# A Vision for Integrating Performance Counters into the Roofline model

Samuel Williams[1,2]

*samw@cs.berkeley.edu*

with Andrew Waterman[1], Heidi Pan[1,3],
David Patterson[1], Krste Asanovic[1], Jim Demmel[1]

[1]University of California, Berkeley
[2]Lawrence Berkeley National Laboratory
[3]Massachusetts Institute of Technology

# Outline

- Auto-tuning
  - Introduction to Auto-tuning
  - BeBOP’s previous performance counter experience
  - BeBOP’s current tuning efforts

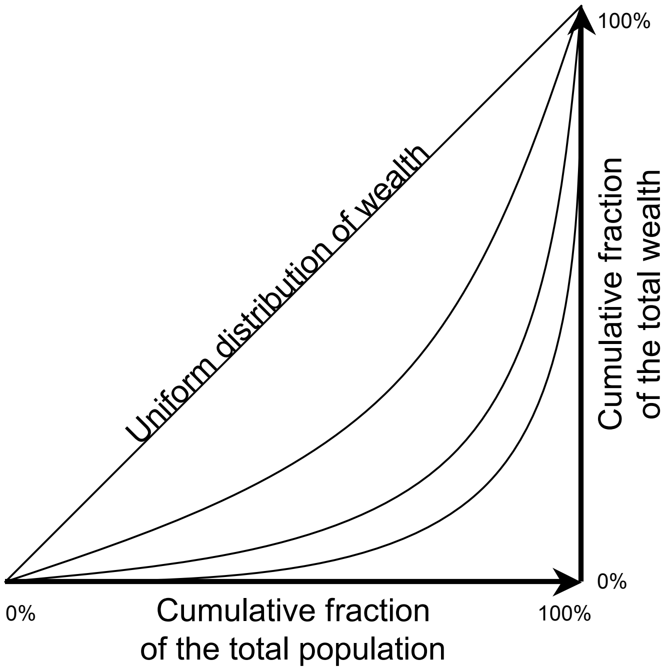
- Roofline Model
  - Motivating Example - SpMV
  - Roofline model
  - Performance counter enhanced Roofline model

# Motivation
# (Folded into Jim's Talk)

# Gini Coefficient

- In economics, the Gini coefficient is a measure of the distribution of wealth within a society
- As wealth becomes concentrated, the value of the coefficient increases, and the curve departs from a straight line.

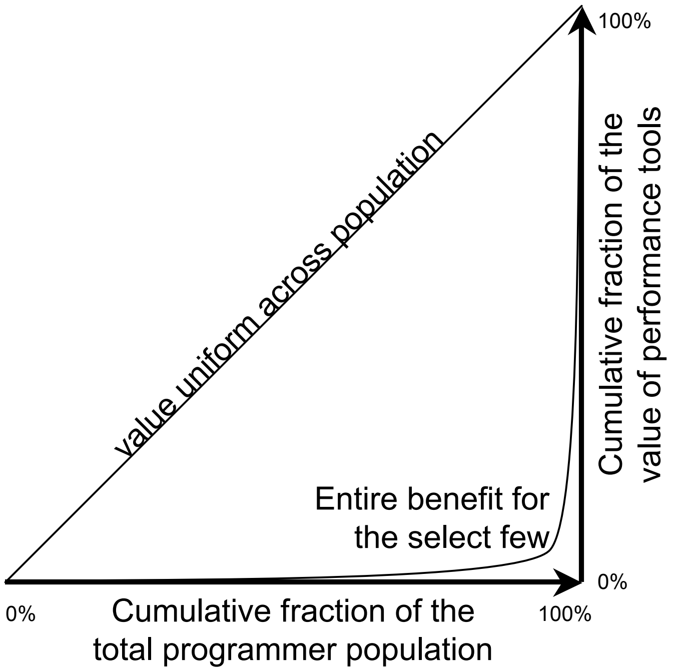
*it's a just an assessment of the distribution, not a commentary on what it should be*



http://en.wikipedia.org/wiki/Gini_coefficient

# What's the Gini Coefficient for our Society?

- By ***our society***, I mean those working in the performance optimization and analysis world (tuners, profilers, counters)
- Our ***wealth***, is knowledge of tools and benefit gained from them.

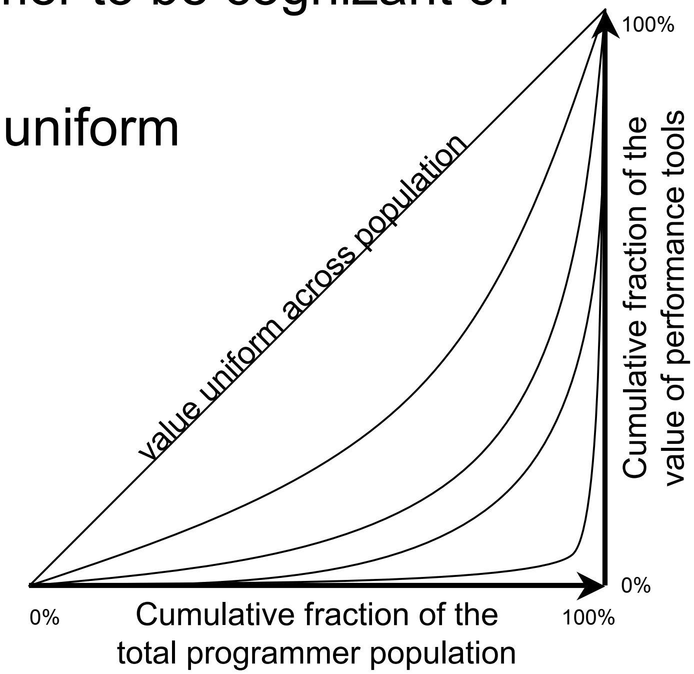


# Why is it so low?

- Apathy
  - Performance only matters after correctness
  - Scalability has won out over efficiency
  - Timescale for Moore's law has been shorter than optimization
- Ignorance / Lack of Specialized Education
  - Tools assume broad and deep architectural knowledge
  - Optimization may require detailed application knowledge
- Significant SysAdmin support
- Cryptic tools/presentation
- Erroneous data
- Frustration

# To what value should we aspire?

- Certainly unreasonable for every programmer to be cognizant of performance counters
- Equally unreasonable for the benefit to be uniform



- Making performance tools
  - more intuitive
  - more robust
  - easier to use (always on?)
  - essential in a multicore era

  will motivate more users to exploit them

- Oblivious to programmers, compilers, architectures, and middleware may exploit performance counters to improve performance

# I

## auto-tuning & performance counter experience

# Introduction to Auto-tuning

# Out-of-the-box Code Problem

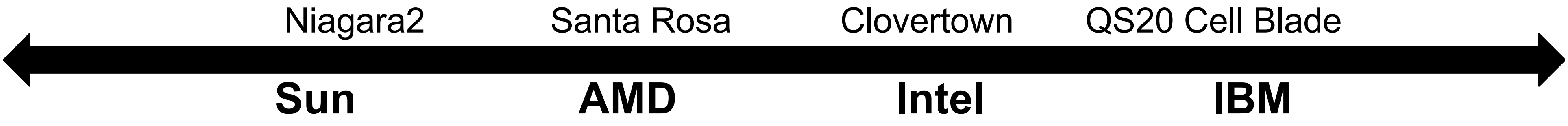
- Out-of-the-box code has (unintentional) assumptions on:
  - cache sizes (>10MB)
  - functional unit latencies(~1 cycle)
  - etc…
- These assumptions may result in poor performance when they exceed the machine characteristics

# Auto-tuning?

- Trade up front loss in productivity cost for continued reuse of automated kernel optimization on other architectures

- Given existing 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)

- Auto-tuners that generate C code provide **performance portability** across the existing breadth of architectures
- Can be extended with ISA specific optimizations (e.g. DMA, SIMD)



*(Breadth of Existing Architectures)*

# BeBOP's Previous Performance Counter Experience (2-5 years ago)

# Performance Counter Usage

- Perennially, performance counters have been used
  - as a *post-mortem* to validate auto-tuning heuristics
  - to bound remaining performance improvement
  - to understand unexpectedly poor performance

- However, this requires:
  - significant kernel and architecture knowledge
  - creation of a performance model specific to each kernel
  - calibration of the model

- Summary: We've experienced a progressively lower benefit and confidence in their use due to the variation in the quality and documentation of performance counter implementations

# Experience (1)

- Sparse Matrix Vector Multiplication (SpMV)
  - *“Performance Optimizations and Bounds for Sparse Matrix-Vector Multiply”*
    - Applied to older Sparc, Pentium III, Itanium machines
    - Model cache misses (compulsory matrix or compulsory matrix+vector)
    - Count cache misses via PAPI
    - Generally well bounded (but large performance bound)
  - *“When Cache Blocking Sparse Matrix Vector Multiply Works and Why”*
    - Similar architectures
    - Adds a fully associative TLB model (benchmarked TLB miss penalty)
    - Count TLB misses (as well as cache misses)
    - Much better correlation to actual performance trends

- Only modeled and counted the total number of misses (bandwidth only).
- Performance counters didn’t distinguish between ‘slow’ and ‘fast’ misses (i.e. didn’t account for exposed memory latency)

# Experience (2)

- MSPc/SIREV papers
  - Stencils (heat equation on a regular grid)
  - Used newer architectures (Opteron, Power5, Itanium2)
  - Attempted to model slow and fast misses (e.g. engaged prefetchers)
  - Modeling generally bounds performance and notes the trends

  - Attempted use performance counters to understand the quirks
  - Opteron and Power5 performance counters didn’t count prefetched data
  - Itanium performance counter trends correlated well with performance

# BeBOP’s Current Tuning Efforts (last 2 years)

# BeBOP’s Current Tuning Efforts

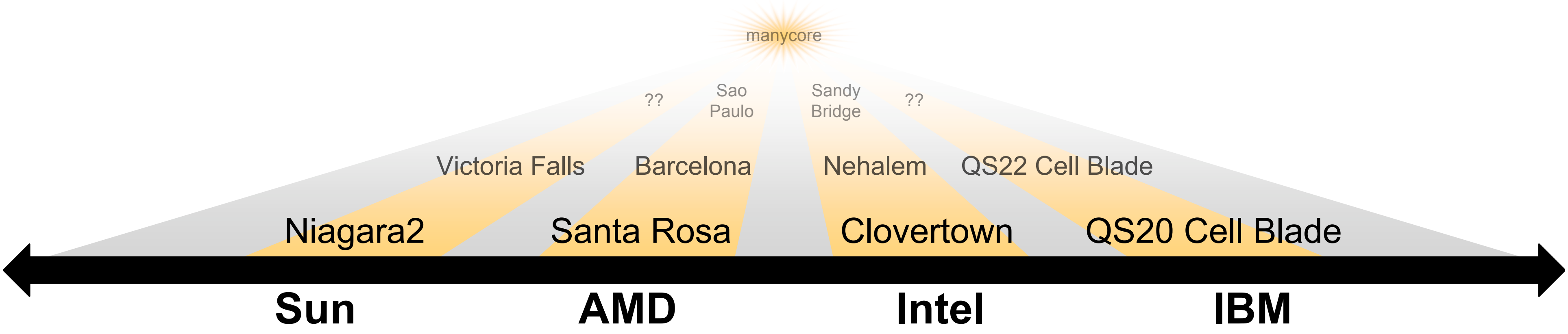
- Multicore (and distributed) oriented

- Throughput Oriented Kernels on Multicore architectures:
  - Dense Linear Algebra (LU, QR, Cholesky, …)
  - Sparse Linear Algebra (SpMV, Iterative Solvers, …)
  - Structured Grids (LBMHD, stencils, …)
  - FFTs
  - SW/HW co-tuning
  - Collectives (e.g. block transfers)

- Latency Oriented Kernels:
  - Collectives (Barriers, scalar transfers)

# (re)design for evolution

- Design auto-tuners for an arbitrary number of threads
- Design auto-tuners to address the limitations of the multicore paradigm
- This will provide **performance portability** across both the existing breadth of multicore architectures as well as their evolution



*(Breadth of Existing Architectures)*

# II

# Roofline Model:

## Facilitating Program Analysis and Optimization

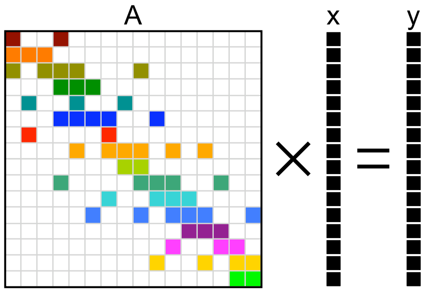
# Motivating Example:

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

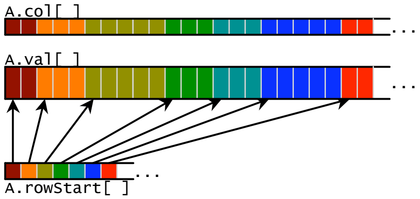
Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, James Demmel, "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms", Supercomputing (SC), 2007.

