FAST CODE

enev.martin @ gmail

Slides source: prof. Markus Pueschel, ETH Zurich

Time is of the essence...

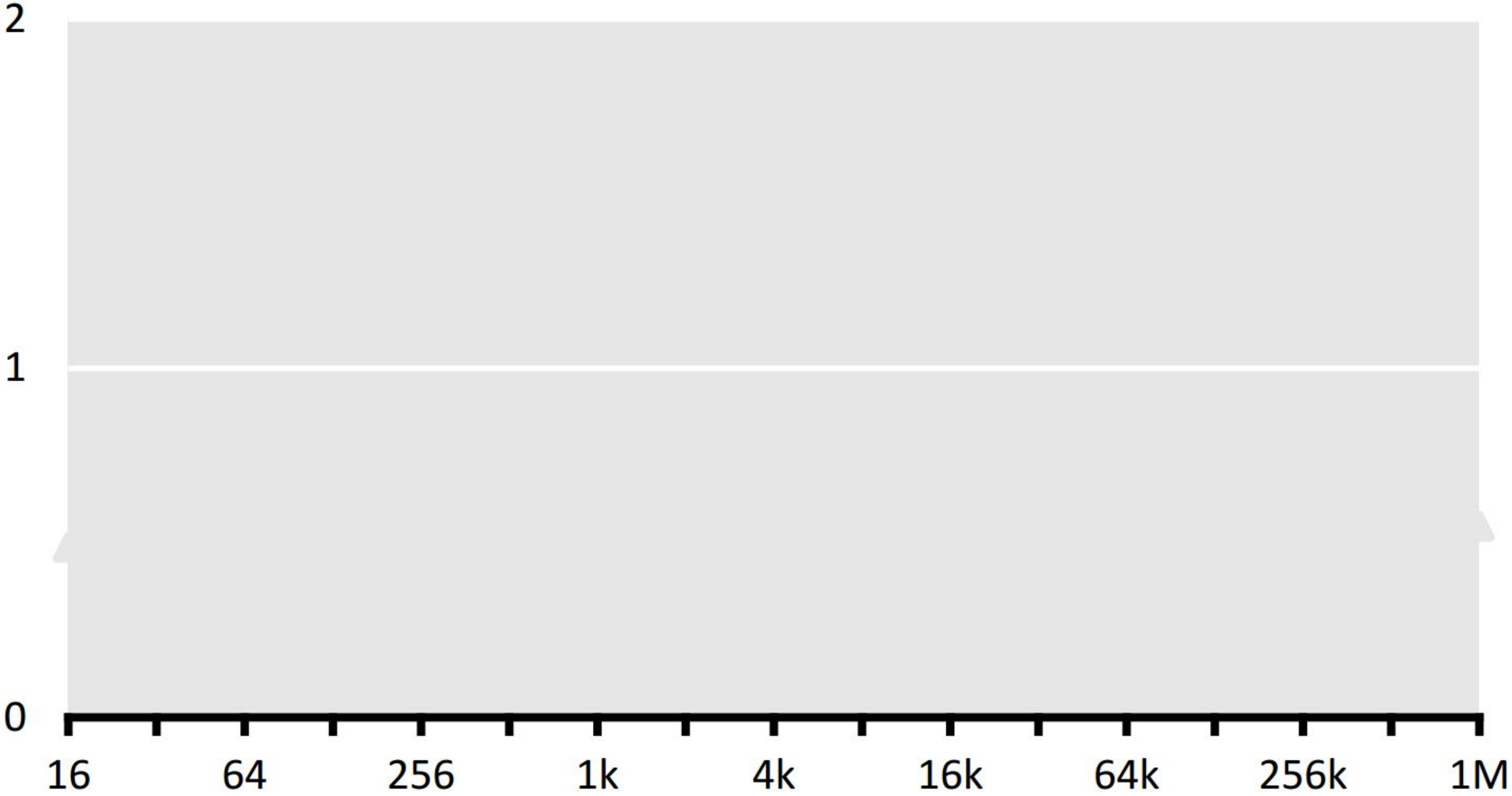
- Scientific computations
- User facing applications
- Embedded systems

How hard is it to get fast code?

- Problem
- Algorithm theory
- Optimal algorithm
- Software developer
- Source code
- Compiler
- Fast executable

