

Παράλληλη Επεξεργασία

Εαρινό Εξάμηνο 2022-23
«Memory and Performance
- Roofline Model»

Παναγιώτης Χατζηδούκας, Ευστράτιος Γαλλόπουλος

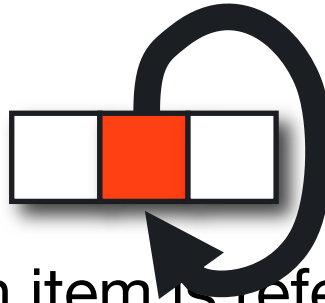
Outline

- Memory
- Performance
 - Amdahl's law
 - Strong and weak scaling
- Roofline Model

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

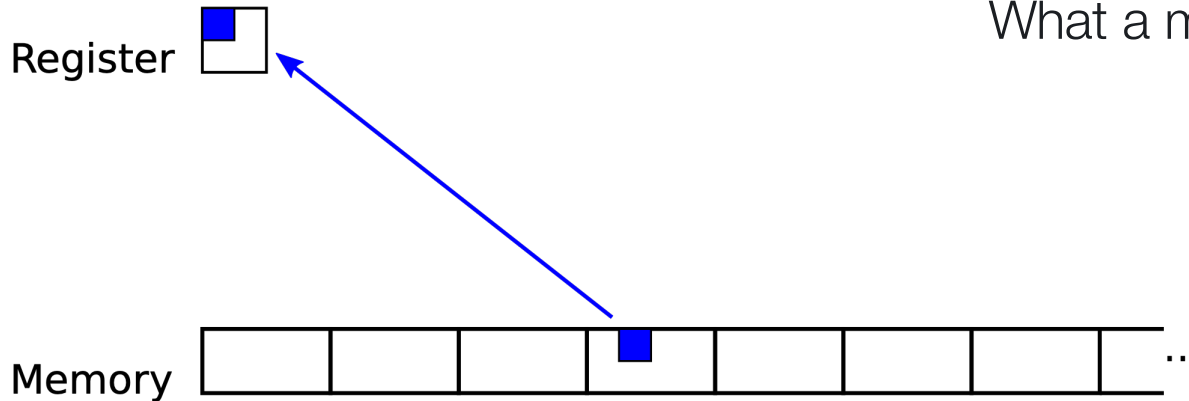


- Spatial locality: if an item is referenced, items whose addresses are close by, will tend to be referenced soon
 - Data access: sequential access to elements of array

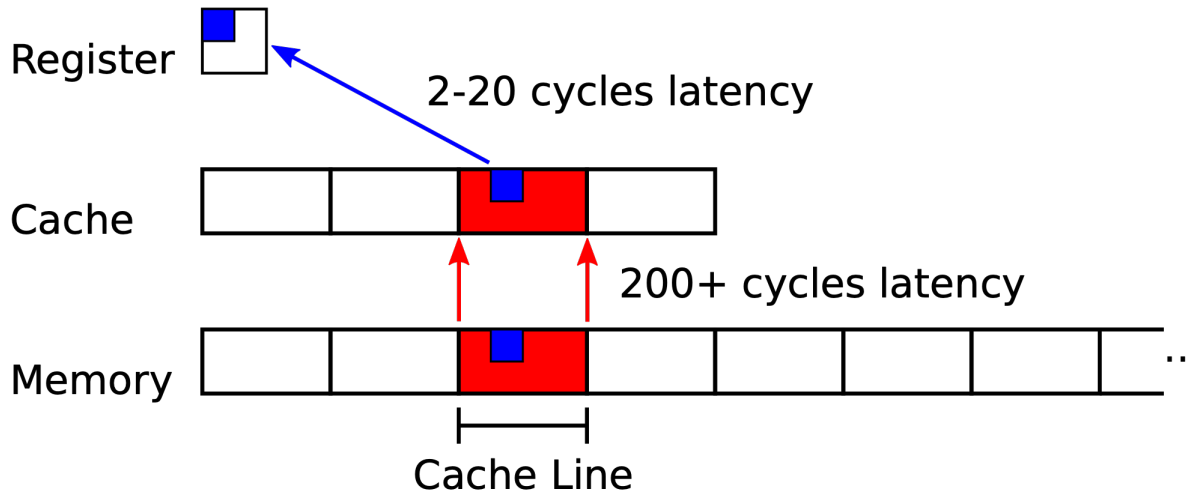


Register and Caches

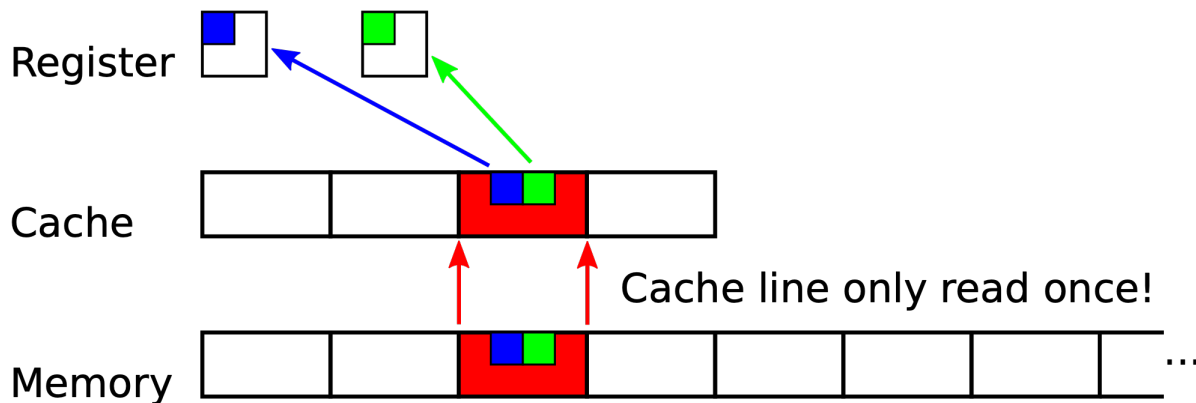
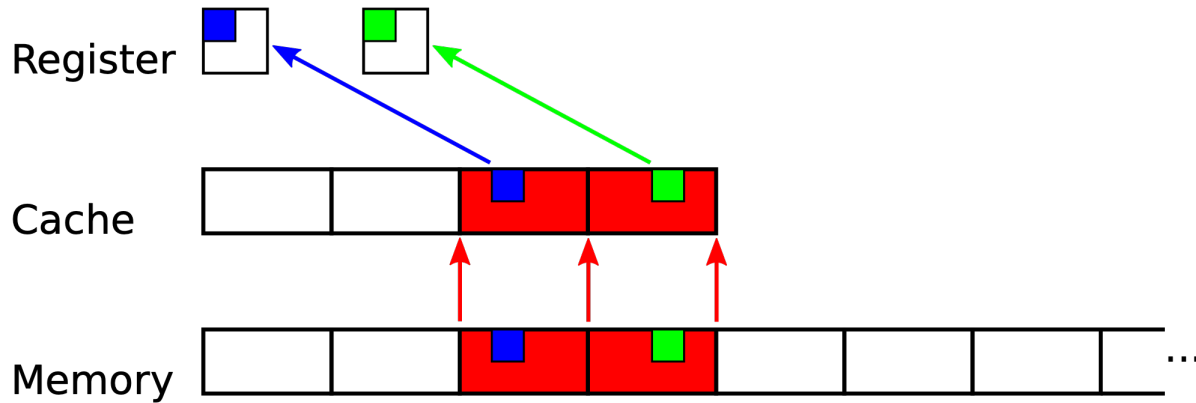
What a memory access looks like



What really happens



Spatial Locality



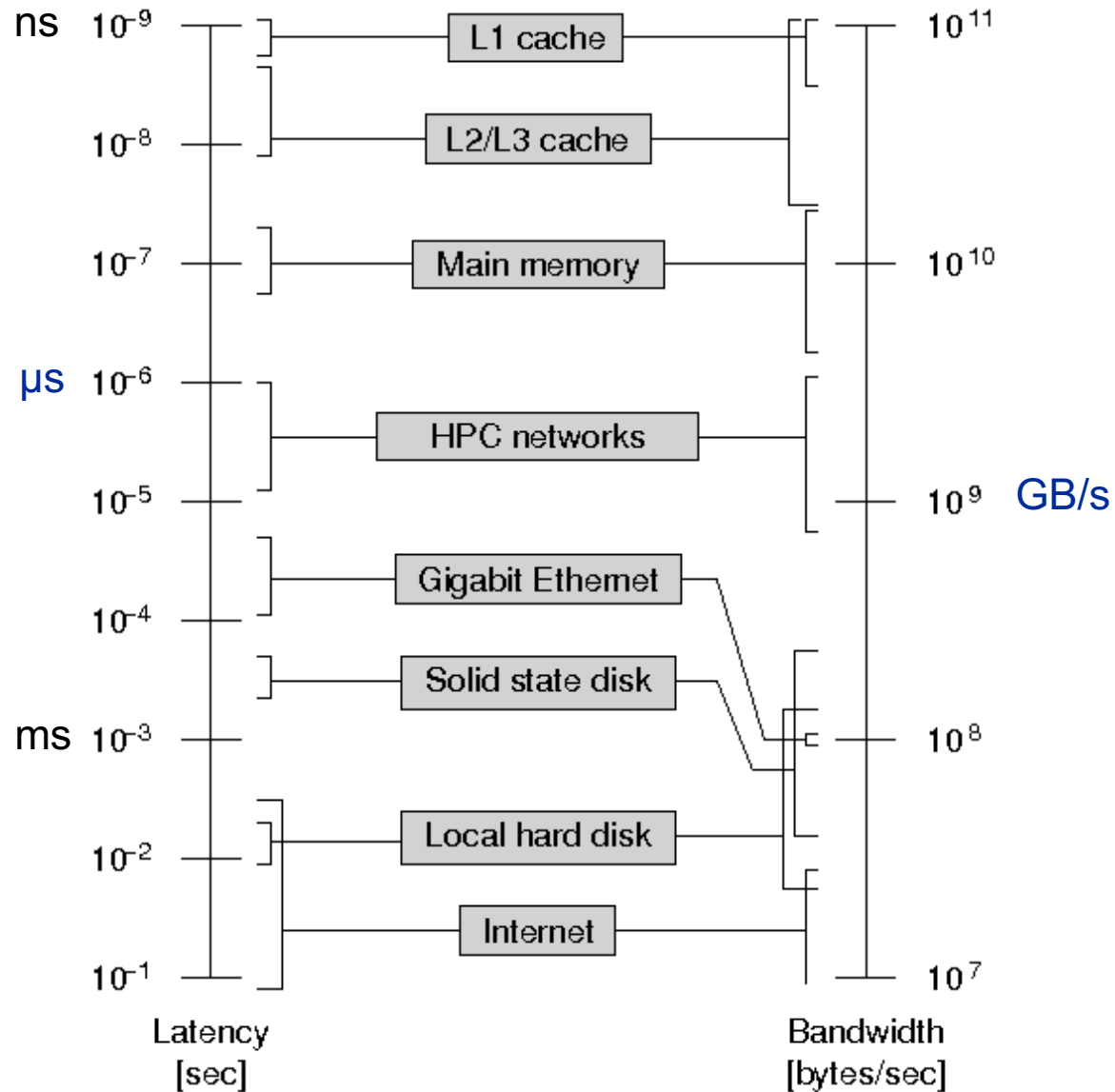
Better spatial locality!

Memory Hierarchy

- Multiple levels of memory with different speeds and sizes.