# 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



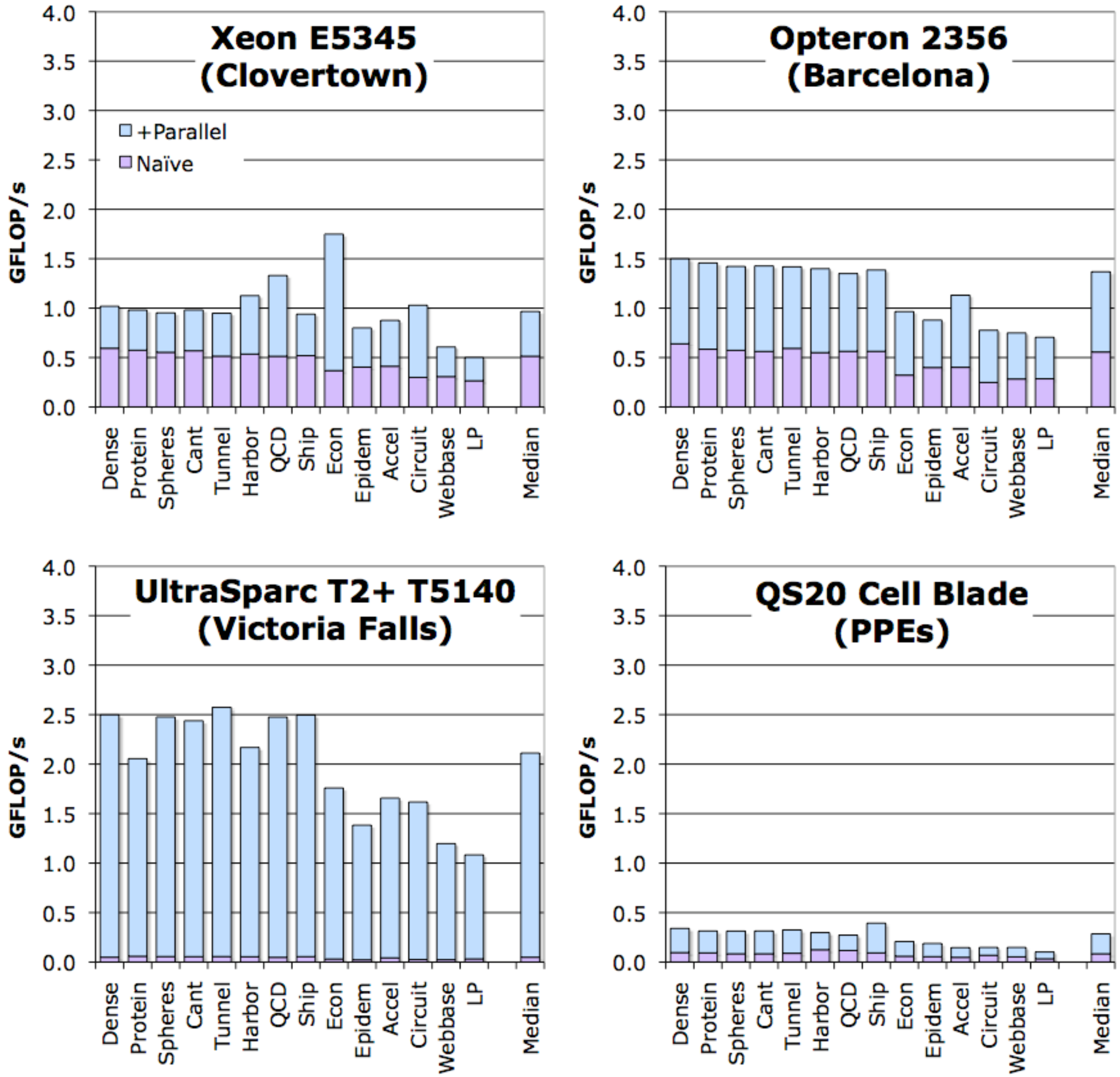
**(b)**
CSR data structure

```
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

# SpMV Performance

## (simple parallelization)



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

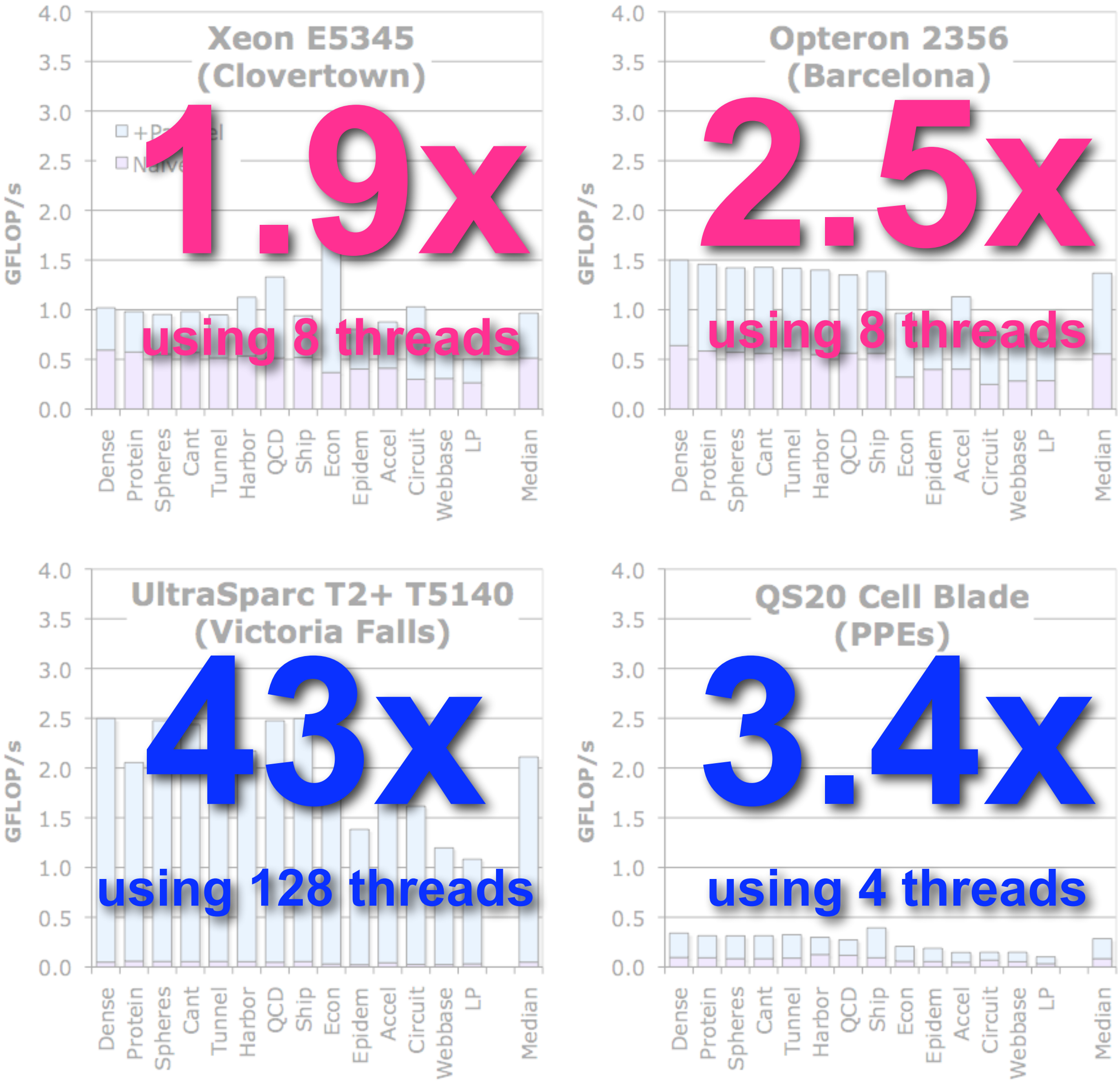
Naïve Pthreads

Naïve

EECS
Electrical Engineering and Computer Sciences

# SpMV Performance
## (simple parallelization)

BERKELEY PAR LAB



1.9x using 8 threads

2.5x using 8 threads

43x using 128 threads

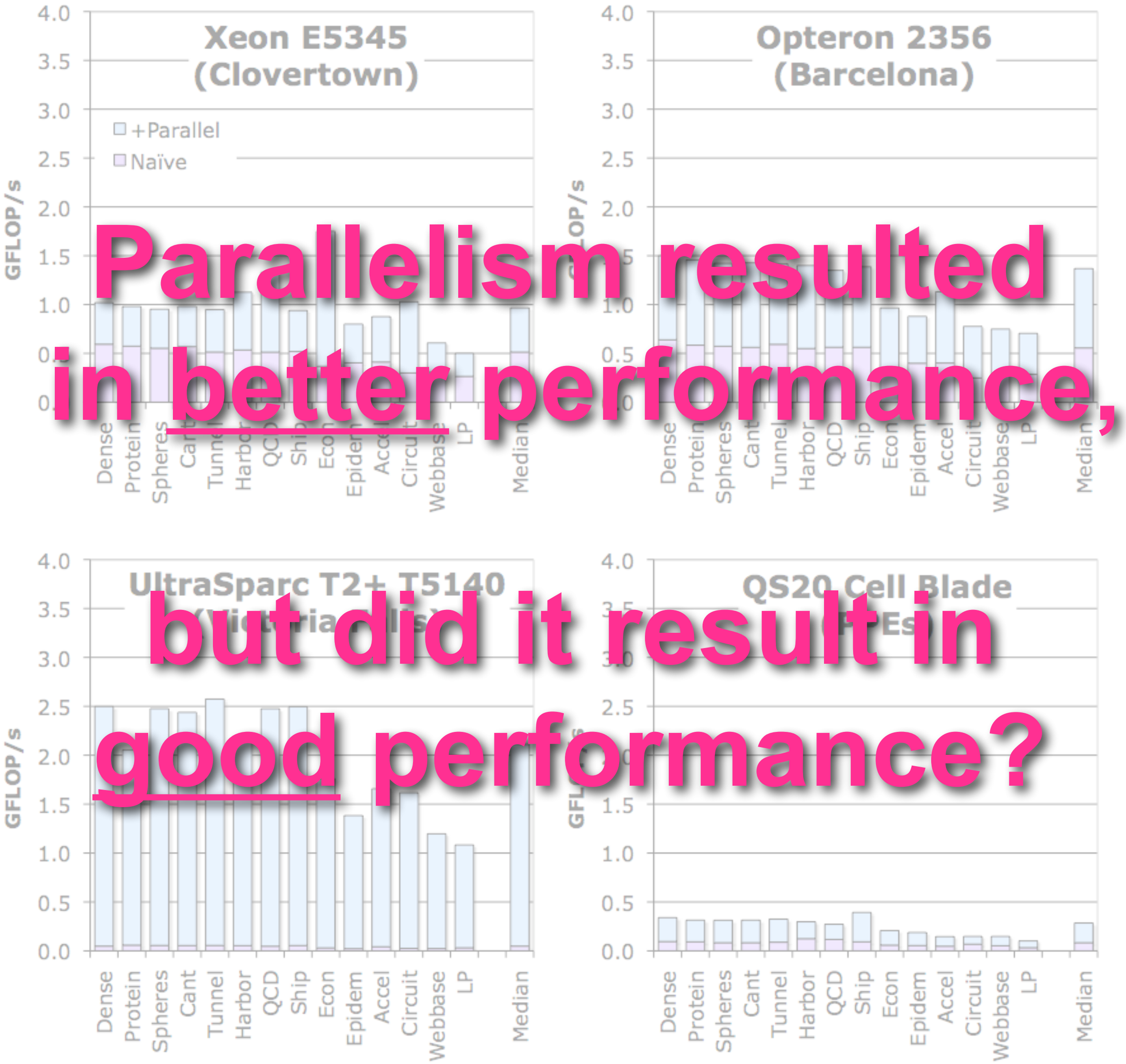
3.4x using 4 threads

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

EECS
Electrical Engineering and
Computer Sciences

# SpMV Performance
## (simple parallelization)

BERKELEY PAR LAB



**Parallelism resulted in <u>better</u> performance, but did it result in <u>good</u> performance?**

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

□ Naïve Pthreads

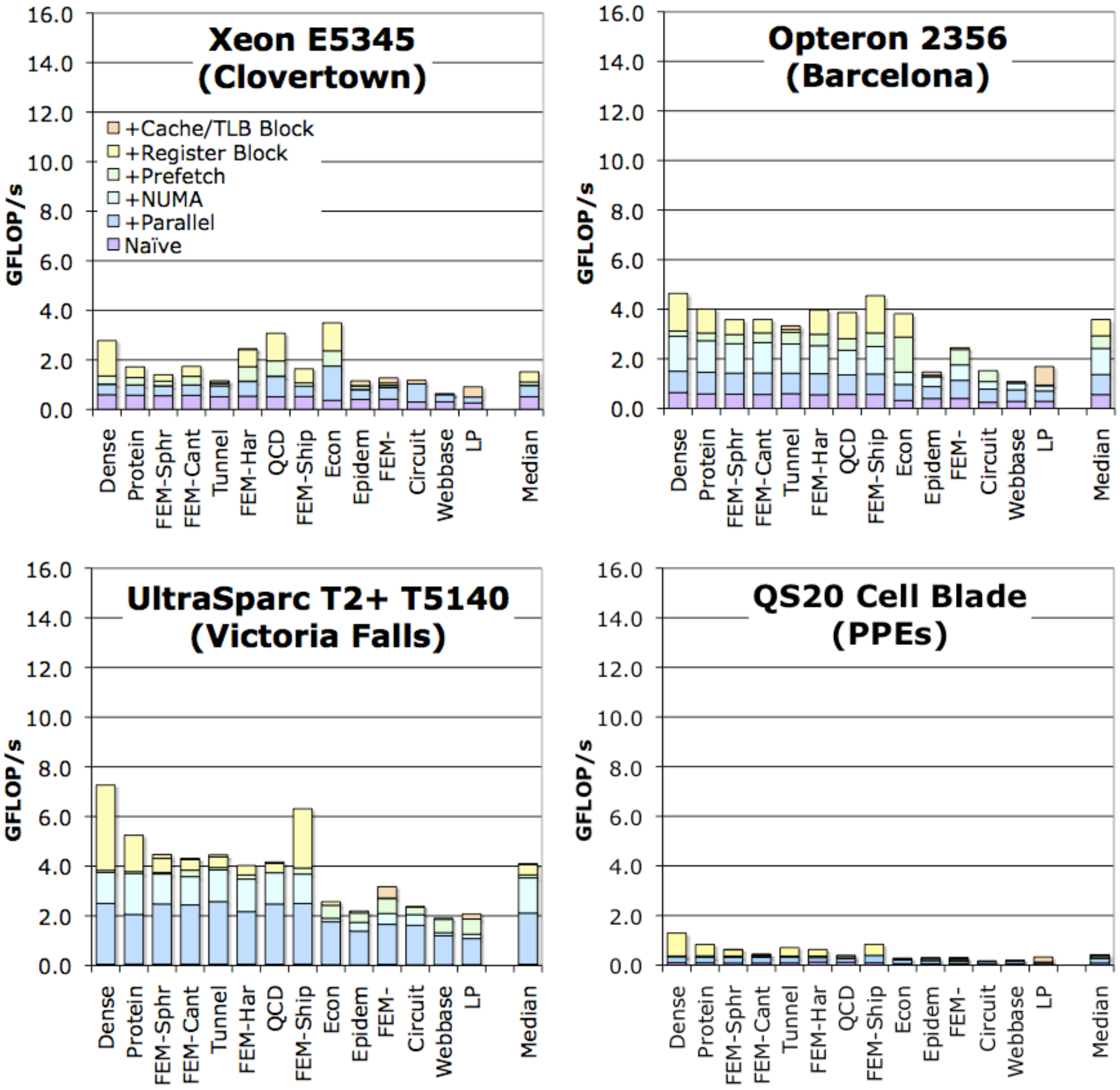
□ Naïve

EECS Electrical Engineering and Computer Sciences

# Auto-tuned SpMV Performance

## (portable C)

BERKELEY PAR LAB



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

🟧 +Cache/LS/TLB Blocking

🟨 +Matrix Compression

🟩 +SW Prefetching

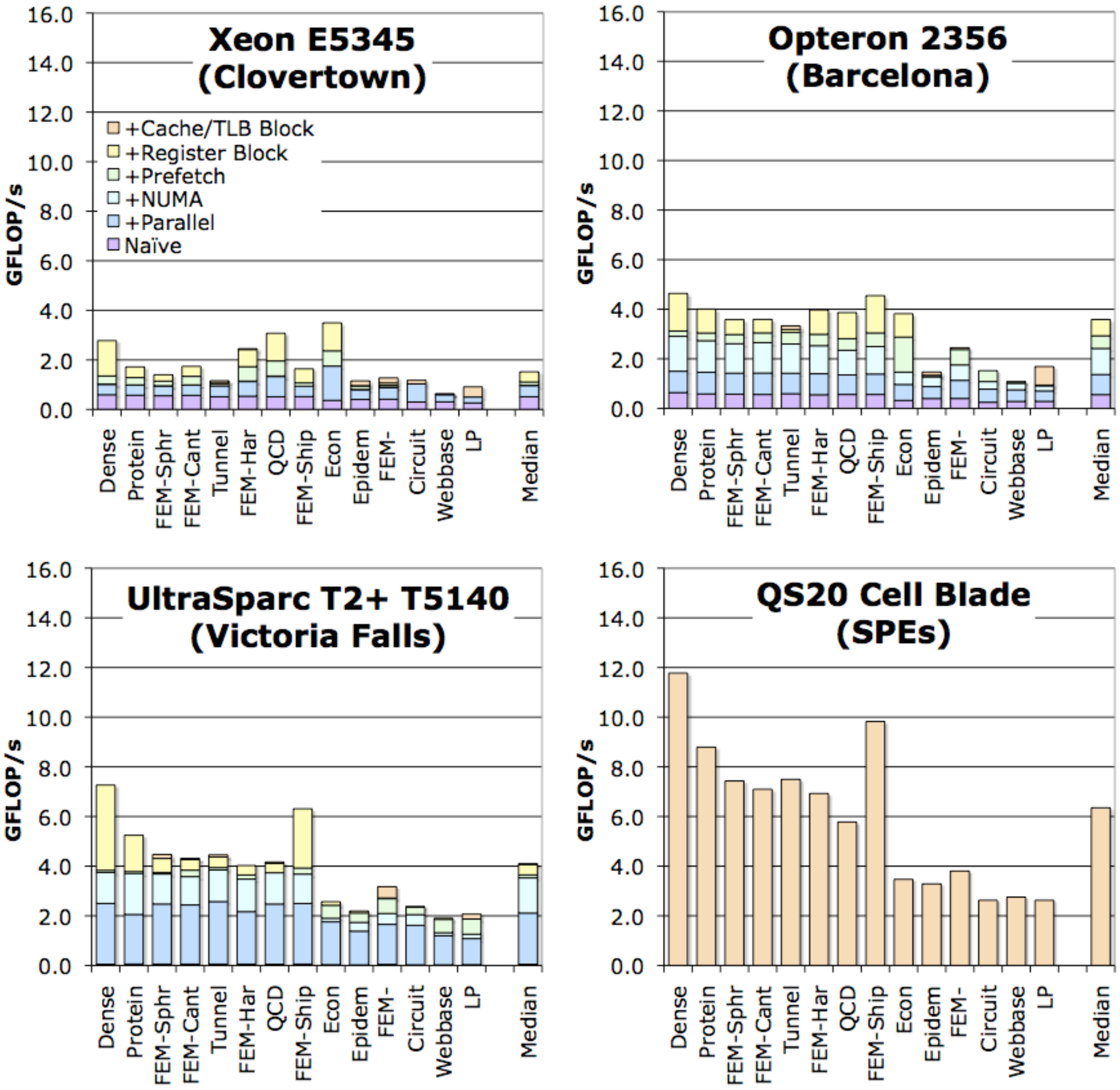
🟦 +NUMA/Affinity

🟦 Naïve Pthreads

🟪 Naïve

# 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?

🟧 +Cache/LS/TLB Blocking

🟨 +Matrix Compression

🟩 +SW Prefetching

□ +NUMA/Affinity

🟦 Naïve Pthreads

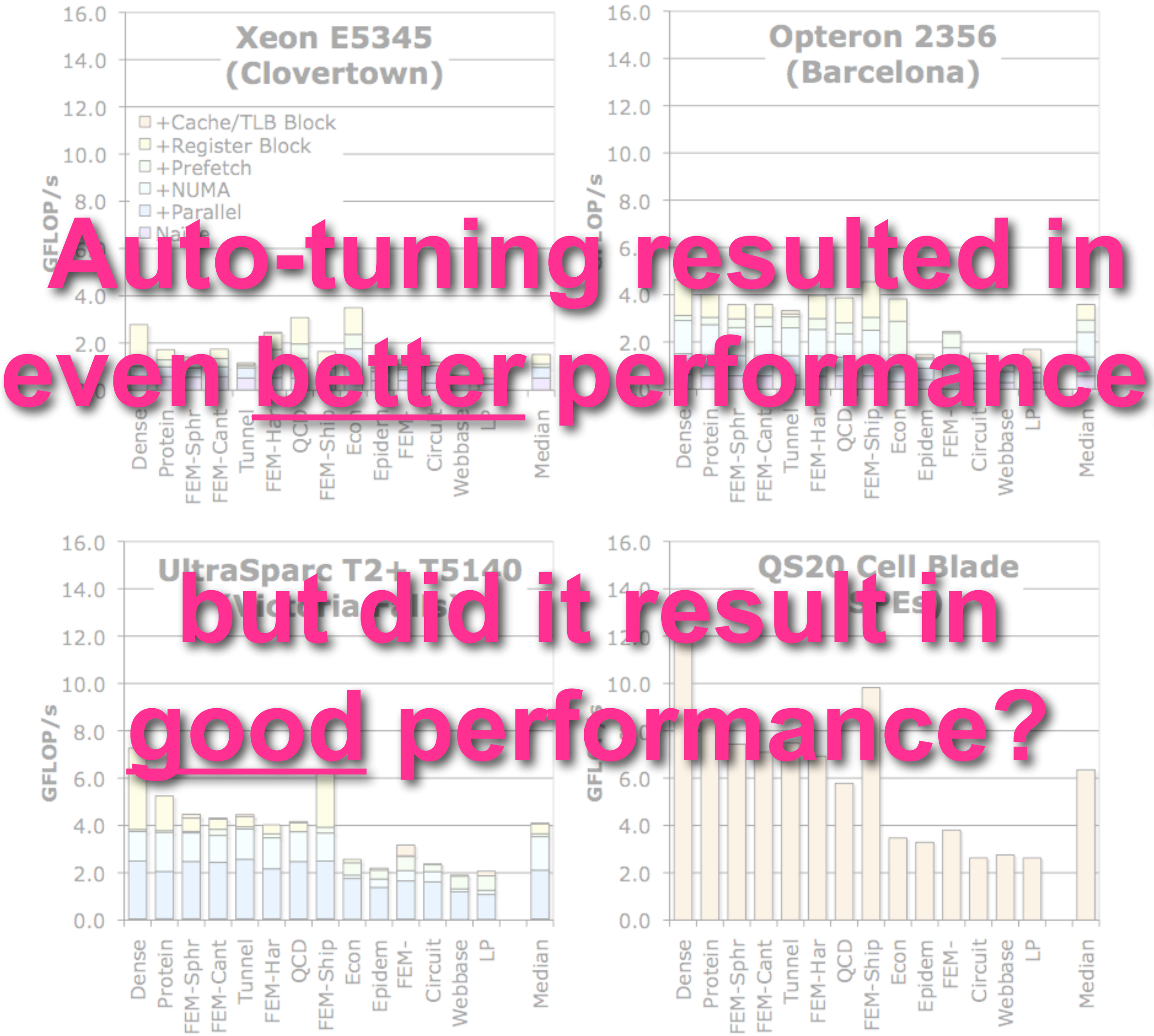
🟪 Naïve

EECS
Electrical Engineering and Computer Sciences

# Auto-tuned SpMV Performance

## (architecture specific optimizations)

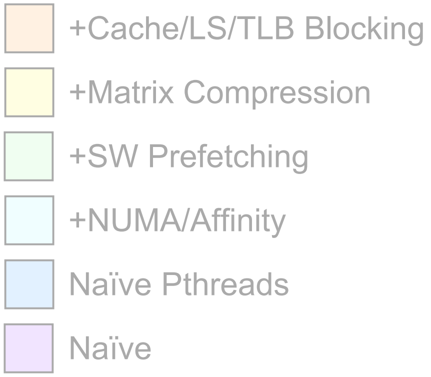
BERKELEY PAR LAB



**Auto-tuning resulted in even 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?**

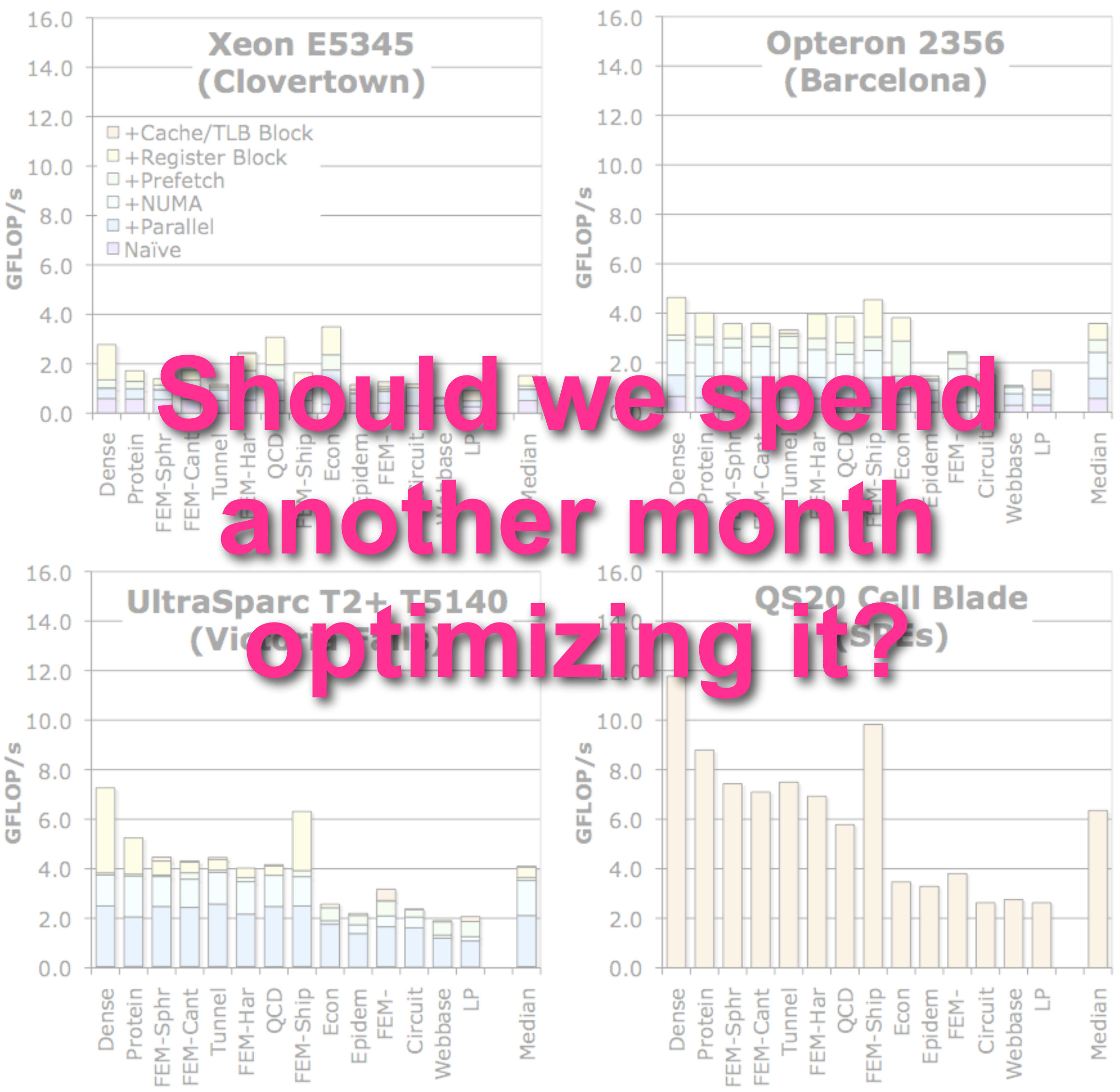


EECS
Electrical Engineering and
Computer Sciences

# Auto-tuned SpMV Performance

## (architecture specific optimizations)

BERKELEY PAR LAB



Should we spend another month optimizing it?

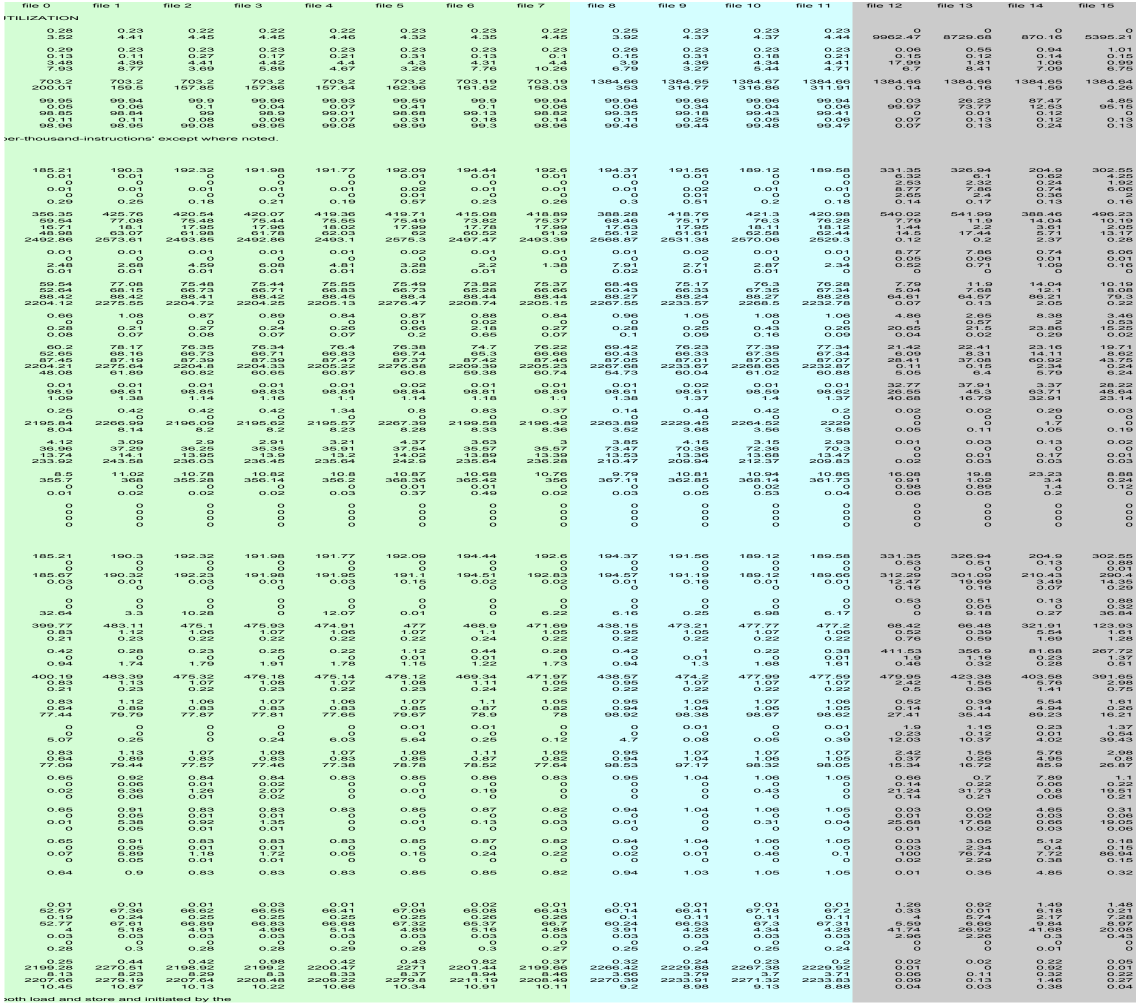
- Fully auto-tuned SpMV performance across the suite of matrices
- Included SPE/local store optimized version
- Why do some optimizations work better on some architectures?
- **Performance is better, but is performance good?**

# How should the bulk of programmers analyze performance ?

EECS
Electrical Engineering and Computer Sciences

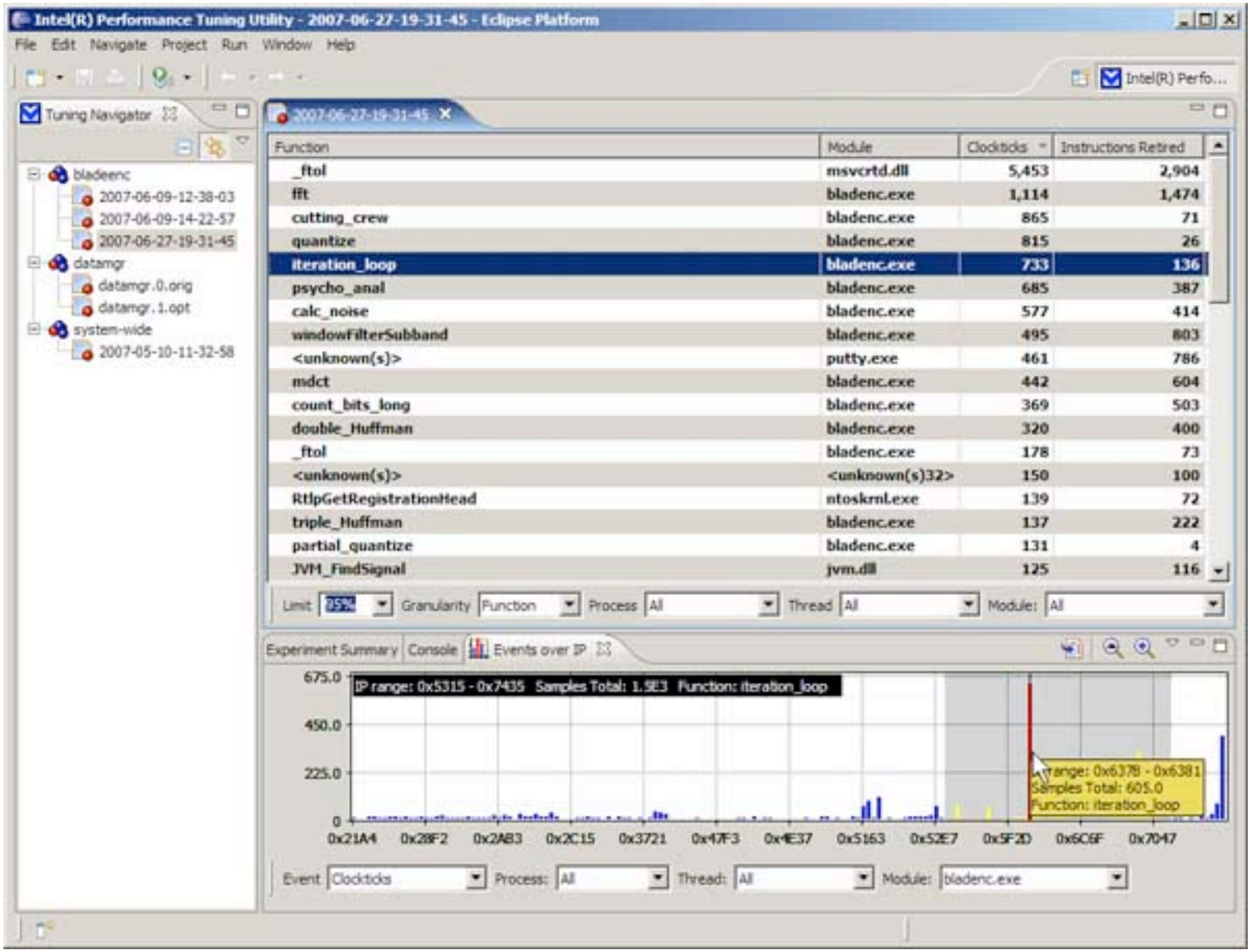
# Spreadsheet of Performance Counters?

BERKELEY PAR LAB



| file 0 | file 1 | file 2 | file 3 | file 4 | file 5 | file 6 | file 7 | file 8 | file 9 | file 10 | file 11 | file 12 | file 13 | file 14 | file 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UTILIZATION | | | | | | | | | | | | | | | |
| 0.28 | 0.23 | 0.22 | 0.22 | 0.22 | 0.23 | 0.23 | 0.22 | 0.25 | 0.23 | 0.23 | 0.23 | 0 | 0 | 0 | 0 |
| 3.52 | 4.41 | 4.45 | 4.45 | 4.46 | 4.32 | 4.35 | 4.45 | 3.92 | 4.37 | 4.37 | 4.44 | 9962.47 | 8729.68 | 870.16 | 5395.21 |
| | | | | | | | | | | | | | | | |
| 0.29 | 0.23 | 0.23 | 0.23 | 0.23 | 0.23 | 0.23 | 0.23 | 0.26 | 0.23 | 0.23 | 0.23 | 0.06 | 0.55 | 0.94 | 1.01 |
| 0.13 | 0.11 | 0.27 | 0.17 | 0.21 | 0.31 | 0.13 | 0.1 | 0.15 | 0.31 | 0.18 | 0.21 | 0.15 | 0.12 | 0.14 | 0.15 |
| 3.48 | 4.36 | 4.41 | 4.42 | 4.4 | 4.3 | 4.31 | 4.4 | 3.9 | 4.36 | 4.34 | 4.41 | 17.99 | 1.81 | 1.06 | 0.99 |
| 7.93 | 8.77 | 3.69 | 5.89 | 4.67 | 3.26 | 7.76 | 10.26 | 6.79 | 3.27 | 5.44 | 4.71 | 6.7 | 8.41 | 7.09 | 6.75 |
| | | | | | | | | | | | | | | | |
| 703.2 | 703.2 | 703.2 | 703.2 | 703.2 | 703.2 | 703.19 | 703.19 | 1384.66 | 1384.65 | 1384.67 | 1384.66 | 1384.66 | 1384.66 | 1384.65 | 1384.64 |
