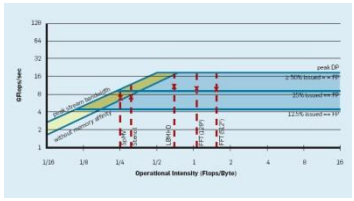
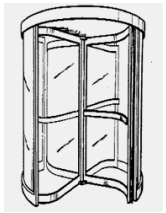


“Simple” performance modeling: The Roofline Model

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

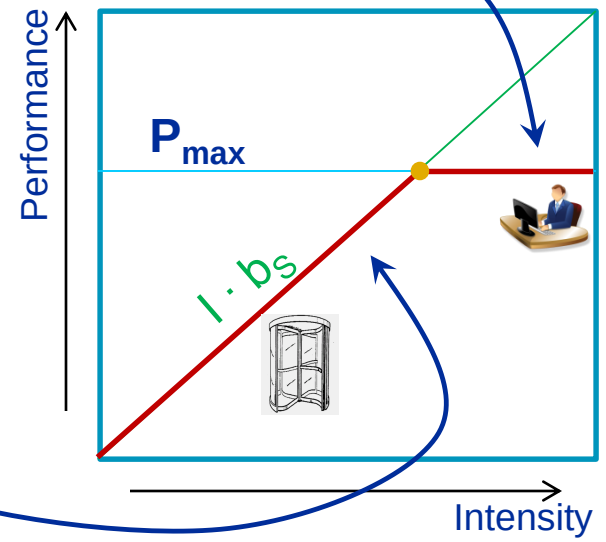
- The service desks (max. tasks/sec): $P_{\max}$
- The revolving door (max. customers/sec): $I \cdot b_S$

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

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:

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

[Byte/s]

[Byte/Flop]

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)

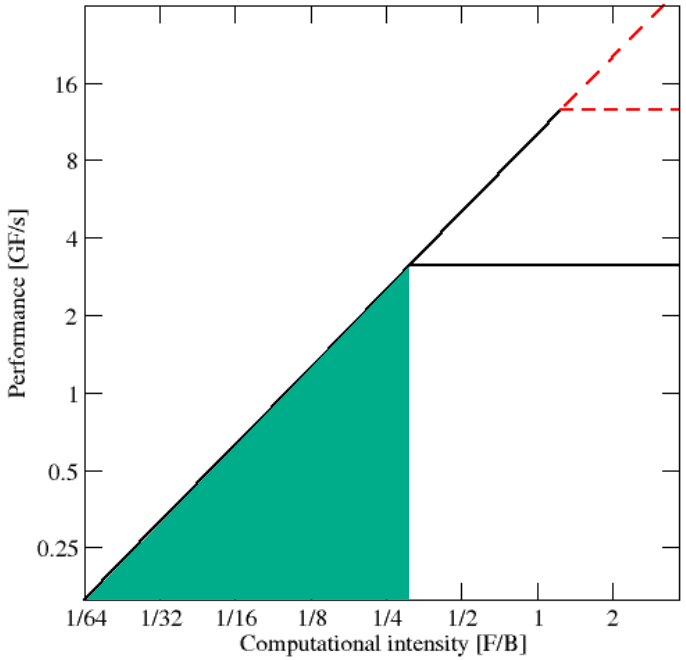
Factors to consider in the roofline model

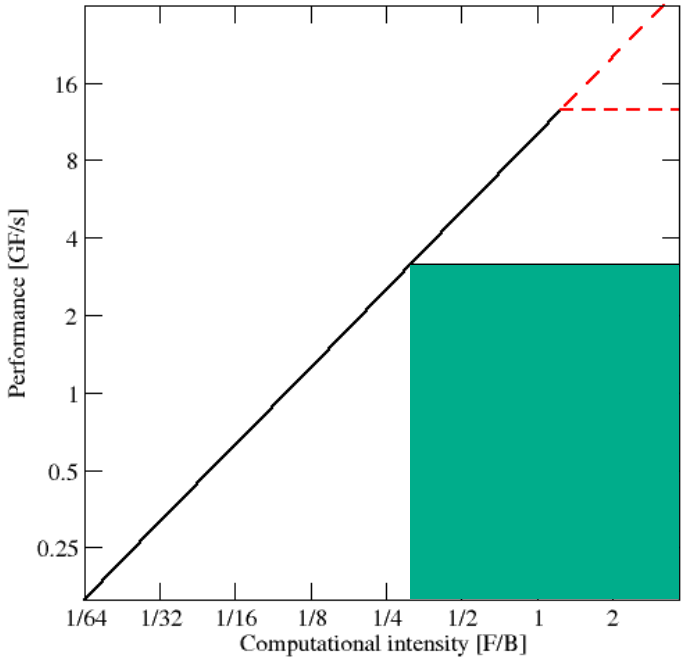
Bandwidth-bound (simple case)

