Introduction to High Performance Computing
Fall Semester 2016

COURSE DESCRIPTION:
High performance computing algorithms and software technology, with an emphasis on using distributed memory systems for scientific computing. Theoretical and practically achievable performance for processors, memory system, and network, for large-scale scientific applications. The state-of-the-art and promise of predictive computational science and engineering. Algorithmic kernels common to linear and nonlinear algebraic systems, partial differential equations, integral equations, particle methods, optimization, and statistics. Computer architecture and the stresses put on scientific applications and their underlying mathematical algorithms by emerging architecture. State-of-the-art discretization techniques, solver libraries, and execution frameworks.

PREREQUISITES:
Experience using C/C++ in a Unix environment, familiarity with basic numerical algorithms, and familiarity with computer architecture.

COURSE FLAVOR:
A good subtitle for this course would be *"Algorithms as if architecture mattered."* Architecture increasingly does matter today. During decades of progress using the paradigm of bulk synchronous processing on systems that were small enough to be considered "flat" and tightly coupled, architecture could largely be abstracted away through the message passing interface (MPI), an excellent example of "separation of concerns" in computer science. One could write in a high-level language without concern about where the compiler and runtime stashed the operands, because flops were relatively slow, which made everything else, including the physical layout of the architecture, appear nearly flat. One could count flops for serial complexity estimation, and determine how many could be done concurrently (between synchronization events) for parallel complexity estimation. Today, however, flops are cheap compared to the cost of moving data, in both time and energy expenditure. Therefore, we must worry about the topology of the network and the latencies and bandwidths of every part of the memory system and network in getting the operands to the FPUs. This gives high performance computing an emphasis different from some other types of computing. The same architecture advances that make it frustrating also make it exciting! What new high performance science and engineering computing users need are an introduction to the concepts, the hardware and software environments, and selected algorithms and applications of parallel scientific computing, with an emphasis on tightly coupled computations that are capable of scaling to thousands of processors and well beyond. The course material ranges (selectively) from high-level descriptions of motivating applications to low-level details of implementation, in order to expose the algorithmic kernels and the shifting balances of computation and communication between them. The homeworks range from simple theoretical studies to running and modifying demonstration codes. Modest programming assignments using MPI and PETSc culminate in an independent project leading to an in-class report.

INSTRUCTORS:
The principal lecturer will be David Keyes, Professor of Applied Mathematics and Computational Science, KAUST. Guest lecturers will be invited to speak on their specialties. Lectures from Extreme Computing Research Center staff members highlighting open source scientific software will be incorporated into the course.

GOALS AND SYLLABUS:

The overall goal is to acquaint students who anticipate doing independent work that may benefit from large-scale simulation with current hardware, software tools, practices, and trends in parallel scientific computing, and to provide an opportunity to build and execute sample parallel codes. The software employed in course examples is freely available. The course is also designed to make students intelligent consumers and critics of parallel scientific computing literature and conferences.

Much of the motivation for parallel scientific computing comes from simulations based on discretizations of partial differential equations (PDEs, typically described with sparse matrices), or integral equations (IEs, typically described with dense matrices), or based on interacting particles (unstructured interaction lists, often embedded in octtrees). Of course, many applications are nonlinear, but these are typically approached as a series of linearized analyses. An understanding of the underlying equations, their physical meaning, and their mathematical analysis is important in some parts of the course and opens up many possibilities for independent projects. Other material is easily abstracted away from its underlying operator equation context to that of a generic bulk-synchronous computation that interleaves flows of data with operations on that data. The intention is to provide a course of benefit to a broad clientele of graduate researchers. In addition to computer scientists and applied mathematicians, students from mechanical engineering, electrical engineering, chemical engineering, materials science, and geophysics should find it of interest and approachable if they already have sufficient background in computing to be motivated towards the high end.

Thirteen algorithmic prototypes that occur regularly in scientific computing have been identified in a famous 2006 Berkeley technical report *"The Landscape of Parallel Computing Research: The View from Berkeley"* (UCB/EECS-2006-183). Though ten years old, students may want to download and devour this report as representative of the motivation and flavor of the course. The Berkeley prototypes are: dense direct solvers, sparse direct solvers, spectral methods, N-body methods, structured grids / iterative solvers, unstructured grids / iterative solvers, Monte Carlo (including "MapReduce"), combinatorial logic, graph traversal, graphical models, finite state machines, dynamic programming, backtrack/branch-and-bound. The first seven are essential floating point kernels and the last six essential integer kernels. The course examines several of these kernels in detail.

