Advanced Systems Lab
Spring 2021, Lecture 1

Instructors: Markus Püschel, Ce Zhang
TAs: Joao Rivera, several more

Picture: www.tapety-na-pulpit.org

Minds open...

... Laptops closed

slide by Bertrand Meyer
Today

Motivation for this course
Organization of this course

Scientific Computing

Physics/biology simulations

Consumer Computing

Audio/image/video processing

Embedded Computing

Signal processing, communication, control

Computing

Unlimited need for performance

Large set of applications, but ...

Relatively small set of critical components

(100s to 1000s)

- Matrix multiplication
- Discrete Fourier transform (DFT)
- Viterbi decoder
- Shortest path computation
- Stencils
- Solving linear system
- ....
Scientific Computing (Clusters/Supercomputers)

- Climate modelling
- Finance simulations
- Molecular dynamics

Other application areas:
- Fluid dynamics
- Chemistry
- Biology
- Medicine
- Geophysics

Methods:
- Mostly linear algebra
- PDE solving
- Linear system solving
- Finite element methods
- Others

Consumer Computing (Desktop, Phone, …)

- Photo/video processing
- Audio coding
- Security

Image compression

- Original
- JPEG
- JPEG2000

Methods:
- Linear algebra
- Transforms
- Filters
- Others
**Embedded Computing (Low-Power Processors)**

*Sensor networks*  
*Cars*  
*Robotics*

**Computation needed:**  
- Signal processing  
- Control  
- Communication

**Methods:**  
- Linear algebra  
- Transforms, Filters  
- Coding

**Classes of Performance-Critical Functions**

- Transforms  
- Filters/correlation/convolution/stencils/interpolators  
- Dense linear algebra functions  
- Sparse linear algebra functions  
- Coder/decoders  
- Graph algorithms  
  *several others*

See also the 13 dwarfs/motifs in  
[http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf](http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf)
How Hard Is It to Get Fast Code?

- **Algorithms**
- **Software**
- **Compilers**
- **Microarchitecture**

```
“compute Fourier transform”
```
```
“fast Fourier transform”
```
```
O(nlog(n)) or 4nlog(n) + 3n
```
```
e.g., a C function
```
```
optimized executable
```
```
high runtime performance
```

How well does this work?

The Problem: Example 1

DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)

Runtime [s]

- Straightforward “good” C code (1 KB)

or

?
The Problem: Example 1

DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)

Performance [Gflop/s]

straightforward “good” C code (1 KB)
The Problem: Example 1

DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)
Performance [Gflop/s]

- Straightforward "good" C code (1 KB)
- Fastest code (1 MB)
- Vendor compiler, best flags

Roughly same operations count
The Problem: Example 2

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz

Performance [Gflop/s]

Vendor compiler, best flags
Exact same operations count (2n³)

Model predictive control
Eigenvalues
LU factorization
Optimal binary search organization
Image color conversions
Image geometry transformations
Enclosing ball of points
Metropolis algorithm, Monte Carlo
Seam carving
SURF feature detection
Submodular function optimization
Graph cuts, Edmond-Karps Algorithm
Gaussian filter
Black Scholes option pricing
Disparity map refinement

Singular-value decomposition
Mean shift algorithm for segmentation
Stencil computations
Displacement based algorithms
Motion estimation
Multiresolution classifier
Kalman filter
Object detection
IIR filters
Arithmetic for large numbers
Optimal binary search organization
Software defined radio
Shortest path problem
Feature set for biomedical imaging
Biometrics identification
“Theorem:”

Let $f$ be a mathematical function to be implemented on a state-of-the-art processor. Then

$$\frac{\text{Performance of optimal implementation of } f}{\text{Performance of straightforward implementation of } f} \approx 10\text{–}100$$

Evolutions of Processors (Intel)

Evolutions of Processors (Intel)

Floating point peak performance [Gflop/s]
CPU Frequency [GHz]

- Cores: 8x
- Vector units: 8x
- parallelism: work required
- ~360 Gflop/s
- ~3 GHz
- free speedup
- 8 cores


Evolutions of Processors (Intel)

Floating point peak performance [Gflop/s]
CPU Frequency [GHz]

- memory bandwidth (normalized)

And there is Processor Variety ...

**ARM Cortex-A7**

**Nvidia Tesla**

**Domain-specific (here: Tile)**

**FPGA accelerators**

---

**DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)**

Performance [Gflop/s]

```
382 = mm_addsub_ps(t128, U247);
383 = mm_add_ps(t268, mm_addsub_ps(U247, mm_shuffle_ps(t275, t275, _MM_SHUFFLE(2, 3, 0, 1))));
384 = mm_add_sub_ps(t268, mm_addsub_ps(U247, mm_shuffle_ps(t275, t275, _MM_SHUFFLE(2, 3, 0, 1))));
386 = mm_sub_ps(t270, U247);
387 = mm_mul_ps(t277, mm_set1_ps(-0.70710678118654757));
```

```
t282 = t283 + t284;
t285 = t283 + t286;
t283 = mm_setzero_ps();
```

Compiler doesn’t do the job
Doing by hand: **nightmare**
Summary and Facts I

Implementations with same operations count can have vastly different performance (up to 100x and more)

- A cache miss can be 100x more expensive than an operation
- Vector instructions
- Multiple cores = processors on one die

Minimizing operations count ≠ maximizing performance

End of free speed-up for legacy code

- Future performance gains through increasing parallelism
Summary and Facts II

It is very difficult to write the fastest code
- Tuning for memory hierarchy
- Vector instructions
- Efficient parallelization (multiple threads)
- Requires expert knowledge in algorithms, coding, and architecture

Fast code can be large
- Can violate "good" software engineering practices

Compilers often can’t do the job
- Often intricate changes in the algorithm required
- Optimization blockers
- No good way of evaluating choices

Highest performance is in general non-portable

Performance is different than other software quality features
Performance/Productivity Challenge

Current Solution

Legions of programmers implement and optimize the same functionality for every platform and whenever a new platform comes out
Better Solution: Autotuning

Automate (parts of) the implementation or optimization

Research efforts

- Linear algebra: Phipac/ATLAS, LAPACK, Sparsity/Bebop/OSKI, Flame
- Tensor computations
- PDE/finite elements: Fenics
- Adaptive sorting
- Fourier transform: FFTW
- Linear transforms: Spiral
- ...many more since then
- New compiler techniques

Promising area but much more work needed ...

