

# The Roofline Model:

A pedagogical tool for program analysis and optimization

Samuel Williams<sup>1,2</sup>, David Patterson<sup>1</sup>,  
Leonid Oliker<sup>1,2</sup>, John Shalf<sup>2</sup>, Katherine Yelick<sup>1,2</sup>

<sup>1</sup>University of California, Berkeley

<sup>2</sup>Lawrence Berkeley National Laboratory

*samw@cs.berkeley.edu*

- ❖ Motivation, Goals, Audience, etc...
- ❖ Survey of multicore architectures
- ❖ Description of the Roofline model
- ❖ Introduction to Auto-tuning
- ❖ Application of the roofline to auto-tuned kernels
  - Example #1 - SpMV
  - Example #2 - LBMHD
- ❖ Conclusions

- ❖ 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?

- ❖ Throughput-oriented kernels (rather than time)
- ❖ Our performance metrics are:  
**Gflop/s** and **% of peak** (efficiency)

*for purposes of this talk, I will focus on memory-intensive 64b floating-point SPMD kernels.*

- ❖ Not focused on algorithmic innovation or computational complexity

## ❖ Goals for Roofline:

- Provide everyone (especially undergrads) 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

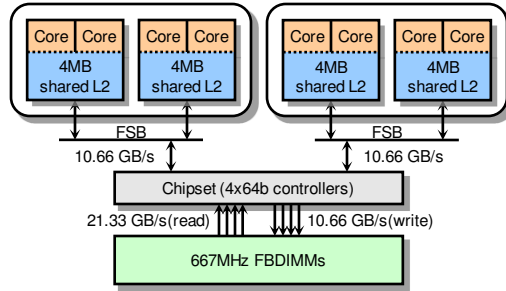
## ❖ Who's not the audience for the Roofline:

- Not for those interested in fine tuning (+10%)
- Not for those challenged by parallel kernel correctness

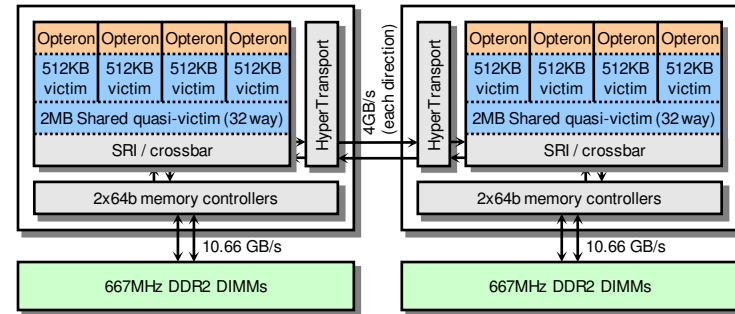
# Multicore SMPs of Interest

*(used throughout the rest of the talk)*

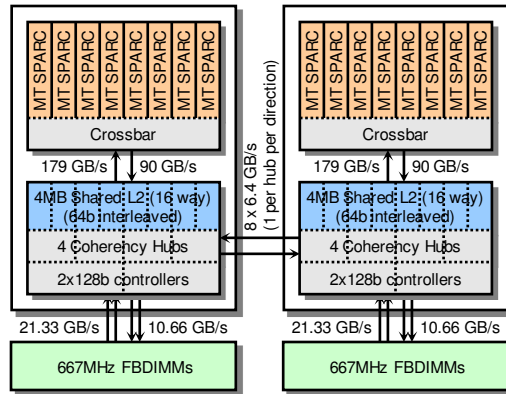
### Intel Xeon E5345 (Clovertown)



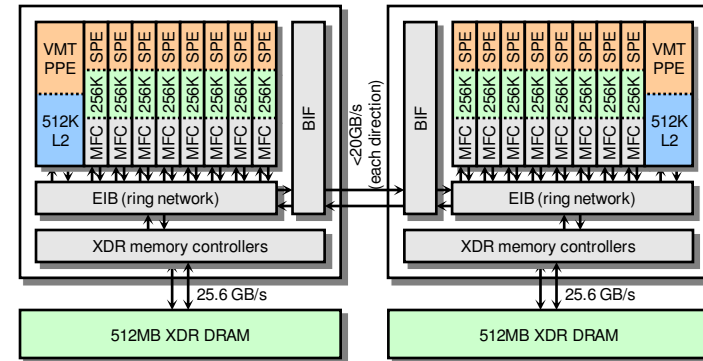
### AMD Opteron 2356 (Barcelona)



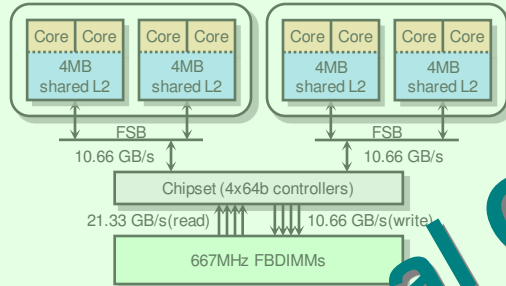
### Sun T2+ T5140 (Victoria Falls)



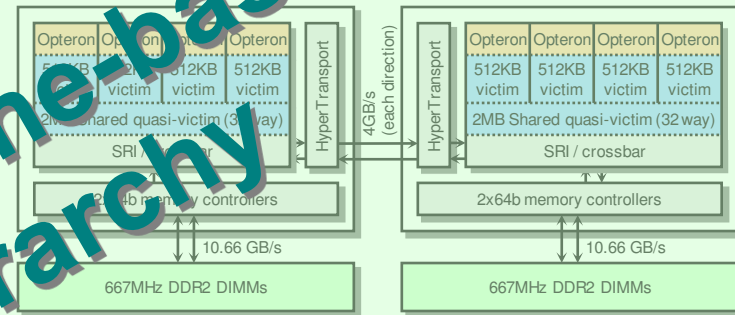
### IBM QS20 Cell Blade



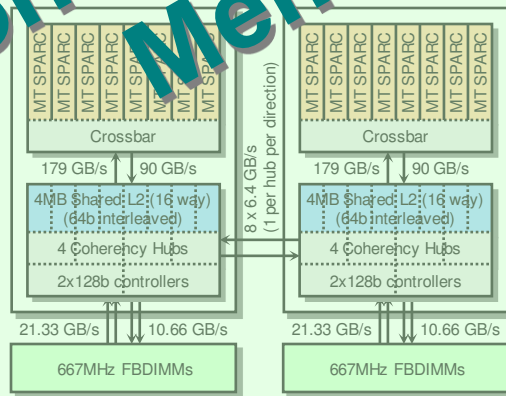
## Intel Xeon E5345 (Clovertown)



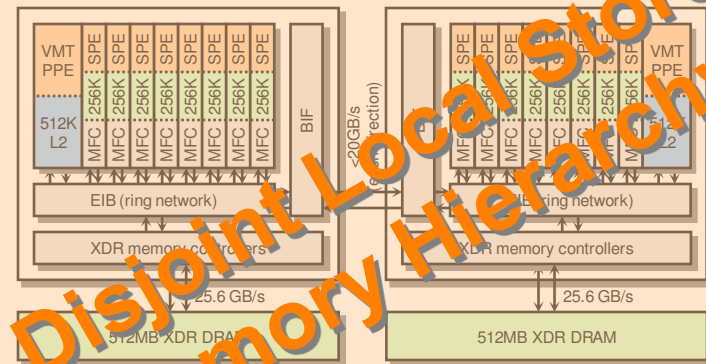
## AMD Opteron 2356 (Barcelona)



## Sun T9-45140 (Victoria Falls)



## IBM QS20 Cell Blade

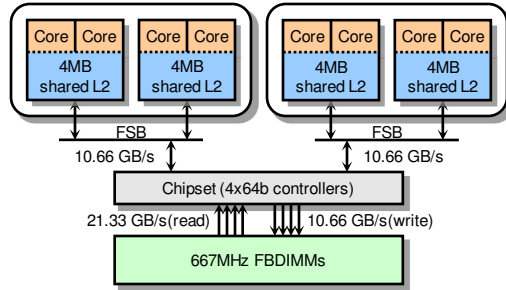


Conventional Cache-based  
Memory Hierarchy

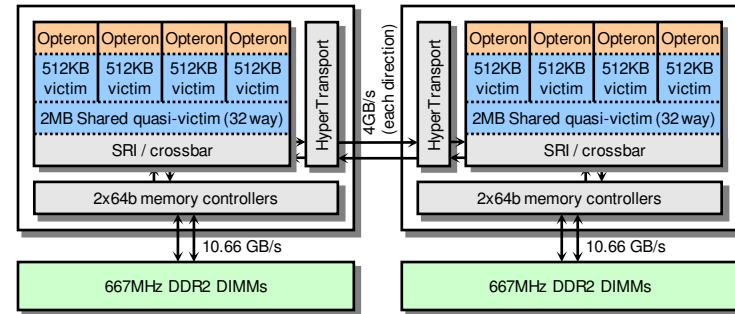
Disjoint Local Store  
Memory Hierarchy



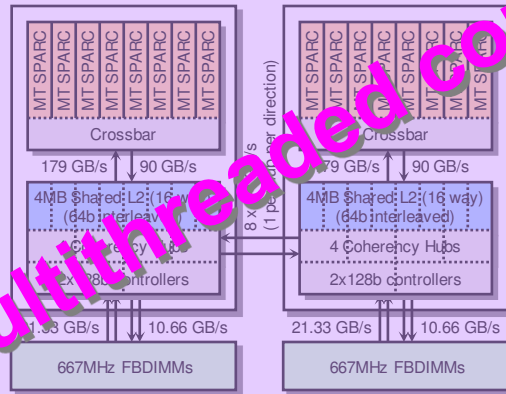
## Intel Xeon E5345 (Clovertown)



## AMD Opteron 2356 (Barcelona)

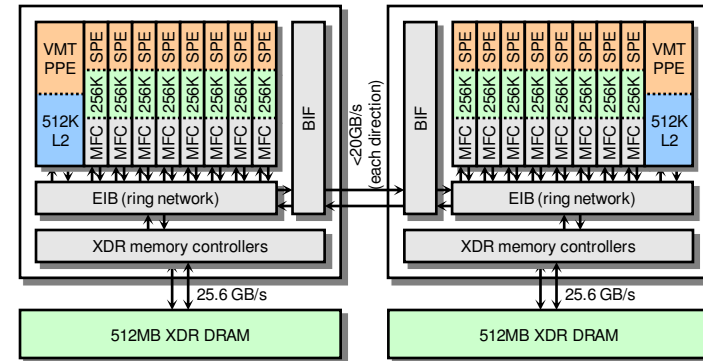


## Sun T2+ T5140 (Victoria Falls)

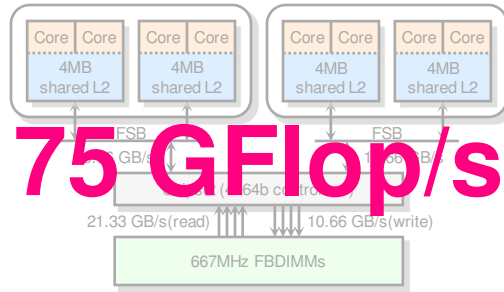


**multi-ported cores**

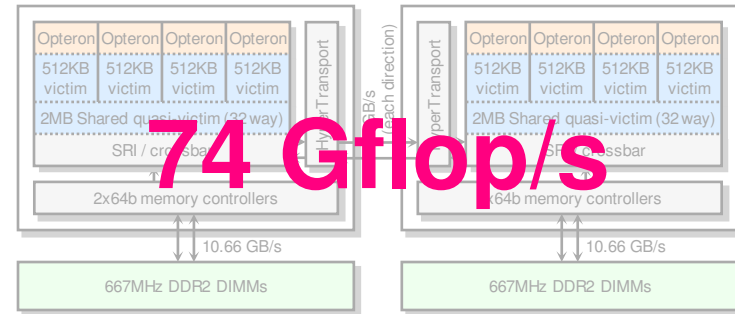
## IBM QS20 Cell Blade



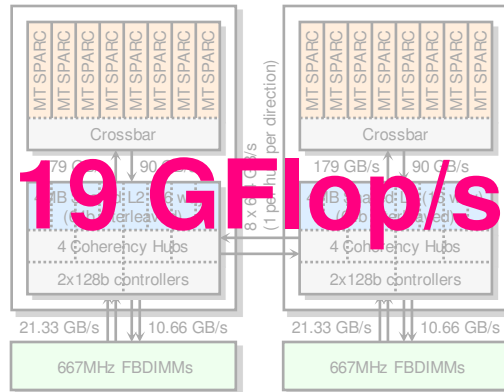
Intel Xeon E5345 (Clovertown)



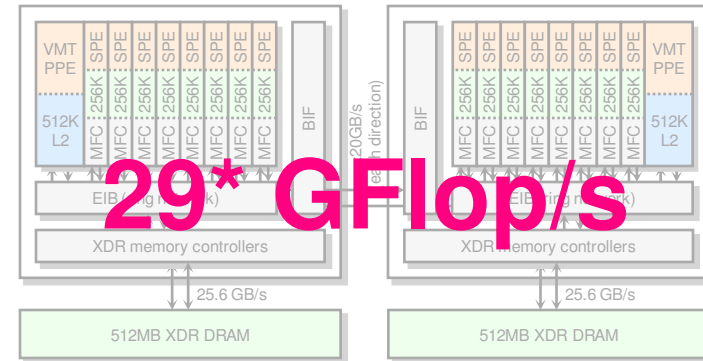
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

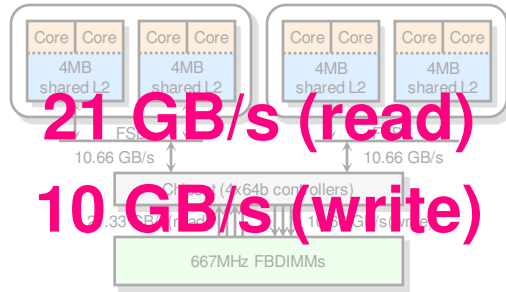


\*SPEs only

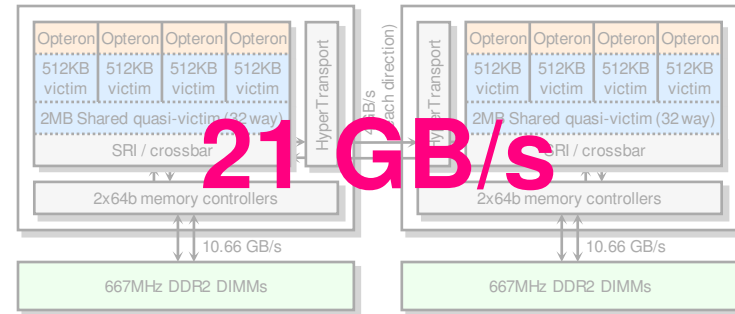
# Multicore SMPs Used

(total DRAM bandwidth)

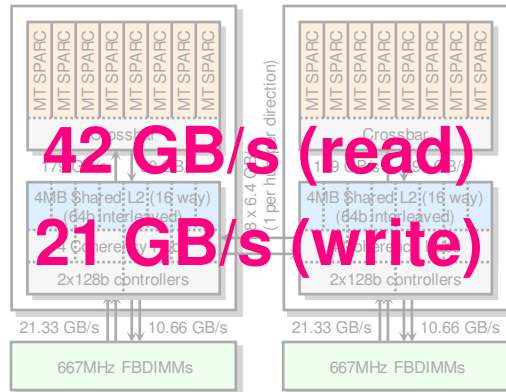
Intel Xeon E5345 (Clovertown)



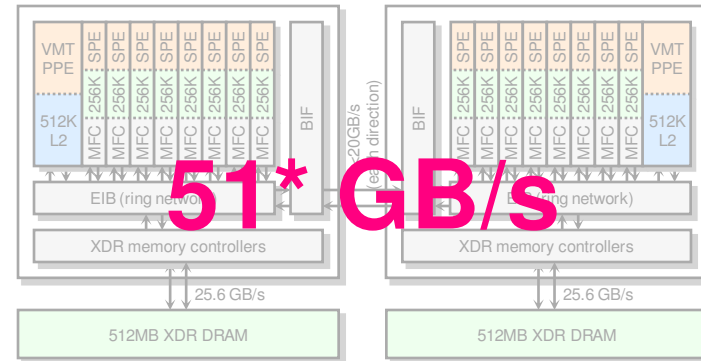
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



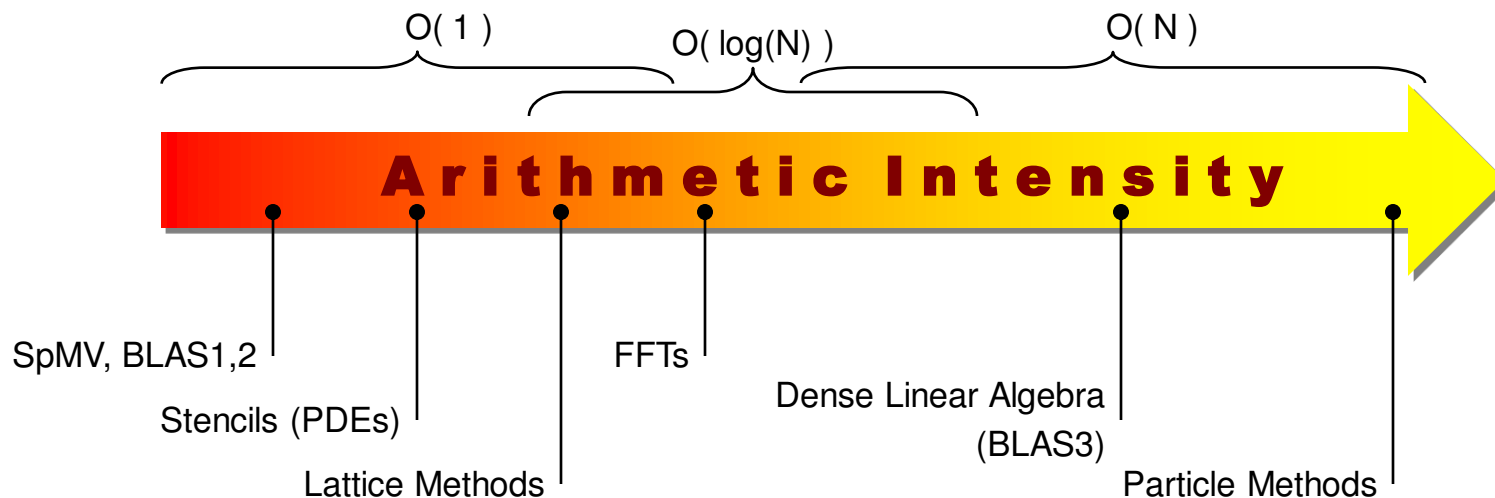
IBM QS20 Cell Blade



\*SPEs only

# Roofline models for multicore SMPs

*(for memory-intensive double precision floating-point kernels)*



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

- constant with respect to problem size for many problems of interest
- ultimately limited by compulsory traffic
- diminished by conflict or capacity misses.

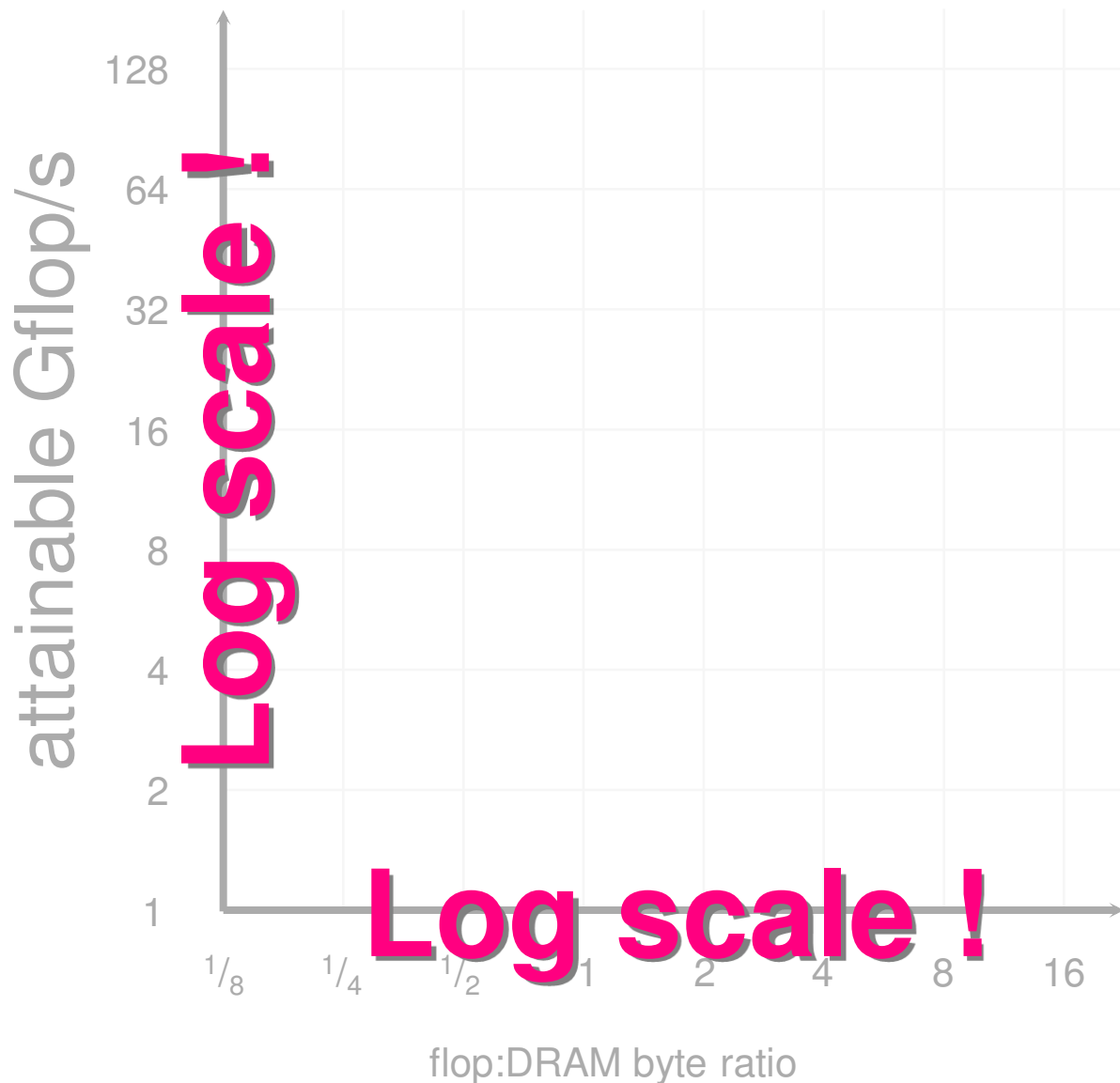
- ❖ Based on **Bound and Bottleneck analysis**<sup>1</sup>
- ❖ Performance is upper bounded by both the peak flop rate, and the product of streaming bandwidth and the flop:byte ratio
- ❖ (well understood in the performance oriented communities)

$$\text{Gflop/s}(AI) = \min \left\{ \begin{array}{l} \text{Peak Gflop/s} \\ AI * \text{StreamBW} \end{array} \right.$$

