which are usually common across the three types of computing platforms considered in this course, and also through implementation details, which often vary with the architecture.

By the end of the course, you should be able to make effective use of the three types of computing platforms discussed above, and obtain significance improvements in speed for practical applications. You may still not be able to obtain the best possible performance; one semester is not enough time for that! However, you will learn enough that you can continue further learning on your own and become an expert after a few years of experience, if you are interested in this.

Learning objectives:

At the end of this course, you should be able to accomplish the objectives given below.

- Define terminology commonly used in parallel computing, such as *efficiency* and *speedup*.
- Describe different multicore architectures and programming models.
- Implement efficient algorithms for common application kernels, such as matrix multiplication, on conventional multicore processors, GPUs, and the Cell processor.
- Given a problem, develop an efficient parallel algorithm to solve it.
- Given a parallel algorithm, analyze its time complexity as a function of the problem size and number of processors.
- Given a parallel algorithm, implement it using MPI, OpenMP, pthreads, CUDA, or a combination of these.
- Given a parallel code, analyze its performance, determine computational bottlenecks both theoretically and empirically, and optimize the performance of the code.
- Given a parallel code, debug it and fix the errors.
- Given a problem, implement an efficient and correct code to solve it, analyze its performance, and give convincing written and oral presentations explaining your achievements.

Your responsibilities:

Deadlines and Instructions

Following the same professional guidelines that you will encounter at work, there are strict deadlines, and instructions that must be followed. Please read instructions carefully, and schedule your activities so that you submit assignments well in time. You should check your **garnet** email account and the class web page regularly, and note other announcements, on-line and in class.

Group Project

You will have one group project, which will account for half your course grade. You should participate in your group's activities and make a fair contribution to the efforts of your group. You can learn more about working in groups at: www.cs.fsu.edu/~asriniva/courses/CP10/groupwork.html. Roughly half of your grade on the group project will be based on the performance of the whole group. The other half will be based on your contribution. However, if your contribution was unacceptably small, then you may be assigned a grade as low as zero. Note that the group project constitutes an important component of your grade. Your work should be of sufficient quality and quantity for it to be accepted in at least a mediocre conference, such as Europar or ICCS. You may want to read some best papers from conferences such as IPDPS or SC, to get an idea of good quality work. While your work may differ from the best ones in quantity, the quality of your work and presentation should be high.

Reading Assignments

After each lecture, you will be given a reading assignment pertaining to that lecture. You should read these, and also practice writing code. **If you learn only the material discussed during the lecture, then you will likely fail the course.** You should learn material from the reading assignment, and also refer to reference material in your programming tasks.

Note that new material builds on the old ones. So, if you have trouble with some material, please get help through the discussion board on Blackboard, or from me, before the next class.

I expect that you will need to spend between one and two hours studying, for each lecture. The programming assignments, project, and exams will consume *additional* time.

The following learning components are important, and you may want to verify if you do satisfactorily on these, after studying the material.

- *Knowledge*: Do you understand the terminology used? Given an algorithm and its input, can you describe the steps carried out by the algorithm and also its output? Given an algorithm, can you write code to implement it? Given an algorithm that you have studied, can you give the time and space complexity?

- *Understanding*: If some aspect of an algorithm were modified, can you analyze the time complexity? If some aspect of an algorithm were modified, can you prove or disprove its correctness? In order to answer these questions, you need to understand how each component of an algorithm affects the time complexity, and why each component of an algorithm is important for its correctness. After you learn about what an algorithm does (and have, thus, acquired "knowledge"), it will be useful for you to think of different things that can be changed, and see how these will affect the time complexity or correctness.
- *Application*: Given a real life, or artificial, problem, can you decide on suitable algorithms from those we have studied, to solve that problem?
- *Creativity*: Can you modify algorithms that we have studied, to make them more efficient for special situations? Given a problem for which our algorithm is not valid as designed, can you modify the algorithm to solve the problem, prove the correctness of your solution, and analyze its time complexity? Can you use multiple algorithms from those that we have studied, to solve a new problem?

