Master Informatics Eng.

2019/20

A.J.Proença

The Roofline Performance Model

(most slides are borrowed)

Motivation

- Multicore guarantees neither good scalability nor good (attained) performance
- Performance and scalability can be extremely non-intuitive even to computer scientists

- Success of the multicore paradigm seems to be premised upon their programmability
- To that end, one must understand the limits to both scalability and efficiency.

- How can we empower programmers?

Goals of the Roofline Model

CONVENTIONAL WISDOM IN computer architecture produced similar designs. Nearly every desktop and server computer uses caches, pipelining, superscalar instruction issue, and out-of-order execution. Although the instruction sets varied, the microprocessors were all from the same school of

Roofline Model

For the foreseeable future, off-chip memory bandwidth will often be the constraining resource in system performance.[23] Hence, we want a model that relates processor performance to off-chip memory traffic. Toward this

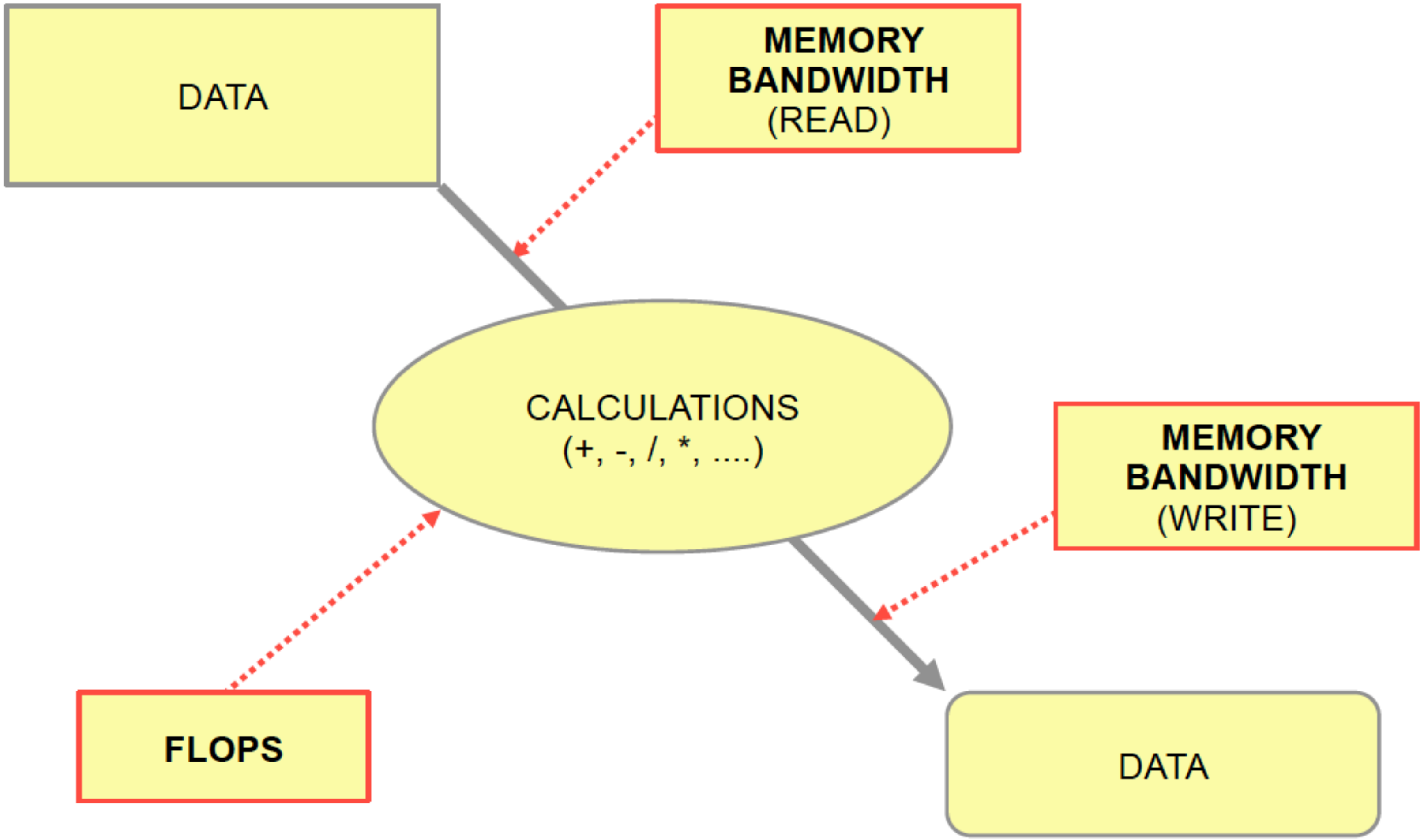
DOI:10.1145/1498765.1498785

The Roofline model offers insight on how to improve the performance of software and hardware.

BY SAMUEL WILLIAMS, ANDREW WATERMAN, AND DAVID PATTERSON

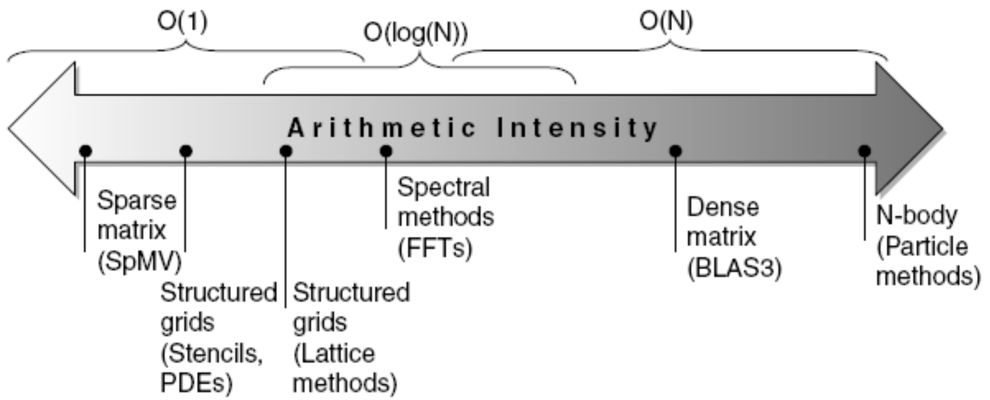
Roofline:

Performance Limiting Factors



Roofline Performance Model

- Basic idea:
  - Plot peak floating-point throughput as a function of arithmetic intensity
  - Ties together floating-point performance and memory performance for a target machine
- Arithmetic intensity
  - Floating-point operations per byte read



Components

- There are three principal components to performance:
  - **Computation**
  - **Communication**
  - **Locality**

- Each architecture has a different balance between these
- Each kernel has a different balance between these

- Performance is a question of how well an kernel's characteristics map to an architecture's characteristics

Computation

- For us, floating point performance (**Gflop/s**) is the metric of interest (typically double precision)

... but we could also consider SP or int

- Peak in-core performance can only be attained if:
  - fully exploit ILP, DLP, FMA, etc…
  - non-FP instructions don't sap instruction bandwidth
  - threads don't diverge (GPUs)
  - transcendental/non pipelined instructions are used sparingly
  - branch mispredictions are rare

- To exploit a form of in-core parallelism, it must be:
  - Inherent in the algorithm
  - Expressed in the high level implementation
  - Explicit in the generated code

Communication

- For us, DRAM bandwidth (**GB/s**) is the metric of interest

- Peak bandwidth can only be attained if certain optimizations are employed:
  - Few unit stride streams
  - NUMA allocation and usage
  - SW Prefetching
  - Memory Coalescing (GPU)

Locality

- Computation is free, Communication is expensive.
- Maximize locality to minimize communication
- **There is a lower limit to communication: compulsory traffic**

- Hardware changes can help minimize communication
  - Larger cache capacities minimize capacity misses
  - Higher cache associativities minimize conflict misses
  - Non-allocating caches minimize compulsory traffic

3Cs model for caches

- Software optimization can also help minimize communication
  - Padding avoids conflict misses
  - Blocking avoids capacity misses
  - Non-allocating stores minimize compulsory traffic

Three Classes of Locality

- Temporal Locality
  - reusing data (either registers or cache lines) multiple times
  - amortizes the impact of limited bandwidth.
  - **transform loops or algorithms to maximize reuse.**

- Spatial Locality
  - data is transferred from cache to registers in words.
  - However, data is transferred to the cache in 64-128Byte lines
  - using every word in a line maximizes spatial locality.
  - **transform data structures into *structure of arrays* (SoA) layout**

- Sequential Locality
  - Many memory address patterns access cache lines sequentially.
  - CPU's hardware stream prefetchers exploit this observation to hide speculatively load data to memory latency.
  - **Transform loops to generate (a few) long, unit-stride accesses.**

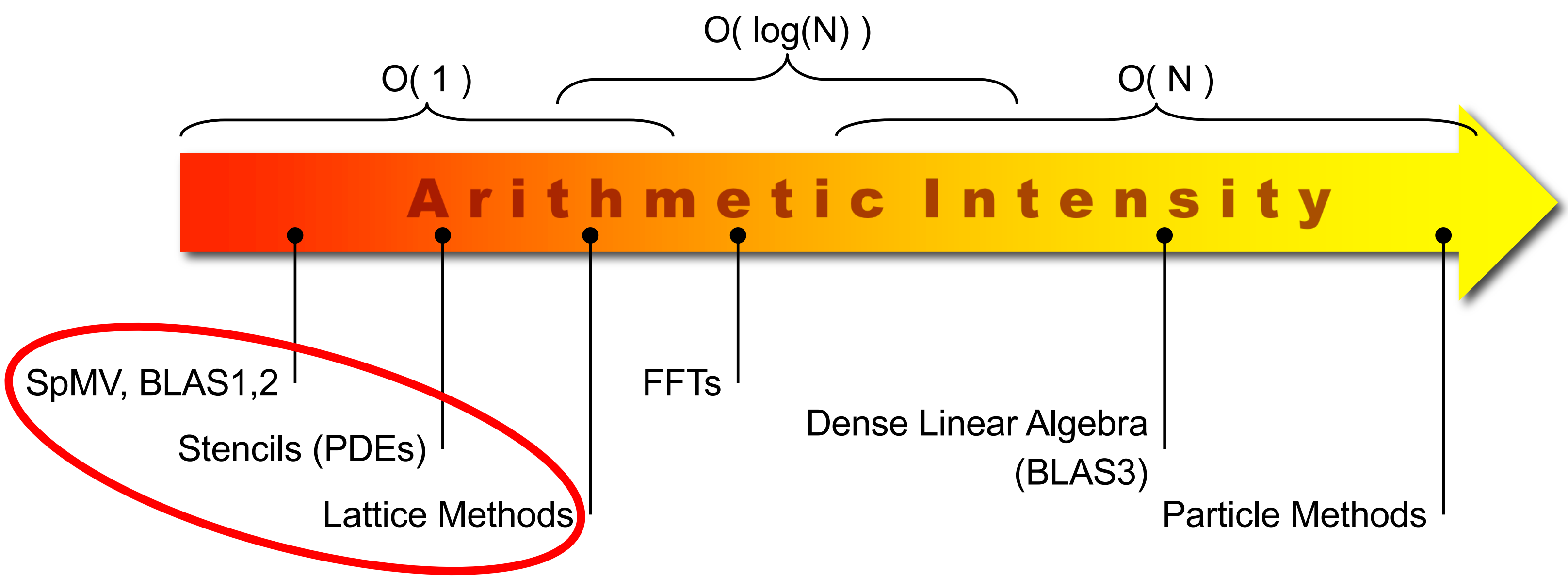
Preliminary notes in the Roofline Model

- goal: integrate in-core performance, memory bandwidth, and locality into a single readily understandable performance figure
- graphically show the penalty associated with not including certain software optimizations
- Roofline model will be unique to each architecture

Key elements in the Roofline Model

- *x*-axis: the "operational intensity", operations per byte of RAM traffic, Flops/byte (traffic between caches and memory)
- *y*-axis: the attainable floating-point performance, GFlops/sec includes both peak processor/memory performance
- *peak processor FP performance*: a horizontal line computed from the processor specs
- *peak memory performance*: bounds the max FP performance of the memory system for a given operational intensity
- *for each kernel*: its performance is a point on a vertical line that crosses the *x*-axis on the kernel operational intensity

Arithmetic Intensity



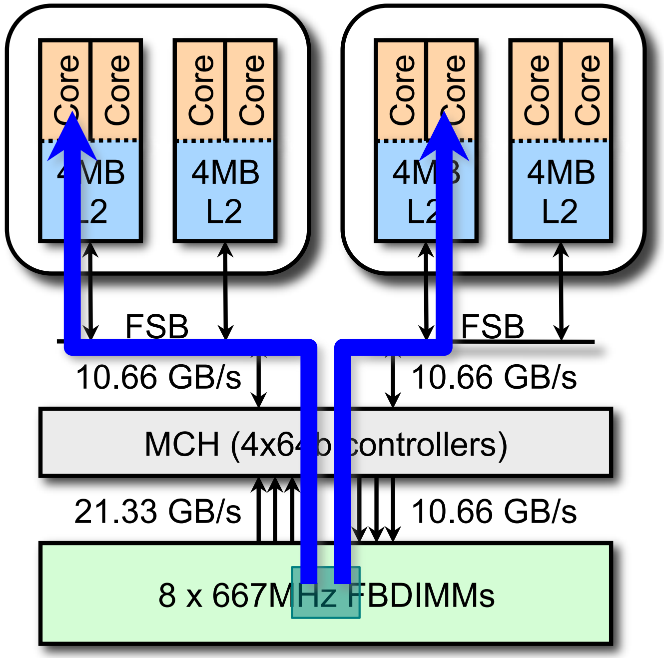
- **True Arithmetic Intensity (AI) ~ Total Flops / Total DRAM Bytes**