- Accurate traffic calculation (write-allocate, strided access, …)
- Practical ≠ theoretical BW limits
- Erratic access patterns

Core-bound (may be complex)

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

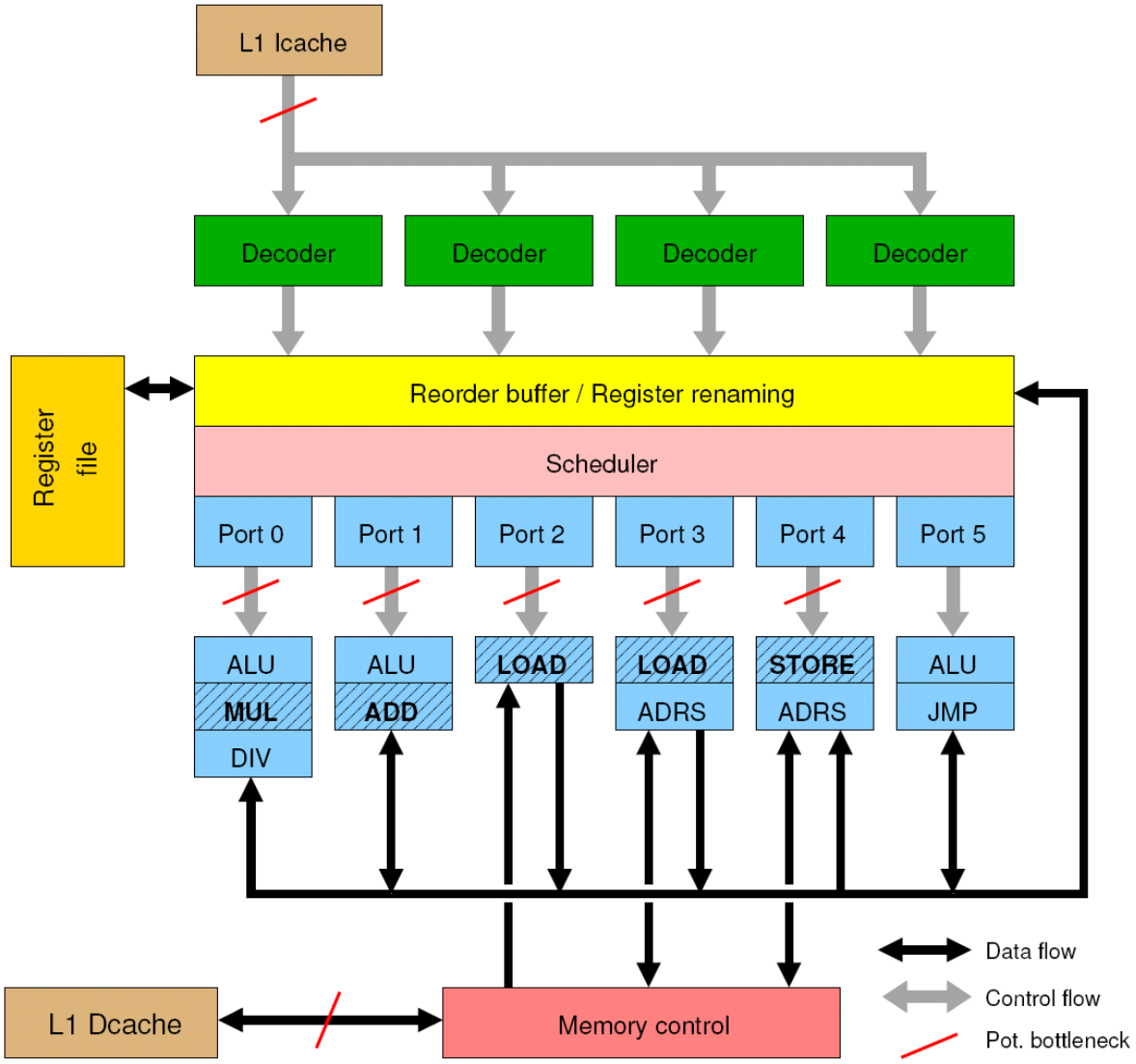




Complexities of in-core execution

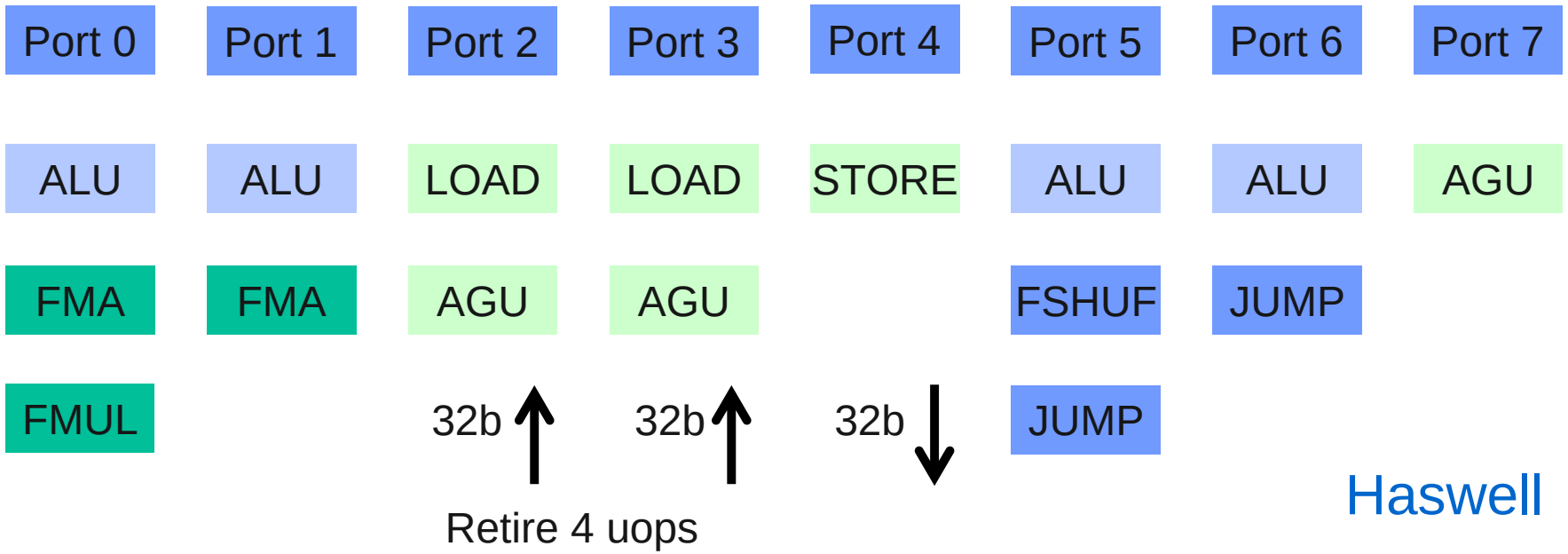
Multiple bottlenecks:

- 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
- …



Preliminary: Estimating P_{max}

Remember the Haswell port scheduler model?



Zeroth order throughput limits on Haswell

- **Per cycle with AVX, SSE, or scalar**
  - 2 LOAD instructions **AND** 1 STORE instruction
  - 2 instructions selected from the following five:
    - 2 FMA (fused multiply-add)
    - 2 MULT