Assignments

You will have several small programming assignments in this course, and you will have around two days to work on each one. The assignments will be announced on the Blackboard course web site under the "Assignments" tab. Parallel programming assignments are **substantially** more difficult than sequential programming assignments, and require substantially more time and effort. Please start working on the assignments as soon as they are announced, if you wish to complete them!

Course calendar:

Week	Lecture	Lecture topics	Other activities
1	6 Jan	1. Introduction	
2	11 Jan	2. Review of multicore architectures, including Cell and GPUs	
	13 Jan	3. Parallel programming using OpenMP and pthreads	
3	18 Jan	No class -- Martin Luther King Jr. Day	
	20 Jan	4. Parallel programming using MPI	
4	25 Jan	5. CUDA programming model	
	27 Jan	6. CUDA memory model	
5	1 Feb	7. Introduction to Cell programming	
	3 Feb	8. A Cell programming framework	
6	8 Feb	9. Single core optimization: Making effective use of the compiler and standard libraries	
	10 Feb	10. Profiling code on conventional processors	
7	15 Feb	11. Profiling code on the Cell and GPU	
	17 Feb	12. SIMD intrinsics	
8	22 Feb	13. Optimizing for memory on a single core	
	24 Feb	14. Cache issues on multicore processors	
9	1 Mar	**Midterm 1**	
	3 Mar	15. Performance models and common communication patterns	
10	8 Mar	Spring Break	
	10 Mar	Spring Break	
11	15 Mar	16. Common parallel algorithm design patterns	
	17 Mar	17. Techniques for optimizing parallel algorithms	
12	22 Mar	18. Cell optimization case studies	
	24 Mar	19. Optimizing CUDA performance	
13	29 Mar	20. CUDA case studies	
	31 Mar	21. Some useful SSE instructions	

14	5 Apr	22. Understanding assembly to identify bottlenecks	
	7 Apr	23. Cell assembly and PTX code	
15	12 Apr	24. Floating point and control flow on GPUs	
	14 Apr	**Midterm 2**	
16	19 Apr	Project presentations	
	21 Apr	Project presentations	
17		**Project demonstrations**	

Grading criteria:

Your overall grade will be based on your performance on (i) midterms, (ii) group project, and (iii) assignments, with weights as given in Table 1. Exams consist of two midterms. Assignments consist of several small programming assignments, whose weights will be announced along with the assignment announcement. The group project has been described above. Note that there is no final exam in this course.

Your average on the midterms should be at least 80% for you to get a course grade of B or better. If you meet this constraint, then the final grade will be determined using Table 2.

Table 1: Course Points	
Item	*Weight*
Midterms	30
Assignments	20
Group Project	50

Table 2: Letter Grades	
Points	*Grade*
92.0 - 100.0	A
90.0 - 91.9	A-
88.0 - 89.9	B+
82.0 - 87.9	B
80.0 - 81.9	B-
0 - 79.9	F - C+

NOTE: You must earn a weighted average of 80% in the midterm exams to be awarded a course grade of B or better. For example, if you obtain a total of 89%, but an exam average of only 78%, then you will not get a B+. Instead, you will get a B-, because that is the highest grade for which you will be eligible without meeting the exam cutoff.

Programming Assignment Assessment

You must understand your assignment work. If you are asked to explain your work, and you are unable to do so, you may be assigned a grade of zero.

Course policies:

Attendance Policy:

The university requires attendance in all classes, and it is also important to your learning. The attendance record may be provided to deans who request it. If your grade is just a little below the cutoff for a higher grade, your attendance will be one of the factors that we consider, in deciding whether to "bump" you up to the higher grade. Three or fewer unexcused absences in lectures and recitations will be considered good attendance.

