The Roofline Model:

A pedagogical tool for program analysis and optimization

ParLab Summer Retreat

Samuel Williams, David Patterson

samw@cs.berkeley.edu

Motivation

- Performance and scalability of multicore architectures can be extremely non-intuitive to novice programmers
- Success of the multicore paradigm should be premised on augmenting the abilities of the world's programmers

Goals & Audience

- Focused on:

  rates and efficiencies (Gflop/s, % of peak),

- Goals for Roofline:
  - Provide everyone with a graphical aid that provides:

    realistic expectations of performance and productivity
  - Show inherent hardware limitations for a given kernel
  - Show potential benefit and priority of optimizations

- Who's not the audience for the Roofline:
  - Not for those interested in fine tuning (+5%)
  - Not for those challenged by parallel kernel correctness

Principal Components of Performance

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)

- 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

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

Roofline Model

Integrating Components…

- Goal: integrate in-core performance, memory bandwidth, and locality into a single readily understandable performance figure
- Also, must graphically show the penalty associated with not including certain software optimizations

- Roofline model will be unique to each architecture
- Coordinates of a kernel are ~unique to each architecture

What relates GB/s to GFlop/s ?

- Through dimensional analysis, its clear that **Flops:Bytes** is the parameter that allows us to convert bandwidth (GB/s) to performance (GFlop/s)

- This is a well known quantity: **Arithmetic Intensity** (discussed later)
- When we measure total bytes, we incorporate all cache behavior (the 3C’s) and Locality

Basic Roofline

- Performance is upper bounded by both the peak flop rate, and the product of streaming bandwidth and the flop:byte ratio

$$\textbf{Gflop/s} = \textbf{min}\begin{cases}\textbf{Peak Gflop/s} \\ \textbf{Stream BW * actual flop:byte ratio}\end{cases}$$

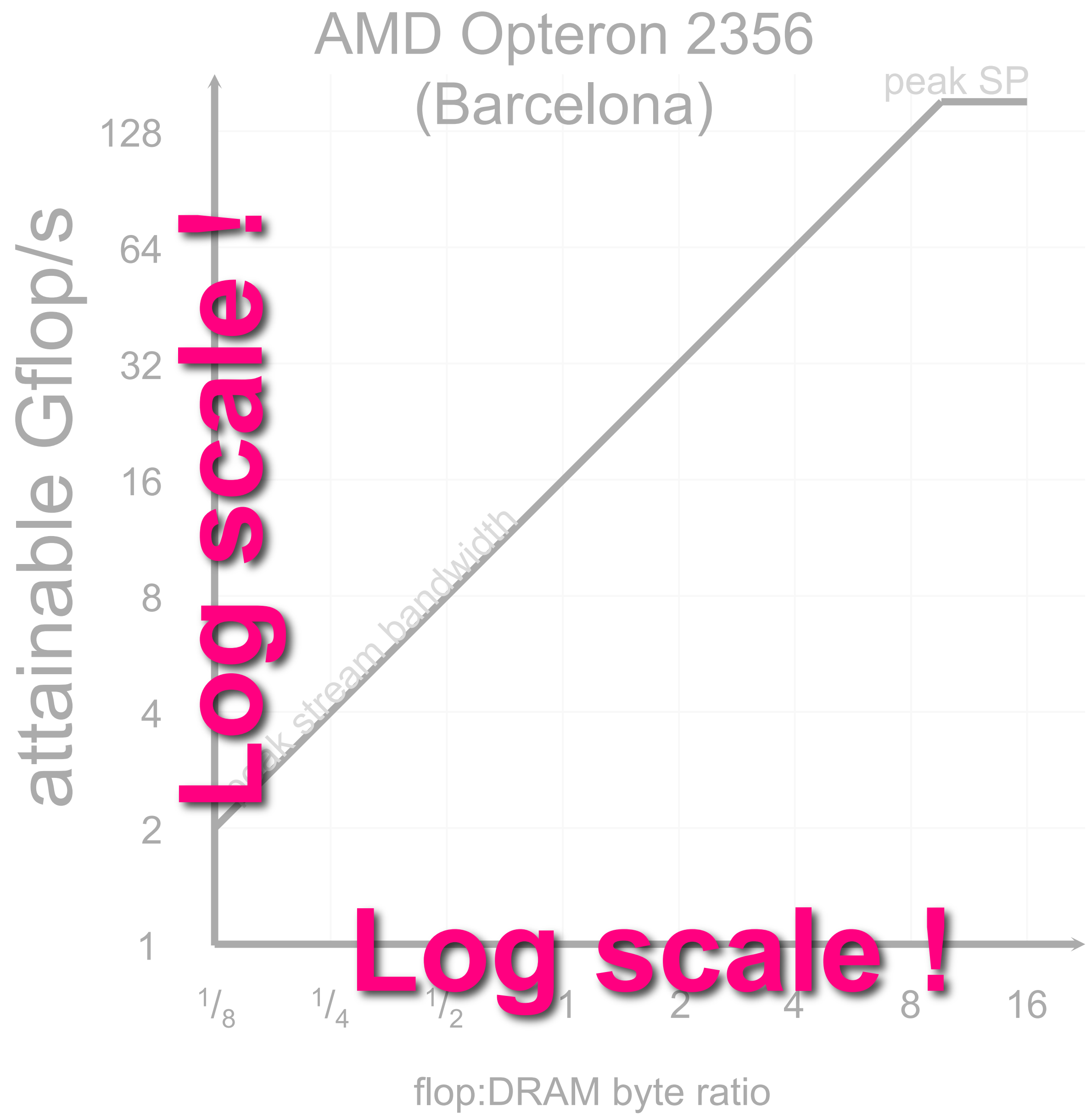
notes

- *Bandwidth #'s collected via micro benchmarks*
- *Computation #'s derived from optimization manuals (pencil and paper)*
- *Assume complete overlap of either communication or computation*

EECS
Electrical Engineering and
Computer Sciences

Roofline model for Opteron
(adding ceilings)

BERKELEY PAR LAB



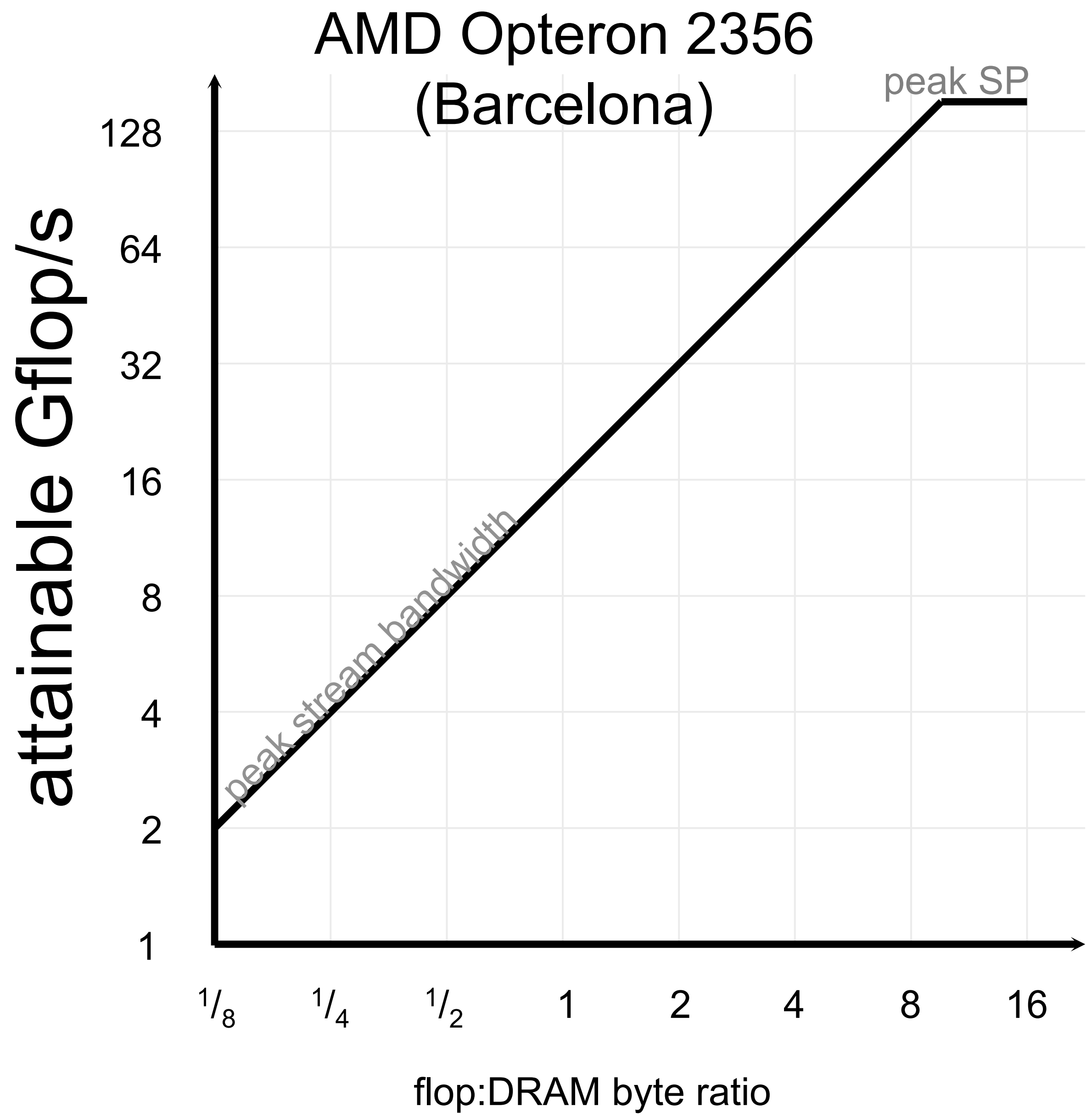
Log scale !

Log scale !

- Peak roofline performance
- based on manual for **single precision peak**
- and a hand tuned stream read for bandwidth

Roofline model for Opteron

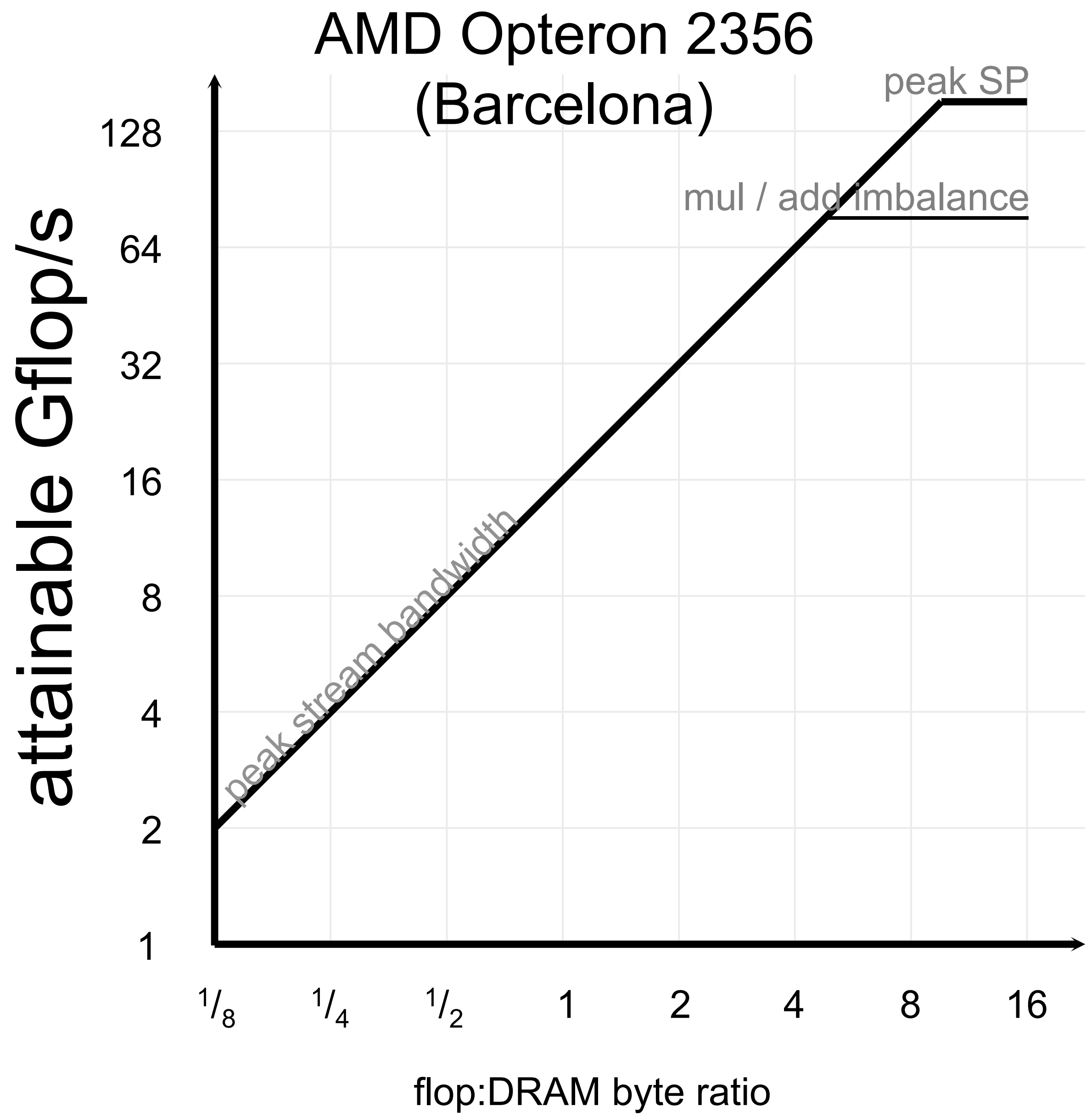
(adding ceilings)



- Peak roofline performance
- based on manual for **single precision peak**
- and a hand tuned stream read for bandwidth

Roofline model for Opteron

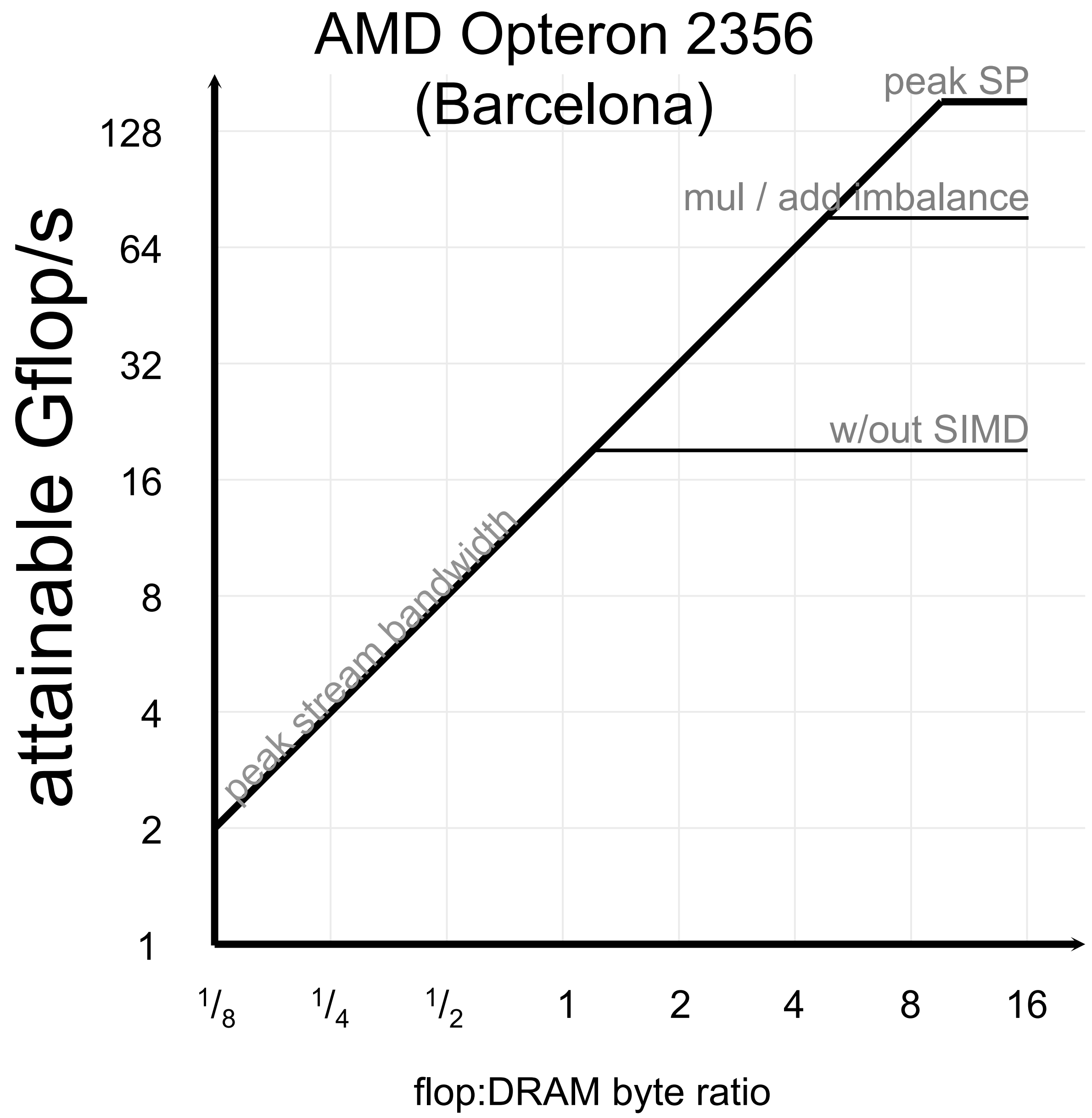
(adding ceilings)



- Opterons have separate multipliers and adders
- 'functional unit parallelism'
- This is a ceiling beneth the roofline

Roofline model for Opteron

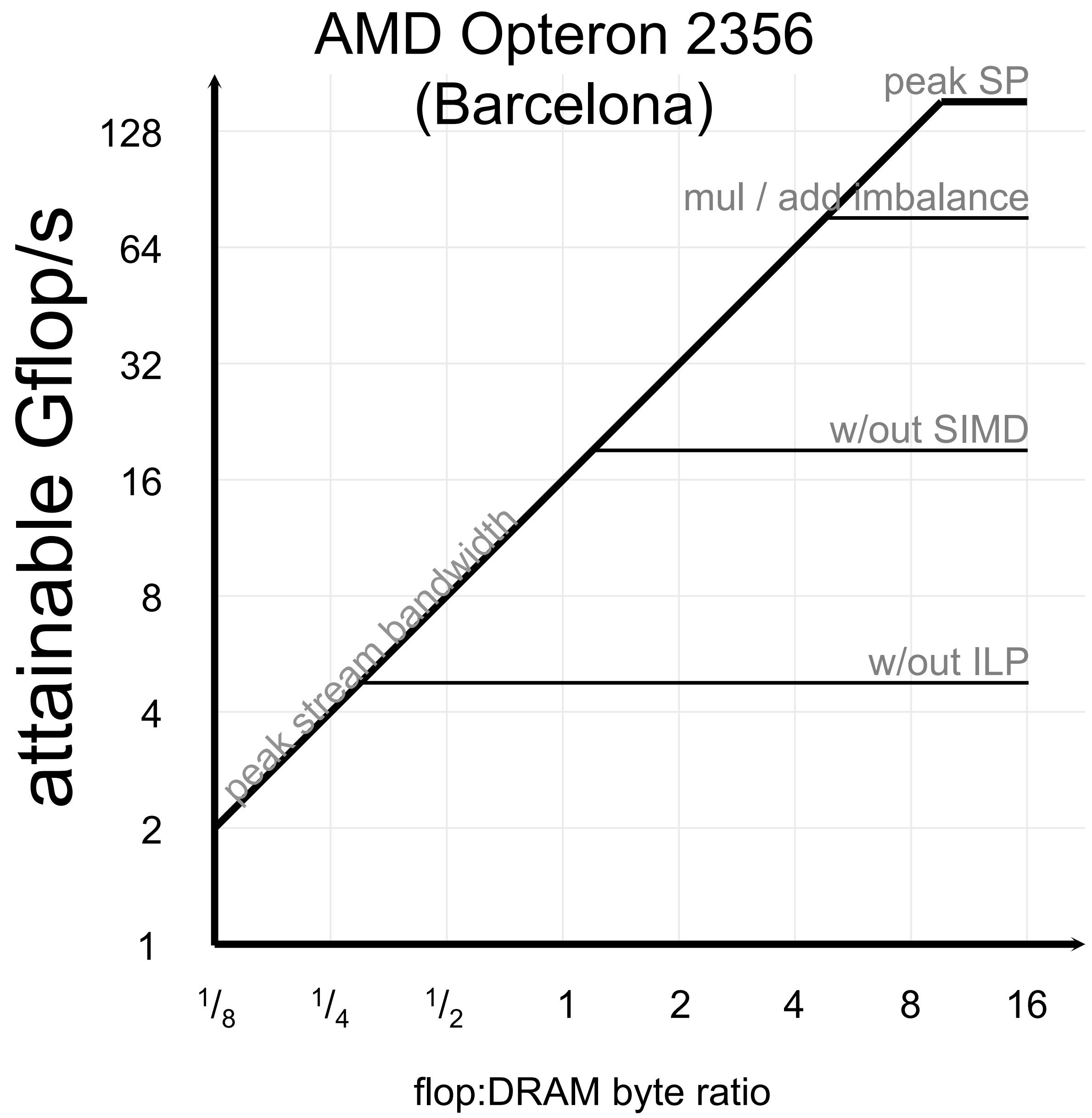
(adding ceilings)



- In single precision, SIMD is 4x32b.
- If only the _ss versions are used, performance is 1/4

Roofline model for Opteron

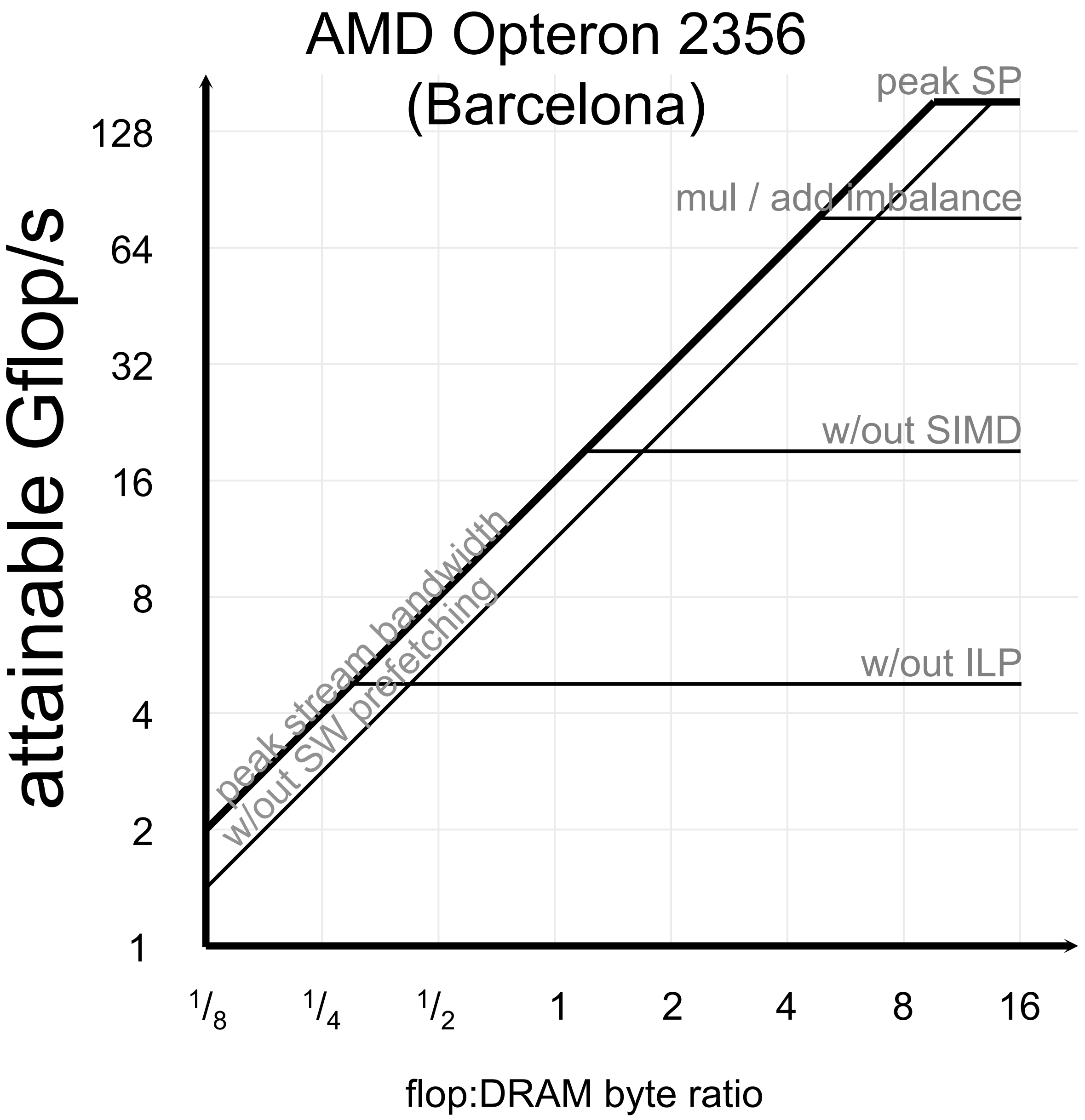
(adding ceilings)



- If 4 independent instructions are kept in the pipeline, performance will fall

Roofline model for Opteron

(adding ceilings)



