How to Write Fast Numerical Code

Spring 2014

Lecture: Cost analysis and performance

Instructor: Markus Püschel

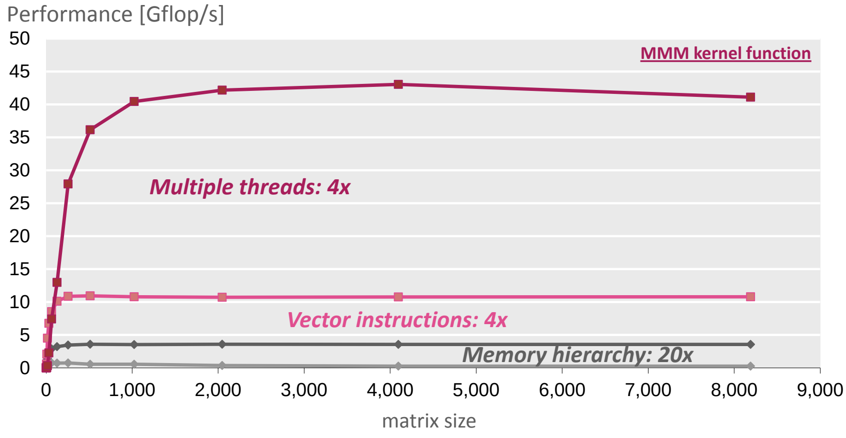
TA: Daniele Spampinato & Alen Stojanov

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Technicalities

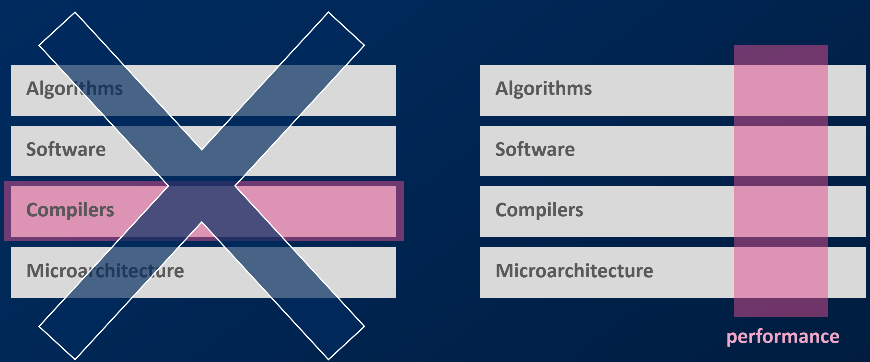
- **Research project: Let us know (fastcode@lists.inf.ethz.ch)**
  - if you know with whom you will work
  - if you have already a project idea
  - current status: on the web
  - Deadline: March 7th
- **If you need partner:** fastcode-forum@lists.inf.ethz.ch
- **If you need partner and project:** fastcode-forum@lists.inf.ethz.ch

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



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

3



Performance is different than other software quality features

4

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 problem*** **(this course): 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 algorithm:*** **Dominated by arithmetic** (adds, mults, …)
- **Examples:**
  - Cooley-Tukey fast Fourier transform (FFT)
  - A description of MMM by definition
  - JPEG encoding
  - Mergesort
  - y = x << 2 + x

Reminder: Do You Know The O?

- O(f(n)) is a … ? set
- How are these related? Θ(f(n) = Ω(f(n)) ∩ O(f(n))
  - O(f(n))
  - Θ(f(n))
  - Ω((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:**
  - $O(100)$ — $O(1)$
  - $O(\log_2(n))$ — $O(\log(n))$
  - $\Theta(n^{1.1} + n \log(n))$ — $\Theta(n^{1.1})$
  - $2n + O(\log(n))$ — yes
  - $O(2n) + \log(n)$ — $O(n)$
  - $\Omega(n \log(m) + m \log(n))$ — yes

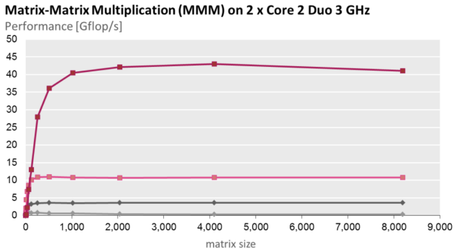
Asymptotic Analysis of Algorithms & Problems

- **Analysis of algorithms for**
  - Runtime
  - Space = memory requirement = memory footprint
- **Asymptotic runtime of an algorithm:**
  - Count "elementary" steps
    numerical algorithms: usually floating point operations
  - 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?

