How to Write Fast Numerical Code

Spring 2012

Lecture 2

Instructor: Markus Püschel

TA: Georg Ofenbeck

ETH

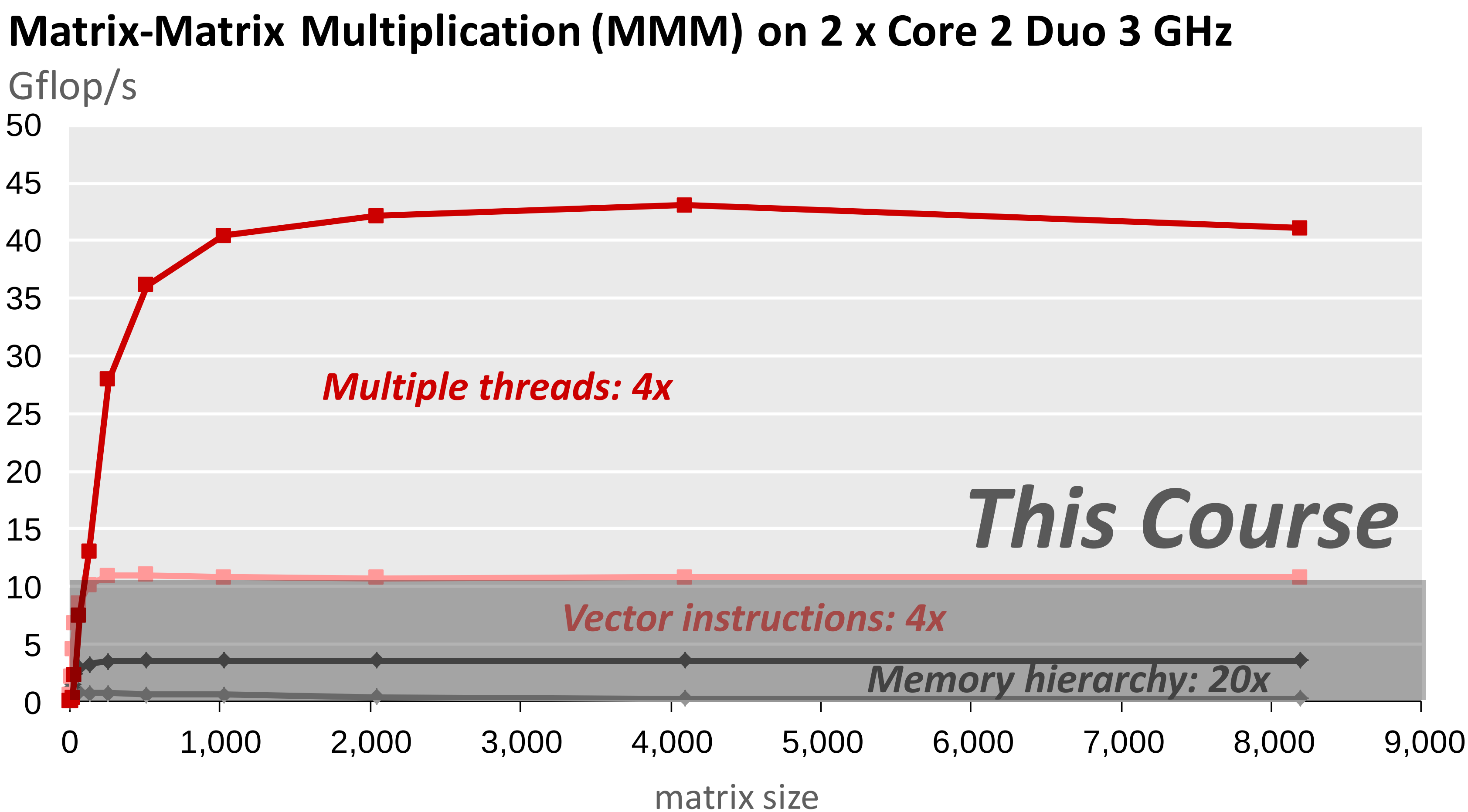
Eidgenössische Technische Hochschule Zürich

Swiss Federal Institute of Technology Zurich

Technicalities

- **Research project: Let me know**
  - if you know with whom you will work
  - if you have already a project idea
  - current status: on the web
  - Deadline: March 7th
- **Email for questions: fastcode@lists.inf.ethz.ch**
  - use for all technical questions
  - received by me and the Tas = ensures timely answer