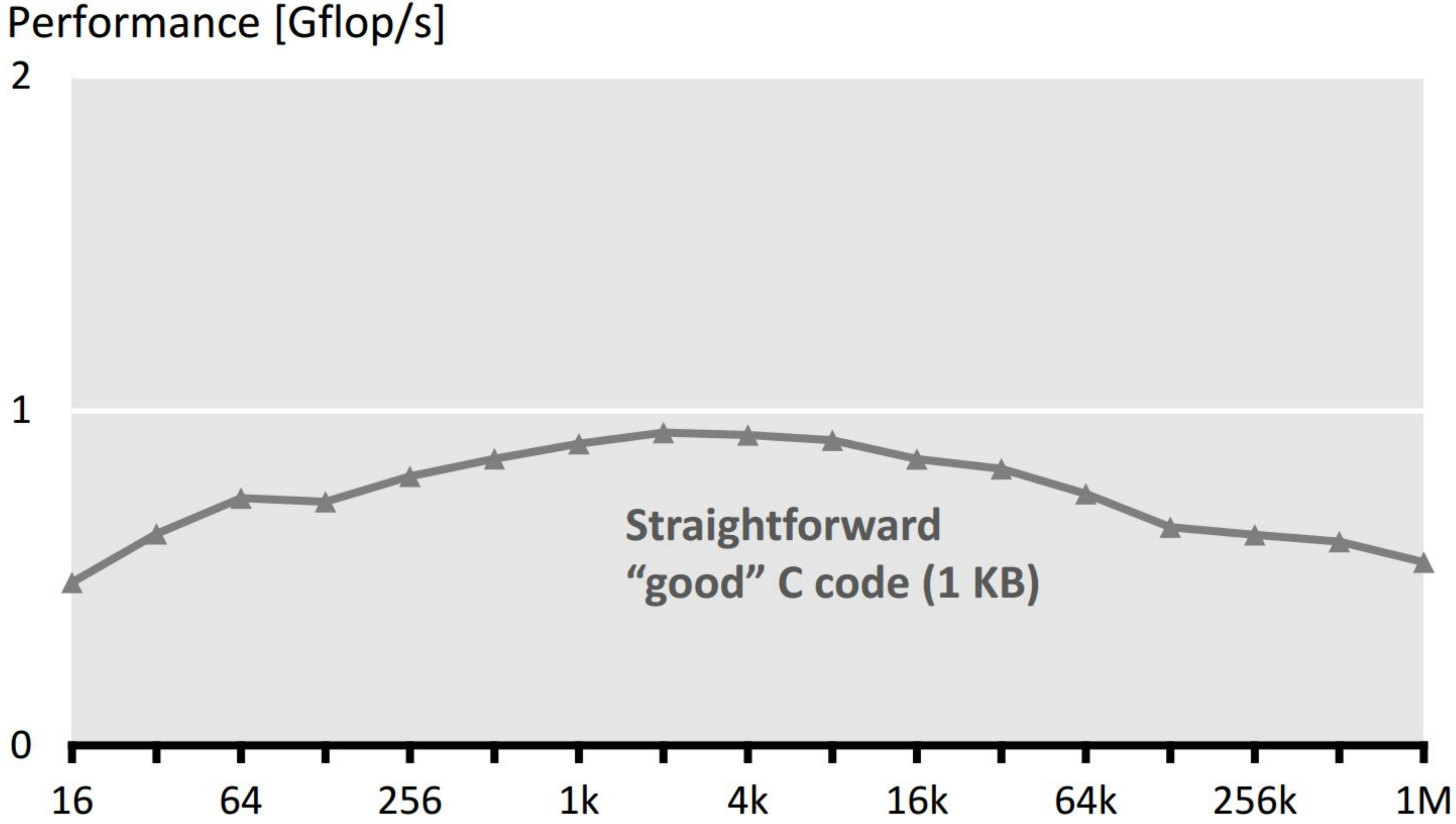
DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)

Performance [Gflop/s]



Source: prof. Markus Pueschel, ETH Zurich

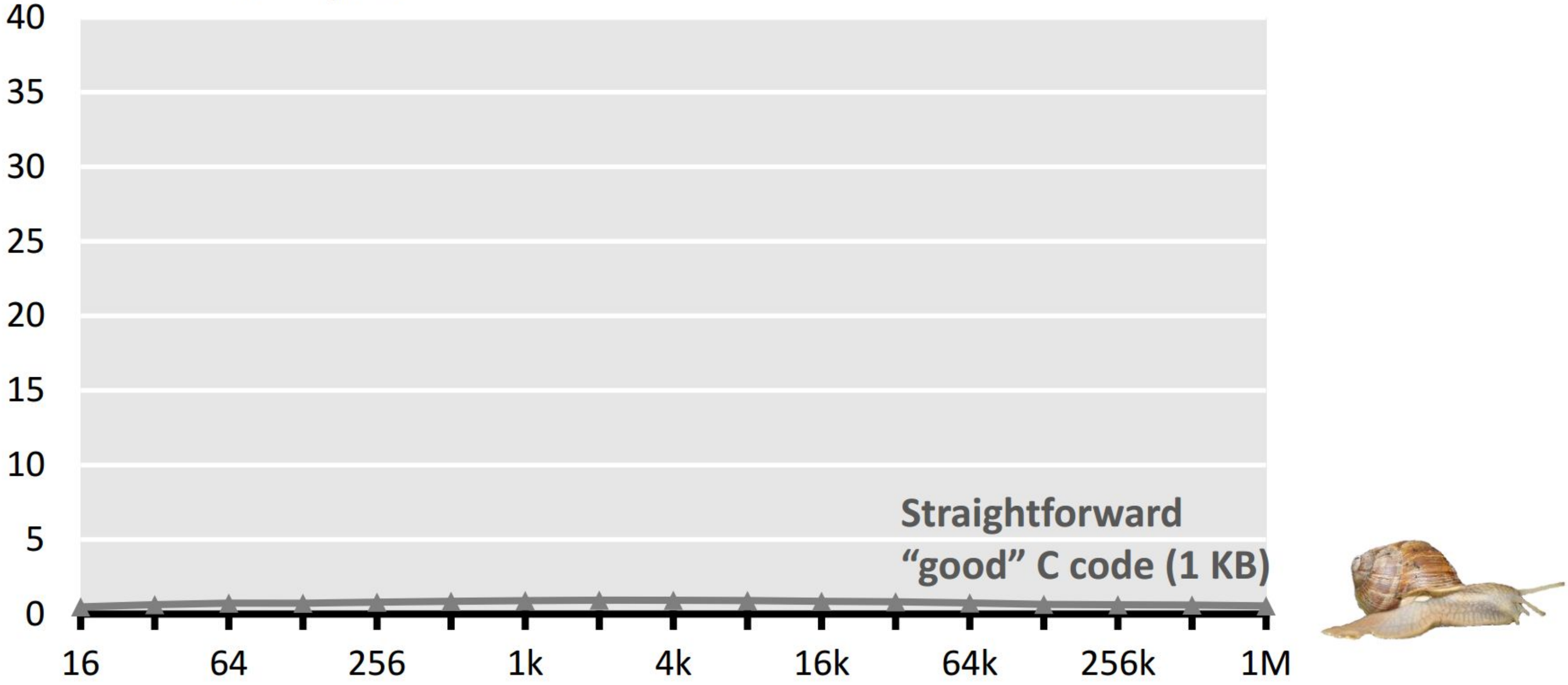
DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)