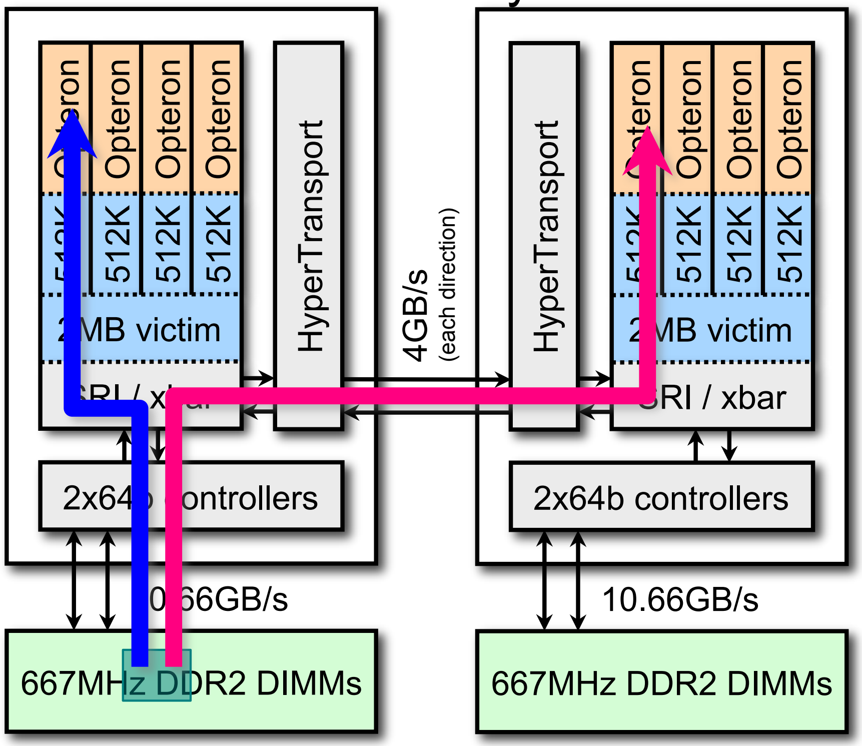
- Some HPC kernels have an arithmetic intensity that scales with problem size (increased temporal locality)
- Others have constant intensity

- Arithmetic intensity is ultimately limited by compulsory traffic
- Arithmetic intensity is diminished by conflict or capacity misses.

NUMA

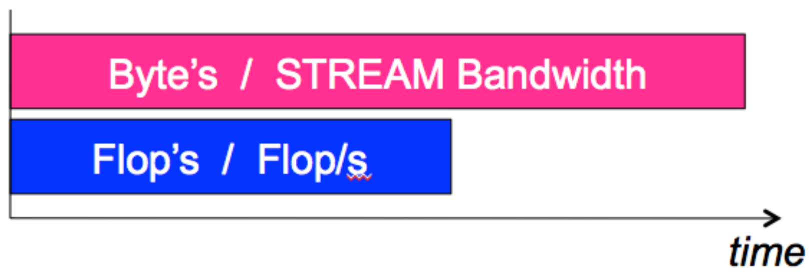
- Recent multicore SMPs have integrated the memory controllers on chip.
- As a result, memory-access is non-uniform (NUMA)
- That is, the bandwidth to read a given address varies dramatically among between cores
- Exploit NUMA (affinity+first touch) when you malloc/init data.
- Concept is similar to data decomposition for distributed memory





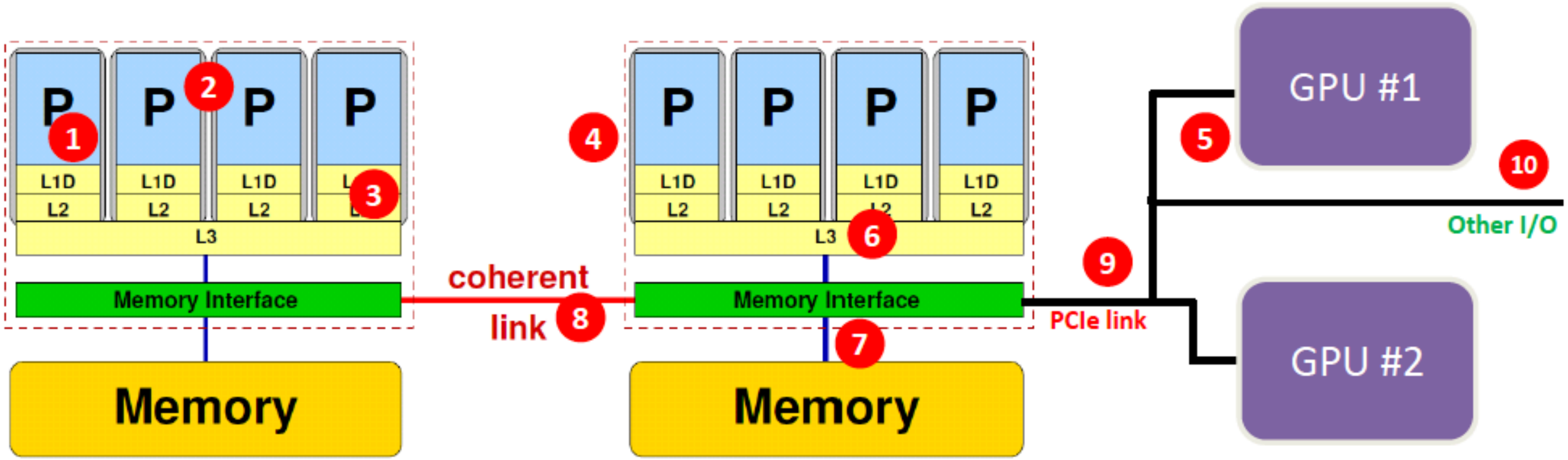
- Memory bandwidth #'s collected via micro benchmarks (or the STREAM benchmark)
- Computation #'s derived from optimization manuals (pencil and paper)
- Assume complete overlap of either communication or computation =>

Gflop/s = min { Peak Gflop/s ; Stream BW * actual flop:byte ratio }



Parallelism in a modern compute node

- Parallel and shared resources within a shared-memory node



Parallel resources:

- Execution/SIMD units (1)
- Cores (2)
- Inner cache levels (3)
- Sockets / ccNUMA domains (4)
- Multiple accelerators (5)

Shared resources ("bottlenecks"):

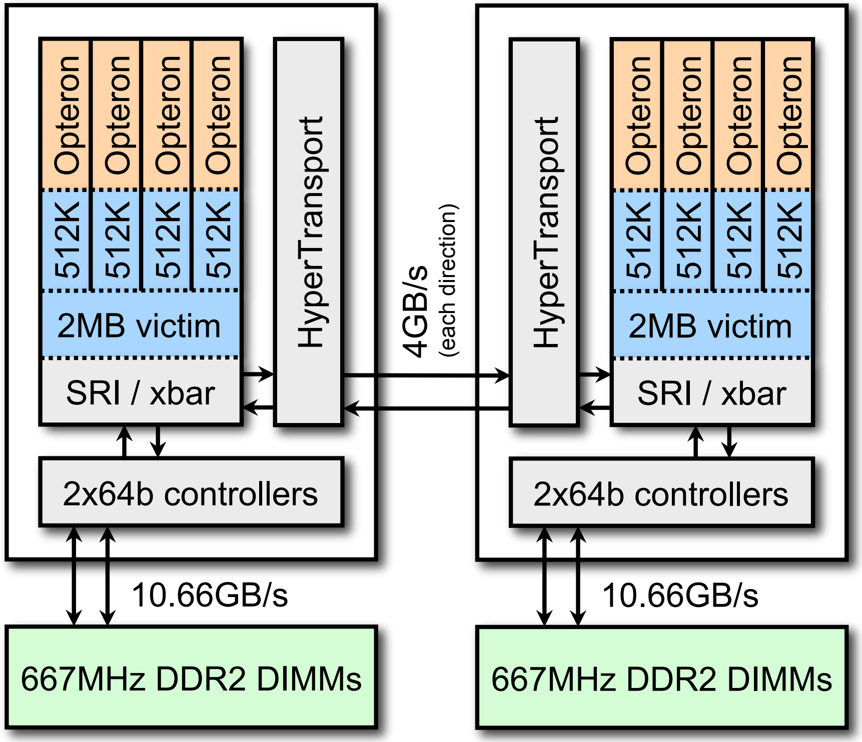
- Outer cache level per socket (6)
- Memory bus per socket (7)
- Intersocket link (8)
- PCIe bus(es) (9)
- Other I/O resources (10)

Where is the bottleneck for your application?

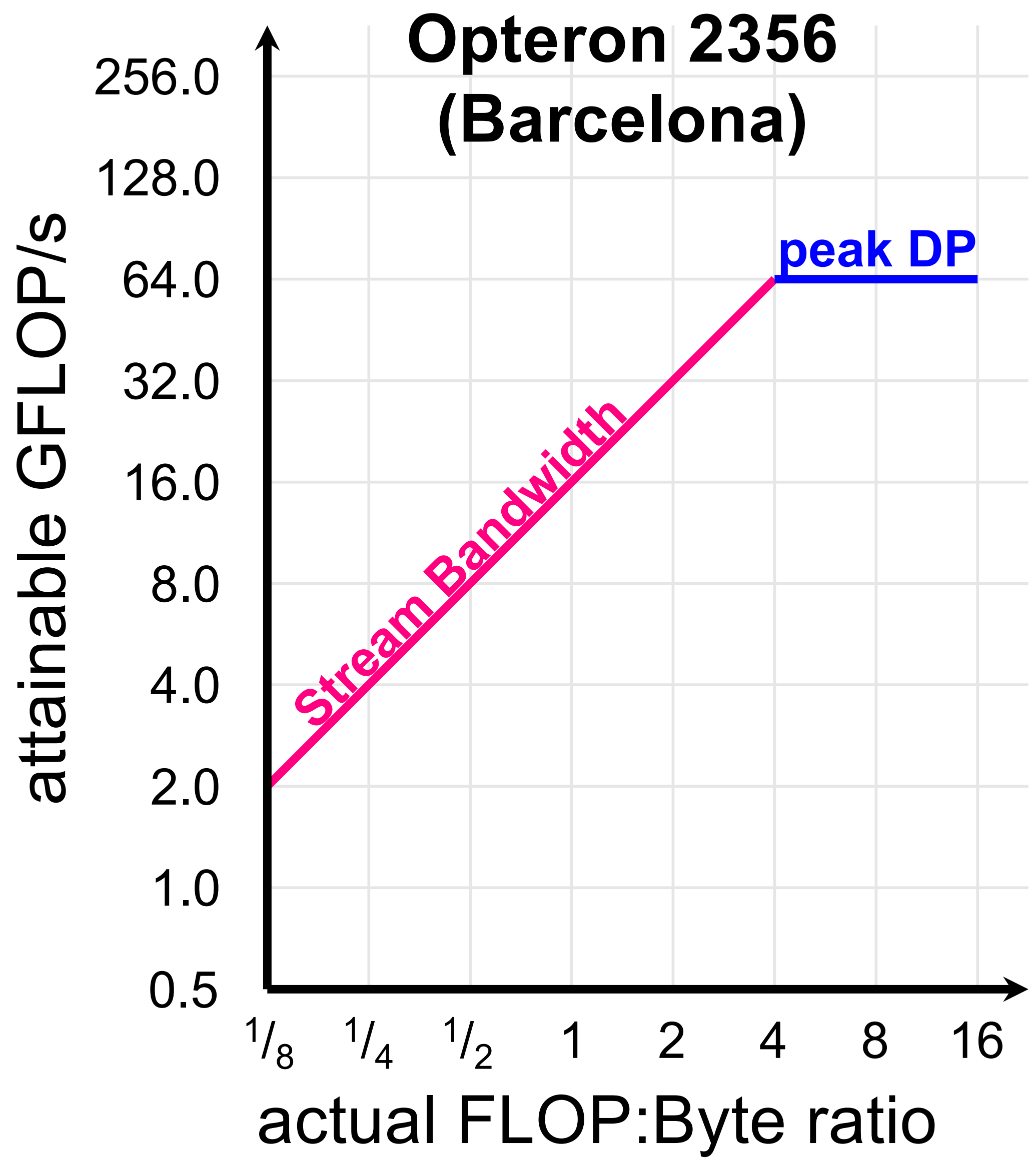
Georg Hager, Jan Treibig, Gerhard Wellein

Example

- Consider the Opteron 2356:
  - Dual Socket (NUMA)
  - limited HW stream prefetchers
  - quad-core (8 total)
  - 2.3GHz
  - 2-way SIMD (DP)
  - separate FPMUL and FPADD datapaths
  - 4-cycle FP latency



- Assuming **expression of parallelism** is the challenge on this architecture, what would the roofline model look like ?

