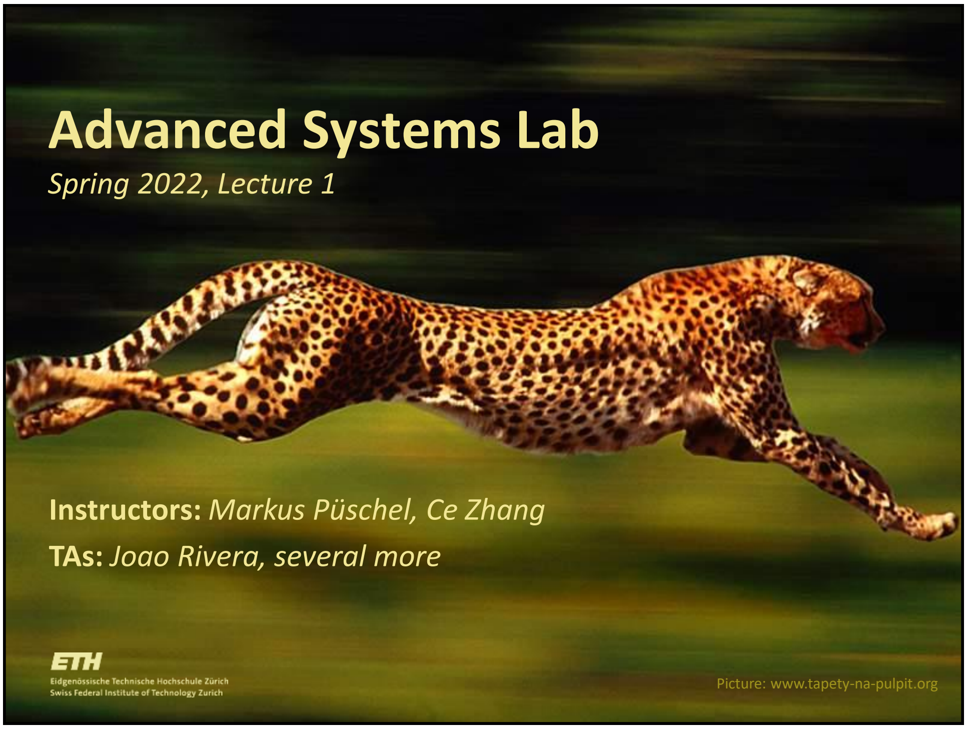


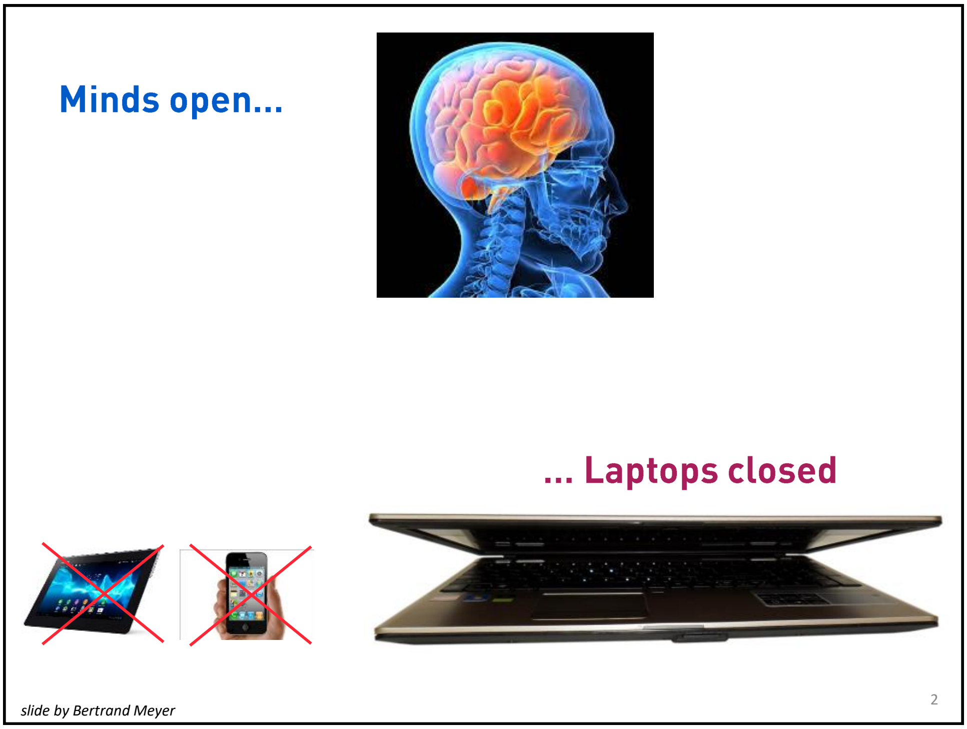
Advanced Systems Lab

Spring 2022, 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

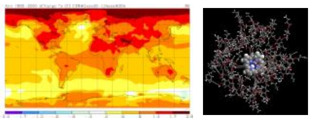
2

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

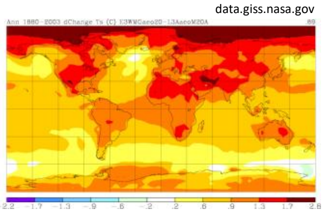
Large set of applications, but …

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)

