PERI :
Auto-tuning Memory Intensive Kernels for Multicore

Samuel Williams[1,2], Kaushik Datta[1],
Jonathan Carter[2], Leonid Oliker[1,2], John Shalf[2],
Katherine Yelick[1,2], David Bailey[2]

[1]University of California, Berkeley
[2]Lawrence Berkeley National Laboratory

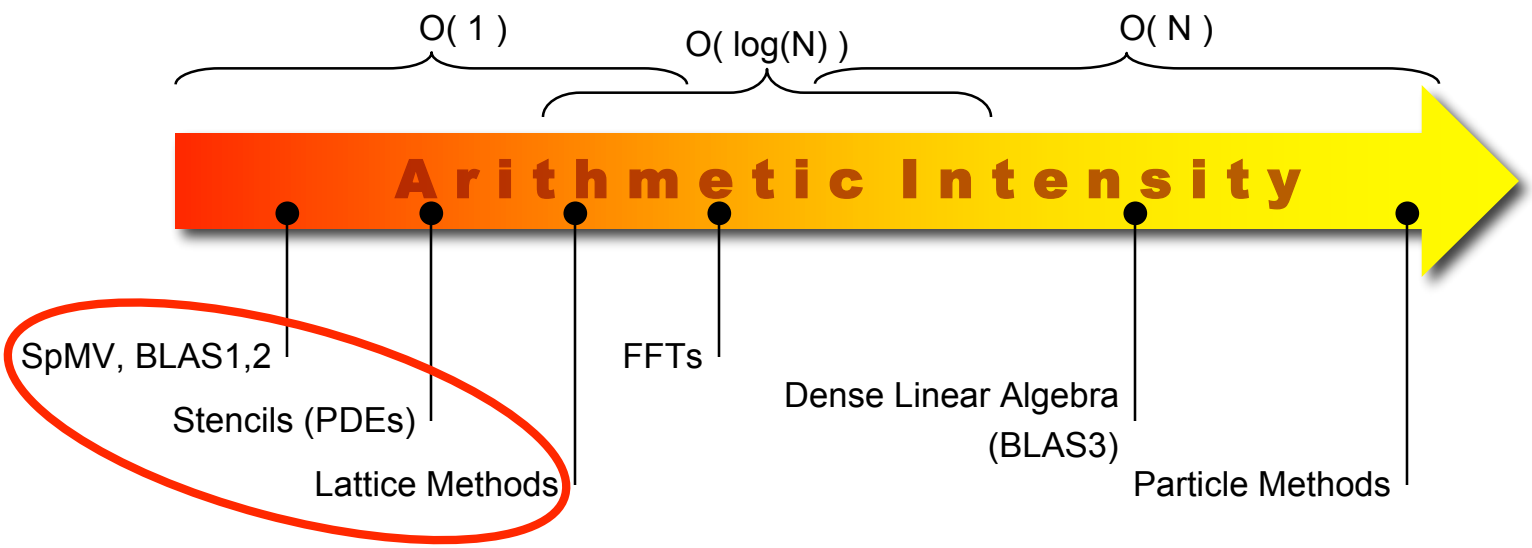
samw@cs.berkeley.edu

Motivation

- Multicore is the de facto solution for increased peak performance for the next decade
- However, given the diversity of architectures, multicore guarantees neither good scalability nor good (attained) performance
- We need a solution that provides **performance portability**

What's a Memory Intensive Kernel?

Arithmetic Intensity in HPC



- **True Arithmetic Intensity (AI) ~ Total Flops / Total DRAM Bytes**
- Some HPC kernels have an arithmetic intensity that scales with problem size (increased temporal locality), but remains constant on others
- Arithmetic intensity is ultimately limited by compulsory traffic
- Arithmetic intensity is diminished by conflict or capacity misses.

Memory Intensive

- Let us define memory intensive to be when the kernel's arithmetic intensity is less than machine's balance (flop:byte)

  Performance ~ Stream BW * Arithmetic Intensity

- Technology allows peak flops to improve faster than bandwidth.

  ⇨**more and more kernels will be considered memory intensive**

Outline

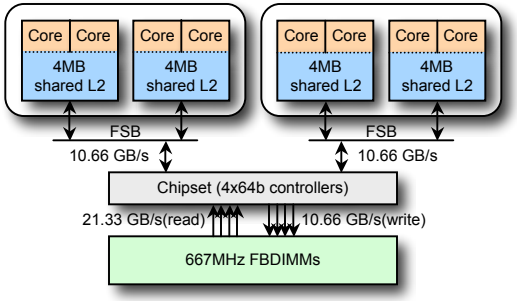
- Motivation
- Memory Intensive Kernels
- Multicore SMPs of Interest
- Software Optimizations
- Introduction to Auto-tuning
- Auto-tuning Memory Intensive Kernels
  - Sparse Matrix Vector Multiplication (SpMV)
  - Lattice-Boltzmann Magneto-Hydrodynamics (LBMHD)
  - Heat Equation Stencil (3D Laplacian)
- Summary

Multicore SMPs of Interest

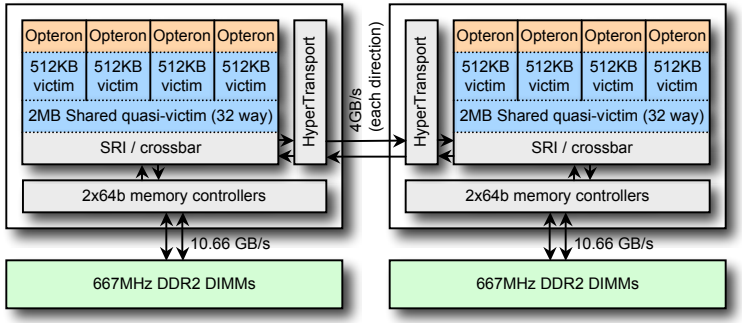
(used throughout the rest of the talk)

Multicore SMPs Used

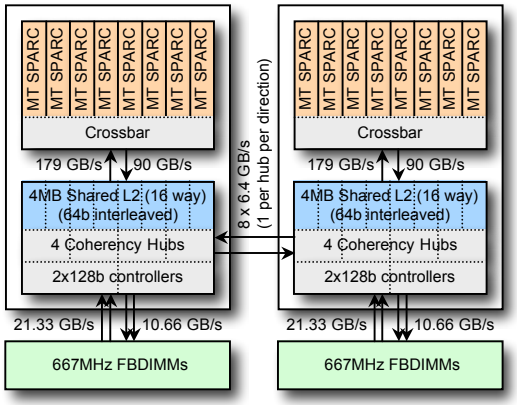
Intel Xeon E5345 (Clovertown)



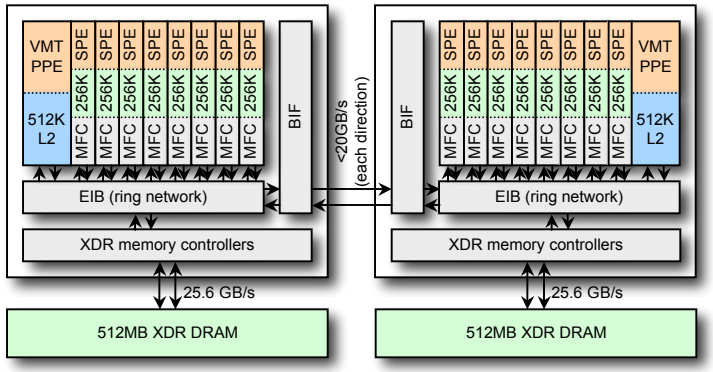
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)

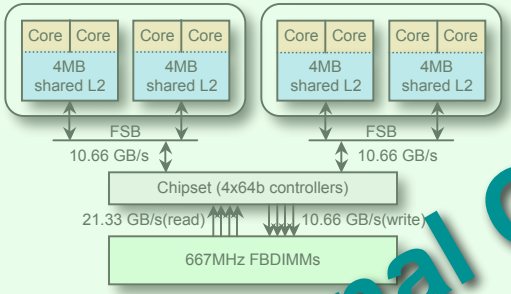


IBM QS20 Cell Blade

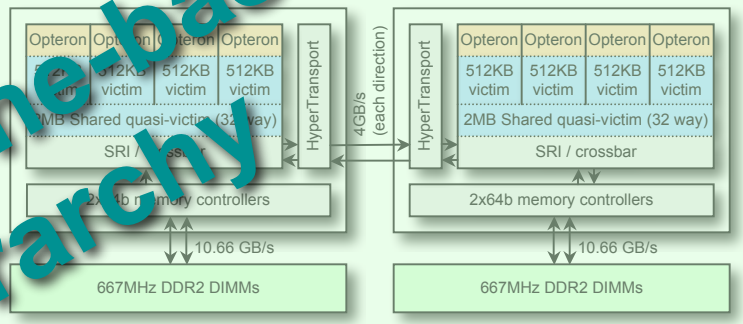


Multicore SMPs Used

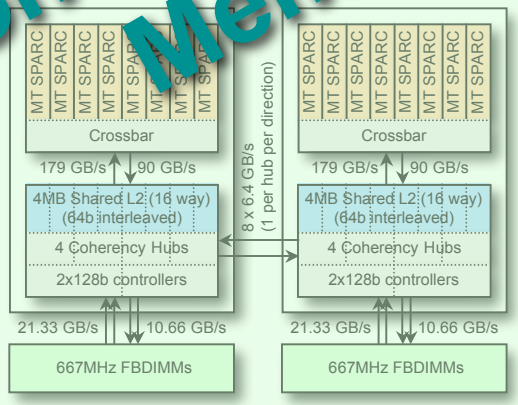
Intel Xeon E5345 (Clovertown)



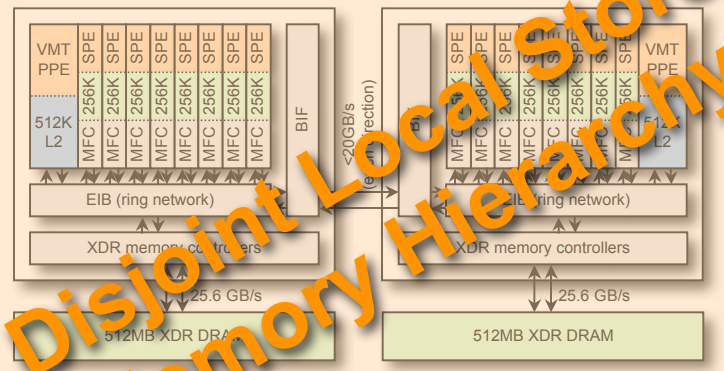
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

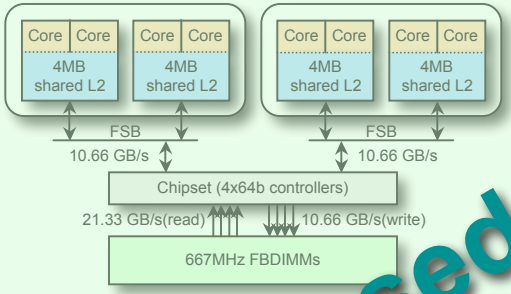


Conventional Cache-based Memory Hierarchy

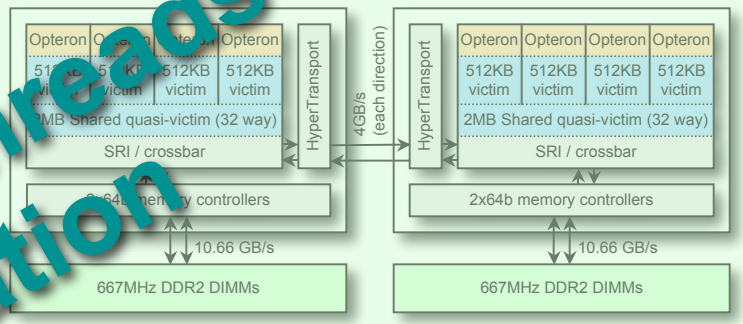
Disjoint Local Store Memory Hierarchy

Multicore SMPs Used

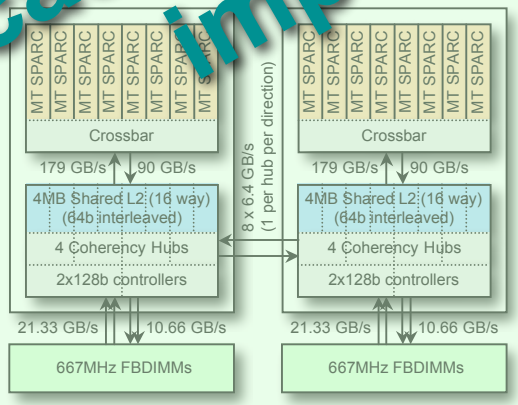
Intel Xeon E5345 (Clovertown)



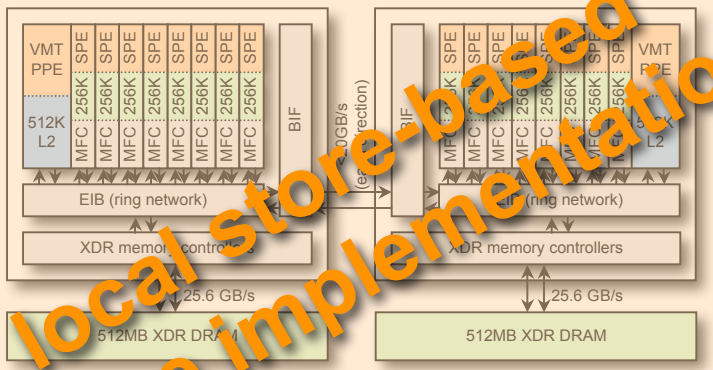
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

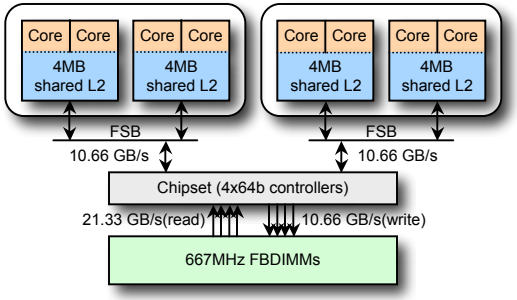


cache-based Pthreads implementation

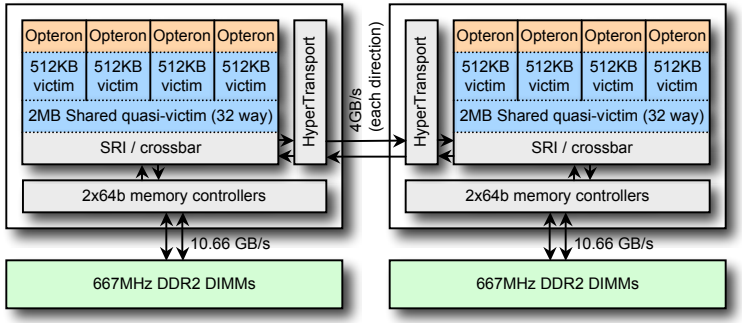
local store-based libspe implementation

Multicore SMPs Used

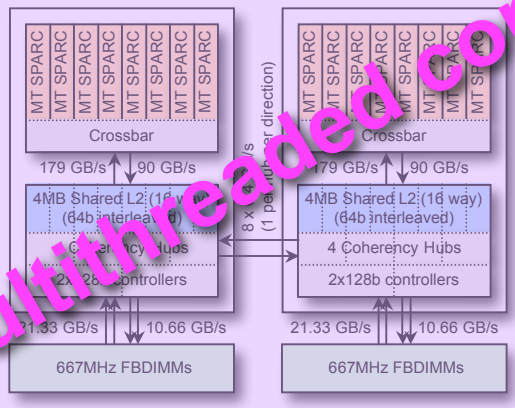
Intel Xeon E5345 (Clovertown)



AMD Opteron 2356 (Barcelona)

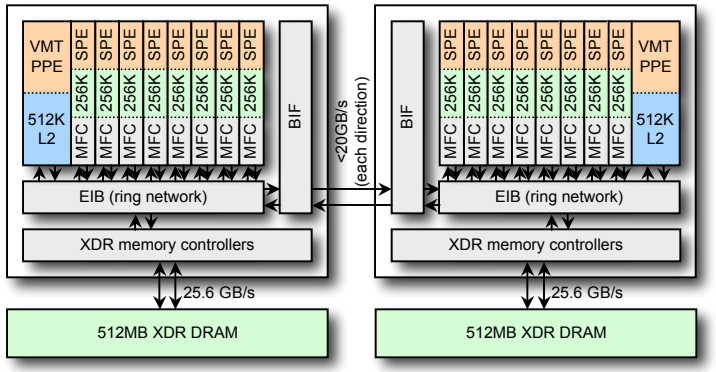


Sun T2+ T5140 (Victoria Falls)



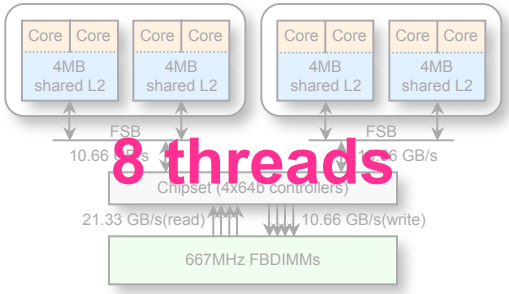
multithreaded cores

IBM QS20 Cell Blade

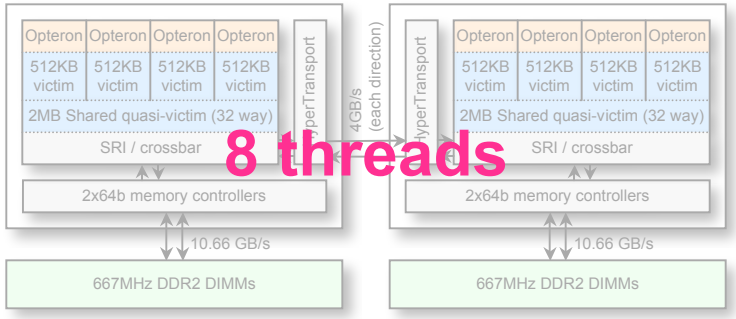


Multicore SMPs Used
(threads)

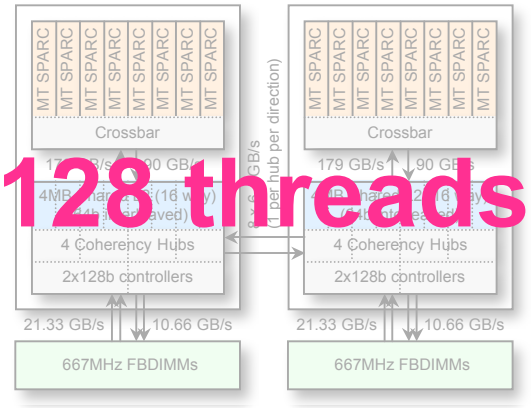
Intel Xeon E5345 (Clovertown)



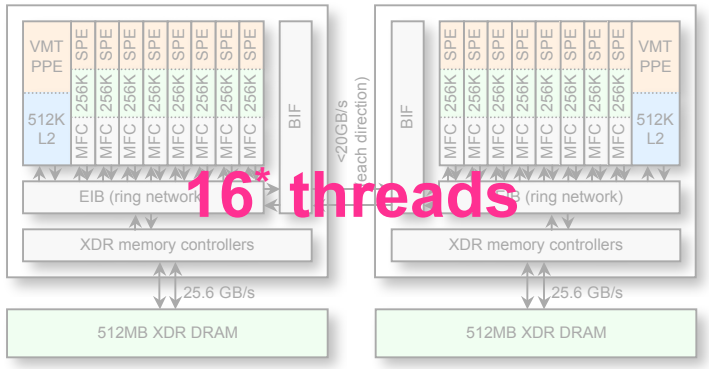
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

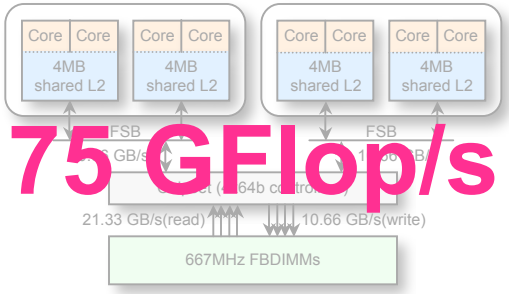


*SPEs only

Multicore SMPs Used

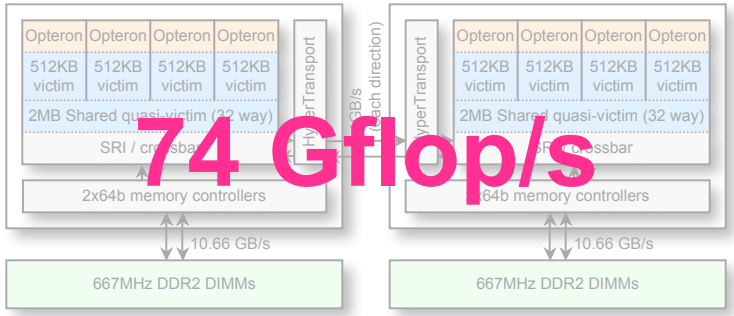
(peak double precision flops)

Intel Xeon E5345 (Clovertown)



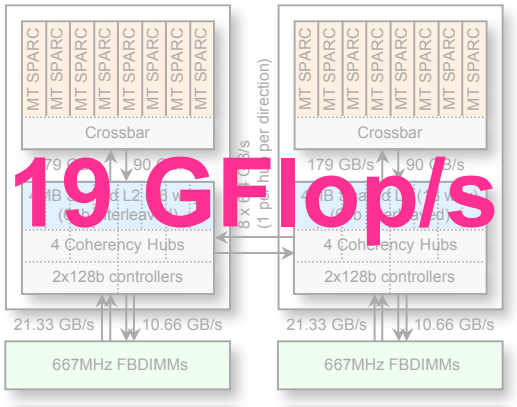
75 GFlop/s

AMD Opteron 2356 (Barcelona)



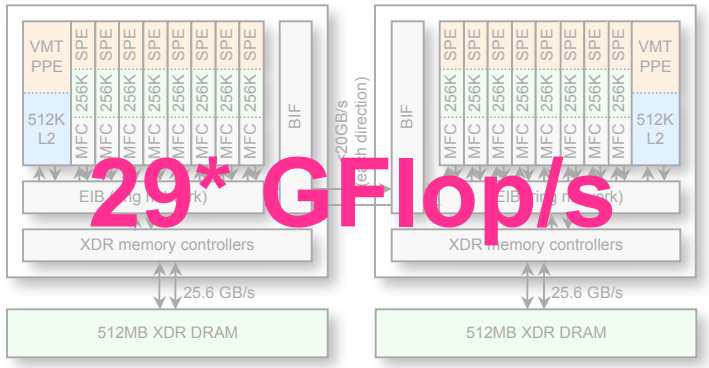
74 Gflop/s

Sun T2+ T5140 (Victoria Falls)



19 GFlop/s

IBM QS20 Cell Blade

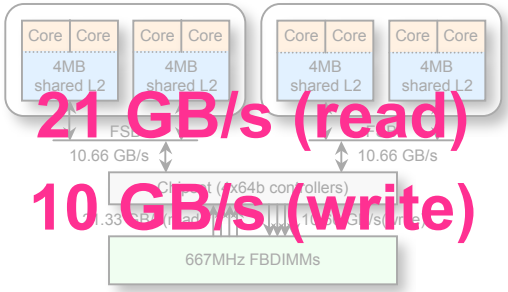


29* GFlop/s

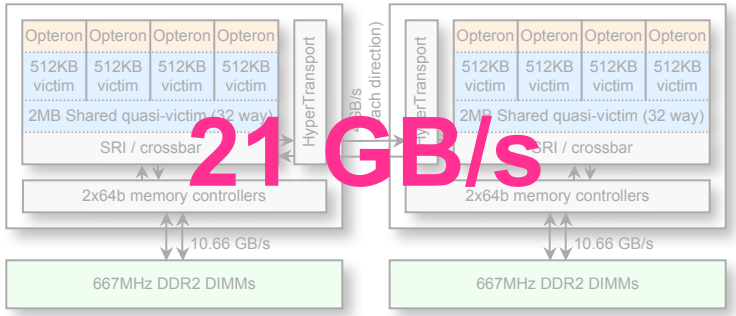
*SPEs only

Multicore SMPs Used
(total DRAM bandwidth)

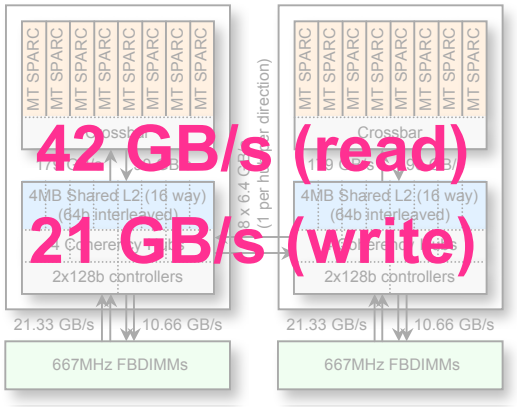
Intel Xeon E5345 (Clovertown)