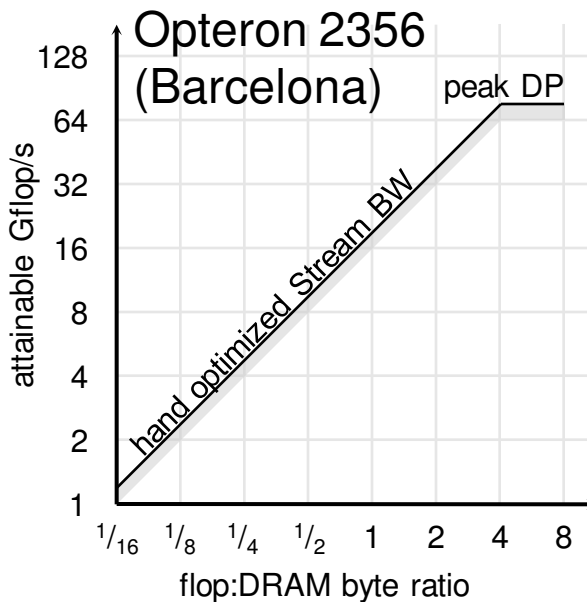
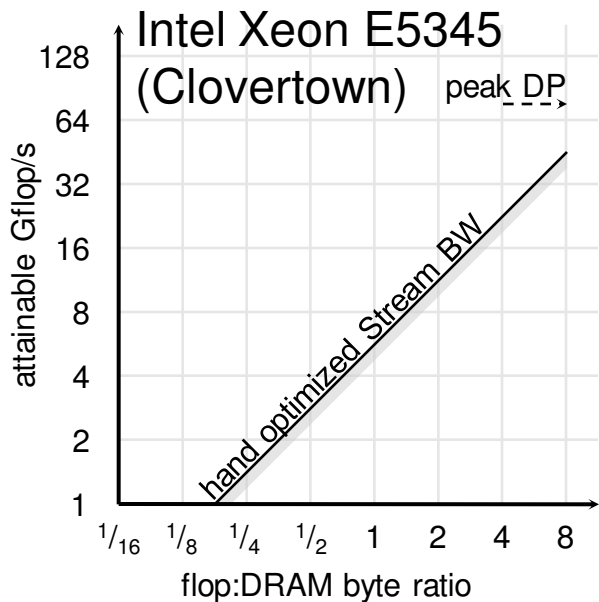
*where AI is the actual arithmetic intensity*

- ❖ Assumptions:
  - Bandwidth is independent on arithmetic intensity
  - Bandwidth is independent of optimization or access pattern
  - Computation is independent of optimization
  - Complete overlap of either communication or computation

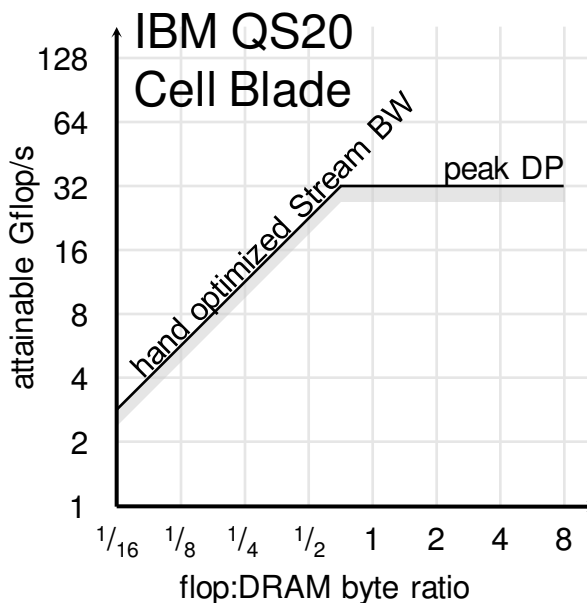
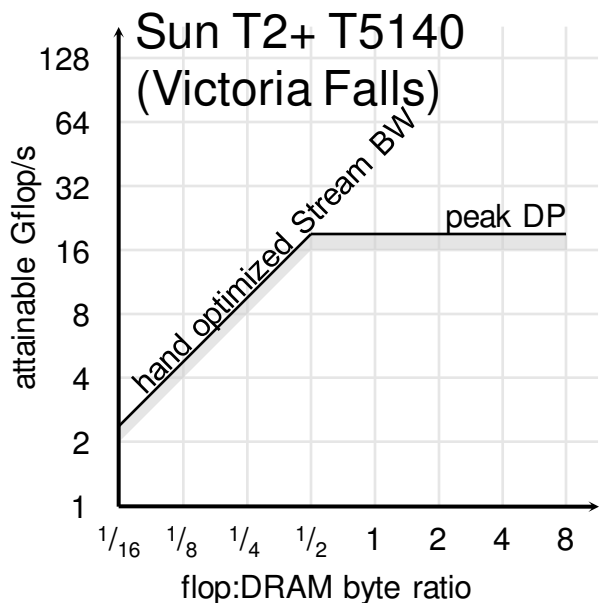
<sup>1</sup>*D. Lazowska, J. Zahorjan, G. Graham, K. Sevcik,  
“Quantitative System Performance”*



# Naïve Roofline Model



- ❖ Unrealistically optimistic model
- ❖ Hand optimized Stream BW benchmark

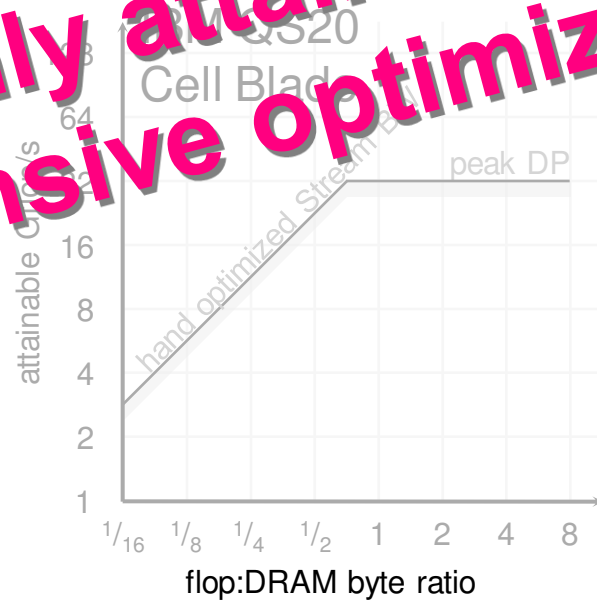
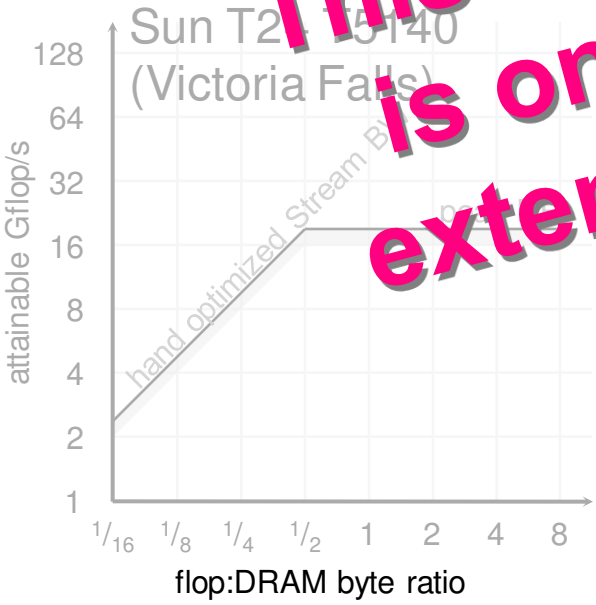




# Naïve Roofline Model

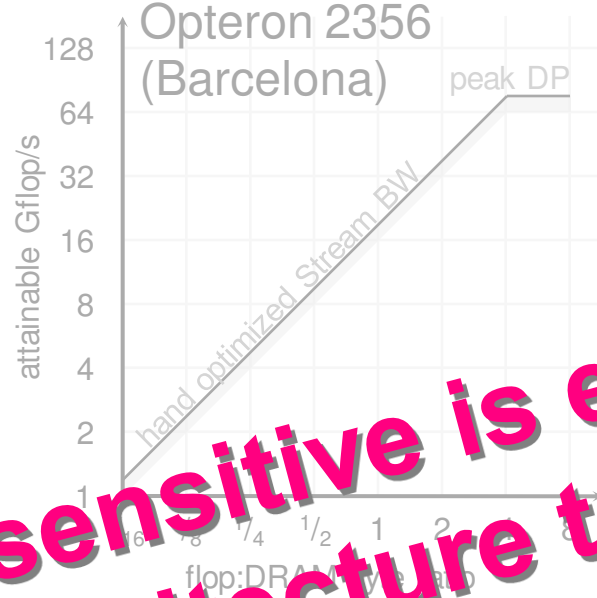


- ❖ Unrealistically optimistic model
- ❖ Hand optimized Stream BW benchmark

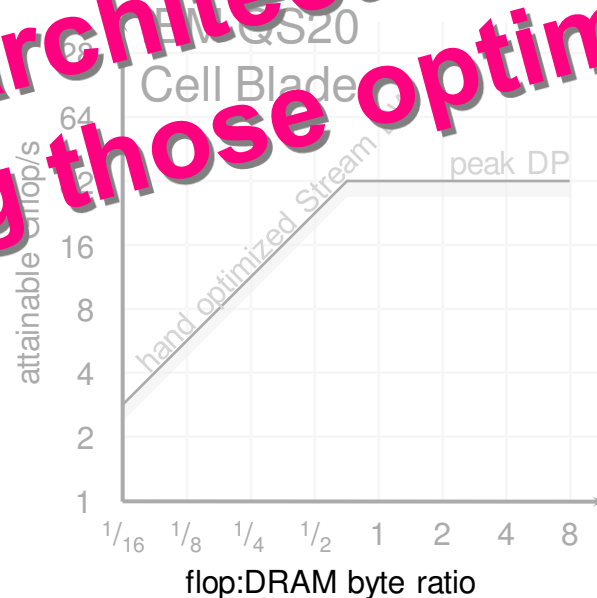
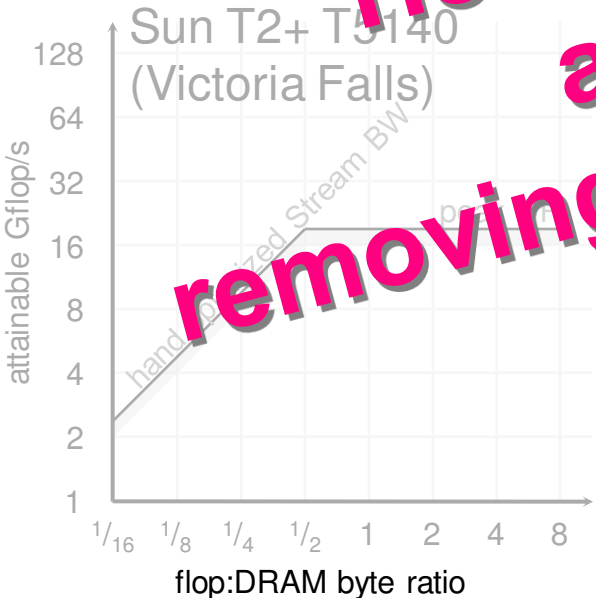


**This level of performance is only attainable with extensive optimizations**

# Naïve Roofline Model



- ❖ Unrealistically optimistic model
- ❖ Hand optimized Stream BW benchmark



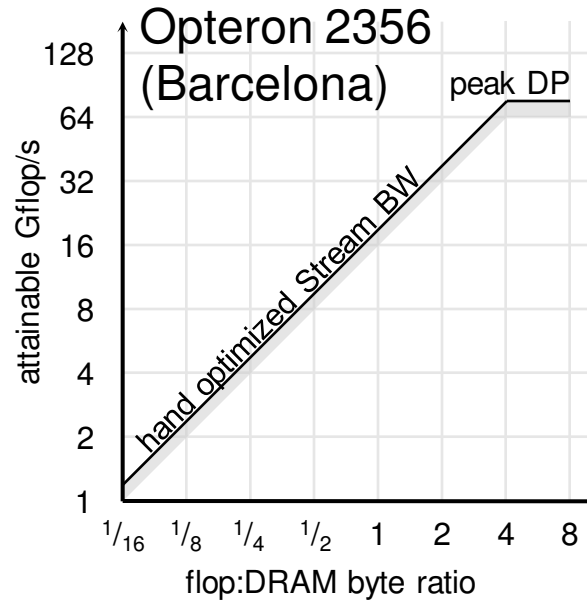
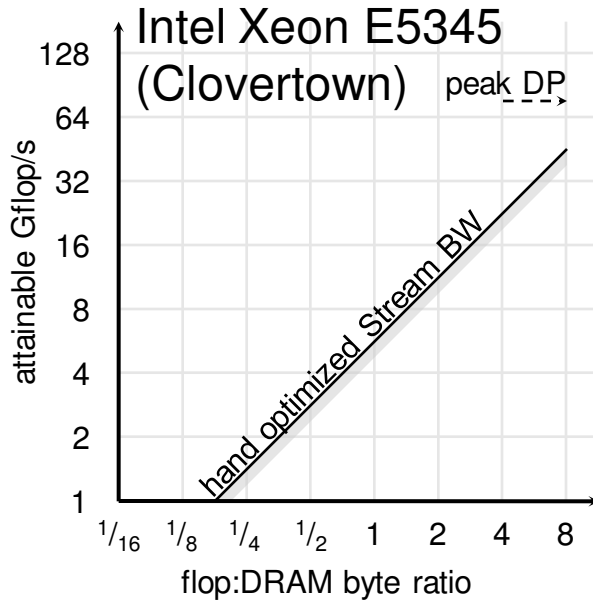
**How sensitive is each architecture to removing those optimizations?**

- ❖ Collect **StreamBW<sub>j</sub>** with progressively fewer optimizations
- ❖ Estimate **InCoreGFlops<sub>i</sub>** with progressively fewer optimizations

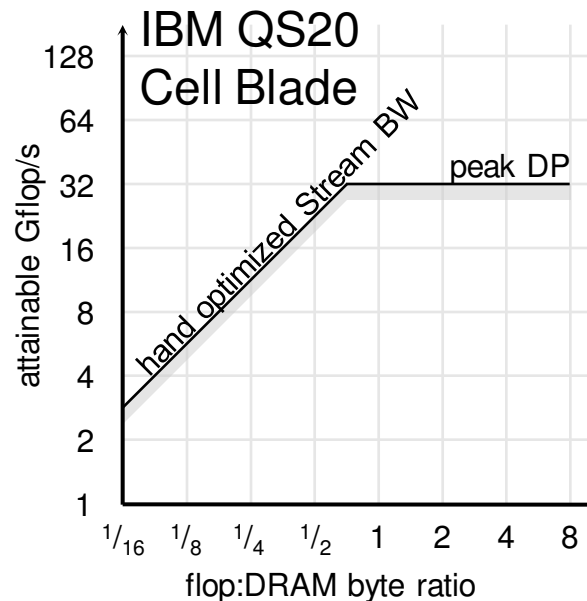
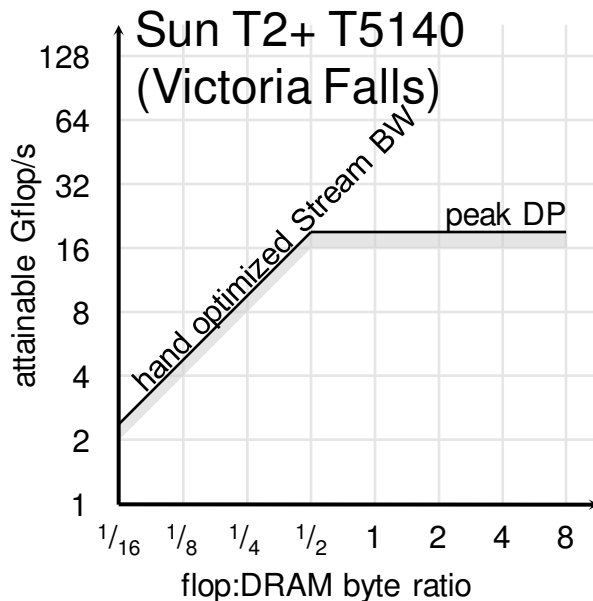
$$\text{GFlops}_{i,j}(\text{AI}) = \min \left\{ \begin{array}{l} \text{InCoreGFlops}_i \\ \text{AI} * \text{StreamBW}_j \end{array} \right.$$

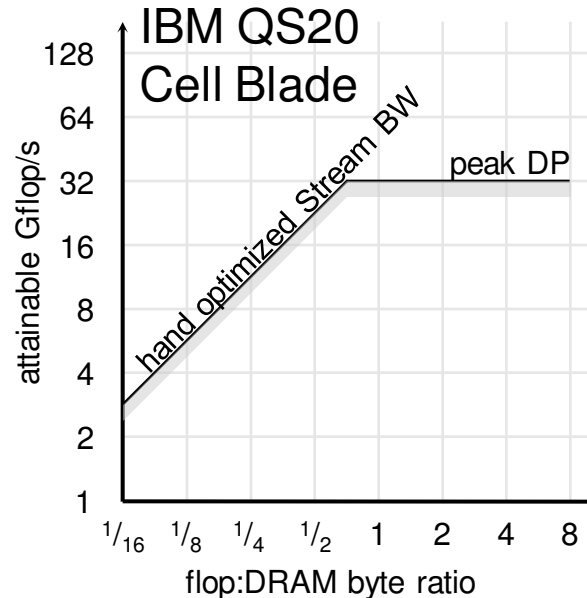
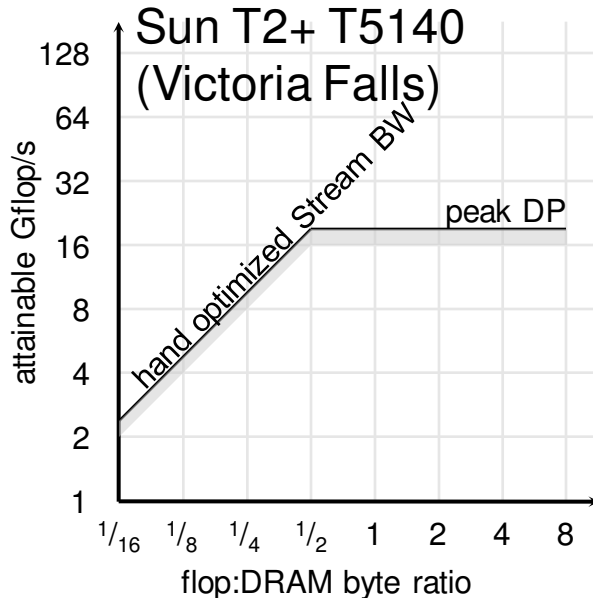
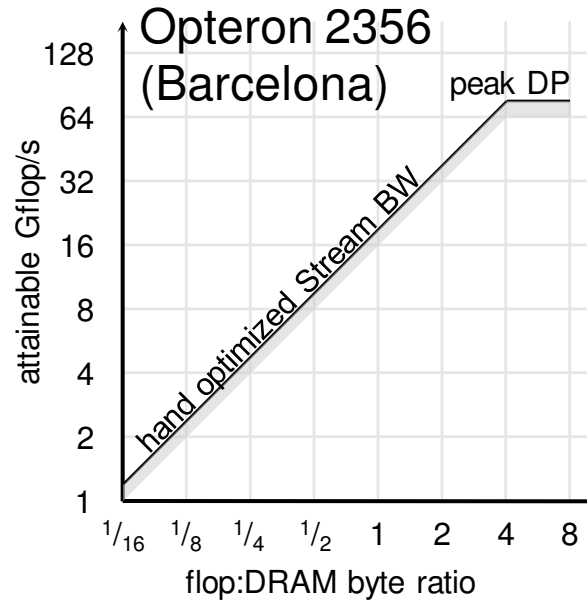
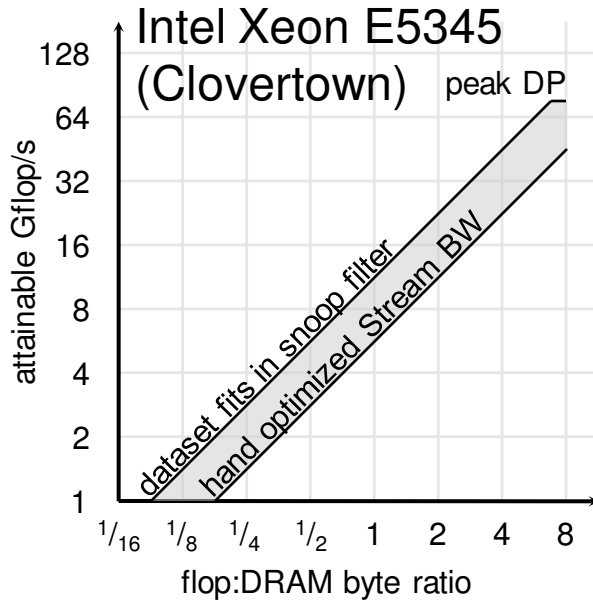
*is the attainable performance with:  
memory optimizations<sub>1...i</sub> - and -  
in-core optimizations<sub>1..j</sub>*

- ❖ These denote a series of **ceilings** below the roofline
- ❖ Assumptions:
  - Bandwidth is independent on arithmetic intensity
  - Complete overlap of either communication or computation

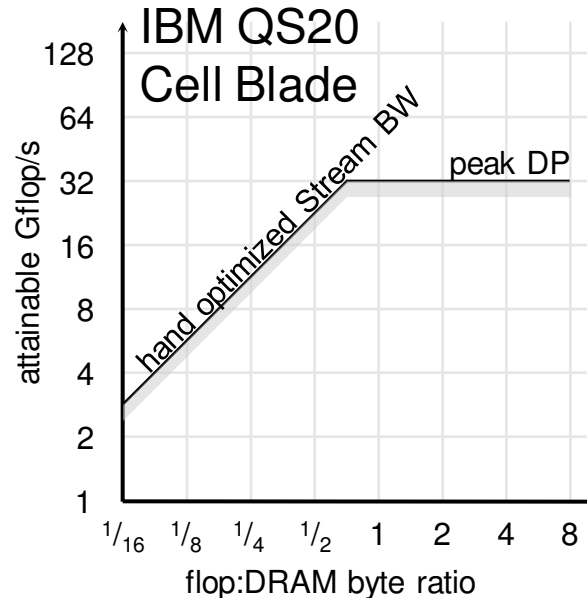
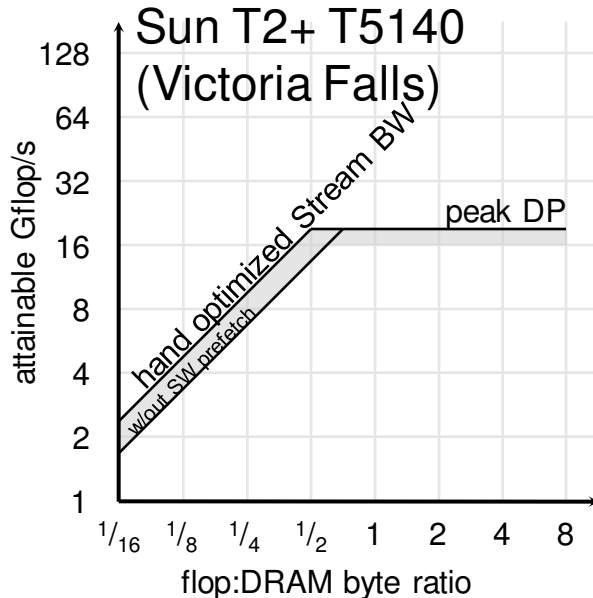
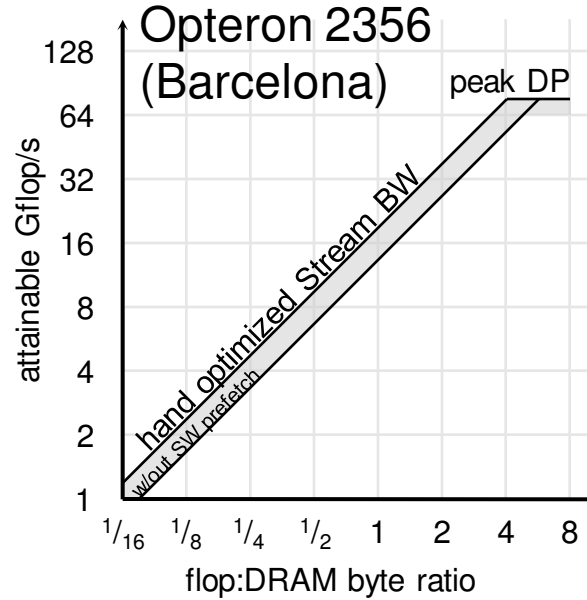
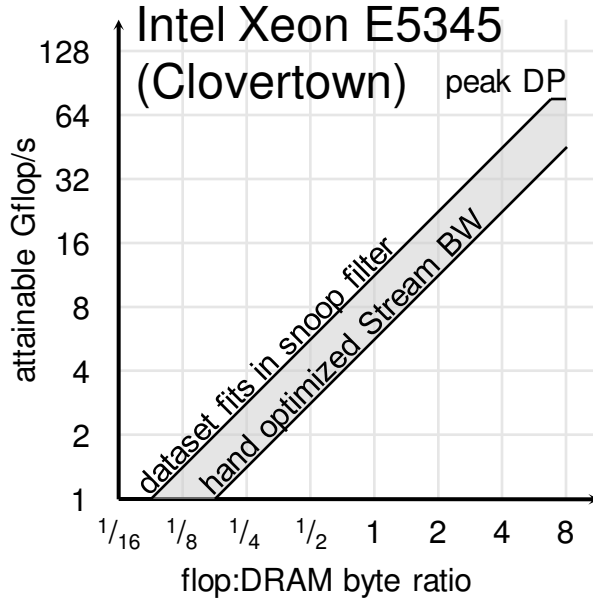


