How to Write Fast Numerical Code

Spring 2011, Lecture 1

Instructor: *Markus Püschel*

TA: *Georg Ofenbeck*

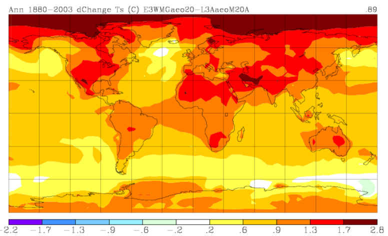
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

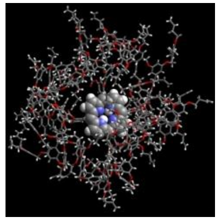
Picture: www.tapety-na-pulpit.org

Today

- Motivation for this course
- Organization of this course

Scientific Computing



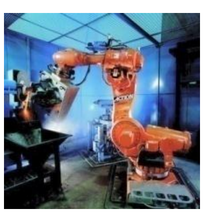


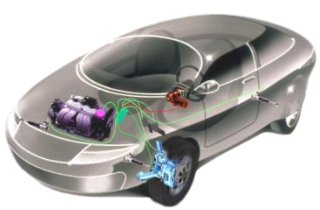
Physics/biology simulations

Consumer Computing

Audio/image/video processing

Embedded Computing





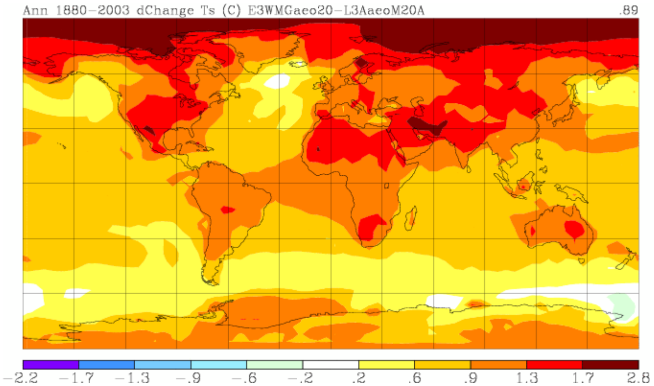
Signal processing, communication, control

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)

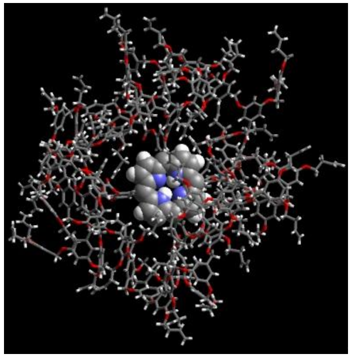
data.giss.nasa.gov



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

Consumer Computing (Desktop, …)

Photo/video processing

Audio coding

Security

Original

JPEG

JPEG2000

Image compression

Methods:

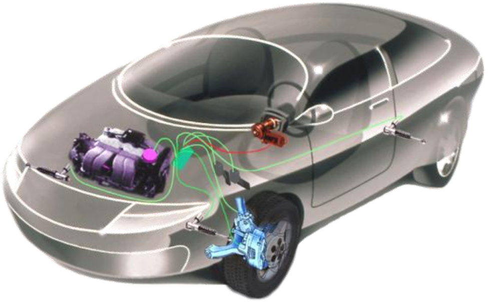
- Linear algebra
- Transforms
- Filters
- Others

Embedded Computing (Low-power processors)

www.dei.unipd.it

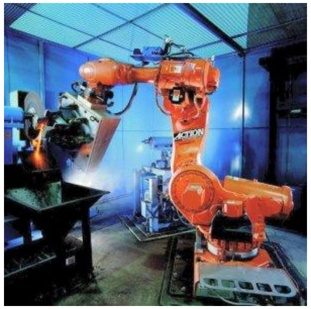
Sensor networks

www.ece.drexel.edu



Cars

www.microway.com.au



Robotics

Computation needed:
- Signal processing
- Control
- Communication

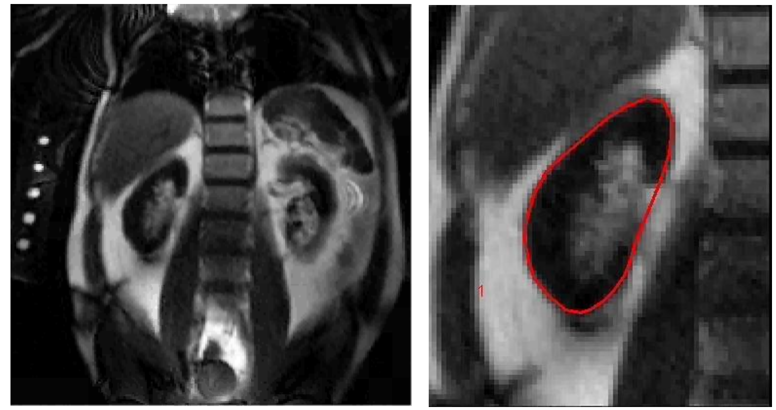
Methods:
- Linear algebra
- Transforms, Filters
- Coding

Research (Examples from Carnegie Mellon)

Bhagavatula/Savvides

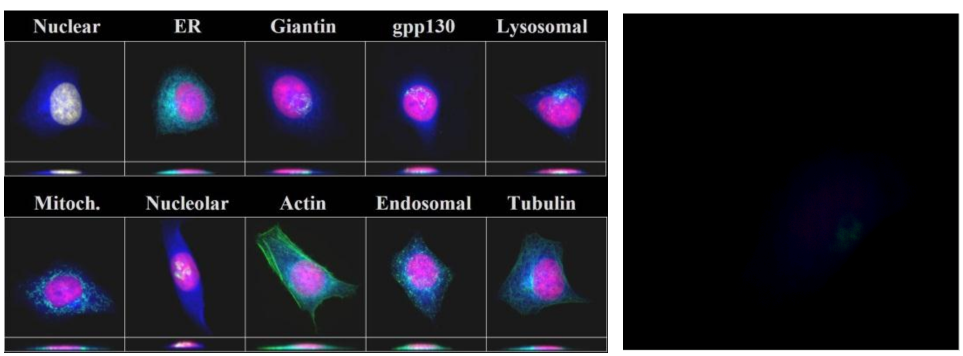
Biometrics

Moura



Medical Imaging

Kovacevic



Bioimaging

Kanade

Computer vision

Classes of Performance-Critical Functions

- **Transforms**
- **Filters/correlation/convolution/stencils/interpolators**
- **Dense linear algebra functions**
- **Sparse linear algebra functions**
- **Coder/decoders**
- ***… 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?