Source: prof. Markus Pueschel, ETH Zurich

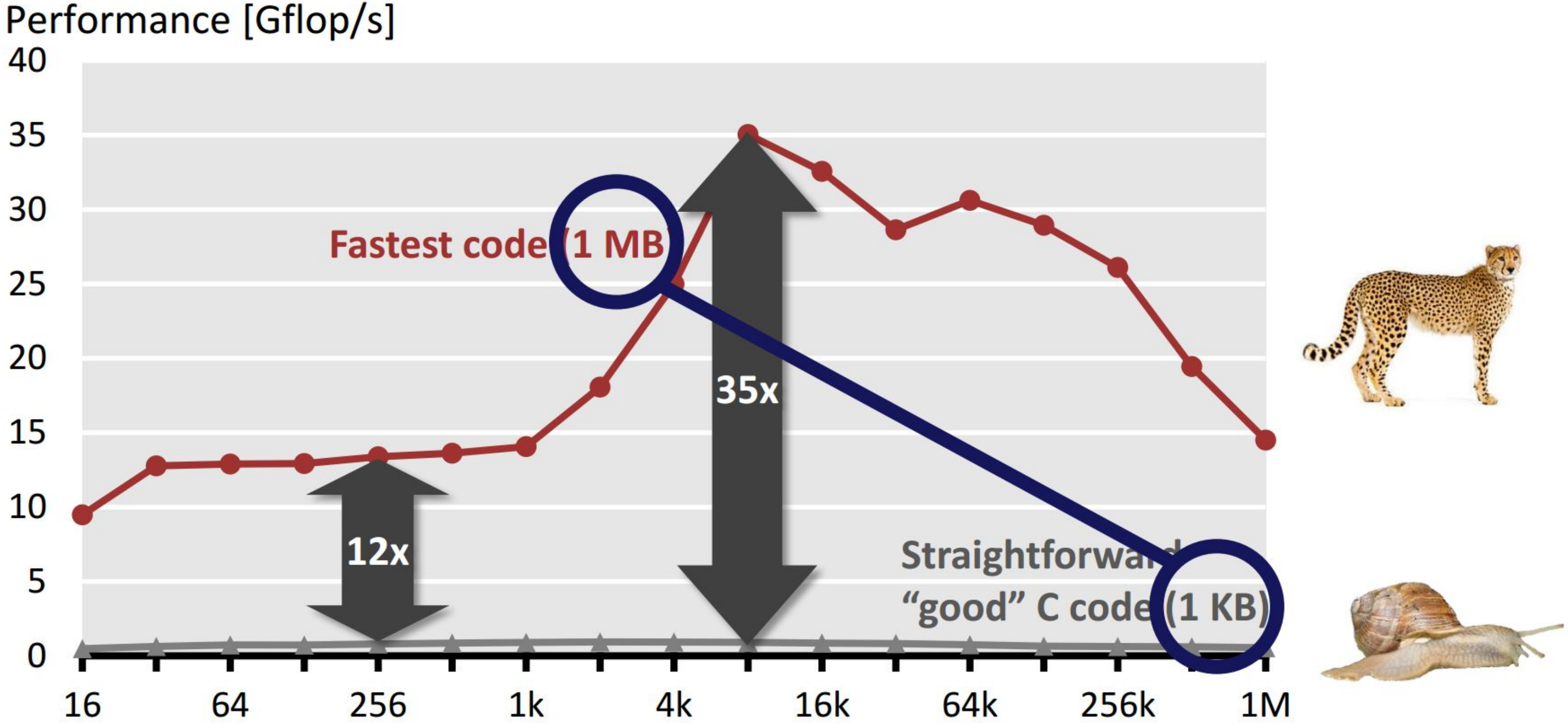
DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)