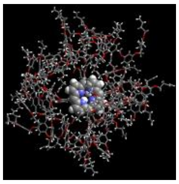
data.giss.nasa.gov



Climate modelling

Finance simulations

www.foresight.org



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

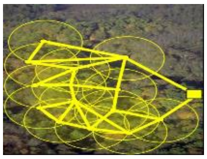
Original JPEG JPEG2000

Image compression

Methods:
- Linear algebra
- Transforms
- Filters
- Others

Embedded Computing (Low-Power Processors)

www.dei.unipd.it



Sensor networks

www.ece.drexel.edu

Cars

www.microway.com.au

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

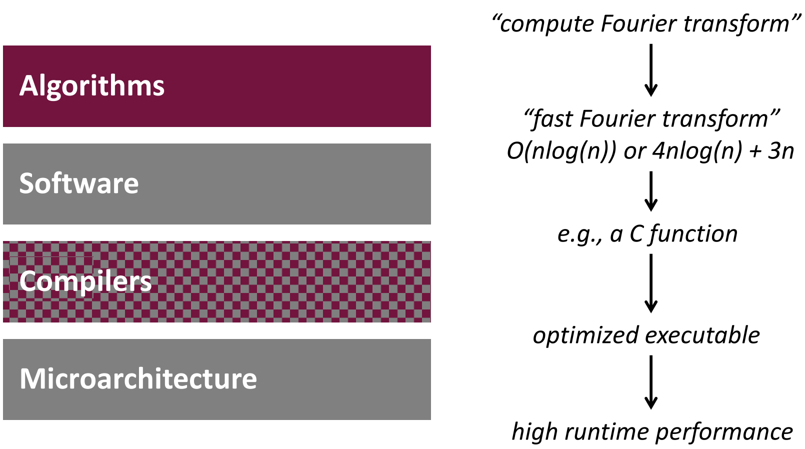
Coder/decoders

Graph algorithms

… several others

See also the 13 dwarfs/motifs in
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

How Hard Is It to Get Fast Code?

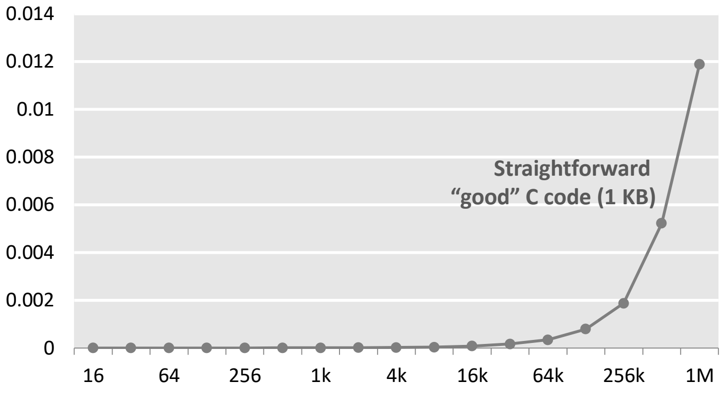


How well does this work?

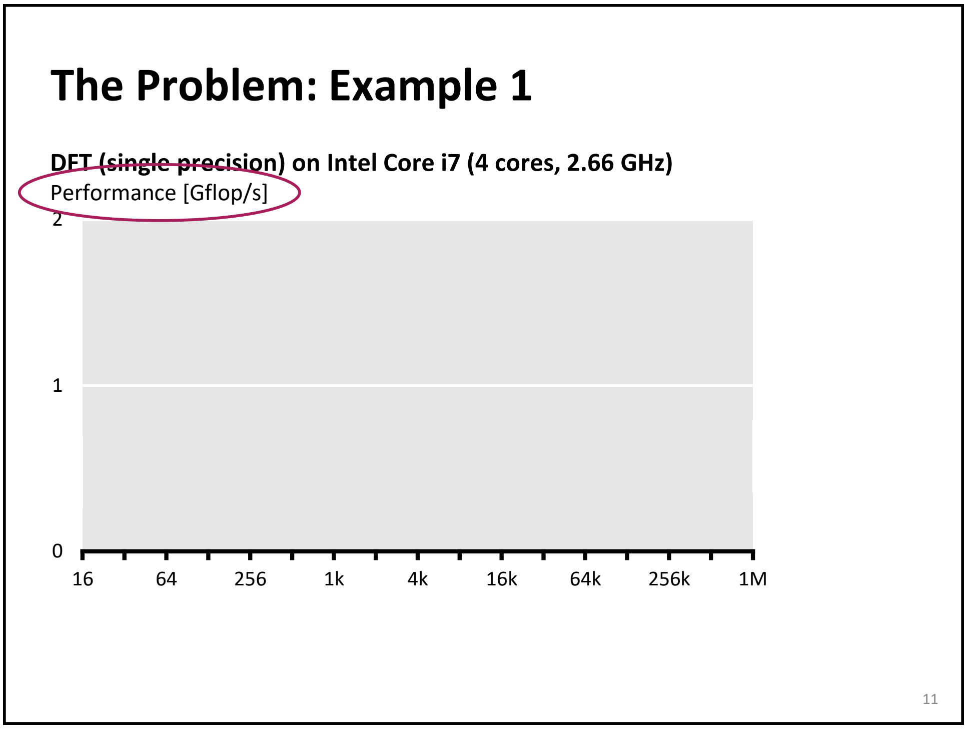
The Problem: Example 1

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

Runtime [s]