- ❖ What happens as bandwidth optimizations are stripped out ?
- ❖ Form a series of bandwidth ceilings below the roofline
- ❖ Small problems fit in the snoop filter in Clovertown's MCH
- ❖ most architectures see NUMA and prefetch variations

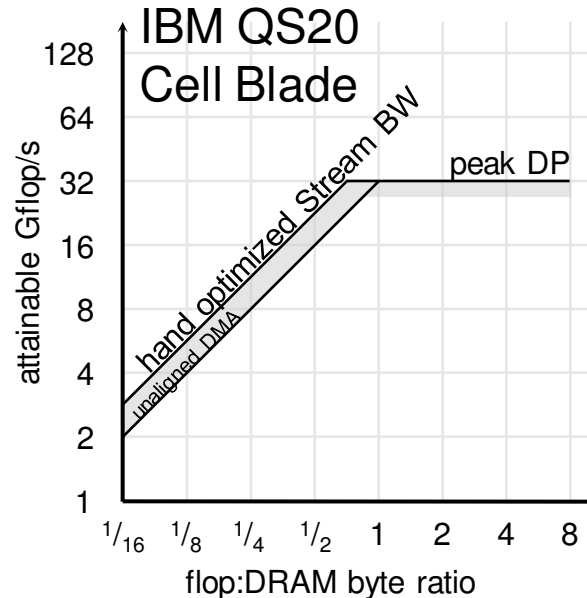
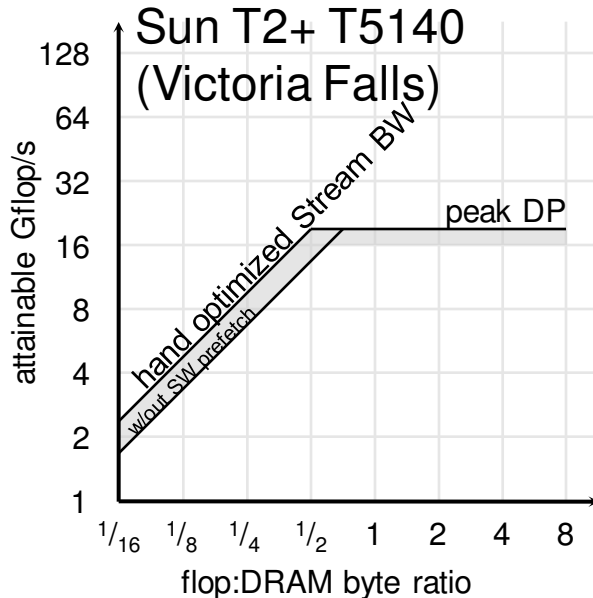
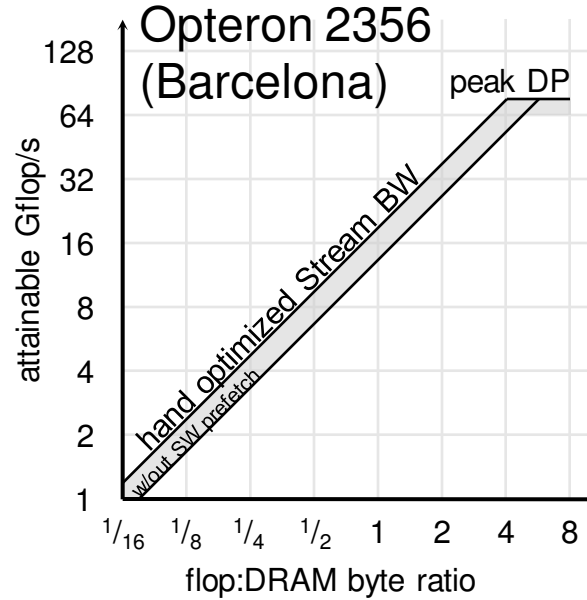
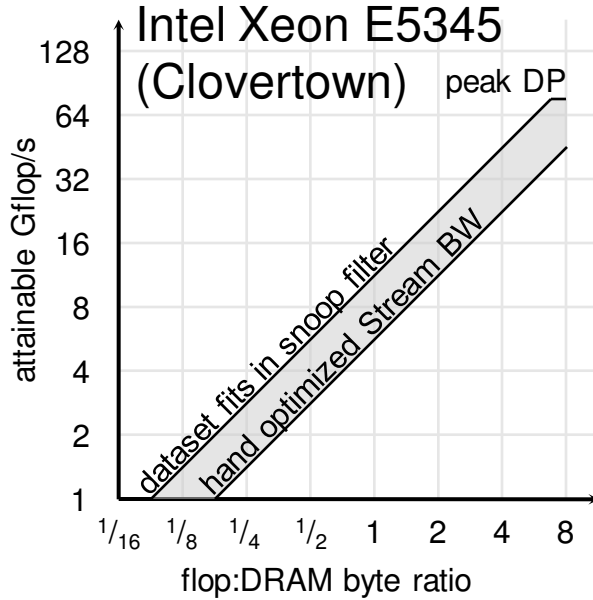




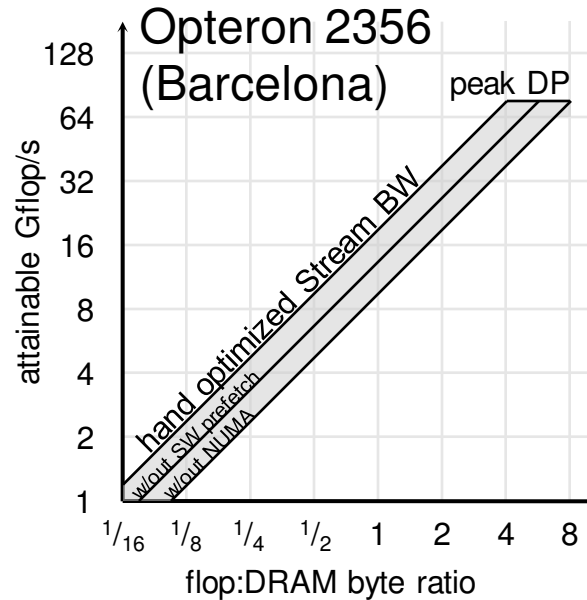
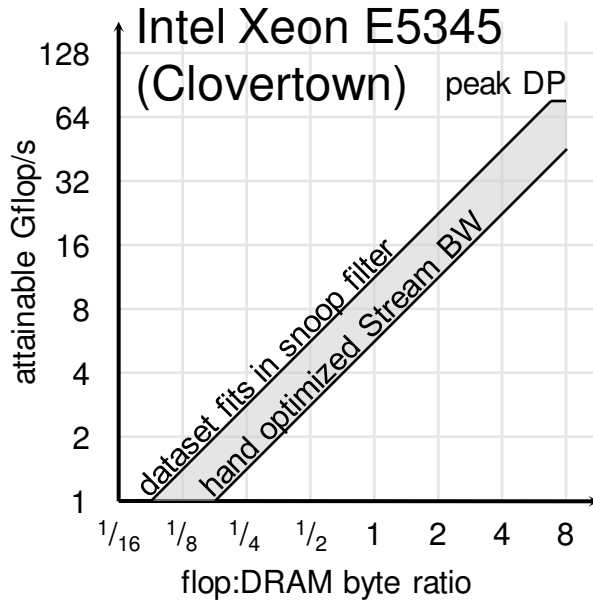
- ❖ What happens as bandwidth optimizations are stripped out ?
- ❖ Form a series of bandwidth ceilings below the roofline
- ❖ Small problems fit in the snoop filter in Clovertown's MCH
- ❖ most architectures see NUMA and prefetch variations



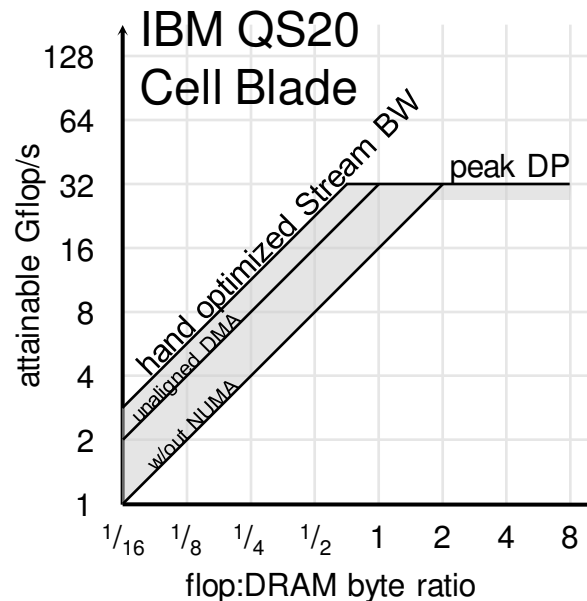
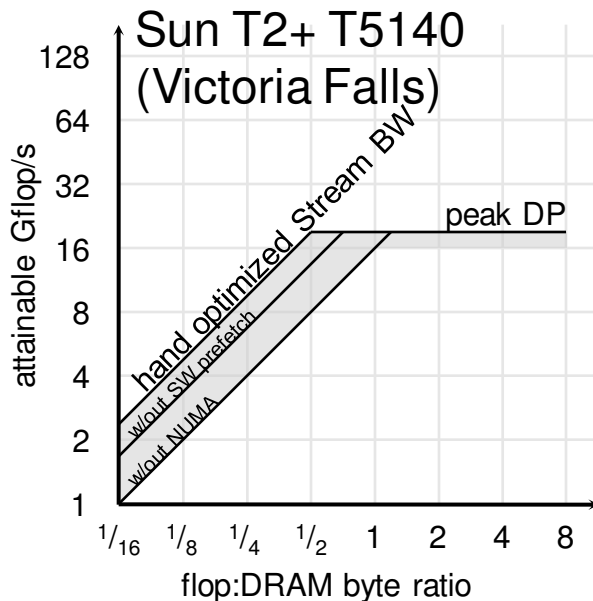
- ❖ What happens as bandwidth optimizations are stripped out ?
- ❖ Form a series of bandwidth ceilings below the roofline
- ❖ Small problems fit in the snoop filter in Clovertown's MCH
- ❖ most architectures see NUMA and prefetch variations



- ❖ What happens as bandwidth optimizations are stripped out ?
- ❖ Form a series of bandwidth ceilings below the roofline
- ❖ Small problems fit in the snoop filter in Clovertown's MCH
- ❖ most architectures see NUMA and prefetch variations

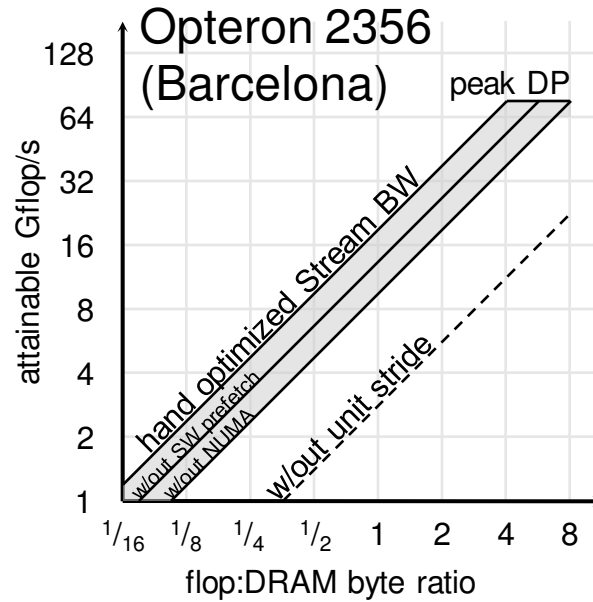
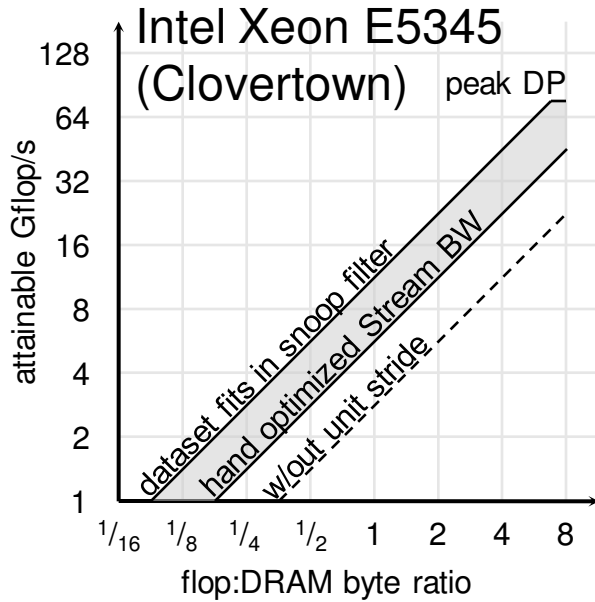


- ❖ What happens as bandwidth optimizations are stripped out ?
- ❖ Form a series of bandwidth ceilings below the roofline
- ❖ Small problems fit in the snoop filter in Clovertown's MCH
- ❖ most architectures see NUMA and prefetch variations

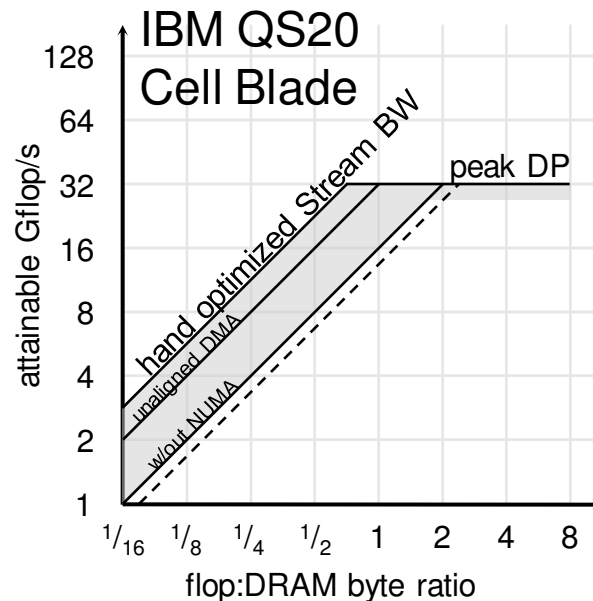
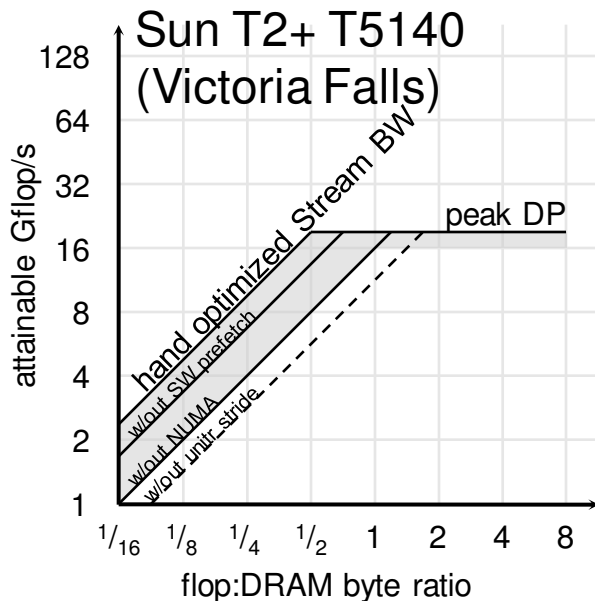




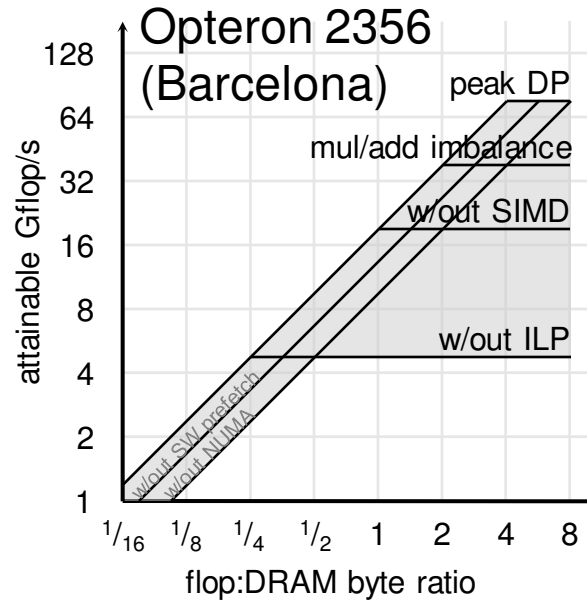
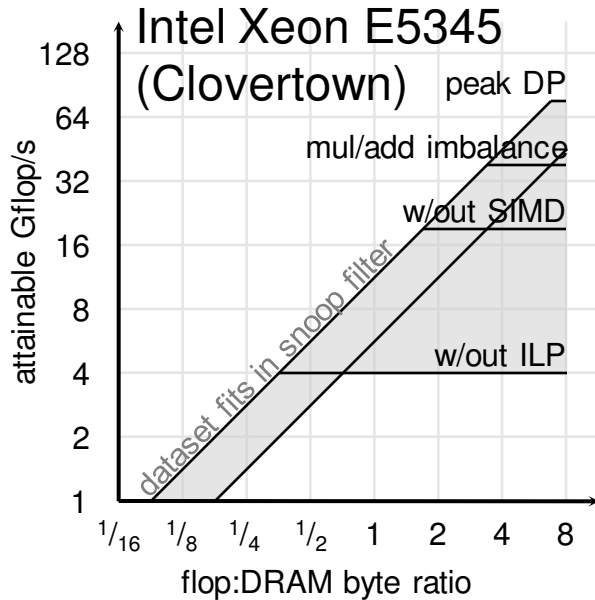
# Roofline models (dram bandwidth)



- ❖ What happens as bandwidth optimizations are stripped out ?
- ❖ Form a series of bandwidth ceilings below the roofline
- ❖ Small problems fit in the snoop filter in Clovertown's MCH
- ❖ most architectures see NUMA and prefetch variations



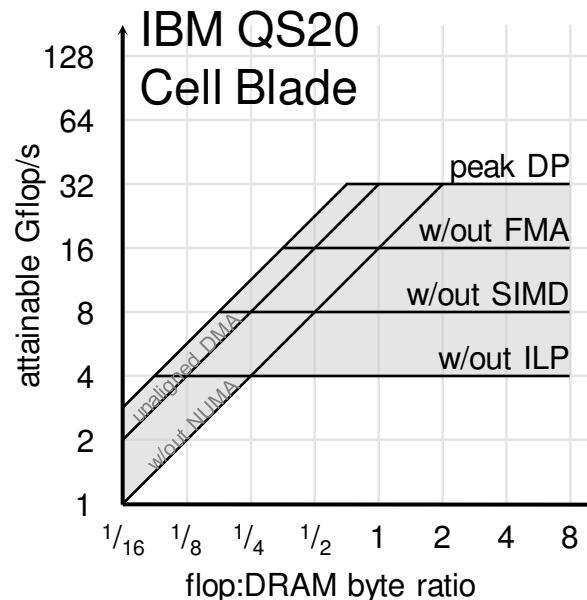
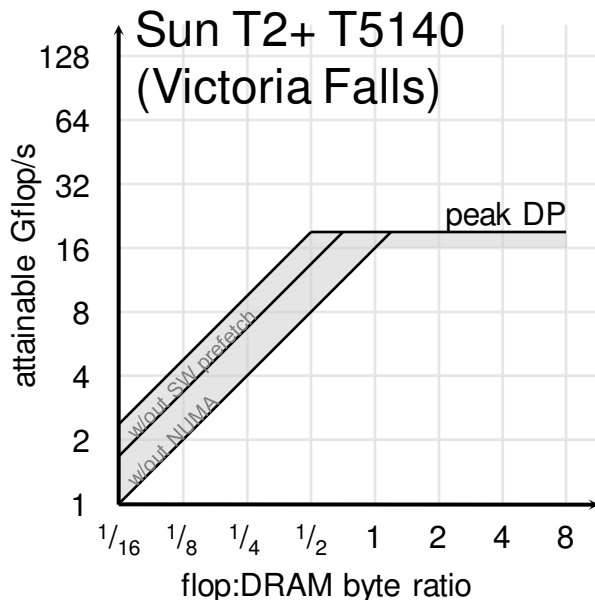
- ❖ Define a similar set of ceilings for in-core performance
  
- ❖ In-core performance can be limited by (among other things):
  - Not satisfying all forms of **in-core parallelism**:
    - Instruction-level parallelism (multi-issue, pipeline, ...)
    - Data-level parallelism (SIMD)
    - Functional unit parallelism (adders + multipliers + ...)
  - Non-FP instructions can consume **instruction issue bandwidth**
    - As the FP fraction decrease, how sensitive is attainable performance?
  
