Math 6590.352: Applications of Parallel Computing
Online Class

Professor: Dana Brunson
E-mail: dana.brunson@okstate.edu (the best way to contact me)
Office: 106 Mathematical Sciences Building
Office Phone: (405) 744-4455
Office Hours: Open door and by appointment.

Overview: Oklahoma State University with XSEDE (www.xsede.org) is pleased to offer this online course on parallel computing for graduate students and advanced undergraduates from diverse disciplines. Students should have programming experience as well as mathematical background in linear algebra and numerical analysis. This is a multi-university course that will be managed and instructed locally by Dana Brunson. The course runs from January 20, 2015 to May 8, 2015. (OSU schedule: Jan 12 - May 8)

Course Description:
This course[i] teaches both graduate and advanced undergraduate students from diverse departments how use parallel computers both efficiently and productively, i.e. how to write programs that run fast while minimizing programming effort. The latter is increasingly important since essentially all computers are (becoming) parallel, from supercomputers to laptops. So beyond teaching the basics about parallel computer architectures and programming languages, we emphasize commonly used patterns that appear in essentially all programs that need to run fast. These patterns include both common computations (eg linear algebra, graph algorithms, structured grids,..) and ways to easily compose these into larger programs. We show how to recognize these patterns in a variety of practical problems, efficient (sometimes optimal) algorithms for implementing them, how to find existing efficient implementations of these patterns when available, and how to compose these patterns into larger applications. We do this in the context of the most important parallel programming models today: shared memory (eg PThreads and OpenMP on your multicore laptop), distributed memory (eg MPI and UPC on a supercomputer), GPUs (eg CUDA and OpenCL, which could be both in your laptop and supercomputer), and cloud computing (eg MapReduce and Hadoop). We also present a variety of useful tools for debugging correctness and performance of parallel programs. Finally, we have a variety of guest lectures by a variety of experts, including parallel climate modeling, astrophysics, and other topics.

- Computer Architectures (at a high level, in order to understand what can and cannot be done in parallel, and the relative costs of operations like arithmetic, moving data, etc.).
    - Sequential computers, including memory hierarchies
    - Shared memory computers and multicore
    - Distributed memory computers
    - GPUs (Graphical Processing Units, eg NVIDIA cards)
    - Cloud and *Grid* Computing
- Programming Languages and Models for these architectures
    - Threads
    - OpenMP
    - Message Passing (MPI)

    - UPC and/or Titanium
    - Communication Collectives (reduce, broadcase, etc.)
    - CUDA/OpenCL etc. (for GPUs)
    - *Cilk*
- Sources of parallelism and locality in simulation: The two most important issues in designing fast algorithms are (1) identifying enough parallelism, and (2) minimizing the movement of data between memories and processors (moving data being much slower than arithmetic or logical operations. We discuss how simulations of real-world processes have naturally exploitable parallelism and "locality" (i.e. data than needs to be combined can naturally be stored close together, to minimize its movement).
- Programming "Patterns": It turns out that there is a relatively short list of basic computing problems that appear over and over again. Good ways to solve these problems exist, and so it is most productive to be able to recognize these "patterns" when they appear, and use the best available algorithms and software to implement them. The list of patterns continues to evolve, but we will present the most common ones, and also illustrate how they arise in a variety of applications.
- Originally, there were 7 such patterns that were identified by examining a variety of high performance computational science problems. Since there were 7, they were called the "7 dwarfs" of high performance computing. For each one, we will discuss its structure and usage, algorithms, measuring and tuning its performance (automatically when possible), and available software tools and libraries.
    - Dense linear algebra (matrix multiply, solving linear systems of equations, etc.)
    - Sparse linear algebra (similar to the dense case, but where the matrices have mostly zero entries and the algorithms neither store nor operate on these zero entries).
    - Structured Grids (where the data is organized to lie on a "grid", eg a 2-dimensional mesh, and the basic operations are the same at each mesh point (eg "average the value at each mesh point with its neighbors").
    - Unstructured Grids (similar to the above, but where "neighbor" can be defined by an arbitrary graph)
    - Spectral Methods (the FFT, or Fast Fourier Transform, is typical).
    - Particle Methods (where many "particles" (eg atoms, planets, people,...) are updated (eg moved) depending on the values of some or all other particles (eg by electrostatic forces, gravity, etc.)
    - Monte Carlo, sometimes also called MapReduce (as used by Google), where every task is completely independent, but may finish at a different time and require different resources, and where the results of all the tasks may be combined ("reduced") to a single answer.
- The next 6 patterns of parallel computing were identified by examining a broad array of nonscientific applications that require higher performance via parallelism; not only did the above "7 dwarfs" appear, but 6 other computational patterns, that we will probably only have time to partially cover: (see here for details):
    - *Finite State Machines, where the "state" is updated using rules based on the current state and most recent input*
    - *Combinational Logic, performing logical operations (Boolean Algebra) on large amounts of data*
    - Graph traversal, traversing a large graph and performing operations on the nodes