or **?**

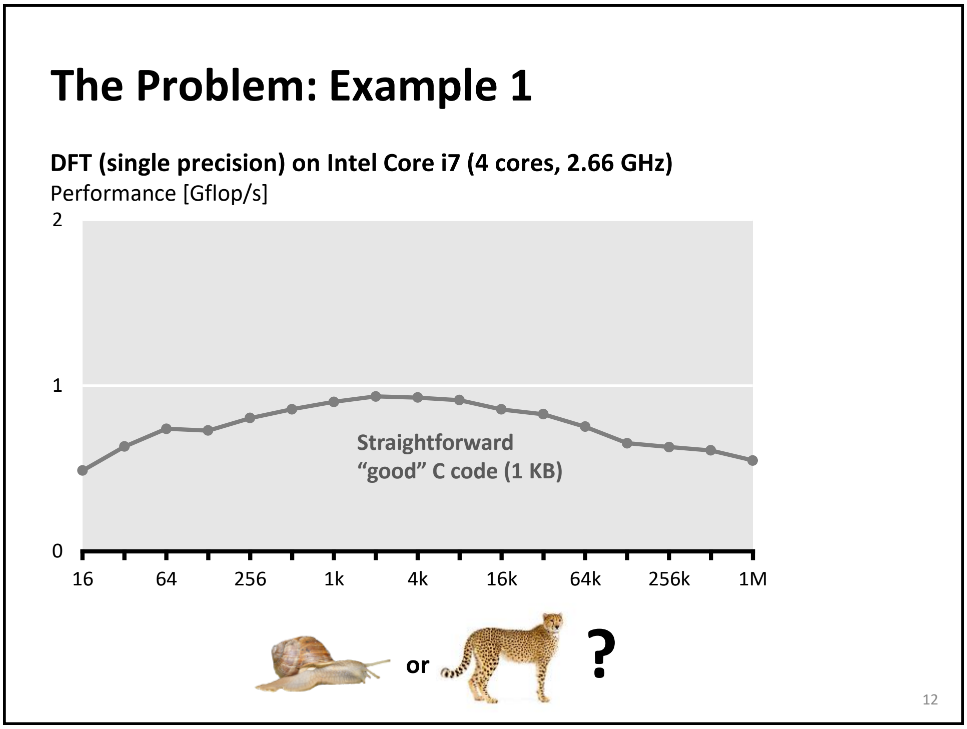


The Problem: Example 1

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

Performance [Gflop/s]

11



The Problem: Example 1

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

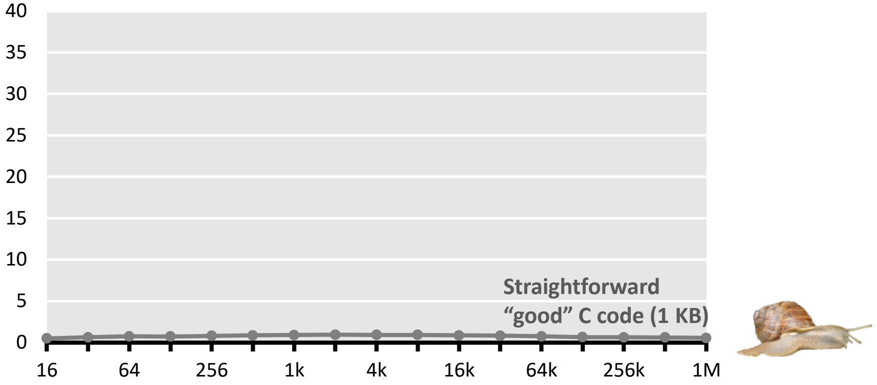
Performance [Gflop/s]

12

The Problem: Example 1

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

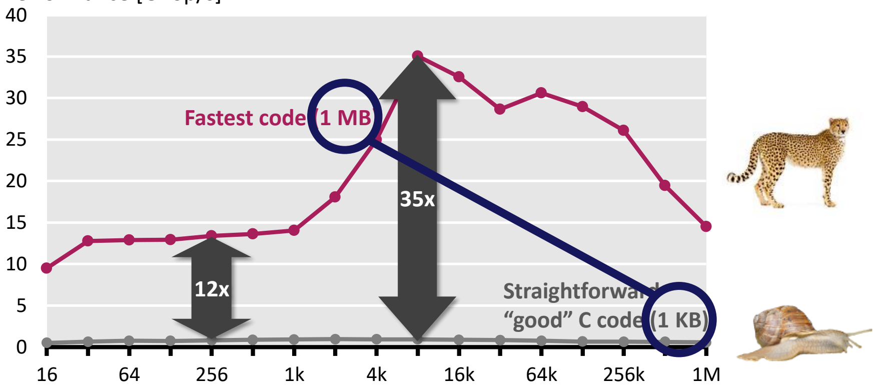
Performance [Gflop/s]