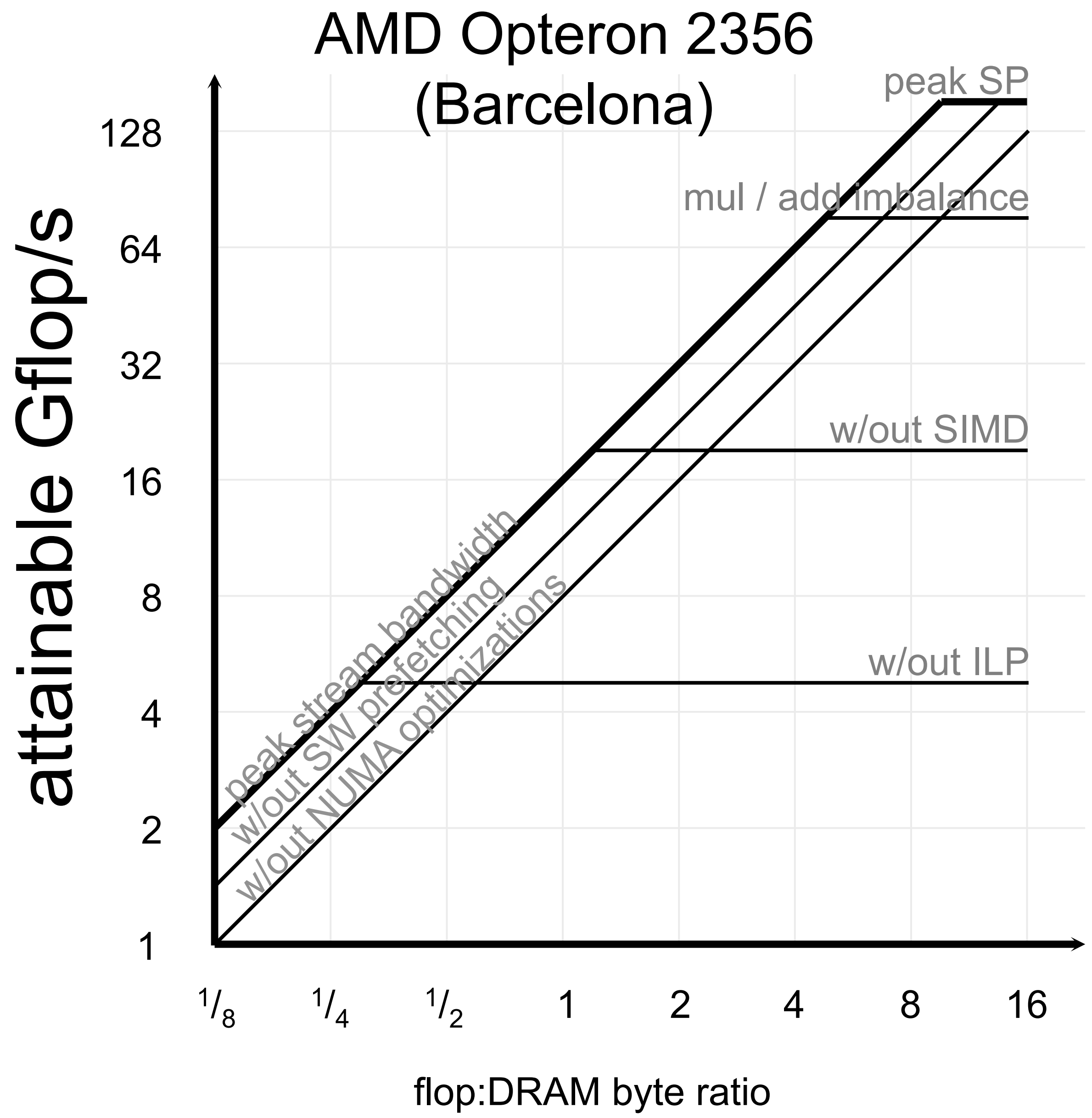
- If SW prefetching is not used, performance will degrade
- These act as ceilings below the bandwidth roofline

EECS
Electrical Engineering and Computer Sciences

Roofline model for Opteron

(adding ceilings)

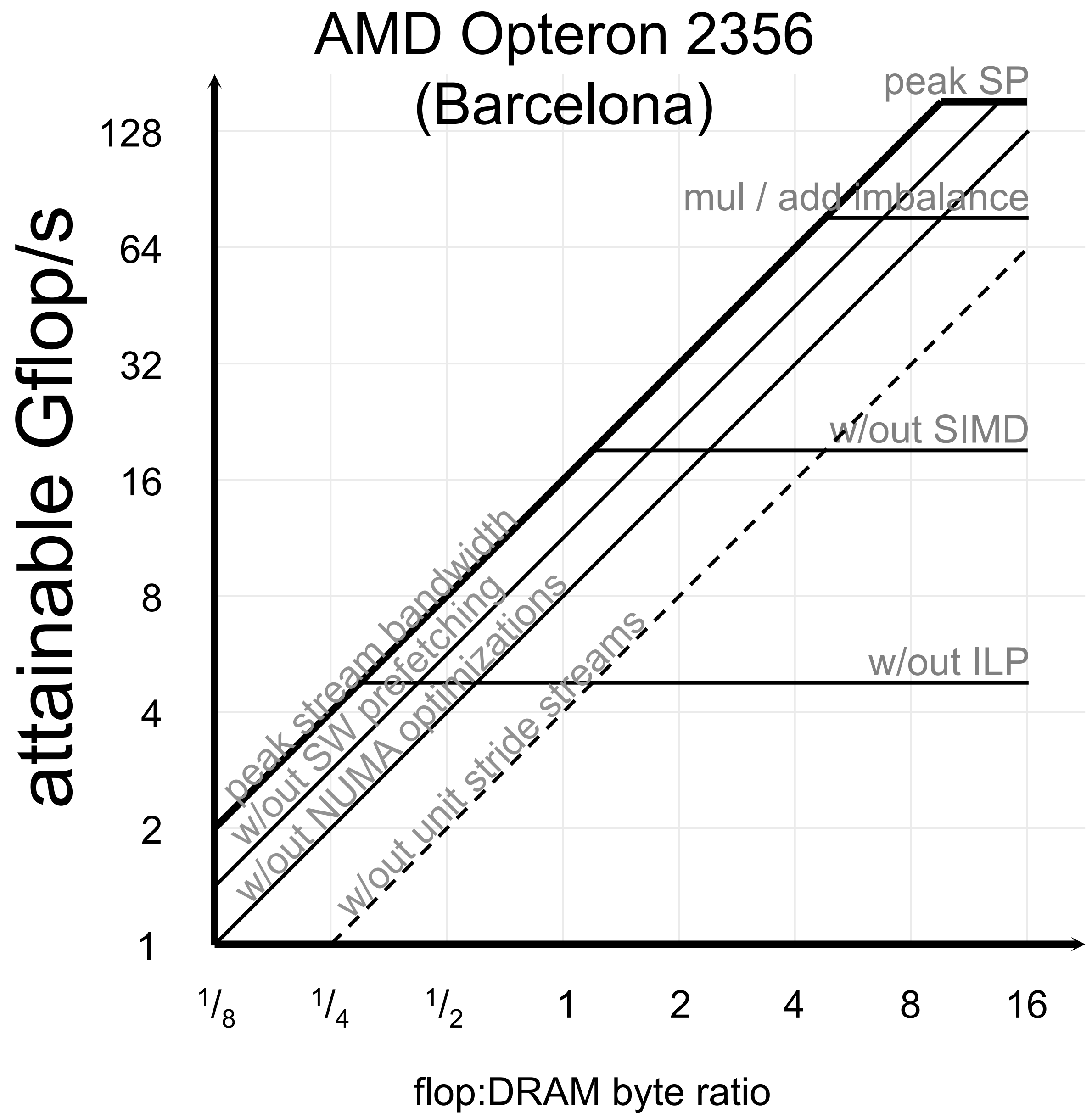
BERKELEY PAR LAB



- Without NUMA optimizations, the memory controllers on the second socket can’t be used.

Roofline model for Opteron

(adding ceilings)



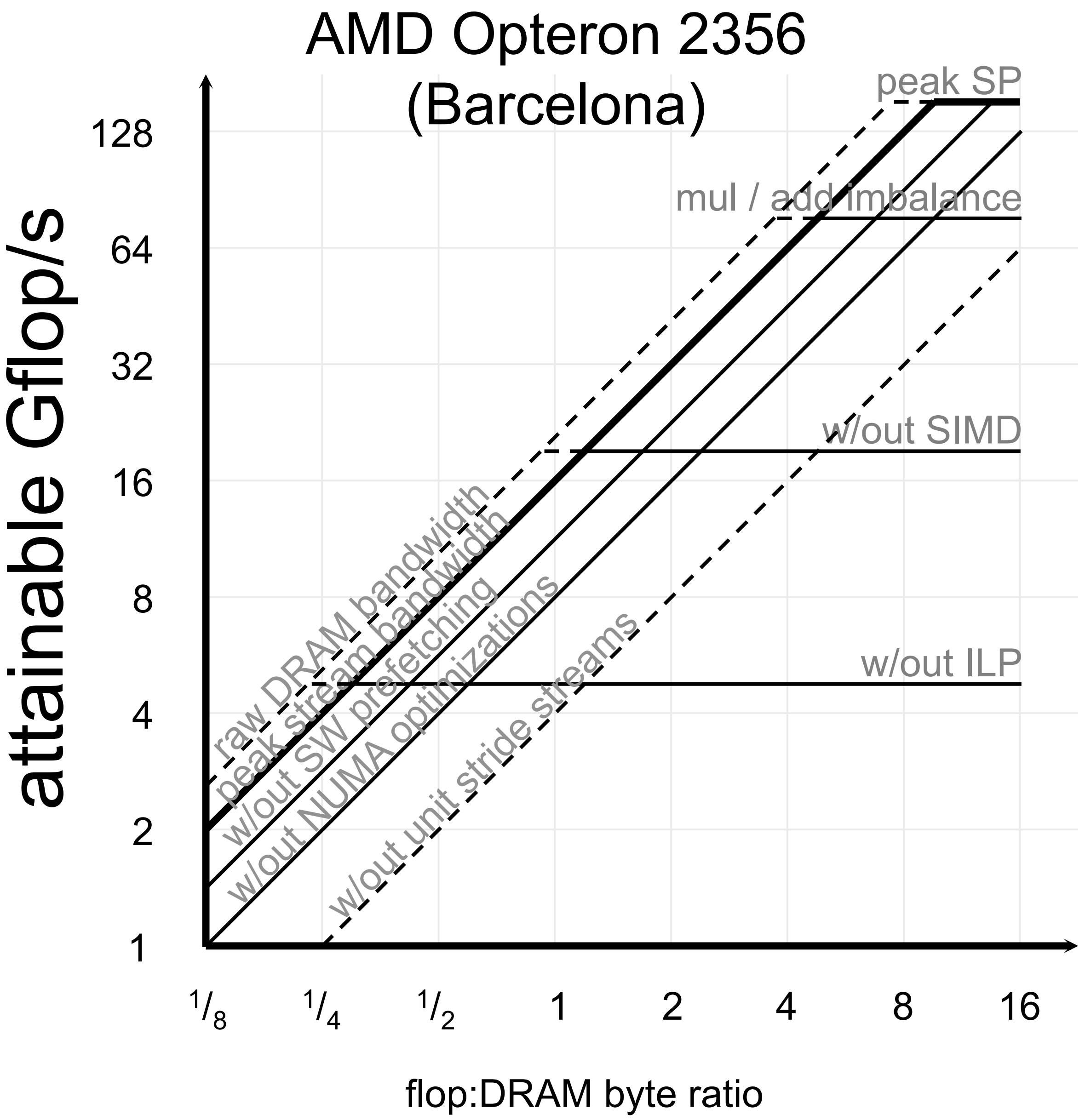
- Bandwidth is much lower without unit stride streams

EECS
Electrical Engineering and
Computer Sciences

Roofline model for Opteron

(adding ceilings)

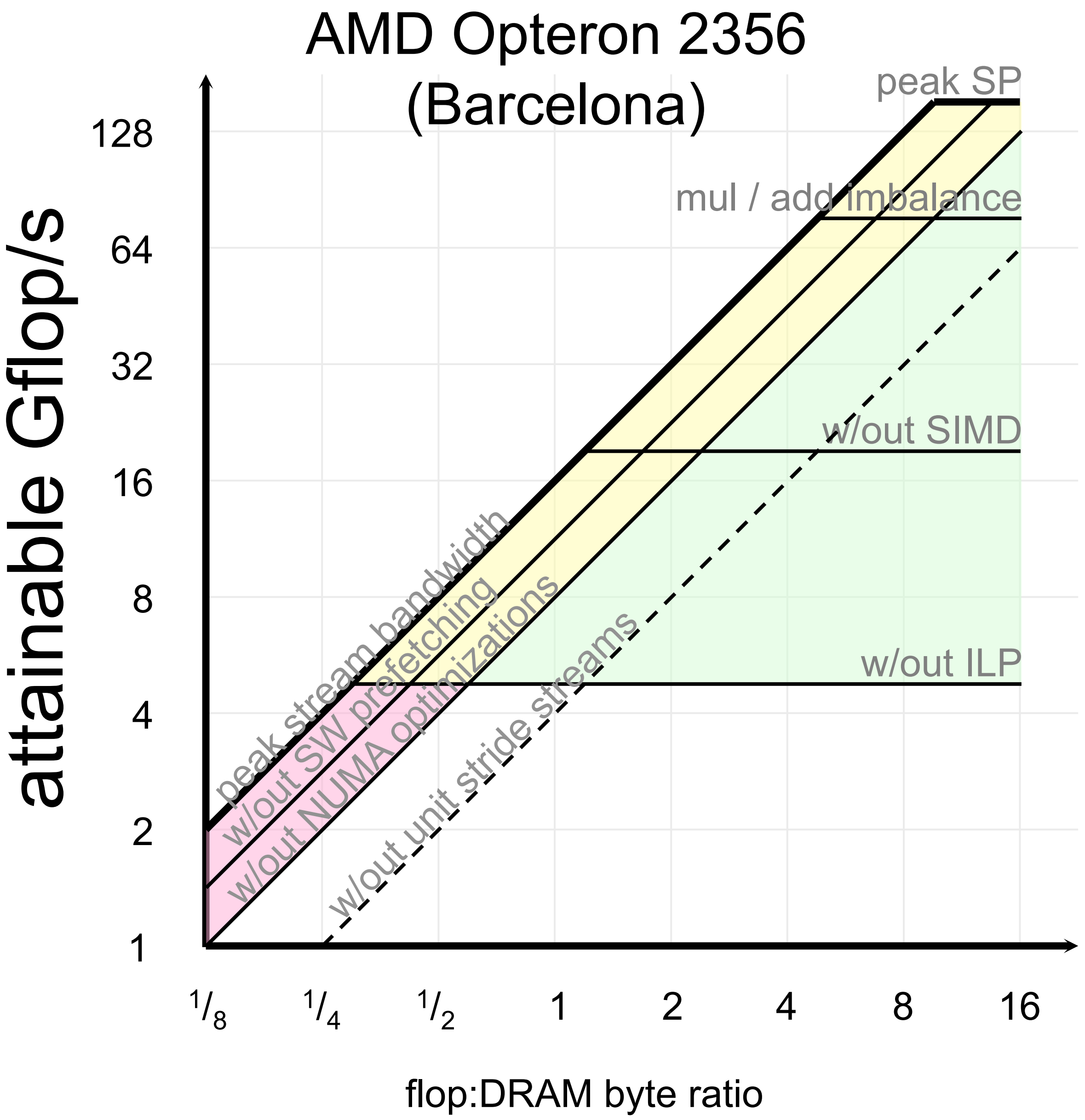
BERKELEY PAR LAB



- Its difficult for any architecture to reach the raw DRAM bandwidth

Roofline model for Opteron

(adding ceilings)

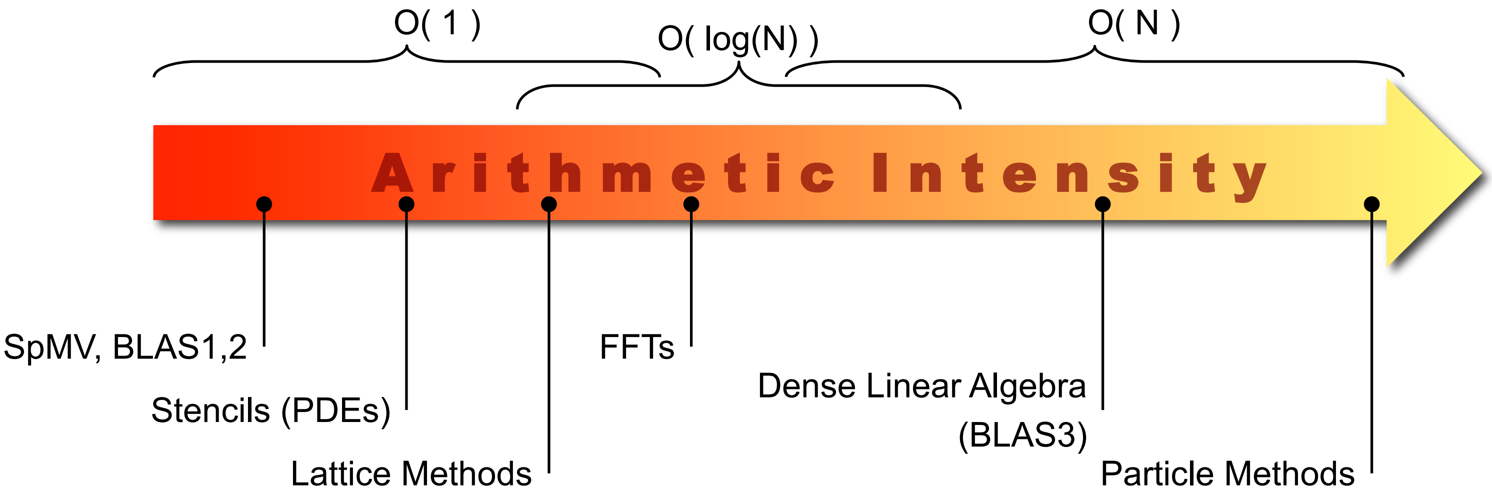


- Partitions the regions of expected performance into three optimization regions:
  - Compute only
  - Memory only
  - Compute+Memory

Uniqueness

- There is no single ordering or roofline model
- The order of ceilings is generally (bottom up):
  - What is inherent in algorithm
  - What a compiler is likely to provide
  - What a programmer could provide
  - What can never be exploited for this kernel
- For example,
  - FMA or mul/add balance is inherent in many linear algebra routines and should be placed at the bottom.
  - However, many stencils are dominated by adds, and thus the multipliers and FMA go underutilized.

Arithmetic Intensity in HPC



- **Arithmetic Intensity (AI) ~ Total Flops / Total DRAM Bytes**
- Some HPC kernels have an arithmetic intensity that's constant, but on others it scales with with problem size (increasing temporal locality)
- Actual arithmetic intensity is capped by cache/local store capacity

Accurately Determining the true Flop:DRAM Byte ratio

- Remember the 3C's of caches

- Calculating the Flop:DRAM byte ratio is:
  - **Compulsory misses**: straightforward
  - **Capacity misses**: pencil and paper (maybe performance counters)
  - **Conflict misses**: must use performance counters

- Flop:actual DRAM Byte ratio < Flop:compulsory DRAM Byte ratio

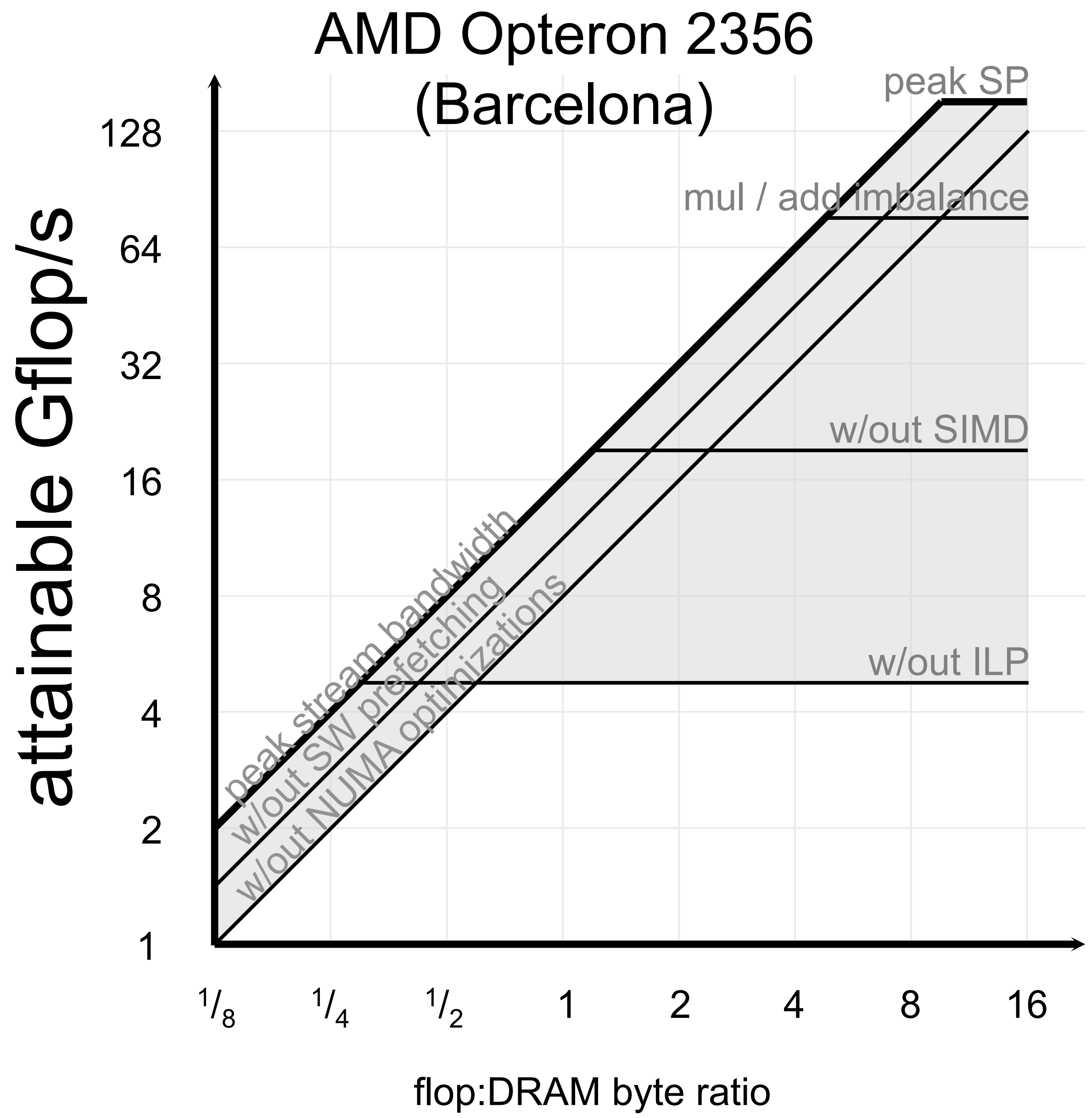
- One might place a range on the arithmetic intensity ratio
- Thus performance is limited to an area between the ceilings and between the upper (compulsory) and lower bounds on arithmetic intensity

EECS Electrical Engineering and Computer Sciences

Roofline model for Opteron

(powerpoint doodle)

BERKELEY PAR LAB



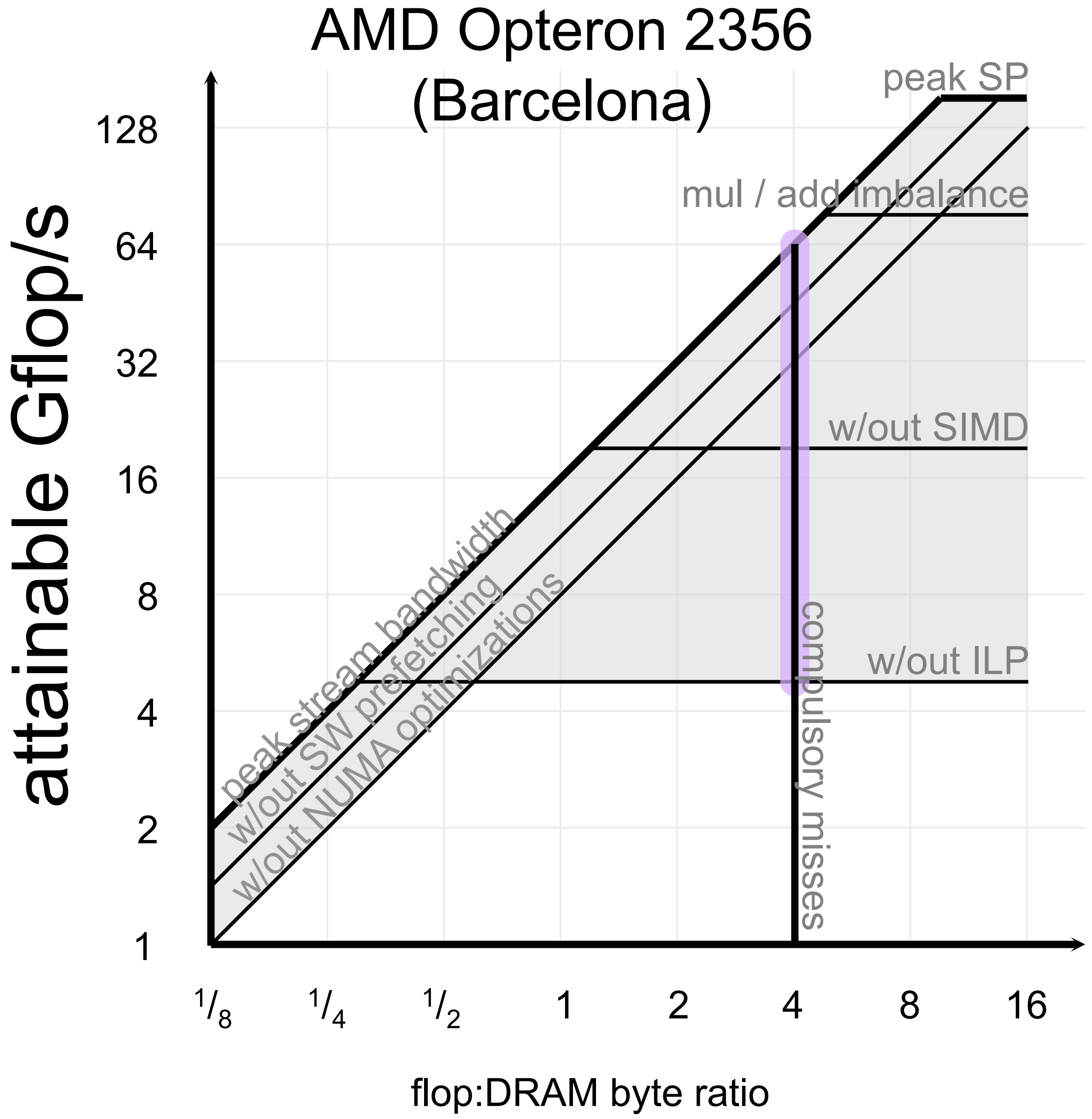
- Final Roofline

EECS
Electrical Engineering and Computer Sciences

Roofline model for Opteron

(powerpoint doodle)