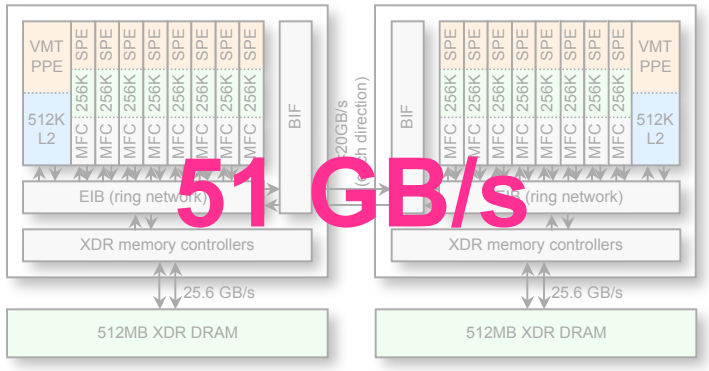
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade



*SPEs only

Categorization of Software Optimizations

Optimization Categorization

Maximizing *(attained)* **In-core Performance**	**Maximizing** *(attained)* **Memory Bandwidth**	**Minimizing** *(total)* **Memory Traffic**

Optimization Categorization

Maximizing In-core Performance	Maximizing Memory Bandwidth	Minimizing Memory Traffic

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

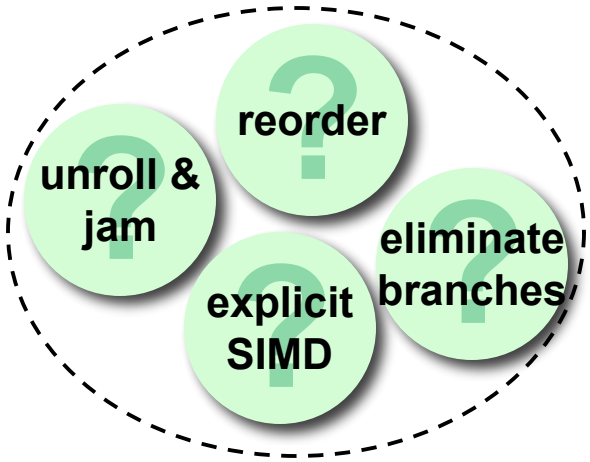
Optimization Categorization

Maximizing In-core Performance

Maximizing Memory Bandwidth

Minimizing Memory Traffic

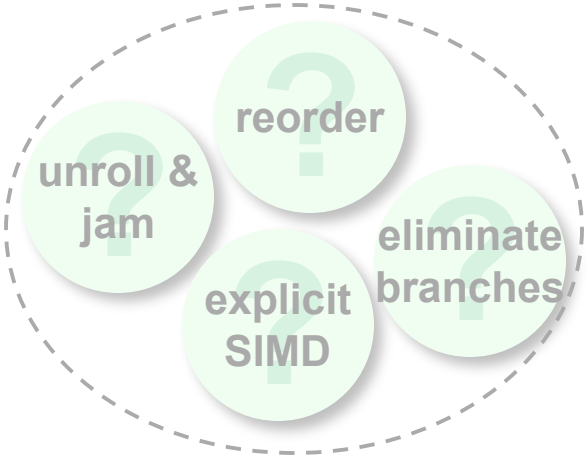
- Exploit in-core parallelism (ILP, DLP, etc…)
- Good (enough) floating-point balance



Optimization Categorization

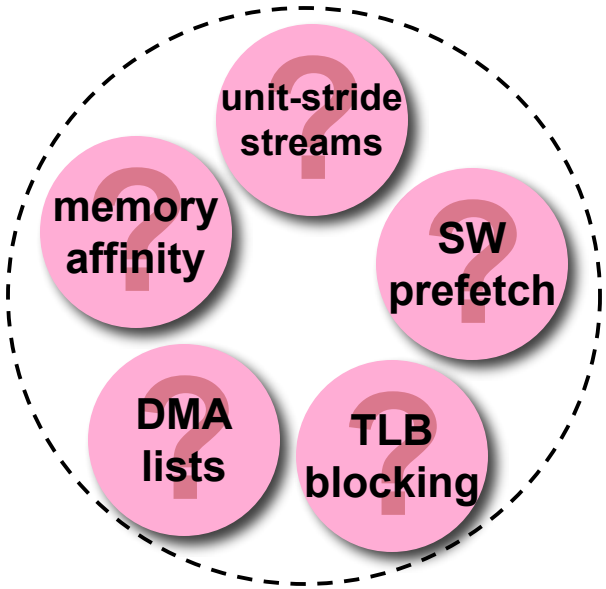
Maximizing In-core Performance

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



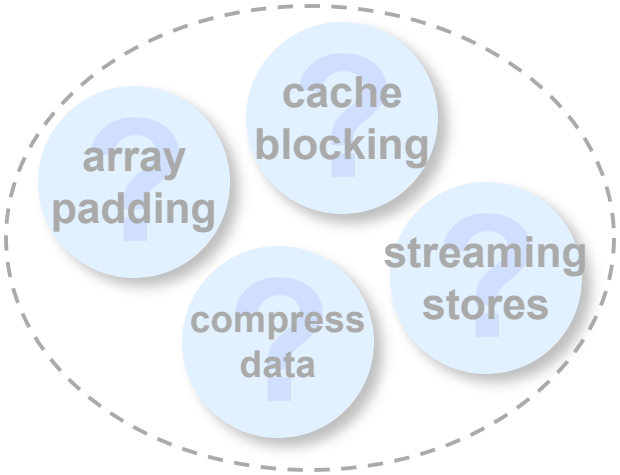
Maximizing Memory Bandwidth

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



Minimizing Memory Traffic

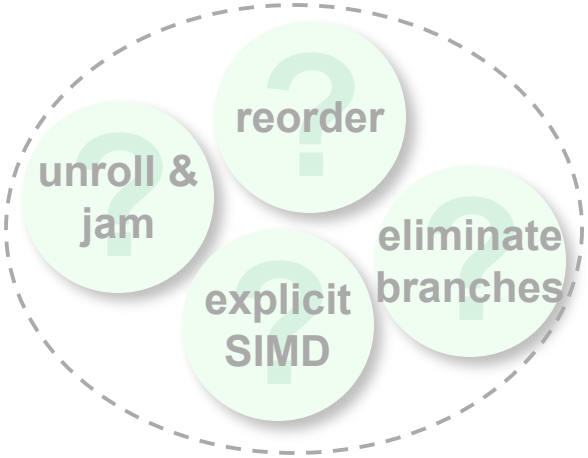
Eliminate:
- Capacity misses
- Conflict misses
- Compulsory misses
- Write allocate behavior



Optimization Categorization

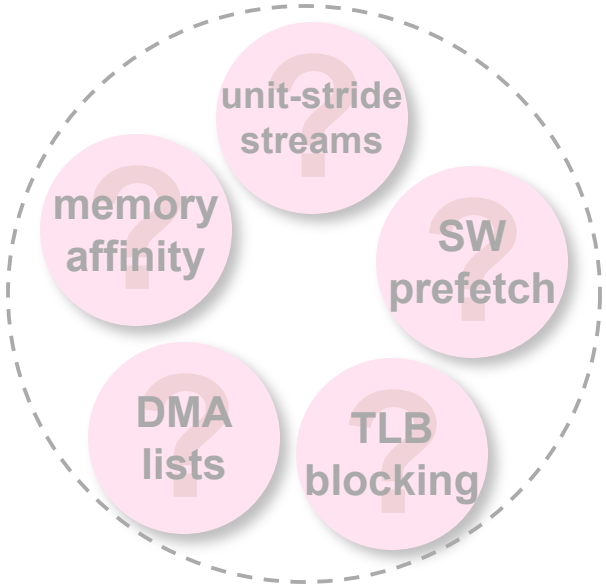
Maximizing In-core Performance

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



Maximizing Memory Bandwidth

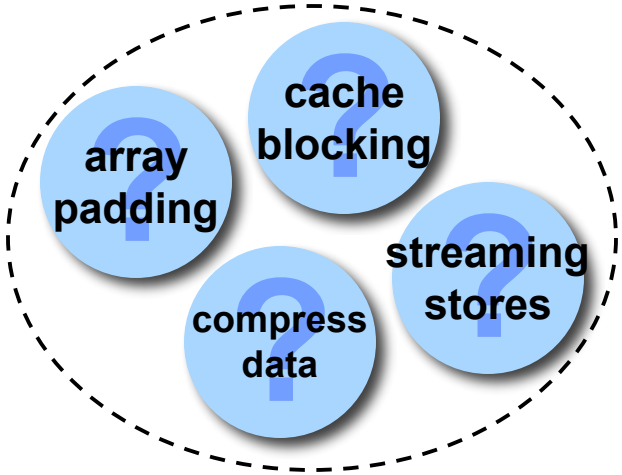
- Exploit NUMA
- Hide memory latency
- Satisfy Little's Law



Minimizing Memory Traffic

Eliminate:

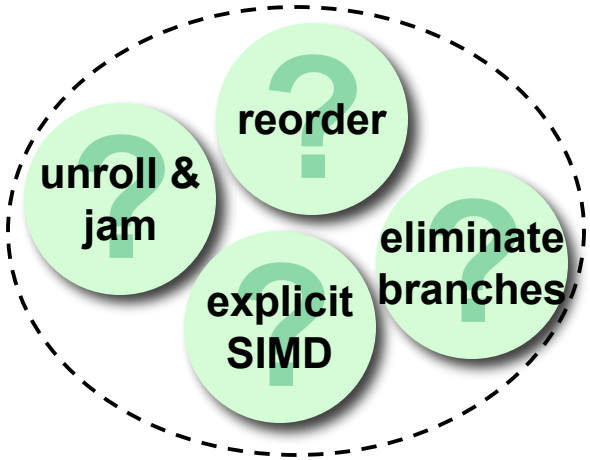
- Capacity misses
- Conflict misses
- Compulsory misses
- Write allocate behavior



Optimization Categorization

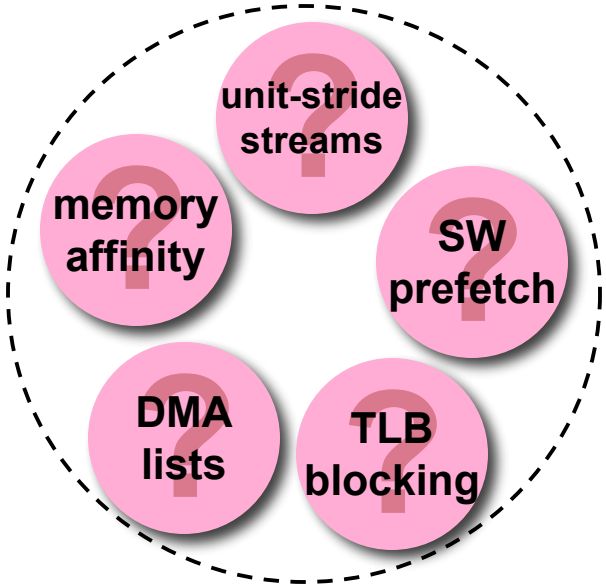
Maximizing In-core Performance

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



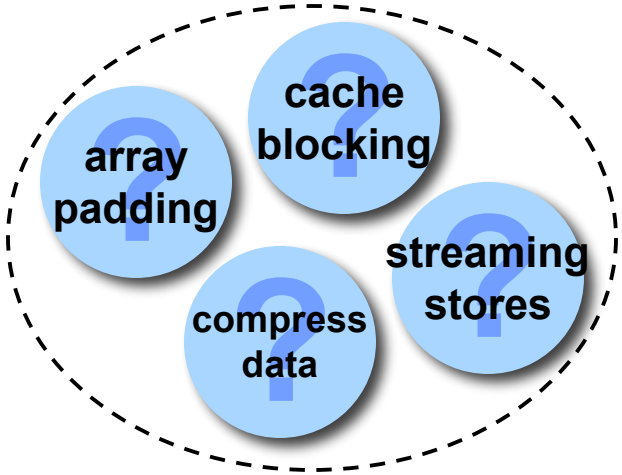
Maximizing Memory Bandwidth

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



Minimizing Memory Traffic

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



Optimization Categorization

Maximizing In-core Performance	Maximizing Memory Bandwidth	Minimizing Memory Traffic
•Exploit in-core parallelism (ILP, DLP, etc…)	•Exploit NUMA	Eliminate:
•Good (enough) floating-point balance	•Hide memory latency	•Capacity misses
	•Satisfy Little's Law	•Conflict misses
		•Compulsory 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 Problem

- 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

Auto-tuning?

- Provides **performance portability** across the existing breadth and evolution of microprocessors
- One time up front productivity cost is 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:
  - parameterized code generator (we wrote ours in Perl)
  - Auto-tuning exploration benchmark
    (combination of heuristics and exhaustive search)
- Can be extended with ISA specific optimizations (e.g. DMA, SIMD)

Auto-tuning Memory Intensive Kernels

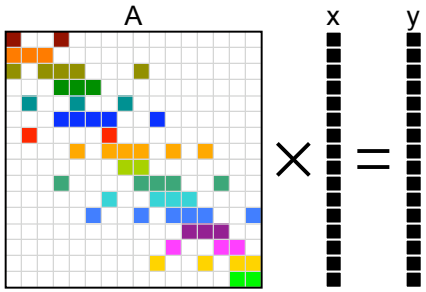
- Sparse Matrix Vector Multiplication (SpMV) — SC'07
- Lattice-Boltzmann Magneto-hydrodynamics (LBMHD) — IPDPS'08
- Explicit Heat Equation (Stencil) — SC'08

Auto-tuning Sparse Matrix-Vector Multiplication (SpMV)

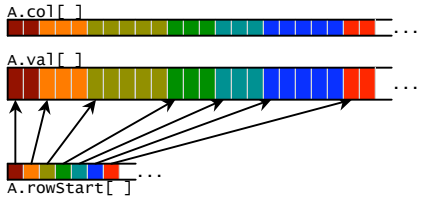
Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, James Demmel, "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms", Supercomputing (SC), 2007.

Sparse Matrix Vector Multiplication

- 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

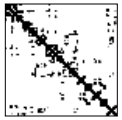
The Dataset (matrices)

- 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

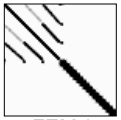
2K x 2K Dense matrix stored in sparse format

Dense

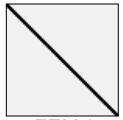
Well Structured (sorted by nonzeros/row)



Protein

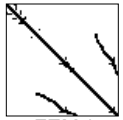


FEM / Spheres



FEM / Cantilever

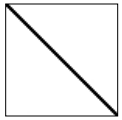
Wind Tunnel



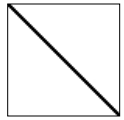
FEM / Harbor

QCD

FEM / Ship



Economics

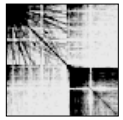


Epidemiology

Poorly Structured hodgepodge

FEM / Accelerator

Circuit

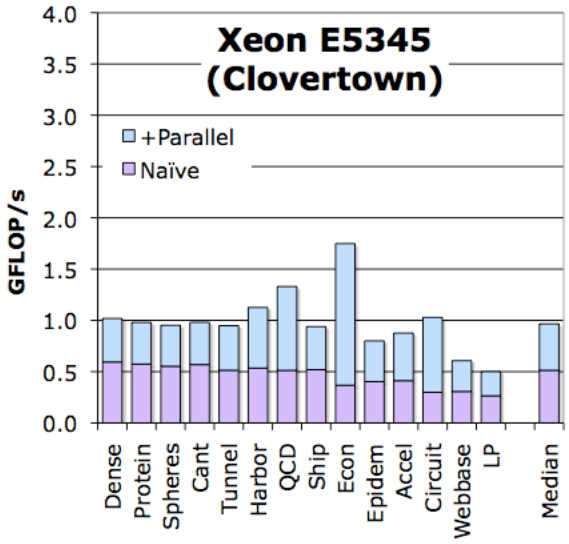


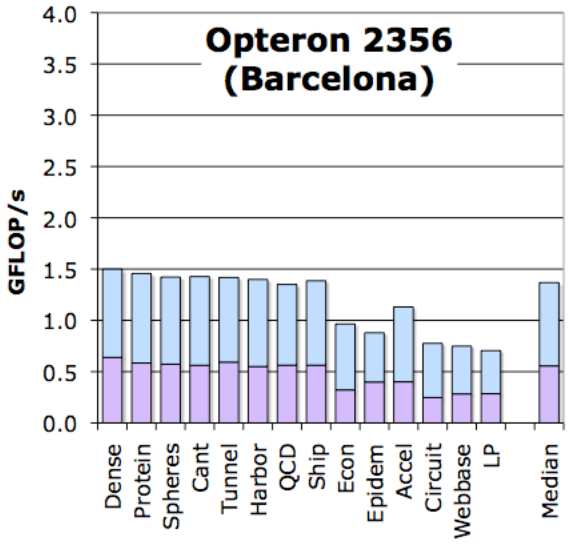
webbase

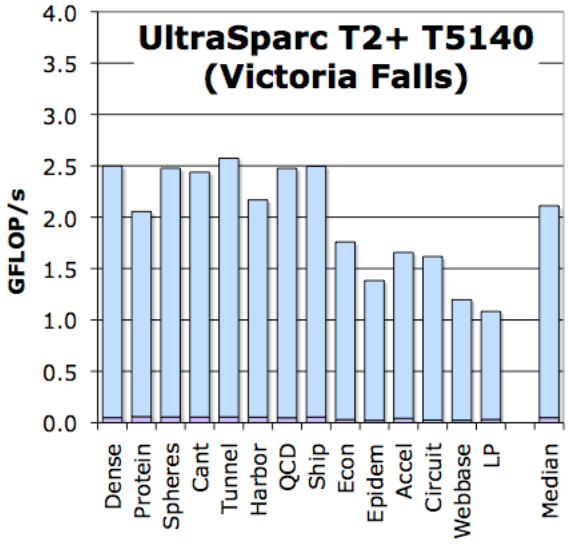
Extreme Aspect Ratio (linear programming)

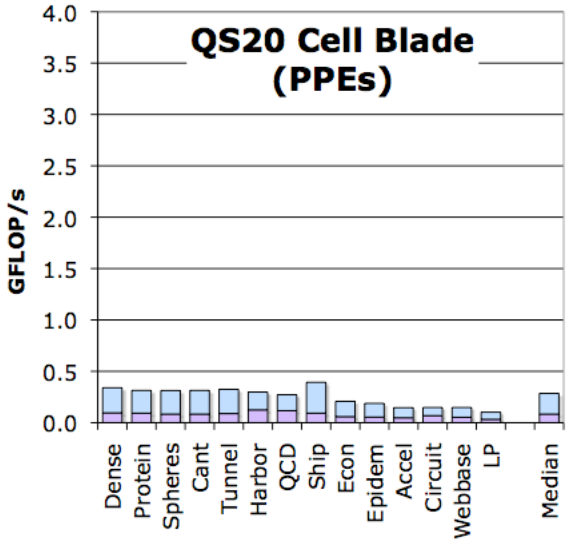
LP

SpMV Performance
(simple parallelization)





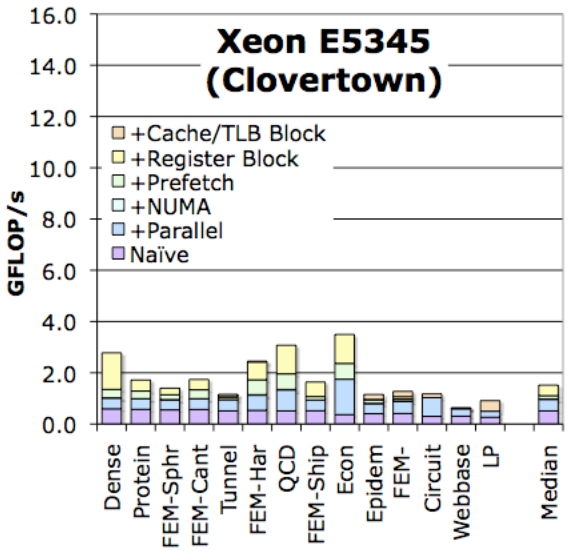


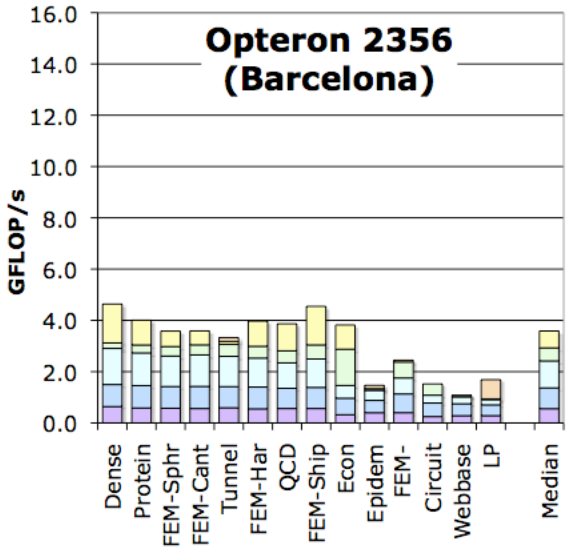


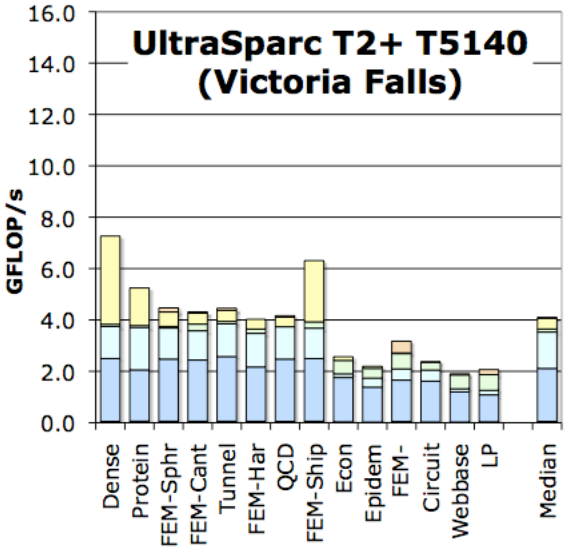
- Out-of-the box SpMV performance on a suite of 14 matrices
- **Scalability isn't great**
- **Is this performance good?**

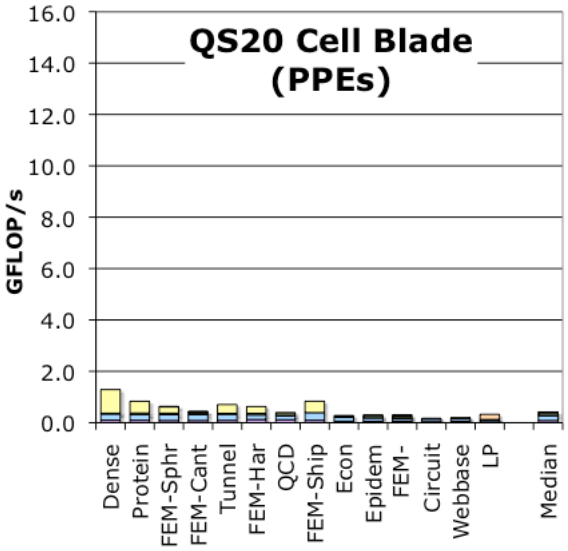
Naïve Pthreads
Naïve

Auto-tuned SpMV Performance
(portable C)

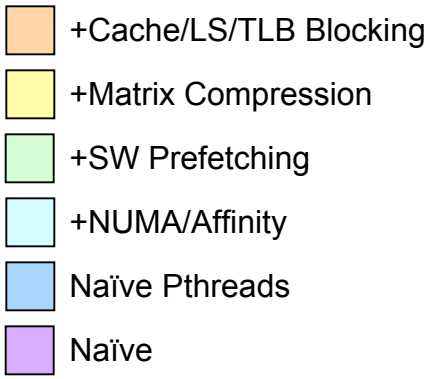






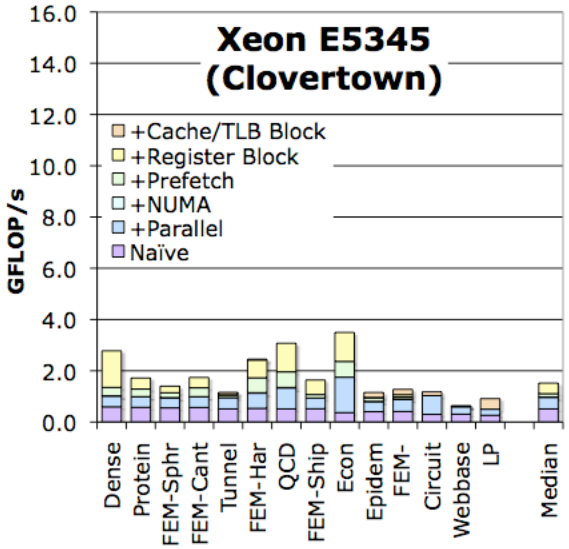


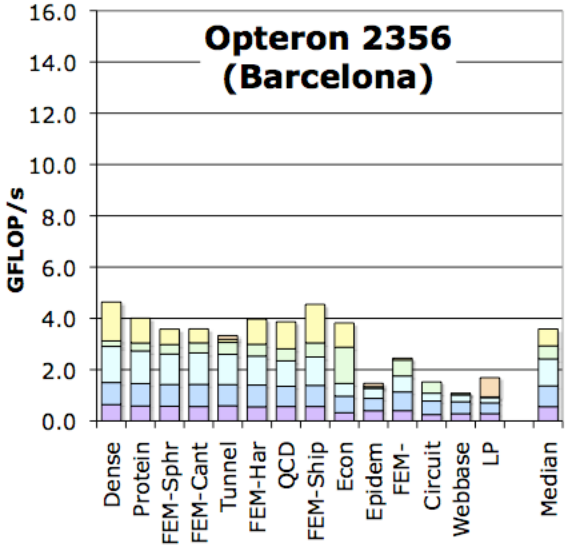
- Fully auto-tuned SpMV performance across the suite of matrices
- Why do some optimizations work better on some architectures?

