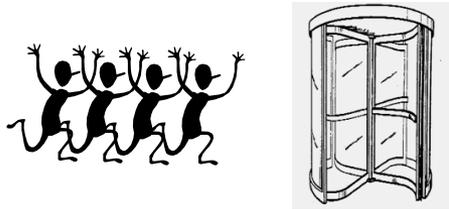# "Simple" performance modeling:
The Roofline Model

Loop-based performance modeling: Execution vs. data transfer

Prelude: Modeling customer dispatch in a bank

Revolving door throughput:
b_S [customers/sec]

Processing capability:
P_{max} [tasks/sec]

Intensity:
I [tasks/customer]

Prelude: Modeling customer dispatch in a bank

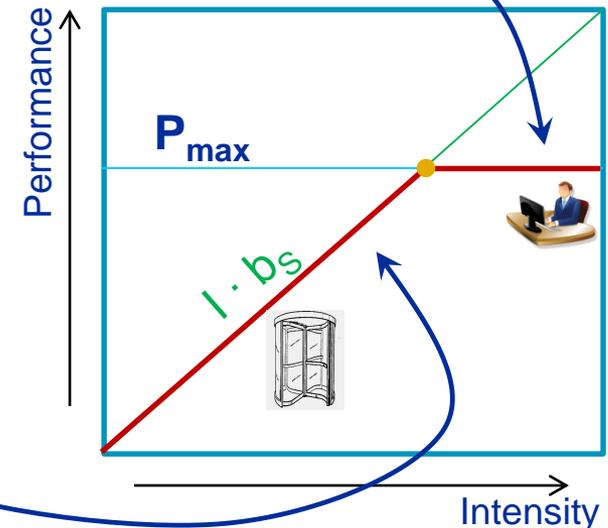## How fast can tasks be processed? P [tasks/sec]

The bottleneck is either

- The service desks (max. tasks/sec): P_{max}
- The revolving door (max. customers/sec): $I \cdot b_S$

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

This is the "Roofline Model"

- High intensity: P limited by "execution"
- Low intensity: P limited by "bottleneck"
- "Knee" at $P_{max} = I \cdot b_S$: Best use of resources
- **Roofline is an "optimistic" model ("light speed")**

The 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
   → e.g., b_S = 56 GByte/s

Expected performance:

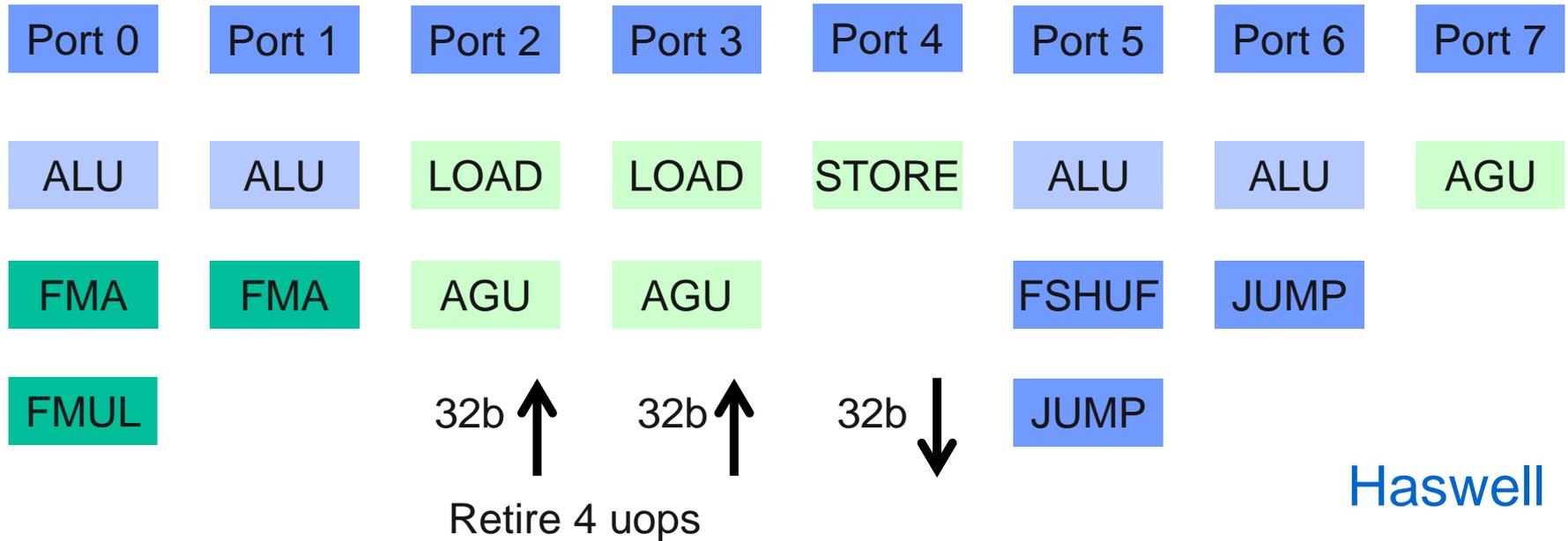$$P = \min(P_{\max}, I \cdot b_S) = \min\left(P_{\max}, \frac{b_S}{B_C}\right)$$

