Syllabus for CS 471/Math 471
High Performance Scientific Computing

Brian T. Smith, CS Dept./HPCERC/ARC
Andrew C. Pineda, HPCERC/ARC
Richard C. Allen, Sandia National Laboratories

Fall, 1997

Lectures: M, W 4:00-4:50 PM, TAPY Rm. 217, Section I
T, Th. 4:30-5:20 PM, ME Rm. 220, Section II

Laboratory: M, W 5:20-6:50 PM, ESCP Rm. 109, Section I.
M, W 5:20-6:50 PM, ESCP Rm. 110, Section II

Office Hours: M, T, W, Th. 2:00-3:30PM (Brian Smith)
??? (Andy Pineda)

Prerequisites

Programming knowledge and experience, Fortran or C, knowledge of calculus, introductory knowledge of ODEs, linear algebra, numerical methods, knowledge and experience with UNIX. Note: most of this material is reviewed and although expertise and current knowledge in all of these is not required nor assumed, the student is responsible for reviewing or gaining knowledge in these areas to be able to cope with the material. I will assist in any way to gain this knowledge for those who are unfamiliar with the material or who have not used it for some time. Lack of current knowledge in all of these areas will lead to problems in keeping up with the lectures and laboratories for this course.

Course Material

Text Book: An Introduction to High Performance Scientific Computing, Fosdick et al., MIT Press.
Laboratory Exercises: ~acpineda/CS471/... on both the CIRT and AHPCC machines.
Auxiliary Material: CSEP Web Text Book, http://csep1.phy.ornl.gov/csep.html
Getting Started: ~acpineda/CS471/GettingStarted.txt

Offices

Prof. Brian T. Smith
2nd floor, Galles Building Showroom
1601 Central Ave. NE.
Albuquerque, NM 87131
Phone: 505-277-8249 or 8337
Email: bsmith@arc.unm.edu

Dr. Richard C. Allen
Org. 9205
SNL
Albuquerque, NM 87185-1110
Phone: 505-845-7825
Email: rcallen@cs.sandia.gov

Dr. Andrew C. Pineda
1st Floor, Galles Building Showroom
1601 Central Ave. NE.
Albuquerque, NM 87131
Phone: 505-277-8249
Email: acpineda@arc.unm.edu

Support

CS 471/Math 471 is being offered this fall semester both as a required course in UNM's Scientific and Engineering Computational Program for graduate students and as part of a new post non-degree graduate program in Computational Science. We thank and acknowledge the support of SNL to help establish this non-degree graduate program. To better accommodate the backgrounds of the students, we are offering two sections this semester, Section I for the post-degree students that treats more advanced application topics and involves a major project, and Section II for the UNM regular students that provides a one or two more introductory application and involves more directed laboratory material. The material presented in the lectures, except for the applications, will be the same.

Lecturers

Lectures will be given by several UNM faculty and staff, and SNL staff. Your primary instructors are: Prof. Brian T. Smith (CS/UNM), Dr. Andrew C. Pineda, and Dr. Richard Allen (SNL).

Acknowledgments

Much of this course is based on material developed under an NSF Educational Infrastructure grant CDA-9017953 at the University of Colorado's Computer Science Department, by investigators Prof. Lloyd Fosdick, Prof. Elizabeth Jessup, Ms. Carolyn Schauble, and Prof. Gitta Domik. Although it has been modified to suit the needs of the various programs at UNM, its unique methodology in Computer Science to inform students of issues in scientific computing is due to the ideas and teaching expertise of Lloyd Fosdick and others at the University of Colorado: many thanks for a very interesting and effective teaching methodology for this cross-disciplinary subject.

Introduction

The principal objective of this course is to introduce you to the use of high performance computing systems in science and engineering. High performance computing systems are systems that deliver computational results at a rate of over a billion floating point operations per second sustained over a few minutes to a few hours or longer; it is expected that, within the next few years, applications that use over a trillion floating point operations over sustained over a few minutes will become commonplace. High performance scientific computing uses such machines to solve science and engineering problems as well as to model and simulated complex physical systems to gain insight and understanding into how they behave.

