How to Write Fast Numerical Code

Spring 2012
Lecture 6

Instructor: Markus Püschel

TAs: Georg Ofenbeck & Daniele Spampinato

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

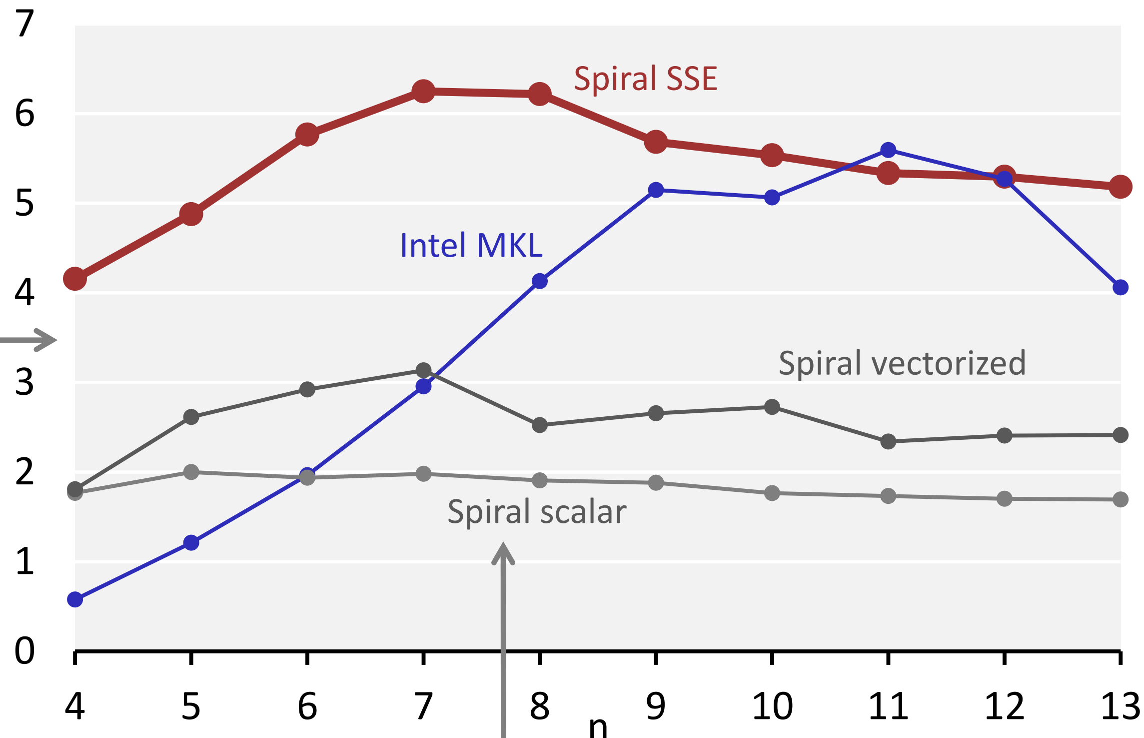
Left alignment

Attractive font (sans serif, avoid Arial)
Calibri, Helvetica, Gill Sans MT, …

DFT 2^n (single precision) on Pentium 4, 2.53 GHz

[Gflop/s]

Horizontal y-label



Main line possibly emphasized (red, thicker)

No y-axis (superfluous)

Background/grid inverted for better layering

No legend; makes decoding easier

Last Time: Optimizing Compilers

-O

- **Always use optimization flags:**
  - gcc: *default is no optimization* (-O0)!
  - icc: some optimization is turned on
- **Good choices for gcc/icc: -O2, -O3, -march=xxx, -mSSE3, -m64**
  - Read in manual what they do
  - Try to understand the differences
- **Try different flags and maybe different compilers**

