“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)

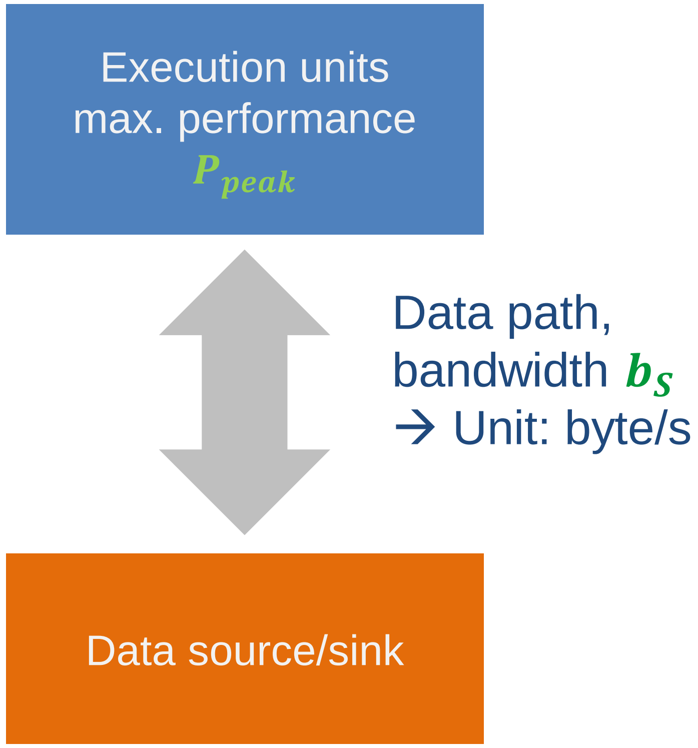
S. Williams: Auto-tuning Performance on Multicore Computers. UCB Technical Report No. UCB/EECS-2008-164. PhD thesis (2008)

Analytic white-box performance models

An analytic white-box performance model is a simplified mathematical description of the hardware and its interaction with software. It is able to predict the runtime/performance of code from “first principles.”

A simple performance model for loops

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

Naïve Roofline Model

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

The bottleneck is either

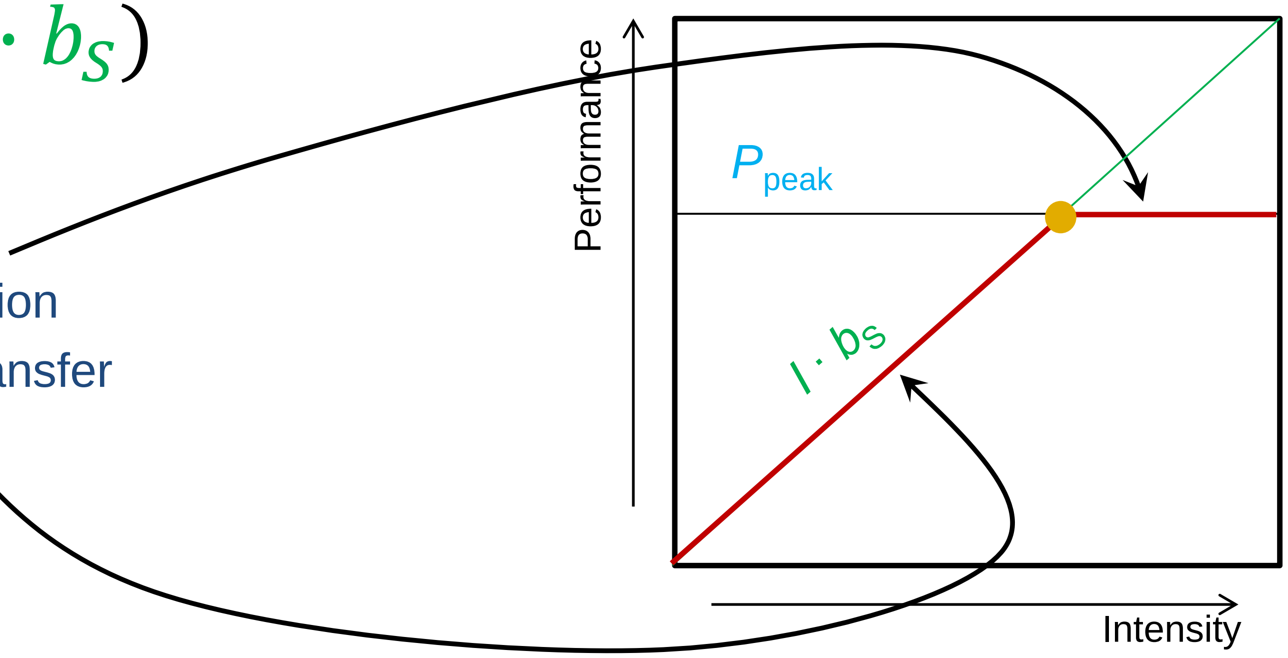
- The execution of work: P_{peak} [flop/s]
- The data path: $I \cdot b_S$ [flop/byte x byte/s]

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

This is the "Naïve Roofline Model"

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

- Roofline is an "optimistic" model
  (think "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 model