    - 1 ADD
- **Overall maximum of 4 instructions per cycle**
  - In practice, 3 is more realistic
- **Remember: one AVX instruction handles**
  - 4 DP operands or
  - 8 SP operands

- **First order correction**
  - Typically only two LD/ST instructions per cycle due to one AGU handling “simple” addresses only
  - See SIMD chapter for more about memory addresses

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 (equivalent to 4 scalar iterations) on one core?

→ All (AVX) iterations are independent → assume full throughput

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)

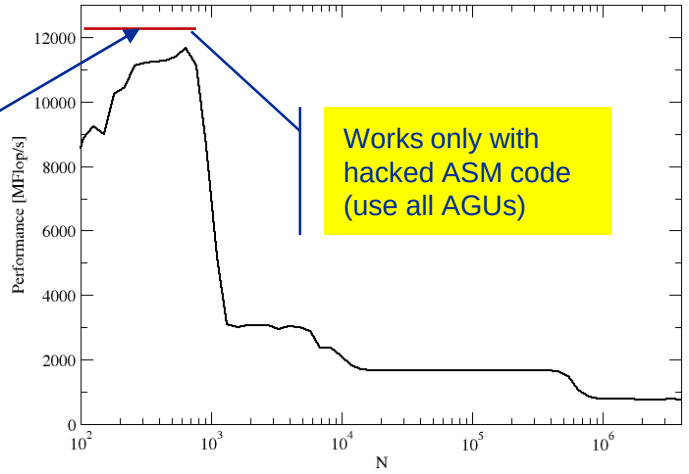
```
double  *A, *B, *C, *D;
for (int i=0; i<N; i++) {
   A[i] = B[i] + C[i] * D[i];
}
```

