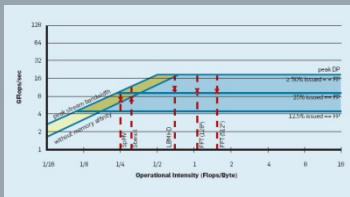


“Simple” performance modeling: The Roofline Model

Loop-based performance modeling: Execution vs. data transfer

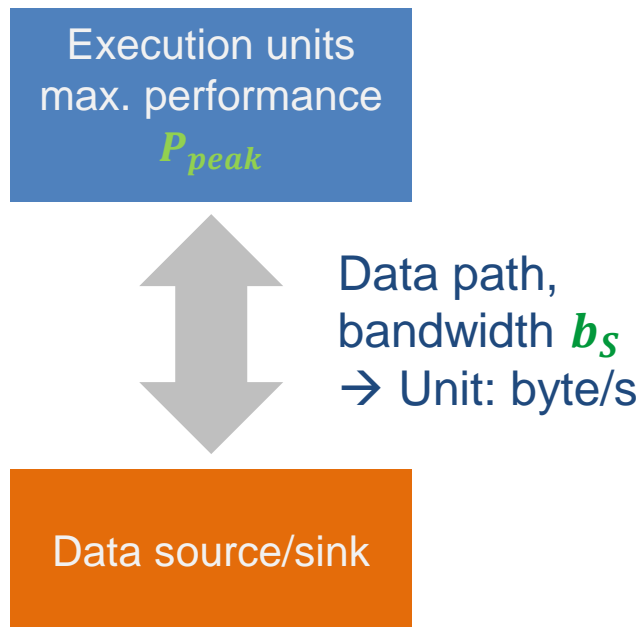


R.W. Hockney and I.J. Curington: $f_{1/2}$: A parameter to characterize memory and communication bottlenecks. *Parallel Computing* 10, 277-286 (1989). [DOI: 10.1016/0167-8191\(89\)90100-2](https://doi.org/10.1016/0167-8191(89)90100-2)

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/ECS-2008-164. PhD thesis (2008)

Simplistic view of the hardware:



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

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

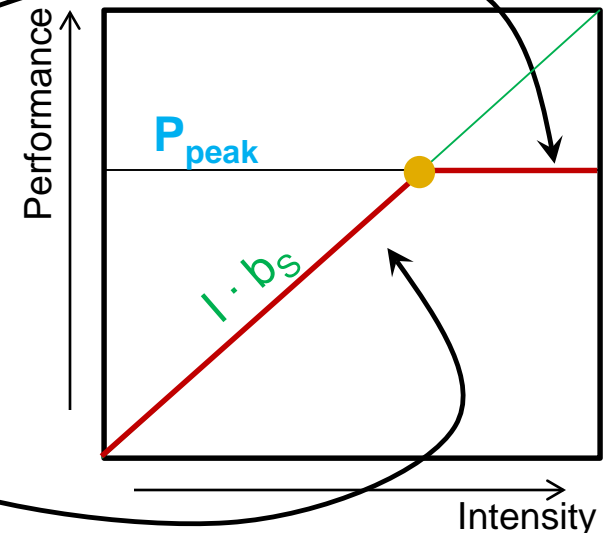
The bottleneck is either

- 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 $P_{\text{max}} = I \cdot b_S$:
Best use of resources
- Roofline is an “optimistic” model
 (“light speed”)