- ❖ One or the other is usually more difficult to satisfy on a given architecture/kernel
  - = **Architecture's Achilles' Heel**

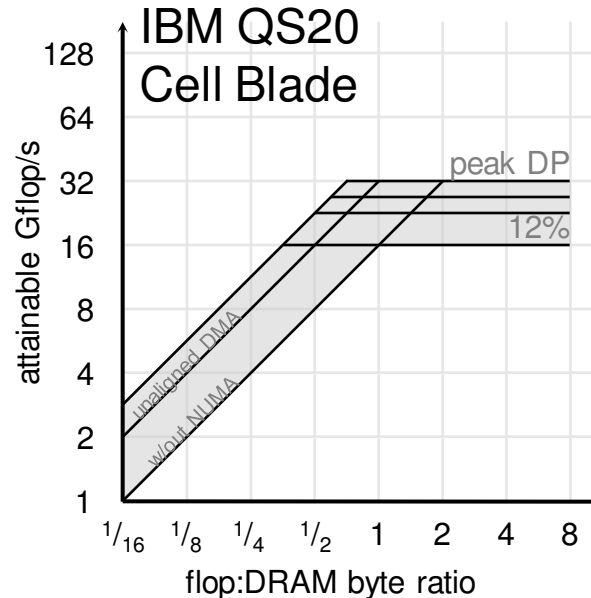
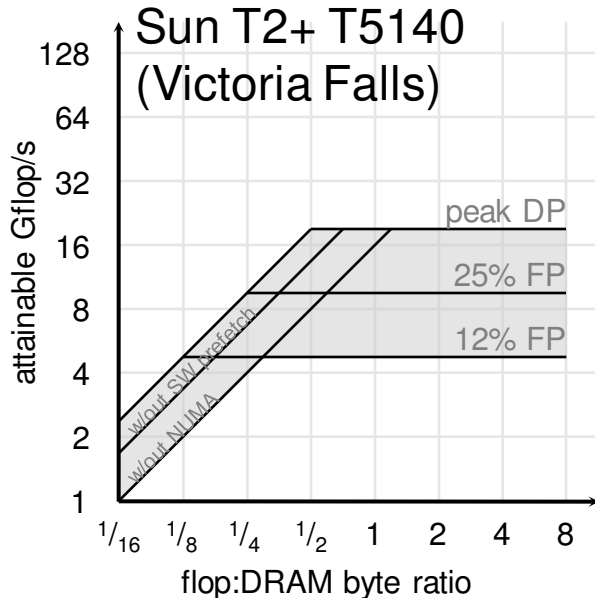
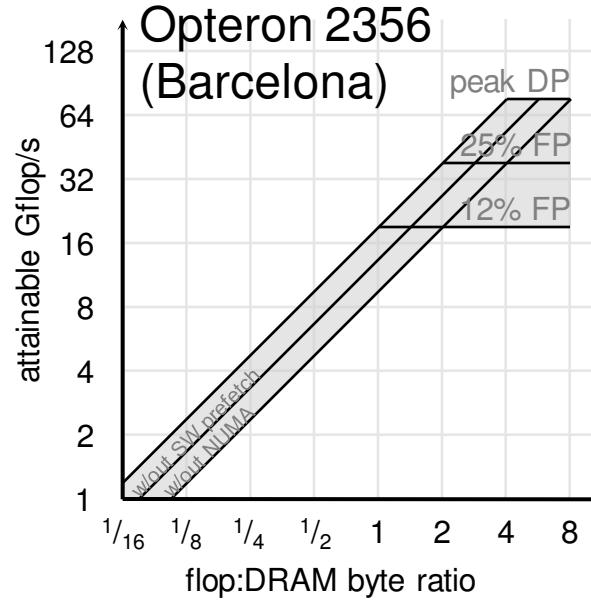
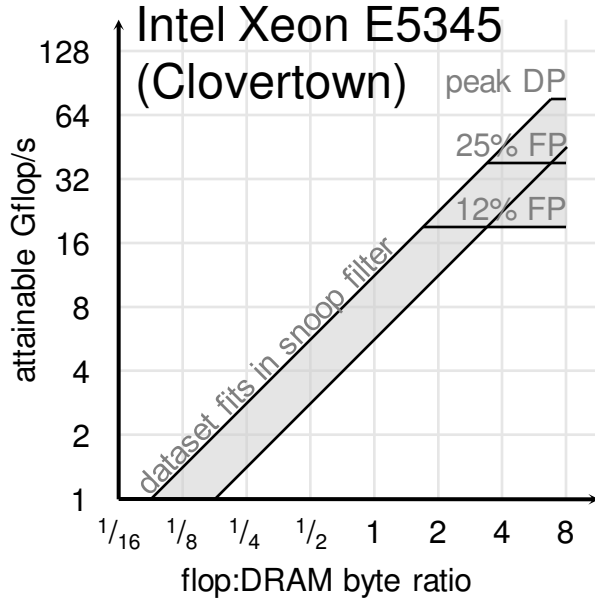


❖ Covering the breadth of in-core parallelism is the preeminent challenge on most architectures

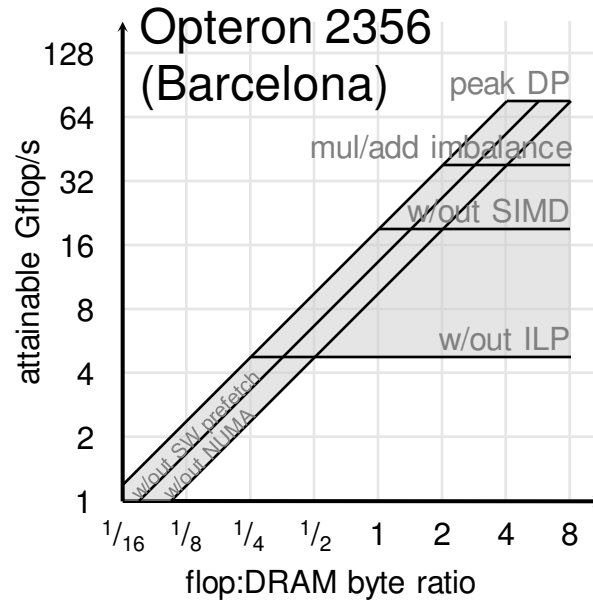
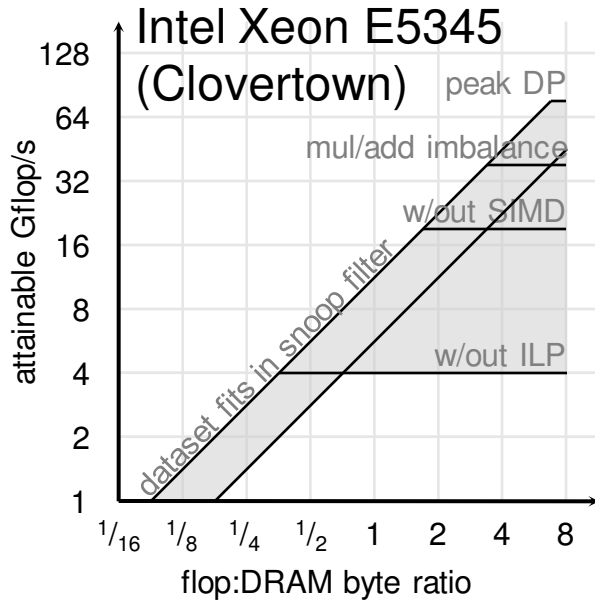
❖ Form a series of parallelism ceilings below the roofline

❖ On Niagara machines, instruction latencies are easily hidden with 8-way multithreading

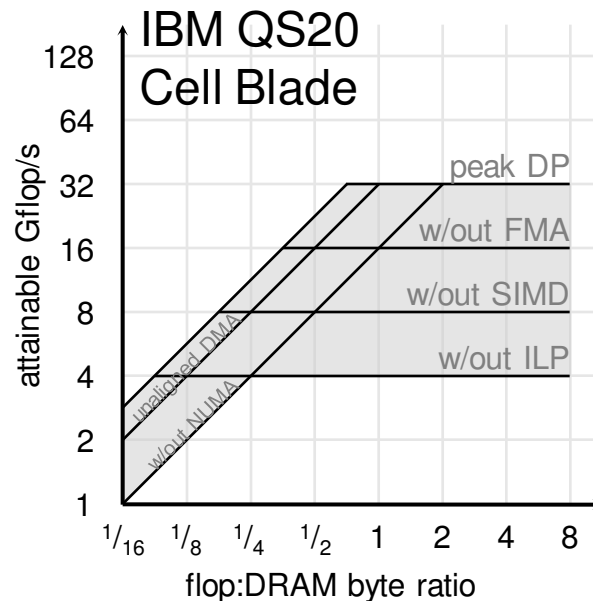
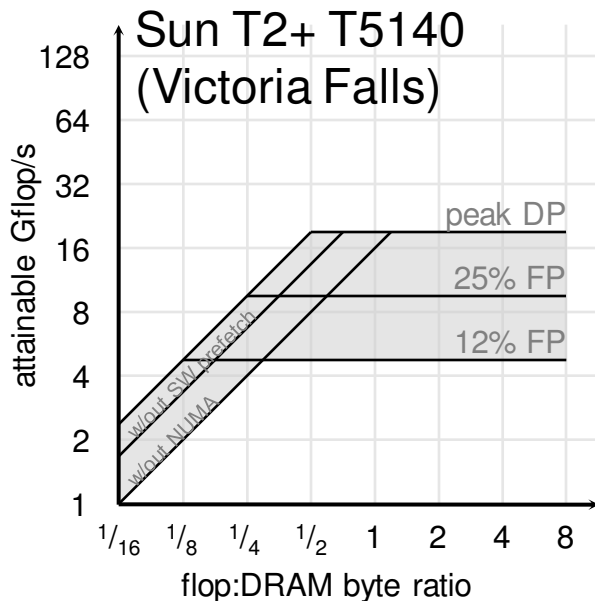




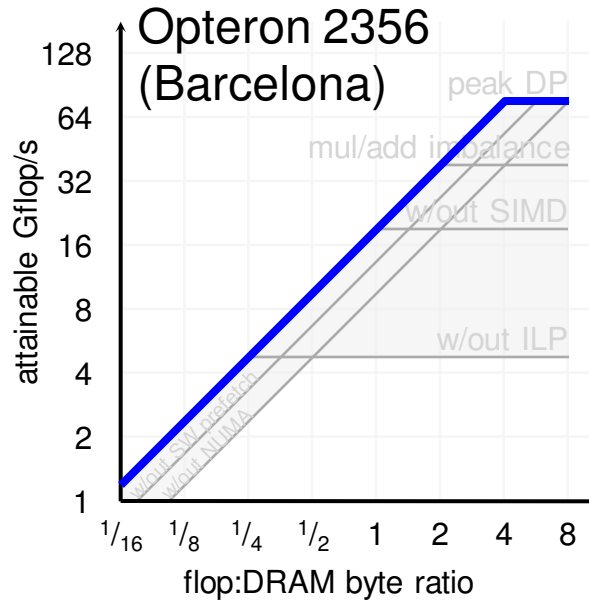
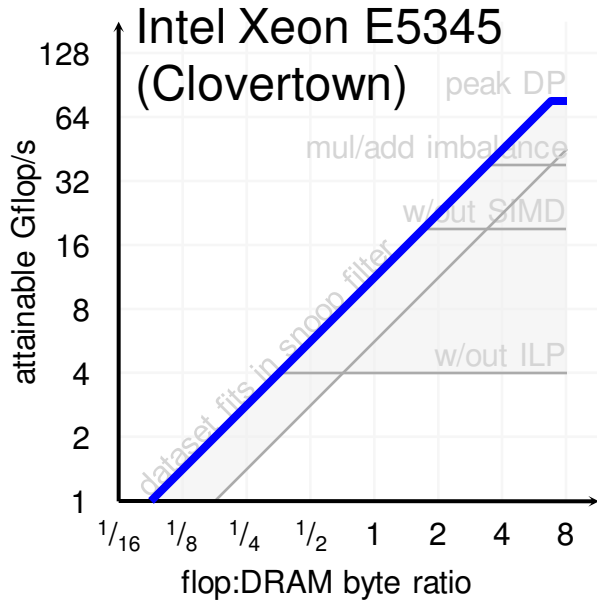
- ❖ All machines have a limited instruction issue bandwidth.
- ❖ non-FP instructions sap instruction issue bandwidth needed by FP instructions
- ❖ As the FP fraction of the dynamic instruction mix decreases, so might performance.
- ❖ On Cell, double precision instructions stall subsequent issues for 7 cycles.



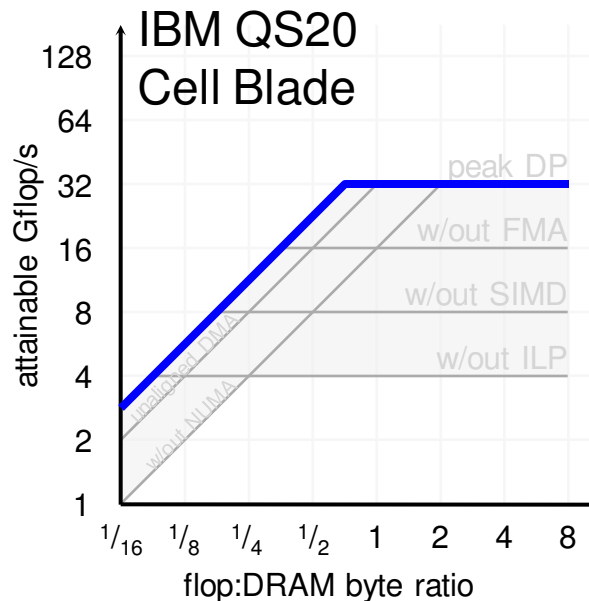
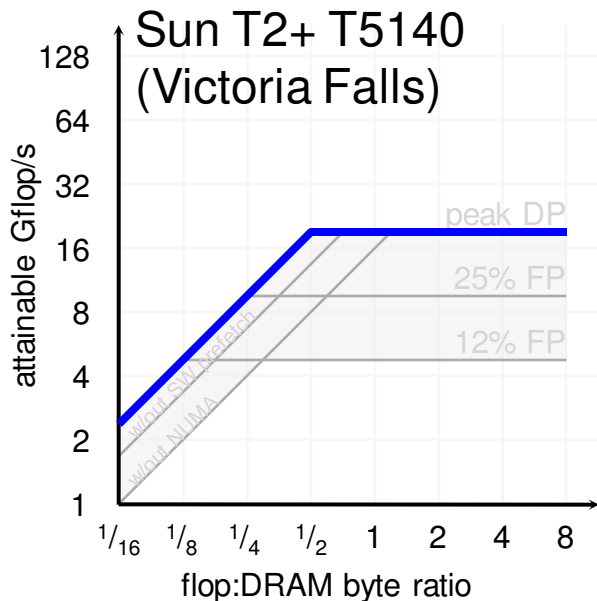
- ❖ Its clear that in-core parallelism is more important on the superscalars
- ❖ Instruction mix is more important on Niagara2
- ❖ Each architecture has its own **Achilles' Heel** when it comes to in-core performance



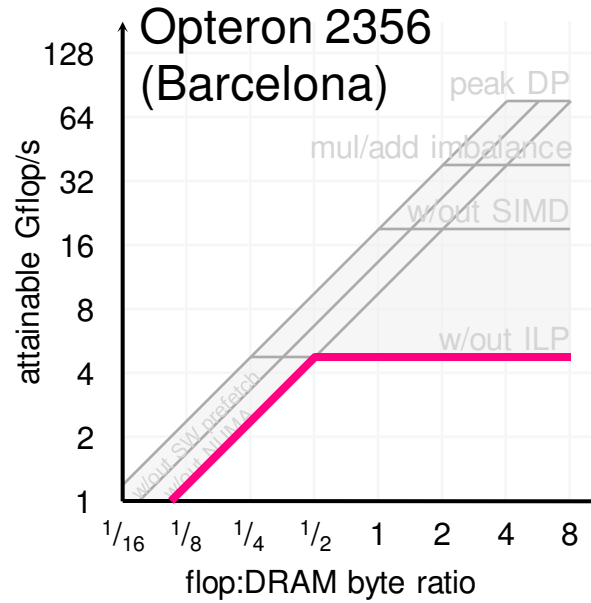
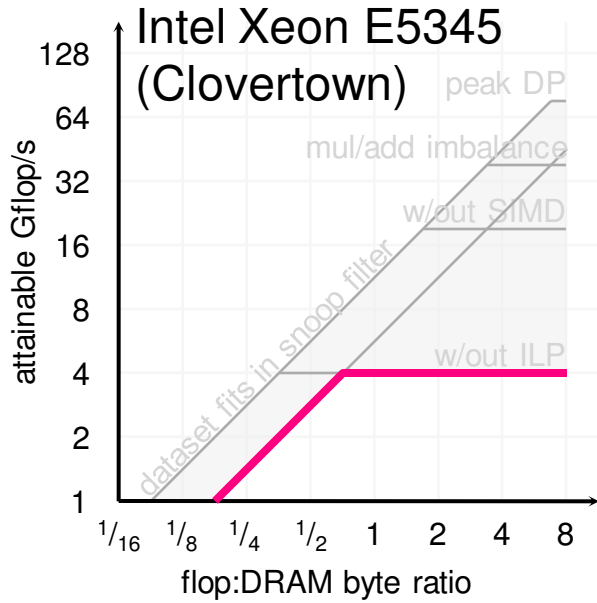
# Roofline models (ceilings constrain performance)



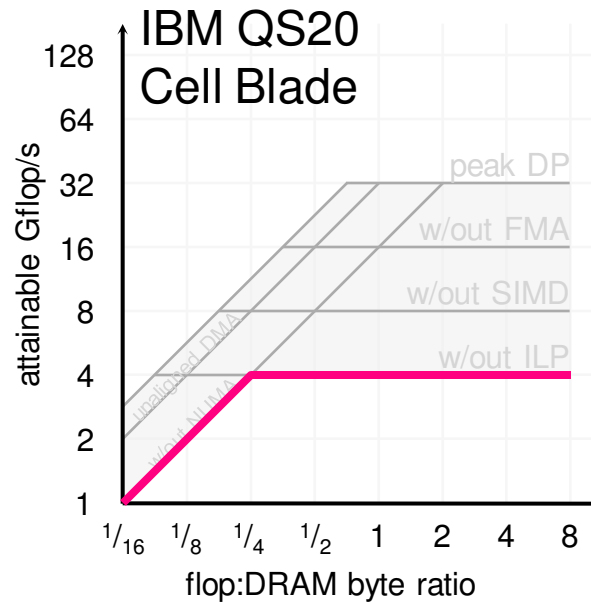
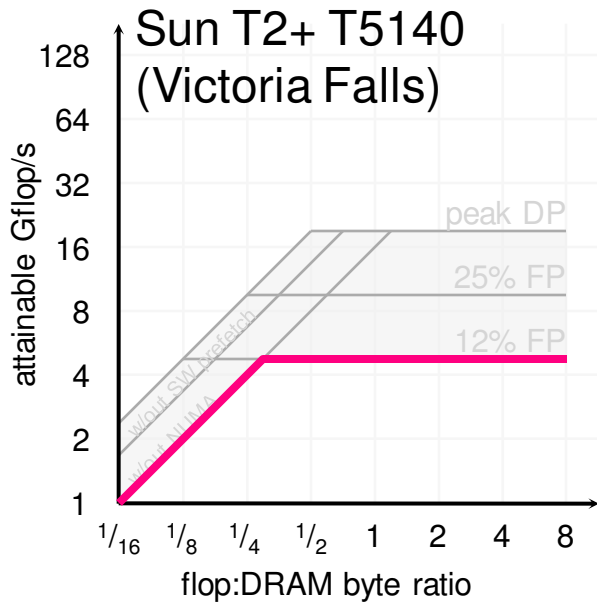
❖ The ceilings act to constrain performance to a much smaller region



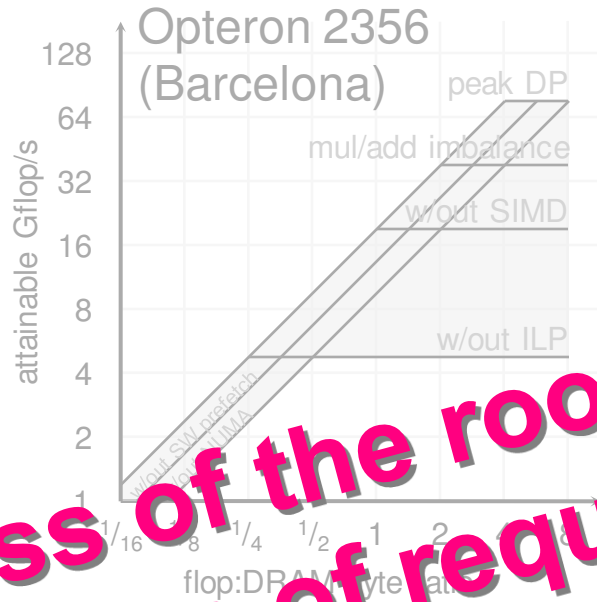
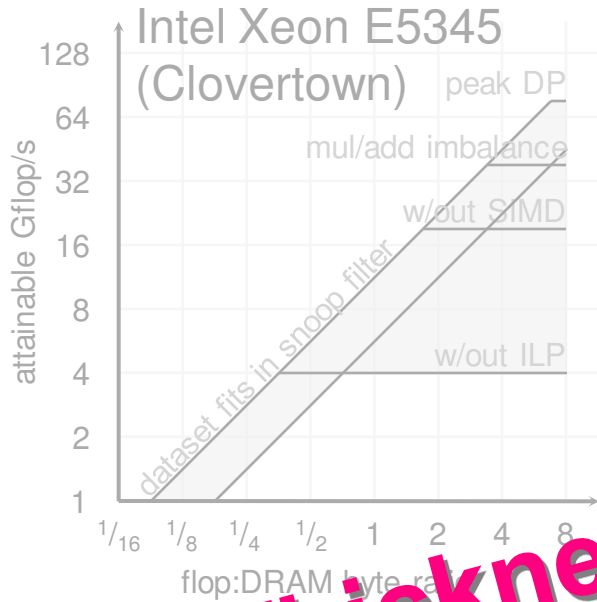
# Roofline models (ceilings constrain performance)



❖ The ceilings act to constrain performance to a much smaller region

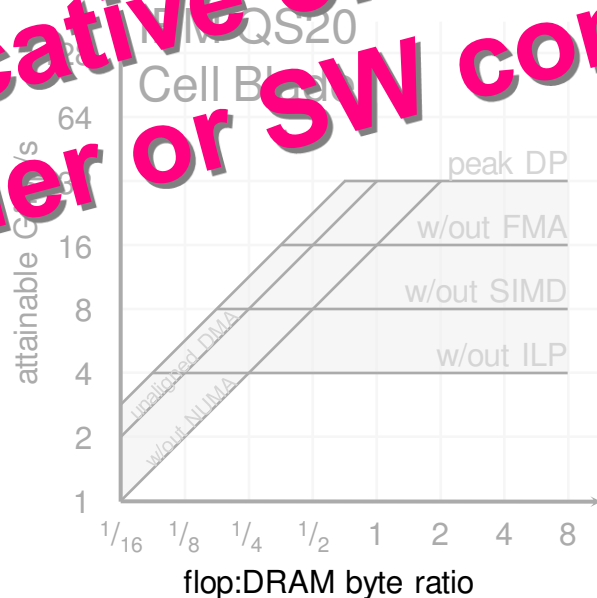
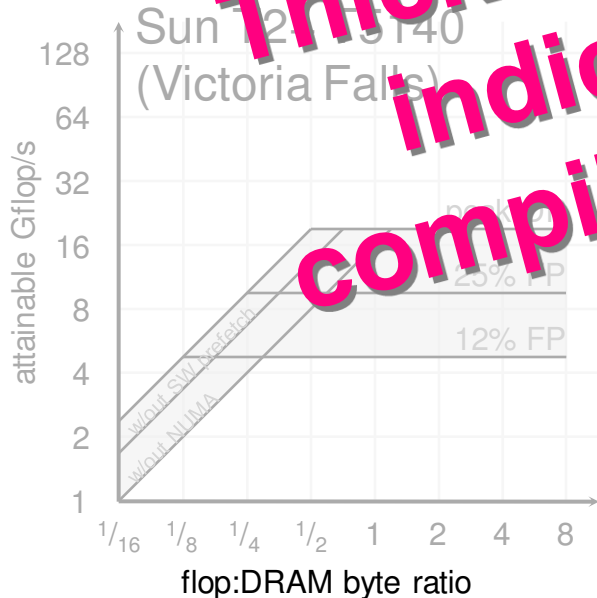