Problem

↓

Algorithm theory

↓

Optimal algorithm

↓

Software developer

↓

Source code

↓

Compiler

↓

Fast executable

"compute Fourier transform"

"fast Fourier transform"
O(nlog(n)) or 4nlog(n) + 3n

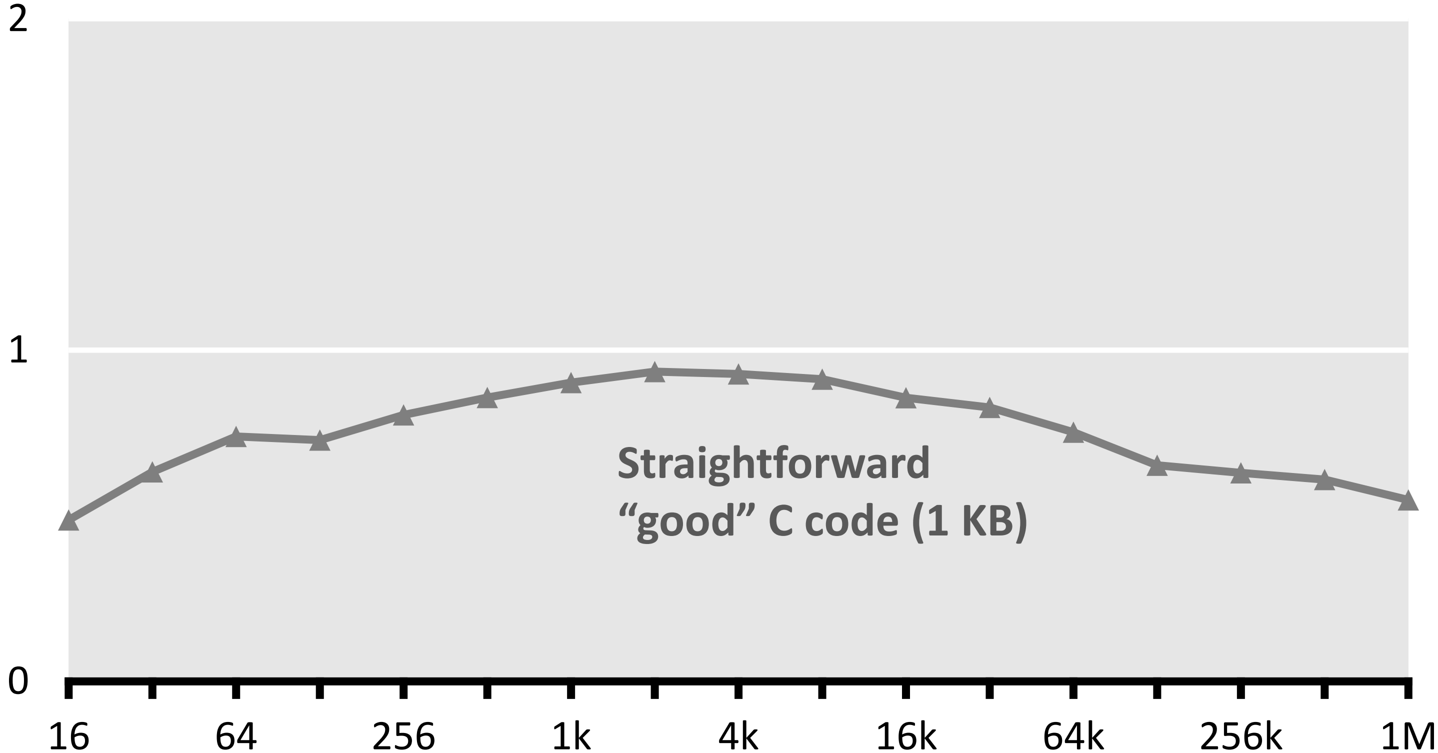
e.g., a C function

How well does this work?

The Problem: Example 1

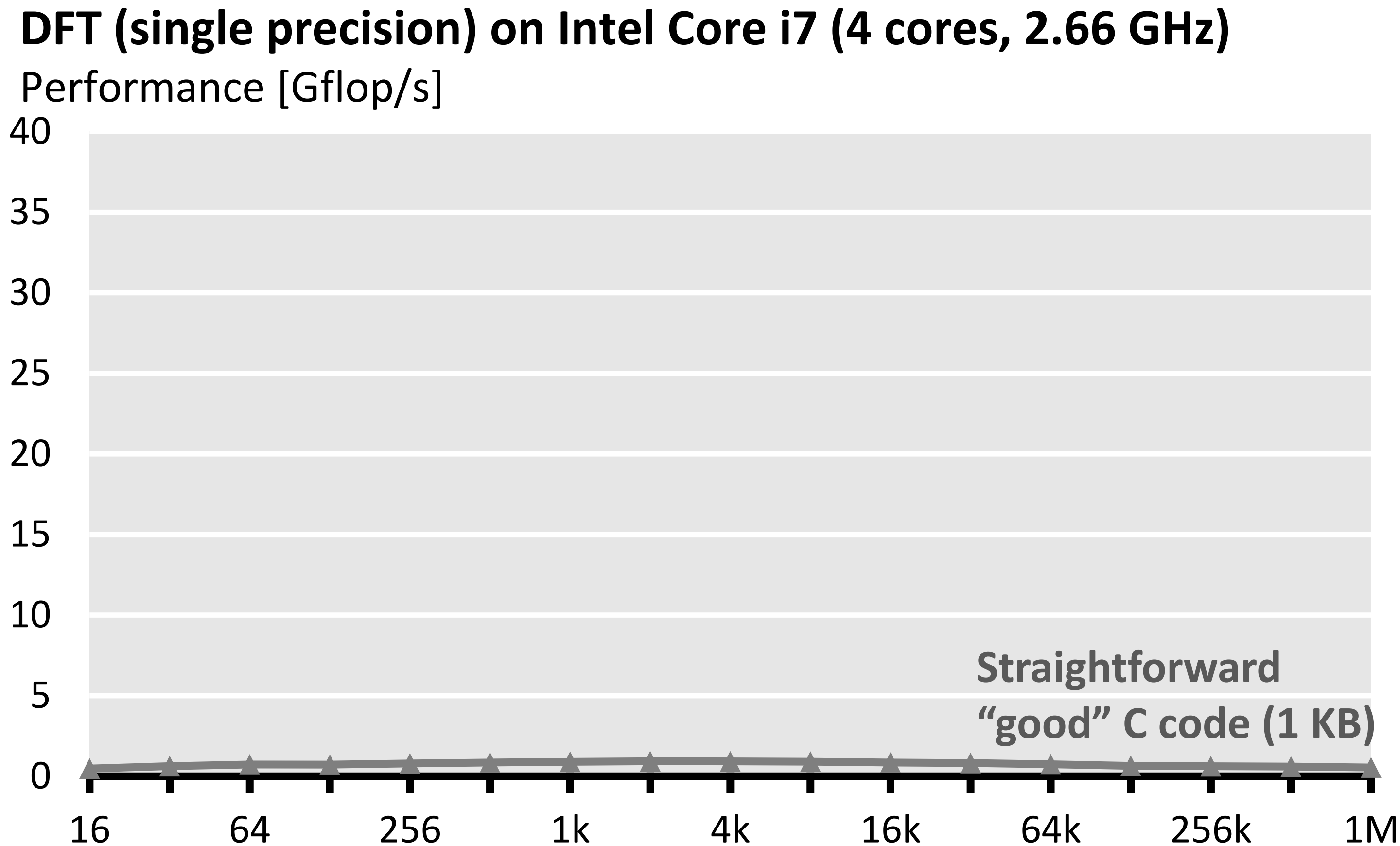
DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)