Performance [Gflop/s]



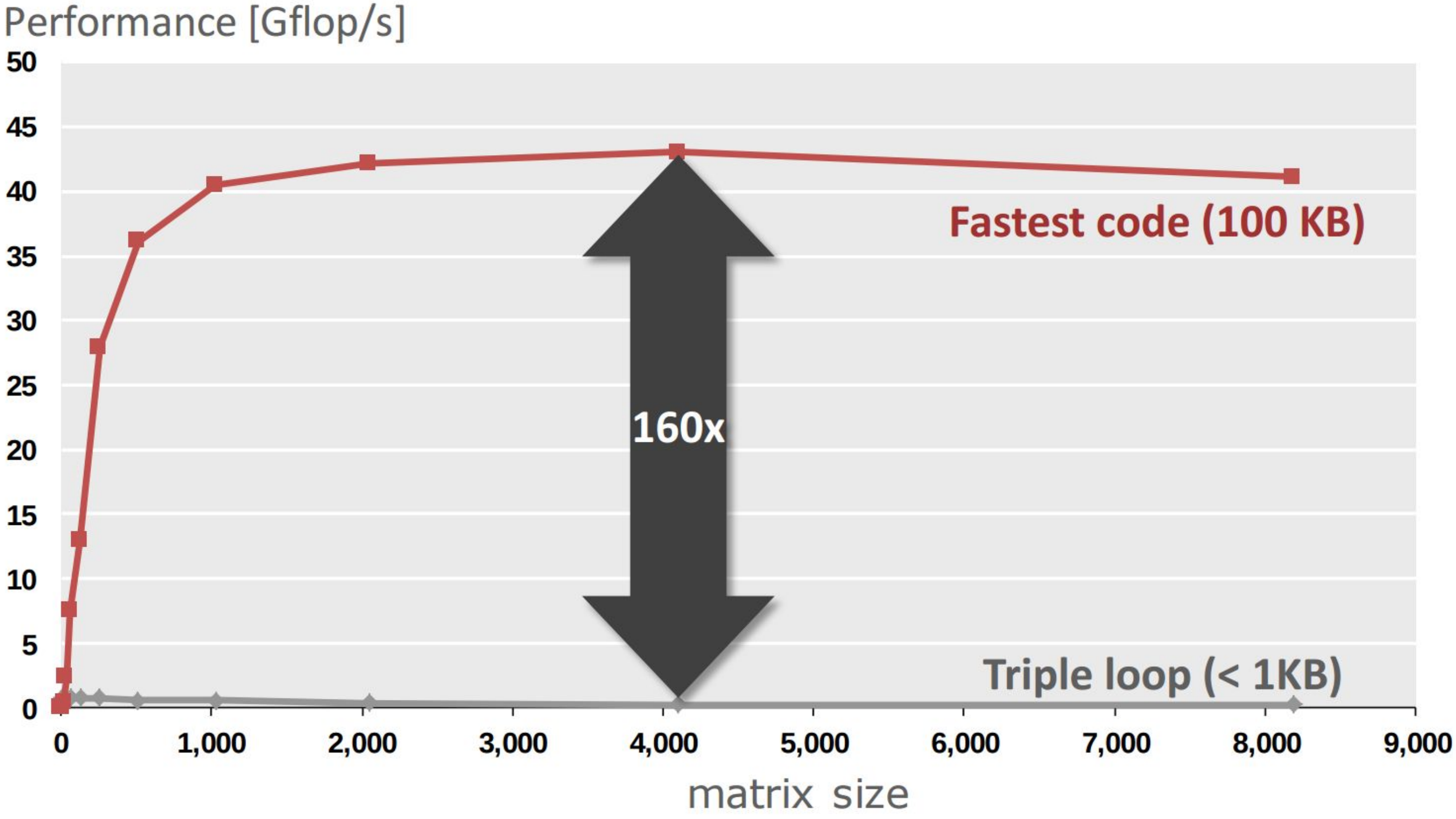
Source: prof. Markus Pueschel, ETH Zurich

