

# The Roofline Model:

A pedagogical tool for program analysis and  
optimization

**ParLab Summer Retreat**

Samuel Williams, David Patterson

*samw@cs.berkeley.edu*

- ❖ 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

- ❖ 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

- ❖ 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

- ❖ 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

- ❖ 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)

- ❖ 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

- ❖ 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

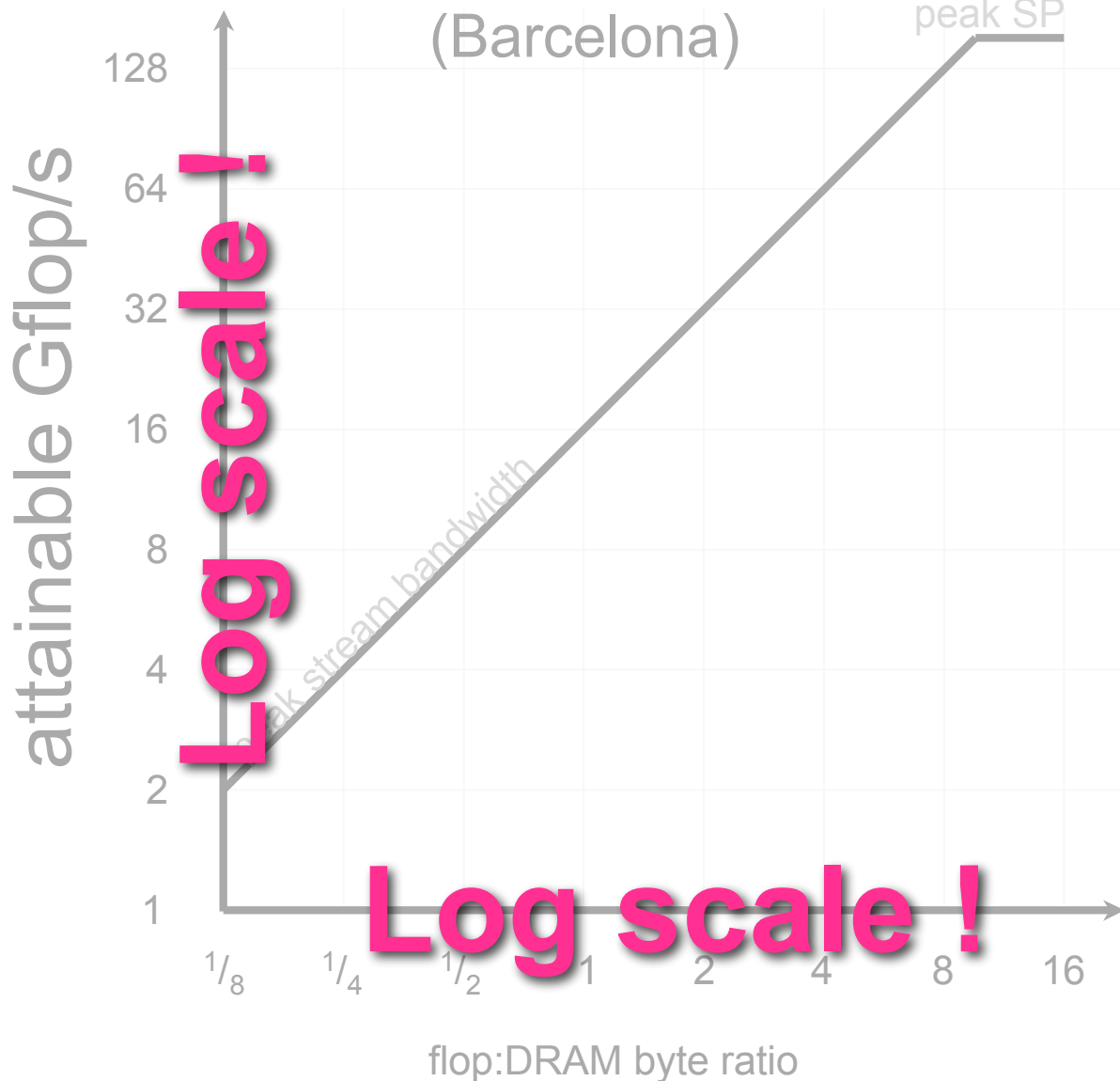
- ❖ Through dimensional analysis, its clear that **Flops:Bytes** is the parameter that allows us to convert bandwidth (GB/s) to performance (GFlop/s)
- ❖ This is a well known quantity: **Arithmetic Intensity** (discussed later)
- ❖ When we measure total bytes, we incorporate all cache behavior (the 3C's) and Locality

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

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \text{Peak Gflop/s} \\ \text{Stream BW} * \text{actual flop:byte ratio} \end{array} \right.$$

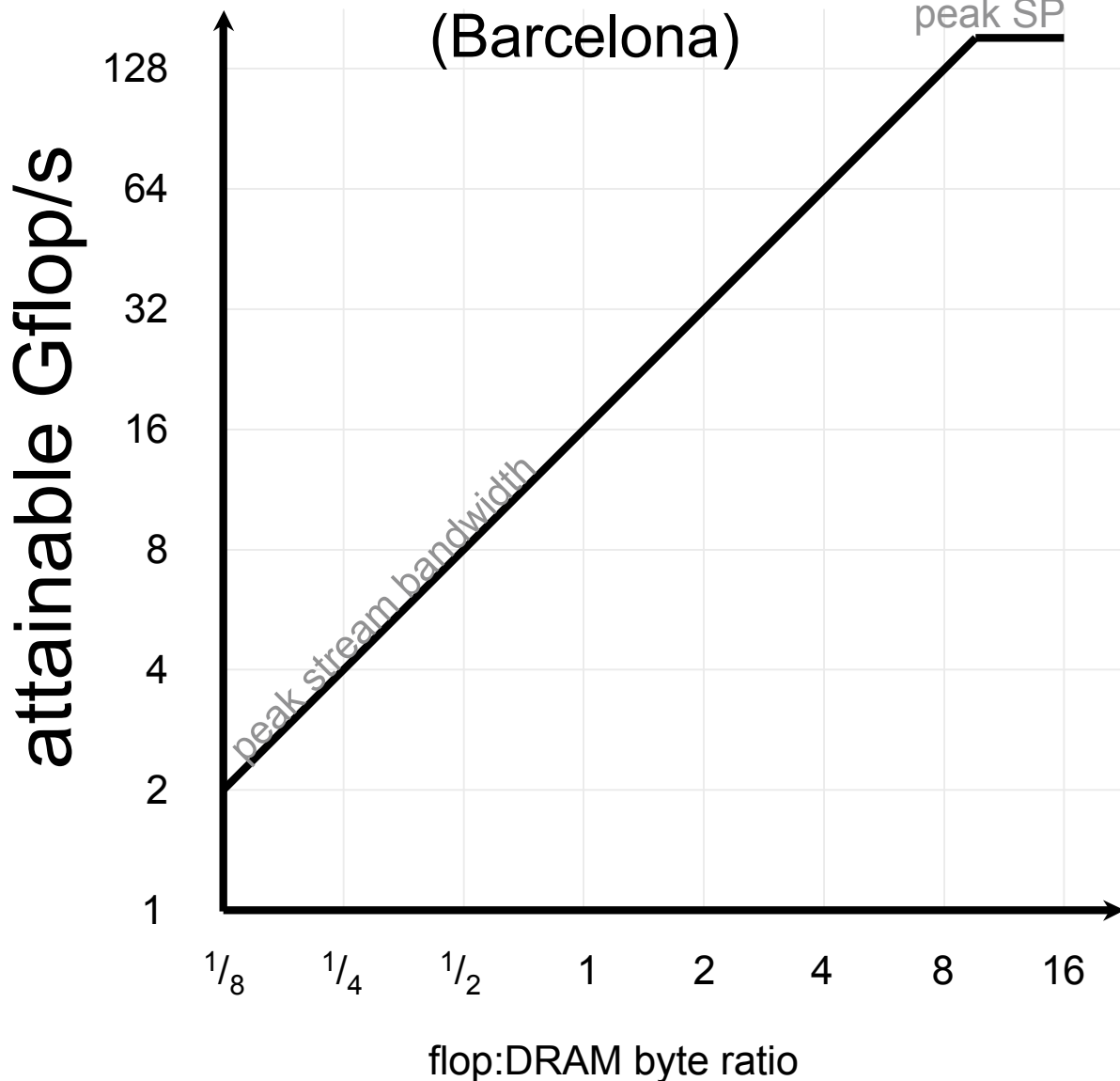
- ❖ *Bandwidth #'s collected via micro benchmarks*
- ❖ *Computation #'s derived from optimization manuals (pencil and paper)*
- ❖ *Assume complete overlap of either communication or computation*

AMD Opteron 2356  
(Barcelona)



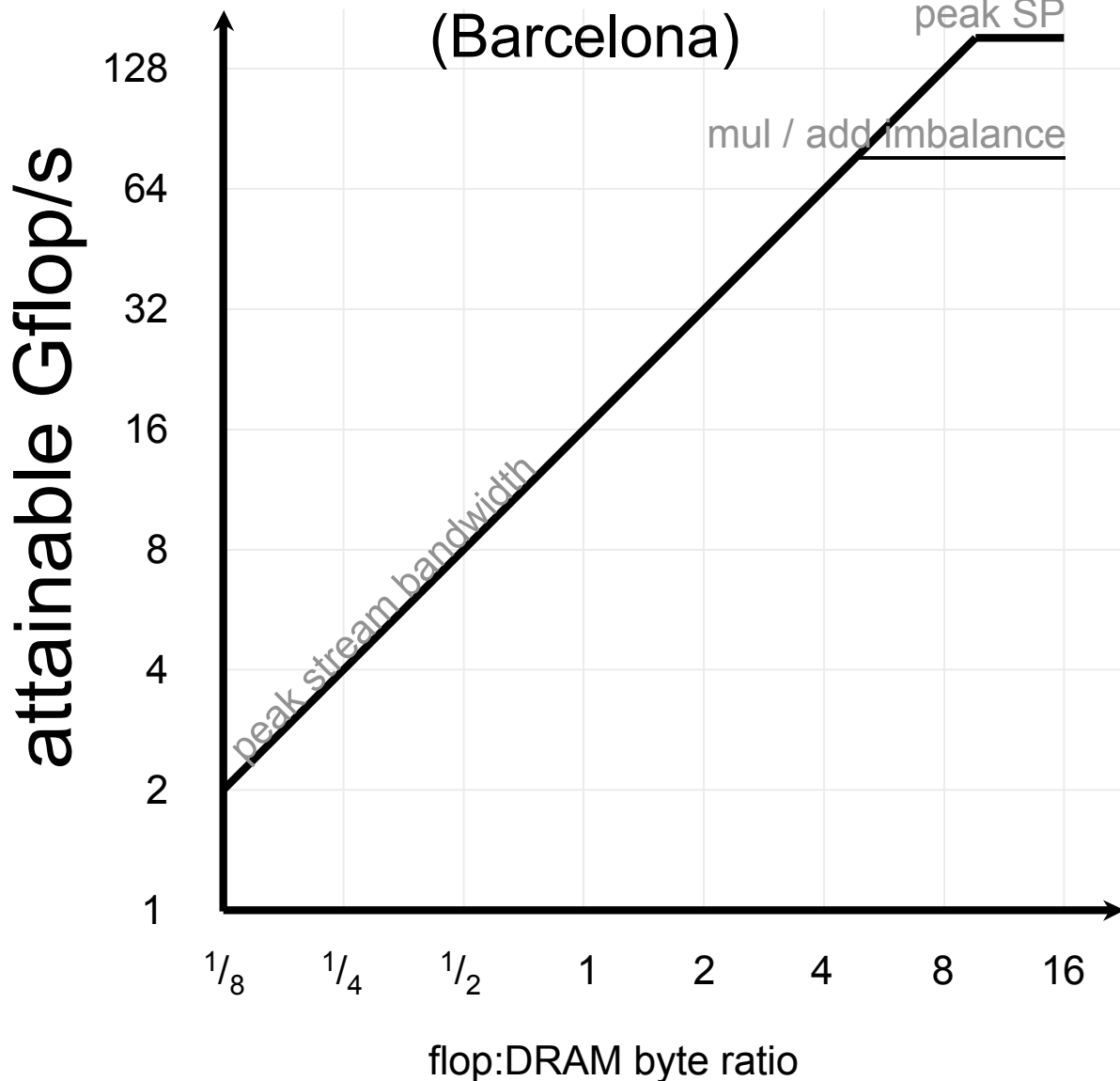
- ❖ Peak roofline performance
- ❖ based on manual for **single precision peak**
- ❖ and a hand tuned stream read for bandwidth

AMD Opteron 2356  
(Barcelona)



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

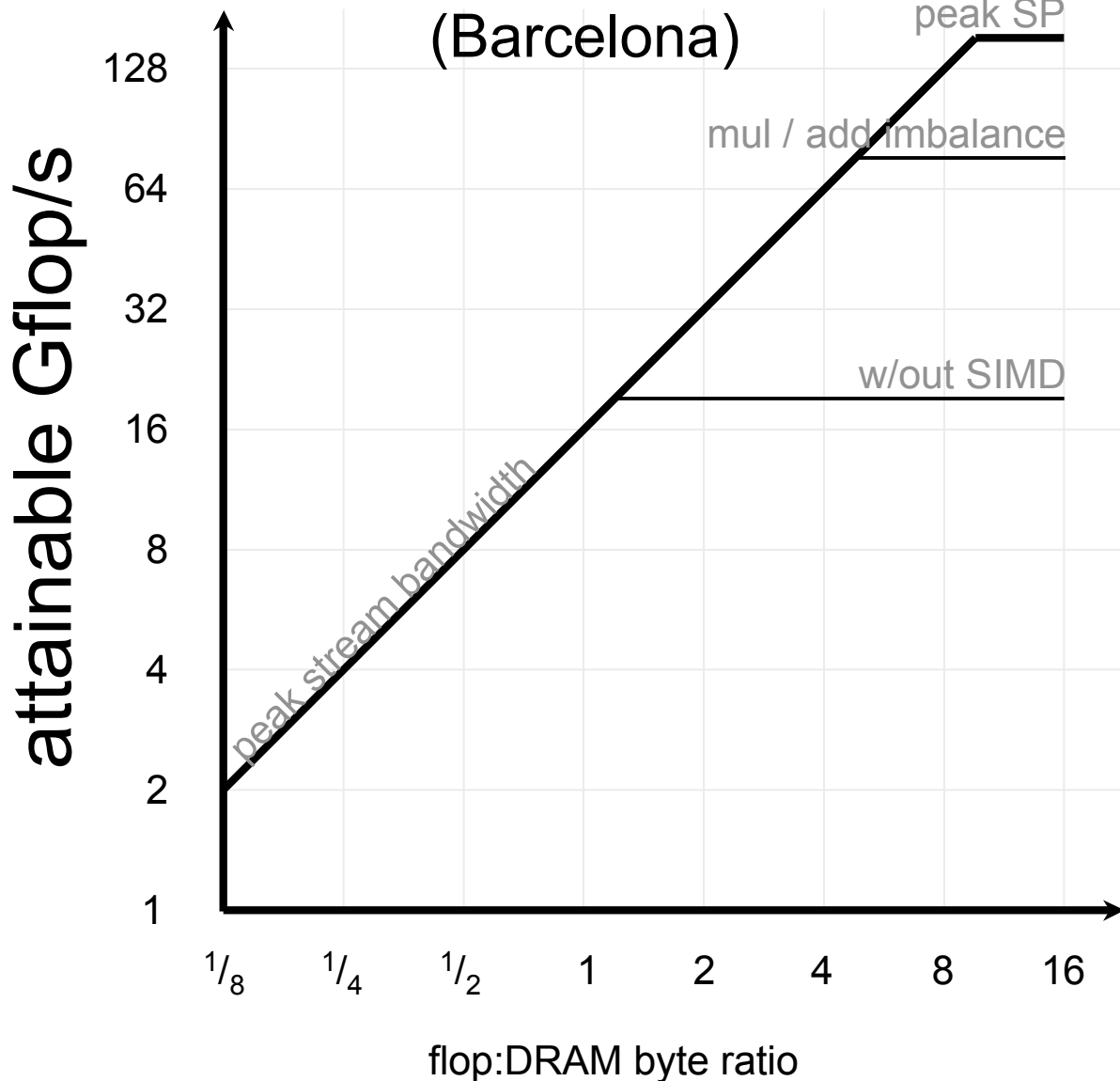
AMD Opteron 2356  
(Barcelona)



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

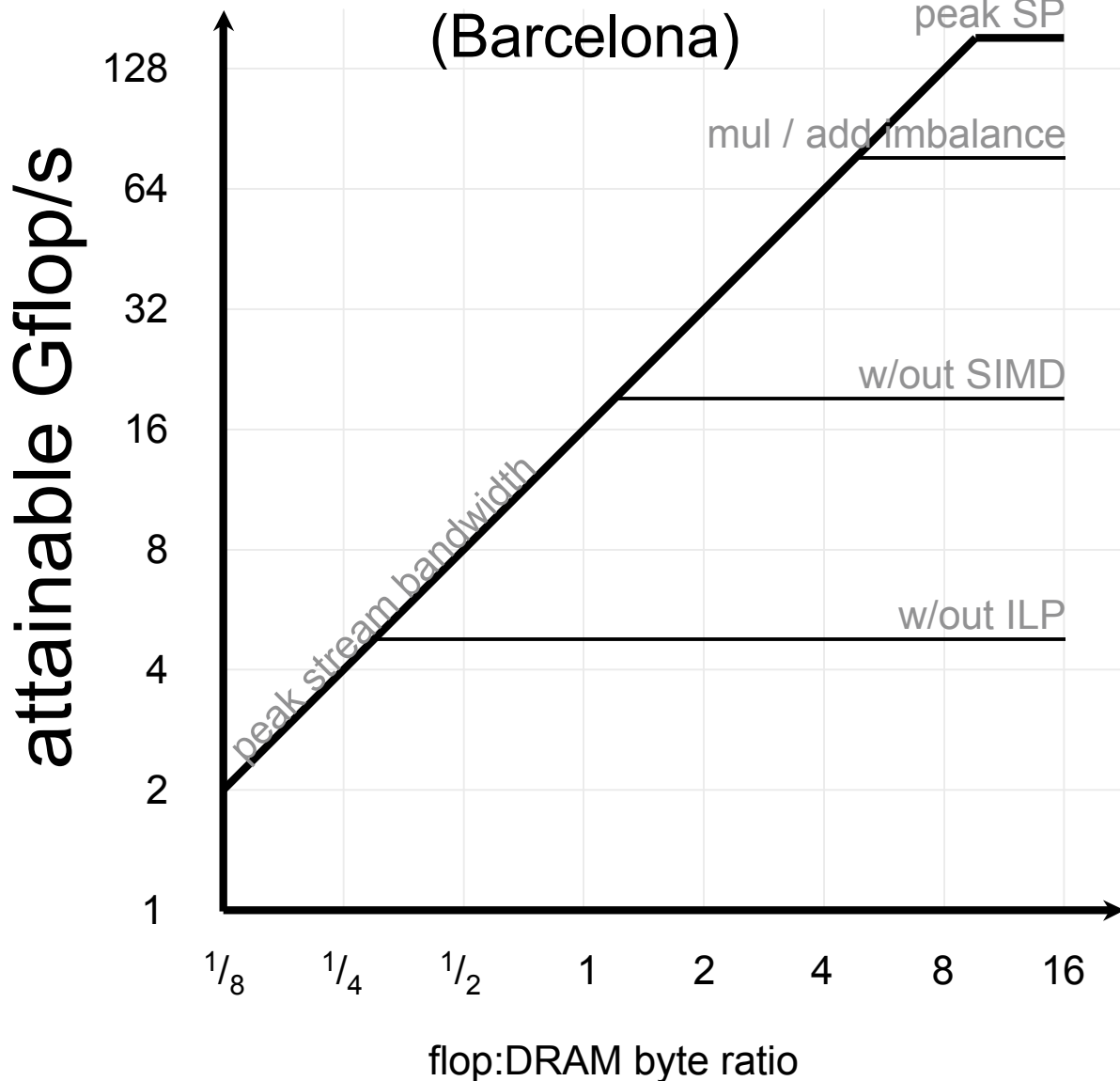


## AMD Opteron 2356 (Barcelona)



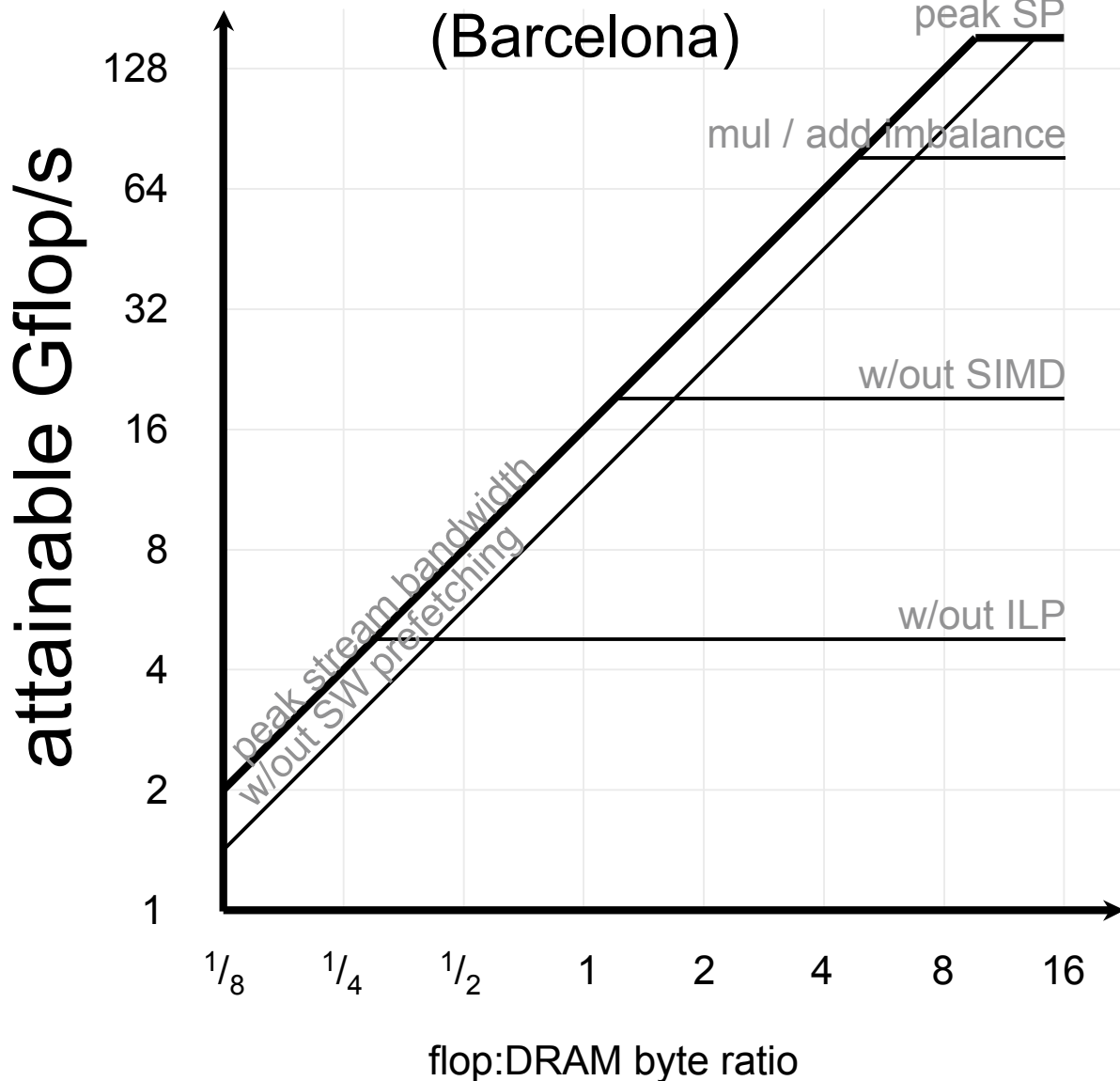
- ❖ In single precision, SIMD is 4x32b.
- ❖ If only the `_ss` versions are used, performance is 1/4

## AMD Opteron 2356 (Barcelona)



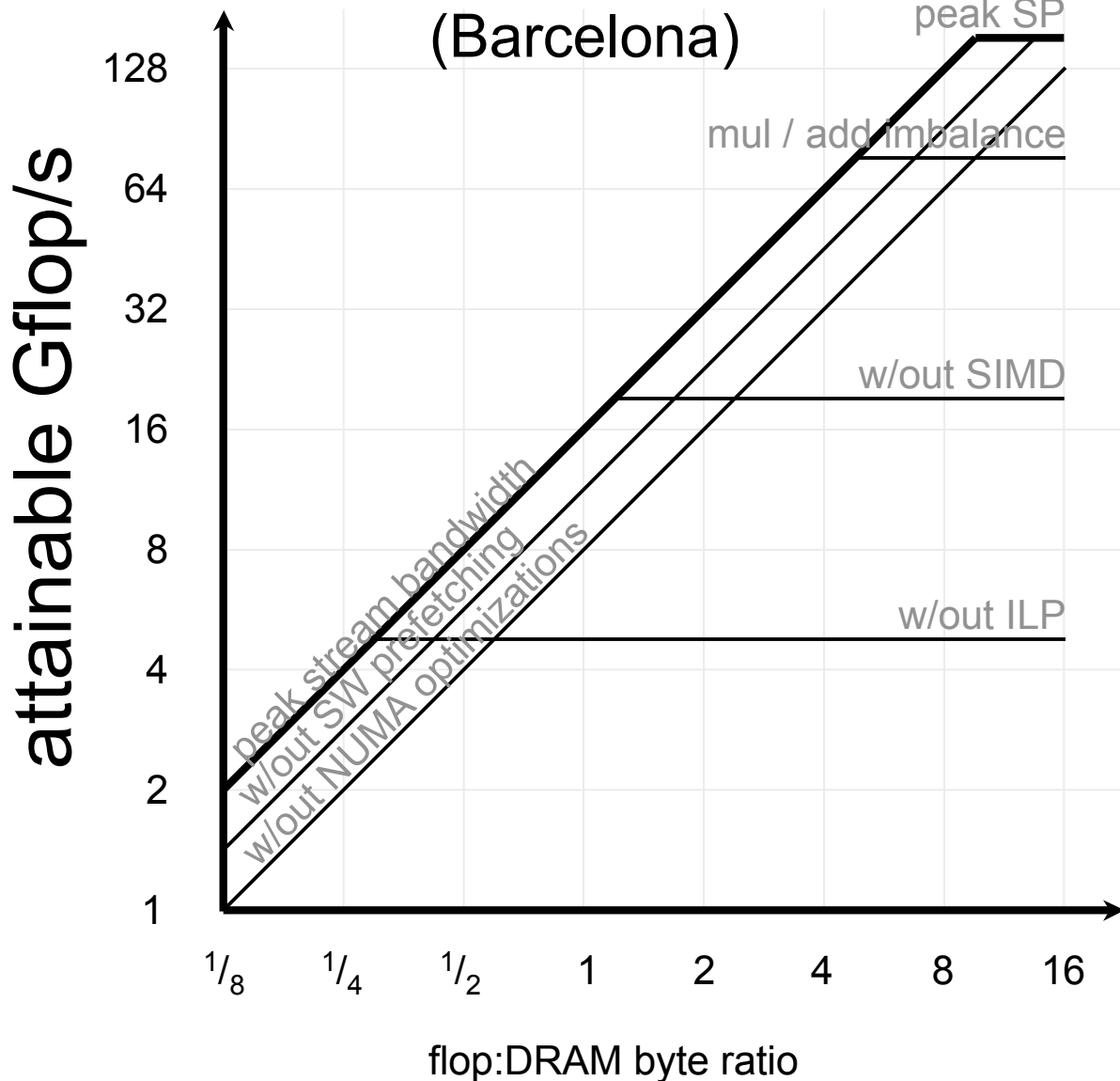
- ❖ If 4 independent instructions are kept in the pipeline, performance will fall