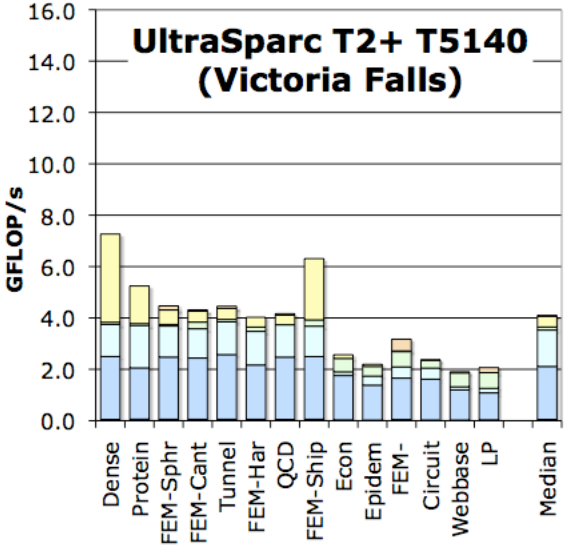


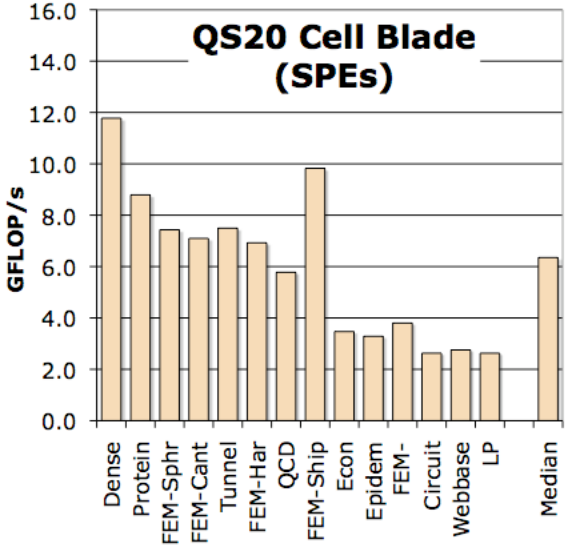
Auto-tuned SpMV Performance

(architecture specific optimizations)

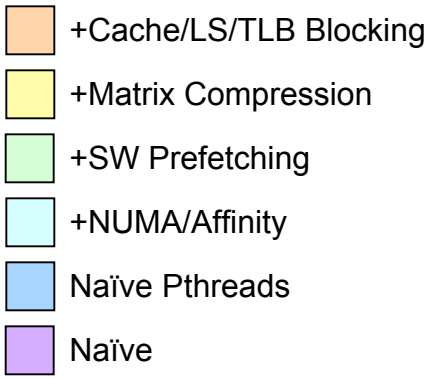






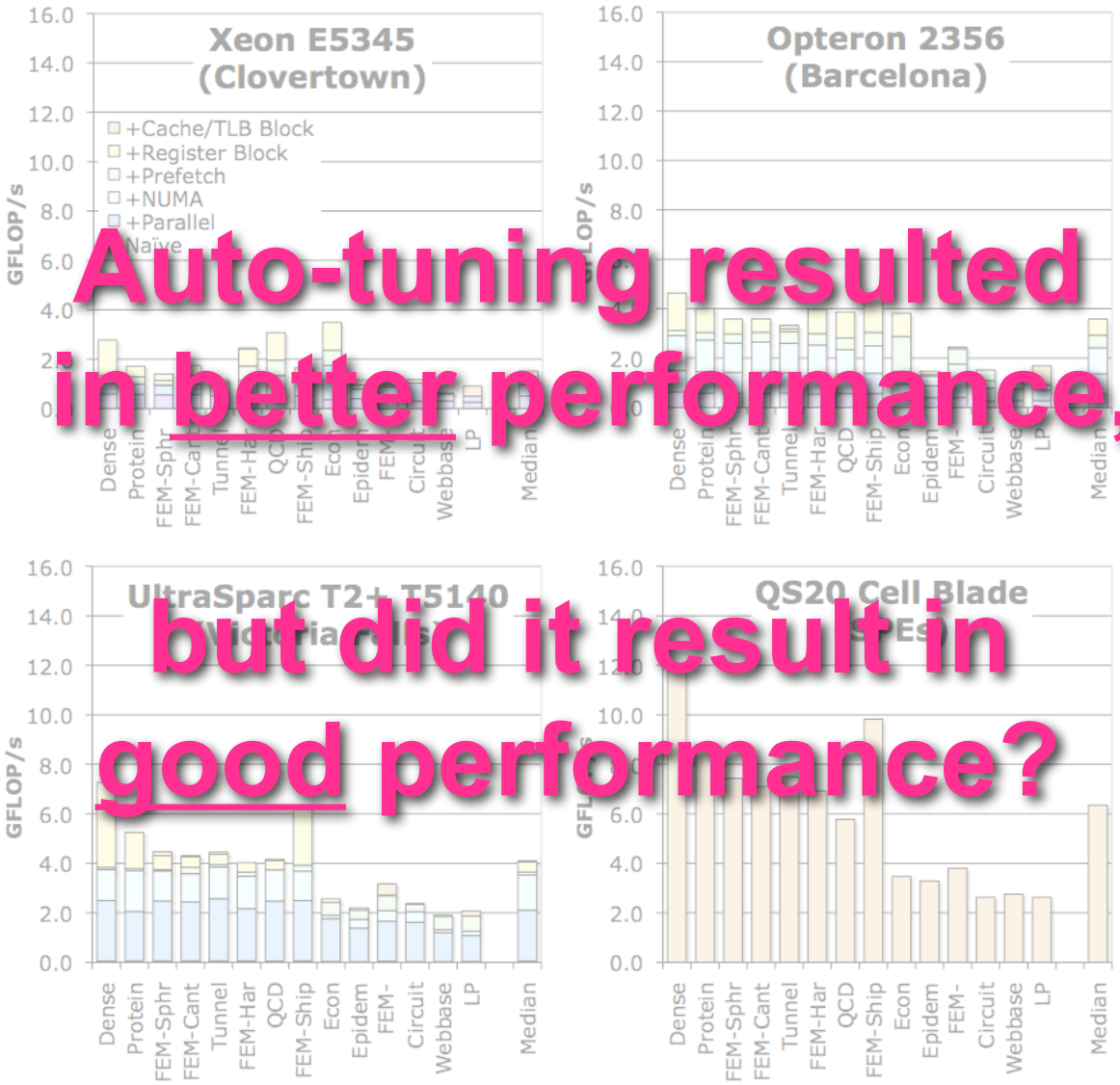


- Fully auto-tuned SpMV performance across the suite of matrices
- Included SPE/local store optimized version
- Why do some optimizations work better on some architectures?



Auto-tuned SpMV Performance

(architecture specific optimizations)

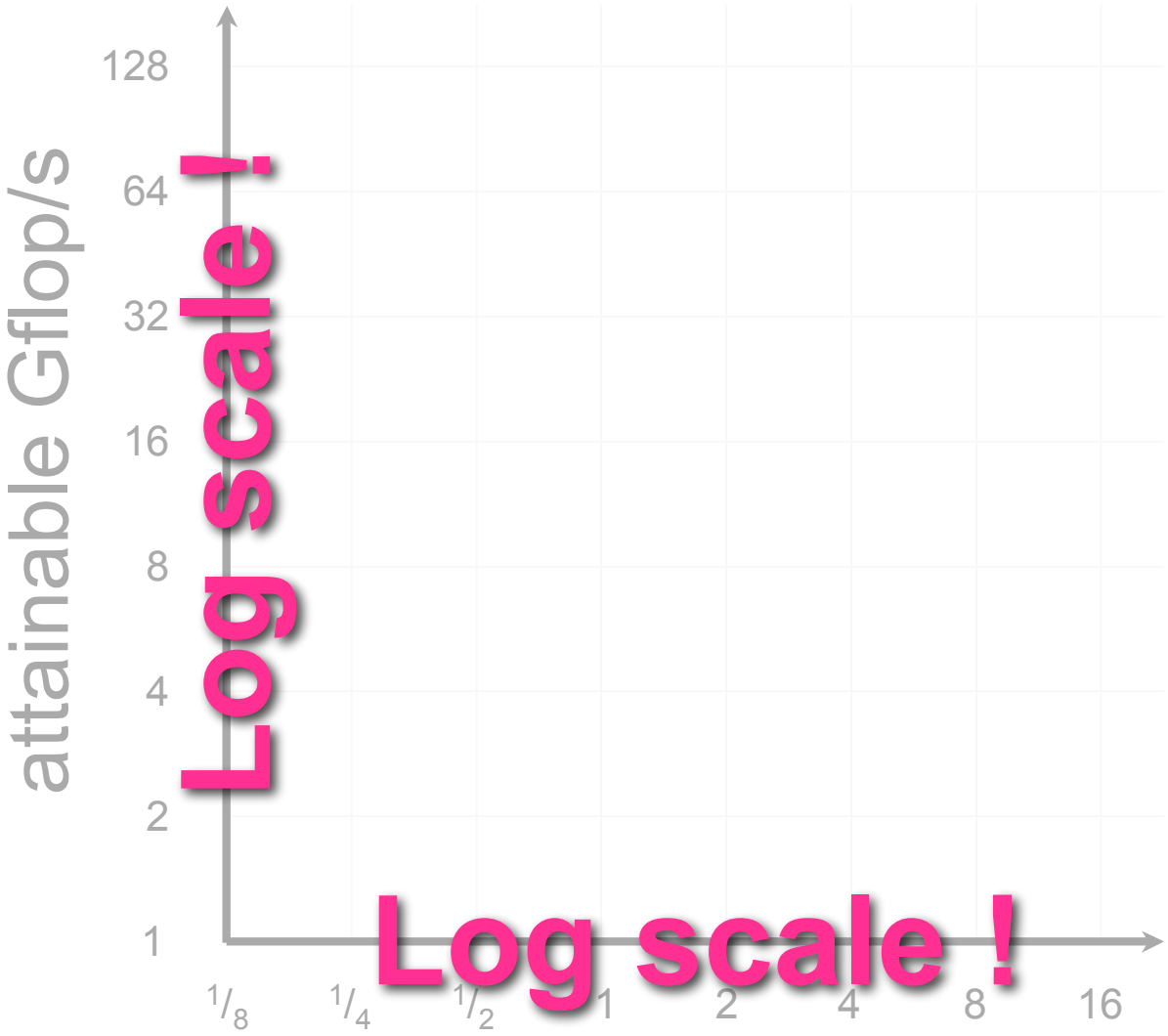


Auto-tuning resulted in better performance, but did it result in good performance?

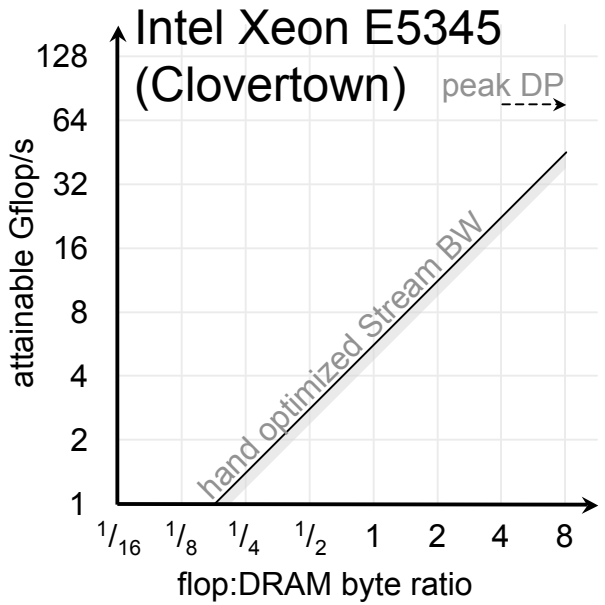
- Fully auto-tuned SpMV performance across the suite of matrices
- Included SPE/local store optimized version
- Why do some optimizations work better on some architectures?
- **Performance is better, but is performance good?**

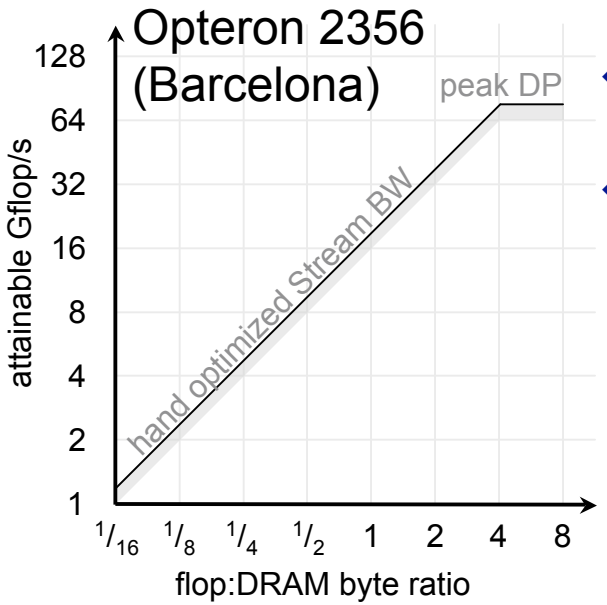
+Cache/LS/TLB Blocking
+Matrix Compression
+SW Prefetching
+NUMA/Affinity
Naïve Pthreads
Naïve

Roofline Model



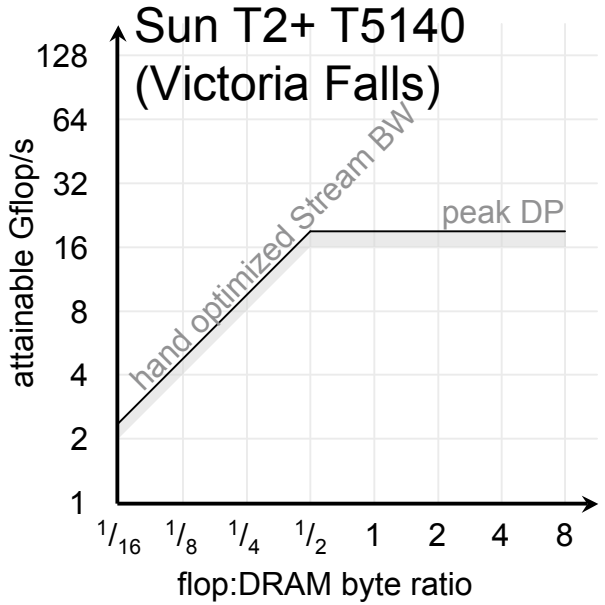
Naïve Roofline Model

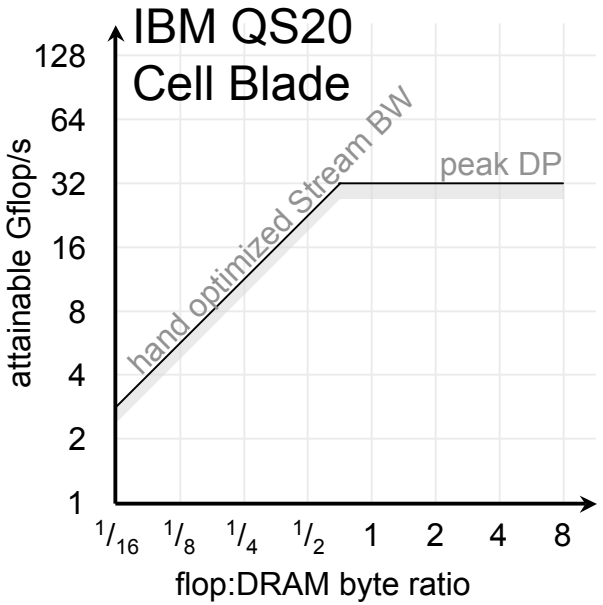




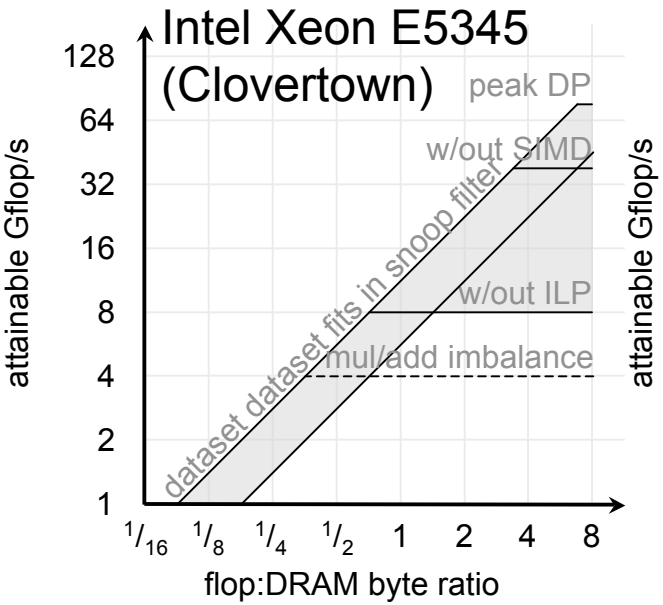
- Unrealistically optimistic model
- Hand optimized Stream BW benchmark

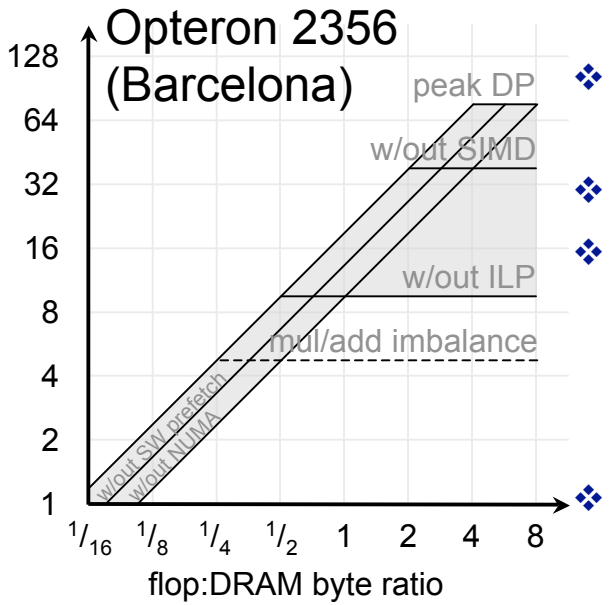
Gflop/s(AI) = min { Peak Gflop/s, StreamBW * AI }





Roofline model for SpMV

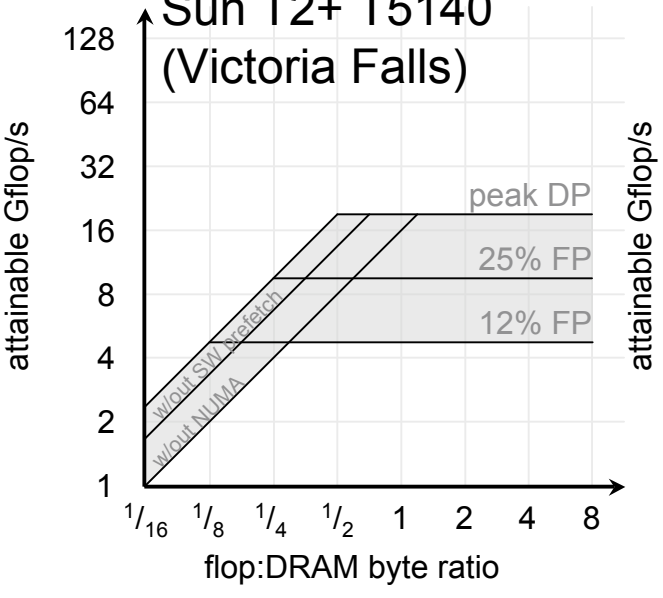


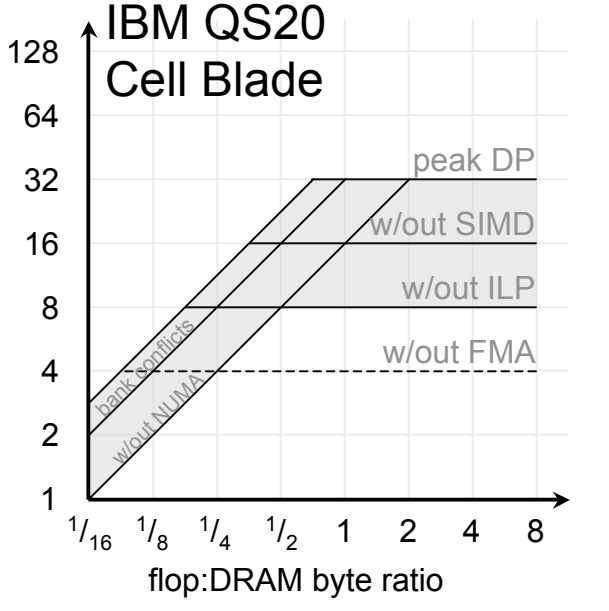


- Double precision roofline models
- In-core optimizations 1..i
- DRAM optimizations 1..j

$$GFlops_{i,j}(AI) = \min \begin{cases} InCoreGFlops_i \\ StreamBW_j * AI \end{cases}$$

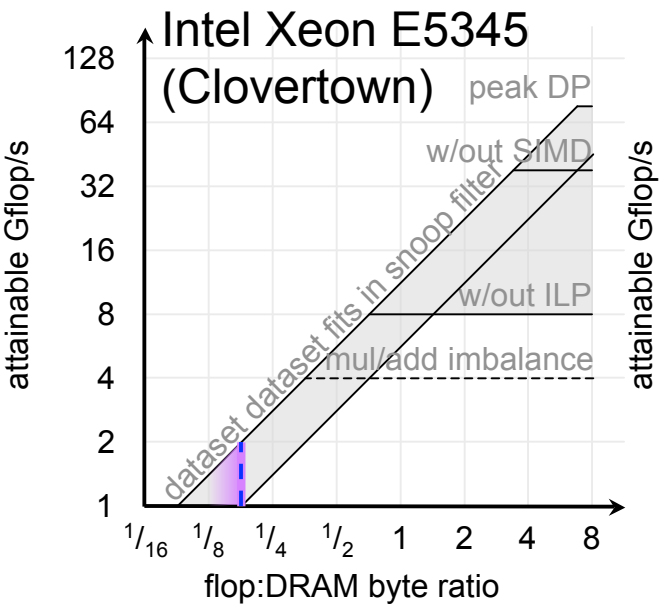
- FMA is inherent in SpMV (place at bottom)

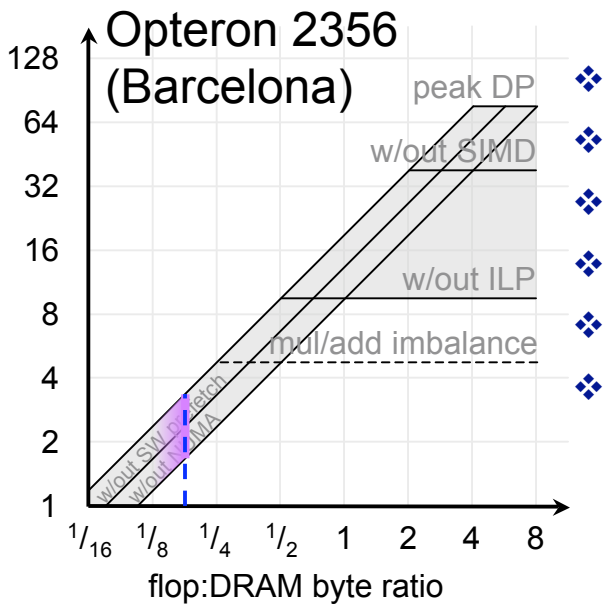




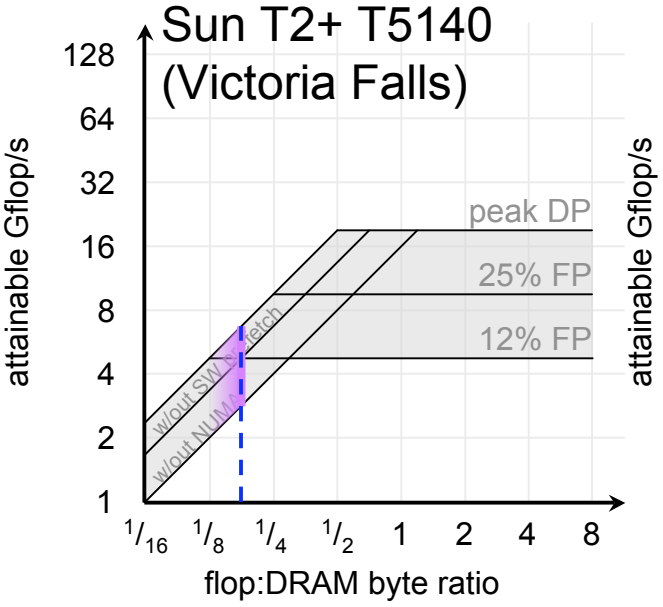
Roofline model for SpMV

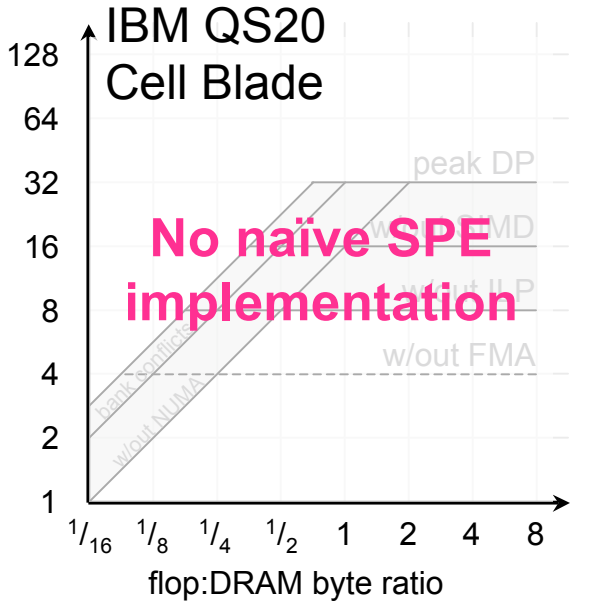
(overlay arithmetic intensity)