Performance [Gflop/s]

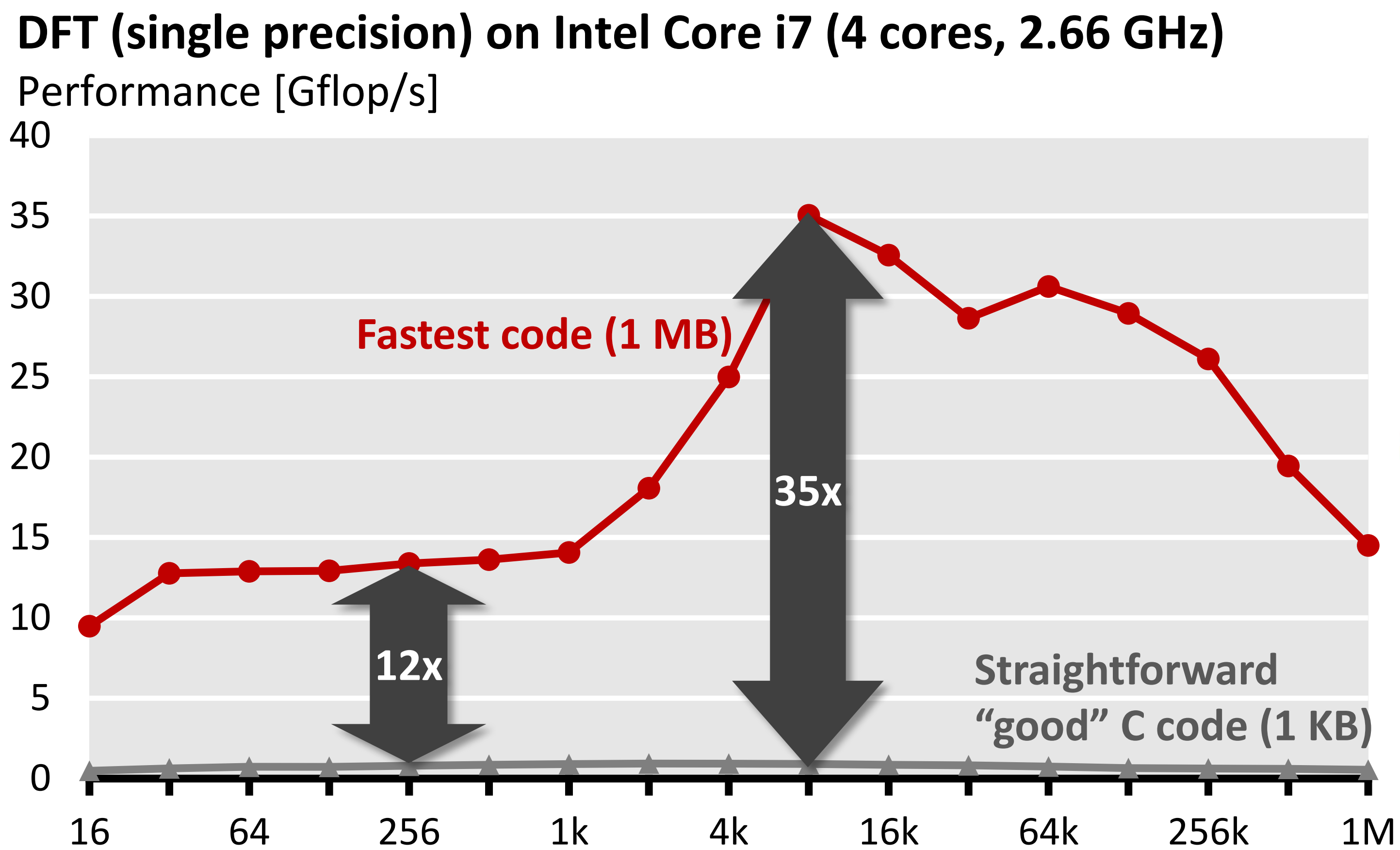


 **or**  **?**

The Problem: Example 1



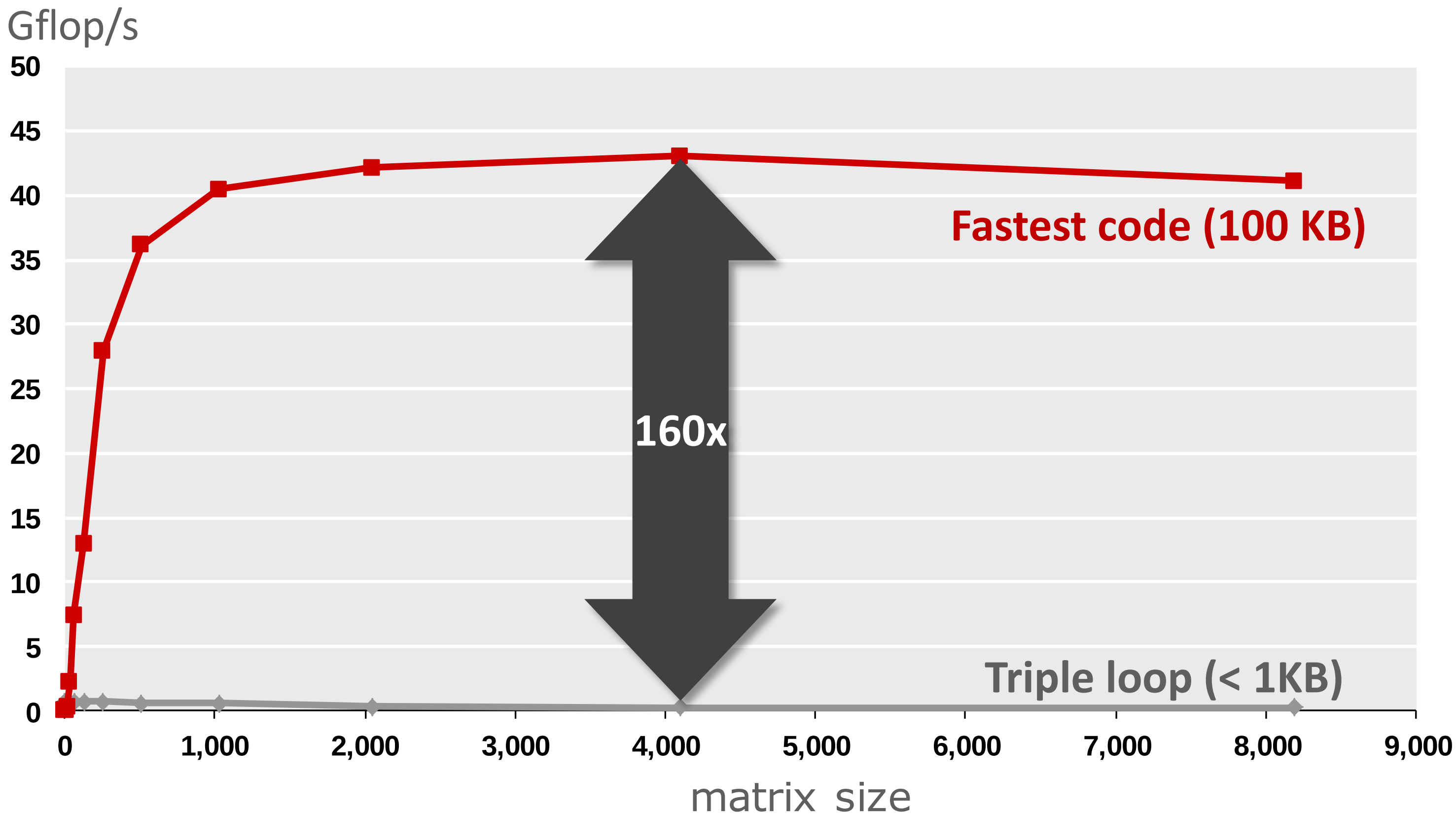
The Problem: Example 1



- Vendor compiler, best flags