Apply the naive Roofline model in practice

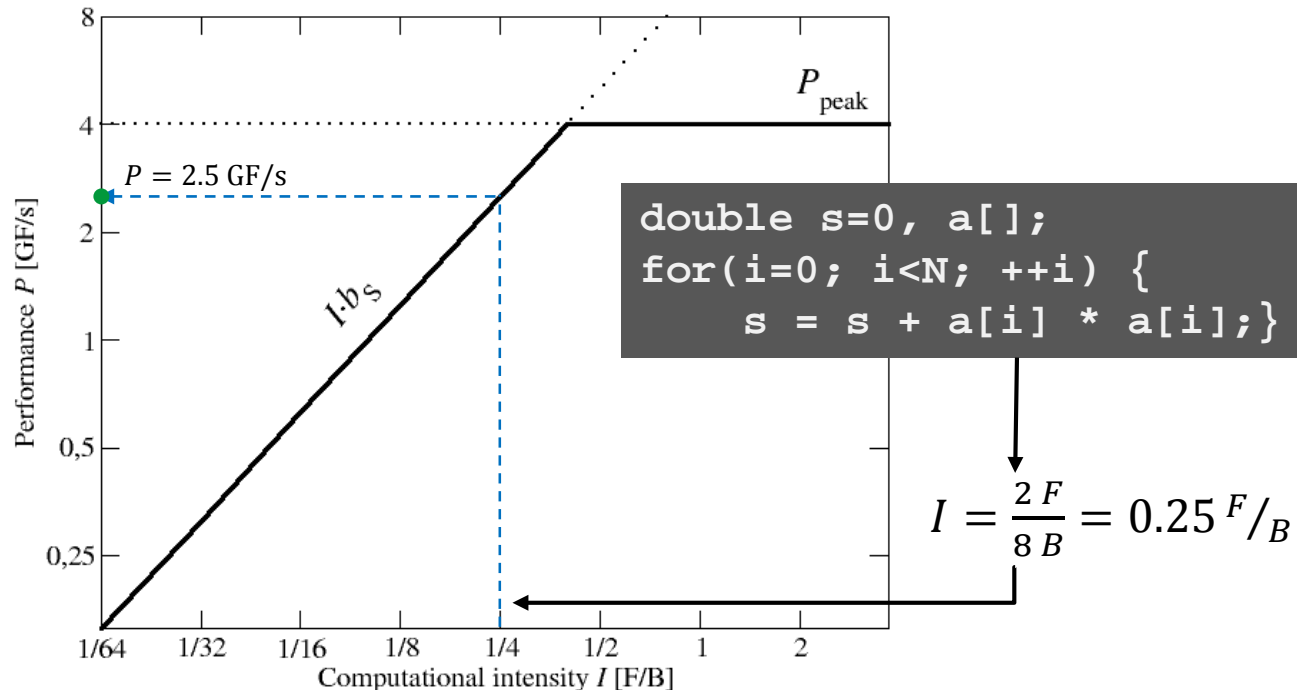
- Machine parameter #1: Peak performance: $P_{peak} \left[\frac{F}{s} \right]$
- Machine parameter #2: Memory bandwidth: $b_S \left[\frac{B}{s} \right]$
- Code characteristic: Computational intensity: $I \left[\frac{F}{B} \right]$

Machine properties:

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

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

Application property: I



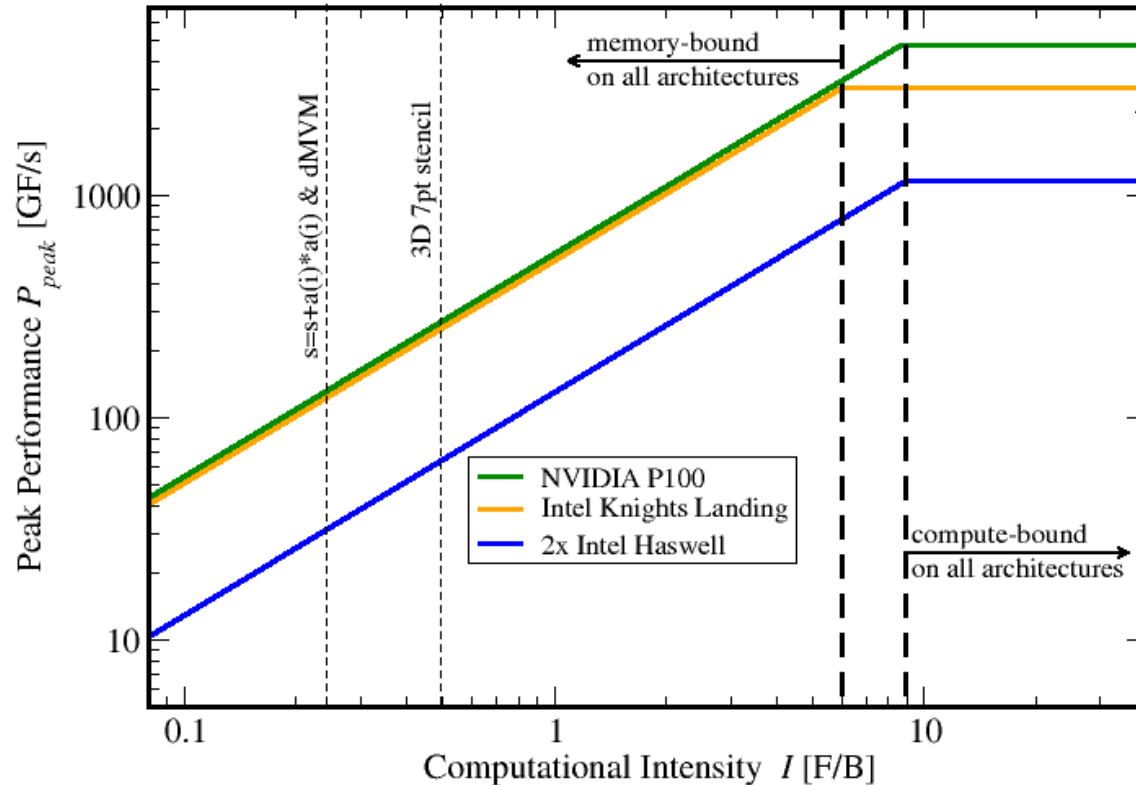
- **The roofline formalism is based on some (crucial) prerequisites:**
 - There is a clear concept of “work” vs. “traffic”
 - “work” = flops, updates, iterations...
 - “traffic” = required data to do “work”