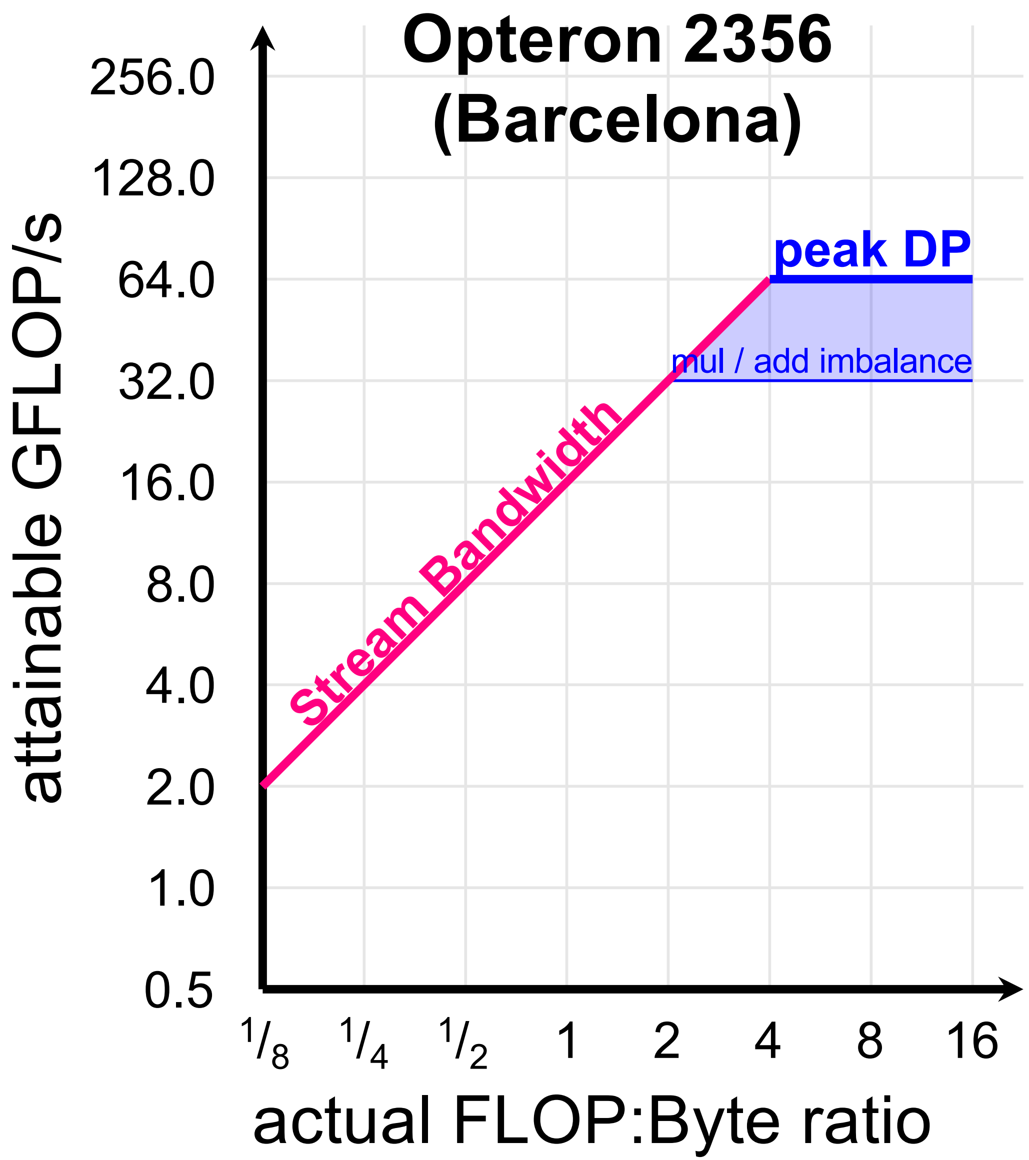


- Plot on log-log scale
- Given AI, we can easily bound performance
- But architectures are much more complicated

- We will bound performance as we eliminate specific forms of in-core parallelism

Roofline Model

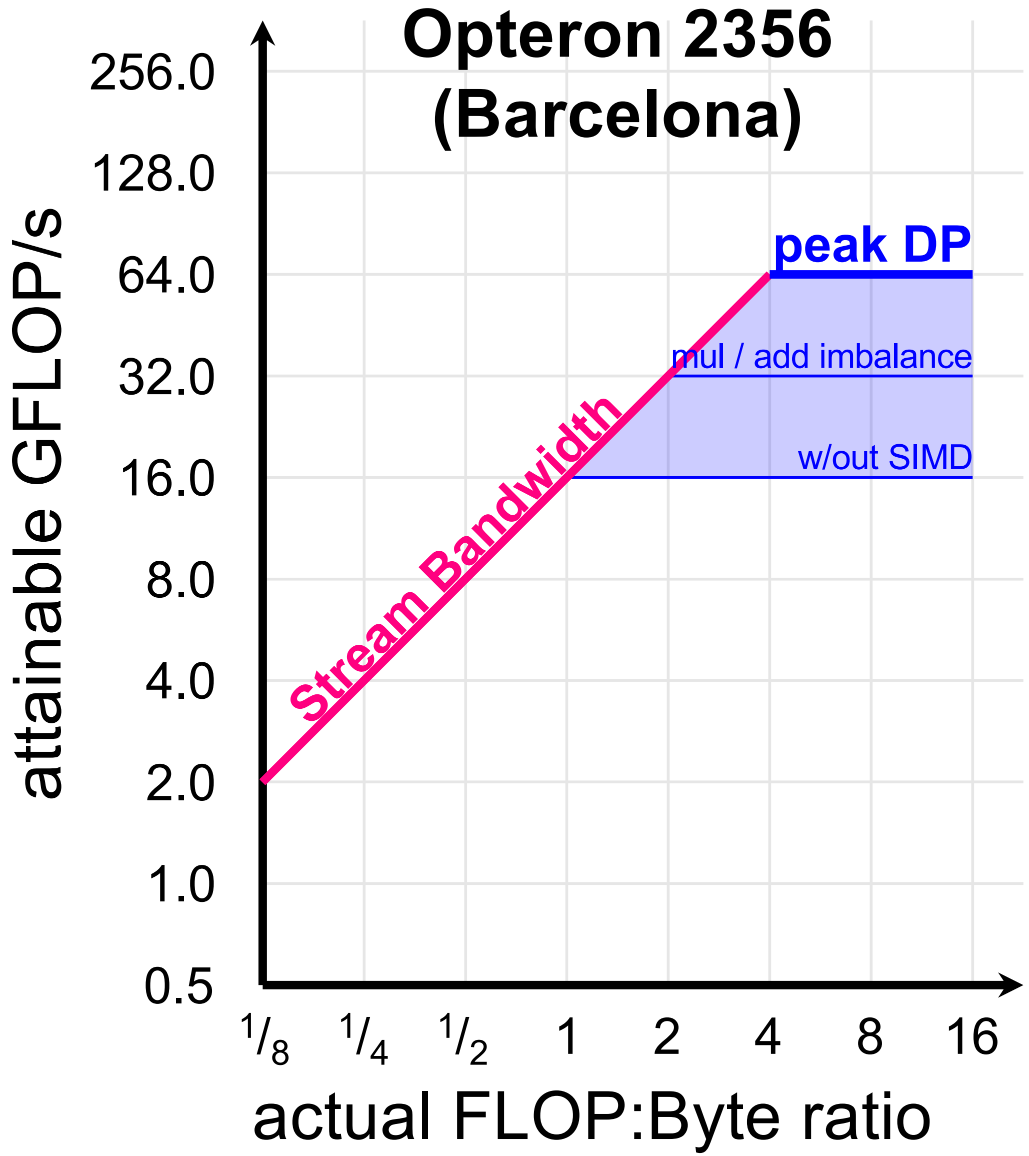
computational ceilings



- Opterons have dedicated multipliers and adders.
- If the code is dominated by adds, then attainable performance is half of peak.
- We call these **Ceilings**
- They act like constraints on performance

Roofline Model

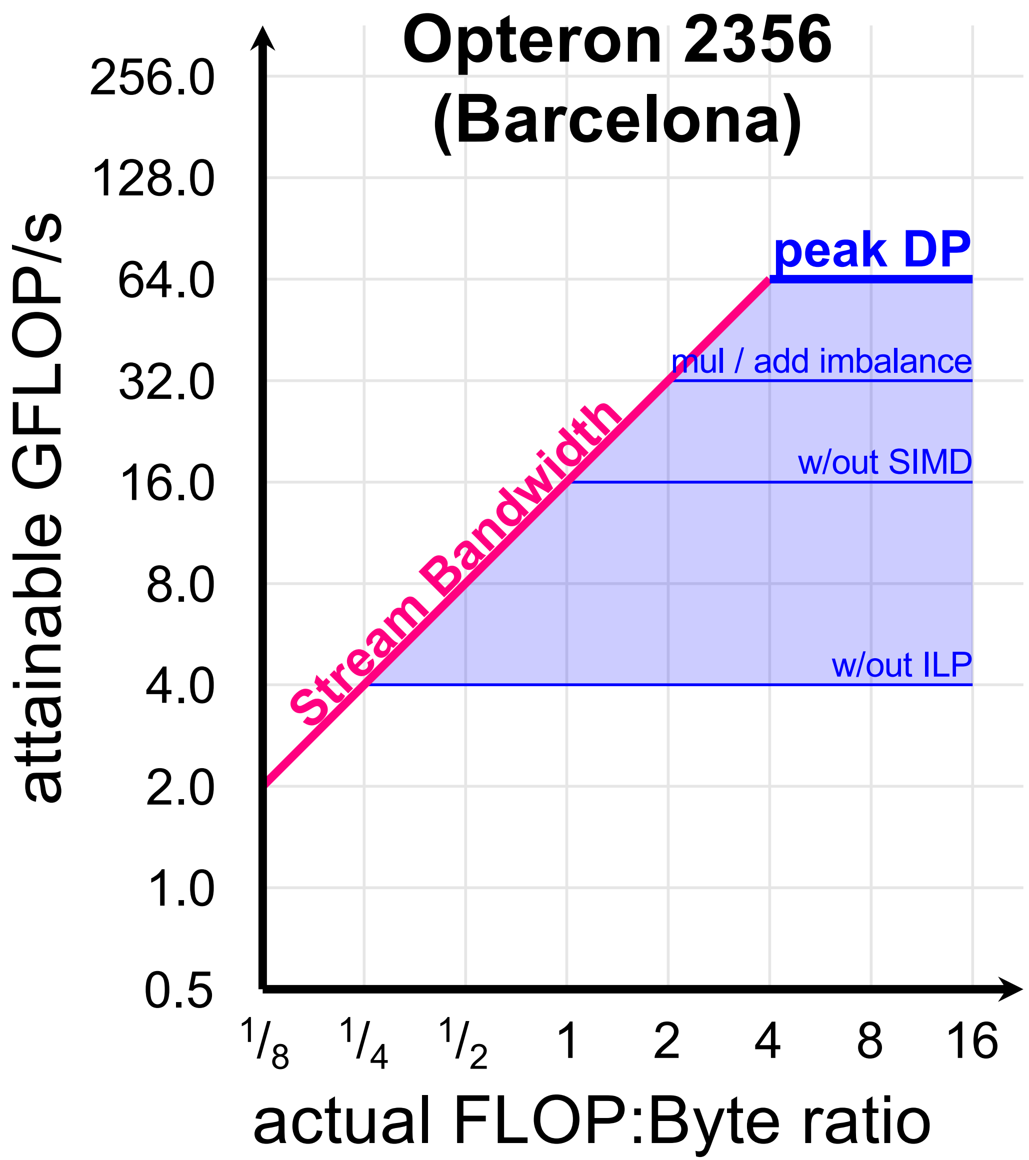
computational ceilings



- Opterons have 128-bit datapaths.
- If instructions aren't SIMDized, attainable performance will be halved

Roofline Model

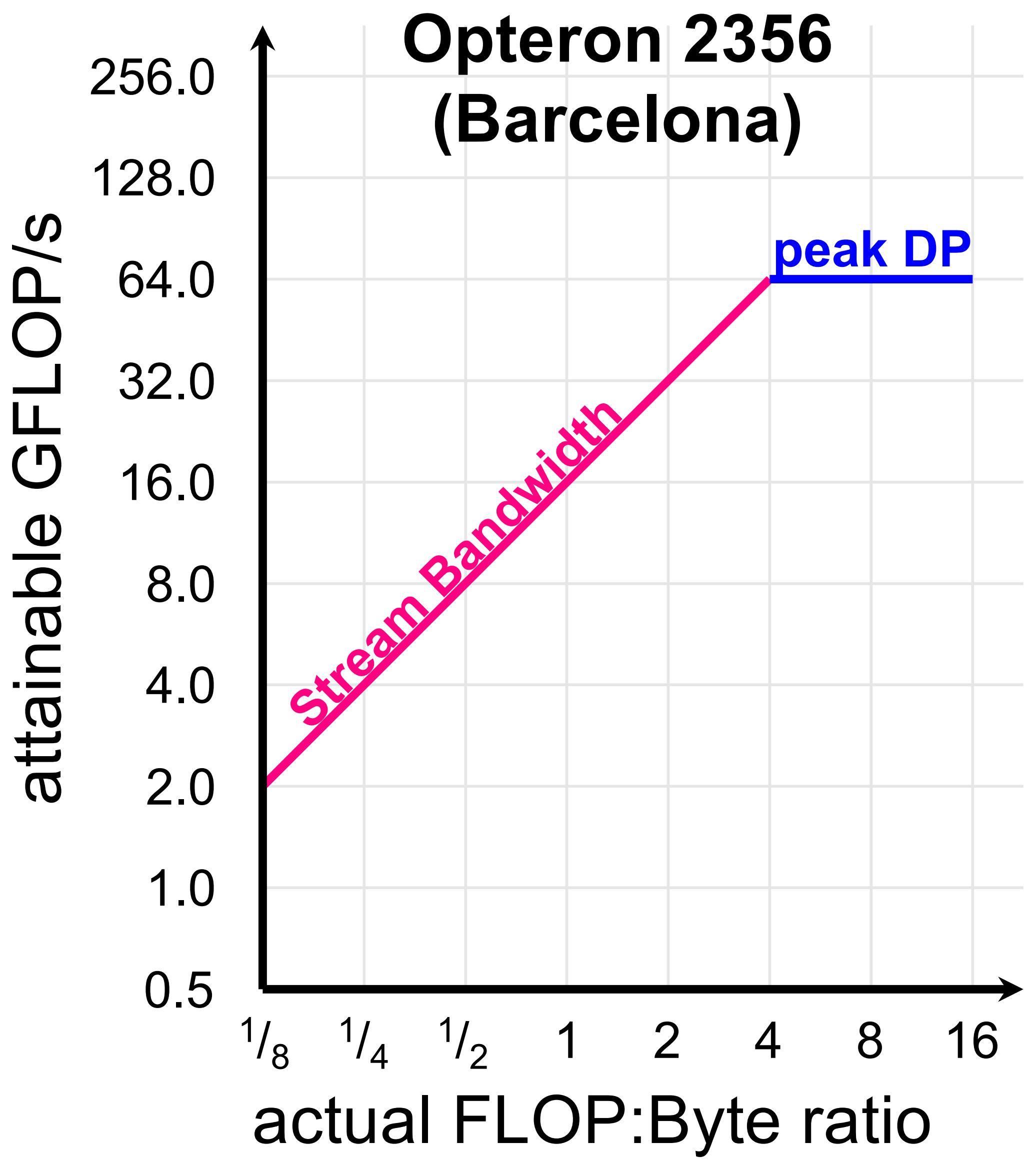
computational ceilings



- On Opterons, floating-point instructions have a 4 cycle latency.
- If we don't express 4-way ILP, performance will drop by as much as 4x

