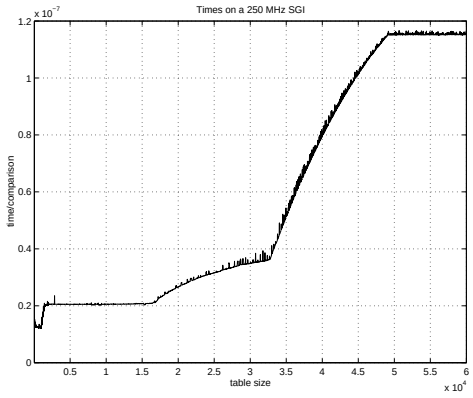
Lecture Notes on High Performance Computing



Thomas Ericsson
Mathematics
Chalmers
2008

High Performance Computing

Thomas Ericsson, computational mathematics, Chalmers

office: L2075
phone: **772 10 91**
e-mail: **thomas@math.chalmers.se**

My homepage:
http://www.math.chalmers.se/~thomas

The course komepage:
http://www.math.chalmers.se/Math/Grundutb/CTH/tma881/0708

Assignments, copies of handouts (lecture notes etc.), schedule can be found on the www-address.

We have two lectures and two labs per week.

Prerequisites:

- a programming course (any language will do)
- the basic numerical analysis course
- an interest in computers and computing; experience of computers and programming

If you do not have a Chalmers/GU-account, you need to contact the helpdesk to get a personal account.

Why do we need HPC?

- Larger and more complex mathematical models require greater computer performance. Scientific computing is replacing physical models, for example.
- When computer performance increases new problems can be attacked.

To solve problems in a reasonable time using available resources in a good way we need competence in the following areas:

1. algorithms
2. hardware; limitations and possibilities
3. code optimization
4. parallelization and decomposition of problems and algorithms
5. software tools

This course deals primarily with points 2-5. When it comes to algorithms we will mostly study the basic building blocks.

The next pages give some, old and new, examples of demanding problems.

For each problem there is a list of typical HPC-questions, some of which you should be able to answer after having passed this course.

A real-time application

Simulation of surgery; deformation of organs; thesis work (examensarbete) Prosolvia. Instead of using a standard deformation more realistic deformations were required. Organs have different structure, lung, liver etc.

Requires convincing computer graphics in real time; no flicker of the screen; refresh rate $\geq$ 72 Hz.

Using a shell model implies solving a sparse linear system, $\boldsymbol{Ax} = \boldsymbol{b}$, in 0.01 s (on a 180 MHz SGI O2 workstation, the typical customer platform).

- What do we expect? Is there any chance at all?
- Is there a better formulation of the problem?
- What linear solver is fastest? Sparse, band, iterative, ...?
  - Datastructures, e.g.
  - General sparse: $\{j, k, a_{j,k}\},\ a_{j,k} \neq 0,\ j \geq k$.
  - Dense banded.
  - Memory accesses versus computation.
- Should we write the program or are there existing ones? Can we us the high performance library made by SGI?
- If we write the code, what language should we use, and how should we code in it for maximum performance?
- Can we find out if we get the most out of the CPU? How? Are we using 20% of top speed or 90%?
- What limits the performance, the CPU (the floating point unit) or the memory?

Integrals and probability

Graduate student in mathematical statistics. What remained of the thesis work was to make large tables of integrals:

$$\int_0^\infty f(x)\,dx, \quad \text{where} \quad f(x) \to \infty \quad \text{when} \quad x \to 0+$$

The integrand was complicated and was defined by a Matlab-program. No chance of finding a primitive function. Using the Matlab routine **quad** (plus a substitution), to approximate the integral numerically, the computer time was estimated to several CPU-years. Each integral took more than an hour to compute.

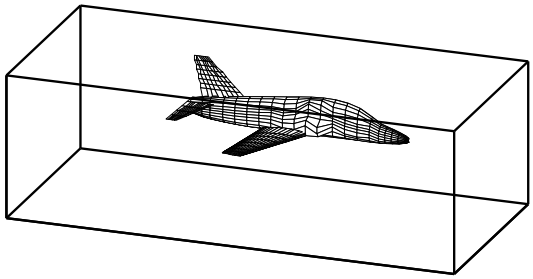
- Is this reasonable, is the problem really this hard?
- Are Matlab and **quad** good tools for such problems?
- How can one handle singularities and infinite intervals?