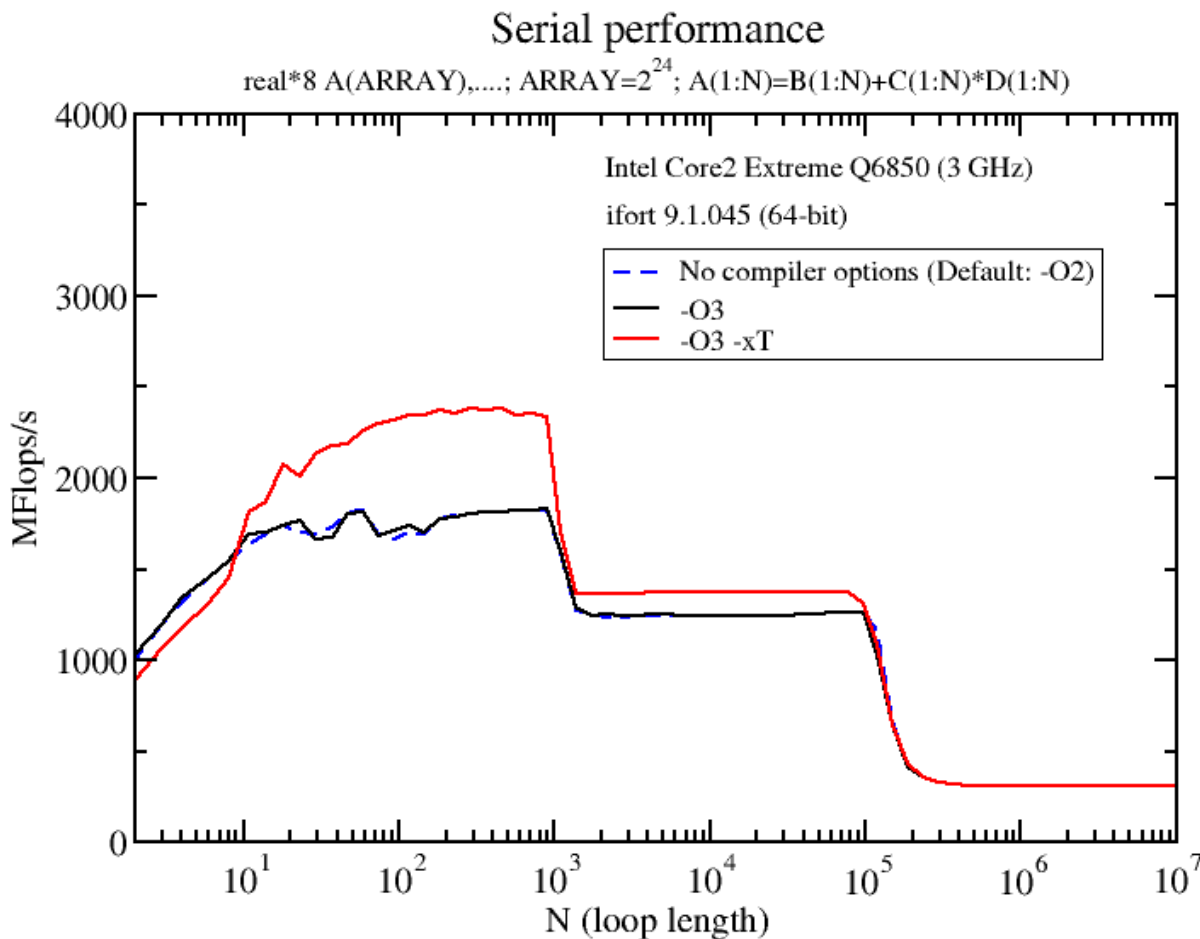
Memory	Access time (ns)	\$ per GB (2008)
SRAM	0.5 - 2.5	2000 - 5000
DRAM	50 - 70	20 - 75
Magnetic Disk	5×10^6 - 20×10^6	0.2 - 2

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)

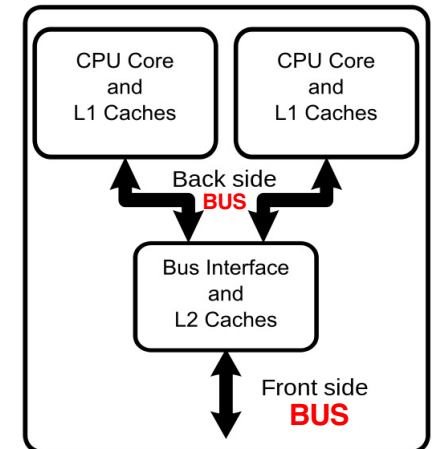
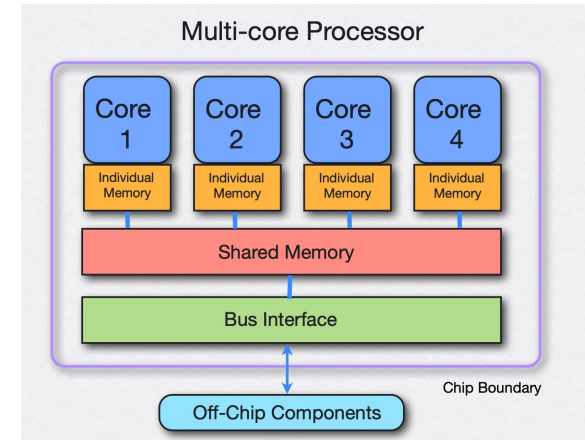
- Suppose that
 - cache access time = 10ns
 - main memory access time = 200ns
 - cache hit rate = 99%
- What is the EAT for non-overlapped access?

$$\text{EAT} = 0.99(10\text{ns}) + 0.01(10\text{ns} + 200\text{ns}) = 9.9\text{ns} + 2.1\text{ns} = 12\text{ns}$$

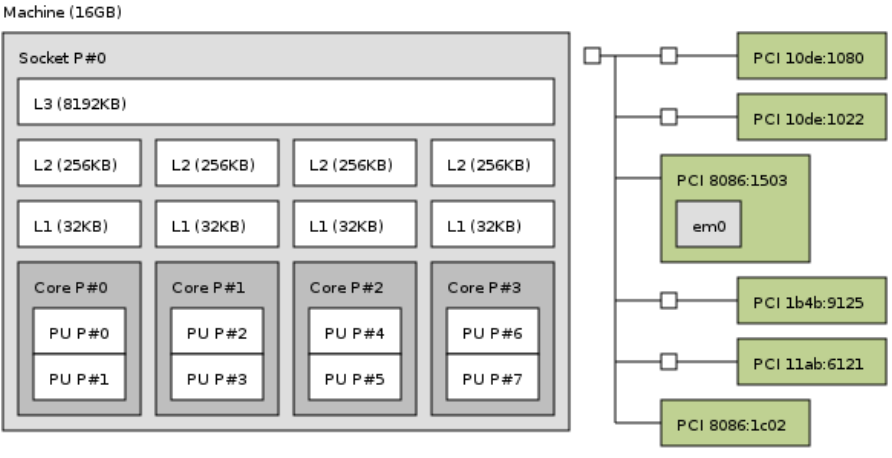
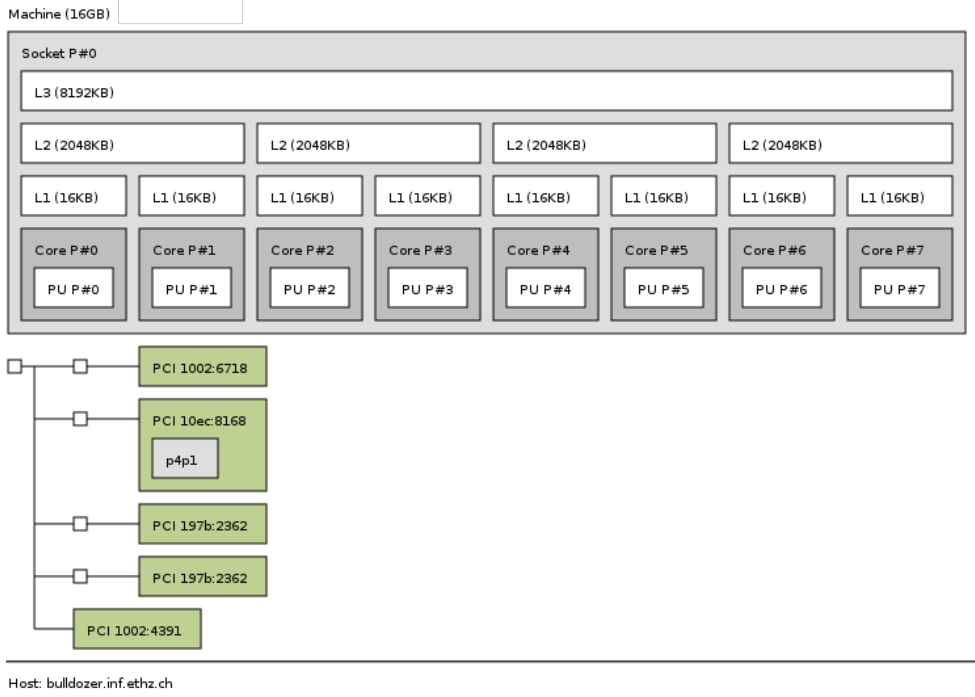
Caches and Multiprocessors

- 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)



Host: cyrus.ethz.ch

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)

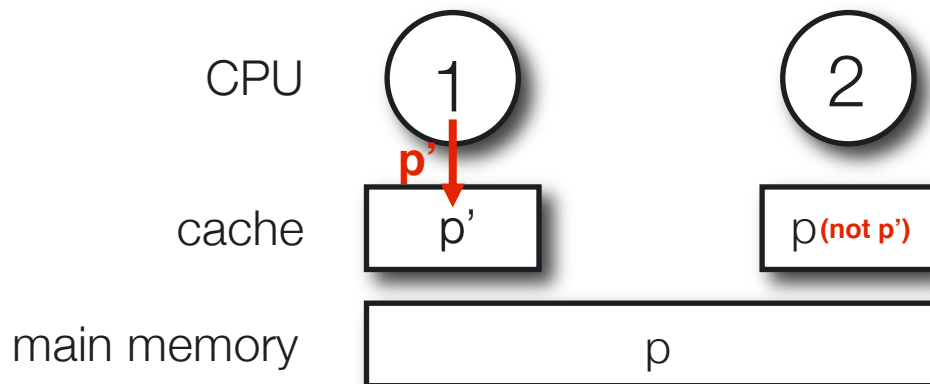
1st NUMA NODE

- 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. \Rightarrow 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

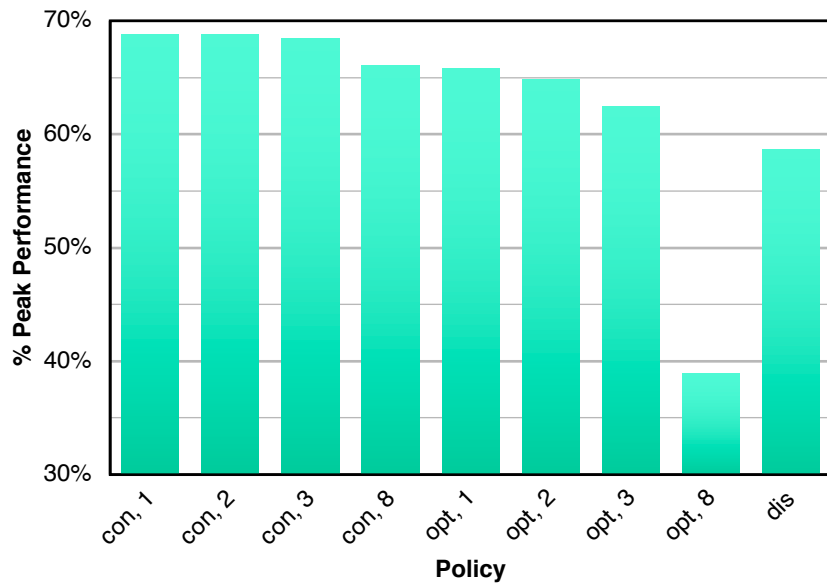
- Protocols are maintained for cache coherence by tracking the state of any sharing of a data block.
 - Example -> Snooping protocols: every cache with a copy of the data from a block of physical memory, also has a copy of the sharing status of the block, but no centralized state is kept.
 - The caches 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