Last Time


Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz
Gflop/s
50
45
40
35
30
25
20
15
10
5
0
Multiple threads: 4x
This Course
Vector instructions: 4x
Memory hierarchy: 20x
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
8,000
9,000
matrix size

Today

- **Problem and Algorithm**
- **Asymptotic analysis**
- **Cost analysis**

- ***Standard book:*** **Introduction to Algorithms (2nd edition), Corman, Leiserson, Rivest, Stein, McGraw Hill 2001)**

Problem

- ***Problem:*** **Specification of the relationship between a given input and a desired output**
- **Numerical problems (this class): In- and Output are numbers** (or lists, vectors, arrays, … of numbers)
- **Examples**
  - Compute the discrete Fourier transform of a given vector x of length n
  - Matrix-matrix multiplication (MMM)
  - Compress an n x n image with a ratio …
  - Sort a given list of integers
  - Multiply by 5, y = 5x, using only additions and shifts

Algorithm

- ***Algorithm:*** **A precise description of a sequence of steps to solve a given problem.**
- **Numerical algorithms: These steps involve arithmetic computation** (additions, multiplications, …)
- **Examples:**
  - Cooley-Tukey fast Fourier transform
  - A description of MMM by definition
  - JPEG encoding
  - Mergesort
  - y = x<<2 + x

Tips for Presenting and Publishing

- **If your topic is an algorithm, *you must first:***
  - Give a formal problem specification, like:
    ***Given** …..; **We want to compute**……*
    or
    Input: ……; Output: …..
- **Analyze the algorithm, at least asymptotic runtime in O-notation**

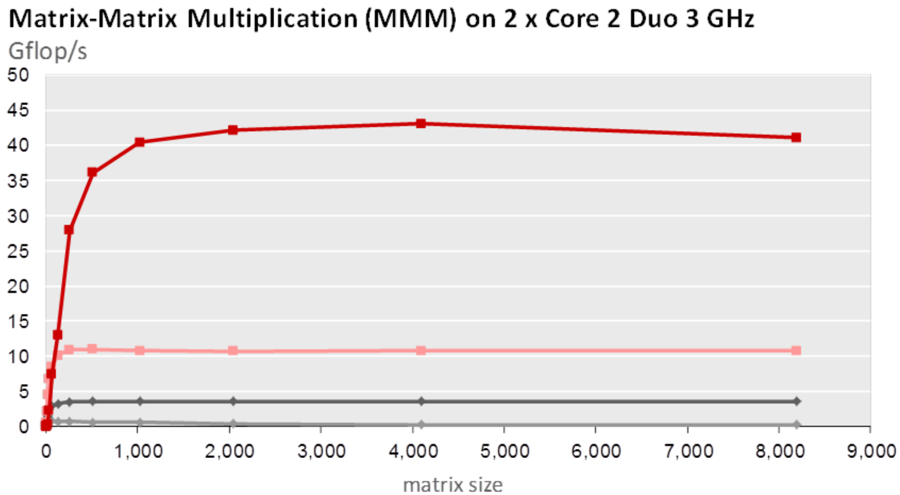
Asymptotic Analysis of Algorithms & Problems

- **Analysis of Algorithms for**
  - Runtime
  - Space = memory requirement (or footprint)
- **Runtime of an algorithm:**
  - Count "elementary" steps
    (for numerical algorithms: usually floating point operations)
    dependent on the input size n (more parameters may be necessary)
  - State result in O-notation
  - Example MMM (square and rectangular): C = A*B + C
- **Runtime complexity of a problem =**
  Minimum of the runtimes of all possible algorithms
  - Result also stated in asymptotic O-notation

Complexity is a property of a problem, not of an algorithm

Valid?

- **Is asymptotic analysis still valid given this?**



- **Memory: yes, if the algorithm is O(f(n)), all memory effects are O(f(n))**
- **Vectorization, parallelization may introduce additional parameters**
  - Vector length ν
  - Number of processors p
  - Example: MMM

Reminder: Do You Know The O?