Roofline Model

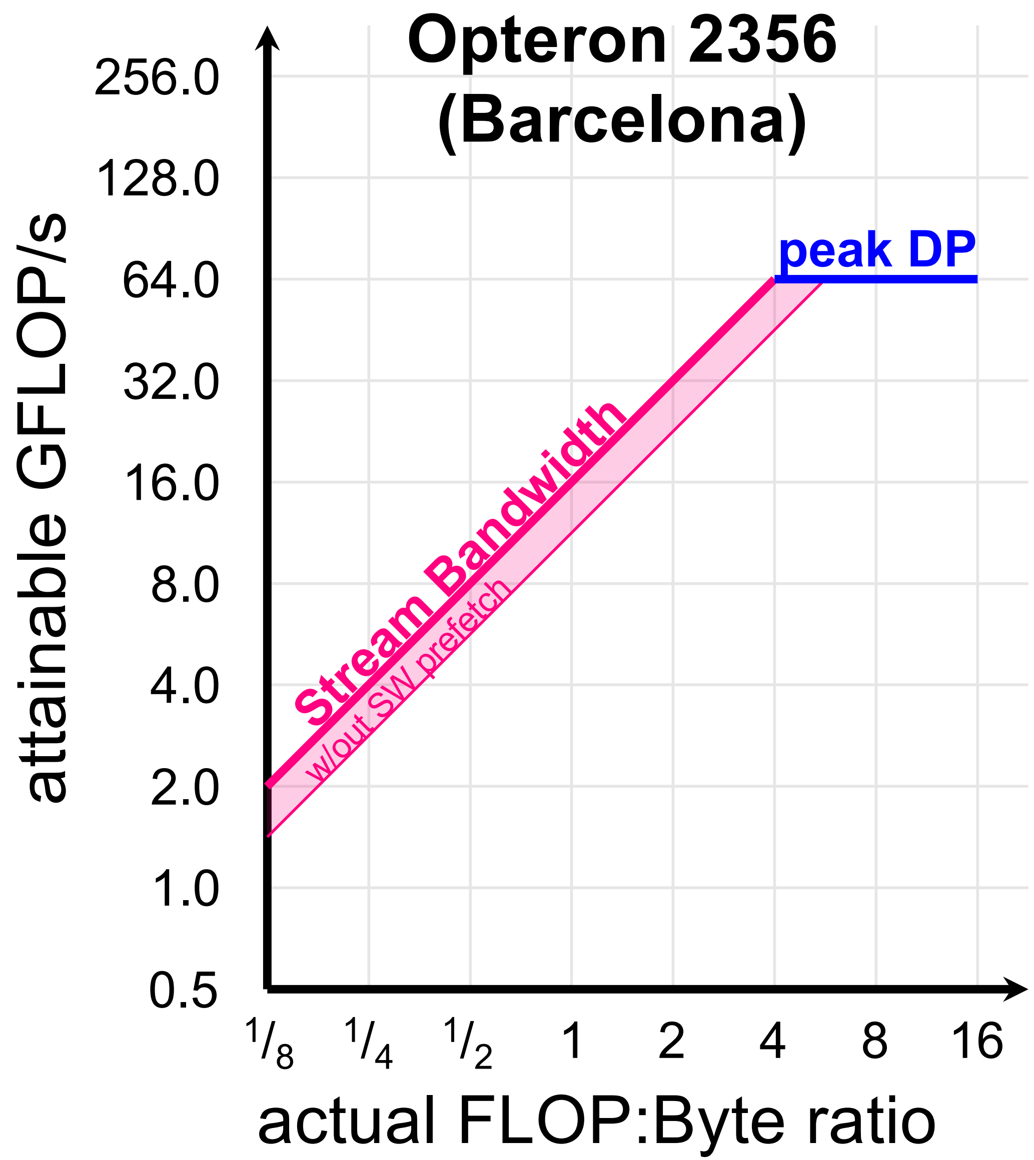
communication ceilings



- We can perform a similar exercise taking away parallelism from the memory subsystem

Roofline Model

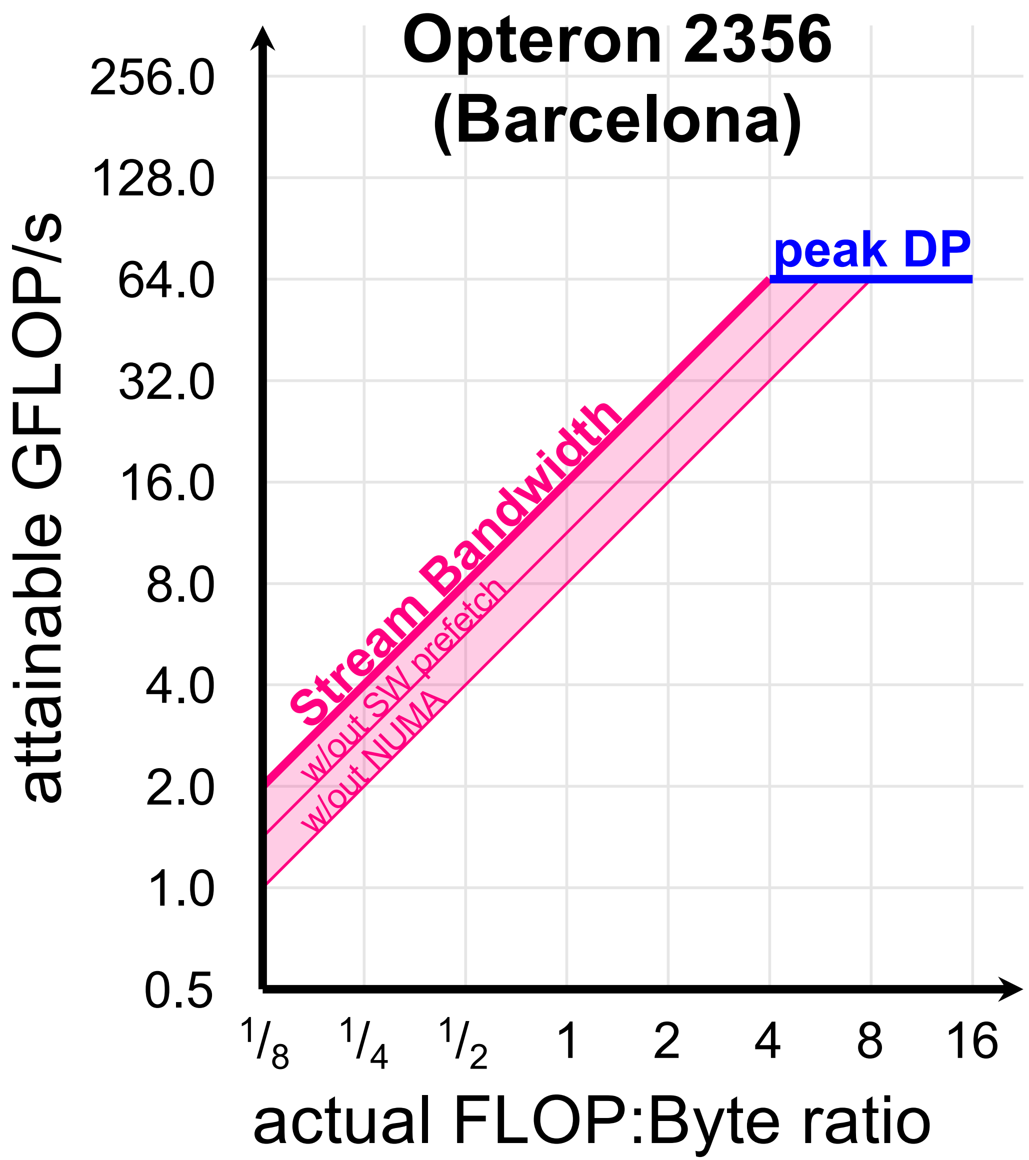
communication ceilings



- Explicit software prefetch instructions are required to achieve peak bandwidth

Roofline Model

communication ceilings

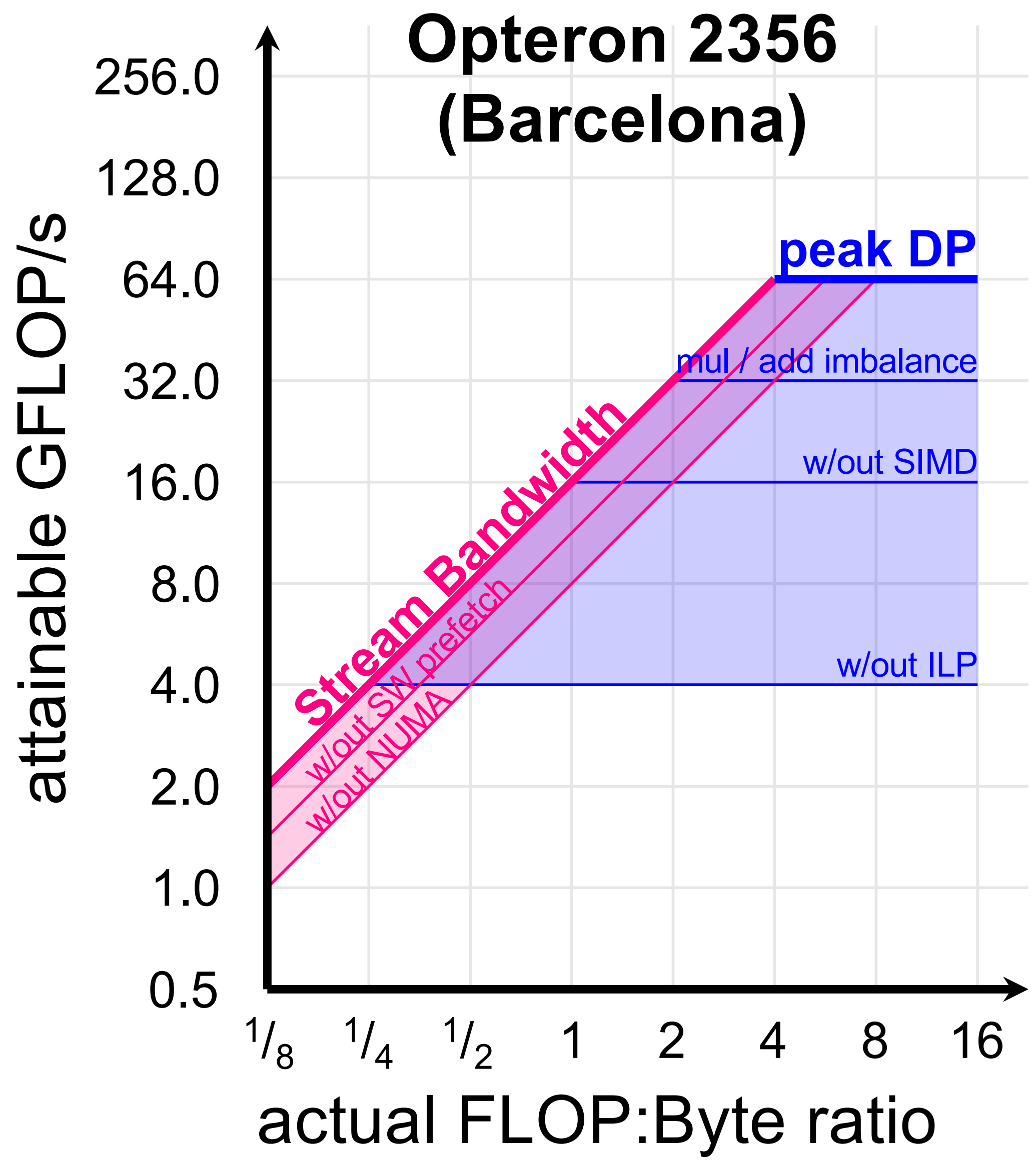


- Opterons are NUMA
- As such memory traffic must be correctly balanced among the two sockets to achieve good Stream bandwidth.

