Μεταγλωττιστές για Ενσωματωμένα Συστήματα

Χειμερινό Εξάμηνο 2023-24
«Memory and Roofline Model»

Παναγιώτης Χατζηδούκας

Outline

- Memory
- Roofline Model
- Hands-on
  - Peak performance
  - Memory bandwidth
  - Memory layout (struct of arrays vs array of structures)
  - Matrix multiplication

Memory

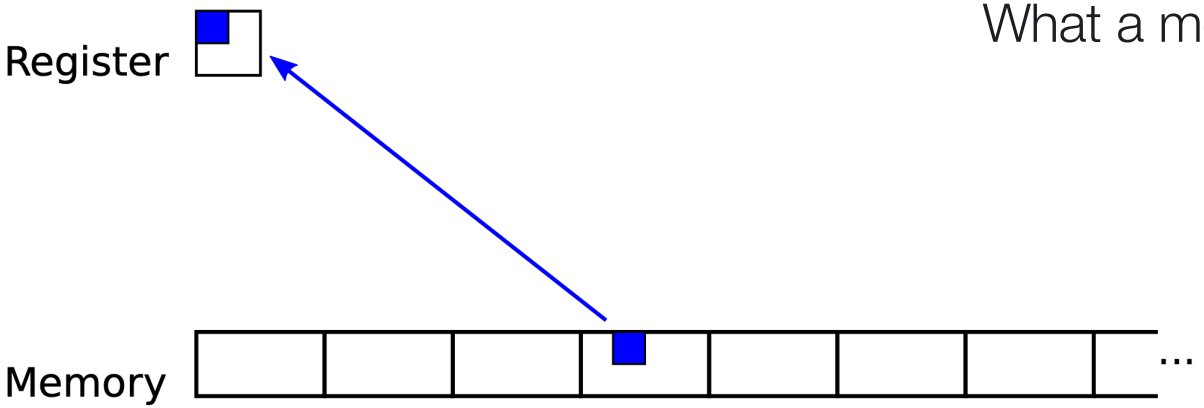
Locality

- Idea: have near you only what you need
- Temporal locality: if an item is referenced, it will need to be referenced again soon
  - Loops: instructions and data accessed repeatedly

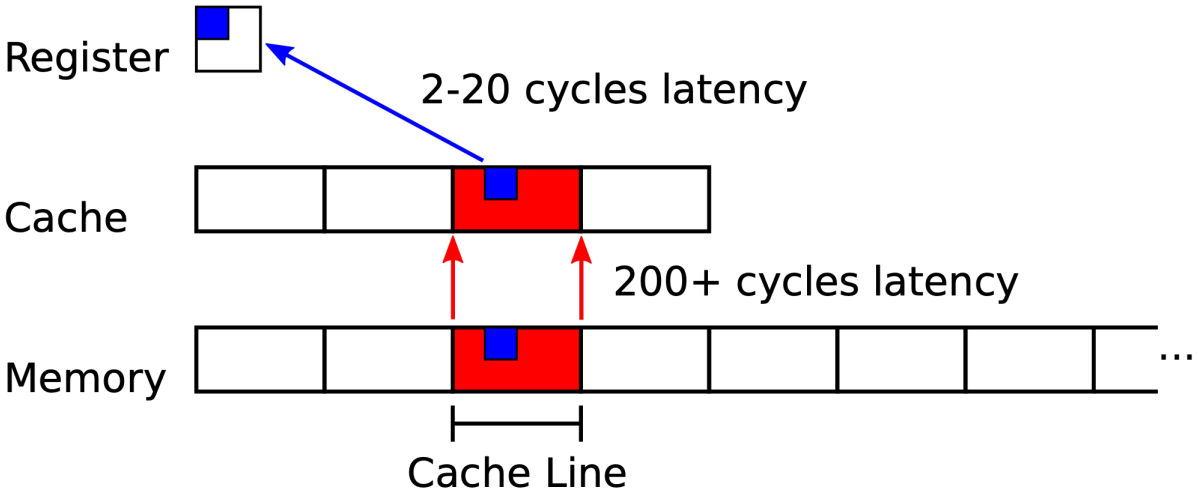
- Spatial locality: if an item is referenced, items whose addresses are close by, will tend to be referenced soon
  - Data access: sequential access to elements of array

Register and Caches

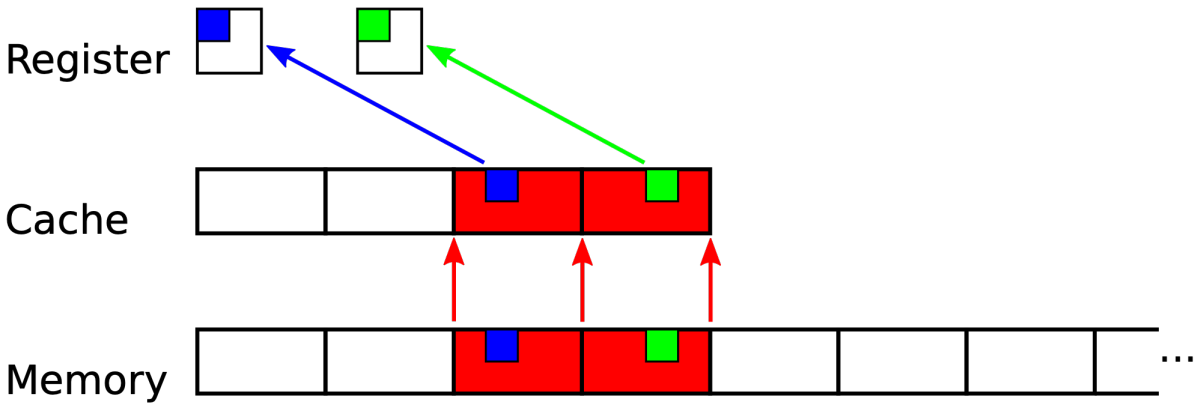
What a memory access looks like

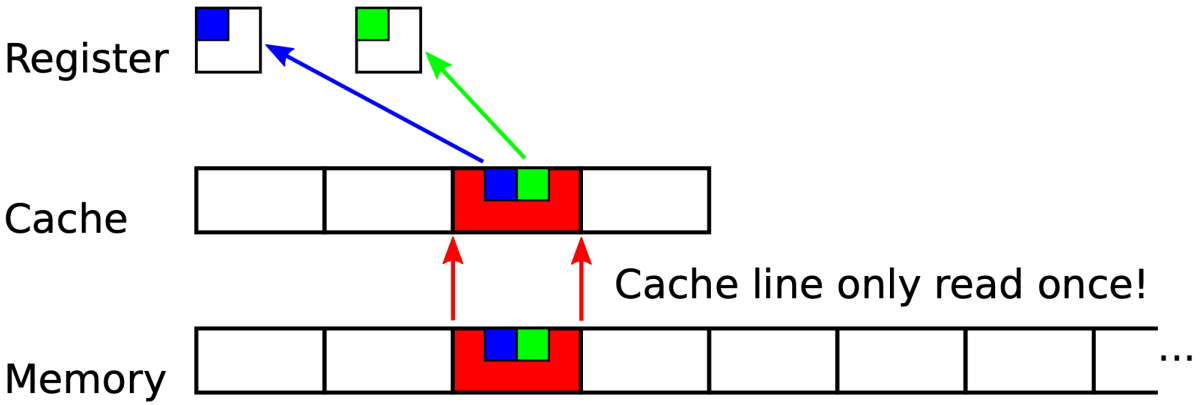


What really happens



Spatial Locality





Better spatial locality!

Locality Example 1

```
sum = 0;
for (i = 0; i < n; i++)
  sum += a[i];

return sum;
```

- Data:
  - Temporal: sum referenced in each iteration
  - Spatial: array a[] accessed consecutively
- Instructions:
  - Temporal: loops cycle through the same instructions
  - Spatial: instructions referenced in sequence
- Being able to assess the locality of code is a crucial skill for a performance programmer

Locality Example 2

```
double sum_array_rows(double a[M][N])
{
  int i, j;
  double sum = 0;

  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
      sum += a[i][j];

  return sum;
}
```

Locality Example 3

```
double sum_array_3d(double a[K][M][N])
{
  int i, j, k;
  double sum = 0;

  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
      for (k = 0; k < K; k++)
        sum += a[k][i][j];

  return sum;
}
```

Memory Hierarchy

- Multiple levels of memory with different speeds and sizes.

	Capacity	Latency	Cost/GB
Register	1000s of bits	20 ps	$$$$
SRAM	~10 KB-10 MB	1-10 ns	~$1000
DRAM	~10 GB	80 ns	~$10
Flash*	~100 GB	100 us	~$1
Hard disk*	~1 TB	10 ms	~$0.10