## AMD Opteron 2356 (Barcelona)



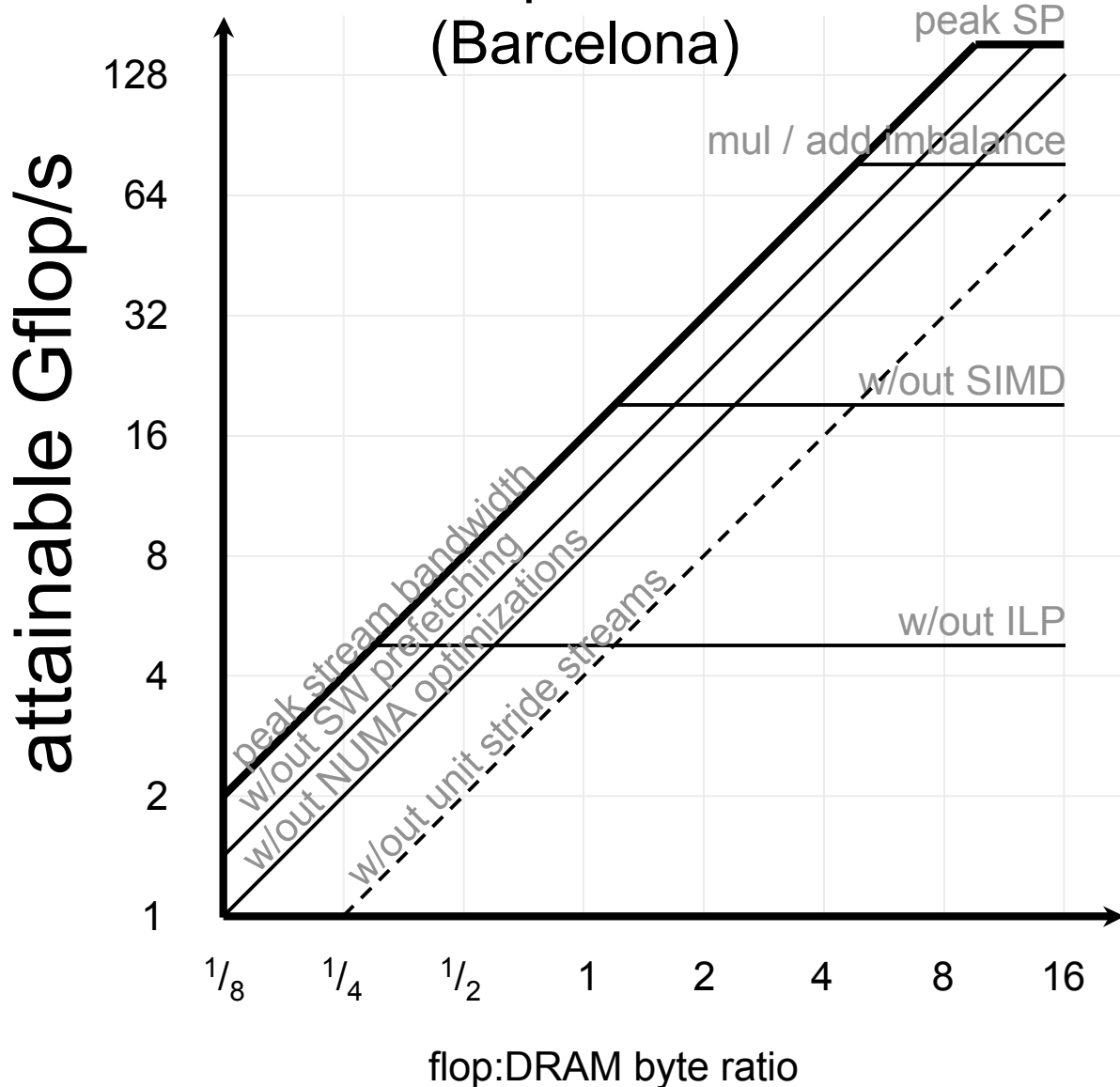
- ❖ If SW prefetching is not used, performance will degrade
- ❖ These act as ceilings below the bandwidth roofline

## AMD Opteron 2356 (Barcelona)



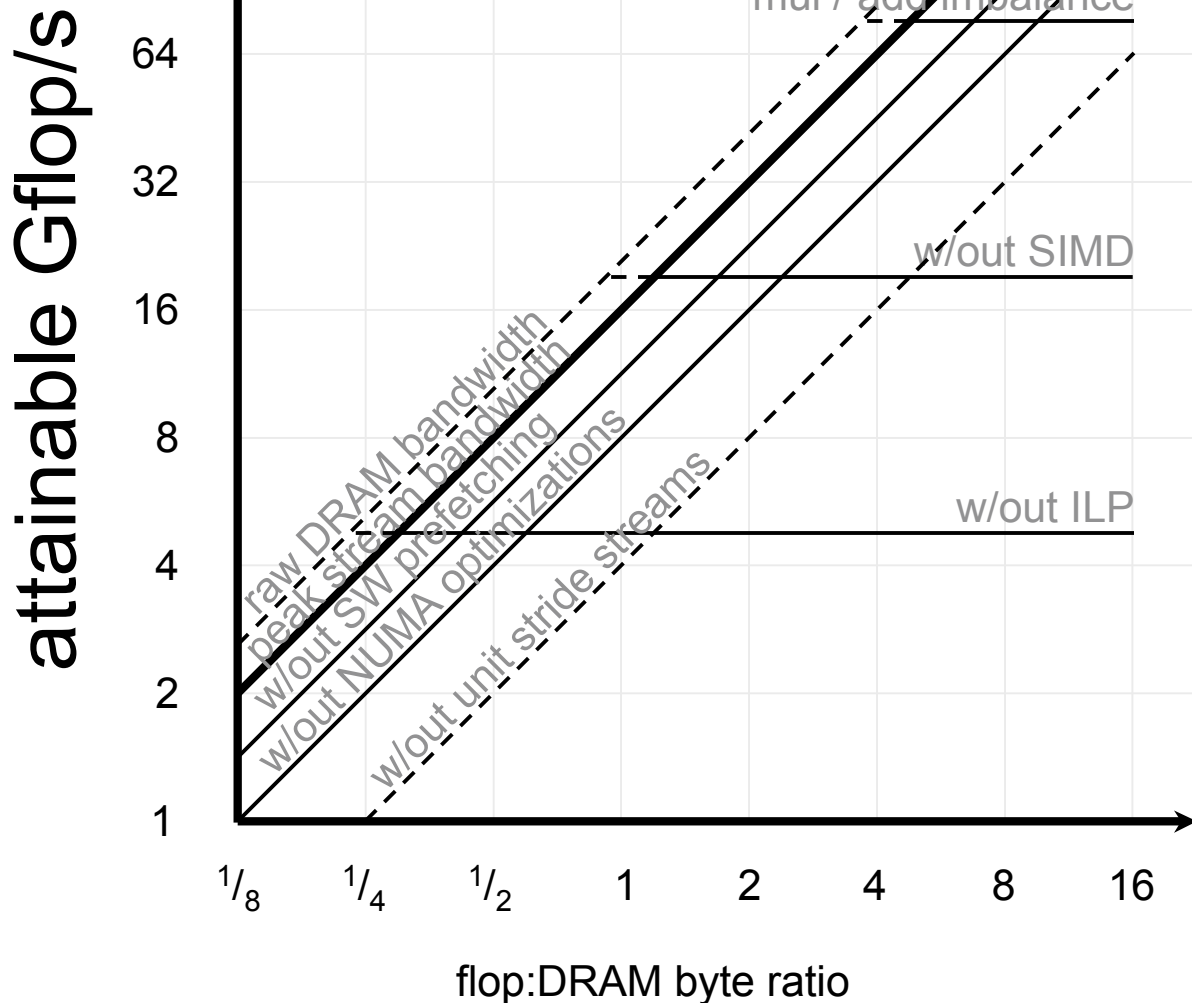
- ❖ Without NUMA optimizations, the memory controllers on the second socket can't be used.

## AMD Opteron 2356 (Barcelona)



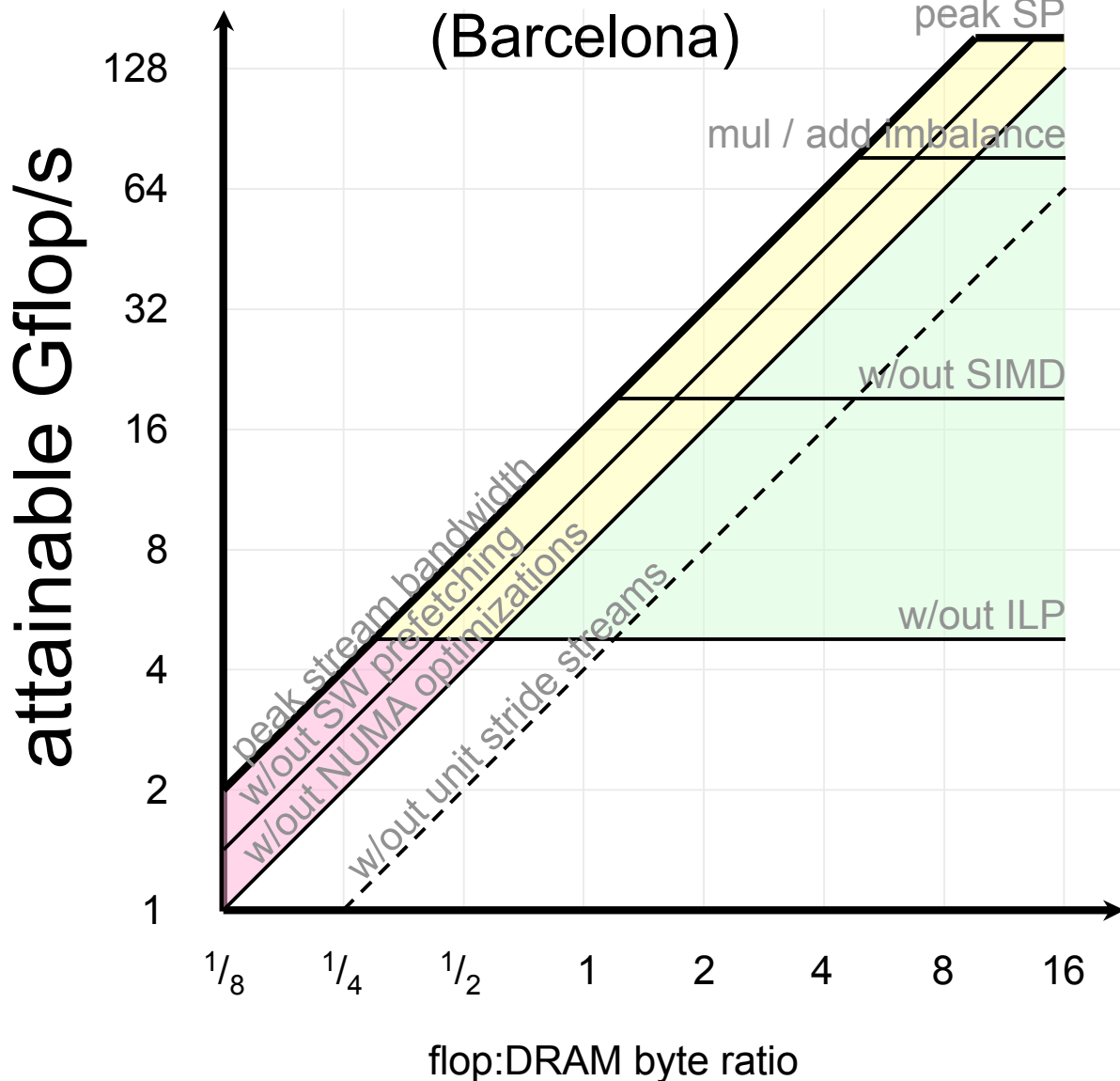
❖ Bandwidth is much lower without unit stride streams

## AMD Opteron 2356 (Barcelona)



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

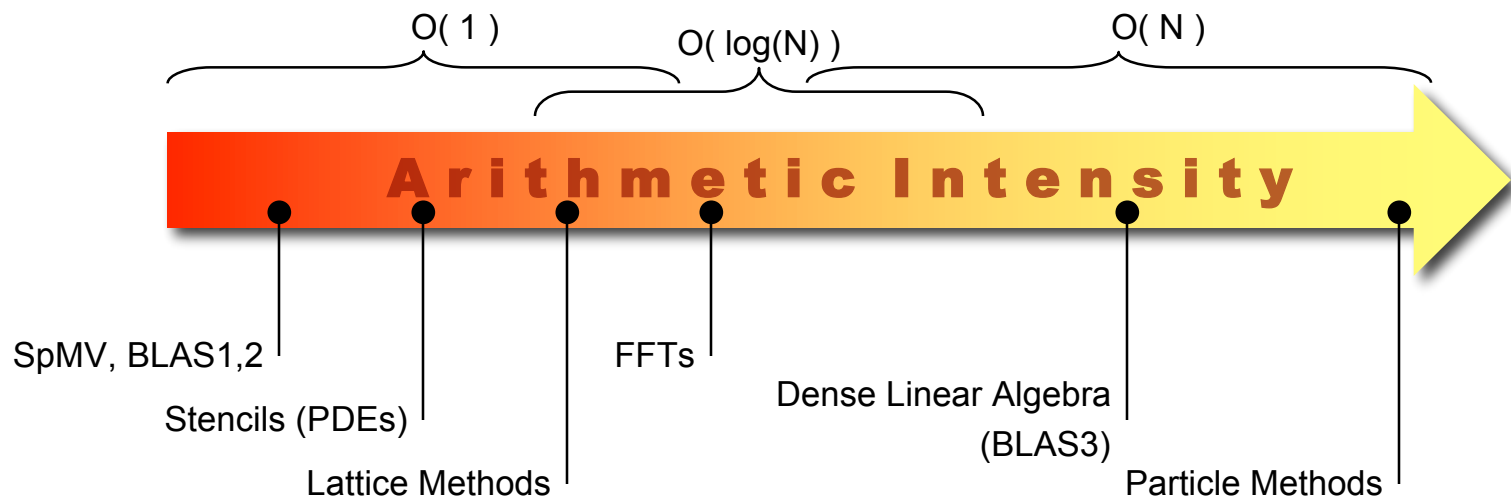
## AMD Opteron 2356 (Barcelona)



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

- ❖ 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.





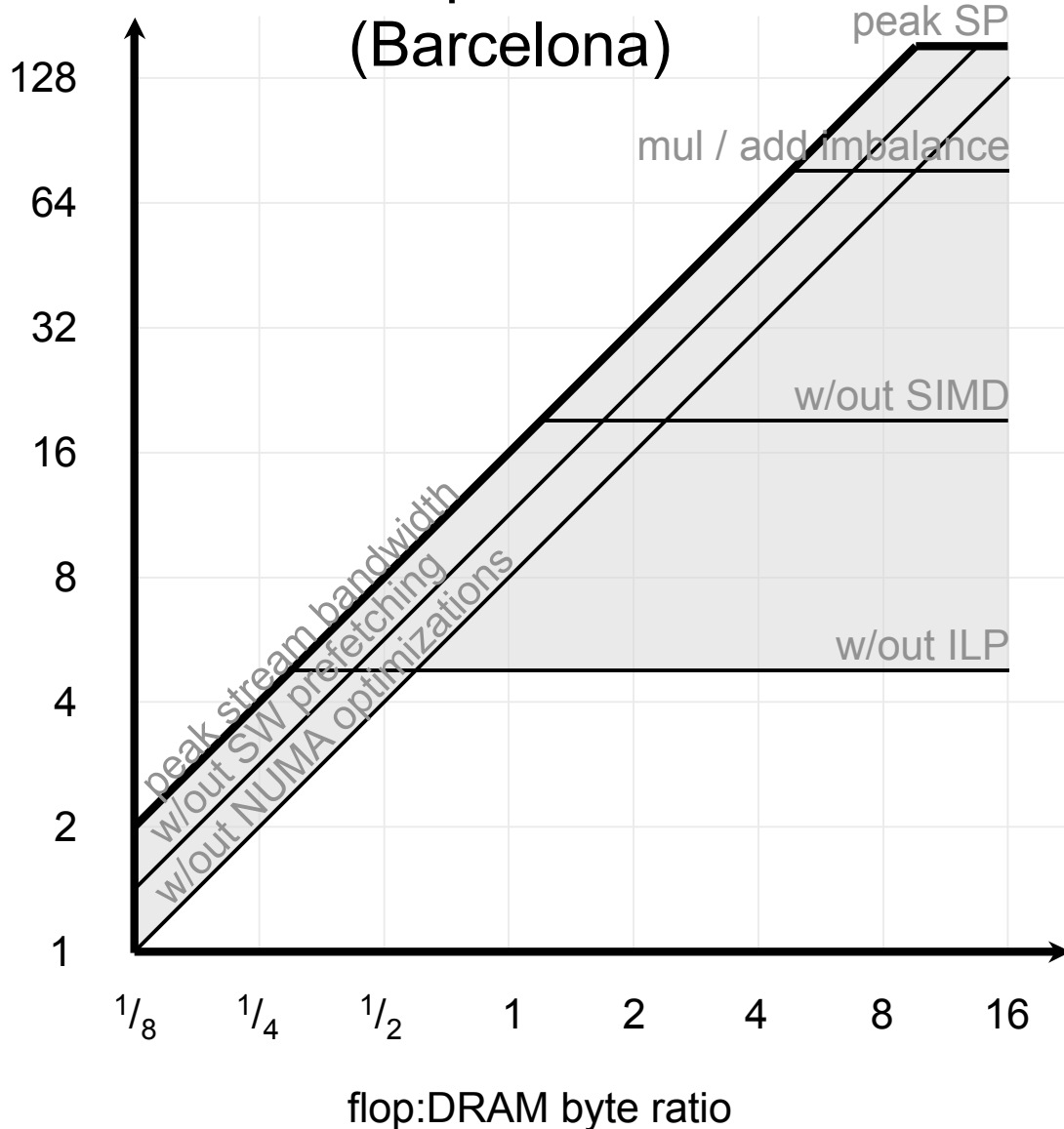
- ❖ **Arithmetic Intensity (AI) ~ Total Flops / Total DRAM Bytes**
- ❖ Some HPC kernels have an arithmetic intensity that's constant, but on others it scales with with problem size (increasing temporal locality)
- ❖ Actual arithmetic intensity is capped by cache/local store capacity

- ❖ 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
  
- ❖  $\text{Flop:actual DRAM Byte ratio} < \text{Flop:compulsory DRAM Byte ratio}$
  
- ❖ 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

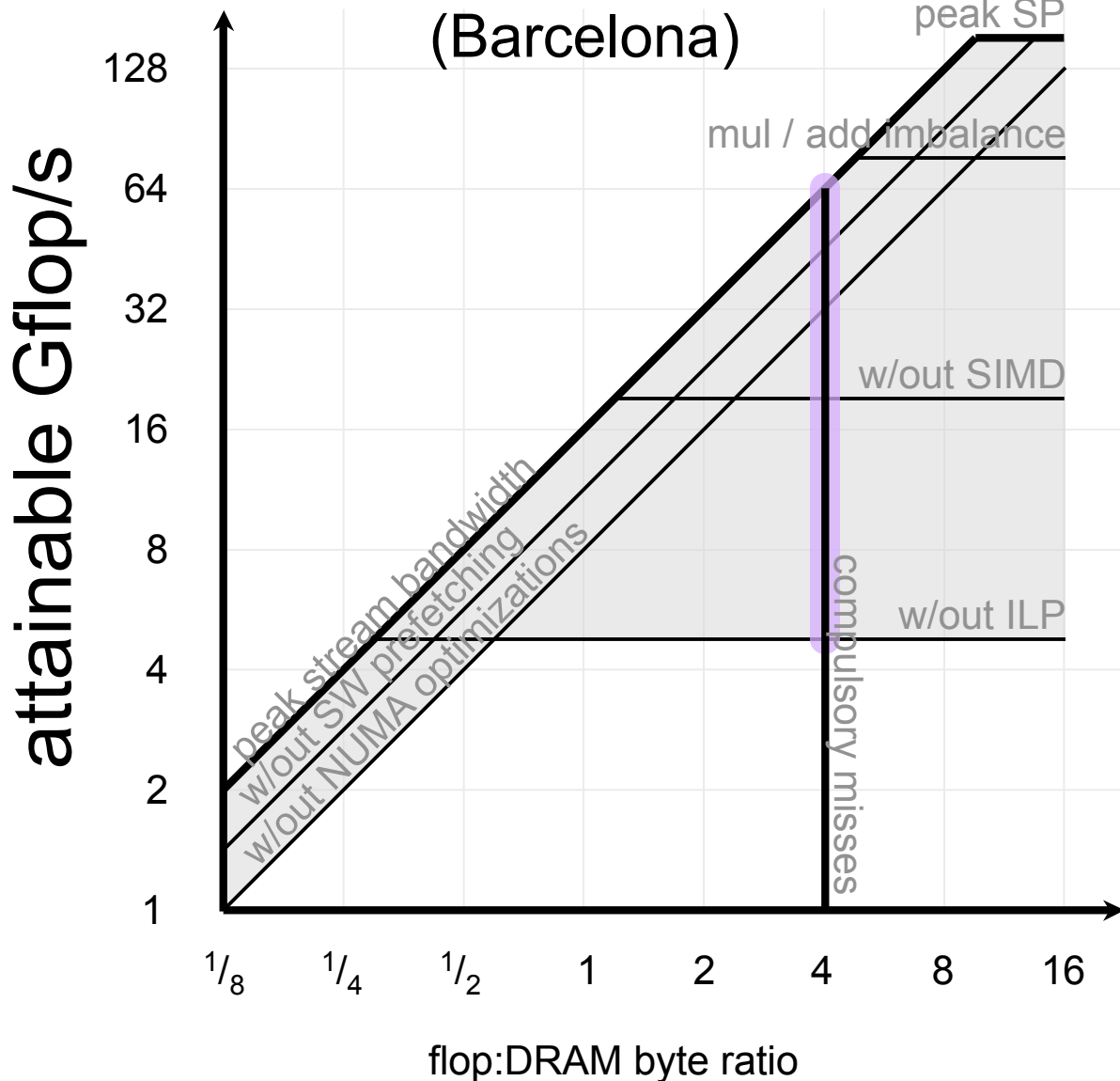
## AMD Opteron 2356 (Barcelona)

