# High-Performance Numerical Computation: A Quasi-Capstone Course Proposal

Andrew A. Anda,
St. Cloud State University

April 12, 2003

# 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.

**numerical analysis** Discretization of continuous assuming infinite precision

**numerical computation** ⇳

**computational science** Computational modeling of natural systems.

- Most significant scientific and engineering computational problem solutions are formulated with an underlying matrix computation structure.
- These matrix computations tend to dominate the demands for resources during the computation of those solutions.
- Problem 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.
- $\Rightarrow$ This course will focus on matrix computation algorithms and their implementation.

- Any two mathematically equivalent numerical algorithms when computed via finite precision arithmetic may exhibit radically different responses to rounding error.
- 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.

# Motivation

- $\exists$ new initiative for curricular development in the CSCI Dept. at SCSU 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.

- This course will discuss various significant, foundational, and fundamental *gotchas* which are often counter-intuitive for computer science students and 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.
- Numerical techniques that a student might be exposed to prior to this course, usually through a math class, are often useful in the context of proofs but are often unsatisfactory or suboptimal for many problems either with respect to feasibility or stability.

- It is 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 added the *quasi-* qualification because the course will not necessarily incorporate a significantly large group project of the type often associated with a capstone course.

## Prerequisites

- 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.

# Objectives

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 are to solve;
- find literature and software describing its solution;
- exploit the best existing software for solving their numerical problems;
- program reasonably simple equations stably.

## Other Issues

- When I teach this course again, because the students are most familiar with C++, I will expect that they will do most of their programming in C++.
- If Matlab and Fortran are available to the students, I will exploit either of both those languages as well.
- Java, however, causes problems in part because of the inability to adjust the level of compiler optimization.
- Large non-square matrix multiplication is a good project for examining a variety of computational issues.
- 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.

# 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;
- discover how mathematically identical algorithms can exhibit such computationally different behaviors;
- 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.