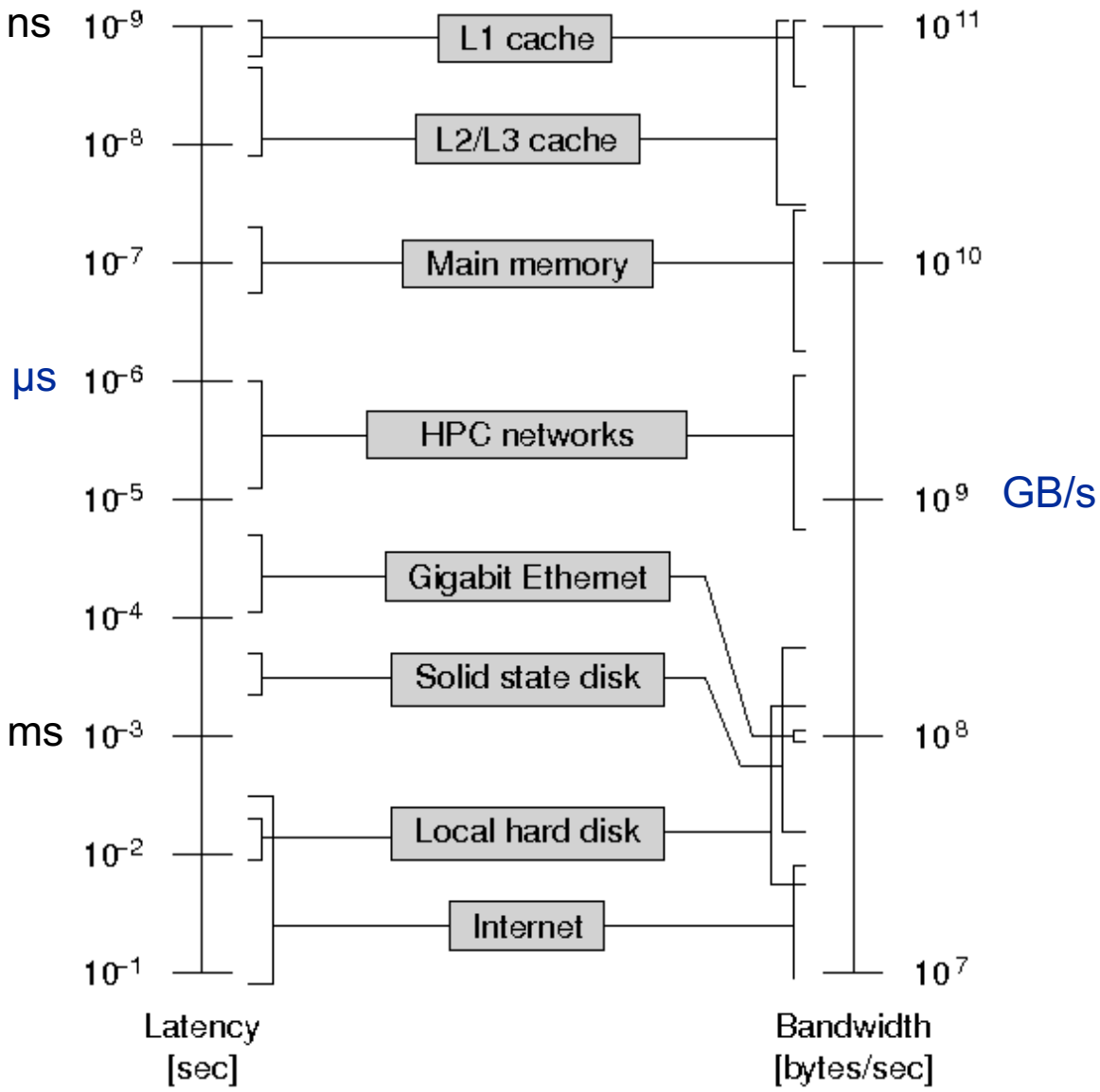
Processor Datapath

Memory Hierarchy

I/O subsystem

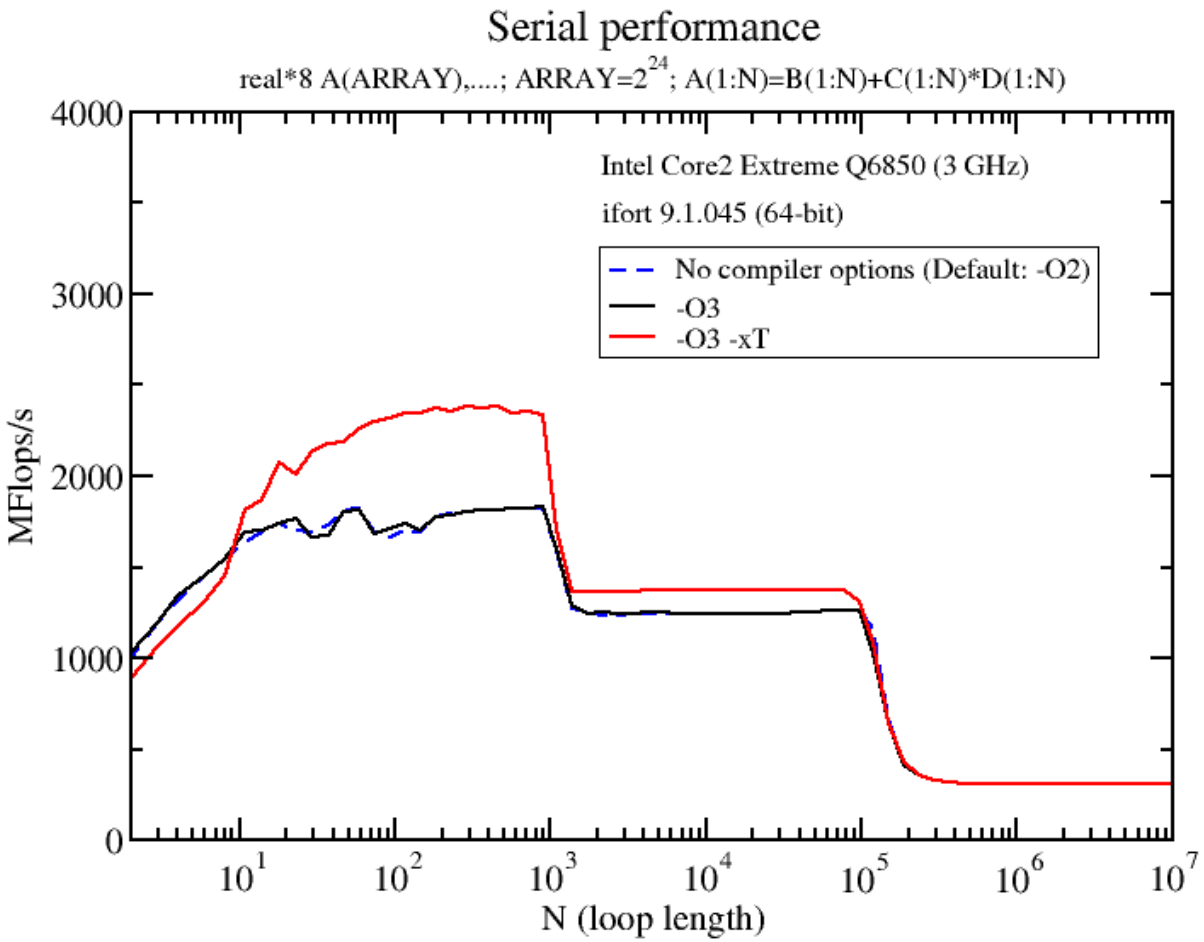
* non-volatile (retains contents when powered off)

Latency and Bandwidth



Characterization of Memory Hierarchies

- **Peak Performance** for 1 core of Intel Core2 Q6850 (DP):
  - 3 GHz * (2 Flops (DP-Add) + 2 Flops (DP-Mult)) = **12 GFlops/s**



Performance decreases if data set exceeds cache size

`-xT` : Enables vectorization & improves in-cache performance: Packed SSE instructions

Cache: Terminology

- **Block/Line**: minimum unit of information that can be present or not present in a cache
- **Hit**: data request by the processor, encountered in some block in the upper (closer to processor) level of memory hierarchy. If the data is not found, it is a miss. Then a lower (further from processor) level is accessed to retrieve the data.
- **Hit rate**: fraction of memory accesses found in the upper level.
- **Hit time**: time to access upper level including time to determine if it is a hit or a miss.
- **Miss penalty**: time to replace a block in the upper level with the corresponding block from the lower level.

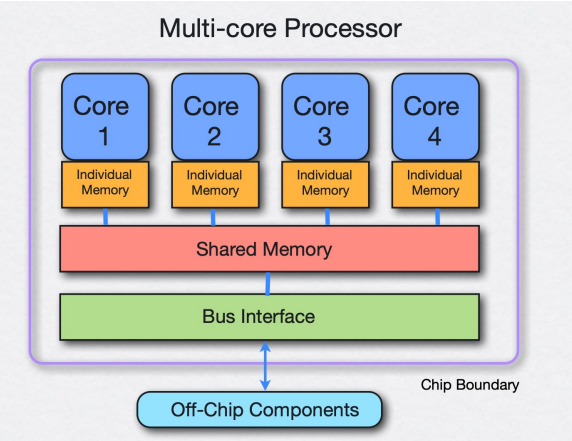
Effective Access Time (EAT)

- Suppose that
  - cache access time = 10ns
  - main memory access time = 200ns
  - cache hit rate = 99%
- What is the EAT for non-overlapped access?

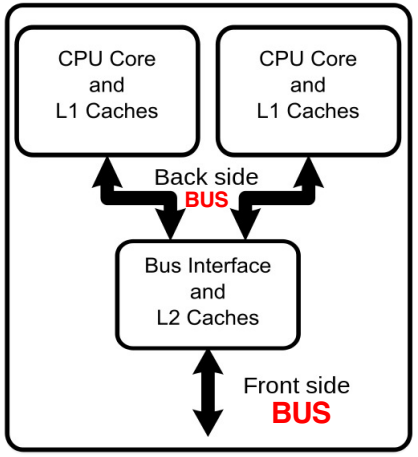
$$EAT = 0.99(10ns) + 0.01(10ns + 200s) = 9.9ns + 2.1ns = 12ns$$

Caches and Multiprocessors

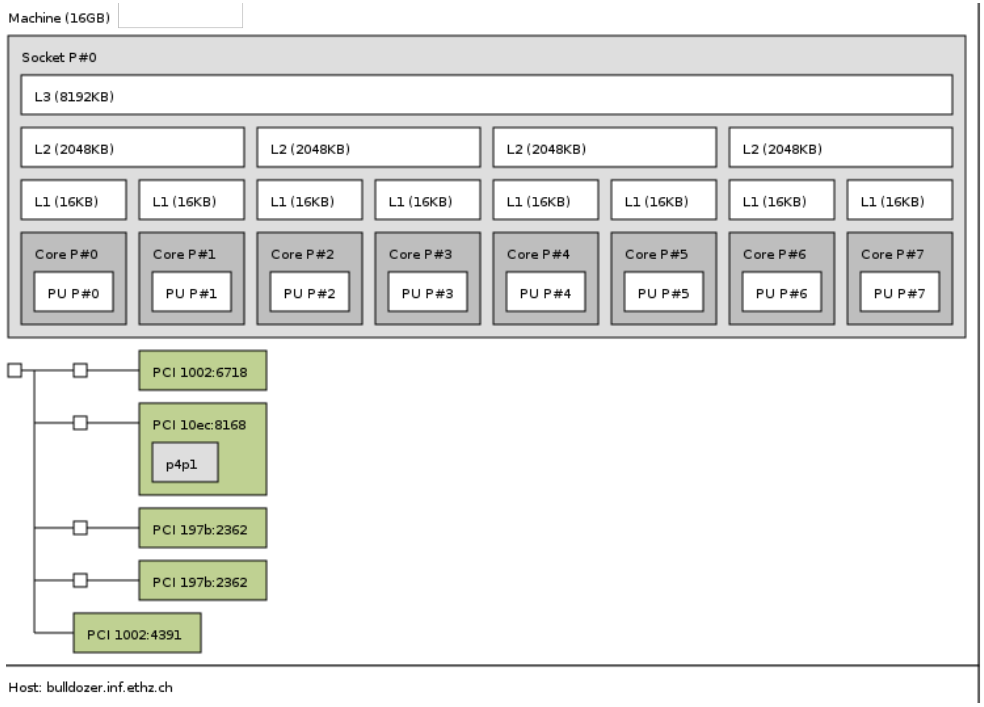
- BUS: a shared communication link, which uses one set of wires to connect multiple subsystems.
- Used for communication between memory, I/O and processors.
  - versatility: since we have a single connection scheme, new devices can be added
  - simplicity: single set of wires is shared in multiple ways
  - communication bottleneck: limiting the I/O throughput as all information passes a single wire

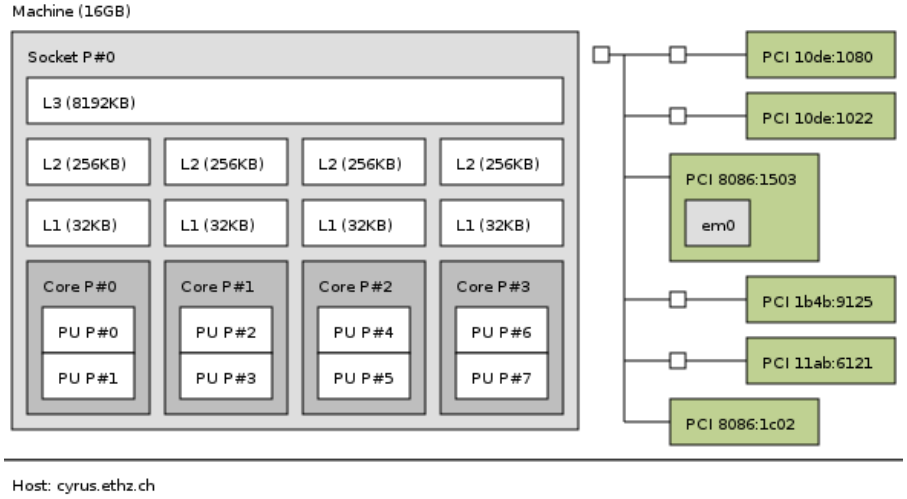


Processor-Memory Bus: bus that connects processor and memory, and that is short, high speed and matched to the memory system so as to *maximize memory-processor bandwidth*.



Portable Hardware Locality (hwloc)





http://www.open-mpi.org/projects/hwloc/

Euler Compute Node (text format)

```
Machine (256GB)
  NUMANode L#0 (P#0 128GB) + Socket L#0 + L3 L#0 (30MB)
    L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0 + PU L#0 (P#0)
    L2 L#1 (256KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1 + PU L#1 (P#1)
    L2 L#2 (256KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2 + PU L#2 (P#2)
    L2 L#3 (256KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3 + PU L#3 (P#3)
    L2 L#4 (256KB) + L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4 + PU L#4 (P#4)
    L2 L#5 (256KB) + L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5 + PU L#5 (P#5)
    L2 L#6 (256KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6 + PU L#6 (P#6)
    L2 L#7 (256KB) + L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7 + PU L#7 (P#7)
    L2 L#8 (256KB) + L1d L#8 (32KB) + L1i L#8 (32KB) + Core L#8 + PU L#8 (P#8)
    L2 L#9 (256KB) + L1d L#9 (32KB) + L1i L#9 (32KB) + Core L#9 + PU L#9 (P#9)
    L2 L#10 (256KB) + L1d L#10 (32KB) + L1i L#10 (32KB) + Core L#10 + PU L#10 (P#10)
    L2 L#11 (256KB) + L1d L#11 (32KB) + L1i L#11 (32KB) + Core L#11 + PU L#11 (P#11)
    HostBridge L#0
      PCIBridge
        PCI 14e4:168e
          Net L#0 "eth0"
        PCI 14e4:168e
          Net L#1 "eth1"
      PCIBridge
        PCI 103c:323b
          Block L#2 "sda"
      PCIBridge
        PCI 15b3:1003
          Net L#3 "ib0"
          Net L#4 "ib1"
          OpenFabrics L#5 "mlx4_0"
      PCIBridge
        PCI 102b:0533
  NUMANode L#1 (P#1 128GB) + Socket L#1 + L3 L#1 (30MB)
    L2 L#12 (256KB) + L1d L#12 (32KB) + L1i L#12 (32KB) + Core L#12 + PU L#12 (P#12)
    L2 L#13 (256KB) + L1d L#13 (32KB) + L1i L#13 (32KB) + Core L#13 + PU L#13 (P#13)
    L2 L#14 (256KB) + L1d L#14 (32KB) + L1i L#14 (32KB) + Core L#14 + PU L#14 (P#14)
    L2 L#15 (256KB) + L1d L#15 (32KB) + L1i L#15 (32KB) + Core L#15 + PU L#15 (P#15)
    L2 L#16 (256KB) + L1d L#16 (32KB) + L1i L#16 (32KB) + Core L#16 + PU L#16 (P#16)
    L2 L#17 (256KB) + L1d L#17 (32KB) + L1i L#17 (32KB) + Core L#17 + PU L#17 (P#17)
    L2 L#18 (256KB) + L1d L#18 (32KB) + L1i L#18 (32KB) + Core L#18 + PU L#18 (P#18)
    L2 L#19 (256KB) + L1d L#19 (32KB) + L1i L#19 (32KB) + Core L#19 + PU L#19 (P#19)
    L2 L#20 (256KB) + L1d L#20 (32KB) + L1i L#20 (32KB) + Core L#20 + PU L#20 (P#20)
    L2 L#21 (256KB) + L1d L#21 (32KB) + L1i L#21 (32KB) + Core L#21 + PU L#21 (P#21)
    L2 L#22 (256KB) + L1d L#22 (32KB) + L1i L#22 (32KB) + Core L#22 + PU L#22 (P#22)
    L2 L#23 (256KB) + L1d L#23 (32KB) + L1i L#23 (32KB) + Core L#23 + PU L#23 (P#23)
```