BERKELEY PAR LAB



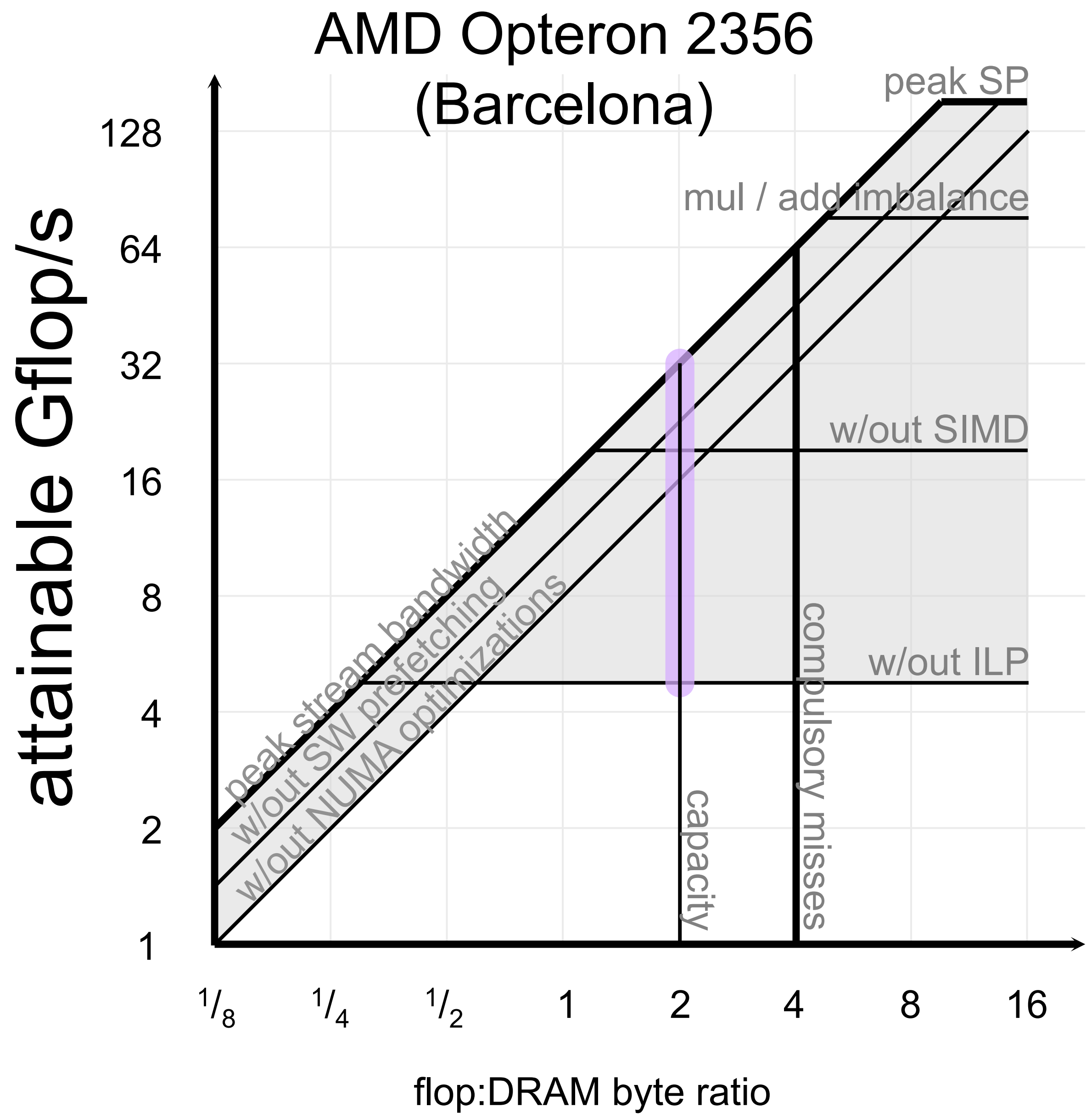
- Some arbitrary kernel has a flop:compulsory byte ratio of 4
- Overlaid on the roofline
- Defines upper bound on range of expected performance
- Also shows which optimizations are likely

EECS
Electrical Engineering and Computer Sciences

Roofline model for Opteron

(powerpoint doodle)

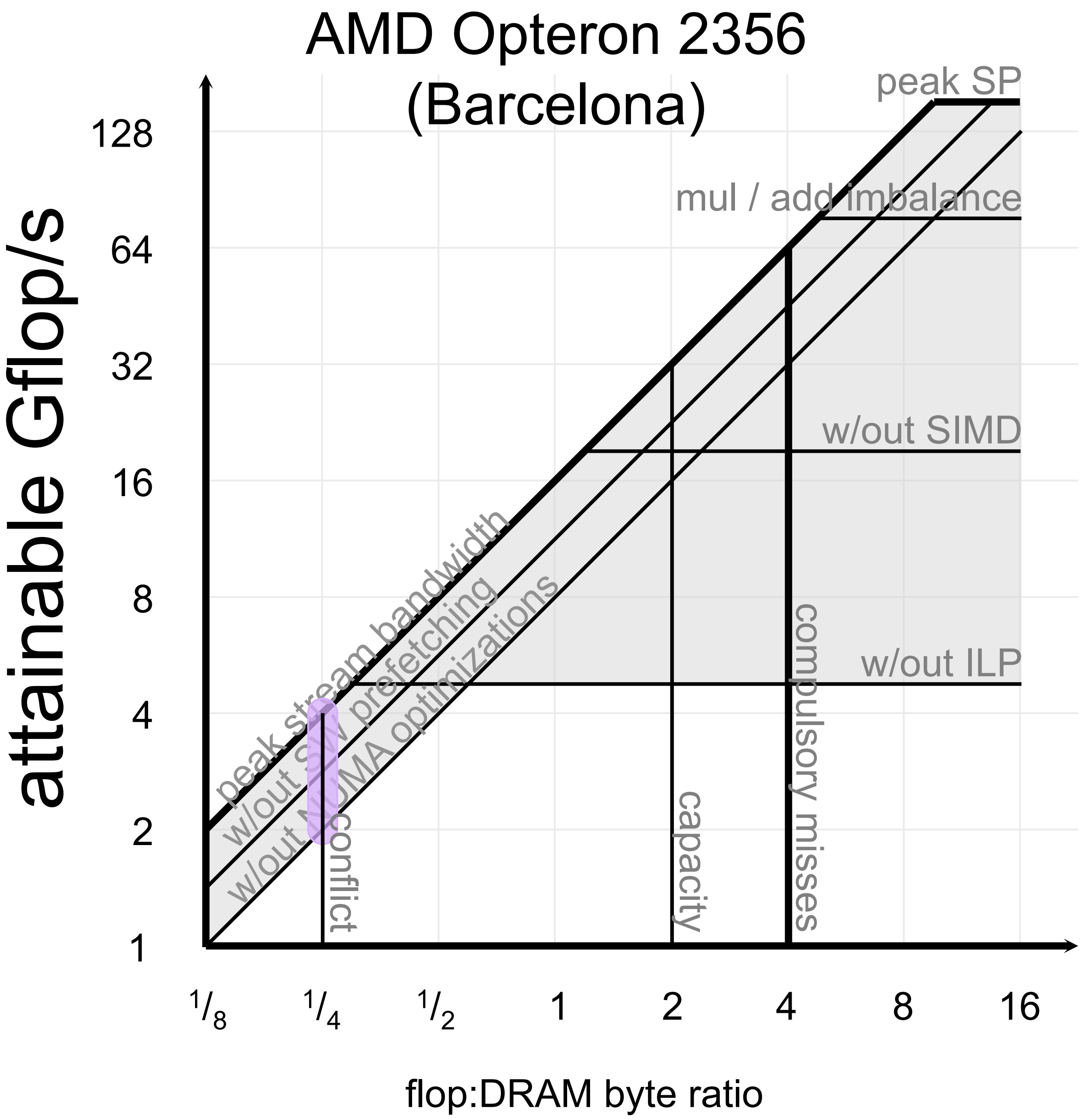
BERKELEY PAR LAB



- Capacity misses reduce the actual flop:byte ratio
- Also reduces attainable performance
- **AI is unique to each combination of kernel and architecture**

Roofline model for Opteron

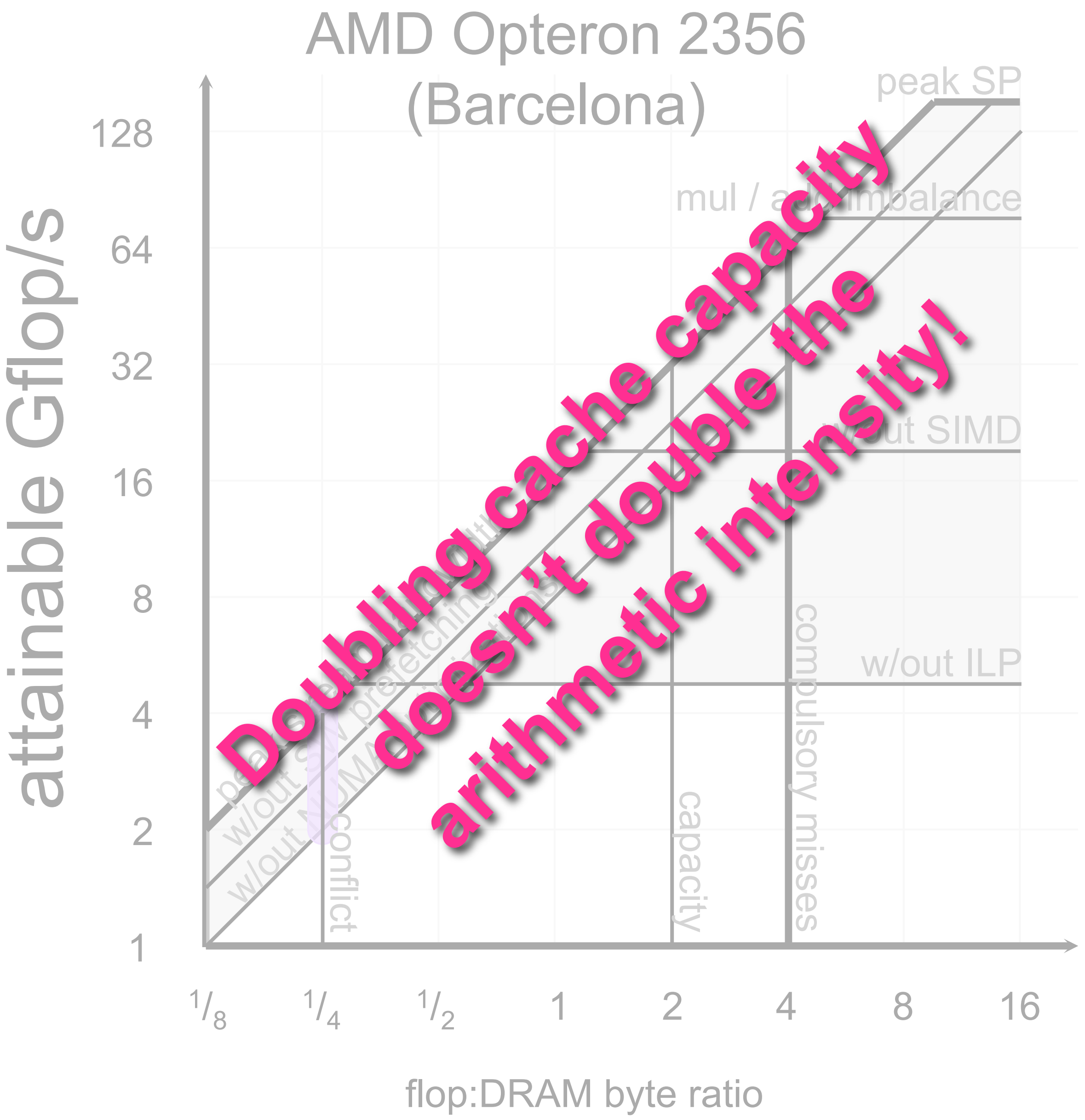
(powerpoint doodle)



- Conflict misses may destroy performance
- **AI is unique to each combination of kernel and architecture**

Roofline model for Opteron

(powerpoint doodle)

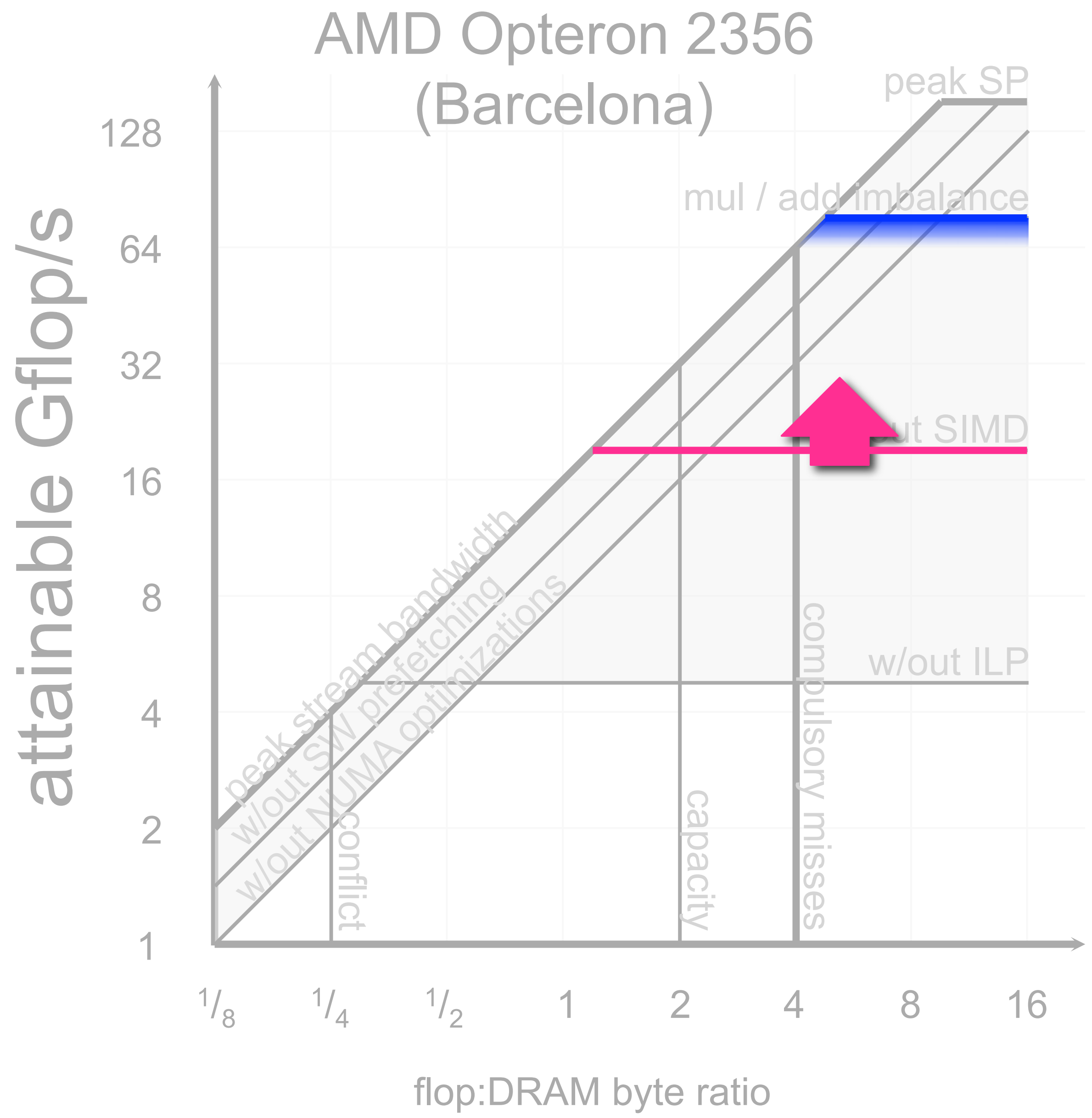


Doubling cache capacity doesn't double the arithmetic intensity!

- Conflict misses may destroy performance

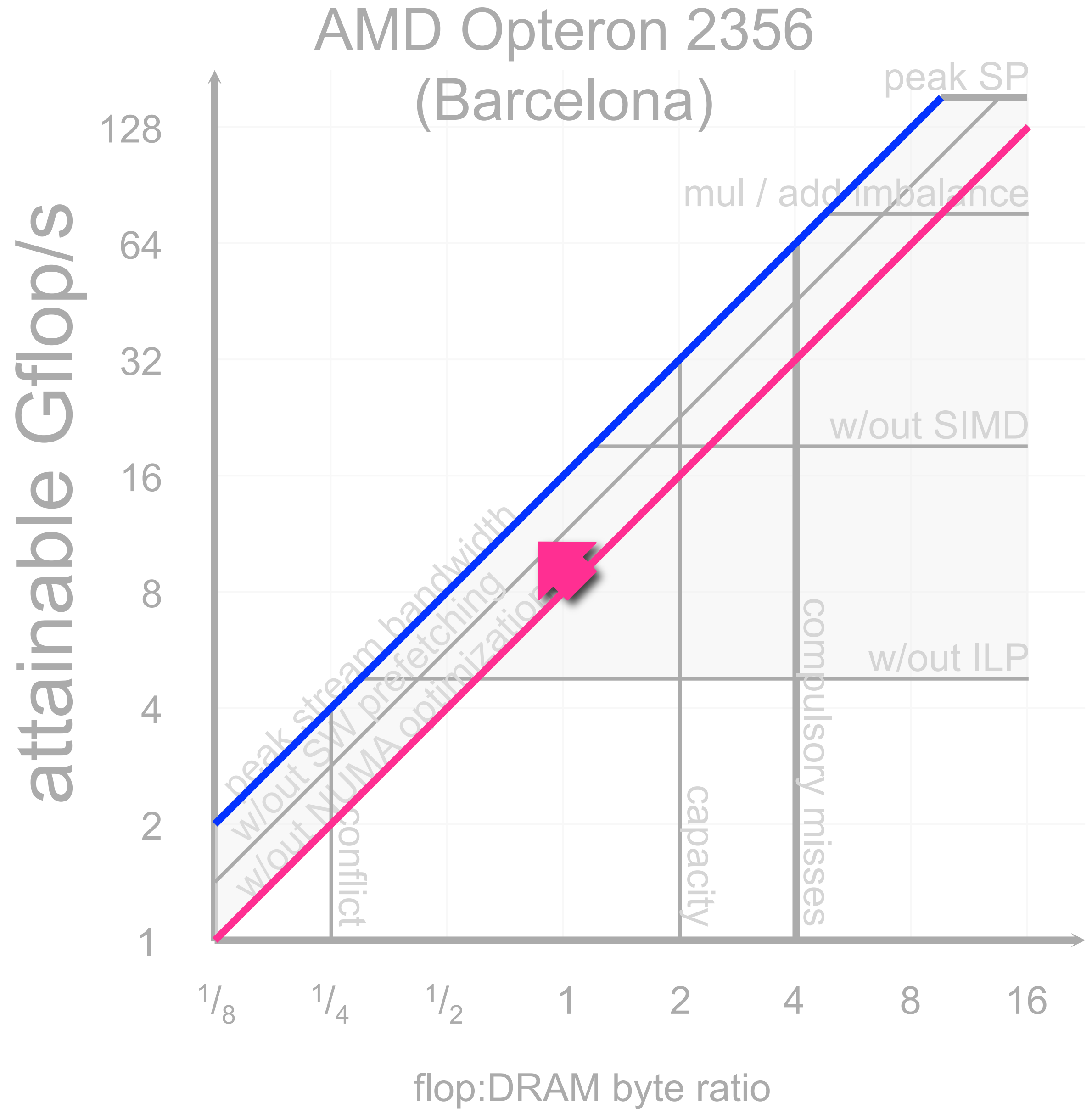
Three Categories of Software Optimization

Maximizing Attained in-core Performance



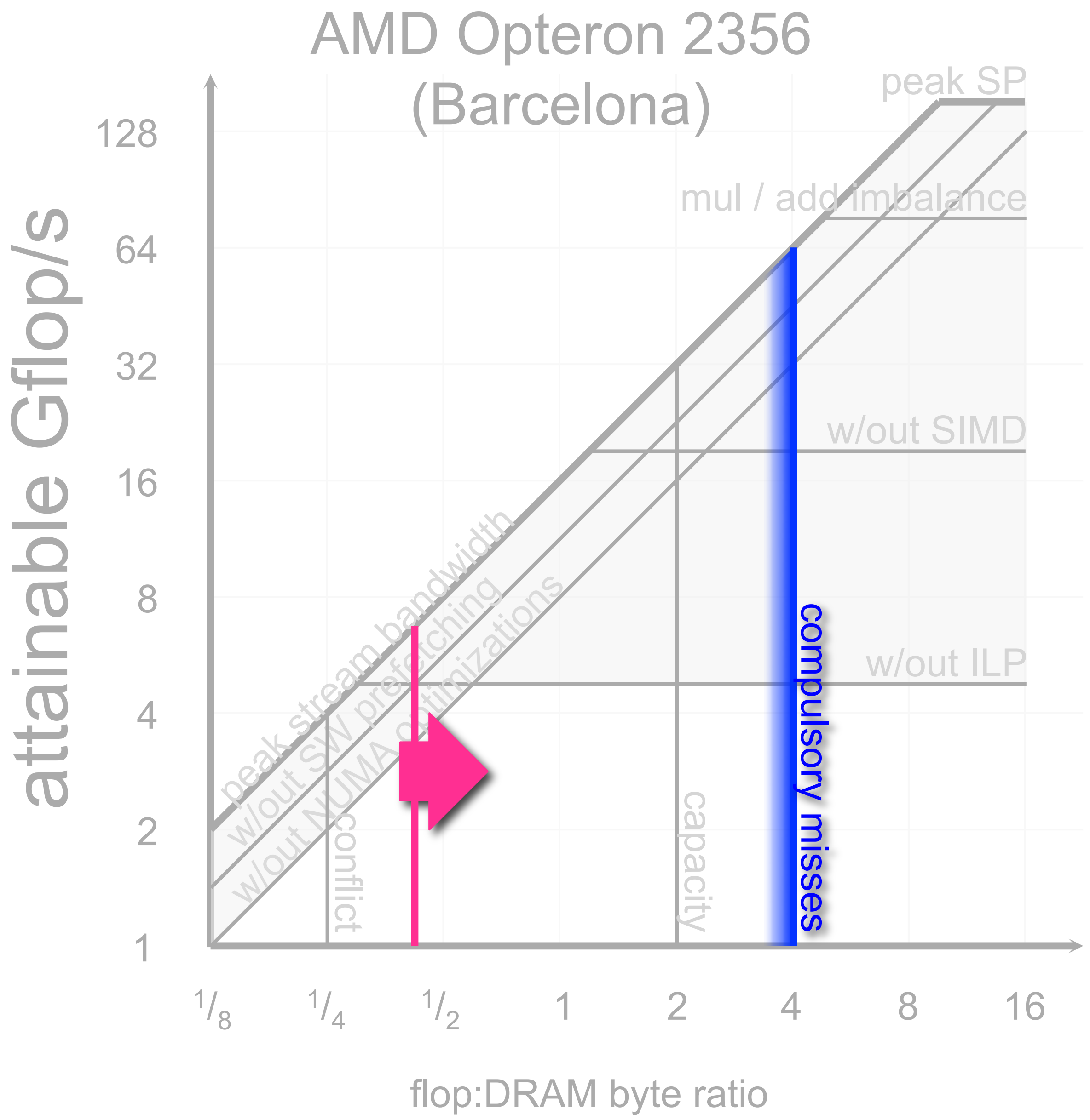
- Software optimizations such as explicit SIMDization can punch through the horizontal ceilings (what can be expected from a compiler)
- Other examples include loop unrolling, reordering, and long running loops

Maximizing Attained Memory Bandwidth



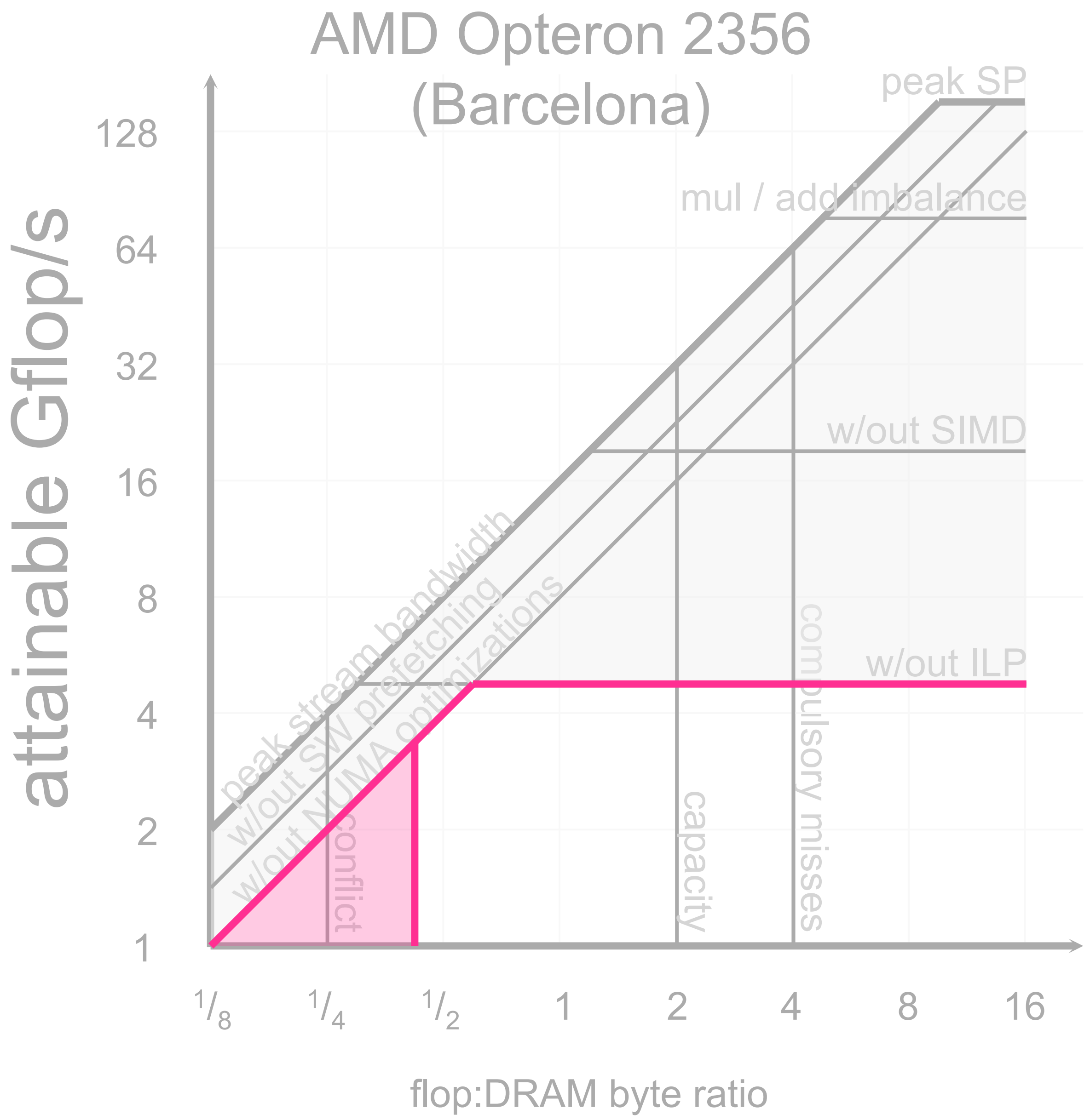
- Compilers won't give great out-of-the box bandwidth
- Punch through bandwidth ceilings:
  - Maximize MLP
  - long unit stride accesses
  - NUMA aware allocation and parallelization
  - SW prefetching

Minimizing Memory Traffic



- Use performance counters to measure flop:byte ratio (AI)
- Out-of-the-box code may have an AI ratio much less than the compulsory ratio
  - Be cognizant of cache capacities, associativities, and threads sharing it
  - Pad structures to avoid conflict misses
  - Use cache blocking to avoid capacity misses
- **These optimizations can be imperative**

Effective Roofline (before)