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:

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

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



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)

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

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

Machine balance for hardware characterization

- **For quick comparisons the concept of machine balance is useful**

$$B_m = \frac{b_S}{P_{\text{peak}}}$$

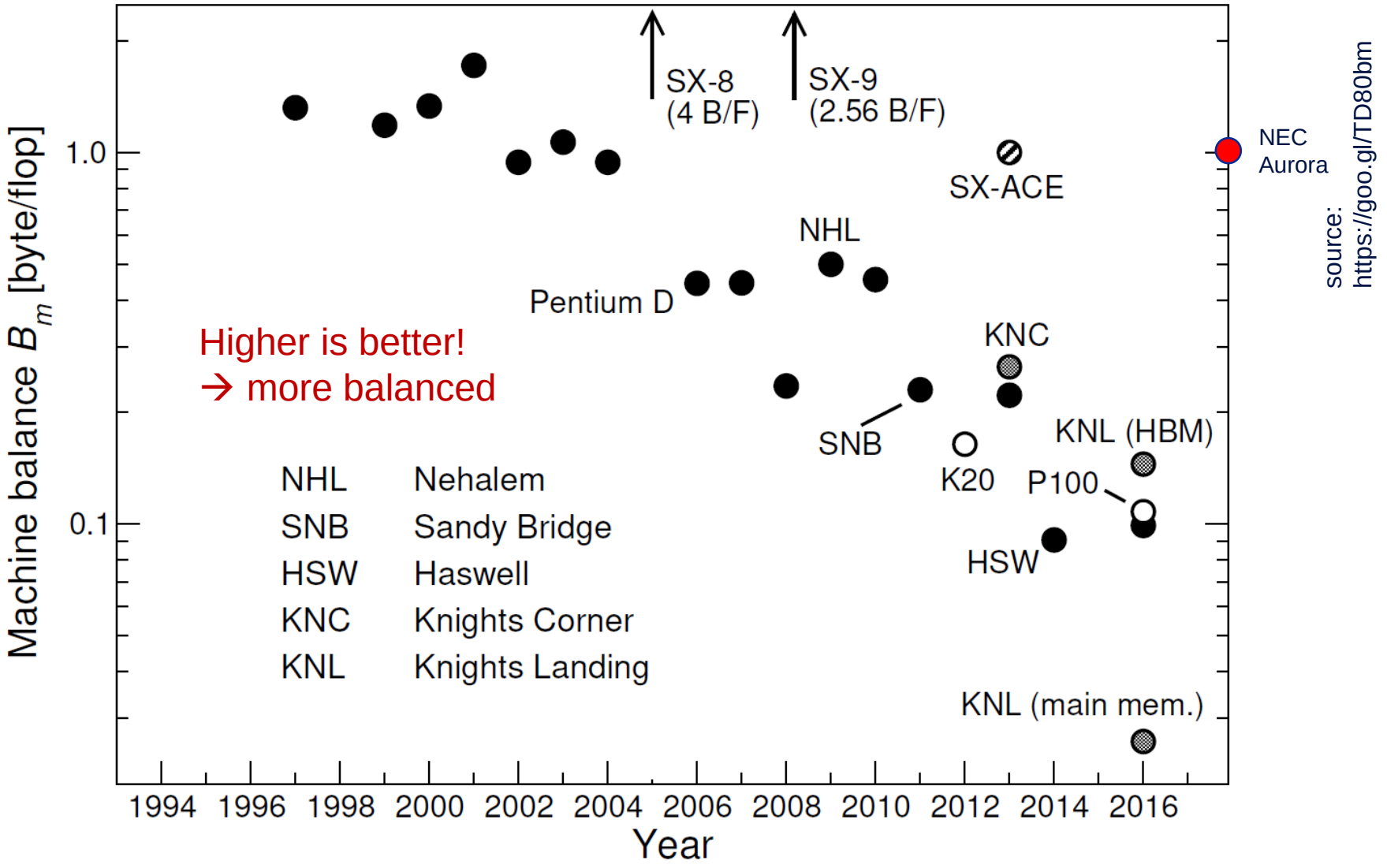
- Machine Balance = How much input data can be delivered for each FP operation? ("Memory Gap characterization")
  - Assuming balanced MULT/ADD
- Rough estimate: $B_m \ll B_c$ → strongly memory-bound code