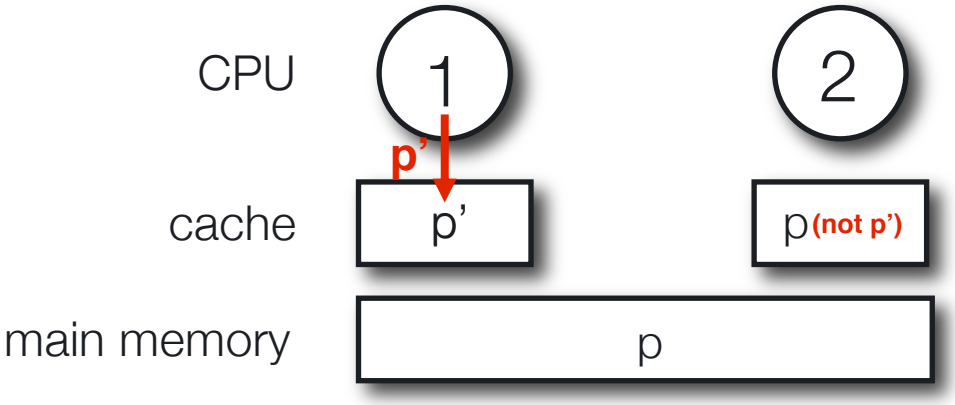
1st NUMA NODE

- Compute node:
  - 256 GB RAM
  - 2 NUMA Nodes
- NUMA Node
  - 128GB RAM
  - 1 Socket
    - 30MB L3 Cache
    - 12 Cores
- Core
  - 256KB L2 Cache
  - 32KB L1 Data Cache
  - 32KB L1 Instruction Cache

2nd NUMA NODE

Parallelism and Memory Hierarchies

- Multicore multiprocessor:
  - Processors (most likely) share a common physical address space
- Caching shared data: view of memory for each processor through their individual caches so it differs if changes are made.
- CAREFUL: 2 different processors can have 2 different values for the same location -> cache coherence problem



Cache Coherency

- A memory system is coherent if:
  - A read by processor P to location X, that follows a write by P to X, with no writes to X by another processor occurring between the write and read by P, always returns the value written by P.
  - A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occurs between the 2 accesses. ⇒ needs controller
  - Writes to the same location are serialized: that is 2 writes to the same location by any 2 processors are seen in the same order by all processors.

Enforcing Coherence

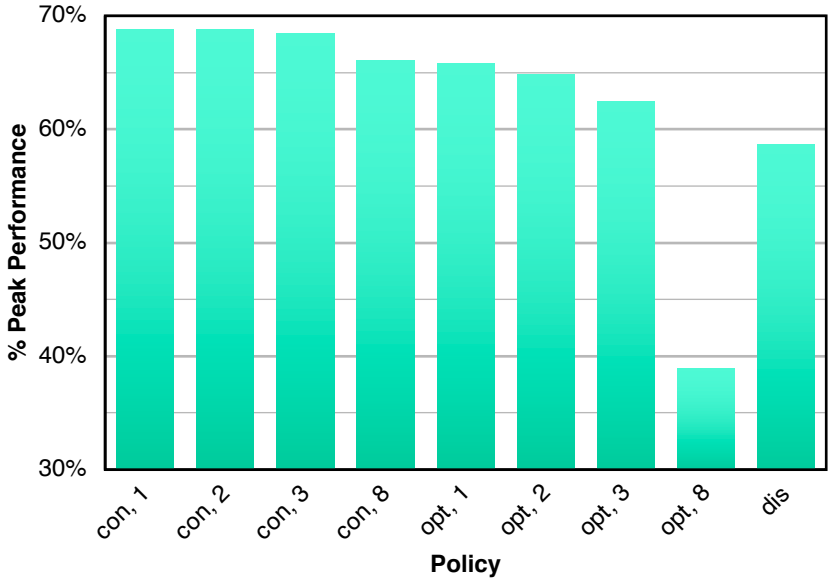
- Protocols are maintained for cache coherence by tracking the state of any sharing of a data block.
  - Example -> Snooping protocols: every cache with a copy of the data from a block of physical memory, also has a copy of the sharing status of the block, but no centralized state is kept.
  - The caches are all accessible via some broadcast medium (bus or network) and all cache controllers monitor (snoop) on the medium to determine whether they have a copy of a block that is requested on a bus or switch access.

Memory Usage: Remarks

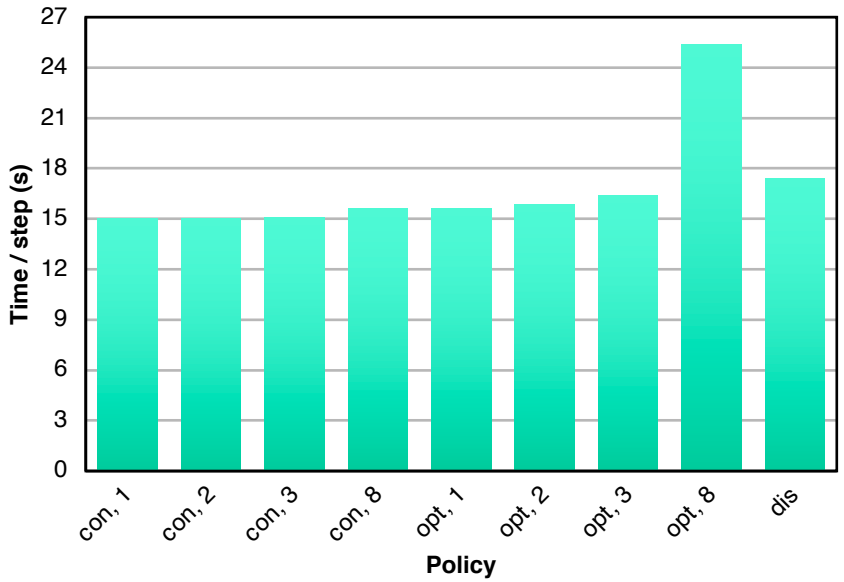
- Software for improved memory usage, assisted by compilers to transform programs.
  - reorganize program to enhance its spatial and temporal locality (loop-oriented programs, using large arrays as the major data structure; e.g. large linear algebra problems) by restructuring the loops (to improve locality and obtain) better cache performance
  - prefetching: a block of data is brought to cache before it is referenced. Hardware to predict accesses that may not be detected by software.
  - cache-aware instructions to optimize memory transfer.

Effect of Data prefetching on BG/Q

- Single BG/Q node, 64 threads



Peak Performance



Time-to-Solution

con: L1P_stream_confirmed
opt: L1P_stream_optimistic
dis: L1P_stream_disable

The roofline model

The Roofline Model

- Proposed by Williams, Waterman and Patterson [1]:
  - Crucial in performance predictions
  - Helpful for software optimization
- “Bound-and-bottleneck” analysis:
  - Provides valuable performance insight
  - Focuses on the primary performance factors
  - Main system bottleneck is highlighted and quantified

[1] Roofline: an insightful visual performance model for multicore architectures, Williams and Waterman and Patterson, Communication to ACM, 2009

Computation-Transfer overlap

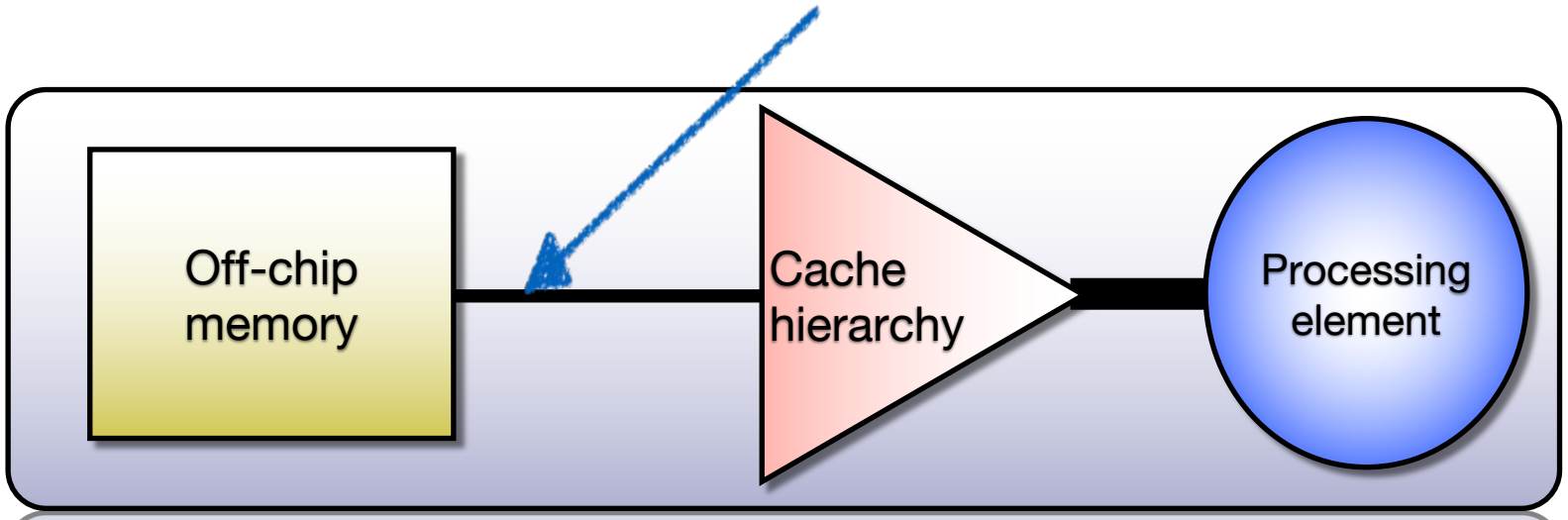
- On CPUs:
  - Superscalar execution (multiple instructions per cycle)
  - In principle: automatic overlap (balanced instructions)
  - In practice: enforced through software prefetching

The Roofline Model

- Main assumptions/issues:
  - The memory bandwidth is the constraining resource (Off-chip system memory)
  - Transfer-computation overlap
  - Memory footprint does not fit in the cache
- We want a model that relates:
  - Computing performance [GFLOP/s]
  - Off-chip memory traffic [GB/s]
- New concept: the operational intensity [FLOP/Byte]