    - *"Graphical models" involve special graphs representing random variables and probabilities, and are used in machine learning techniques*
    - *Dynamic Programming, an algorithmic technique for combining solutions of small subproblems into solutions of larger problems*
    - *Branch-and-Bound search, a divide-and-conquer technique for searching extremely large search spaces, like those arising in games like chess*
- More Patterns - there are various other structural patterns that are useful for organizing software (parallel or sequential) that we will cover as well.
- Measuring performance and finding bottlenecks
- Load balancing techniques, both dynamic and static
- Parallel Sorting
- Assorted possible guest lectures (some repeats, some new; depends on availability of lecturers)
    - Performance Measuring and Debugging Tools
    - Parallel Debugging Tools
    - Climate Modeling
    - Computational Astrophysics
    - Computational Biology
    - Computational Nanoscience
    - V*olunteer Computing (eg how seti@home etc work)*
    - *Simulating the Human Brain*
    - *Musical performance and delivery (ParLab application)*
    - *Image Processing (ParLab application)*
    - *Speech Recognition (ParLab application)*
    - *Modeling Circulatory System of Stroke Victims (ParLab application)*
    - *Parallel Web Browers (ParLab application)*

Course Organization:

- Weekly meetings: Monday 3:30-5:00, 445 MSCS
- https://moodle.xsede.org: Homework assignments and reading material, upload homework assignments, discussion forums. We'll follow the due dates shown on the homework submissions pages.
- https://portal.xsede.org: Lectures and quizzes: Login to the portal, go to online training, scroll down and click the link "Applications of Parallel Computers" under the Comprehensive courses which goes to the Cornell Virtual Workshop site (https://www.cac.cornell.edu/VW/apc/default.aspx) with the list of lectures. Note that these all have 3-4 quizzes each. Your score will be the best of the first 3 attempts. To keep pace, you need to complete two lectures per week.
- http://www.cs.berkeley.edu/~demmel/cs267_Spr15/: This is the 2015 version of the course - from this page you can get to the new version of the videos after they are posted on YouTube. You can also find previous versions of the course.

Syllabus Attachment: Please read the OSU syllabus attachment on the web, linked at http://academicaffairs.okstate.edu/faculty-a-staff. This has a lot of important information, including instructions about disability accommodations. Please contact me privately during the first week of the course if you need accommodations as a result of a disability.

Grading: Your grade will be based on quizzes from the lectures, homework assignments and your final project. Most of the grade will be based on a final project (in which students are encouraged to work in small interdisciplinary teams), which could involve parallelizing an interesting application, or developing or evaluating a novel parallel computing tool. Students are expected to have identified a likely project by mid semester, so that they can begin working on it. We will provide many suggestions of possible projects as the class proceeds.

---

[i] http://www.cs.berkeley.edu/~demmel/cs267_Spr12/Syllabus.html