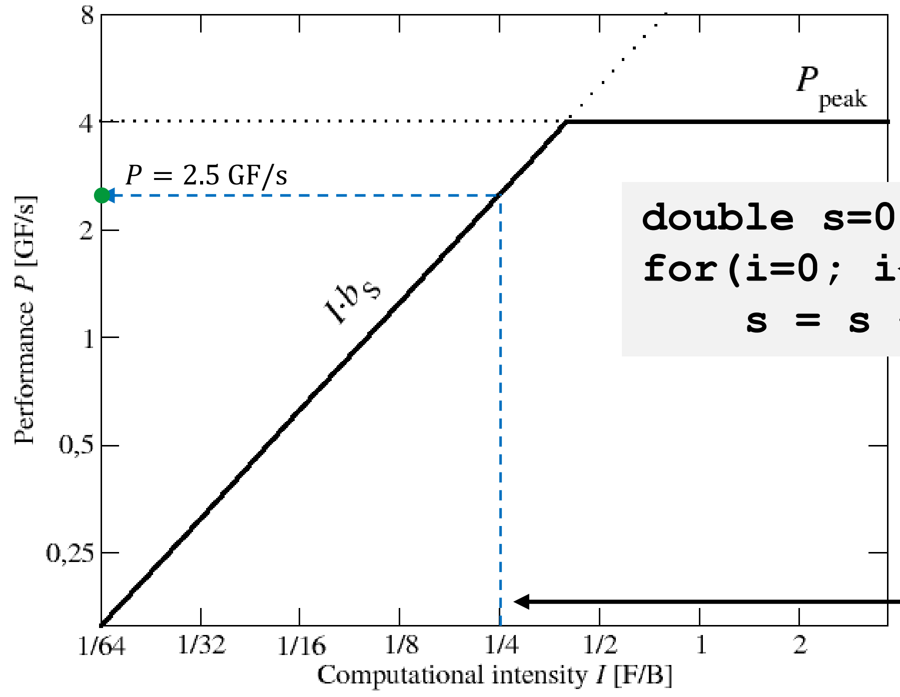
Application model

Machine properties:

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

$$\boldsymbol{b_S} = 10\,\frac{\mathrm{GB}}{\mathrm{s}}$$

Application property: I



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

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

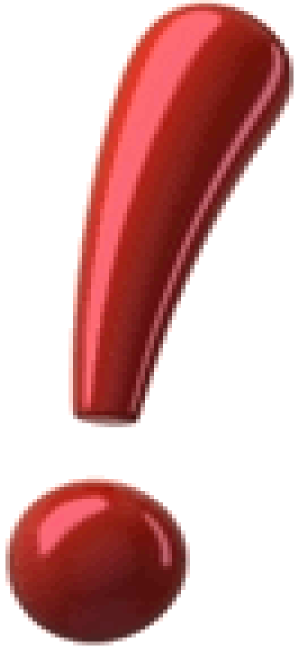
Prerequisites for the Roofline Model

- 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
  - If two bottlenecks are "close," no interaction is assumed

- Data access latency is ignored, i.e. perfect streaming mode
  - Achievable bandwidth is the limit

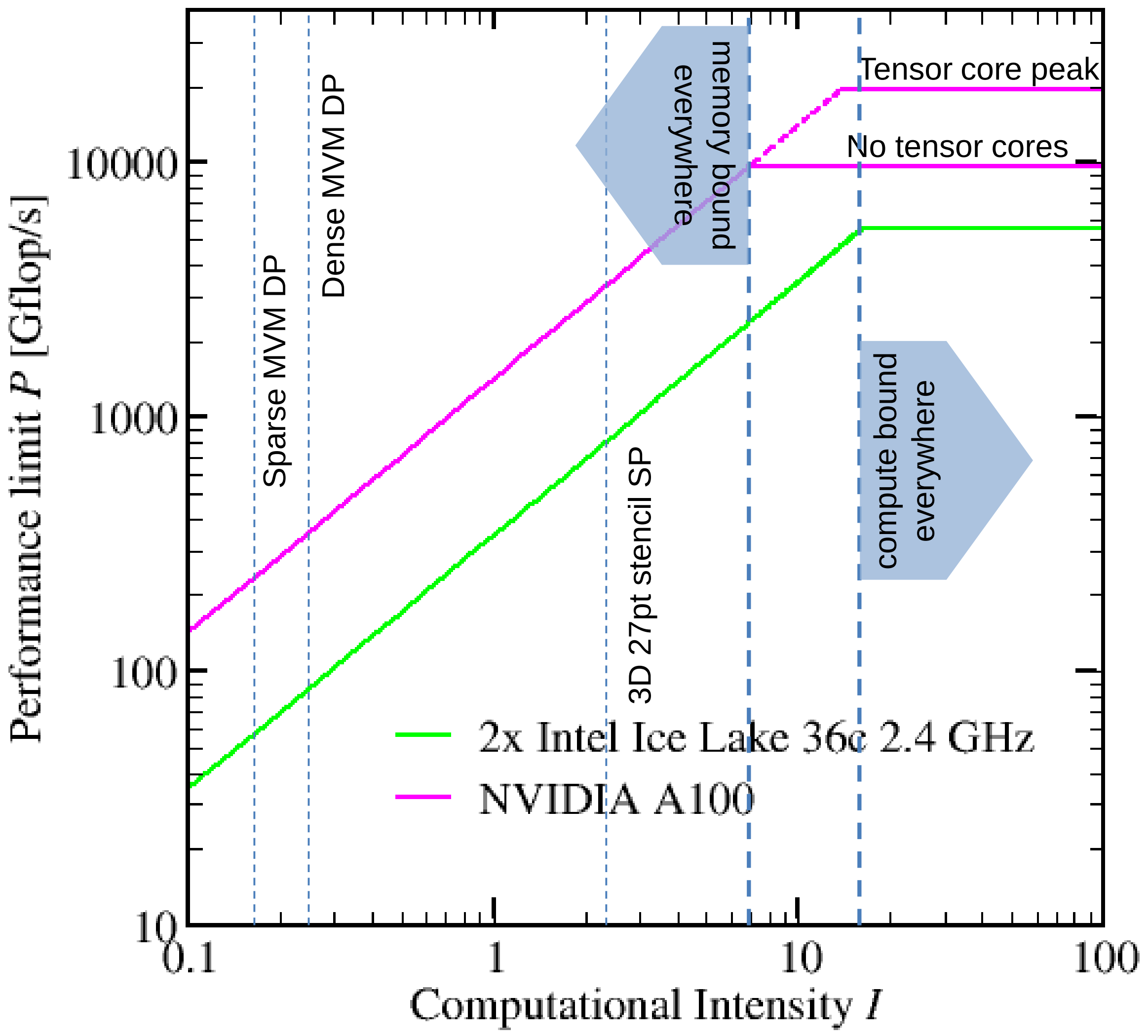
- Chip must be able to saturate the bandwidth bottleneck(s)
  - Always model the full chip



Roofline for architecture and code comparison

With Roofline, we can

- Compare capabilities of different machines
- Compare performance expectations for different loops



- Roofline always provides upper bound – but is it realistic?
  - Simple case: Loop kernel has loop-carried dependecncies → cannot achieve peak
  - Other bandwidth bottlenecks may apply

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

3. 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

"Flop" is not the only useful unit of work!