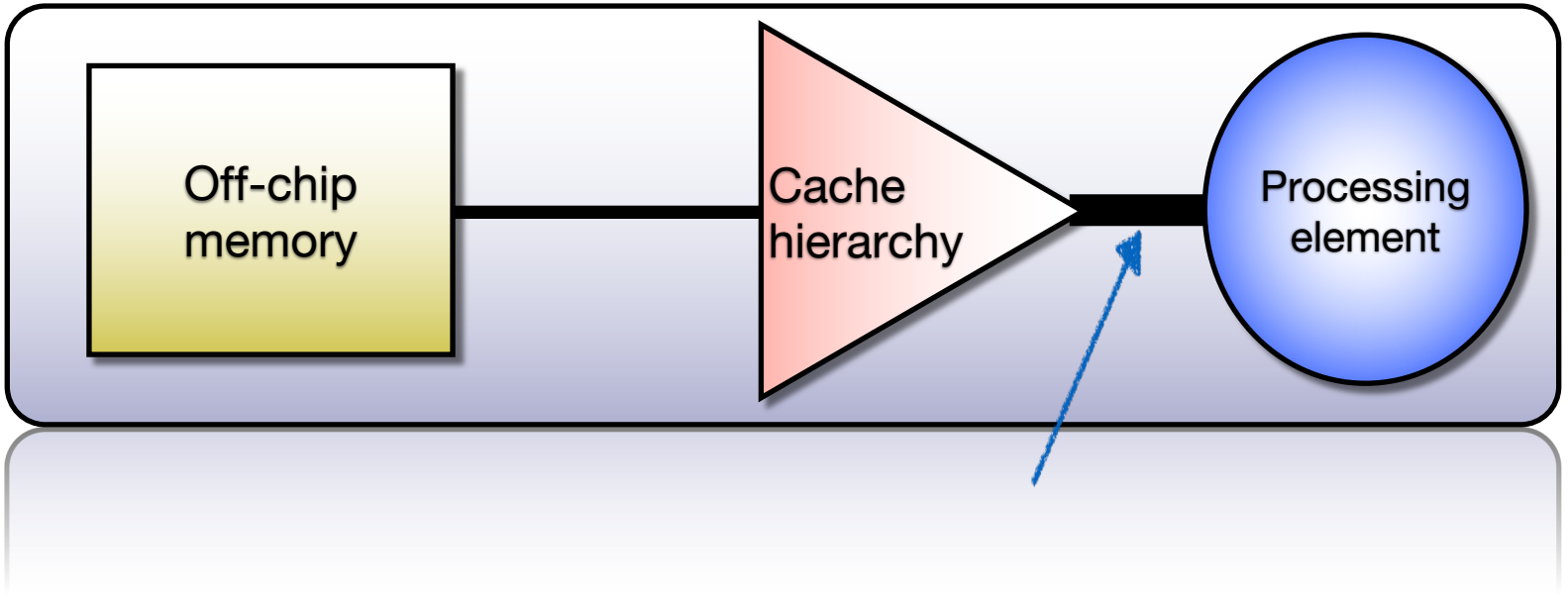
Operational Intensity

- Operations per byte of DRAM traffic
- It measures the traffic between the DRAM and the Last Level Cache (further away from processor)
- It excludes the bytes filtered by the cache hierarchy



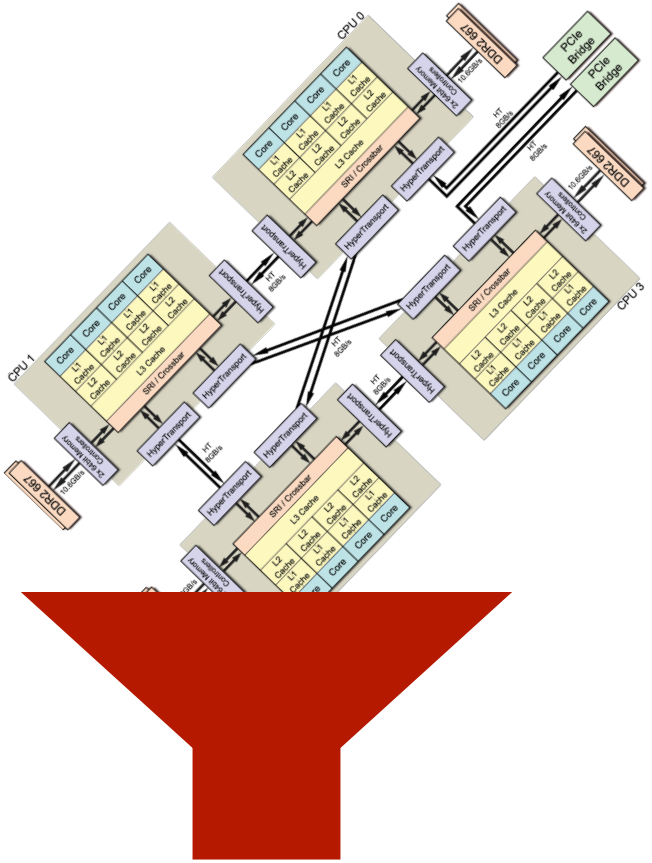
Operational Intensity

- Not equivalent to arithmetic intensity [1], machine balance[2]
  - which refer to traffic between the processor and the cache
- Not forcedly bound to FLOP/Bytes (e.g. Comparison/Byte)



[1] Harris, M. Mapping Computational Concepts To Gpus. In ACM SIGGRAPH Courses, 2005.
[2] Callahan, D., Cocke, J., Kennedy, K. Estimating Interlock and Improving Balance For Pipelined Machines (1988)

Abstraction



GFLOP/s

GB/s

FLOP/B

Operational Intensity:

FLOP-to-byte of off-chip memory transfers

The Roofline Model

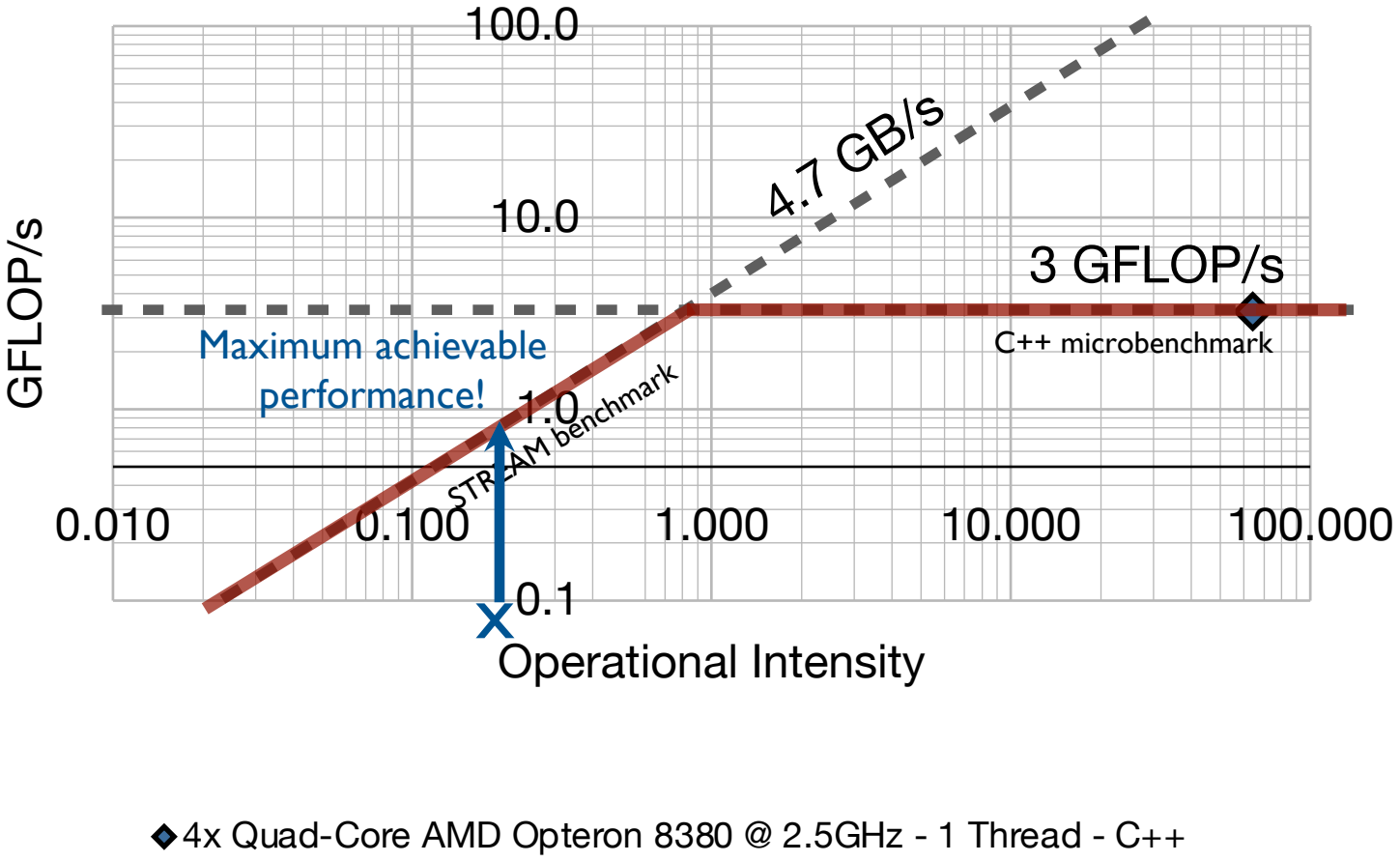
- The roofline is a log-log plot
- It relates:
  - Performance f [FLOP/s] with
  - Operational intensity r [FLOP/Byte]

- Two theoretical regimes for a kernel k:
  - Performance of k is limited by the DRAM bandwidth:

    $\rightarrow f(r_k) = r_k \; b_{peak}$
  - Performance of k is limited by the compute power:

    $\rightarrow f(r_k) = f_{peak}$

The Roofline Model



Nominal Performance

- How to estimate nominal f_{peak} and b_{peak}?
- From the hardware specifications of the platform
- Examples

Processor Clock/sec | Vector size ((SSE, AVX,..) | instructions per clock, FMAs | No. of cores

PP: 2.5 [Ghz] * 4 [SIMD-width] * 2 [issued FLOP/clock] * 16 [cores] = 320 [GFLOP/s]

Memory Clock/sec | Channel size | No. channels | bits/Byte

PB: 1.3 [Ghz] * 64 [bits] * 2 [channels] / 8 [bits/Byte] = 21.3 [GB/s]

Performance=min(OI*PB,PP)

Measured Performance

- Microbenchmarks:
  - STREAM benchmark or similar
  - Nominal peak or vectorized
    - https://github.com/Mysticial/Flops
- Expected discrepancy from nominal quantities:
  - FLOP/s: 90-100% of nominal performance
  - GByte/s : 50-70% of nominal performance
- Discrepancies reveal:
  - Programming skills in extracting system performance
  - Best case scenario for more complex kernels

The Roofline Model

- Run once per platform, not once per kernel
- Estimation of operational intensities (Flops/byte) can be tricky
- What happens if you compute them wrong?

	add $z_i = x_i + y_i$	scale $z_i = \alpha x_i$	triad $z_i = \alpha x_i + y_i$
Intel Xeon W3520	$\frac{1}{12}$ 2 read (x,y) 1 write (z)	$\frac{1}{8}$ 1 read (x) 1 write (z)	$\frac{2}{12}$ 2 read (x) 1 write (z)
4P AMD Opteron 8380	$\frac{1}{16}$ 3 read (x,y,z) 1 write (z)	$\frac{1}{12}$ 2 read (x,z) 1 write (z)	$\frac{2}{16}$ 3 read (x,y,z) 1 write (z)
2P AMD Opteron 2435	$\frac{1}{12}$	$\frac{1}{8}$	$\frac{2}{12}$
NVIDIA Tesla S1070	$\frac{1}{12}$	$\frac{1}{8}$	$\frac{2}{12}$

NOTE: Cache Dependent Numbers

Operational Intensity: Example

- Given

```
for (int ix=1; ix<N-1; ix++)
   out[ix] = in[ix-1]-2*in[ix]+in[ix+1]
```

- where in and out are float arrays of size N

1. What is the number of floating-point operations?

2. What is the number of memory accesses from main memory if:

   a) there is no caching

   b) there is a perfect cache of infinite size

Operational Intensity: Example

```
for (int ix=1; ix<N-1; ix++)
  out[ix] = in[ix-1]-2.*in[ix]+in[ix+1]
```

- Floating point operations: 3*(N-2) FLOP
- Memory accesses (no caching): 4*(N-2) floats accessed
  - every data accessed is counted
- Memory accesses (perfect caching): 2*N-2 floats accessed
  - data is read only once and written only once

Example: 2D Heat Equation

- 2D heat equation:

$$\frac{\partial q}{\partial t} - D\Delta q = 0$$

- q: singe precision (4 Bytes)

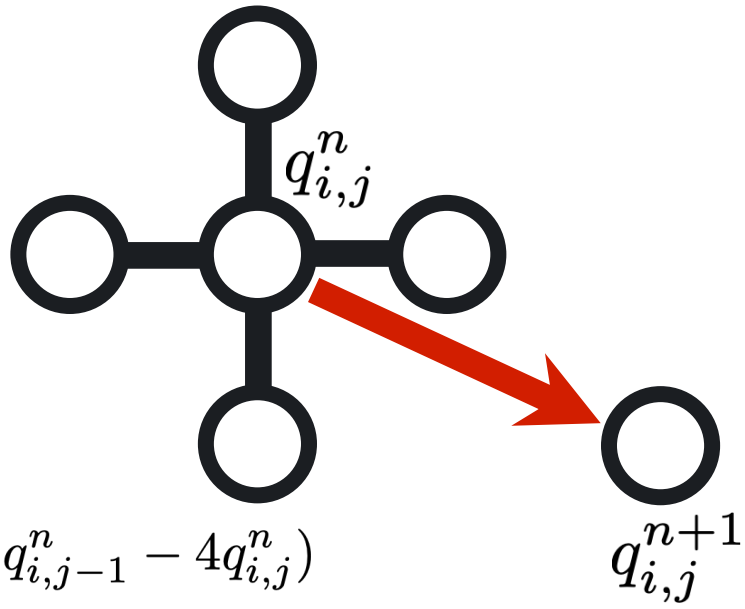
- Algorithm
  - 1. Laplace Operator

$$RHS_{i,j} = C_1(q^n_{i+1,j} + q^n_{i-1,j} + q^n_{i,j+1} + q^n_{i,j-1} - 4q^n_{i,j})$$