Solution: Switching to Fortran and Quadpack (a professional package) the time for one integral came down to 0.02 s (with the same accuracy in the result).

- Matlab may be quite slow.
- **quad** is a simple algorithm; there are better methods available now, e.g. **quadl**.

Mesh generation for PDEs in 3D

require huge amounts of storage and computer time.
Airflow around an aircraft; 3 space dimensions and time.
CFD (Computational Fluid Dynamics).



Discretize (divide into small volume elements) the air in the box and outside the aircraft. Mesh generation (using **m3d**, an ITM-project, Swedish Institute of Applied Mathematics) on one RS6000 processor:

```
wed may 29 12:54:44 metdst 1996  So this is old stuff
thu may 30 07:05:53 metdst 1996

183463307 may  2 13:46 s2000r.mu

tetrahedrons in structured mesh:   4 520 413
tetrahedrons in unstructured mesh: 4 811 373
```

- Choice of programming language and data structures.
- Handling files and disk.

Now we must solve the PDE given the mesh...

More PDEs: Weather forecasts

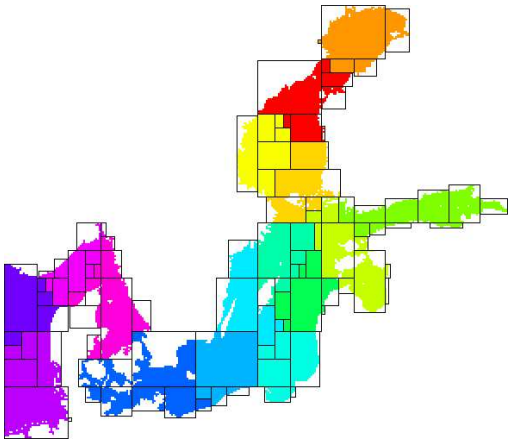
SMHI, Swedish Meteorological and Hydrological Institute.
HIRLAM (HIgh Resolution Limited Area Model).
HIROMB (HIgh Resolution Operational Model of the Baltic).

Must be fast. "Here is the forecast for yesterday."

Parallel implementation of HIROMB, lic-thesis KTH.

Divide the water volume into pieces and distribute the pieces onto the CPUs.

- Domain decomposition (MPI, Message Passing).
- Load balancing. Communication versus computation.
- Difficult sub-problems. Implicit solver for the ice equations. Large (10^5 equations) sparse Jacobians in Newton's method.



Three weeks runtime

Consultant work, for Ericsson, by a colleague of mine.

Find the shape of a TV-satellite antenna such that the "image" on the earth has a given form (some TV-programs must no be shown in certain countries).

Algorithm: shape optimization + ray tracing.

Three weeks runtime (for one antenna) on a fast single-CPU PC. One of the weeks for evaluating trigonometric functions.

You do not know much about the code (a common situation). Can you make it faster? How? Where should you optimize?

- Choice of algorithm.
- Profiling.
- Faster trigonometric functions. Vector forms?
- Parallel version? How? Speedup? (OpenMP)

A Problem from Medicine

Inject a radionuclide that attacks a tumour.
How does the radioactivity affect the surrounding tissue?

To be realistic the simulation should contain some $7 \cdot 10^7$ cells.
Matlab-program computing a huge number of integrals (absorbed dose).
The original program would take some 26 000 years runtime.

After switching to Fortran90, changing the algorithm, by precomputing many quantities (the most important part) and cleaning up the code, the code solved the problem in 9 hours on a fast PC (a speedup by a factor of $2.6 \cdot 10^7$).

Contents of the Course

There are several short assignments which illustrate typical problems and possibilities.

- Matlab (get started exercises, not so technical).
- Uniprocessor optimization.
- Low level parallel threads programming.
- MPI, parallel programming using Message Passing.
- OpenMP, more automatic threads programming.
- Talk, something relevant to the course.

Many small and short programs, matrix- and vector computations (often the basic operations in applications). Simple algorithms.