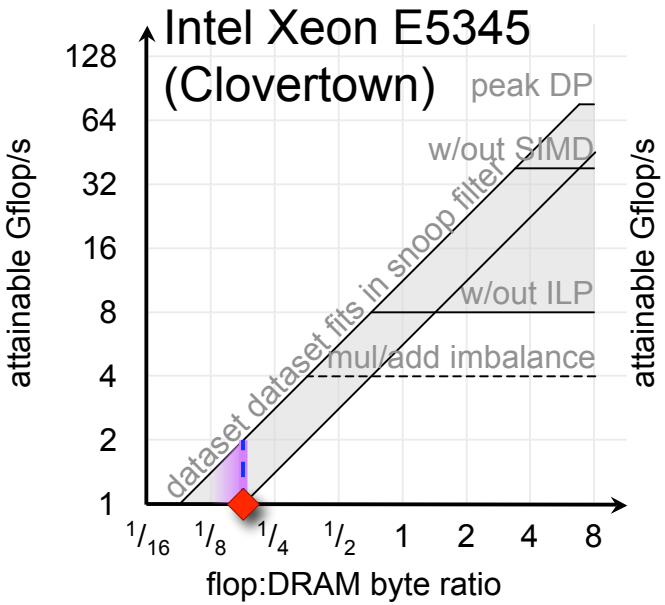
- Two unit stride streams
- Inherent FMA
- No ILP
- No DLP
- FP is 12-25%
- Naïve compulsory flop:byte < 0.166

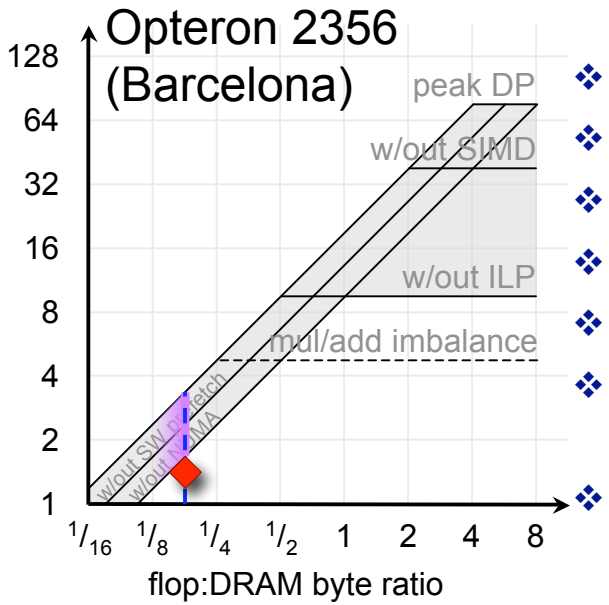


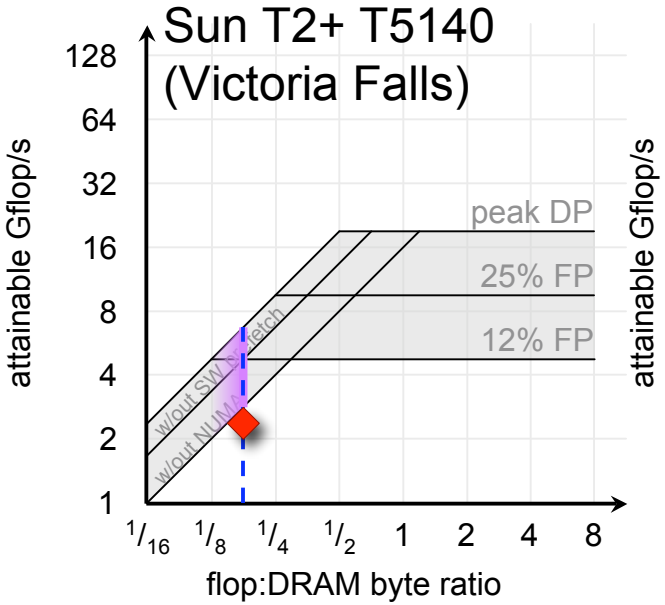


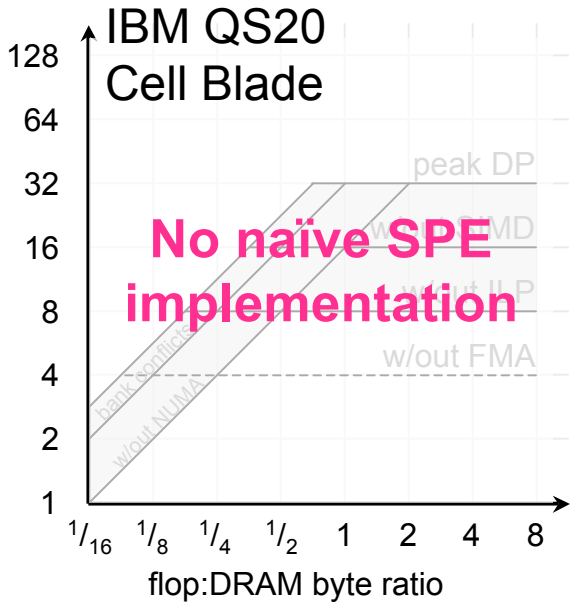
Roofline model for SpMV

(out-of-the-box parallel)





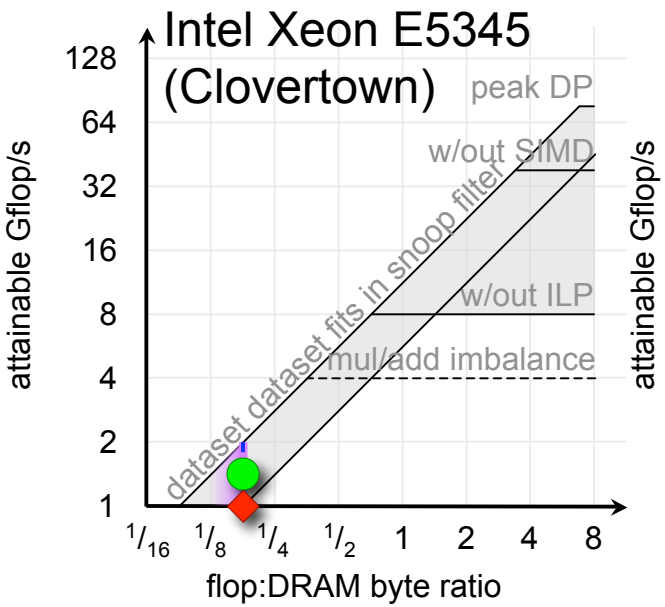


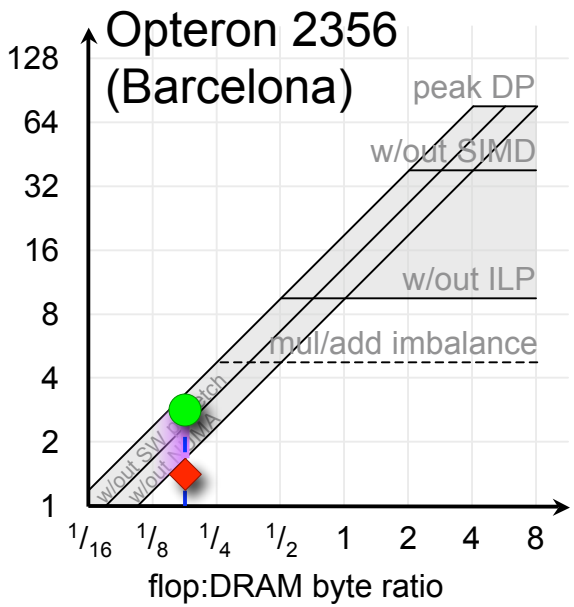


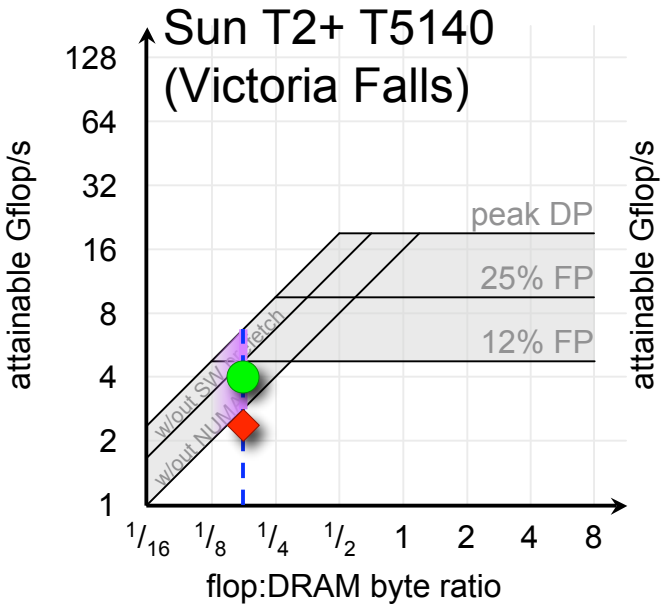
- Two unit stride streams
- Inherent FMA
- No ILP
- No DLP
- FP is 12-25%
- Naïve compulsory flop:byte < 0.166
- For simplicity: **dense matrix in sparse format**

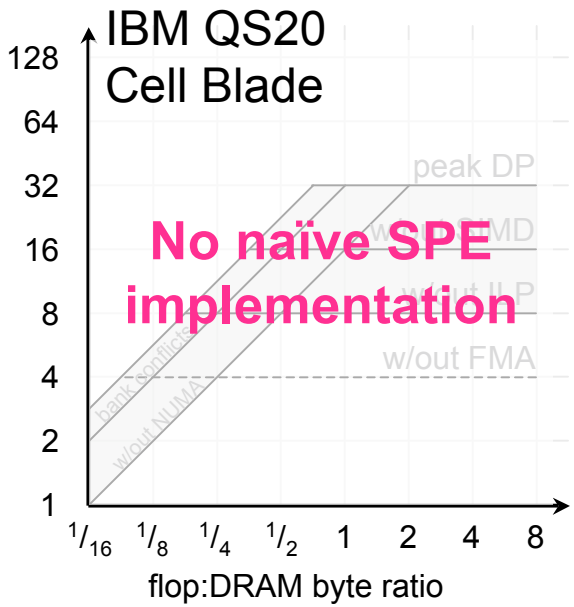
Roofline model for SpMV

(NUMA & SW prefetch)





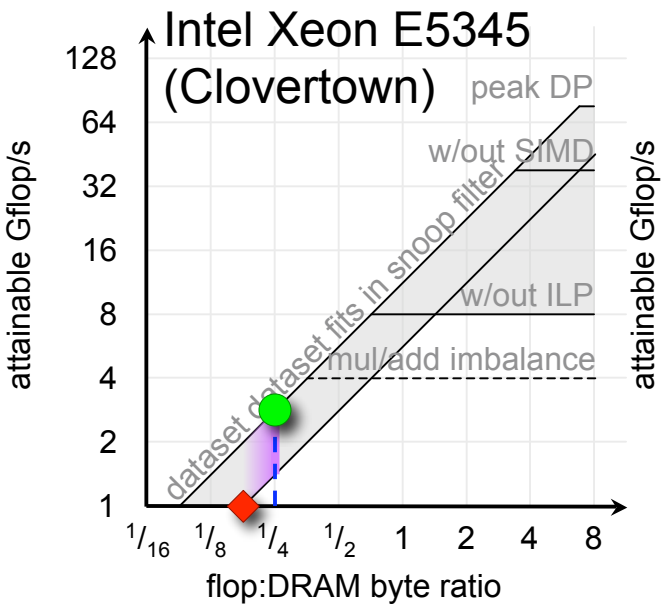


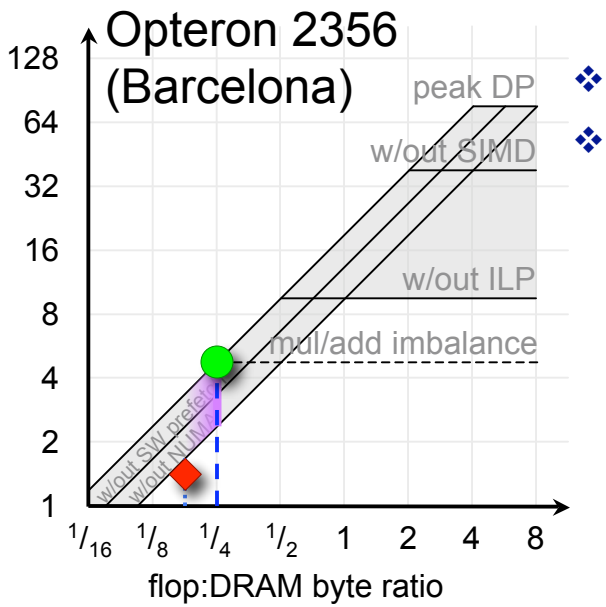


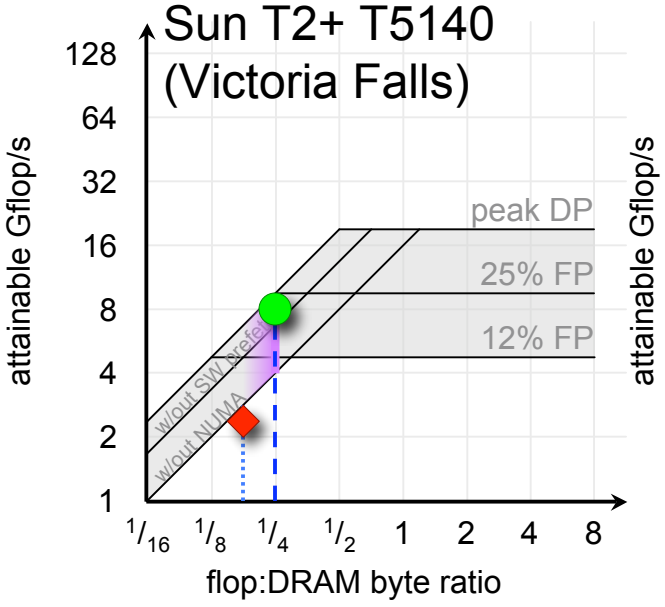
- compulsory flop:byte ~ 0.166
- utilize all memory channels

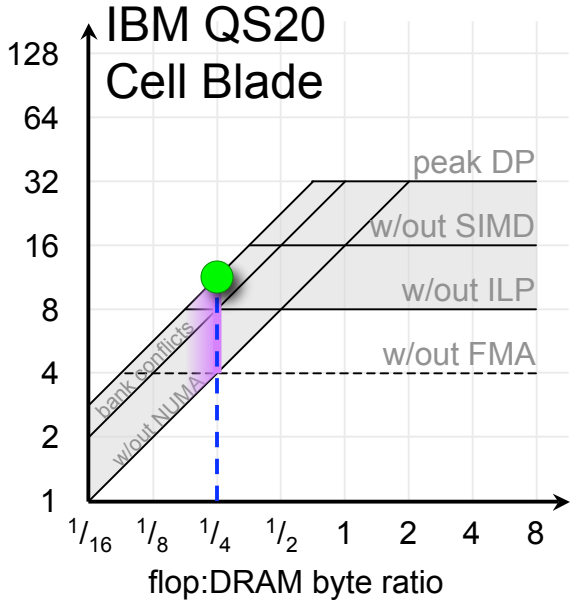
Roofline model for SpMV

(matrix compression)



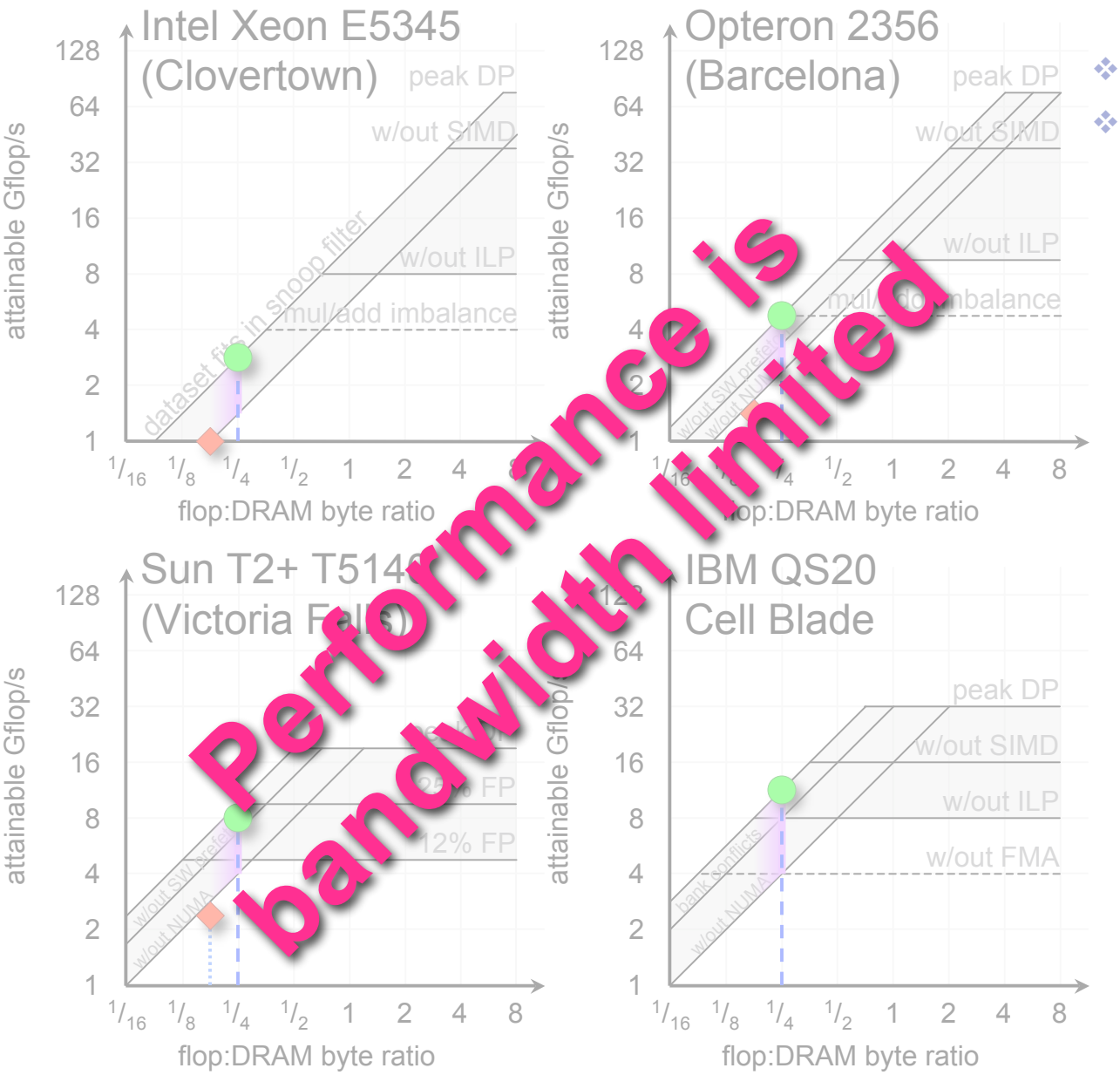






- Inherent FMA
- Register blocking improves ILP, DLP, flop:byte ratio, and FP% of instructions

Roofline model for SpMV
(matrix compression)



Performance is bandwidth limited

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

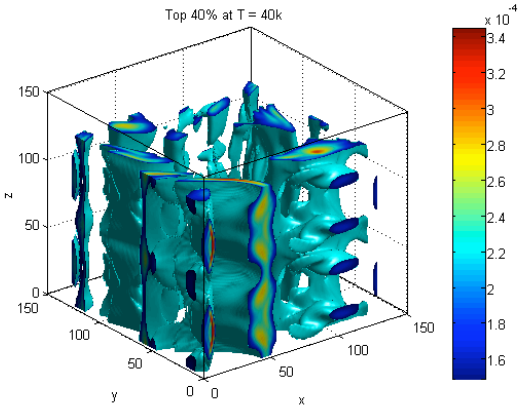
Auto-tuning Lattice-Boltzmann Magneto-Hydrodynamics (LBMHD)

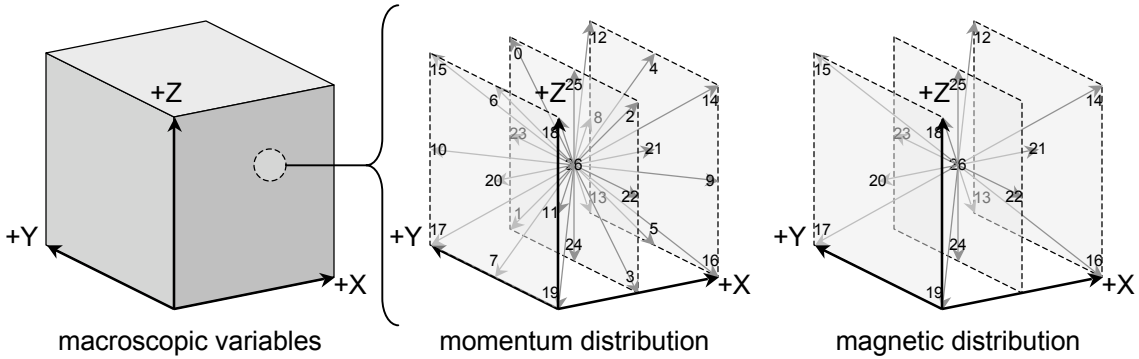
Samuel Williams, Jonathan Carter, Leonid Oliker, John Shalf, Katherine Yelick, "Lattice Boltzmann Simulation Optimization on Leading Multicore Platforms", International Parallel & Distributed Processing Symposium (IPDPS), 2008.