[Byte/s]

[Byte/Flop]

D. Callahan et al.: Estimating interlock and improving balance for pipelined architectures. Journal for Parallel and Distributed Computing 5(4), 334 (1988). DOI: 10.1016/0743-7315(88)90002-0
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)

Remember the Haswell port scheduler model?

Port 0	Port 1	Port 2	Port 3	Port 4	Port 5	Port 6	Port 7
ALU	ALU	LOAD	LOAD	STORE	ALU	ALU	AGU
FMA	FMA	AGU	AGU		FSHUF	JUMP	
FMUL		32b ↑	32b ↑	32b ↓	JUMP		

Retire 4 uops

Haswell

Example: Estimate 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 (one core)?

→ Equivalent to 4 scalar iterations

Cycle 1: LOAD + LOAD + STORE
Cycle 2: LOAD + LOAD + FMA + FMA
Cycle 3: LOAD + LOAD + STORE   **Answer: 1.5 cycles**

Example: Estimate 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:

$$\frac{2.3 \cdot 10^9 \text{ cy/s}}{1.5 \text{ cy}} \cdot 4 \text{ updates} \cdot \frac{2 \text{ flops}}{\text{update}} = \mathbf{12.27} \frac{\text{Gflops}}{\text{s}}$$

$$6.13 \cdot 10^9 \frac{\text{updates}}{\text{s}} \cdot 32 \frac{\text{bytes}}{\text{update}} = 196 \frac{\text{Gbyte}}{\text{s}}$$

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

- Peak memory bandwidth: b_S = **50 GB/s**
- Memory code balance:
  B_c = (4+1) Words / 2 Flops = 20 B/F (including write allocate)
  $\rightarrow$ ***I* = 0.05 F/B**

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

- P_{peak} = 515.2 Gflop/s (14 cores x (8+8) Flops/cy x 2.3 GHz)
- P_{max} = 14 x 12.27 Gflop/s =**172 Gflop/s** (33% peak)