❖ Final Roofline

attainable Gflop/s

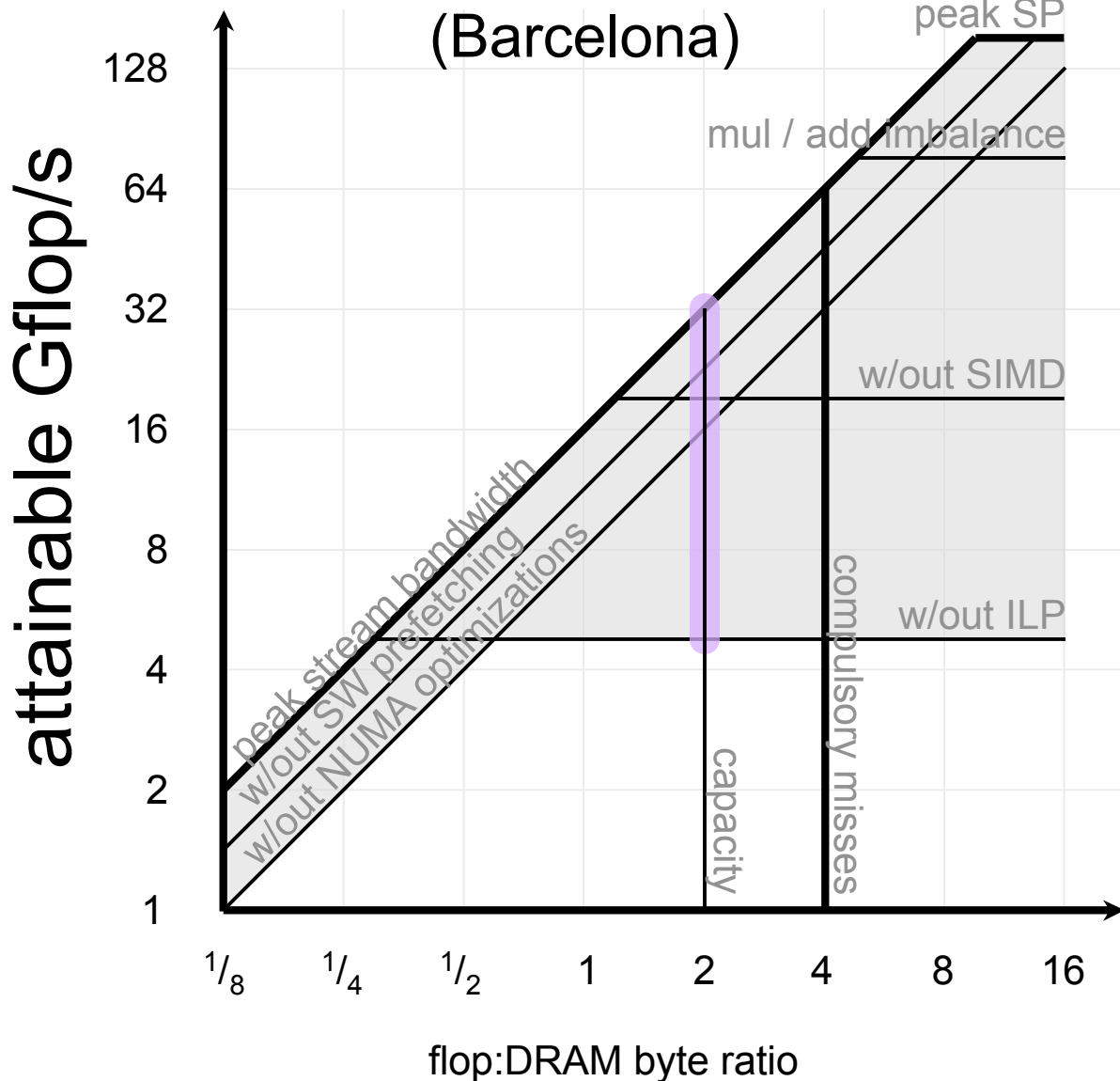


## AMD Opteron 2356 (Barcelona)



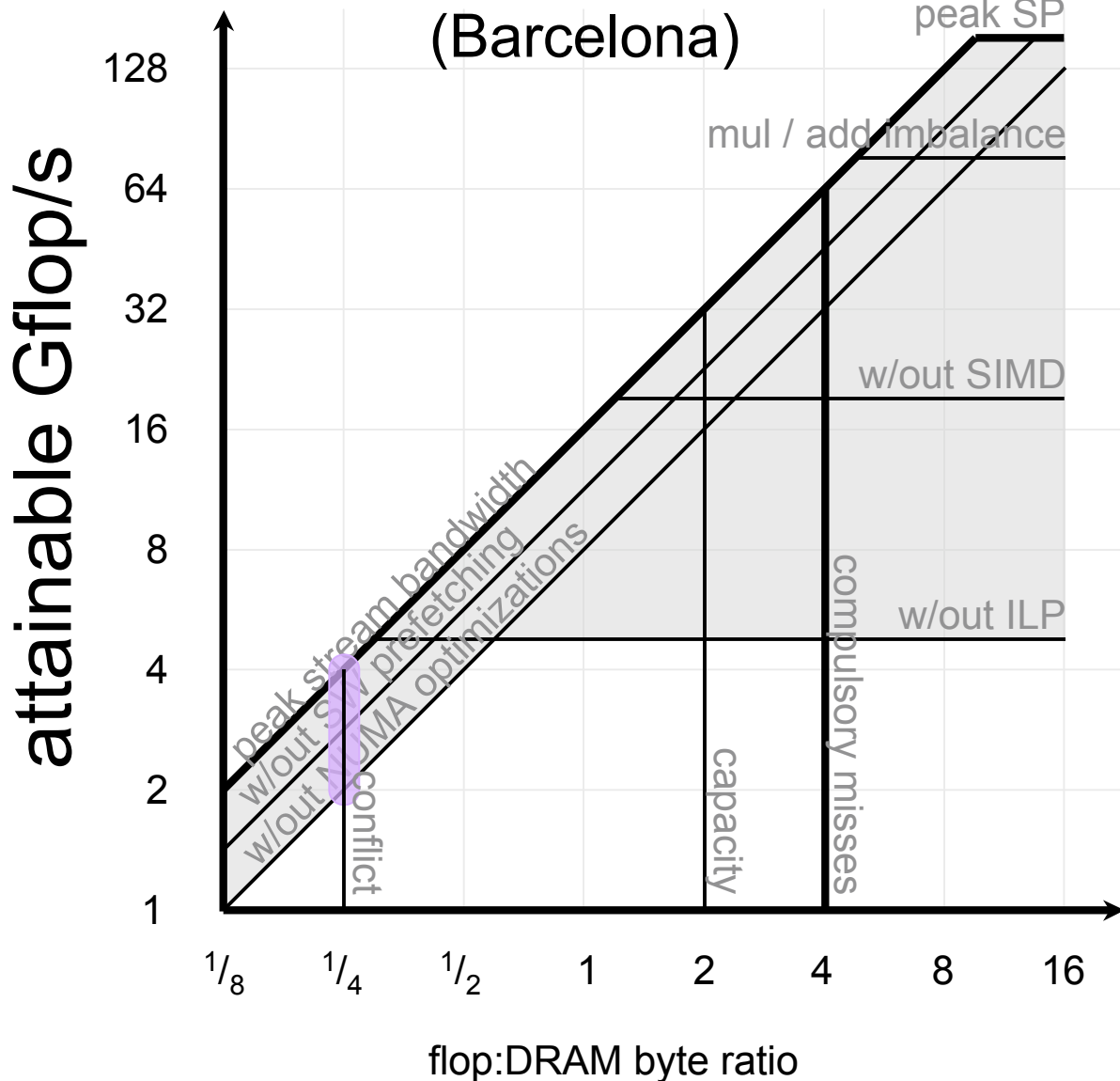
- ❖ Some arbitrary kernel has a flop:compulsory byte ratio of 4
- ❖ Overlaid on the roofline
- ❖ Defines upper bound on range of expected performance
- ❖ Also shows which optimizations are likely

## AMD Opteron 2356 (Barcelona)



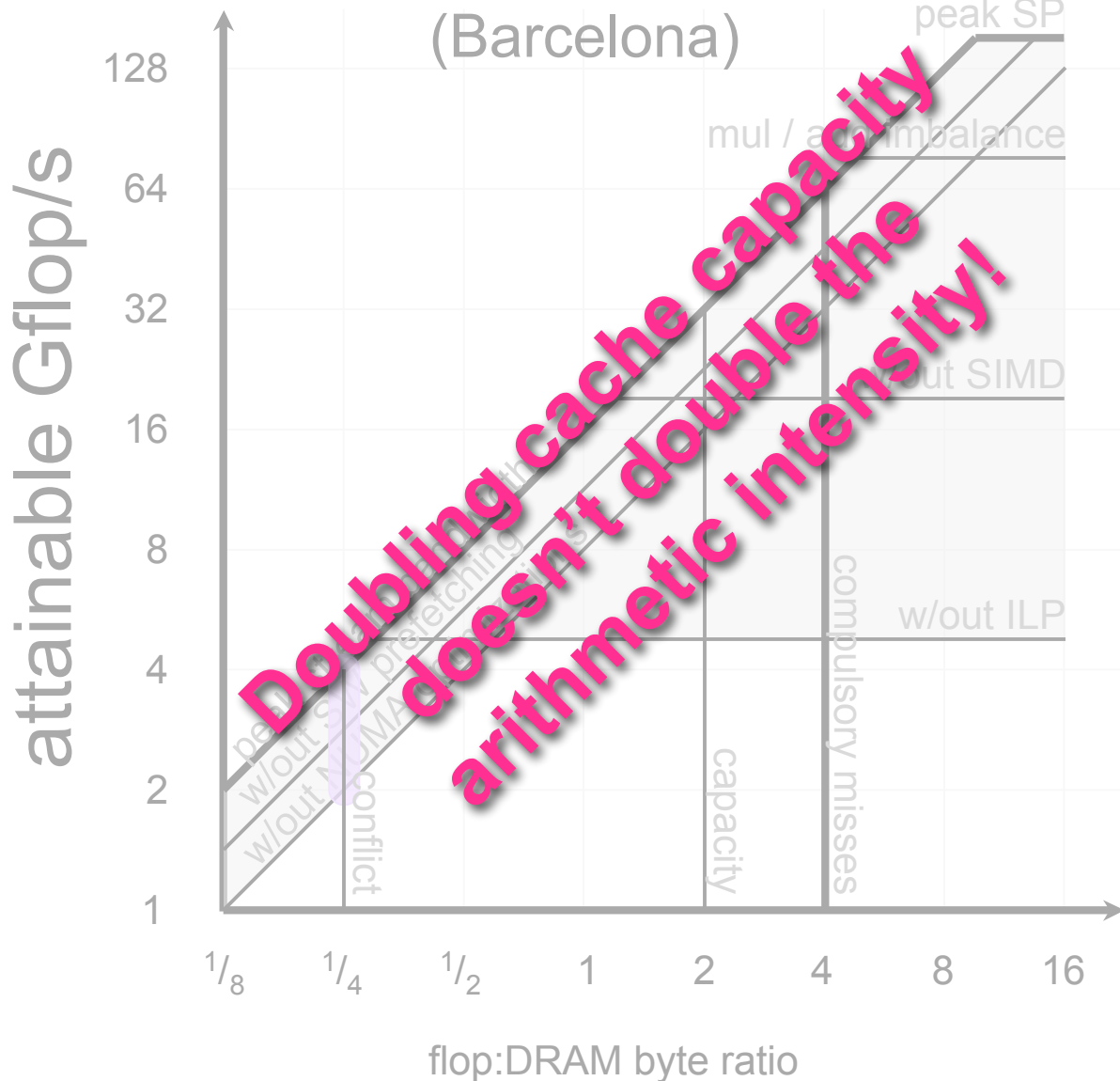
- ❖ Capacity misses reduce the actual flop:byte ratio
- ❖ Also reduces attainable performance
- ❖ **AI is unique to each combination of kernel and architecture**

## AMD Opteron 2356 (Barcelona)



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

AMD Opteron 2356  
(Barcelona)



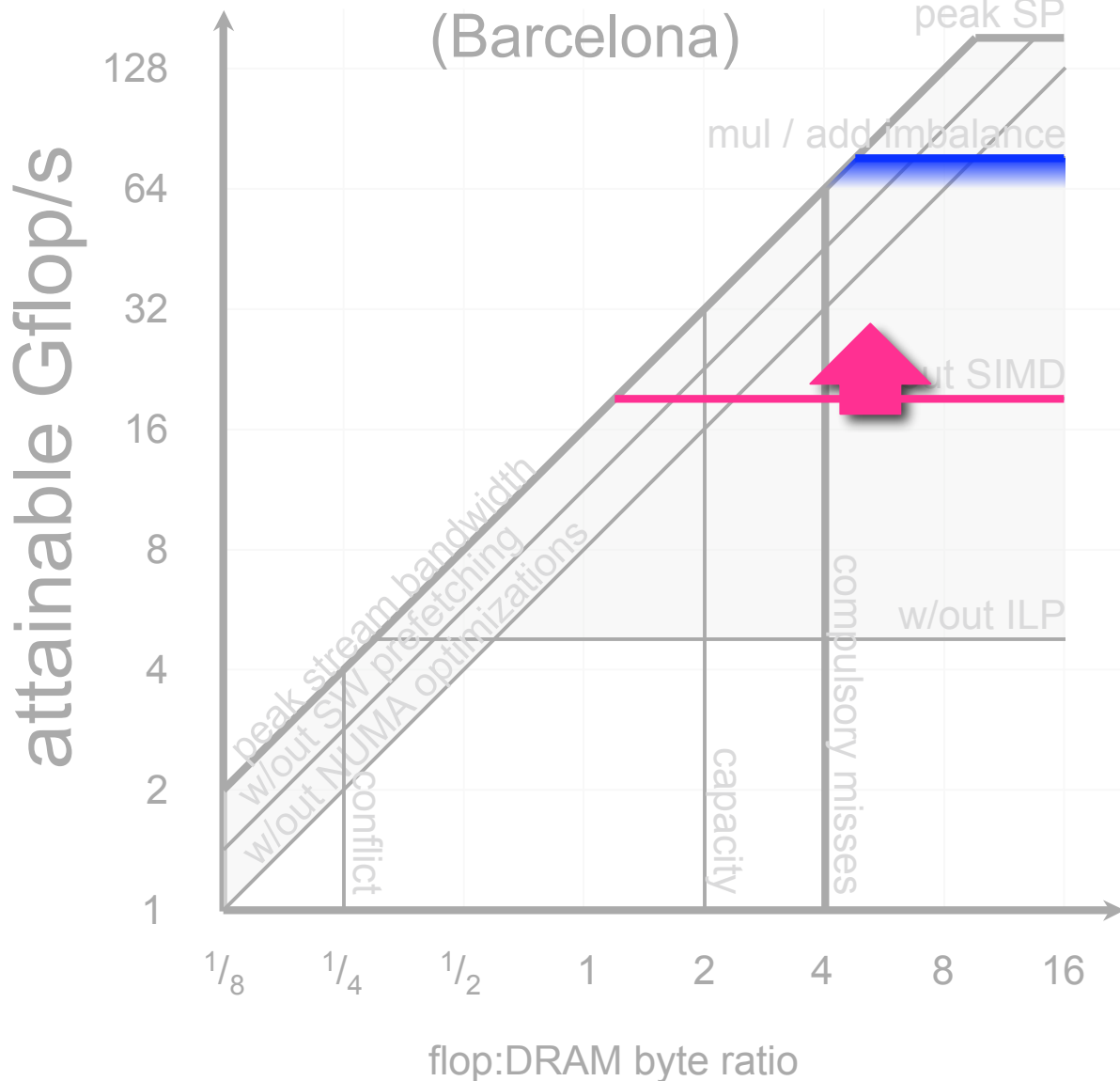
❖ Conflict misses may destroy performance

# Three Categories of Software Optimization



# Maximizing Attained in-core Performance

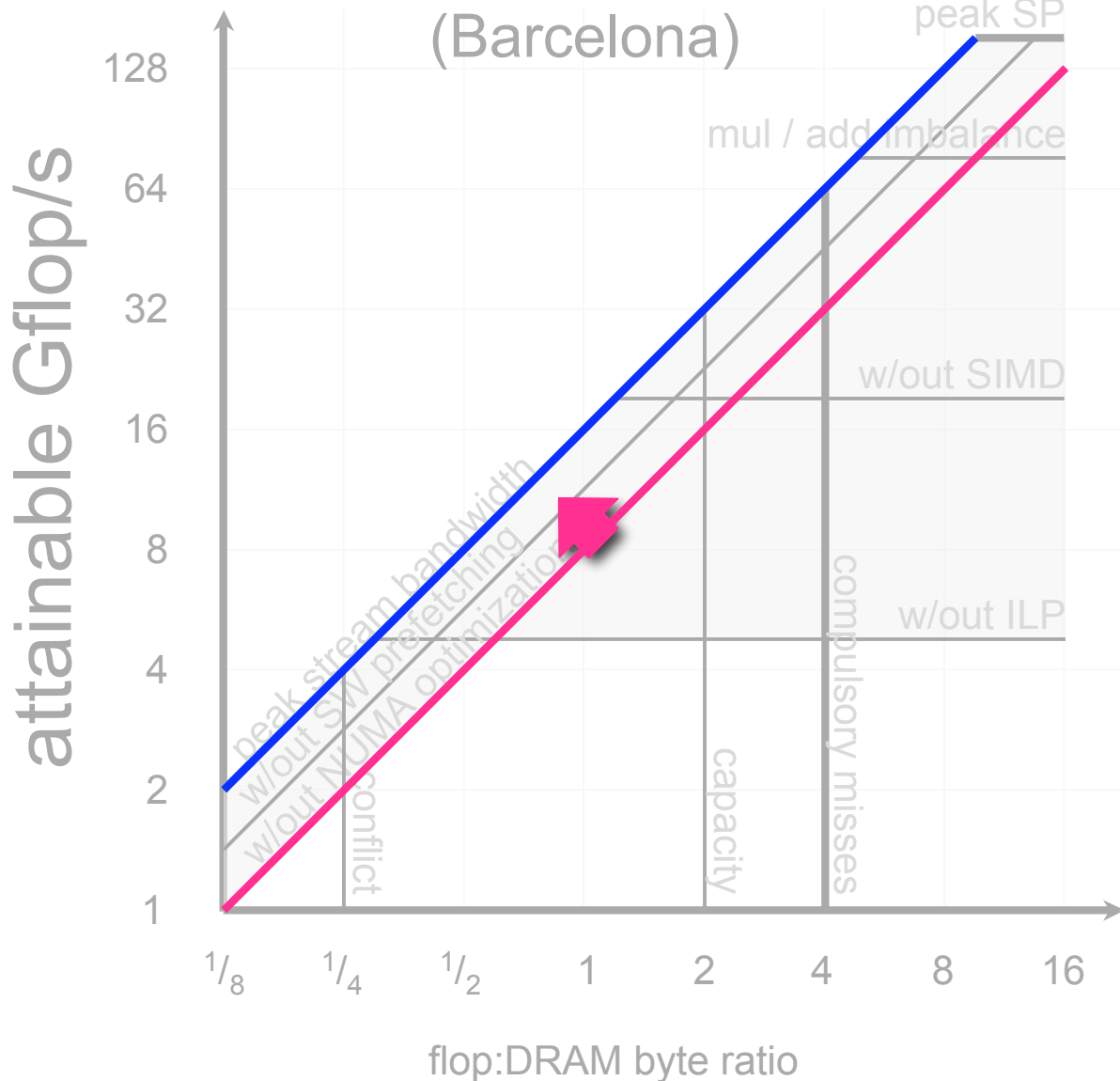
AMD Opteron 2356  
(Barcelona)



- ❖ 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

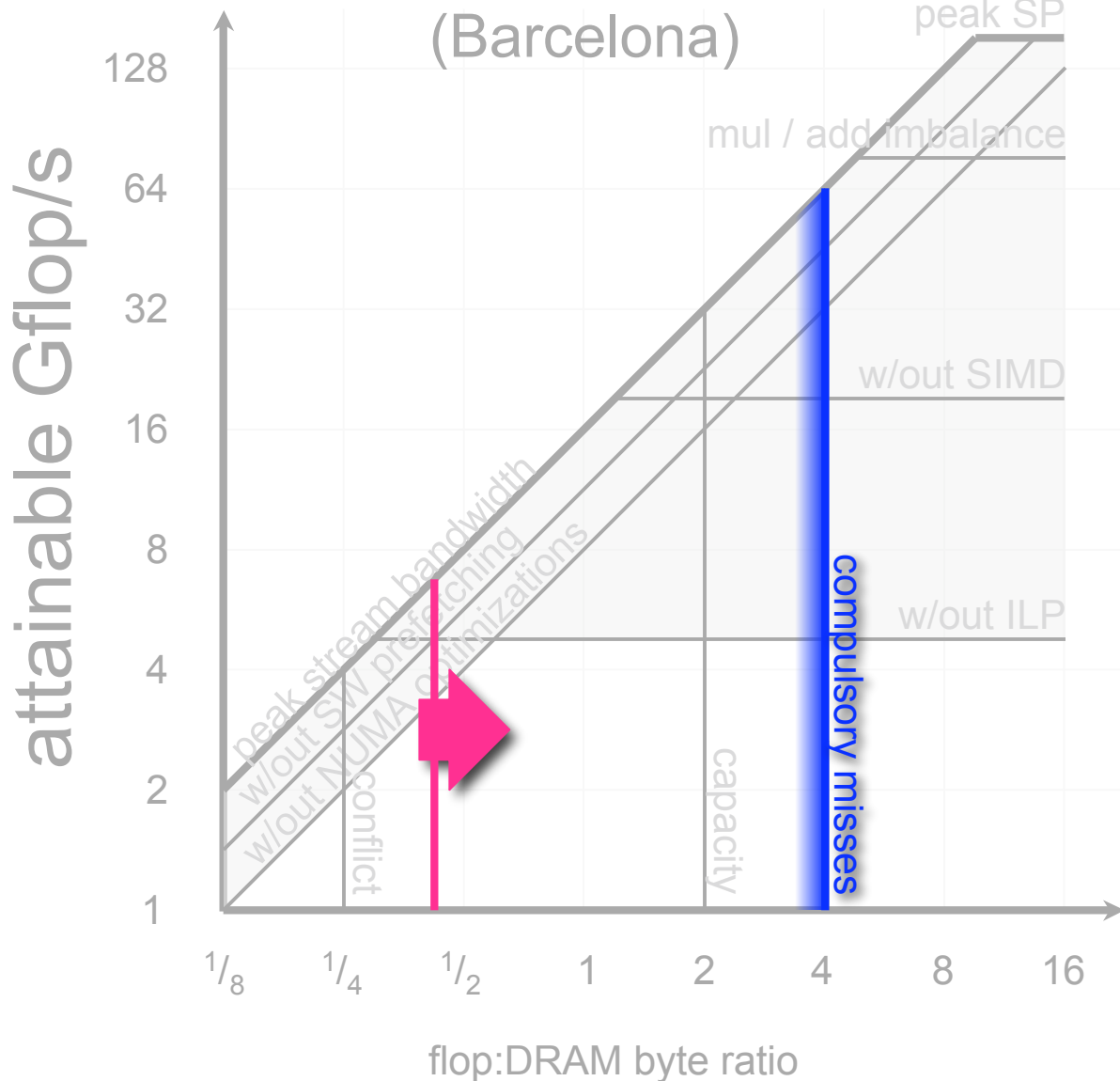
# Maximizing Attained Memory Bandwidth

AMD Opteron 2356  
(Barcelona)



- ❖ 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

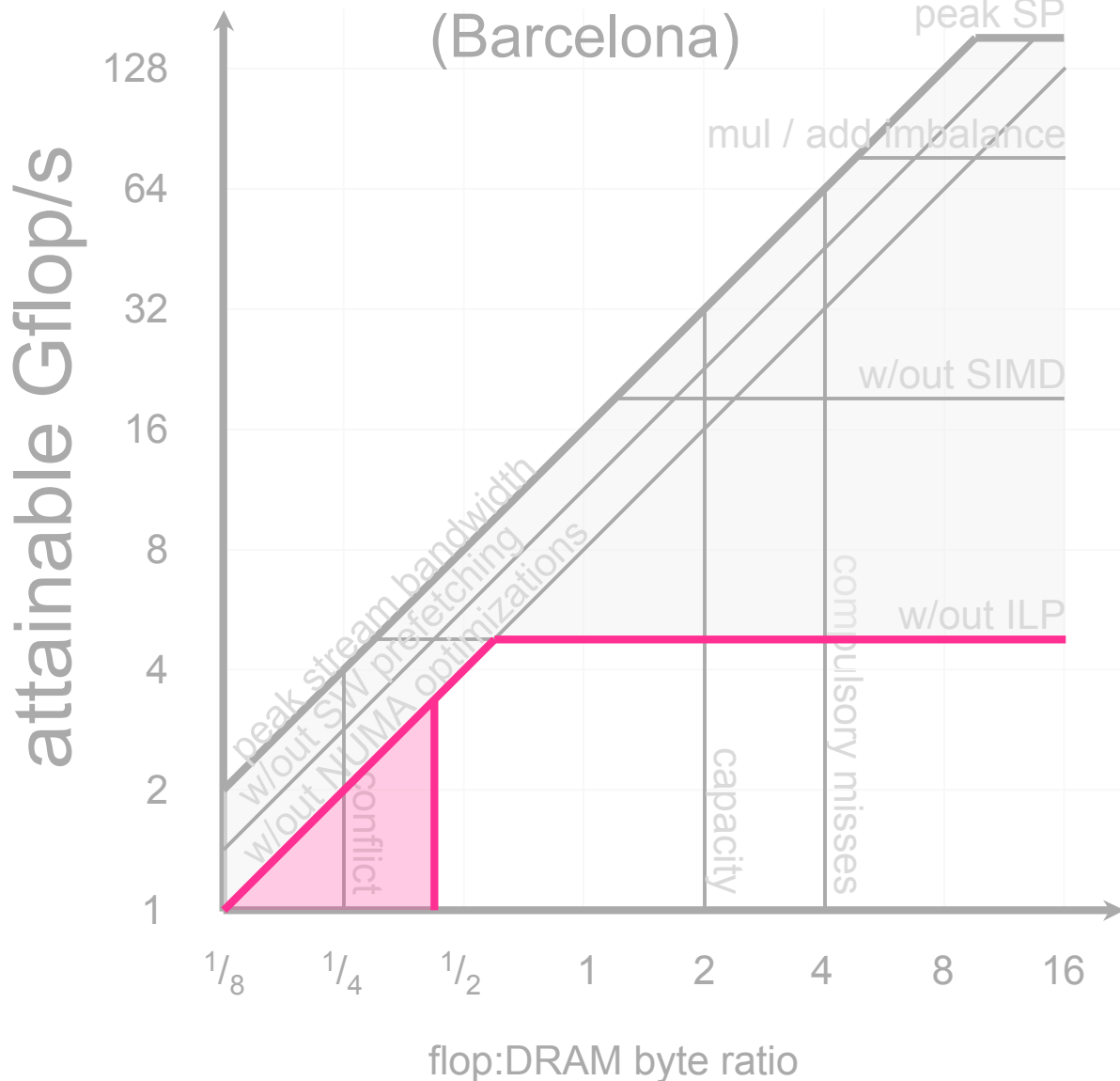
AMD Opteron 2356  
(Barcelona)



- ❖ 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**

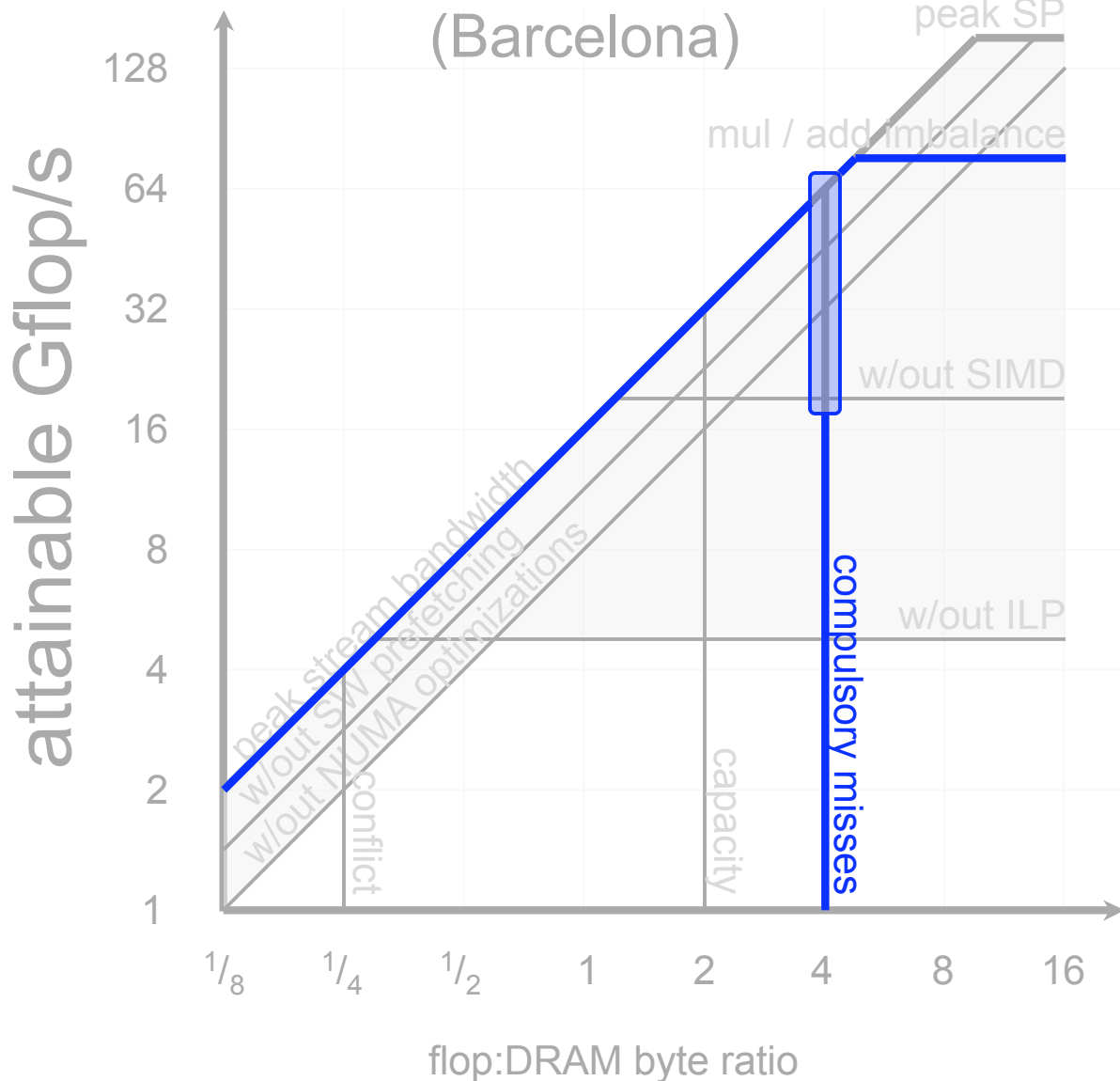
# Effective Roofline (before)

