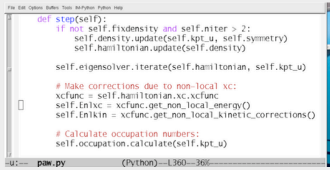
Software development for computational science

Jussi Enkovaara

Software Engineering

CSC – The finnish IT center for science



CSC

Outline

- **CSC – The finnish IT center for science**
- **Computational science**
- **Software development in high performance computing**
  - General needs
  - Programming languages
  - Optimization
  - Parallel processing
- **Summary**

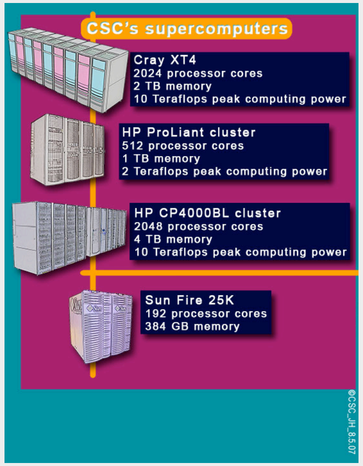
CSC

CSC – The finnish IT centre for science

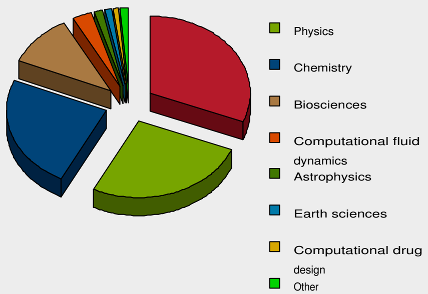
- **Limited non-profit company, owned by the Finnish Ministry of Education**
- **~ 150 emplyees**
- **Computational services for Finnish universities and research institutes**
  - Supercomputing
  - Scientific applications and databases
  - Network service
  - Information management services

CSC

Computing environment of CSC



Usage of processor time by discipline 2006

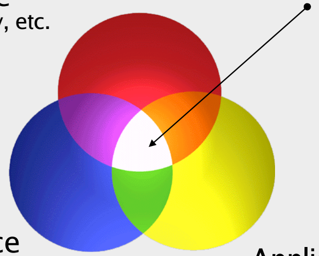


CSC

What is computational science?

Computational Science
Teamwork and Collaboration

Science Discipline
Physics, Chemistry, Biology, etc.

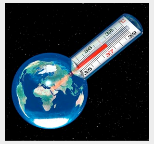


Computer Science
Hardware/Software

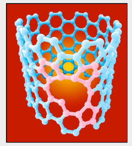
Applied Mathematics
Numerical Analysis, Modeling, Simulation

CSC

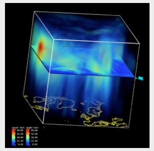
Applications of computational science



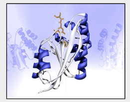
Climate change



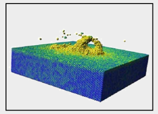
Nanoscience



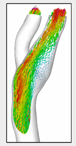
Weather forecasts



Drug design



Material physics



Fluid dynamics

CSC

Ingredients of computational science

- **Modeling**
  - set up the mathematical equations describing the problem
  - the governing equations are often known e.g. Navier-Stokes equation, Schrödinger equation etc.
- **Simulation**
  - write or use existing computer program which solves the equations numerically
  - high performance computing
- **Analysis and visualization**
  - analysis of results may also require programming

CSC

Software development in high performance computing

- **General development needs**
  - version control
  - choice of programming language(s)
  - debugging tools
  - testing schemes
- **Specific needs in high performance computing**
  - achieving high performance with single processor in different platforms
  - efficient parallel processing
  - profiling tools

CSC

Development tools

- **Version control**
  - cvs, svn, etc.
  - easy access to version history
  - code syncronization with multiple developers
- **Regression testing**
  - test sets
  - automatic testing
- **Programming tools**
  - syntax highlighting in editors
  - automated building
  - debugging utilities
  - integrated development environments (IDE)
- **Profiling tools**
  - help to pinpoint the bottlenecks of the program efficiency