E.g. test how indirect addressing (using pointers) affects performance:

```
do k = 1, n
  j    = p(k)   ! p is a pointer array
  y(j) = y(j) + a * x(j)
end do
```

- You will work with C, Fortran and some Matlab and several software tools and packages.
- Java is not so interesting for HPC (so only in the lectures).

At most two students per group. Each group should hand in written reports on paper (Swedish is OK). Not e-mail.

There are deadlines, see www.

In more detail...

- A sufficient amount of Fortran90 (77), C for the labs.
- Computer architecture, RISC/CISC, pipelining, caches...
- Writing efficient programs for uniprocessors, libraries, Lapack, ...
- Necessary tools: `make`, `ld`, `prof`, ...
- Introduction to parallel systems, SIMD, MIMD, shared memory, distributed memory, network topologies, ...
- Unix processes, `fork` and `exec`.
- Communicating using shared memory, `shmget`, `shmctl`, ...
- POSIX threads, pthreads.
- MPI, the Message Passing Interface.
- Shared memory parallelism, OpenMP.
- Parallel numerical analysis, packages.

Note: this is not a numerical analysis course. We will study simple algorithms (mainly from linear algebra).

You will not become an expert in any of the above topics (smörgåsbord), but you will have a good practical understanding of high performance computing.

The course will give you a good basis for future work.

Literature

You can survive on the lecture notes, web-pages and man-pages.

I used to recommend:

K. Dowd & C. Severance, High Performance Computing, O'Reilly, 2nd Edition July 1998, 466 pages.
`http://www.ora.com`. The book is a bit dated and can be hard to find (out of print).

Here follow some reference books I have (or will have) in my bookshelf (nice to have, but not necessary for the course). There are many special books in HPC and a huge number of books dealing with parallel computers. Some books are available as E-books (Books24x7) through the Chalmers library homepage.

A web-version of Designing and Building Parallel Programs, by Ian Foster, 1995, can be found at:
`http://www-unix.mcs.anl.gov/dbpp` .

For more free manuals for code optimization and system reference see the course homepage.

A few C- and Fortran books (there are many)

- B. W. Kernighan, D. M. Ritchie, The C Programming Language (2nd ed.), Prentice Hall, 1988. Get the ANSI C version.
- P. van der Linden, Expert C Programming : Deep C Secrets, Prentice Hall, 1994.
- M. Metcalf, J. Reid, M. Cohen, Fortran 95/2003 Explained, 3rd ed. (2nd ed., 1999, is OK as well, E-book).
- Suely Oliveira, David E. Stewart, Writing Scientific Software: A Guide for Good Style, Cambridge UP, 2006. E-book.

Code Optimization

- Randy Allen, Ken Kennedy, Optimizing Compilers for Modern Architectures: A Dependence-based Approach, Morgan Kaufmann, 2001.
- Stefan Andersson, Ron Bell, John Hague, Holger Holthoff, Peter Mayes, Jun Nakano, Danny Shieh, Jim Tuccillo, RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide, 1998,
IBM Redbook (`www.redbooks.ibm.com`, free).
- Steve Behling, Ron Bell, Peter Farrell, Holger Holthoff, Frank O'Connell, Will Weir, The POWER4 Processor Introduction and Tuning Guide, 2001,
IBM Redbook (`www.redbooks.ibm.com`, free).
- Originally written by David Cortesi, based on the first edition by Jeff Fier; updated by Jean Wilson and Julie Boney, Origin 2000 and Onyx2 Performance Tuning and Optimization Guide, Document Number: 007-3430-003, 2001, (free `techpubs.sgi.com`).
- I. Crawford, K. Wadleigh, Software Optimization for High Performance Computing: Creating Faster Applications, Prentice Hall, 2000 (Hewlett Packard).
- Rajat P. Garg, Ilya Sharapov, Illya Sharapov, Techniques for Optimizing Applications: High Performance Computing, Prentice Hall, 2001 (Sun UltraSPARC platforms).
- Kevin Smith, Richard Gerber, Aart J. C. Bik, The Software Optimization Cookbook, High Performance Recipes for IA 32 Platforms, Intel Press, 2005. E-book.