AMD Opteron 2356  
(Barcelona)



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

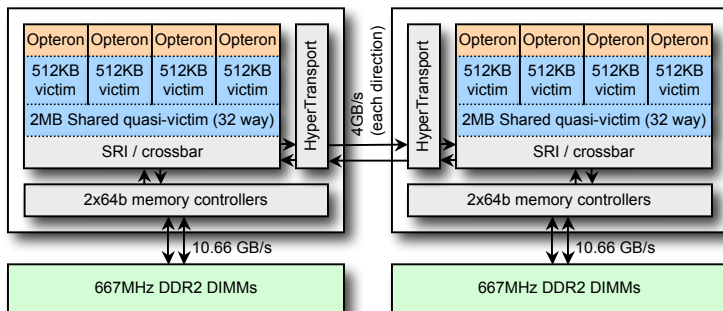
AMD Opteron 2356  
(Barcelona)



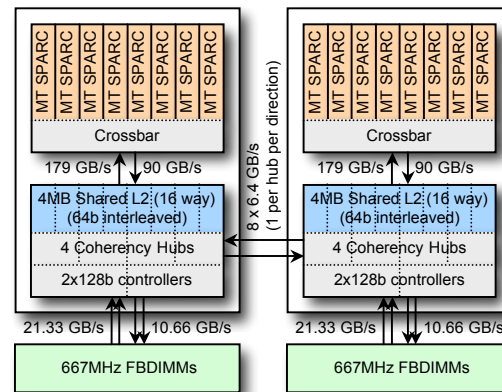
❖ After optimization, ideally, performance is significantly better

# Applicable to Other Architectural Paradigms ?

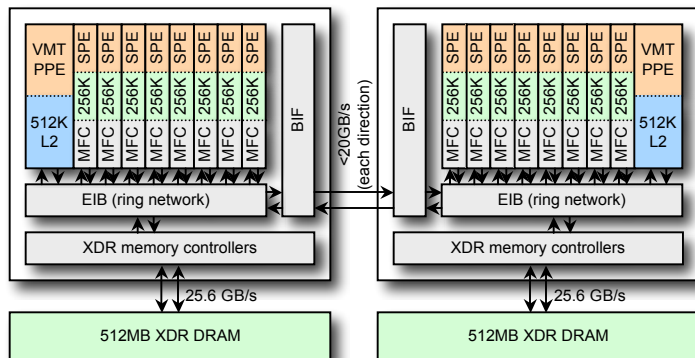
## AMD Barcelona



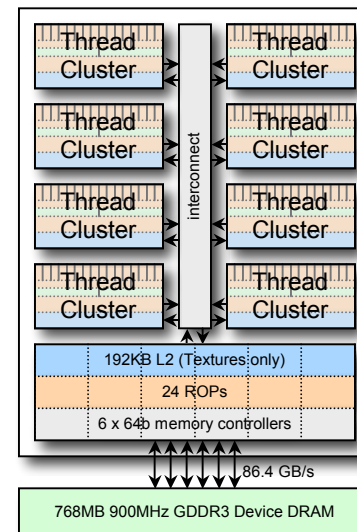
## Sun Victoria Falls



## IBM Cell Blade

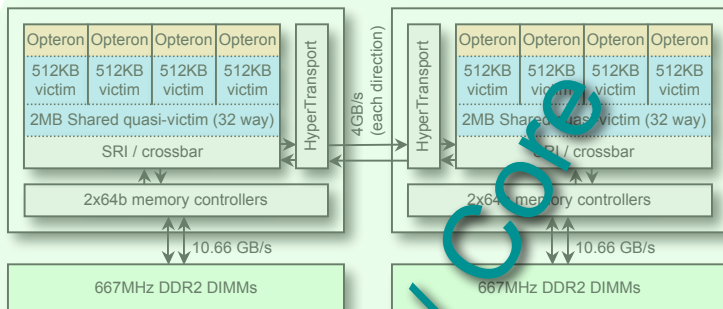


## NVIDIA G80

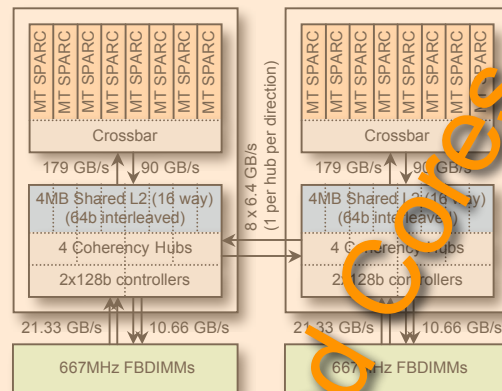


# Four Architectures

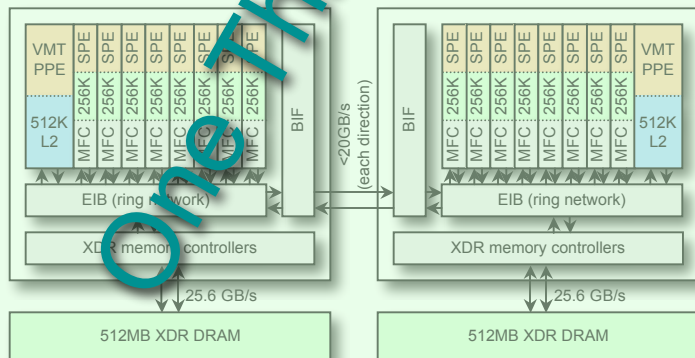
## AMD Barcelona



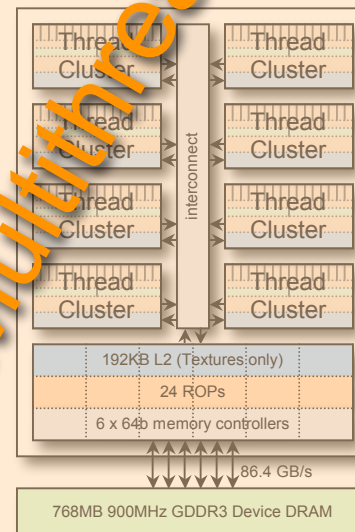
## Sun Victoria Falls



## IBM Cell Blade



## NVIDIA G80



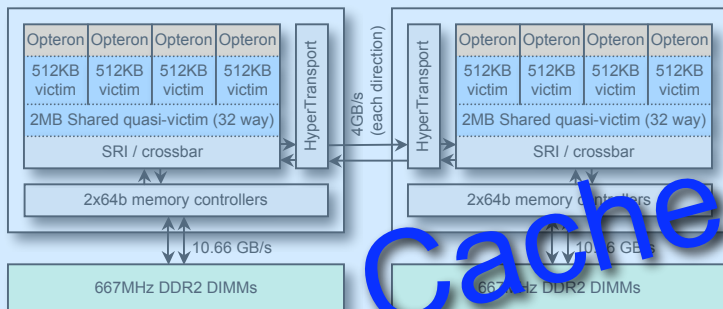
One Thread / Core

Multi-threaded Cores

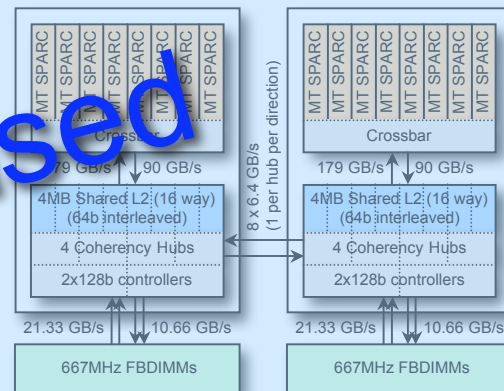


# Four Architectures

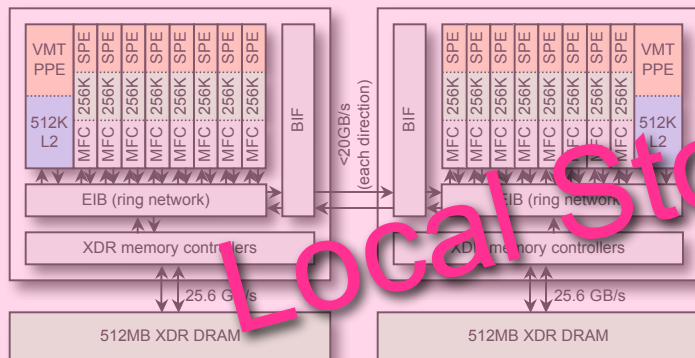
## AMD Barcelona



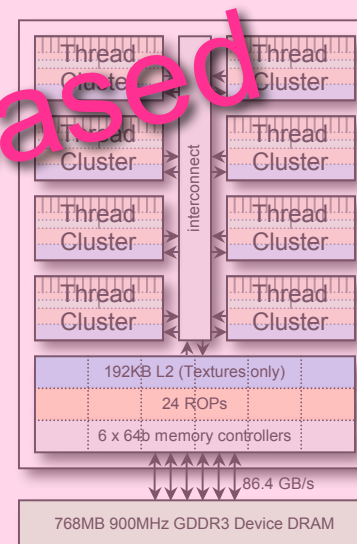
## Sun Victoria Falls



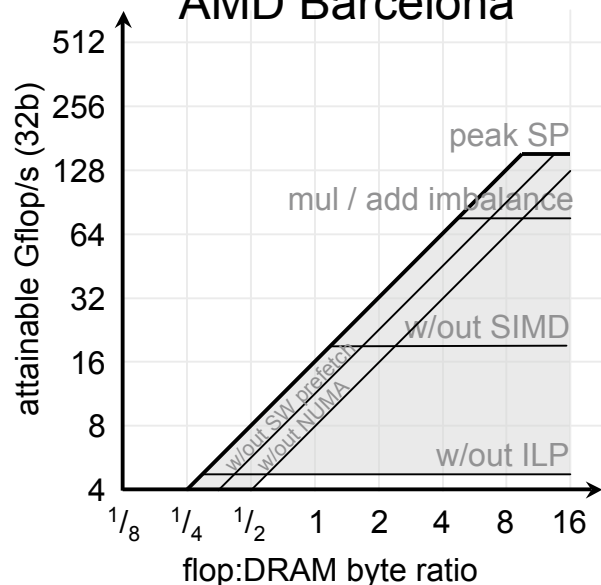
## IBM Cell Blade



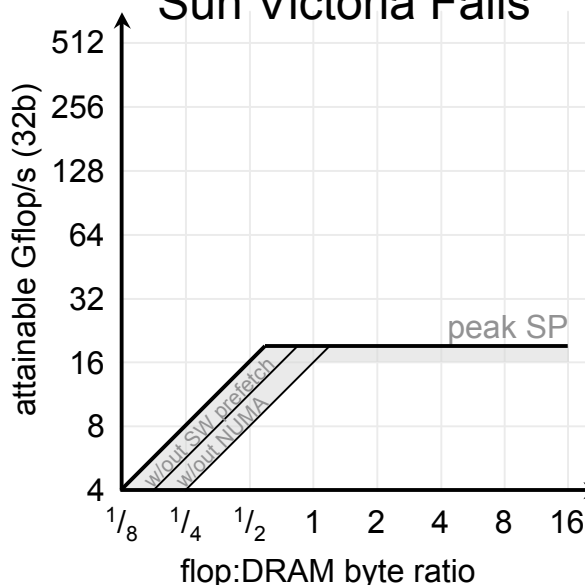
## NVIDIA G80



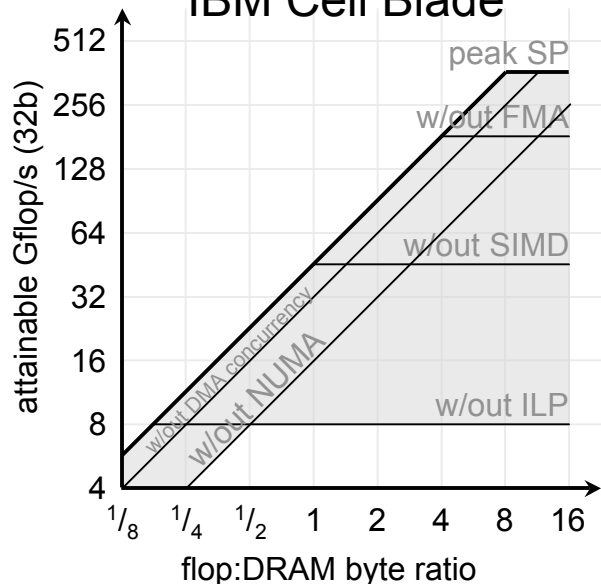
### AMD Barcelona



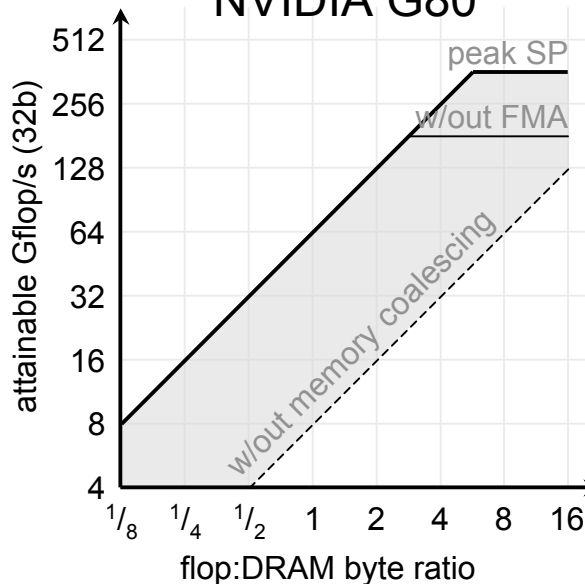
### Sun Victoria Falls



### IBM Cell Blade



### NVIDIA G80

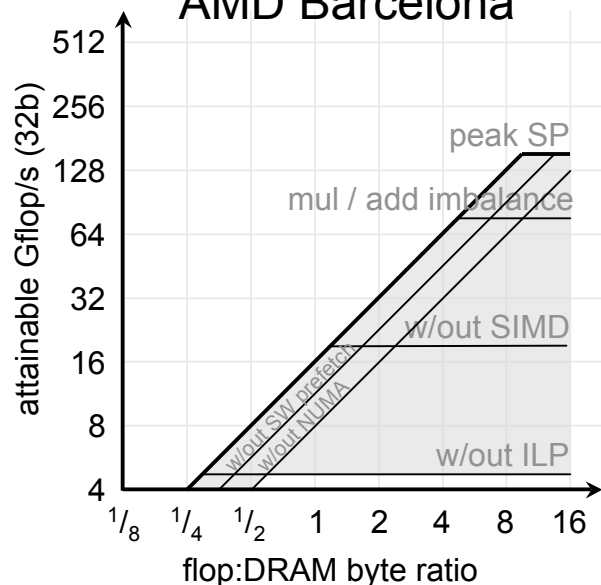


- ❖ Single Precision Roofline models for the SMPs used in this work.
- ❖ Based on micro-benchmarks, experience, and manuals
- ❖ Ceilings = in-core parallelism

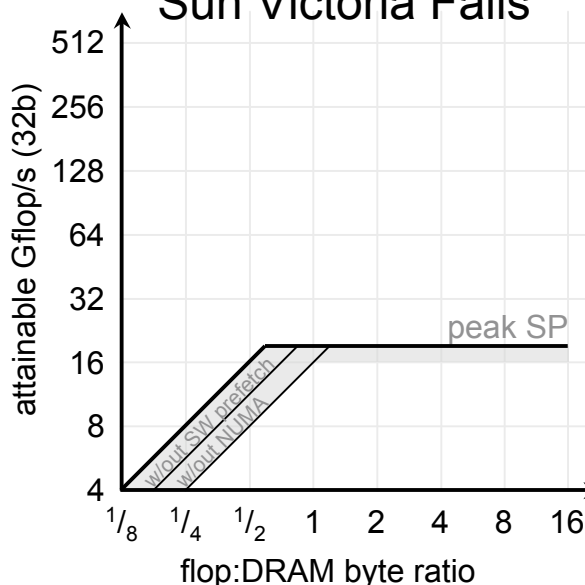
**Can the compiler find all this parallelism ?**

- ❖ NOTE:
  - log-log scale
  - Assumes perfect SPMD

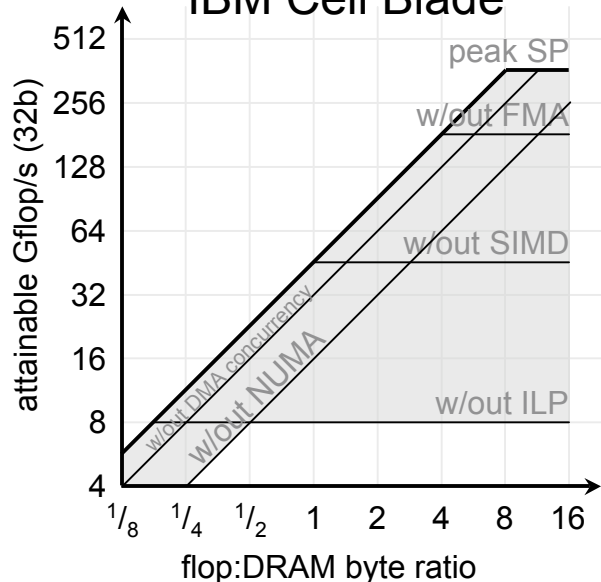
### AMD Barcelona



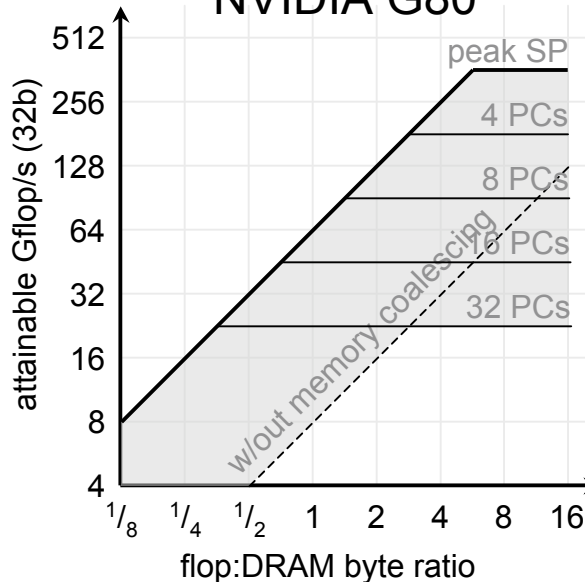
### Sun Victoria Falls



### IBM Cell Blade

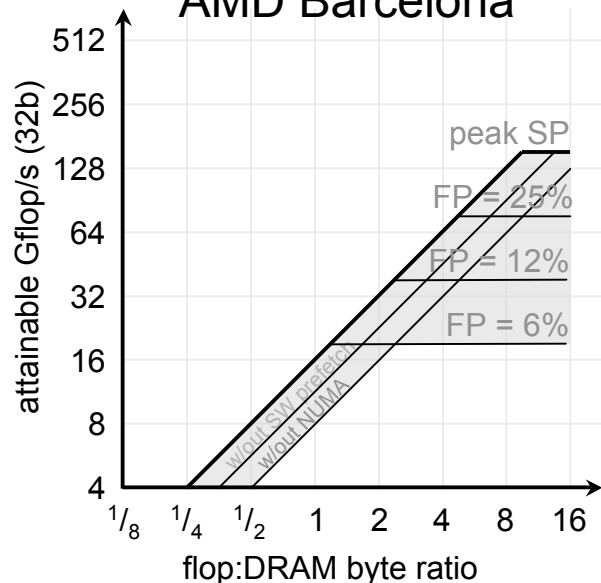


### NVIDIA G80

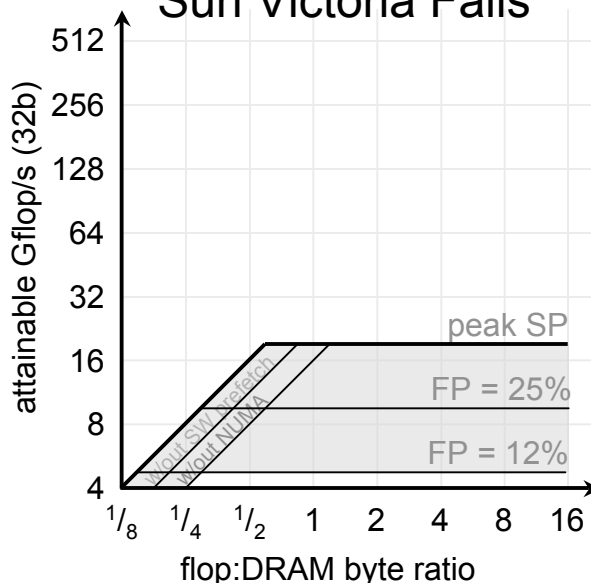


- ❖ G80 dynamically finds DLP (shared instruction fetch)
- ❖ **SIMT**
- ❖ If threads of a warp diverge from SIMD execution, performance is limited by instruction issue bandwidth
- ❖ Ceilings on G80 = number of unique PCs when threads diverge

### AMD Barcelona

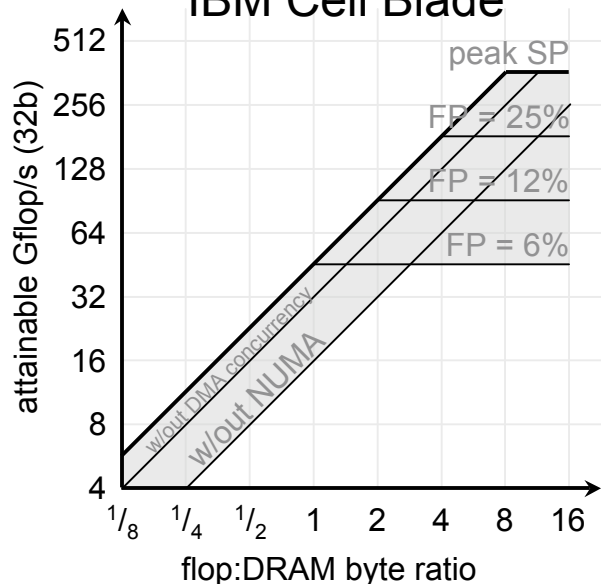


### Sun Victoria Falls

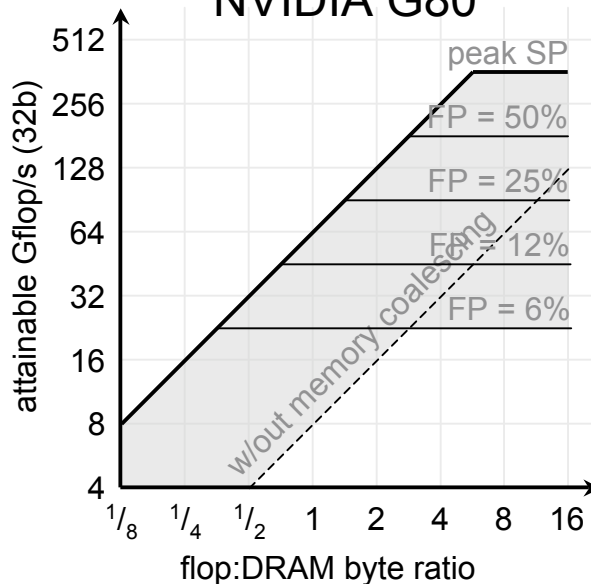


- ❖ Some kernels have large numbers of non FP instructions
- ❖ Saps instruction issue bandwidth
- ❖ Ceilings = FP fraction of dynamic instruction mix
- ❖ NOTE:
  - Assumes perfect in-core parallelism

### IBM Cell Blade



### NVIDIA G80

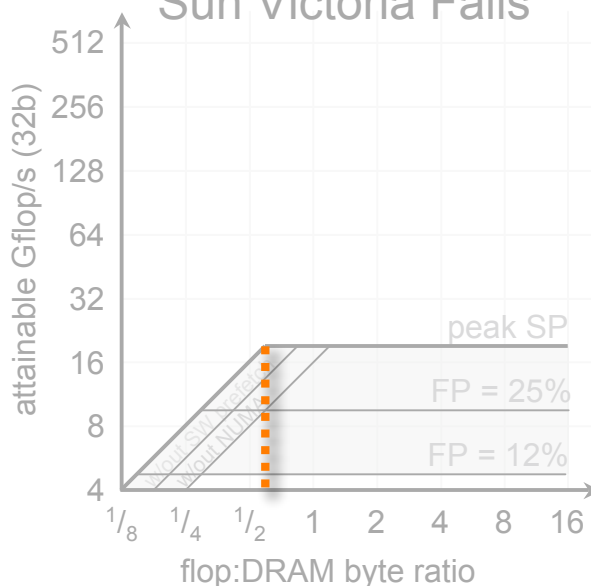
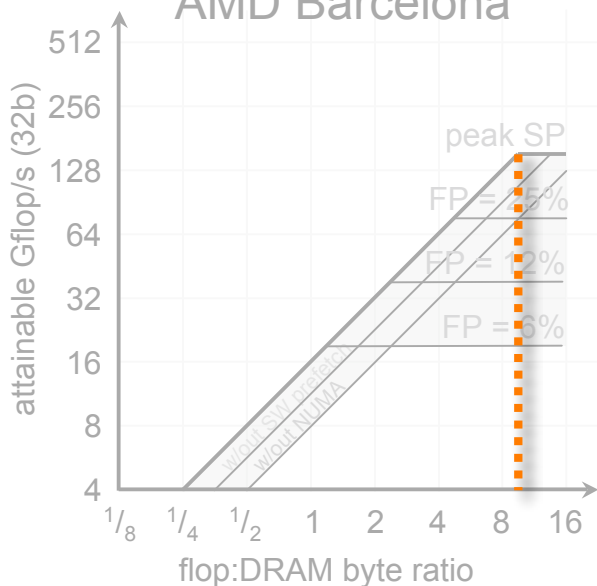