- Roughly same operations count

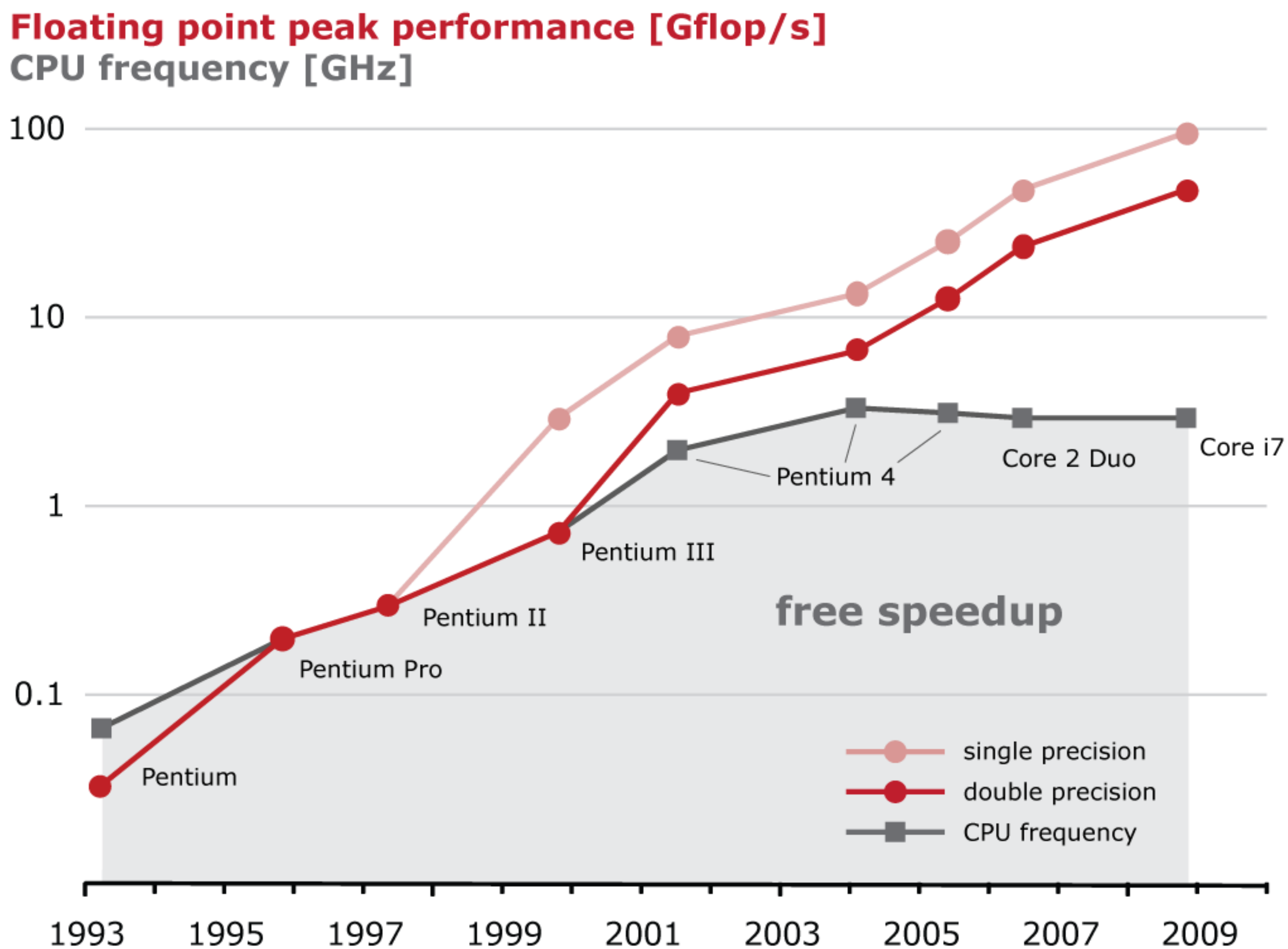
The Problem: Example 2

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

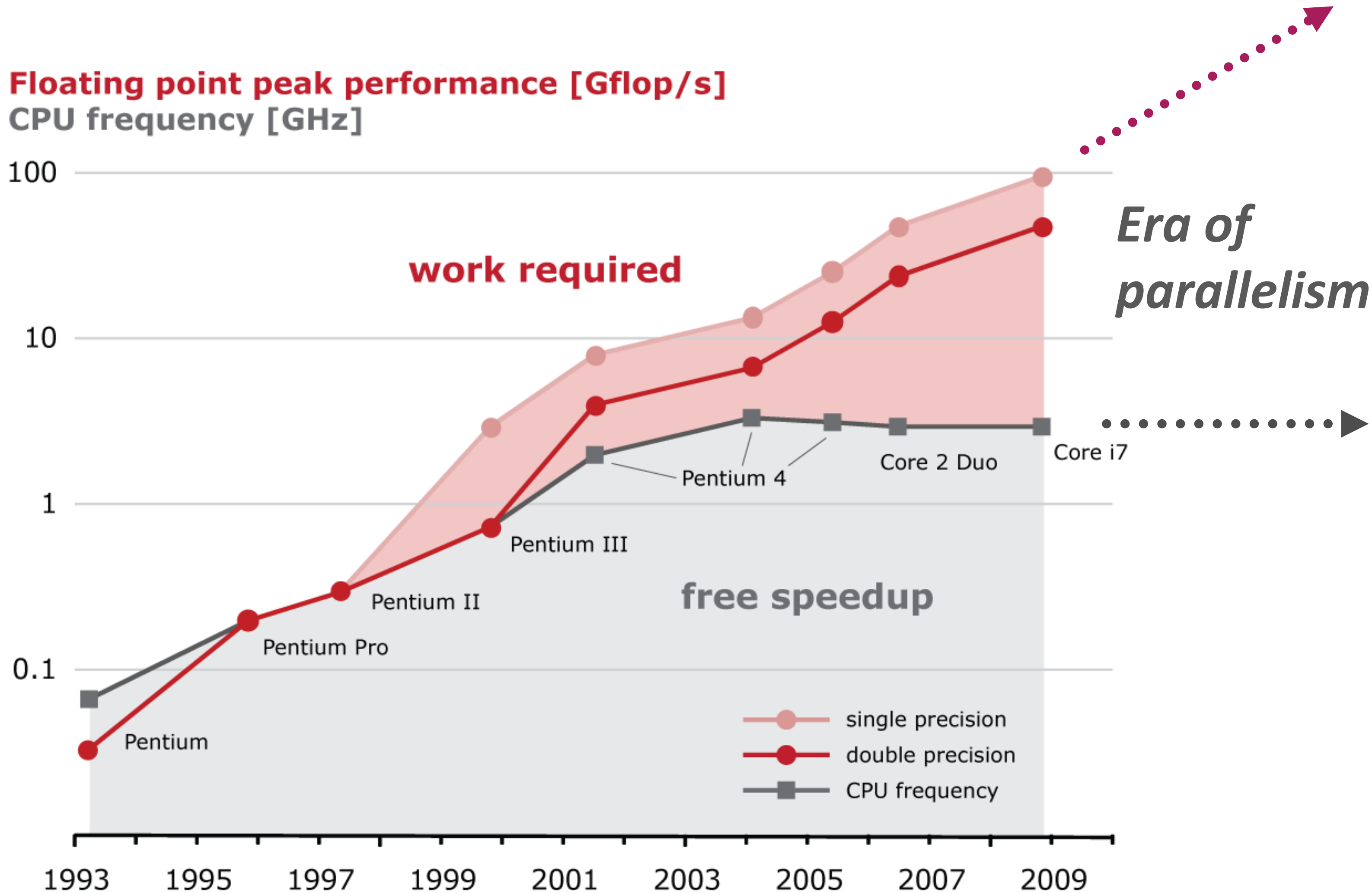


- Vendor compiler, best flags
- Exact same operations count ($2n^3$)
- ***What is going on?***

Evolution of Processors (Intel)

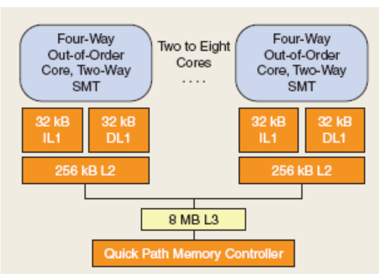


Evolution of Processors (Intel)