- **Typical values (main memory):**

Intel Haswell 14-core 2.3 GHz
B_m = 60 GB/s / (14 x 2.3 x 16) GF/s ≈ **0.12 B/F**

Intel Sandy Bridge 8-core 2.7 GHz ≈ **0.23 B/F**
Nvidia K20x ≈ **0.14 B/F**
Intel Xeon Phi Knights Corner ≈ **0.16 B/F**

Machine balance over time

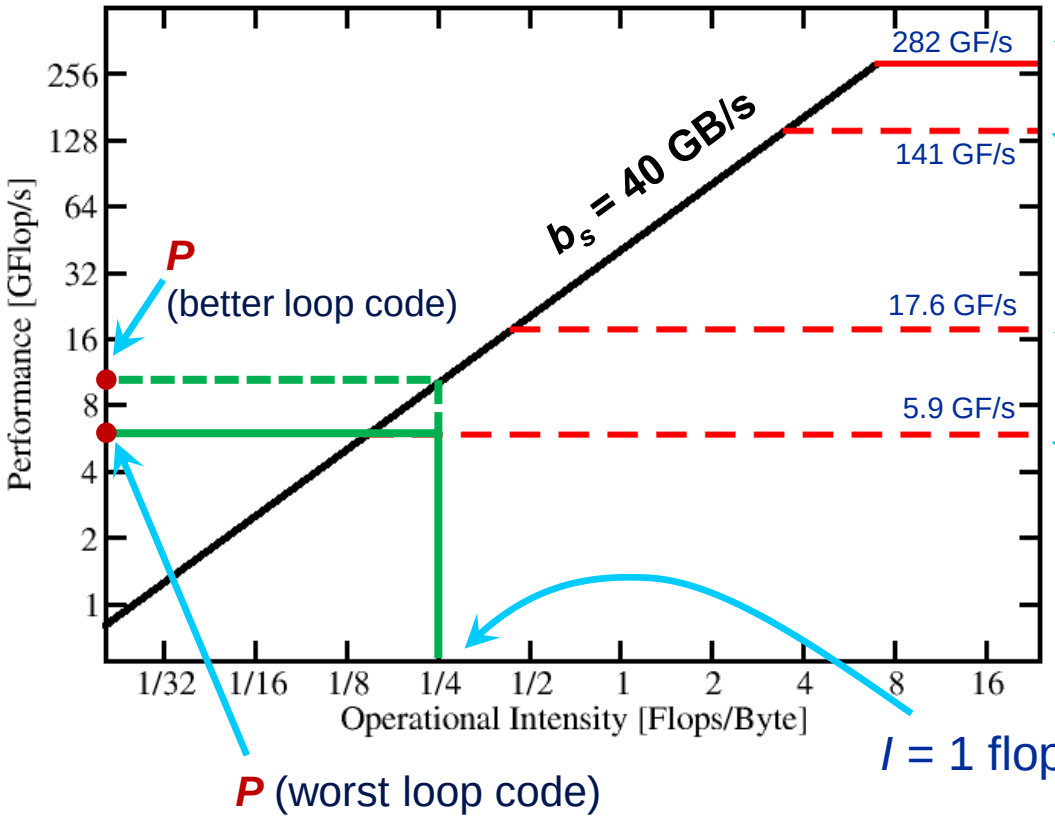


A not so simple Roofline example

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

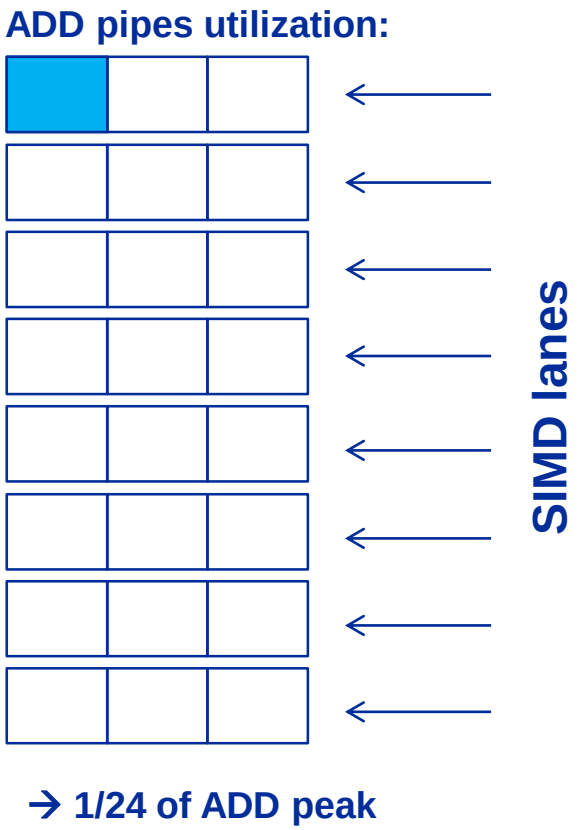
How do we get these numbers???

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
