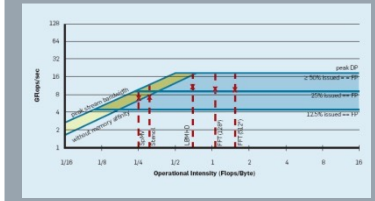
Erlangen Regional Computing Center

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

“Simple” performance modeling: The Roofline Model

Loop-based performance modeling: Execution vs. data transfer



R.W. Hockney and I.J. Curington: $f_{1/2}$: A parameter to characterize memory and communication bottlenecks. Parallel Computing 10, 277-286 (1989). DOI: 10.1016/0167-8191(89)90100-2

W. Schönauer: Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers. Self-edition (2000)

S. Williams: Auto-tuning Performance on Multicore Computers. UCB Technical Report No. UCB/EECS-2008-164. PhD thesis (2008)

Performance Modeling – Motivation

Maximum floating point (FP) performance:

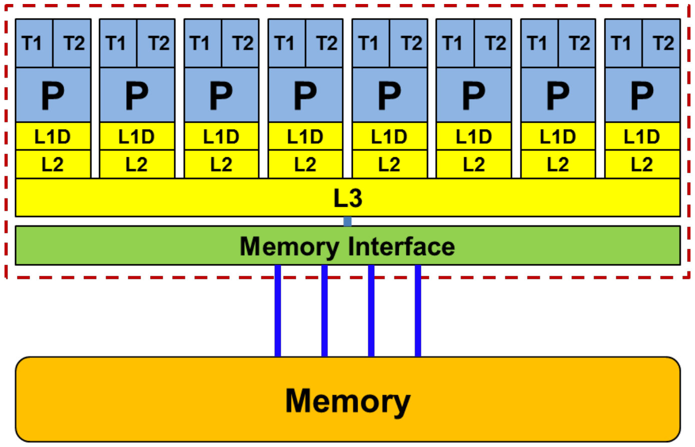
Clock Speed

$$P_{peak} = n_{core} \cdot n_{super}^{FP} \cdot n_{FMA} \cdot n_{SIMD} \cdot f \; [\frac{Flop}{s}]$$

Superscalarity

FMA factor

SIMD factor



?

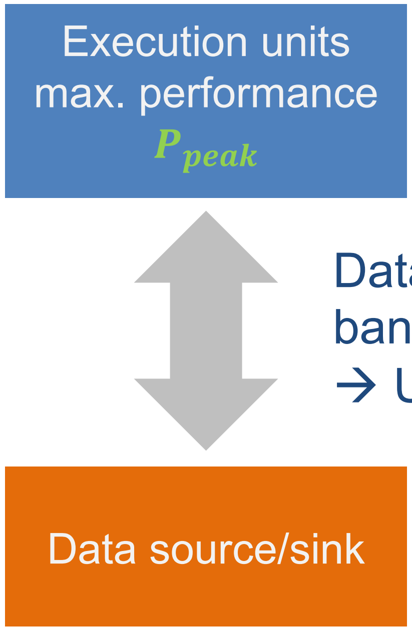
```
!$OMP PARALLEL DO
do k = 1 , 400
   do j = 1 , 400; do i = 1 , 400
      y(i,j,k)= b*(    x(i-1,j,k)+ x(i+1,j,k)+ x(i,j-1,k)+
                       x(i,j+1,k)+ x(i,j,k-1)+ x(i,j,k+1))
    enddo; enddo
enddo
!$OMP END PARALLEL DO
```

Maximum main memory bandwidth

$$b_S = n_{Channel} \cdot 8\,Byte \cdot f \; [\frac{Byte}{s}]$$

A simple performance model for loops

Simplistic view of the hardware:



Data path,
bandwidth b_S
→ Unit: byte/s

Simplistic view of the software:

```
! may be multiple levels
do i = 1,<sufficient>
  <complicated stuff doing
    N flops causing
    V bytes of data transfer>
enddo
```