Best Paper, Application Track

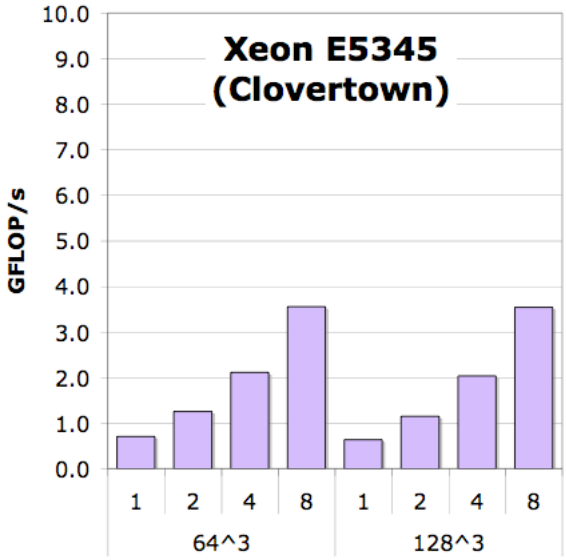
LBMHD

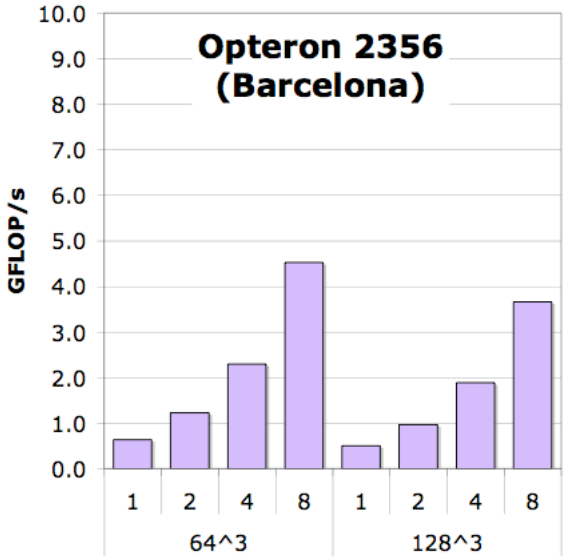
- Plasma turbulence simulation via Lattice Boltzmann Method
- Two distributions:
  - momentum distribution (27 scalar components)
  - magnetic distribution (15 vector components)
- Three macroscopic quantities:
  - Density
  - Momentum (vector)
  - Magnetic Field (vector)
- Arithmetic Intensity:
  - 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)**
- Cache capacity requirements are independent of problem size
- Two problem sizes:
  - 64^3 (0.3 GB)
  - 128^3 (2.5 GB)
- periodic boundary conditions

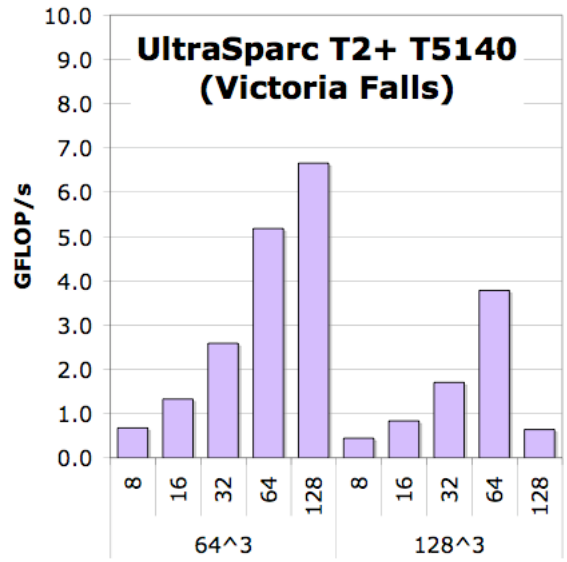


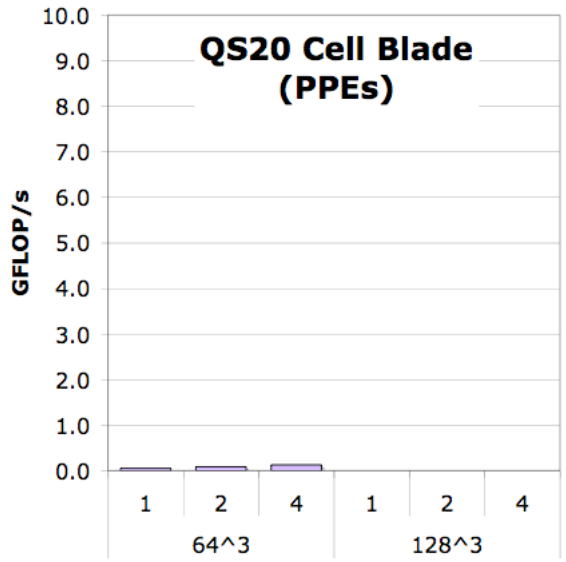


Initial LBMHD Performance







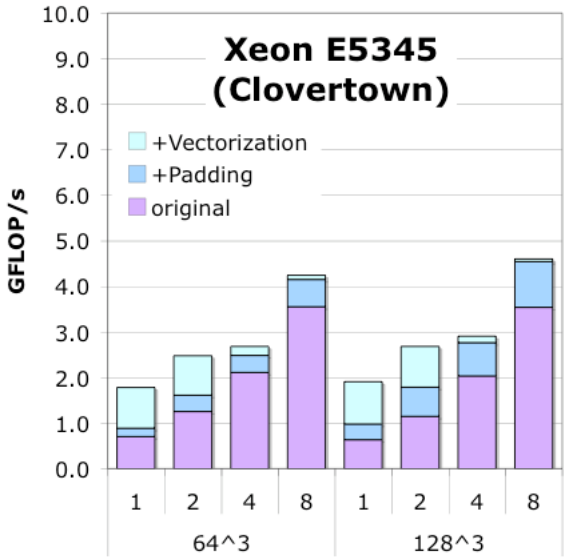


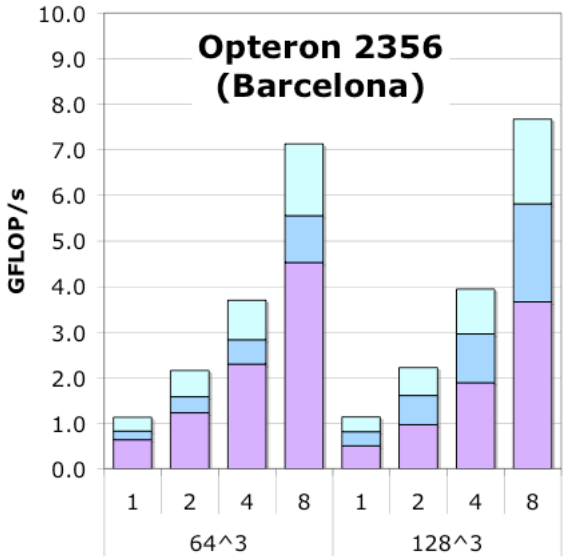
- Generally, scalability looks good
- **Scalability is good**
- **but is performance good?**

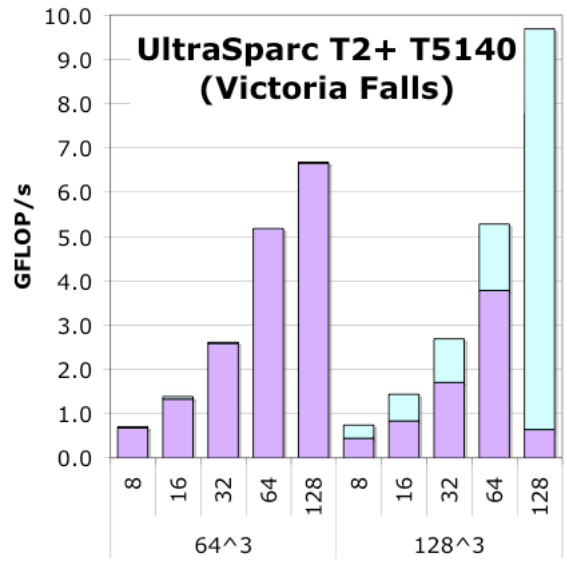
Naïve+NUMA

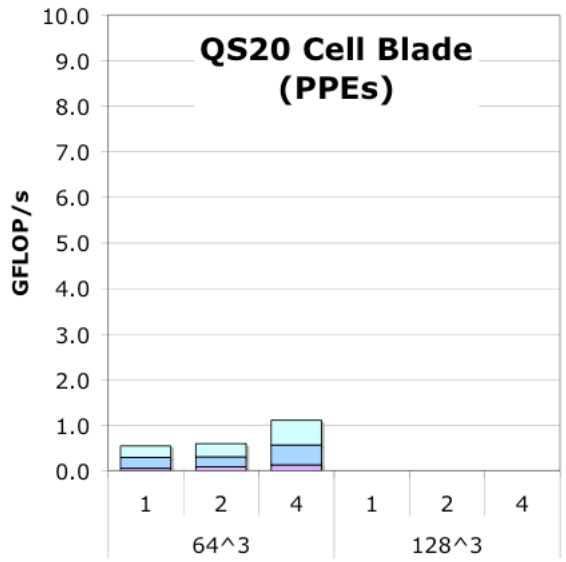
*collision() only

Auto-tuned LBMHD Performance (portable C)









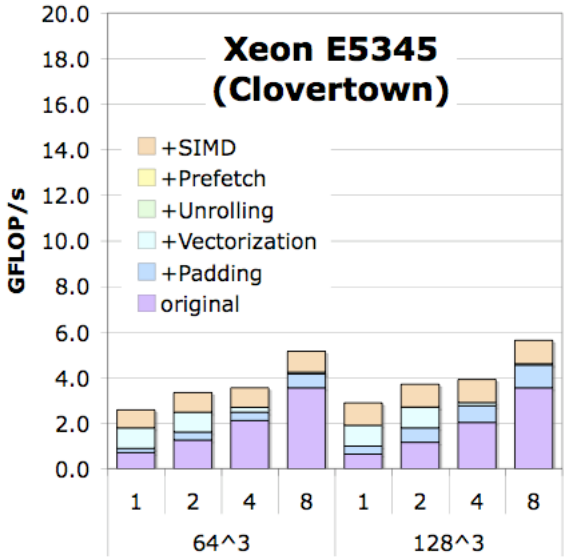
- Auto-tuning avoids cache conflict and TLB capacity misses
- Additionally, it exploits SIMD where the compiler doesn't
- Include a SPE/Local Store optimized version

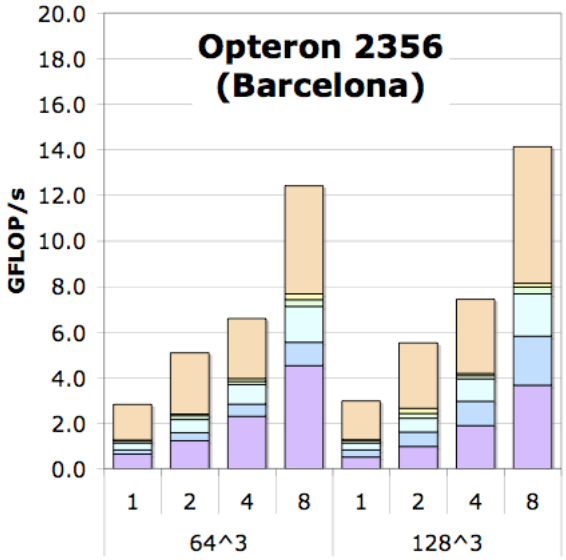
+Vectorization
+Padding
Naïve+NUMA

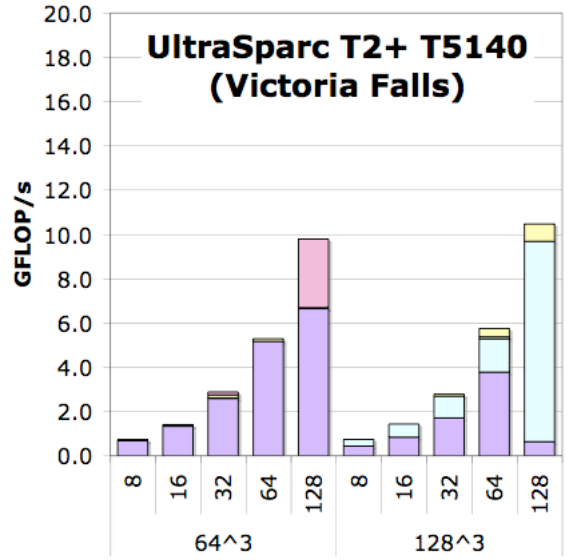
*collision() only

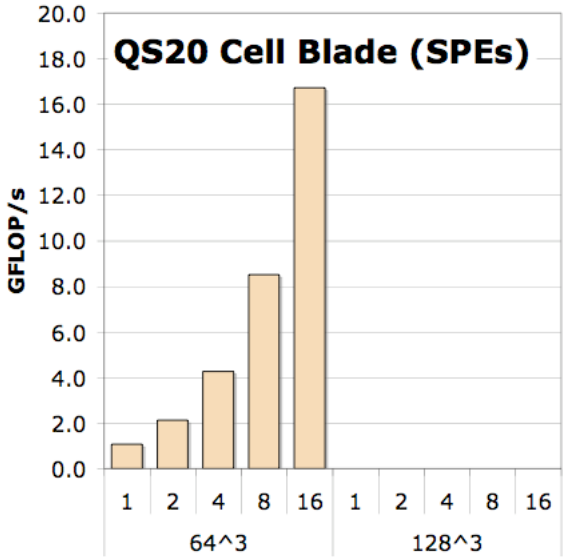
Auto-tuned LBMHD Performance

(architecture specific optimizations)

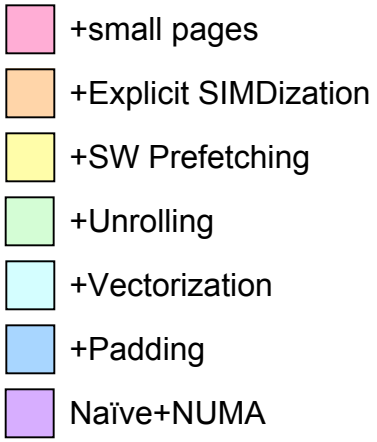






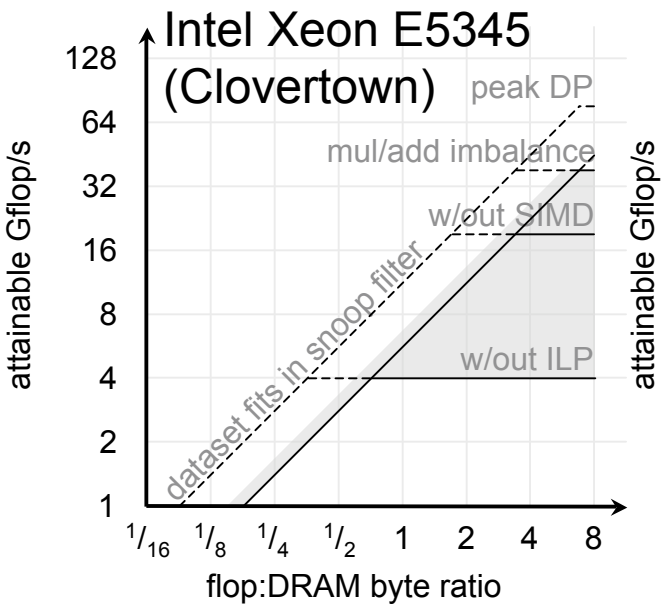


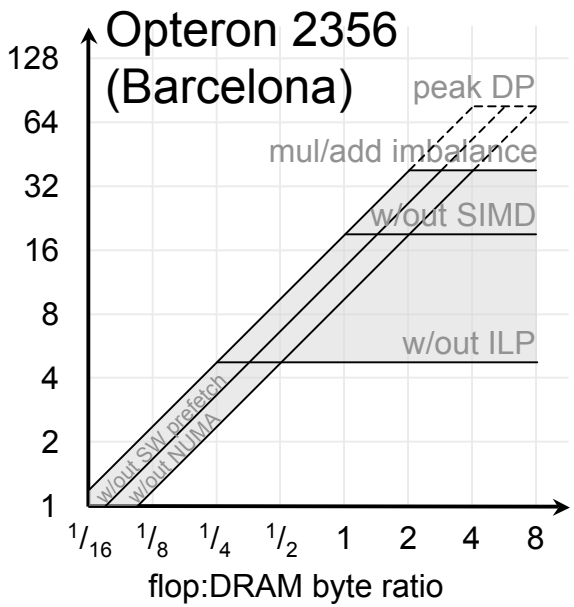
- Auto-tuning avoids cache conflict and TLB capacity misses
- Additionally, it exploits SIMD where the compiler doesn't
- Include a SPE/Local Store optimized version

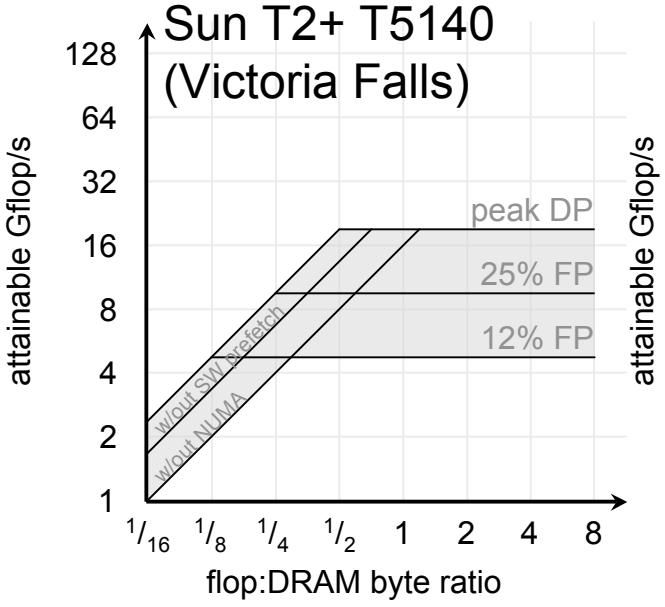


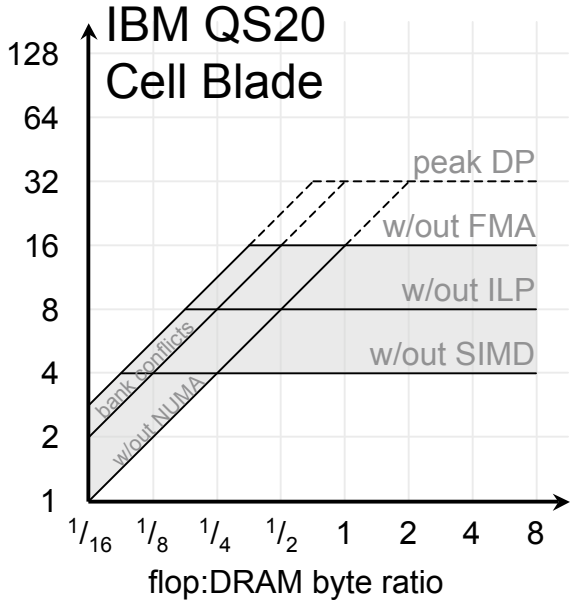
*collision() only

Roofline model for LBMHD





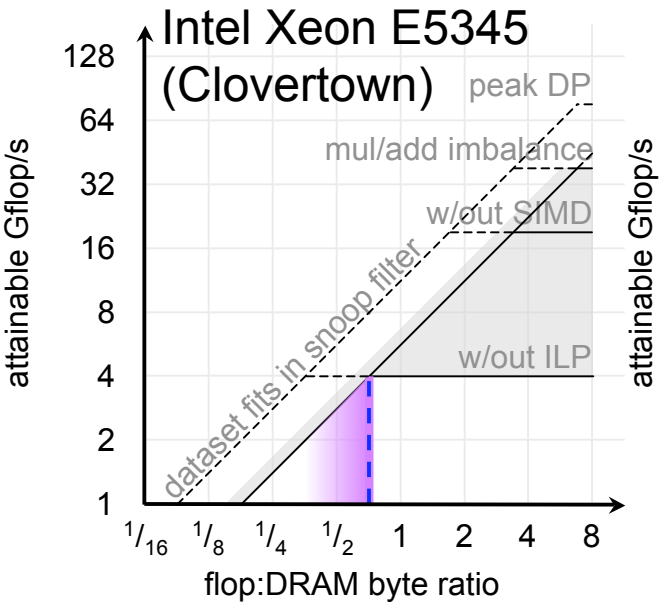


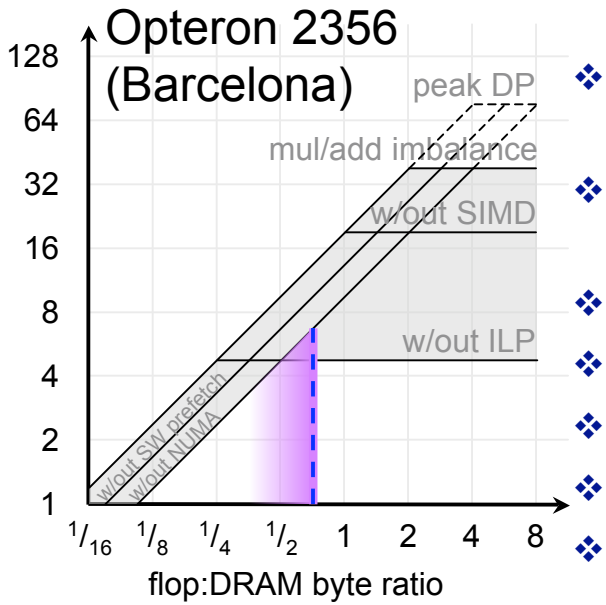


- Far more adds than multiplies (imbalance)
- Huge data sets

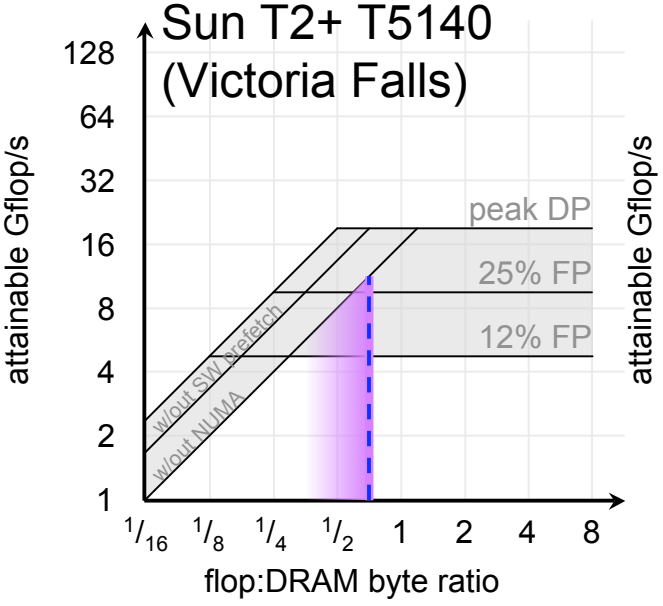
Roofline model for LBMHD

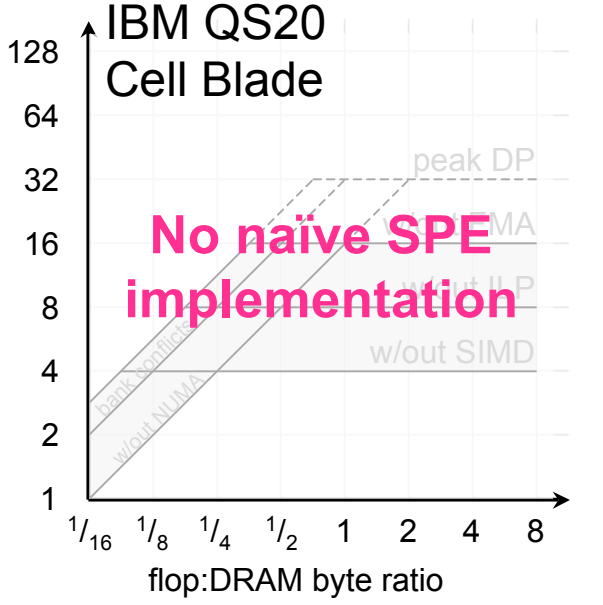
(overlay arithmetic intensity)





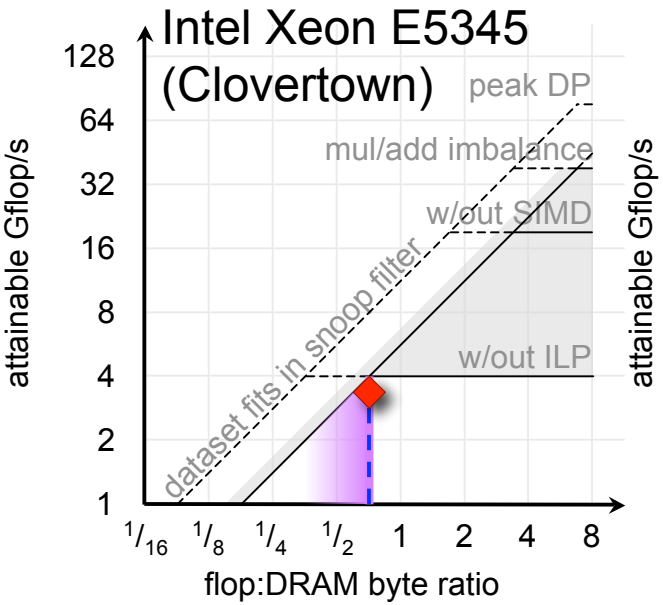
- Far more adds than multiplies (imbalance)
- **Essentially random access to memory**
- Flop:byte ratio ~0.7
- NUMA allocation/access
- Little ILP
- No DLP
- High conflict misses

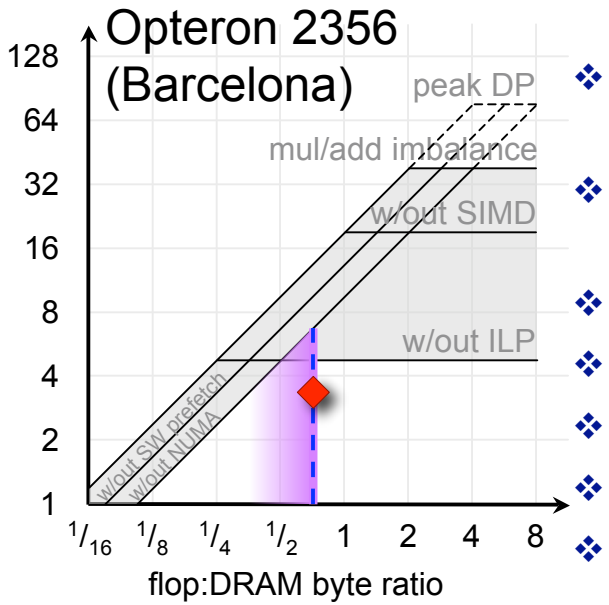




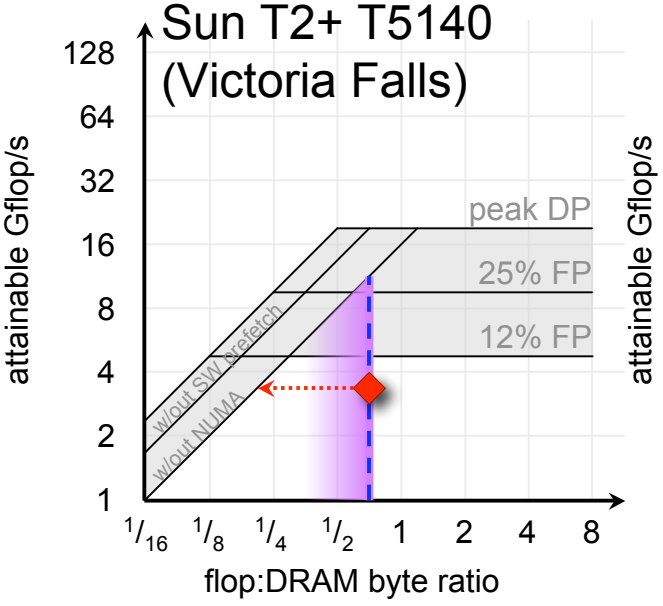
Roofline model for LBMHD

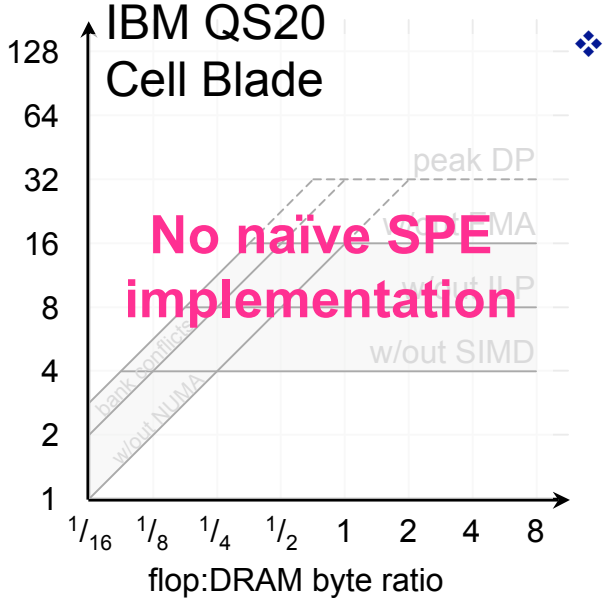
(out-of-the-box parallel performance)