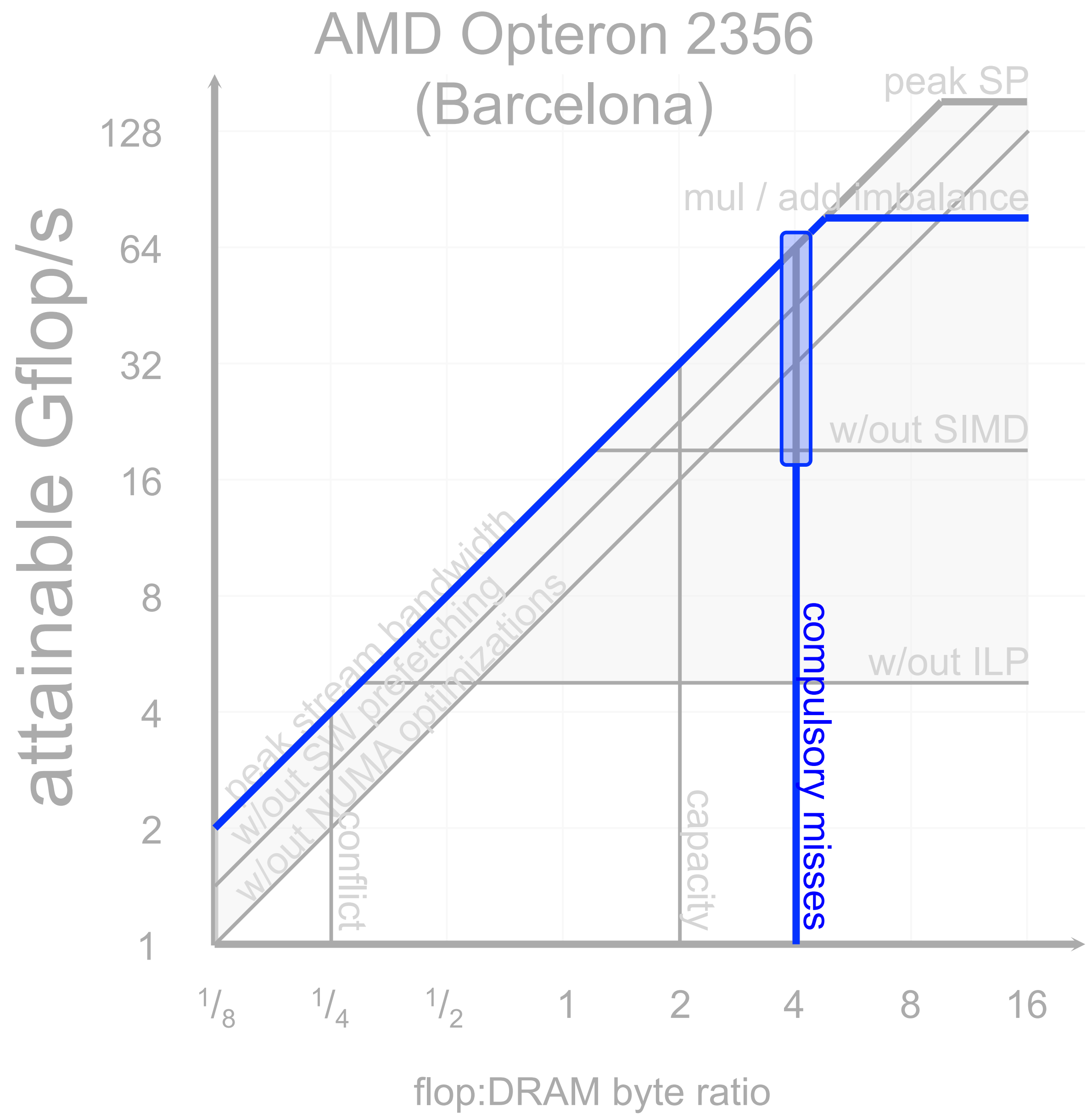


- Before optimization, traffic, and limited bandwidth optimization limits performance to a very narrow window

EECS
Electrical Engineering and
Computer Sciences

Effective Roofline (after)

BERKELEY PAR LAB

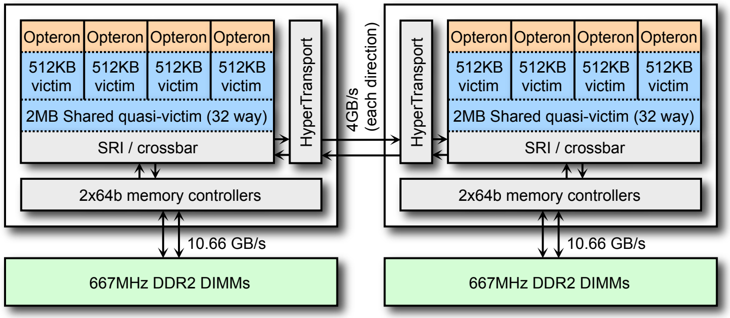


- After optimization, ideally, performance is significantly better

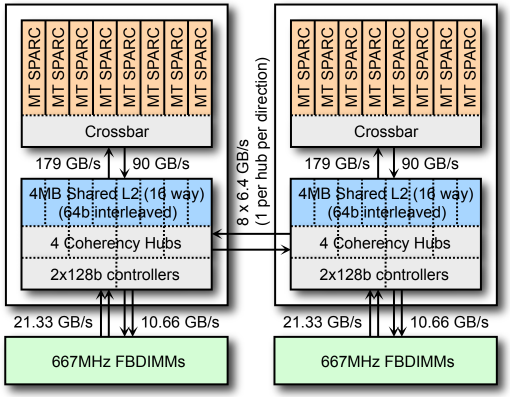
Applicable to Other Architectural Paradigms ?

Four Architectures

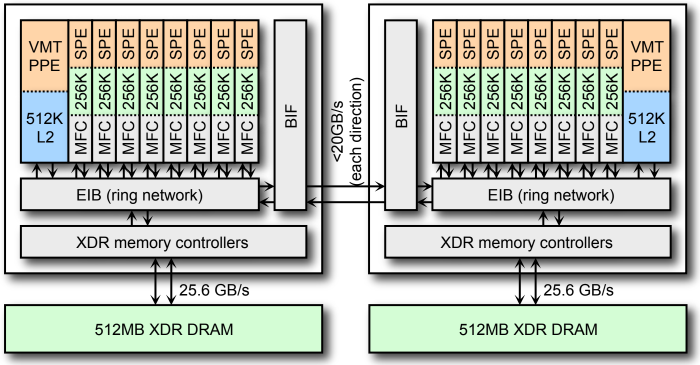
AMD Barcelona



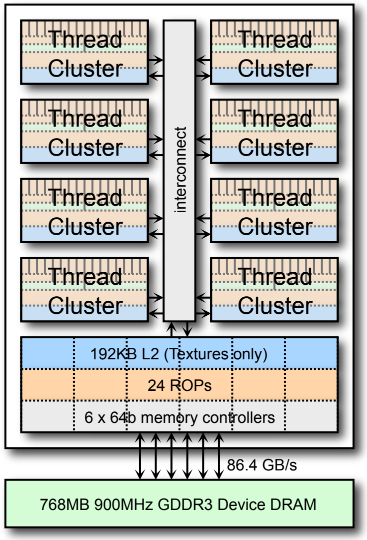
Sun Victoria Falls



IBM Cell Blade

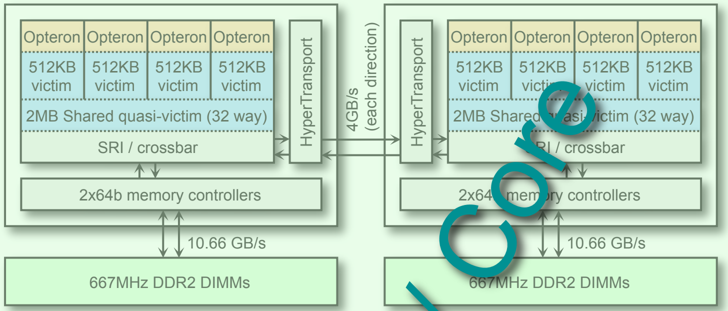


NVIDIA G80

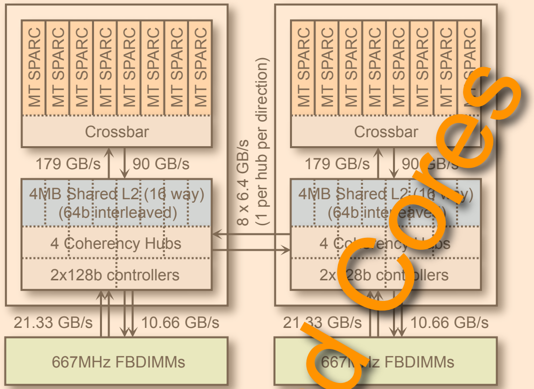


Four Architectures

AMD Barcelona



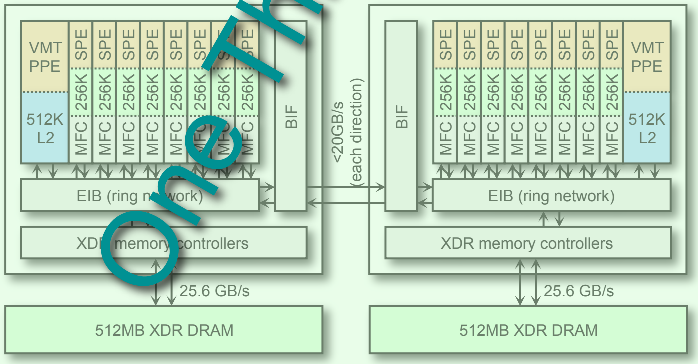
Sun Victoria Falls



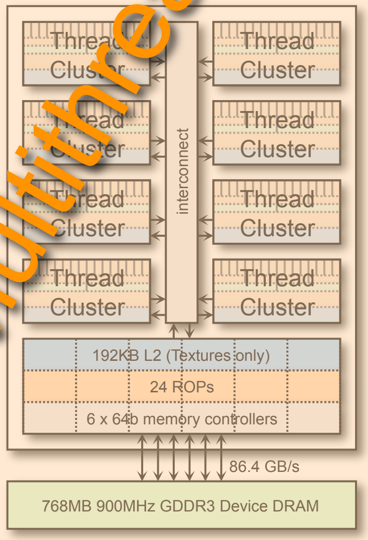
One Thread / Core

Multithreaded Cores

IBM Cell Blade



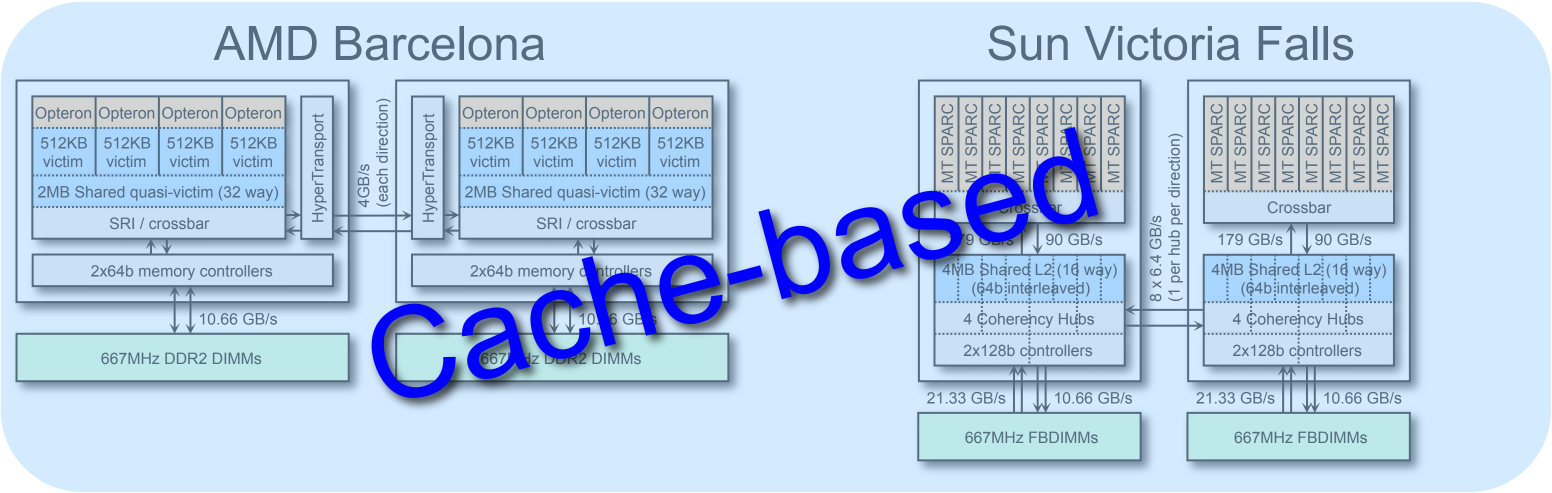
NVIDIA G80



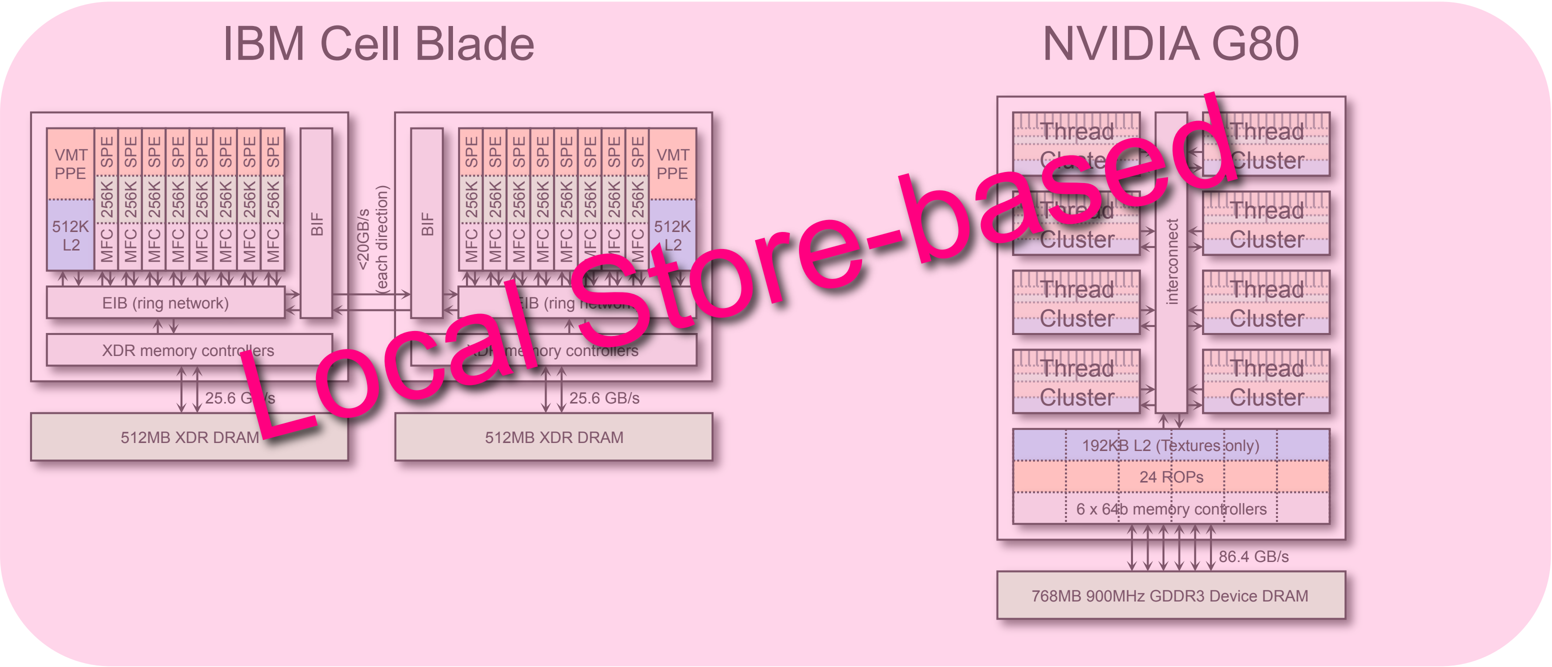
EECS
Electrical Engineering and Computer Sciences

Four Architectures

BERKELEY PAR LAB



Cache-based



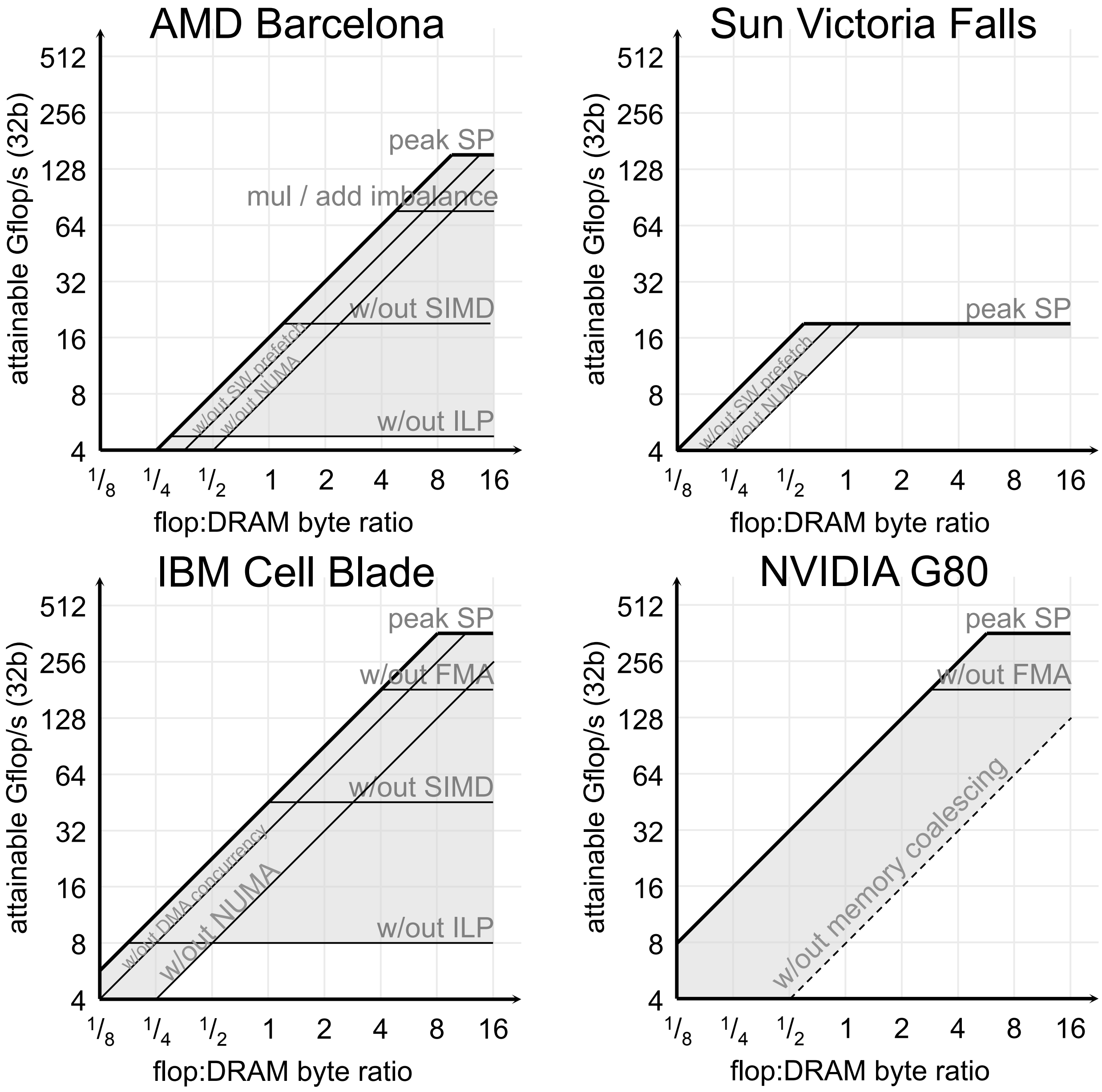
Local Store-based

EECS
Electrical Engineering and Computer Sciences

32b Rooflines for the Four

(in-core parallelism)

BERKELEY PAR LAB



- Single Precision Roofline models for the SMPs used in this work.
- Based on micro-benchmarks, experience, and manuals
- Ceilings =

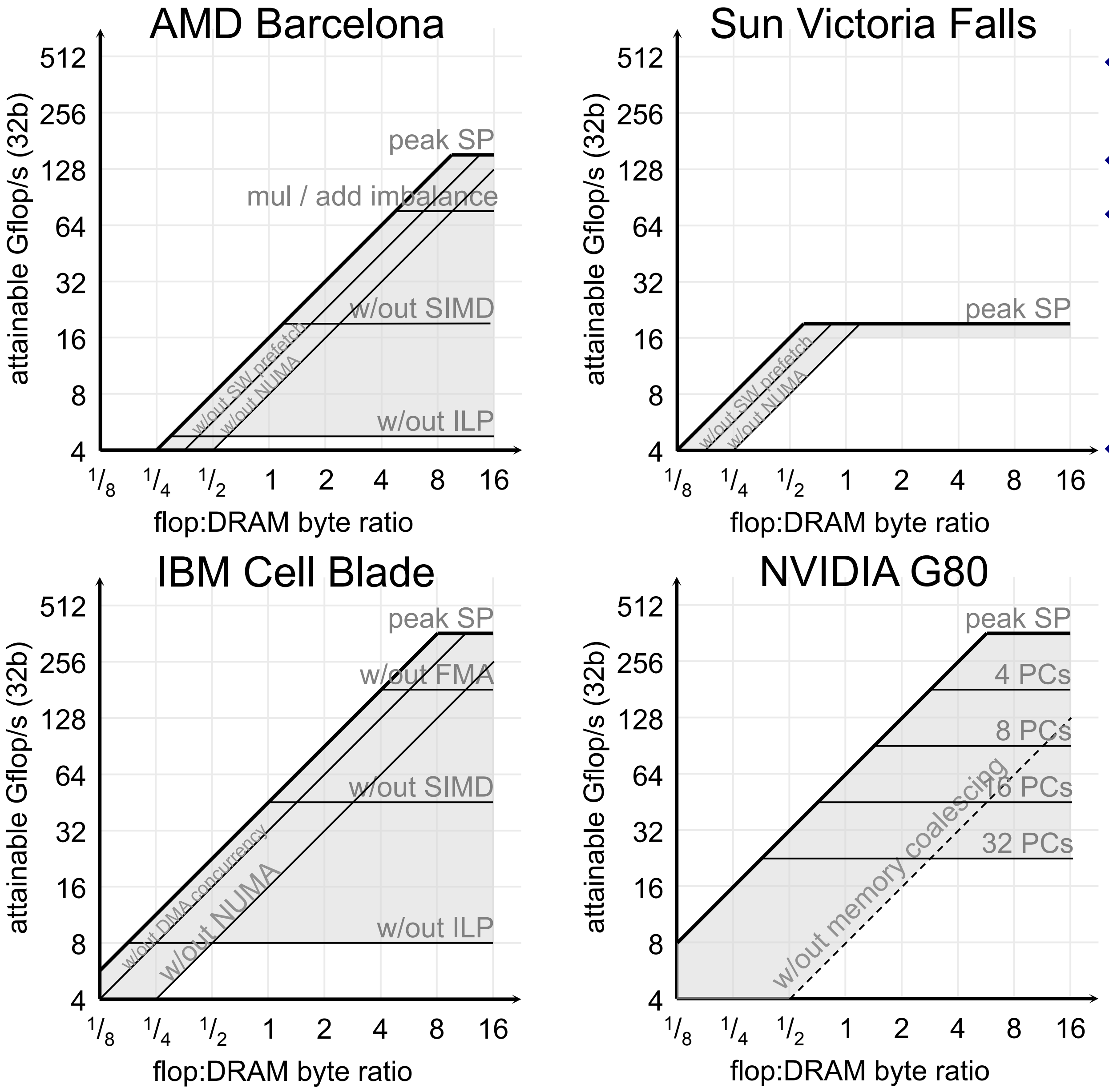
  in-core parallelism
- **Can the compiler find all this parallelism ?**
- NOTE:
  - log-log scale
  - Assumes perfect SPMD

Electrical Engineering and Computer Sciences

32b Rooflines for the Four

(diverged threads)

BERKELEY PAR LAB

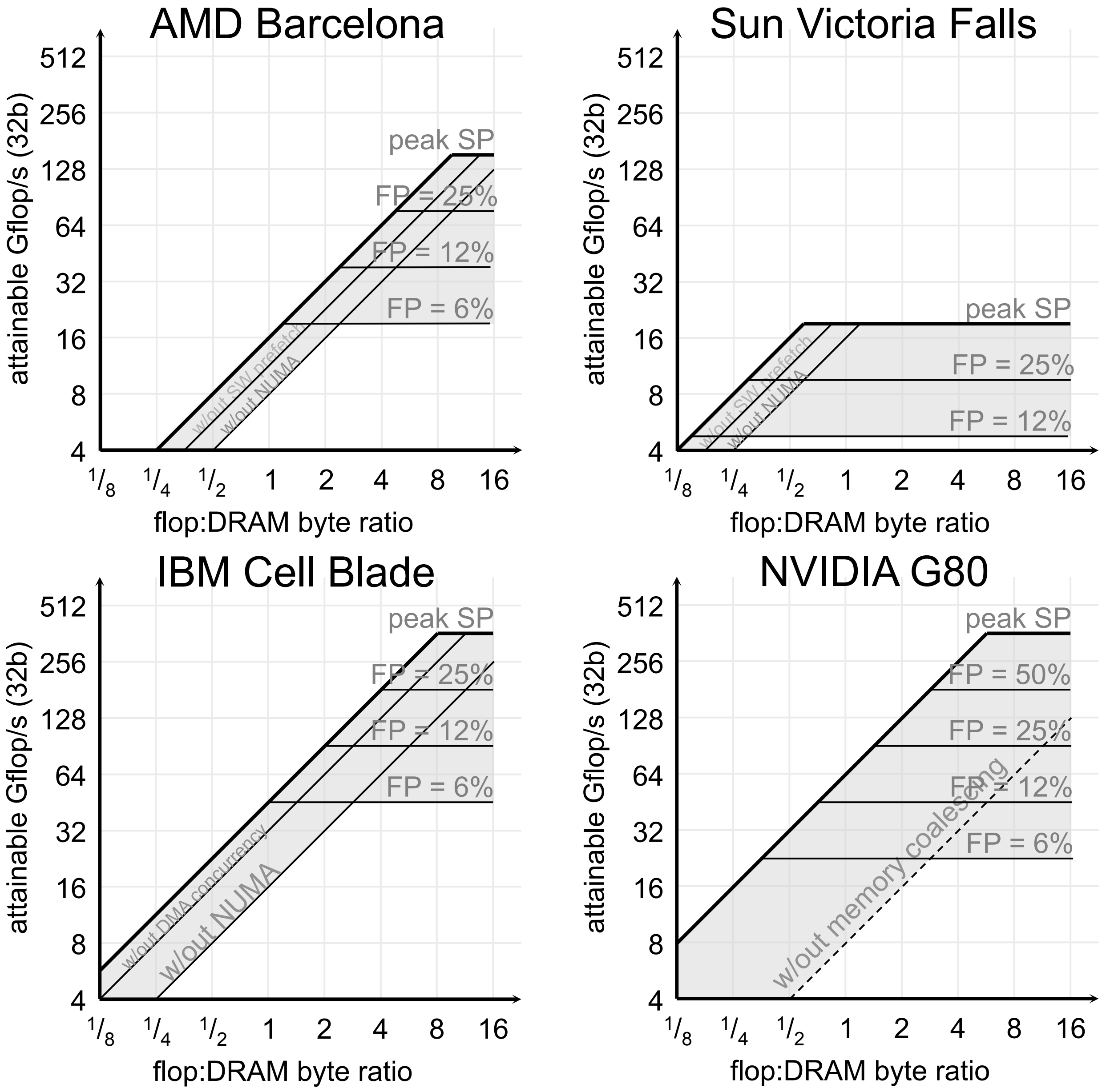


- G80 dynamically finds DLP (shared instruction fetch)
- **SIMT**
- If threads of a warp diverge from SIMD execution, performance is limited by instruction issue bandwidth
- Ceilings on G80 =

  number of unique PCs when threads diverge

32b Rooflines for the Four

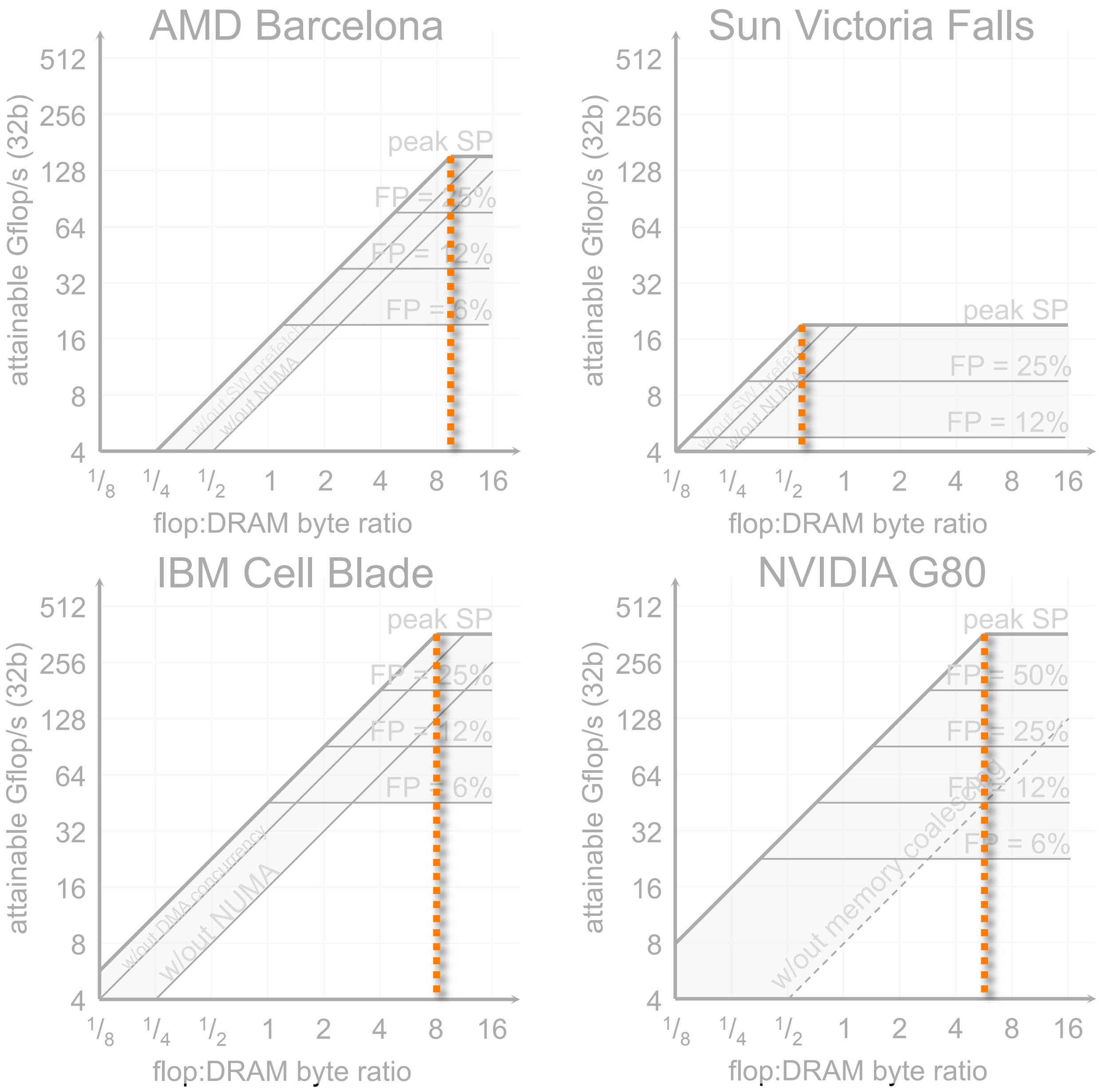
(FP fraction of dynamic instructions)