Computational intensity

$$I = \frac{N}{V}$$

→ Unit: flop/byte

Naïve Roofline Model

How fast can tasks be processed? ***P* [flop/s]**

The bottleneck is either

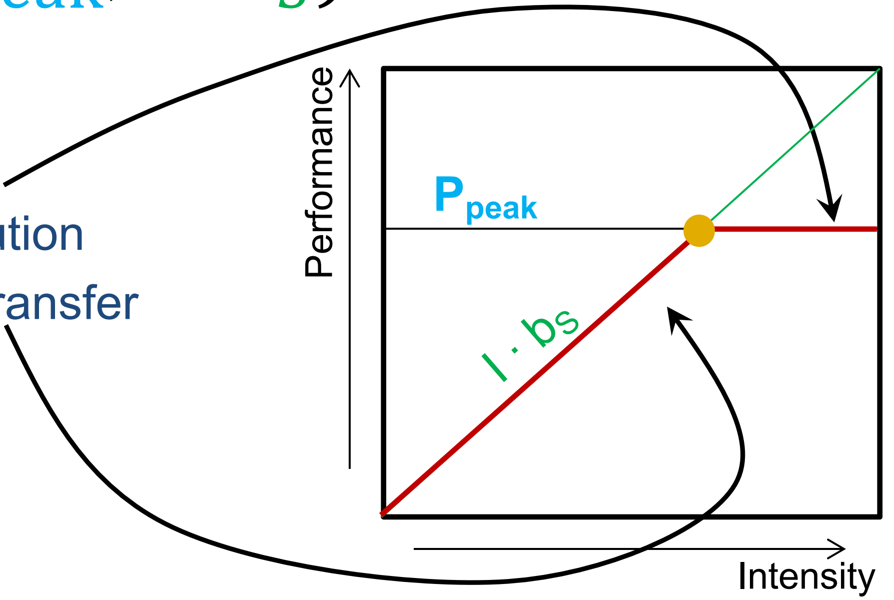
- The execution of work: P_{peak} [flop/s]
- The data path: $I \cdot b_S$ [flop/byte x byte/s]

$$P = \min(P_{\text{peak}}, I \cdot b_S)$$

This is the "Naïve Roofline Model"

- High intensity: P limited by execution
- Low intensity: P limited by data transfer
- "Knee" at $P_{max} = I \cdot b_S$: Best use of resources

- Roofline is an "optimistic" model ("light speed")



The Roofline Model in computing – Basics

Apply the naive Roofline model in practice

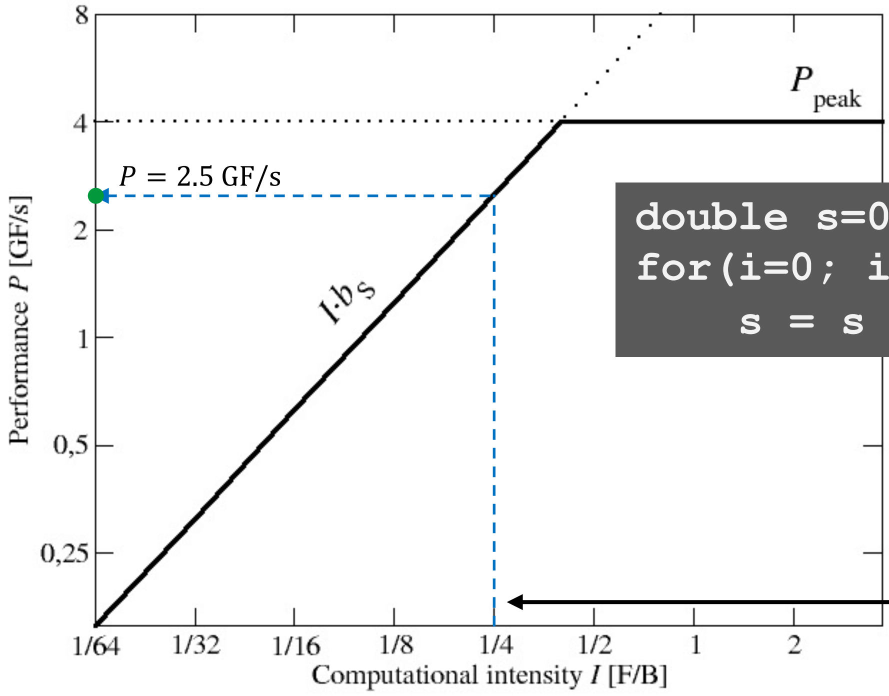
- Machine parameter #1: Peak performance: $P_{peak}\ \left[\frac{F}{s}\right]$
- Machine parameter #2: Memory bandwidth: $b_S\ \left[\frac{B}{s}\right]$
- Code characteristic: Computational intensity: $I\ \left[\frac{F}{B}\right]$

Machine properties:

$$P_{peak} = 4\,\frac{\text{GF}}{\text{s}}$$

$$b_S = 10\,\frac{\text{GB}}{\text{s}}$$

Application property: I



```
double s=0, a[];
for(i=0; i<N; ++i) {
    s = s + a[i] * a[i];}
```

$$I = \frac{2\ F}{8\ B} = 0.25\ {}^{F}/_{B}$$

Code balance: more examples

```
double a[], b[];
for(i=0; i<N; ++i) {
    a[i] = a[i] + b[i];}
```