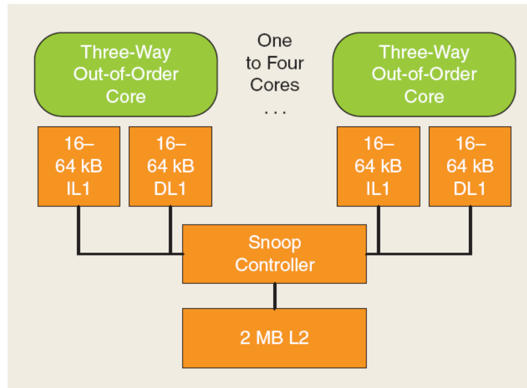


And There Will Be Variety …

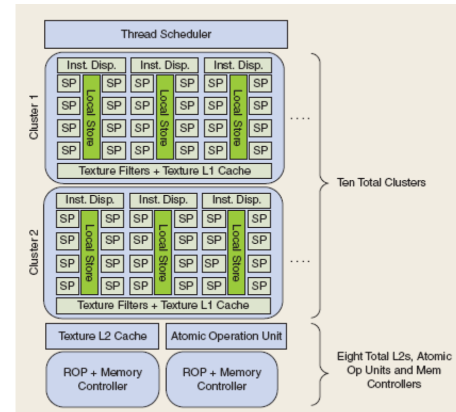
Core i7



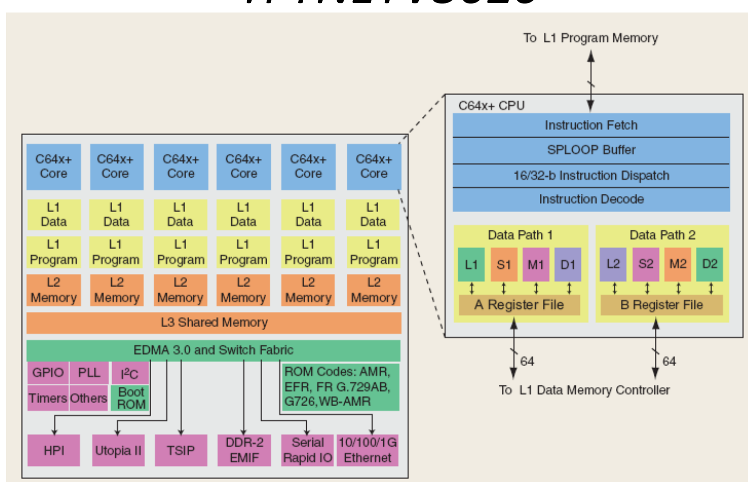
Arm Cortex A9



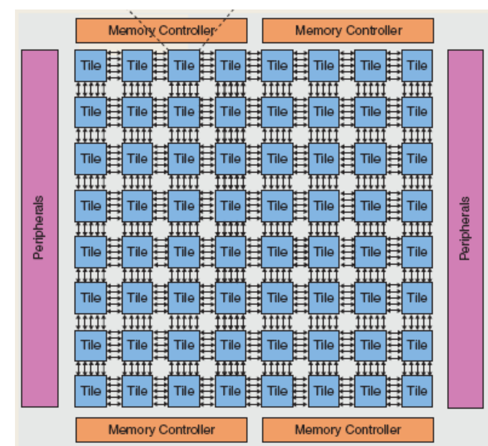
Nvidia G200



TI TNETV3020

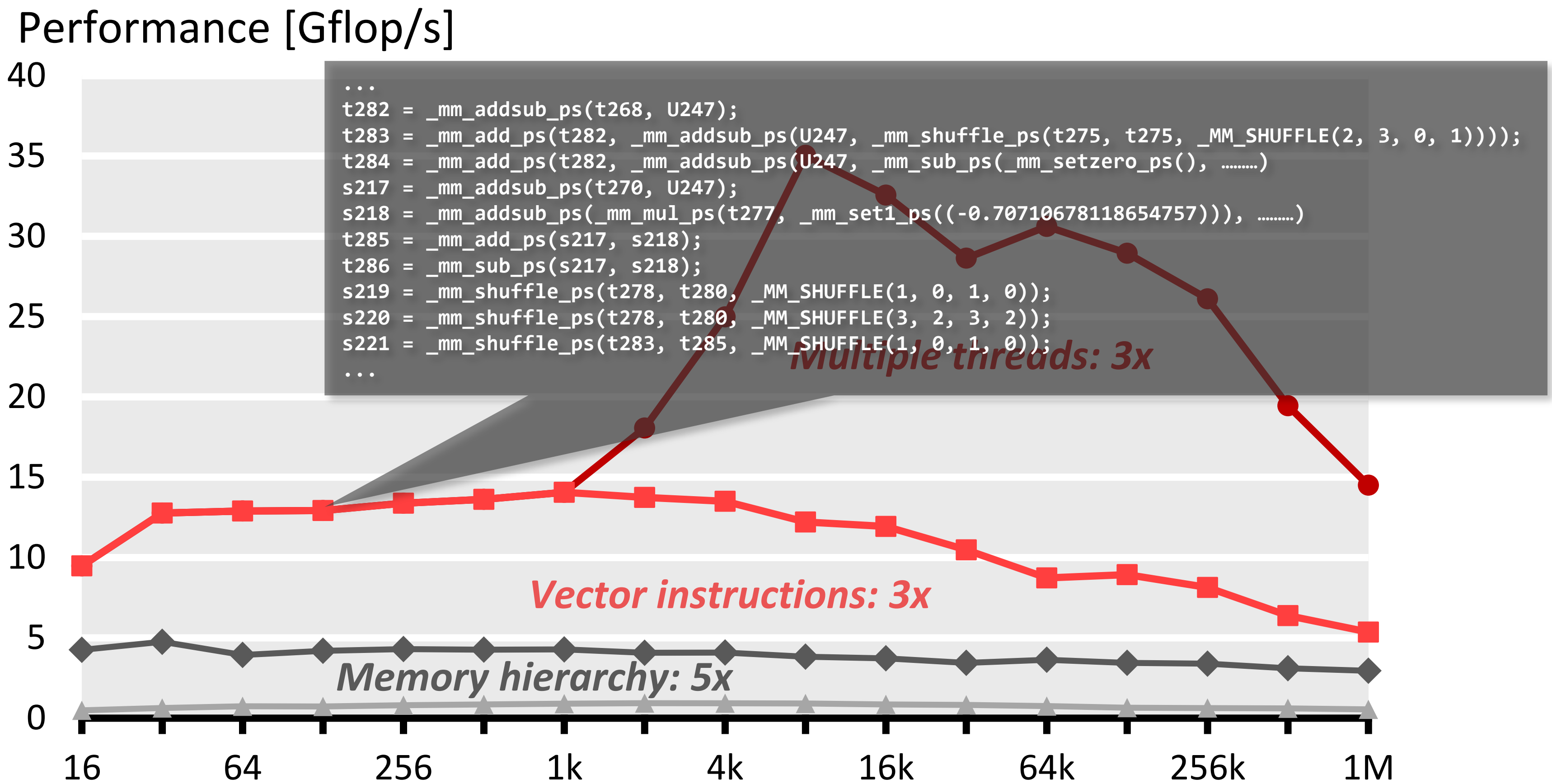


Tilera Tile64



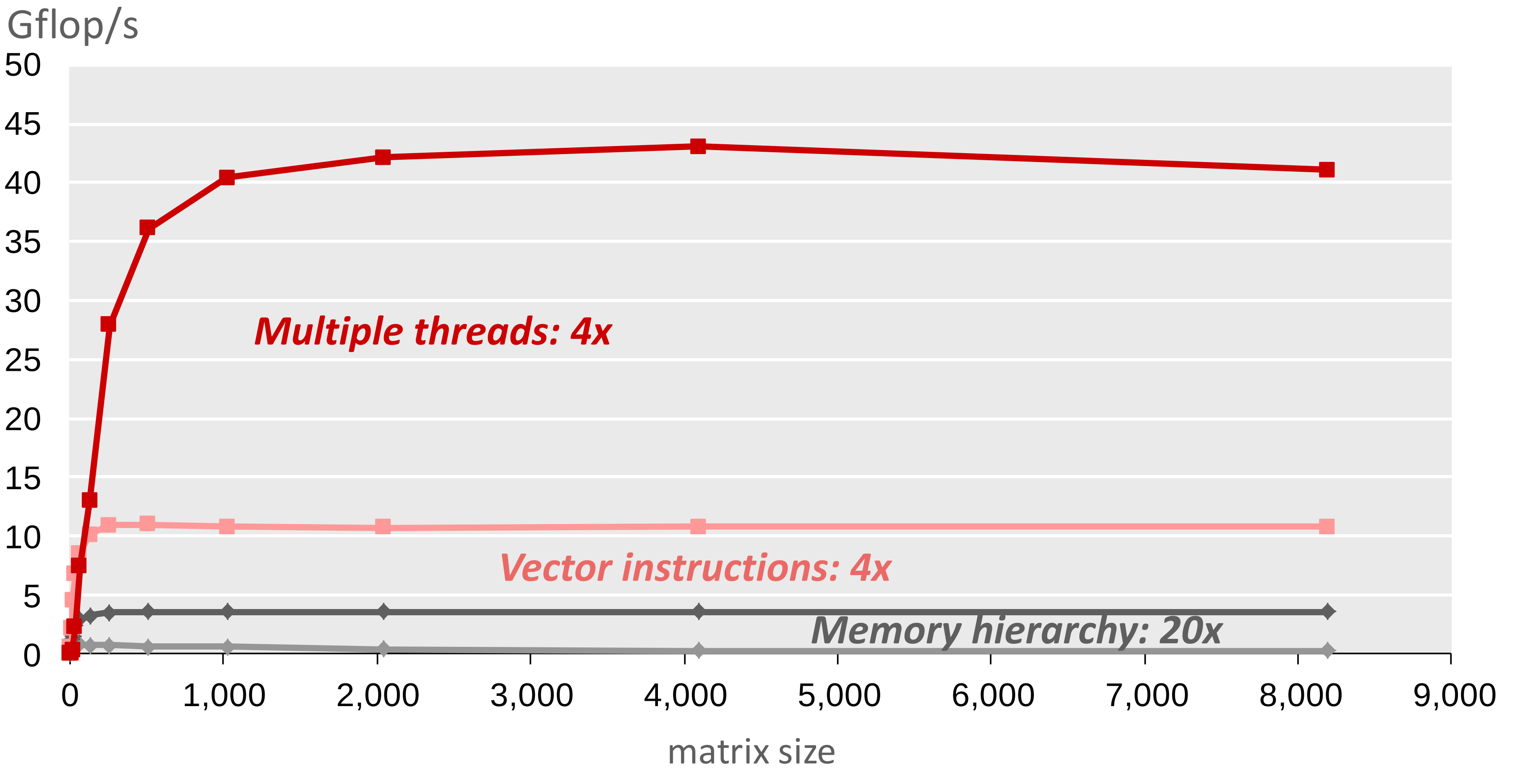
Source: IEEE SP Magazine, Vol. 26, November 2009

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