CSC

Programming languages

- **Efficiency of development vs. execution efficiency**
  - human work is expensive
  - ease of development and maintanance
- **Portability**
  - (super)computers have life cycle of few years
  - programs may be run in different architectures
- **Access to external libraries**
  - most time consuming parts e.g. linear algebra operations can be performed by optimized external libraries
- **Traditional languages**
  - Fortran, C
- **"Modern" languages**
  - C++, Java, Python, Perl

CSC

Programming languages

- **Compiled, "low" level languages**
  - Fortran, C, C++
  - produce typically fast code
  - development may be more cumbersome, e.g. each code change requires recompiling
  - interfaces to external libraries
- **Interpreted scripting languages**
  - Python, Perl
  - fast development and prototyping
  - native code relatively slow
  - possible to combine with compiled languages
  - special (e.g. parallel) profiling tools do not necessarily support these languages

CSC

Optimization

- **Find the bottlenecks of the program**
  - timing calls in program
  - profiling tools
  - typically, a program spends 90 % of time in 10 % of code
- **Efficiency of algorithm**
  - optimization of the algorithm is typically the most efficient way to improve the performance of the program
- **Try to use optimized external libraries**
  - many vendors provide highly tuned libraries for linear algebra (BLAS, LAPACK), FFTs etc.
- **Optimize the actual code**
  - in current systems, memory access is often the crucial factor

CSC

Basic linear algebra subprograms (BLAS)

- **Many algorithms can be written in terms of matrix-vector operations**
- **BLAS provides standard building blocks for many vector and matrix operations**
- **In addition to standard BLAS (www.netlib.org/blas) there are implementations optimized for specific architectures**
- **Interfaces to Fortran, C and Java**
- **Example: matrix-matrix multiplication in Opteron 2.6 GHz**
  - 600x600 double precision matrix
  - simple implementation with compiler optimizations takes 0.58 s
  - BLAS call 0.094 s

Parallel processing

- **Parallel processing is used to**
  - speed-up program execution
  - recuce the needed memory per processor
- **Todays desktop processors start to contain multiple cores (dual core, quad core, ...)**
- **Current supercomputers can use thousands of commodity processor**
  - Top 10 supercomputers have all over 10000 processing cores, the top one having over 200 000 processors!
- **Utilization of large number of processors is major challenge in software development**

Classes of parallel computers

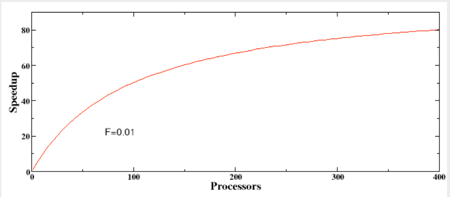
- **Shared memory systems**
  - all the processors can access all of the memory
  - OpenMP compiler directives
  - simple to use
  - limited scalability
- **Distributed memory systems**
  - message passing is used to communicate between processors
  - MPI interface
  - requires more work from the programmer
  - in trivial problems can scale to arbitrarily large number of processors

Speedup of parallel program

- **Upper limit by Amdahl's law:**

$$Speedup = \frac{1}{F + (1-F)/N},$$

where F is sequential fraction and N number of processors



- **Maximum speedup is 1/F**
  - if 1 % of program is sequential, maximum speedup is only 100!
  - When number of processors is doubled from 200 to 400, program will execute only 20 % faster
- **In practice, scaling is much worse**

Software development projects in CSC

- **Elmer**
  - multiphysical modeling with finite element method
  - Fortran90, MPI-parallelization
- **GPAW**
  - atomistic modeling within density functional theory
  - Python + C, MPI-parallelization
- **FinHPC**
  - optimization of selected programs
  - only implementations are optimized
- **Chipster**
  - user-friendly interface for DNA microarray data analysis
  - Java
- **SOMA**
  - molecular modelling environment
  - Perl programs and XML-schemas, operated through www-browser

Open vs. closed source software

- **Proprietary software**
  - there are high quality commercial programs for some problems
  - user friendly graphical interfaces
  - “black boxes”
  - no access to source code, no own extensions
- **Open source software**
  - correct functioning of program (i.e. correct results) can be checked
  - code can be extended depending on users needs