| 200.01 | 159.5 | 157.85 | 157.86 | 157.64 | 162.96 | 161.62 | 158.03 | 353 | 316.77 | 316.86 | 311.91 | 0.14 | 0.16 | 1.59 | 0.26 |
| | | | | | | | | | | | | | | | |
| 99.95 | 99.94 | 99.9 | 99.96 | 99.93 | 99.59 | 99.9 | 99.94 | 99.94 | 99.66 | 99.96 | 99.94 | 0.03 | 26.23 | 87.47 | 4.85 |
| 0.05 | 0.06 | 0.1 | 0.04 | 0.07 | 0.41 | 0.1 | 0.06 | 0.06 | 0.34 | 0.04 | 0.06 | 99.97 | 73.77 | 12.53 | 95.15 |
| 98.85 | 98.84 | 99 | 98.9 | 99.01 | 98.68 | 99.13 | 98.82 | 99.35 | 99.18 | 99.43 | 99.41 | 0 | 0.01 | 0.12 | 0 |
| 0.11 | 0.11 | 0.08 | 0.06 | 0.07 | 0.31 | 0.18 | 0.14 | 0.11 | 0.25 | 0.05 | 0.06 | 0.07 | 0.13 | 0.12 | 0.13 |
| 98.96 | 98.95 | 99.08 | 98.95 | 99.08 | 98.99 | 99.3 | 98.96 | 99.46 | 99.44 | 99.48 | 99.47 | 0.07 | 0.13 | 0.24 | 0.13 |
| per-thousand-instructions' except where noted. | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| 185.21 | 190.3 | 192.32 | 191.98 | 191.77 | 192.09 | 194.44 | 192.6 | 194.37 | 191.56 | 189.12 | 189.58 | 331.35 | 326.94 | 204.9 | 302.55 |
| 0.01 | 0.01 | 0 | 0 | 0 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0 | 0 | 6.32 | 6.1 | 0.62 | 4.25 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2.53 | 2.32 | 0.24 | 1.92 |
| 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.02 | 0.01 | 0.01 | 0.01 | 0.02 | 0.01 | 0.01 | 8.77 | 7.86 | 0.74 | 6.06 |
| 0 | 0 | 0 | 0 | 0 | 0.01 | 0 | 0 | 0 | 0.01 | 0 | 0 | 2.65 | 2.4 | 0.36 | 2 |
| 0.29 | 0.25 | 0.18 | 0.21 | 0.19 | 0.57 | 0.23 | 0.26 | 0.3 | 0.51 | 0.2 | 0.18 | 0.14 | 0.17 | 0.13 | 0.16 |
| | | | | | | | | | | | | | | | |
| 356.35 | 425.76 | 420.54 | 420.07 | 419.36 | 419.71 | 415.08 | 418.89 | 388.28 | 418.76 | 421.3 | 420.98 | 540.02 | 541.99 | 388.46 | 496.23 |
| 59.54 | 77.08 | 75.48 | 75.44 | 75.55 | 75.49 | 73.82 | 75.37 | 68.46 | 75.17 | 76.3 | 76.28 | 7.79 | 11.9 | 14.04 | 10.19 |
| 16.71 | 18.1 | 17.95 | 17.96 | 18.02 | 17.99 | 17.78 | 17.99 | 17.63 | 17.95 | 18.11 | 18.12 | 1.44 | 2.2 | 3.61 | 2.05 |
| 48.98 | 63.07 | 61.98 | 61.78 | 62.03 | 62 | 60.52 | 61.9 | 56.12 | 61.61 | 62.58 | 62.44 | 14.5 | 17.44 | 5.71 | 13.17 |
| 2492.86 | 2573.61 | 2493.85 | 2492.86 | 2493.1 | 2575.3 | 2497.47 | 2493.39 | 2568.87 | 2531.38 | 2570.06 | 2529.3 | 0.12 | 0.2 | 2.37 | 0.28 |
| | | | | | | | | | | | | | | | |
| 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.02 | 0.01 | 0.01 | 0.01 | 0.02 | 0.01 | 0.01 | 8.77 | 7.86 | 0.74 | 6.06 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.05 | 0.06 | 0.01 | 0.01 |
| 2.48 | 2.68 | 4.59 | 6.08 | 4.81 | 3.28 | 2.2 | 1.38 | 7.91 | 2.71 | 2.87 | 2.34 | 0.52 | 0.71 | 1.09 | 0.16 |
| 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.02 | 0.01 | 0 | 0.02 | 0.01 | 0.01 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | |
| 59.54 | 77.08 | 75.48 | 75.44 | 75.55 | 75.49 | 73.82 | 75.37 | 68.46 | 75.17 | 76.3 | 76.28 | 7.79 | 11.9 | 14.04 | 10.19 |
| 52.64 | 68.15 | 66.73 | 66.71 | 66.83 | 66.73 | 65.28 | 66.66 | 60.43 | 66.33 | 67.35 | 67.34 | 5.04 | 7.68 | 12.1 | 8.08 |
| 88.42 | 88.42 | 88.41 | 88.42 | 88.45 | 88.4 | 88.44 | 88.44 | 88.27 | 88.24 | 88.27 | 88.28 | 64.61 | 64.57 | 86.21 | 79.3 |
| 2204.12 | 2275.55 | 2204.72 | 2204.25 | 2205.13 | 2276.47 | 2208.74 | 2205.15 | 2267.55 | 2233.57 | 2268.5 | 2232.78 | 0.07 | 0.13 | 2.05 | 0.22 |
| | | | | | | | | | | | | | | | |
| 0.66 | 1.08 | 0.87 | 0.89 | 0.84 | 0.87 | 0.88 | 0.84 | 0.96 | 1.05 | 1.08 | 1.06 | 4.86 | 2.65 | 8.38 | 3.46 |
| 0 | 0 | 0 | 0 | 0 | 0.01 | 0.02 | 0 | 0 | 0 | 0 | 0 | 1 | 0.57 | 2 | 0.53 |
| 0.28 | 0.21 | 0.27 | 0.24 | 0.26 | 0.66 | 2.18 | 0.27 | 0.28 | 0.25 | 0.43 | 0.26 | 20.65 | 21.5 | 23.86 | 15.25 |
| 0.08 | 0.07 | 0.08 | 0.07 | 0.07 | 0.2 | 0.65 | 0.07 | 0.1 | 0.09 | 0.16 | 0.09 | 0.04 | 0.02 | 0.29 | 0.02 |