 - **Machine input parameters: Peak Performance and Peak Bandwidth**
Application/kernel is expected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



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., $P_{peak} \rightarrow P_{peak}/2$)

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:

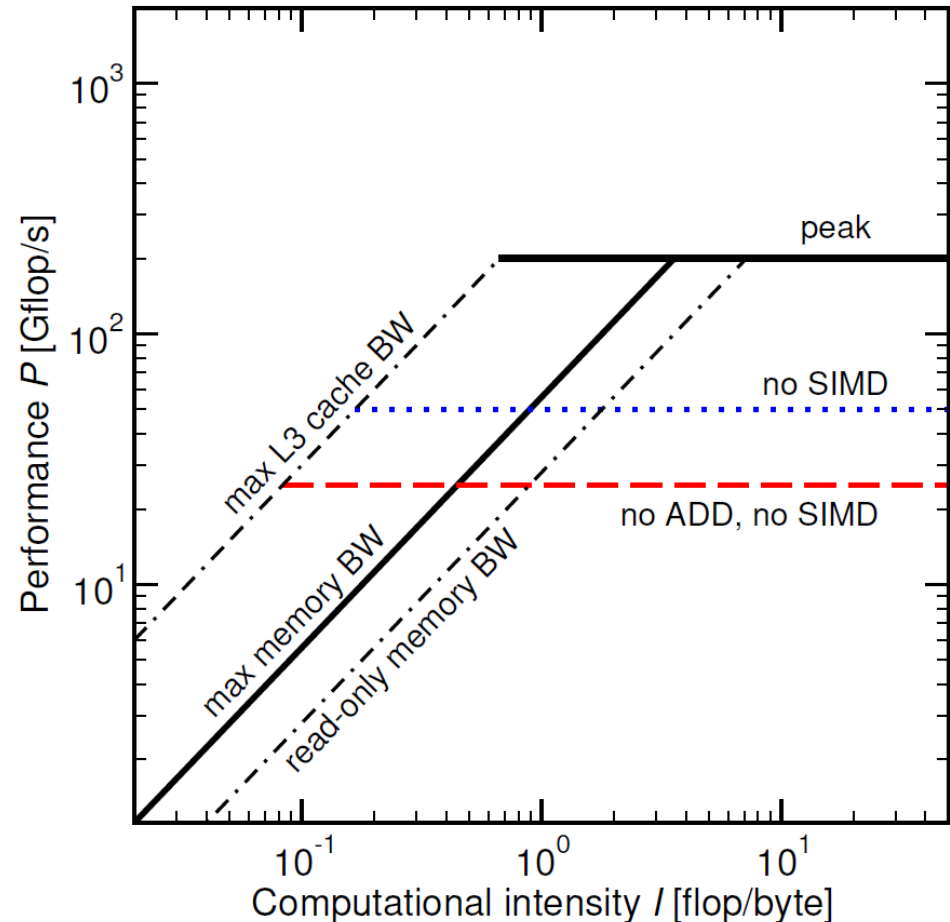
$$P = \min(P_{\max}, I \cdot b_S) = \min\left(P_{\max}, \frac{b_S}{B_C}\right)$$