# Roofline models (thickness)



❖ The ceilings act to constrain performance to a much smaller region

**Thickness of the roofline is indicative of requisite compiler or SW complexity**

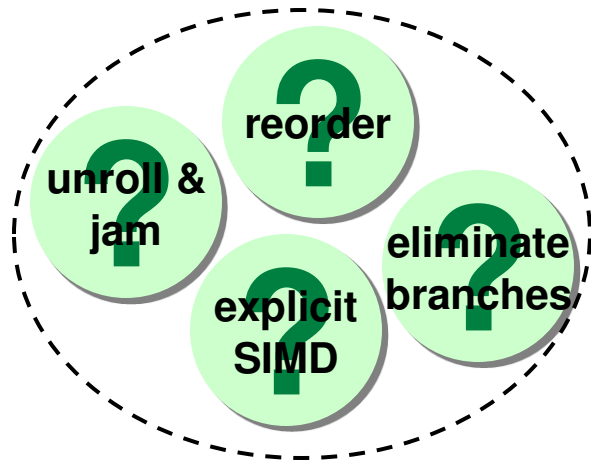




# Three Categories of Software Optimization

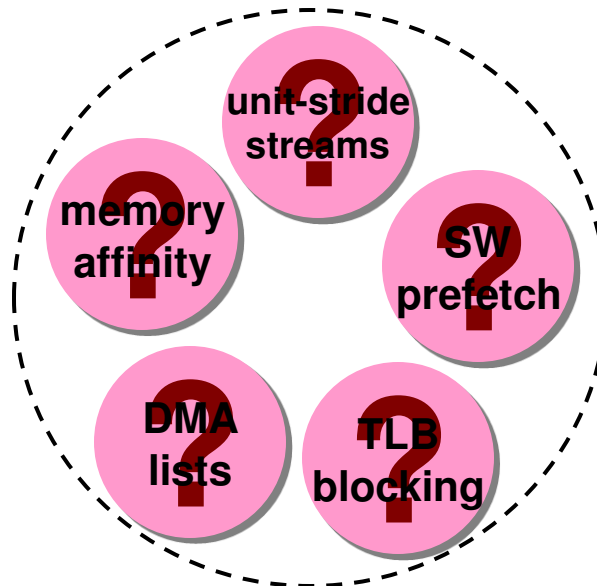
## Maximizing In-core Performance

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



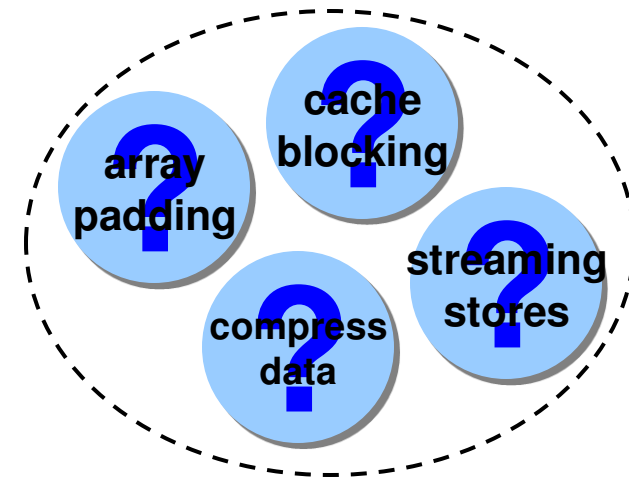
## Maximizing Memory Bandwidth

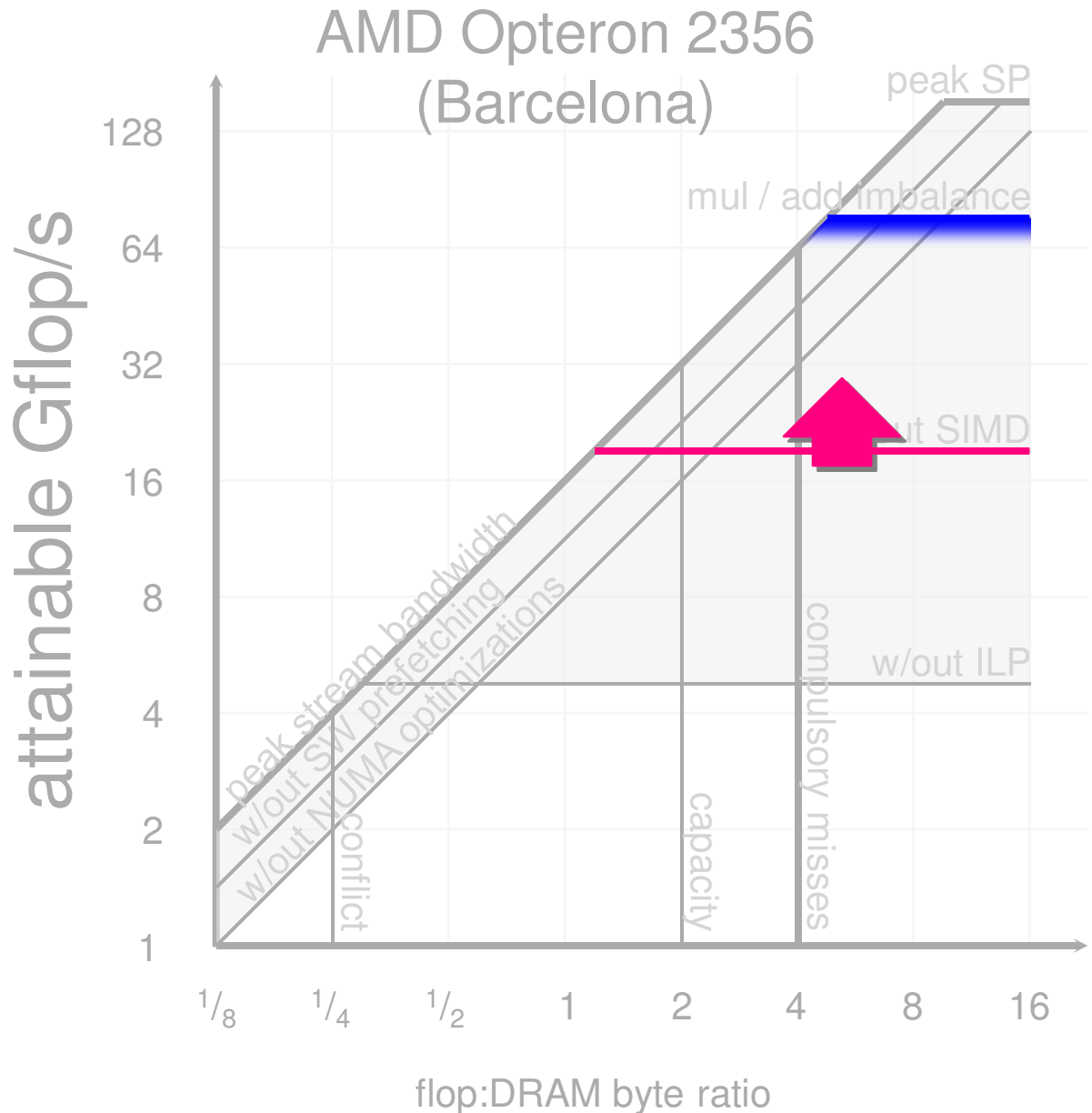
- Exploit NUMA
- Hide memory latency
- Satisfy Little's Law



## Minimizing Memory Traffic

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

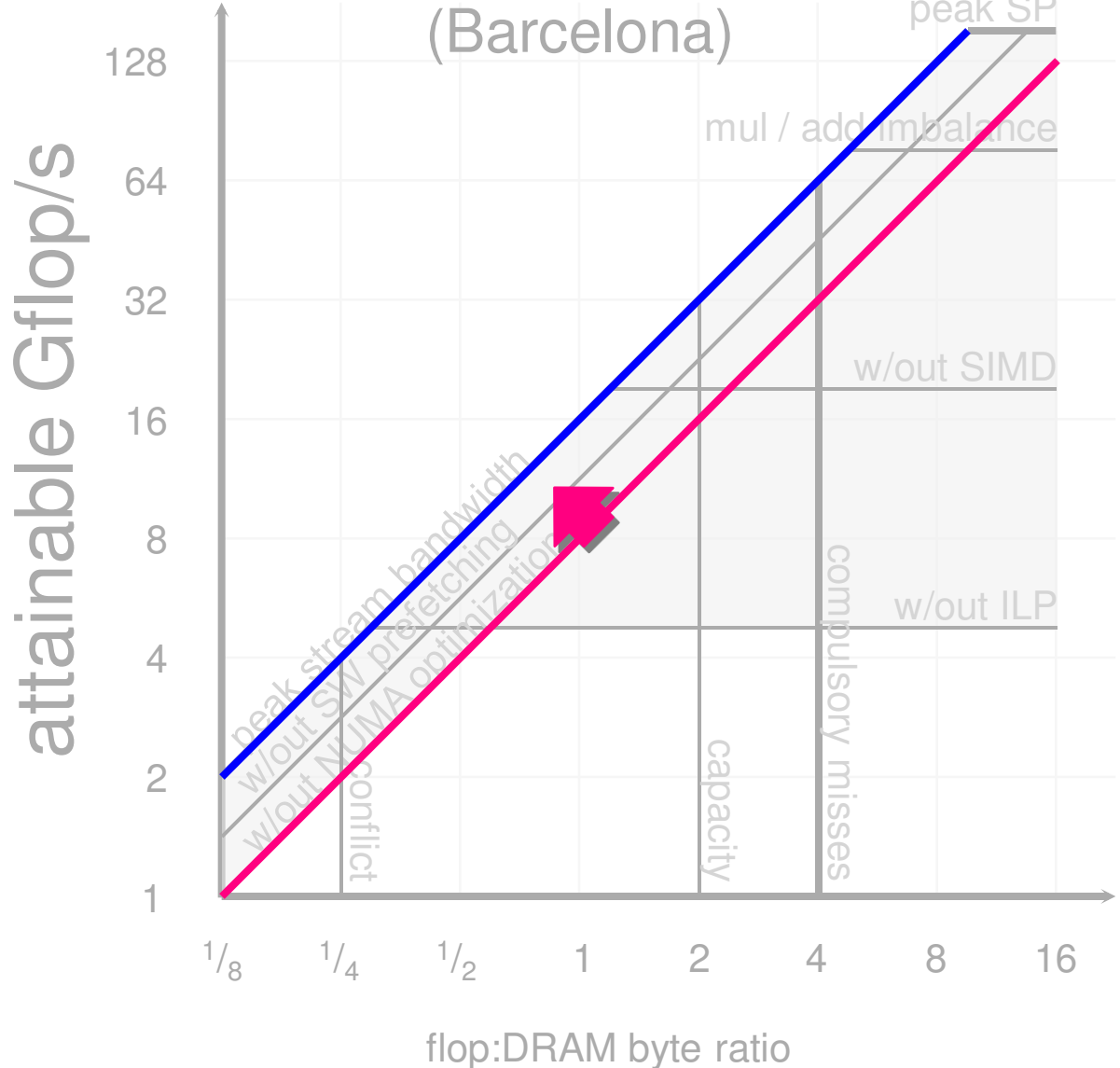




- ❖ Compilers may not have as much knowledge as the programmer
- ❖ Express more in-core parallelism and amortize non-FP instructions
- ❖ Software optimizations:
  - Explicit SIMDization
  - Loop unrolling
  - Unroll and jam
  - Reordering
  - Predication
- ❖ **Punch through ceilings**

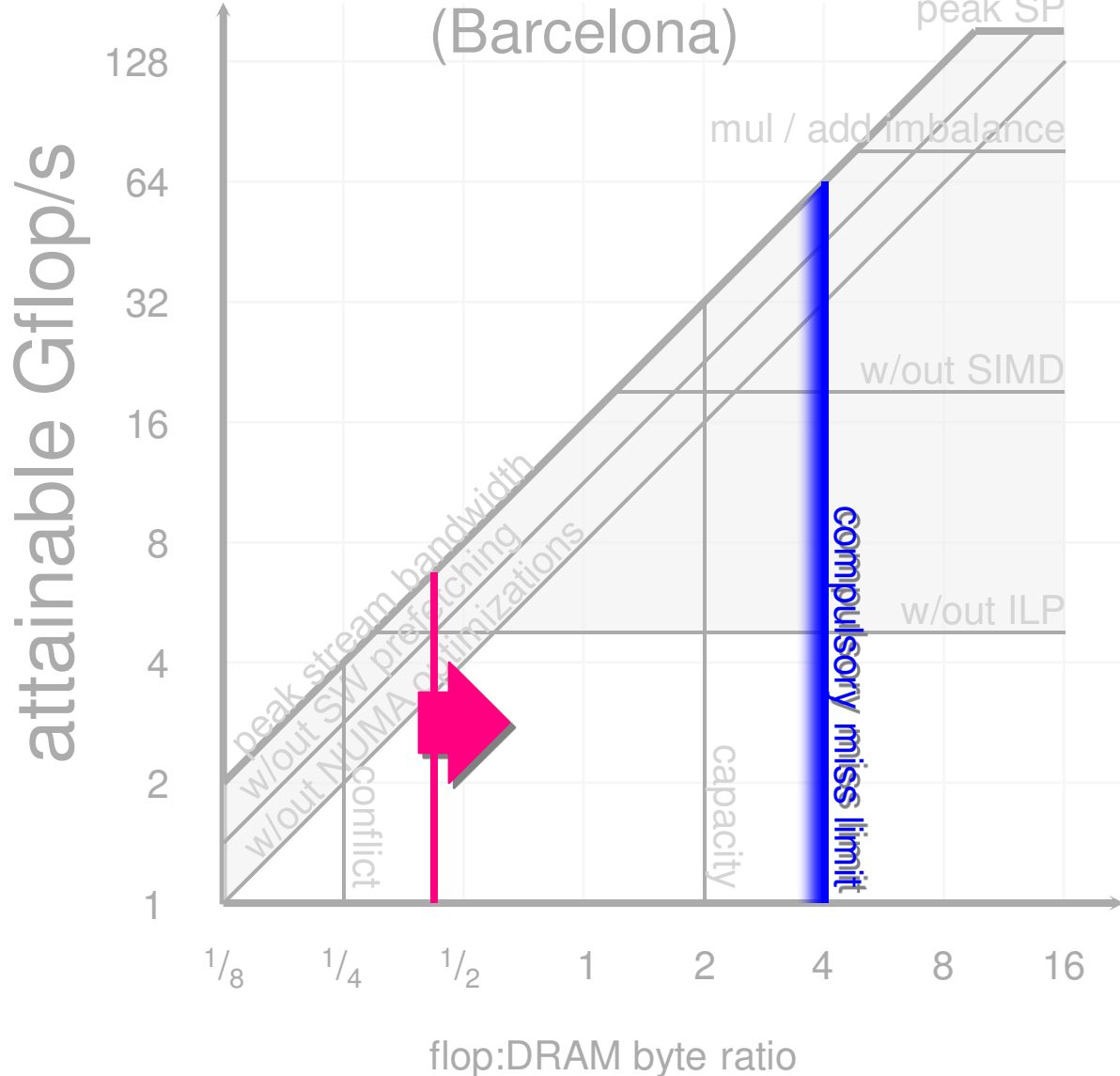
# Maximizing Attained Memory Bandwidth

AMD Opteron 2356  
(Barcelona)



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

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
- ❖ Optimizations:
  - Array padding: conflict
  - Cache blocking: capacity
  - Cache bypass: compulsory
- ❖ **Push arithmetic intensity to the compulsory limit**

## Maximizing In-core Performance

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

## Maximizing Memory Bandwidth

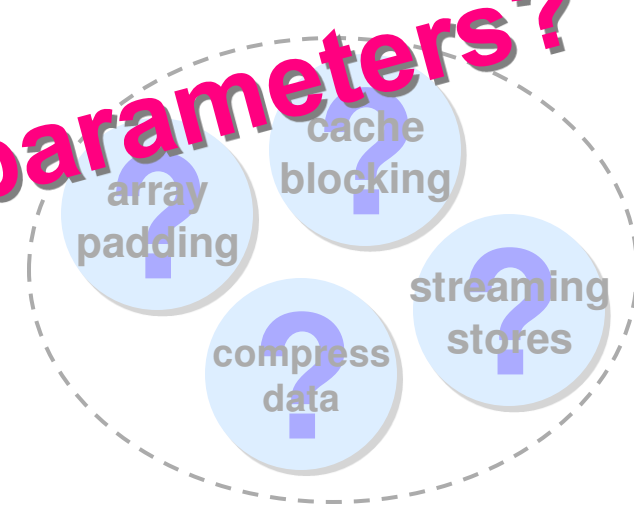
- Exploit NUMA
- Hide memory latency
- Apply Little's Law

## Minimizing Memory Traffic

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

**Each optimization has a large parameter space**

**What are the optimal parameters?**



# Introduction to Auto-tuning

- ❖ Out-of-the-box code has (unintentional) assumptions on:
  - cache sizes (>10MB)
  - functional unit latencies(~1 cycle)
  - etc...
- ❖ These assumptions may result in poor performance when they exceed the machine characteristics



- ❖ Goal: provide **performance portability** across the existing breadth and evolution of microprocessors
- ❖ At the expense of a one time up front productivity cost that's amortized by the number of machines its used on
  
- ❖ Auto-tuning does not invent new optimizations
- ❖ **Auto-tuning automates the exploration of the optimization and parameter space**
- ❖ Two components:
  1. parameterized code generator (we wrote ours in Perl)
  2. Auto-tuning exploration benchmark  
(combination of heuristics and exhaustive search)
- ❖ Can be extended with ISA specific optimizations (e.g. DMA, SIMD)

- ❖ Roofline specifies what's deficient, but not how to fix it.
- ❖ Auto-tuning attempts to fix it by searching the parameter space for the existing body of optimization work

# Application of the Roofline Model to sample Kernels

*Does the roofline model provide insight into the limitations of architecture, implementation, and algorithm?*

❖ Things to watch for:

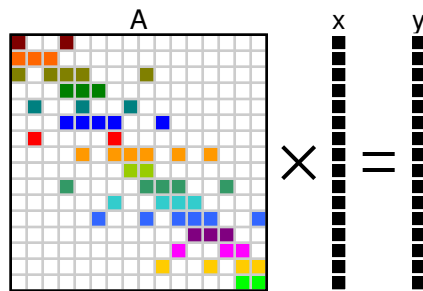
1. do performance graphs alone provide insight into the limitations of kernel or architecture ?
2. does the roofline show the ultimate performance limitations of kernel and architecture ?
3. does the roofline show which optimizations will be necessary ?

# Example #1:

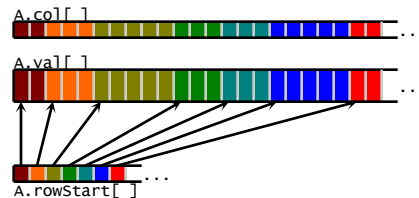
## Auto-tuning Sparse Matrix-Vector Multiplication (SpMV)

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.

- ❖ What's a Sparse Matrix ?
  - Most entries are 0.0
  - Performance advantage in only storing/operating on the nonzeros
  - Requires significant meta data to reconstruct the matrix structure
- ❖ What's SpMV ?
  - Evaluate  $y=Ax$
  - A is a sparse matrix, x & y are dense vectors
- ❖ Challenges
  - **Very low arithmetic intensity (often <0.166 flops/byte)**
  - Difficult to exploit ILP(bad for superscalar),
  - Difficult to exploit DLP(bad for SIMD)



(a)  
algebra conceptualization



(b)  
CSR data structure

```

for (r=0; r<A.rows; r++) {
    double y0 = 0.0;
    for (i=A.rowstart[r]; i<A.rowstart[r+1]; i++){
        y0 += A.val[i] * x[A.col[i]];
    }
    y[r] = y0;
}
    
```

(c)  
CSR reference code

- ❖ Unlike dense BLAS, performance is dictated by sparsity
- ❖ Suite of 14 matrices
- ❖ All bigger than the caches of our SMPs
- ❖ We'll also include a median performance number

2K x 2K Dense matrix  
stored in sparse format



Dense

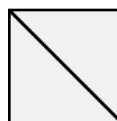
Well Structured  
(sorted by nonzeros/row)



Protein



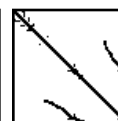
FEM /  
Spheres



FEM /  
Cantilever



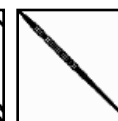
Wind  
Tunnel



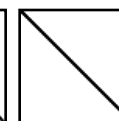
FEM /  
Harbor



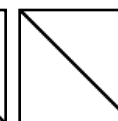
QCD



FEM /  
Ship



Economics



Epidemiology

Poorly Structured  
hodgepodge



FEM /  
Accelerator



Circuit



webbase

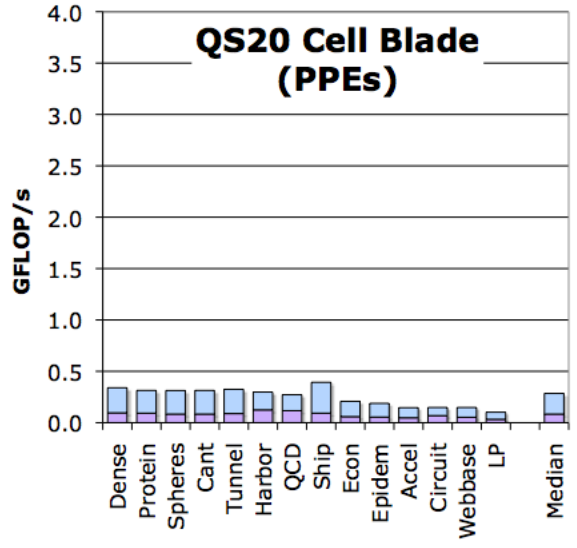
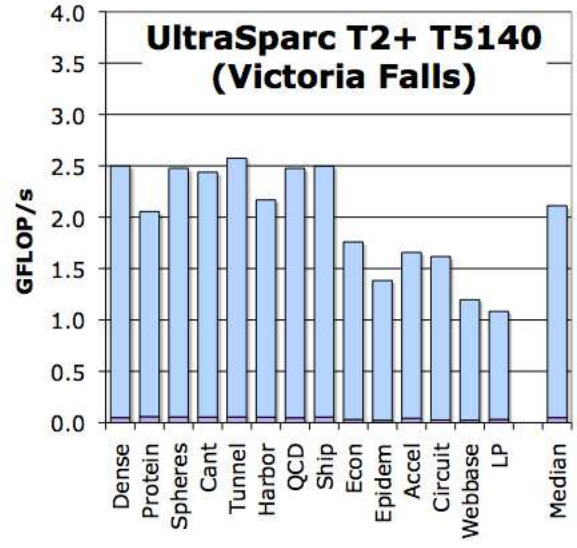
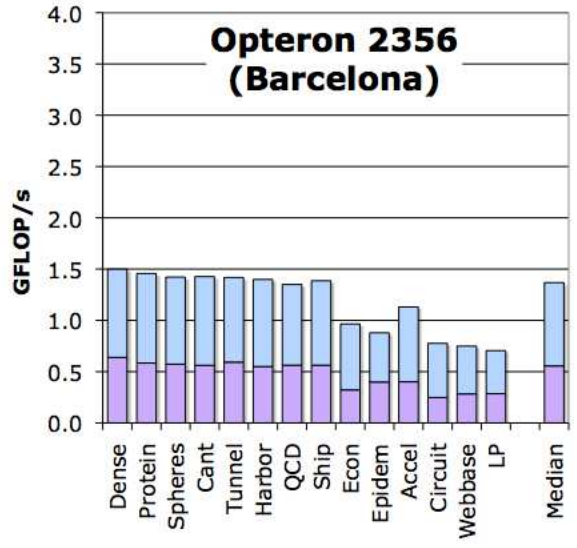
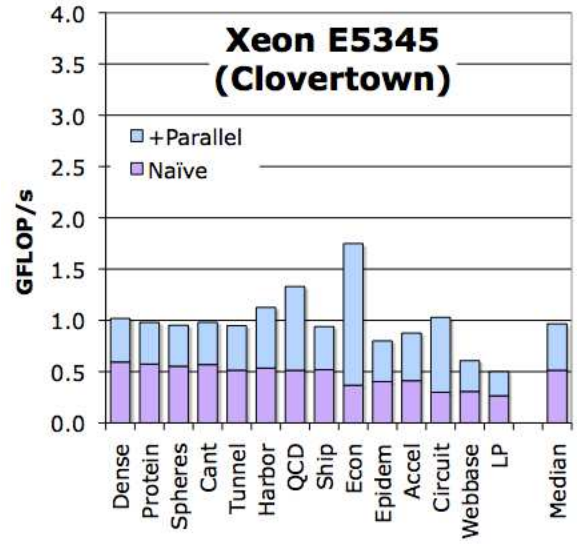
Extreme Aspect Ratio  
(linear programming)



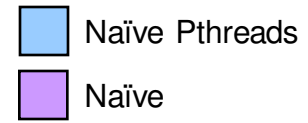
LP

# SpMV Performance

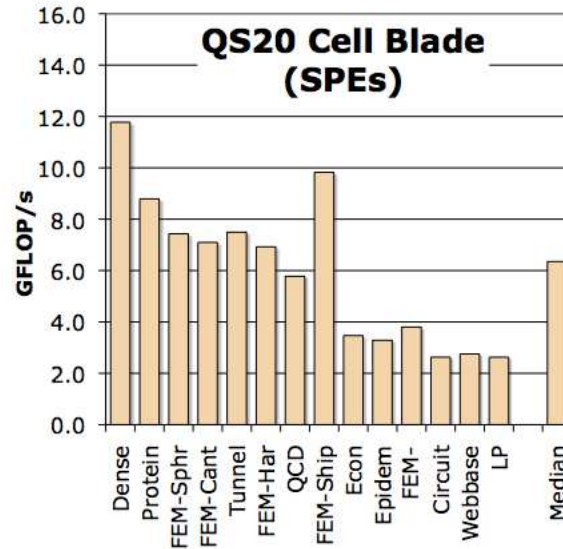
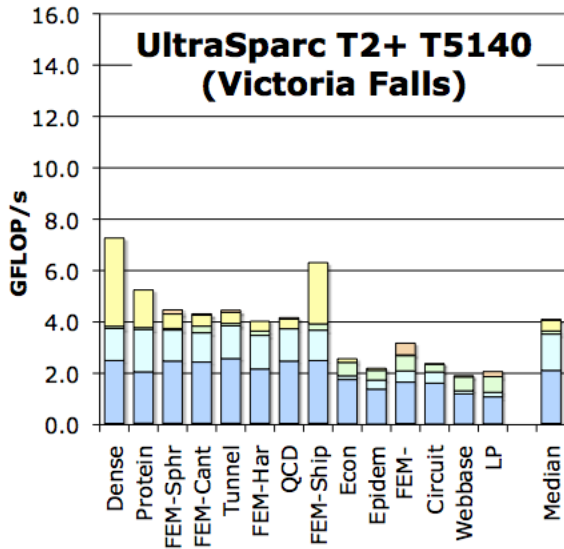
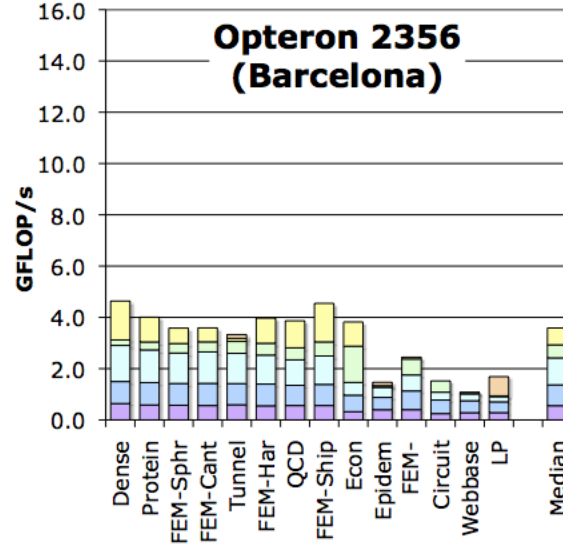
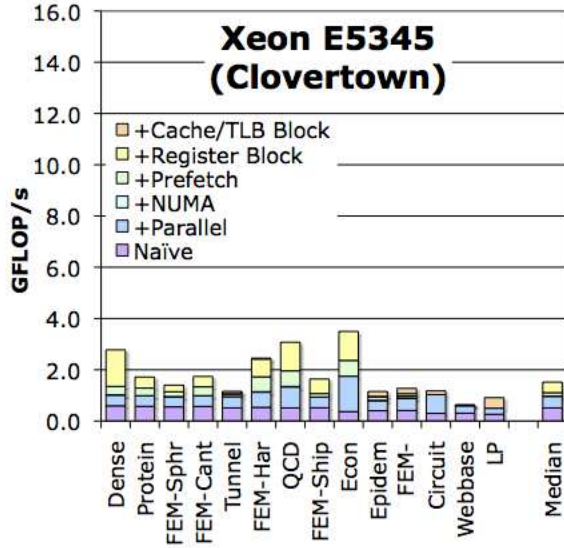
(simple parallelization)