| | | | | | | | | | | | | | | | |
| 60.2 | 78.17 | 76.35 | 76.34 | 76.4 | 76.38 | 74.7 | 76.22 | 69.42 | 76.23 | 77.39 | 77.34 | 21.42 | 22.41 | 23.16 | 19.71 |
| 52.65 | 68.16 | 66.73 | 66.71 | 66.83 | 66.74 | 65.3 | 66.66 | 60.43 | 66.33 | 67.35 | 67.34 | 6.09 | 8.31 | 14.11 | 8.62 |
| 87.45 | 87.19 | 87.39 | 87.39 | 87.47 | 87.37 | 87.42 | 87.46 | 87.05 | 87.01 | 87.03 | 87.07 | 28.41 | 37.08 | 60.92 | 43.75 |
| 2204.21 | 2275.64 | 2204.8 | 2204.33 | 2205.22 | 2276.68 | 2209.39 | 2205.23 | 2267.68 | 2233.67 | 2268.66 | 2232.87 | 0.11 | 0.15 | 2.34 | 0.24 |
| 48.08 | 61.89 | 60.82 | 60.65 | 60.87 | 60.8 | 59.38 | 60.74 | 54.73 | 60.04 | 61.02 | 60.88 | 5.05 | 6.4 | 5.79 | 6.24 |
| | | | | | | | | | | | | | | | |
| 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.02 | 0.01 | 0.01 | 0.01 | 0.02 | 0.01 | 0.01 | 32.77 | 37.91 | 3.37 | 28.22 |
| 98.9 | 98.61 | 98.85 | 98.83 | 98.89 | 98.84 | 98.81 | 98.89 | 98.61 | 98.61 | 98.59 | 98.62 | 26.55 | 45.3 | 63.71 | 48.64 |
| 1.09 | 1.38 | 1.14 | 1.16 | 1.1 | 1.14 | 1.18 | 1.1 | 1.38 | 1.37 | 1.4 | 1.37 | 40.68 | 16.79 | 32.91 | 23.14 |
| | | | | | | | | | | | | | | | |
| 0.25 | 0.42 | 0.42 | 0.42 | 1.34 | 0.8 | 0.83 | 0.37 | 0.14 | 0.44 | 0.42 | 0.2 | 0.02 | 0.02 | 0.29 | 0.03 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2195.84 | 2266.99 | 2196.09 | 2195.62 | 2195.57 | 2267.39 | 2199.58 | 2196.42 | 2263.89 | 2229.45 | 2264.52 | 2229 | 0 | 0 | 1.7 | 0 |
| 8.04 | 8.14 | 8.2 | 8.2 | 8.23 | 8.28 | 8.33 | 8.36 | 3.52 | 3.68 | 3.56 | 3.58 | 0.05 | 0.11 | 0.05 | 0.19 |
| | | | | | | | | | | | | | | | |
| 4.12 | 3.09 | 2.9 | 2.91 | 3.21 | 4.37 | 3.63 | 3 | 3.85 | 4.15 | 3.15 | 2.93 | 0.01 | 0.03 | 0.13 | 0.02 |
| 36.96 | 37.29 | 36.25 | 35.35 | 35.91 | 37.54 | 35.57 | 35.57 | 73.47 | 70.36 | 72.36 | 70.3 | 0 | 0 | 0 | 0 |
| 13.74 | 14.1 | 13.95 | 13.9 | 13.2 | 14.02 | 13.89 | 13.39 | 13.53 | 13.36 | 13.68 | 13.47 | 0 | 0.01 | 0.17 | 0.01 |
| 233.92 | 243.58 | 236.03 | 236.45 | 235.64 | 242.9 | 235.64 | 236.28 | 210.47 | 209.94 | 212.37 | 209.83 | 0.02 | 0.03 | 0.03 | 0.03 |
| | | | | | | | | | | | | | | | |
| 8.5 | 11.02 | 10.78 | 10.82 | 10.8 | 10.87 | 10.68 | 10.76 | 9.79 | 10.81 | 10.94 | 10.86 | 16.08 | 19.8 | 23.23 | 8.88 |
| 355.7 | 368 | 355.28 | 356.14 | 356.2 | 368.36 | 365.42 | 356 | 367.11 | 362.85 | 368.14 | 361.73 | 0.91 | 1.02 | 3.4 | 0.24 |
| 0 | 0 | 0 | 0 | 0 | 0.01 | 0.01 | 0 | 0 | 0 | 0.02 | 0 | 0.98 | 0.89 | 1.4 | 0.12 |
| 0.01 | 0.02 | 0.02 | 0.02 | 0.03 | 0.37 | 0.49 | 0.02 | 0.03 | 0.05 | 0.53 | 0.04 | 0.06 | 0.05 | 0.2 | 0 |
| | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | |
| 185.21 | 190.3 | 192.32 | 191.98 | 191.77 | 192.09 | 194.44 | 192.6 | 194.37 | 191.56 | 189.12 | 189.58 | 331.35 | 326.94 | 204.9 | 302.55 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.53 | 0.51 | 0.13 | 0.88 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.01 |
| 185.67 | 190.32 | 192.23 | 191.98 | 191.95 | 191.1 | 194.51 | 192.83 | 194.57 | 191.19 | 189.12 | 189.66 | 312.29 | 301.09 | 210.43 | 290.4 |
| 0.03 | 0.01 | 0.03 | 0.01 | 0.03 | 0.15 | 0.02 | 0.02 | 0.01 | 0.16 | 0.01 | 0.01 | 12.47 | 19.69 | 3.49 | 14.35 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.16 | 0.16 | 0.07 | 0.29 |
| | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.53 | 0.51 | 0.13 | 0.88 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.05 | 0 | 0.32 |
| 32.64 | 3.3 | 10.28 | 0 | 12.07 | 0.01 | 0 | 6.22 | 6.16 | 0.25 | 6.98 | 6.17 | 0 | 9.18 | 0.27 | 36.84 |
| | | | | | | | | | | | | | | | |
| 399.77 | 483.11 | 475.1 | 475.93 | 474.91 | 477 | 468.9 | 471.69 | 438.15 | 473.21 | 477.77 | 477.2 | 68.42 | 66.48 | 321.91 | 123.93 |