Last Time: Optimization Blockers

overhead through abstract data types

```
...
  for (i = 0; i < n; i++) {
    get_vec_element(v, i, &t);
    *res += t;
  }
  return res;
}
```

optimization blocker: procedure

```
void lower(char *s)
{
  int i;
  for (i = 0; i < strlen(s); i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
      s[i] -= ('A' - 'a');
}
```

optimization blocker: potential aliasing

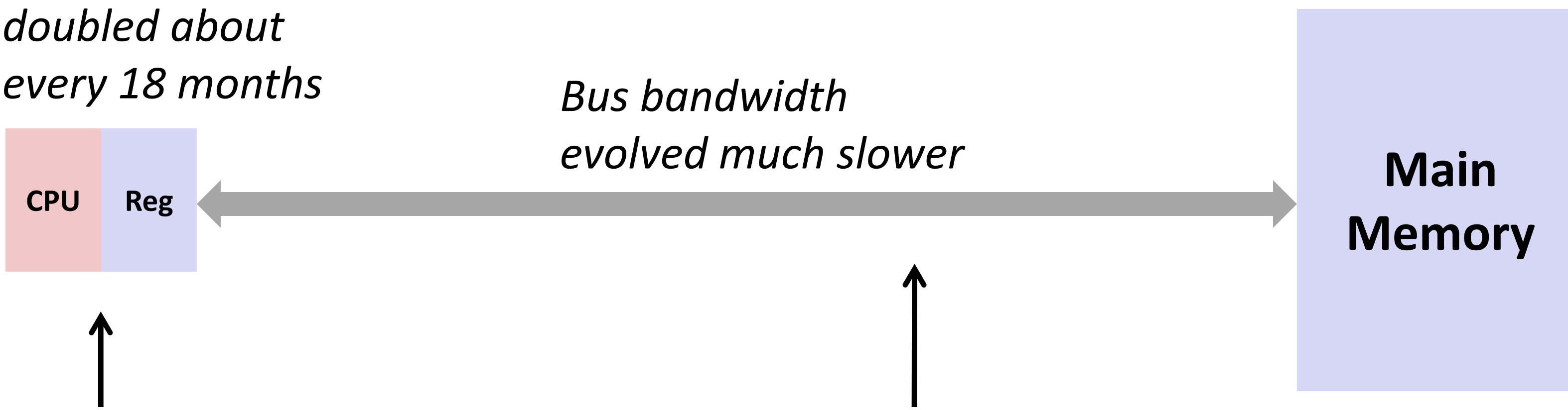
```
/* Sums rows of n x n matrix a
   and stores in vector b   */
void sum_rows1(double *a, double *b, int n) {
  int i, j;

  for (i = 0; i < n; i++) {
    b[i] = 0;
    for (j = 0; j < n; j++)
      b[i] += a[i*n + j];
  }
}
```