B_C = 24B / 1F = 24 B/F

***I* = 0.042 F/B**

```
double a[], b[];
for(i=0; i<N; ++i) {
    a[i] = a[i]+ s * b[i];}
```

B_C = 24B / 2F = 12 B/F

***I* = 0.083 F/B**

Scalar – can be kept in register

```
float s=0, a[];
for(i=0; i<N; ++i) {
    s = s + a[i] * a[i];}
```

B_C = 4B / 2F = 2 B/F

***I* = 0.5 F/B**

Scalar – can be kept in register

```
float s=0, a[], b[];
for(i=0; i<N; ++i) {
    s = s + a[i] * b[i];}
```

B_C = 8B / 2F = 4 B/F

***I* = 0.25 F/B**

Scalar – can be kept in register

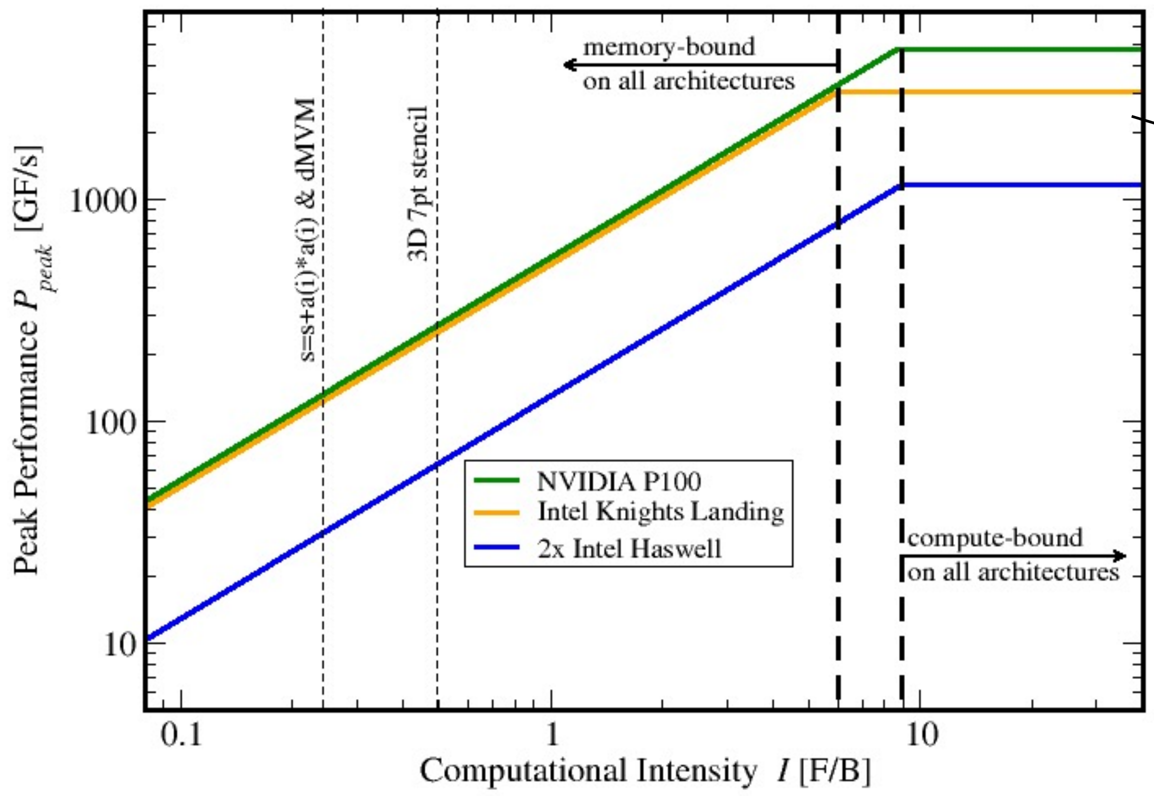
Prerequisites for the Roofline Model

- **The roofline formalism is based on some (crucial) prerequisites:**
  - There is a clear concept of "work" vs. "traffic"
    - "work" = flops, updates, iterations…
    - "traffic" = required data to do "work"
  - Machine input parameters: Peak Performance and Peak Bandwidth
    Application/kernel is expected to achieve is limits theoretically