[Byte/s] (connected to b_S)
[Byte/Flop] (connected to B_C)

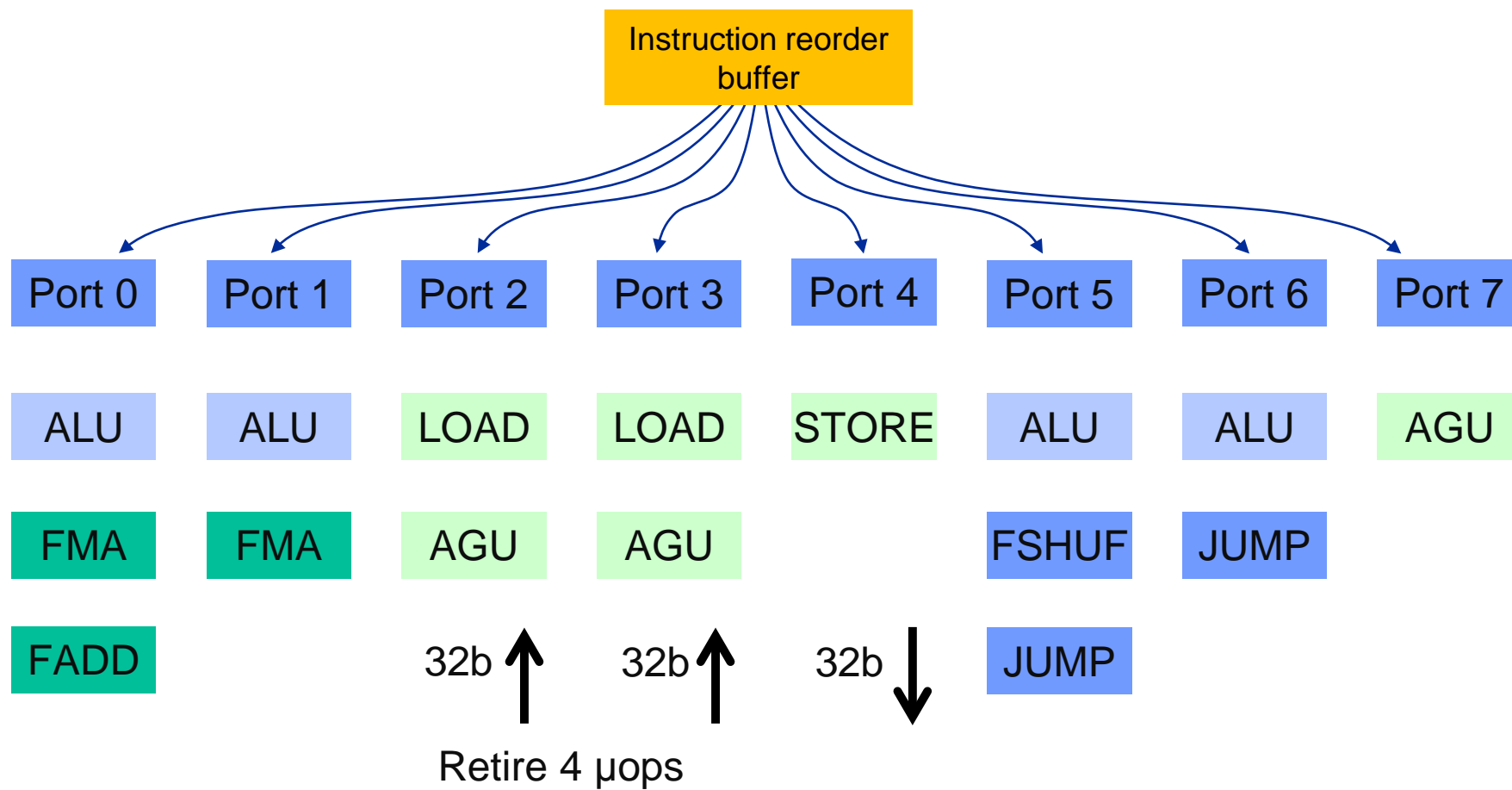
Multiple ceilings may apply

- 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



Haswell/Broadwell port scheduler model:



Haswell/Broadwell

```
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

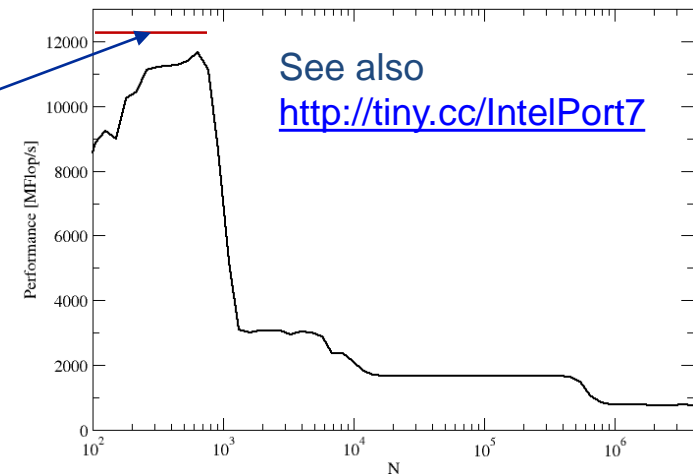
```
double *A, *B, *C, *D;
for (int i=0; i<N; i++) {
    A[i] = B[i] + C[i] * D[i];
}
```

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

One AVX iteration (1.5 cycles) does $4 \times 2 = 8$ flops:

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

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



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

Consider full chip (14 cores):

Memory bandwidth: $b_S = 50 \text{ GB/s}$

Code balance (incl. write allocate):

$B_c = (4+1) \text{ Words} / 2 \text{ Flops} = 20 \text{ B/F} \rightarrow I = 0.05 \text{ F/B}$