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

A not so simple Roofline example

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

in single precision on a 2.2 GHz Sandy Bridge socket @ "large" N
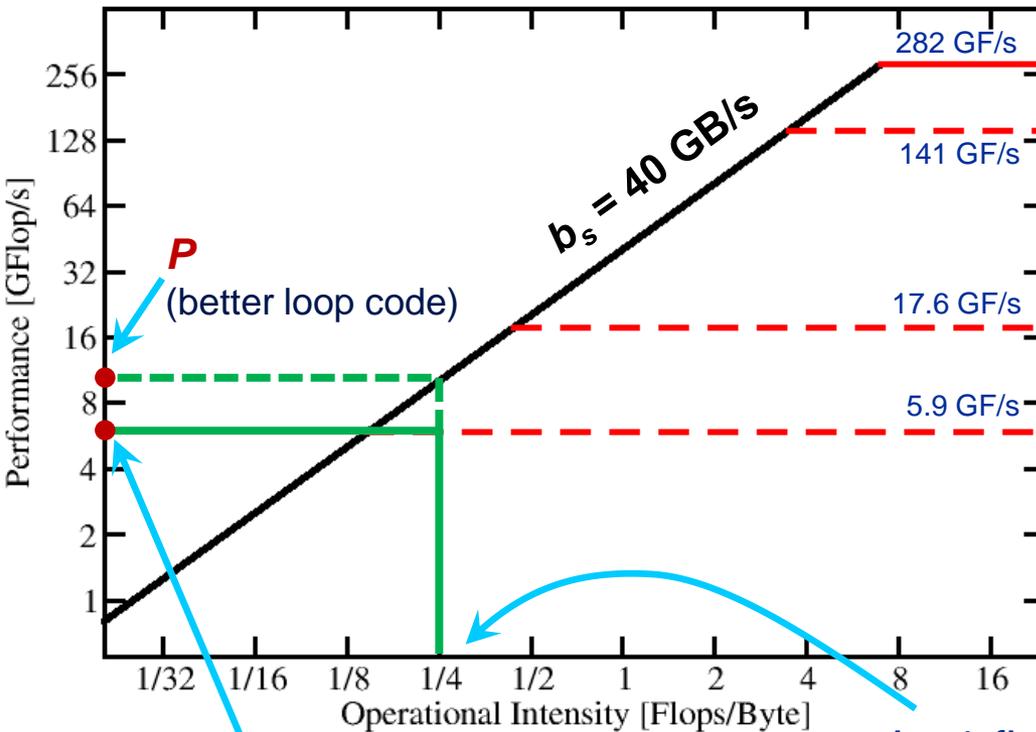
$$P = \min(P_{\max}, I \cdot b_S)$$

Machine peak (ADD+MULT) Out of reach for this code — 282 GF/s

ADD peak (best possible code) — 141 GF/s

no SIMD — 17.6 GF/s

3-cycle latency per ADD if not unrolled — 5.9 GF/s

How do we get these numbers???

b_s = 40 GB/s

P (better loop code)

P (worst loop code)

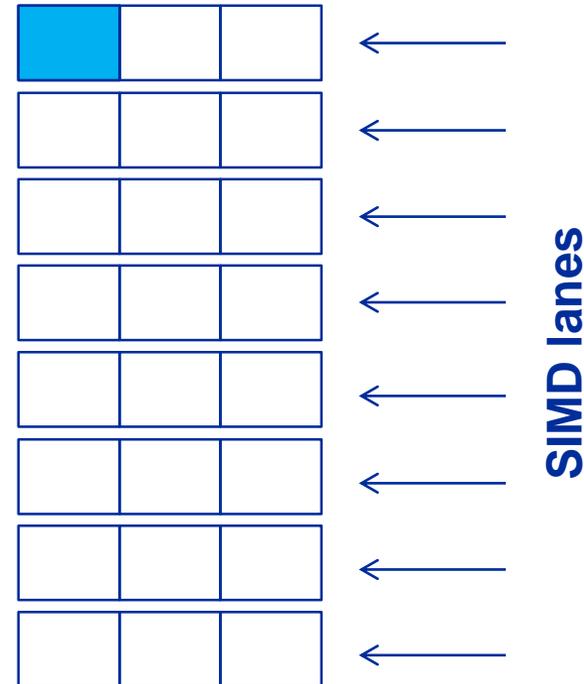I = 1 flop / 4 byte (SP!)

Applicable peak for the summation loop

Plain scalar code, no SIMD

```
LOAD r1.0 ← 0
i ← 1
loop:
  LOAD r2.0 ← a(i)
  ADD r1.0 ← r1.0+r2.0
  ++i →? loop
result ← r1.0
```

ADD pipes utilization:

SIMD lanes

→ **1/24 of ADD peak**

Scalar code, 3-way unrolling

```
LOAD r1.0 ← 0
LOAD r2.0 ← 0
LOAD r3.0 ← 0
i ← 1

loop:
  LOAD r4.0 ← a(i)
  LOAD r5.0 ← a(i+1)
  LOAD r6.0 ← a(i+2)

  ADD r1.0 ← r1.0 + r4.0
  ADD r2.0 ← r2.0 + r5.0
  ADD r3.0 ← r3.0 + r6.0

  i+=3 →? loop
result ← r1.0+r2.0+r3.0
```
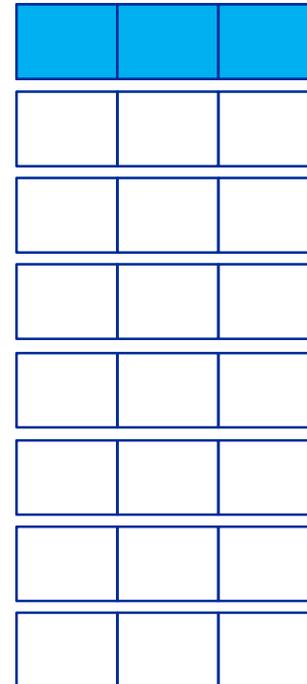
ADD pipes utilization:

→ **1/8 of ADD peak**

SIMD-vectorized, 3-way unrolled

```