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

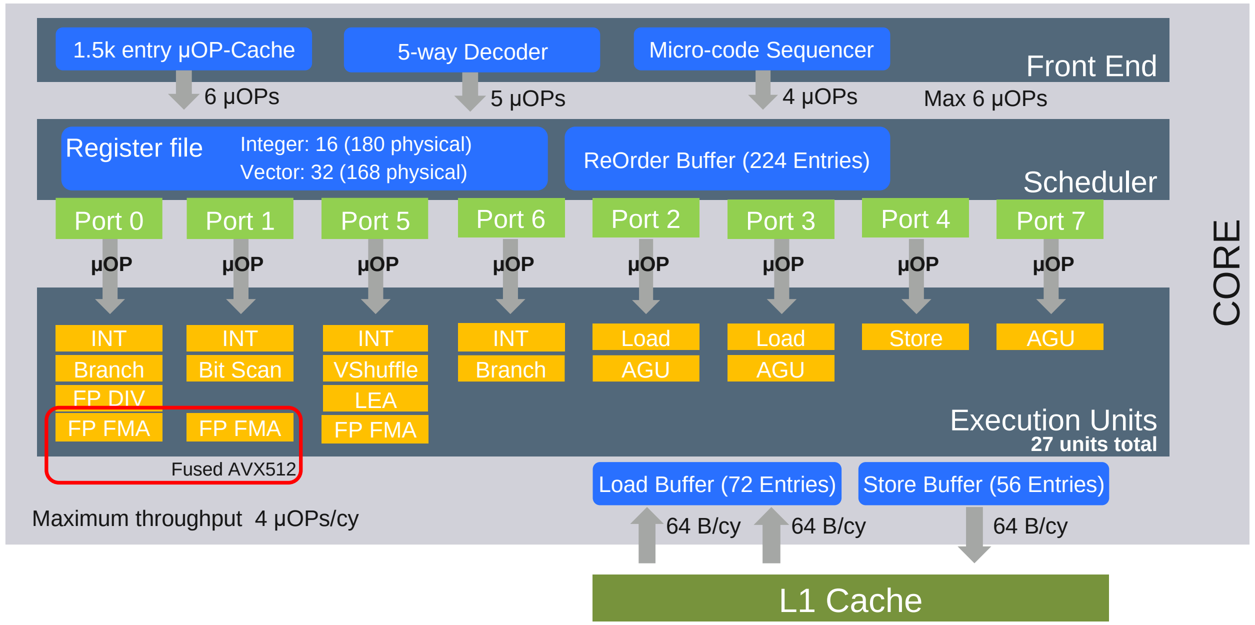
[Byte/s]

[Byte/Flop]

Complexities of in-core execution (P_{max})

Multiple bottlenecks:

- Decode/retirement throughput
- Port contention (direct or indirect)
- Arithmetic pipeline stalls (dependencies)
- Overall pipeline stalls (branching)
- L1 Dcache bandwidth (LD/ST throughput)
- Scalar vs. SIMD execution
- L1 Icache (LD/ST) bandwidth
- Alignment issues
- …



Skylake

Tool for P_{max} analysis: OSACA
http://tiny.cc/OSACA
DOI: 10.1109/PMBS49563.2019.00006
DOI: 10.1109/PMBS.2018.8641578

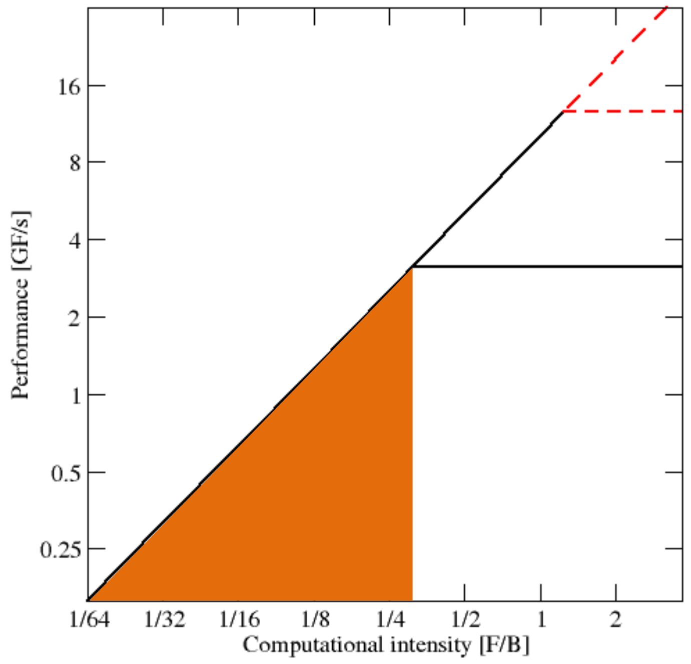
Factors to consider in the Roofline Model

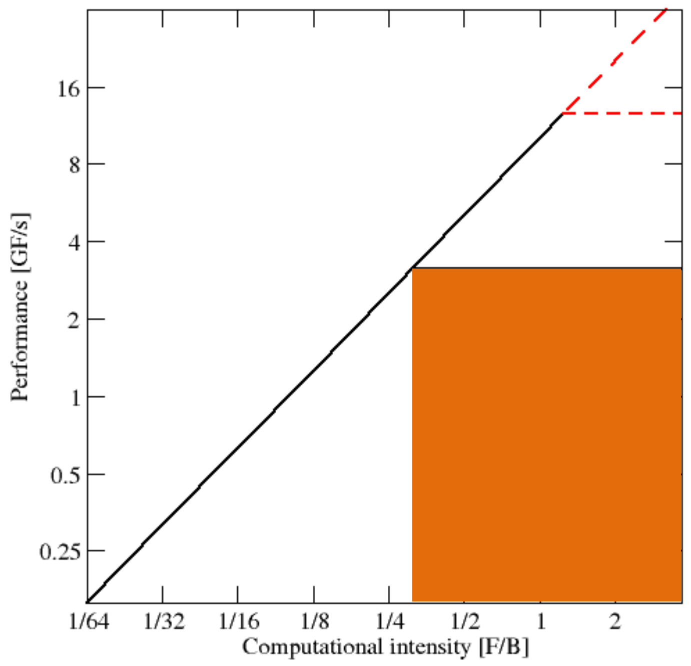
Bandwidth-bound (simple case)