# 32b Rooflines for the Four (ridge point)

AMD Barcelona

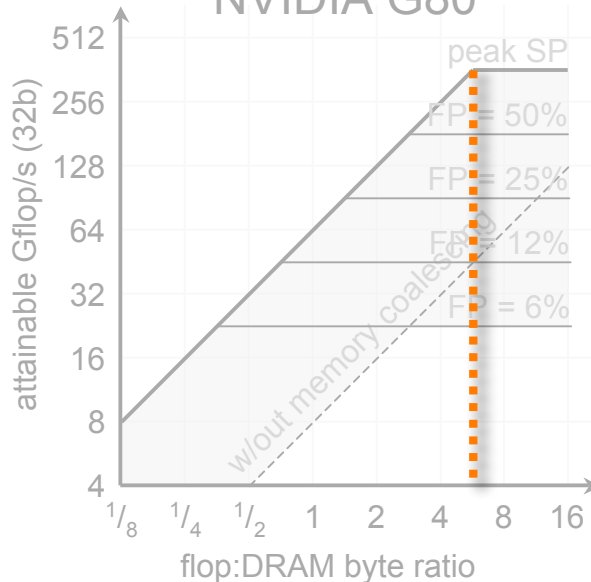
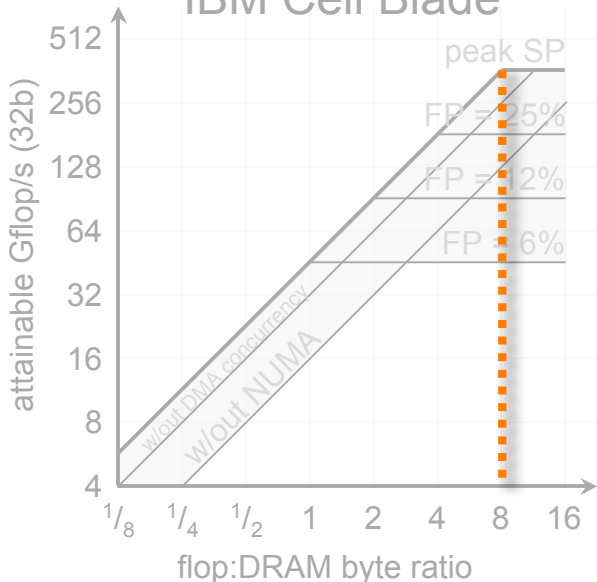
Sun Victoria Falls

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



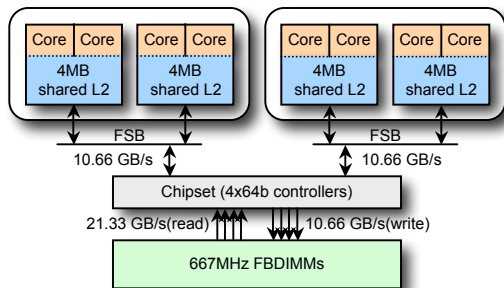
IBM Cell Blade

NVIDIA G80

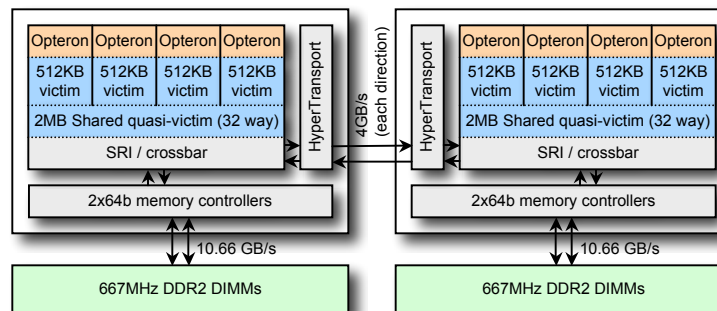


# Using Roofline when Auto-tuning HPC Kernels

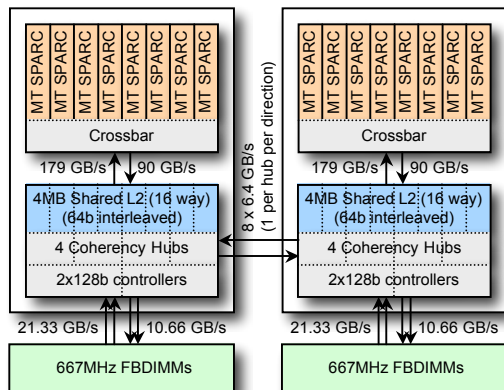
## Intel Xeon E5345 (Clovertown)



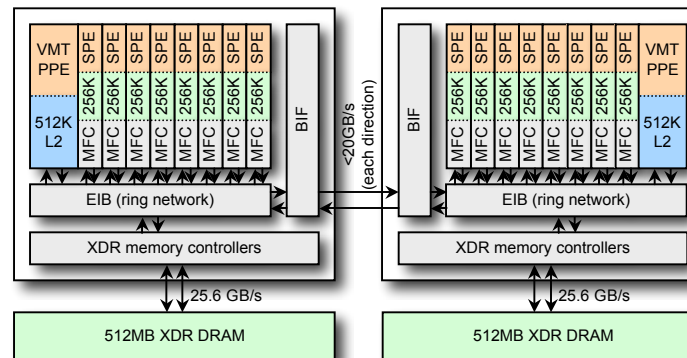
## AMD Opteron 2356 (Barcelona)



## 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

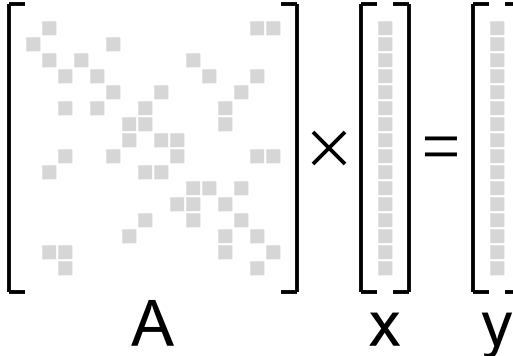
- Most entries are 0.0
- Performance advantage in only storing/operating on the nonzeros
- Requires significant meta data

## ❖ Evaluate $y=Ax$

- A is a sparse matrix
- x & y are dense vectors

## ❖ Challenges

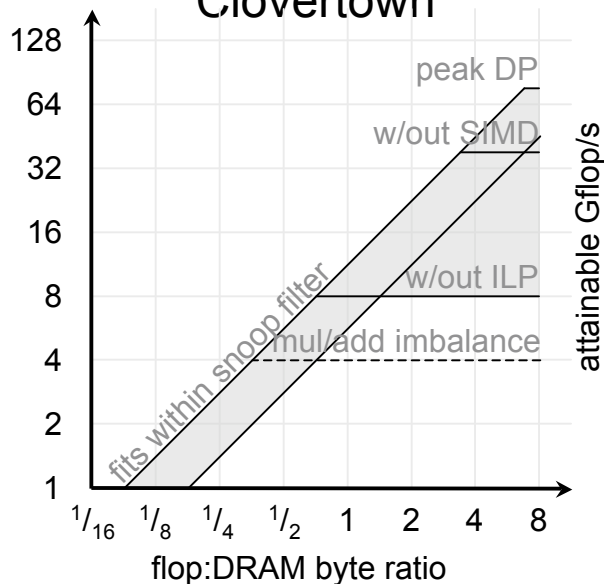
- Difficult to exploit ILP(bad for superscalar),
- Difficult to exploit DLP(bad for SIMD)
- Irregular memory access to source vector
- Difficult to load balance
- **Very low arithmetic intensity (often  $<0.166$  flops/byte)**  
**= likely memory bound**



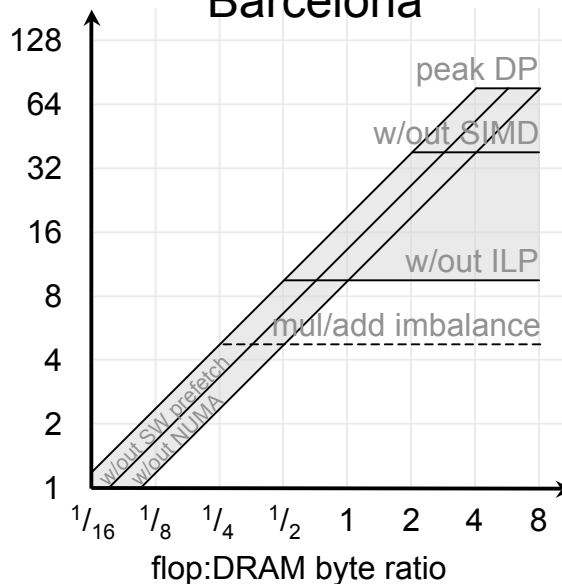
$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \times \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

A                      x                      y

Clovertown

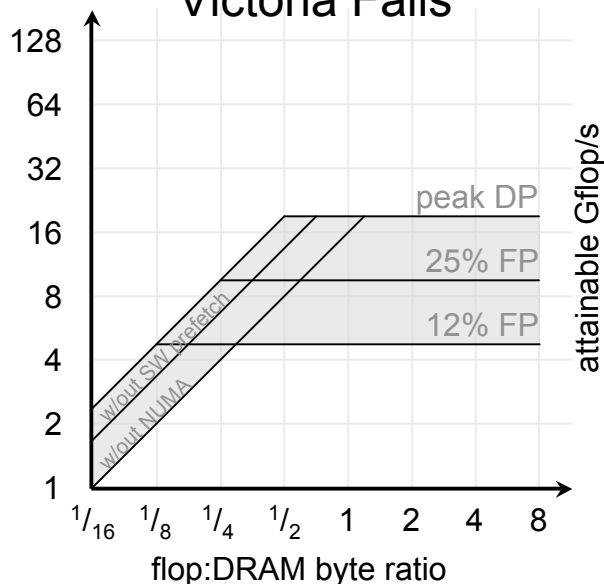


Barcelona

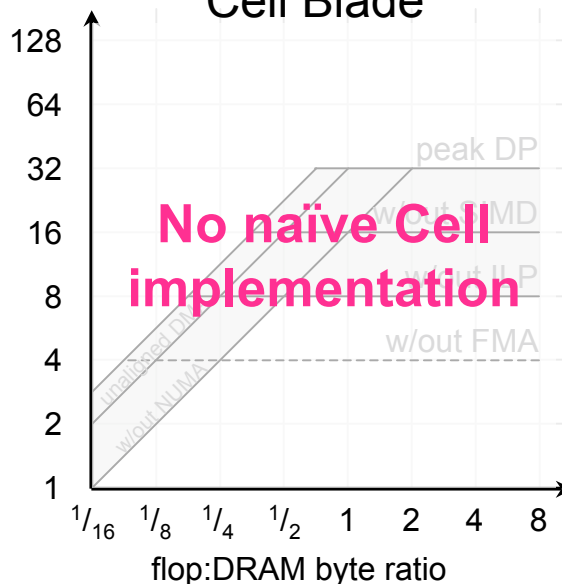


- ❖ Double precision roofline models
- ❖ FMA is inherent in SpMV (place at bottom)

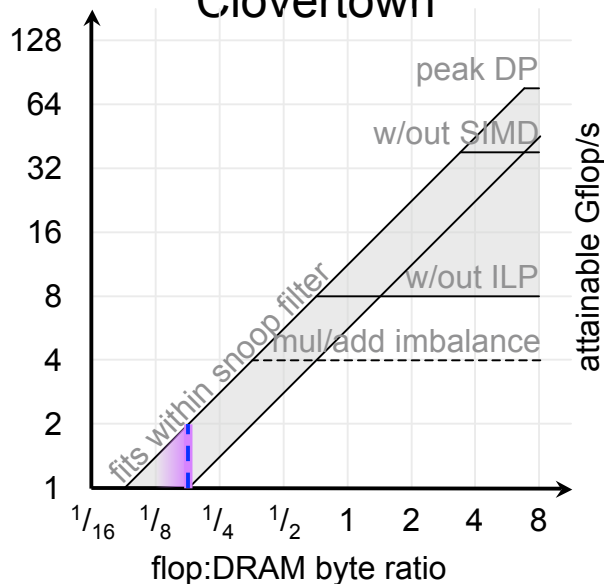
Victoria Falls



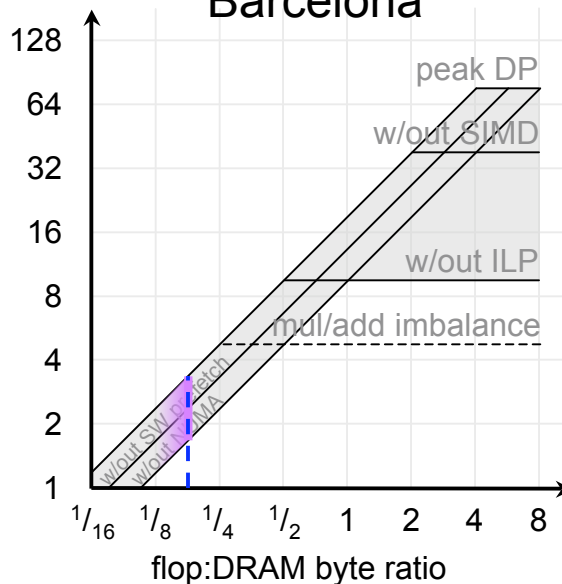
Cell Blade



Clovertown

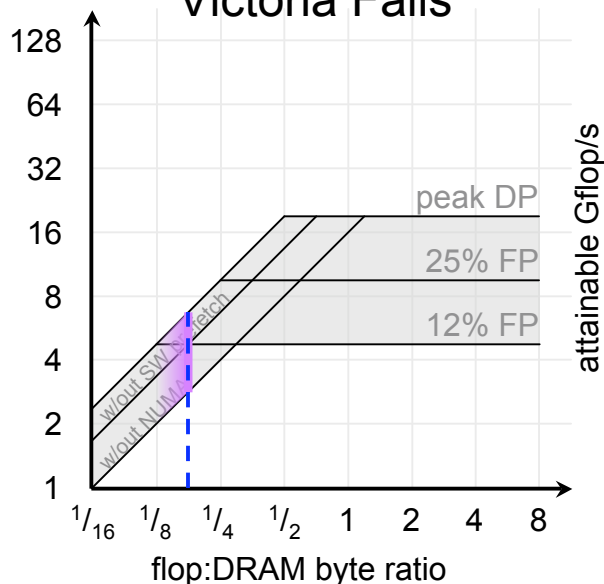


Barcelona

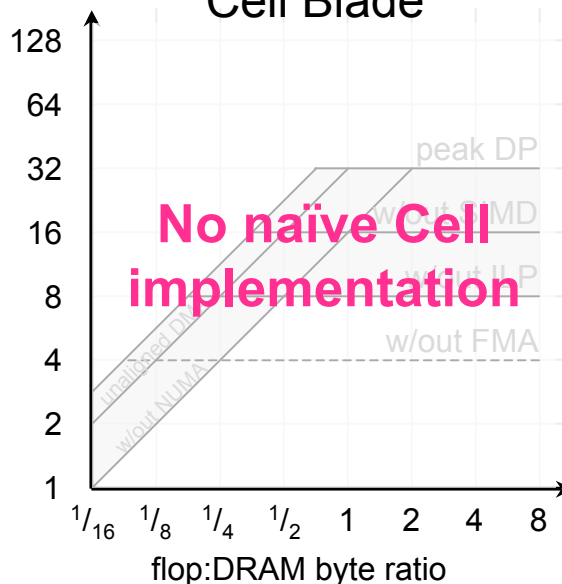


- ❖ Two unit stride streams
- ❖ Inherent FMA
- ❖ No ILP
- ❖ No DLP
- ❖ FP is 12-25%
- ❖ Naïve compulsory  
flop:byte < 0.166

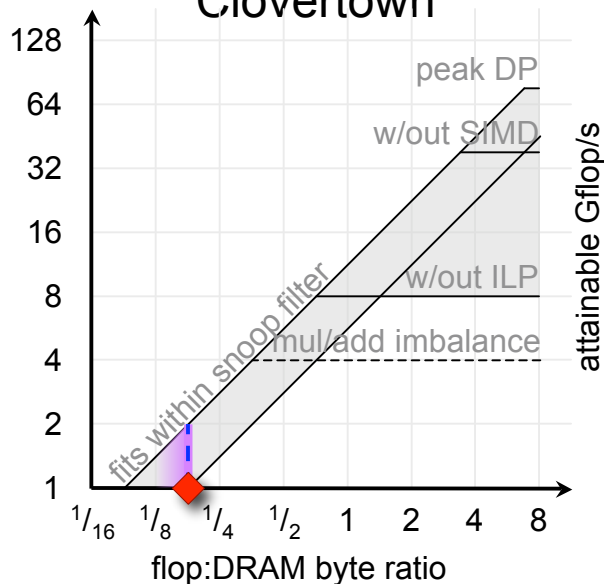
Victoria Falls



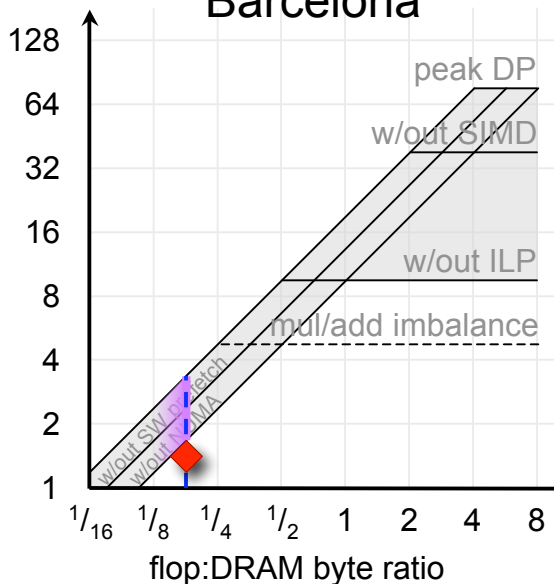
Cell Blade



Clovertown

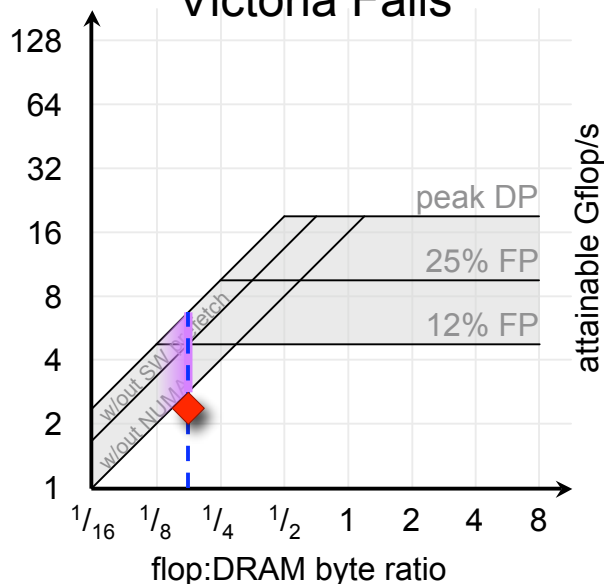


Barcelona

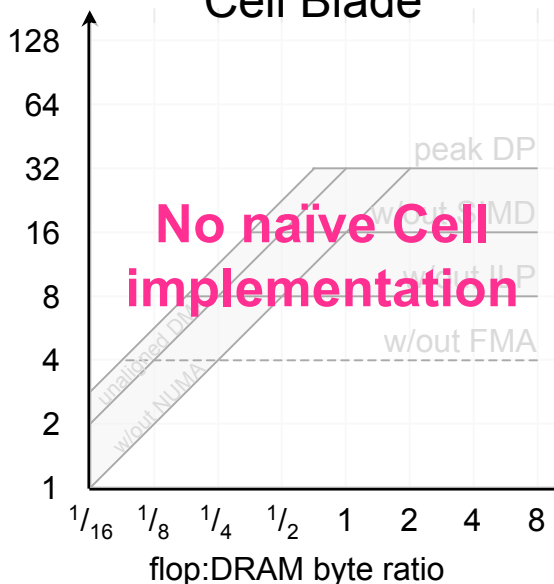


- ❖ Two unit stride streams
- ❖ Inherent FMA
- ❖ No ILP
- ❖ No DLP
- ❖ FP is 12-25%
- ❖ Naïve compulsory  
flop:byte < 0.166
- ❖ For simplicity: dense  
matrix in sparse format

Victoria Falls

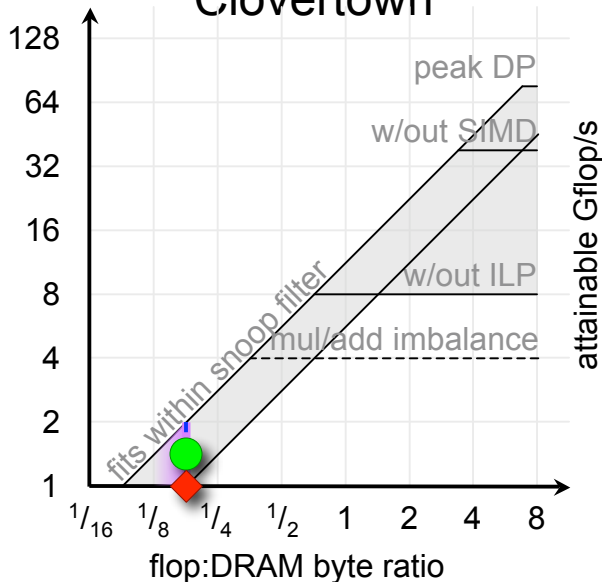


Cell Blade

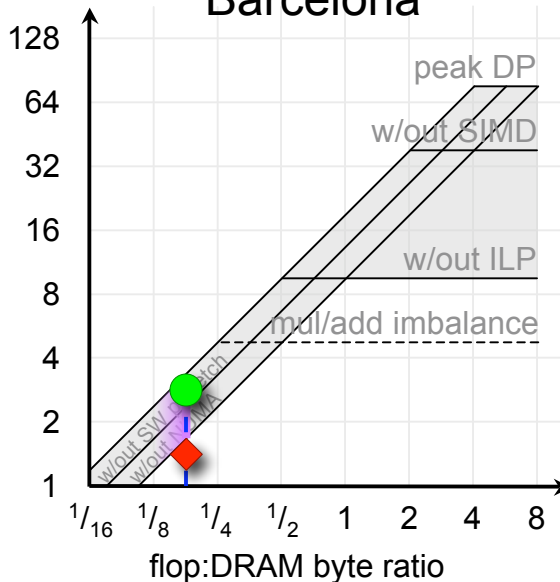


**No naïve Cell  
implementation**

Clovertown

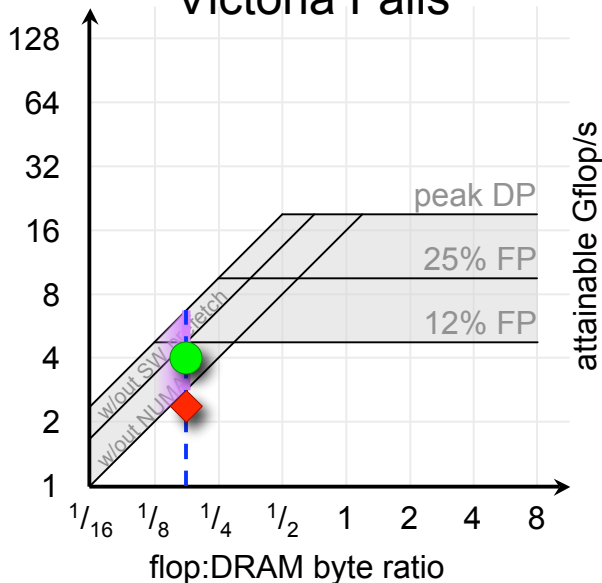


Barcelona

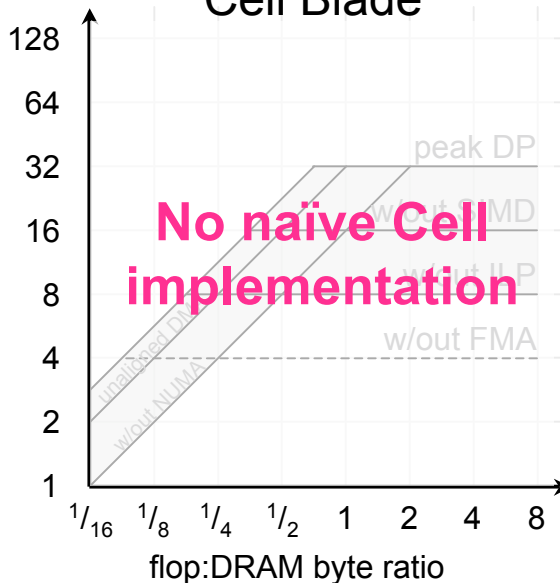


- ❖ compulsory flop:byte ~ 0.166
- ❖ utilize all memory channels

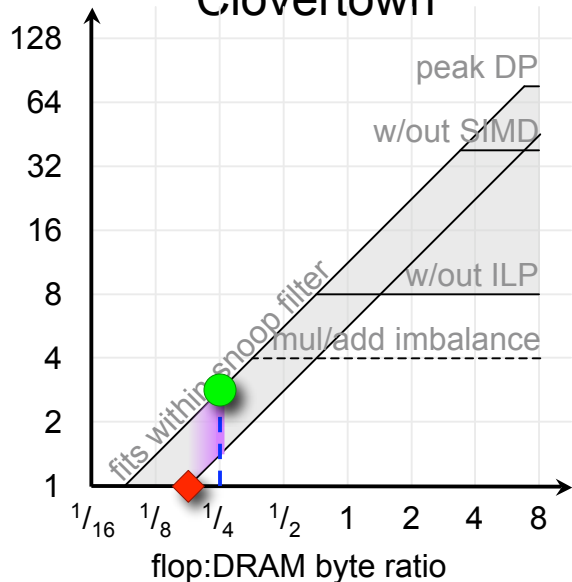
Victoria Falls



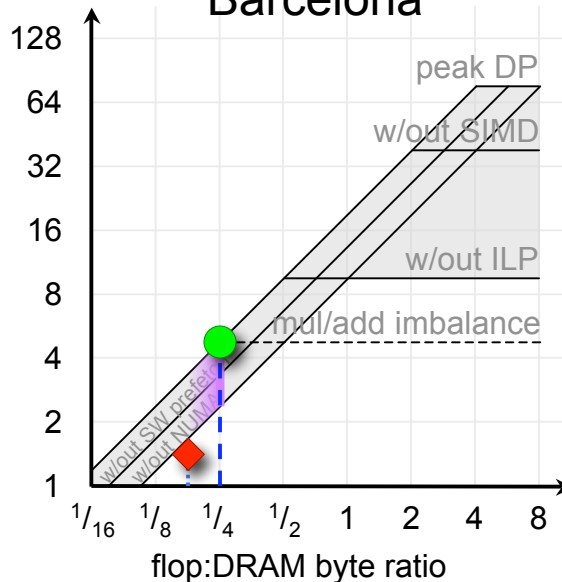
Cell Blade



Clovertown

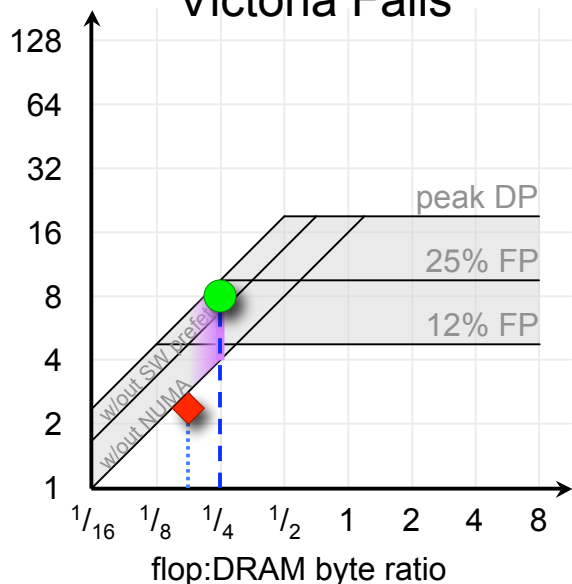


Barcelona

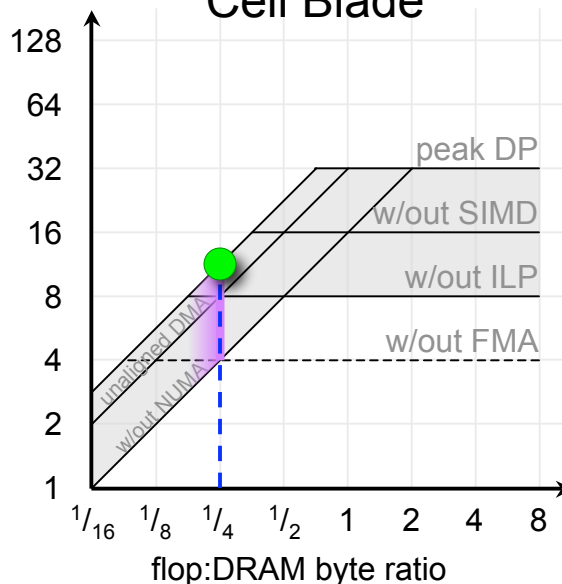


- ❖ Inherent FMA
- ❖ Register blocking improves ILP, DLP, flop:byte ratio, and FP% of instructions

Victoria Falls



Cell Blade

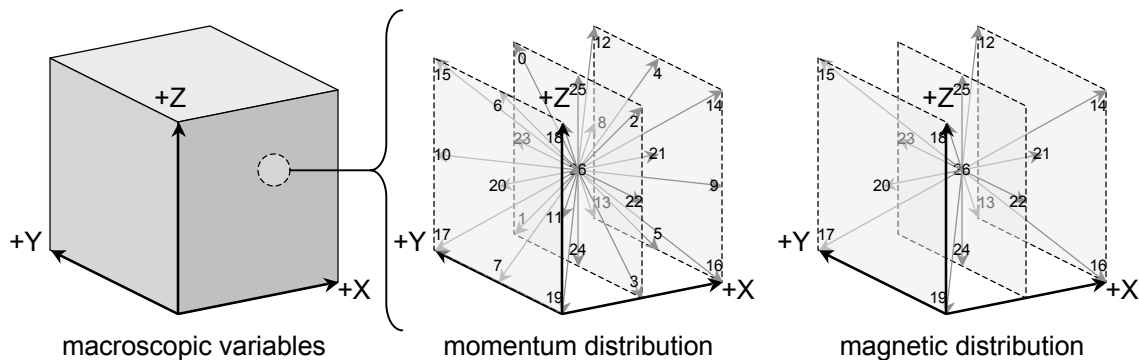


# Lattice-Boltzmann Magneto-Hydrodynamics (LBMHD)

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.