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



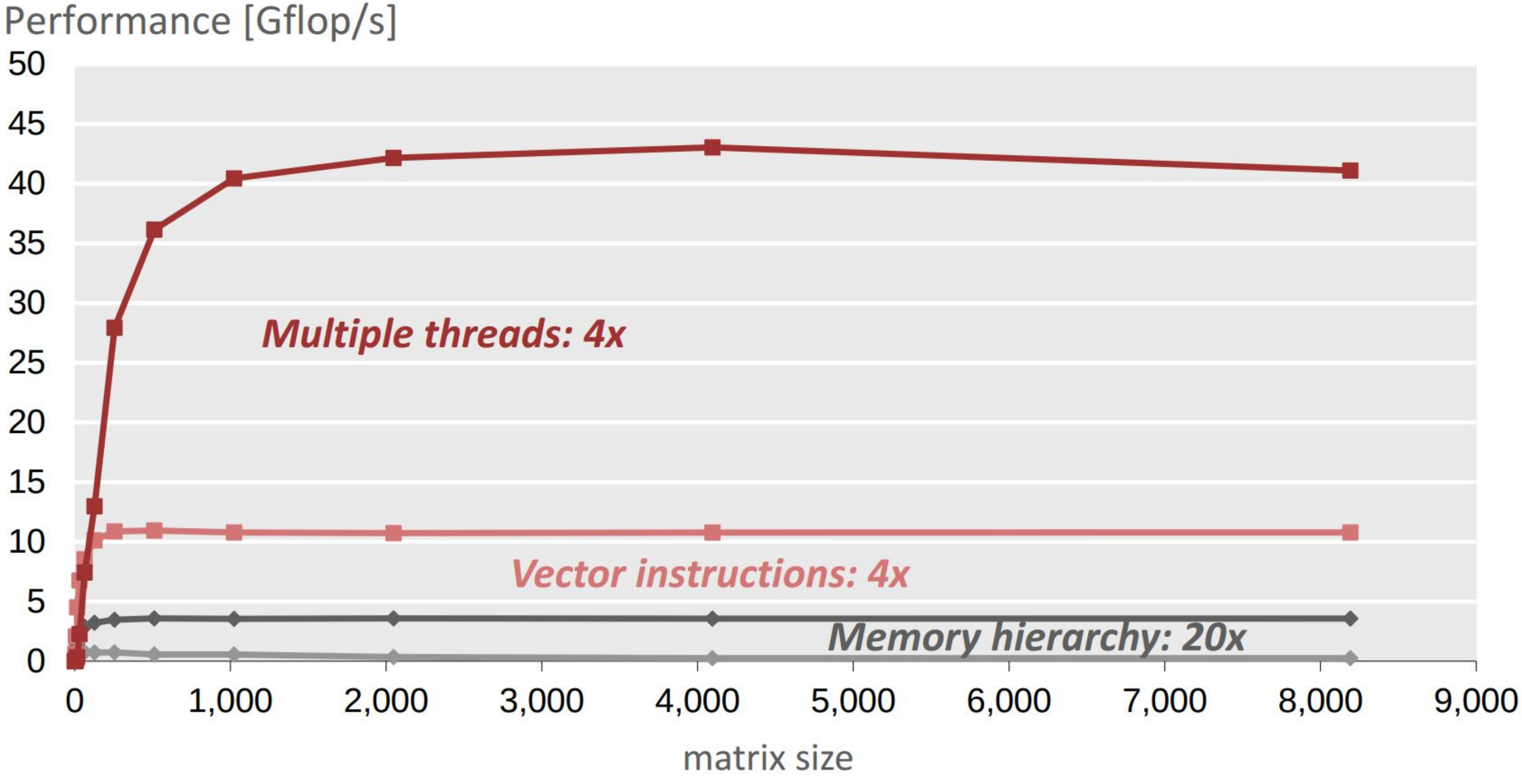
Source: prof. Markus Pueschel, ETH Zurich

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



Source: prof. Markus Pueschel, ETH Zurich

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



The compiler doesn't do the job

Source: prof. Markus Pueschel, ETH Zurich

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
- 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
- Highest performance is in general non-portable

Floating point peak performance

- Scalar:
  - 1 add and 1 mult / cycle
  - Assume 3 GHz:
  - **6 Gflop/s scalar peak performance on one core**
- Vector double precision (SSE)
  - 1 vadd and 1 vmult / cycle (2-way)
  - Assume 3 GHz:
  - **12 Gflop/s peak performance on one core**
- Vector single precision (SSE)
  - 1 vadd and 1 vmult / cycle (4-way)
  - Assume 3 GHz:
  - **24 Gflop/s peak performance on one core**

Floating point peak performance

- Overall peak on 3 GHz Core 2 and Core i3/i5/i7 Nehalem: (2 cores, SSE)
  - Double precision: 24 Gflop/s
  - Single precision: 48 Gflop/s
- Overall peak on 3 GHz Core i3/i5/i7 Sandy Bridge: (4 cores, AVX)
  - Double precision: 96 Gflop/s
  - Single precision: 192 Gflop/s

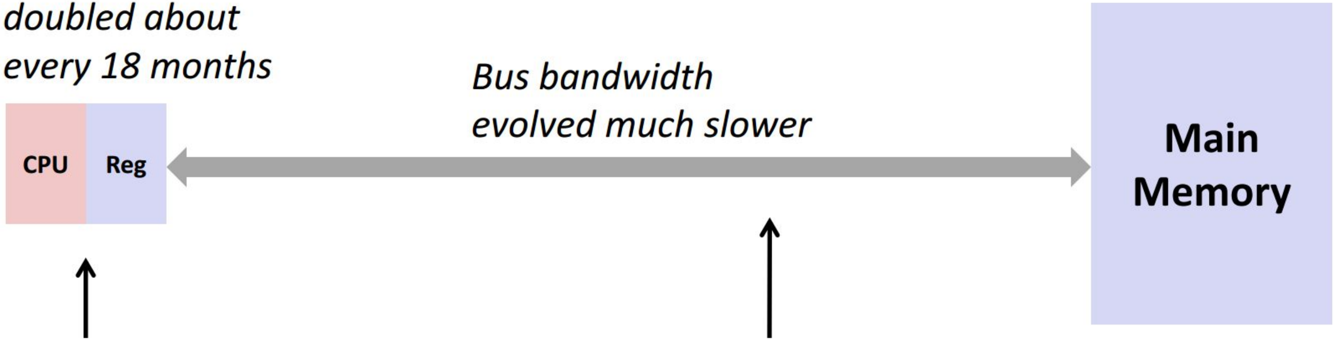
How To Make Code Faster?