- **Assumptions behind the model:**
  - Data transfer and core execution overlap perfectly!
    - **Either** the limit is core execution **or** it is data transfer
    - **Slowest limiting factor "wins";** all others are assumed to have no impact
  - Latency effects are ignored, i.e., perfect streaming mode
  - "Steady-state" code execution (no wind-up/-down effects)

The Roofline Model in computing – Basics

Compare capabilities of different machines:



Assuming double precision – for single precision: $P_{peak} \rightarrow 2 \cdot P_{peak}$

- Roofline always provides upper bound – but is it realistic?
- If code is not able to reach this limit (e.g., contains add operations only), machine parameters need to be redefined (e.g., $P_{peak} \rightarrow P_{peak}/2$)

A refined Roofline Model

1. $P_{\max}$ = Applicable peak performance of a loop, assuming that data comes from the level 1 cache (this is not necessarily P_{peak})
   → e.g., $P_{\max}$ = 176 GFlop/s

2. I = Computational intensity ("work" per byte transferred) over the slowest data path utilized (code balance $B_C = I^{-1}$)
   → e.g., I = 0.167 Flop/Byte → B_C = 6 Byte/Flop

3. b_S = Applicable (saturated) peak bandwidth of the slowest data path utilized (measure attainable bandwidth using, e.g. STREAM)
   → e.g., b_S = 56 GByte/s

Expected performance:

$$P = \min(P_{\max}, I \cdot b_S) = \min\left(P_{\max}, \frac{b_S}{B_C}\right)$$

[Byte/s]

[Byte/Flop]

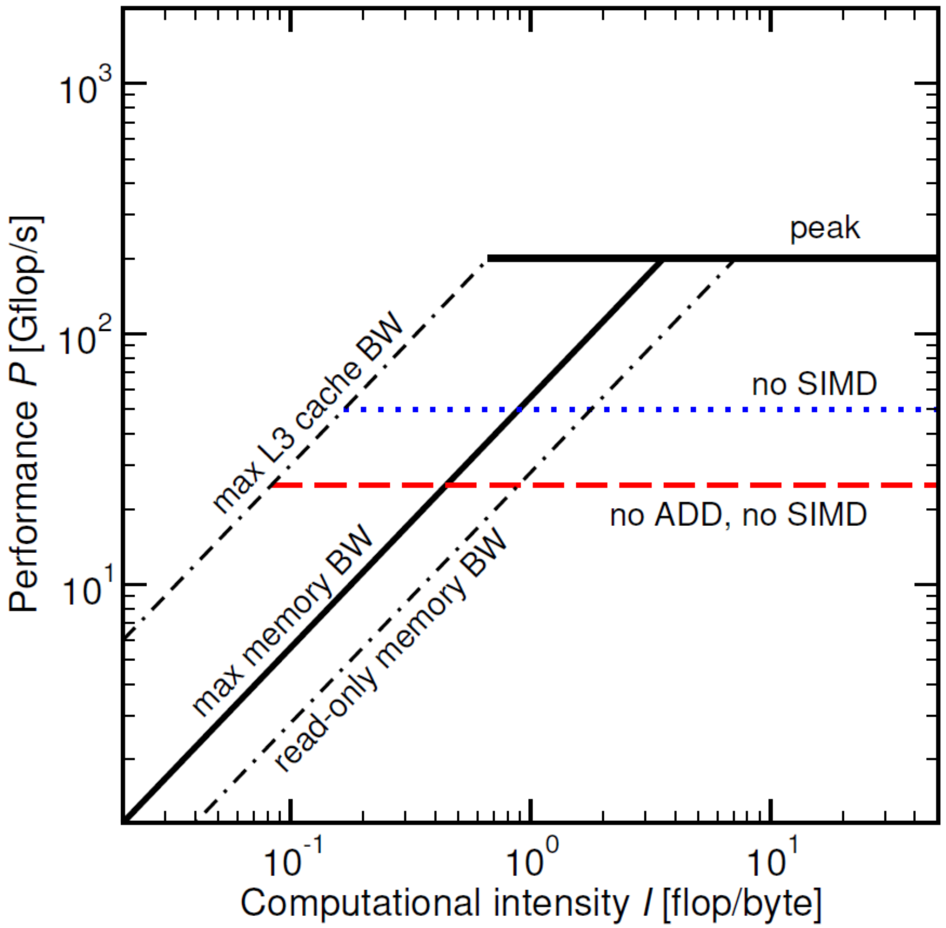
Refined Roofline model: graphical representation