- We could continue this by examining strided or random memory access patterns

Roofline Model

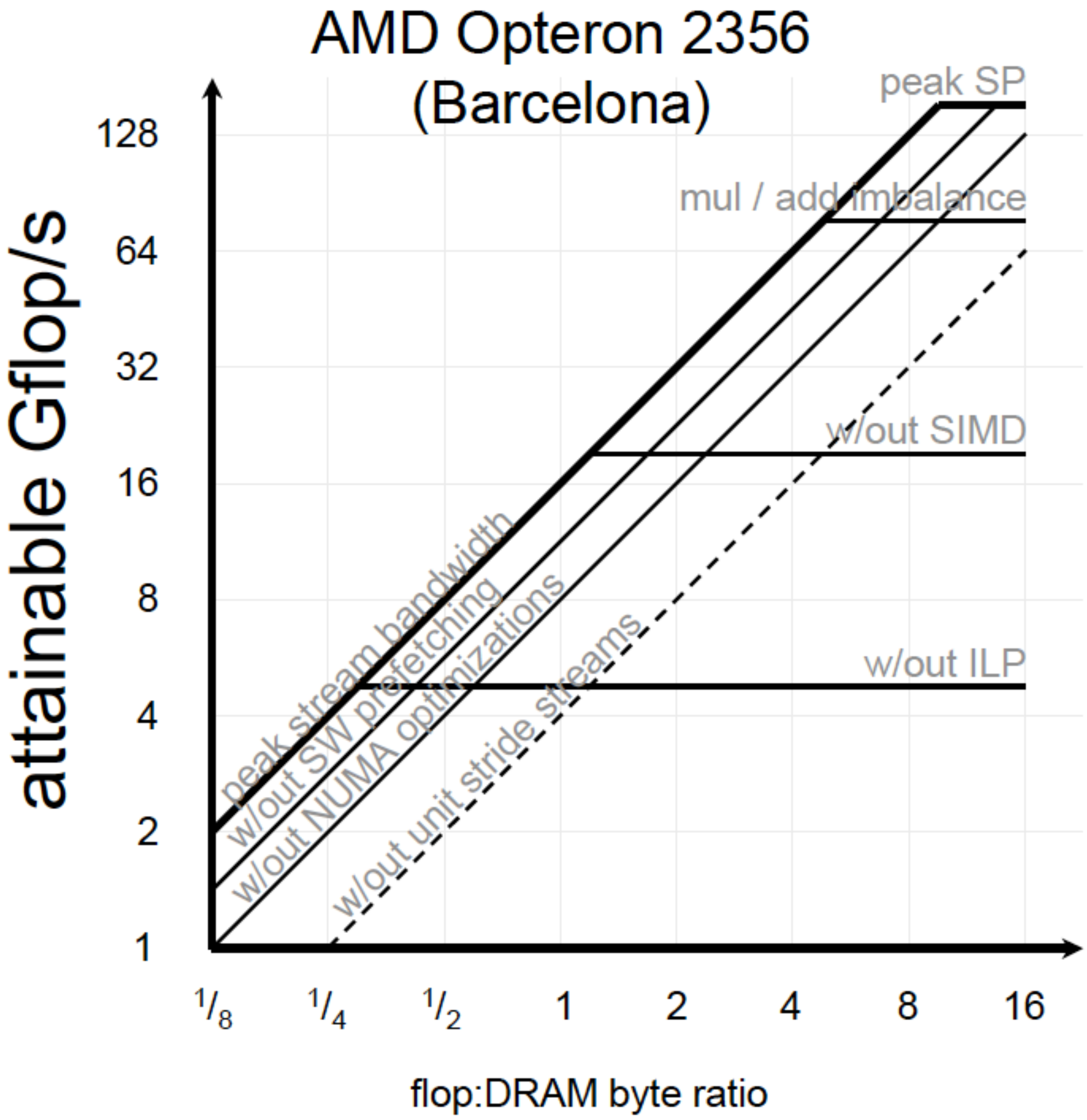
computation + communication ceilings



- We may bound performance based on the combination of expressed in-core parallelism and attained bandwidth.

Roofline model for Opteron

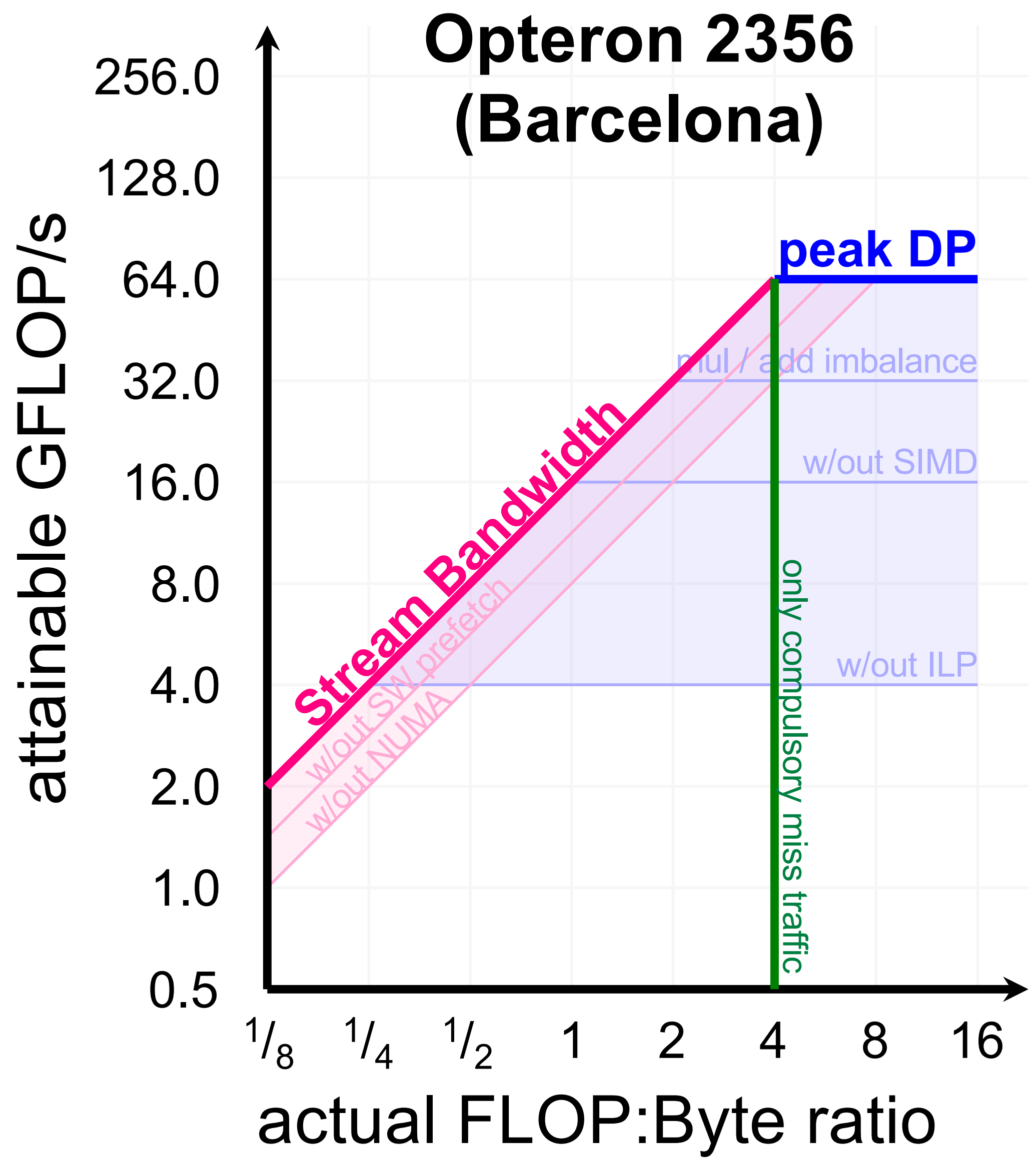
(adding ceilings)



- Bandwidth is much lower without unit stride streams

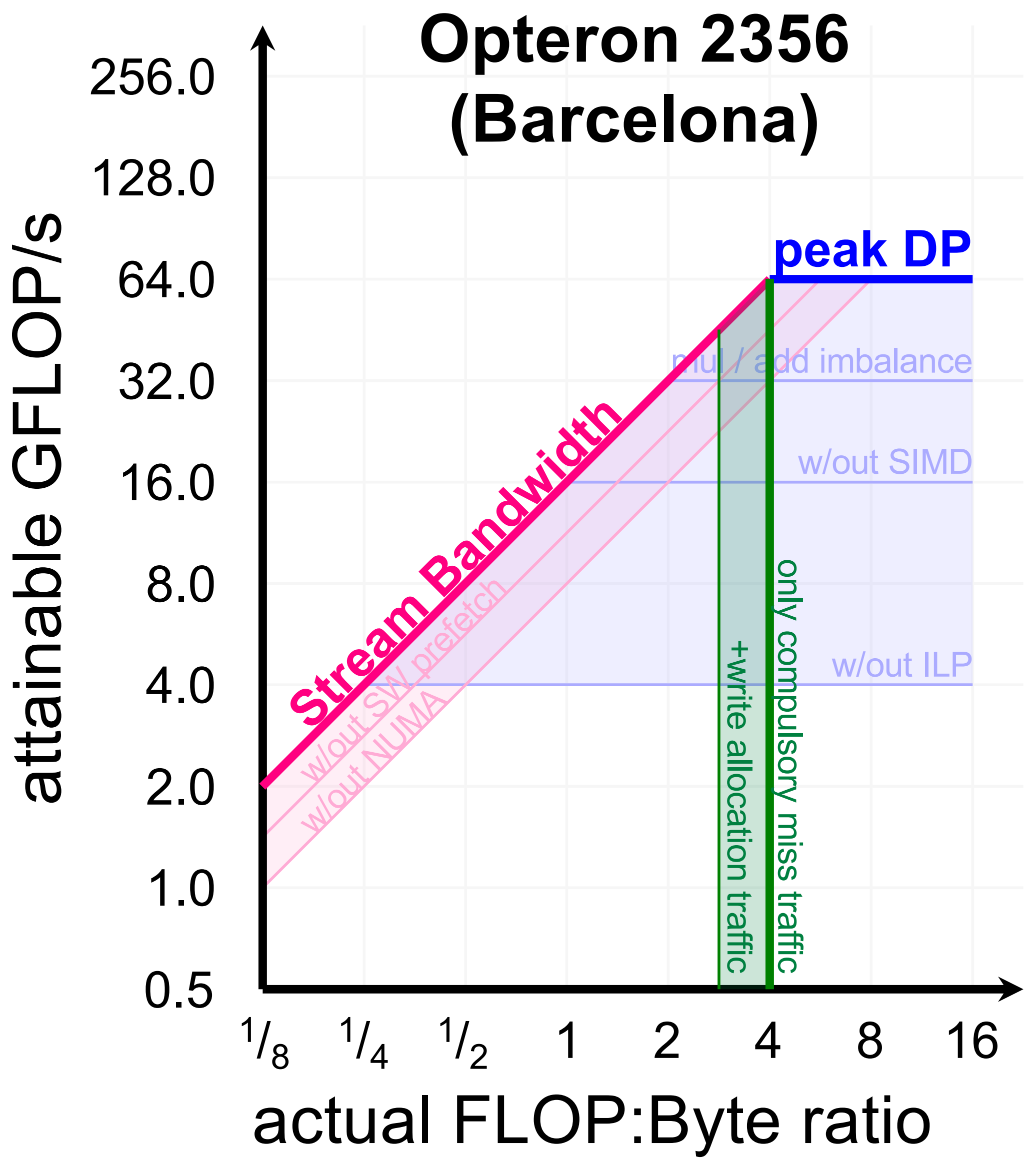
Roofline Model

locality walls



- Remember, memory traffic includes more than just compulsory misses.
- As such, actual arithmetic intensity may be substantially lower.
- Walls are unique to the architecture-kernel combination