The Problem: Example 1

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

Performance [Gflop/s]

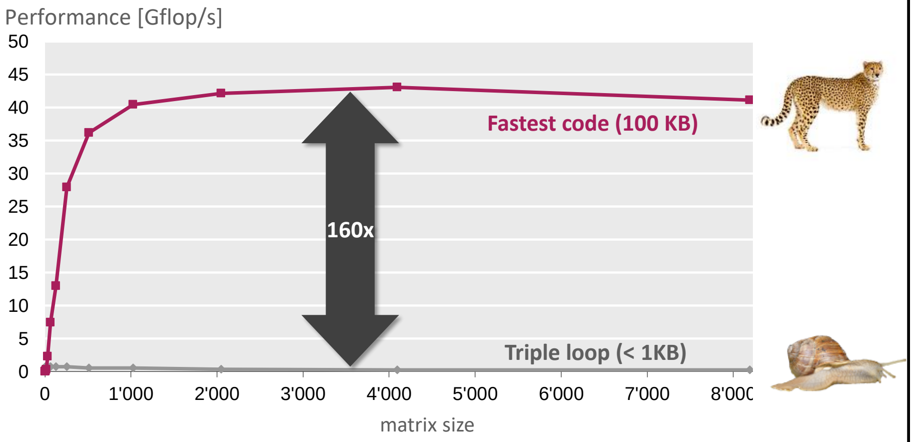


Vendor compiler, best flags

Roughly same operations count

The Problem: Example 2

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



Vendor compiler, best flags

Exact same operations count ($2n^3$)

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

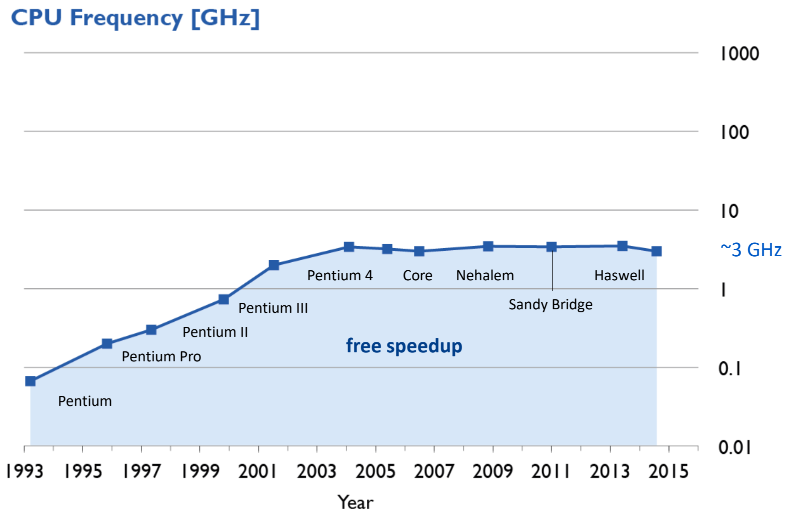
Biometrics identification

“Theorem:”

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

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

Evolutions of Processors (Intel)

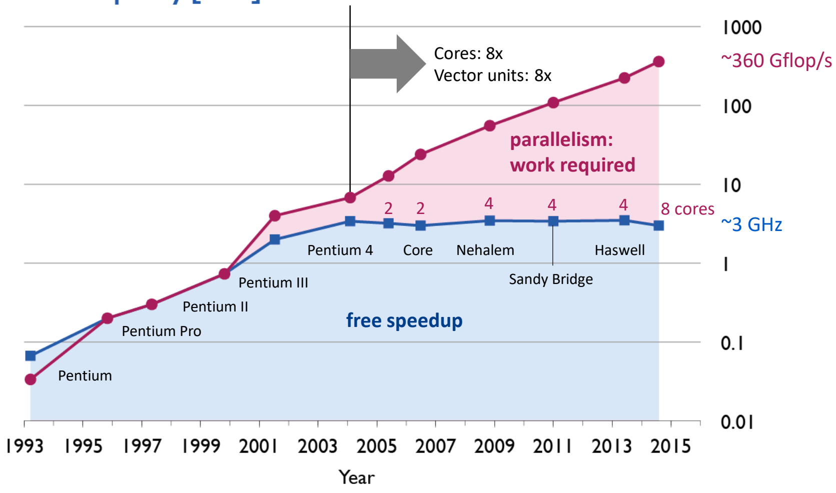


Source: Wikipedia/Intel/PCGuide

Evolutions of Processors (Intel)

Floating point peak performance [Gflop/s]

CPU Frequency [GHz]