Lecture coverage includes:

- Introduction to large-scale predictive simulations: the combined culture of CS&E and HPC
- Introduction to parallel architecture and programming models
- Introduction to MPI, PETSc, and other software frameworks for HPC
- Parallel algorithms for the solution of large, sparse linear systems and nonlinear systems with large, sparse Jacobians
- Parallel algorithms for partial differential equations
- Parallel algorithms for N-body particle dynamics

EVALUATION AND GRADING:

Evaluation consists of four components: problem sets, project, final exam, and class participation at the flipped local site. *Problem sets* may be undertaken cooperatively (and this is encouraged), but each student must submit the homework separately under their own name, vouching for their own responsibility for the answers. The quality of the write-up is part of the grade. It is intended that all students should be able to score well on the problem sets, because they will be announced well in advance of their due dates and students have unlimited time for their own reading and research of the topics consultations with one another. The problem sets should create an extended ongoing discussion for the class community. The *project* is intended to be individual. If students want to team to undertake a "bigger" project and earn the same grade for it, this should be negotiated when projects are launched in mid-course. Projects will be submitted in report form, and each project will be featured for a short presentation to the class at the end of the semester. The *final exam* is, of course, individual.

RECOMMENDED RESOURCES:

None of these written resources are "required," but are of potential reference interest. They are not intended to be interchangeable, but are composed for different audiences, with different objectives, but unified around the challenges and opportunities of HPC. Proceeding chronologically back in time, most are out of date in architectural details, due to the rapid evolution of the field, but the principles are mostly timeless.

1) *Introduction to High Performance Scientific Computing*, by V. Eijkhout (Creative Commons, 2015)
2) *Using MPI; Portable Parallel Programming with the Message-Passing Interface, Third Edition*, by W. D. Gropp *et al.* (MIT Press, 2014)
3) *Introduction to High Performance Computing for Scientists and Engineers*, by G. Hager and G. Wellein (CRC Press, 2011)
4) *Applied Parallel Computing*, by Y. Deng (World Scientific, 2011)
5) *Petascale Computing: Algorithms and Applications*, by D. Bader, ed. (Chapman and Hall, 2008)
6) *Scientific Parallel Computing*, by L. R. Scott *et al*. (Princeton, 2005)
7) *An Introduction to Parallel Computing: Design and Analysis of Algorithms,2nd ed.*, by A. Grama *et al*. (Pearson Addison Wesley, 2003)
8) *Sourcebook of Parallel Computing* by J. Dongarra *et al*., eds. (Morgan-Kaufmann, 2002)
9) *Parallel Computer Architecture: A Hardware/Software Approach* by D. E. Culler *et al*. (Morgan Kaufmann, 1999)
10) *High Performance Computing*, K. Dowd and C. Severance (O'Reilly, 1998)

FREQUENTLY ASKED QUESTIONS:

Must I understand PDEs and Linear Algebra well to take this course?

Algorithms for partial differential equation and linear algebraic computations motivate this course and add knowledge of their mathematics adds substance to the parallel applications. *However, the aspects of these subjects that are important to success in this course have to do with understanding the choreography of data and hardware*. If you are comfortable with following the data in these algorithms without a theoretical understanding of how they approximate the real world (modeling) or how rapidly they converge to it (analysis), you can survive this course and even excel in it. Mathematical theorems, e.g., tying convergence of an iterative method to condition number of a matrix have a quality of subroutines: if the upstream hypotheses (inputs) are verified, the consequences (outputs) may be chained into downstream uses in this course, e.g., complexity analyses.

Must I be facile in Unix and C/C++ to take this course?

In this course, you will work with sample applications written in C and you will build and execute on Linux-based distributed systems. *One can pick up what one needs without being an expert in the tools applied.*

Do you have a motto for success in difficult endeavors like high performance computing?

Actually, this is *not* a frequently asked question, but it should be ☺. I do have a motto, taken from the most successful college football coach in history, Bear Bryant (1913—1983), as measured by the number of career wins amassed: **"It's not the will to win, but the will to *prepare* to win that makes the difference."**