Multiple ceilings may apply

- Different bandwidths /data paths
  → different inclined ceilings

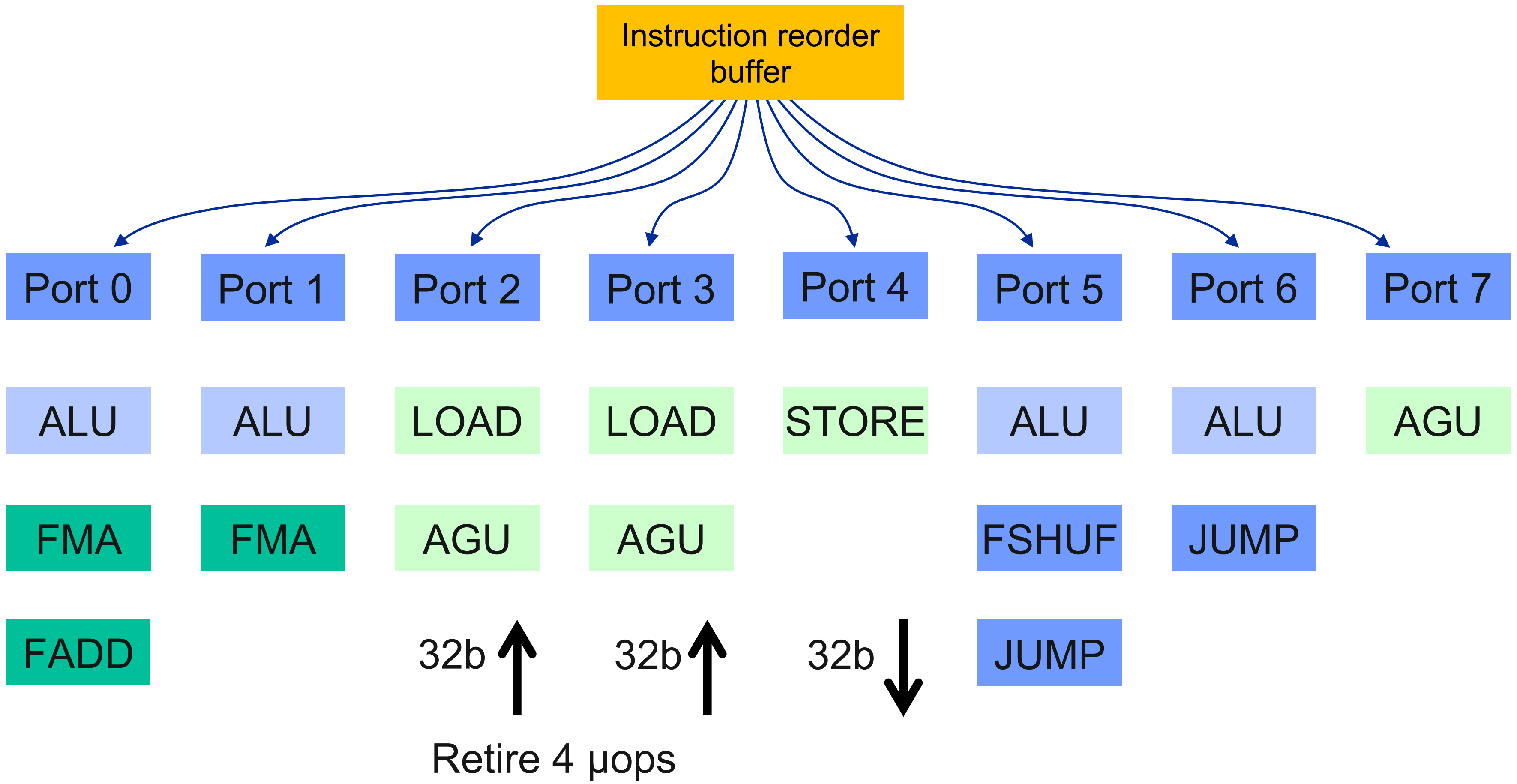
- Different P_{max}
  → different flat ceilings

  In fact, P_{max} should always come from code analysis; generic ceilings are usually impossible to attain



Estimating per-core P_{max} on a given architecture

Haswell/Broadwell port scheduler model:



Haswell/Broadwell

Example: P_{max} of vector triad on Haswell

```
double  *A, *B, *C, *D;
for (int i=0; i<N; i++) {
   A[i] = B[i] + C[i] * D[i];
}
```

Minimum number of cycles to process **one AVX-vectorized iteration** (equivalent to 4 scalar iterations) on one core?