- Some kernels have large numbers of non FP instructions
- Saps instruction issue bandwidth
- Ceilings = FP fraction of dynamic instruction mix
- NOTE:
  - Assumes perfect in-core parallelism

32b Rooflines for the Four

(ridge point)



- ❖ Some architectures have drastically different ridge points
- ❖ VF may be compute bound on many kernels
- ❖ Clovertown has $^{1}/_{3}$ the BW of Barcelona = ridge point to the right

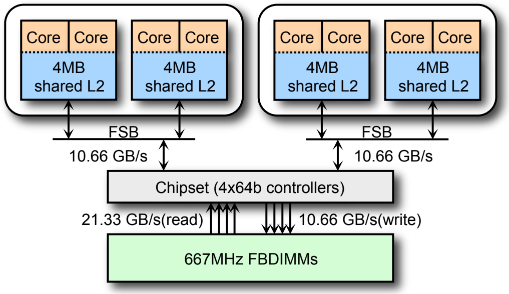
Using Roofline when Auto-tuning HPC Kernels

EECS
Electrical Engineering and
Computer Sciences

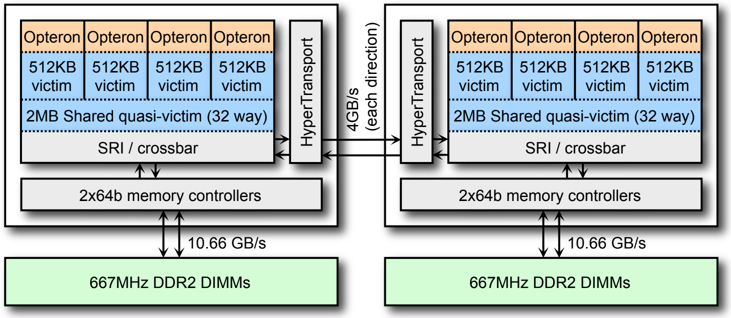
Multicore SMPs Used

BERKELEY PAR LAB

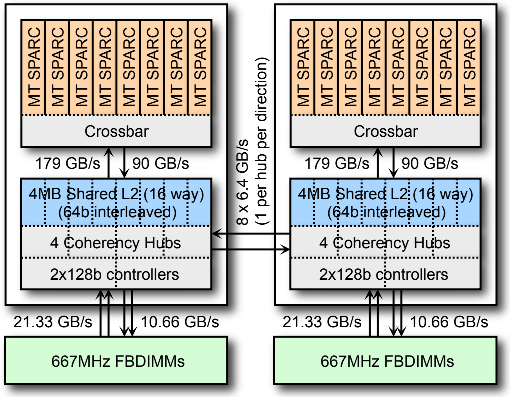
Intel Xeon E5345 (Clovertown)



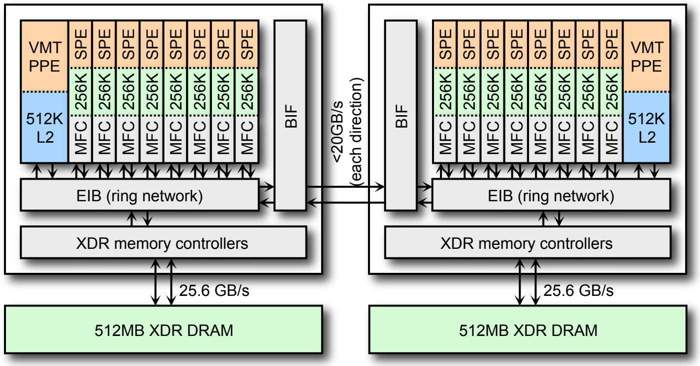
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

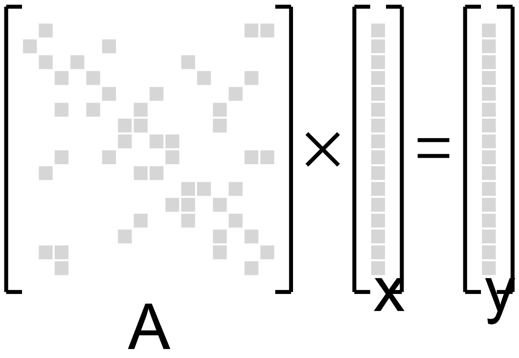


Sparse Matrix-Vector Multiplication (SpMV)

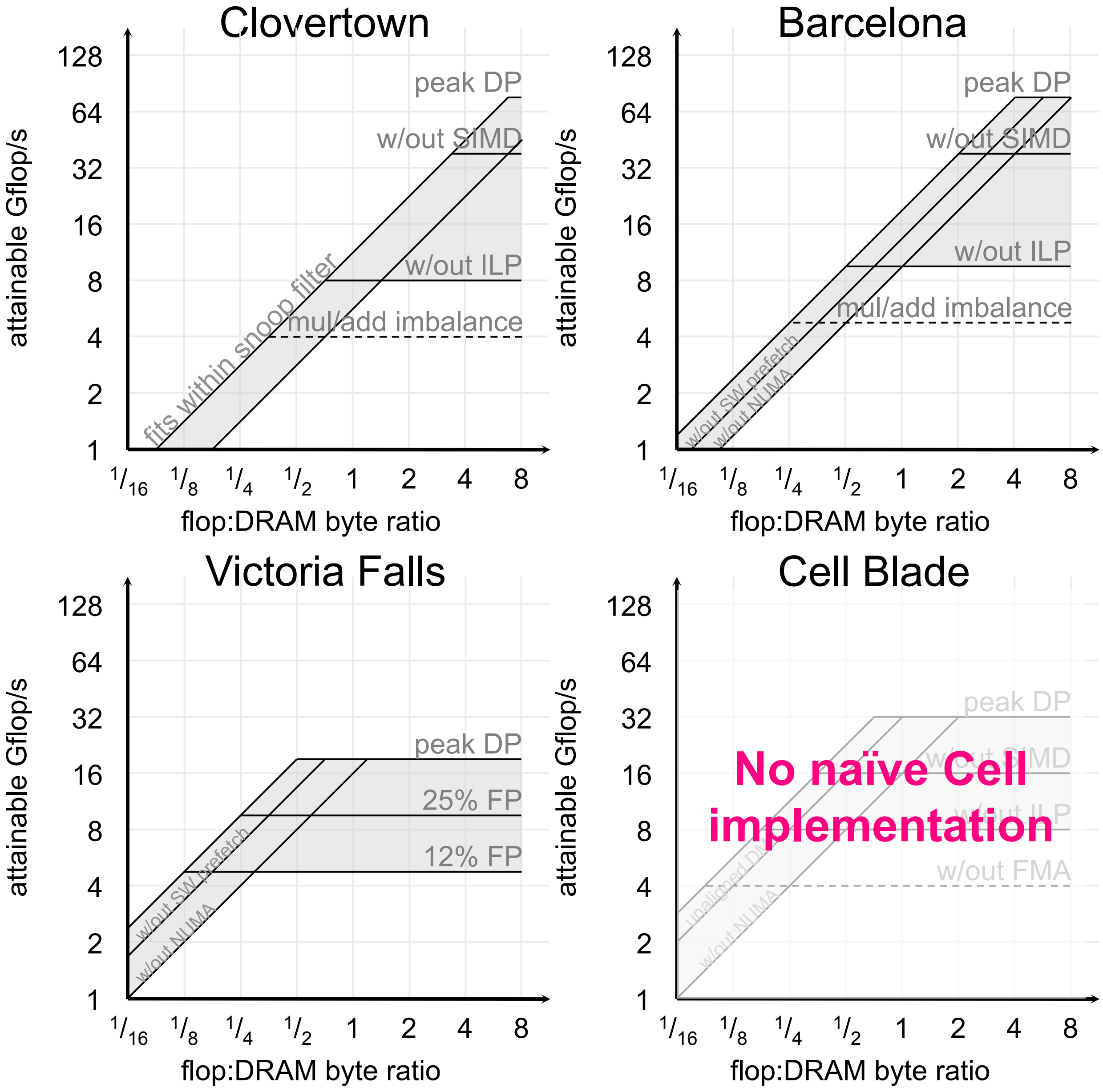
Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, James Demmel, "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms", Supercomputing (SC), 2007.

Sparse Matrix Vector Multiplication

- Sparse Matrix
  - Most entries are 0.0
  - Performance advantage in only storing/operating on the nonzeros
  - Requires significant meta data
- Evaluate y=Ax
  - A is a sparse matrix
  - x & y are dense vectors
- Challenges
  - Difficult to exploit ILP(bad for superscalar),
  - Difficult to exploit DLP(bad for SIMD)
  - Irregular memory access to source vector
  - Difficult to load balance
  - **Very low arithmetic intensity (often <0.166 flops/byte) = likely memory bound**



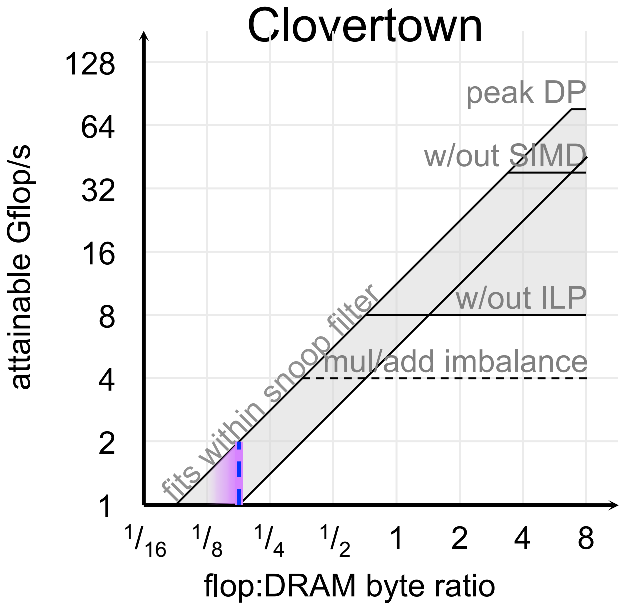
Roofline model for SpMV

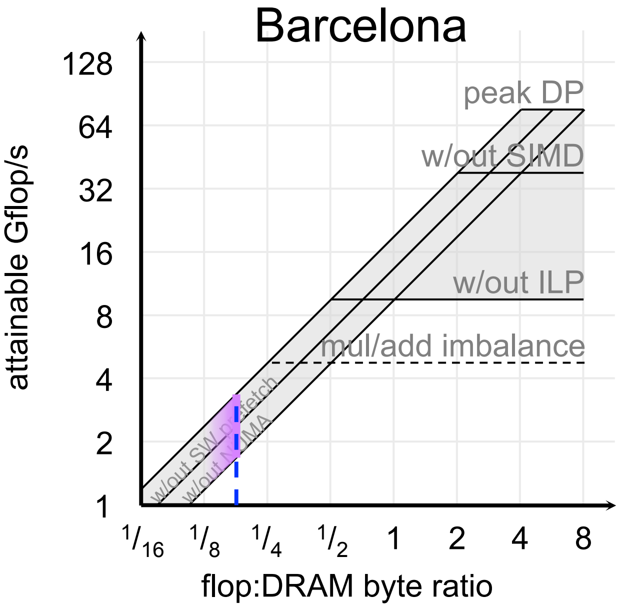


No naïve Cell implementation

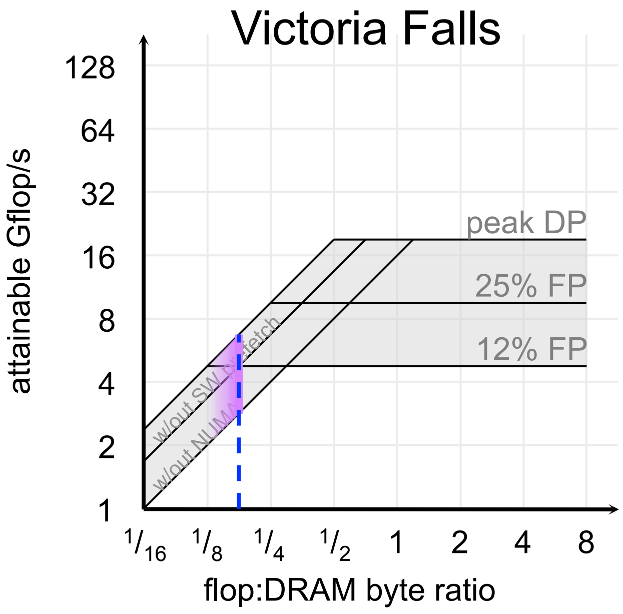
- Double precision roofline models
- FMA is inherent in SpMV (place at bottom)

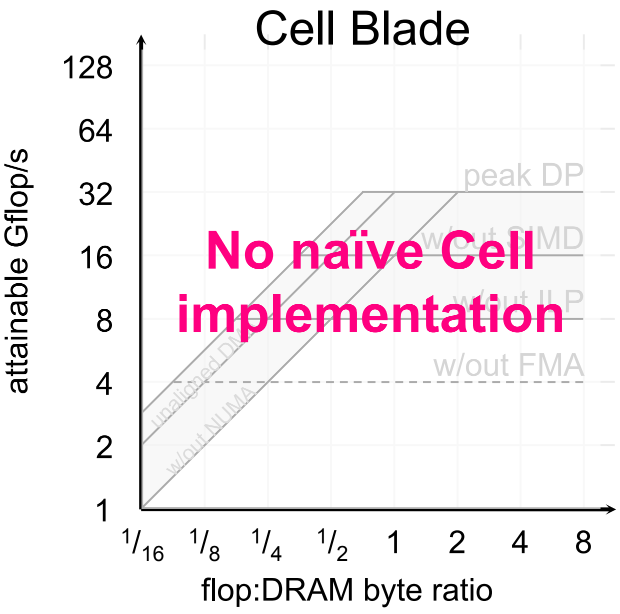
Roofline model for SpMV


Clovertown


Barcelona

- ❖ Two unit stride streams
- ❖ Inherent FMA
- ❖ No ILP
- ❖ No DLP
- ❖ FP is 12-25%
- ❖ Naïve compulsory flop:byte < 0.166

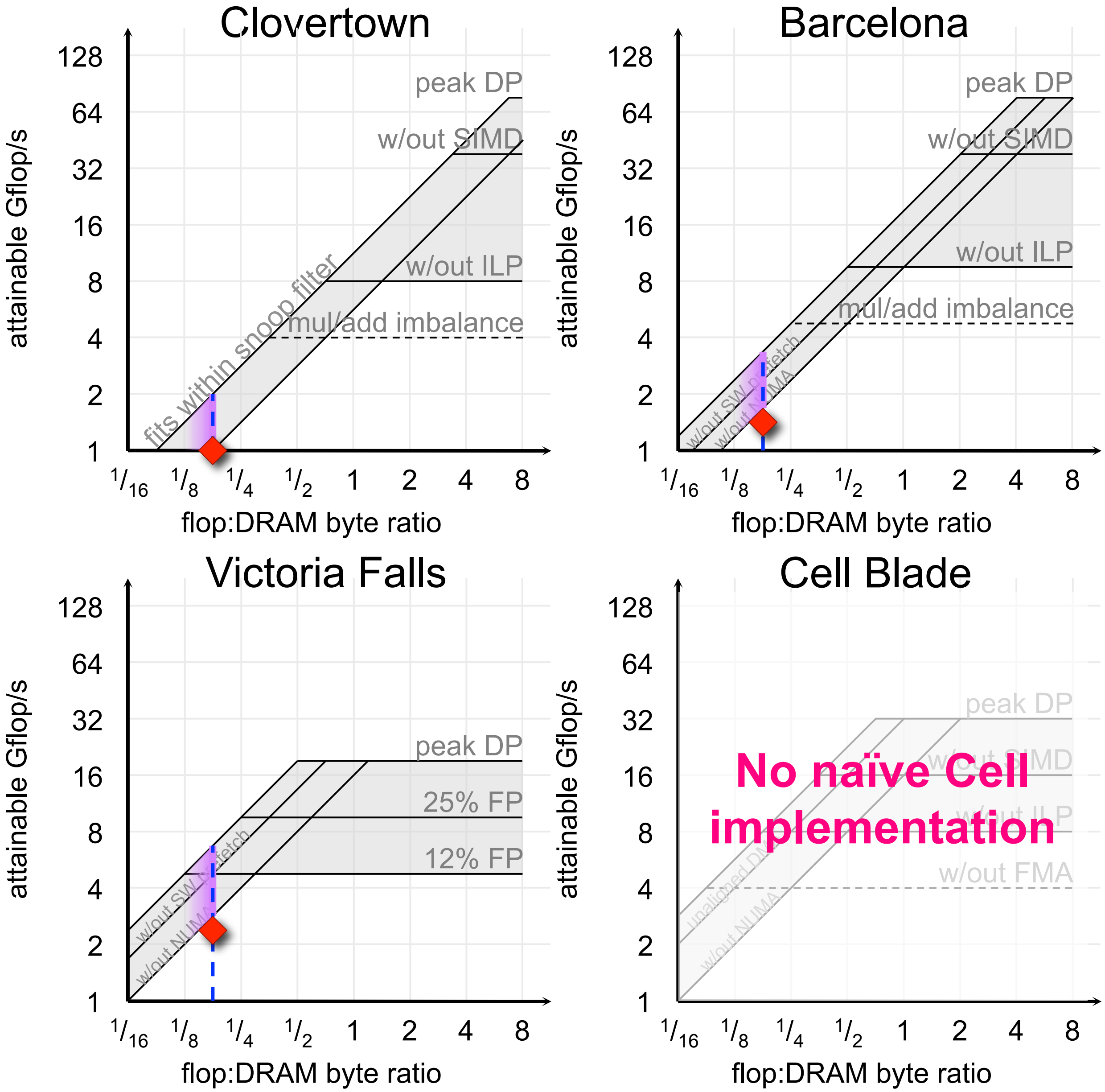

Victoria Falls


Cell Blade

No naïve Cell implementation

Roofline model for SpMV

(out-of-the-box parallel)

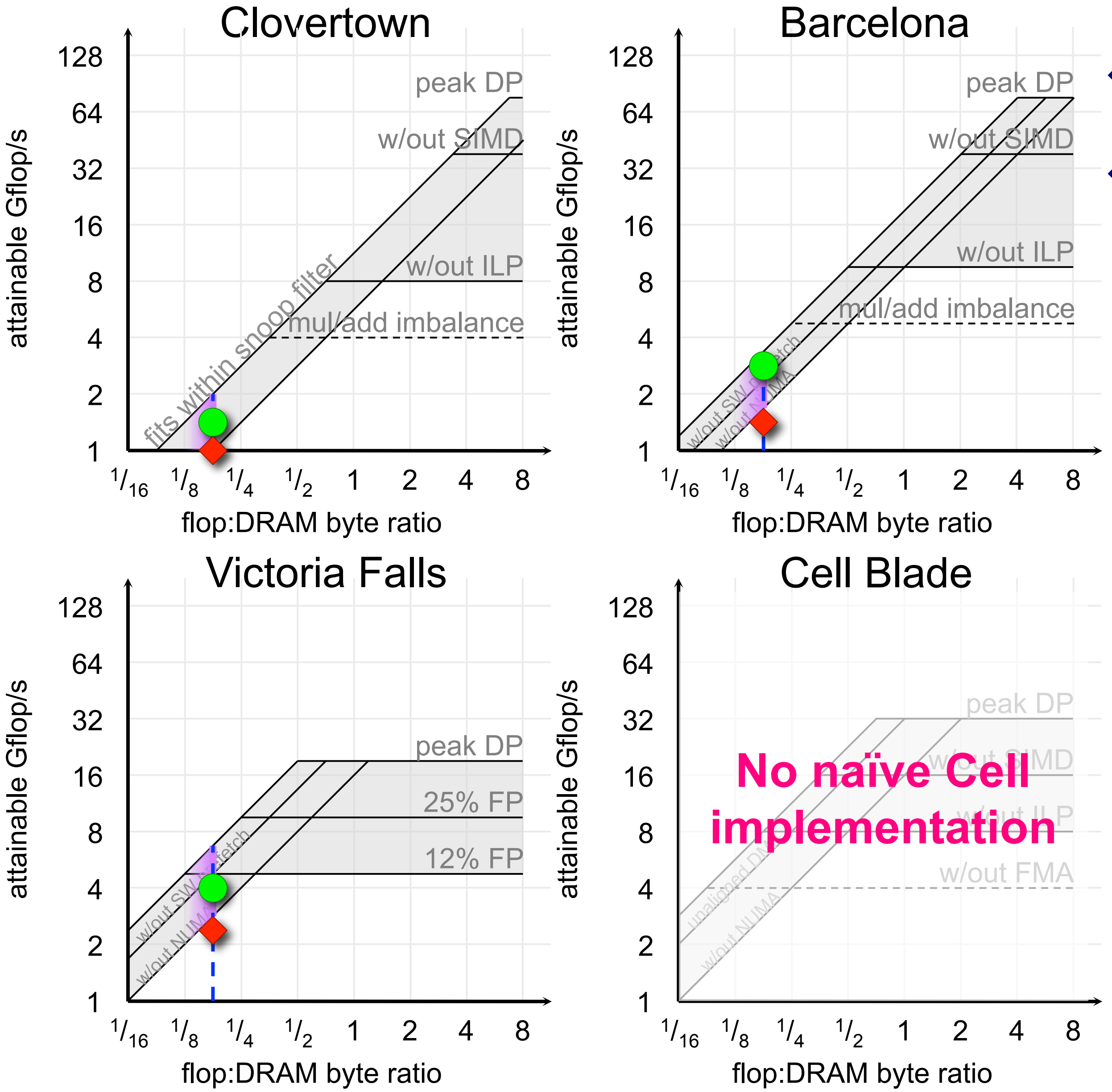


No naïve Cell implementation

- Two unit stride streams
- Inherent FMA
- No ILP
- No DLP
- FP is 12-25%
- Naïve compulsory flop:byte < 0.166
- For simplicity: dense matrix in sparse format

Roofline model for SpMV

(NUMA & SW prefetch)

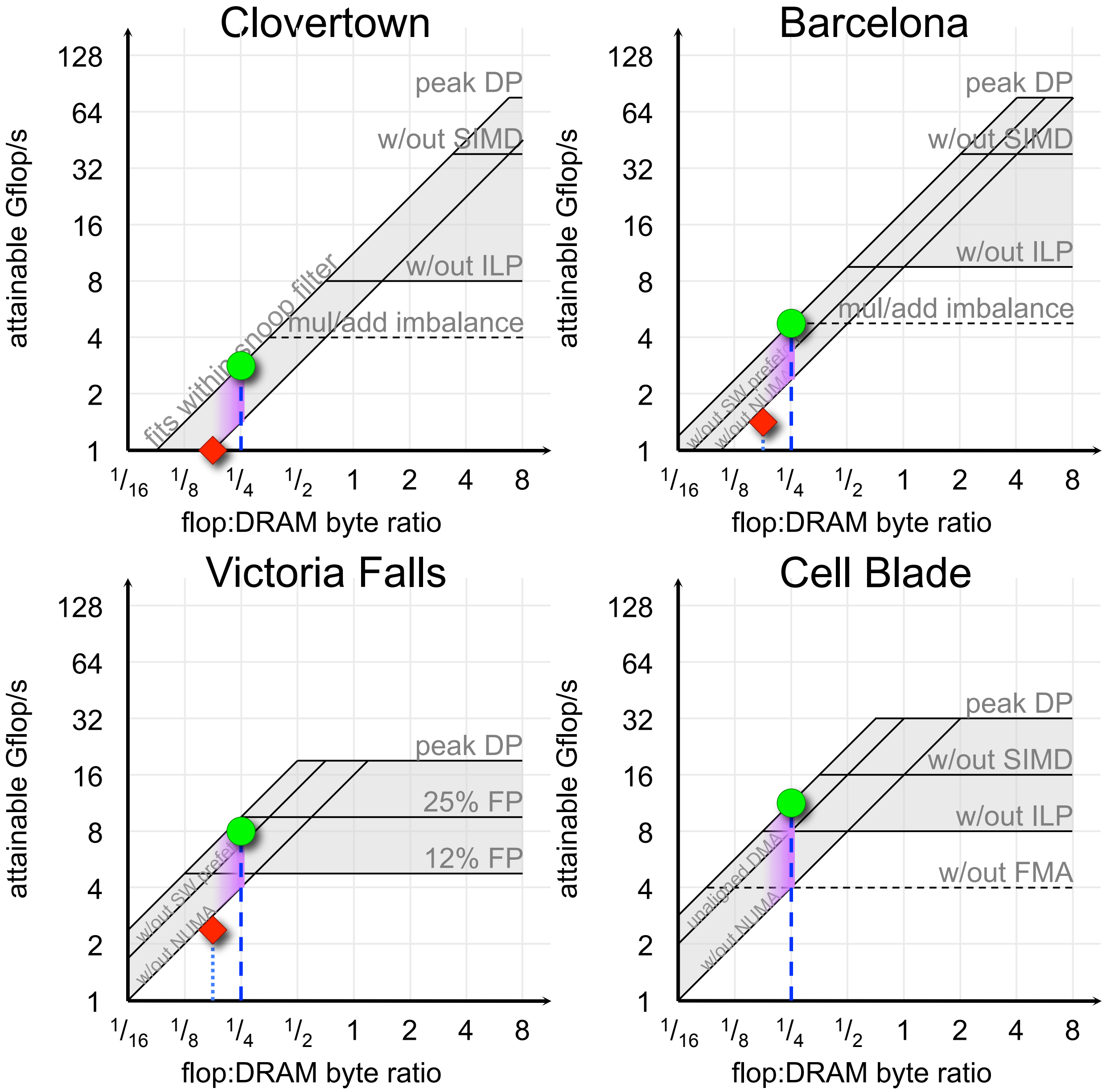


No naïve Cell implementation

- compulsory flop:byte ~ 0.166
- utilize all memory channels

Roofline model for SpMV

(matrix compression)



- Inherent FMA
- Register blocking improves ILP, DLP, flop:byte ratio, and FP% of instructions

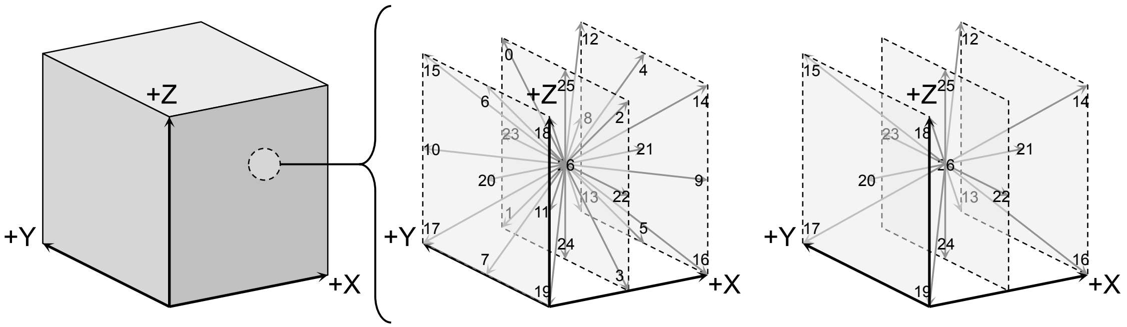
Lattice-Boltzmann Magneto-Hydrodynamics (LBMHD)

Samuel Williams, Jonathan Carter, Leonid Oliker, John Shalf, Katherine Yelick, "Lattice Boltzmann Simulation Optimization on Leading Multicore Platforms", International Parallel & Distributed Processing Symposium (IPDPS), 2008.