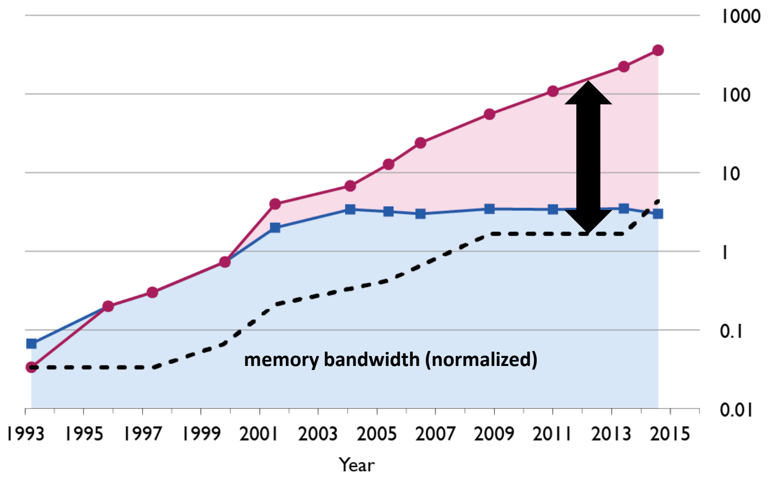


Source: Wikipedia/Intel/PCGuide

Evolutions of Processors (Intel)

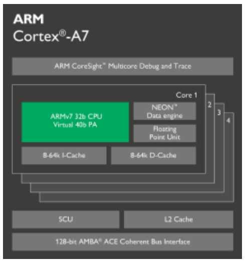
Floating point peak performance [Gflop/s]

CPU Frequency [GHz]



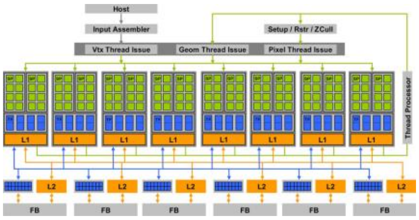
Source: Wikipedia/Intel/PCGuide

And there is Processor Variety …



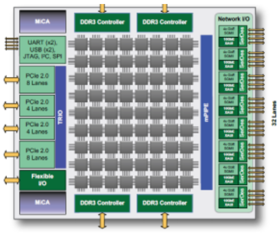
arm.com

Nvidia Tesla



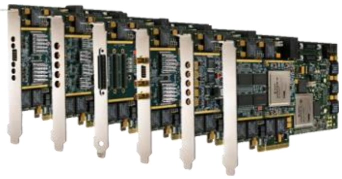
beyond3d.com

Domain-specific (here: Tile)



mellanox.com

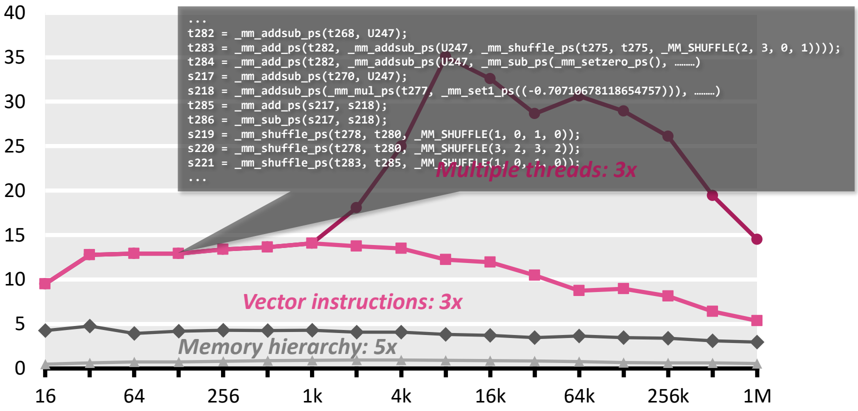
FPGA accelerators



nallatech.com

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

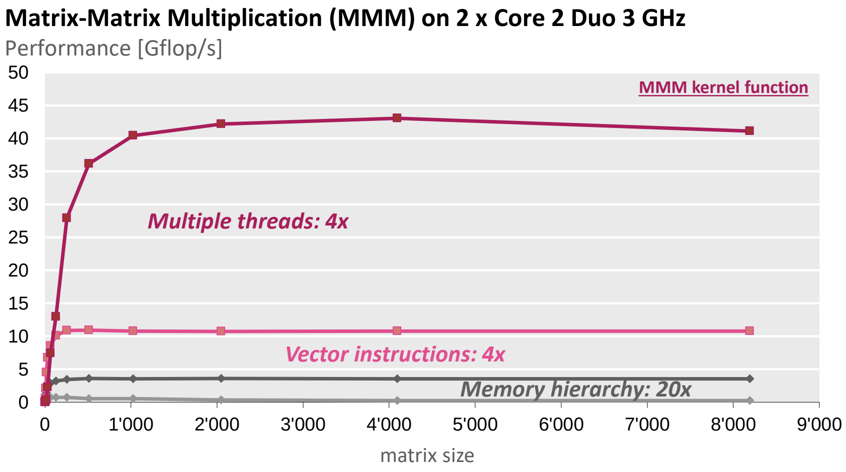
Performance [Gflop/s]



```
...
t282 = _mm_addsub_ps(t268, U247);
t283 = _mm_add_ps(t282, _mm_addsub_ps(U247, _mm_shuffle_ps(t275, t275, _MM_SHUFFLE(2, 3, 0, 1))));
t284 = _mm_add_ps(t282, _mm_addsub_ps(U247, _mm_sub_ps(_mm_setzero_ps(), ………)
s217 = _mm_addsub_ps(t270, U247);
s218 = _mm_addsub_ps(_mm_mul_ps(t277, _mm_set1_ps((-0.707106781186547))), ………)
t285 = _mm_add_ps(s217, s218);
t286 = _mm_sub_ps(s217, s218);
s219 = _mm_shuffle_ps(t278, t280, _MM_SHUFFLE(1, 0, 1, 0));
s220 = _mm_shuffle_ps(t278, t280, _MM_SHUFFLE(3, 2, 3, 2));
s221 = _mm_shuffle_ps(t283, t285, _MM_SHUFFLE(1, 0, 1, 0));
...
```