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





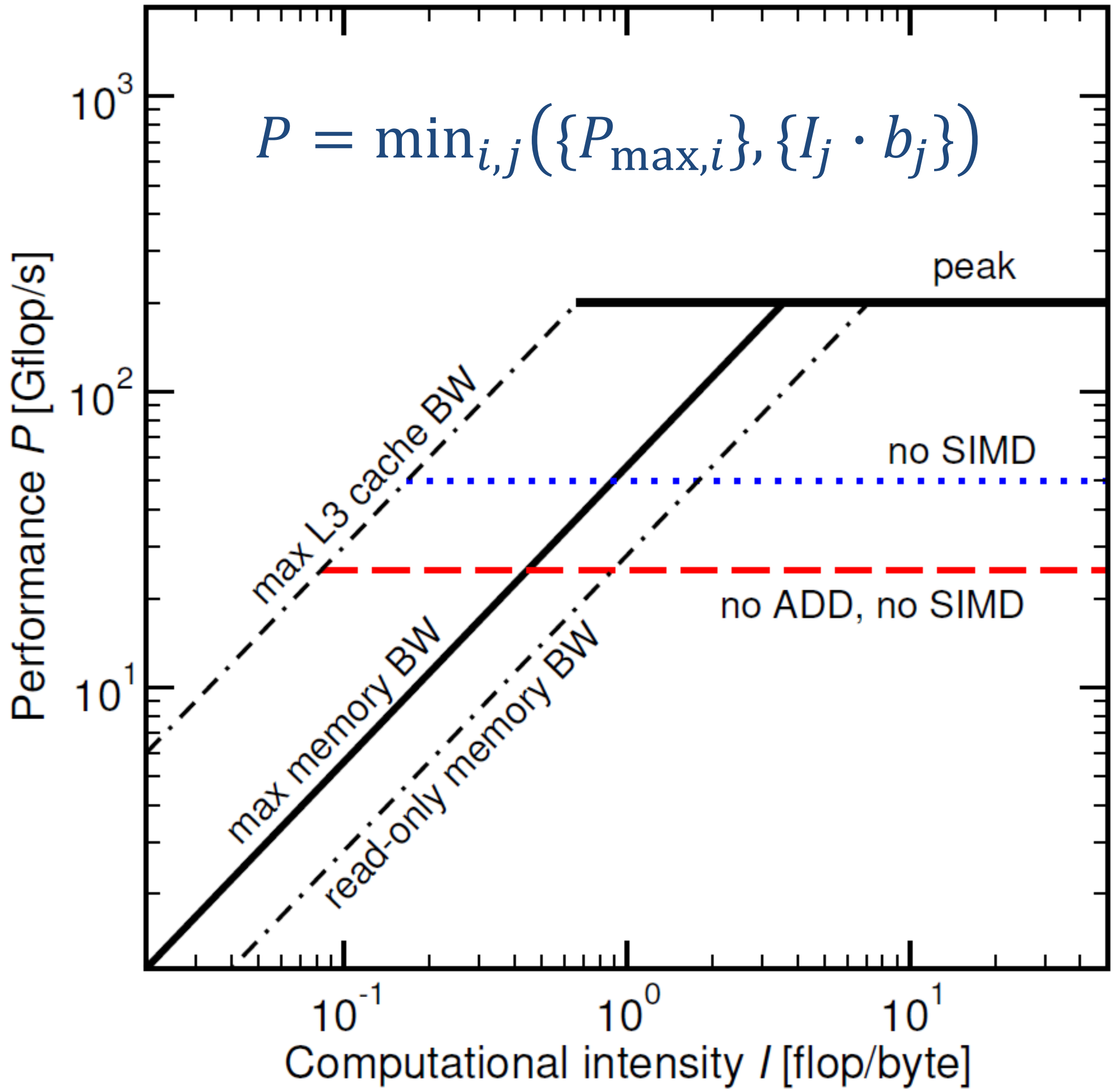
Refined Roofline model: graphical representation

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



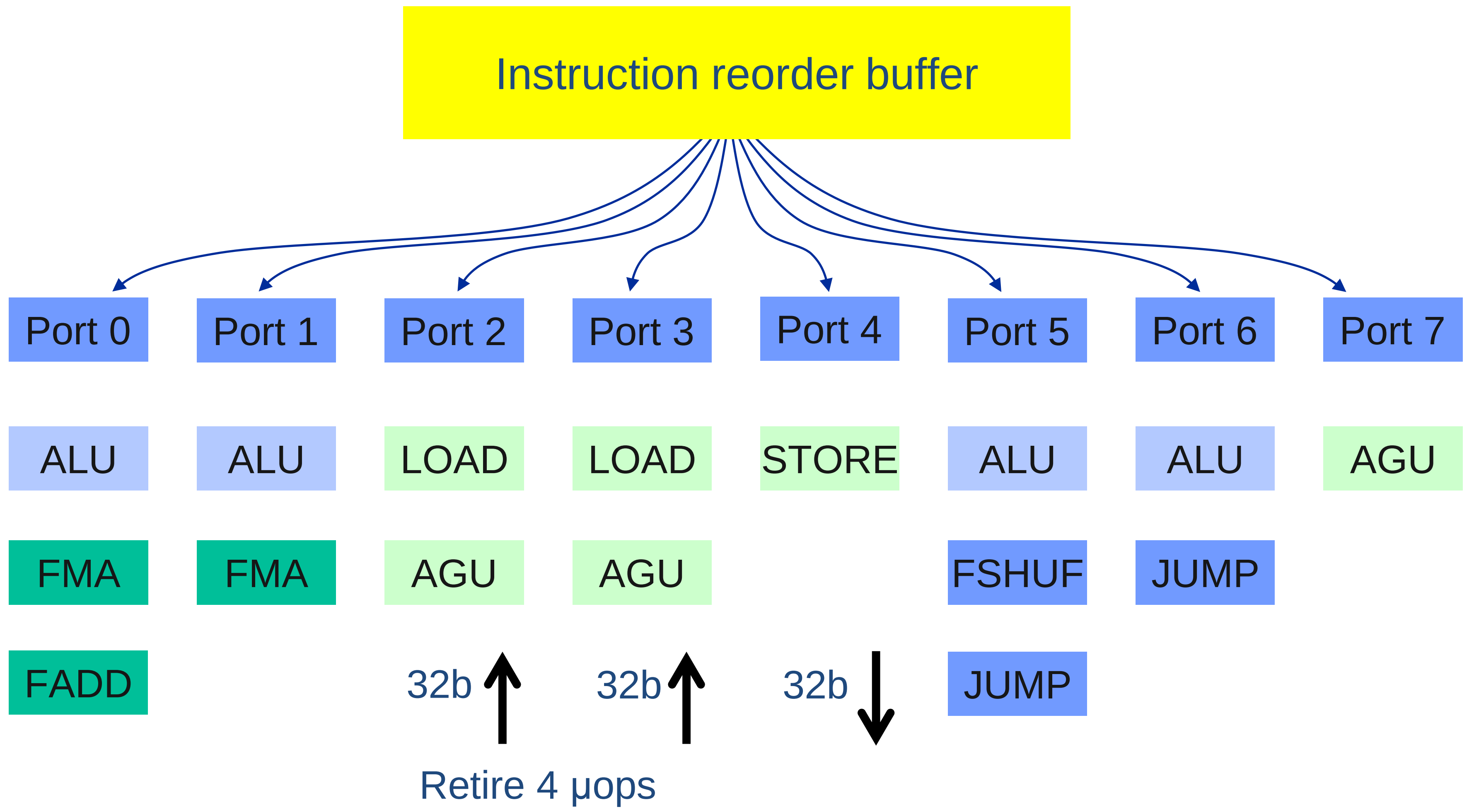
Hardware features of (some) Intel Xeon processors

