

# Advanced Systems Lab

Spring 2025

*Lecture:* Cost analysis and performance

**Instructor:** Markus Püschel

**TA:** Tommaso Pegolotti, several more

**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

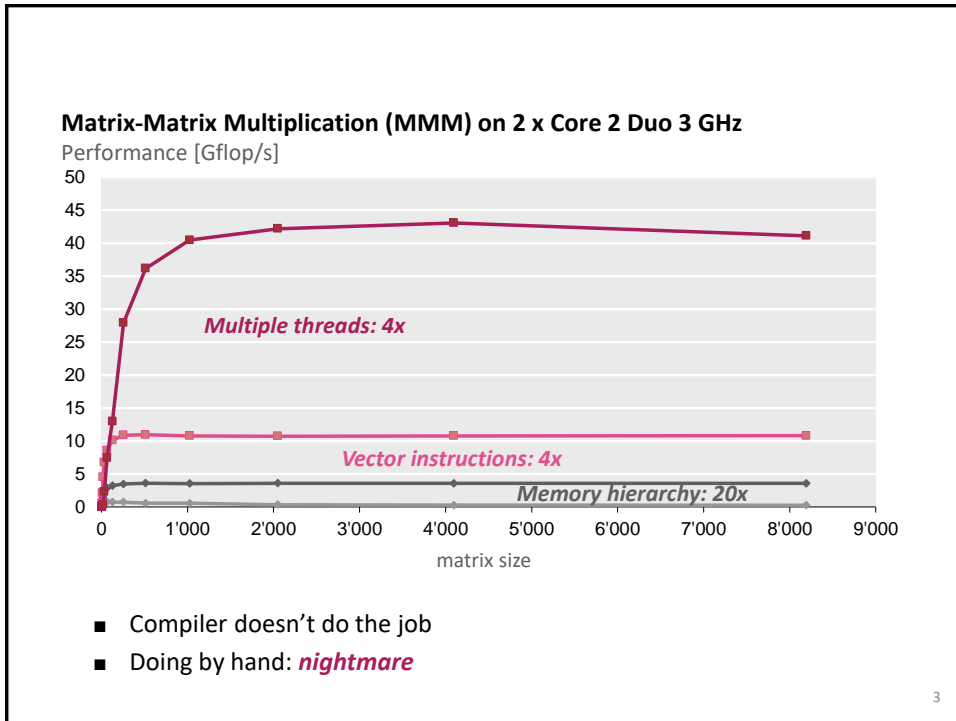
1

## Organization

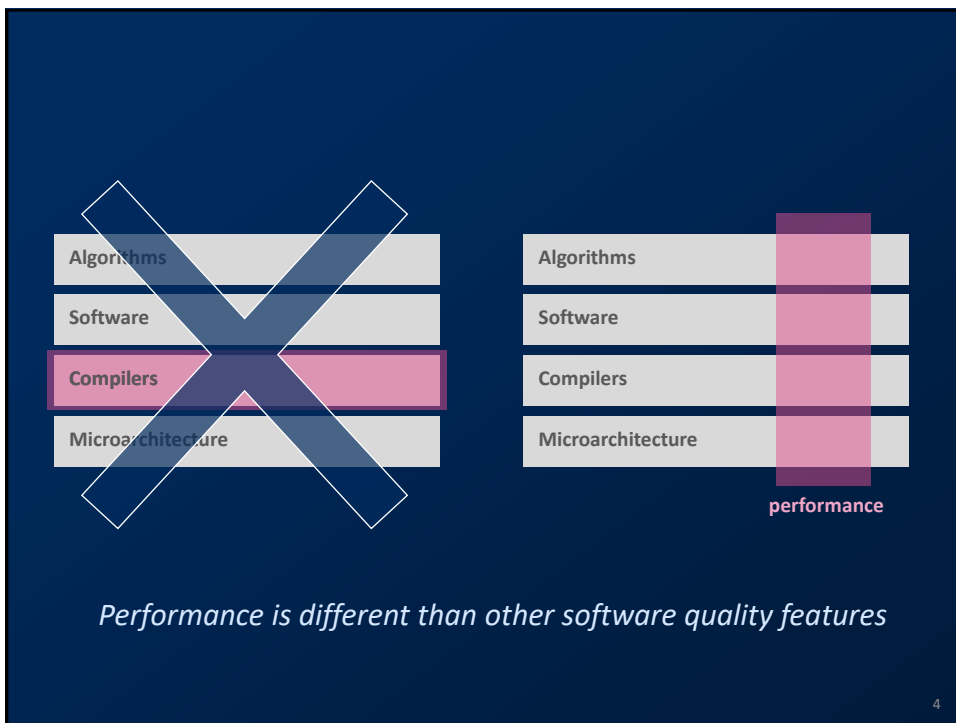
Team and research project: Deadline: *March 7<sup>th</sup>*

If you need team: [fastcode-forum@lists.inf.ethz.ch](mailto:fastcode-forum@lists.inf.ethz.ch)