  - 2. Forward Euler Operator:

$$q^{n+1}_{i,j} = q^n_{i,j} + \delta t \cdot RHS_{i,j}$$

5 point stencil



A-Priori Performance Analysis

$$RHS_{i,j} = C_1(q^n_{i+1,j} + q^n_{i-1,j} + q^n_{i,j+1} + q^n_{i,j-1} - 4q^n_{i,j})$$

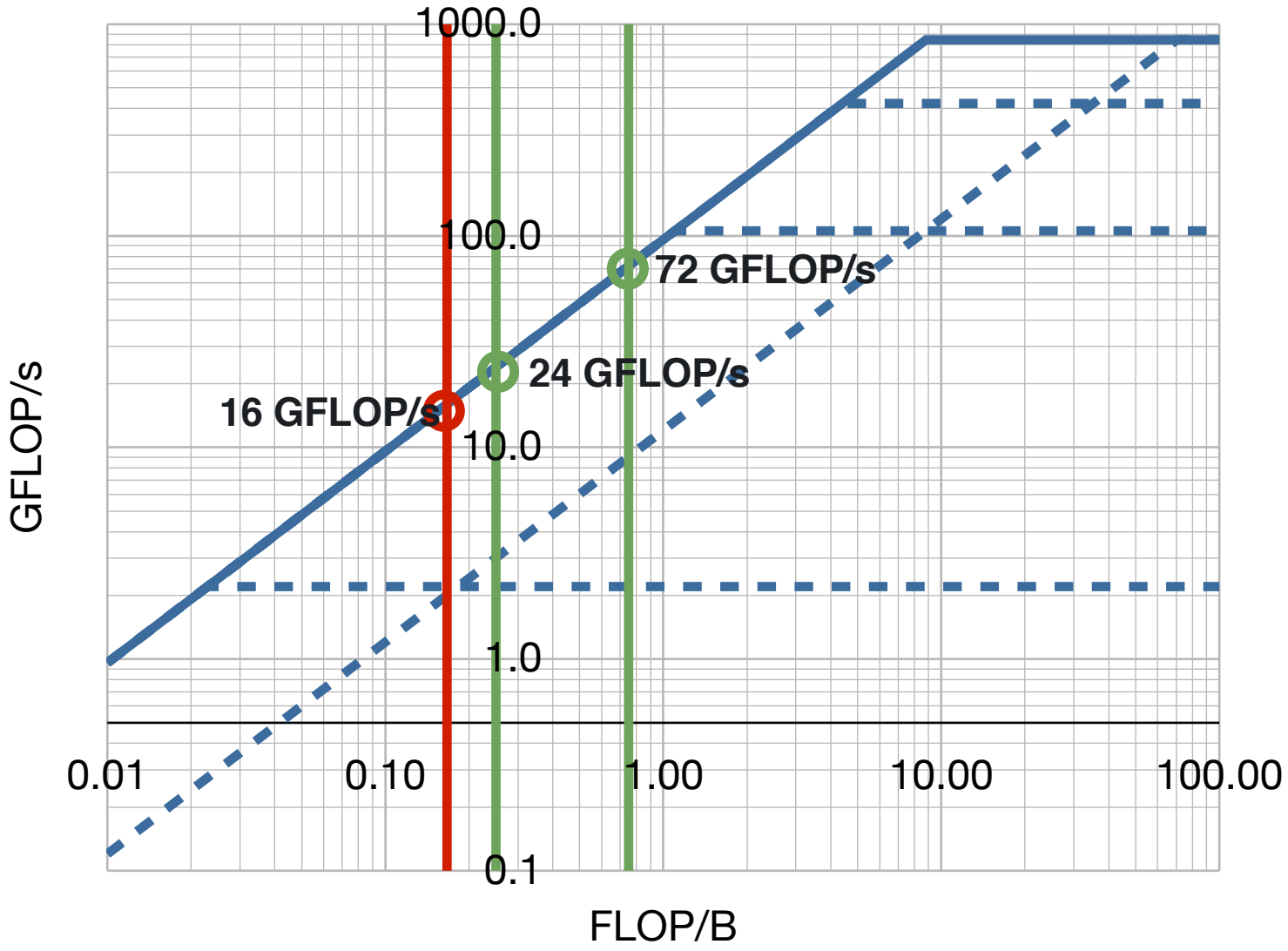
- Floating point operations per point: 4 ADD + 2 MUL
- Memory accesses per point:
  - Worst case: 5 read + 1 write
  - Best case: 1 read + 1 write
- Operational Intensity:
  - Worst case: 6 FLOP / (6*4 B) = 0.25 FLOP/B
  - Best case: 6 FLOP / (2*4 B) = 0.75 FLOP/B

A-Priori Performance Analysis

$$q_{i,j}^{n+1} = q_{i,j}^{n} + \delta t \cdot RHS_{i,j}$$

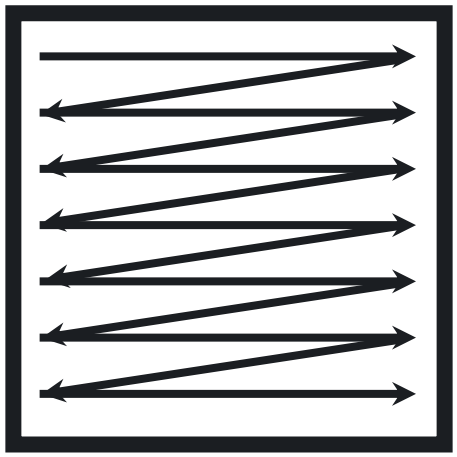
- Floating point operations per point: 1 ADD + 1 MUL
- Memory accesses per point:
  - Worst case: 2 read + 1 write
  - Best case: 2 read + 1 write
- Operational Intensity:
  - Worst case: 2 FLOP / (3*4 B) = 0.17 FLOP/B
  - Best case: 2 FLOP / (3*4 B) = 0.17 FLOP/B

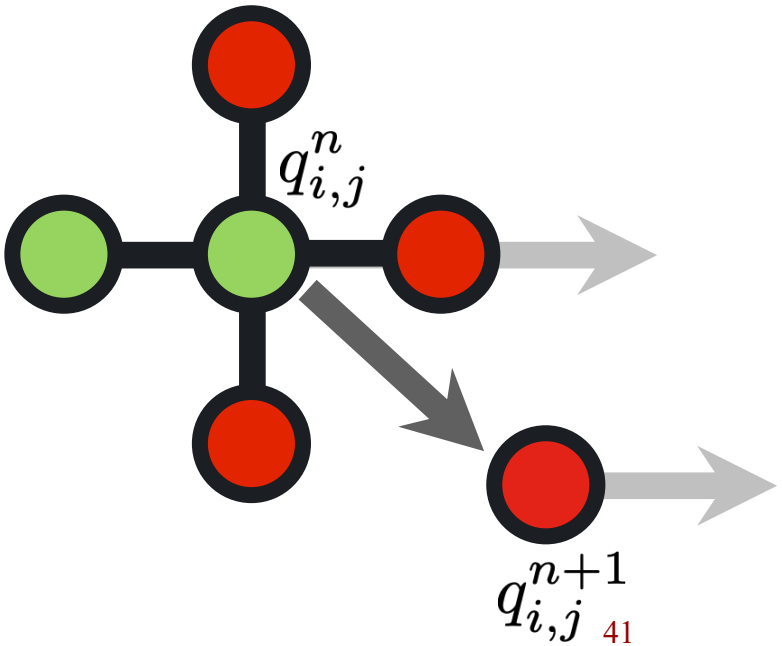
Roofline


1000.0
100.0
10.0
1.0
0.1
GFLOP/s
0.01
0.10
1.00
10.00
100.00
FLOP/B
72 GFLOP/s
24 GFLOP/s
16 GFLOP/s

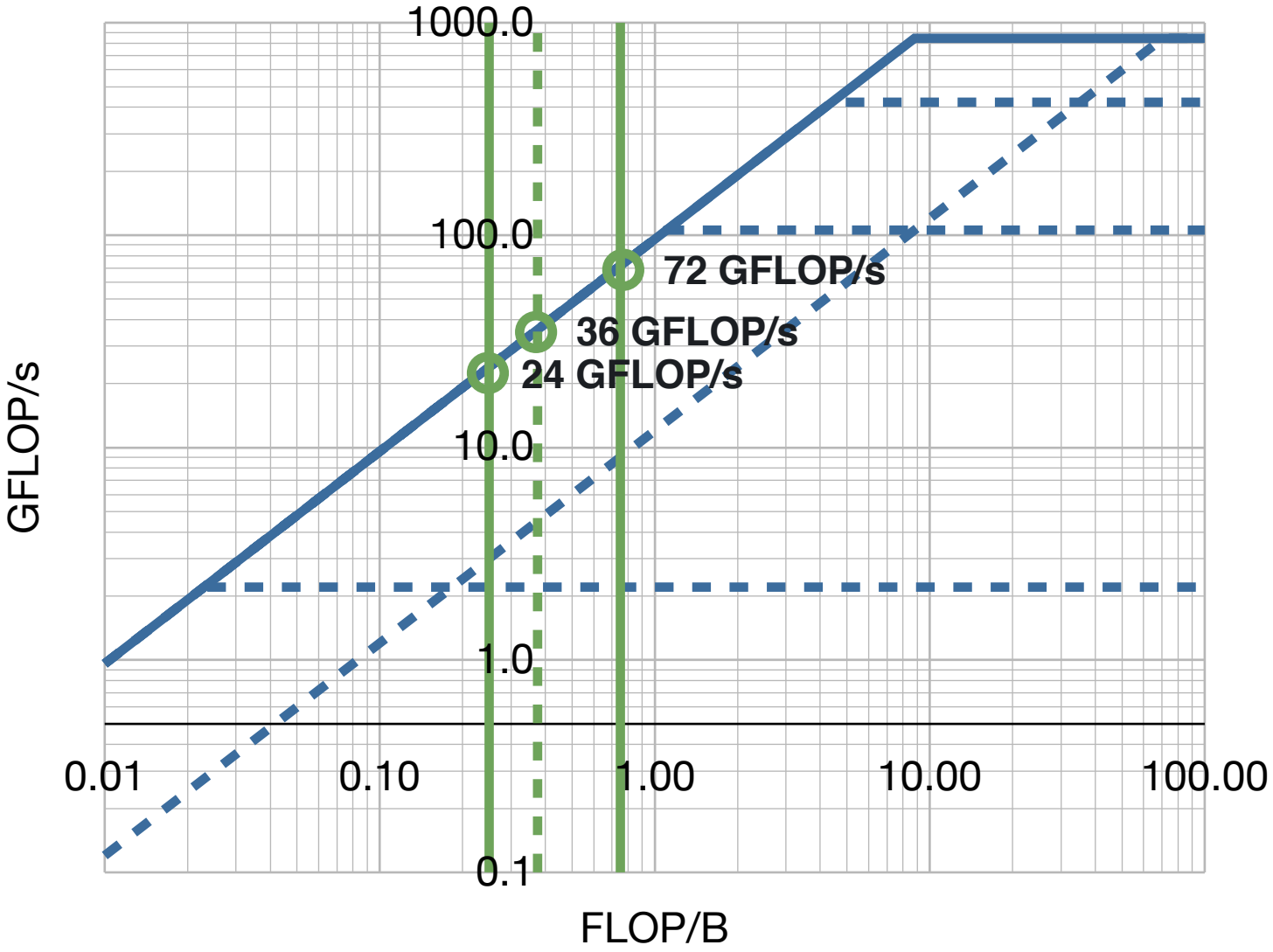
A More Accurate Analysis

- We have locality!
- Memory accesses per point:
  - 3 read + 1 write
- Operational Intensity:
  - 6 FLOP / (4*4 B) = 0.375 FLOP/B