- Compiler doesn't do the job
- Doing by hand: ***nightmare***

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



- 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

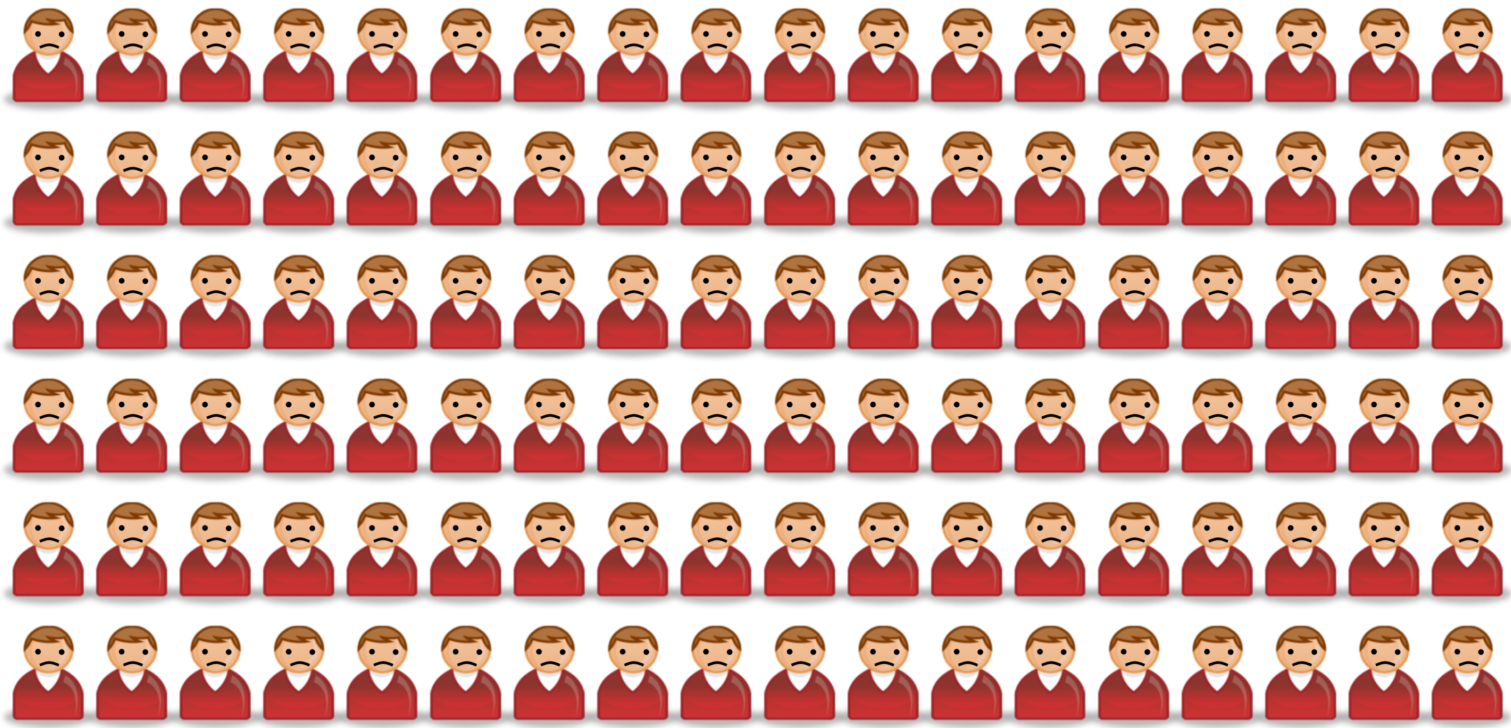
- **Fast code can be large**
  - Can violate “good” software engineering practices

- **Compilers often can’t do the job**
  - Often intricate changes in the algorithm required
  - Parallelization/vectorization still unsolved