→ Assuming full throughput:

Cycle 1: LOAD + LOAD + STORE
Cycle 2: LOAD + LOAD + FMA + FMA
Cycle 3: LOAD + LOAD + STORE

Answer: 1.5 cycles

Example: P_{max} of vector triad on Haswell@2.3

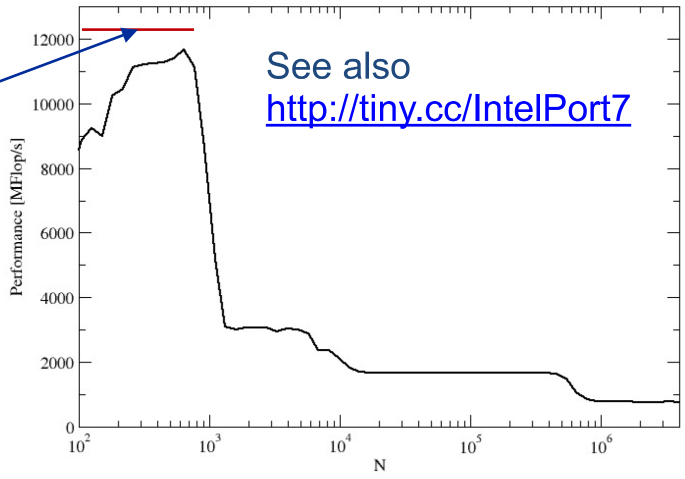
```
double  *A, *B, *C, *D;
for (int i=0; i<N; i++) {
   A[i] = B[i] + C[i] * D[i];
}
```

What is the **performance in GFlops/s per core** and the bandwidth in GBytes/s?

One AVX iteration (1.5 cycles) does 4 x 2 = 8 flops:

$$2.3 \cdot 10^9 \text{ cy/s} \cdot \frac{8 \text{ flops}}{1.5 \text{ cy}} = \mathbf{12.27} \frac{\mathbf{Gflops}}{\mathbf{s}}$$

$$12.27 \frac{\text{Gflops}}{\text{s}} \cdot 16 \frac{\text{bytes}}{\text{flop}} = 196 \frac{\text{Gbyte}}{\text{s}}$$



See also http://tiny.cc/IntelPort7

$P_{\max}$ + bandwidth limitations: The vector triad

Vector triad `A(:)=B(:)+C(:)*D(:)` on a 2.3 GHz 14-core Haswell chip

Consider full chip (14 cores):

Memory bandwidth: b_S = **50 GB/s**

Code balance (incl. write allocate):
B_c = (4+1) Words / 2 Flops = 20 B/F → ***I* = 0.05 F/B**

→ ***I* · b_S = 2.5 GF/s** (0.5% of peak performance)

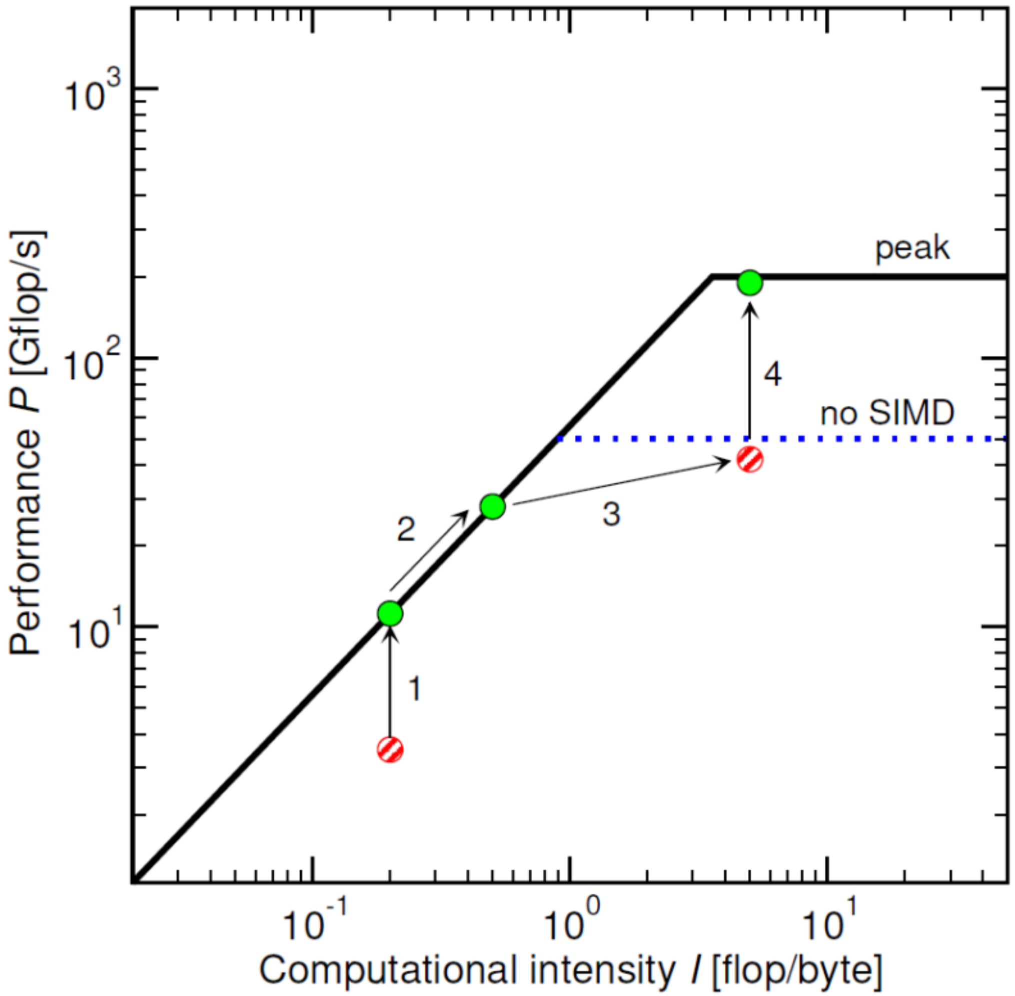
P_{peak} / core = 36.8 Gflop/s ((8+8) Flops/cy x 2.3 GHz)
$P_{\max}$ / core = 12.27 Gflop/s (see prev. slide)