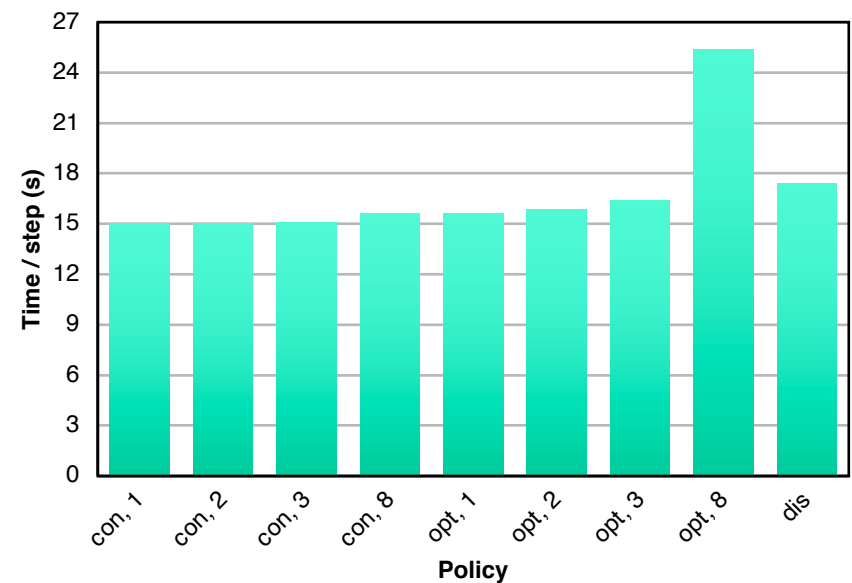
- Software for improved memory usage, assisted by compilers to transform programs.
 - reorganize program to enhance its spatial and temporal locality (loop-oriented programs, using large arrays as the major data structure; e.g. large linear algebra problems) by restructuring the loops (to improve locality 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

Performance

The Difficulty of Parallel Processing Programs

- We must get efficiency - else use single processor
- Instruction level parallelism done by the compiler can help (out of order execution, etc)

- Challenges
 - scheduling
 - load balancing
 - time for synchronization
 - overhead for communication between parts

Amdahl's Law

- How much can a problem be improved?
- e.g.: We want a speed up of 90x faster with 100 processors

$$\text{Exec time after improvement} = \frac{\text{Exec time affected by improvement}}{\text{Amount of improvement}} + \text{Exec time unaffected}$$
$$\text{Speed-up} = \frac{\text{Exec time before}}{(\text{Exec time before} - \text{Exec time affected}) + \frac{\text{Exec time affected}}{100}}$$

Processors

$$\text{Speed-up} = \frac{1}{(1 - \text{Fraction time affected}) + \frac{\text{Fraction time affected}}{100}}$$

Assumption of perfect load balancing

Amdahl's Law : Example 1

- We want a speed up of 90x faster with 100 processors

$$90 = \frac{1}{(1 - \text{Fraction time affected}) + \frac{\text{Fraction time affected}}{100}}$$
$$\rightarrow \text{Fraction time affected} = \frac{89}{89.1} = 0.999$$

- So to get a speed-up of 90 from 100 processors the sequential part can only be 0.1%

Amdahl's Law : Example 2

- Suppose we want to perform two sums:
 - sum of 10 scalar variables
 - matrix sum of pairs of 2D arrays with dimension 10x10
- What speedup you get with 10 and 100 processors?
- Calculate the speedups assuming matrices grow to 100x100.
- Assume time t for the performance of an addition. Then there are (for 100 processors) 100 additions that scale and 10 that do not.

$$\text{Time for 1 processor} = 100t + 10t = 110t$$

Amdahl's Law : Example 2

- For 110 numbers we get 55% of the potential speedup with 10 processors but only 10% of 100 processors.

Time for 10 processors =

$$\begin{aligned}\text{Time after improvement} &= \frac{\text{Exec time affected}}{\text{Amount of improvement}} + \text{Exec time unaffected} \\ &= \frac{100t}{10} + 10t = 20t\end{aligned}$$

$$\text{Speed-up}_{10} = \frac{110t}{20t} = 5.5 \text{ (out of 10)}$$

$$\text{Time for 100 processors} = \frac{100t}{100} + 10t = 11t$$

$$\text{Speed-up}_{100} = \frac{110t}{11t} = 10 \text{ (out of 100)}$$

Amdahl's Law : Example 2

- What happens when we increase the matrix order?

Time for 1 processor = $10t + 10000t = 10010t$

$$\text{Exec time after improvement} = \frac{10000t}{10} + 10t = 1010t$$

$$\text{Speed-up}_{10} = \frac{10010t}{1010t} = 9.9 \text{ (out of 10)}$$

$$\text{Exec time after improvement} = \frac{10000t}{100} + 10t = 110t$$

$$\text{Speed-up}_{100} = \frac{10010t}{110t} = 91 \text{ (out of 100)}$$

- For larger problem size we get 99% with 10 processors and 91% with 100 processors

Scaling

- Strong scaling:
speed-up on multiprocessors **without** increase on problem size
 - Amdahl's law considers the strong scaling
- Weak scaling:
speed-up on multiprocessors, while increasing the size of the problem **proportionally** to the increase in the number of processors

Strong Scaling

- Strong scaling: defines how the solution time varies with increasing number of processors p for a fixed total problem size (i.e. fix workload is split among cores):

► **Speedup** for strong scaling:

$$S(p) = \frac{T(1)}{T(p)}$$

► **Efficiency** for strong scaling:

$$E_s(p) = \frac{S(p)}{p}$$

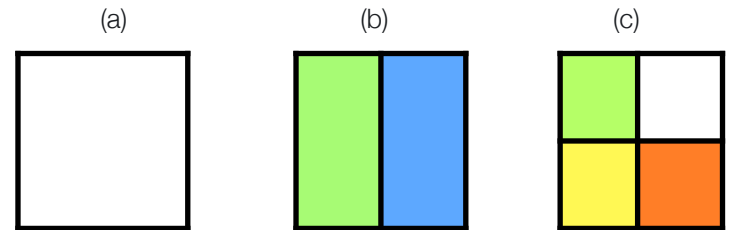
$T(1)$ = time of one thread to process the data

$T(p)$ = time of p threads to process the same data

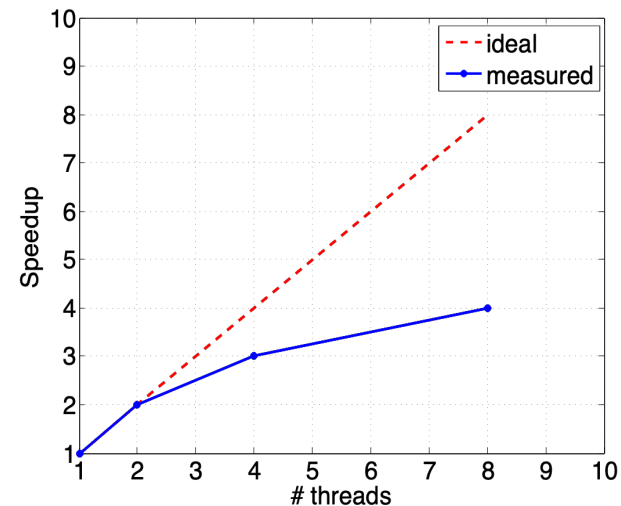
Example:

System with fix problem size N (e.g. # particles)

Problem size:	Execution time:	Speedup:
$N = 100$	$T(1) = 12.0$ s	$S(1) = 1.0$
$N = 100$	$T(2) = 6.0$ s	$S(2) = 2.0$
$N = 100$	$T(4) = 4.0$ s	$S(4) = 3.0$
$N = 100$	$T(8) = 3.0$ s	$S(8) = 4.0$



Split of the fixed problem size over a) 1 core, b) 2 cores, c) 4 cores



Weak Scaling

- Weak scaling: defines how the solution time varies with the number of processors p for a fixed problem size per processor

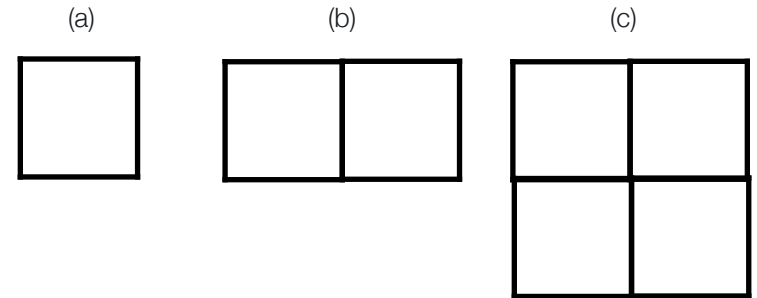
▶ Speedup for weak scaling doesn't make sense

▶ **Efficiency** for weak scaling:

$$E_w(p) = \frac{T(1)}{T(p)}$$

$T(1)$ = time of one thread to process the data

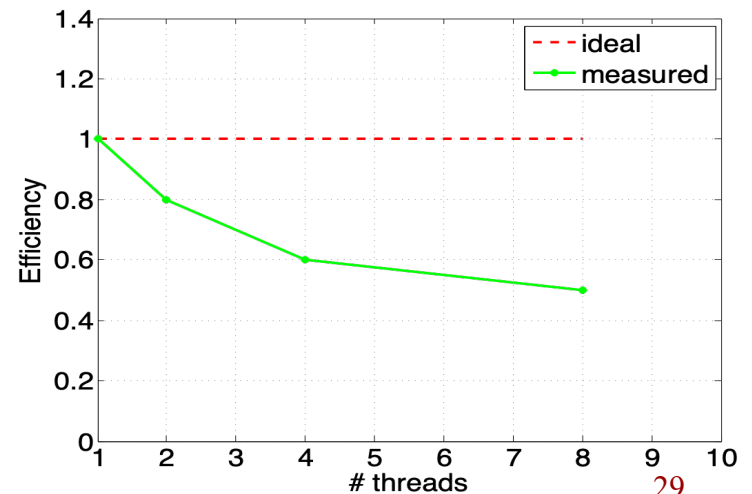
$T(p)$ = time of p threads to process p times the data



Problem size with a) 1 core, b) 2 cores, c) 4 cores

Example:

Problem size:	Execution time:	Efficiency:
$N = 100$	$T(1) = 12$ s	$E(1) = 1.0$
$N = 200$	$T(2) = 15$ s	$E(2) = 0.8$
$N = 400$	$T(4) = 20$ s	$E(4) = 0.6$
$N = 800$	$T(8) = 24$ s	$E(8) = 0.5$



Amdahl's Law: Example 3

- Load balancing
- In the previous example, in order to achieve the speed-up of 91 (for the larger problem) with 100 processors, we assumed that the load was perfectly balanced.
 - **Perfect balance:** each of the 100 processors has 1% of the work to do
- What if: 1 of the 100 processors load is higher than all the other 99?
Calculate for increased loads of 2% and 5%

Amdahl's Law: Example 3

- If one has 2% of the parallel load then it must do $2\% \times 10000$ (larger problem) = 200 additions
 - The other 99 will share 9800
- Since they operate simultaneously:

$$\text{Exec time after improvement} = \max\left(\frac{9800t}{99}, \frac{200t}{1}\right) + 10t = 210t$$

- The speedup drops to $\text{Speed-up}_{100} = \frac{10010t}{210t} = 48$

$$\text{Speed-up}_{100}^{10000} \Big|^{NB} = 48\% \text{ (instead of } 91\%)$$

- If one processor has 5% of the load, then it must perform $5\% \times 10000 = 500$ additions

