How to Write Fast Code
18-645, spring 2008
1st Lecture, Jan. 14th

Instructor: Markus Püschel
TAs: Srinivas Chellappa (Vas) and Frédéric de Mesmay (Fred)
Today

- Motivation and idea behind this course
- Technicalities
- Motivation: Concrete applications
Motivation and idea behind this course
Scope

- **Numerical computing**: algorithms and implementation that are dominated by additions and multiplications, usually floating point.

- **Three domains of numerical computing:**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Platform</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scientific computing</td>
<td>Large computer clusters</td>
<td>Climate modeling, Physics simulations</td>
</tr>
<tr>
<td>Consumer computing</td>
<td>Standard desktop</td>
<td>Adobe Photoshop, Audio/Video coding</td>
</tr>
<tr>
<td>Embedded computing</td>
<td>Small low-power processor</td>
<td>Signal processing, Control</td>
</tr>
</tbody>
</table>

- Usually there is an **unlimited need for performance** large datasets, realtime.
The Problem

- Standard desktop computer, vendor compiler, using optimization flags
- All implementations have roughly the same operations count (~ $4n\log_2(n)$)
- *Maybe the DFT is just difficult?*

Discrete Fourier Transform (DFT) on 2 x Core 2 Duo 3 GHz (single precision)

Gflop/s

Numerical recipes

Best code
The Problem

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz (double precision)
Gflop/s

- Standard desktop computer, vendor compiler, using optimization flags
- All implementations have exactly the same operations count ($2n^3$)
- What is going on?
Evolution of Processors (Intel)

Floating point peak performance [Mflop/s]
CPU frequency [MHz]

Year

1993 1995 1997 1999 2001 2003 2005 2007

data: www.sandpile.org
Evolution of Processors (Intel)

Floating point peak performance [Mflop/s]
CPU frequency [MHz]

Year
1993 1995 1997 1999 2001 2003 2005 2007

Pentium Pro
Pentium II
Pentium III
Pentium 4
Core 2 Duo
Core 2 Quad

Free speedup

Data: www.sandpile.org
Evolution of Processors (Intel)

High performance software development becomes a nightmare
Evolution of Processors: The Future

Before 2000

- Core2 Duo
- Core2 Extreme
- ClearSpeed
- nVIDIA GPUs

2007

- Sun Niagara 32 threads
- Cell BE 8+1 cores
- IBM Chameleon Cell + FPGA
- Virtex 5 FPGA + 4 CPUs
- SGI RASC Itanium + FPGA

2010 and later

- IBM Cyclops64 80 cores
- ATI/AMD merger CPU+GPU fusion
- Xtreme DATA Opteron + FPGA
- Opteron + FPGA
- Core2 Extreme

A clean slate for concurrent architectures

CPU platforms

Programmability
Discrete Fourier Transform (DFT) on 2 x Core 2 Duo 3 GHz

Gflop/s

input size

Multiple threads: 2x
Vector instructions: 3x
Memory hierarchy: 5x
Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz

Gflop/s

Multiple threads: 4x

Vector instructions: 4x

Memory hierarchy: 20x
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 addition or multiplication
  - Vector instructions can perform 2 or 3 operations in parallel
  - All recent desktop computers have multiple cores = processors on one die

Minimizing operations count does not mean maximizing performance

End of free speed-up: Legacy code will not get automatically faster anymore
  - CPU frequency scaling has hit the power wall
  - Future performance gains through increasing parallelism
  - It is not clear how future platforms will look
Summary and Facts II

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

- Compilers can rarely perform the necessary optimization on numerical code
  - Often intricate changes in the algorithm required
  - Automatic parallelization/vectorization still unsolved

- Highest performance is in general non-portable
  - Best code on one computer may be suboptimal on another
  - Best code is tuned to microarchitecture
  - Often assembly code is hand-written for optimal tuning
Current Practice

- **Legions** of programmers implement and optimize the **same** functionality for **every** platform and **whenever** a new platform comes out
Current Research: Automatic Performance Tuning

- 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

Proceedings of the IEEE special issue, Feb. 2005
Learn how to write **fast code** for numerical problems

- Requires multi-disciplinary knowledge
- Principles studied using important examples
- Applied in homeworks and a semester-long research project

**Fast implementations of numerical problems**

- Algorithms
- Software
- Compilers
- Computer architecture
This Course cont’d

- **Background**
  - Algorithm analysis
  - Compilers
  - Computer architecture

- **Performance optimization**
  - Benchmarking, optimization techniques (memory hierarchy, vector instructions, multithreading)
  - Case studies: important numerical kernels (transforms, linear algebra, filters, convolution, …)
  - Automatic performance tuning (state-of-the-art research)

- **Other knowledge**
  - History, tips for publishing and presenting, …
About this Course

- **Requirements**
  - solid C programming skills
  - matrix algebra
  - senior or above

- **Grading**
  - 40% research project
  - 15% midterm
  - 35% homework
  - 10% class participation

- **No textbook**

- **Office Hours:** yet to be determined

- **Website:** [www.ece.cmu.edu/~pueschel](http://www.ece.cmu.edu/~pueschel) → teaching → 18-645
Research Project

- Team up in pairs
- **Topic:** Very fast implementation of a numerical problem
- **Jan 28th:** suggest to me a problem or I give you a problem
  Tip: pick something from your research (for PhD students)
- Show “milestones” during semester
- Write 4 page standard conference paper (template will be provided)
- Give short presentation end of semester
Midterm

- Mostly about algorithm analysis
- Some multiple-choice

Final Exam

- There is no final exam
Homework

- Exercises on algorithm 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
- Homework scheduled to leave time for research project
Classes/Class Participation

- I’ll start on time, duration ~1:30 (without break)
  - be on time, it’s good style

- 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)
Questions?
Motivation: Concrete Applications
Scientific Computing (Large Clusters)

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

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

Climate modelling
Finance simulations
Molecular dynamics
Consumer Computing (Desktop, ...)

Photo/video processing

Audio coding

Security

Image compression

Methods:
- Linear algebra
- Transforms
- Filters
- Many others
Embedded Computing (Low-power processors)

Sensor networks

Cars

Robotics

Computation needed:
- Signal processing
- Control
- Communication

Methods:
- Linear algebra
- Transforms, Filters
- Coding
Research (Examples at ECE/CMU)

**Biometrics**

Bhagavatula/Savvides

**Medical Imaging**

Moura

**Bioimaging**

Kovacevic

**Computer vision**

Kanade
Summary

- A very large number of diverse applications in engineering, science, consumer market rely on numerical computation
- The computations are diverse but rely on basic mathematical functionality (see 13 dwarfs, Berkeley report on parallel computing landscape)
  - Linear algebra (dense/sparse)
  - Transforms/filters
  - Grid methods
  - Encryption
  - Graph traversals, sorting
  - ...
- Unlimited need for performance
- In this course you learn how to make numerical applications run fast on modern computing platforms (focus desktop)