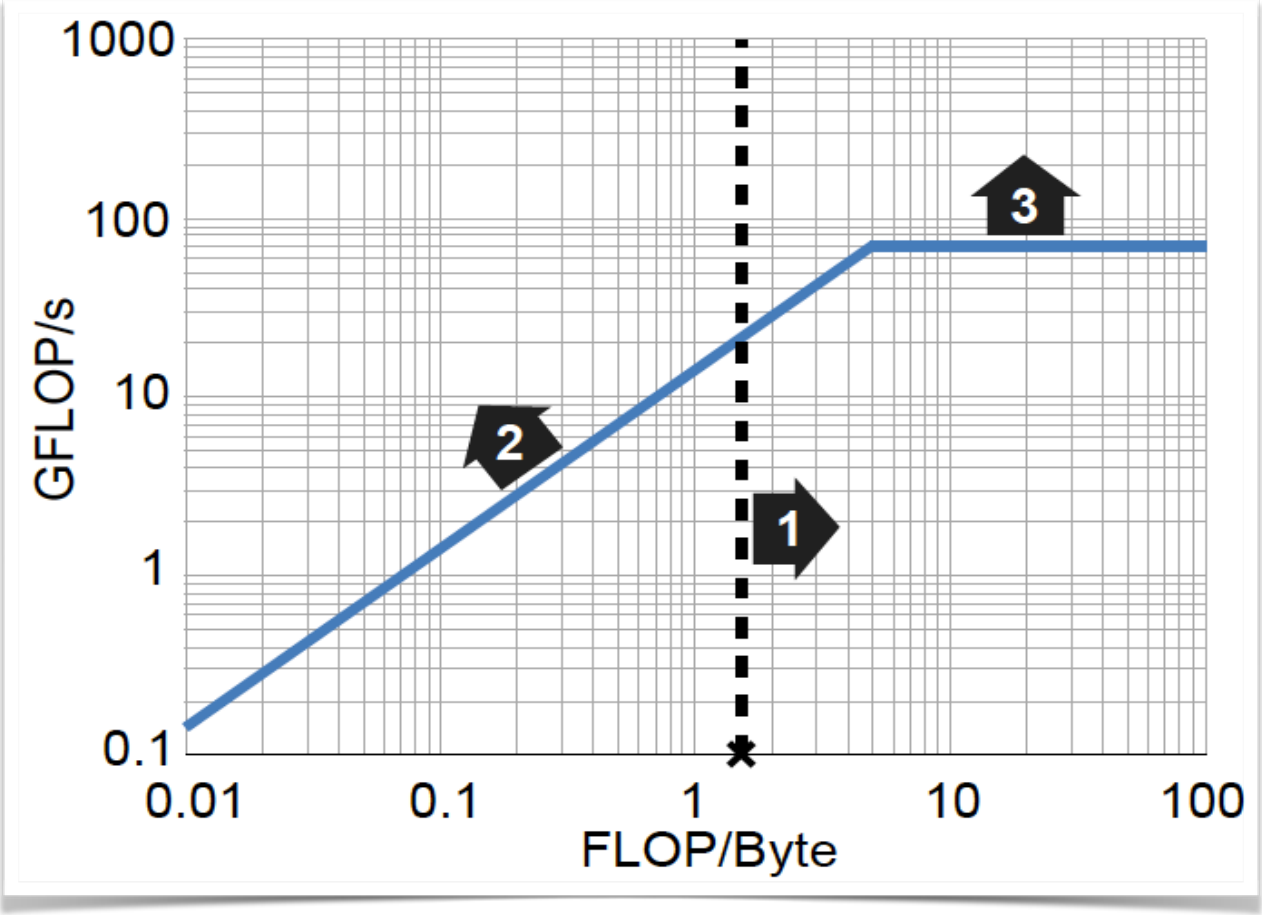


Roofline

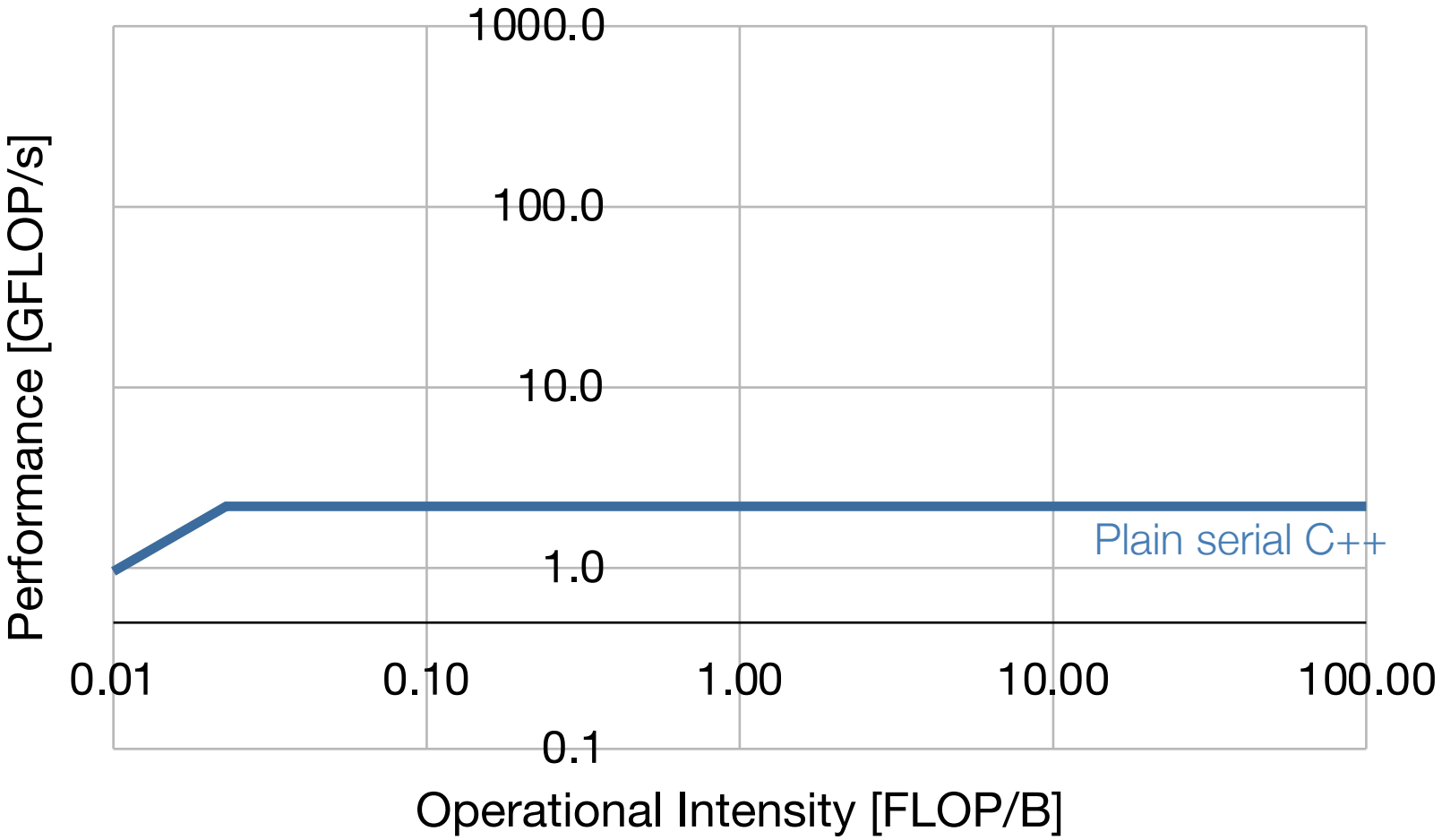

1000.0
100.0
10.0
1.0
0.1
GFLOP/s
0.01
0.10
1.00
10.00
100.00
FLOP/B
72 GFLOP/s
36 GFLOP/s
24 GFLOP/s

Optimization

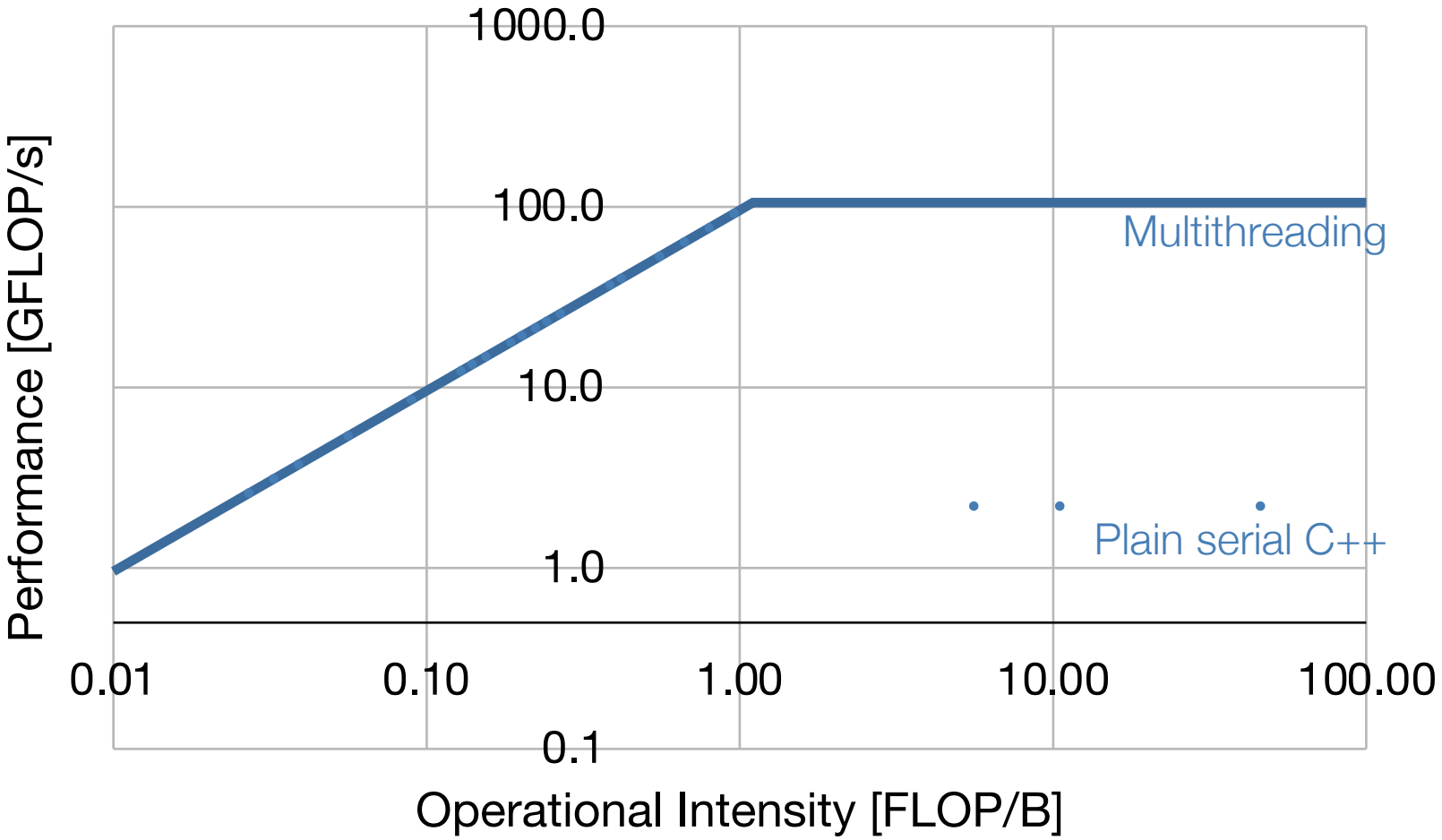
1. Locality
2. Communication
3. Computation