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

Computation-Transfer overlap

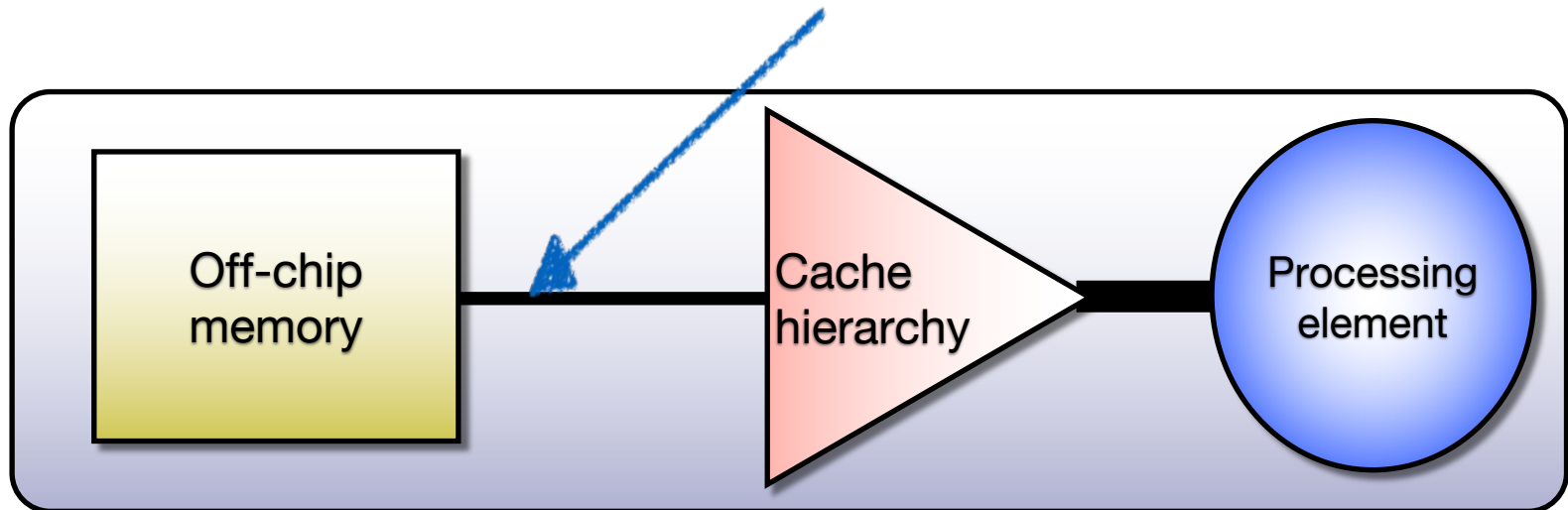
- On CPUs:
 - Superscalar execution (multiple instructions per cycle)
 - In principle: automatic overlap (balanced instructions)
 - In practice: enforced through software prefetching

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]

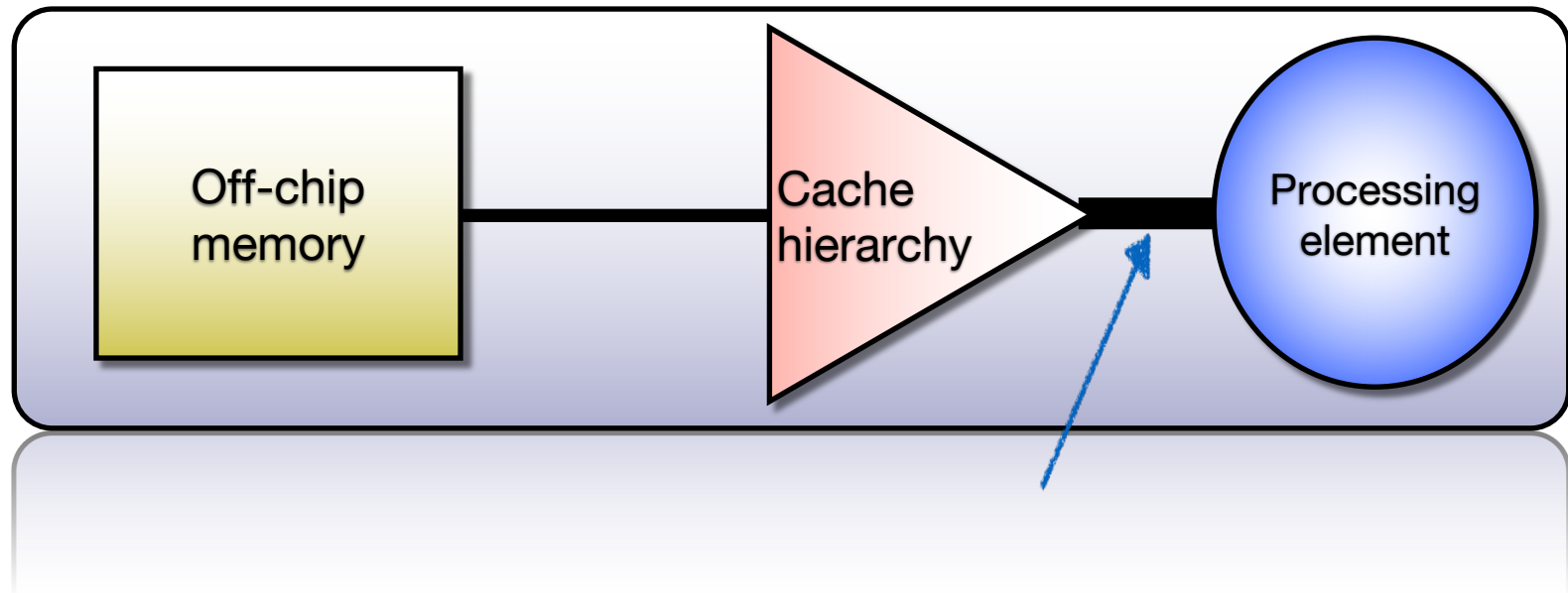
Operational Intensity

- 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

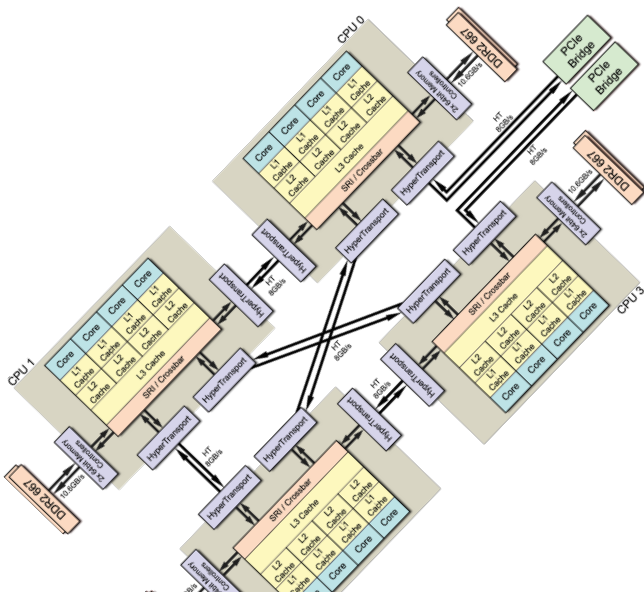
- Not equivalent to arithmetic intensity [1], machine balance[2]
 - which refer to traffic between the processor and the cache
- Not forcedly bound to FLOP/Bytes (e.g. Comparison/Byte)



[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



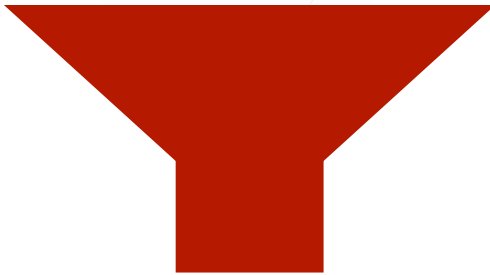
```
#include <vector>
#include <sstream>
#include <cmath>
#include <cassert>
#include <stdint.h>

// a saxpy version requiring alignment
void saxpy(int n, float a, float* x, float* y)
{
    // load the scale factor requiring alignment
    _mm128 x8 = _mm_load_ps(a);
    // we assume alignment
    std::size_t xv = reinterpret_cast<std::size_t*>(&x);
    assert(xv % 16 == 0 && yv % 16 == 0);

    int ndiv4 = n/4;
    for (int i=0; i<ndiv4; ++i) {
        _mm_prefetch(y+i*4, MM_HINT_NTA);
        _mm128 x2 = _mm_load_ps(y+i*4);
        _mm128 x4 = _mm_load_ps(y+i*4);
        _mm_store_ps(y+i*4, x2);
    }
    // do the remaining entries
    for (int i=ndiv4*4; i<n; ++i)
        y[i] += a*x[i];
}
```



GFLOP/s
GB/s



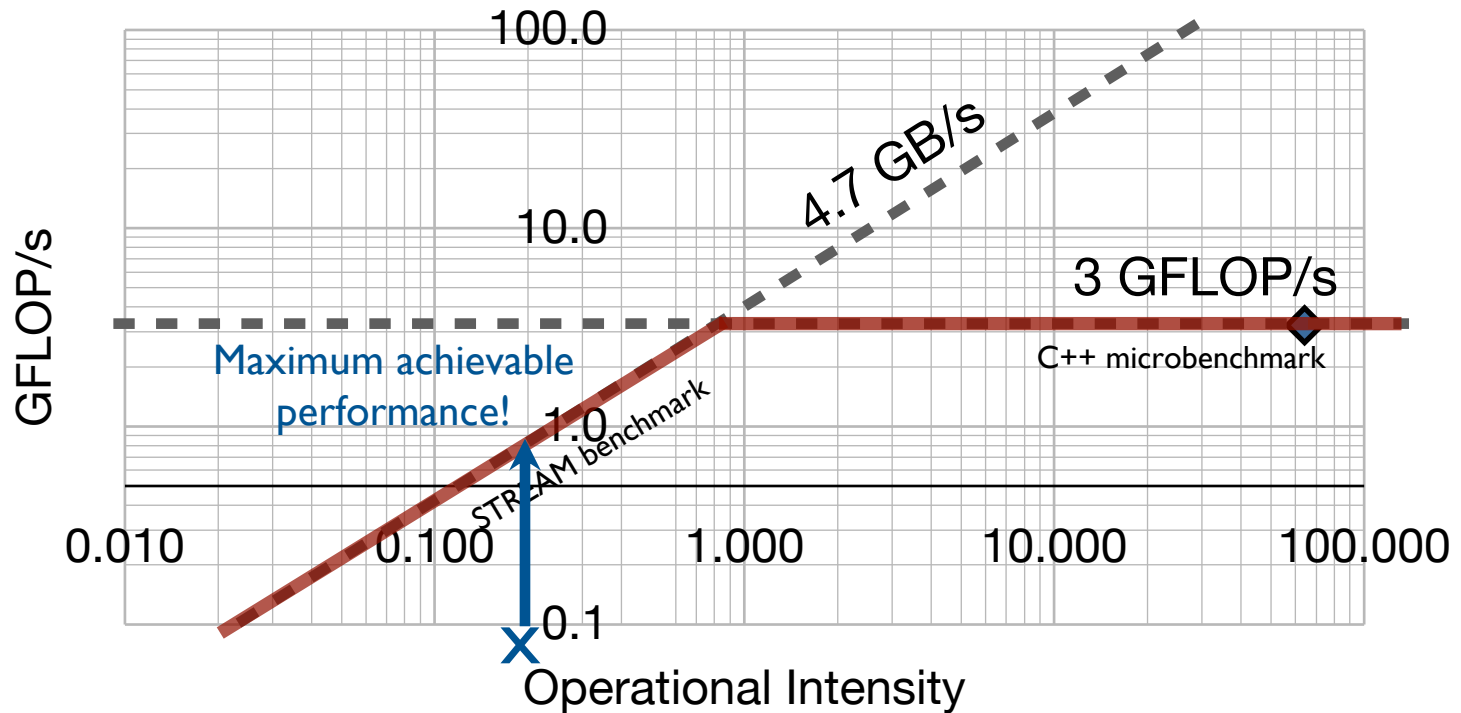
FLOP/B

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

The Roofline Model



◆ 4x Quad-Core AMD Opteron 8380 @ 2.5GHz - 1 Thread - C++

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]

$$\text{Performance} = \min(\text{OI} * \text{PB}, \text{PP})$$

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: $3 \cdot (N-2)$ FLOP
- Memory accesses (no caching): $4 \cdot (N-2)$ floats accessed
 - every data accessed is counted
- Memory accesses (perfect caching): $2 \cdot N - 2$ 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 : single precision (4 Bytes)