- **Highest performance is in general non-portable**

Performance/Productivity **Challenge**

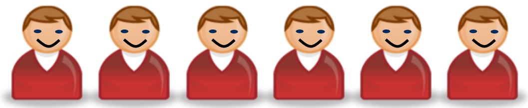
Current Solution



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

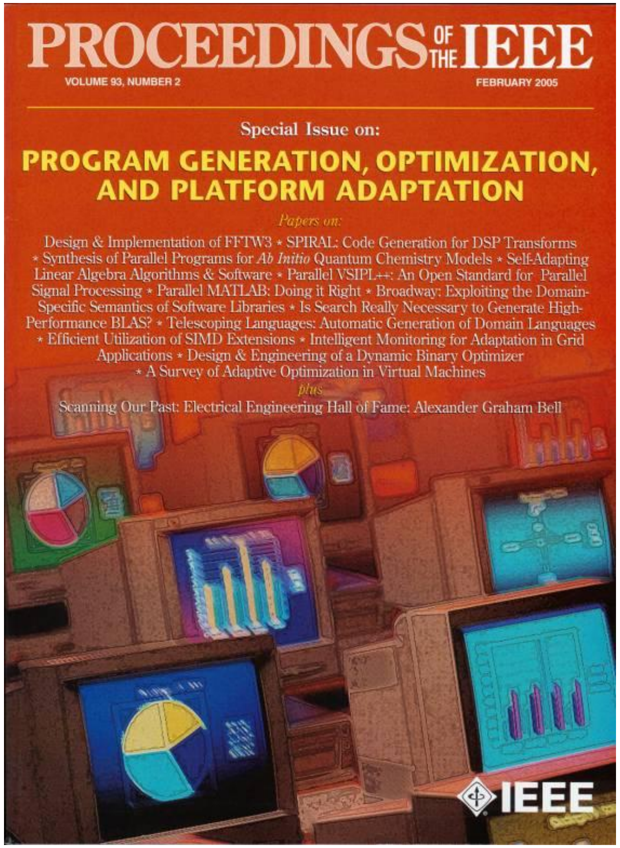
Better Solution: Autotuning

- **Automate (parts of) the implementation or optimization**



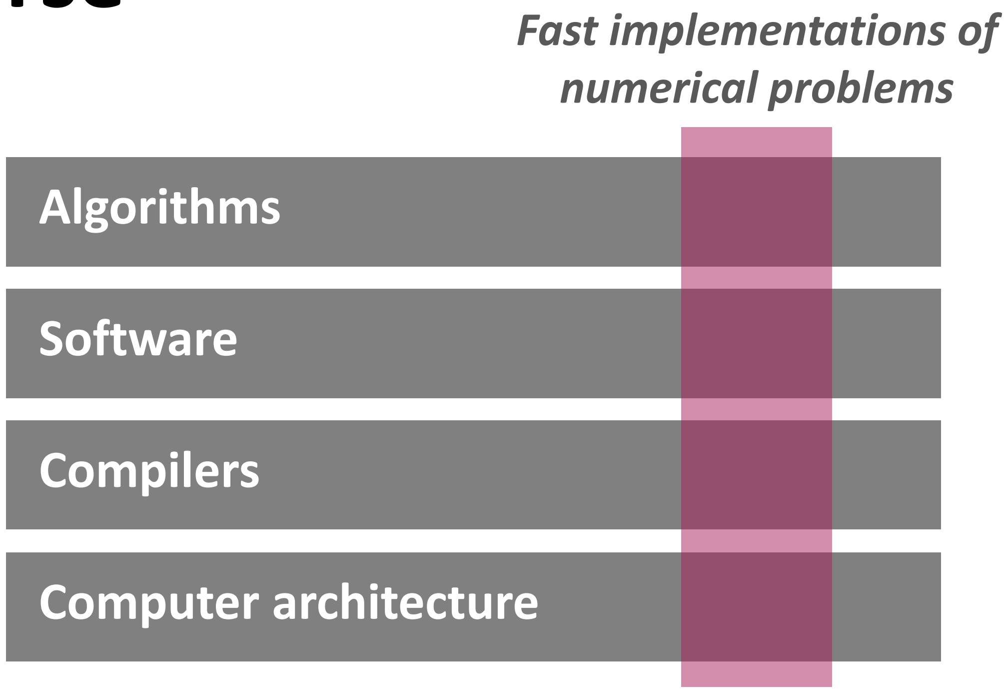
- **Research efforts**
  - Linear algebra: ***Phipac/ATLAS***, LAPACK, ***Sparsity/Bebop/OSKI***, Flame
  - Tensor computations
  - PDE/finite elements: Fenics
  - Adaptive sorting
  - ***Fourier transform: FFTW***
  - ***Linear transforms: Spiral***
  - ...others
  - New compiler techniques

Promising new area but much more work needed …



Proceedings of the IEEE special issue, Feb. 2005

This Course



- **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**

About this Course

- **Team**
  - Me
  - TA: Georg Ofenbeck

- **Office hours: to be determined**
- **Email address for any questions: fastcode@lists.inf.ethz.ch**
- **Course website has *ALL* information**

About this Course (cont'd)

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

- **Grading**
  - 40% research project
  - 20% midterm exam
  - 40% homework

- **Friday slot**
  - Gives you scheduled time to work together
  - Occasionally I will move lecture there

Research Project

- Team up in pairs
- *Topic:* Very fast implementation of a numerical problem
- *Until March 9th:* suggest to me a problem or I give you a problem
  Tip: pick something from your research or that you are interested in
- Show "milestones" during semester
- Write 4 page standard conference paper (template will be provided)
- Give short presentation end of semester
- Submit final code (early semester break)

Midterm Exam

- **Some algorithm analysis**
- **Memory hierarchy**
- **Other**

- ***There is no final exam***

Homework

- **Exercises on algorithm/performance analysis (Math)**
- **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)
  - Some templates will be provided
  - ***Does everybody have access to an Intel processor?***
- **Homework 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
  - You can use at most 2 for a homework

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
  - Ok to discuss things – but then you have to do it alone

- **Code may be checked with tools**

Background Material

- **Course website**

- *Chapter 5 in:*
  Computer Systems: A Programmer's Perspective, 2nd edition
  Randal E. Bryant and David R. O'Hallaron
  (several ones are in the library)
  web: http://csapp.cs.cmu.edu/

- **Prior version of this course:**
  spring 2008 at ECE/CMU

Class Participation

- **I'll start on time**

- **It is important to attend**
  - Many things I'll teach are not in books
  - I'll use part slides part blackboard

- **Ask questions**

- **I will provide some anonymous feedback mechanism (maybe every 3–4 weeks)**