This course is designed to expose students to many of the tools and techniques used to solve advanced and complex scientific problems. The course surveys basic tools such as programming languages, problem solving environments, visualization environments, and basic computing methodologies from modeling and simulation techniques to visualization techniques. The course emphasizes hands-on experience through the use of computational laboratories that require experiments with computational models and the

reporting of these experiments through logbooks to be maintained throughout the semester.

What We Will Study

The fastest computers today operate at peak speeds of around several hundred gigaflops (several hundred billion floating point operations per second). Recently Intel and Sandia announced that they were able to sustain a computation at the rate of around one teraflop (10^12 floating point operations per second) on the ASCI Red machine! You can get some feeling for this speed when you realize a machine operating at 100 gigaflops can do about 100 arithmetic operations in the time it takes for light to travel one foot. Today's high speed have been achieved by improved chip technology and by new computing architectures that support parallel computing. These new architectures are a challenge to the algorithm designer and the programmer because of their complexity and the fact that failure to take into account their special characteristics, such as individual central processing unit (CPU) performance and the interconnection network, can result in significant loss of performance.

Therefore, we will study algorithms and software designed to achieve good performance on these systems. This will be done in the context of solving problems that are typical of those solved in the "real world". Problems of practical interest arise in such diverse areas of science as medicine, celestial mechanics, the study of fluid flows, the design of buildings and airplanes, and quantum mechanics. The problems to be solved are computationally intensive and tend to require large amounts of data. We will see that data access patterns on modern CPUs and on parallel machines are critically important to achieving high performance.

A good understanding of algorithms for high performance computing requires a knowledge of numerical analysis. Without this knowledge you cannot understand why one algorithm is more accurate than another or why one will produce a result of given accuracy in less time. Furthermore, and of great practical importance, you cannot be confident of the results of a computation without the understanding of numerical methods used to produce them and how numerical rounding errors propagate (imagine how rounding errors can accumulate when you are performing 10^12 computations per second and your program runs for several days for each result). Thus, we will study aspects of numerical analysis, with particular attention to linear algebra and the solution of differential equations.

Another important issue in high performance computing is how to present the results of a computation. In most cases this must be done in a graphic form, not only because it is the most natural way to do it and leads to insights about the physical system, but also because the sheer quantity of numbers produced (as many as 10^12 per second) makes it impractical, if not impossible, to print them all. We will study how to present the results of computation in graphic form, not only as simple x-y plots but as realistic pictures of the physical system. This subject is often called "scientific visualization".

Parallel computing is becoming the only way to reach

the high performance needed to solve the challenging modeling and simulation problems of our times. Of the many programming paradigms and tools which support parallel program development, we study just two of them: the message passing paradigm as illustrated by the Message Passing Interface, and data parallelism, as illustrated by High Performance Fortran. Each of these parallel programming tools also is used to illustrate a second major difference in approach: the hands-on detailed control of MPI versus the "compiler-knows-best" strategy of the High Performance Fortran. We still study the pros and cons of each approach.

Finally, we will study computer performance itself. We will study how to measure performance, how to characterize it in terms of performance parameters, and how to use tools to enhance performance.

The Approach of This Course

The approach taken in this course is to use laboratory exercises and case studies to learn the important aspects of high performance scientific computing and in particular the use of this computing tool to perform modeling and simulation of our most significant scientific and engineering problems facing you in the future. The basic material is presented in lectures followed by time to perform the exercises and experiments described in a series of laboratory manuals during lab periods. At least one person will be present at all labs to assist you with the assigned exercises.

Log books, describing your work in the lab periods, must be kept by you and will be handed in and marked three times during the semester. A Fortran 90 quiz is set for week ??? to allow you to assess your performance in the class. A second quiz will be scheduled ??? (week 11). Depending on the section, a further quiz will be given or projects will be assigned; this are to be determined later in the semester.

