High-Performance Numerical Computation: A Course Proposal

Andrew A. Anda
Department of Computer Science
St. Cloud State University
aanda@eeyore.stcloudstate.edu

Abstract

This paper presents a proposal for a quasi-capstone course surveying selected topics in high-performance numerical computing. This course is intended for computer science majors. At St. Cloud State University, we are promoting a new initiative to integrate and incorporate into the computer science major curriculum the study of the effect of computer hardware and architecture on software performance. The study of high-performance numerical computing (HPNC) aptly fulfills this objective.

Introduction

Numerical computation essentially consists of the study, and implementation in software on some architecture, of those algorithms which compute via finite precision (usually floating point) arithmetic. The high-performance qualification in the course title incorporates the issues of relative efficiency and accuracy in comparing those algorithms and software. High-Performance computing requires an understanding of the influence of computer hardware, architecture, and system software resources on efficiency and accuracy.

The boundaries between disciplines tend to be quite fuzzy and indeterminate, but numerical computation may be described as providing a bridge between the mathematical discipline of numerical analysis and the relatively recently defined discipline of computational science. In practice, numerical analysis provides a piecewise-linear discretization of a mathematical continuum which must then be formulated into an algorithm. Software is then developed to implement that algorithm. The latter two processes of algorithm and software development and their analyses are the purview of the computer science subfield of numerical computation. The application of numerical software to infer principles and behaviors of the natural world constitutes the broader field of computational science.

The discretization errors studied in numerical analysis exist regardless of the precision of computation. Usually, the mathematical analyses assume infinite precision calculation. However, algorithms formulated from these discretizations must be implemented via finite precision arithmetic. Finite precision arithmetic adds some degree of computation related rounding error. Any two mathematically equivalent numerical algorithms may exhibit radically different responses to rounding error. Also, on modern computer architectures, in large part due to an inherent multilevel memory hierarchy, the analysis of memory traffic has become at least as important to the analysis of execution time requirements as the more traditional counting of operations. I.e., when and where operations are performed on a set of data has become at least as critical a measure as the number of operations.

Most significant scientific and engineering computational problem solutions are formulated with an underlying matrix computation structure. And those matrix computations tend to dominate the demands for resources during the computation of those solutions. Additionally, problems formulations are often made larger to minimize the effects of discretization errors. Many of these large problem formulations push the limits of feasibility. The feasibility of a numerical computation with respect to the criteria of accuracy and execution time may be effected only through the careful application of the theory and techniques of numerical computation and computer science in general. Therefore, the algorithmic focus of this course will be with matrix computations.

Motivation

A new initiative for curricular development in the Computer Science Department at St. Cloud State University (SCSU) is to develop new courses or modify some existing courses to better demonstrate and study the effect of computer hardware and architecture on software and its performance. The study of the high-performance solution of large matrix problems requires an understanding of the synergy between hardware and architecture on one hand and software and its performance on the other.

There has also been a progressive process of reduction in numerical computation content in the sequence of ACM Computing Curricula recommendations. In fact, the current 2001 Report (Computing Curricula IEEE Computer Society, 1991) has eliminated from the core curriculum all seven lecture hours of *numerical and symbolic computation* recommended by the 1991 Report (Tucker, 1991). While this course is intended as an elective, and will therefore not be servicing the curricular core, it will at least offer an option for student to elect in addressing their deficiencies in the numerical computation knowledge base. Also, the 2001 report eliminated numerical computation as an elective, replacing it with a set of four courses including a course in numerical analysis, which doesn't discuss hardware/architectural considerations, and a course in high-performance computing, which seems to be defined as being about half computational science in content.

This course will discuss various significant, foundational, and fundamental *gotchas* which are often counter-intuitive for students and computer science practitioners. Relatively few students choose a numerical computation related career, but many will occasionally encounter problems relating to it throughout their careers. An ignorance of some of the important principles may have harmful or at least unintended effects. E.g. Kernighan and Richie, in specifying the C language, allowed the compiler to perform associative arithmetic transformations based on mathematical rather than computational equivalence. Even parentheses were insufficient to indicate intent to the compiler. This deficiency was addressed in the subsequent ANSI standard. But, this example serves to illustrate the importance of the numerical computation knowledge base to a computer scientists who aren't performing numerical calculation. In some respects, the world of numerical computation can be likened to falling down the Carrollian rabbit hole – A computer scientist should at least know that they're working on the other side of the looking glass when they encounter numerical computation issues.

The numerical techniques that a student might be exposed to prior to this course, usually through a math class, is often useful in the context of proofs but is often unsatisfactory for many problems either with respect to feasibility or stability. Most introductory matrix computation courses focus on solving only two or three matrix equations (linear, eigenvalue, SVD) assuming general dense matrices where each element is uncorrelated with other elements. Some simple symmetry conditions and structures (e.g. banded or triangular) matrices might be discussed as well. In practice, sparse, low rank, structured matrices, and more complicated matrix equations are often generated by applications. The extra conditions on the structure of the matrices can be exploited to significantly reduce the complexity of some matrix operations and sometimes increase the stability. More complicated matrix equations can usually be decomposed into a set of simpler ones. A student should be able to identify these special matrices and matrix equations so that they can find appropriate software. Also, the complexity of several fundamental matrix operations is reduced significantly if a related problem is solved subsequently, e.g. a rank-one update to a least squares problem, or solving a linear system using another right-hand side vector but the same matrix.