- ❖ Out-of-the box SpMV performance on a suite of 14 matrices
- ❖ **Scalability isn't great**
- ❖ **Is this performance good?**

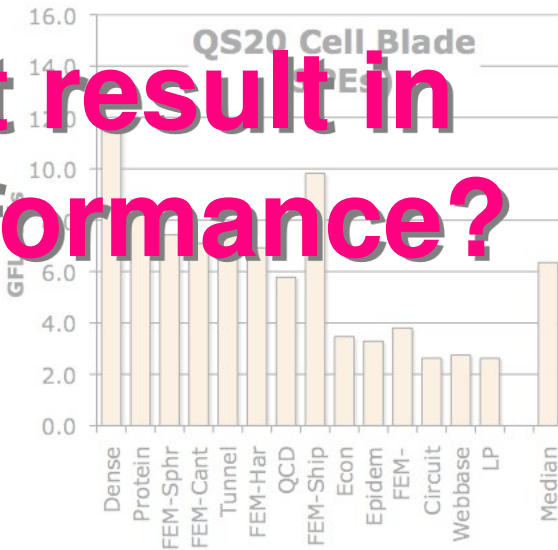
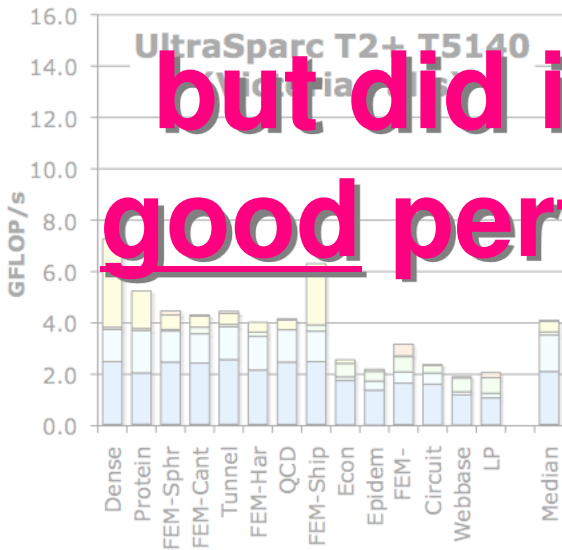
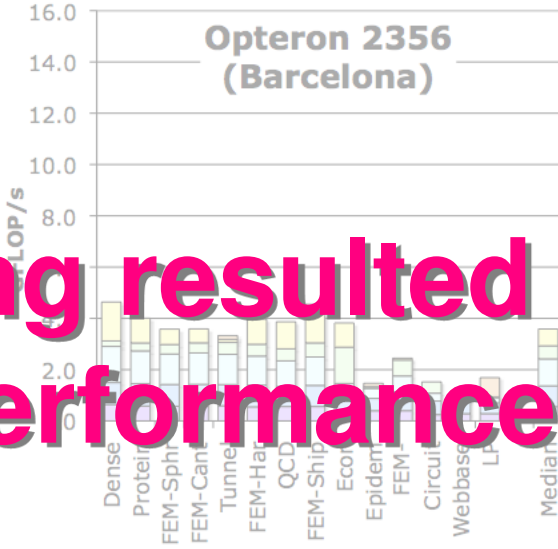






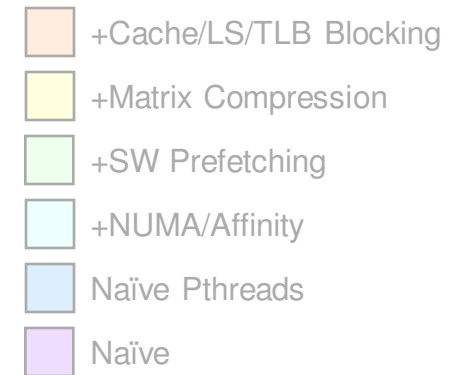
- ❖ Fully auto-tuned SpMV performance across the suite of matrices
- ❖ Included SPE/local store optimized version
- ❖ Why do some optimizations work better on some architectures?
- ❖ **Performance is better, but is performance good?**

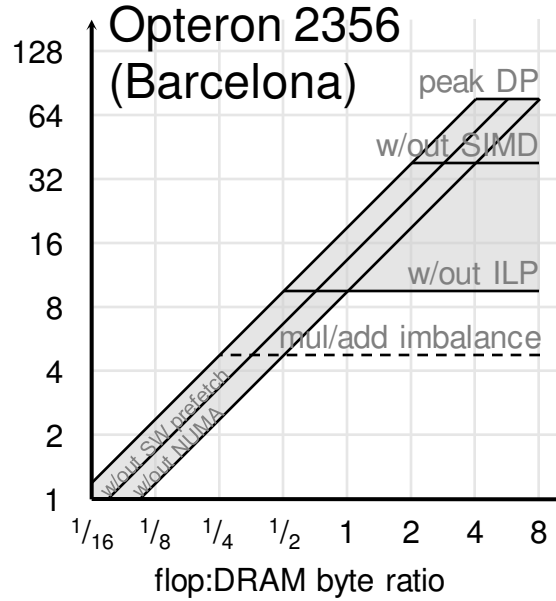
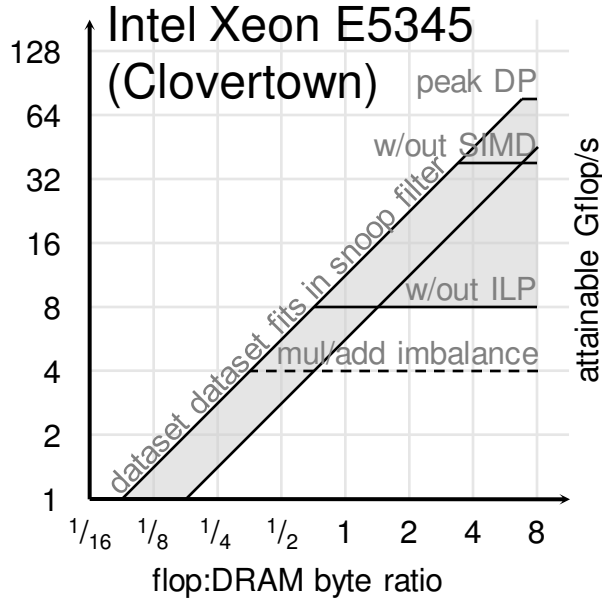
- +Cache/LS/TLB Blocking
- +Matrix Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve



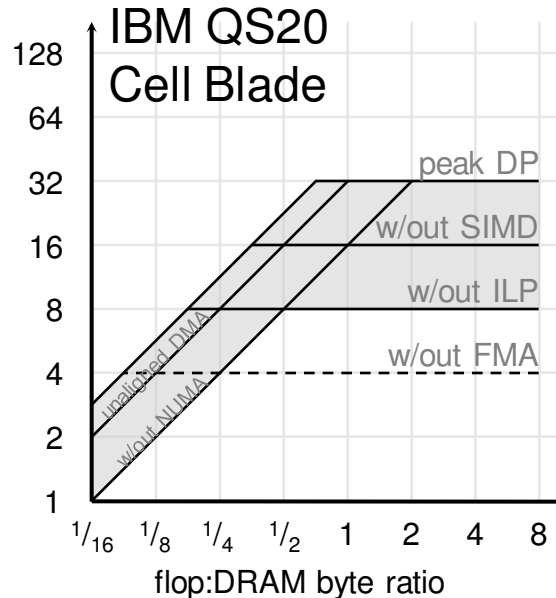
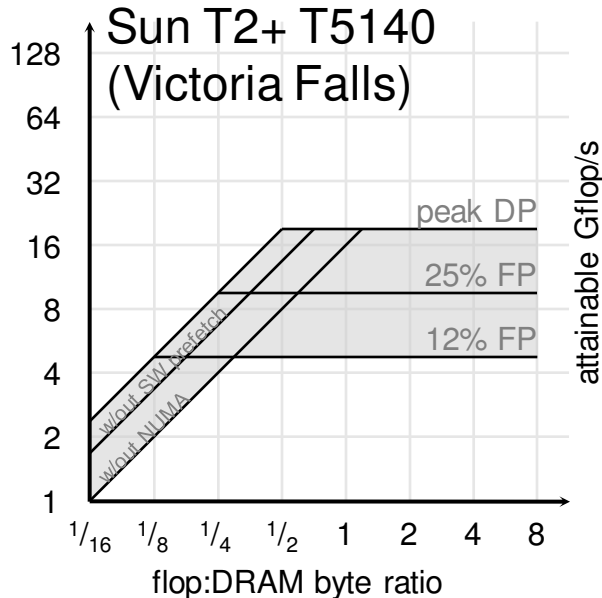
**Auto-tuning resulted in better performance, but did it result in good performance?**

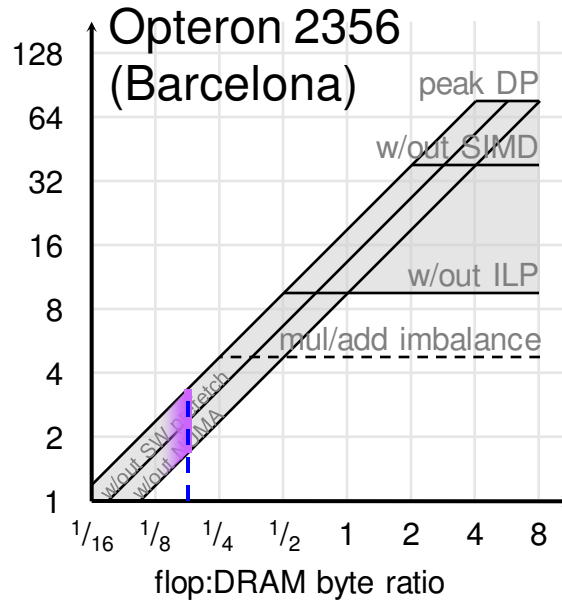
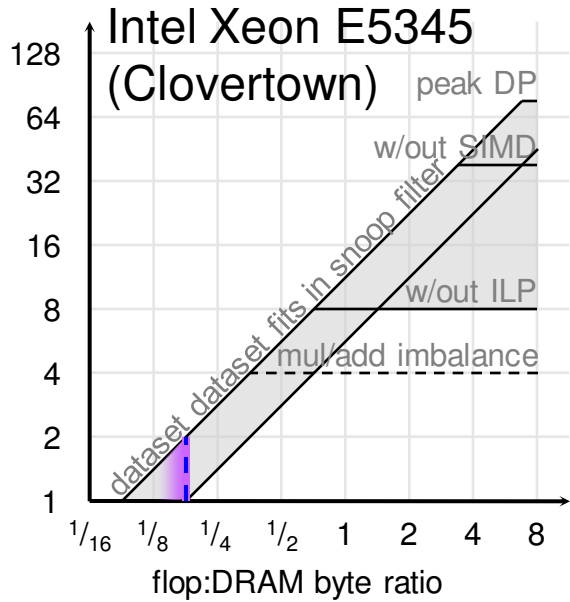
- ❖ Fully auto-tuned SpMV performance across the suite of matrices
- ❖ Included SPE/local store optimized version
- ❖ Why do some optimizations work better on some architectures?
- ❖ **Performance is better, but is performance good?**



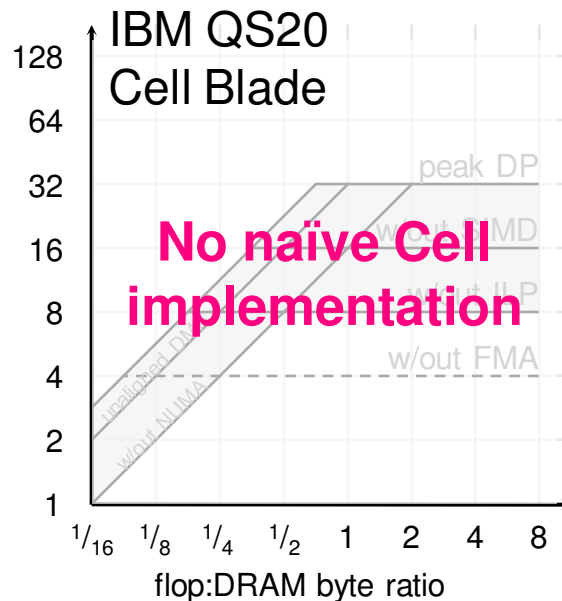
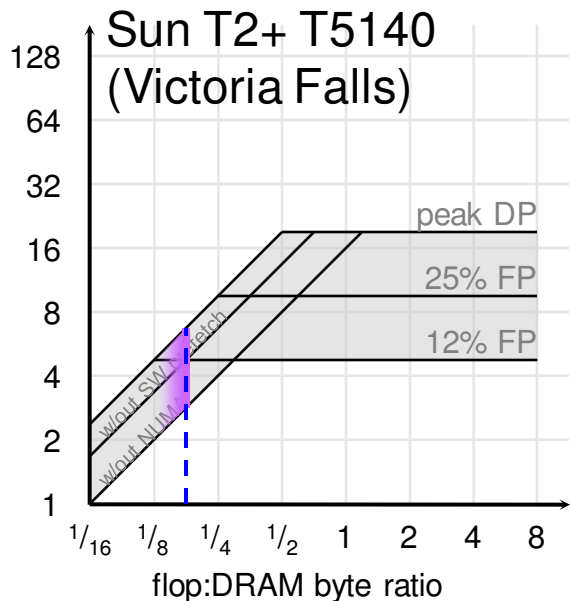


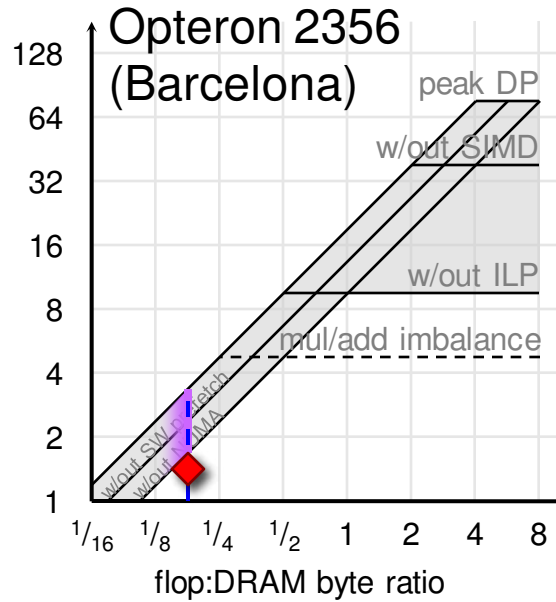
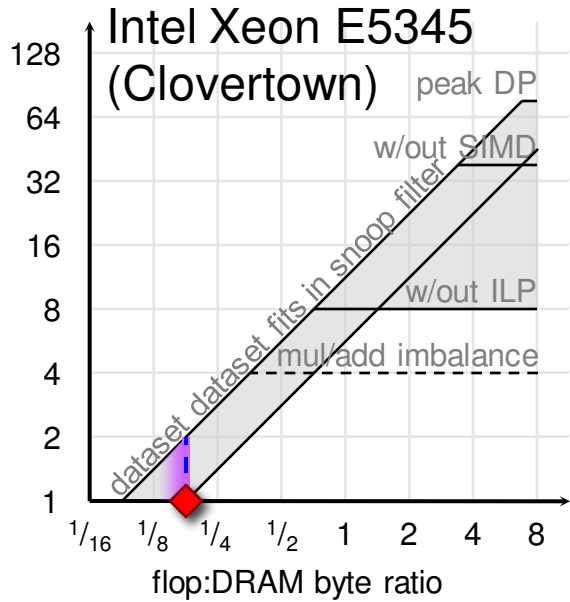
- ❖ Double precision roofline models
- ❖ FMA is inherent in SpMV (place at bottom)



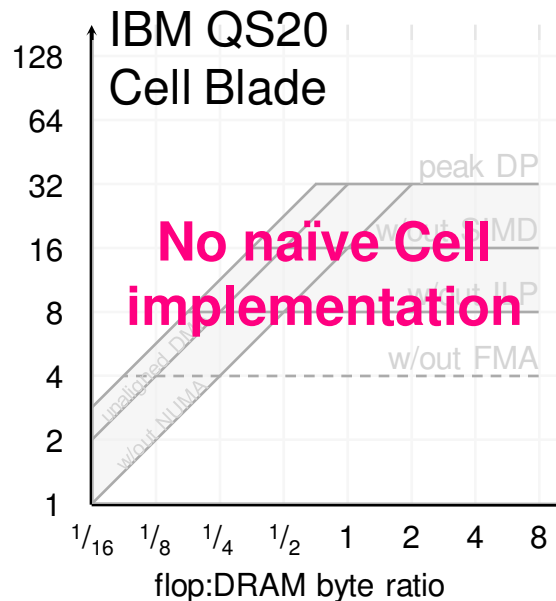
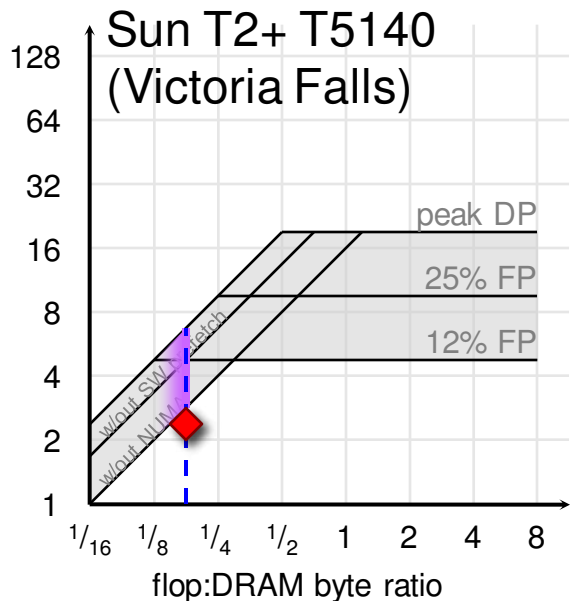


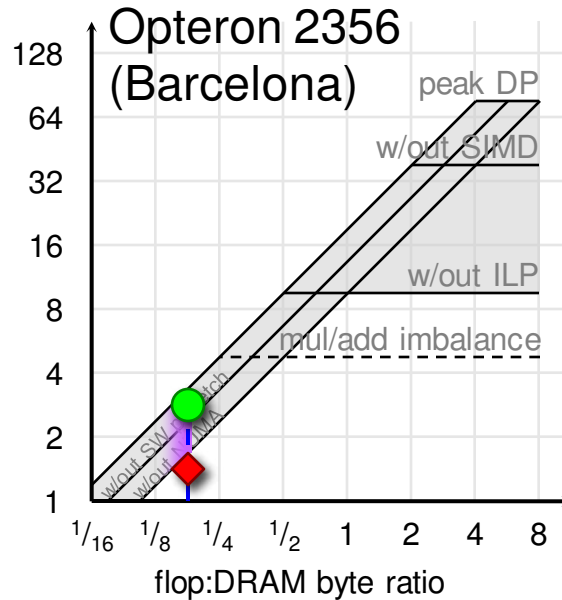
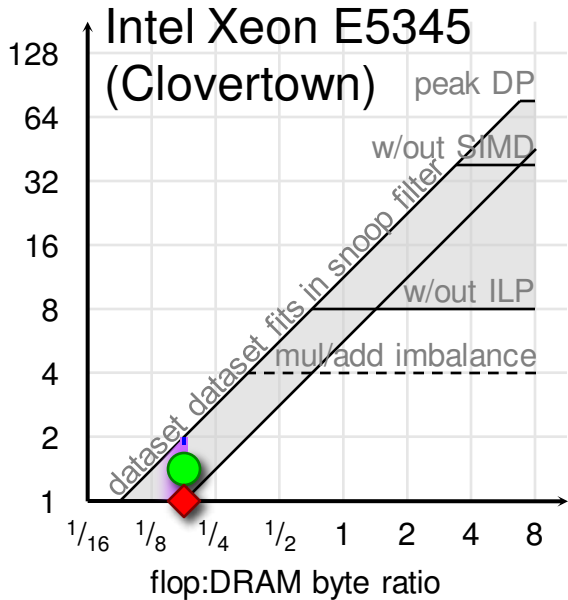
- ❖ Two unit stride streams
- ❖ Inherent FMA
- ❖ No ILP
- ❖ No DLP
- ❖ FP is 12-25%
- ❖ Naïve compulsory  
flop:byte < 0.166



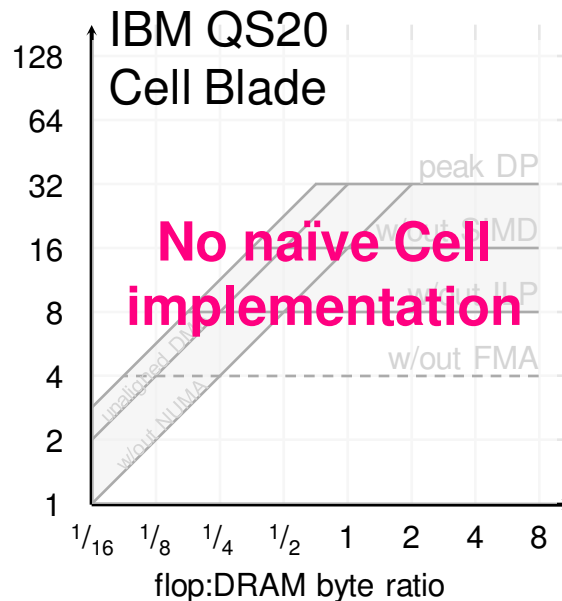
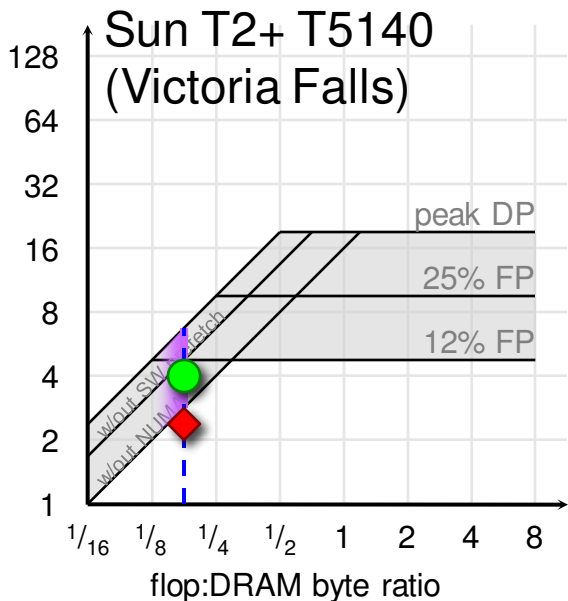