$\rightarrow I \cdot b_S = 2.5 \text{ GF/s}$ (0.5% of peak performance)

$P_{\text{peak}} / \text{core} = 36.8 \text{ Gflop/s}$ ((8+8) Flops/cy x 2.3 GHz)

$P_{\text{max}} / \text{core} = 12.27 \text{ Gflop/s}$ (see prev. slide)

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

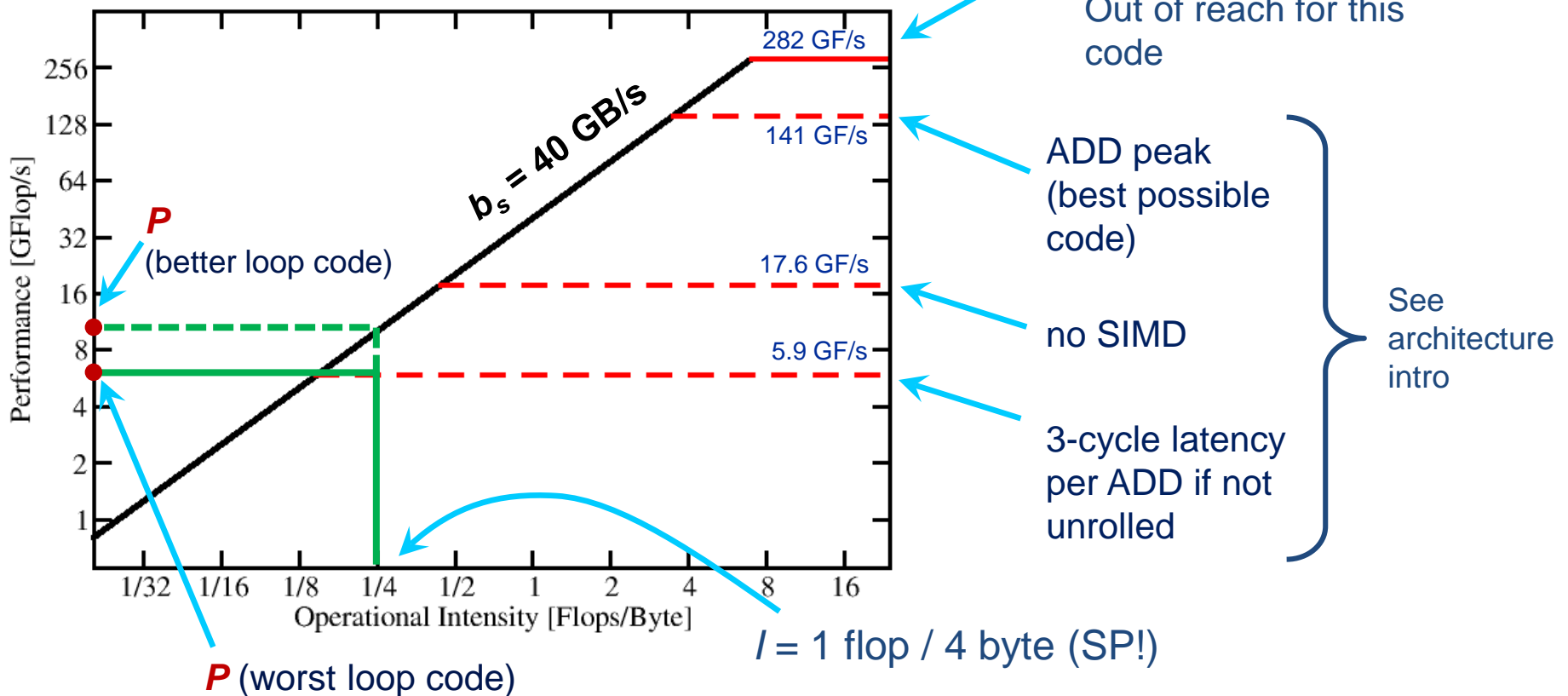
$$P = \min(P_{\text{max}}, I \cdot b_S) = \min(172, 2.5) \text{ GFlop/s} = 2.5 \text{ GFlop/s}$$

A not so simple Roofline example

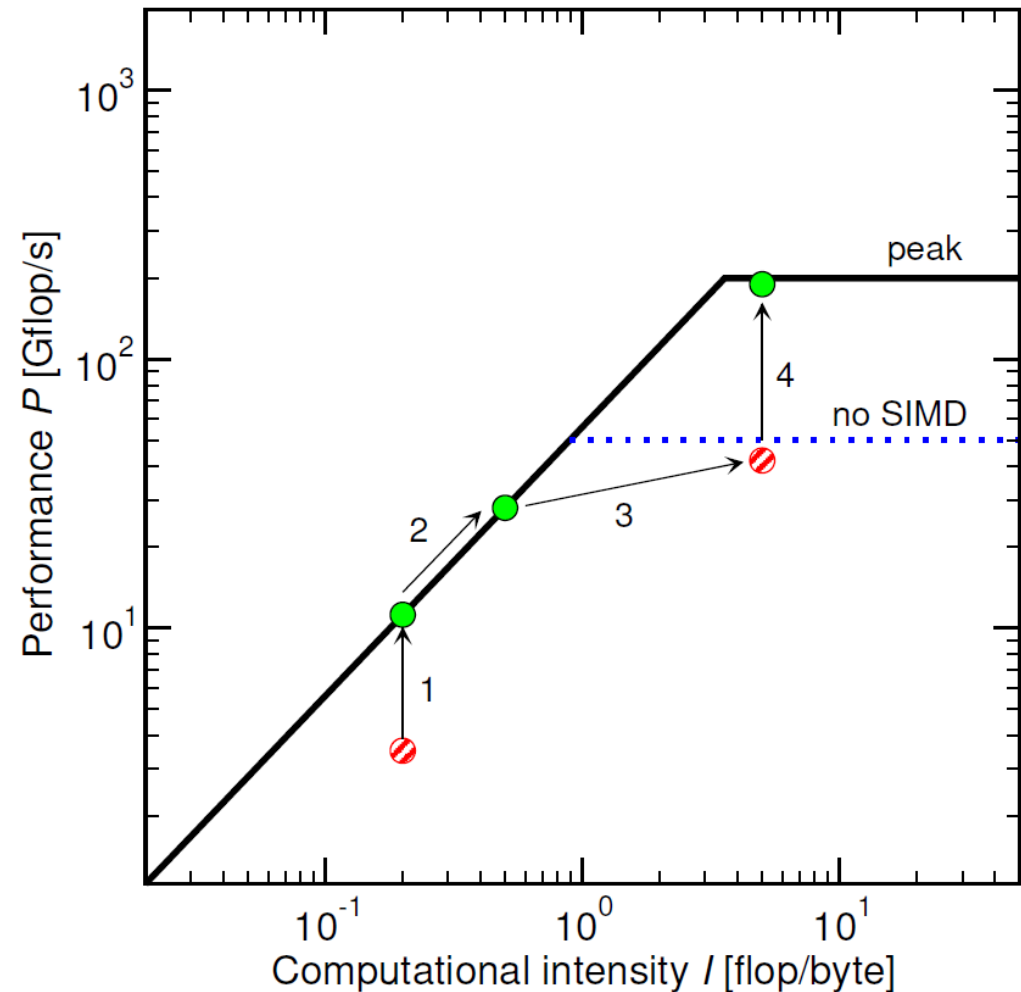
Example: `do i=1,N; s=s+a(i); enddo`

in **single precision** on an **8-core 2.2 GHz** Sandy Bridge socket @ “large” N

$$P = \min(P_{\max}, I \cdot b_s)$$

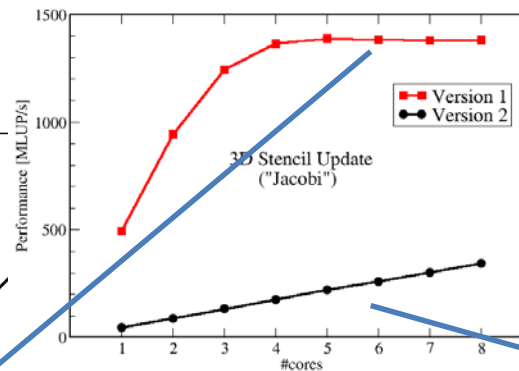
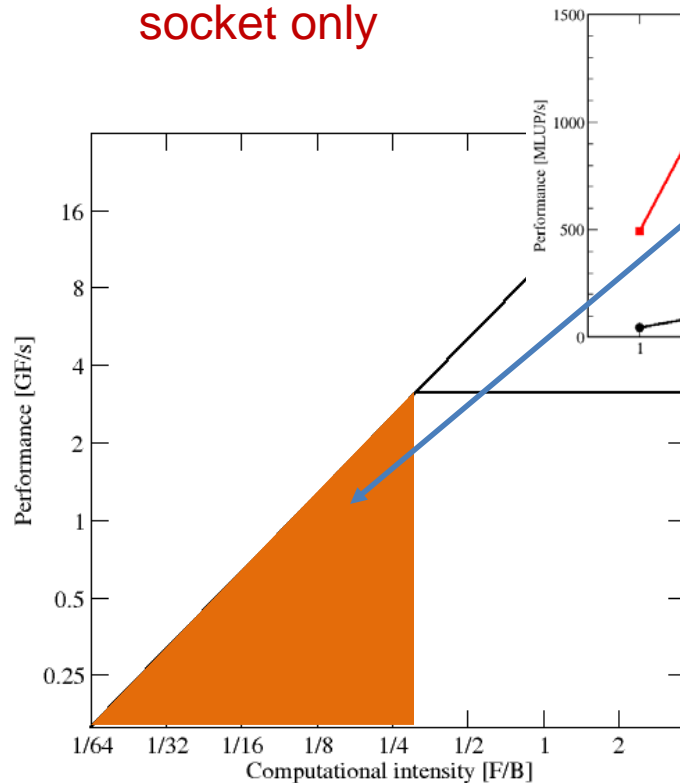