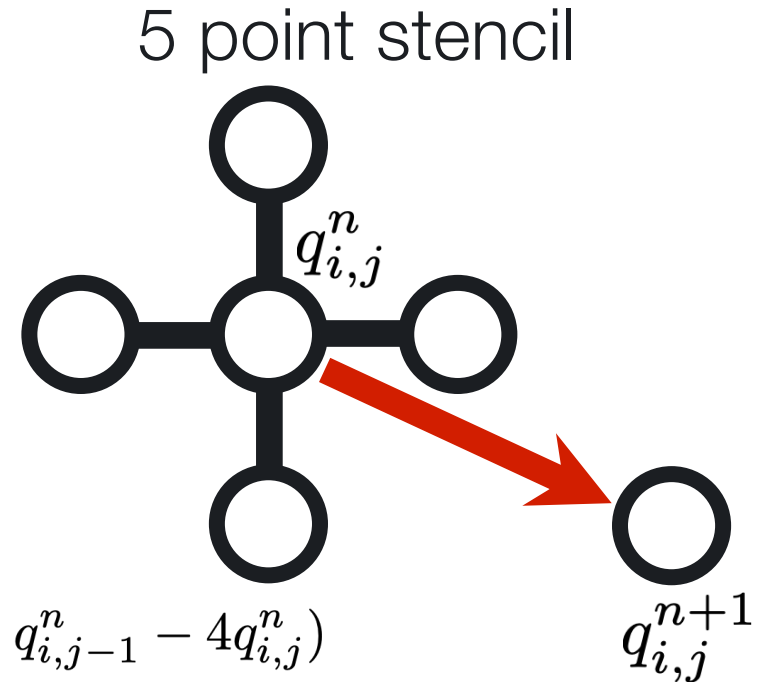
- Algorithm

- 1. Laplace Operator

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

- 2. Forward Euler Operator:

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



A-Priori Performance Analysis

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

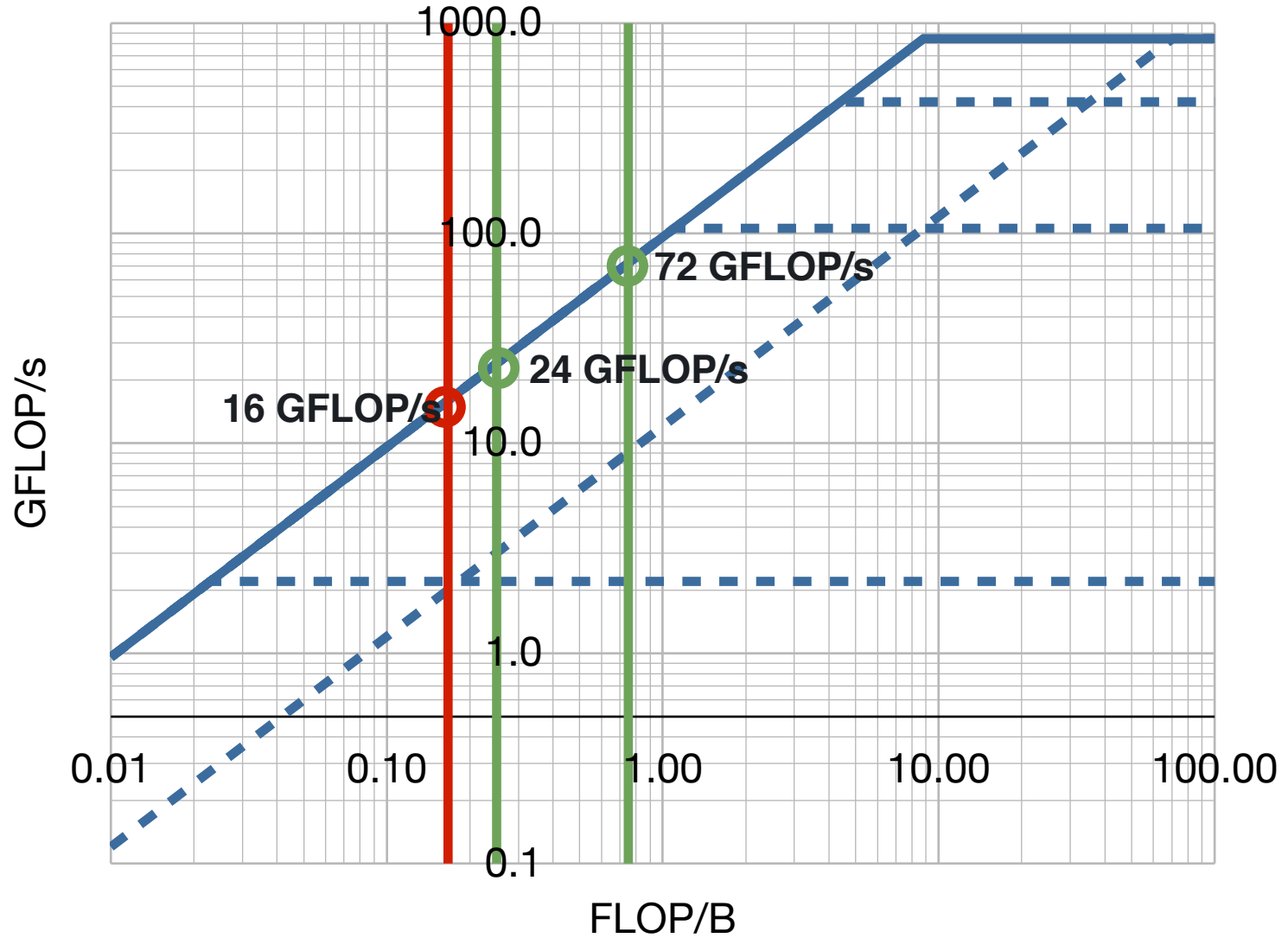
- Floating point operations per point: 4 ADD + 2 MUL
- Memory accesses per point:
 - Worst case: 5 read + 1 write
 - Best case: 1 read + 1 write
- Operational Intensity:
 - Worst case: 6 FLOP / (6*4 B) = 0.25 FLOP/B
 - Best case: 6 FLOP / (2*4 B) = 0.75 FLOP/B

A-Priori Performance Analysis

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

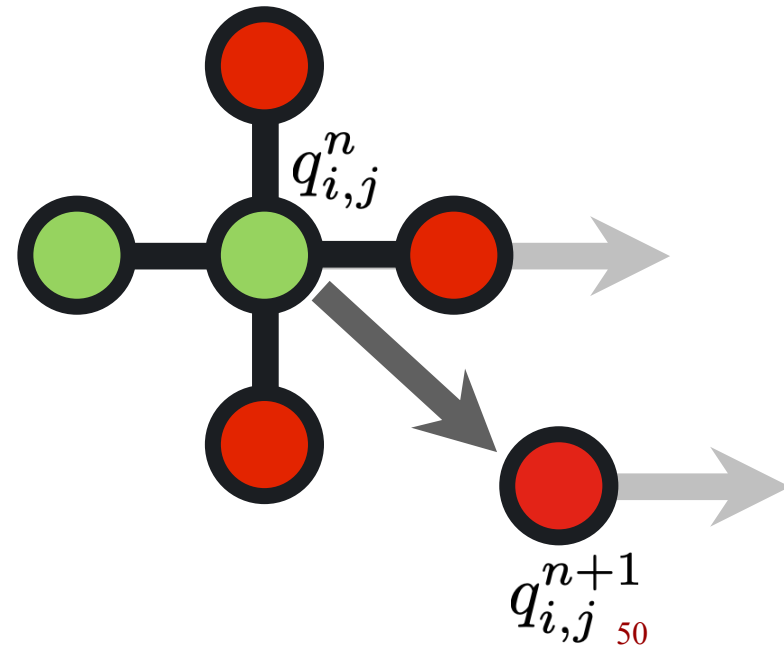
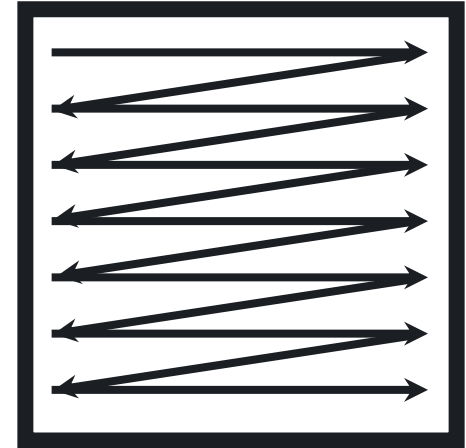
- Floating point operations per point: 1 ADD + 1 MUL
- Memory accesses per point:
 - Worst case: 2 read + 1 write
 - Best case: 2 read + 1 write
- Operational Intensity:
 - Worst case: 2 FLOP / (3*4 B) = 0.17 FLOP/B
 - Best case: 2 FLOP / (3*4 B) = 0.17 FLOP/B

Roofline

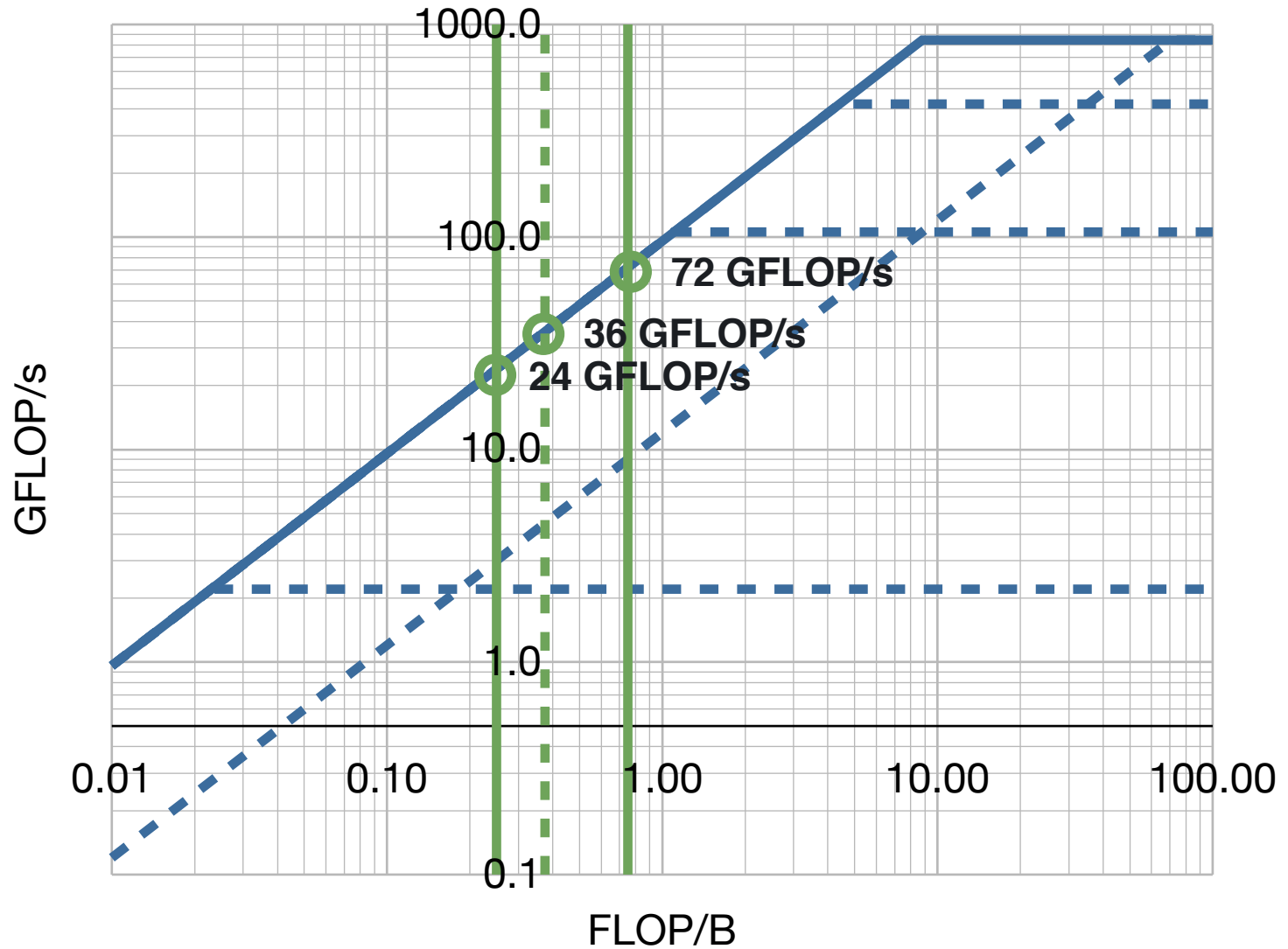


A More Accurate Analysis

- We have locality!
- Memory accesses per point:
 - 3 read + 1 write
- Operational Intensity:
 - $6 \text{ FLOP} / (4 \times 4 \text{ B}) = 0.375 \text{ FLOP/B}$