| 0.83 | 1.12 | 1.06 | 1.07 | 1.06 | 1.07 | 1.1 | 1.05 | 0.95 | 1.05 | 1.07 | 1.06 | 0.52 | 0.39 | 5.54 | 1.61 |
| 0.21 | 0.23 | 0.22 | 0.22 | 0.22 | 0.22 | 0.24 | 0.22 | 0.22 | 0.22 | 0.22 | 0.22 | 0.76 | 0.59 | 1.69 | 1.28 |
| | | | | | | | | | | | | | | | |
| 0.42 | 0.28 | 0.23 | 0.25 | 0.22 | 1.12 | 0.44 | 0.28 | 0.42 | 1 | 0.22 | 0.38 | 411.53 | 356.9 | 81.68 | 267.72 |
| 0 | 0 | 0 | 0 | 0 | 0.01 | 0.01 | 0 | 0 | 0.01 | 0 | 0.01 | 1.9 | 1.16 | 0.23 | 1.37 |
| 0.94 | 1.74 | 1.79 | 1.91 | 1.78 | 1.15 | 1.22 | 1.73 | 0.94 | 1.3 | 1.68 | 1.61 | 0.46 | 0.32 | 0.28 | 0.51 |
| | | | | | | | | | | | | | | | |
| 400.19 | 483.39 | 475.32 | 476.18 | 475.14 | 478.12 | 469.34 | 471.97 | 438.57 | 474.2 | 477.99 | 477.59 | 479.95 | 423.38 | 403.58 | 391.65 |
| 0.83 | 1.13 | 1.07 | 1.08 | 1.07 | 1.08 | 1.11 | 1.05 | 0.95 | 1.07 | 1.07 | 1.07 | 2.42 | 1.55 | 5.76 | 2.98 |
| 0.21 | 0.23 | 0.22 | 0.23 | 0.22 | 0.23 | 0.24 | 0.22 | 0.22 | 0.22 | 0.22 | 0.22 | 0.5 | 0.36 | 1.41 | 0.75 |
| | | | | | | | | | | | | | | | |
| 0.83 | 1.12 | 1.06 | 1.07 | 1.06 | 1.07 | 1.1 | 1.05 | 0.95 | 1.05 | 1.07 | 1.06 | 0.52 | 0.39 | 5.54 | 1.61 |
| 0.64 | 0.89 | 0.83 | 0.83 | 0.83 | 0.85 | 0.87 | 0.82 | 0.94 | 1.04 | 1.06 | 1.05 | 0.14 | 0.14 | 4.94 | 0.26 |
| 77.44 | 79.79 | 77.87 | 77.81 | 77.65 | 79.67 | 78.9 | 78 | 98.92 | 98.38 | 98.67 | 98.62 | 27.41 | 35.44 | 89.23 | 16.21 |
| | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0.01 | 0.01 | 0 | 0 | 0.01 | 0 | 0.01 | 1.9 | 1.16 | 0.23 | 1.37 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.23 | 0.12 | 0.01 | 0.54 |
| 5.07 | 0.25 | 0 | 0.24 | 6.03 | 5.64 | 0.25 | 0.12 | 4.7 | 0.08 | 0.05 | 0.39 | 12.03 | 10.37 | 4.02 | 39.43 |
| | | | | | | | | | | | | | | | |
| 0.83 | 1.13 | 1.07 | 1.08 | 1.07 | 1.08 | 1.11 | 1.05 | 0.95 | 1.07 | 1.07 | 1.07 | 2.42 | 1.55 | 5.76 | 2.98 |
| 0.64 | 0.89 | 0.83 | 0.83 | 0.83 | 0.85 | 0.87 | 0.82 | 0.94 | 1.04 | 1.06 | 1.05 | 0.37 | 0.26 | 4.95 | 0.8 |
| 77.09 | 79.44 | 77.57 | 77.46 | 77.38 | 78.78 | 78.52 | 77.64 | 98.53 | 97.17 | 98.32 | 98.05 | 15.34 | 16.72 | 85.9 | 26.87 |
| | | | | | | | | | | | | | | | |
| 0.65 | 0.92 | 0.84 | 0.84 | 0.83 | 0.85 | 0.86 | 0.83 | 0.95 | 1.04 | 1.06 | 1.05 | 0.66 | 0.7 | 7.89 | 1.1 |
| 0 | 0.06 | 0.01 | 0.02 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.14 | 0.22 | 0.06 | 0.22 |
| 0.02 | 6.36 | 1.26 | 2.07 | 0 | 0.01 | 0.19 | 0 | 0 | 0 | 0.43 | 0 | 21.24 | 31.73 | 0.8 | 19.51 |
| 0 | 0.06 | 0.01 | 0.02 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.14 | 0.21 | 0.06 | 0.21 |
| | | | | | | | | | | | | | | | |
| 0.65 | 0.91 | 0.83 | 0.83 | 0.83 | 0.85 | 0.87 | 0.82 | 0.94 | 1.04 | 1.06 | 1.05 | 0.03 | 0.09 | 4.65 | 0.31 |
| 0 | 0.05 | 0.01 | 0.01 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.01 | 0.02 | 0.03 | 0.06 |
| 0.01 | 5.38 | 0.92 | 1.35 | 0 | 0.01 | 0.13 | 0.03 | 0.01 | 0 | 0.31 | 0.04 | 25.68 | 17.68 | 0.66 | 19.05 |
| 0 | 0.05 | 0.01 | 0.01 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.01 | 0.02 | 0.03 | 0.06 |
| | | | | | | | | | | | | | | | |
| 0.65 | 0.91 | 0.83 | 0.83 | 0.83 | 0.85 | 0.87 | 0.82 | 0.94 | 1.04 | 1.06 | 1.05 | 0.03 | 3.05 | 5.12 | 0.18 |
| 0 | 0.05 | 0.01 | 0.01 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.03 | 2.34 | 0.4 | 0.15 |
| 0.07 | 5.89 | 1.18 | 1.72 | 0.05 | 0.15 | 0.24 | 0.22 | 0.02 | 0.01 | 0.46 | 0.1 | 100 | 76.74 | 7.72 | 86.94 |
| 0 | 0.05 | 0.01 | 0.01 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.02 | 2.29 | 0.38 | 0.15 |
| | | | | | | | | | | | | | | | |
| 0.64 | 0.9 | 0.83 | 0.83 | 0.83 | 0.85 | 0.85 | 0.82 | 0.94 | 1.03 | 1.05 | 1.05 | 0.01 | 0.35 | 4.85 | 0.32 |
| | | | | | | | | | | | | | | | |
