How to Write Fast Numerical Code

Spring 2011
Lecture 6

Instructor: Markus Püschel

TA: Georg Ofenbeck

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Organizational

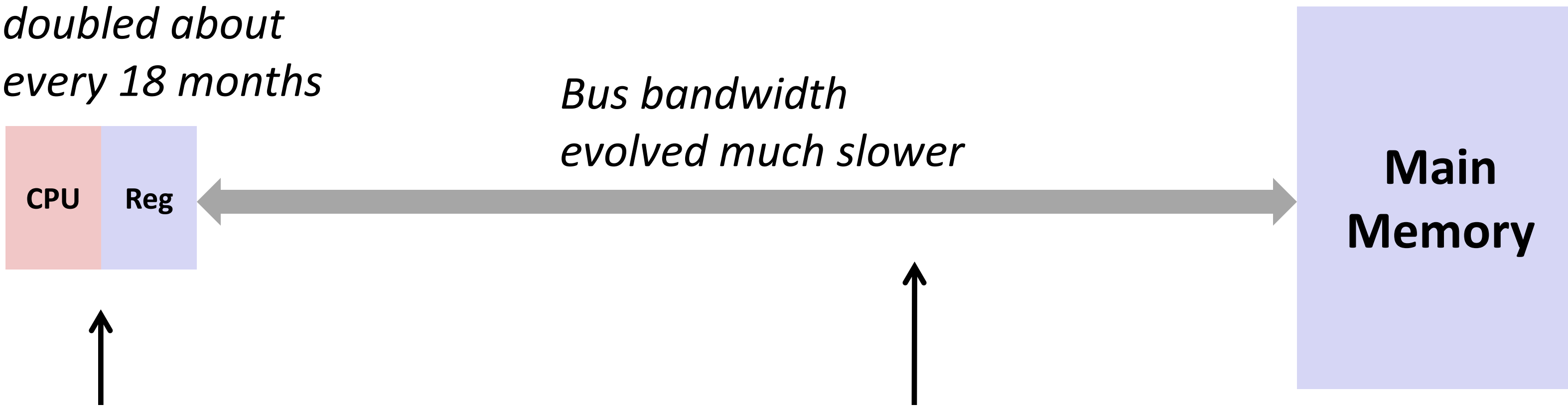
- *Class this Friday 18.3*
- **Exam?**
  - Monday April 11 (Sechseläuten, afternoon is off)
  - Friday April 15