→ **$P_{\max}$ = 14 * 12.27 Gflop/s =172 Gflop/s** (33% peak)

$$\boxed{P = \min(P_{\max}, I \cdot b_S) = \min(172{,}2.5)\,\text{GFlop/s} = 2.5\,\text{GFlop/s}}$$

Tracking code optimizations in the Roofline Model

1. Hit the BW bottleneck by good serial code
   (e.g., Ninja C++ → Fortran)
2. Increase intensity to make better use of BW bottleneck
   (e.g., spatial loop blocking [see later])
3. Increase intensity and go from memory bound to core bound
   (e.g., temporal blocking)
4. Hit the core bottleneck by good serial code
   (e.g., `-fno-alias` [see later])



Factors to consider in the Roofline Model

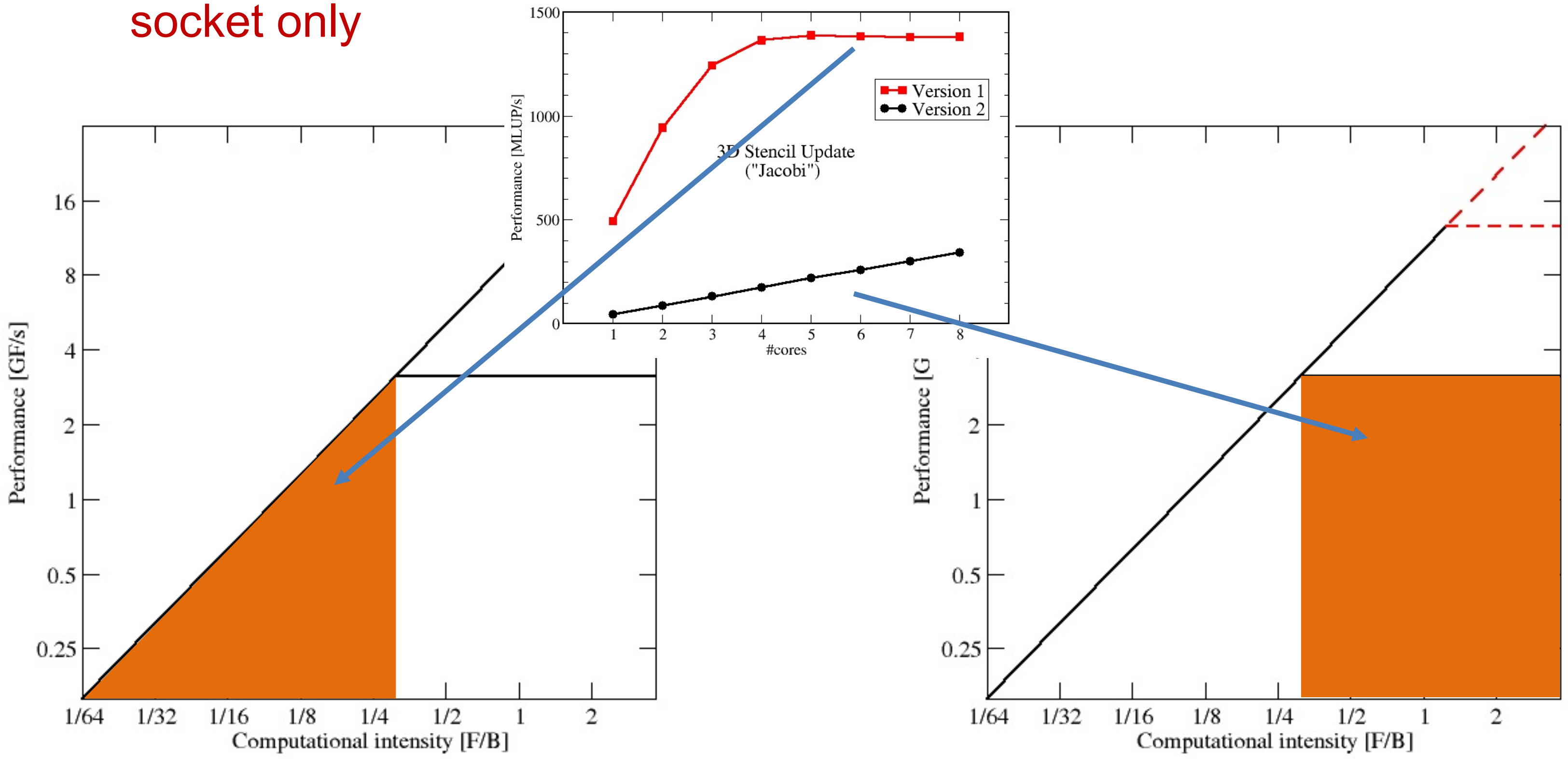
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Bandwidth-bound (simple case)