Programming Languages

In scientific computing, Fortran is the predominant language, mainly due to its development as the first high level programming language to be officially standardized, but also do to its strong support for numerical computing. Fortran 90 is introduced because of its strong support for numeric precision control and array processing. In addition, it is the basis for the data parallel programming paradigm used in this course, namely standard High Performance Fortran. Fortran 95, now a standard, has adopted many of the array extensions to Fortran 90 used in High Performance Fortran. You are welcome to use other languages such as C and data parallel C, now just becoming available, but as stated below, the laboratory material is based on Fortran.

The course uses much sample code written in Fortran 77 that needs to be modified. The student is free to rewrite the exercises and perform the laboratory experiments in an other language of his/her choice but little if any sample

code in other languages is available at this time. (Much of the example code is being rewritten in Fortran 90 and will be rewritten at the same time in C and C++, but it is unlikely that much of it will available during this semester.)

About The Computing Machines

The computers used are those available for general use at UNM and Sandia National Laboratory. The UNM computers are those at the Albuquerque High Performance Computing Center (AHPCC) and the central computing facility CIRT. The AHPCC machines include currently a 32-node IBM SP1 with the SP2 high performance switch, an HP 735, a Sun Sparc 20, a Sun Ultrasparc clone, an IBM RS/6000 model 370, several IBM PowerPCs (models 25T, 43P), an SGI Onyx Reality, an SGI Onyx Reality 2, and several Pentiums running Linux. The CIRT machines include a 4-node IBM SP2 with an SP3 high performance switch, a large collection of IBM RS/6000 370 machines, an IBM RS/6000 J30 4-processor shared memory processor, and an SGI Onyx Reality. The Sandia machine is 54-node Paragon. All of these machines run a version of UNIX, compatible with either System V or BSD 4.2 Unix. CIRT maintains a very large modem bank, accessed via the phone number 277-9990, from which both CIRT and AHPCC machines are accessible. AHPCC has a small modem back (approximately 10 lines), available from the phone number 277-????

About The Software And Software Tools

Software, provided on the local UNM machines, includes several Fortran 90 compilers, Matlab Version 4.3, and various parallel computing libraries such as MPI, PVM, and P4. The compilers behave differently and have different support and debugging tools available from the vendor. For example, the NAG f90 compiler is a good debugging compiler with various debugging flags like -C (check subscripts), -P (checks pointers) but is a very poor code optimizer (at least on IBM workstations -- however, there is an optimizing NAG f90 compiler for Sun platforms). On the other hand, the IBM xlf, xlhpf, and xlc compilers are very good optimizing compilers with often very surprising optimizations (specified by options on the compiler) that drastically improve the performance of programs used in the laboratory experiments; these performance improvements are not described in the laboratory manuals on computer performance.

Beware that many of the UNIX tools such as Make, Vi, debuggers, etc. behave differently depending on which version is being used. The versions may be Gnu software, vendor supplied (IBM, HP, Sun), or Linux, all of which can have slightly different behaviors. Before spending a lot of time looking for bugs, ask me, the lecturer, or your colleagues for help. This course relies on these tools but is not the subject of the course. See the document GettingStarted in ~acpineda/CS471 for further information on some of these tools.

Classes And Laboratory Sessions

The classes and laboratories are held in the rooms listed at the beginning of this document. The lecture material is presented before it is needed in the laboratory sessions. The laboratory sessions are available for

performing the exercises and experiments with someone present at all times to answer questions about the lectures, laboratory exercises and experiments, and projects.

In some cases, the results described in the laboratory material are based on the use of a DECstation 5000/240, particularly the computer performance laboratory. This machine is a RISC (reduced instruction set computer) architecture, like the IBM RS/6000 machines, but is considerable slower with much less sophisticated hardware. Thus, the descriptions in the lab manuals for DECstation are similar but not the same as for the IBM machines but they still remain appropriate. The major differences arises from the fact that the IBM RS/6000 has multiple functional units and relies on cache storage more so than the DECstation.