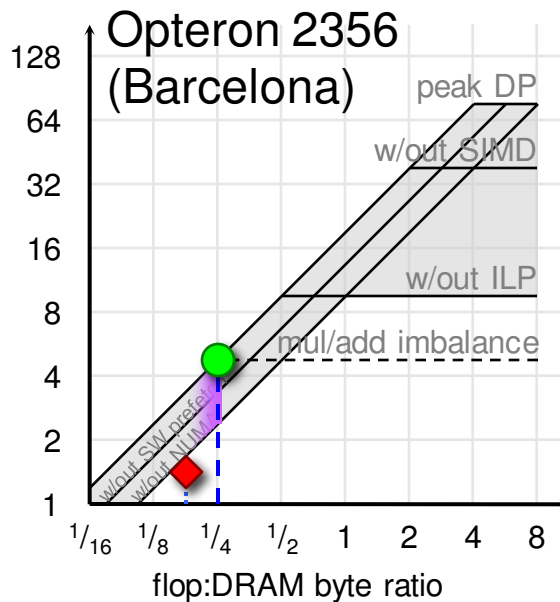
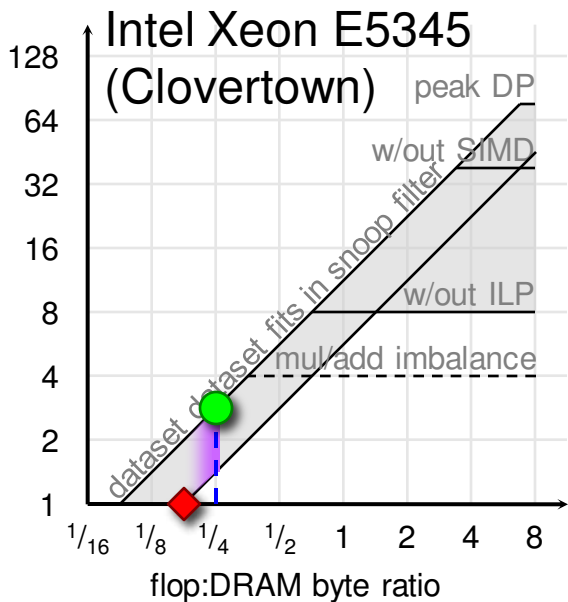
- ❖ Two unit stride streams
- ❖ Inherent FMA
- ❖ No ILP
- ❖ No DLP
- ❖ FP is 12-25%
- ❖ Naïve compulsory  
flop:byte < 0.166
- ❖ For simplicity: **dense matrix in sparse format**



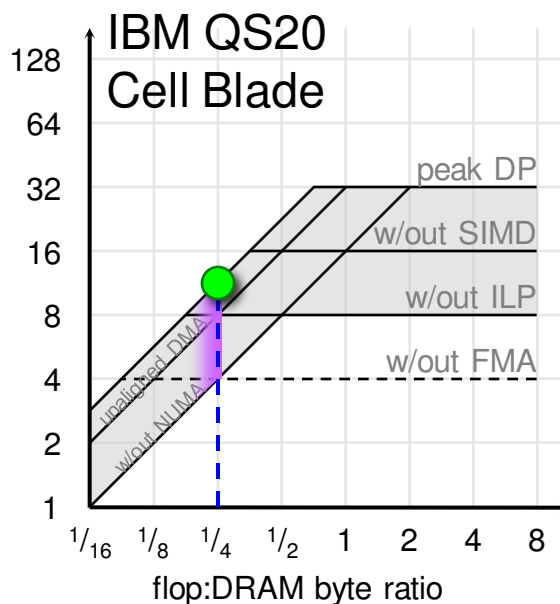
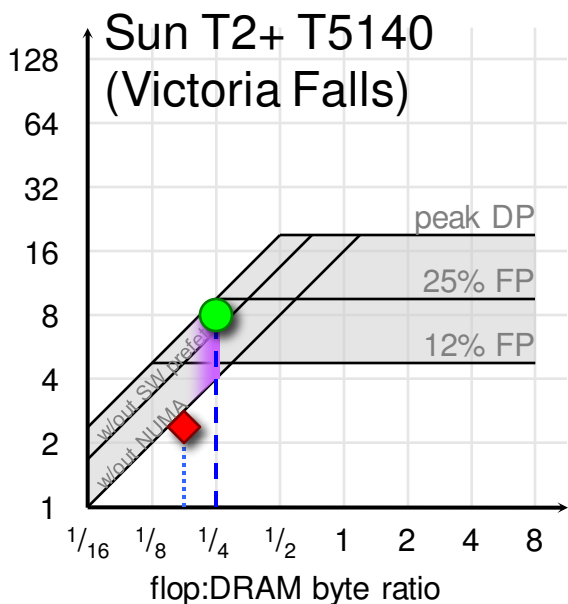


- ❖ compulsory flop:byte ~ 0.166
- ❖ utilize all memory channels

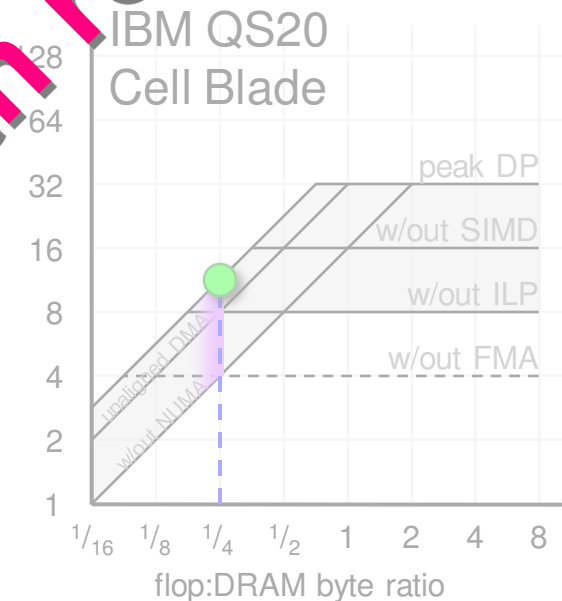
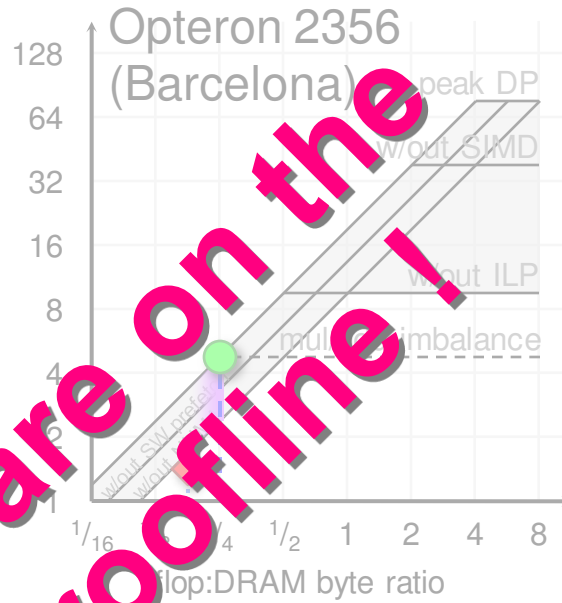
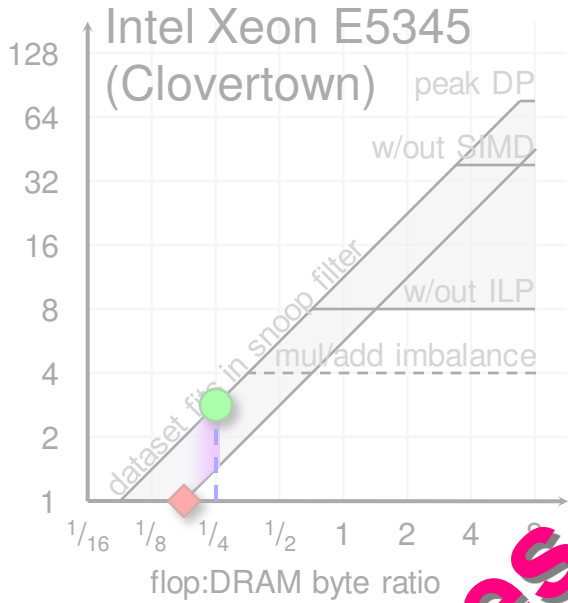




- ❖ Inherent FMA
- ❖ Register blocking improves ILP, DLP, flop:byte ratio, and FP% of instructions



# Roofline model for SpMV (matrix compression)



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

All machines are on the bandwidth roofline!



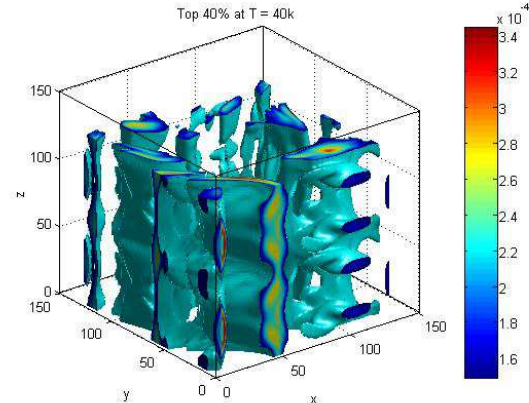
# Example #2:

## Auto-tuning Lattice-Boltzmann Magneto-Hydrodynamics (LBMHD)

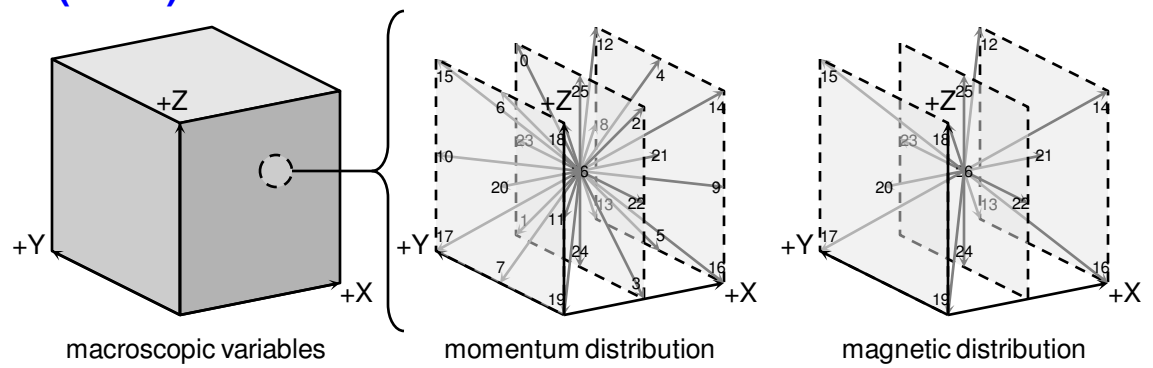
Samuel Williams, Jonathan Carter, Leonid Oliker, John Shalf, Katherine Yelick, "Lattice Boltzmann Simulation Optimization on Leading Multicore Platforms", International Parallel & Distributed Processing Symposium (IPDPS), 2008.

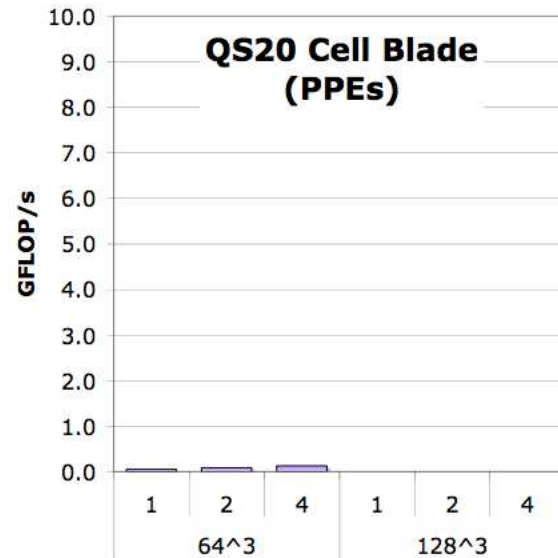
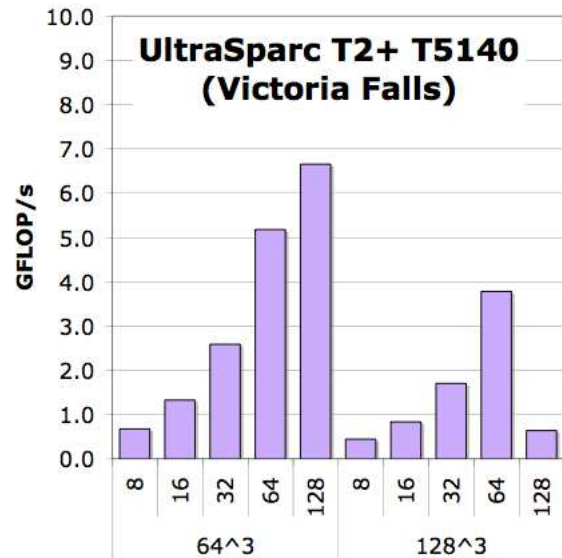
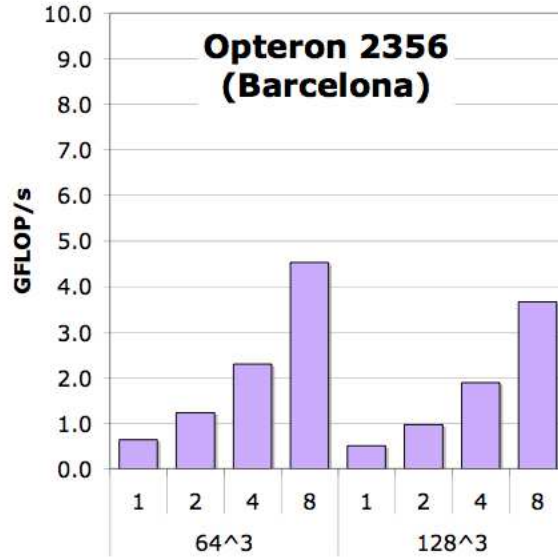
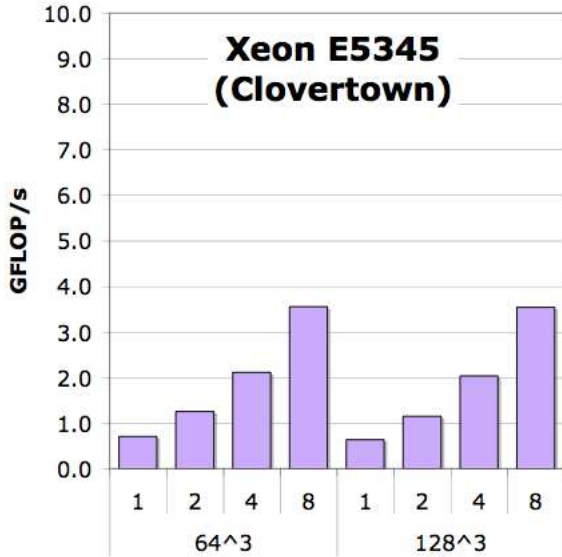
**Best Paper, Application Track**

- ❖ 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)
- ❖ Arithmetic Intensity:
  - Must read 73 doubles, and update 79 doubles per lattice update (1216 bytes)
  - Requires about 1300 floating point operations per lattice update
  - **Just over 1.0 flops/byte (ideal)**



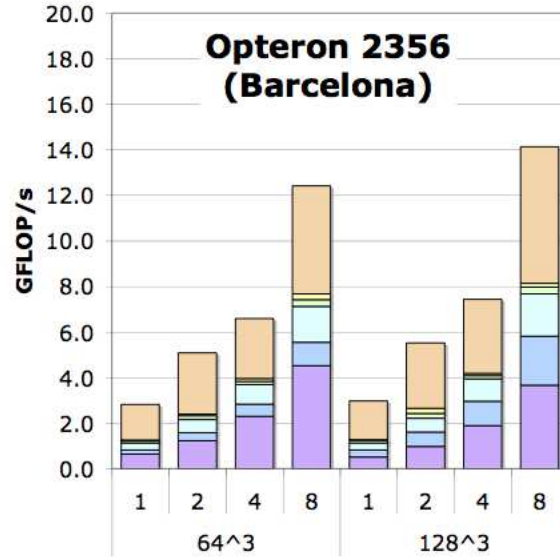
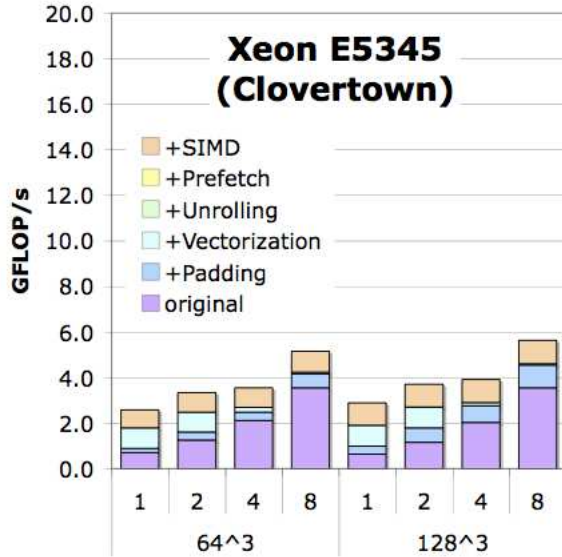
❖ Out-of-the-box,  
no unit stride memory  
access patterns



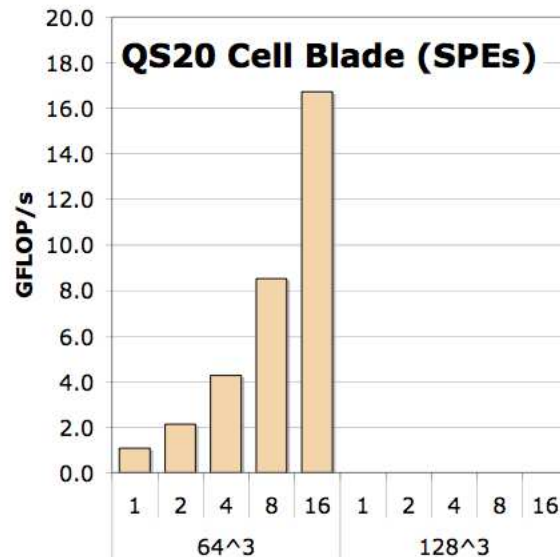
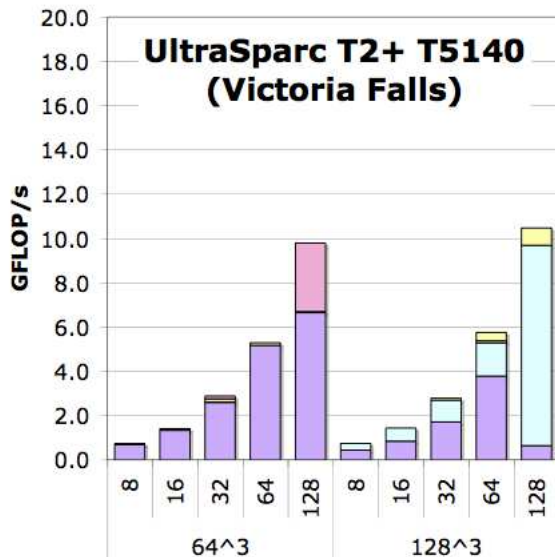


- ❖ Generally, scalability looks good
- ❖ **but is performance good?**

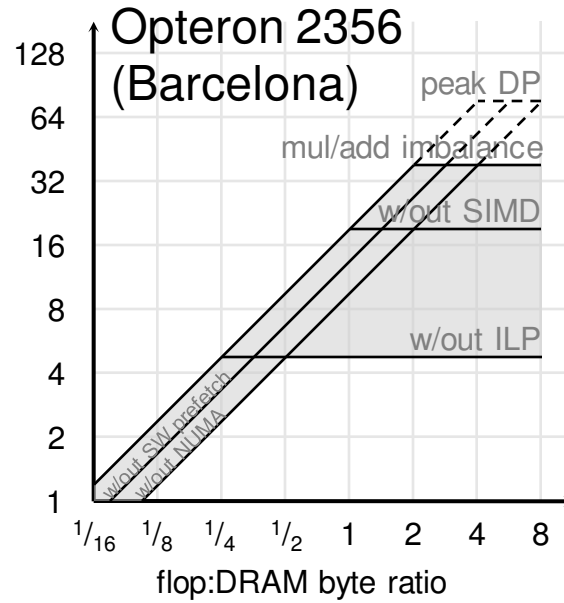
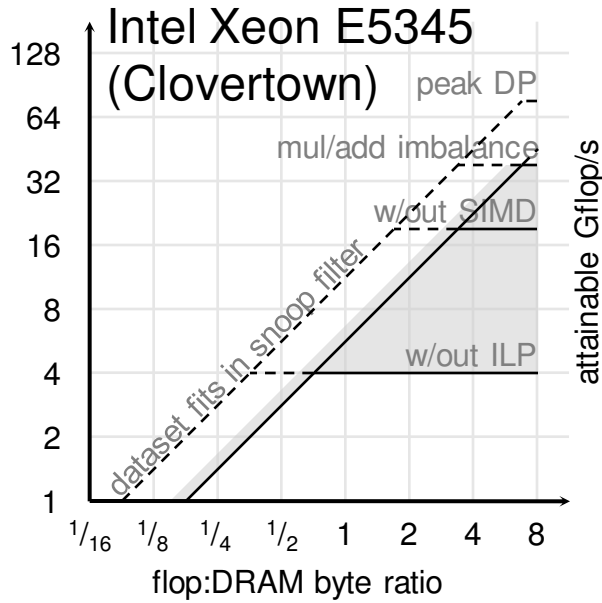
 Naïve+NUMA



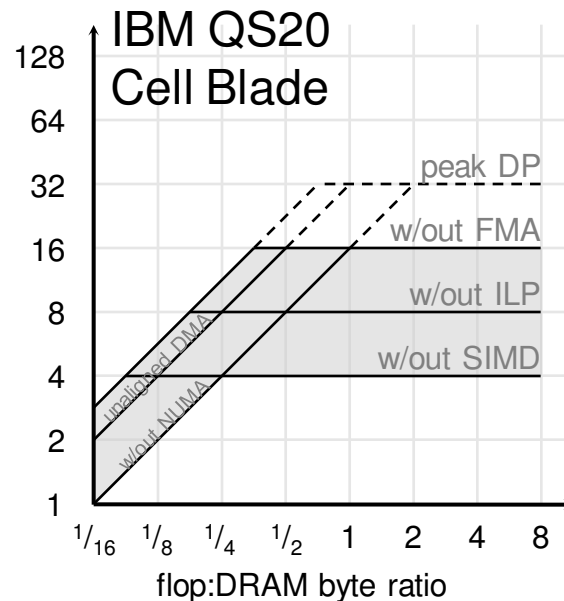
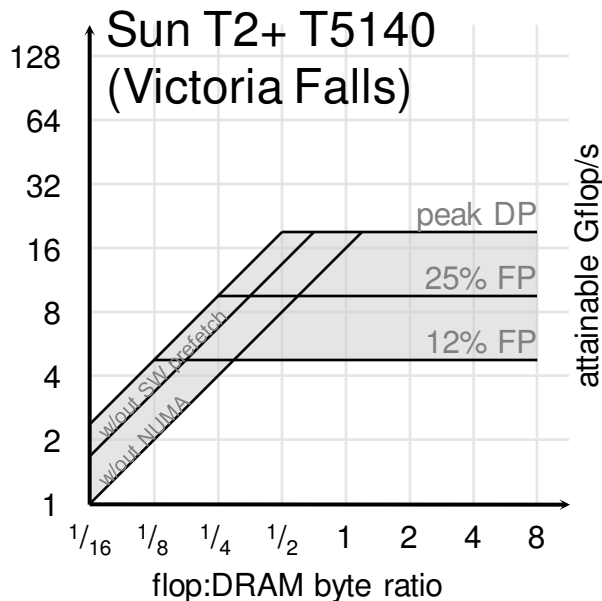
- ❖ Auto-tuning avoids cache conflict and TLB capacity misses
- ❖ Additionally, it exploits SIMD where the compiler doesn't
- ❖ Include a SPE/Local Store optimized version