1. Hit the BW bottleneck by good serial code
(e.g., Ninja C++ → Fortran)
2. Increase intensity to make better use of BW bottleneck
(e.g., spatial loop blocking [see later])
3. Increase intensity and go from memory bound to core bound
(e.g., temporal blocking)
4. Hit the core bottleneck by good serial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



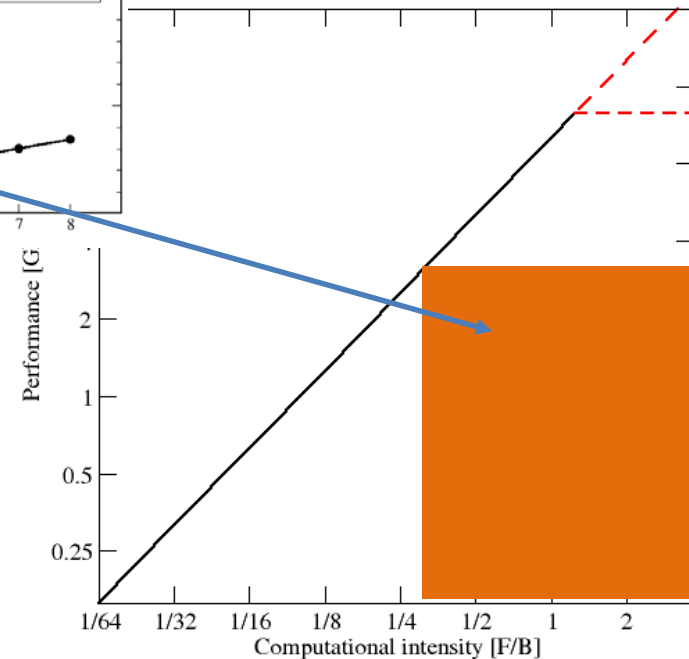
Bandwidth-bound (simple case)

1. Accurate traffic calculation (write-allocate, strided access, ...)
2. Practical \neq theoretical BW limits
3. Saturation effects \rightarrow 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



- **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)