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz

Performance [Gflop/s]

- Multiple threads: 4x
- Vector instructions: 4x
- Memory hierarchy: 20x

This Course
This Course: Goals

Obtain an understanding of performance (runtime)

Learn how to write *fast code* for numerical problems
- *Focus: Memory hierarchy and vector instructions*
- *Principles studied using important examples*
- *Applied in homeworks and a semester-long research project*

Learn about autotuning

Today

Motivation for this course

Organization of this course
Course: Times and Places

Lectures:
- Monday 10-12, HG F3
- Thursday 9-10, HG F3

Extra sessions: Only used when announced on website
- Wednesday 14-16, ETF C1

Course deregistration rule:
- Deadline: Second Friday in March
- After that: drop out = fail

Course Website Has all Info

https://acl.inf.ethz.ch/teaching/fastcode/
About this Course

Head TA: Joao Rivera

Other TAs: Eliza Wszola, Konstantin Taranov, Theodoros Theodoridis

Course website has ALL information

Questions: fastcode@lists.inf.ethz.ch

Finding project partner: fastcode-forum@lists.inf.ethz.ch

About this Course (cont’d)

Requirements

- solid C programming skills
- matrix algebra
- Master student or above

Grading

- 40% research project
- 30% midterm exam
- 30% homework

Wednesday slot

- Gives you scheduled time to work together
- Occasionally I will move lecture there (check course website)
- By default will not take place
Research Project: Overview

Teams of 4
Yes: 4

*Topic:* Very fast implementation of a numerical problem

*Until March 12th:*
- find a project team
- suggest to me a problem or pick from list (on course website)
  - Tip: pick something from your research or that you are interested in
- Register on project website + you get a git repo for project

Show “milestones” during semester: One-on-one meetings

Give short presentation end of semester

Write 8 page standard conference paper (template on website)

Submit final code

Finding Project Team

Teams of 4: no exceptions

Use fastcode-forum@lists.inf.ethz.ch:
- “I have a project (short description) and am looking for partners”
- “I am looking for a team, am interested in anything related to visual computing”

In the beginning all of you are registered to that list

Once team is formed register it in our project system
Finding Project

Pick from list on website or select on yourself

Projects from website: number of teams is limited, once picked it is final

Select yourself:
- Pick something you are interested in
- Nothing that is dominated by standard linear algebra (matrix-matrix mult, solving linear systems), no stencil computations
- Send me a short explanation plus a publication with algorithm

Exact scope can be adapted during semester
- reduced to critical component
- specialized

You are in charge of your project!
- If too big, adapt
- If too easy, expand
- Don't come after 2 months and say project does not work

Organize Project

Work as a team

Start asap with a team meeting

Keep communicating regularly during semester

Be fair to your team members

Being able to work as a team is part of the exercise

Be a team player

If you don't contribute I will fail you for the project
Research Project: Possible Failures

Don’t do this:
- never meet
- not respond to emails
- “I don’t have time right to work on this project in the next few months, why don’t you start and I catch up later”
- “I have a paper deadline in 1 month, cannot do anything else right now”
- while not desperate(project-partners) do
  “I do my part until end of next week”
  ... nothing happens ...
- end
- “why don’t you take care of the presentation”
- “why don’t you take care of the report, I’ll do the project presentation”

Single point of failure:
- One team member is the expert on the project and says: I quickly code up the basic infrastructure, then the three of you can join working on parts
- 1 month later, the “quickly coding up” ...

Midterm Exam

Covers first part of course

*Date:* April 21st

*No substitute date*

*There is no final exam*
Homework

Done individually

Solving homeworks analogous to homeworks in prior years is no guarantee for full points

Exercises on algorithm/performance analysis

Implementation exercises
- Concrete numerical problems
- Study the effect of program optimizations, use of compilers, use of special instructions, etc. (Writing C code + creating runtime/performance plots)

Homework is scheduled to leave time for research project

Small part of homework grade for neatness

Late homework policy:
- No deadline extensions, but
- 3 late days for the entire semester (at most 2 for one homework)

Workload During Semester (Sketch)
Academic Integrity

Zero tolerance cheating policy (cheat = fail + being reported)

Homeworks
- All single-student
- Don’t look at other students code
- Don’t copy code from anywhere
- Don’t share your code or solutions
- Ok to discuss things – but then you have to do it alone

We use Moss to check copying (check out what it can do)

*Don’t do copy-paste*
- code
- ANY text
- pictures
- especially not from Wikipedia

Background Material

See course website and links in slides

Prior versions of this course: see website

I post all slides, notes, etc. on the course website

Training material: midterms and homeworks from prior years
Class Participation

I’ll start on time

All material I cover goes on the website, but not my verbal explanations

It is important to attend but not obligatory (obviously)

Do ask questions

*If you drop the course, please unregister in mystudies*