- It depends!
- Memory bound:
    - Reduce memory traffic
    - Reduce cache misses, register spills
    - Compress data
- Compute bound:
    - Keep floating point units busy
    - Reduce cache misses, register spills
    - Instruction level parallelism (ILP)
    - Vectorization

Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months

Bus bandwidth evolved much slower



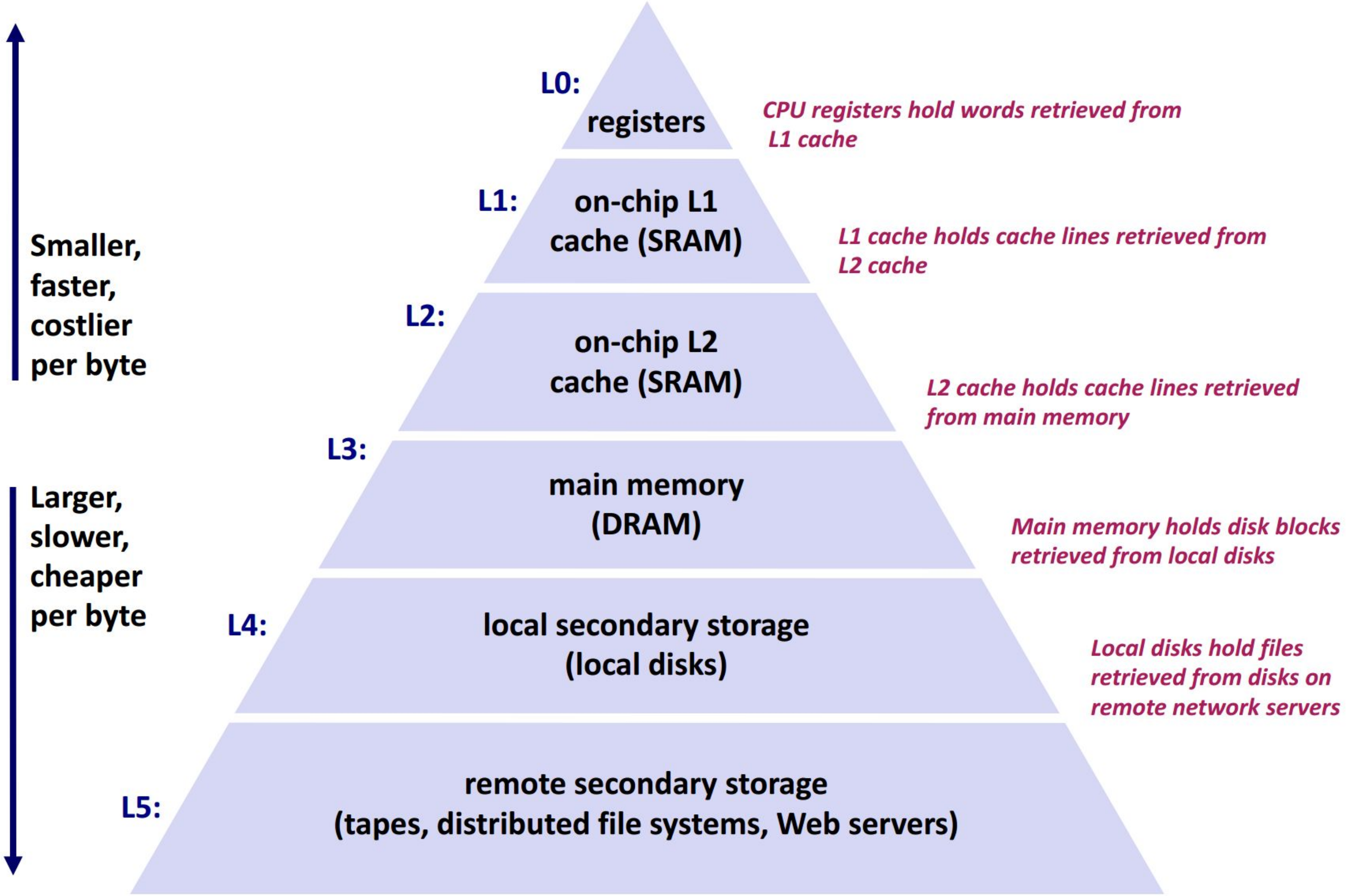
Core 2 Duo:
Peak performance:
2 SSE two operand ops/cycles
consumes up to 64 Bytes/cycle