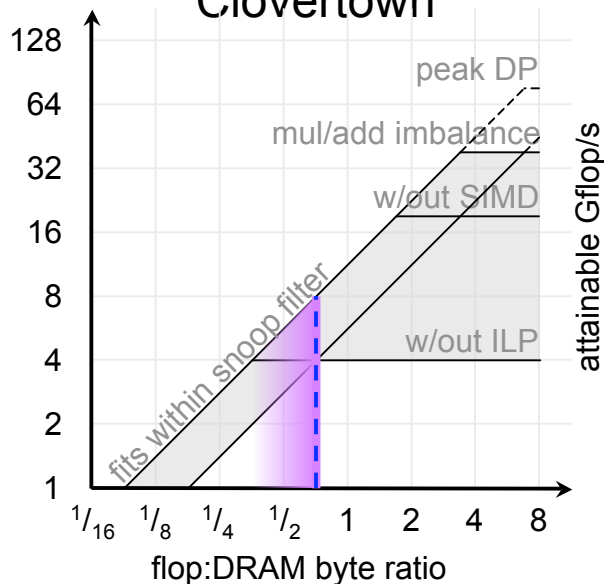
**Best Paper, Application Track**

- ❖ 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)

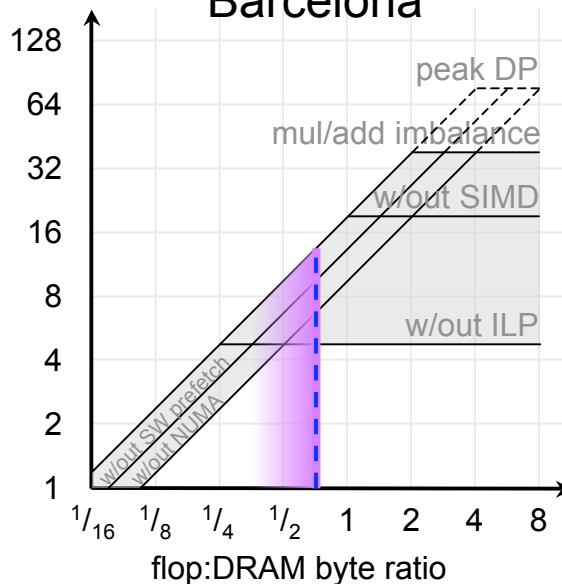




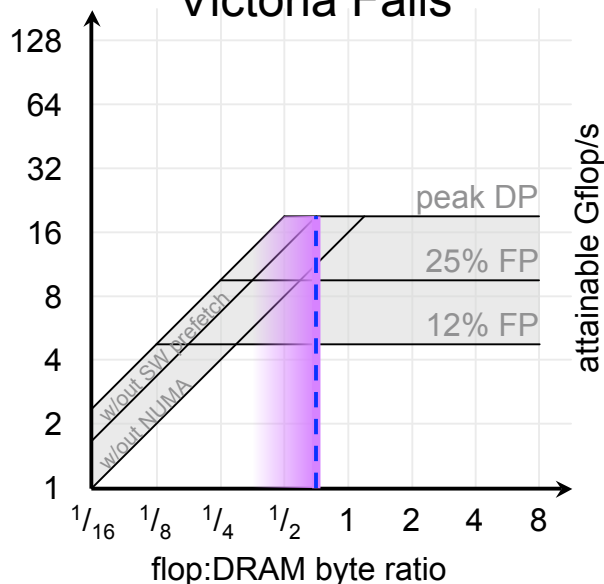
Clovertown



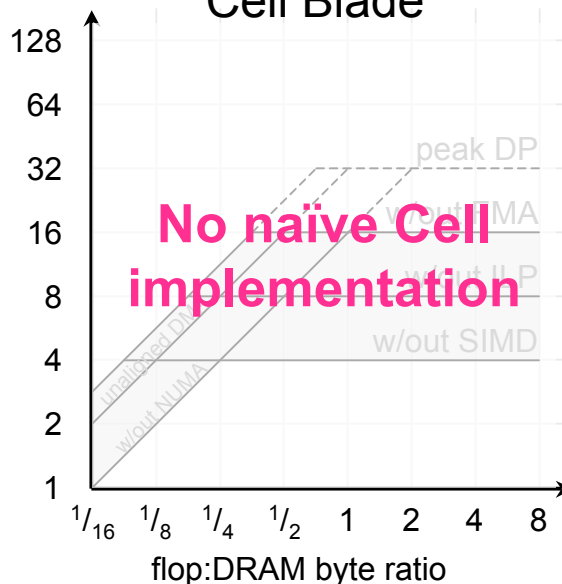
Barcelona



Victoria Falls

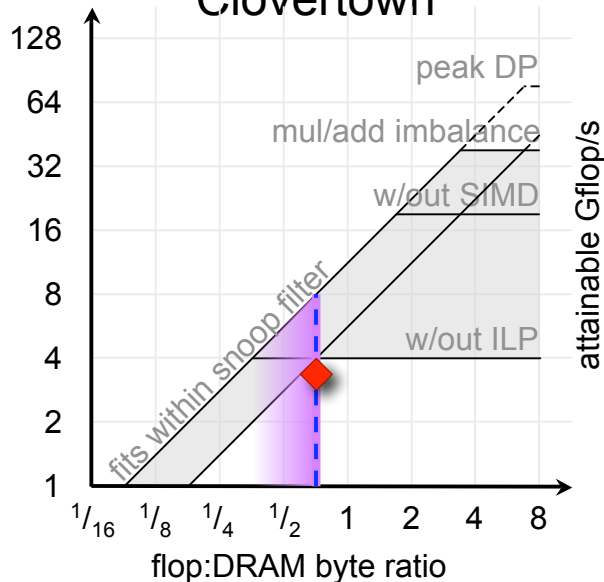


Cell Blade

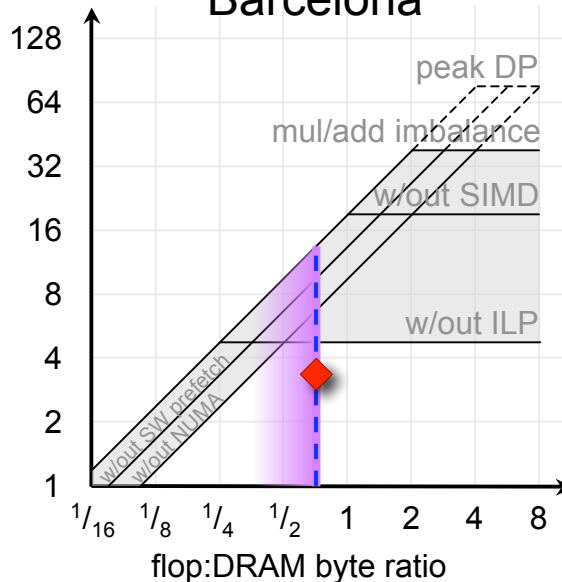


- ❖ Huge datasets
- ❖ NUMA allocation/access
- ❖ Little ILP
- ❖ No DLP
- ❖ Far more adds than multiplies (imbalance)
- ❖ **Essentially random access to memory**
- ❖ Flop:byte ratio ~0.7
- ❖ High conflict misses

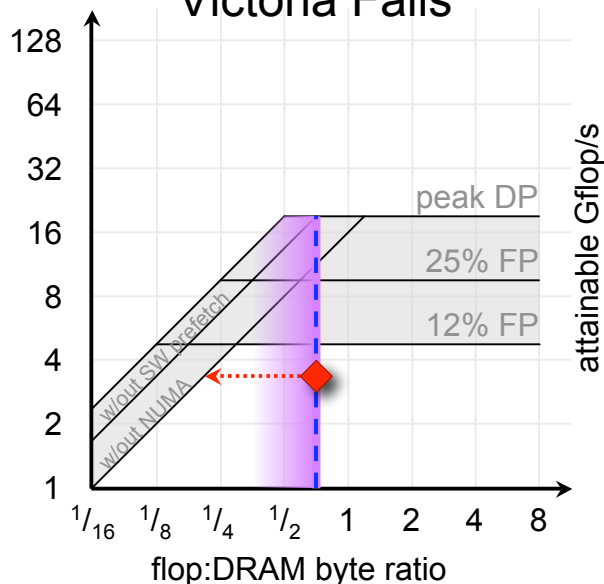
Clovertown



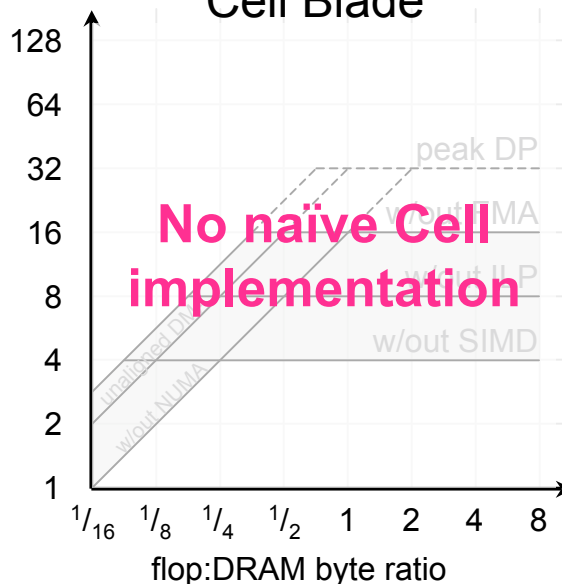
Barcelona



Victoria Falls

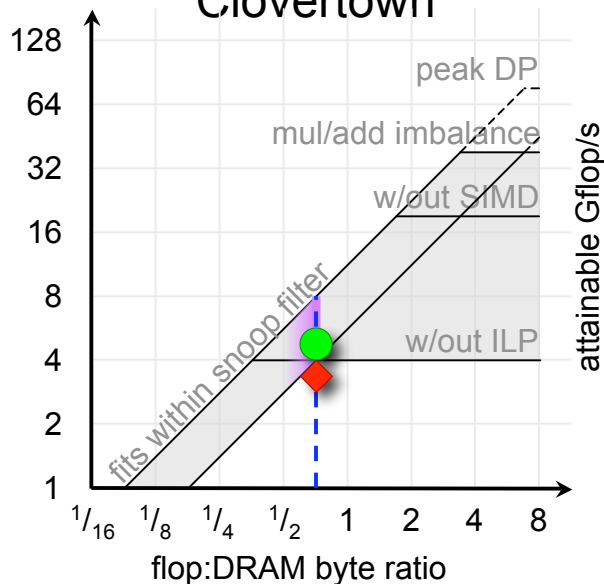


Cell Blade

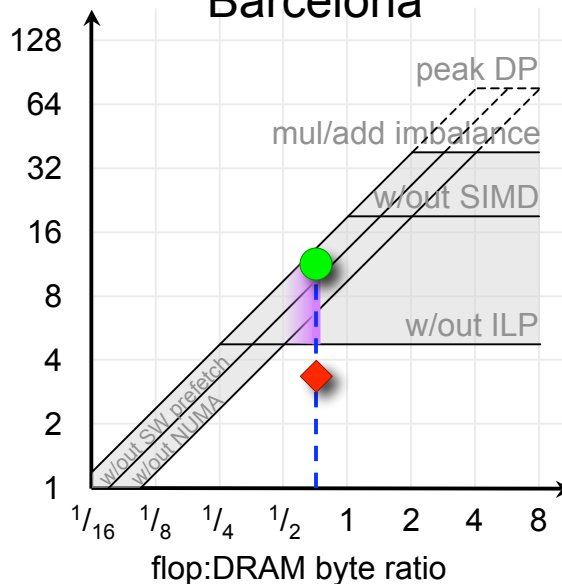


- ❖ Huge datasets
- ❖ NUMA allocation/access
- ❖ Little ILP
- ❖ No DLP
- ❖ Far more adds than multiplies (imbalance)
- ❖ **Essentially random access to memory**
- ❖ Flop:byte ratio ~0.7
- ❖ High conflict misses
- ❖ Peak VF performance with 64 threads (our of 128) - high conflict misses

Clovertown

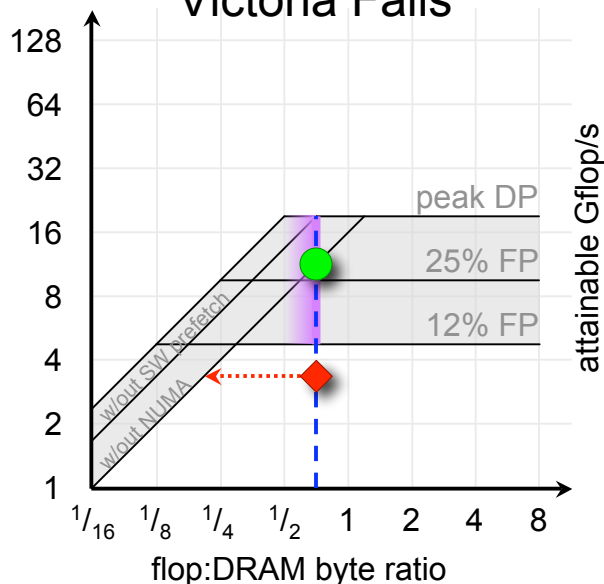


Barcelona

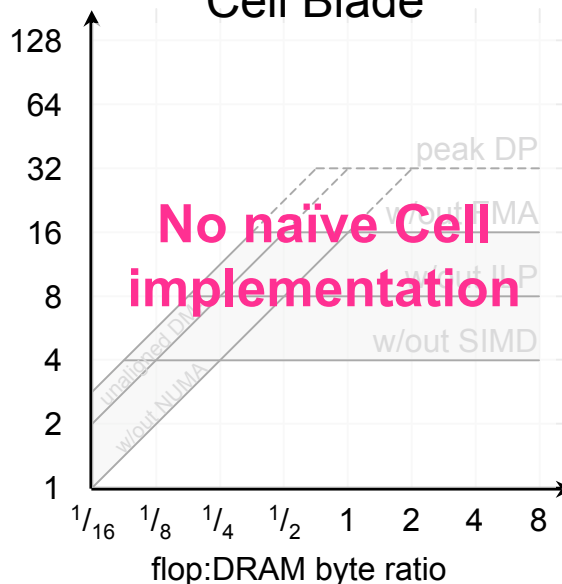


- ❖ Vectorize the code to eliminate TLB capacity misses
- ❖ Ensures unit stride access (bottom bandwidth ceiling)
- ❖ Tune for optimal VL
- ❖ Clovertown pinned to lower BW ceiling

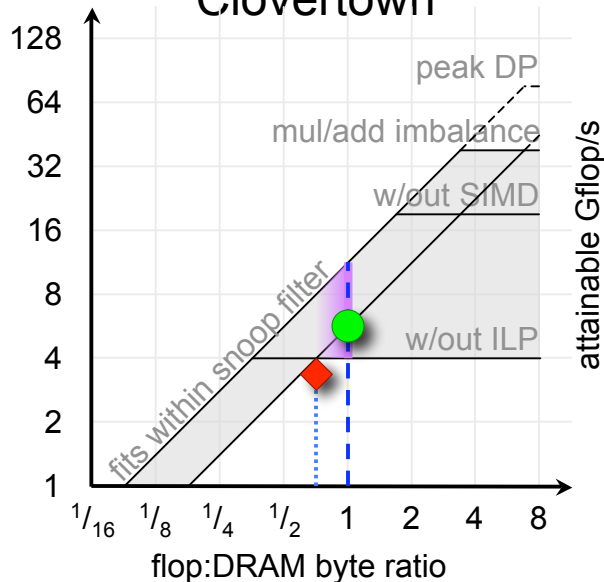
Victoria Falls



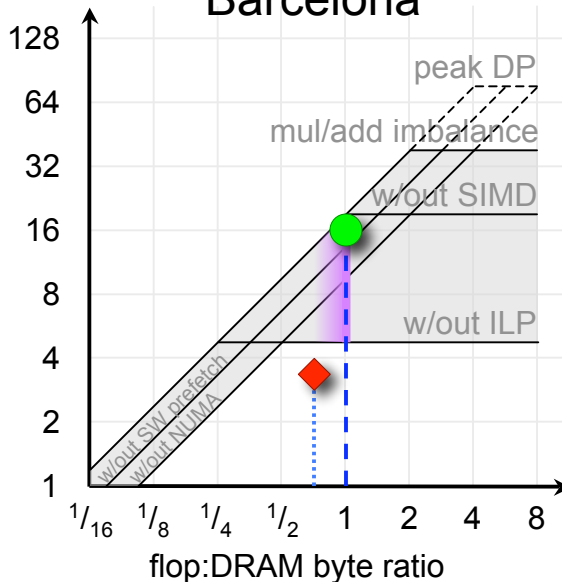
Cell Blade



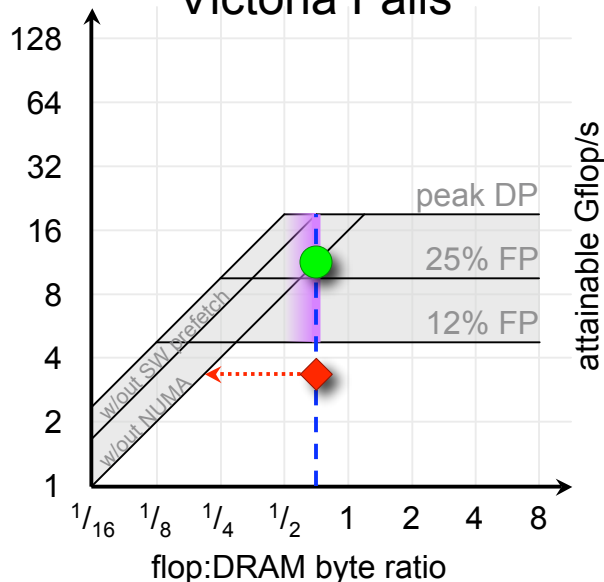
### Clovertown



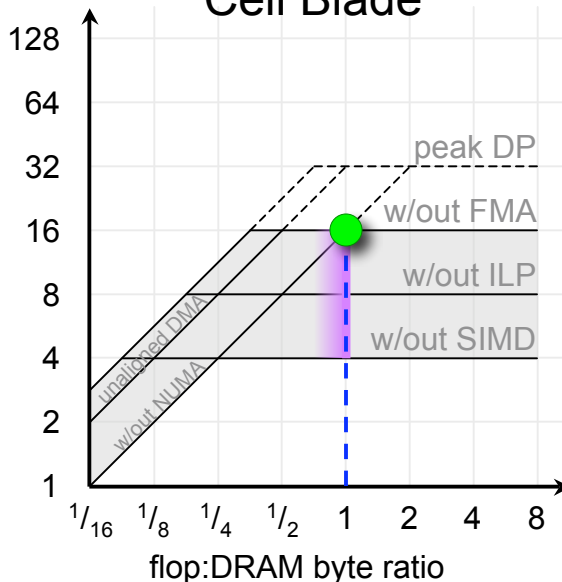
### Barcelona



### Victoria Falls



### Cell Blade

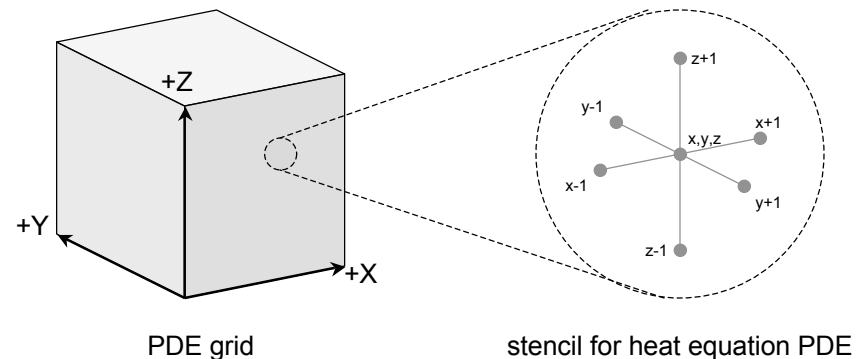


- ❖ Make SIMDization explicit
- ❖ Technically, this swaps ILP and SIMD ceilings
- ❖ Use cache bypass instruction: ***movntpd***
- ❖ Increases flop:byte ratio to ~1.0 on x86/Cell

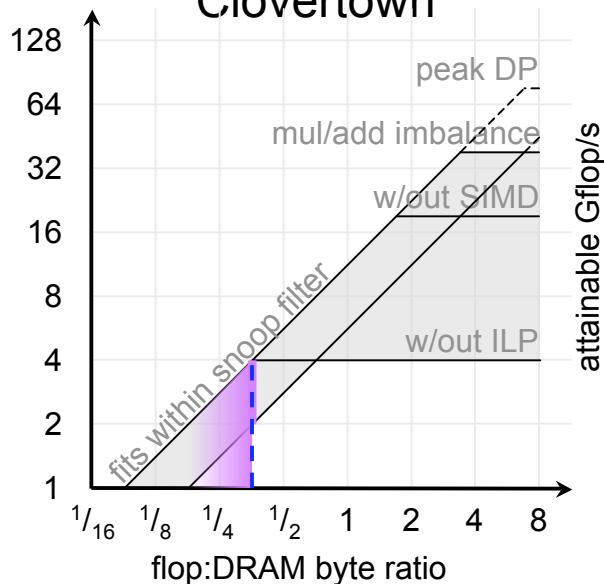
# The Heat Equation Stencil

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.