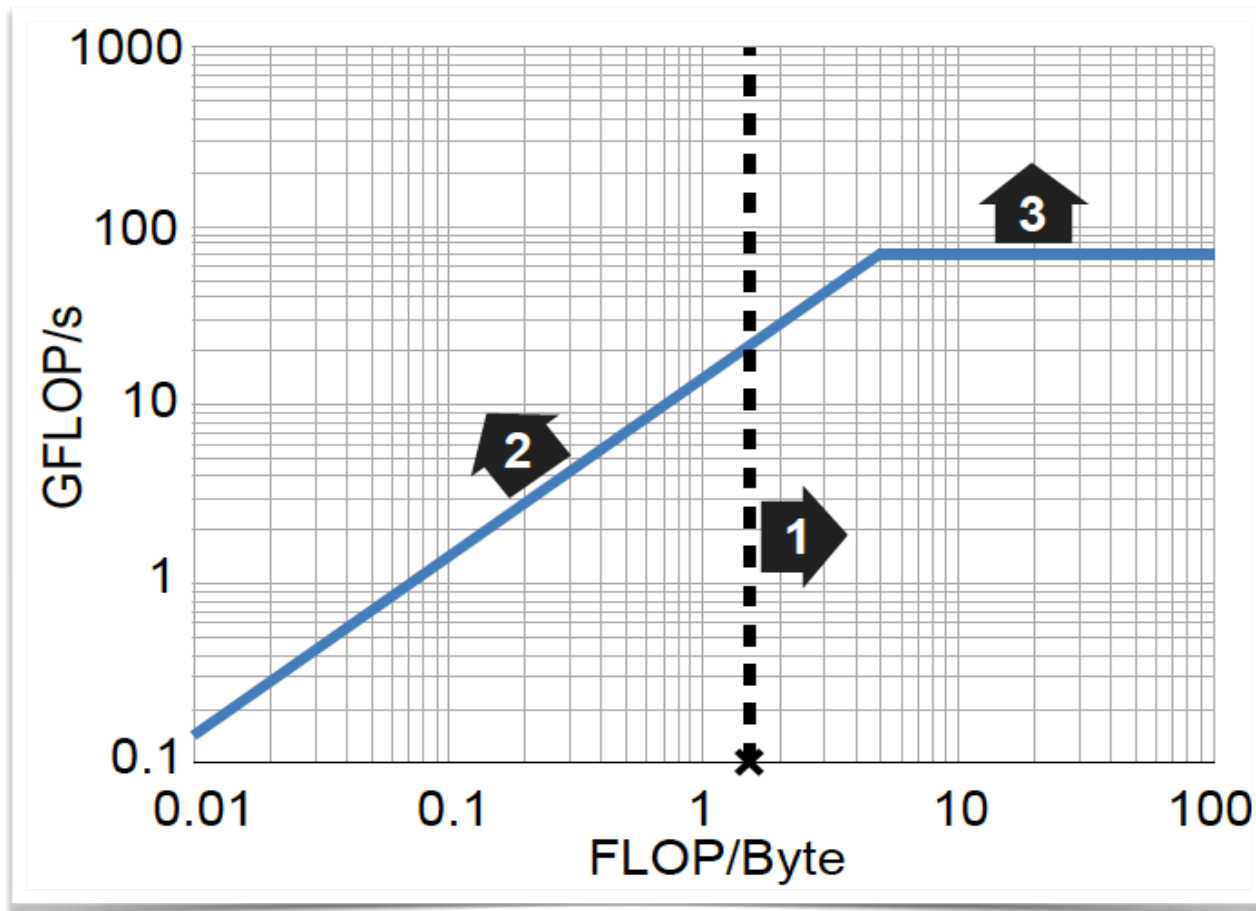


Roofline

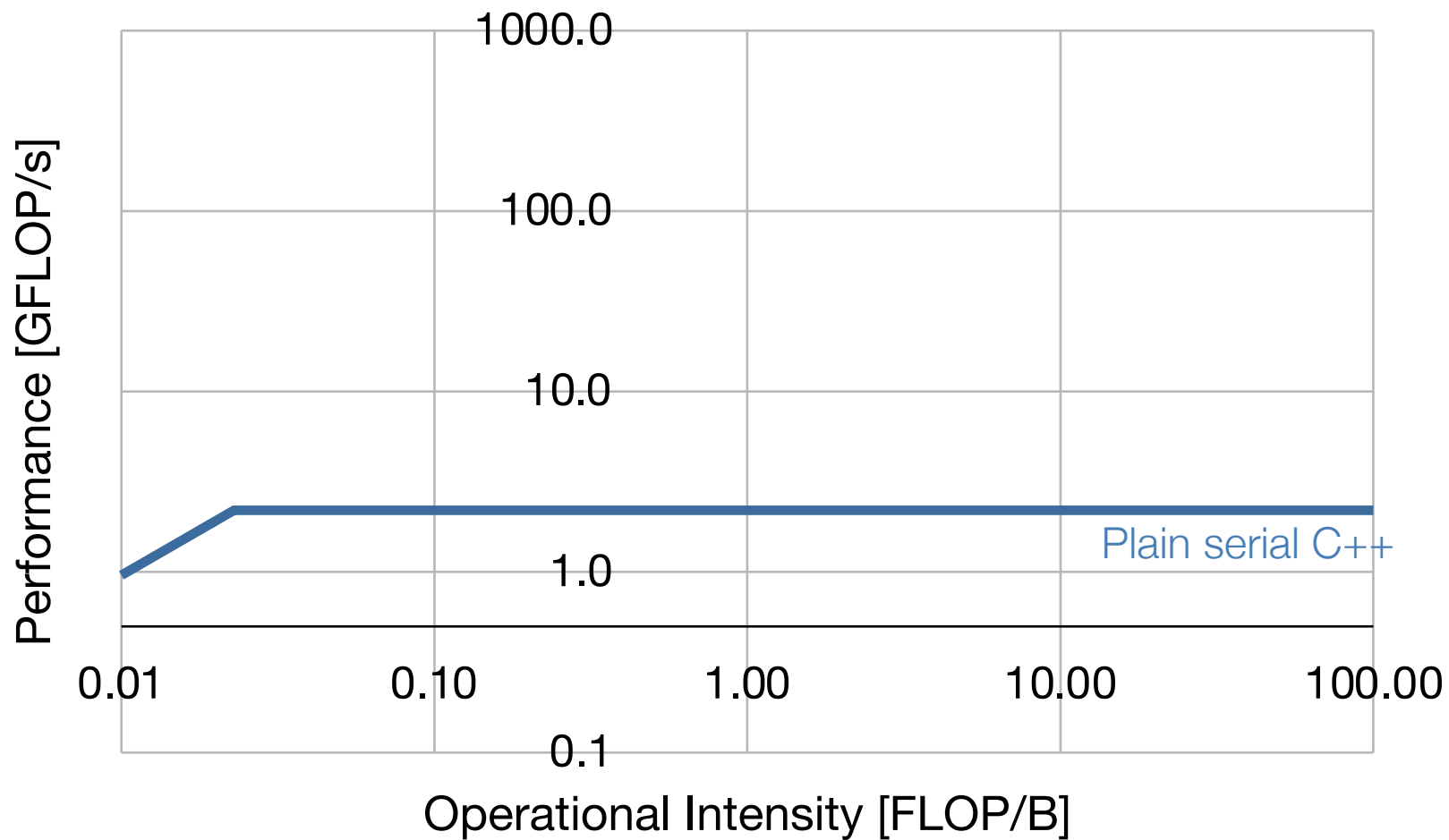


Optimization

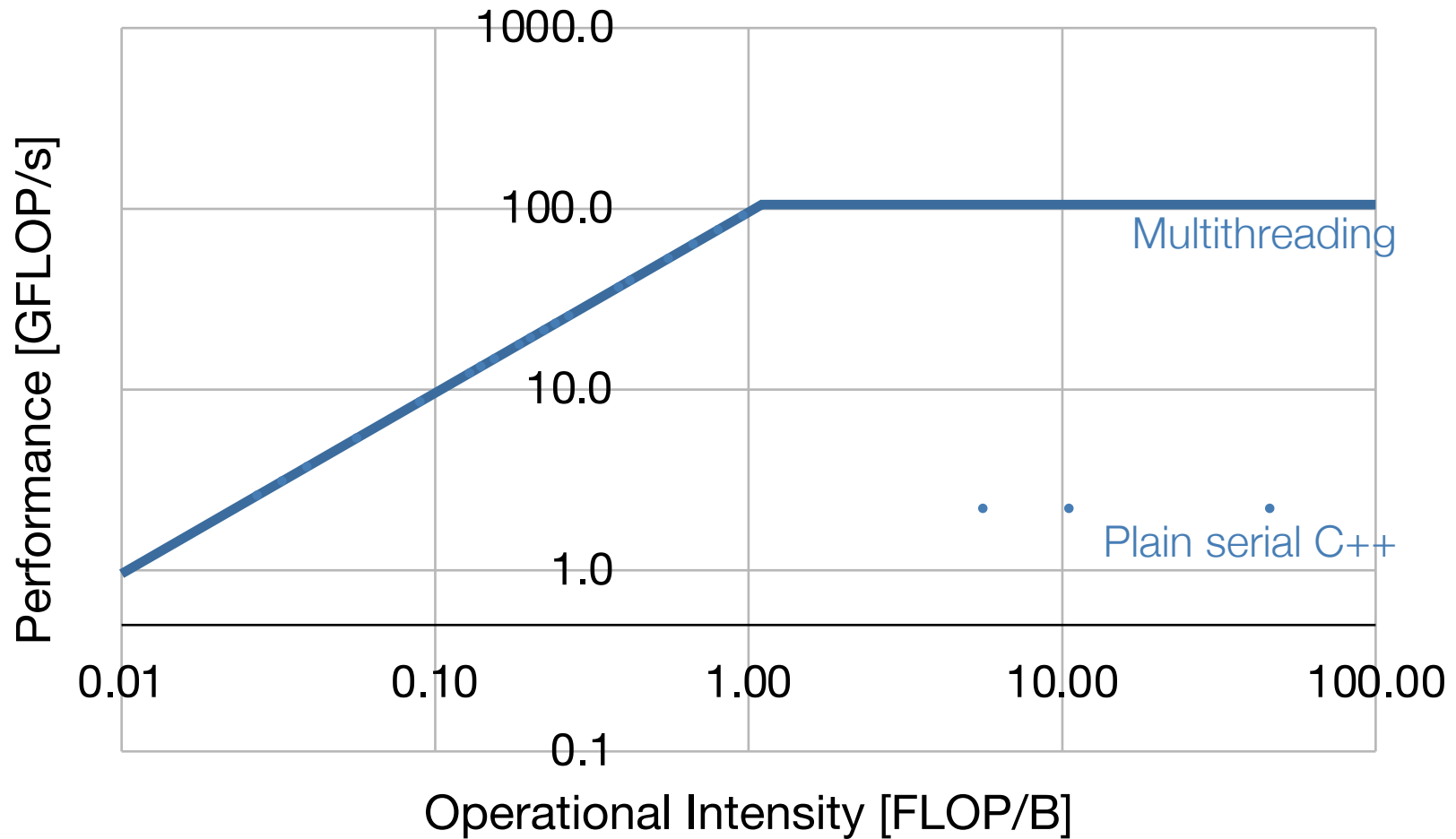
1. Locality
2. Communication
3. Computation