- +small pages
- +Explicit SIMDization
- +SW Prefetching
- +Unrolling
- +Vectorization
- +Padding
- Naïve+NUMA

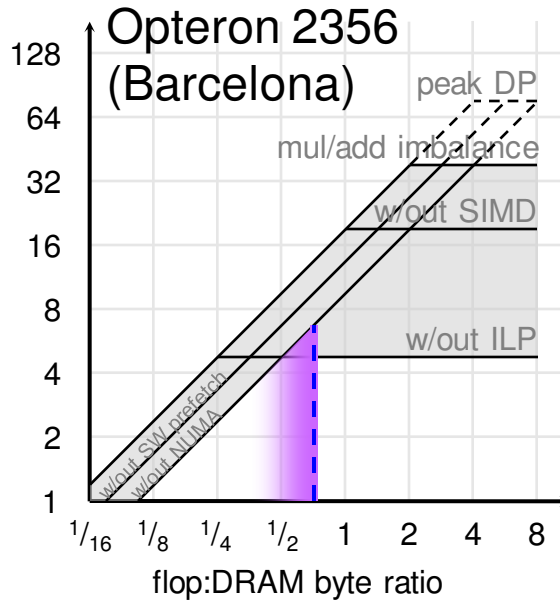
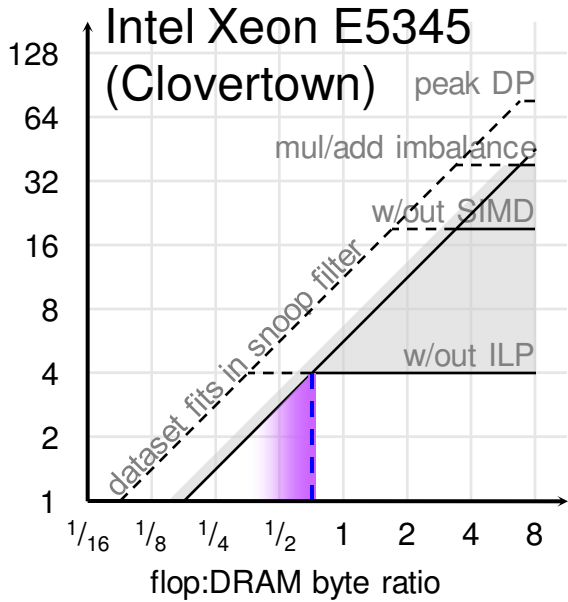


- ❖ Far more adds than multiplies (imbalance)
- ❖ Huge data sets

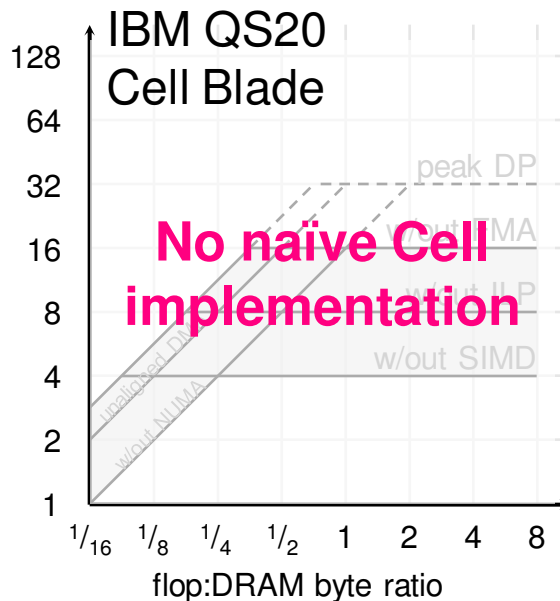
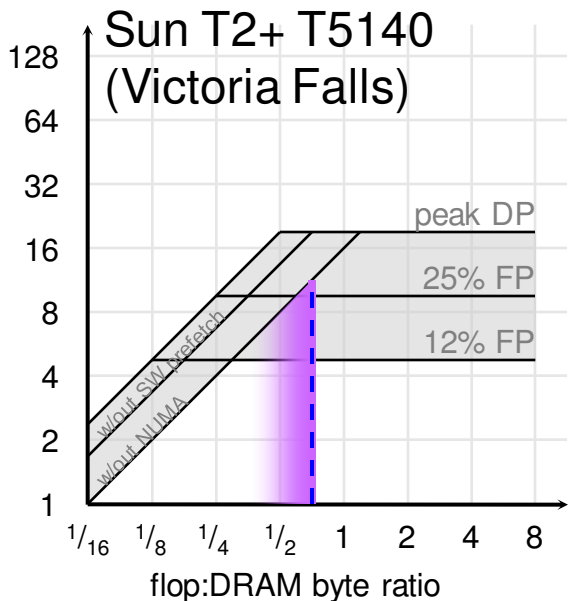


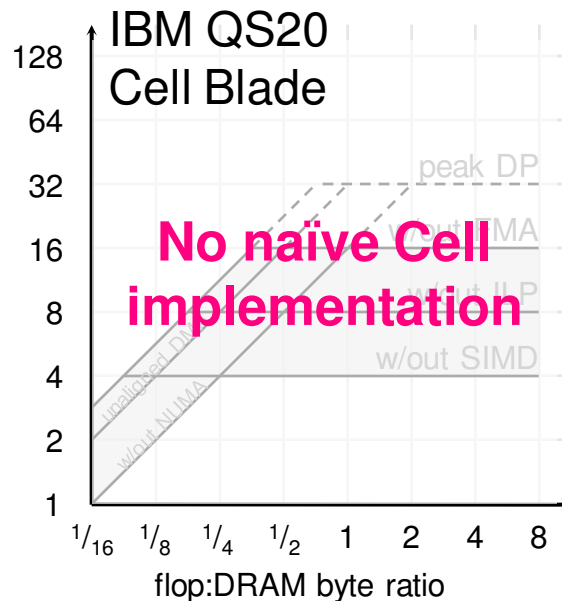
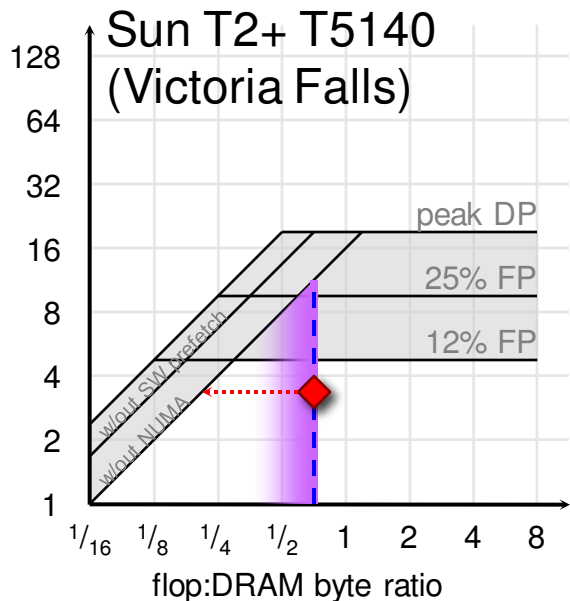
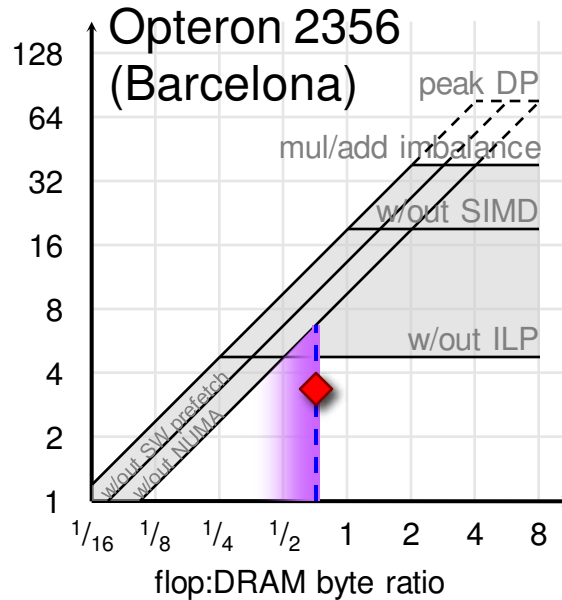
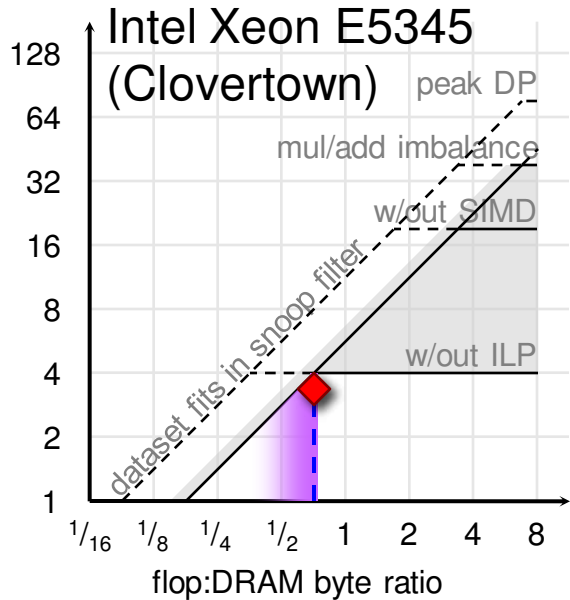
# Roofline model for LBMHD

(overlay arithmetic intensity)



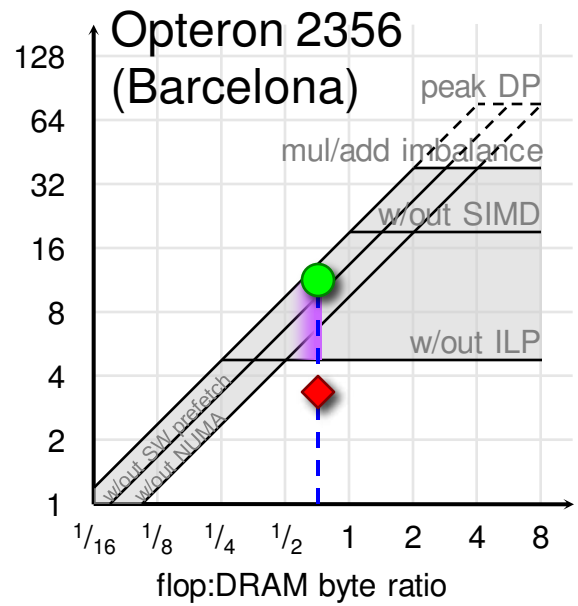
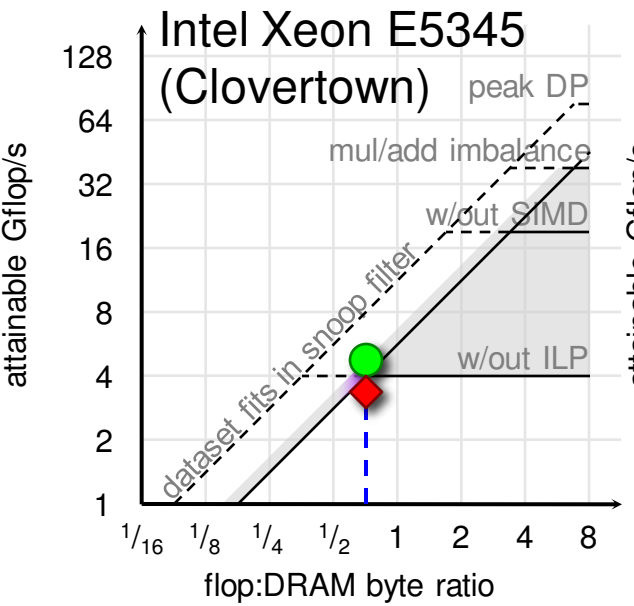
- ❖ Far more adds than multiplies (imbalance)
- ❖ **Essentially random access to memory**
- ❖ Flop:byte ratio ~0.7
- ❖ NUMA allocation/access
- ❖ Little ILP
- ❖ No DLP
- ❖ High conflict misses



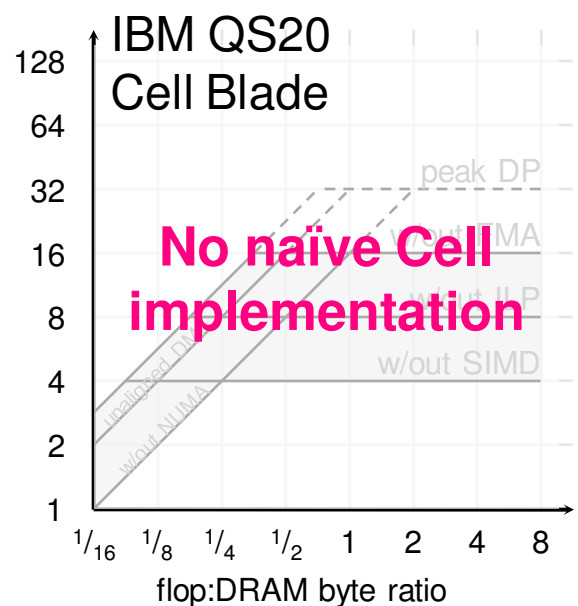
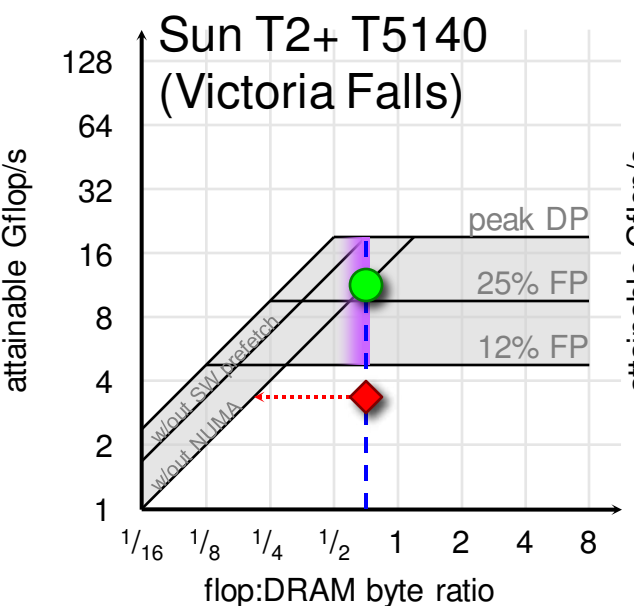


- ❖ Far more adds than multiplies (imbalance)
- ❖ **Essentially random access to memory**
- ❖ Flop:byte ratio  $\sim 0.7$
- ❖ NUMA allocation/access
- ❖ Little ILP
- ❖ No DLP
- ❖ High conflict misses

- ❖ Peak VF performance with 64 threads (out of 128) - high conflict misses

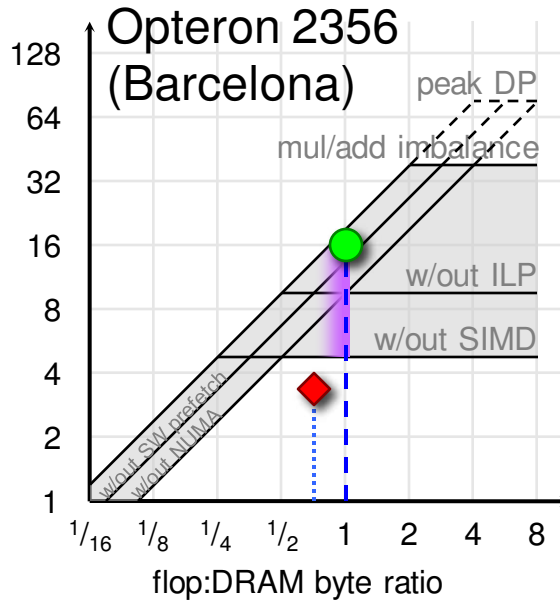
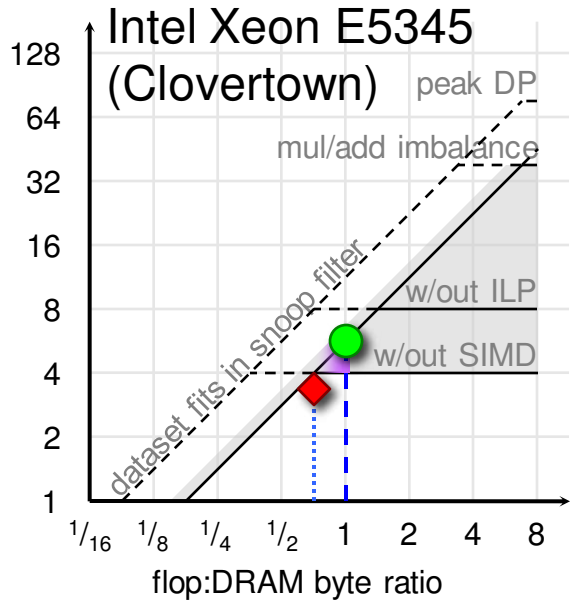


- ❖ Vectorize the code to eliminate TLB capacity misses
- ❖ Ensures unit stride access (bottom bandwidth ceiling)
- ❖ Tune for optimal VL
- ❖ Clovertown pinned to lower BW ceiling

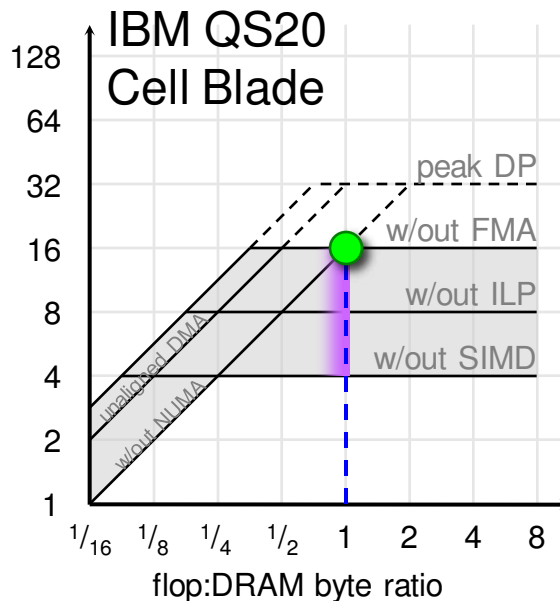
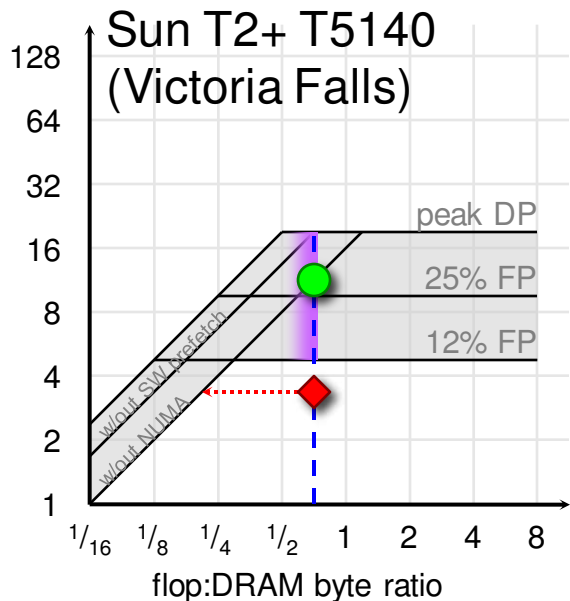




# Roofline model for LBMHD (SIMDization + cache bypass)

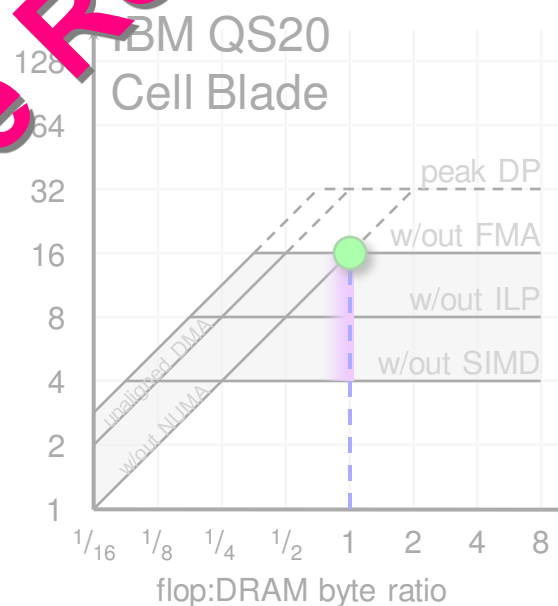
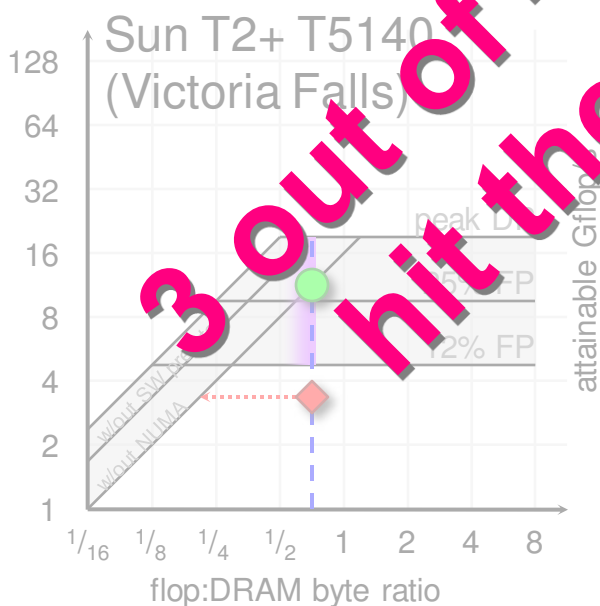
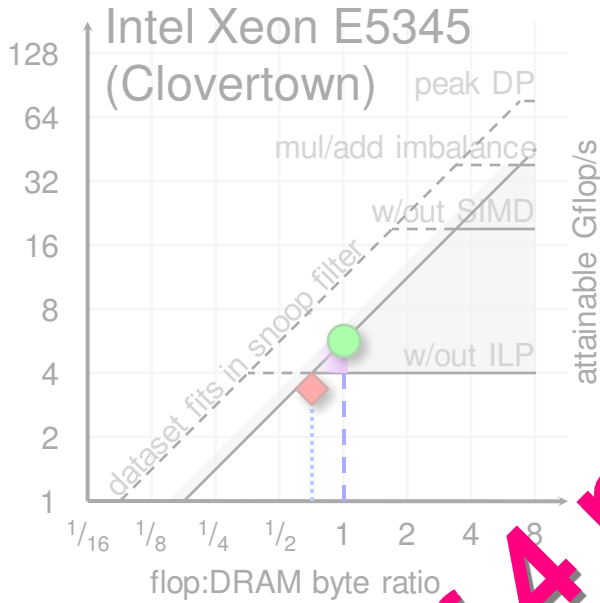


- ❖ Make SIMDization explicit
- ❖ Technically, this swaps ILP and SIMD ceilings
- ❖ Use cache bypass instruction: *movntpd*
- ❖ Increases flop:byte ratio to ~1.0 on x86/Cell



# Roofline model for LBMHD

(SIMDization + cache bypass)



- ❖ 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

3 out of 4 machines hit the Roofline

# Conclusions

- ❖ The Roofline model is a visually intuitive figure for kernel analysis and optimization
- ❖ We believe undergraduates will find it useful in assessing performance and scalability limitations
- ❖ It is easily extended to other architectural paradigms
- ❖ We believe it is easily extendable to other metrics:
  - performance (sort, graphics, crypto...)
  - bandwidth (L2, PCIe, ...)
- ❖ We believe that a performance counters could be used to generate a **runtime-specific** roofline that would greatly aide the optimization

- ❖ As architectures are presented over the next two days, we invite you to create a roofline model for each.
- ❖ Estimate the ceilings.
- ❖ Then contemplate performance and productivity among them

❖ Research supported by:

- Microsoft and Intel funding (Award #20080469)
- DOE Office of Science under contract number DE-AC02-05CH11231
- NSF contract CNS-0325873
- Sun Microsystems - Niagara2 / Victoria Falls machines
- AMD - access to Quad-core Opteron (barcelona) access
- Forschungszentrum Jülich - access to QS20 Cell blades
- IBM - virtual loaner program to QS20 Cell blades

# Questions ?

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.

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**

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*", Supercomputing (SC) (to appear), 2008.