Computers

- John L. Hennessy, David A. Patterson, Andrea C. Arpaci-Dusseau, Computer Architecture: A Quantitative Approach (with CDROM), Morgan Kaufmann, 2006.
- W. R. Stevens, Advanced Programming in the UNIX Environment, Addison Wesley, 1992.

- A. Oram, S. Talbott, Managing Projects with make, 2nd ed., O'Reilly, 1991.
- R. Mecklenburg, Managing Projects with GNU Make, 3rd ed, O'Reilly, 2004.
- G. Anderson, P. Anderson, Unix C Shell Field Guide, Prentice Hall, 1986.

Parallel Programming

- Yukiya Aoyama, Jun Nakano, RS/6000 SP: Practical MPI Programming, IBM Redbook (`www.redbooks.ibm.com`, free).
- D. R. Butenhof, Programming With Posix Threads, Addison Wesley, 1997.
- Rohit Chandra, Dave Kohr, Ramesh Menon, Leo Dagum, Dror Maydan, Jeff McDonald, Parallel Programming in OpenMP, Morgan Kaufmann, 2001. E-book.
- Barbara Chapman, Gabriele Jost, Ruud van der Pas, David J. Kuck, Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation), MIT Press, 2007.
- William Gropp, Ewing Lusk, and Anthony Skjellum, Using MPI, 2nd Edition, MIT Press.
- Lucio Grandinetti (ed), Grid Computing: The New Frontier of High Performance Computing, Elsevier, 2005. E-book.
- Timothy G. Mattson, Beverly A. Sanders, and Berna L. Massingill, Patterns for Parallel Programming, Addison Wesley Professional, 2004.
- Peter Pacheco, Parallel Programming with Mpi, Morgan Kaufmann, 1997. (Not MPI-2)
- Wesley Petersen, Peter Arbenz, Introduction to Parallel Computing A practical guide with examples in C, 2004, Oxford UP (E-book)
- Michael J. Quinn, Parallel Programming in C with MPI and OpenMP, McGraw-Hill Education, 2003.

- Laurence T. Yang and Minyi Guo (eds), High Performance Computing: Paradigm and Infrastructure, John Wiley, 2006. E-book.

Beowulf computers

- Robert Lucke, Robert W. Lucke, Building Clustered Linux Systems, Prentice Hall, 2004
- Karl Kopper, The Linux Enterprise Cluster, No Starch Press, 2005
- William Gropp, Thomas Sterling, Ewing Lusk, Beowulf Cluster Computing with Linux, MIT press, 2003

Numerical methods

- D. P. Bertsekas, J. N. Tsitsiklis, Parallel and distributed computation, numerical methods, Prentice-Hall, 1989
- P. Bjorstad, Domain Decomposition Methods in Sciences and Engineering, John Wiley & Sons, 1997.
- J. J. Dongarra, I. S. Duff, D. C. Sorensen, H. A. van der Vorst, Numerical Linear Algebra on High-performance Computers, SIAM, 1998.
- Robert A. van de Geijn and Enrique S. Quintana-Ortí, The Science of Programming Matrix Computations, Free download `http://www.lulu.com/content/1911788`. About FLAME-project (Formal Linear Algebra Methods Environment).

Programming languages for HPC

The two dominating language groups are Fortran and C/C++.

Fortran90/95/2003 is more adapted to numerical computations. It has support for complex numbers, array operations, handling of arithmetic etc. New code is written in Fortran90/95/2003 (Fortran66/77 is used for existing code only).

Fortran90/95 has simple OO (Object Oriented) capabilities (modules, overloading, but no inheritance).

C++ is OO and has support for some numerics (standard library) or can be adapted using classes, overloading etc.

C is less suited for numerical computing (my opinion).
Too few built-in tools and it cannot be modified.

C/C++ is almost the only choice when it comes to low level unix programming. It is not uncommon to code the computational part in Fortran and a computer graphics (or unix) part in C/C++.

Commercial Fortran compilers generate fast code: competition (benchmarks), simple language, no aliasing. One of Fortran's most important advantages. Speed is (almost) everything in many applications.

It is harder to generate fast code from C/C++.
It is very easy to write inefficient programs in C++.

More about performance later on.