High-performance computing and programming

Introduction to this course
- Practical matters
- Goals
- Motivation
First and foremost

Make **sure** that you have a working UNIX account (log in on the systems in P2510)

Make **sure** that you have access to this course in Studentportalen (studentportalen.uu.se)

You must register for this course online.
Personal introductions

Marcus (marcus.holm@it.uu.se, P2404)
Lectures, labs, assignments

Pavol (pavol.bauer@it.uu.se, P2403)
Lectures, labs, assignments
Course goals

- To pass, the student should be able to
  - transform algorithms in the computational science area to efficient code
  - write, organize and handle large programs for numerical computations
  - use tools for performance optimization and debugging
  - analyze code with respect to performance and suggest and implement performance improvements.
Course outline

Lectures:

http://www.it.uu.se/edu/course/homepage/hpb/vt13/lectures

Week 12-15: C programming and debugging in our lab environment (also Easter break)

Week 16: Performance analysis

Week 17-20: Optimization techniques

Week 20-21: Various topics in HPC
Labs

Lab 1: getting started with C, makefiles, etc
Lab 2: debugging with ddd/gdb
Lab 3: profiling
Lab 4: vectorization

No report needed, oral examination in the lab room

*Individual* reports required for anyone unable to attend

Lab 1 optional but *highly recommended* if you are unfamiliar with C or our UNIX systems.
Assignments

Emphasis is placed on the analysis and insights in your reports.

Assignment 1: Performance analysis

Describe the performance behavior of a few different implementations of a given algorithm. Code will be provided.

Assignment 2: Program coding and optimization

N-body problem

- Simulating a gravitational system efficiently
Examination

Mandatory *individual* miniproject
Examinatory lab on May 27

Make arrangements ahead of time if you cannot make this date.
Practical aspects, assignments

Assignments preferably done in pairs

Assignments should be runnable on lab machines

Handed in through Studentportalen
Practical aspects, labs

Get your UNIX account in order
Labs will take place in room 2510D.

You can work in the assignment groups you choose

Lab reports (for missing students) and late assignment reports will be corrected at a pace that suits your teachers.

Please try logging in to studentportalen.uu.se
A lecture schedule is published on the course site, linked through Studentportalen.

Reading suggestions provided in that schedule.

Lectures and feedback on assignments are critical aspects of course content.

Lecture slides will be published in Studentportalen.

(but they're lecture slides)
Information about corrections, room changes, etc will be emailed out through Studentportalen and written to the Studentportalen homepage.

Make sure you receive emails from Studentportalen!
Moore's law

Moore’s law's end?
High-performance computers

Tianhe-1A. 14,336 Xeon CPUs + 7,168 Nvidia Tesla GPGPUs, 229 TB main memory, 2.5 petaFLOPS peak performance (Nov 2010).

Kalkyl, 348x2 Xeon CPUs, 9.5 TB main memory, 20.5 TFlops.
What is happening?

Computations used to be expensive
  Cheaper than ever
Memory used to be expensive
  Cheaper than ever
…but not as cheap as computing
Moving data becomes expensive
Bandwidth and latency are limiting factors
What is happening?
What do we learn in this course?

Writing code for a single core/thread:

1. Access memory
2. Do something
3. Write back to memory

"We should forget about small inefficiencies, about 97% of the time. Premature optimization is the root of all evil".

~Donald Knuth, living legend in computer science
It is much easier to optimize correct code than it is to correct optimized code.

Priorities (in order):
Correctness
Flexibility
Performance
Old software is good software

Well-documented and tested software is a very valuable asset

Always **consider** which **existing libraries** that you can build on

Corollary: What resources can *you* provide to the rest of the world?

Slow code can also be incorrect and inflexible

What happens when you get more data, what if you want higher precision?

Make your code age with dignity!
Productivity

Programmer time is not free
Not when writing new code
Not when maintaining old code

Consider the total time needed to solve the problem you need to solve
Complex software

Hardware complexity is increasing
Software complexity increases to make use of it

Scientific computations are often multidisciplinary, multiscale, multiphysics, multiplatform, ...

More tasks are performed in real-time or subject to other constraints (e.g. on embedded systems)
Algorithm choice and complexity

Performance is dependent on what you do
Only rarely on exactly how you do it

Examples:
Searching for a text by scanning every file, or using a small index
Doing 3D by ray-tracing versus Z-buffer rendering
Processing a small preview in real-time versus recomputing the full dataset for each change

Algorithm choice affects performance vastly
Implementation choices affects performance only a little
Example

- Solution using
  - Gauss-Seidel Method
  - Multi-Grid Method
Final notes

Remember:

Make sure that you Upunet-S account works

- studentportalen.uu.se

Make sure that your UNIX account works

- Log in on the systems in situ or connect by ssh

Lab 1 strongly recommended