Best Paper, Application Track

LBMHD

- Plasma turbulence simulation via Lattice Boltzmann Method
- Two distributions:
  - momentum distribution (27 scalar components)
  - magnetic distribution (15 vector components)
- Three macroscopic quantities:
  - Density
  - Momentum (vector)
  - Magnetic Field (vector)
- Must read 73 doubles, and update 79 doubles per point in space
- Requires about 1300 floating point operations per point in space
- Just over 1.0 flops/byte (ideal)

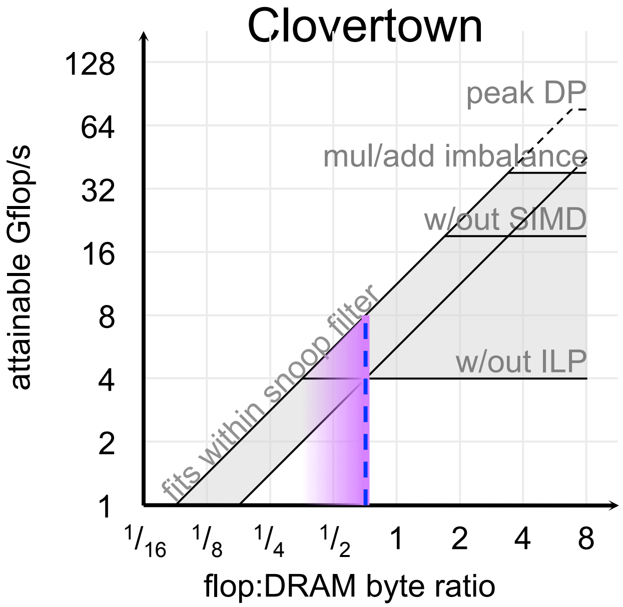


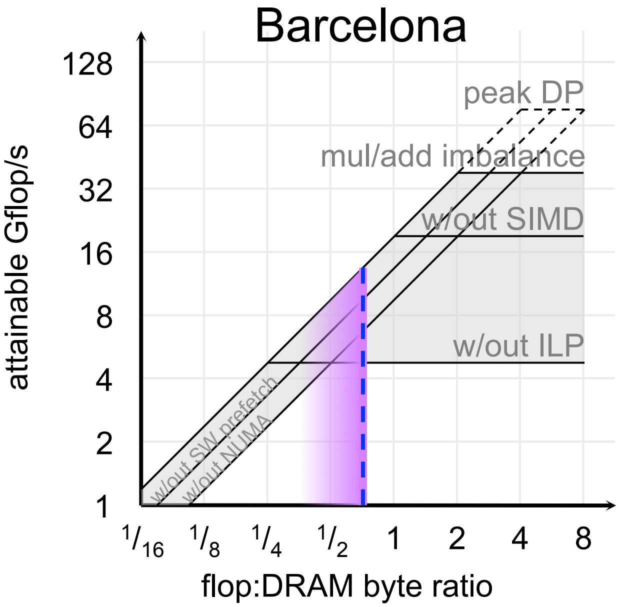
macroscopic variables

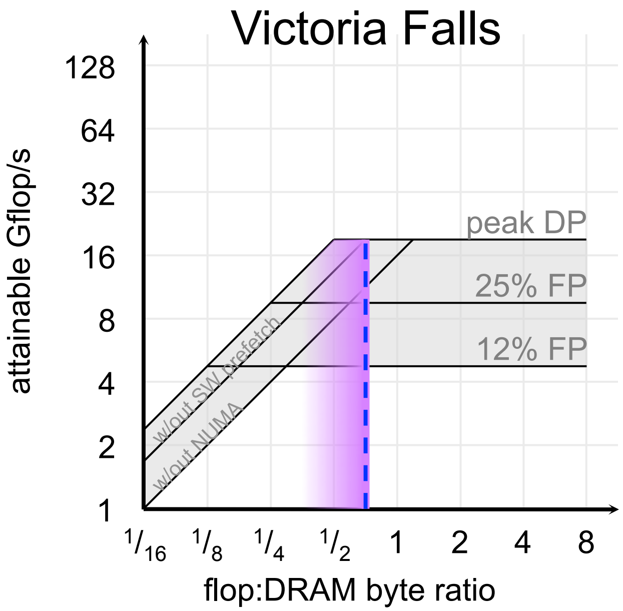
momentum distribution

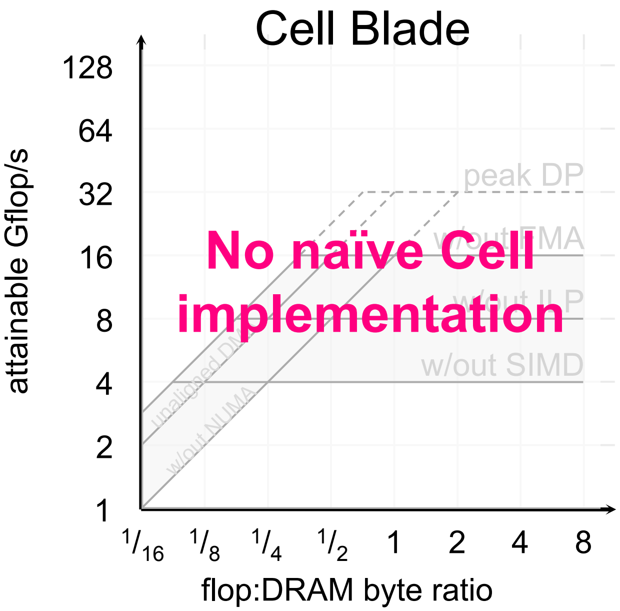
magnetic distribution

Roofline model for LBMHD







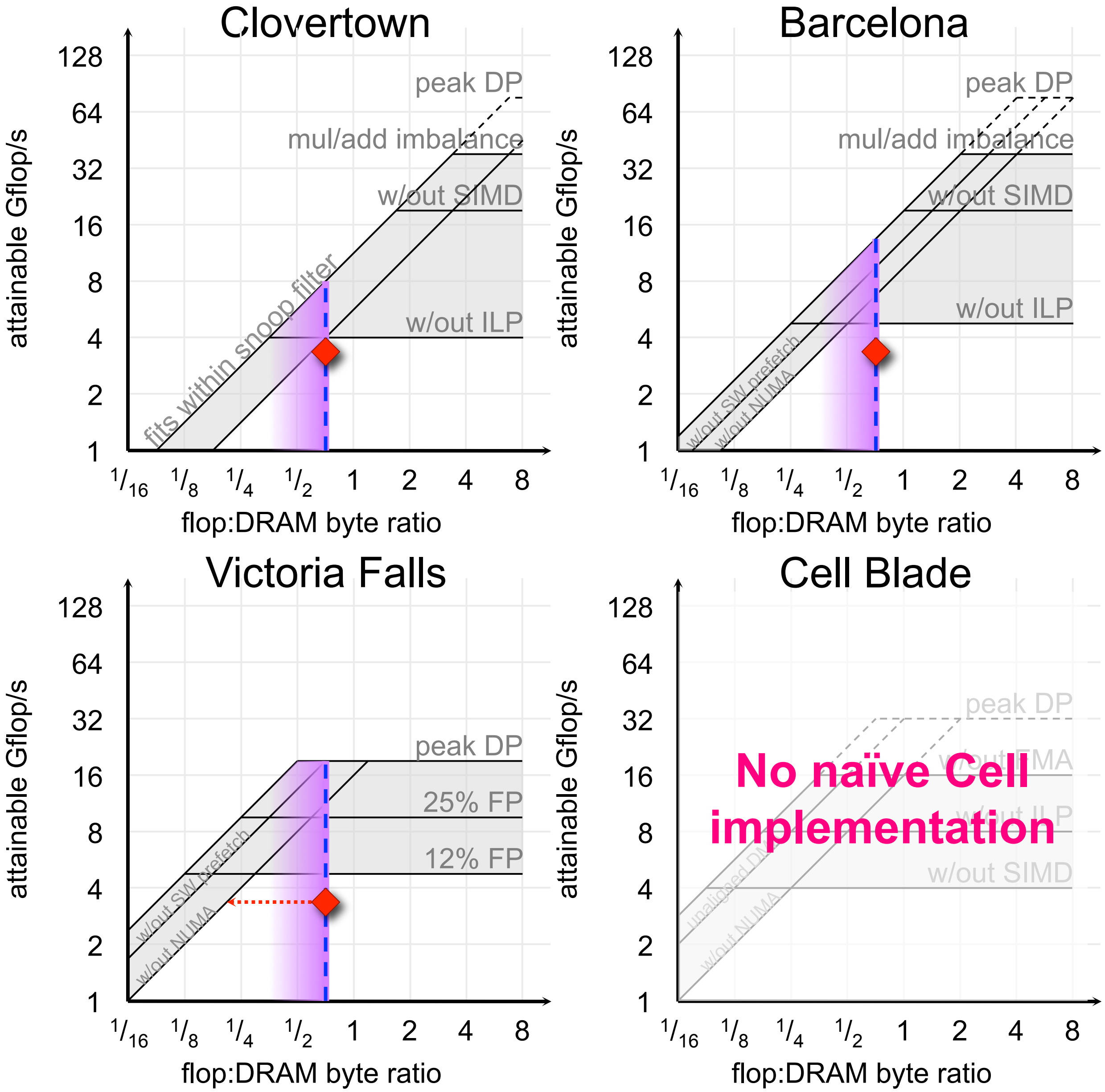


No naïve Cell implementation

- Huge datasets
- NUMA allocation/access
- Little ILP
- No DLP
- Far more adds than multiplies (imbalance)
- **Essentially random access to memory**
- Flop:byte ratio ~0.7
- High conflict misses

Roofline model for LBMHD

(out-of-the-box code)

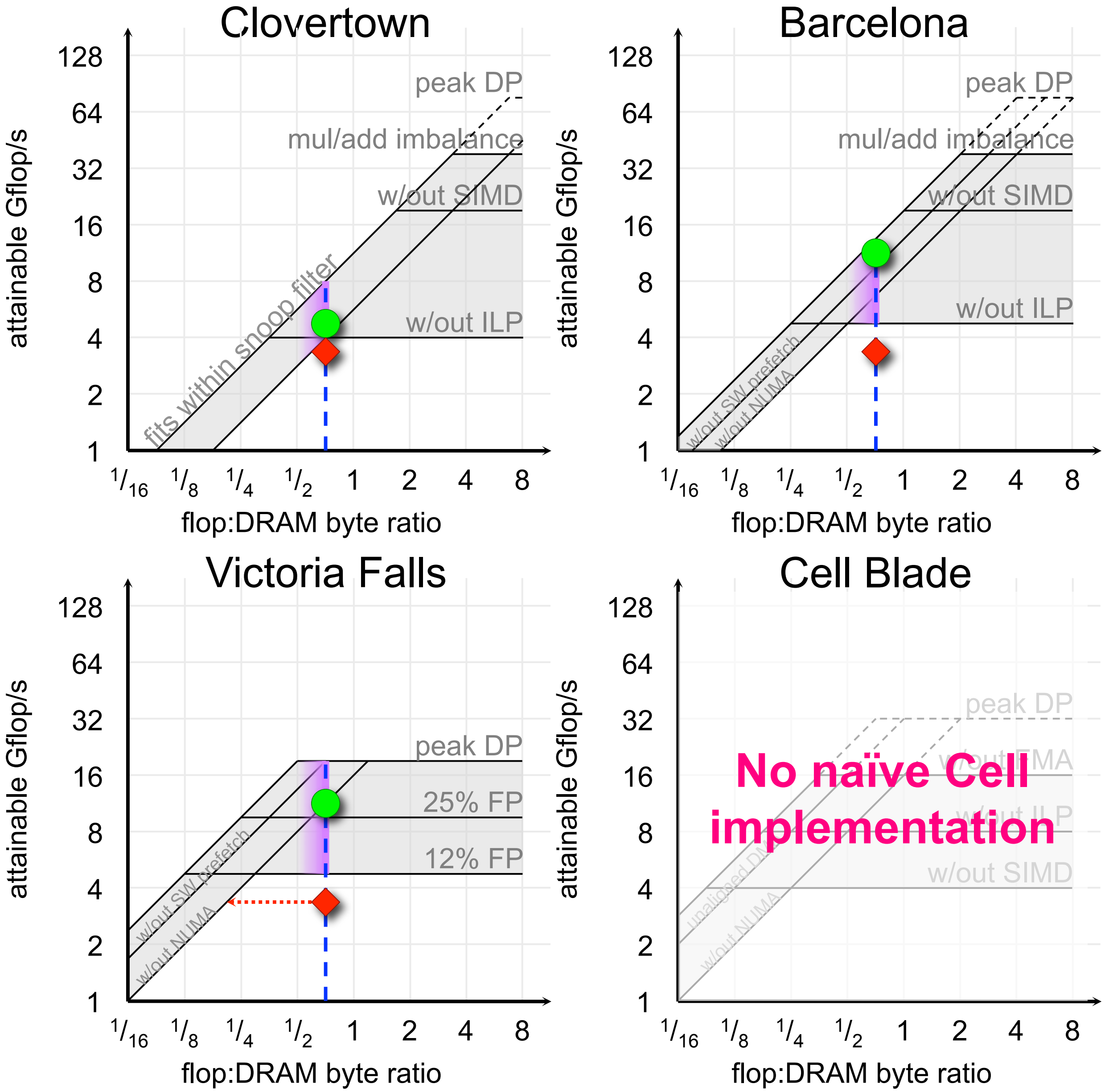


No naïve Cell implementation

- ❖ Huge datasets
- ❖ NUMA allocation/access
- ❖ Little ILP
- ❖ No DLP
- ❖ Far more adds than multiplies (imbalance)
- ❖ **Essentially random access to memory**
- ❖ Flop:byte ratio ~0.7
- ❖ High conflict misses
- ❖ Peak VF performance with 64 threads (our of 128) - high conflict misses

Roofline model for LBMHD

(Padding, Vectorization, Unrolling, Reordering)



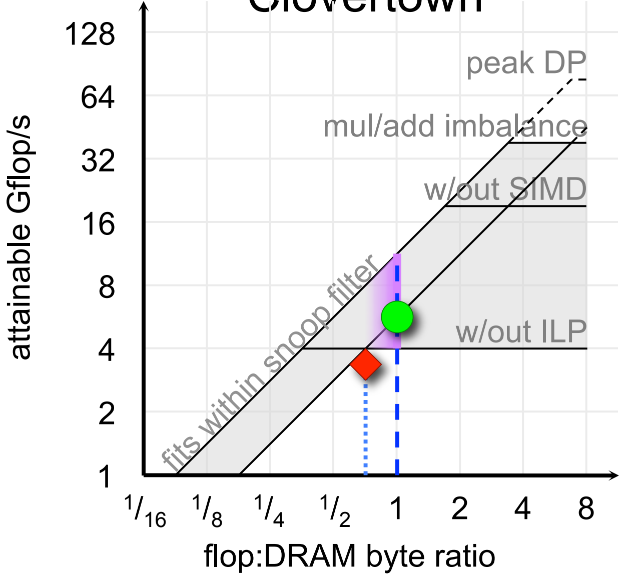
No naïve Cell implementation

- ❖ Vectorize the code to eliminate TLB capacity misses
- ❖ Ensures unit stride access (bottom bandwidth ceiling)
- ❖ Tune for optimal VL
- ❖ Clovertown pinned to lower BW ceiling

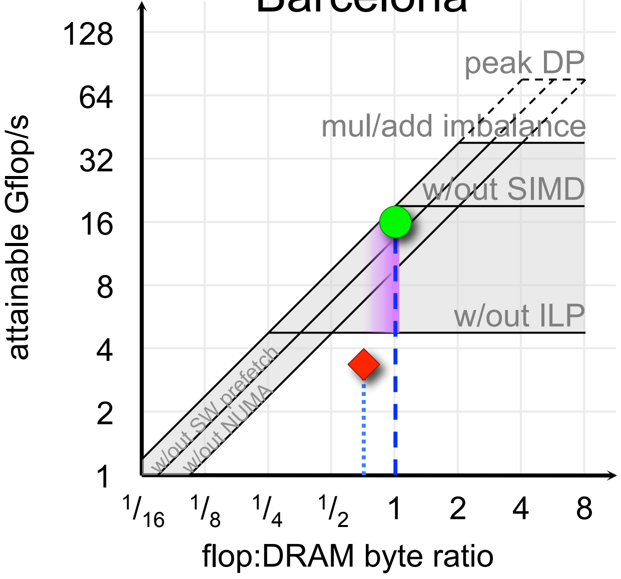
Roofline model for LBMHD

(SIMDization + cache bypass)

Clovertown

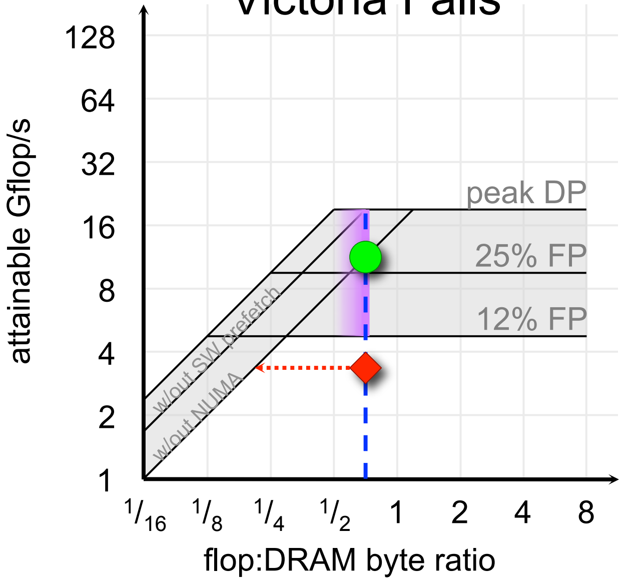


Barcelona

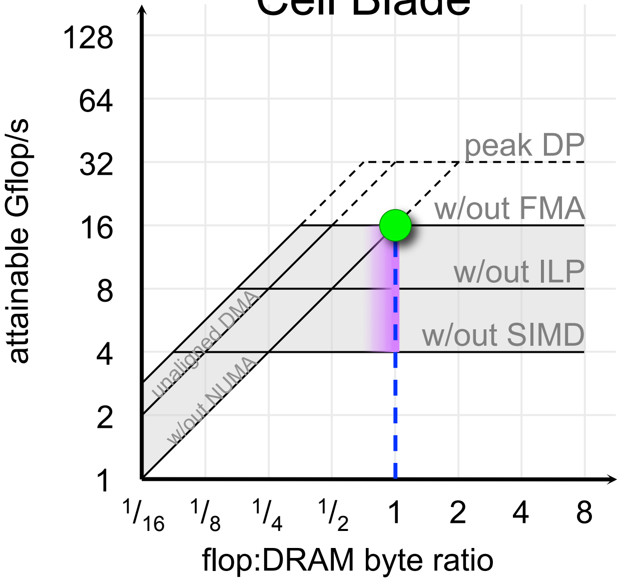


- ❖ Make SIMDization explicit
- ❖ Technically, this swaps ILP and SIMD ceilings
- ❖ Use cache bypass instruction: ***movntpd***
- ❖ Increases flop:byte ratio to ~1.0 on x86/Cell

Victoria Falls



Cell Blade

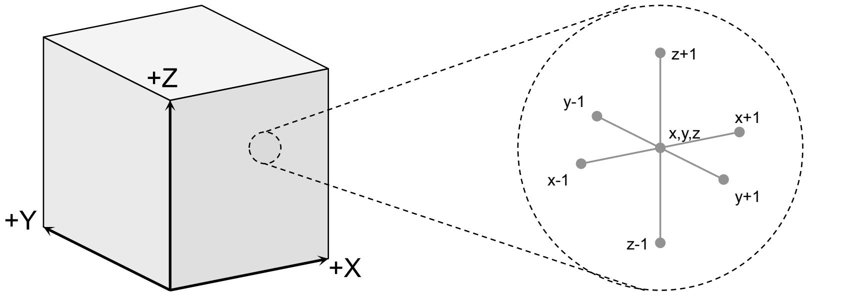


The Heat Equation Stencil

Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, Katherine Yelick, “Stencil Computation Optimization and Autotuning on State-of-the-Art Multicore Architecture”, submitted to Supercomputing (SC), 2008.