Excused absences include documented illness, deaths in the immediate family and other documented crises, call to active military duty or jury duty, religious holy days, and official University activities. Accommodations for these excused absences will be made and will do so in a way that does not penalize students who have a valid excuse. Consideration will also be given to students whose dependent children experience serious illness.

You should let me know in advance, when possible, and submit your documentation. You should make up for any materials missed due to absences.

Missed exam Policy:

A missed exam will be recorded as a grade of zero. We will follow the university rules regarding missed final exams (see http://registrar.fsu.edu/dir_class/spring/exam_schedule.htm), for all the exams, including the final exam.

Late Assignment Policy:

In order to enable us to provide timely solutions to assignments, we have the following policy regarding submission of late assignments.

- An assignment that is turned in no more than 24 hours late will be scored with a 10% penalty.
- An assignment that is turned in no more than 48 hours late will be scored with a 20% penalty.
- An assignment that is turned in more than 48 hours late will receive the score of zero, though we will review it and comment on it.
- You should not be late for, or miss, the project demonstration. You will get a grade of zero for the group project if you do so.

Grade of 'I' Policy:

The grade of 'I' will be assigned only under the following exceptional circumstances:

- Due to an extended illness or other extraordinary circumstance, with appropriate documentation, the student is unable to participate in class for an extended period. In this case, arrangements must be made to make up the missed portion of the course prior to the end of the next semester.

Professional ethics:

You will gain confidence in your ability to design and implement algorithms only when you write the code yourself. On the other hand, one does learn a lot through discussions with ones peers. In order to balance these two goals, I give below a list of things that you may, and may not, do.

Things you may not do: You should not copy code from others. This includes directly copying the files, replacing variable names in their code with different names, altering indentation, or making other modifications to others' code, and submitting it as your own. (You may also wish to note that many of the modifications that make codes look very different in a higher level language, yield lower level representations that are very close, and are hence easy to detect.) Furthermore, you should take steps to ensure that others cannot copy code from you -- in particular, you should have all permissions on assignment files and directories set off for others.

Things you may do: You may discuss specific problems related to use of the computer, useful utilities, and some good programming practices, with others. For example, you may ask others about how to submit your homework, or how to use the debugger or text editor.

Honor Code: The Florida State University Academic Honor Policy outlines the University's expectations for the integrity of students' academic work, the procedures for resolving alleged violations of those expectations, and the rights and responsibilities of students and faculty members throughout the process. Students are responsible for reading the Academic Honor Policy and for living up to their pledge *to be honest and truthful and [to] strive for personal and institutional integrity at Florida State University*. (Florida State University Academic Honor Policy can be found at dof.fsu.edu/honorpolicy.htm.)

Plagiarism:

> Plagiarism is "representing another's work or any part thereof, be it published or unpublished, as ones own. For example, plagiarism includes failure to use quotation marks or other conventional markings around material quoted from any source" (Florida State University General Bulletin 1998-1999, p. 69). Failure to document material properly, that is, to indicate that the material came from another source, is also considered a form of plagiarism. Copying someone else's program, and turning it in as if it were your own work, is also considered plagiarism.

ADA: Students with disabilities needing academic accommodation should: (1) register with and provide documentation to the Student Disability Center, and (2) bring a letter to the instructor indicating the need for accommodation and what type. This should be done during the first week of class. This syllabus and other class materials are available in alternative format upon request. For more information about services available to FSU students with disabilities, contact:

> Student Disability Resource Center
> 874 Traditions Way
> 108 Student Services Building
> Florida State University
> Tallahassee, FL 32306-4167

(850) 644-9566 (voice)
(850) 644-8504 (TDD)
sdrc@admin.fsu.edu
http://www.disabilitycenter.fsu.edu

SYLLABUS CHANGE POLICY:

Except for changes that substantially affect implementation of the evaluation (grading) statement, this syllabus is a guide for the course and is subject to change with advance notice.

Last modified: 12 Dec 2009