```



Applicable peak for the summation loop

Scalar code, 3-way unrolling

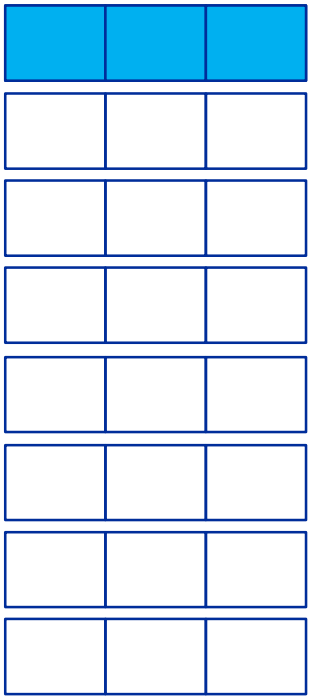
```
LOAD r1.0 ← 0
LOAD r2.0 ← 0
LOAD r3.0 ← 0
i ← 1

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

Applicable peak for the summation loop

SIMD-vectorized, 3-way unrolled

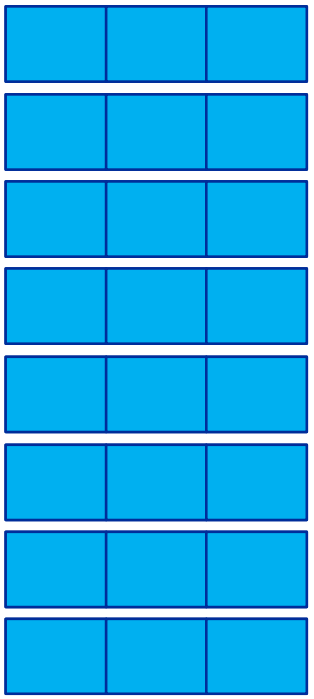
```
LOAD [r1.0,…,r1.7] ← [0,…,0]
LOAD [r2.0,…,r2.7] ← [0,…,0]
LOAD [r3.0,…,r3.7] ← [0,…,0]
i ← 1

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
```