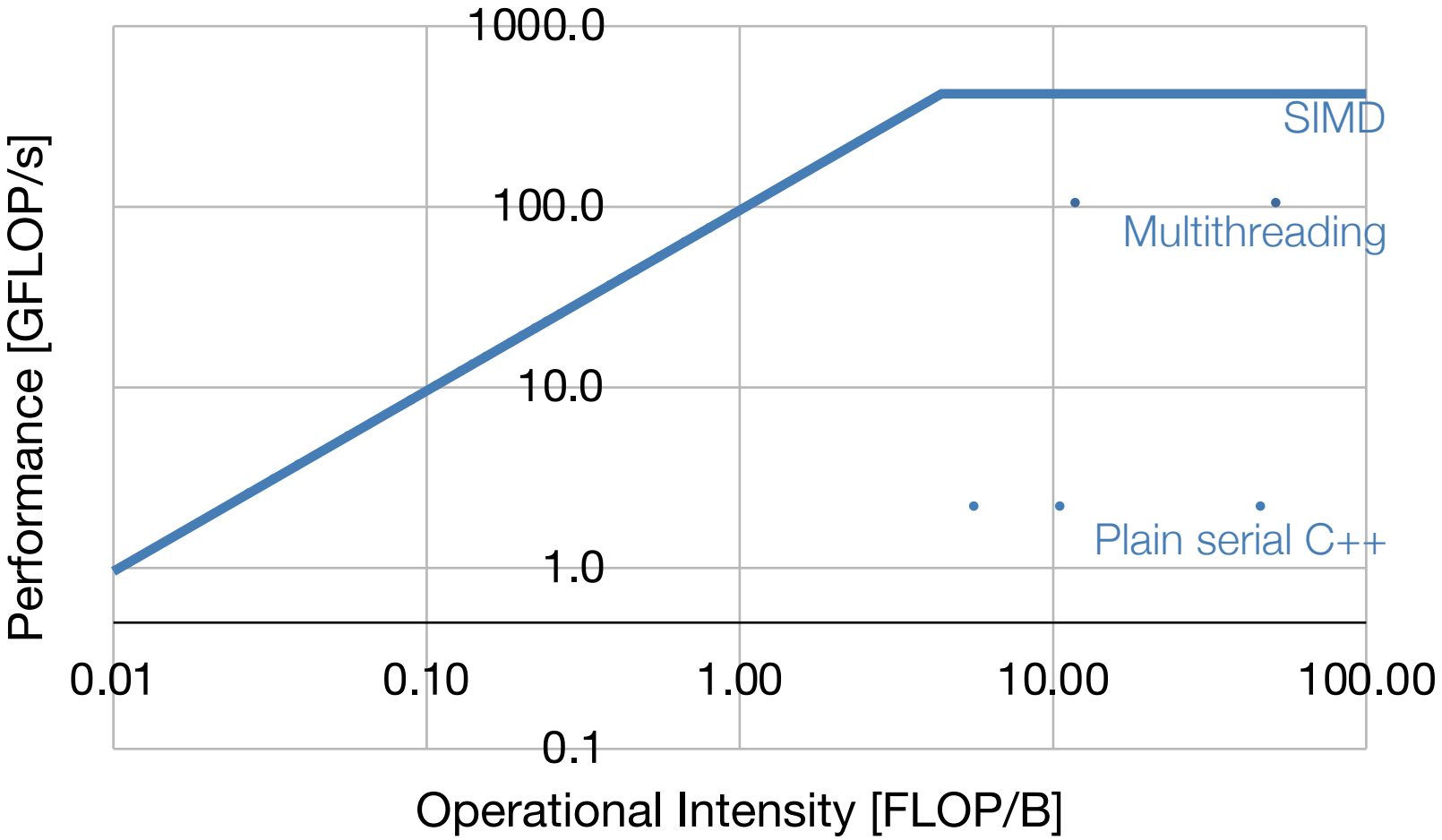
Ceilings on Brutus (single precision)



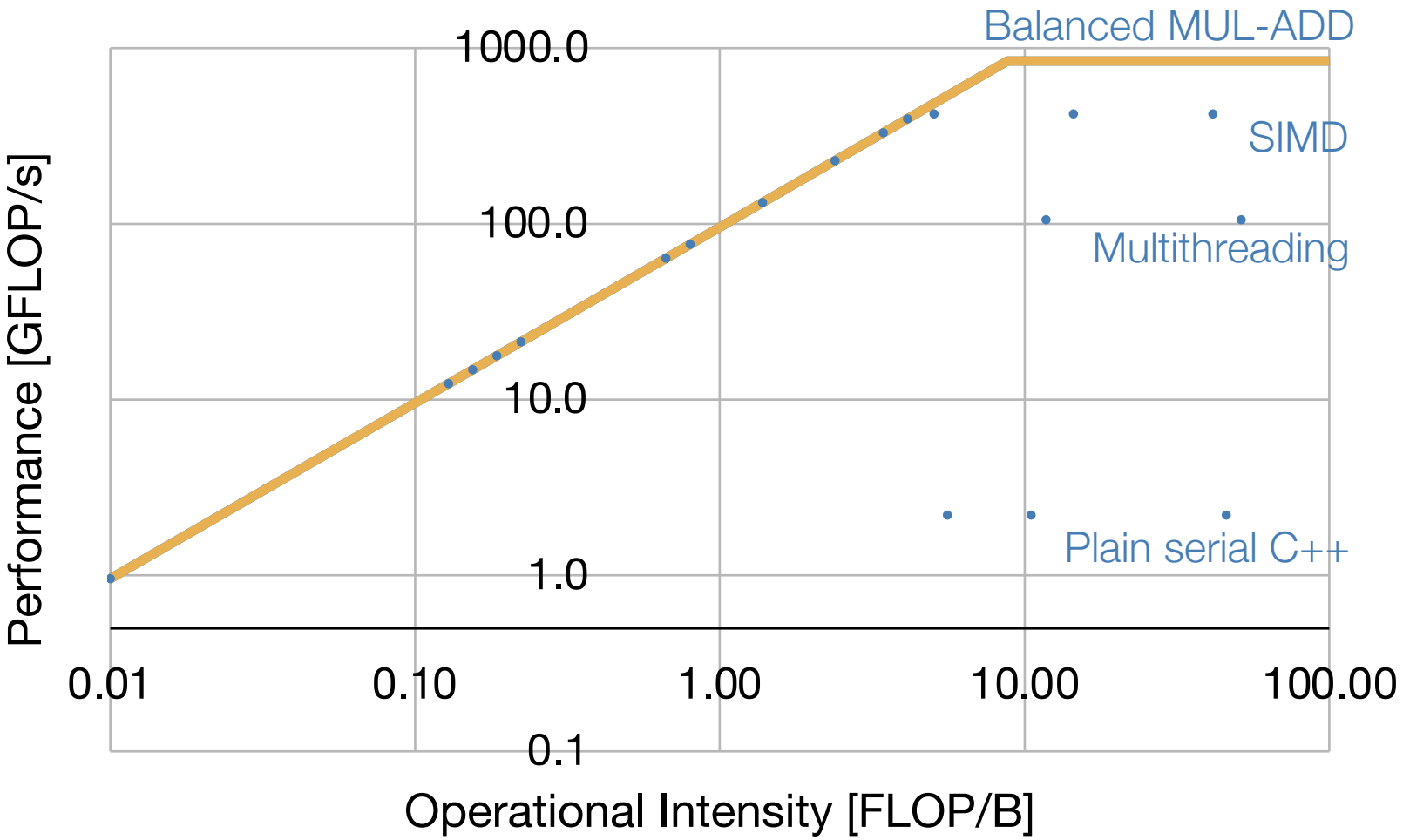
Ceilings on Brutus (single precision)



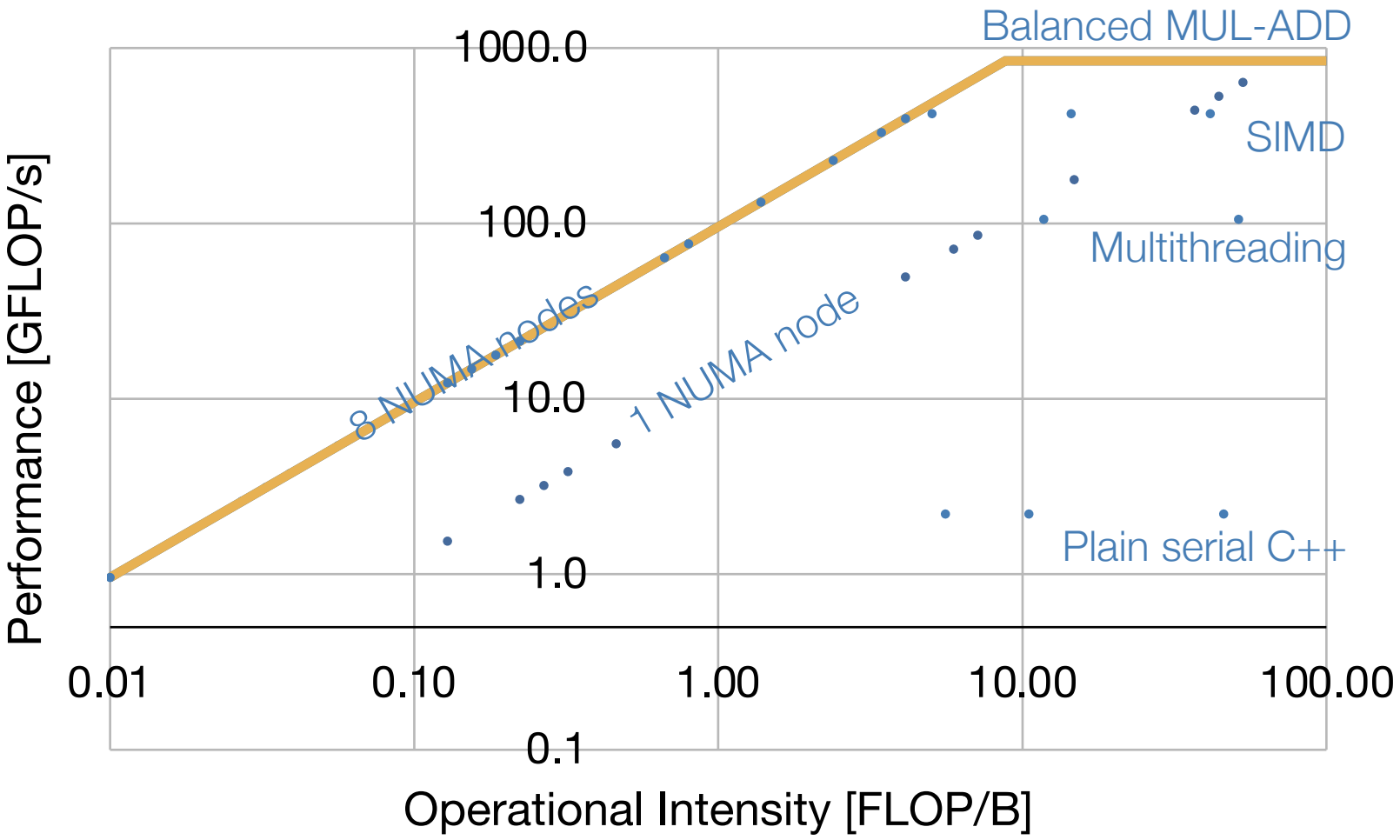
Ceilings on Brutus (single precision)



Ceilings on Brutus (single precision)

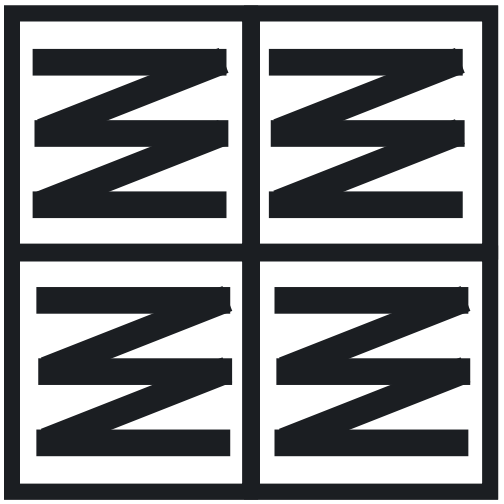


Ceilings on Brutus (single precision)