The Heat Equation Stencil

- Explicit Heat equation on a regular grid
- Jacobi
- One double per point in space
- 7-point nearest neighbor stencil
- Must:
  - read every point from DRAM
  - perform 8 flops (linear combination)
  - write every point back to DRAM
- Just over 0.5 flops/byte (ideal)
- Cache locality is important

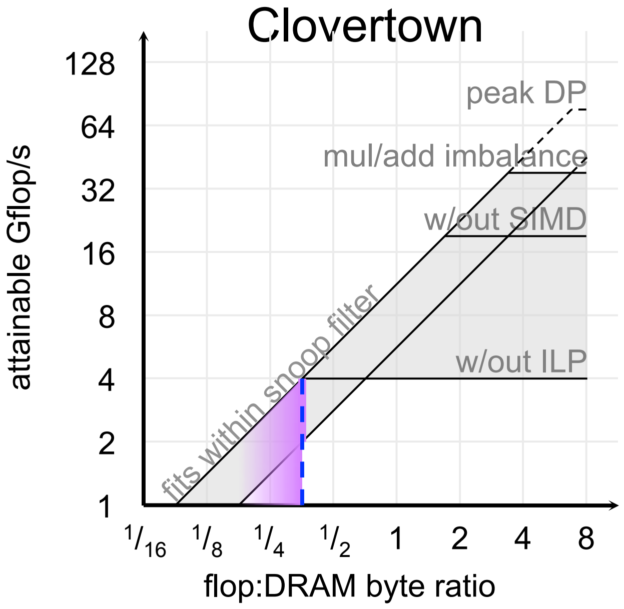


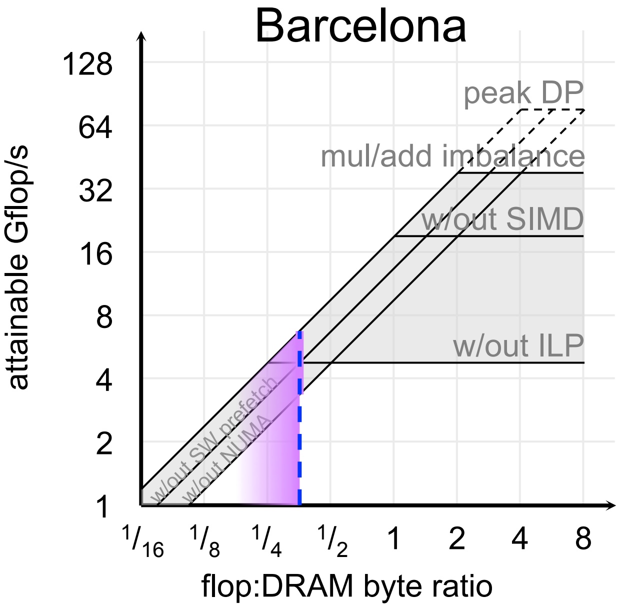
PDE grid

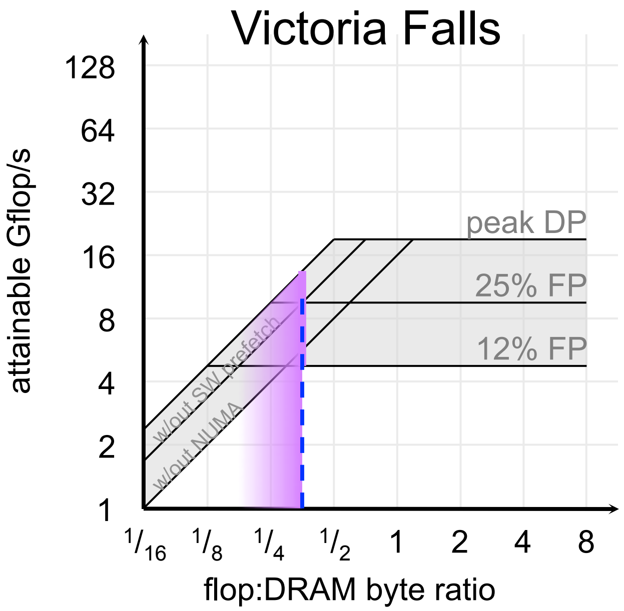
stencil for heat equation PDE

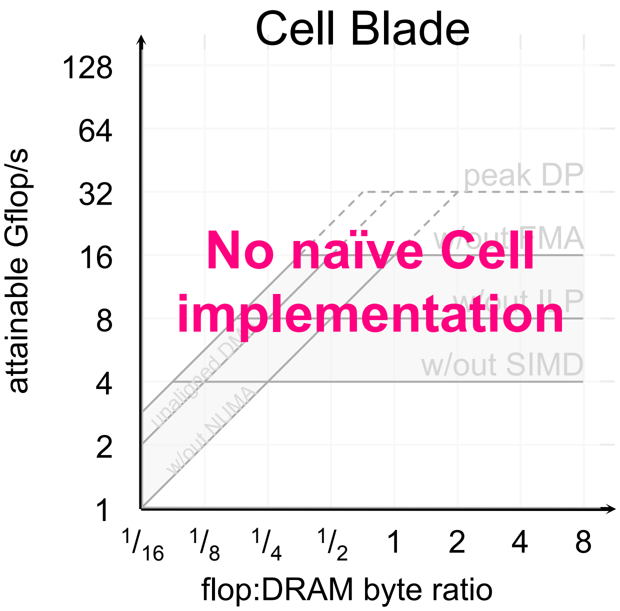
Roofline model for Stencil

(out-of-the-box code)


Clovertown


Barcelona


Victoria Falls


Cell Blade

No naïve Cell implementation

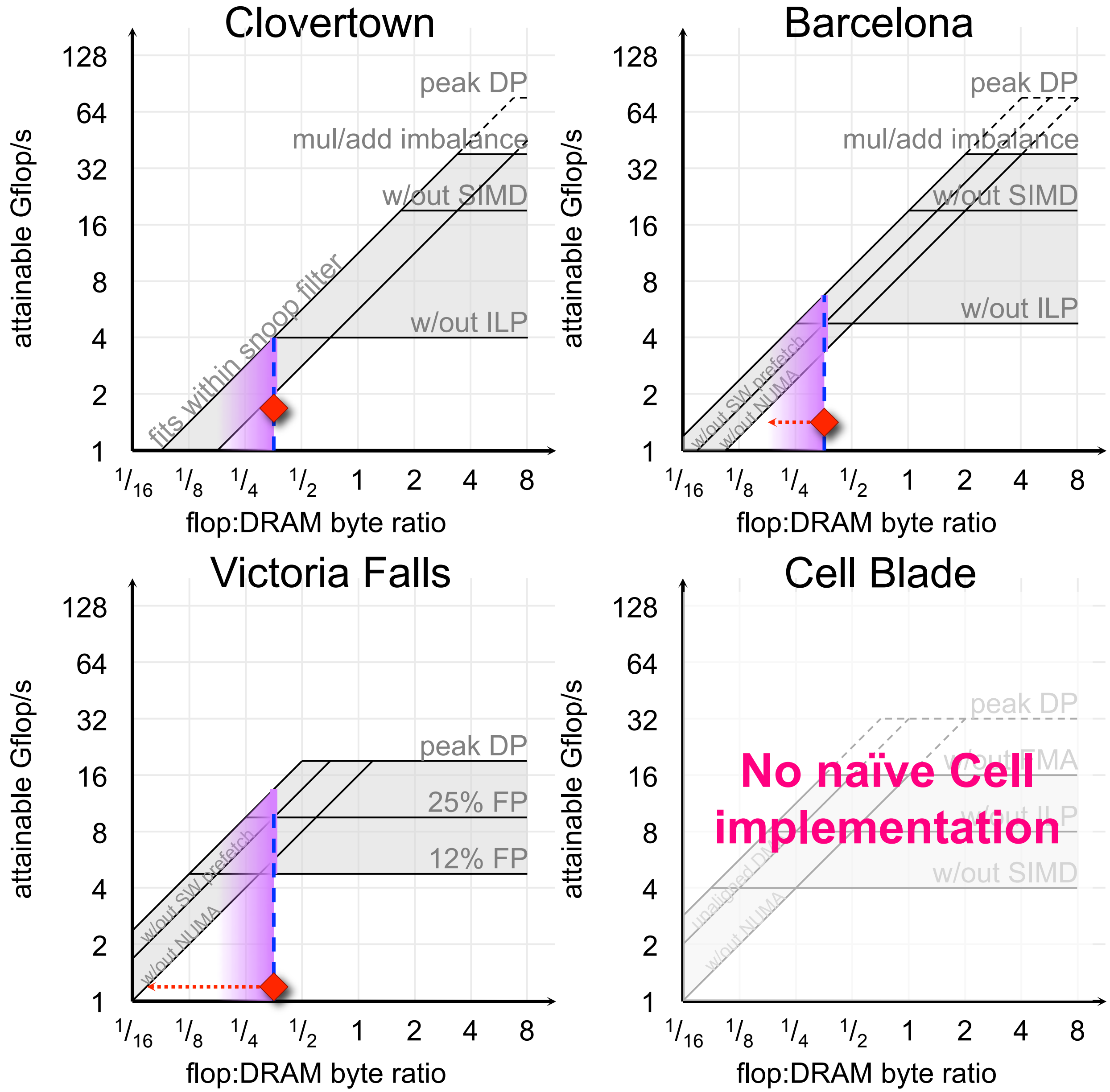
- Large datasets
- 2 unit stride streams
- No NUMA
- Little ILP
- No DLP
- Far more adds than multiplies (imbalance)
- Ideal flop:byte ratio $^{1}/_{3}$
- High locality requirements
- Capacity and conflict misses will severely impair flop:byte ratio

EECS
Electrical Engineering and
Computer Sciences

Roofline model for Stencil

(out-of-the-box code)

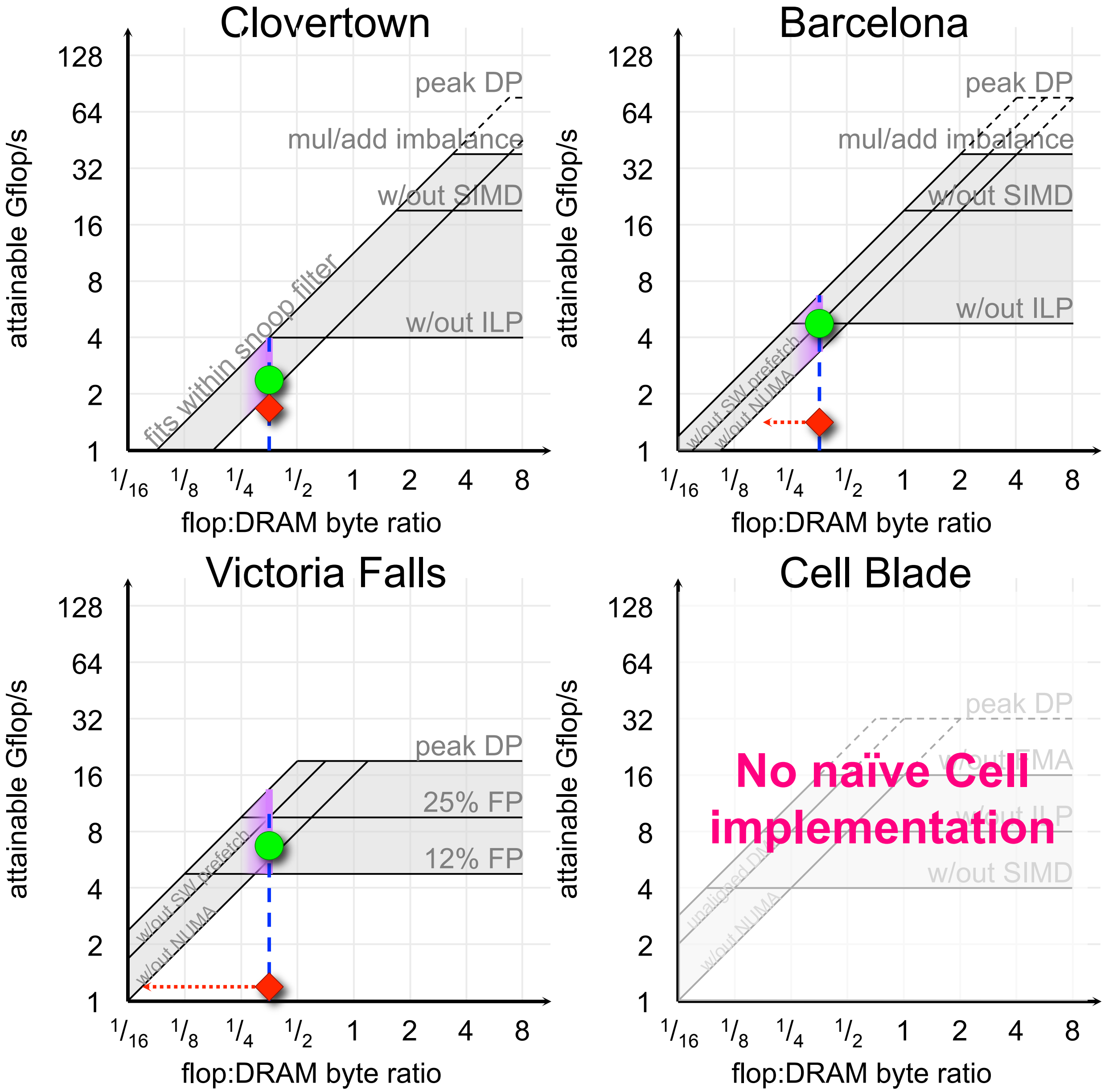
BERKELEY PAR LAB



- Large datasets
- 2 unit stride streams
- No NUMA
- Little ILP
- No DLP
- Far more adds than multiplies (imbalance)
- Ideal flop:byte ratio $^1/_3$
- High locality requirements
- Capacity and conflict misses will severely impair flop:byte ratio

Roofline model for Stencil

(NUMA, cache blocking, unrolling, prefetch, …)



Clovertown

Barcelona

Victoria Falls

Cell Blade

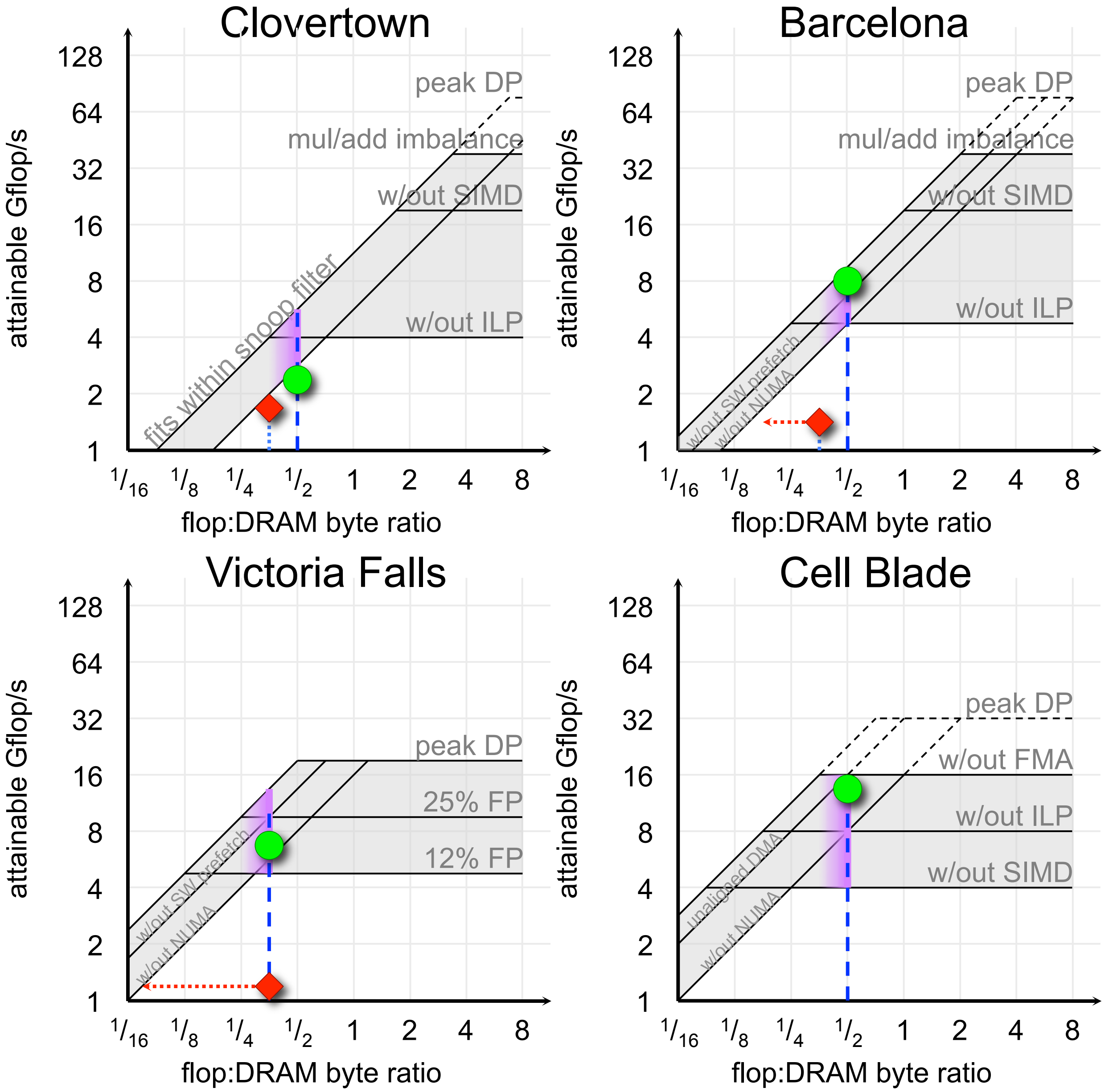
No naïve Cell implementation

- ❖ Cache blocking helps ensure flop:byte ratio is as close as possible to $^1/_3$
- ❖ Clovertown has huge caches but is pinned to lower BW ceiling
- ❖ Cache management is essential when capacity/ thread is low

EECS
Electrical Engineering and Computer Sciences

Roofline model for Stencil
(SIMDization + cache bypass)

BERKELEY PAR LAB


Clovertown

Barcelona

Victoria Falls

Cell Blade

- ❖ Make SIMDization explicit
- ❖ Technically, this swaps ILP and SIMD ceilings
- ❖ Use cache bypass instruction: ***movntpd***
- ❖ Increases flop:byte ratio to ~0.5 on x86/Cell

Refining the Roofline

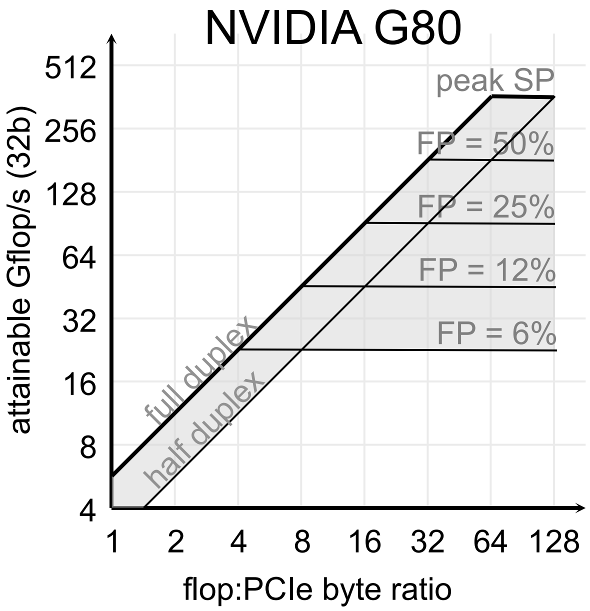
Other performance metrics

- There is no reason either floating point (Gflop/s) must be the performance metric

- Could also use:
  - Graphics (Pixels, Vertices, Textures)
  - Crypto
  - Integer
  - Bitwise
  - etc…

Other bandwidths

- For our kernels, DRAM bandwidth is the key communication component.
- For other kernels, other bandwidths might be more appropriate
  - L2 bandwidth (e.g. DGEMM)
  - PCIe bandwidth (offload to GPU)
  - Network bandwidth
- The example below shows zero overhead double buffered transfers to/from a GPU over PCIe x16
  - How bad is a SP stencil ?
  - What about SGEMM ?
- No overlap / high overhead tends to smooth performance
  - Performance is half at ridge point



Mix and match

- In general, you can mix and match as the kernel/architecture requires:
- e.g. all posibilities is the cross product of performance metrics with bandwidths

{Gflop/s, GIPS, crypto, …} × {L2, DRAM, PCIe, Network}

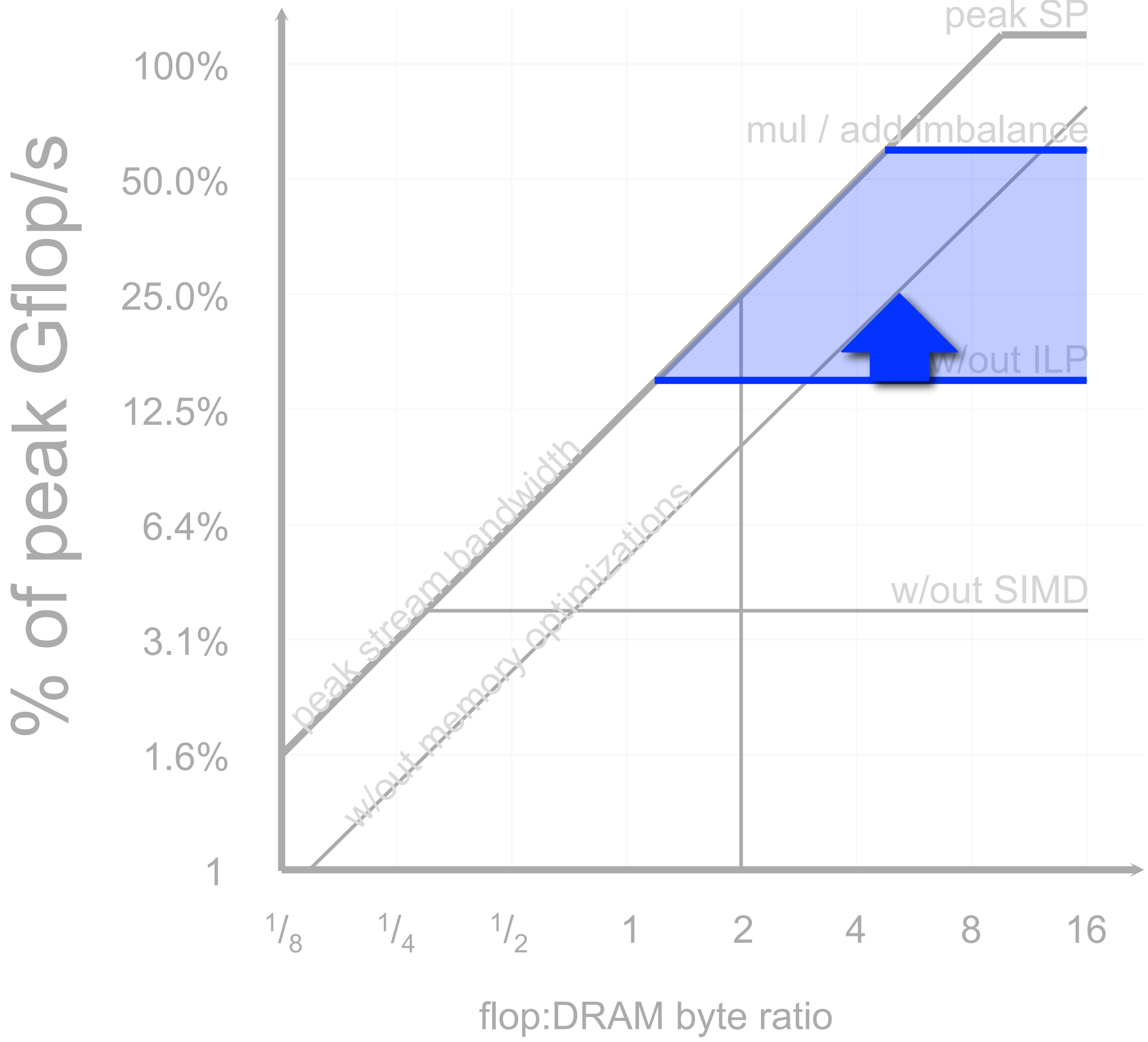
Conclusions

- The Roofline Model provides an intuitive graph for kernel analysis and optimization
- Easily extendable to other architectural paradigms
- Easily extendable to other communication or computation metrics

Questions ?

BACKUP SLIDES

Trends

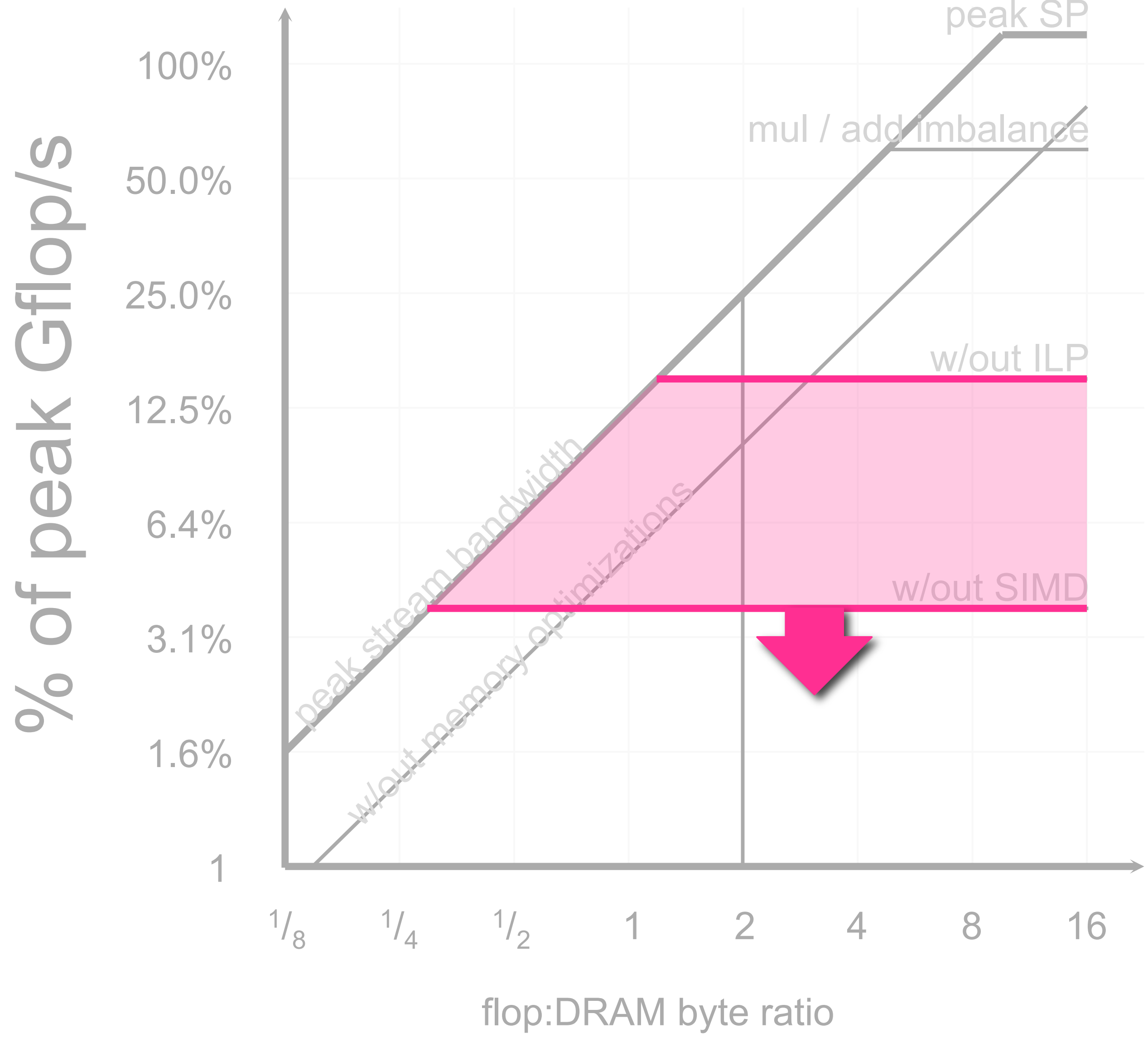


- ❖ ILP is decreasing (shorter pipelines, multithreading)
- ❖ SIMD is becoming wider
- ❖ Bandwidth isn’t keeping up with #cores
- ❖ Application flop:byte is decreasing
- ❖ Cache/Local Store management is becoming critical