Compiler doesn't do the job

Doing by hand: *nightmare*



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*

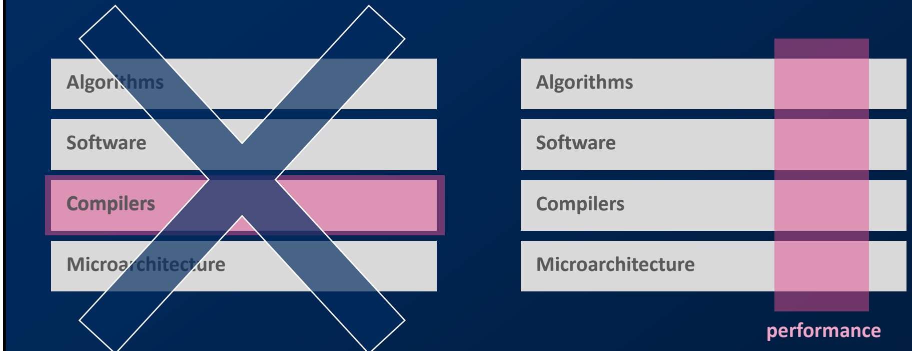
Fast code can be large

- *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*

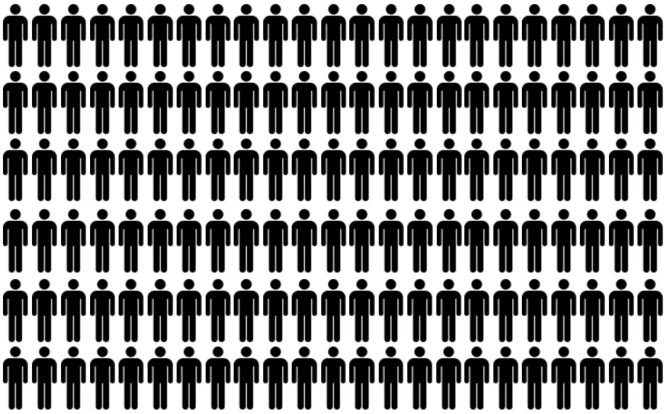
Highest performance is in general non-portable



Performance is different than other software quality features

Performance/Productivity **Challenge**

Current Solution



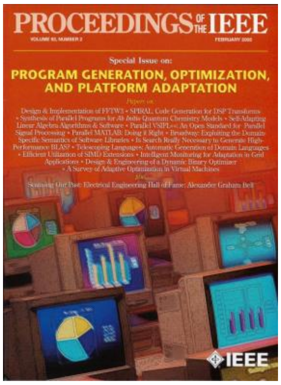
Legions of programmers implement and optimize the *same* functionality for *every* platform and *whenever* a new platform comes out

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*



PROCEEDINGS OF THE IEEE

VOLUME 93, NUMBER 2 FEBRUARY 2005

Special Issue on:

PROGRAM GENERATION, OPTIMIZATION, AND PLATFORM ADAPTATION

Papers on:

Design & Implementation of FFTW3 • SPIRAL: Code Generation for DSP Transforms • Synthesis of Parallel Programs for Ab Initio Quantum Chemistry Models • Self-Adapting Linear Algebra Algorithms & Software • Parallel VSIPL++: An Open Standard for Parallel Signal Processing • Parallel MATLAB: Doing it Right • Broadway: Exploiting the Domain Specific Semantics of Software Libraries • Is Search Really Necessary to Generate High-Performance BLAS? • Telescoping Languages: Automatic Generation of Domain Languages • Efficient Utilization of SIMD Extensions • Intelligent Monitoring for Adaptation in Grid Applications • Design & Engineering of a Dynamic Binary Optimizer • A Survey of Adaptive Optimization in Virtual Machines

Plus:

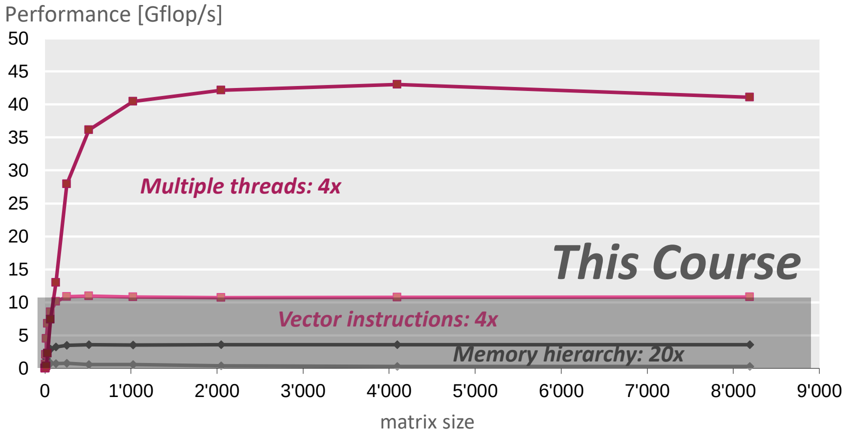
Scanning Our Past: Electrical Engineering Hall of Fame: Alexander Graham Bell