Core 2 Duo:
Bandwidth
2 Bytes/cycle

Solution: Caches/Memory hierarchy

Source: prof. Markus Pueschel, ETH Zurich

Typical Memory Hierarchy



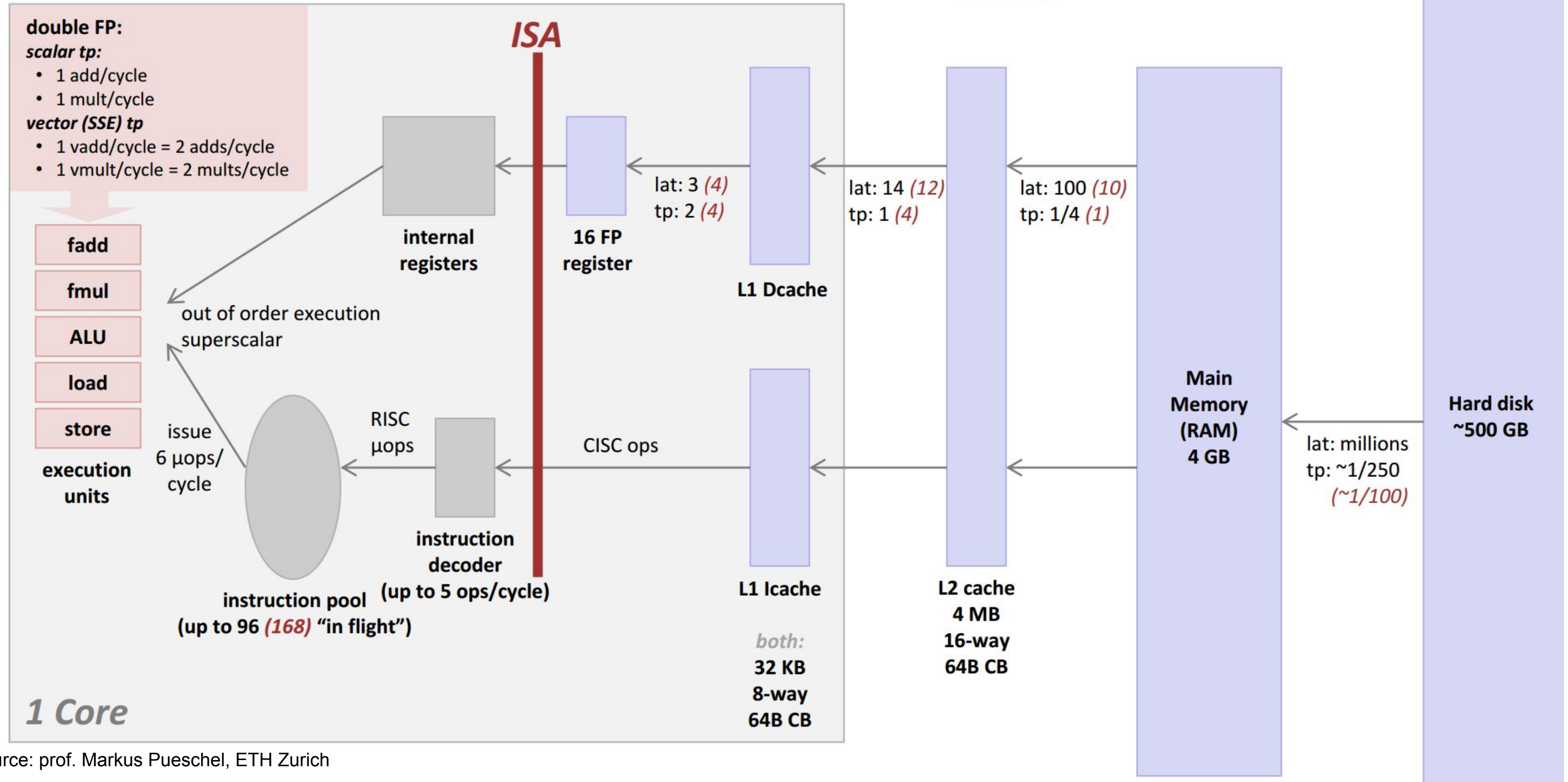
Source: prof. Markus Pueschel, ETH Zurich

Abstracted Microarchitecture: Example Core

Throughput (tp) is measured in doubles/cycle. For example: 2 *(4)*
Latency (lat) is measured in cycles
1 double floating point (FP) = 8 bytes
Rectangles not to scale

Core 2 (2008) — *Core i7 Sandy Bridge (2011)*

Memory hierarchy:
- Registers
- L1 cache
- L2 cache
- Main memory
- Hard disk



Source: prof. Markus Pueschel, ETH Zurich

Memory/Compute bound

- Operational intensity of a program/algorithm:

$$I = \frac{\textit{Number of operations}}{\textit{Amount of data transferred cache} \leftrightarrow \textit{RAM}}$$