Microarchitecture	Ivy Bridge EP	Broadwell EP	Cascade Lake SP	Ice Lake SP
Introduced	09/2013	03/2016	04/2019	06/2021
Cores	≤ 12	≤ 22	≤ 28	≤ 40
LD/ST throughput per cy:				
AVX(2), AVX512	1 LD + ½ ST	2 LD + 1 ST	2 LD + 1 ST	2 LD + 1 ST
SSE/scalar	2 LD || 1 LD & 1 ST			
ADD throughput	1 / cy	1 / cy	2 / cy	2 / cy
MUL throughput	1 / cy	2 / cy	2 / cy	2 / cy
FMA throughput	N/A	2 / cy	2 / cy	2 / cy
L1-L2 data bus	32 B/cy	64 B/cy	64 B/cy	64 B/cy
L2-L3 data bus	32 B/cy	32 B/cy	16+16 B/cy	16+16 B/cy
L1/L2 per core	32 KiB / 256 KiB	32 KiB / 256 KiB	32 KiB / 1 MiB	48 KiB / 1.25 MiB
LLC	2.5 MiB/core inclusive	2.5 MiB/core inclusive	1.375 MiB/core exclusive/victim	1.5 MiB/core exclusive/victim
Memory	4ch DDR3	4ch DDR3	6ch DDR4	8ch DDR4
Memory BW (meas.)	~ 48 GB/s	~ 62 GB/s	~ 115 GB/s	~ 160 GB/s

Source: https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-optimization-reference-manual.html

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

Haswell/Broadwell port scheduler model:



Intel Haswell/Broadwell

Example: P_{max} of vector triad on Haswell/Broadwell

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

Assembly code (AVX2+FMA, no additional unrolling):

```
..B2.9:
  vmovupd      ymm2, [rdx+rax*8]       # LOAD
  vmovupd      ymm1, [r12+rax*8]       # LOAD
  vfmadd213pd ymm1, ymm2, [rbx+rax*8] # LOAD+FMA
  vmovupd      [rdi+rax*8], ymm2       # STORE
  add          rax,4
  cmp          rax,r11
  jb           ..B2.9

# remainder loop omitted
```

Iterations are independent → throughput assumption justified!

Best-case execution time?

Example: P_{max} of vector triad on 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

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

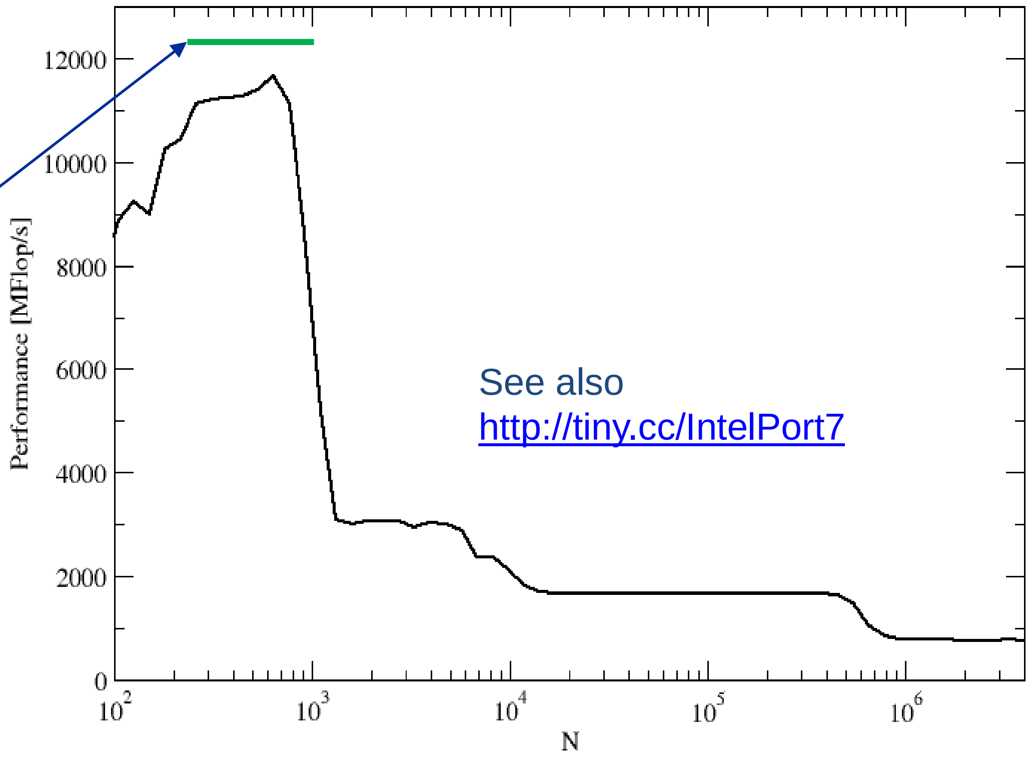
```
double  *A, *B, *C, *D;
for (int i=0; i<N; i++) {
   A[i] = B[i] + C[i] * D[i];
}
```