| 0.01 | 0.01 | 0.01 | 0.03 | 0.01 | 0.01 | 0.02 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 1.26 | 0.92 | 1.49 | 1.48 |
| 52.57 | 67.36 | 66.62 | 66.55 | 66.41 | 67.06 | 65.08 | 66.43 | 60.14 | 66.41 | 67.18 | 67.2 | 0.33 | 0.01 | 6.18 | 0.21 |
| 0.19 | 0.24 | 0.25 | 0.25 | 0.25 | 0.25 | 0.26 | 0.26 | 0.1 | 0.11 | 0.11 | 0.11 | 4 | 5.74 | 2.17 | 7.28 |
| 52.77 | 67.61 | 66.89 | 66.83 | 66.68 | 67.32 | 65.37 | 66.7 | 60.24 | 66.53 | 67.3 | 67.31 | 5.59 | 6.66 | 9.84 | 8.97 |
| 4 | 5.18 | 4.91 | 4.96 | 5.14 | 4.89 | 5.16 | 4.88 | 3.91 | 4.28 | 4.34 | 4.28 | 41.74 | 26.92 | 41.68 | 20.08 |
| 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 2.96 | 2.26 | 0.3 | 0.43 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.28 | 0.3 | 0.28 | 0.28 | 0.29 | 0.28 | 0.3 | 0.27 | 0.25 | 0.24 | 0.25 | 0.24 | 0 | 0 | 0.01 | 0 |
| | | | | | | | | | | | | | | | |
| 0.25 | 0.44 | 0.42 | 0.98 | 0.42 | 0.43 | 0.82 | 0.37 | 0.32 | 0.24 | 0.23 | 0.2 | 0.02 | 0.02 | 0.22 | 0.05 |
| 2199.28 | 2270.51 | 2198.92 | 2199.2 | 2200.47 | 2271 | 2201.44 | 2199.66 | 2266.42 | 2229.88 | 2267.38 | 2229.92 | 0.01 | 0 | 0.92 | 0.01 |
| 8.13 | 8.23 | 8.29 | 8.3 | 8.33 | 8.37 | 8.94 | 8.46 | 3.66 | 3.79 | 3.7 | 3.71 | 0.06 | 0.11 | 0.32 | 0.22 |
| 2207.66 | 2279.19 | 2207.64 | 2208.48 | 2209.22 | 2279.8 | 2211.19 | 2208.49 | 2270.39 | 2233.91 | 2271.32 | 2233.83 | 0.09 | 0.13 | 1.46 | 0.27 |
| 10.45 | 10.87 | 10.13 | 10.22 | 10.66 | 10.34 | 10.91 | 10.11 | 9.2 | 8.98 | 9.13 | 8.88 | 0.04 | 0.03 | 0.38 | 0.04 |
| both load and store and initiated by the | | | | | | | | | | | | | | | |

# VTUNE ?



| Function | Module | Clockticks | Instructions Retired |
|---|---|---|---|
| _ftol | msvcrtd.dll | 5,453 | 2,904 |
| fft | bladenc.exe | 1,114 | 1,474 |
| cutting_crew | bladenc.exe | 865 | 71 |
| quantize | bladenc.exe | 815 | 26 |
| iteration_loop | bladenc.exe | 733 | 136 |
| psycho_anal | bladenc.exe | 685 | 387 |
| calc_noise | bladenc.exe | 577 | 414 |
| windowFilterSubband | bladenc.exe | 495 | 803 |
| <unknown(s)> | putty.exe | 461 | 786 |
| mdct | bladenc.exe | 442 | 604 |
| count_bits_long | bladenc.exe | 369 | 503 |
| double_Huffman | bladenc.exe | 320 | 400 |
| _ftol | bladenc.exe | 178 | 73 |
| <unknown(s)> | <unknown(s)32> | 150 | 100 |
| RtlpGetRegistrationHead | ntoskrnl.exe | 139 | 72 |
| triple_Huffman | bladenc.exe | 137 | 222 |
| partial_quantize | bladenc.exe | 131 | 4 |
| JVM_FindSignal | jvm.dll | 125 | 116 |

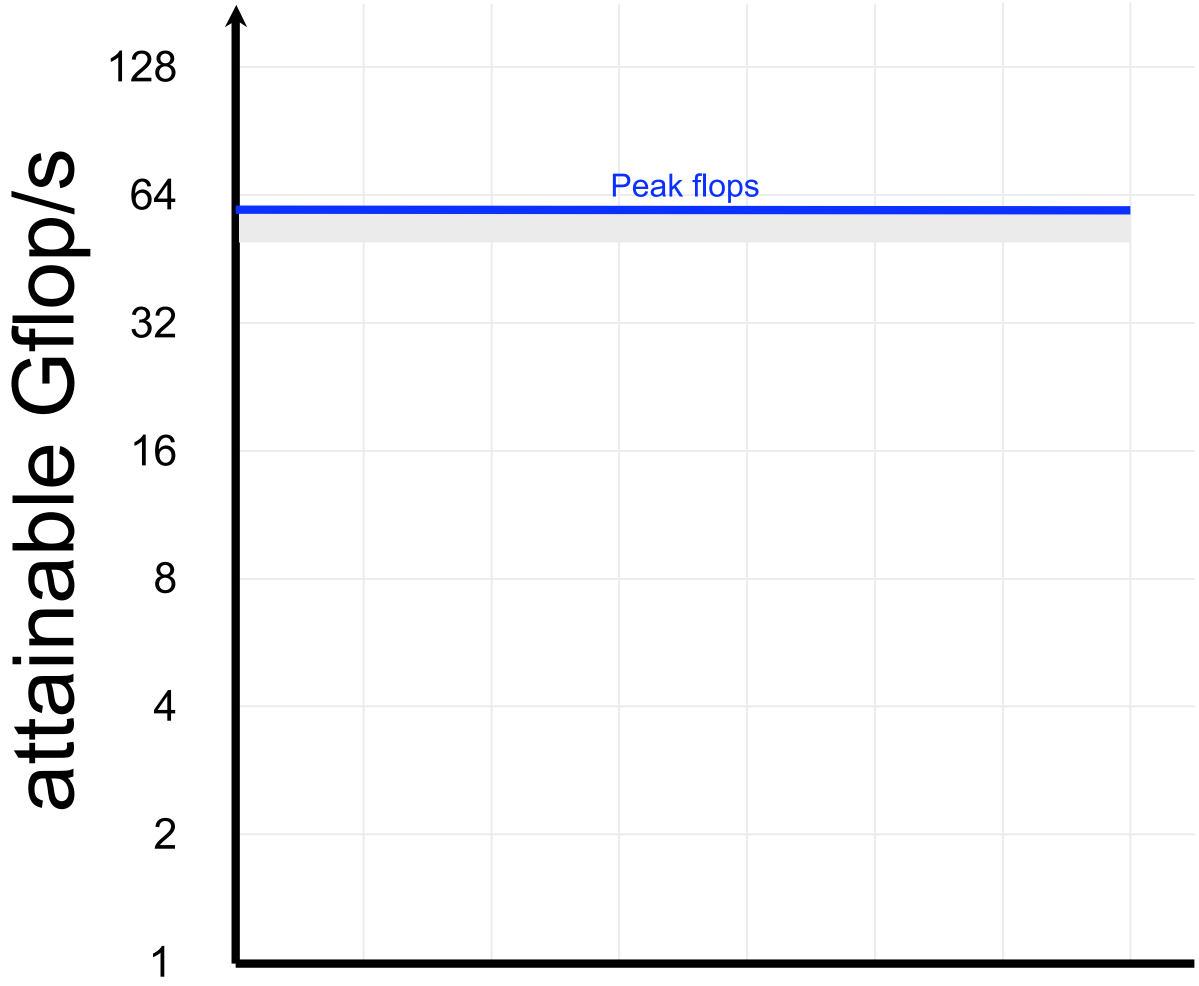
Limit: 95% Granularity: Function Process: All Thread: All Module: All

IP range: 0x5315 - 0x7435 Samples Total: 1.5E3 Function: iteration_loop

range: 0x6378 - 0x6381
Samples Total: 605.0
Function: iteration_loop

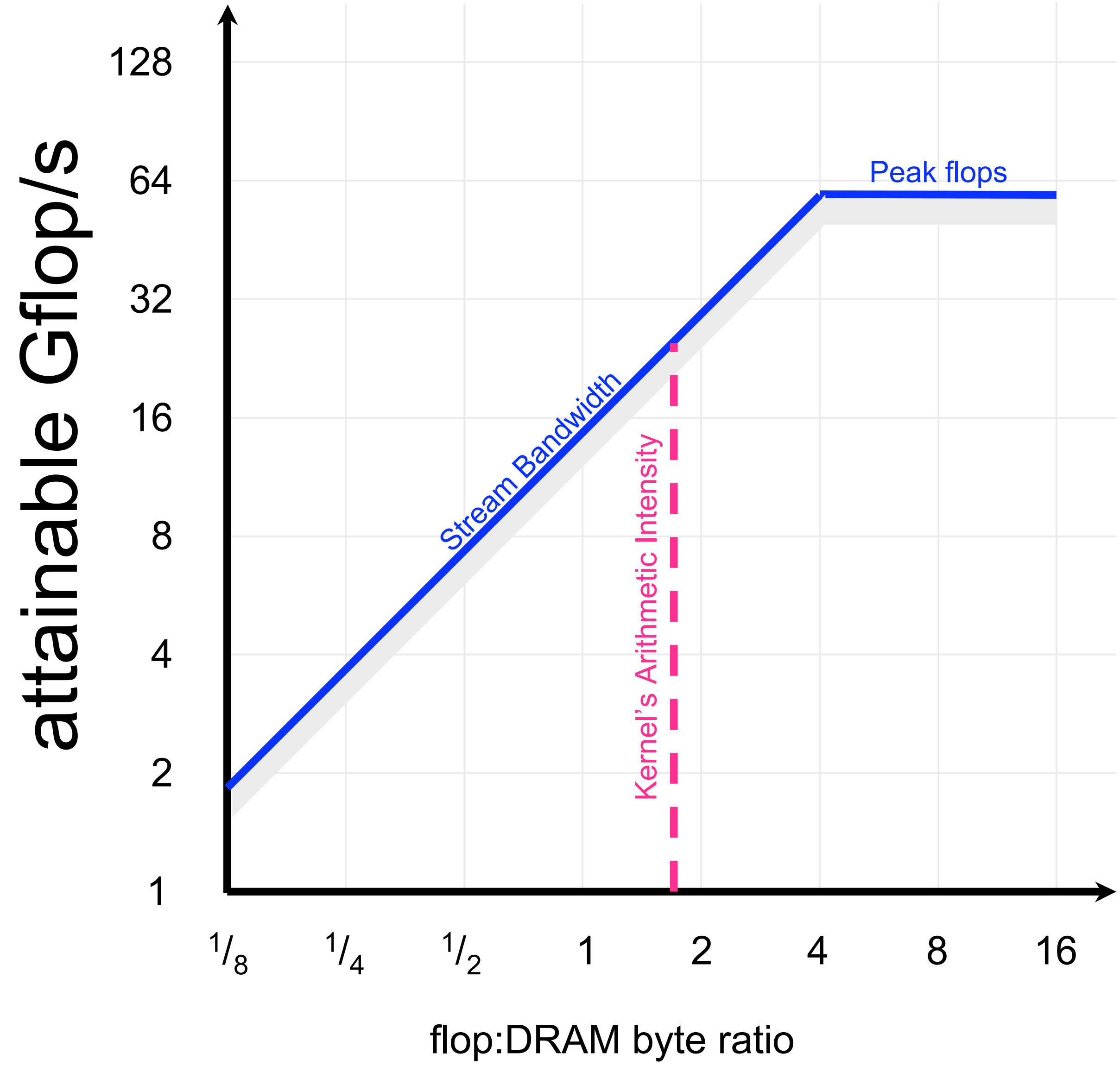
Event: Clockticks Process: All Thread: All Module: bladenc.exe

# Roofline Model



- It would be great if we could always get peak performance

# Roofline Model (2)



- Machines have finite memory bandwidth
- Apply a Bound and Bottleneck Analysis
- Still Unrealistically optimistic model

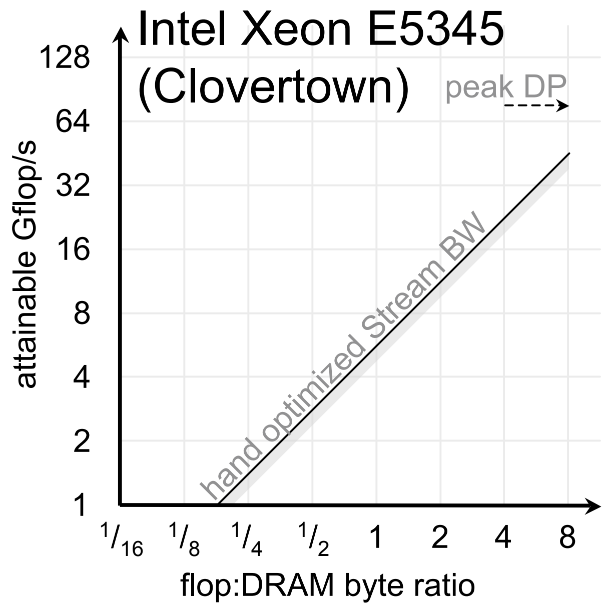
$$\text{Gflop/s(AI)} = \min \begin{cases} \text{Peak Gflop/s} \\ \text{StreamBW * AI} \end{cases}$$

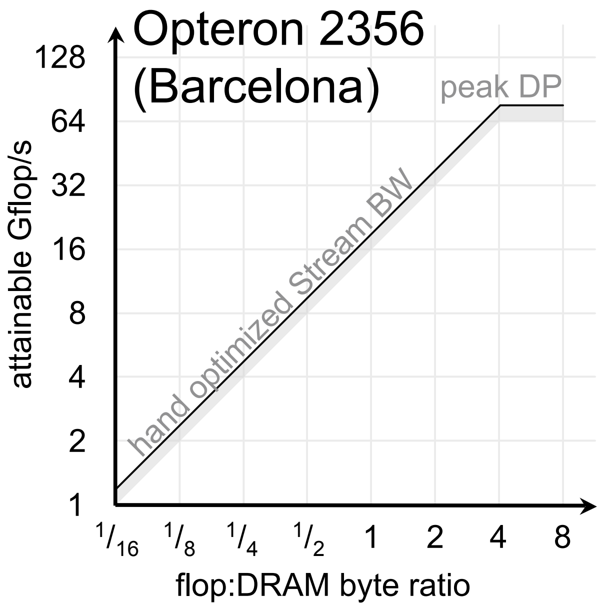
EECS
Electrical Engineering and Computer Sciences

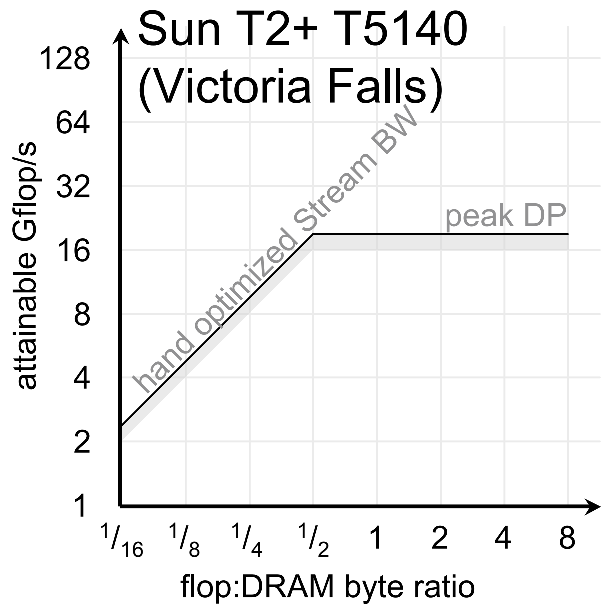
# Naïve Roofline Model

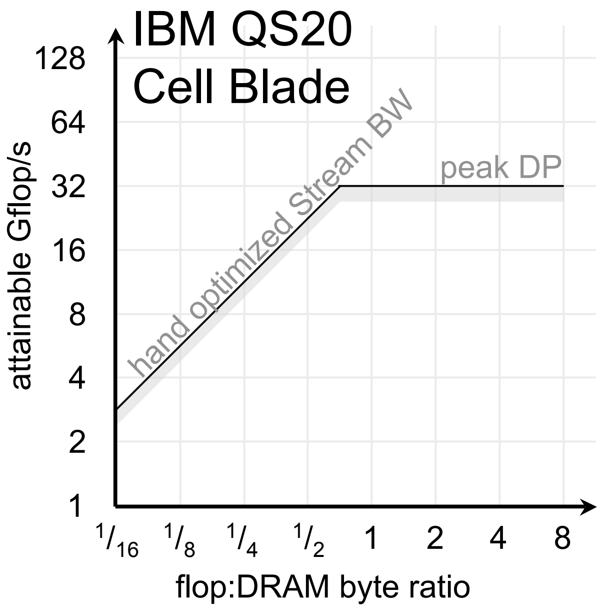
## (applied to four architectures)

BERKELEY PAR LAB









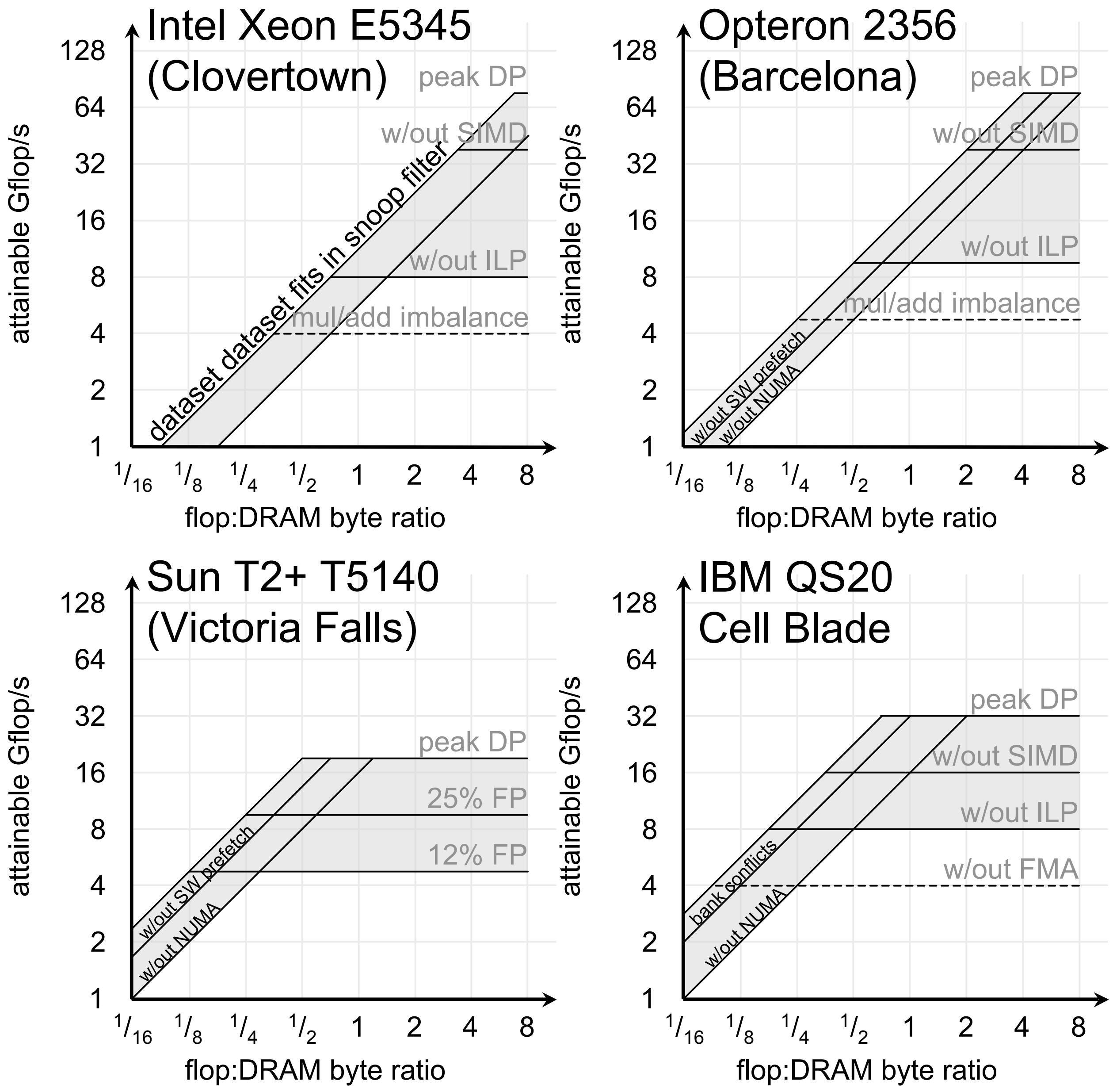
- Bound and Bottleneck Analysis
- Unrealistically optimistic model
- Hand optimized Stream BW benchmark

$$\text{Gflop/s(AI)} = \min \begin{cases} \text{Peak Gflop/s} \\ \text{StreamBW * AI} \end{cases}$$

EECS