Organization

- Temporal and spatial locality
- Operational intensity, memory/compute bound

Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months

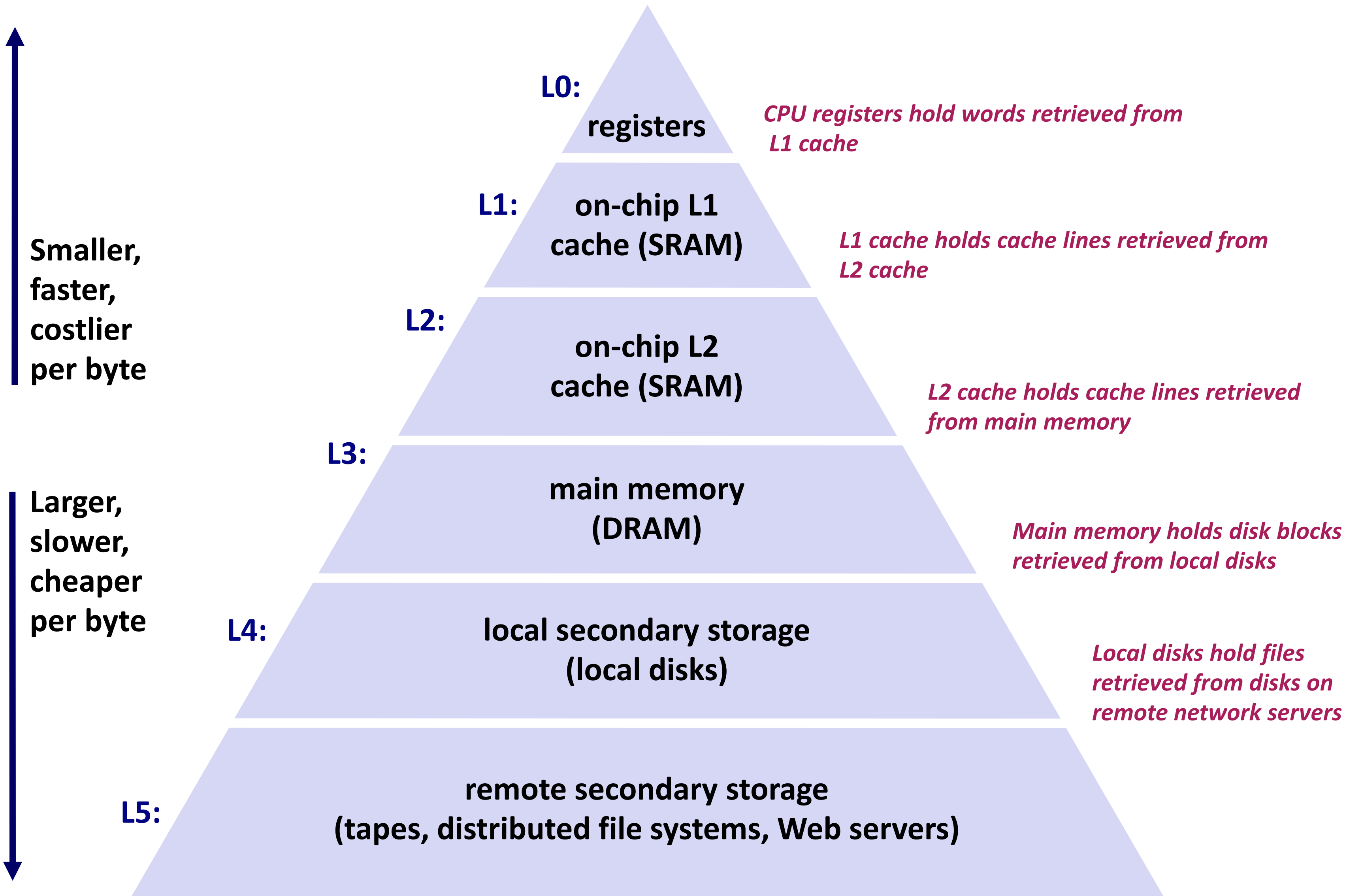


Core 2 Duo:
Peak performance:
2 SSE two operand ops/cycles
consumes up to 64 Bytes/cycle