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:

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

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



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

→ $I \cdot b_S$ = 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)

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

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

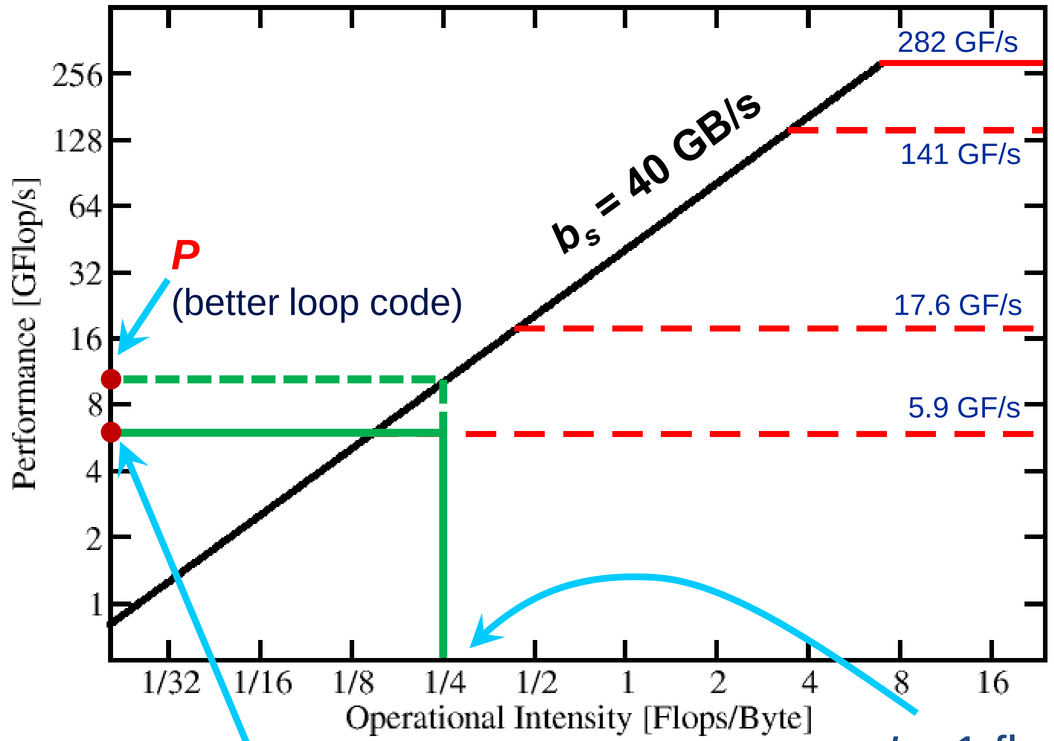
Scalar – can be kept in register

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

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



Machine peak (ADD+MULT) Out of reach for this code

ADD peak (best possible code)