- $O(f(n))$ is a … ? — set
- How are these related? — $\Theta(f(n) = \Omega(f(n)) \cap O(f(n))$
  - $O(f(n))$
  - $\Theta(f(n))$
  - $\Omega((f(n))$
- $O(2^n) = O(3^n)$? — no
- $O(\log_2(n)) = O(\log_3(n))$ — yes
- $O(n^2 + m) = O(n^2)$? — no

Always Use Canonical Expressions

- **Example:**
  - ***not*** $O(2n + \log(n))$, ***but*** $O(n)$
- **Canonical? If not replace:**
  - $O(100)$ — $O(1)$
  - $O(\log_2(n))$ — $O(\log(n))$
  - $\Theta(n^{1.1} + n \log(n))$ — $O(n^{1.1})$
  - $2n + O(\log(n))$ — yes
  - $O(2n) + \log(n)$ — $O(n)$
  - $\Omega(n \log(m) + m \log(n))$ — yes

Master Theorem: Divide-And Conquer Algorithms

Recurrence

Runtime for problem size *n* — **Cost of conquer step**

$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$$

a* subproblems of size *n/b

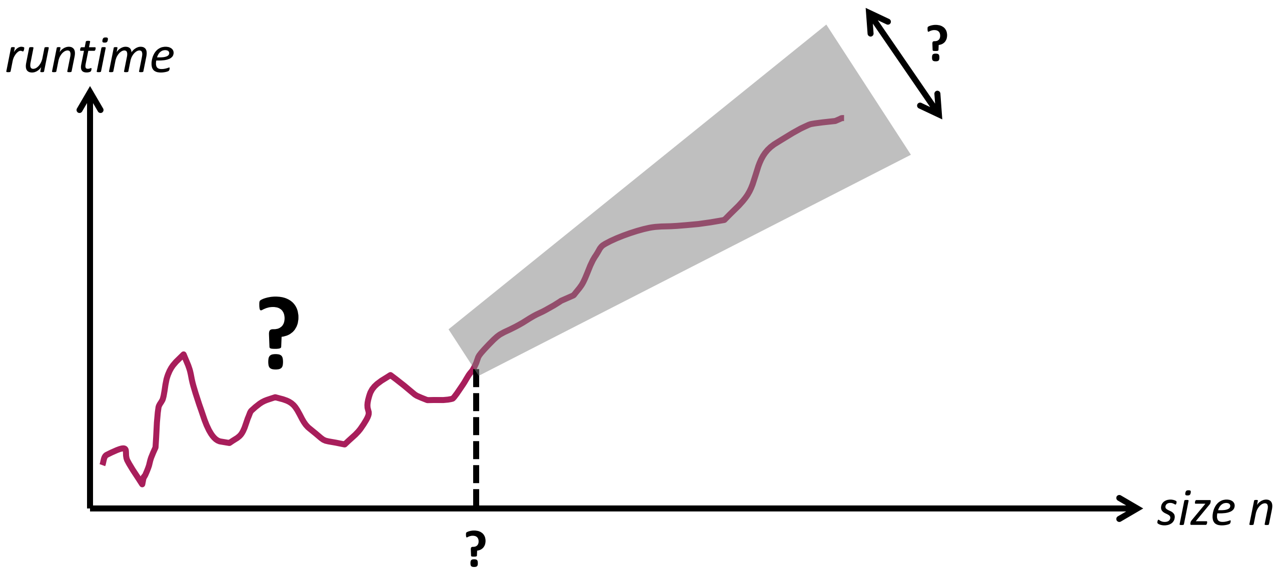
Solution

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & f(n) = O(n^{\log_b a - \epsilon}),\ \text{for some } \epsilon > 0 \\ \Theta(n^{\log_b a} \log(n)), & f(n) = \Theta(n^{\log_b (a)}) \\ \Theta(f(n)), & f(n) = \Omega(n^{\log_b a + \epsilon}),\ \text{for some } \epsilon > 0 \end{cases}$$

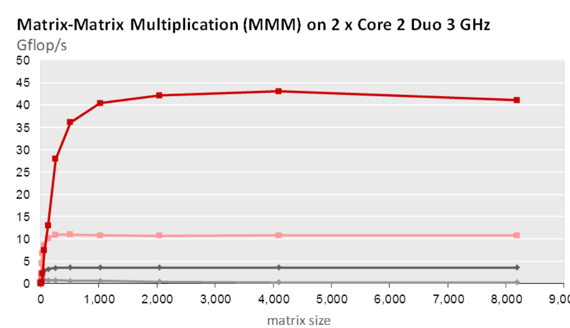
Stays valid if *n/b* is replaced by its floor or ceiling