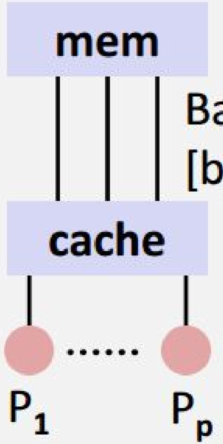
- Notes:
    - I depends on the computer (e.g., the cache size and structure)
    - Q: Relation to cache misses?
    - A: Denominator determined by misses in lowest level cache
- “Definition:” Programs with high I are called compute bound, programs with low I are called memory bound

Roofline model (Williams et al. 2008)

Resources in a processor that bound performance:

- peak performance [flops/cycle]
- memory bandwidth [bytes/cycle]
- <others>

Platform model



Bandwidth β [bytes/cycle] ← carefully measured

- raw bandwidth from manual is unattainable (maybe 60% is)
- Stream benchmark may be conservative

Peak performance π [ops/cycle]

Source: prof. Markus Pueschel, ETH Zurich

Algorithm model (n is the input size)

Operational intensity $I(n) = W(n)/Q(n) =$

$$\frac{\text{number of flops (cost)}}{\text{number of bytes transferred between memory and cache}} \quad \text{[ops/bytes]}$$

$Q(n)$: assumes empty cache;
best measured with performance counters

Notes

In general, Q and hence W/Q depend on the cache size m [bytes].
For some functions the optimal achievable W/Q is known:

FFT/sorting: $\Theta(\log(m))$

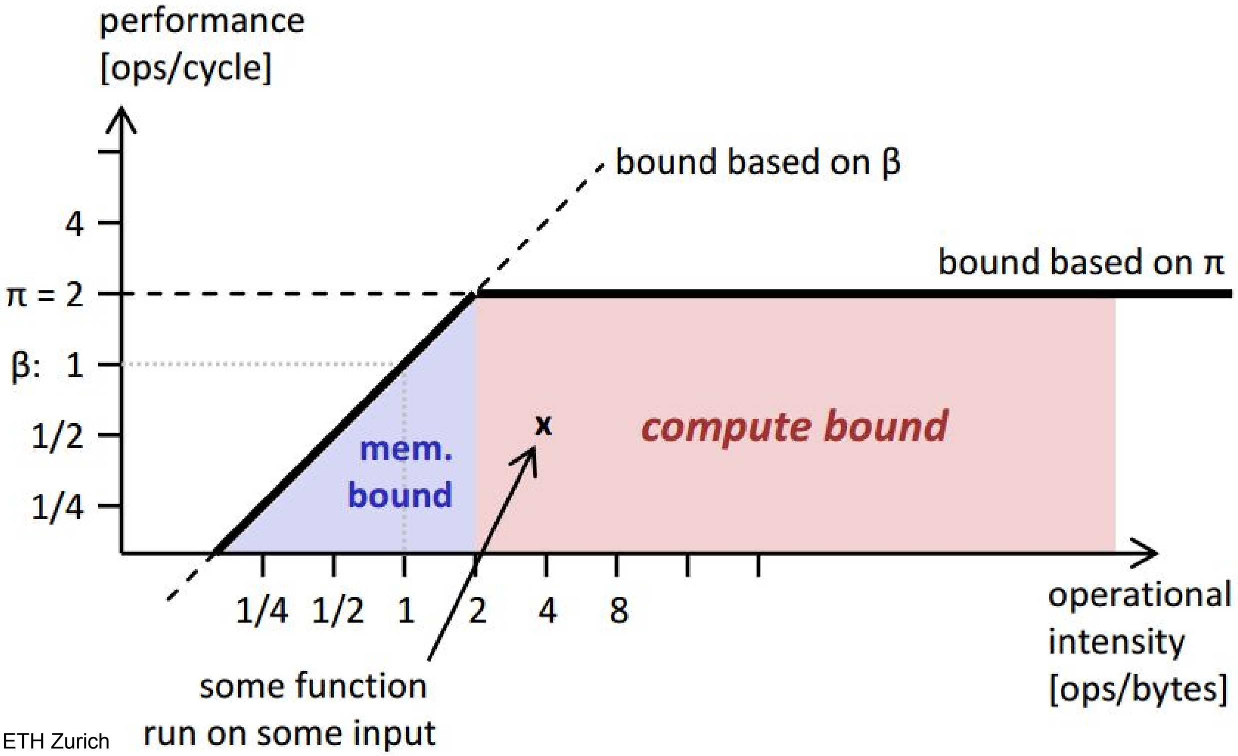
Matrix multiplication: $\Theta(\mathrm{sqrt}(m))$

Source: prof. Markus Pueschel, ETH Zurich

Roofline model

Example: one core with π = 2 and β = 1 and no SSE

ops are double precision flops



Source: prof. Markus Pueschel, ETH Zurich

Bound based on β?

- assume program as operational intensity of x ops/byte
- it can get only β bytes/cycle
- hence: performance = $y \leq \beta x$
- in log scale: $\log_2(y) \leq \log_2(\beta) + \log_2(x)$
- line with slope 1; $y = \beta$ for $x = 1$

More lines (bounds) can be added

- single core but with SSE (π goes up by 2x)
- four cores (π goes up by 4x)
- accesses with no spatial locality (β comes down by 8x)

Source: prof. Markus Pueschel, ETH Zurich