- 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

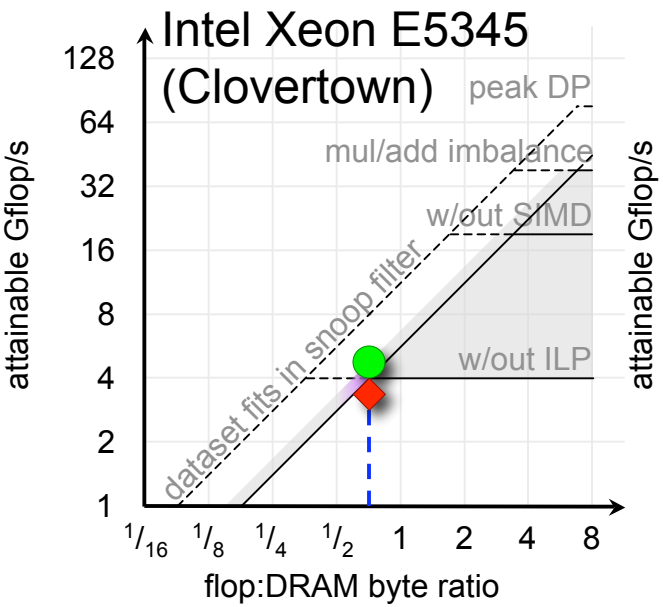


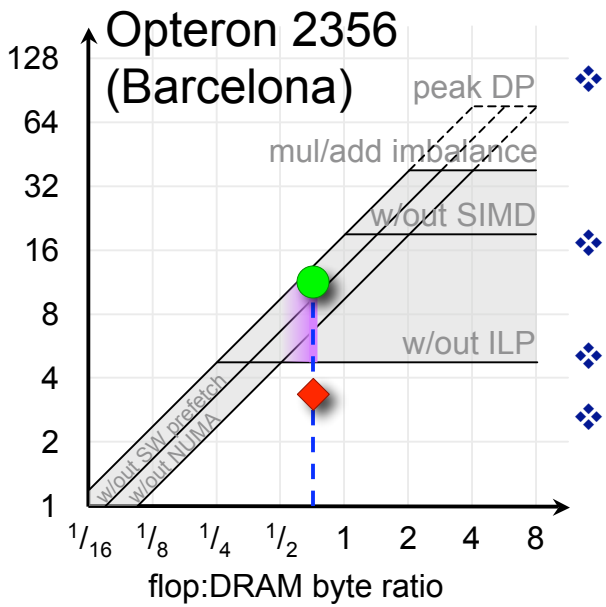


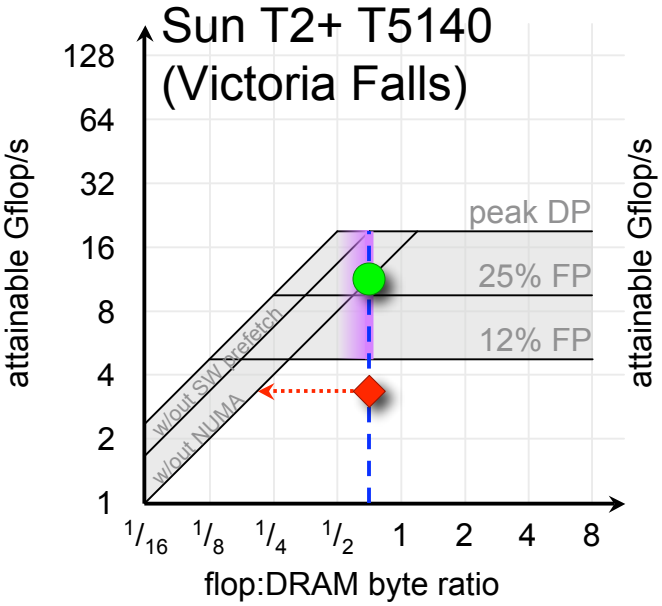
- Peak VF performance with 64 threads (out of 128) - high conflict misses

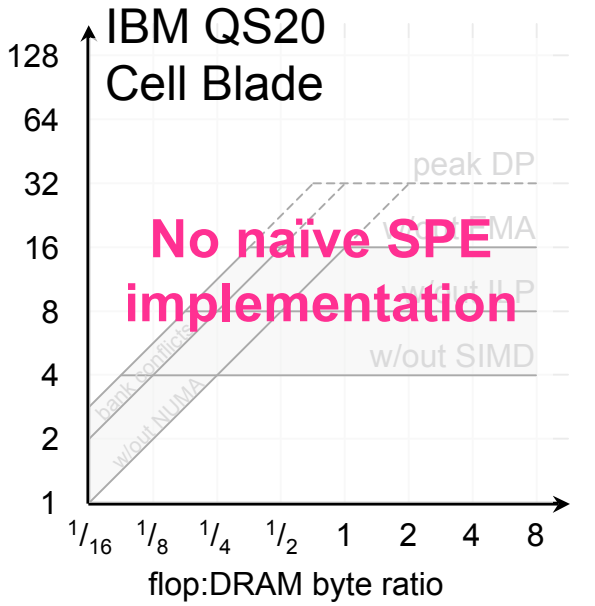
Roofline model for LBMHD

(Padding, Vectorization, Unrolling, Reordering, …)





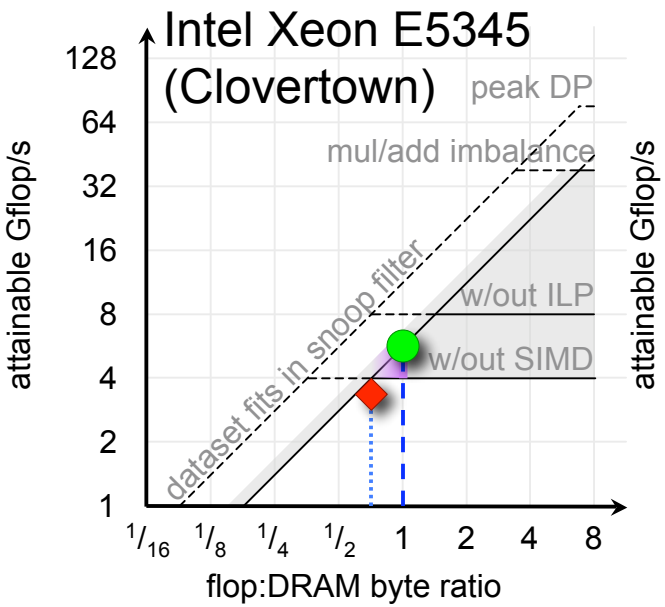


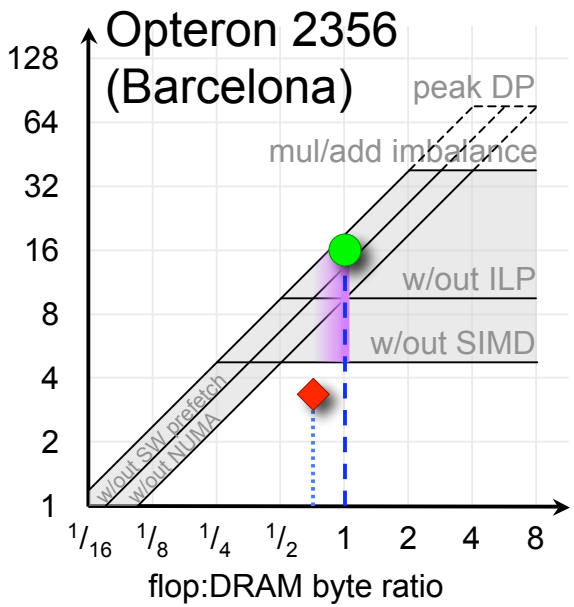


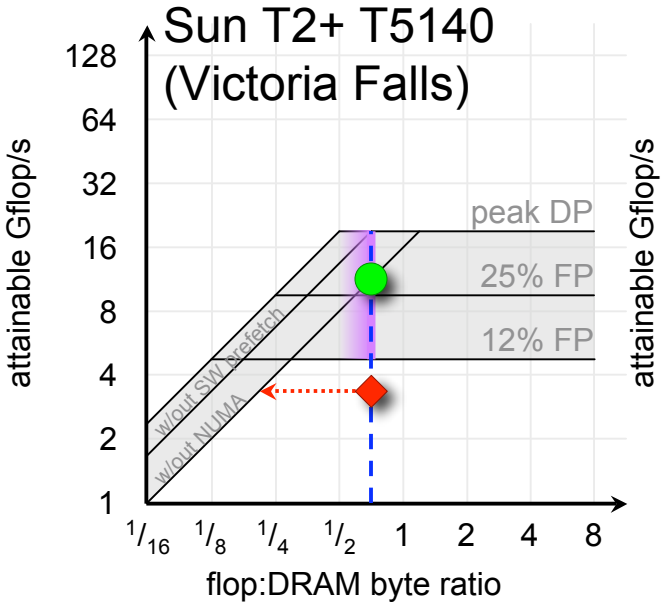
- Vectorize the code to eliminate TLB capacity misses
- Ensures unit stride access (bottom bandwidth ceiling)
- Tune for optimal VL
- Clovertown pinned to lower BW ceiling

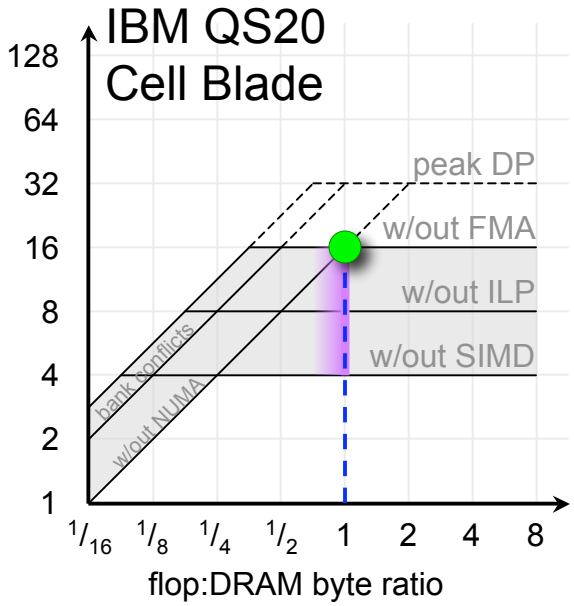
Roofline model for LBMHD

(SIMDization + cache bypass)





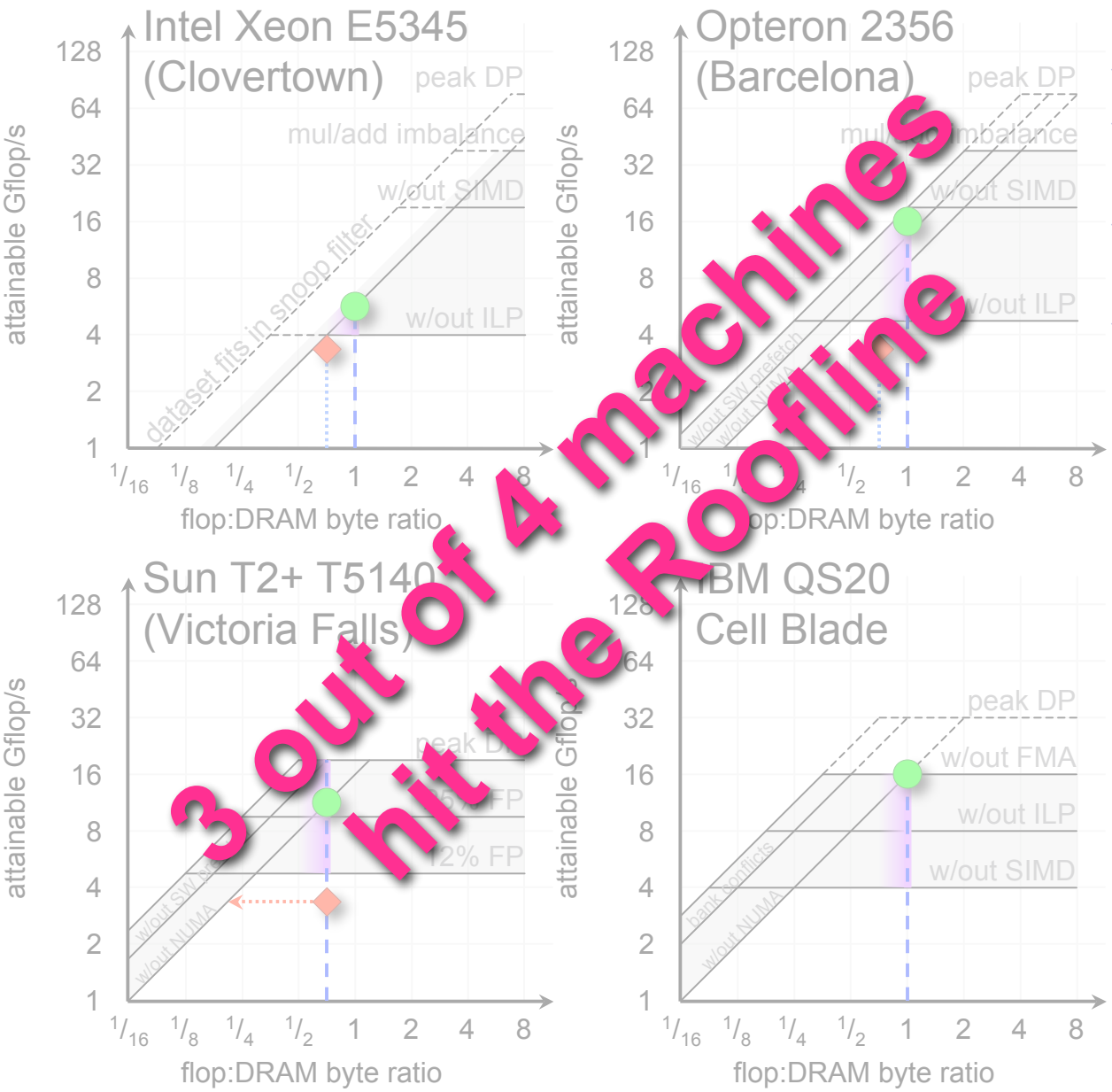




- Make SIMDization explicit
- Technically, this swaps ILP and SIMD ceilings
- Use cache bypass instruction: ***movntpd***
- Increases flop:byte ratio to ~1.0 on x86/Cell

Roofline model for LBMHD

(SIMDization + cache bypass)



3 out of 4 machines hit the Roofline

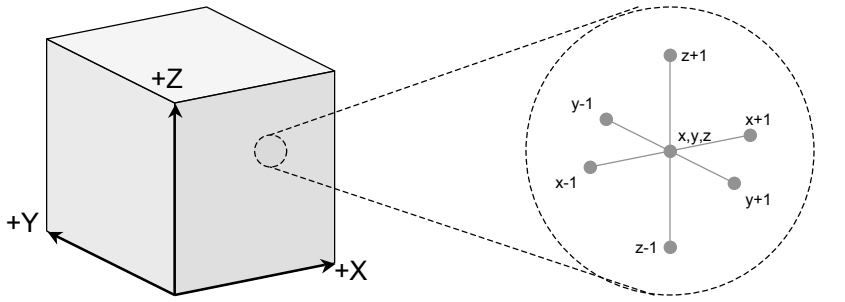
- Make SIMDization explicit
- Technically, this swaps ILP and SIMD ceilings
- Use cache bypass instruction: ***movntpd***
- Increases flop:byte ratio to ~1.0 on x86/Cell

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”, Supercomputing (SC) (to appear), 2008.

The Heat Equation Stencil

- Explicit Heat equation (Laplacian ~ $\nabla^2 F(x,y,z)$) on a regular grid
- Storage: 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
- Run one problem size: 256^3

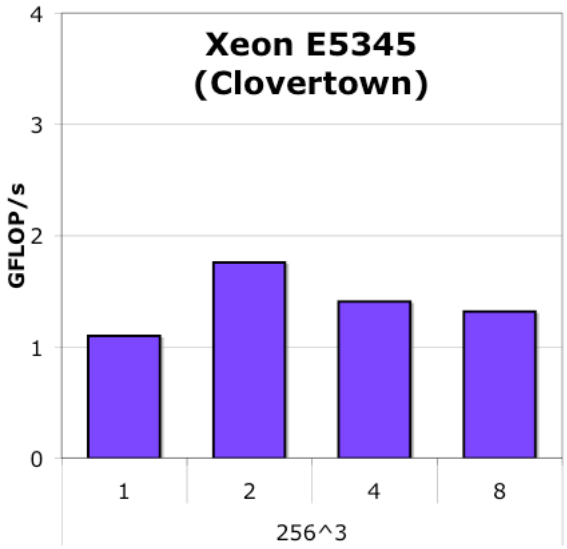


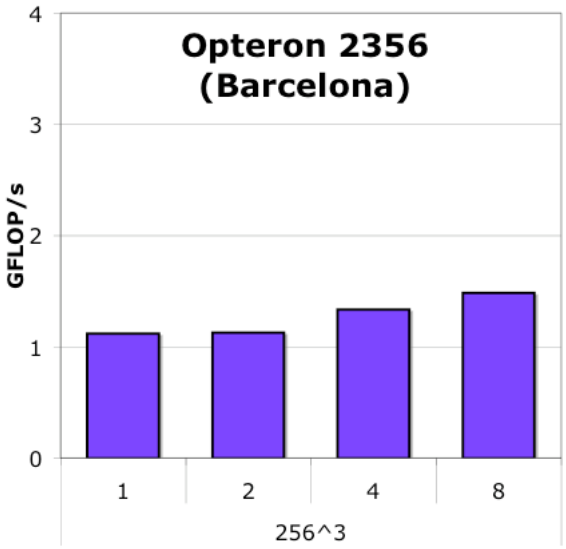
PDE grid

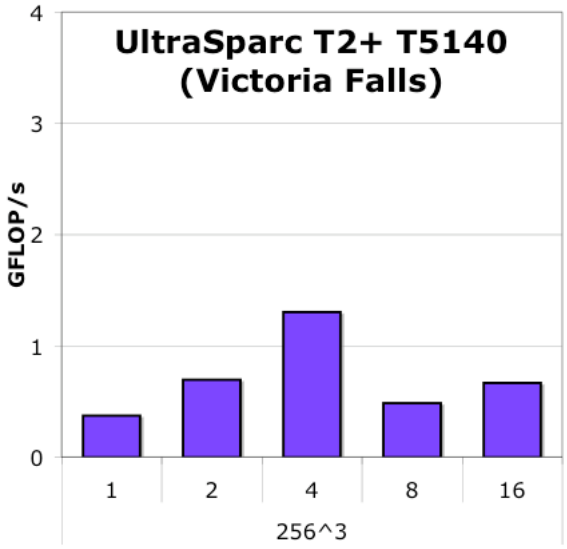
stencil for heat equation PDE

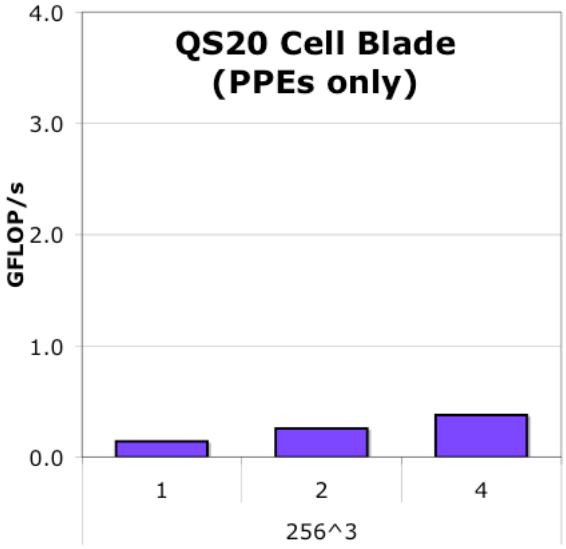
Stencil Performance

(out-of-the-box code)





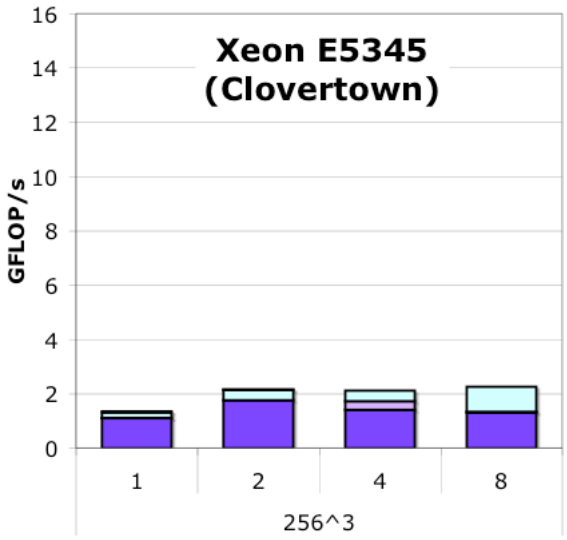


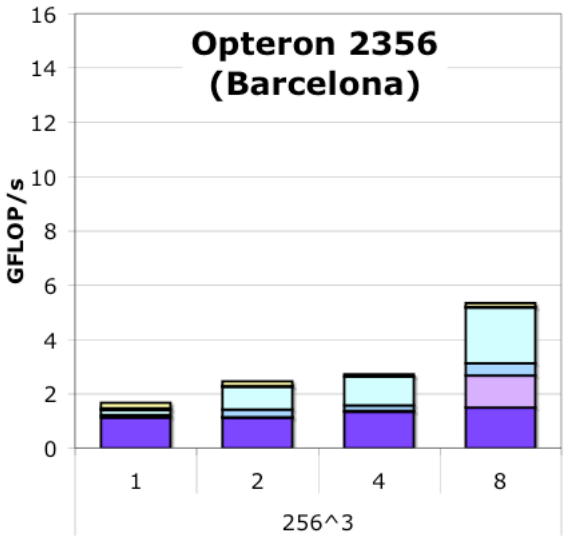


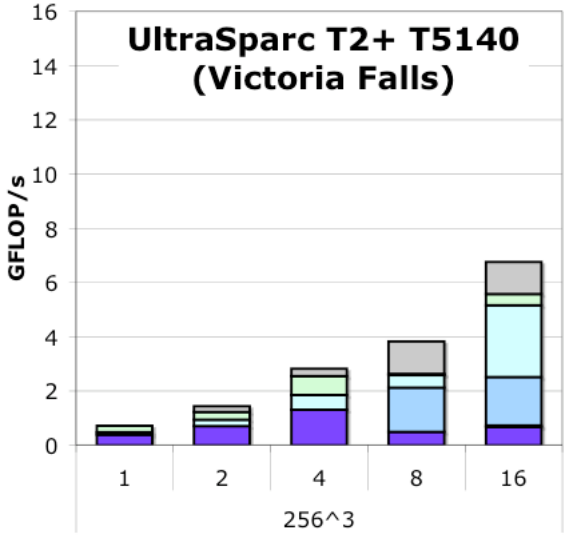
- Expect performance to be between SpMV and LBMHD
- **Scalability is universally poor**
- **Performance is poor**

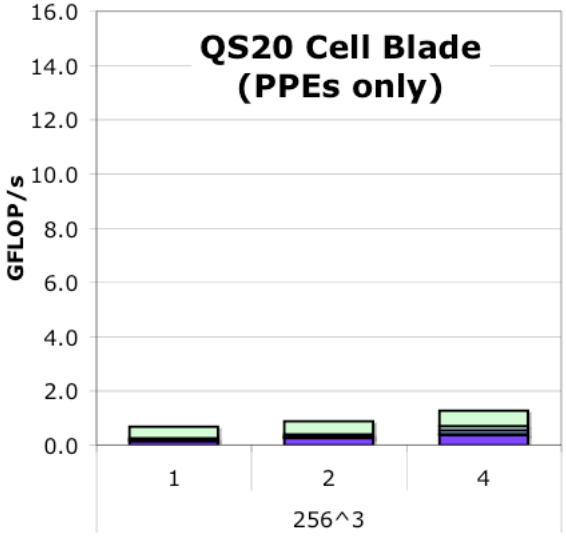
Naïve

Auto-tuned Stencil Performance (portable C)