no SIMD

3-cycle latency per ADD if not unrolled

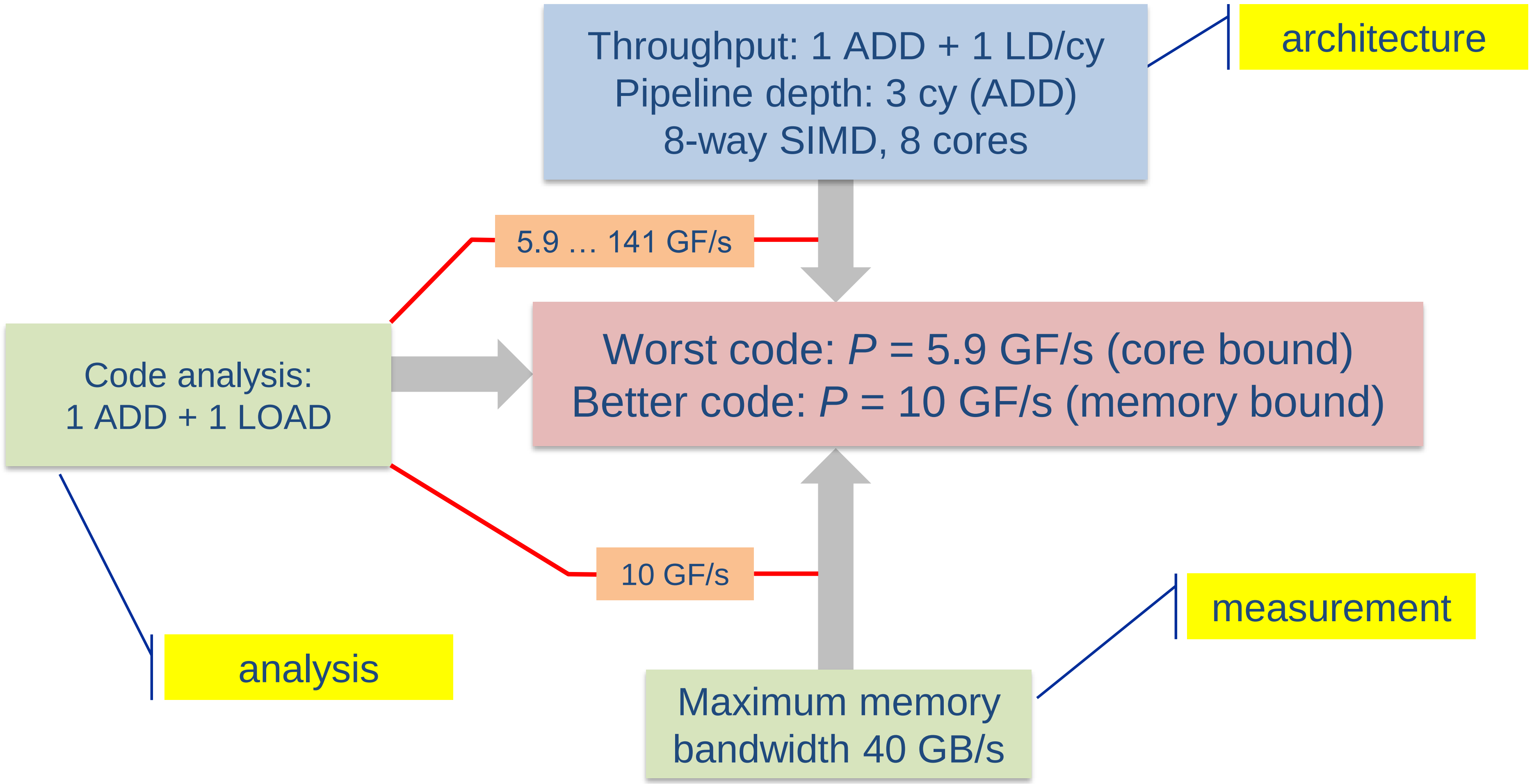
See architecture intro

I = 1 flop / 4 byte (SP!)

P (worst loop code)

Input to the roofline model

… on the example of `do i=1,N; s=s+a(i); enddo`
in single precision


Throughput: 1 ADD + 1 LD/cy
Pipeline depth: 3 cy (ADD)
8-way SIMD, 8 cores
architecture
5.9 … 141 GF/s
Code analysis:
1 ADD + 1 LOAD
Worst code: P = 5.9 GF/s (core bound)
Better code: P = 10 GF/s (memory bound)
10 GF/s
measurement
analysis
Maximum memory bandwidth 40 GB/s

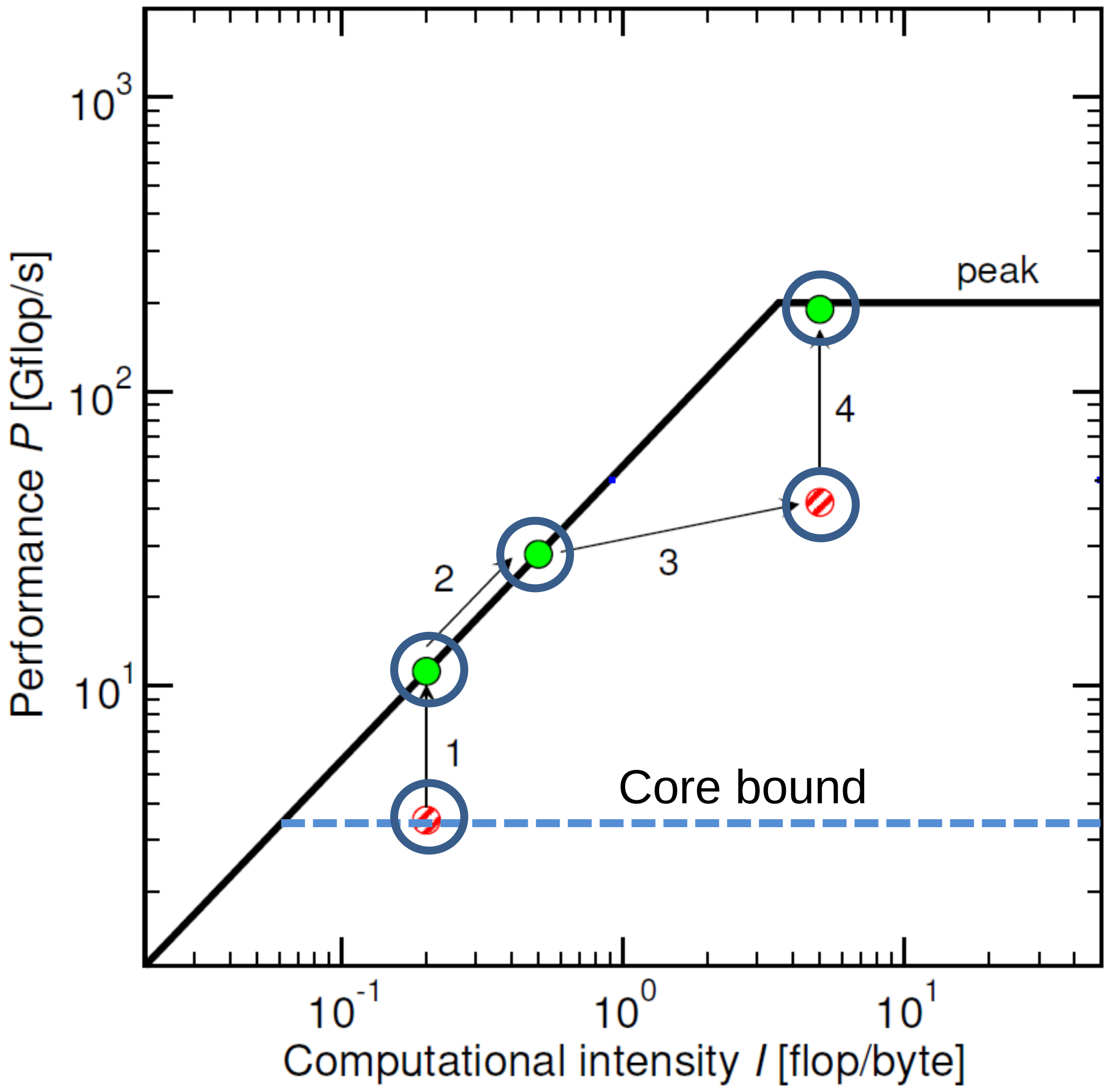
Tracking code optimizations in the Roofline Model

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)