IEEE

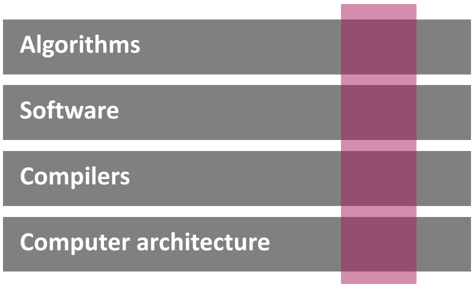
Proceedings of the IEEE special issue, Feb. 2005

Promising area but much more work needed …

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



This Course: Goals



Fast implementations of numerical problems

Algorithms

Software

Compilers

Computer architecture

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

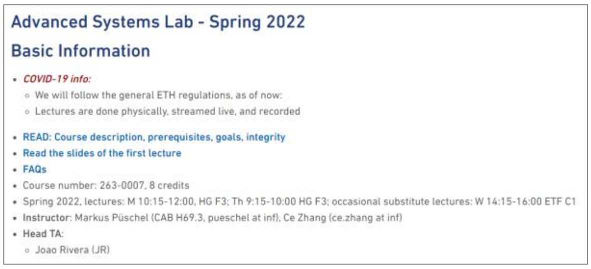
- *Wednesday 14-16,* ETF C1

Course deregistration rule:

- *Deadline: Second Friday in March*
- *After that: drop out = fail*

Course Website Has all Info

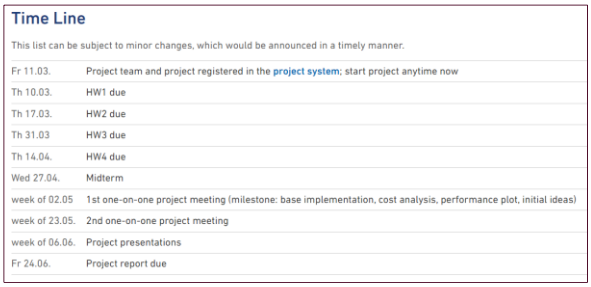
https://acl.inf.ethz.ch/teaching/fastcode/



Advanced Systems Lab - Spring 2022

Basic Information

- *COVID-19 info:*
  - We will follow the general ETH regulations, as of now:
  - Lectures are done physically, streamed live, and recorded
- READ: Course description, prerequisites, goals, integrity
- Read the slides of the first lecture
- FAQs
- Course number: 263-0007, 8 credits
- Spring 2022, lectures: M 10:15-12:00, HG F3; Th 9:15-10:00 HG F3; occasional substitute lectures: W 14:15-16:00 ETF C1
- **Instructor**: Markus Püschel (CAB H69.3, pueschel at inf), Ce Zhang (ce.zhang at inf)
- **Head TA**:
  - Joao Rivera (JR)



Time Line

This list can be subject to minor changes, which would be announced in a timely manner.

Fr 11.03.	Project team and project registered in the project system; start project anytime now
Th 10.03.	HW1 due
Th 17.03.	HW2 due
Th 31.03	HW3 due
Th 14.04.	HW4 due
Wed 27.04.	Midterm
week of 02.05	1st one-on-one project meeting (milestone: base implementation, cost analysis, performance plot, initial ideas)
week of 23.05.	2nd one-on-one project meeting
week of 06.06.	Project presentations
Fr 24.06.	Project report due

Team and Communication

Lecturers: Markus Püschel and Ce Zhang

Head TA: Joao Rivera

Other TAs: Tommaso Pegolotti, Konstantin Taranov, Theodoros Theodoridis

Course website has *ALL* information

Questions:

- *Office hours (during HW period): see website*
- *fastcode@lists.inf.ethz.ch: goes to TAs and lecturers*

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

Prerequisites and Organization

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 we will move lecture there (will communicate if so)*
- *By default will not take place*

Research Project: Overview

Teams of 4

Yes: 4

Topic: Very fast implementation of a numerical problem

Until March 11th:

- *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 in our project system + 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”*
- *“We are a group of three with a project on xxx and are looking for a fourth team member”*

In the beginning all of you are registered to that list

Once team is formed register it in our project system,
tell Joao, and we deregister you

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) or FFT, no stencil computations*
- *Send me a short explanation plus a publication with algorithm for approval*

Exact scope can be adapted during semester

- *reduced to critical component*
- *specialized*

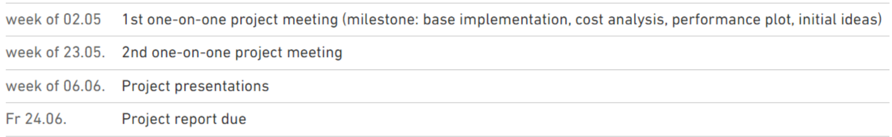
You are in charge of your project!