Electrical Engineering and Computer Sciences

# Roofline model for SpMV

BERKELEY PAR LAB



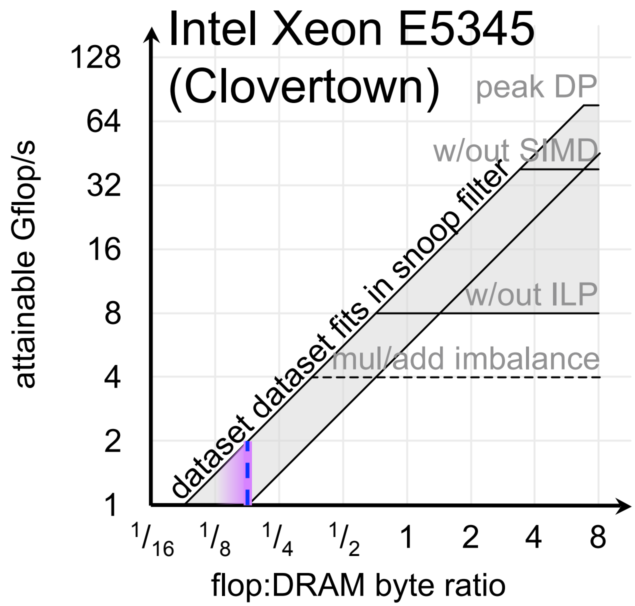
- Delineate performance by architectural paradigm = **'ceilings'**
- In-core optimizations 1..i
- DRAM optimizations 1..j

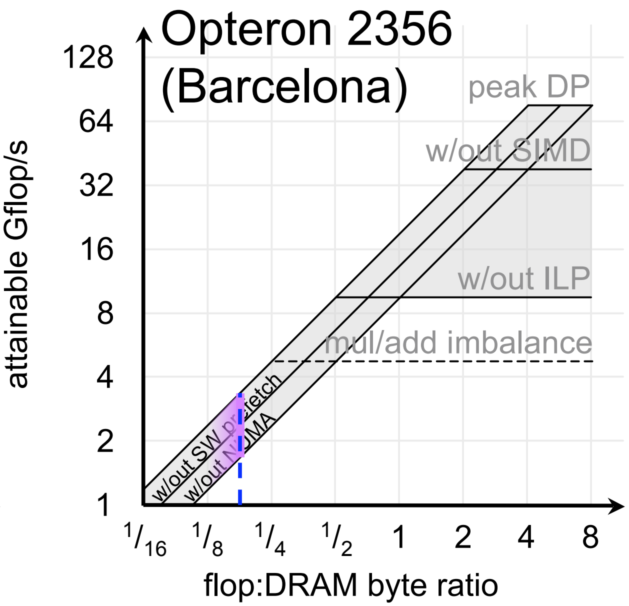
$$\mathbf{GFlops}_{i,j}\mathbf{(AI) = min} \begin{cases} \mathbf{InCoreGFlops}_i \\ \mathbf{StreamBW}_j \mathbf{\ *\ AI} \end{cases}$$

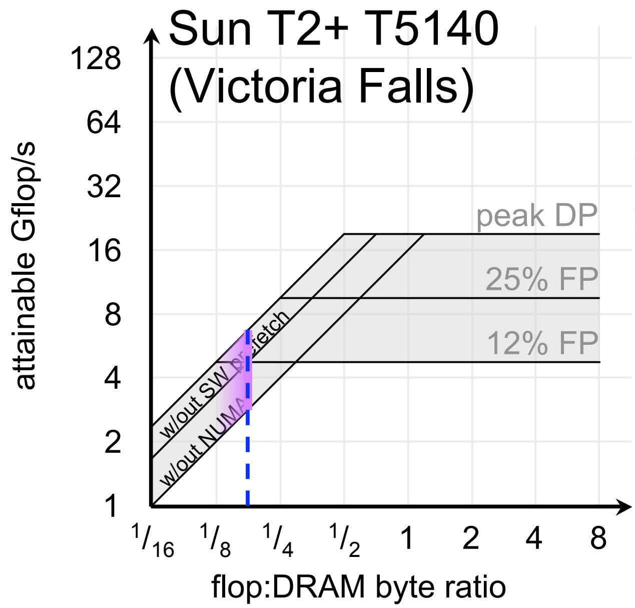
- FMA is inherent in SpMV (place at bottom)

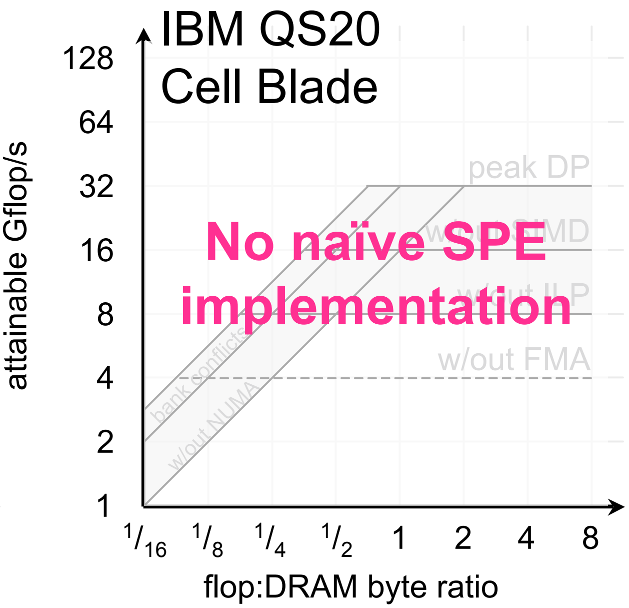
# Roofline model for SpMV

## (overlay arithmetic intensity)


Intel Xeon E5345 (Clovertown)


Opteron 2356 (Barcelona)


Sun T2+ T5140 (Victoria Falls)


IBM QS20 Cell Blade

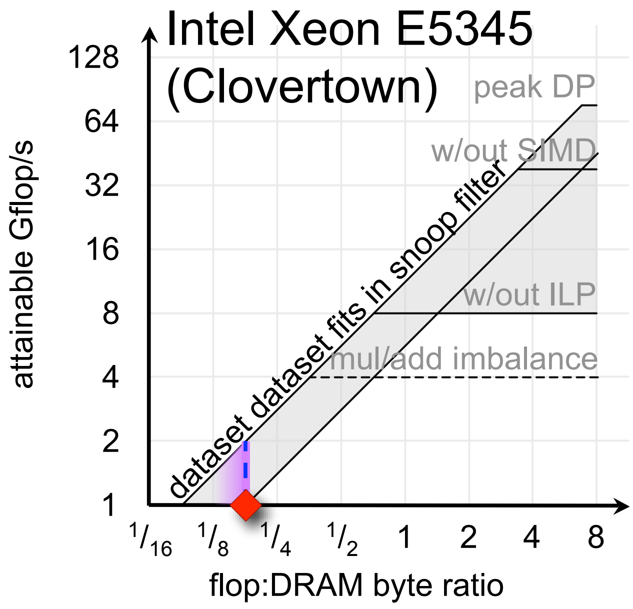
- Two unit stride streams
- Inherent FMA
- No ILP
- No DLP
- FP is 12-25%
- Naïve compulsory flop:byte < 0.166

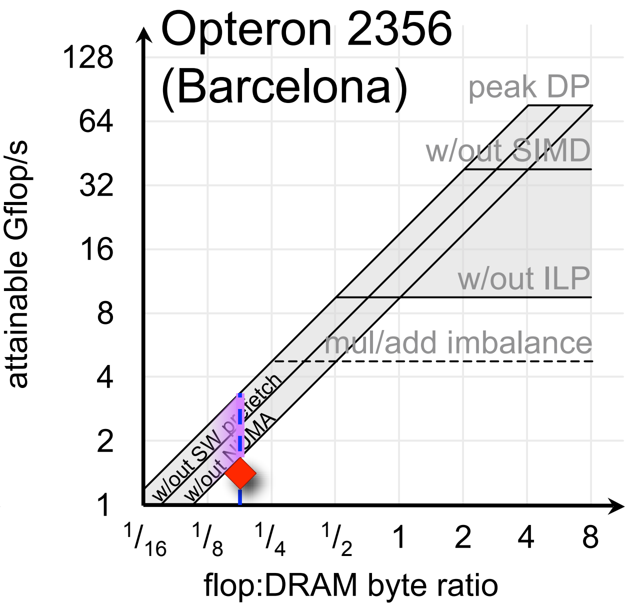
EECS
Electrical Engineering and Computer Sciences

# Roofline model for SpMV

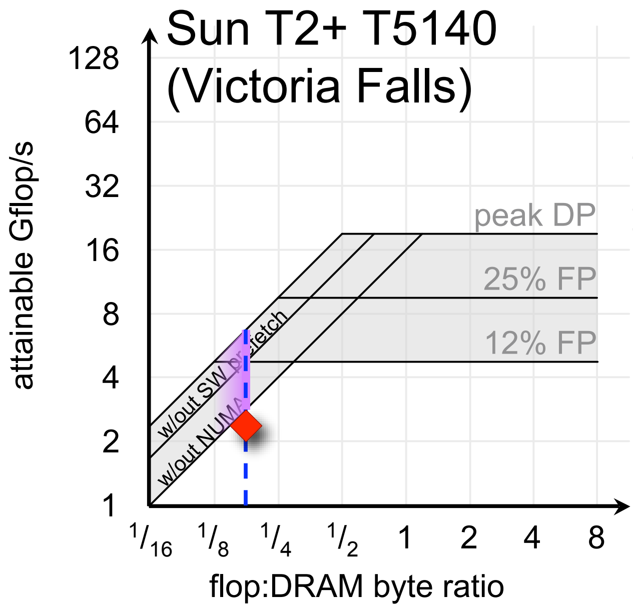
## (out-of-the-box parallel)

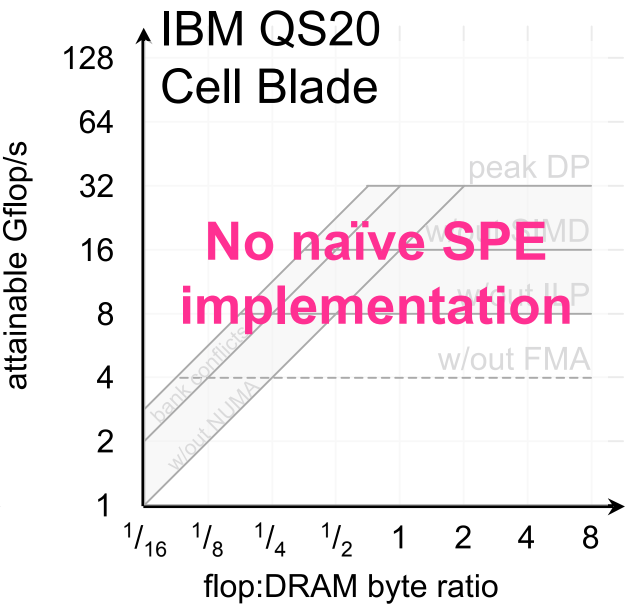
BERKELEY PAR LAB


Intel Xeon E5345 (Clovertown)


Opteron 2356 (Barcelona)

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


Sun T2+ T5140 (Victoria Falls)

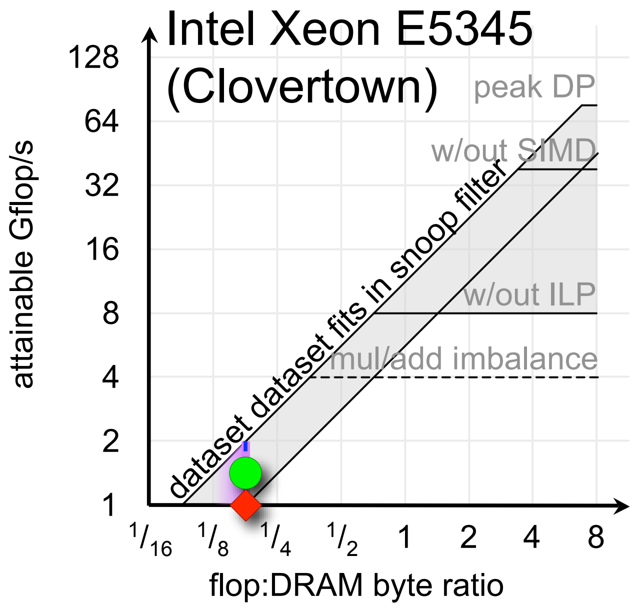

IBM QS20 Cell Blade

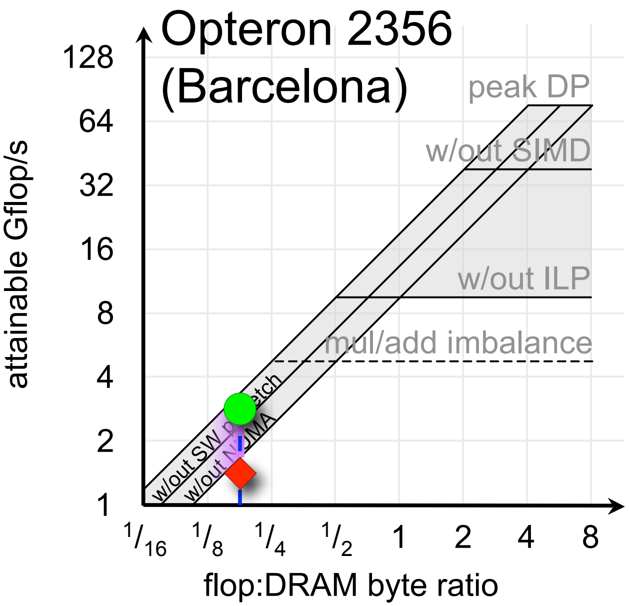
EECS
Electrical Engineering and
Computer Sciences

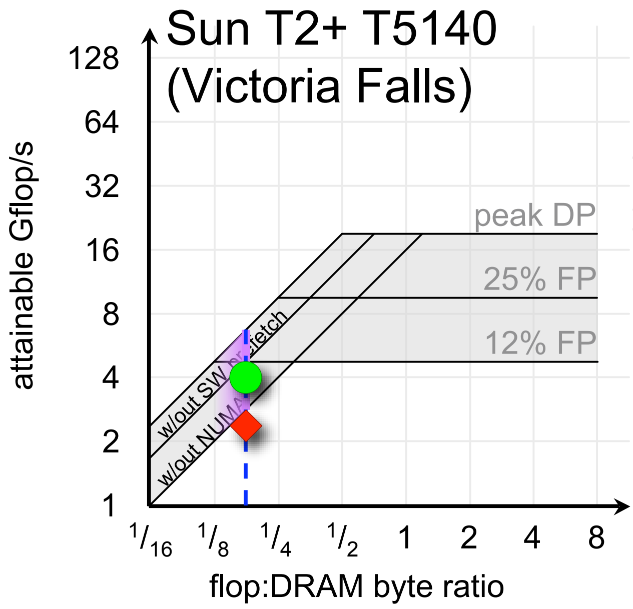
# Roofline model for SpMV

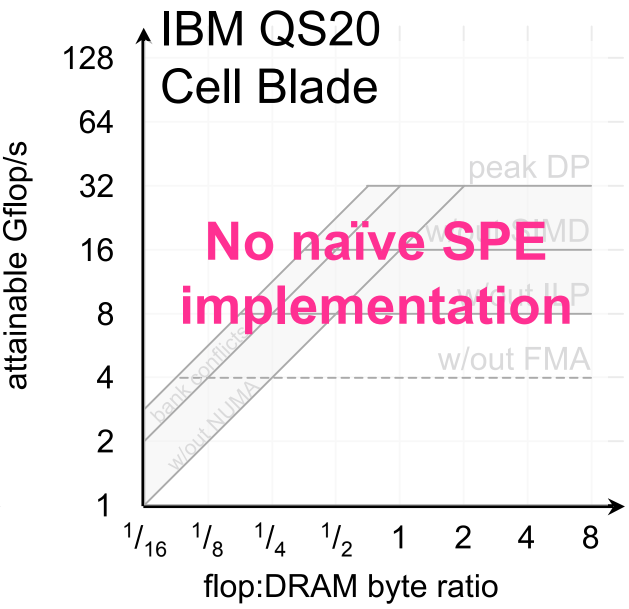
## (NUMA & SW prefetch)

BERKELEY PAR LAB









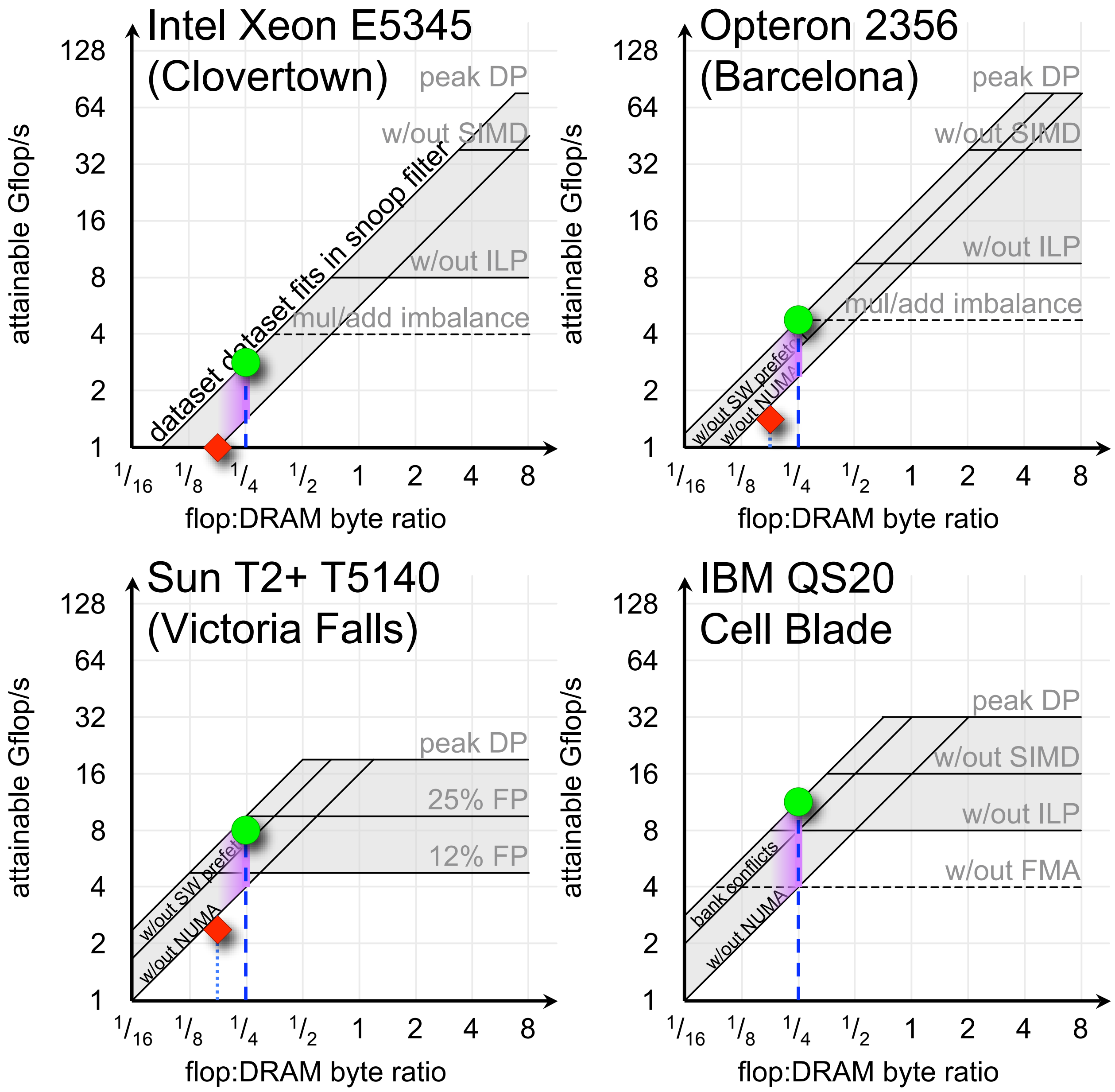
- compulsory flop:byte ~ 0.166
- utilize all memory channels

EECS
Electrical Engineering and Computer Sciences

# Roofline model for SpMV

## (matrix compression)

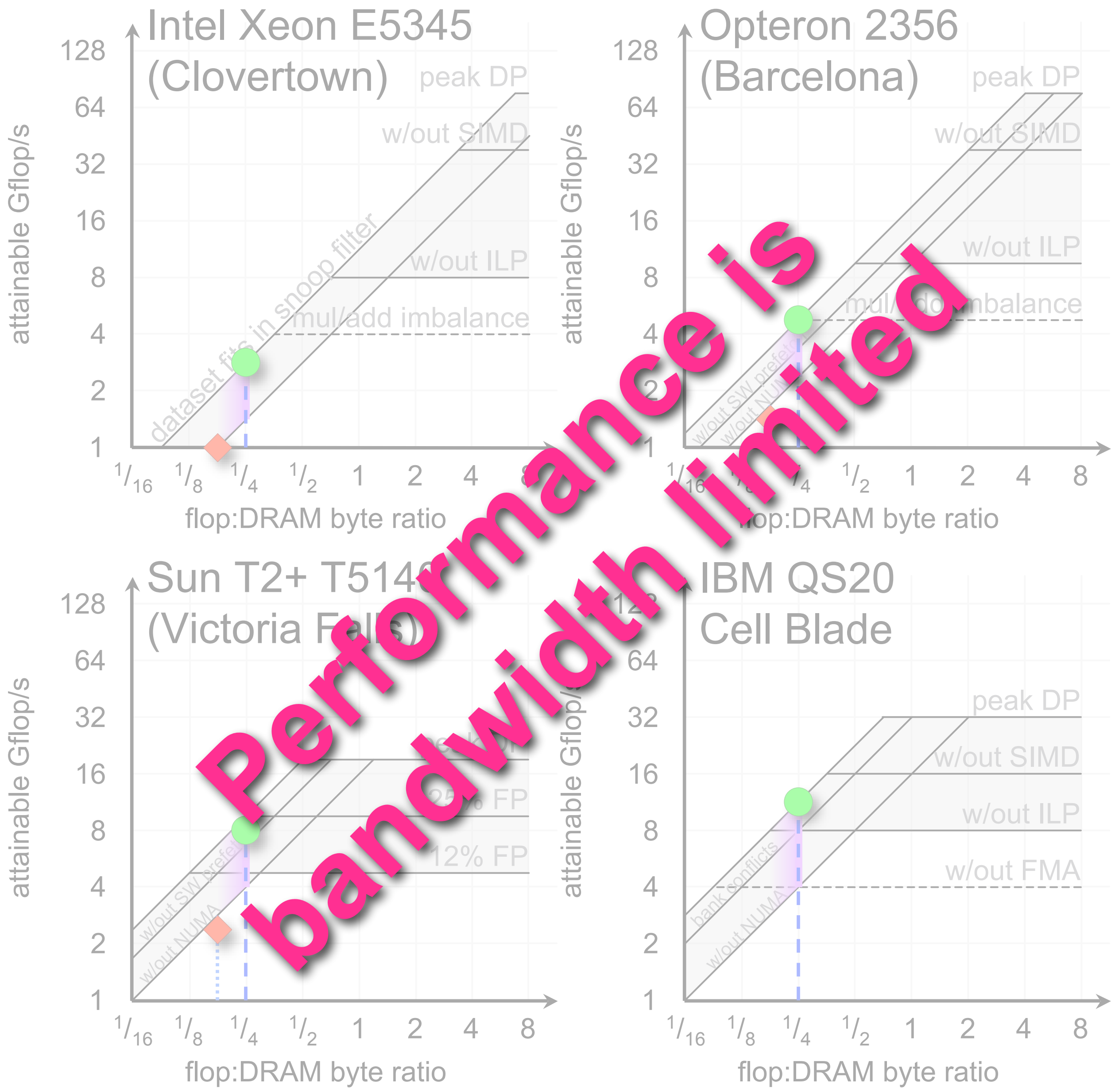
BERKELEY PAR LAB



- 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

# A Vision for BeBOP's Future Performance Counter Usage

# Deficiencies of the Roofline

- The Roofline and its ceilings are architecture specific
- They are not execution(runtime) specific

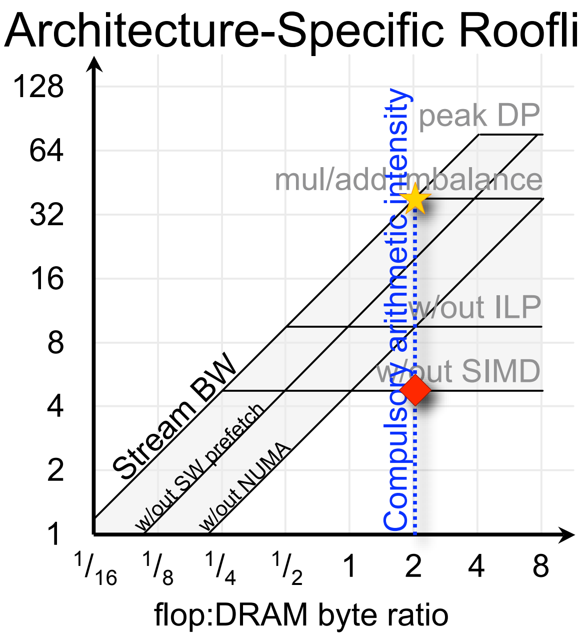
- It requires the user to calculate the true arithmetic intensity including cache conflict and capacity misses.

- Although the roofline is extremely visually intuitive, it only says what must be done by some agent (by compilers, by hand, by libraries)
- It does not state in what aspect was the code deficient

# Performance Counter Roofline

## (understanding performance)

- In the worst case, without performance counter data performance analysis can be extremely non-intuitive
- (delete the ceilings)



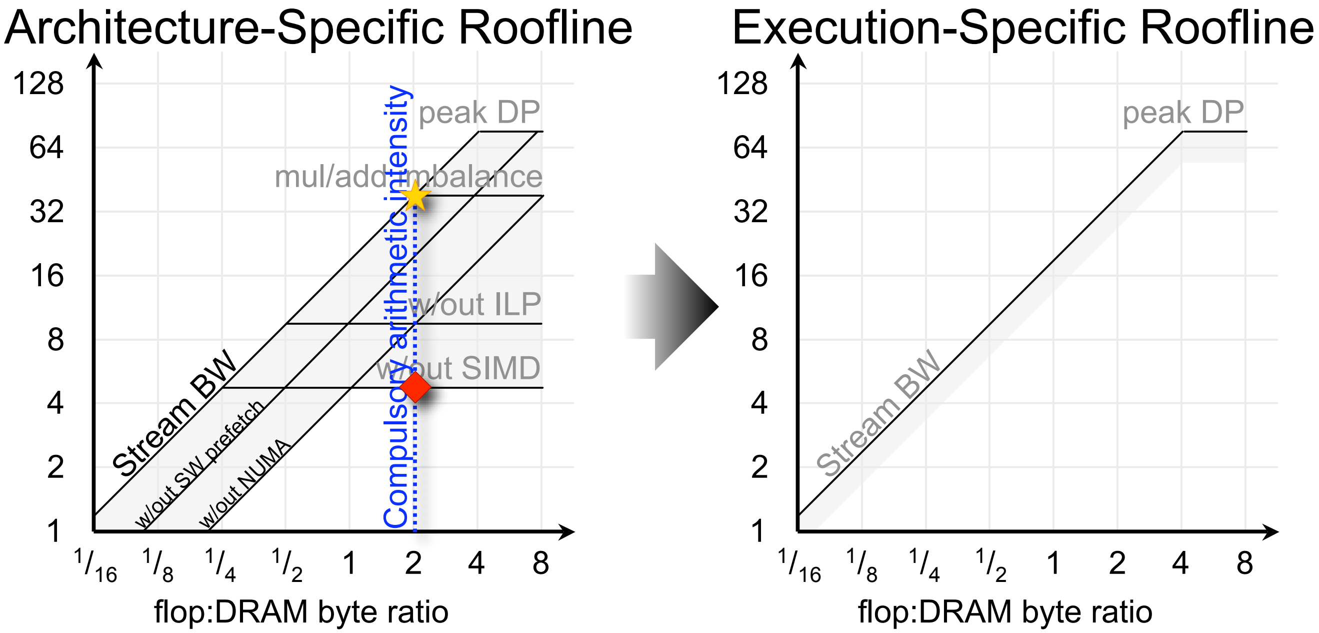
EECS
Electrical Engineering and
Computer Sciences

# Performance Counter Roofline

## (execution specific roofline)

BERKELEY PAR LAB

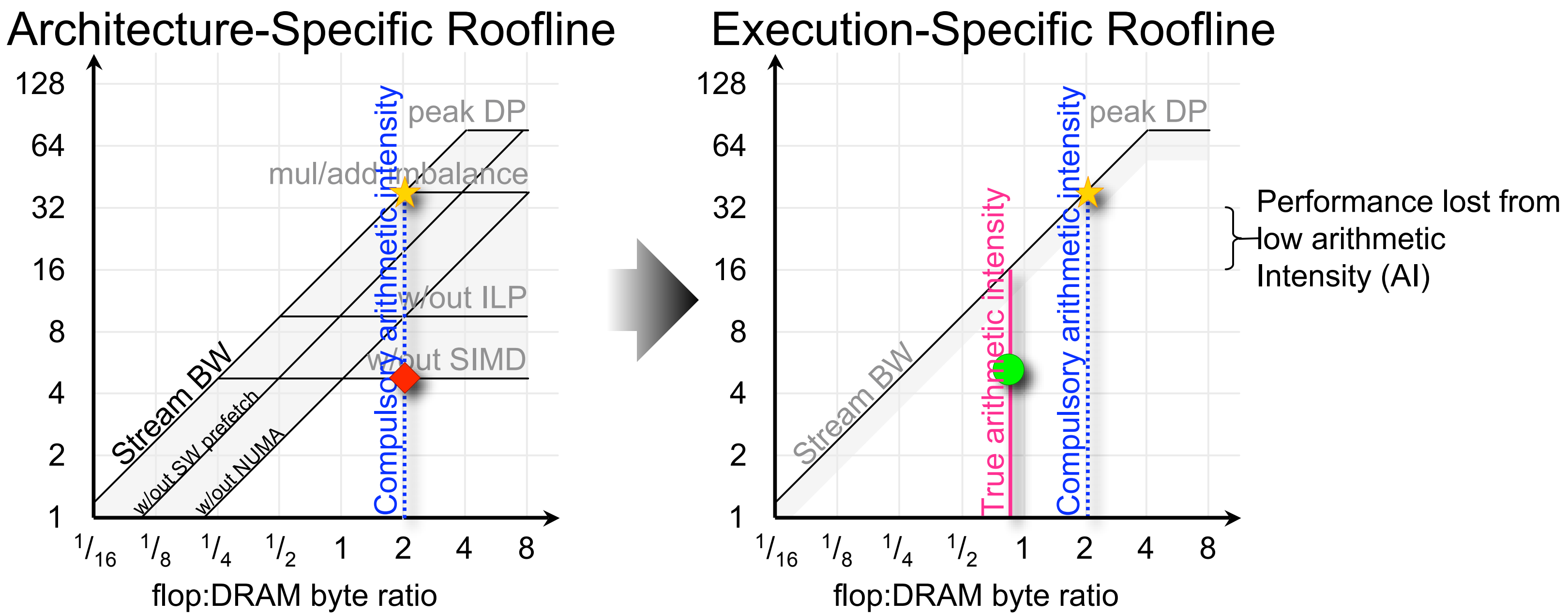
- Transition from an architecture specific roofline,
to an execution-specific roofline



# Performance Counter Roofline

## (true arithmetic intensity)

- Performance counters tell us the true memory traffic
- Algorithmic Analysis tells us the useful flops
- Combined we can calculate the true arithmetic intensity



Architecture-Specific Roofline

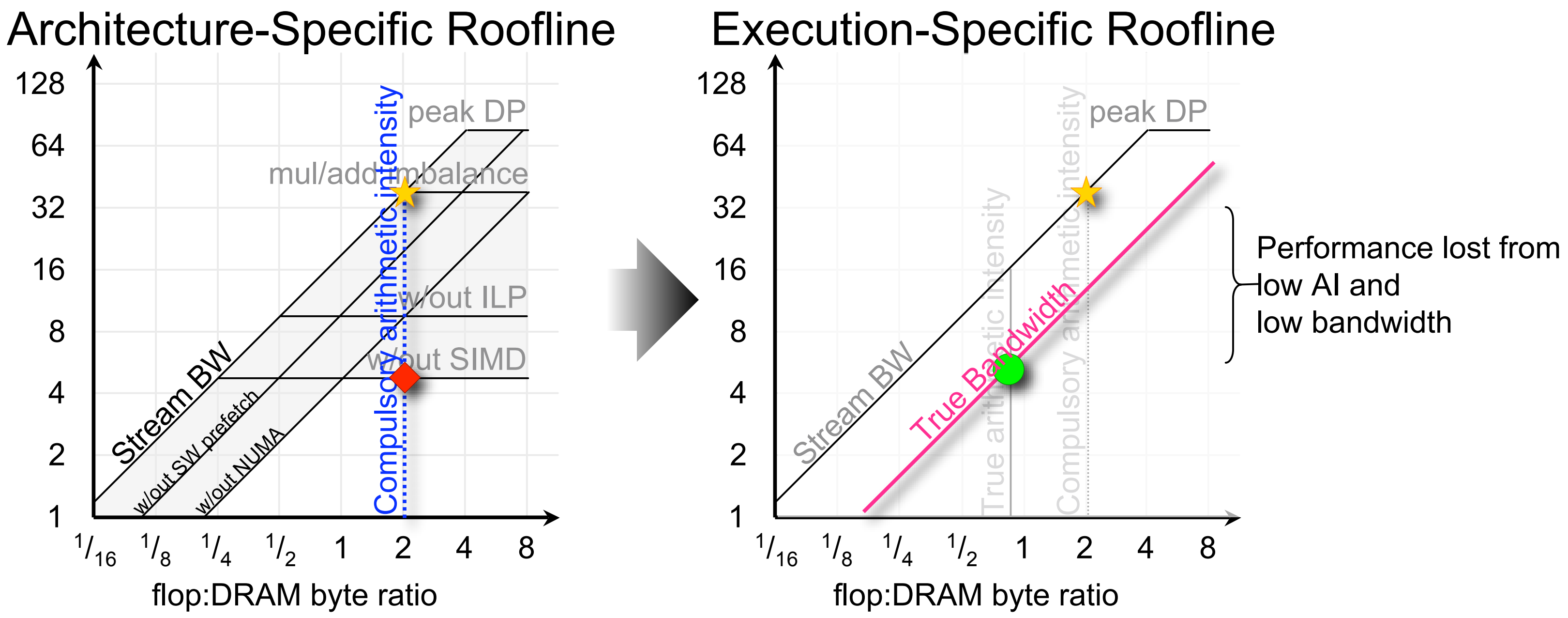
Execution-Specific Roofline

Performance lost from low arithmetic Intensity (AI)

# Performance Counter Roofline

## (true memory bandwidth)

- Given the total memory traffic and total kernel time, we may also calculate the true memory bandwidth
- Must include 3C's + speculative loads

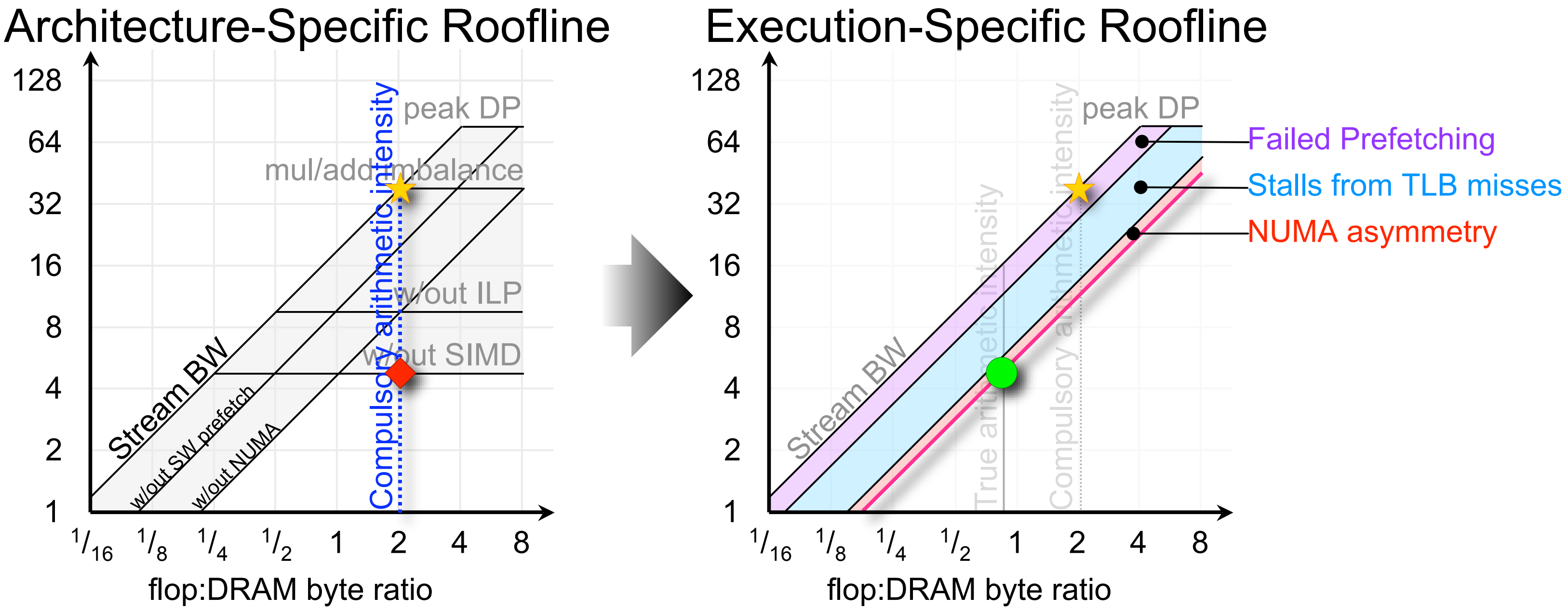


Performance lost from low AI and low bandwidth

# Performance Counter Roofline

## (bandwidth ceilings)

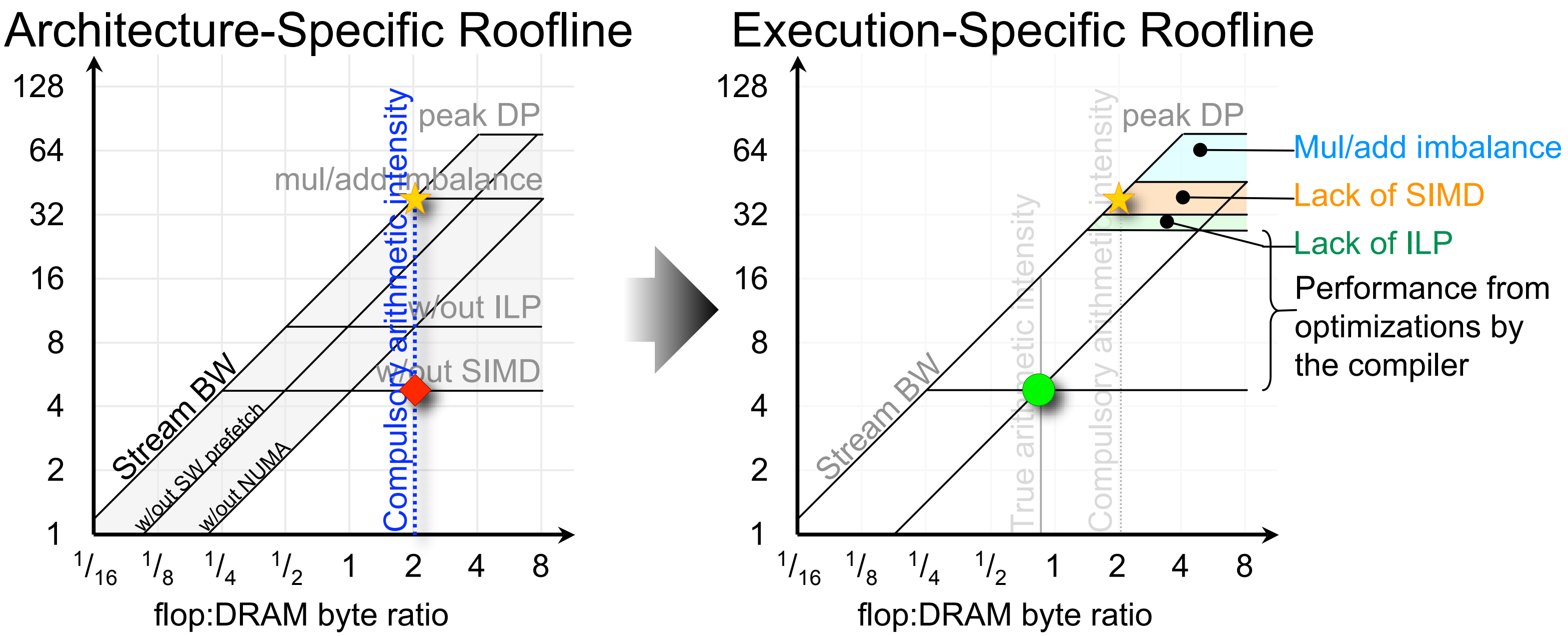