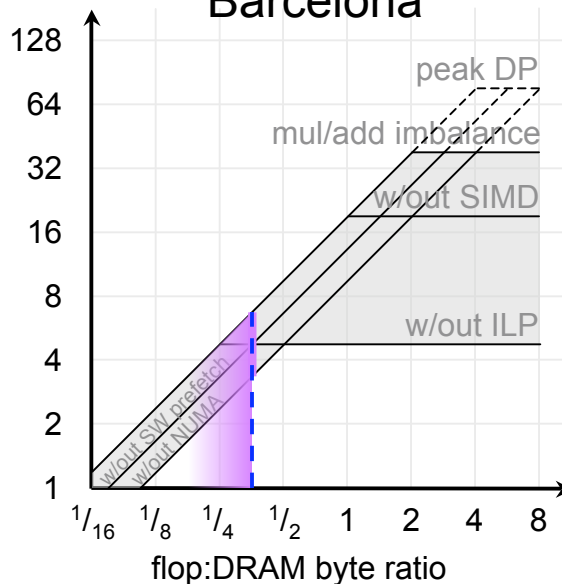
- ❖ Explicit Heat equation on a regular grid
- ❖ Jacobi
- ❖ One double per point in space
- ❖ 7-point nearest neighbor stencil
- ❖ Must:
  - read every point from DRAM
  - perform 8 flops (linear combination)
  - write every point back to DRAM
- ❖ Just over 0.5 flops/byte (ideal)
- ❖ Cache locality is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



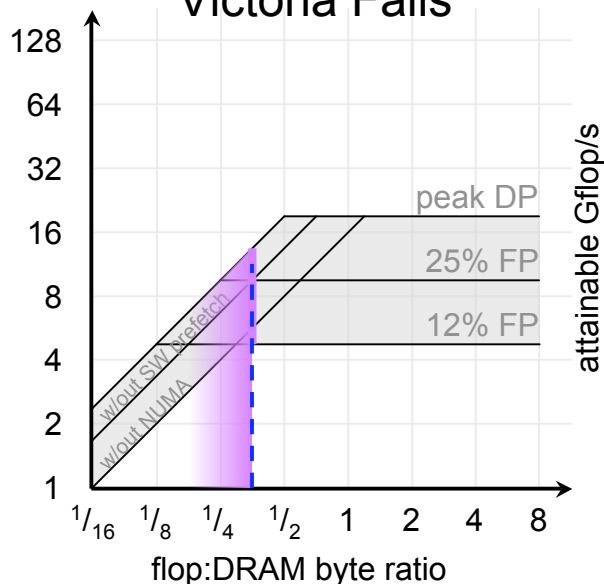
Clovertown



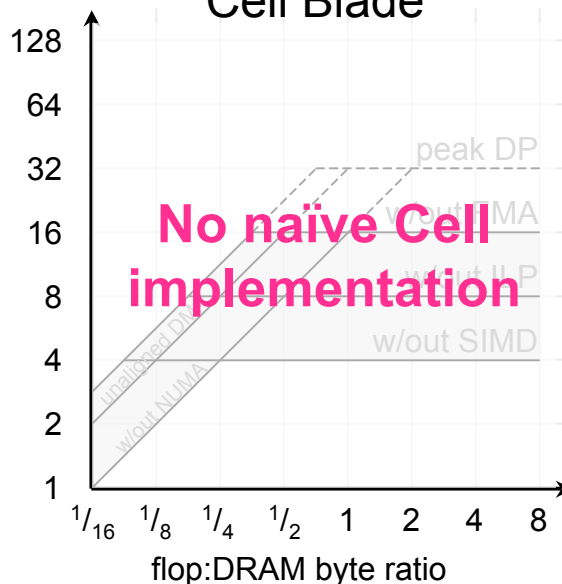
Barcelona



Victoria Falls

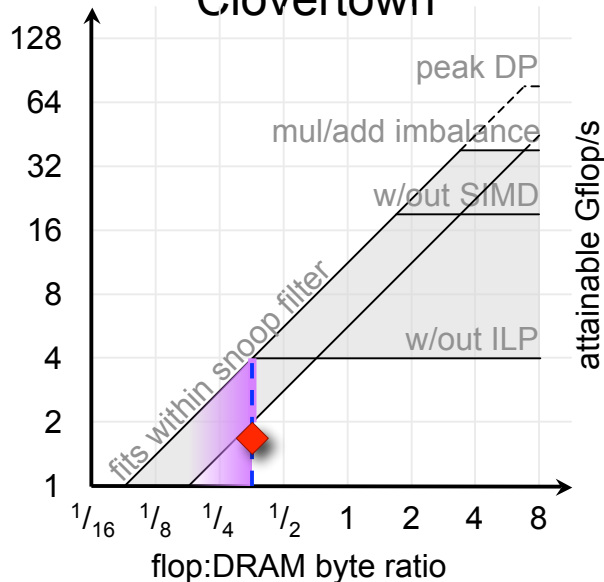


Cell Blade

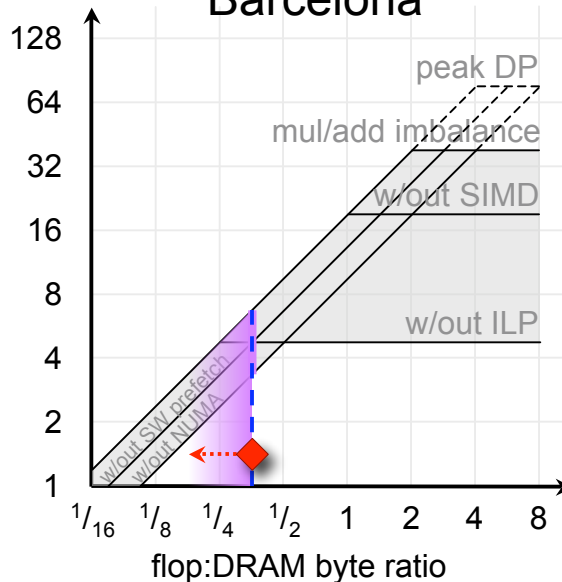


- ❖ Large datasets
- ❖ 2 unit stride streams
- ❖ No NUMA
- ❖ Little ILP
- ❖ No DLP
- ❖ Far more adds than multiplies (imbalance)
- ❖ Ideal flop:byte ratio  $1/3$
- ❖ High locality requirements
- ❖ Capacity and conflict misses will severely impair flop:byte ratio

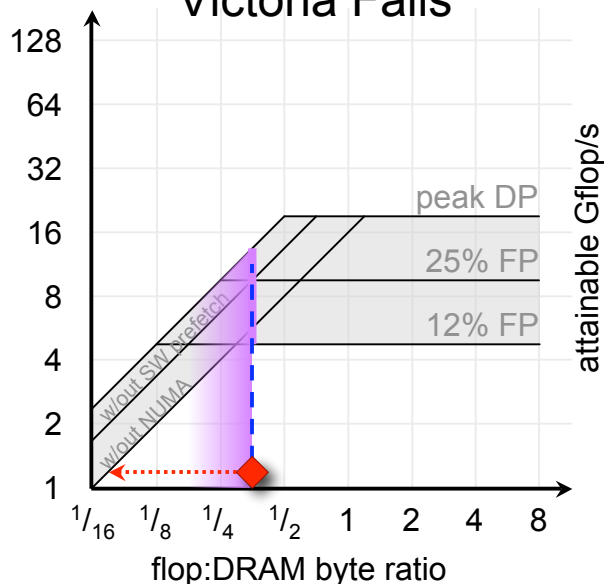
Clovertown



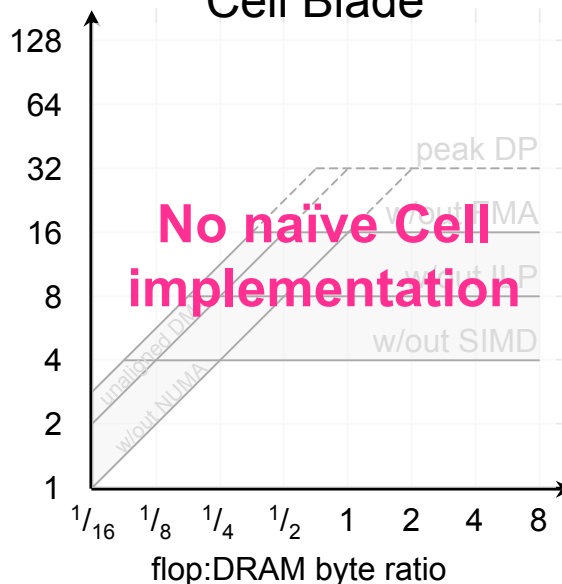
Barcelona



Victoria Falls



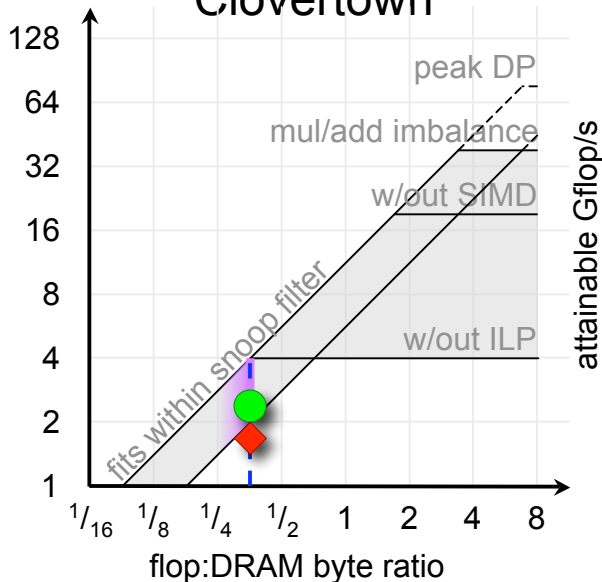
Cell Blade



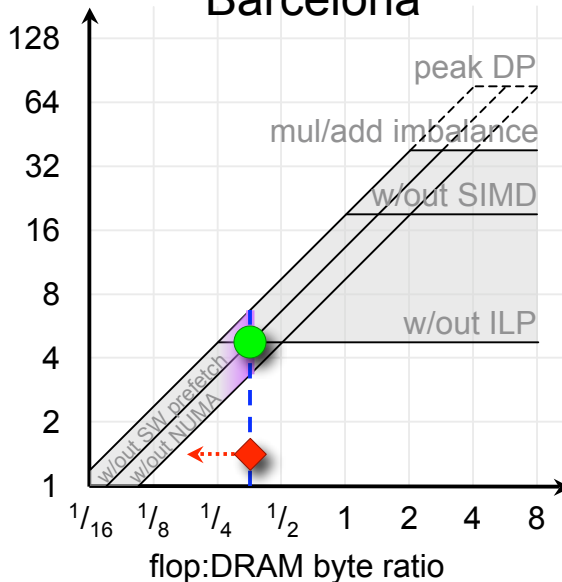
- ❖ Large datasets
- ❖ 2 unit stride streams
- ❖ No NUMA
- ❖ Little ILP
- ❖ No DLP
- ❖ Far more adds than multiplies (imbalance)
- ❖ Ideal flop:byte ratio  $1/3$
- ❖ High locality requirements
- ❖ Capacity and conflict misses will severely impair flop:byte ratio



Clovertown

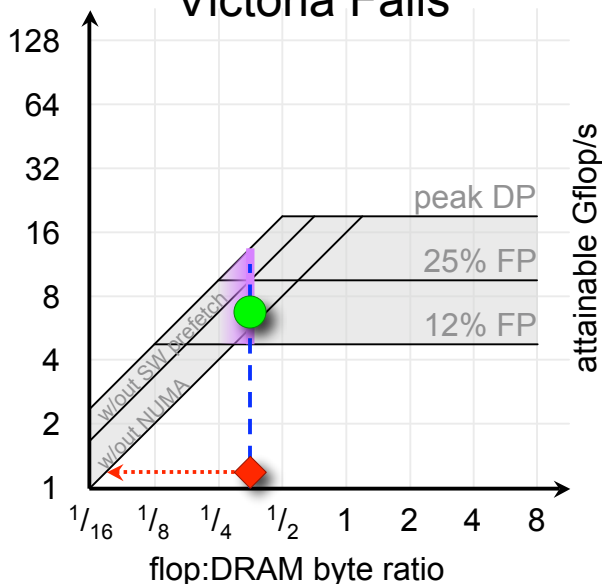


Barcelona

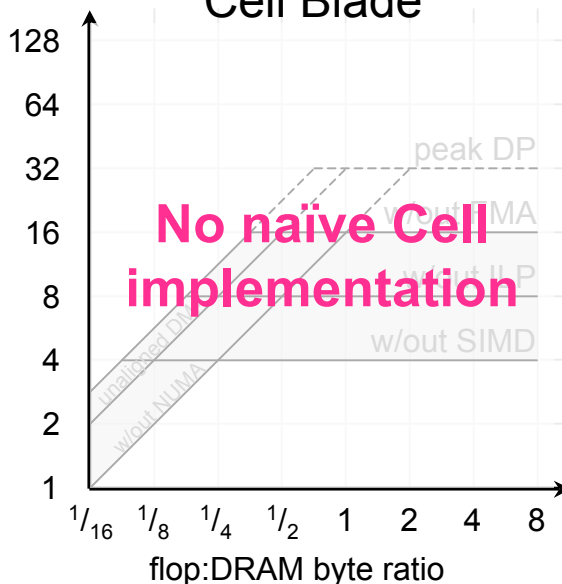


- ❖ Cache blocking helps ensure flop:byte ratio is as close as possible to  $1/3$
- ❖ Clovertown has huge caches but is pinned to lower BW ceiling
- ❖ Cache management is essential when capacity/thread is low

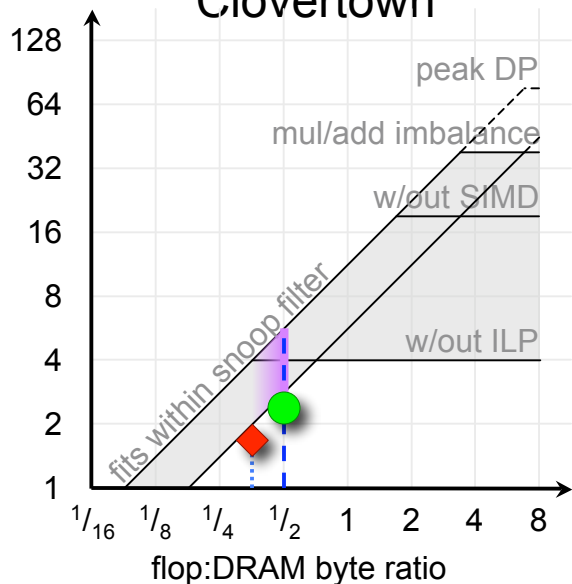
Victoria Falls



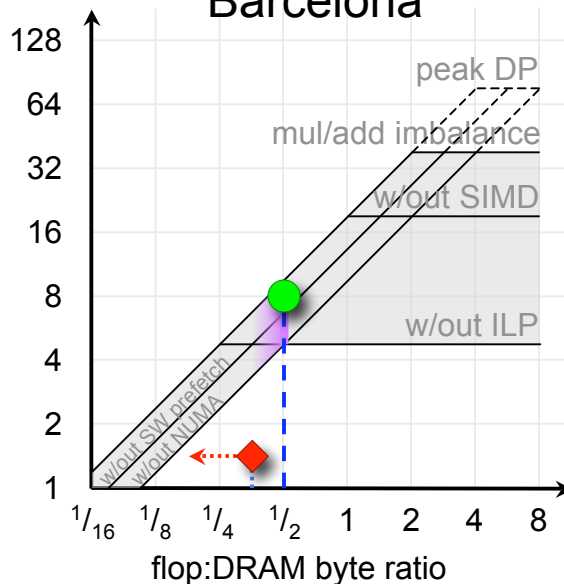
Cell Blade



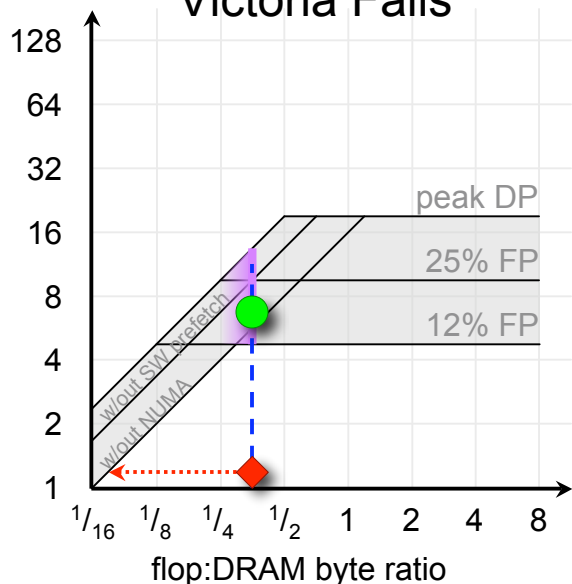
Clovertown



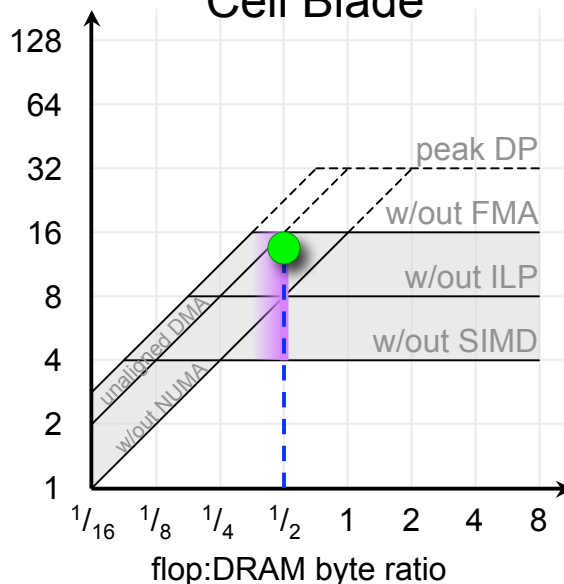
Barcelona



Victoria Falls



Cell Blade

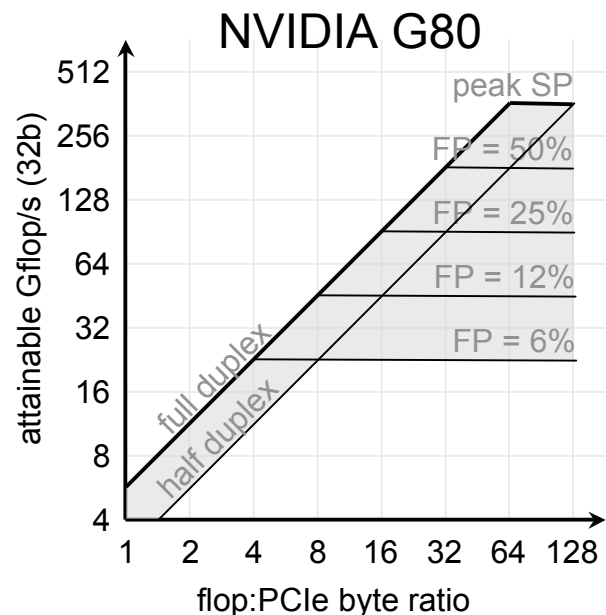


- ❖ Make SIMDization explicit
- ❖ Technically, this swaps ILP and SIMD ceilings
- ❖ Use cache bypass instruction: ***movntpd***
- ❖ Increases flop:byte ratio to ~0.5 on x86/Cell

# Refining the Roofline

- ❖ 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...

- ❖ 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



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

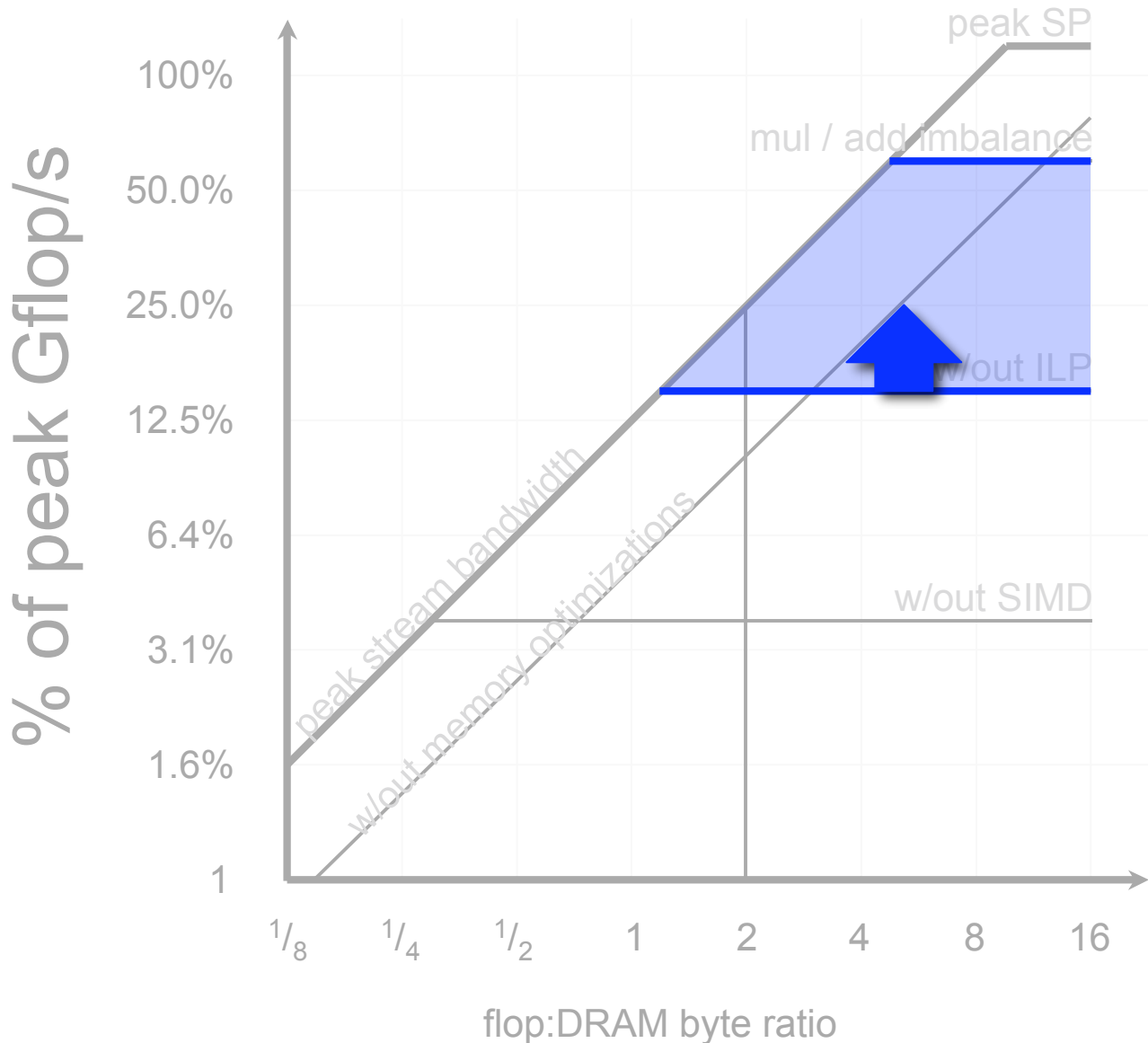
**{Gflop/s, GIPS, crypto, ...} × {L2, DRAM, PCIe, Network}**

- ❖ 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

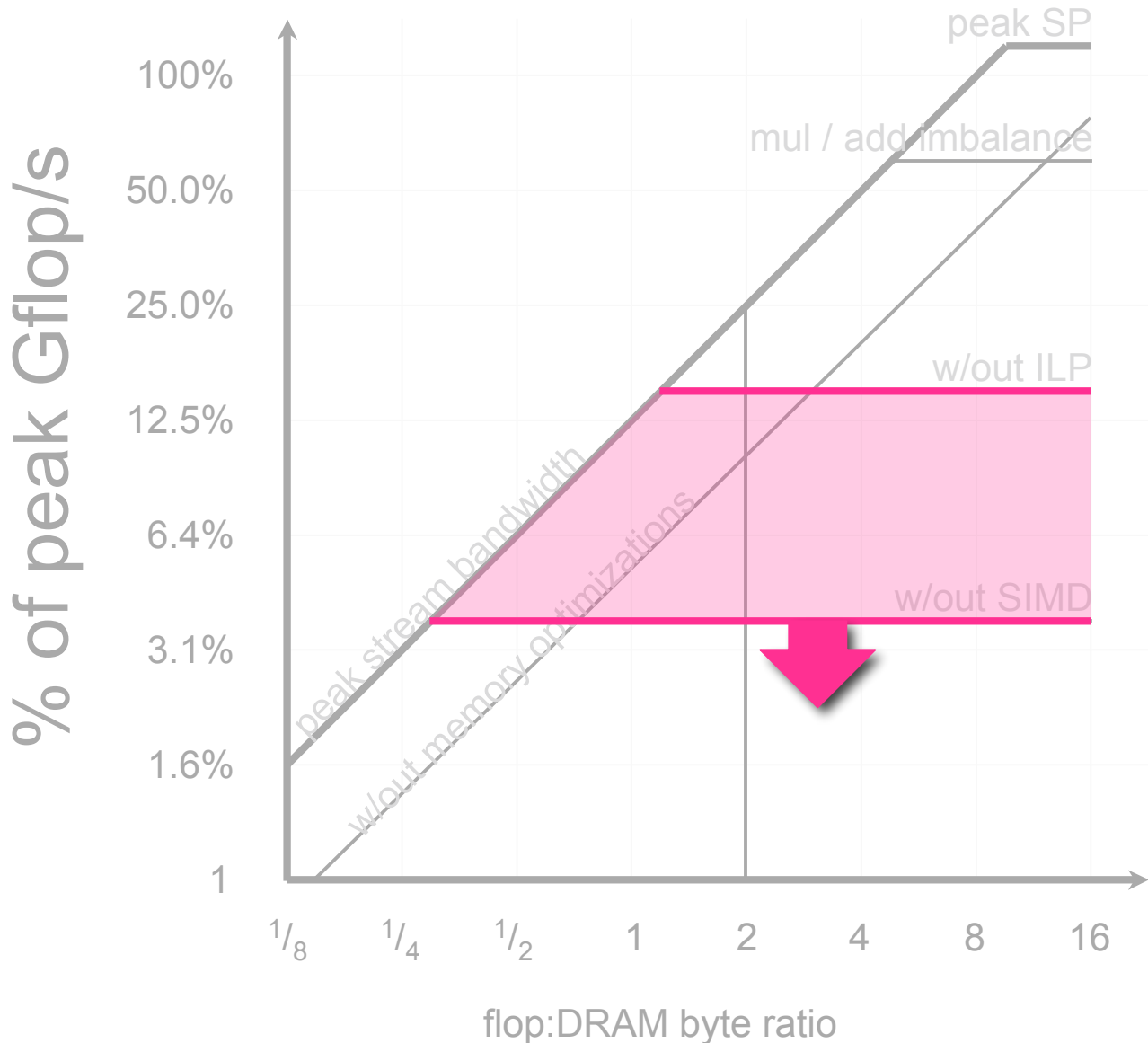
# Questions ?