Quizzes, Exams, Laboratory Log Books, And Projects:

The schedule for the quizzes, submission of the log books, and final projects is as follows:

Item	Date Held or Due
Fortran 90 Quiz	????
Log Book 1	????
Log Book 2	????
Project Proposals	????
Quiz	????
Log Book 3	????
Project Presentations and Report	????

All quizzes are open book and are expected to take an hour.

Logbooks

The logbooks describe the observations and results obtained from the exercises performed in the laboratory manuals. The answers to any questions in the exercises should be provided in the logbooks. Comments on the exercises, what you have learned from the exercises, and questions that remain in your mind should be recorded in the logbook. The purpose of the logbook is to record your thoughts and questions as you proceed through the exercises.

To provide more structure and detailed specification for the logbooks, it should reference the each exercise as you commenting on and should include the following points as appropriate for each exercise:

- the results of performing the exercise
- whether the exercise worked as expected or the results were anomalous
- any questions you had about the results
- what did you learn from the exercise
- analysis of the results

The logbook can either be a written record in an exercise book or recorded in a file created as you are working on the exercises in the lab period. The logbook should be printed, and if you wish, formatted with a text formatter of your choice.

For example, suppose the following exercise was given, after having attended a lecture on IEEE floating point number representations and how the programming language does its arithmetic (that is, what precision it uses).

Exercise 6.1: Write a program (C or Fortran) that executes the following statements:

```
Assign to the variable x the value of the constant 1.1
Print out x with lots of digits
Assign to y the value 1.0 and  z the value 3.0
Assign to zz the value of y/z
Print  out  the value  of  the expression  1.0 - 3.0*(y/z)
Print the machine precision for the type used for x
```

Repeat the program making sure x is represented at a higher precision and the division is performed at this higher precision.

Logbook entry: I wrote the following Fortran program for exercise 6.1

```
x = 1.1
print '(a,e20.12)', 'x= ', x
y = 1.0; z = 3.0; zz = y/z
print '(a,e20.12)', 'The result of the expression 1.0-3.0*zz is ', 1.0 - 3.0*zz
print *, 'The epsilon (machine precision) for the type used to represent x is, epsilon(
x)
end
```

The results were: 1.10000002384 and -2.98023223877E-06 (these are obtained from a PC using the Salford f90 compiler -- what is printed is very dependent on the hardware and software being used). To my surprise, 1.1 did not print as 1.1 but a number slightly larger than 1.1. On second thought, 1.1 is not representable so the input/output conversion routines print the decimal expansion of the binary representation of 1.1 which when listed to 12 digits is larger than 1.1 The second result is not zero as expected but is a small number. It looks reasonable because the division by 3.0 will produce a number near one-third but not exactly as one-third cannot be presented and when multiplied by 3. produces a number near 1.0 but slightly larger. The size of the result is reasonable because the precision of the arithmetic with this compiler and machine is 32 bits (standard IEEE binary arithmetic) which has a precision of 23 binary digits, approximately 10^(-7). (Actually 2^(-23) is 1.1920929E-07.)

For the second part of the exercise, I declared x, zz and y to be double precision and repeated the calculation. This time, .the result for x was the same and the result for the expression was 0.55e-16. I expected initially x to be a number close to 1.1 within 16 digits, but on second thought about how Fortran represents literal constants, I realized that 1.1 is a single precision constant and this is an approximation to 1.1 to single precision. To obtain what I expected, I would have to change 1.1 to 1.1d0. For the second result, 1.0 and 3.0 are represented exactly in double precision, the division is done in double precision, and thus the result is accurate to double precision, that is, approximately 16 digits.

Depending on the Section of the course, two or three case cases will be presented. For both sections, the first case study will be an introduction to molecular dynamics taken from your text book (IHPSC), followed by the corresponding lab. For Section I, the next lab will be a follow-on in the areas of molecular dynamics using a more advanced algorithm in the case study and then will be followed by the case study on advection. Instead of the follow-on molecular dynamics lab and advection case study, Section II will be presented with case studies from two guest lecturers to be determined later. The case studies will involve three lectures on the topic with exercises to be worked on during the lab sessions.