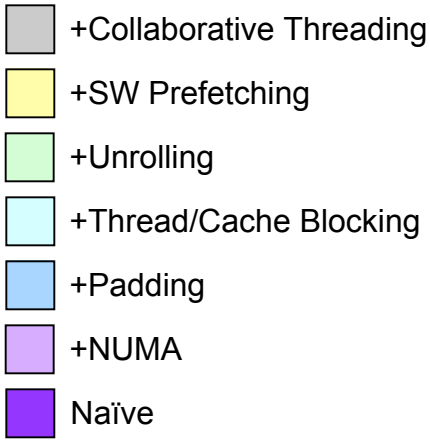






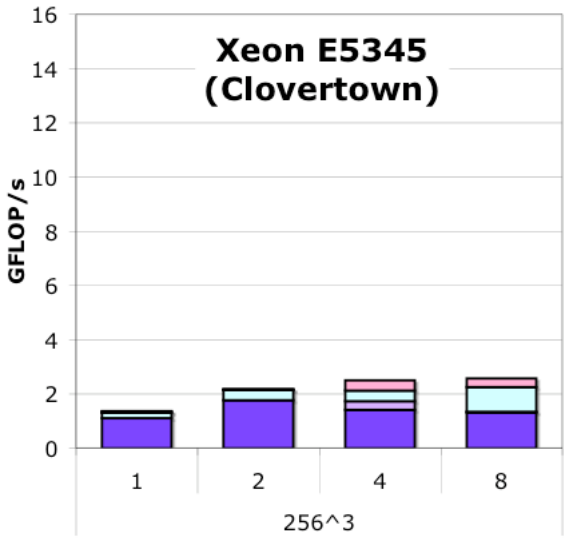


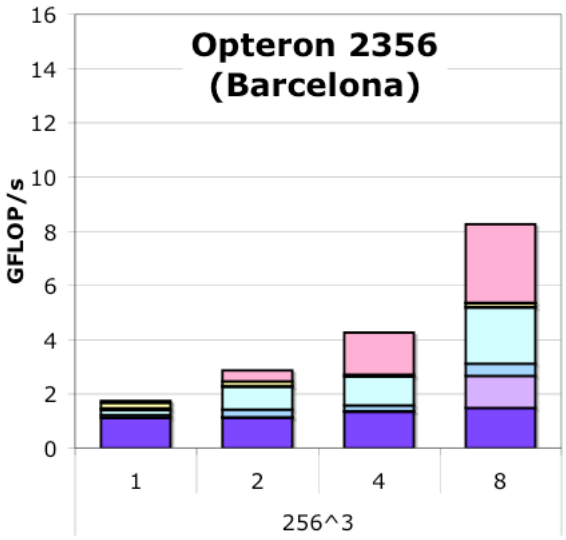
- Proper NUMA management is essential on most architectures
- Moreover proper cache blocking is still essential even with MB's of cache

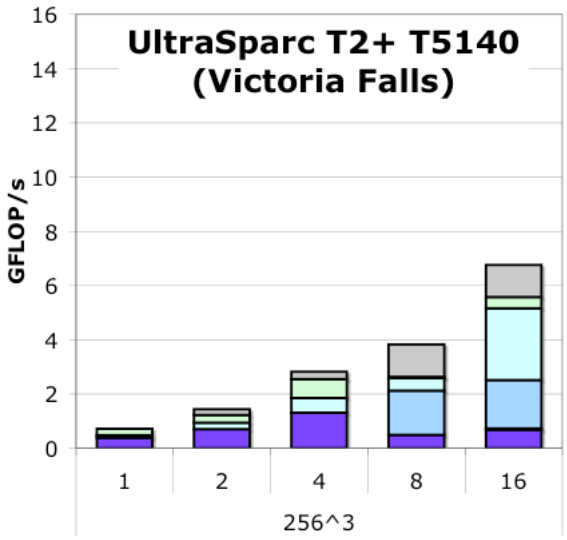


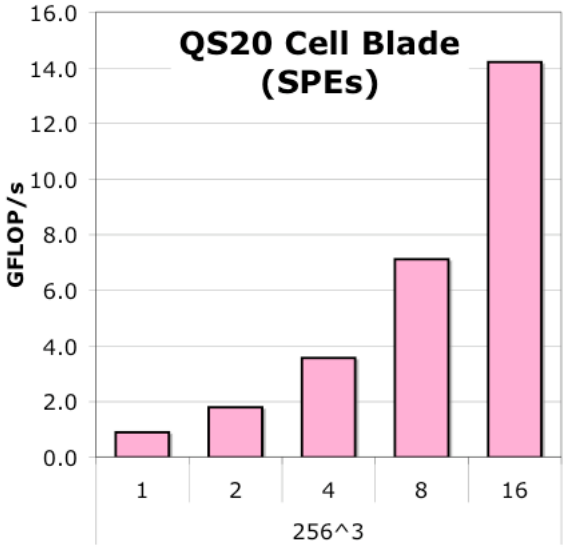
Auto-tuned Stencil Performance

(architecture specific optimizations)

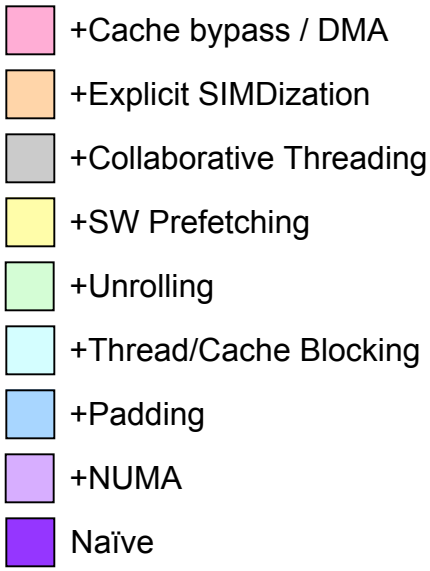






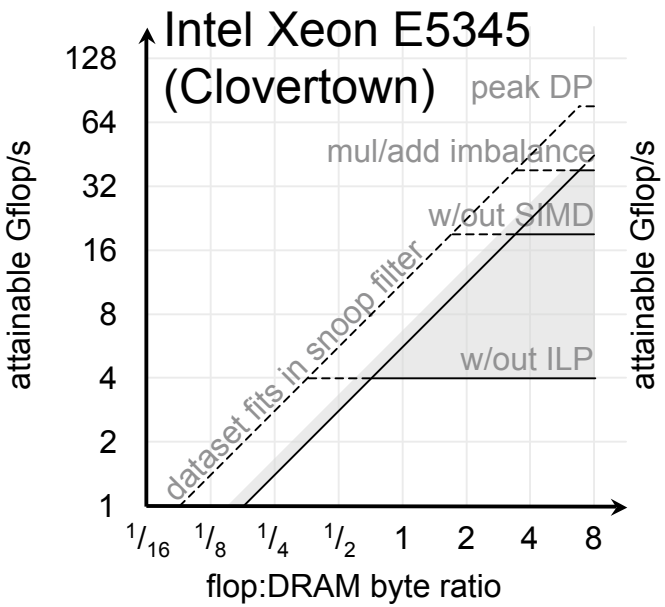


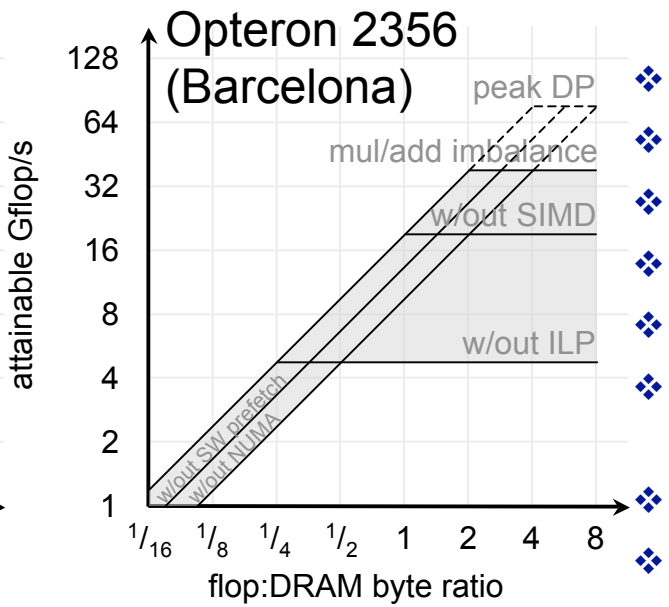
- Cache bypass can significantly improve Barcelona performance.
- DMA, SIMD, and cache blocking were essential on Cell

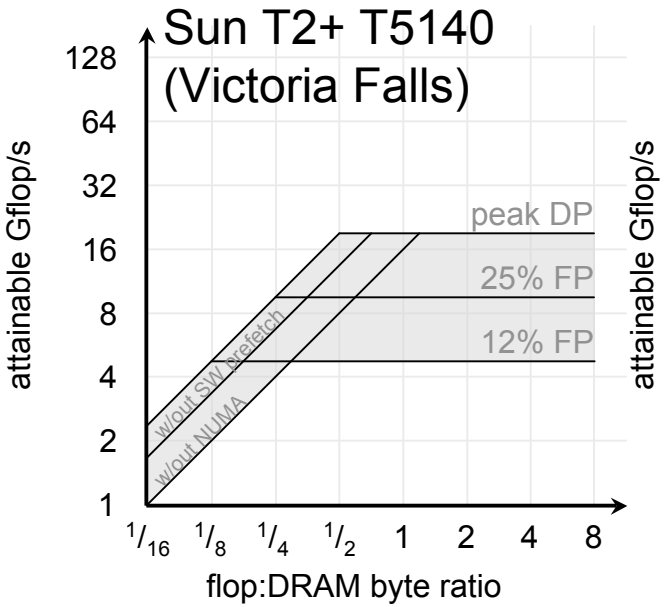


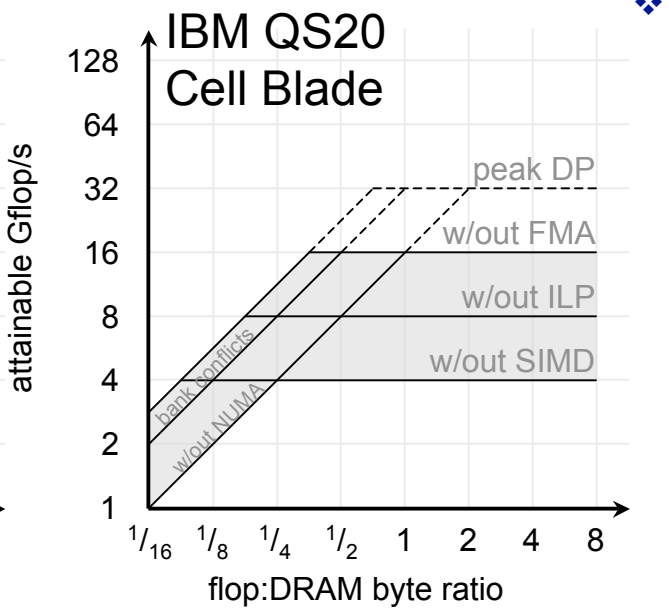
Roofline model for Stencil

(out-of-the-box code)





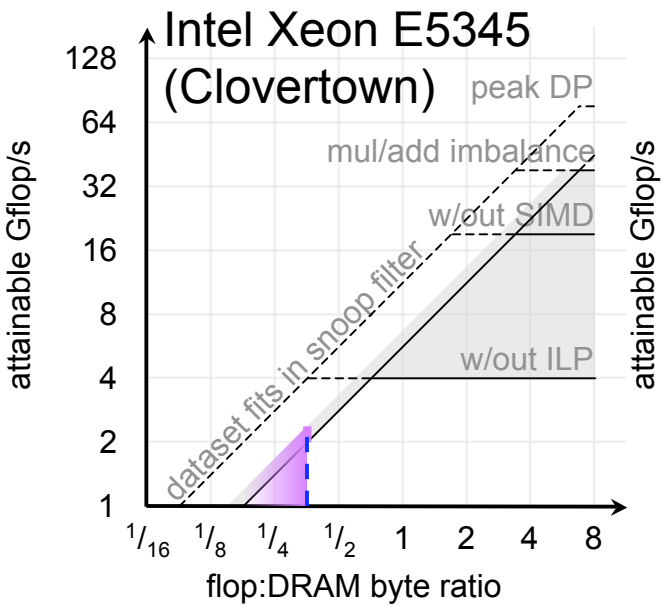


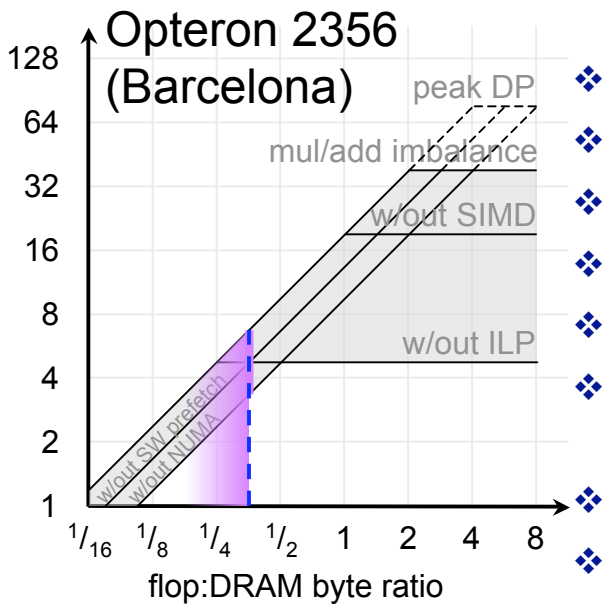


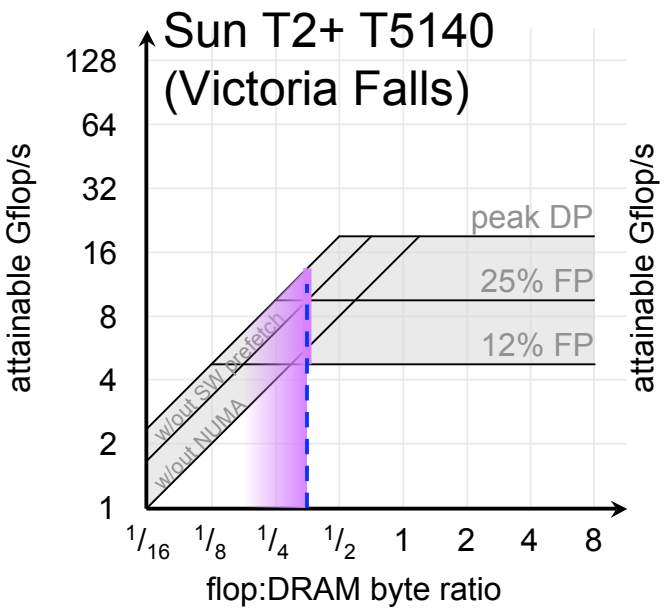
- Large datasets
- 2 unit stride streams
- No NUMA
- Little ILP
- No DLP
- Far more adds than multiplies (imbalance)
- Ideal flop:byte ratio $^1/_3$
- High locality requirements
- Capacity and conflict misses will severely impair flop:byte ratio

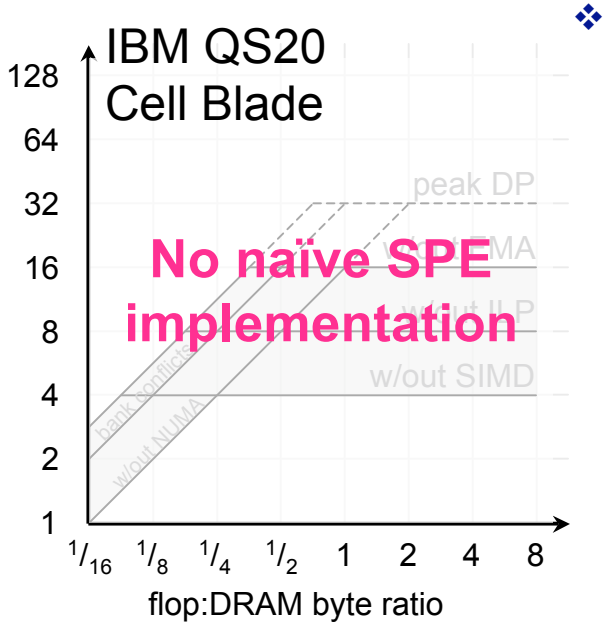
Roofline model for Stencil

(out-of-the-box code)





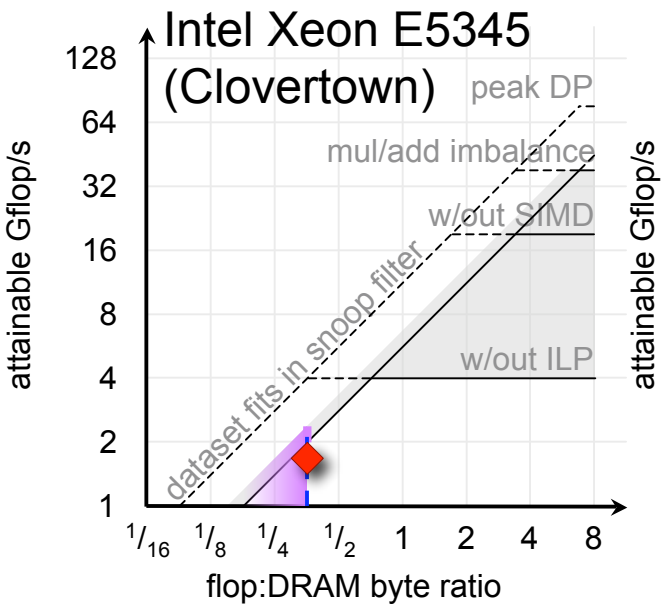


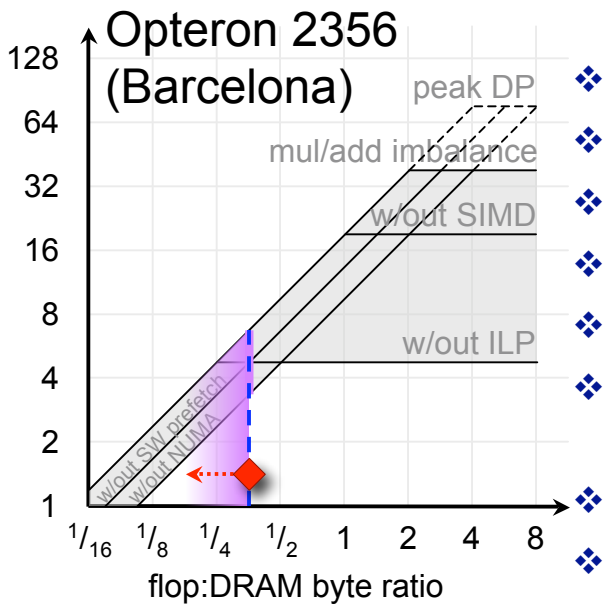


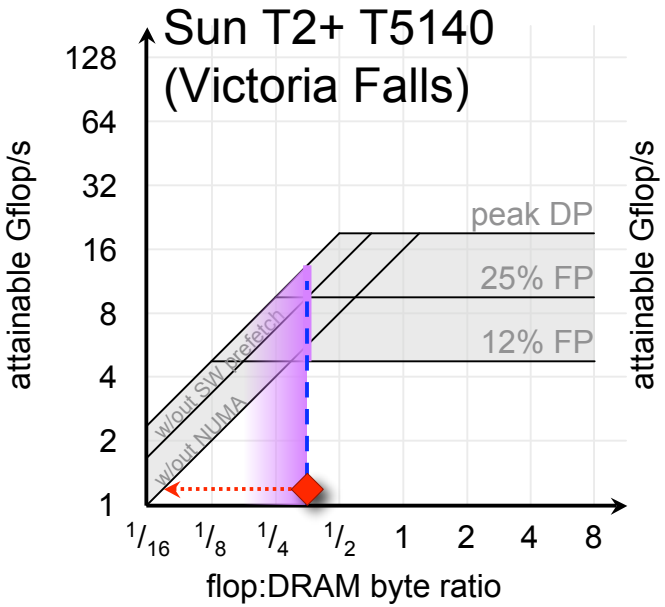
- Large datasets
- 2 unit stride streams
- No NUMA
- Little ILP
- No DLP
- Far more adds than multiplies (imbalance)
- Ideal flop:byte ratio $^1/_3$
- High locality requirements
- Capacity and conflict misses will severely impair flop:byte ratio

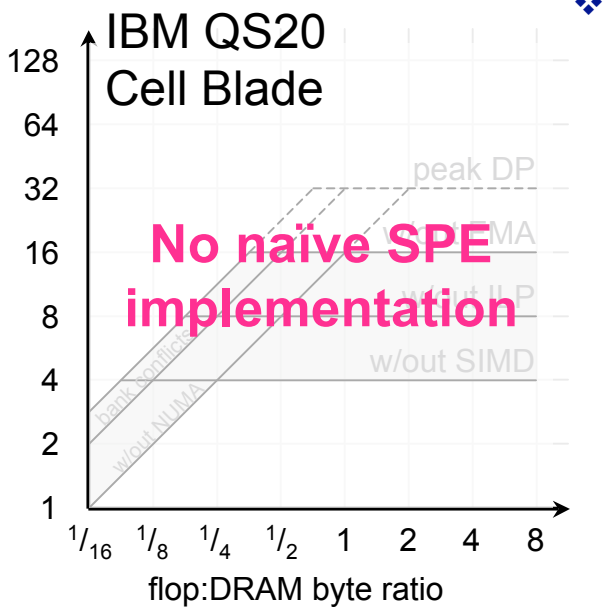
Roofline model for Stencil

(out-of-the-box code)





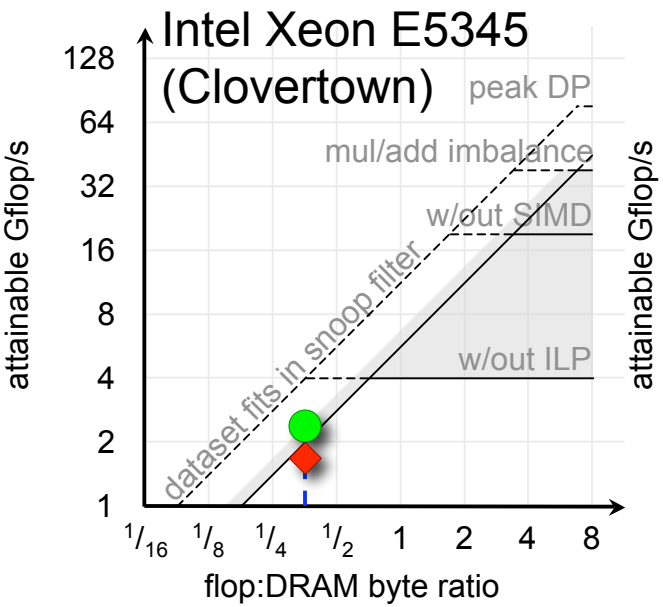


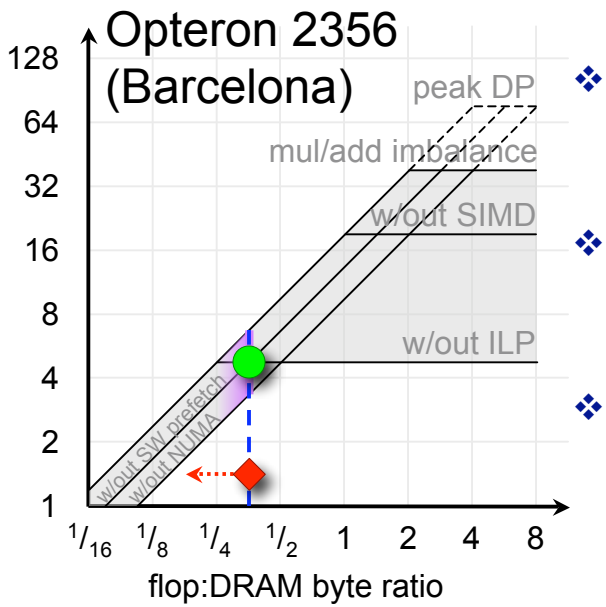


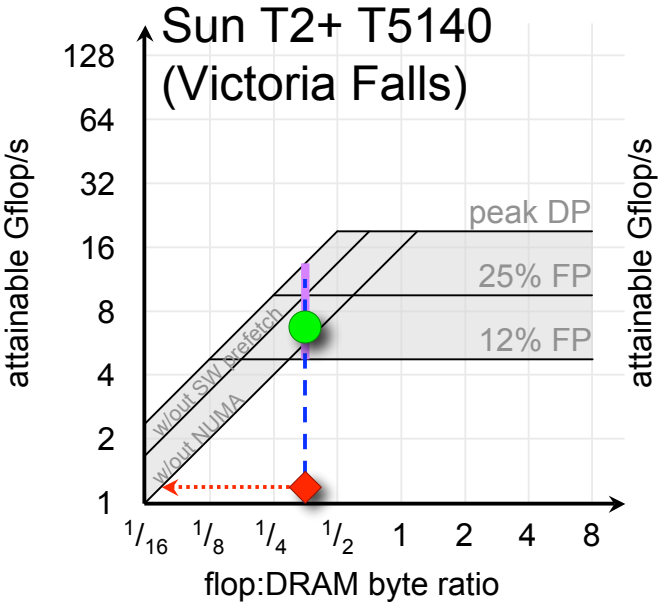
- Large datasets
- 2 unit stride streams
- No NUMA
- Little ILP
- No DLP
- Far more adds than multiplies (imbalance)
- Ideal flop:byte ratio $^1/_3$
- High locality requirements
- Capacity and conflict misses will severely impair flop:byte ratio