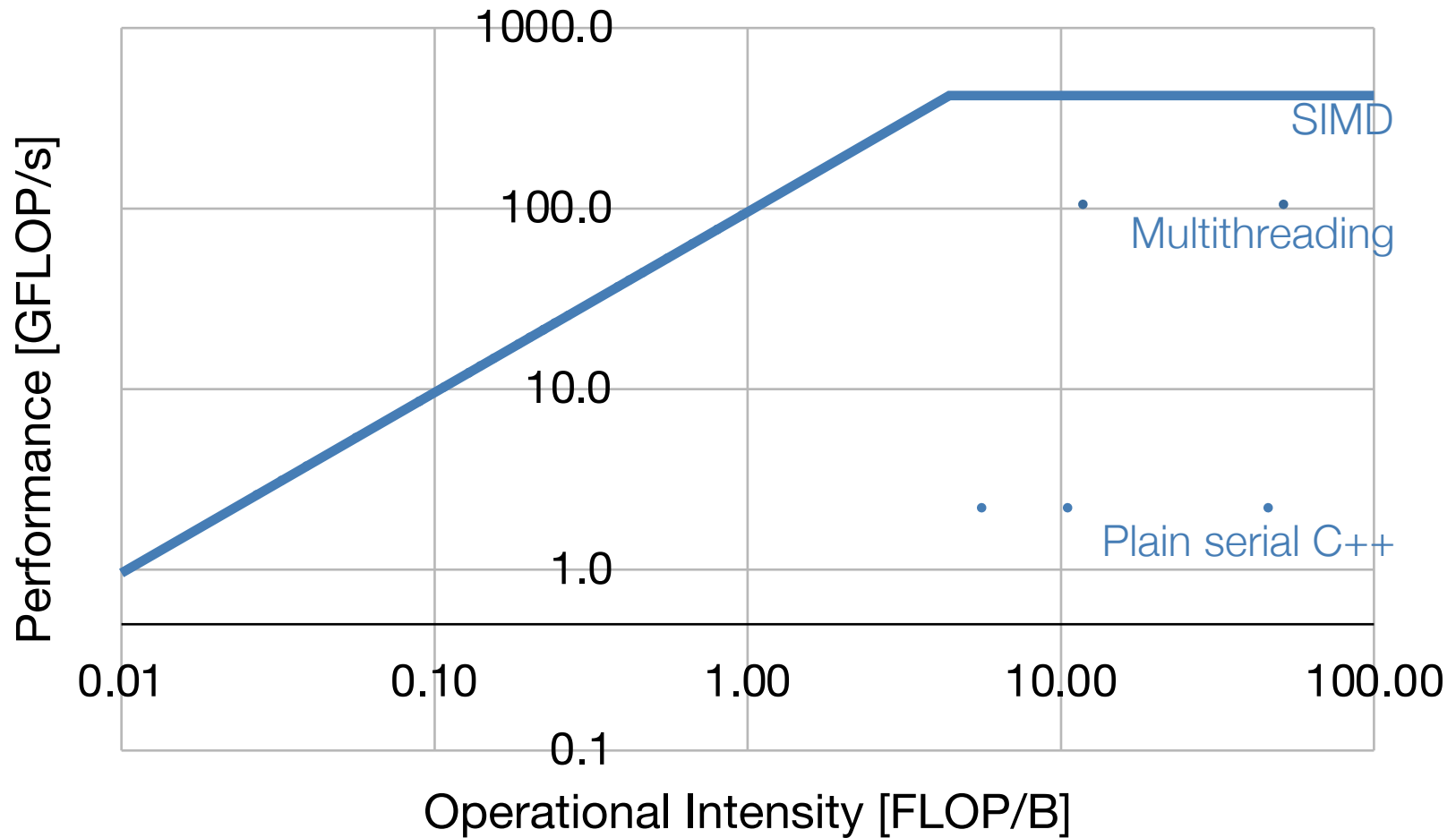
Ceilings on Brutus (single precision)



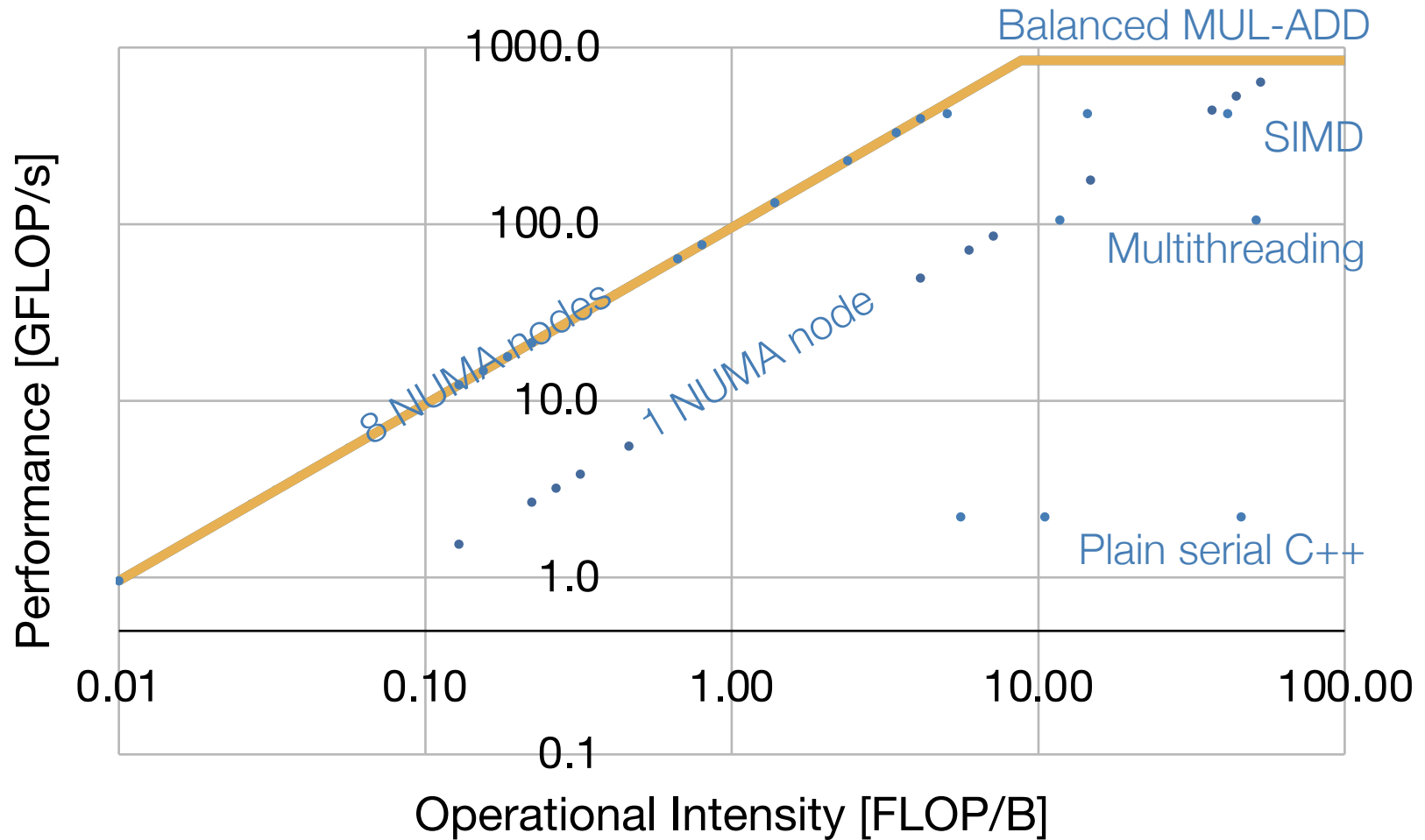
Ceilings on Brutus (single precision)



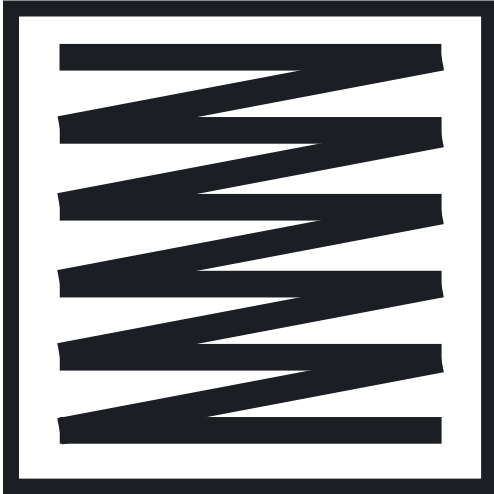
Ceilings on Brutus (single precision)



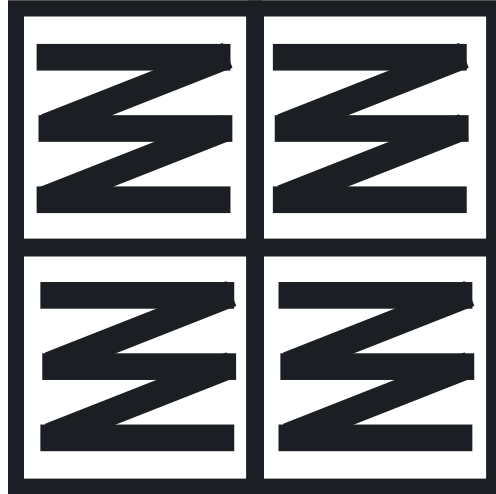
Ceilings on Brutus (single precision)