- Every idle bus cycle diminishes memory bandwidth
- Use performance counters to bin memory stall cycles



# Performance Counter Roofline

(in-core ceilings)

- Measure imbalance between FP add/mul issue rates as well as stalls from lack of ILP and ratio of scalar to SIMD instructions
- Must be modified by the compulsory work
  - e.g. placing a 0 in a SIMD register to execute the _PD form increases the SIMD rate but not the useful execution rate



# Relevance to Typical Programmer

- Visually Intuitive
- With performance counter data its clear which optimizations should be attempted and what the potential benefit is.

  (must still be familiar with possible optimizations)

# Relevance to Auto-tuning?

- Exhaustive search is intractable (search space explosion)

- Propose using performance counters to guide tuning:
  - Generate an execution-specific roofline to determine which optimization(s) should be attempted next
  - From the roofline, its clear what doesn't limit performance
  - Select the optimization that provides the largest potential gain

    e.g. bandwidth, arithmetic intensity, in-core performance
  - and iterate

# Summary

# Concluding Remarks

- Existing performance counter tools miss the bulk of programmers

- The Roofline provides a nice (albeit imperfect) approach to performance/architectural visualization
- We believe that performance counters can be used to generate execution-specific rooflines that will facilitate optimizations

- However, real applications will run concurrently with other applications sharing resouces.  This will complicate performance analysis
- next speaker…

# Acknowledgements

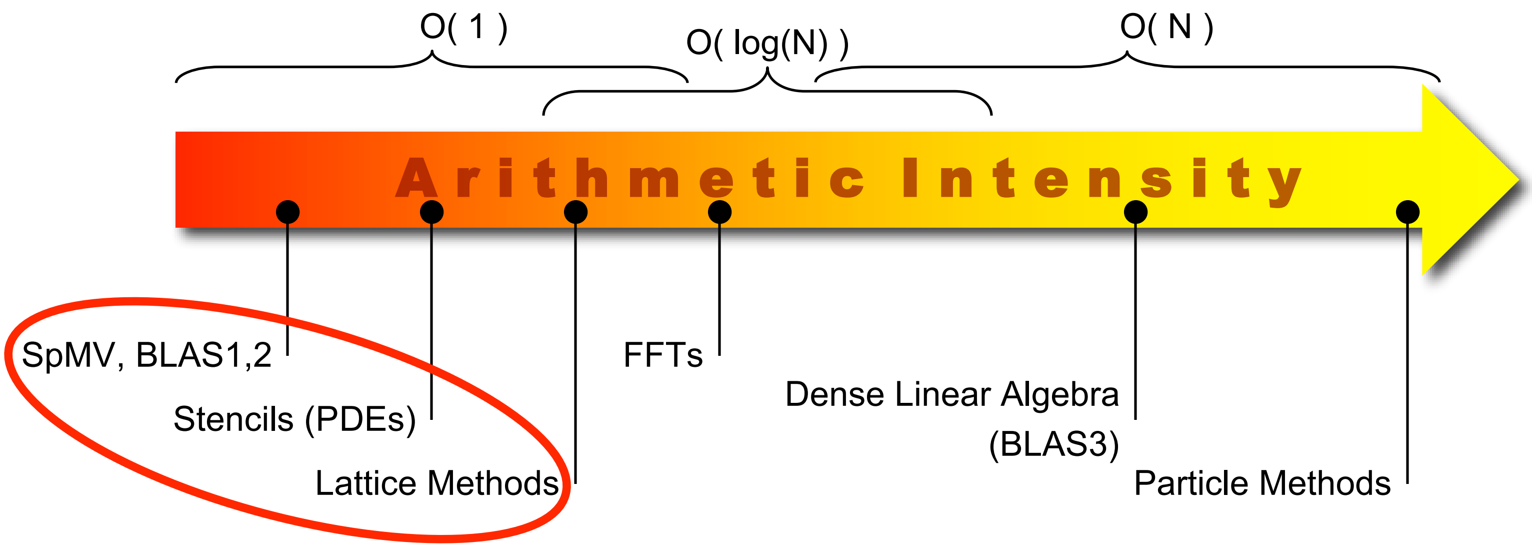
- Research supported by:
  - Microsoft and Intel funding (Award #20080469)
  - DOE Office of Science under contract number DE-AC02-05CH11231
  - NSF contract CNS-0325873
  - Sun Microsystems - Niagara2 / Victoria Falls machines
  - AMD - access to Quad-core Opteron (barcelona) access
  - Forschungszentrum Jülich - access to QS20 Cell blades
  - IBM - virtual loaner program to QS20/QS22 Cell blades

# Questions ?

# BACKUP SLIDES

# What’s a Memory Intensive Kernel?

# Arithmetic Intensity in HPC



- **True Arithmetic Intensity (AI) ~ Total Flops / Total DRAM Bytes**
- Arithmetic intensity is:
  - ultimately limited by compulsory traffic
  - diminished by conflict or capacity misses

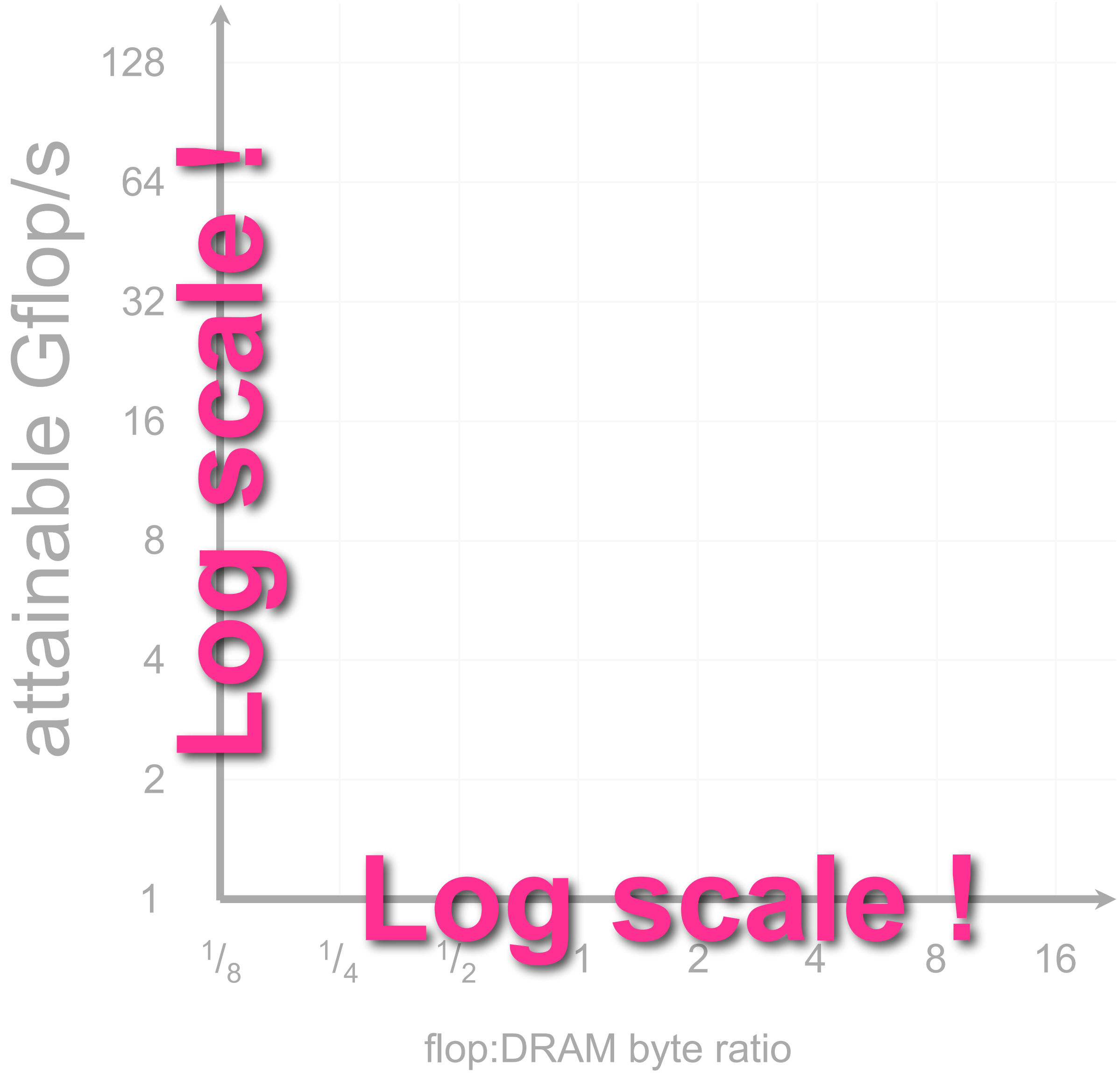
# Memory Intensive

- A kernel is memory intensive when:
  the kernel's arithmetic intensity < the machine's balance (flop:byte)

- If so, then we expect:
  **Performance ~ Stream BW * Arithmetic Intensity**

- Technology allows peak flops to improve faster than bandwidth.
  ⇨**more and more kernels will be considered memory intensive**

# The Roofline Model



# Deficiencies of Auto-tuning

- There has been an explosion in the optimization parameter space.
- Complicates the generation of kernels and their exploration

- Currently we either:
  - Exhaustively search the space (increasingly intractable)
  - Apply very high level heuristics to eliminate much of it

- Need a guided search that is cognizant of both architecture and performance counters.

# Deficiencies in usage of Performance Counters

- Only counted the number of cache/TLB misses

- We didn’t count exposed memory stalls (e.g. prefetchers)
- We didn’t count NUMA asymmetry in memory traffic
- We didn’t count coherency traffic
- Tools can be buggy or not portable
- Even worse is just giving a spread sheet filled with numbers and cryptic event names

- In-core events are less interesting as more and more kernels become memory bound