It is also desirable for an upper-division undergraduate or masters level graduate course to draw upon knowledge from a broad set of allied areas of the computer science core, either to introduce or to reinforce knowledge of and experience with these concepts from other core areas. It is in this context that this course is intended to serve a quasi-capstone function. Because the extent and breadth of the foundational core curricular material that this course draws upon is so significant, I consider this course to be a quasi-capstone course for upper division undergraduate and masters students. I have prepended the qualification "quasi" because there will be a significant amount of new material presented, and because the course will not necessarily incorporate a significantly large group project of the type often associated with a capstone course.

This course will serve as as response to these motivations. I.e. what a degreed computer scientist in general should know about high-performance numerical computation.

Objectives

The prerequisite for this course will be a senior level undergraduate preparation, i.e., the essential completion of the core, and a sufficient exposure to the principles and notation of linear and matrix algebra. Because of this high level of required preparation, this course will serve masters students as well. The core curricular subject areas that this course will draw upon are:

data structures E.g. representations for mathematical objects such as polynomials and matrices; quad- and oct-trees.

discrete algorithms Several discrete algorithms (e.g. the matrix chain algorithm) and algorithmic paradigms (e.g. dynamic programming or divide and conquer) are directly applicable to problems in high-performance numerical computation. Conversely, some discrete problems, such as some graph theory problems, can be solved more efficiently with assistance from numerical computation.

algorithm analysis and complexity Analysis of the time and space complexity of nested loop structures, recursive computation, and sequences of simpler operations.

programming languages Programming language features which are beneficial or detrimental to numerical computation.

software engineering Commonalities in developing and maintaining numerical and non-numerical software

hardware and architecture complexity of arithmetic functional units, the memory systems and networks.

Because the discipline of high-performance numerical computation cannot be covered in depth, even with a primary focus on matrix computations, this course is intended to serve primarily as a survey course with some programming assignments intended to reinforce concepts. There are, however, some topics that will be emphasized:

conditioning Algorithm vs. problem condition and stability. Matrix norms.

IEEE 754 floating point standard Standard floating point arithmetic and exceptions – problems and solutions: precision, range, rounding, wobble, ULPs, cancellation, flags, hardware and architecture, programming language support, and mixed precision.

programming languages Which programming language design elements aid or impede effective high-performance numerical computation

libraries essential software libraries: Lapack, C++ template libraries.

memory The memory hierarchy, bandwidth, distributed memory, promoting data locality: cache, banks

BLAS 1-3 level BLAS, blocking, recursive BLAS, use of the BLAS in numerical software.

FFT Complexity and implementation of the FFT, and applications of the FFT to reducing the complexity of some structured matrix problems.

parallel A basic parallel architecture taxonomy – shared vs. distributed; systolic.

canonical matrix decompositions algorithms, analysis, and nomenclature

matrix equations identification and nomenclature

discrete algorithms applications of the matrix chain, recurrence relations, recursion, graphs, Strassen's matrix product, and binary powers algorithms.

matrix algorithms direct vs. iterative; exploiting structure.

sparse matrices algorithms and data structures; general sparse, symmetric, banded, blocked.

optimization optimizing compiler levels and directives, optimizing source code for space, time, portability, and architecture.

My expectation for students who successfully complete the course will be that they can identify the kind of numerical problem they have, find literature and software describing its solution, and exploit the best existing software for solving their numerical problems. They should also be able to program reasonably simple equations stably.

Other Issues

My own research area and area of graduate coursework and research in computer science is that of matrix computations. I have taught a subset of this course once before as an experimental course. My pacing proved to be problematical. I believe I was covering some topics in too much detail at the expense of breadth. I allowed any programming language to be used, but Java caused problems because of the inability to adjust the level of compiler optimization. Because the students are most familiar with C++, I expect that they will do most of their programming in C++. If Matlab and Fortran are available to the students when I offer this course again, I

will use them as well. Large non-square matrix multiplication is a good project for examining a variety of computational issues including. There is no one text which covers all of the above objectives. However, A basic text on matrix computations may be supplemented by a wealth of online materials including tutorials and research papers, e.g. (Goldberg, 1991) and (Microsystems, 1991).

Closing

I would hope students who complete this course would gain an understanding of how essential computer science theory and practice is in supporting the computational solution of science and engineering problems, and that they can discover how mathematically identical algorithms can exhibit such computationally different behaviors. I would also hope that the students would enjoy a course which ties together and exhibits the relevance of a significant number of computer science theory and practice that they had been previously exposed to in a more compartmentalized manner. My intent is to ultimately have this course approved as a standard elective in my department.

References

Computing Curricula IEEE Computer Society, A. The Joint Task Force on. (1991). *Computing curricula 2001; computer science; final report.* Los Alamitos, CA: IEEE Computer Society.

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computer Surveys*, *23*, 5–48.

Microsystems, S. (1991). *Numerical computations guide.* (Sun Microsystems, 1991. Numerical Computations Guide, Document number 800-527710.)

Tucker, e. a., A.B. (1991). *Computing curricula 1991: Report of the acm/ieee-cs joint curriculum task force.* New York: Association for Computing Machinery.