Improving Locality

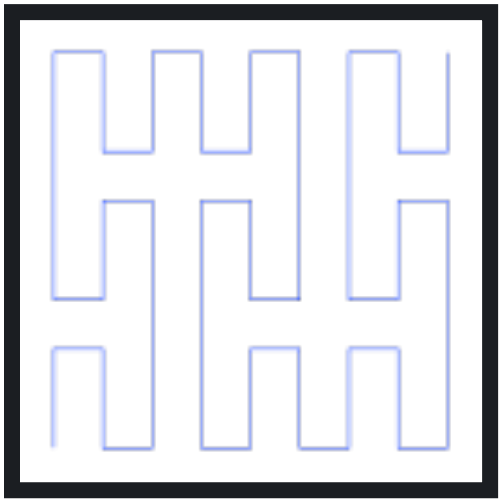
Linear



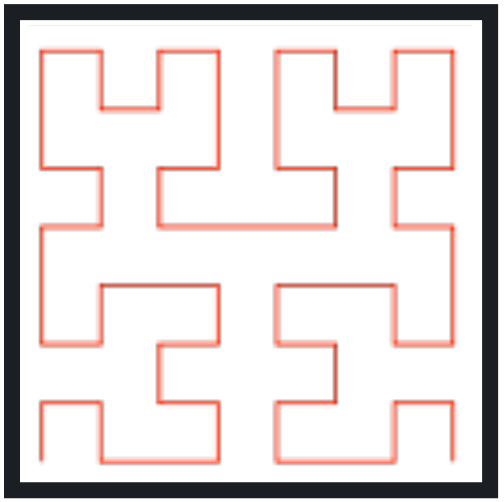
Blocked

Blocking Hierarchy

Morton or Z-Order



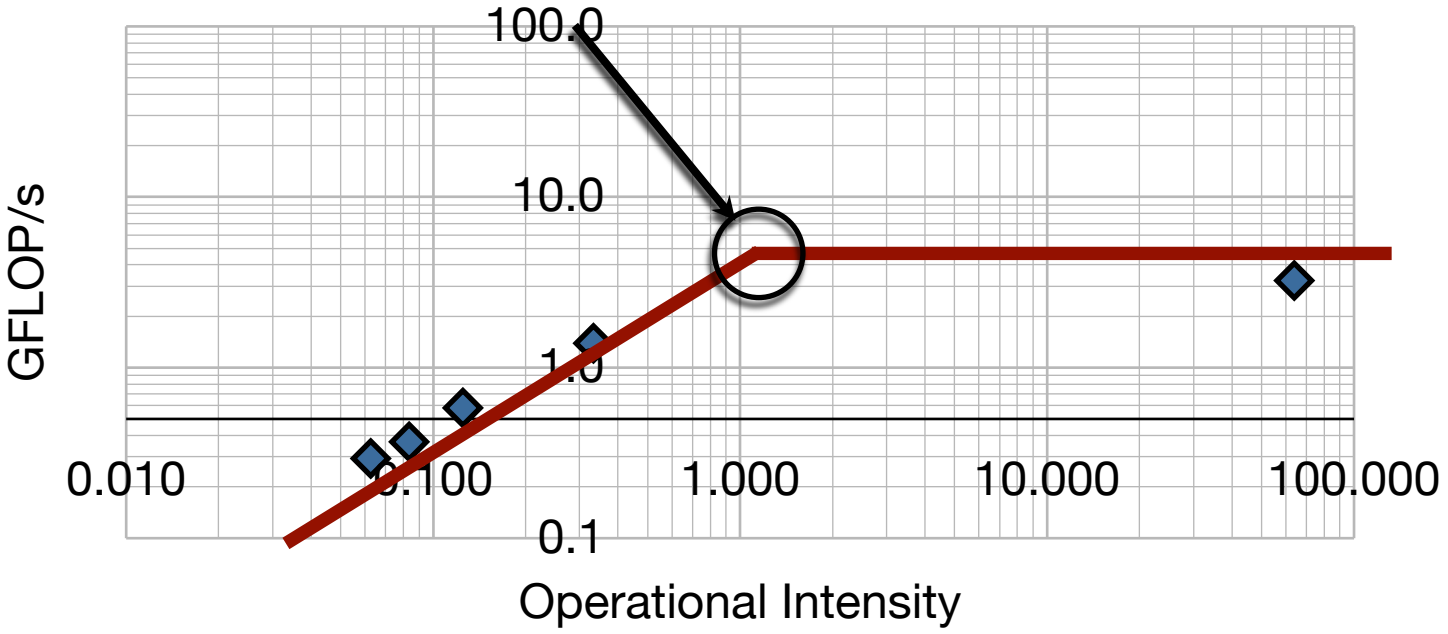
Peano



Hilbert

The Ridge Point

- Ridge point characterizes the overall machine performance
  - Ridge point “to the left”: it is relatively easy to get peak performance
  - Ridge point “to the right”: it is difficult to get peak performance

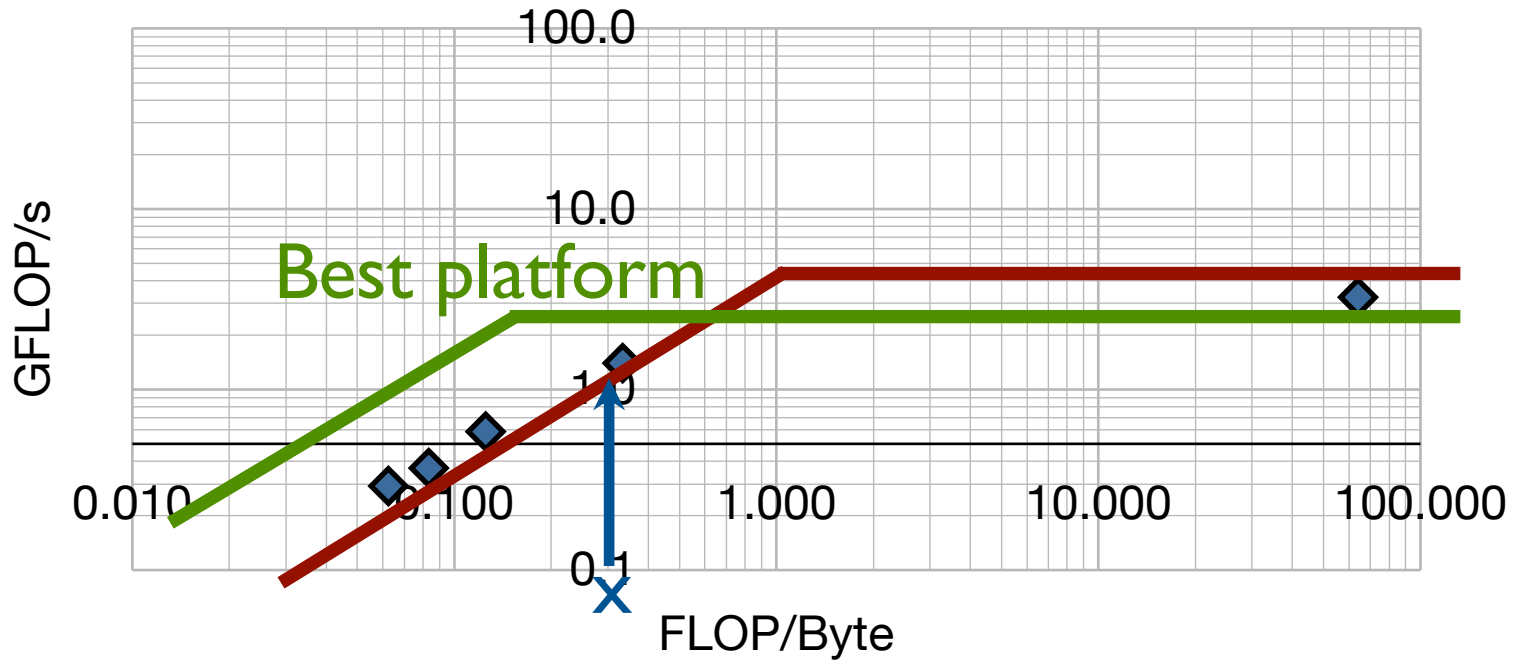


What does it mean “a ridge point to the right” anyway?

Production Software

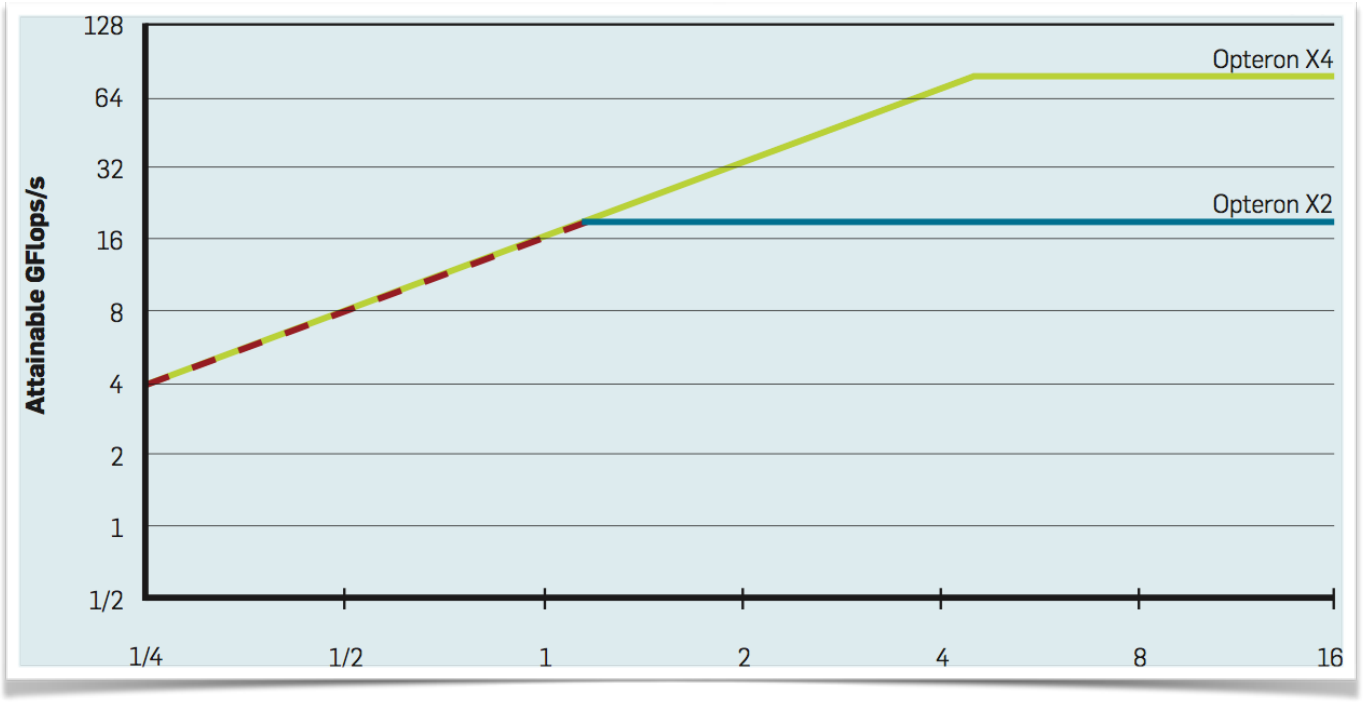
- Assumption: production-ready software
  - Limited set of algorithms
  - Fixed set of kernels
  - Fixed operational intensities

} Best hardware solution?



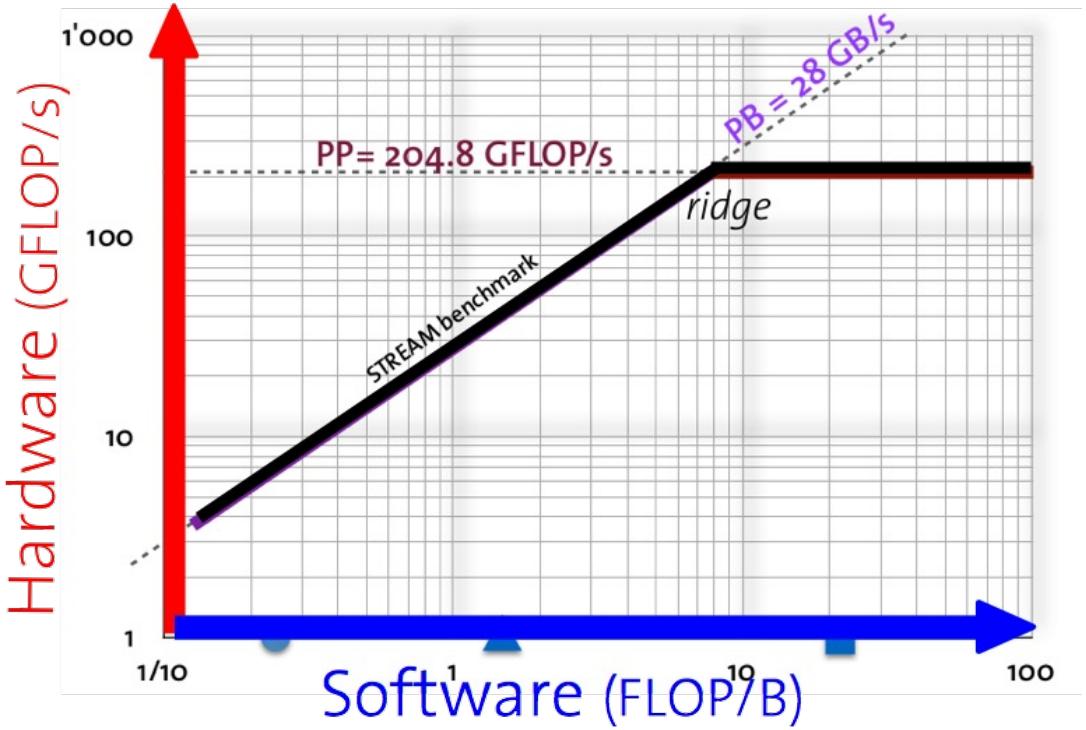
Is Moore worth?

- It depends:
  - On the ridge point
  - On the operational intensity of the considered kernels



The Roofline Model: Summary

- It visually relates hardware with software
- Performance = min(PB x OI, PP)
- Ridge point characterizes the model



Conclusions

- When is the roofline model useless?
  - When you discuss performance in terms to time-to-solution.
- When is the roofline model crucial?
  - When you want to optimize your code (data reuse, ceilings)
  - To predict maximum achievable performance (roofline, ridge point)
  - To systematically assess your performance (roofline, op. int.)
- What do you do if all your kernels have a bad op. int.?
  - Either live with it
  - Go back to equations, pick better discretization schemes/algorithms (leading to a higher op. int.)
  - Wanted: less simulation steps, but more costly (high order schemes)

Hands-on

- Peak performance
- Memory bandwidth
- Memory layout (struct of arrays vs array of structures)
- Matrix multiplication