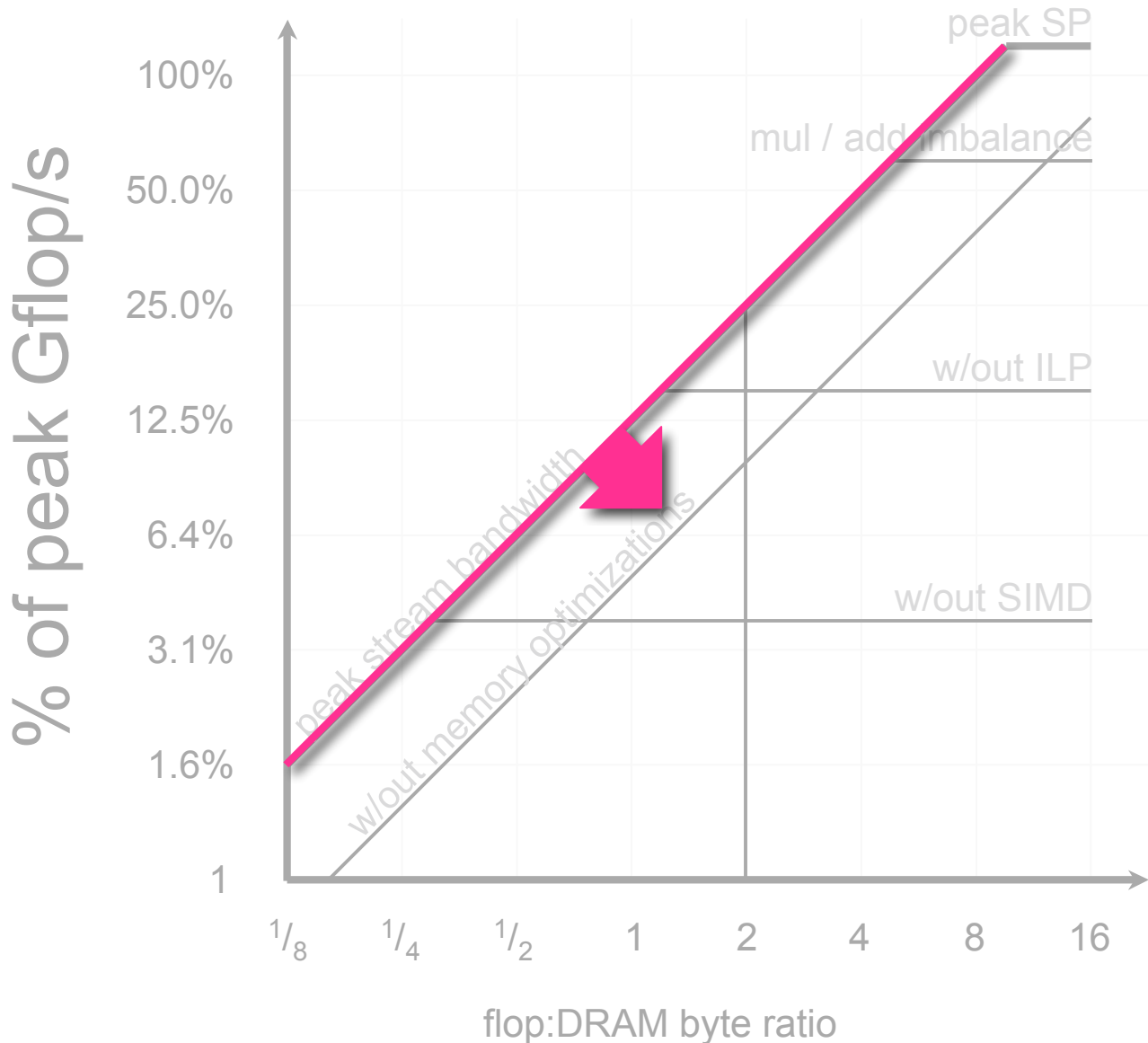
# BACKUP SLIDES



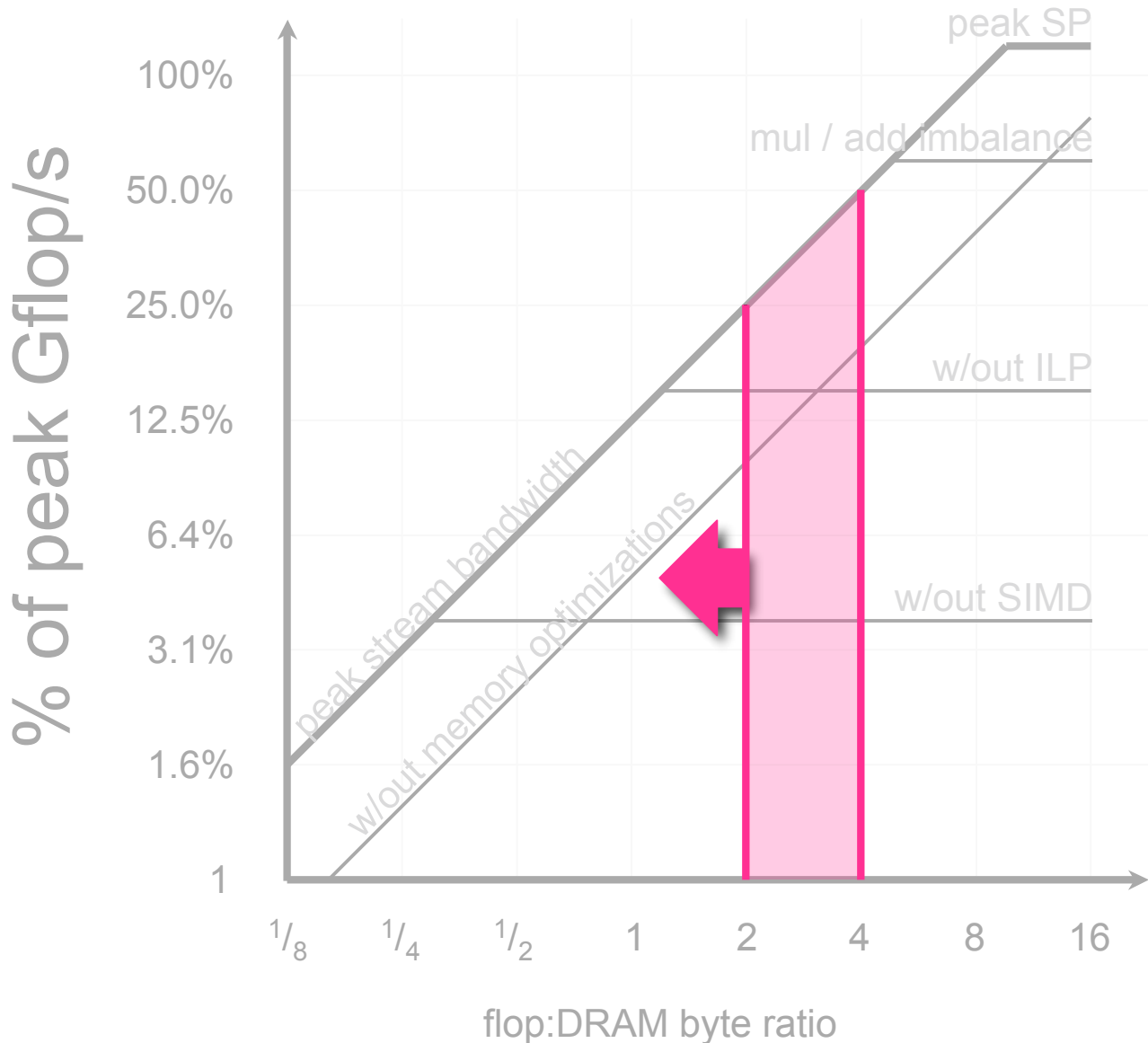
- ❖ ILP is decreasing (shorter pipelines, multithreading)
- ❖ SIMD is becoming wider
- ❖ Bandwidth isn't keeping up with #cores
- ❖ Application flop:byte is decreasing
- ❖ Cache/Local Store management is becoming critical



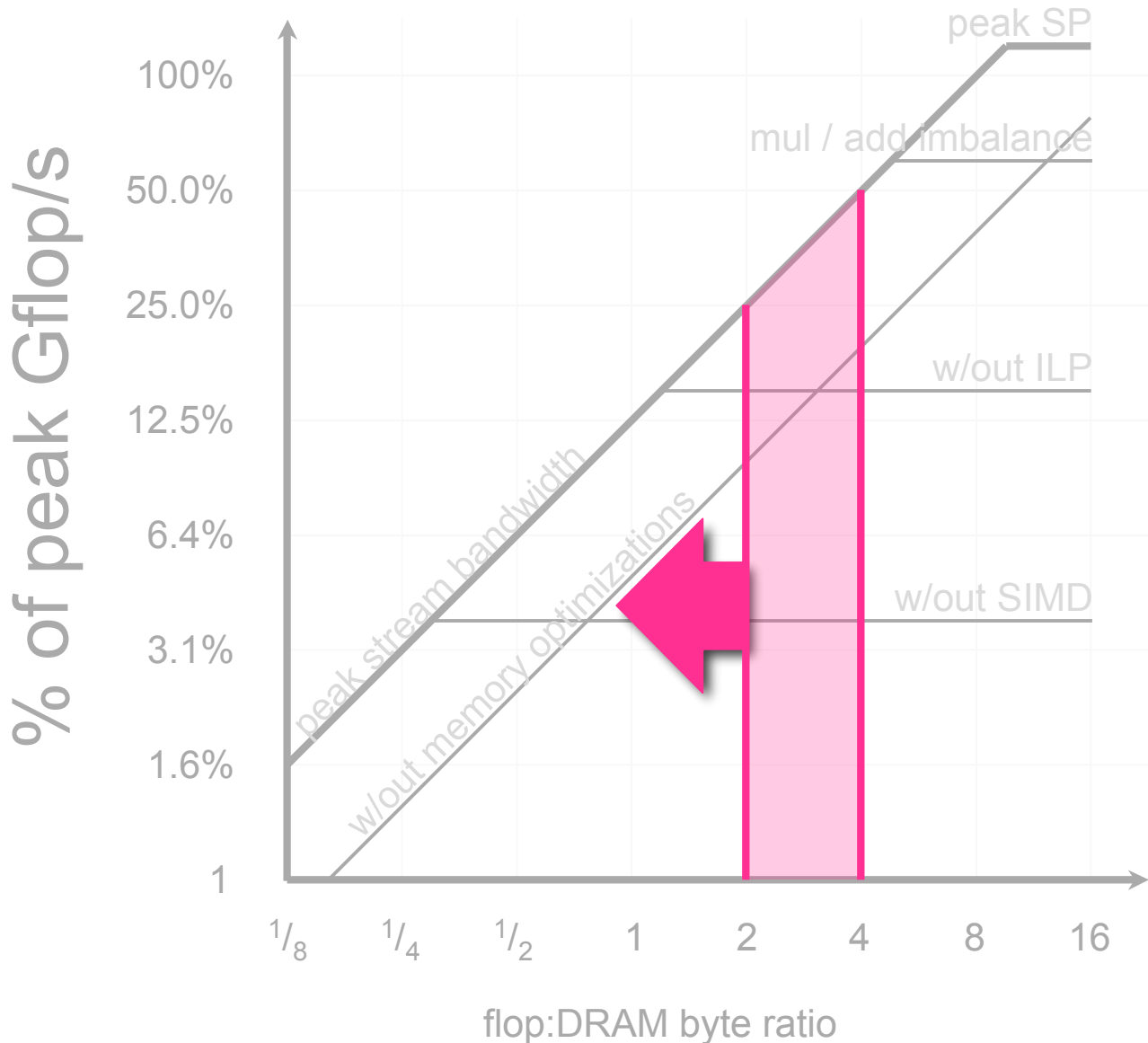
- ❖ ILP is decreasing (shorter pipelines, multithreading)
- ❖ SIMD is becoming wider
- ❖ Bandwidth isn't keeping up with #cores
- ❖ Application flop:byte is decreasing
- ❖ Cache/Local Store management is becoming critical



- ❖ 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

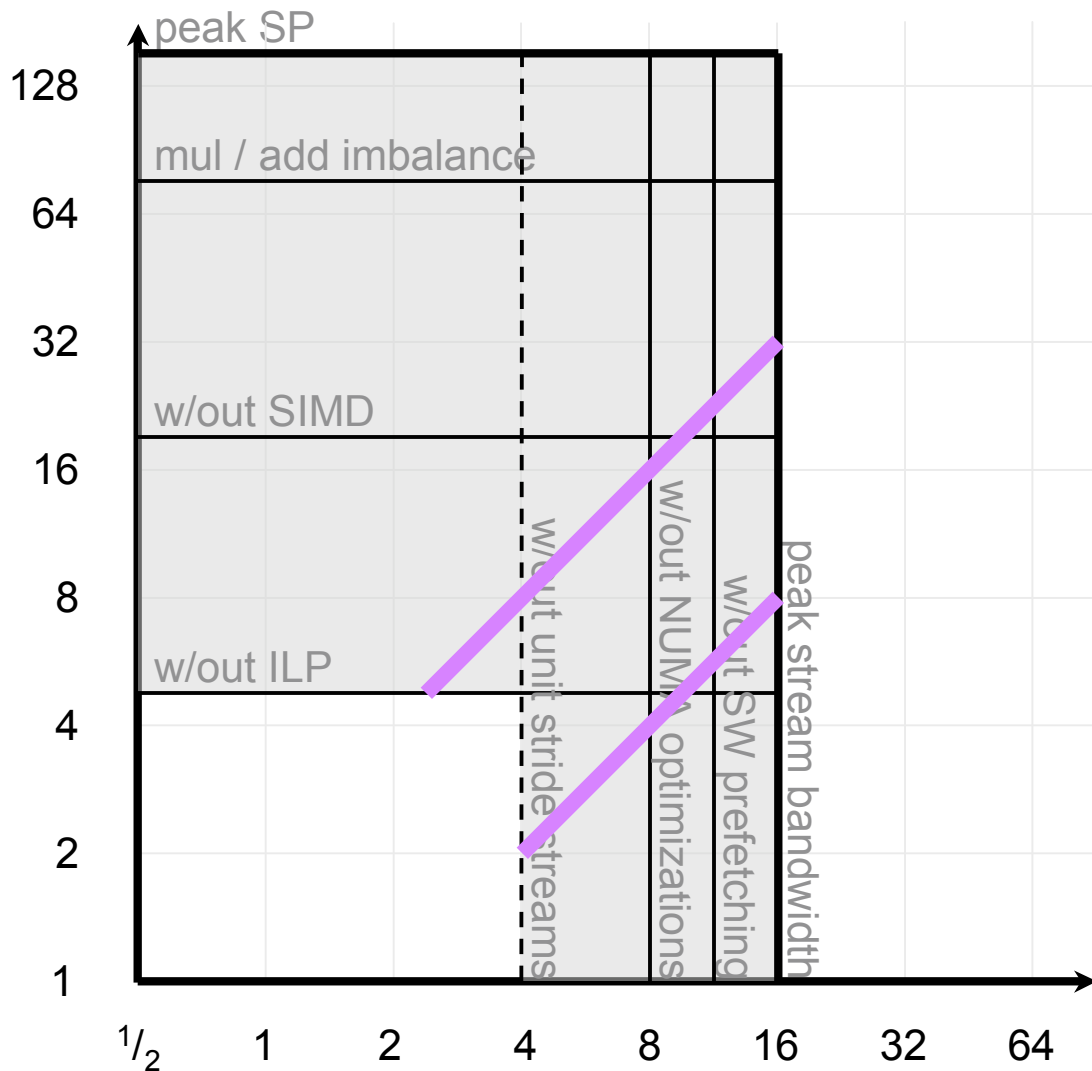


- ❖ ILP is decreasing (shorter pipelines, multithreading)
- ❖ SIMD is becoming wider
- ❖ Bandwidth isn't keeping up with #cores
- ❖ Application flop:byte is decreasing
- ❖ Cache/Local Store management is becoming critical



- ❖ 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

attainable Gflop/s (32b)



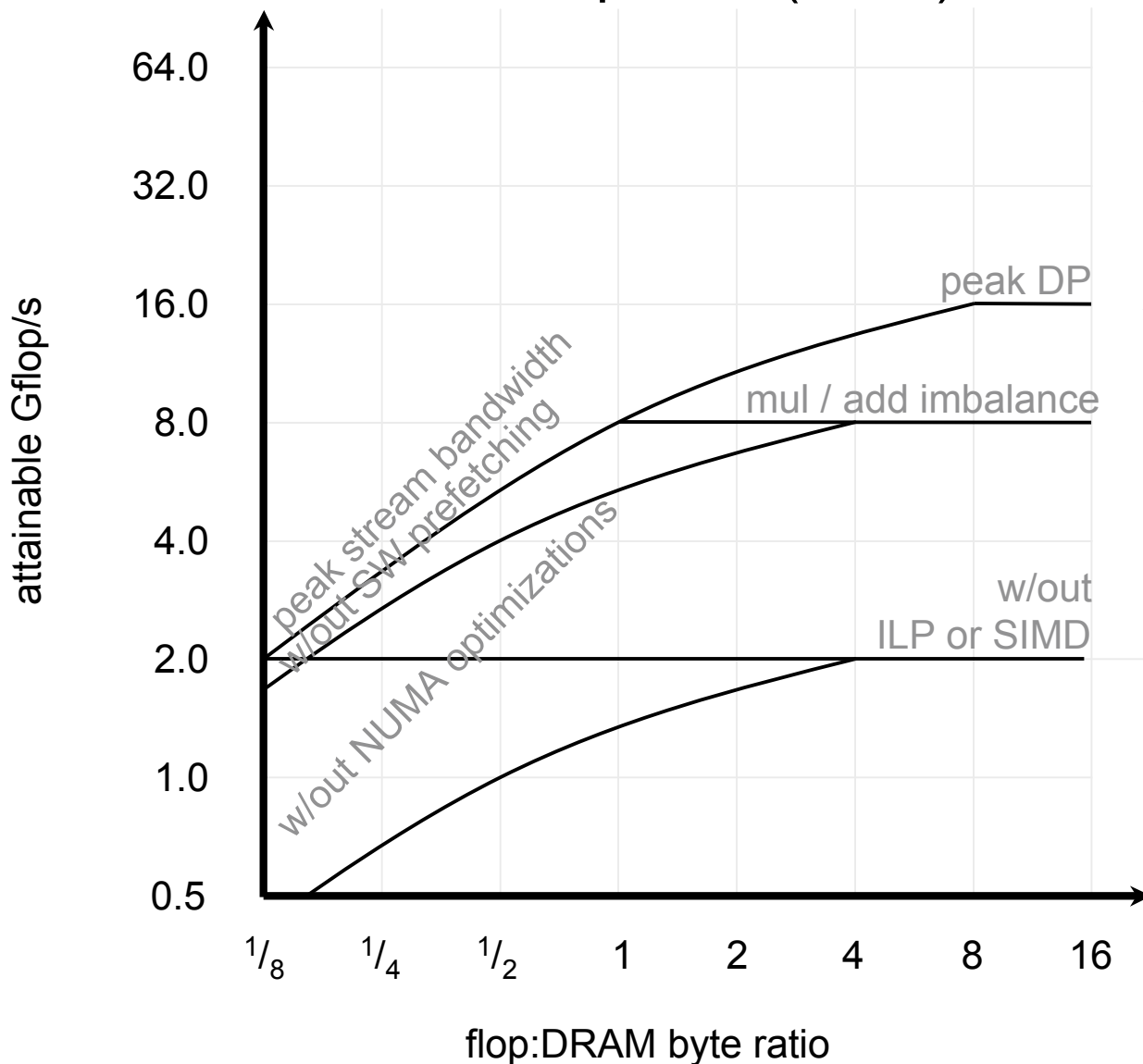
- ❖ Same formalism can be plotted on different axis
- ❖ Difficult to plot AI

attainable GB/s

- ❖ 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 synchronous MPI communication, atypical for DRAM access on modern architectures

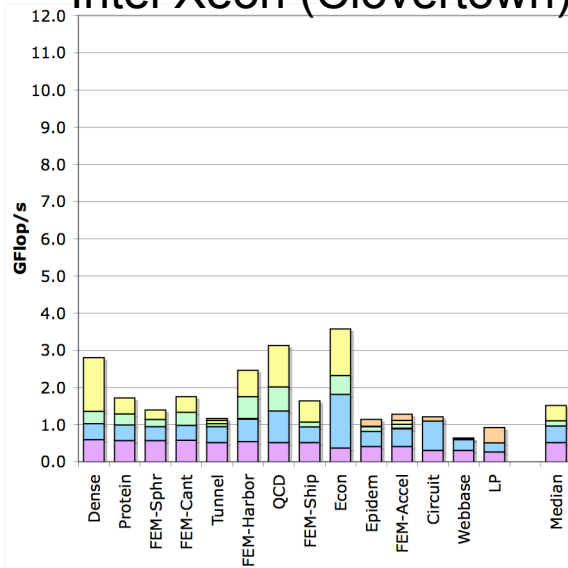


## AMD Opteron (rev.F)

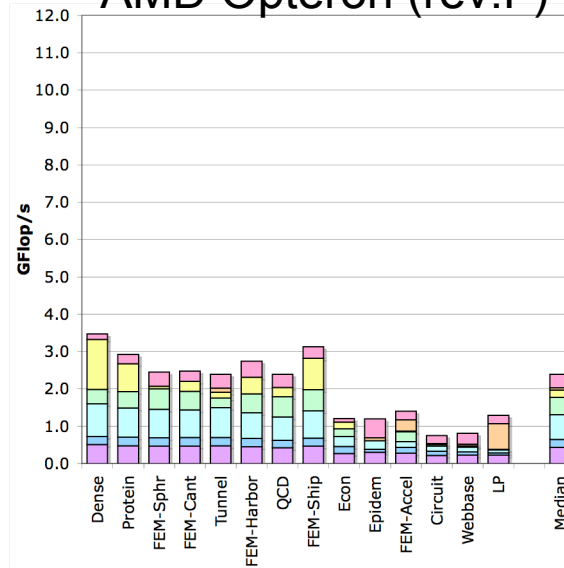


- ❖ Not typical of multi-thread/core architectures
- ❖ Not typical of architectures with ooo or HW prefetchers
- ❖ More common in network accesses

## Intel Xeon (Clovertown)

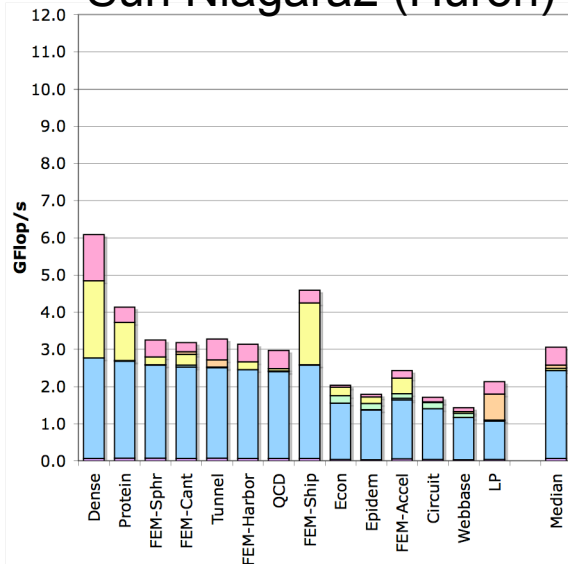


## AMD Opteron (rev.F)

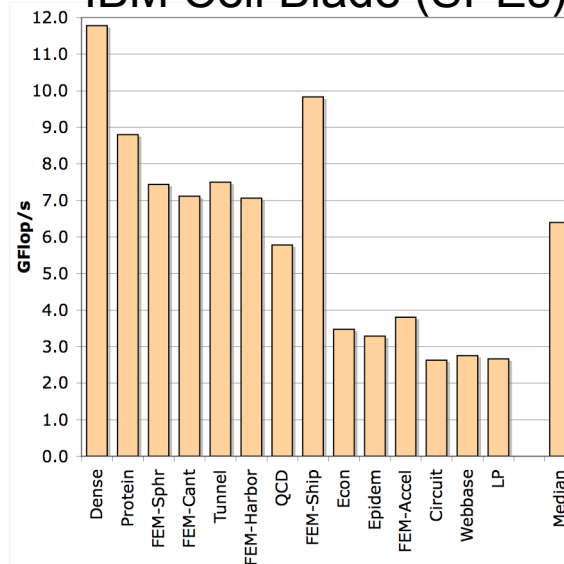


- ❖ Wrote a double precision Cell/SPE version
- ❖ DMA, local store blocked, NUMA aware, etc...
- ❖ Only 2x1 and larger BCOO
- ❖ Only the SpMV-proper routine changed
- ❖ About 12x faster (median) than using the PPEs alone.

## Sun Niagara2 (Huron)

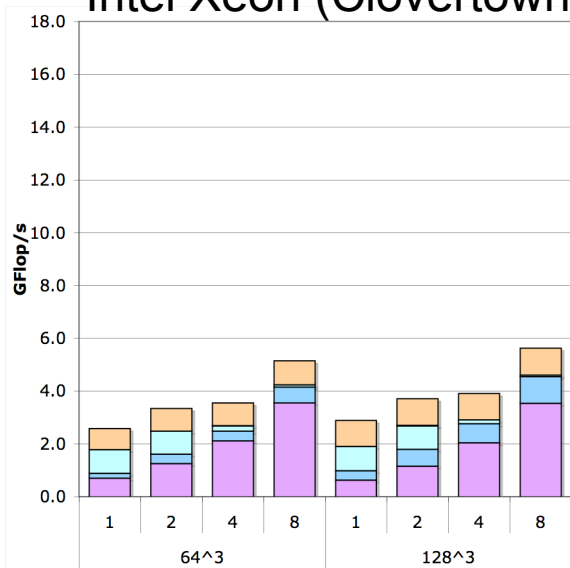


## IBM Cell Blade (SPEs)

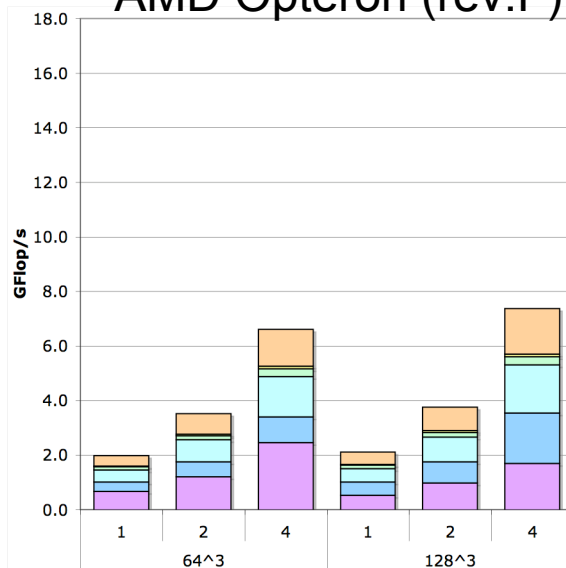


- +More DIMMs(opteron), +FW fix, array padding(N2), etc...
- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naive Pthreads
- Naive

### Intel Xeon (Clovertown)

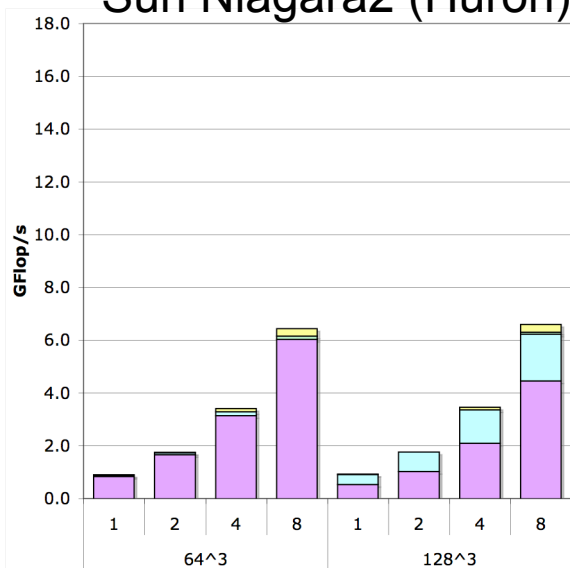


### AMD Opteron (rev.F)

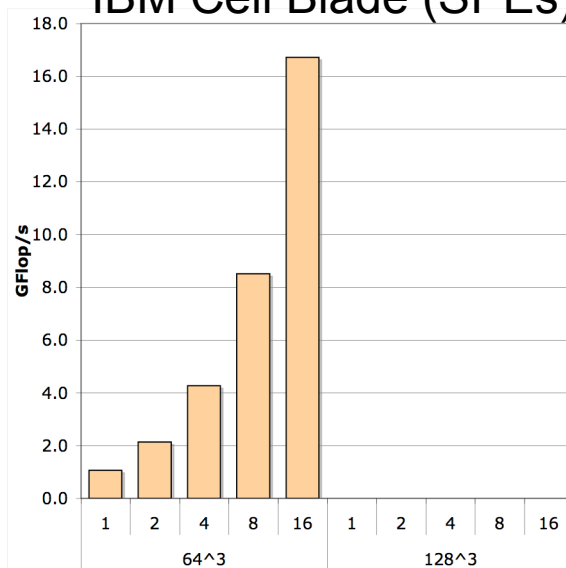


- ❖ First attempt at cell implementation.
- ❖ VL, unrolling, reordering fixed
- ❖ No NUMA
- ❖ Exploits DMA and double buffering to load vectors
- ❖ Straight to SIMD intrinsics.
- ❖ Despite the relative performance, Cell's DP implementation severely impairs performance

### Sun Niagara2 (Huron)



### IBM Cell Blade (SPEs)\*



- +SIMDization
- +SW Prefetching
- +Unrolling
- +Vectorization
- +Padding
- Naïve+NUMA

	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