$$AI = \frac{\text{FLOPs}}{\text{Compulsory Misses}}$$

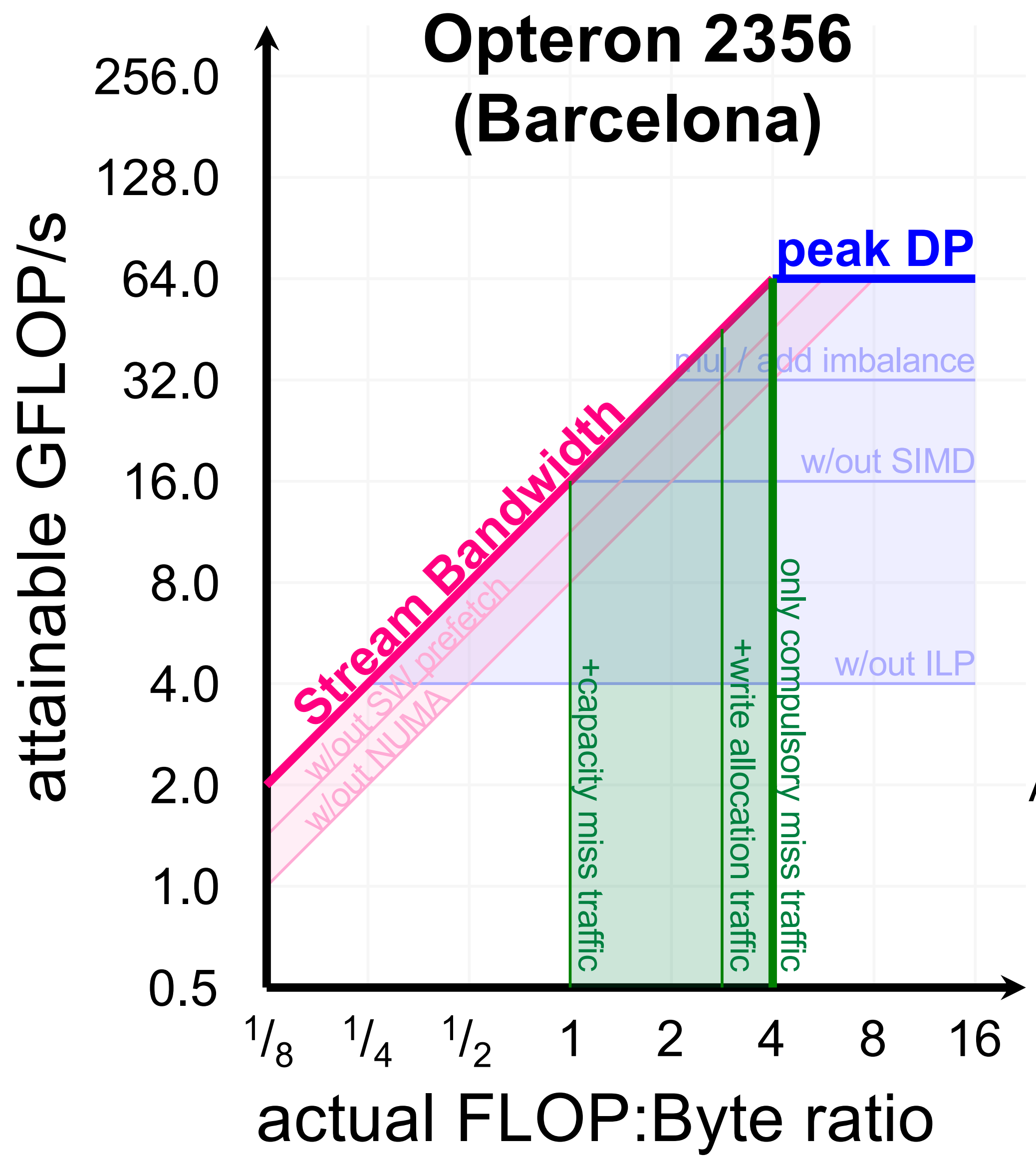


- Remember, memory traffic includes more than just compulsory misses.
- As such, actual arithmetic intensity may be substantially lower.
- Walls are unique to the architecture-kernel combination

$$AI = \frac{\text{FLOPs}}{\text{Allocations + Compulsory Misses}}$$

Roofline Model

locality walls

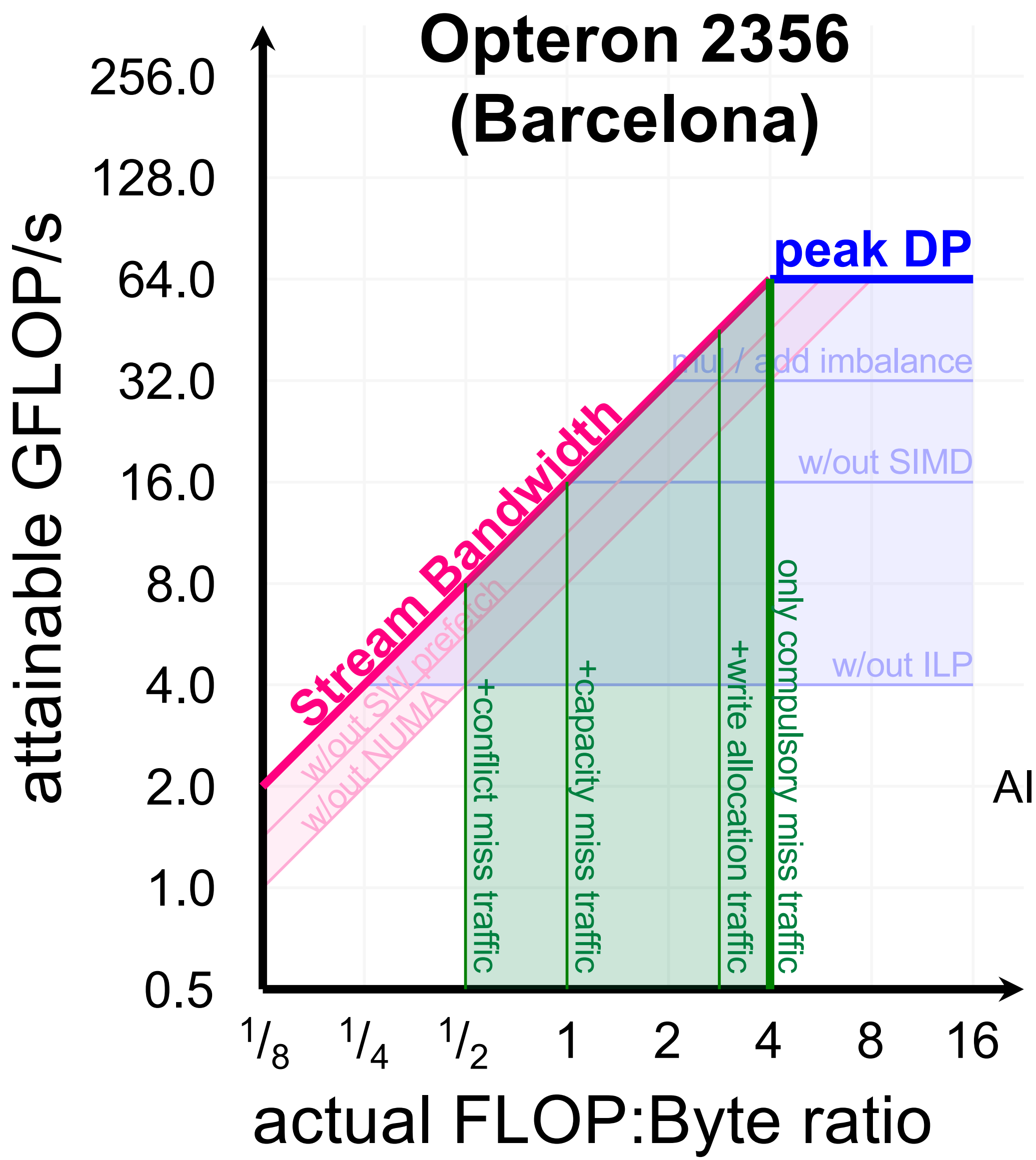


- Remember, memory traffic includes more than just compulsory misses.
- As such, actual arithmetic intensity may be substantially lower.
- Walls are unique to the architecture-kernel combination

$$AI = \frac{\text{FLOPs}}{\text{Capacity + Allocations + Compulsory}}$$

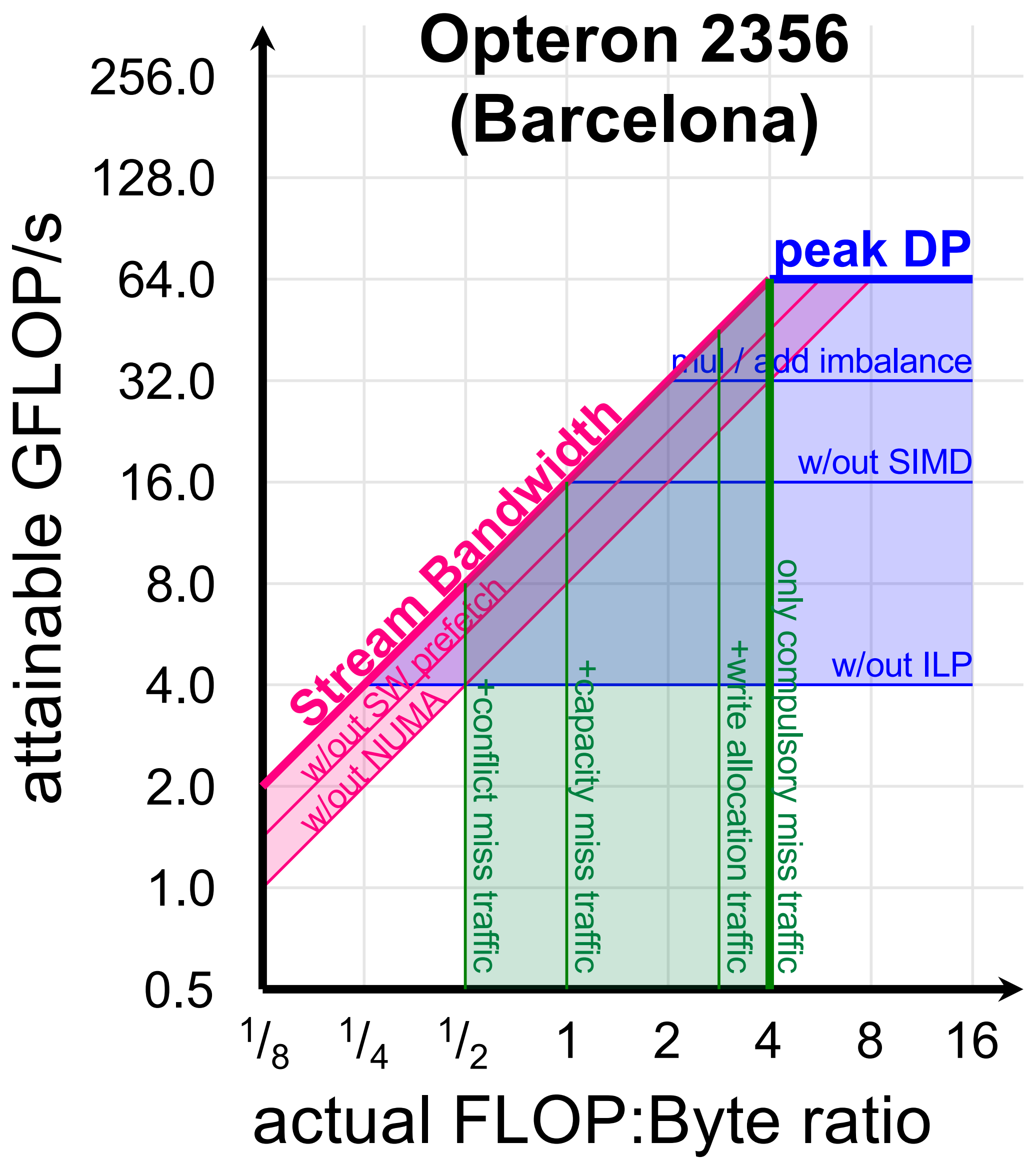
Roofline Model

locality walls



- Remember, memory traffic includes more than just compulsory misses.
- As such, actual arithmetic intensity may be substantially lower.
- Walls are unique to the architecture-kernel combination

$$AI = \frac{\text{FLOPs}}{\text{Conflict + Capacity + Allocations + Compulsory}}$$

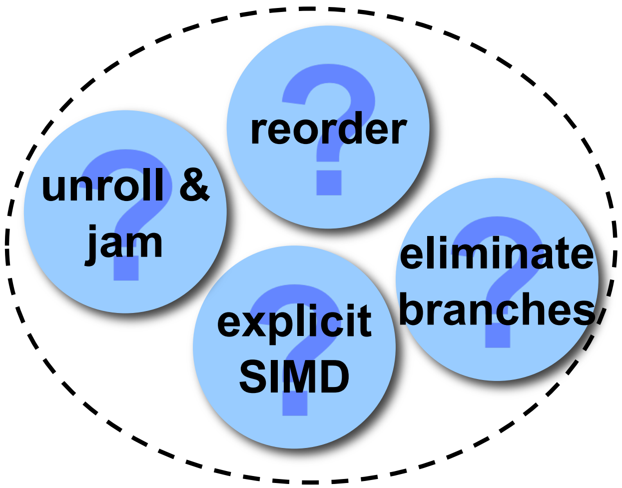


- Optimizations remove these walls and ceilings which act to constrain performance.

Optimization Categorization

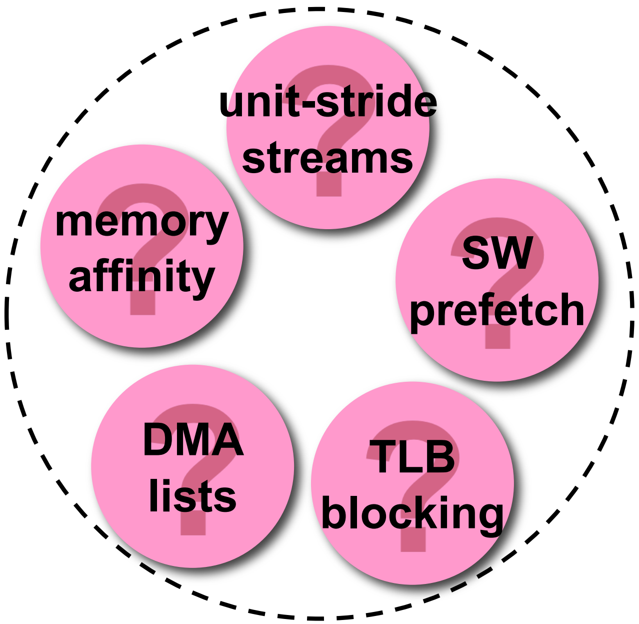
Maximizing In-core Performance

- Exploit in-core parallelism (ILP, DLP, etc…)
- Good (enough) floating-point balance