Asymptotic Analysis: Limitations

- $\Theta(f(n))$ describes only the *eventual shape* of the runtime



- **Constants matter**
  - n^2 is likely better than $1000n^2$
  - $10000000000n$ is likely worse than n^2
- **But remember: even exact op count ≠ runtime**



Refined Analysis for Numerical Problems

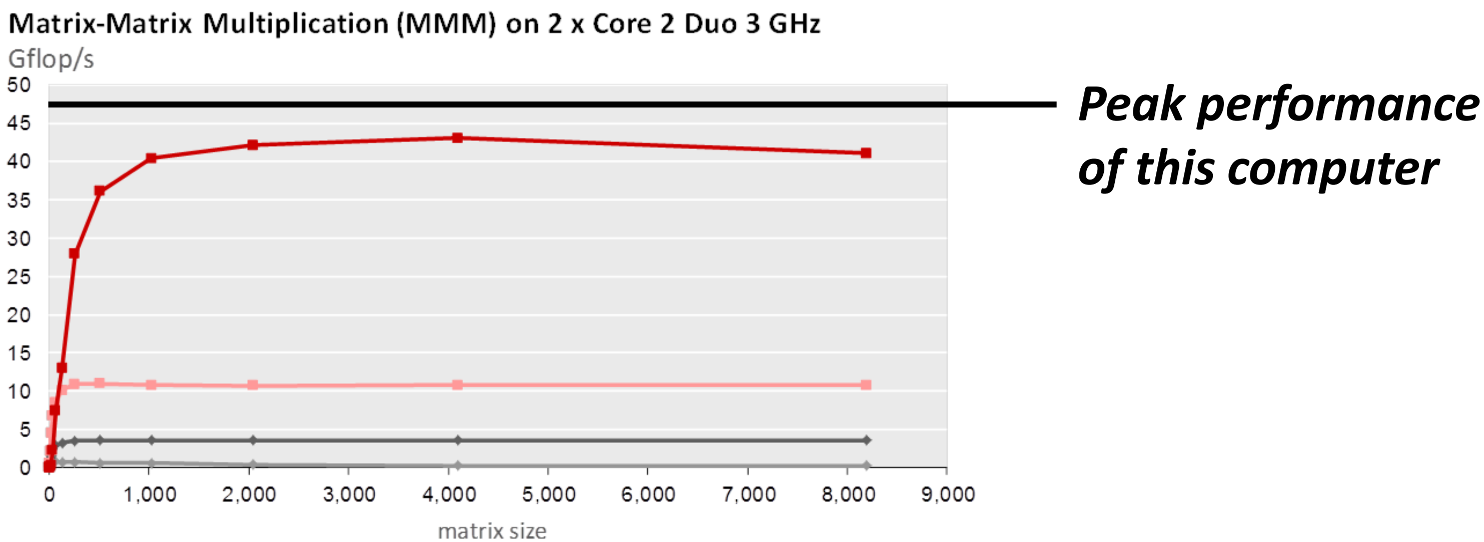
- *Goal:* **determine exact "cost" of an algorithm**
- **Approach (use MMM as running example):**
  - Fix an appropriate cost measure C: "what do I count"
  - Determine cost of algorithm as function C(n) of input size n, or, more generally, of all relevant input parameters:

    $$C(n_1,..,n_k)$$
  - Cost can be multi-dimensional

    $$C(n_1,..,n_k) = (c_1,..,c_m)$$
- **Exact cost is:**
  - More precise than asymptotic runtime
  - Absolutely not the exact runtime
- **Example cost measures:**
  - #floating point operations
  - (#floating point adds, #floating point mults)

Why Cost Analysis?

- **Enables performance analysis**
- **Upper bound through machine's peak performance**



Cost Analysis: How To Do

- **In this class: Cost usually given by floating point ops**
- **Count in algorithm or code**
- **Divide-and-conquer algorithm/code: Solve recurrence**
  - Easy case: formula (blackboard)
  - More involved cases: Graham, Knuth, Patashnik, "Concrete Mathematics," 2nd edition, Addison Wesley 1994
- **If not possible**
  - Instrument code
  - Use performance counters