Core 2 Duo:
Bandwidth
2 Bytes/cycle

Solution: Caches/Memory hierarchy

Typical Memory Hierarchy



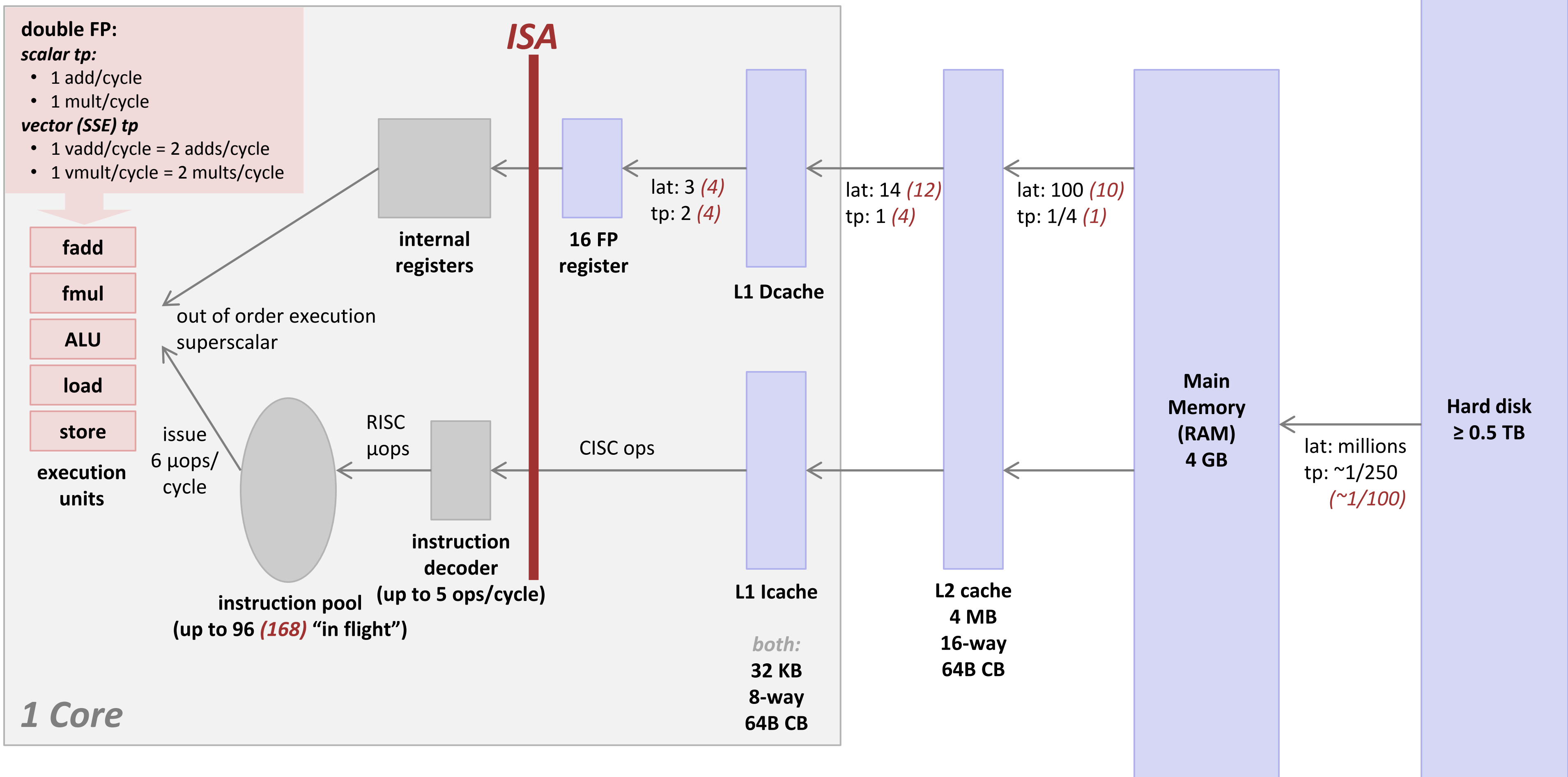
Abstracted Microarchitecture: Example Core

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

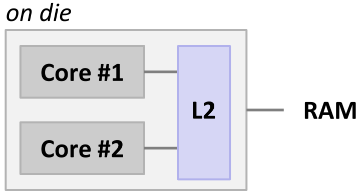
Core 2 (2008)
Core i7 Sandy Bridge (2011)

Memory hierarchy:

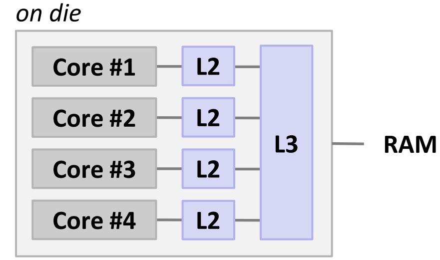
- Registers
- L1 cache
- L2 cache
- Main memory
- Hard disk



Core 2 Duo:



Core i7 Sandy Bridge:
256 KB L2 cache
2–8MB L3 cache: lat 26-31, tp 4
vector (AVX) tp
- 1 vadd/cycle = 4 adds/cycle
- 1 vmult/cycle = 4 mults/cycle

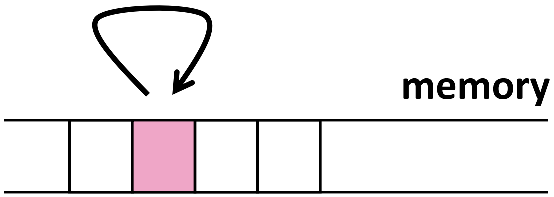


Source: Intel

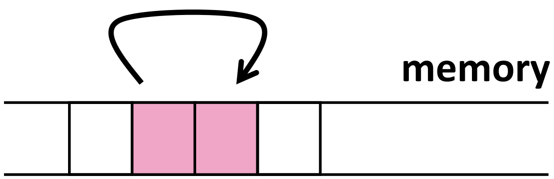
Why Caches Work: Locality

- *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?