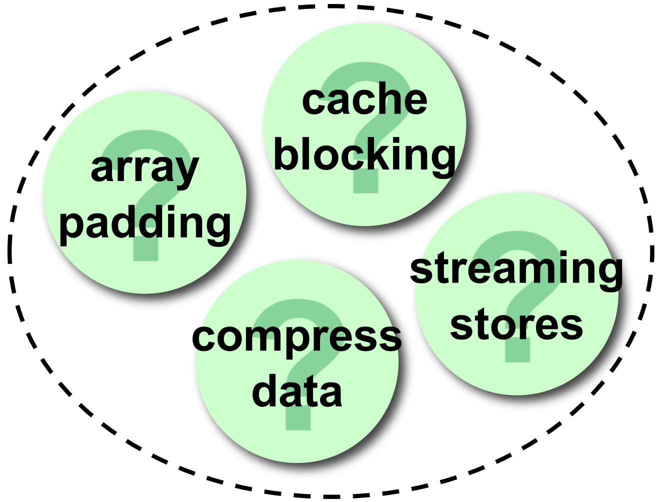
Maximizing Memory Bandwidth

- Exploit NUMA
- Hide memory latency
- Satisfy Little's Law



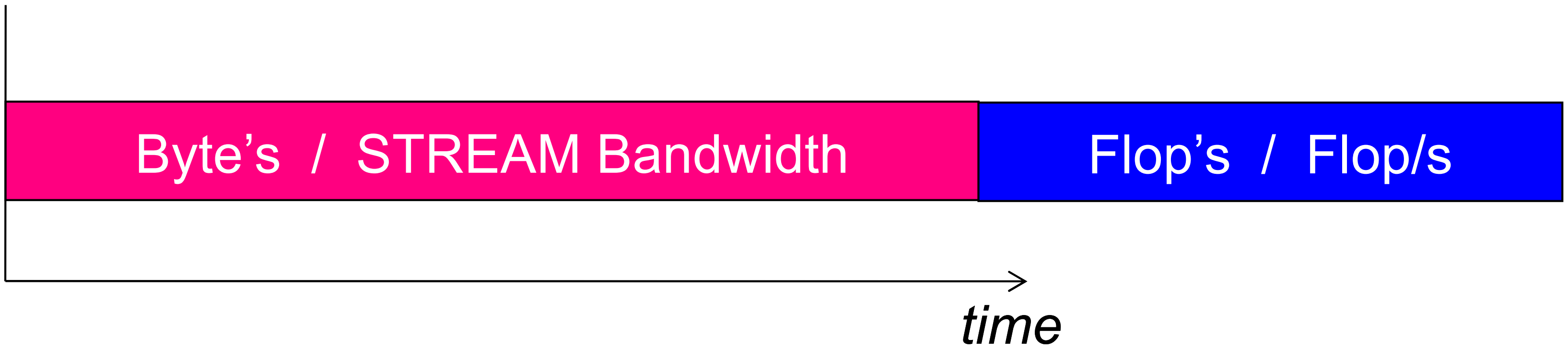
Minimizing Memory Traffic

Eliminate:
- Capacity misses
- Conflict misses
- Compulsory misses
- Write allocate behavior



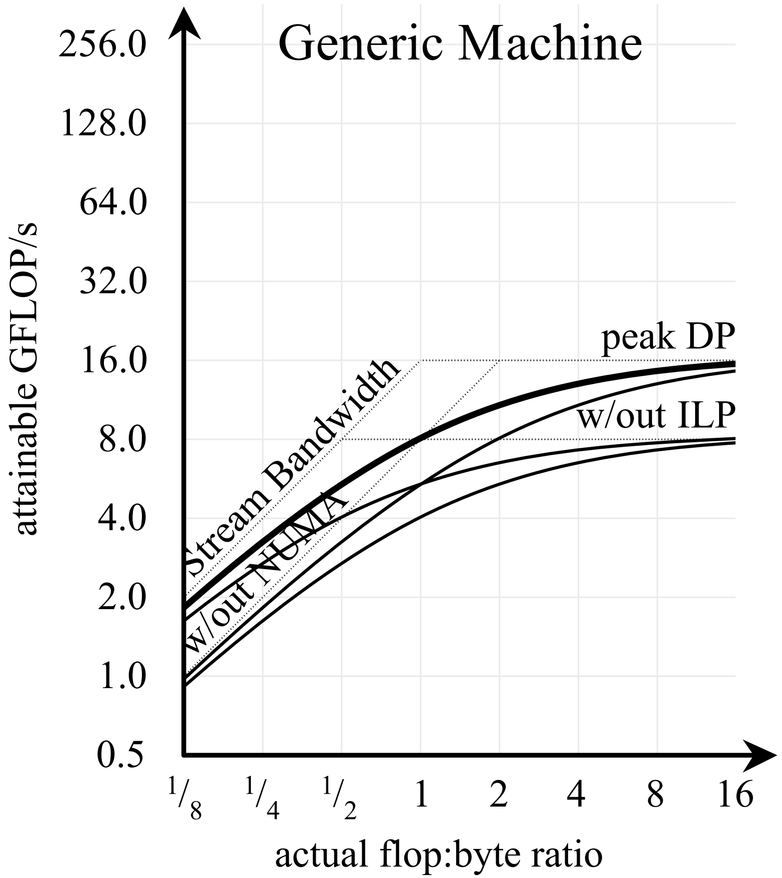
No overlap of communication and computation

- Previously, we assumed perfect overlap of communication or computation.
- What happens if there is a dependency (either inherent or by a lack of optimization) that serializes communication and computation ?



- Time is the sum of communication time and computation time.
- **The result is that flop/s grows asymptotically.**

No overlap of communication and computation



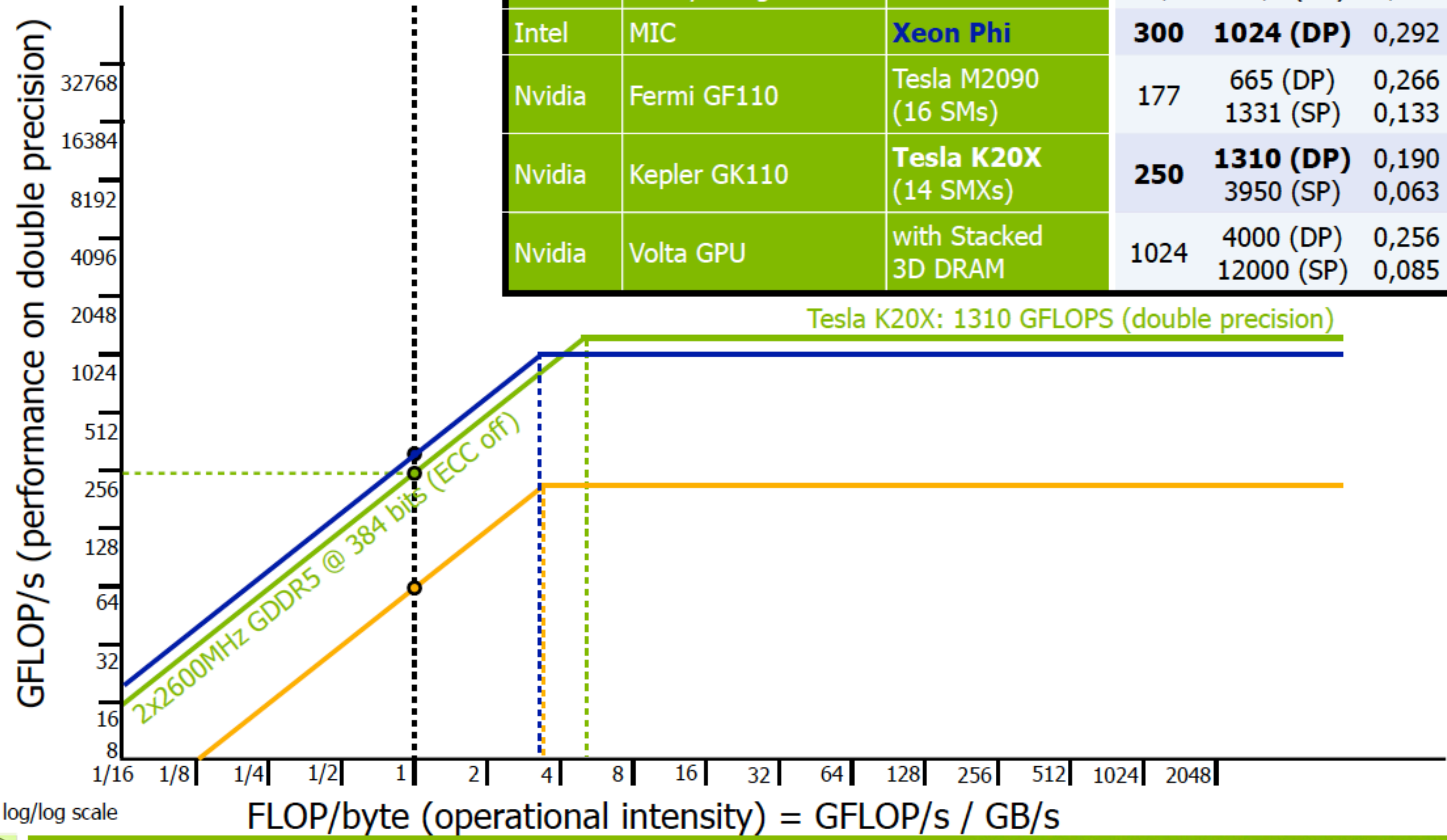
- Consider a generic machine
- If we can perfectly decouple and overlap communication with computation, the roofline is sharp/angular.
- However, without overlap, the roofline is smoothed, **and attainable performance is degraded by up to a factor of 2x.**

Alternate Bandwidths

- Thus far, we assumed a synergy between streaming applications and bandwidth (proxied by the STREAM benchmark)
- **STREAM is NOT a good proxy for short stanza/random cacheline access patterns as memory latency (instead of just bandwidth) is being exposed.**
- Thus one might conceive of alternate memory benchmarks to provide a bandwidth upper bound (ceiling)
- Similarly, if data is primarily local in the LLC cache, one should construct rooflines based on LLC bandwidth and flop:LLC byte ratios.
- For GPUs/accelerators, PCIe bandwidth can be an impediment. Thus one can construct a roofline model based on PCIe bandwidth and the flop:PCIe byte ratio.

Platforms to compare

Vendor	Microarchitecture	Model	GB/s.	GFLOP/s.	Byte/ FLOP
AMD	Bulldozer	Opteron 6284	**59,7**	**217,6 (DP)**	0,235
AMD	Souther Islands	Radeon HD7970	288	1010 (DP)	0,285
Intel	Sandy Bridge	Xeon E5-2690	51,2	243,2 (DP)	0,211
Intel	MIC	**Xeon Phi**	**300**	**1024 (DP)**	0,292
Nvidia	Fermi GF110	Tesla M2090 (16 SMs)	177	665 (DP) 1331 (SP)	0,266 0,133
Nvidia	Kepler GK110	**Tesla K20X** (14 SMXs)	**250**	**1310 (DP)** 3950 (SP)	0,190 0,063
Nvidia	Volta GPU	with Stacked 3D DRAM	1024	4000 (DP) 12000 (SP)	0,256 0,085