LOAD [r1.0,…,r1.7] ← [0,…,0]
LOAD [r2.0,…,r2.7] ← [0,…,0]
LOAD [r3.0,…,r3.7] ← [0,…,0]
i ← 1

loop:
  LOAD [r4.0,…,r4.7] ← [a(i),…,a(i+7)]
  LOAD [r5.0,…,r5.7] ← [a(i+8),…,a(i+15)]
  LOAD [r6.0,…,r6.7] ← [a(i+16),…,a(i+23)]

  ADD r1 ← r1 + r4
  ADD r2 ← r2 + r5
  ADD r3 ← r3 + r6

  i+=24 →? loop
result ← r1.0+r1.1+...+r3.6+r3.7
```
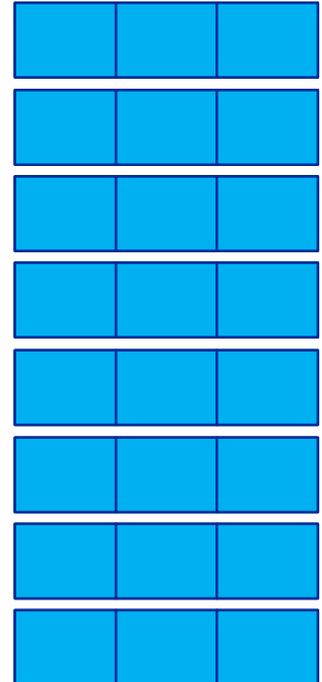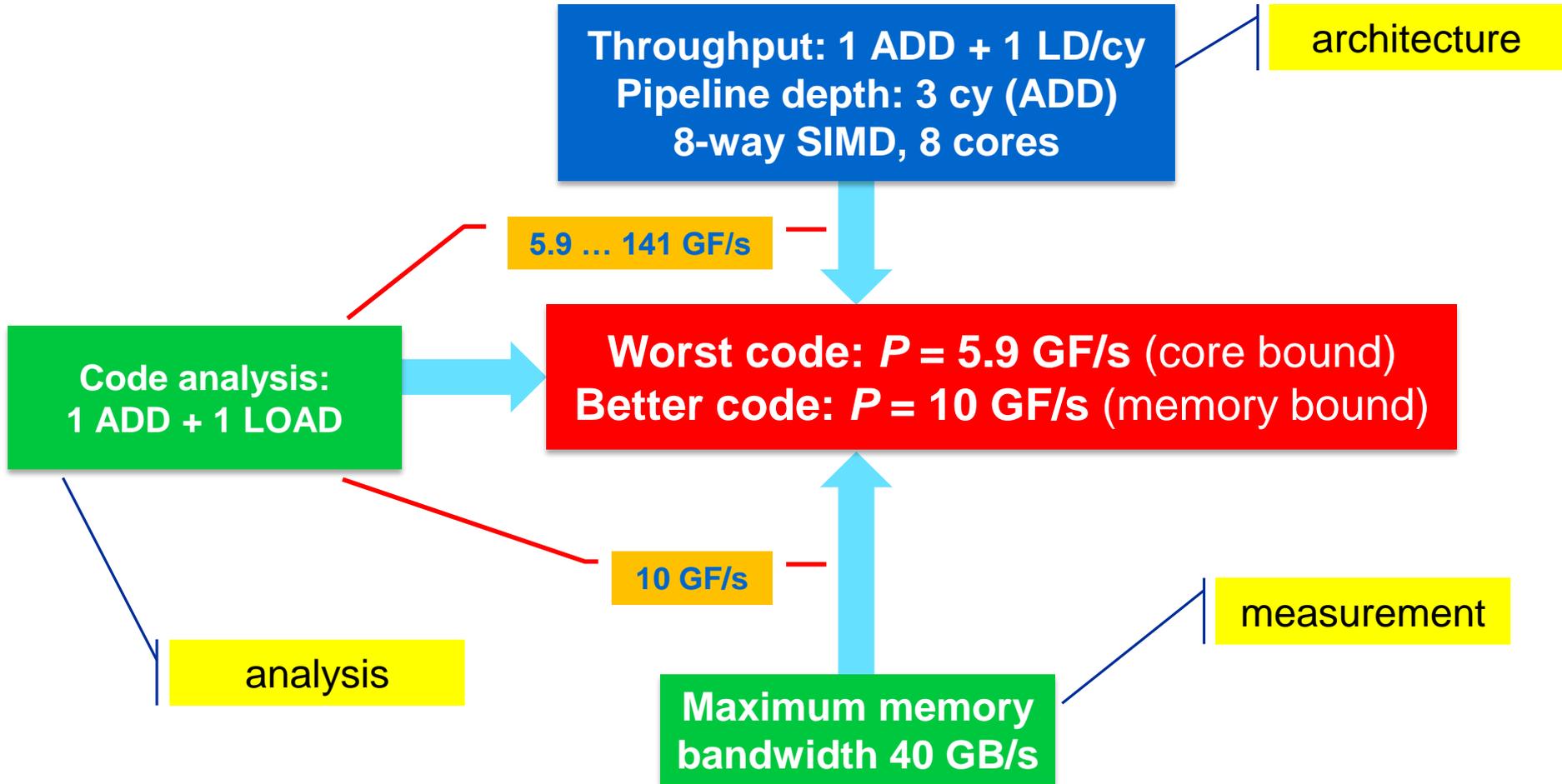
ADD pipes utilization:

→ **ADD peak**

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

analysis

Maximum memory
bandwidth 40 GB/s

measurement

Prerequisites for the Roofline Model

- **The roofline formalism is based on some (crucial) assumptions:**
  - There is a clear concept of "work" vs. "traffic"
    - "work" = flops, updates, iterations…
    - "traffic" = required data to do "work"

  - Attainable bandwidth of code = input parameter! Determine effective bandwidth via simple streaming benchmarks to model more complex kernels and applications

  - 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

# Typical code optimizations in the Roofline Model

1. Hit the BW bottleneck by good serial code
    (e.g.,   Perl   → Fortran)

2. Increase intensity to make better use of BW bottleneck
    (e.g., 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])

5. Shift P_{max} by accessing additional hardware features or using a different algorithm/implementation
    (e.g., scalar → SIMD)

Shortcomings of the roofline model
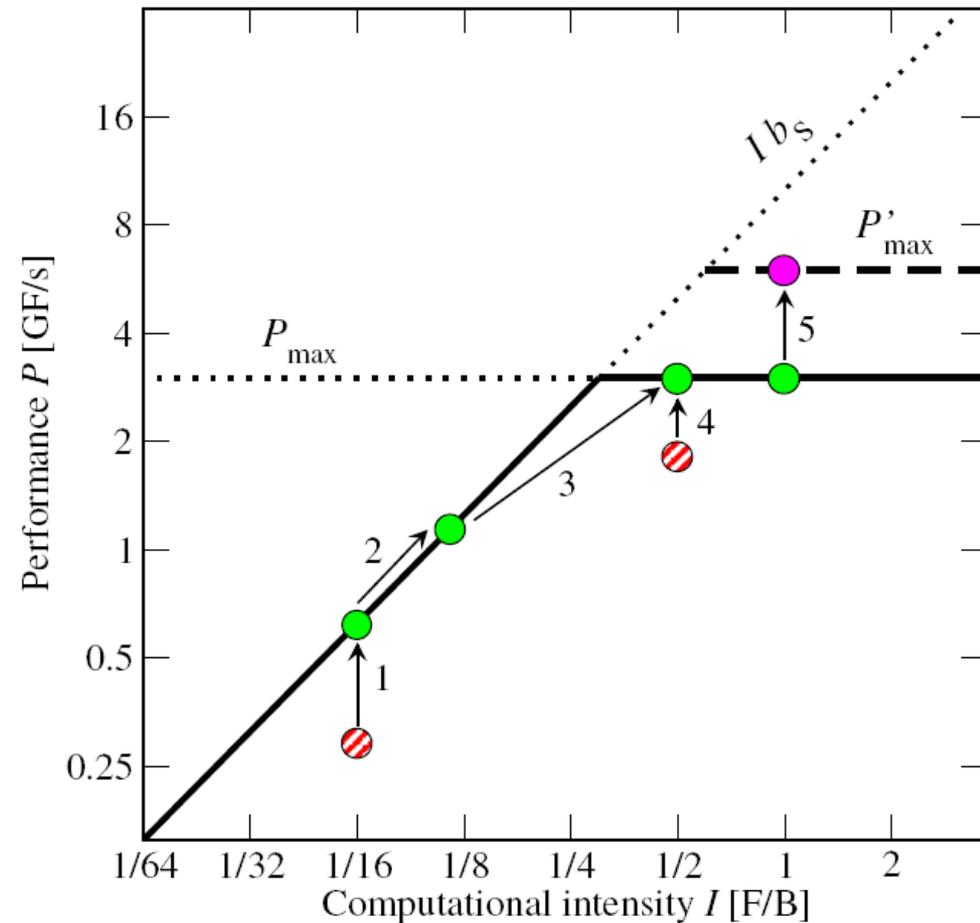
- **Saturation effects in multicore chips are not explained**
  - Reason: "saturation assumption"
  - Cache line transfers and core execution do sometimes not overlap perfectly
  - It is not sufficient to measure single-core STREAM to make it work
  - Only increased "pressure" on the memory interface 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 Preprint: arXiv:1208.2908

$$A(:)=B(:)+C(:)*D(:)$$
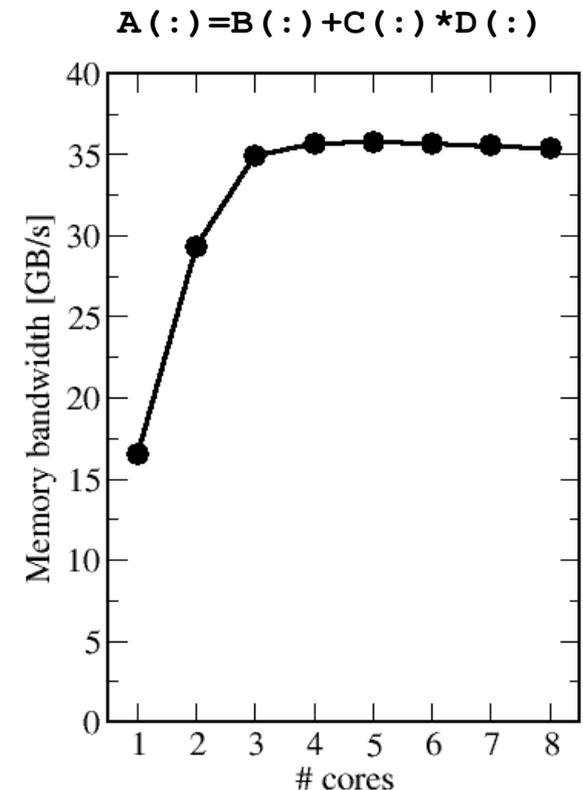
Interlude: Hardware Performance Counters

Ganglia Data / Roofline (04. Feb. 2016 - 14:12:24)

Click and drag to zoom in. Hold down shift key to x-pan.

Where are the "good" and the "bad" jobs in this diagram?