3. Increase intensity and go from memory bound to core bound
   (e.g., temporal blocking)

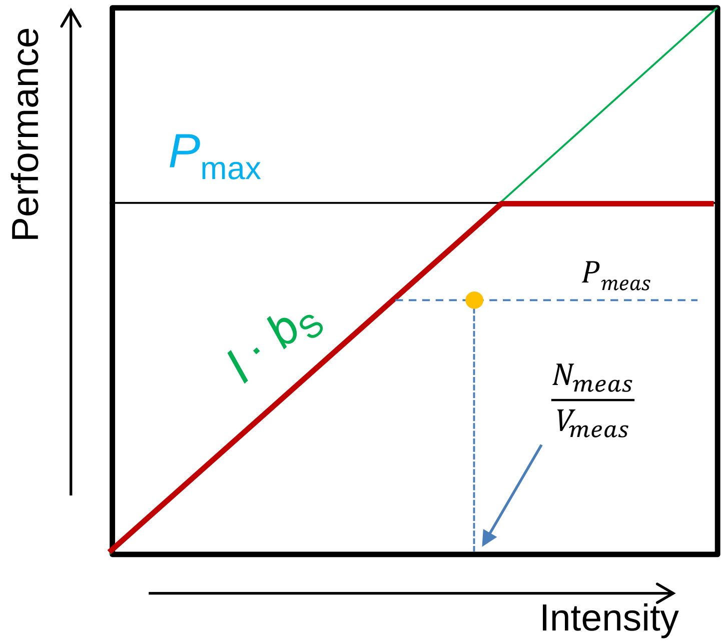
4. Hit the core bottleneck by good serial code
   (e.g., `-fno-alias`, SIMD intrinsics)



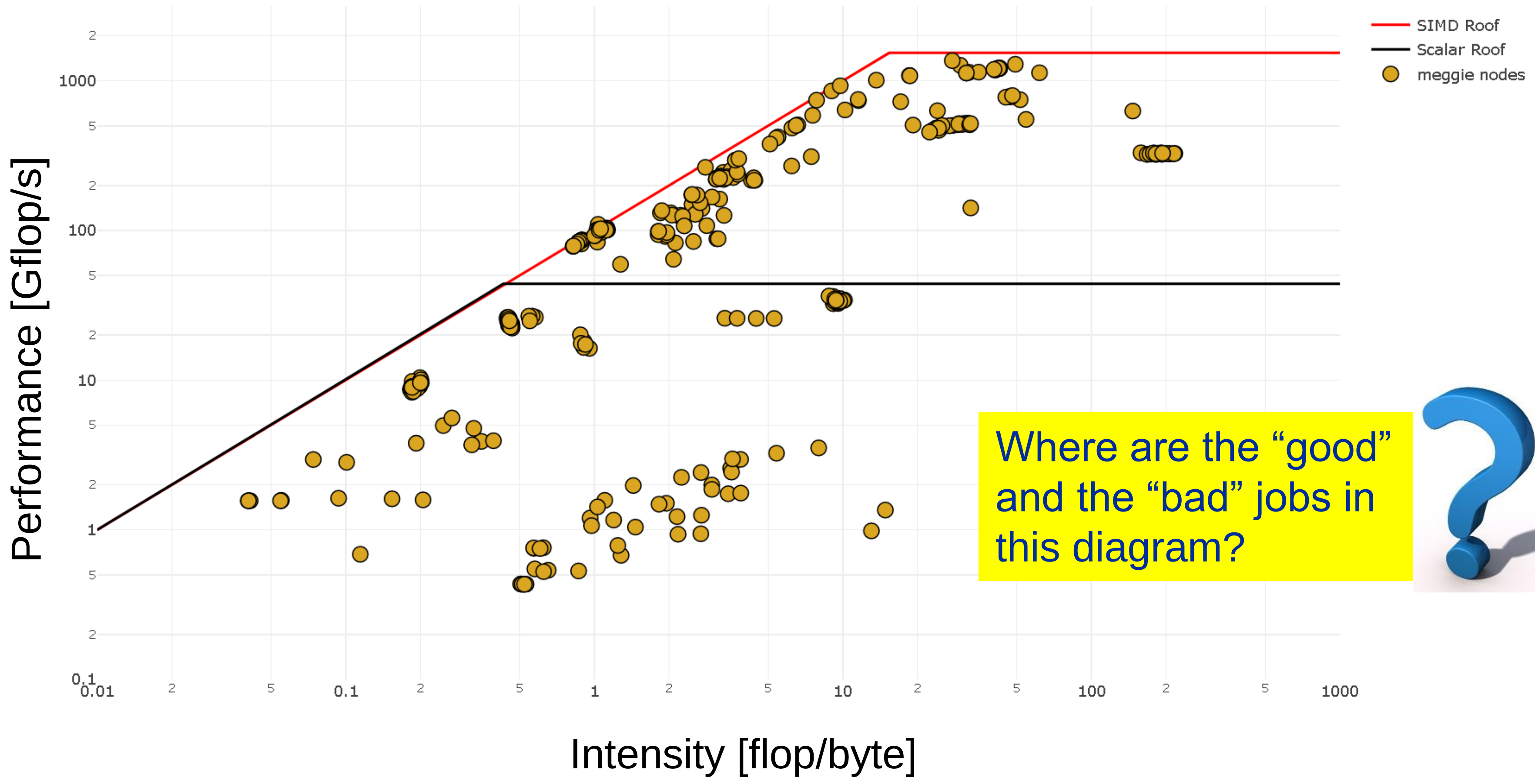
Diagnostic / phenomenological Roofline modeling

Diagnostic modeling

- What if we cannot predict the intensity/balance?
  - Code very complicated
  - Code not available
  - Parameters unknown
  - Doubts about correctness of analysis
- Measure data volume V_{meas} (and work N_{meas})
  - Hardware performance counters
  - Tools: likwid-perfctr, PAPI, Intel Vtune,…
- Insights + benefits
  - Compare analytic model and measurement → validate model
  - Can be applied (semi-)automatically
  - Useful in performace monitoring of user jobs on clusters



Roofline and performance monitoring of clusters



Where are the “good” and the “bad” jobs in this diagram?

https://github.com/RRZE-HPC/likwid/wiki/Tutorial%3A-Empirical-Roofline-Model

Roofline conclusion

- Roofline = simple first-principle model for upper performance limit of data-streaming loops
  - Machine model (P_{max}, b_S) + application model (I)
  - Conditions apply, extensions exist

- Two modes of operation
  - Predictive: Calculate I, calculate upper limit, validate model, optimize, iterate
  - Diagnostic: Measure I and P, compare with roof

- Challenge of predictive modeling: Getting P_{max} and I right