Organization

- **Temporal and spatial locality**
- **Caches**

Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months

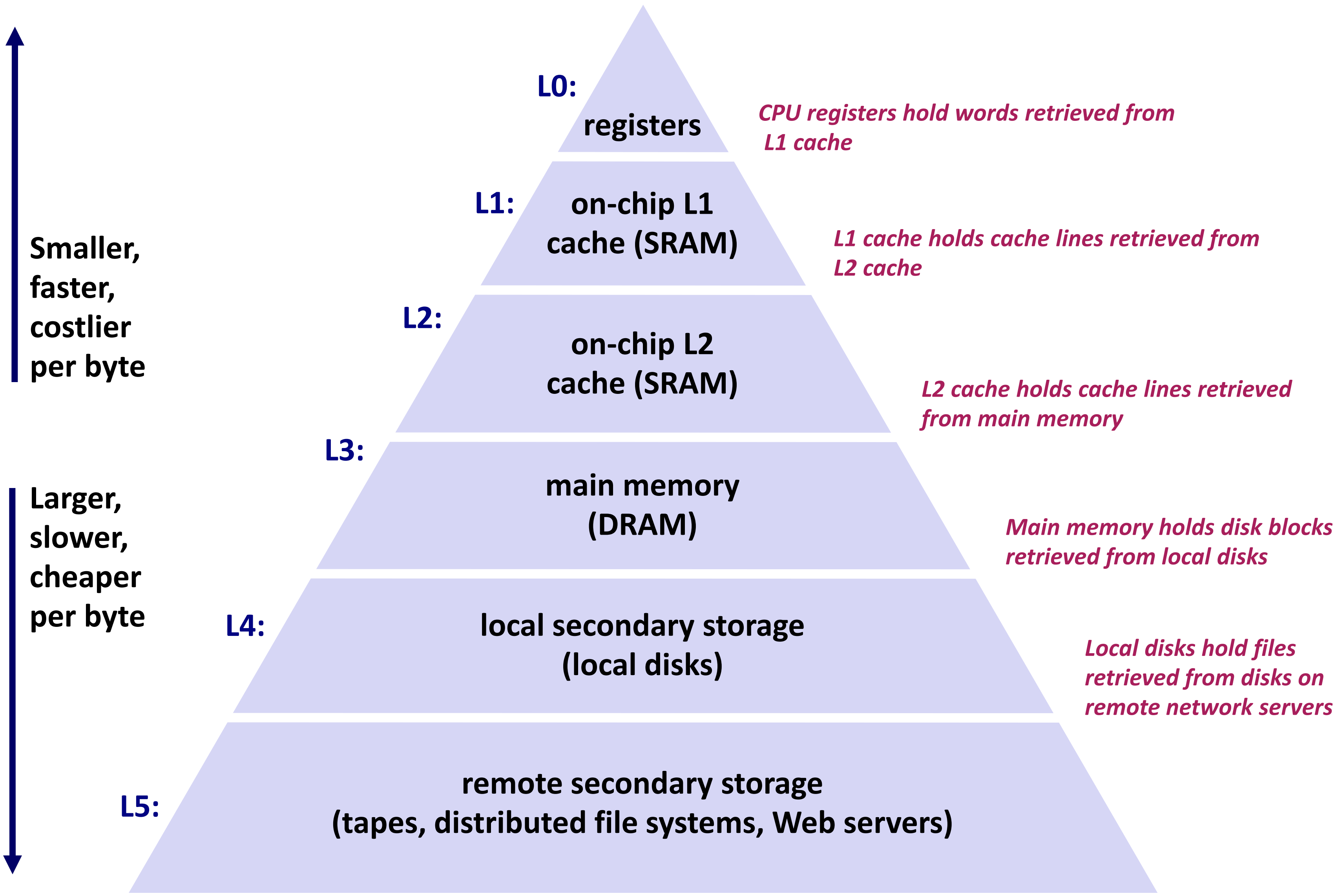


Core 2 Duo:
Peak performance:
2 SSE two operand ops/cycles
512 Bytes/cycle

Core 2 Duo:
Bandwidth
2 Bytes/cycle
Latency
100 cycles

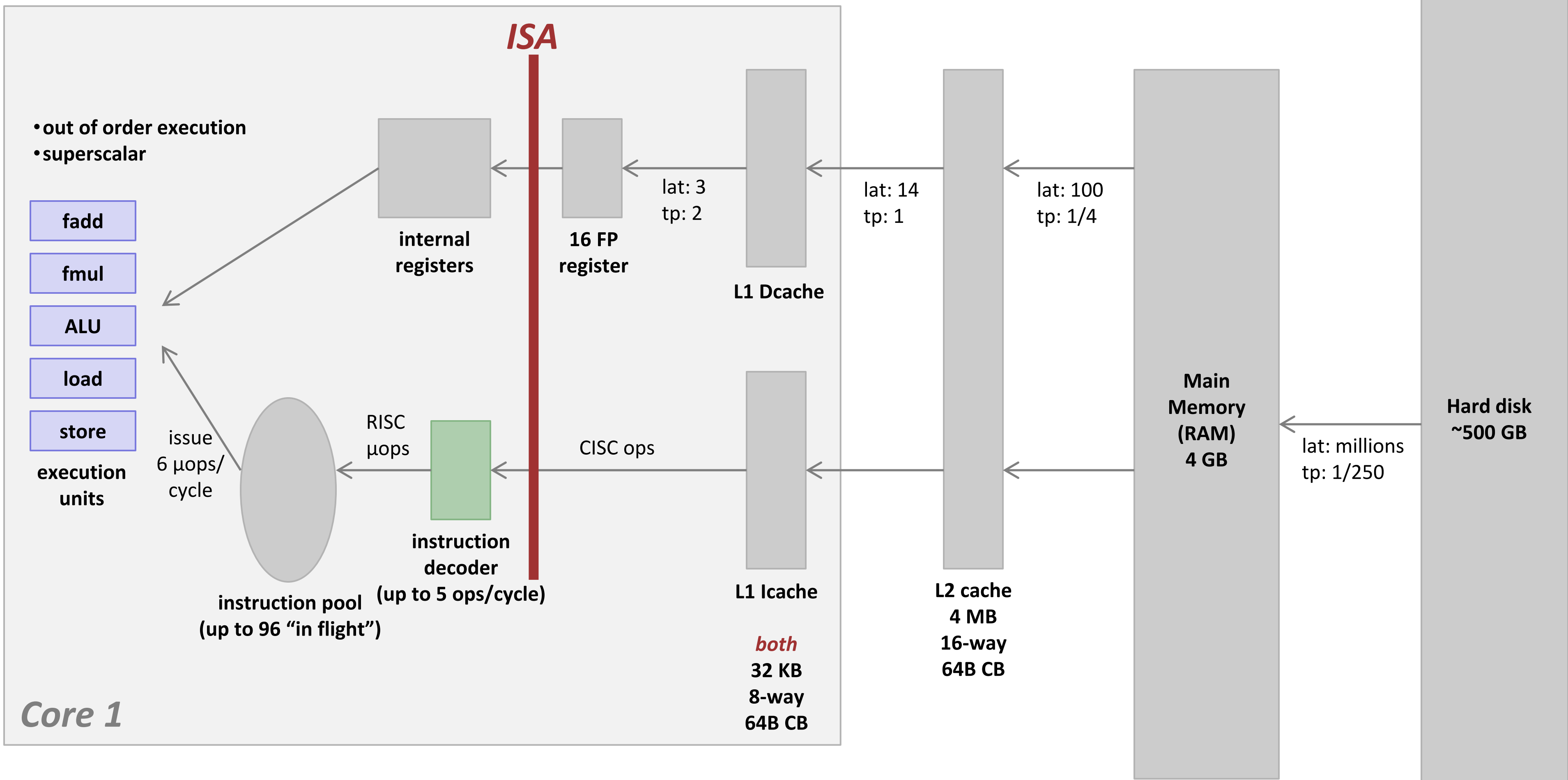
Solution: Caches/Memory hierarchy

Typical Memory Hierarchy

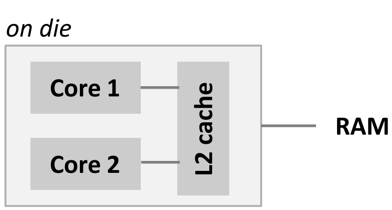


Abstracted Microarchitecture: Example Core (2008)

Throughput is measured in doubles/cycle
Latency in cycles for one double
1 double = 8 bytes
Rectangles not to scale



Core 2 Duo:



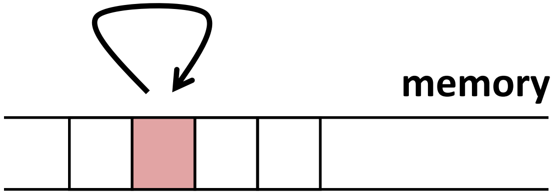
Memory hierarchy:

- Registers
- L1 cache
- L2 cache
- Main memory
- Hard disk

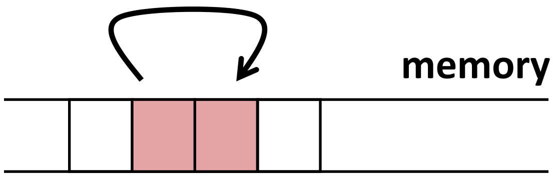
Why Caches Work: Locality

- *Locality:* **Programs tend to use data and instructions with addresses near or equal to those they have used recently**
  History of locality

- *Temporal locality:*
  - Recently referenced items are likely to be referenced again in the near future



- *Spatial locality:*
  - Items with nearby addresses tend to be referenced close together in time



Example: Locality?

```
sum = 0;
for (i = 0; i < n; i++)
  sum += a[i];
return sum;
```

- **Data:**
  - Temporal: **`sum`** referenced in each iteration
  - Spatial: array **`a[]`** accessed in stride-1 pattern
- **Instructions:**
  - Temporal: loops cycle through the same instructions
  - Spatial: instructions referenced in sequence

- ***Being able to assess the locality of code is a crucial skill for a performance programmer***