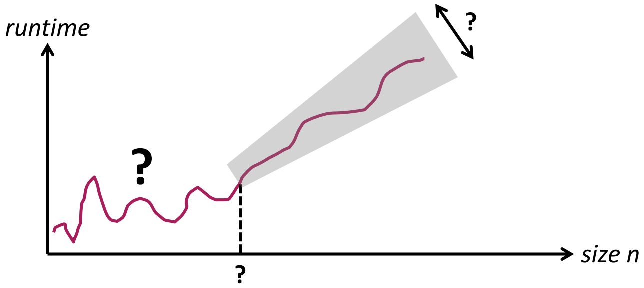
- **Is asymptotic analysis still valid given this?**



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

Asymptotic Analysis: Limitations

- **Θ(f(n)) describes only the *eventual trend* of the runtime**



- **Constants matter**
  - Not clear when "eventual" starts
  - n^2 is likely better than $1000n^2$
  - $10000000000n$ is likely worse than n^2

Cost Analysis for Numerical Problems

- ***Goal:*** **determine exact "cost" of an algorithm**
- **Cost = number of relevant operations**
- **Numerical code (this course):**
  - Number of floating point adds
  - Number of floating point mults
  - *Possibly:* Number of sin, cos, div, sqrt, …

```
/* Multiply n x n matrices a and b  */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            for (k = 0; k < n; k++)
                c[i*n+j] += a[i*n + k]*b[k*n + j];
}
```

Asymptotic runtime: $O(n^3)$

Cost: (fl. adds, fl. mults) = (n^3, n^3)

Cost: flops = $2n^3$

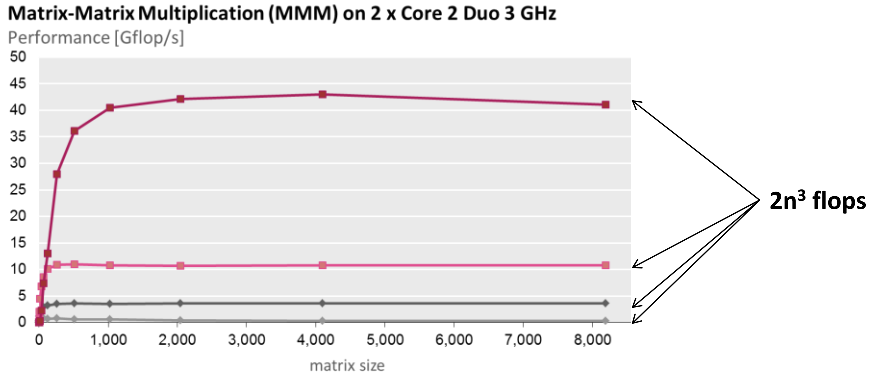
13

Cost Analysis: How To Do

- **Count in algorithm or code**
  - Recursive function: solve recurrence
- **Instrument code**
- **Use performance counters (maybe in a later lecture)**
  - Intel PCM
  - Intel Vtune
  - Perfmon (open source)

14

Remember: Even Exact Cost ≠ Runtime

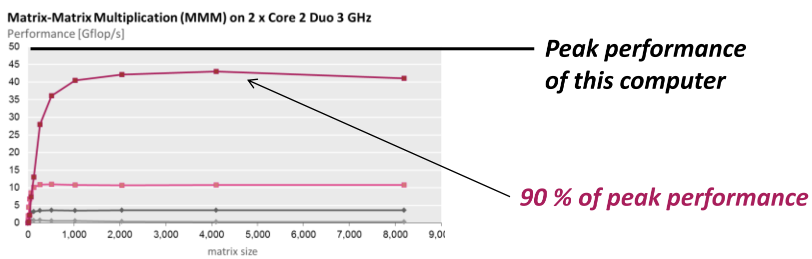


Why Cost Analysis?

- **Enables performance analysis:**

$$\textit{performance} = \frac{\textit{cost}}{\textit{runtime}}$$ *[flops/cycle] or [flops/sec]*

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



Example

```
/* Matrix-vector multiplication y = Ax + y */
void mmm(double *A, double *x, double *y, int n) {
    int i, j, k;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            y[i] += A[i*n + j]*x[j];
}
```

- **Flops? For n = 10?**
  - $2n^2$, 200
- **Performance for n = 10 if runs in 400 cycles**
  - 0.5 flops/cycle
- **Assume peak performance: 2 flops/cycle percentage peak?**
  - 25%

Summary

- **Asymptotic runtime gives only an idea of the runtime *trend***
- **Exact number of operations (cost):**
  - Also no good indicator of runtime
  - But enables performance analysis
- **Always measure performance (if possible)**
  - Gives idea of efficiency
  - Gives percentage of peak