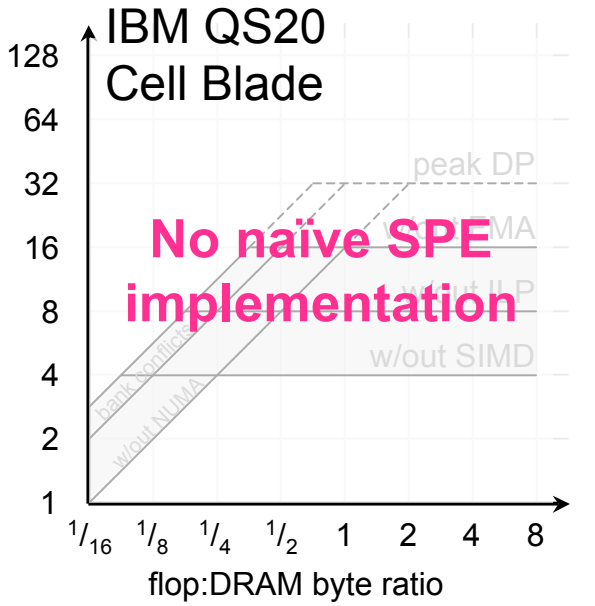
Roofline model for Stencil

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





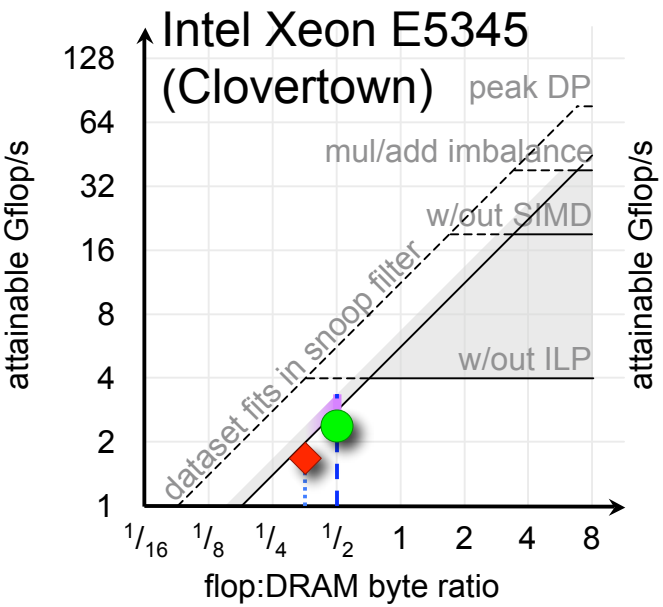


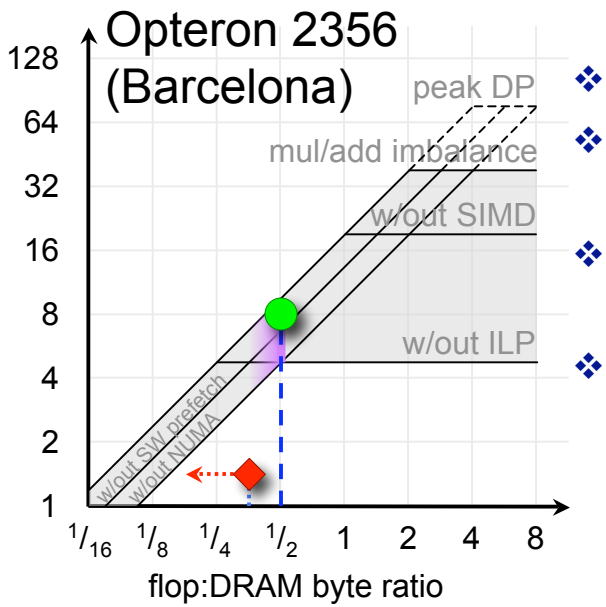


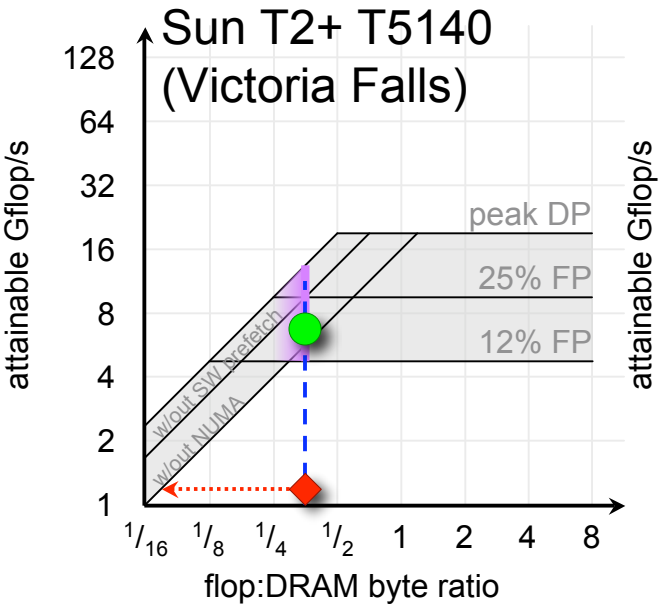
- Cache blocking helps ensure flop:byte ratio is as close as possible to $^1/_3$
- Clovertown has huge caches but is pinned to lower BW ceiling
- Cache management is essential when capacity/thread is low

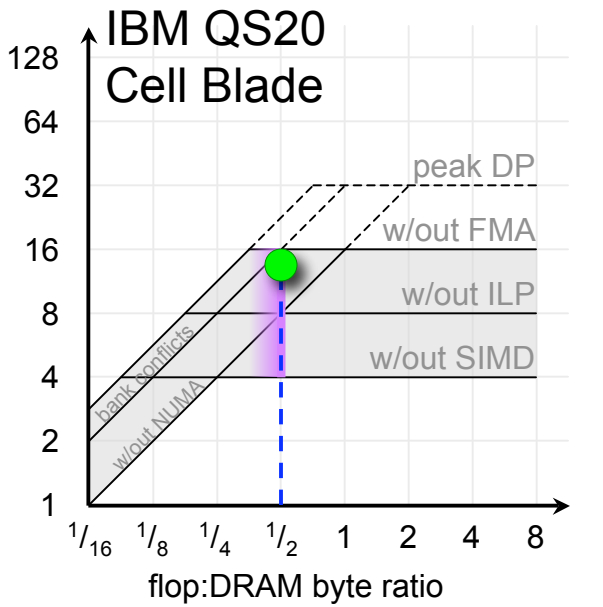
Roofline model for Stencil

(SIMDization + cache bypass)





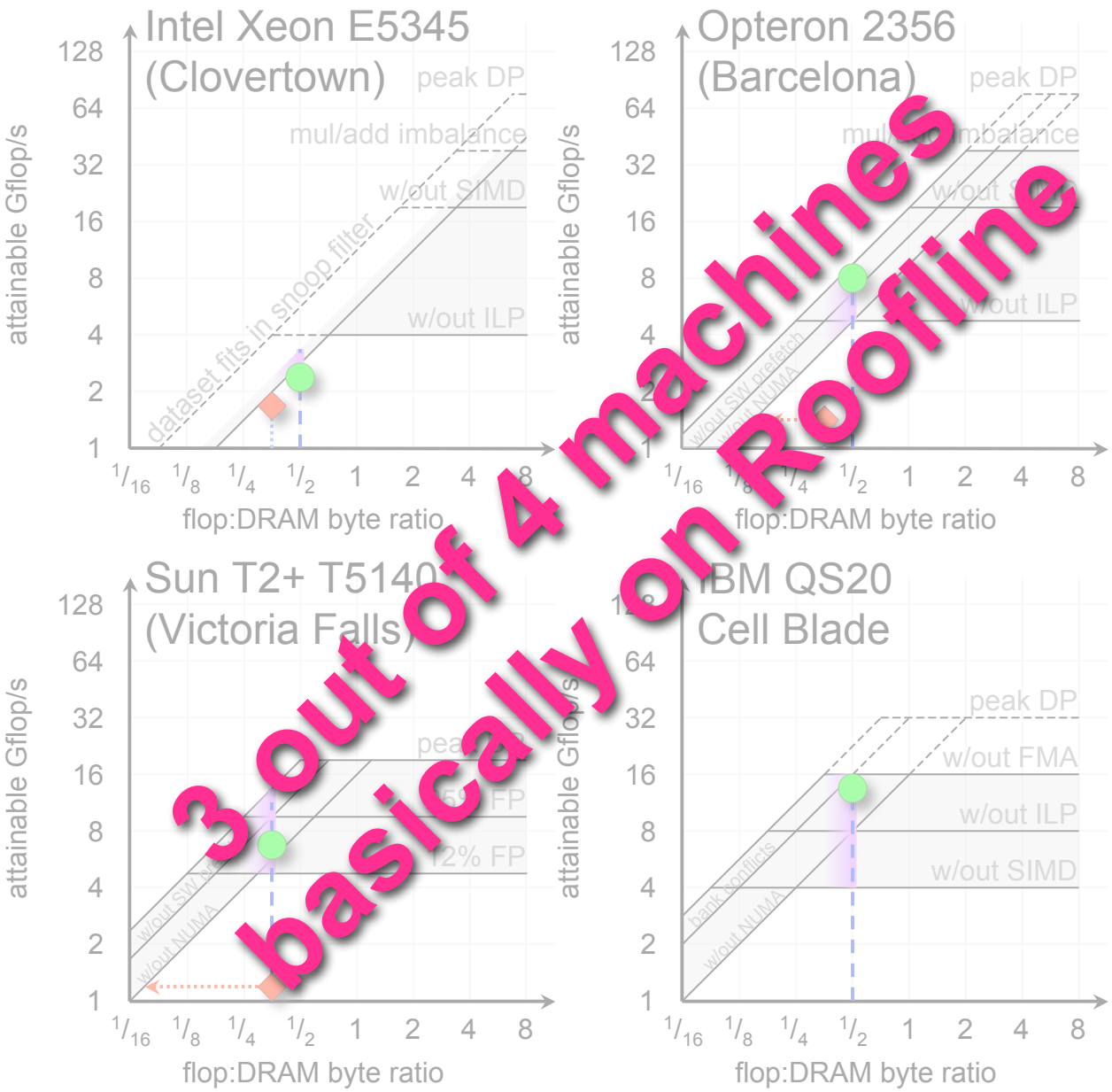




- Make SIMDization explicit
- Technically, this swaps ILP and SIMD ceilings
- Use cache bypass instruction: ***movntpd***
- Increases flop:byte ratio to ~0.5 on x86/Cell

Roofline model for Stencil

(SIMDization + cache bypass)



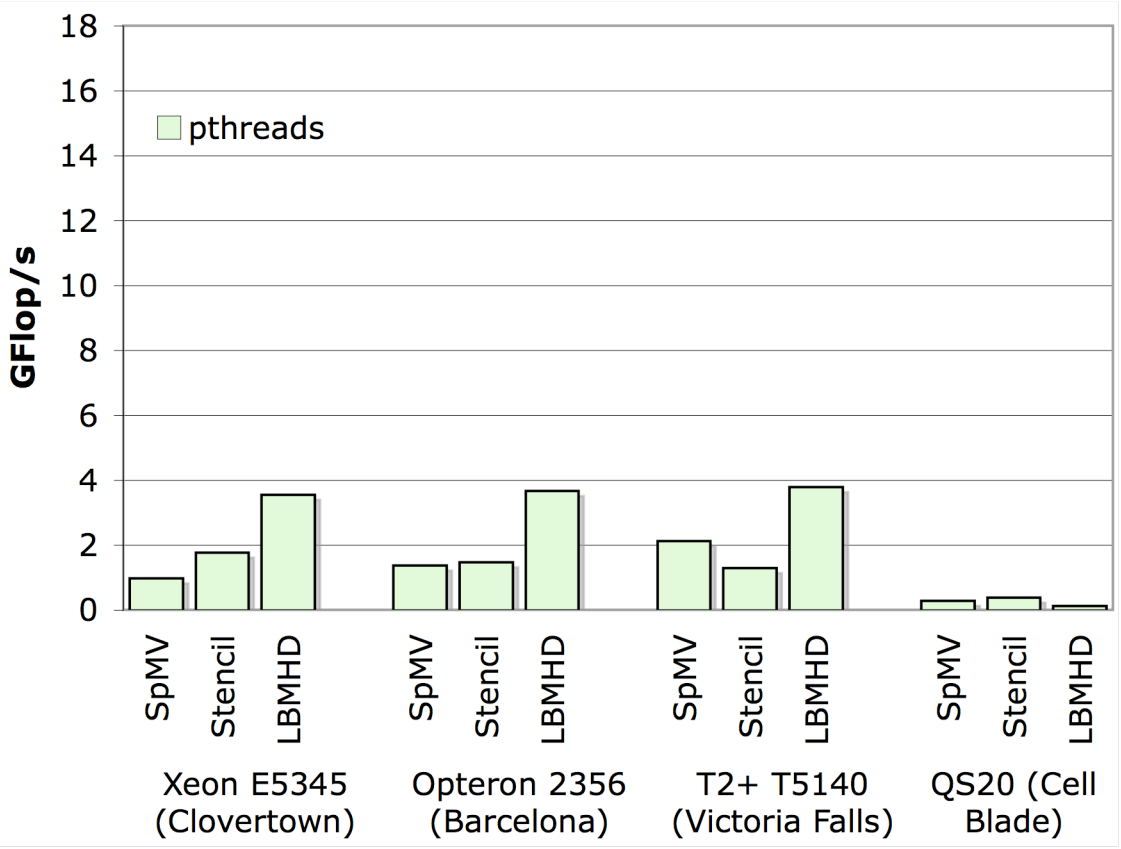
3 out of 4 machines basically on Roofline

- 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

Summary

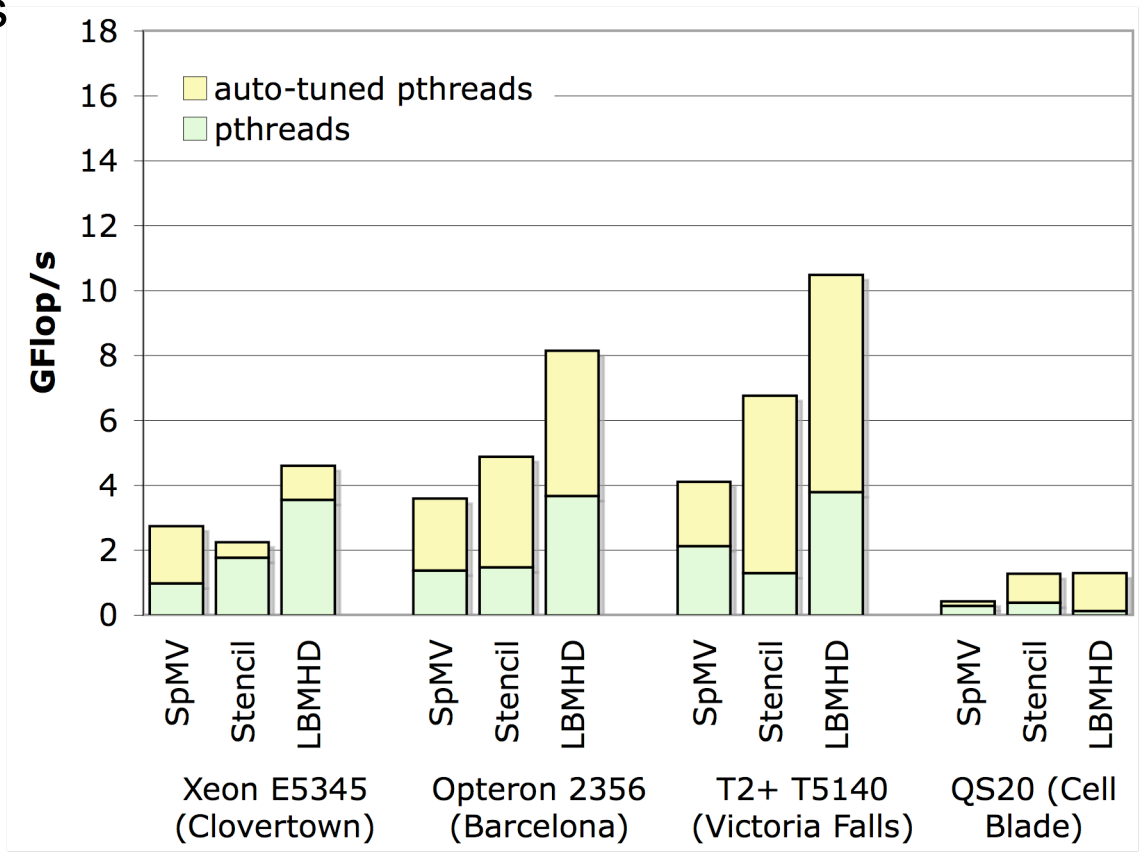
Out-of-the-box Performance

- **Ideal productivity (just type 'make')**
- Kernels sorted by arithmetic intensity
- Maximum performance with any concurrency
- Note: Cell = 4 PPE threads (no SPEs)
- **Surprisingly, most architectures got similar performance**



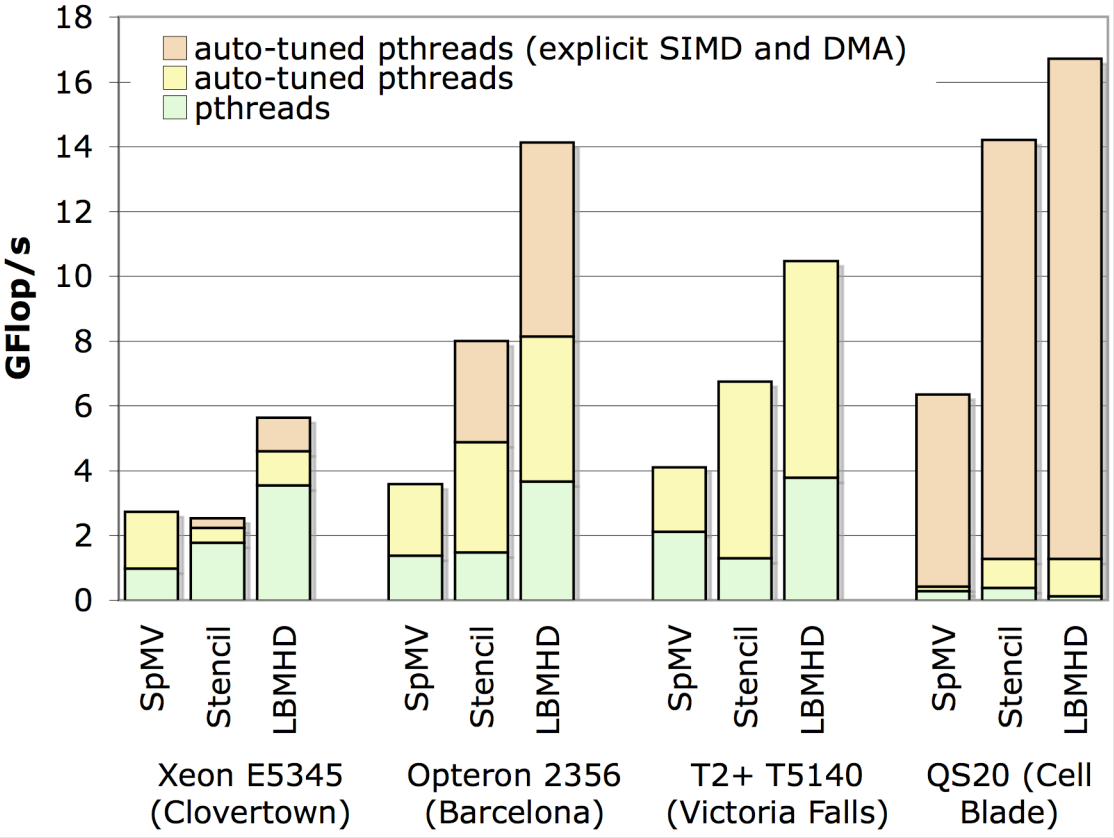
Portable Performance

- Portable (C only) auto-tuning
- **Sacrifice some productivity upfront, amortize by reuse**
- Dramatic increases in performance on Barcelona and Victoria Falls
- Clovertown is increasingly memory bound
- Cell = 4 PPE threads (no SPEs)



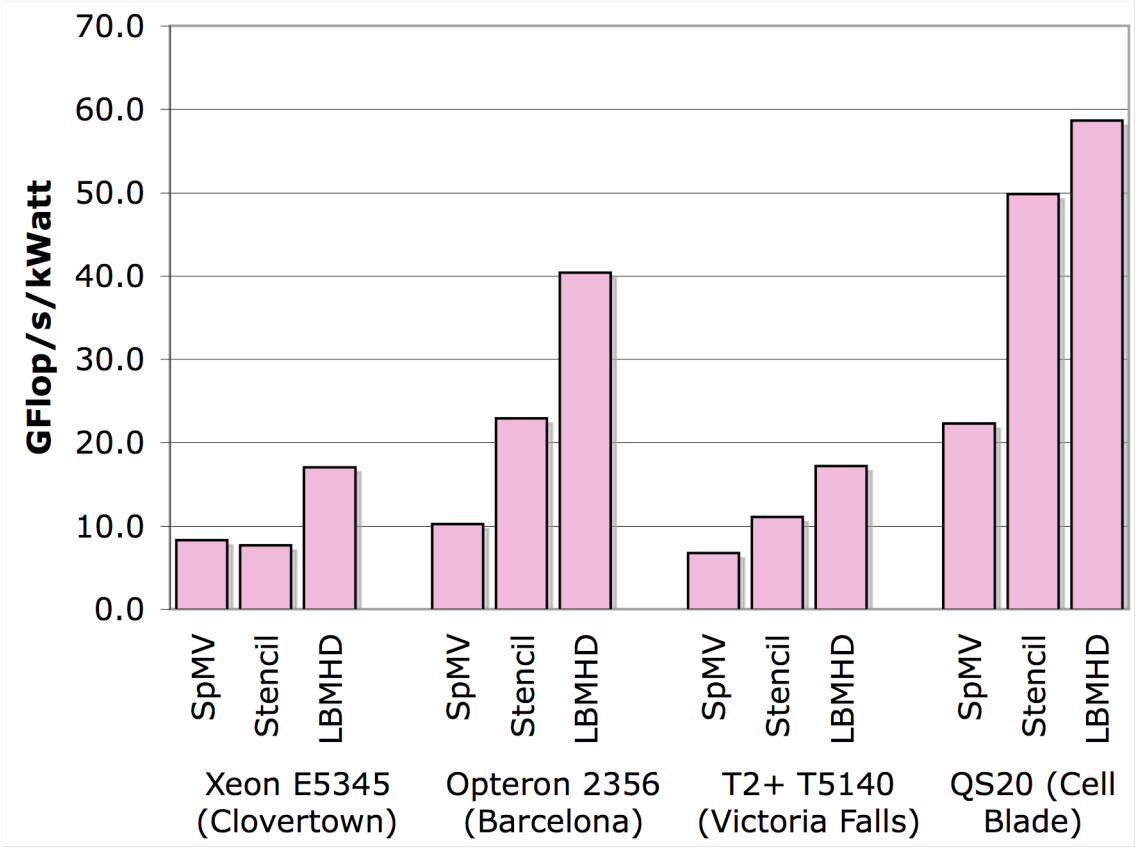
Architecture Specific Performance

- **ISA specific auto-tuning (reduced portability & productivity)**
- explicit:
  - SPE parallelization with DMAs
  - SIMDization(SSE+Cell)
  - cache bypass
- Cell gets all its performance from the SPEs (DP limits perf)
- Barcelona gets half its performance from SIMDization



Power Efficiency

- Used a digital power meter to measure sustained power under load
- **Efficiency = Sustained Performance / Sustained Power**
- Victoria Falls' power efficiency is severely limited by FBDIMM power
- Despite Cell's weak double precision, it delivers the highest power efficiency



Summary
(auto-tuning)

- Auto-tuning provides portable performance
- Auto-tuning with architecture (ISA) specific optimizations are essential on some machines

- The roofline model qualifies performance
  - Also tells us which optimizations are important
  - As well as how much further improvement is possible

Summary
(architectures)

- **Clovertown**
  - severely bandwidth limited
- **Barcelona**
  - delivers good performance with architecture specific auto-tuning
  - bandwidth limited after ISA optimizations
- **Victoria Falls**
  - challenged by interaction of TLP with shared caches
  - often limited by in-core performance
- **Cell Broadband Engine**
  - performance entirely from architecture specific auto-tuning
  - bandwidth limited on SpMV
  - DP FP limited on Stencil(slightly) and LBMHD(severely)
  - delivers the best power efficiency

Acknowledgements

- 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.