EECS Electrical Engineering and Computer Sciences

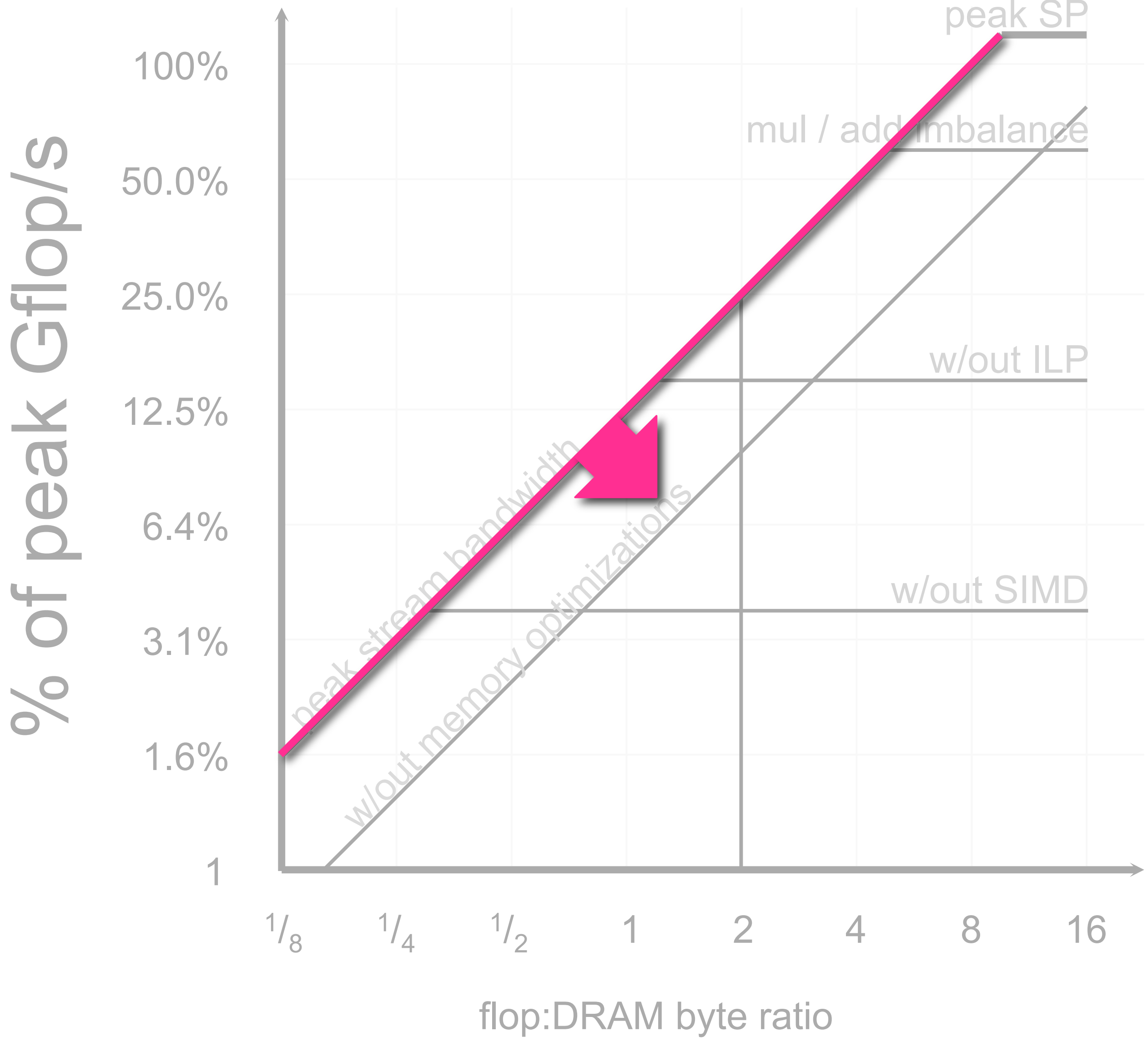
Trends

Berkeley Par Lab



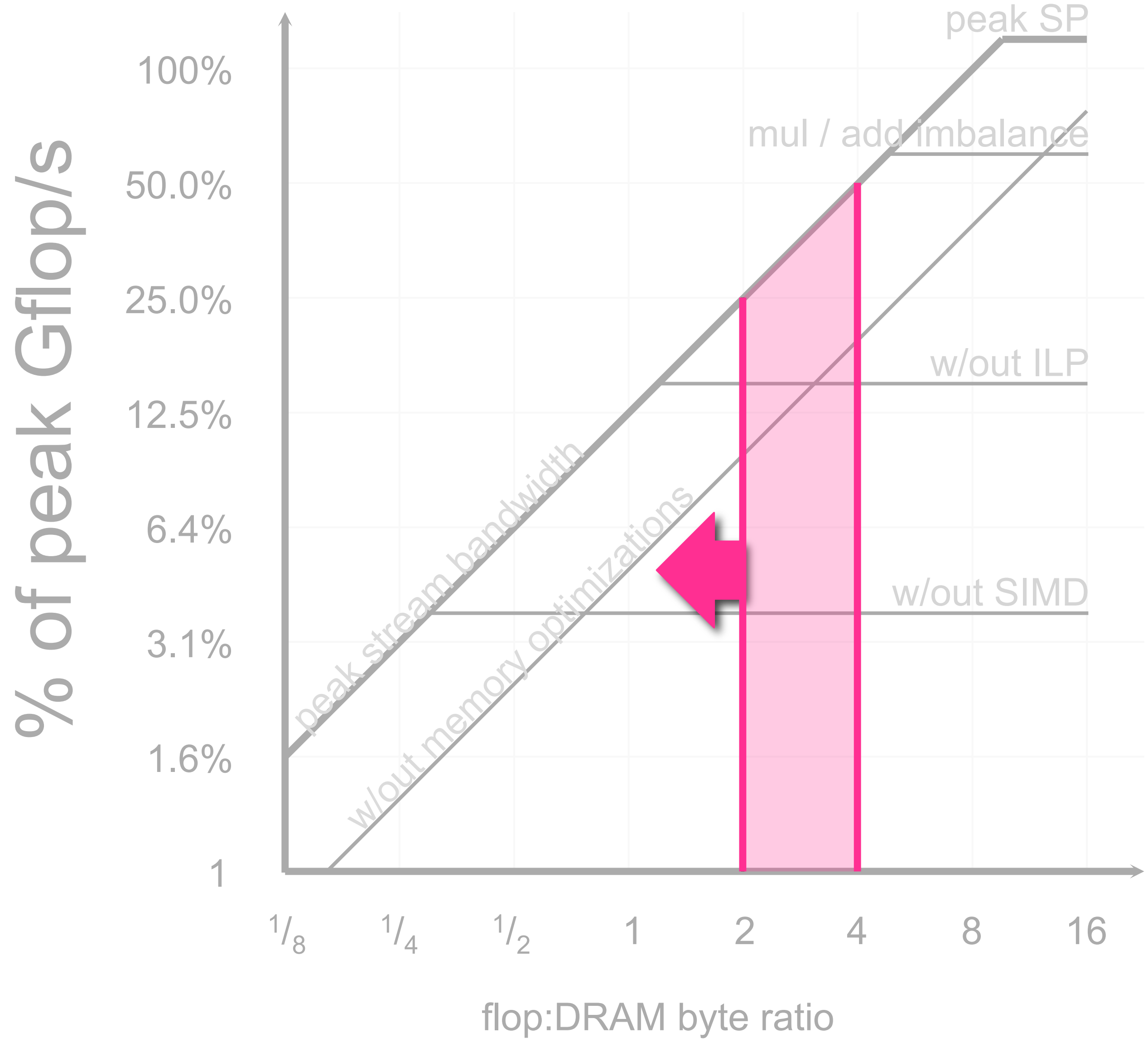
- ❖ ILP is decreasing (shorter pipelines, multithreading)
- ❖ SIMD is becoming wider
- ❖ Bandwidth isn’t keeping up with #cores
- ❖ Application flop:byte is decreasing
- ❖ Cache/Local Store management is becoming critical

Trends



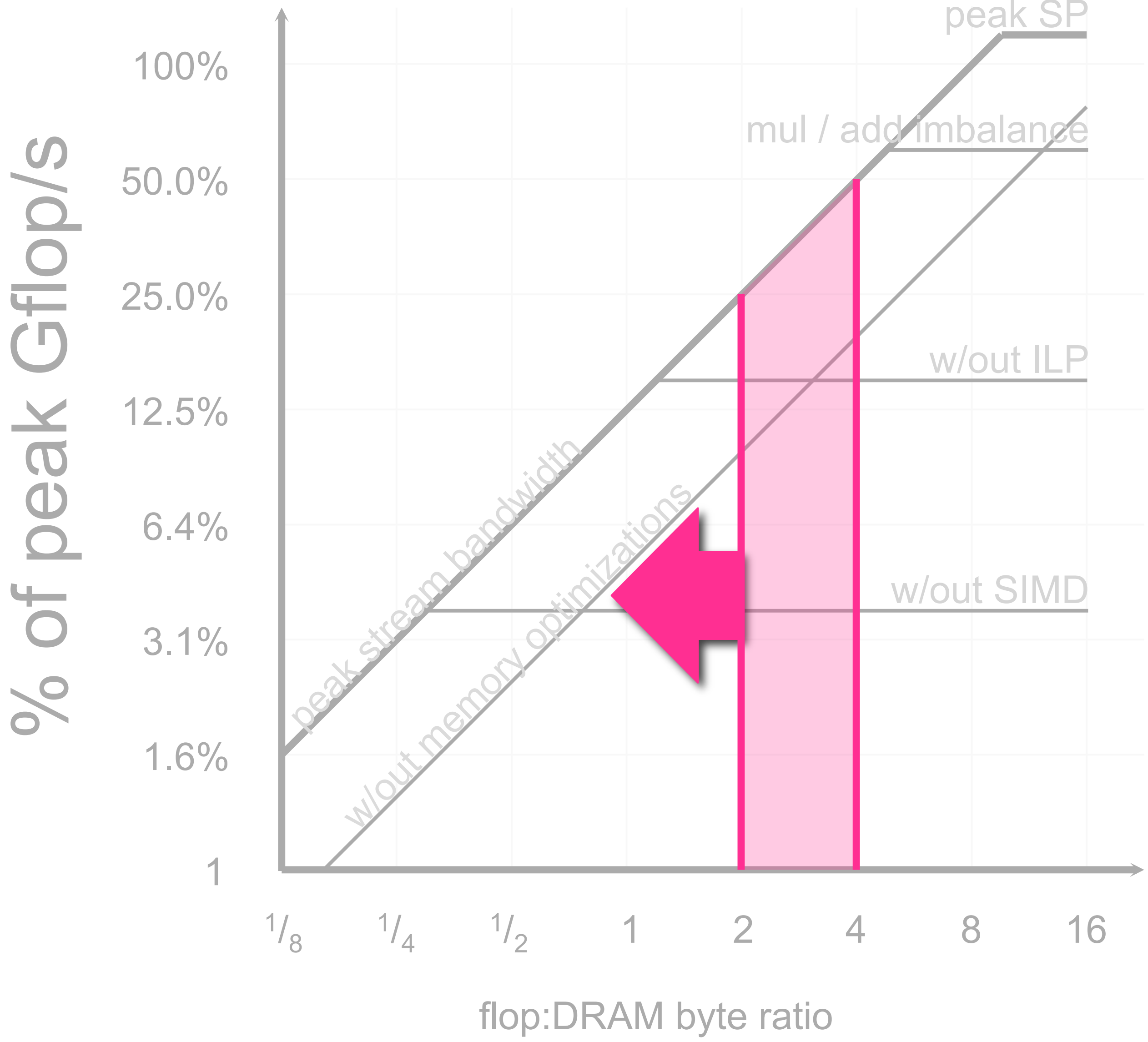
- ILP is decreasing (shorter pipelines, multithreading)
- SIMD is becoming wider
- Bandwidth isn’t keeping up with #cores
- Application flop:byte is decreasing
- Cache/Local Store management is becoming critical

Trends



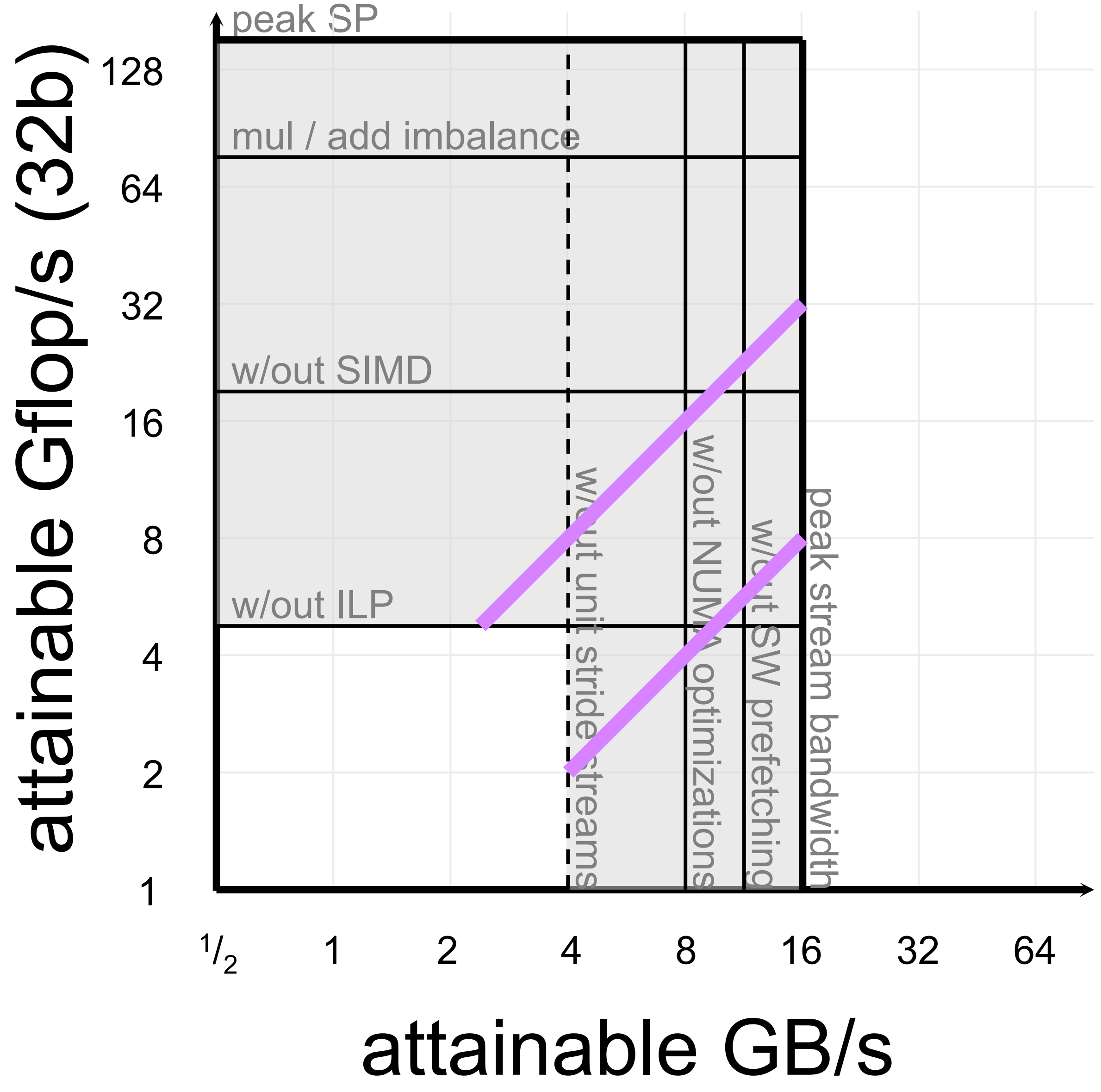
- ILP is decreasing (shorter pipelines, multithreading)
- SIMD is becoming wider
- Bandwidth isn’t keeping up with #cores
- Application flop:byte is decreasing
- Cache/Local Store management is becoming critical

Trends



- ILP is decreasing (shorter pipelines, multithreading)
- SIMD is becoming wider
- Bandwidth isn’t keeping up with #cores
- Application flop:byte is decreasing
- Cache/Local Store management is becoming critical

Alternate View (Barcelona)



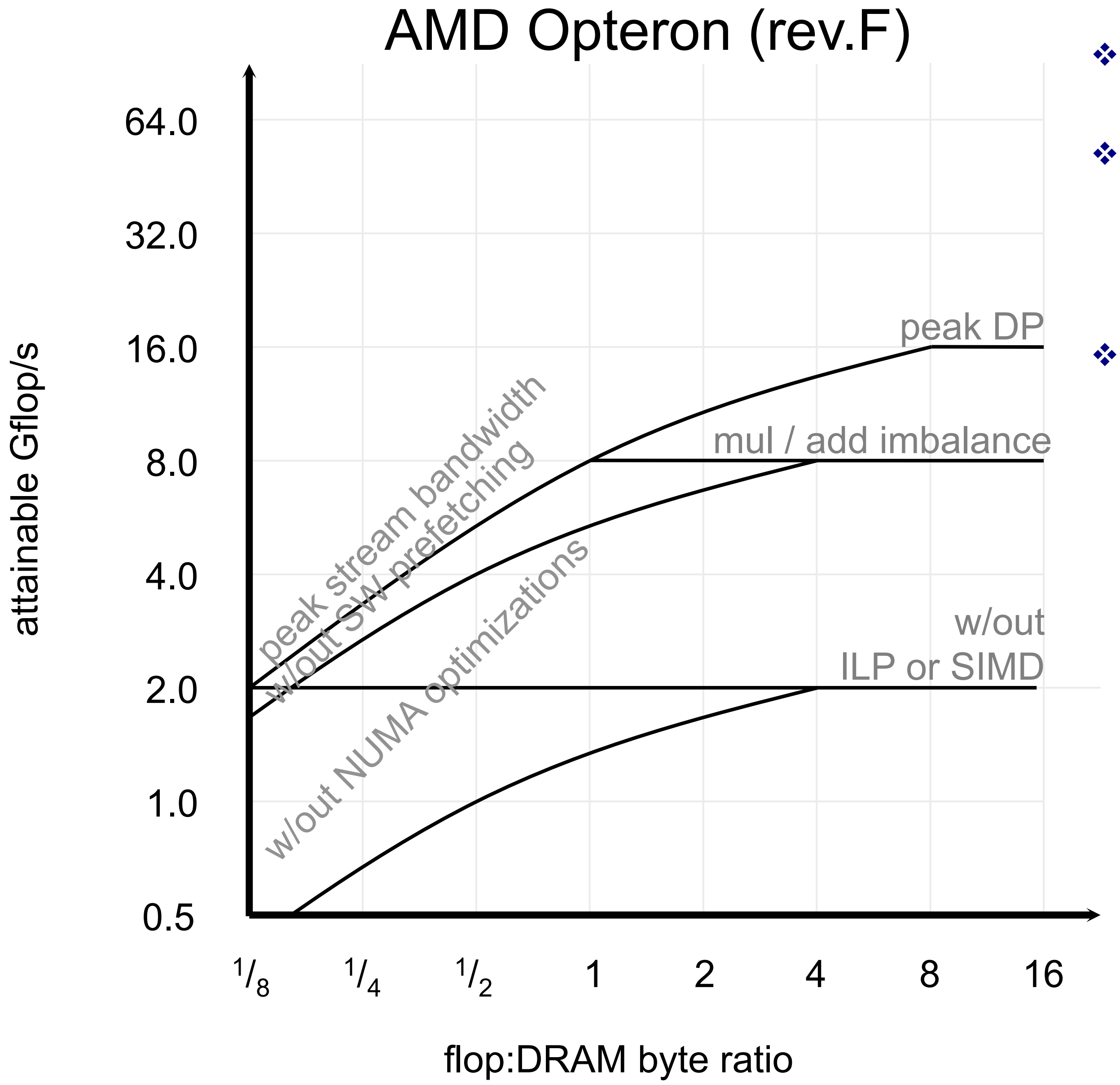
- Same formalism can be plotted on different axis
- Difficult to plot AI

No Overlap

- What if computation or communication isn't totally overlapped
- At ridgepoint 50% of the time is spent in each, so performance is cut in half
- In effect, the curves are smoothed
- Common for bulk synchonous MPI communication, atypical for DRAM access on modern architectures

Roofline model for Opteron

(non-overlapped)



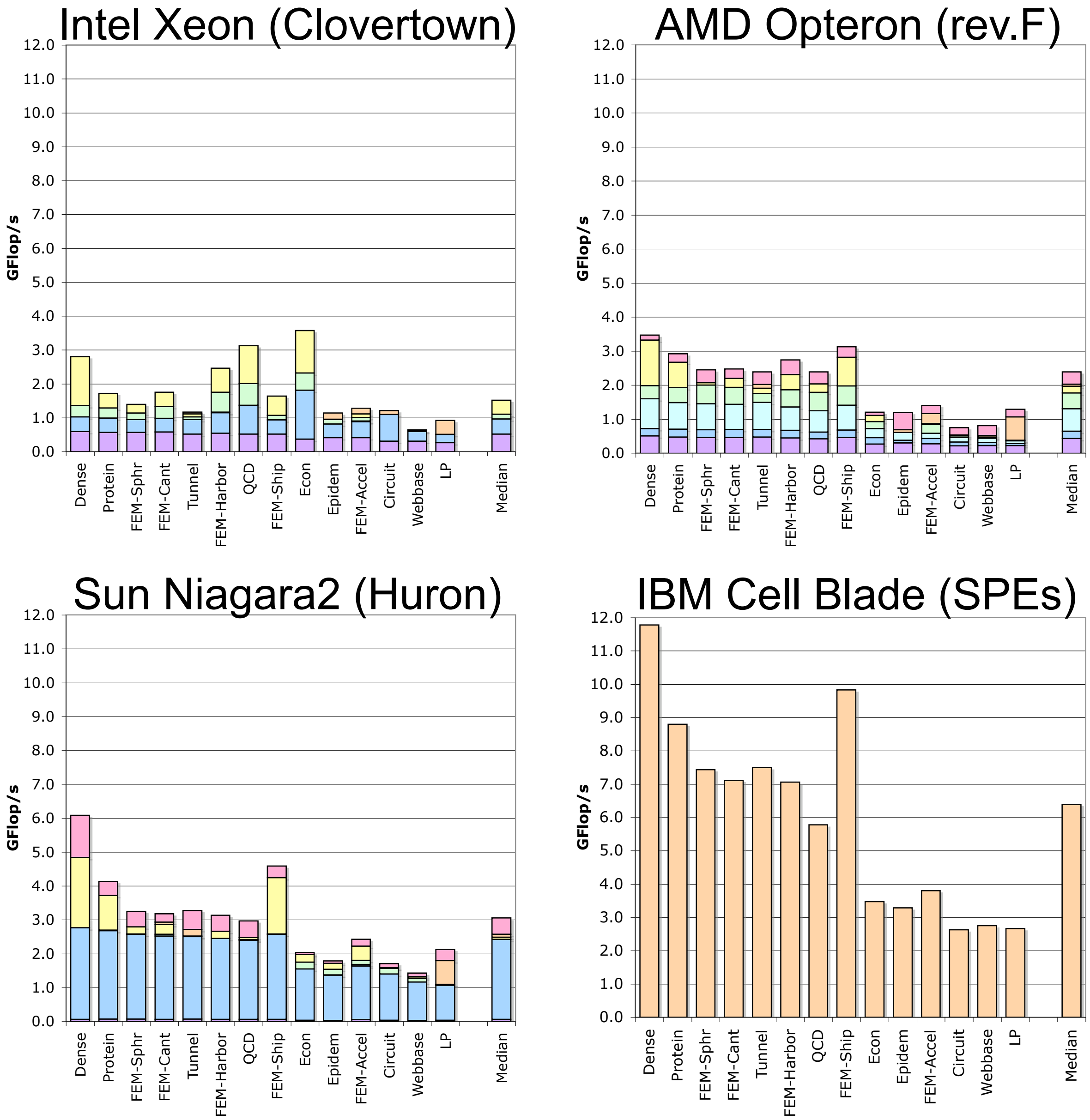
- ❖ Not typical of multi-thread/ core architectures
- ❖ Not typical of architectures with ooo or HW prefetchers
- ❖ More common in network accesses

Auto-tuned Performance

(+Cell/SPE version)

- Wrote a double precision Cell/SPE version
- DMA, local store blocked, NUMA aware, etc…
- Only 2x1 and larger BCOO
- Only the SpMV-proper routine changed

- About 12x faster (median) than using the PPEs alone.



🟪 +More DIMMs(opteron), +FW fix, array padding(N2), etc…

🟧 +Cache/TLB Blocking

🟨 +Compression

🟩 +SW Prefetching

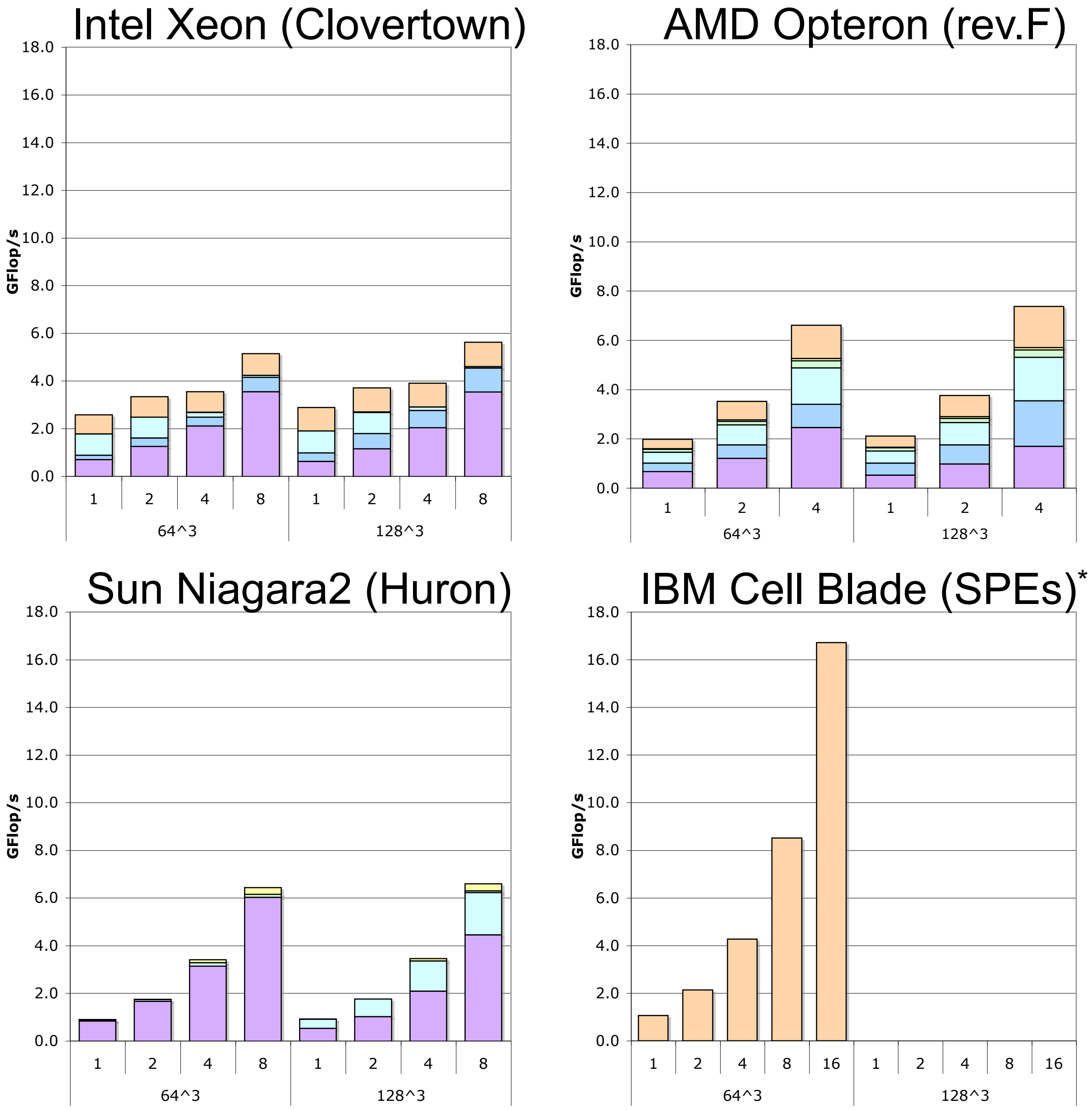
⬜ +NUMA/Affinity

🟦 Naïve Pthreads

🟪 Naïve

Auto-tuned Performance

(Local Store Implementation)



- ❖ First attempt at cell implementation.
- ❖ VL, unrolling, reordering fixed
- ❖ No NUMA
- ❖ Exploits DMA and double buffering to load vectors
- ❖ Straight to SIMD intrinsics.
- ❖ Despite the relative performance, Cell's DP implementation severely impairs performance

🟧 +SIMDization
🟨 +SW Prefetching
🟩 +Unrolling
▢ +Vectorization
🟦 +Padding
🟪 Naïve+NUMA

*collision() only

Where do GPUs fit in?

	Expressed at compile time	Discovered at run time
Instruction Level Parallelism	VLIW	superscalar, SMT, etc…
Data Level Parallelism	SIMD, Vector	G80

- GPUs discover data level parallelism from thread level parallelism at runtime