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

- **This course usually:**
  - #ops = #flops
  - unit: flops/byte or flops/double

- *"Definition:"* **Programs with high *I* are called *compute bound*, programs with low *I* are called *memory bound***

Questions

- *Q:* How high is high enough for compute bound?

  A: Depends on the computer; we will make this precise later with the roofline model

- *Q:* Estimate the operational intensity

```
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;
}
```

Upper Bound on I

- **Assume cold (empty) cache:**

  Amount of data transferred cache ↔ RAM

  ≥ Size of input data + size of output data

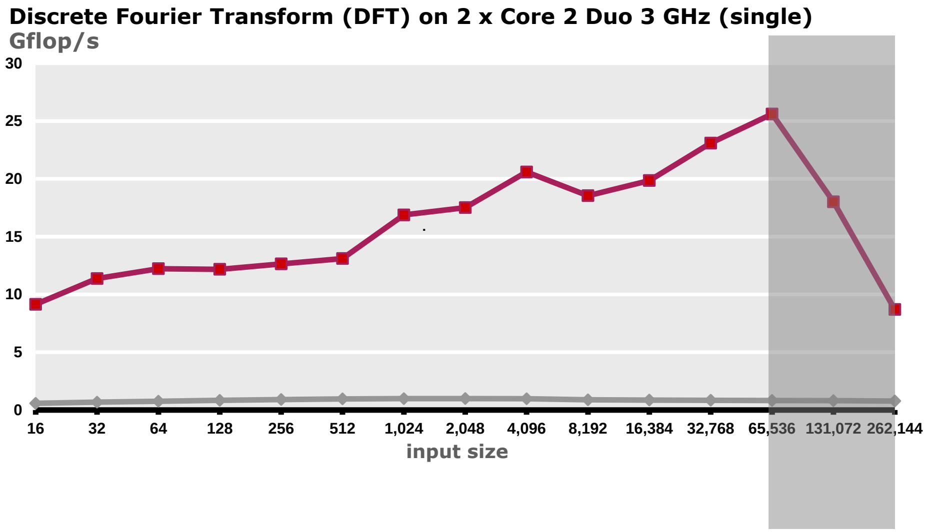
- **Hence:**

$$I \leq \frac{\text{Number of operations}}{\text{Size of input data + size of output data}}$$

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

Effects

FFT: *I(n) ≤ O(log(n))*

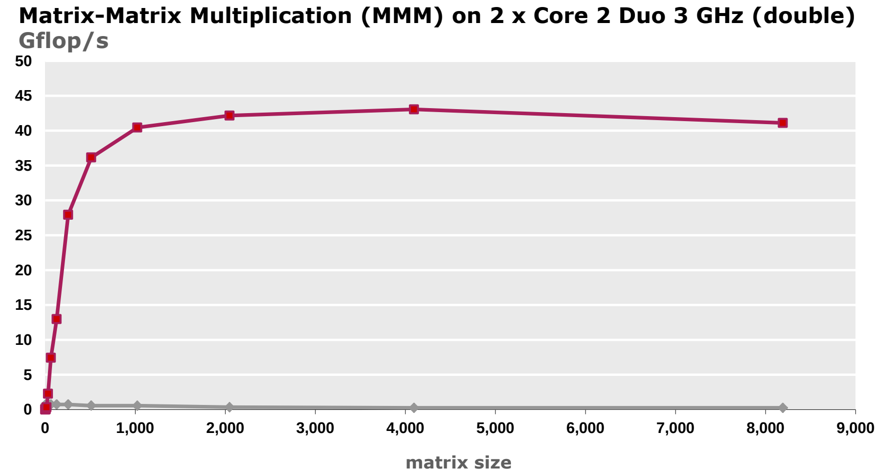


Up to 40-50% peak

Performance drop outside L2 cache

Most time spent transferring data

MMM: *I(n) ≤ O(n)*



Up to 80-90% peak

Performance can be maintained

Cache miss time compensated/hidden by computation