1. Accurate traffic calculation (write-allocate, strided access, …)
2. Practical ≠ theoretical BW limits
3. Saturation effects → consider full socket only

Core-bound (may be complex)

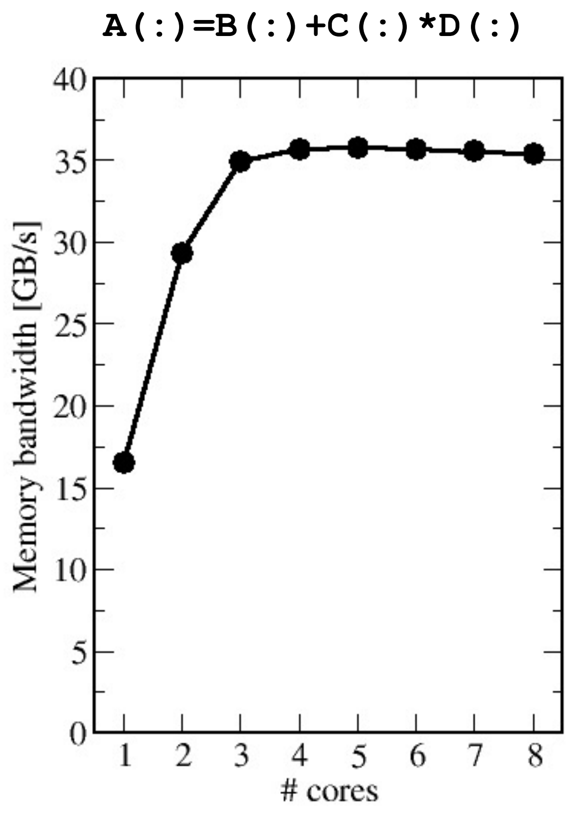
1. Multiple bottlenecks: LD/ST, arithmetic, pipelines, SIMD, execution ports
2. Limit is linear in # of cores



Shortcomings of the roofline model

- **Saturation effects in multicore chips are not explained**
  - Reason: “saturation assumption”
  - Cache line transfers and core execution do sometimes not overlap perfectly
  - It is not sufficient to measure single-core STREAM to make it work
  - Only increased “pressure” on the memory interface can saturate the bus
    → need more cores!
- **In-cache performance is not correctly predicted**
- **The ECM performance model gives more insight:**

  G. Hager, J. Treibig, J. Habich, and G. Wellein: Exploring performance and power properties of modern multicore chips via simple machine models. Concurrency and Computation: Practice and Experience (2013).
  [DOI: 10.1002/cpe.3180](https://doi.org/10.1002/cpe.3180) Preprint: [arXiv:1208.2908](https://arxiv.org/abs/1208.2908)


A(:)=B(:)+C(:)*D(:)
Memory bandwidth [GB/s]
# cores