Locality Example #1

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Locality Example #2

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Locality Example #3

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];
    return sum;
}
```

- **How to improve locality?**

Reuse (Inherent Temporal Locality)

- *Reuse of an algorithm:*

$$\frac{\textit{Number of operations}}{\textit{Size of input + size of output data}}$$

- **Typically:**
  no. operations = arithmetic cost = no. floating point adds and mults

- **Intuitively measures how often every input element is on average needed in the computation**

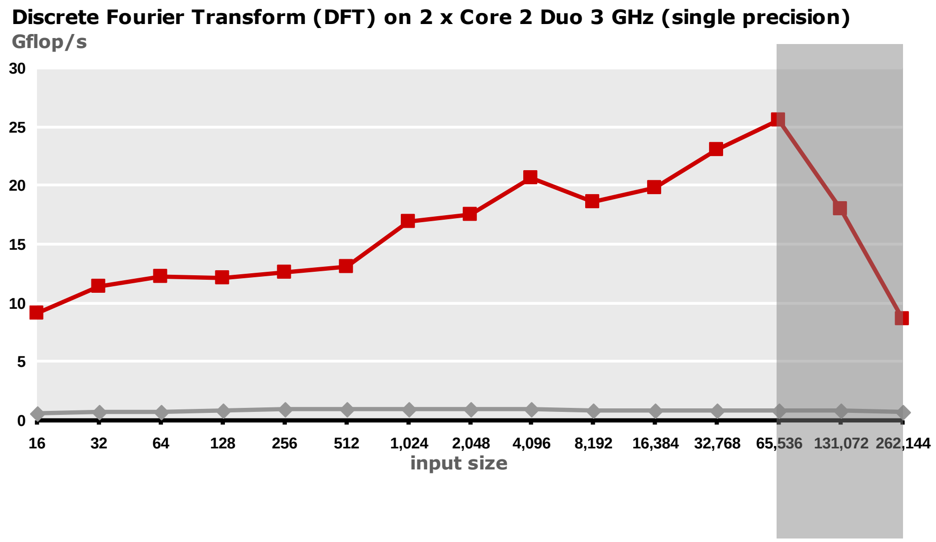
- **Examples:**
  - Matrix multiplication C = AB + C $\frac{2n^3}{3n^2} = \frac{2}{3}n = O(n)$
  - Discrete Fourier transform $\approx \frac{5n\log_2(n)}{2n} = \frac{5}{2}\log_2(n) = O(\log(n))$
  - Adding two vectors x = x+y $\frac{n}{2n} = \frac{1}{2} = O(1)$

CPU bound versus Memory bound

- **Definitions are not precise**
- **An algorithm with high reuse is called *CPU bound***
  - Most time is spent computing
  - Will run faster if CPU is faster
- **An algorithm with low reuse is called *memory bound***
  - Most time spent transferring data in the memory hierarchy
  - Will run faster if memory bus is faster
- ***Performance optimization:* Make sure that high reuse actually translates into few cache misses, i.e., into temporal locality with respect to the cache**

Effects

FFT: O(log(n)) reuse

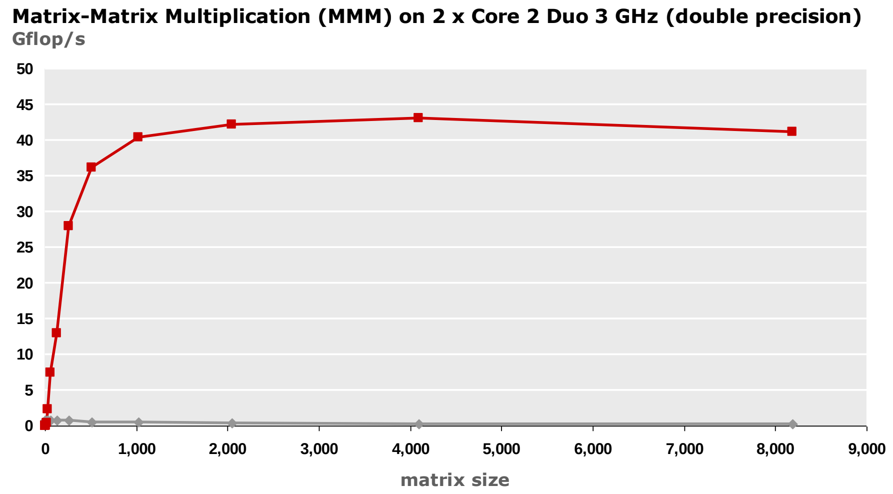


Up to 40-50% peak

Performance drop outside L2 cache

Most time spent transferring data

MMM: O(n) reuse



Up to 80-90% peak

Performance can be maintained

Cache miss time compensated/hidden by computation