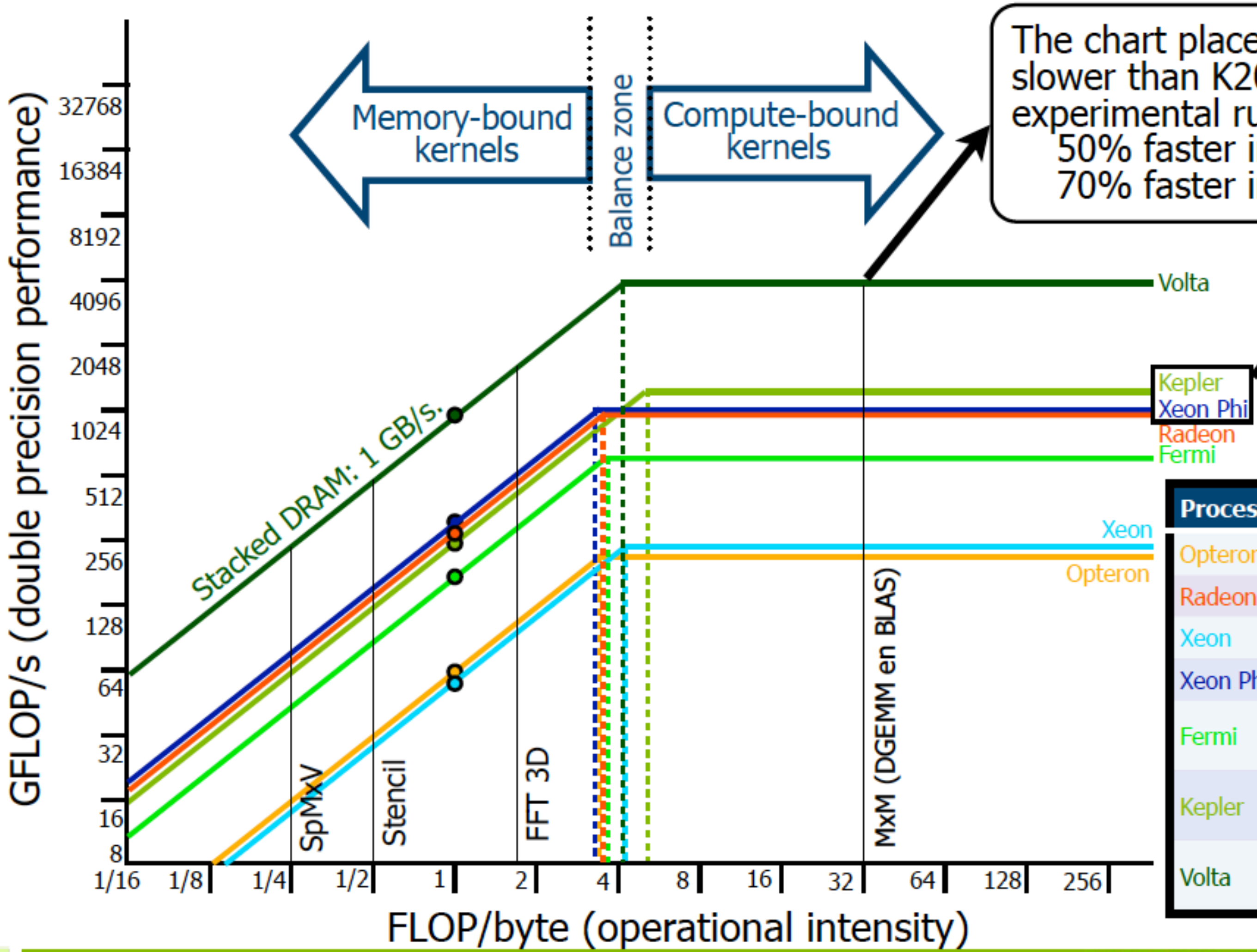


34

Manuel Ujaldon - Nvidia CUDA Fellow

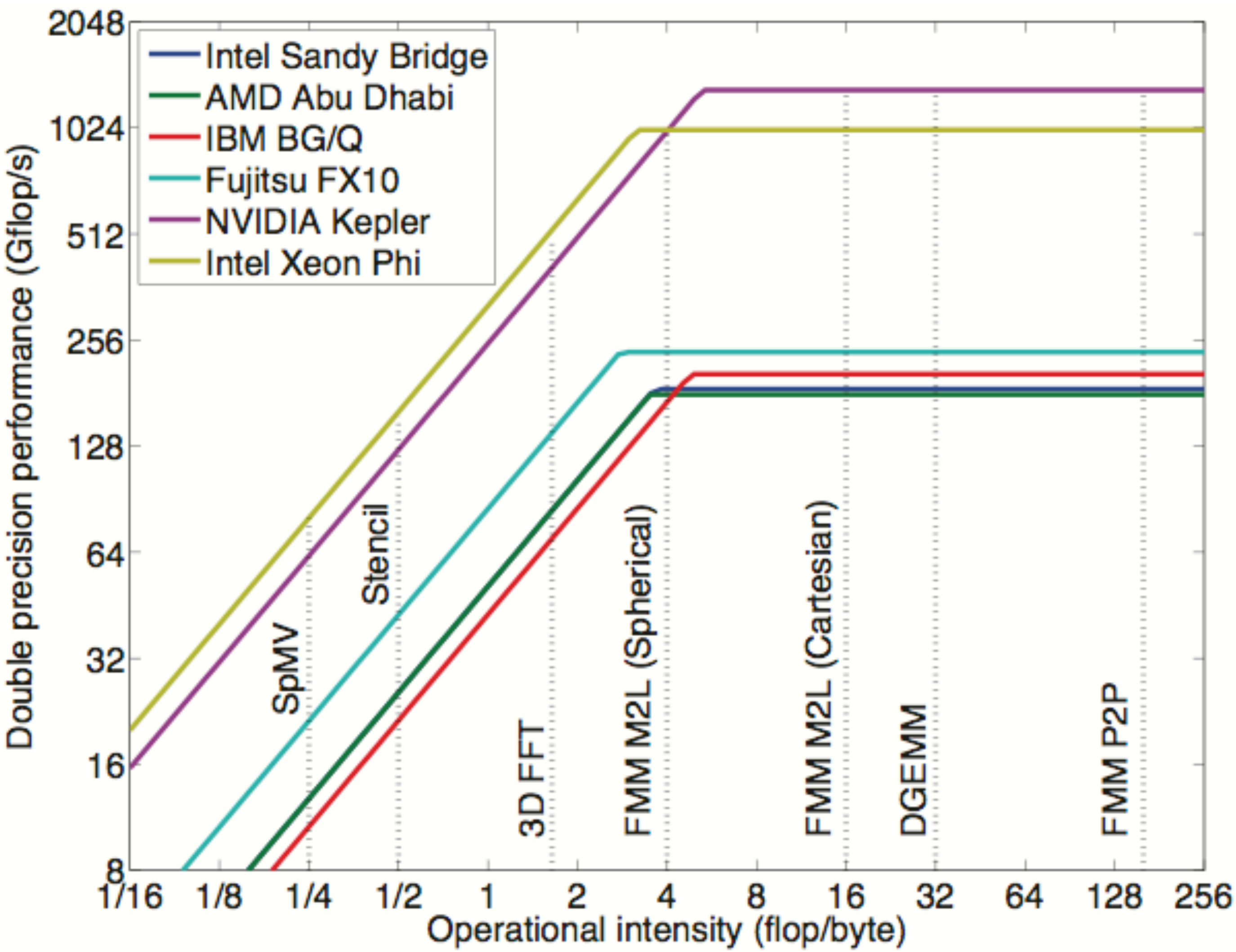
The Roofline model: Hardware vs. Software

The chart places Xeon Phi 225 as 30% slower than K20X on DGEMM, but our experimental runs say that K20X is:
50% faster in double precision.
70% faster in single precision.

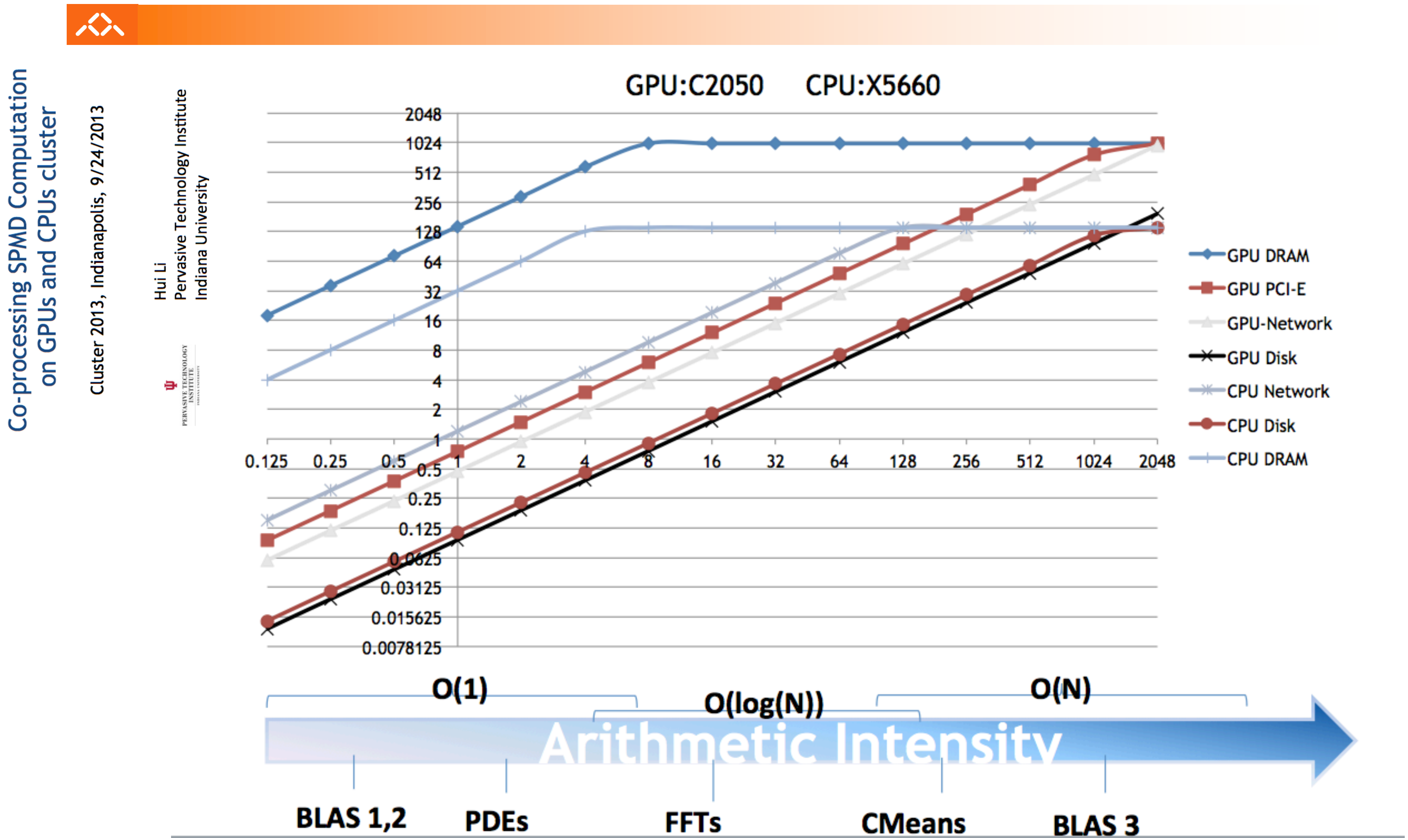


Processor	GB/s.	GFLOP/s.	B/FLOP
Opteron	60	217 (DP)	0,235
Radeon	288	1010 (DP)	0,285
Xeon	51	243 (DP)	0,211
Xeon Phi	300	1024 (DP)	0,292
Fermi	177	665 (DP) 1331 (SP)	0,266 0,133
Kepler	250	1310 (DP) 3950 (SP)	0,190 0,063
Volta	1024	4000 (DP) 12000 (SP)	0,256 0,085

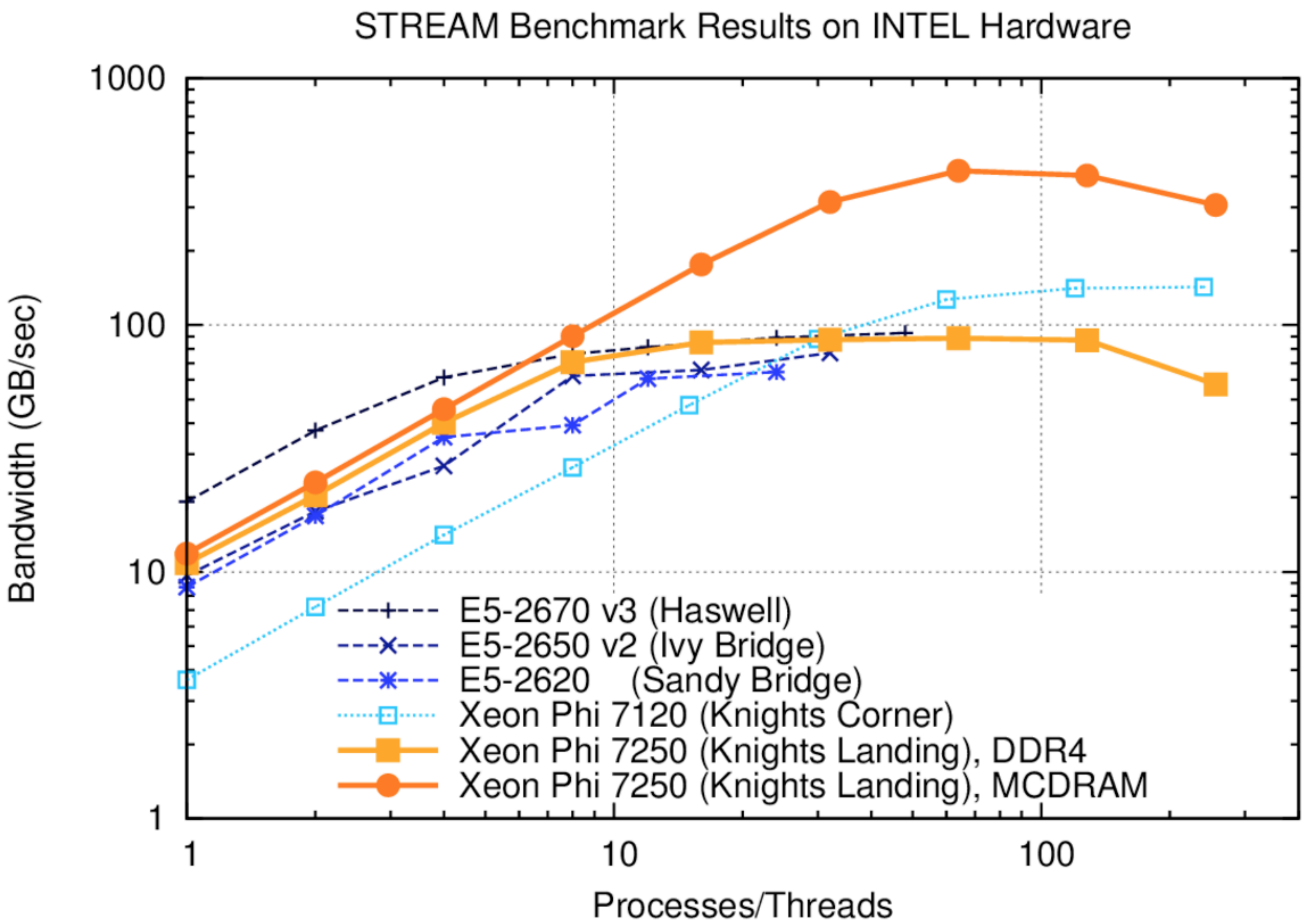
Some more examples



Some more examples



Some more examples



STREAM triad benchmark results for Knights Landing in comparison to three Intel Xeon generations (dual socket) and Knights Corner. A single KNL core achieves about the same memory bandwidth as Ivy Bridge and Sandy Bridge Xeons for small thread counts. Peak bandwidth is obtained with one thread per core; oversubscription reduces memory bandwidth slightly.

https://www.karlrupp.net/2016/07/knights-landing-vs-knights-corner-haswell-ivy-bridge-and-sandy-bridge-stream-benchmark-results/#more-636

Alternate Computations

- Arising from HPC kernels, its no surprise roofline use DP Flop/s.
- Of course, it could use
  - SP flop/s,
  - integer ops,
  - bit operations,
  - pairwise comparisons (sorting),
  - graphics operations,
  - etc…