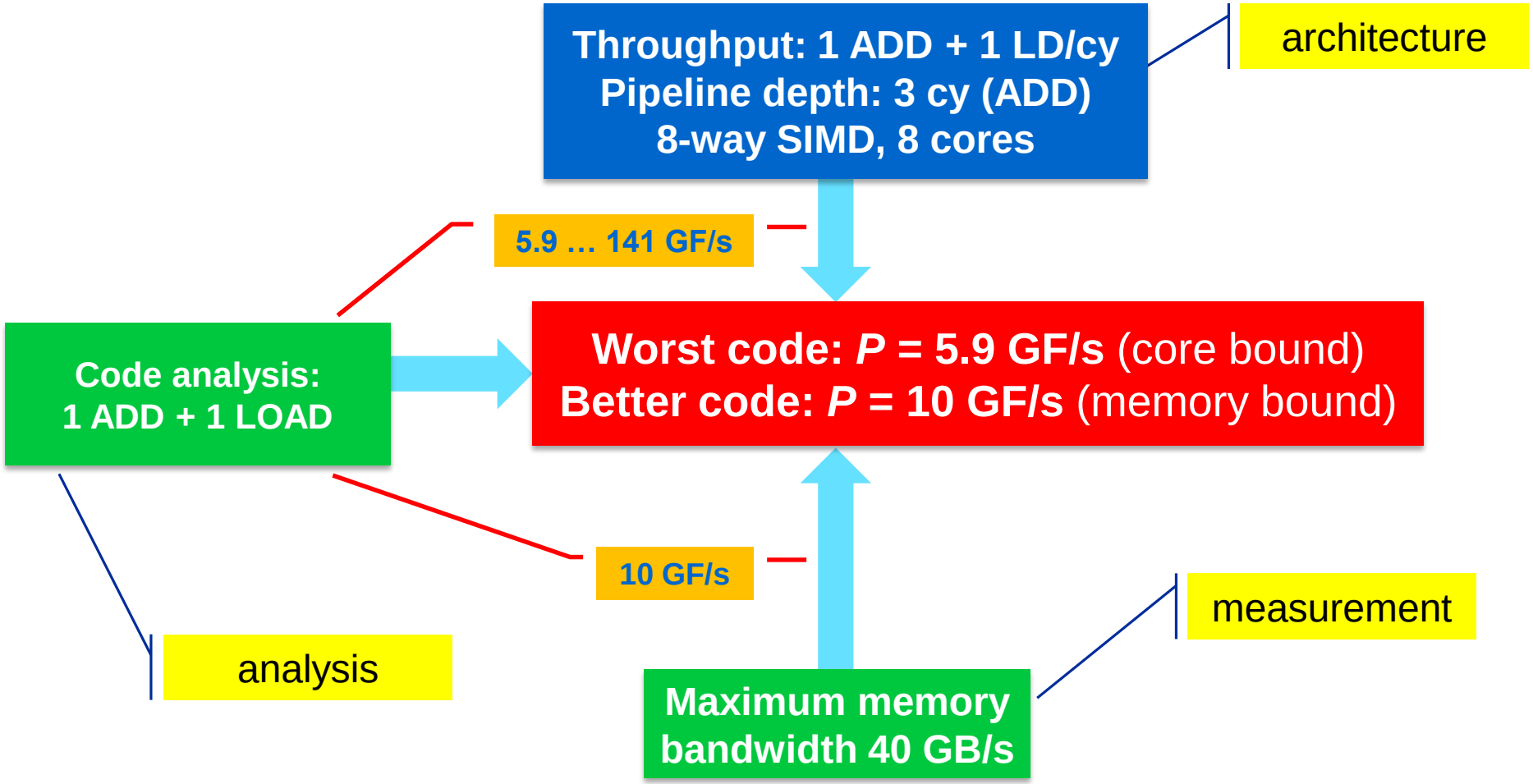
ADD pipes utilization:



→ ADD peak

Input to the roofline model

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

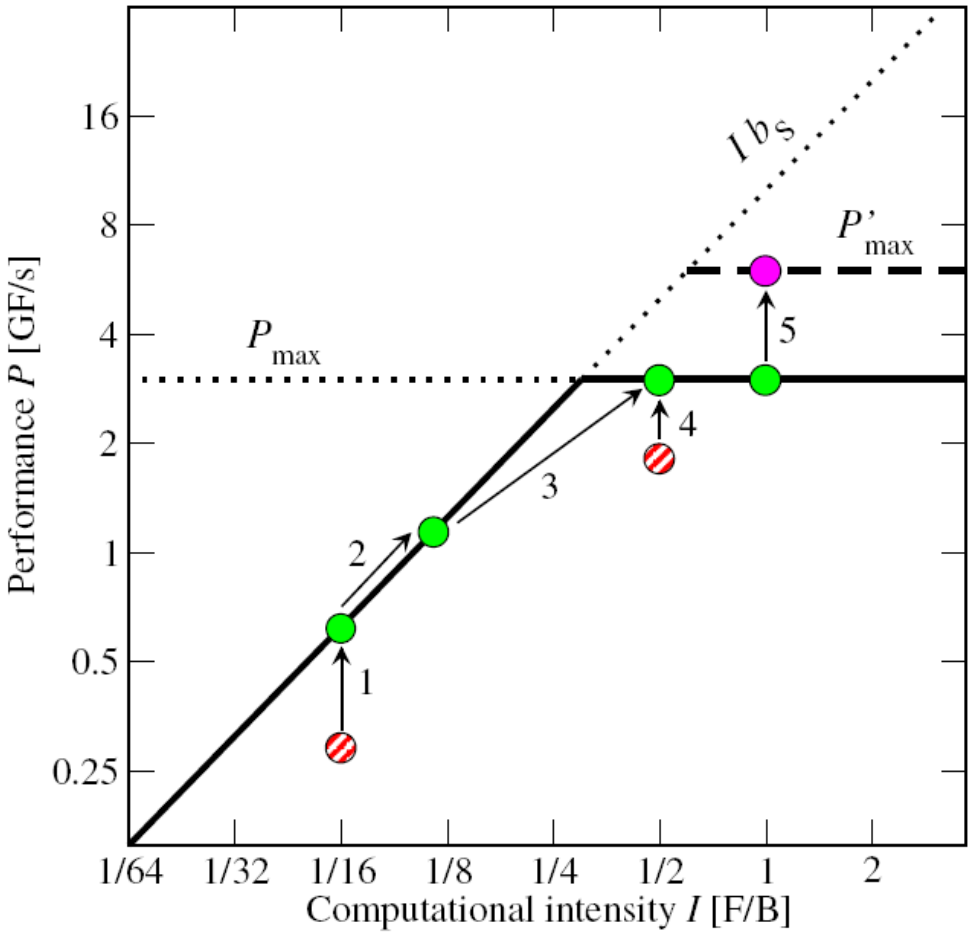


Prerequisites for the Roofline Model

- **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 saturated bandwidth of the chip 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

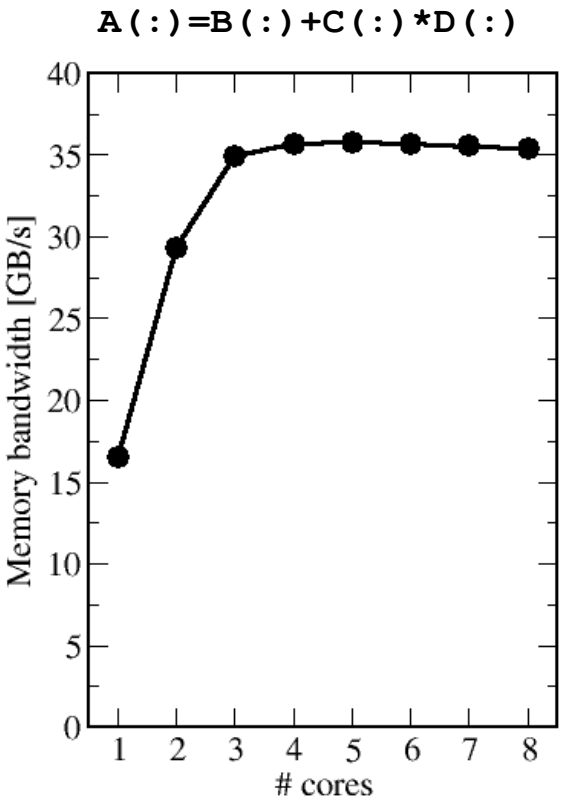
1. Hit the BW bottleneck by good serial code
   (e.g., Perl → Fortran)
2. Increase intensity to make better use of BW bottleneck
   (e.g., loop blocking [see later])
3. Increase intensity and go from memory-bound to core-bound
   (e.g., temporal blocking)
4. Hit the core bottleneck by good serial code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



Exercise: Dense matrix-vector multiplication in DP

```
do c = 1 , C
 do r = 1 , R
     y(r)=y(r) + A(r,c)* x(c)
 enddo
enddo
```

```
do c = 1 , C
   tmp=x(c)
   do r = 1 , R
      y(r)=y(r) + A(r,c)* tmp
   enddo
enddo
```

- Assume C = R ≈ 10,000
- Applicable peak performance?
- Relevant data path?
- Computational Intensity?

Dense MVM Roofline discussion for Haswell

7 cores (one CoD domain), 2.3 GHz

- P_{max}
  - AVX kernel: 2 LOADs, 1 STORE, 1 FMA
    → 1 cy per AVX iteration → per-core P_{max} = 8 F/cy = ½ P_{peak}
  - Full domain (CoD, 7 cores): P_{max} = (7 x 2.3 x 8) F/s = 129 GF/s

- B_c
  - `X()` does not cause significant data traffic
  - R=10000 → memory for `Y()` = 80 kB → fits into L2 cache of each core
    → only `A(,)` causes traffic from main memory of 8 bytes for 2 Flops

    → B_c = 4 B/F

- b_S = 32 GB/sec (saturated memory domain BW measurement [CoD])

- **Roofline model:**

$$P = \min(P_{\max}, I \cdot b_S) = \min(129,8)\,\mathrm{GFlop/s} = 8\,\mathrm{GFlop/s}$$