- *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, check milestones in project system



week of 02.05	1st one-on-one project meeting (milestone: base implementation, cost analysis, performance plot, initial ideas)
week of 23.05.	2nd one-on-one project meeting
week of 06.06.	Project presentations
Fr 24.06.	Project report due

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 desparate(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” …*

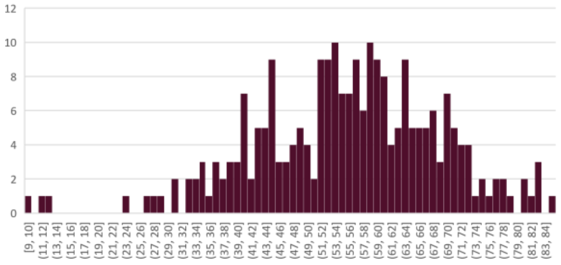
Midterm Exam

Covers first part of course

Date: Wed, April 27th

No substitute date

Point distribution 2020 (max = 100)



There is no final exam

Homework

4 homeworks, beginning of course

Done individually, we use Moodle and Code Expert for some autograding

Exercises on algorithm/performance analysis, check out previous years

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)*

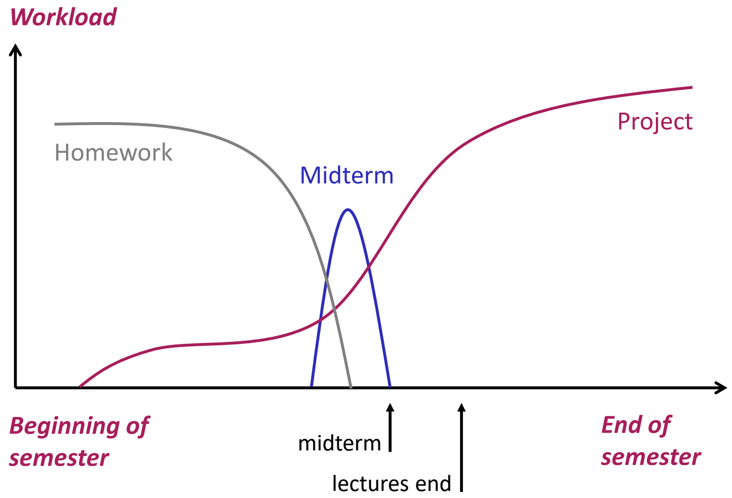
Small part of homework grade for neatness

Late homework policy:

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

Solving homeworks analogous to homeworks in prior years is no 100% guarantee for full points – the material gets updated occasionally

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*

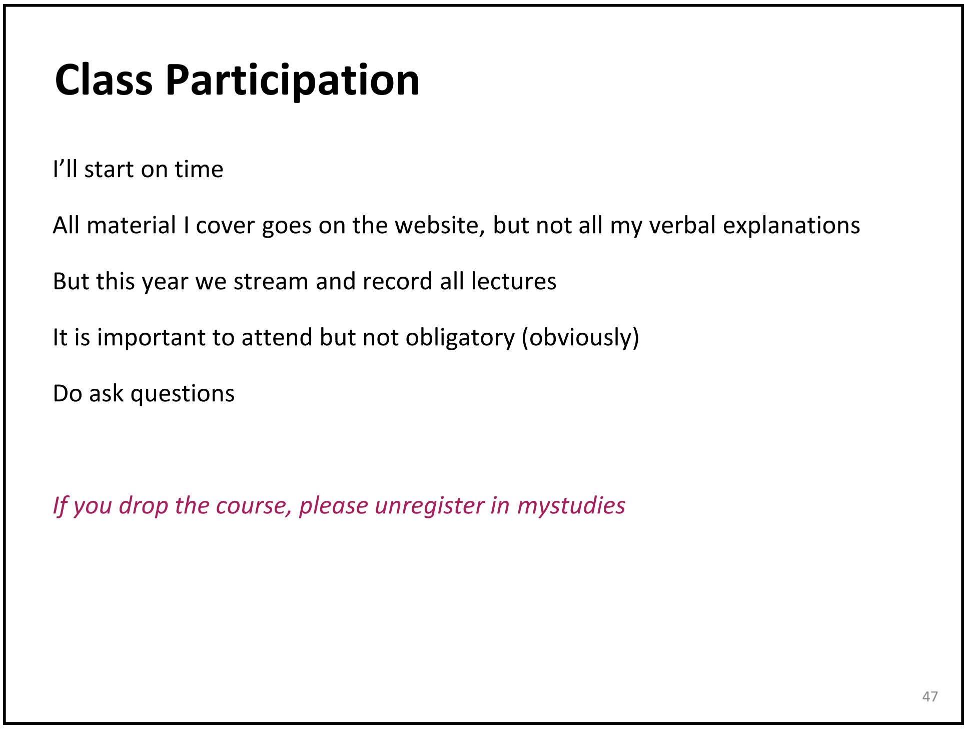
Background Material

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 all my verbal explanations

But this year we stream and record all lectures

It is important to attend but not obligatory (obviously)

Do ask questions

If you drop the course, please unregister in mystudies