2



3



4

# Today

Asymptotic analysis

Cost analysis and performance

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

5

5

## Asymptotic Analysis of Algorithms

Analysis for

- *Runtime*
- *Space (= memory footprint)*
- *Data movement (e.g., between cache and memory)*

Example MMM:  $C = A * B + C$ ,      A,B,C are all  $n \times n$

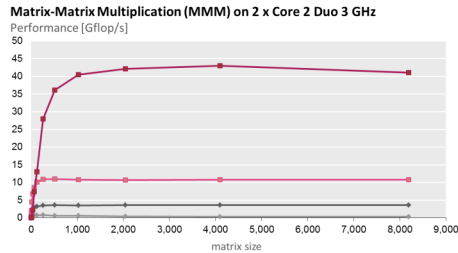
- *Runtime:*     $O(n^3)$
- *Space:*       $O(n^2)$

6

6

## Valid?

Is asymptotic analysis still valid given this?



All algorithms are  $O(n^3)$  when counting flops.

What happens to asymptotics if I take memory accesses into account?

No problem:  $O(f(n))$  flops means at most  $O(f(n))$  memory accesses

What happens if I take vectorization/parallelization into account?

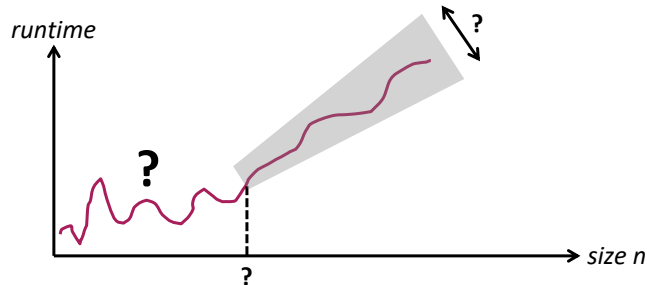
More parameters needed: E.g.,  $O(n^3/p)$  on  $p$  processors

7

7

## Asymptotic Analysis: Limitations

$\Theta(f(n))$  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  $n^2$

8

8

# Cost Analysis for Numerical Problems

*Goal:* determine exact “cost” of an algorithm

Cost = number of relevant operations

Formally: define *cost measure*  $C(n)$ . Examples:

- Counting adds and mults separately:  $C(n) = (adds(n), mults(n))$
- Counting adds, mults, divs separately:  $C(n) = (adds(n), mults(n), divs(n))$
- Counting all flops together:  $C(n) = flops(n)$

This course: focus on floating point operations

The cost measure usually counts *only the operations that constitute the mathematical algorithm* (e.g., as written on paper) and not operations that arise due to its mapping on a computer (e.g., index computations, data movement).

*Example:* next slide

9

9

## Example

```
/* 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 measure?

- $C(n) = (fladds(n), flmults(n)) = (n^3, n^3)$
- $C(n) = flops(n) = 2n^3$

10

10

# Cost Analysis: How To Do

Define suitable cost measure

Count in algorithm or code

- *Recursive function: solve recurrence*

Instrument code

Use performance counters

- [Intel PCM](#)
- [Intel Vtune](#)
- [Perfmon \(open source\)](#)
- *Counters for floating point may not always be available*

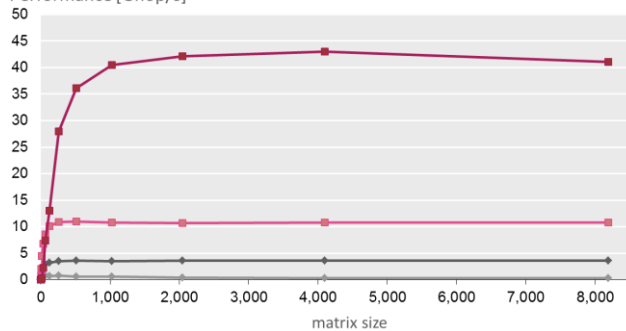
11

11

# Remember: Even Exact Cost $\neq$ Runtime

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

Performance [Gflop/s]



$2n^3$  flops

12

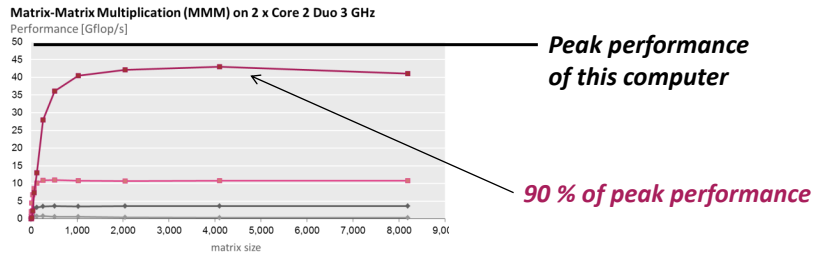
12

# Why Cost Analysis?

Enables performance analysis:

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

Upper bound through machine's peak performance



13

13

# 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%

14

14

## 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*
- *Upper bound on performance through computer's peak performance = lower bound on achievable runtime*

Always measure performance (if possible)

- *Gives idea of efficiency*
- *Gives percentage of peak*

In the project, you should create performance plots whenever possible

15

15