Improving Locality



Linear



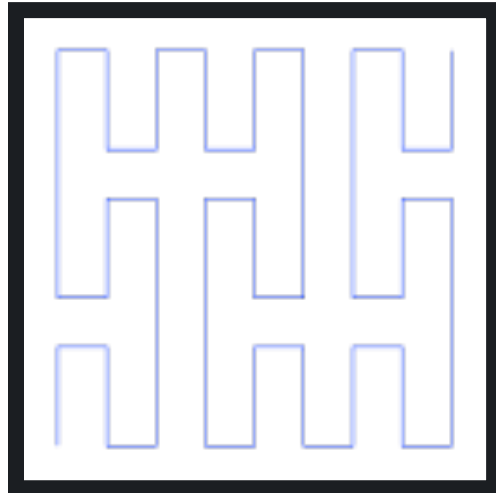
Blocked



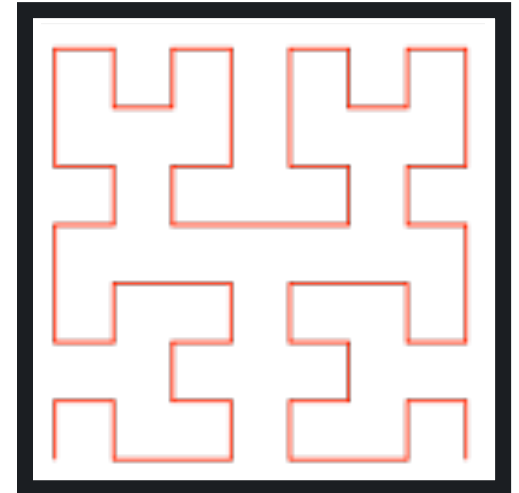
Blocking Hierarchy



Morton or Z-Order



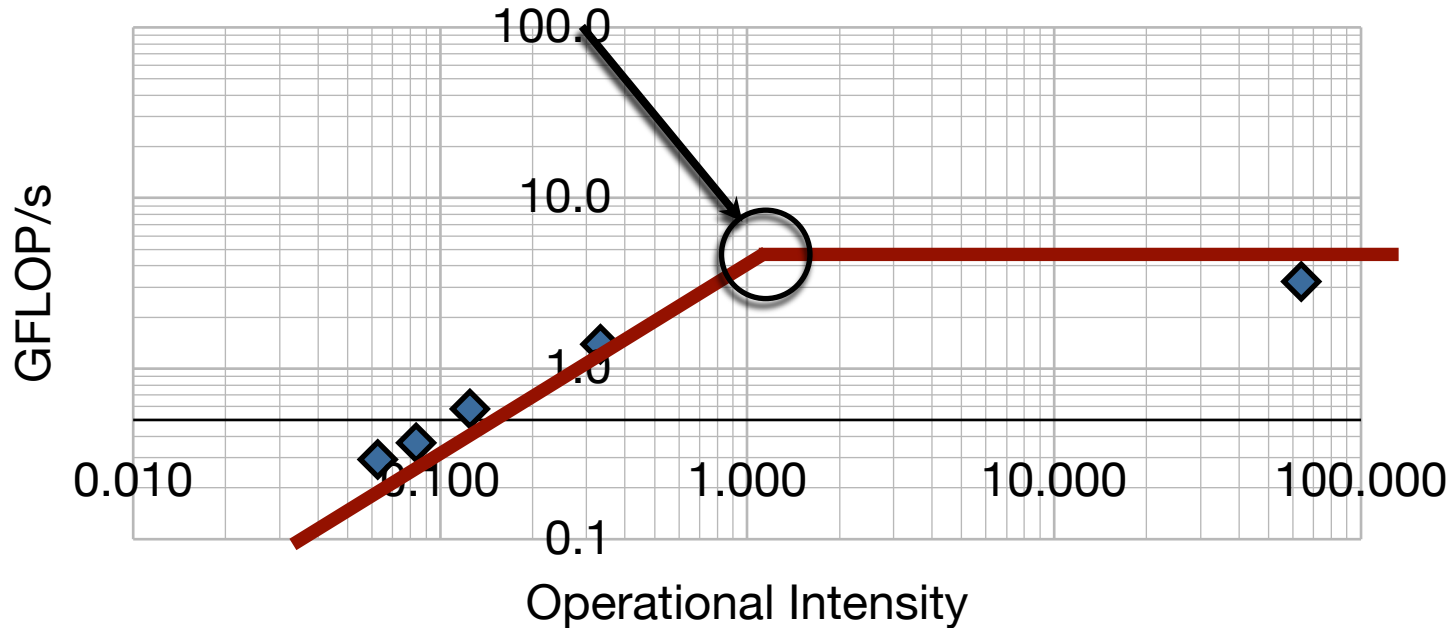
Peano



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?

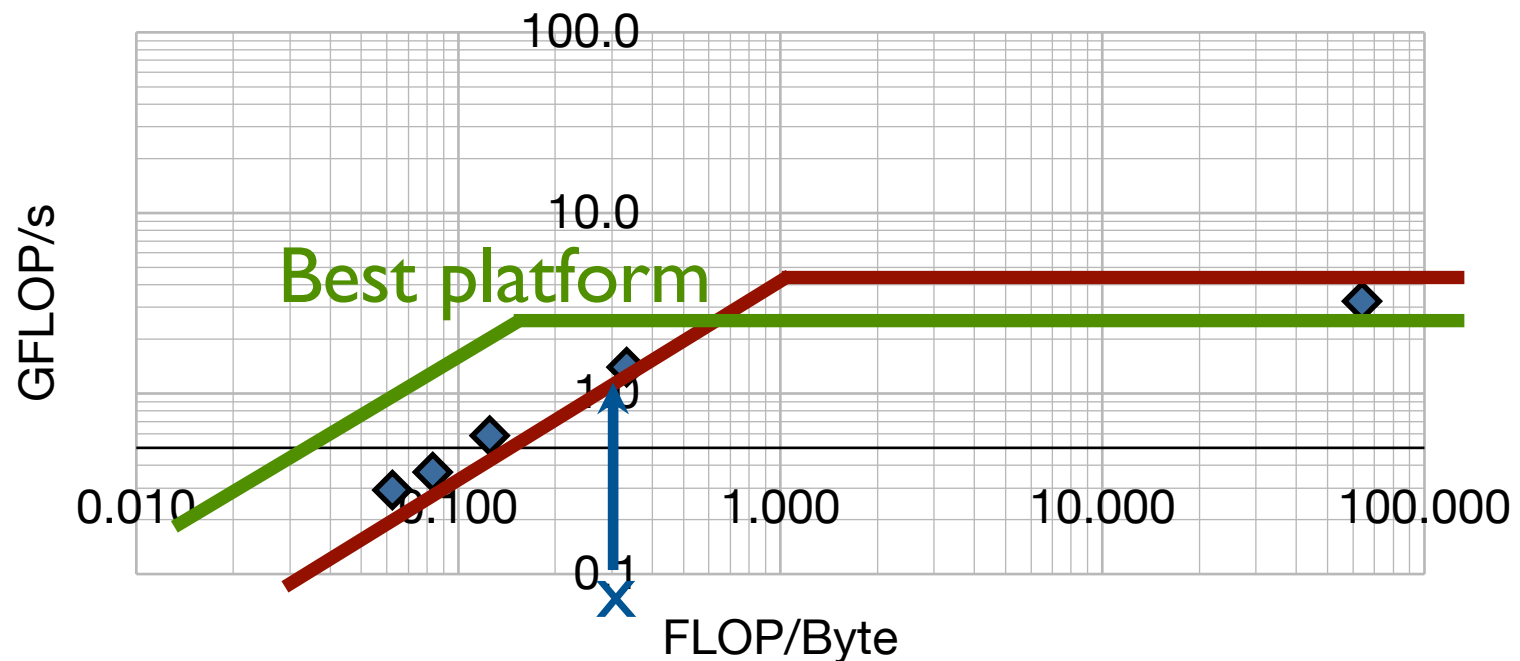
Production Software

- Assumption: production-ready software

- Limited set of algorithms
- Fixed set of kernels
- Fixed operational intensities

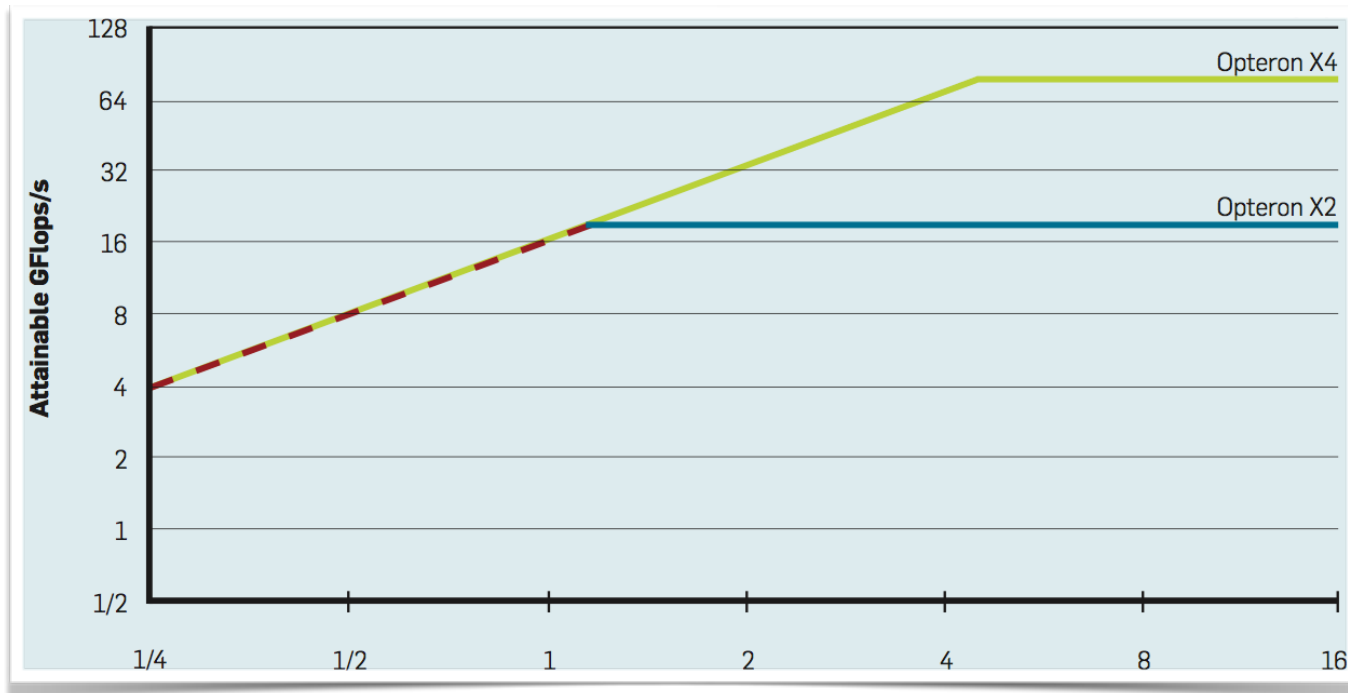


Best hardware solution?



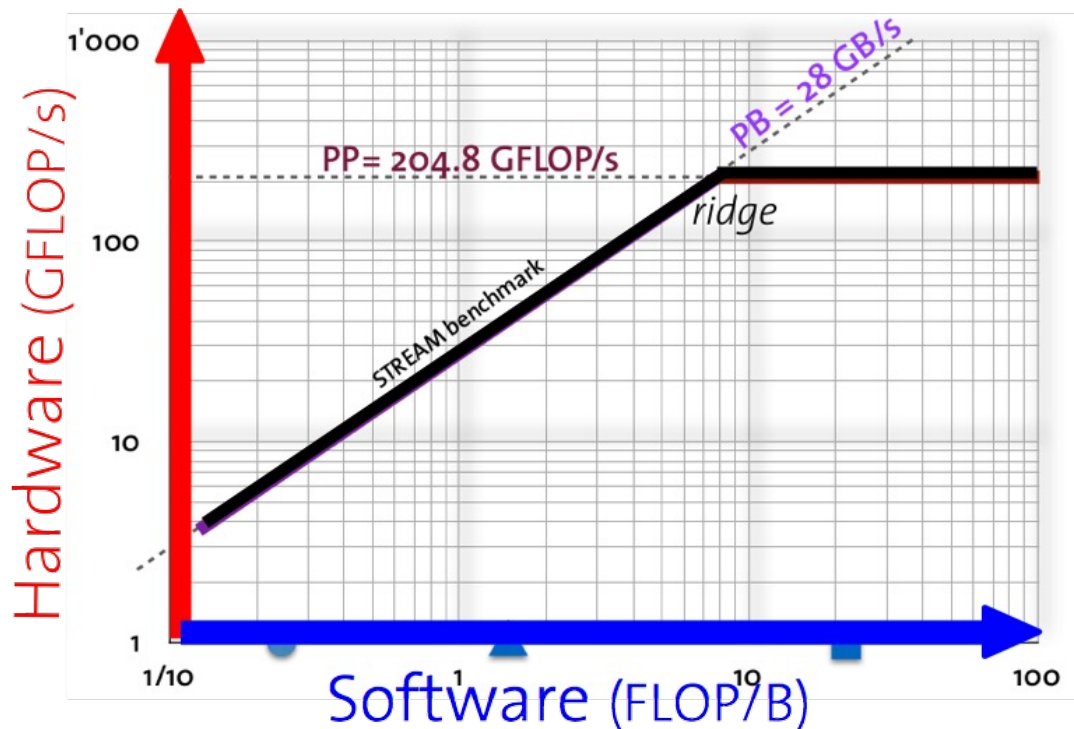
Is Moore worth?

- 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 = $\min(\text{PB} \times \text{OI}, \text{PP})$
- 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)