The Roofline Model:
A pedagogical tool for program analysis and optimization

ParLab Summer Retreat
Samuel Williams, David Patterson

samw@cs.berkeley.edu
Motivation

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

- Focused on:
  - rates and efficiencies (Gflop/s, % of peak),

- Goals for Roofline:
  - Provide everyone with a graphical aid that provides:
    - realistic expectations of performance and productivity
  - Show inherent hardware limitations for a given kernel
  - Show potential benefit and priority of optimizations

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

- There are three principal components to performance:
  - Computation
  - Communication
  - Locality

- Each architecture has a different balance between these
- Each kernel has a different balance between these

- Performance is a question of how well an kernel’s characteristics map to an architecture’s characteristics
For us, floating point performance (Gflop/s) is the metric of interest (typically double precision)

Peak in-core performance can only be attained if:
- fully exploit ILP, DLP, FMA, etc…
- non-FP instructions don’t sap instruction bandwidth
- threads don’t diverge (GPUs)
- transcendental/non pipelined instructions are used sparingly
- branch mispredictions are rare

To exploit a form of in-core parallelism, it must be:
- Inherent in the algorithm
- Expressed in the high level implementation
- Explicit in the generated code
Communication

- For us, DRAM bandwidth (GB/s) is the metric of interest.

- Peak bandwidth can only be attained if certain optimizations are employed:
  - Few unit stride streams
  - NUMA allocation and usage
  - SW Prefetching
  - Memory Coalescing (GPU)
Locality

- Computation is free, Communication is expensive.
- Maximize locality to minimize communication
- There is a lower limit to communication: compulsory traffic

- Hardware changes can help minimize communication
  - Larger cache capacities minimize capacity misses
  - Higher cache associativities minimize conflict misses
  - Non-allocating caches minimize compulsory traffic

- Software optimization can also help minimize communication
  - Padding avoids conflict misses
  - Blocking avoids capacity misses
  - Non-allocating stores minimize compulsory traffic
Roofline Model
Integrating Components...

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

- Roofline model will be unique to each architecture
- Coordinates of a kernel are ~unique to each architecture
What relates GB/s to GFlop/s?

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

- This is a well-known quantity: **Arithmetic Intensity** (discussed later).

- When we measure total bytes, we incorporate all cache behavior (the 3C’s) and Locality.
Performance is upper bounded by both the peak flop rate, and the product of streaming bandwidth and the flop:byte ratio.

\[ \text{Gflop/s} = \min \left\{ \text{Peak Gflop/s}, \text{Stream BW} \times \text{actual flop:byte ratio} \right\} \]
Bandwidth #’s collected via micro benchmarks

Computation #’s derived from optimization manuals (pencil and paper)

Assume complete overlap of either communication or computation
Roofline model for Opteron
(adding ceilings)

AMD Opteron 2356
(Barcelona)

- Peak roofline performance
- Based on manual for single precision peak
- And a hand tuned stream read for bandwidth

Log scale!
Roofline model for Opteron
(adding ceilings)

AMD Opteron 2356
(Barcelona)

- Peak roofline performance
- Based on manual for
  single precision peak
- And a hand tuned stream
  read for bandwidth
Roofline model for Opteron
(adding ceilings)

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

**AMD Opteron 2356 (Barcelona)**

-능력성 Gflop/s
- flop:DRAM byte ratio

- peak SP
- mul / add imbalance
- peak stream bandwidth
Roofline model for Opteron
(adding ceilings)

- In single precision, SIMD is 4x32b.
- If only the _ss versions are used, performance is 1/4 peak SP mul / add imbalance
- peak stream bandwidth w/out SIMD

AMD Opteron 2356
(Barcelona)

attainable Gflop/s

flop:DRAM byte ratio

1

2

4

8

16

1/2

1/4

1/8

8

16

32

64

128

In single precision, SIMD is 4x32b.
If only the _ss versions are used, performance is 1/4 peak SP mul / add imbalance
peak stream bandwidth w/out SIMD

1

2

4

8

16

1/2

1/4

1/8
Roofline model for Opteron
(adding ceilings)

AMD Opteron 2356
(Barcelona)

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

If 4 independent instructions are kept in the pipeline, performance will fall.
Roofline model for Opteron (adding ceilings)

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

**AMD Opteron 2356** (Barcelona)

- **peak SP**
- **mul / add imbalance**
- **w/out SIMD**
- **w/out ILP**
- **w/out SW prefetching**
- **peak stream bandwidth**

**attainable Gflop/s**

**flop:DRAM byte ratio**
Without NUMA optimizations, the memory controllers on the second socket can’t be used.
Roofline model for Opteron
(adding ceilings)

**AMD Opteron 2356 (Barcelona)**

- **Bandwidth is much lower without unit stride streams**

![Graph showing the roofline model for AMD Opteron 2356 (Barcelona). The graph illustrates the attainable Gflop/s against the flop:DRAM byte ratio, with various performance impacts such as peak SP, mul/add imbalance, peak stream bandwidth without ILP, peak stream bandwidth without SW prefetching, peak stream bandwidth without NUMA optimizations, peak stream bandwidth without SIMD, and peak stream bandwidth without unit stride streams.]
Roofline model for Opteron
(adding ceilings)

AMD Opteron 2356
(Barcelona)

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

- peak SP
- mul / add imbalance
- w/out SIMD
- w/out ILP
- raw DRAM bandwidth
- peak stream bandwidth
- w/out SW prefetching
- w/out NUMA optimizations
- w/out unit stride streams
Roofline model for Opteron
(adding ceilings)

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

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

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

- Remember the 3C’s of caches

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

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

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

AMD Opteron 2356
(Barcelona)

attainable Gflop/s

peak SP

mul / add imbalance

w/out SIMD

w/out ILP

w/out SW prefetching

w/out NUMA optimizations

peak stream bandwidth

flop:DRAM byte ratio

Final Roofline

1
1/2
1/4
1/8
Some arbitrary kernel has a flop:compulsory byte ratio of 4

- Overlaid on the roofline
- Defines upper bound on range of expected performance
- Also shows which optimizations are likely
Roofline model for Opteron
(powerpoint doodle)

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

**AMD Opteron 2356 (Barcelona)**

- Peak SP
- Mul / add imbalance
- W/out SIMD
- W/out ILP
- Compulsory misses
- Capacity
- W/out NUMA optimizations
- W/out SW prefetching
- W/out stream bandwidth
- W/out peak stream bandwidth

**attainable Gflop/s**

**flop:DRAM byte ratio**
Roofline model for Opteron
(powerpoint doodle)

- Conflict misses may destroy performance
- AI is unique to each combination of kernel and architecture
Roofline model for Opteron
(powerpoint doodle)

AMD Opteron 2356
(Barcelona)

Conflict misses may destroy performance

Doubling cache capacity doesn’t double the arithmetic intensity!
Three Categories of Software Optimization
Maximizing Attained in-core Performance

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

**AMD Opteron 2356 (Barcelona)**

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

![Graph showing the relationship between attainable Gflop/s and flop:DRAM byte ratio.](image)
Minimizing Memory Traffic

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

- Before optimization, traffic, and limited bandwidth optimization limits performance to a very narrow window
After optimization, ideally, performance is significantly better.
Applicable to Other Architectural Paradigms?
Four Architectures

**AMD Barcelona**
- Opteron
- 512KB victim
- 2MB Shared quasi-victim (32 way)
- 667MHz DDR2 DIMMs
- 2x64b memory controllers

**IBM Cell Blade**
- VMT PPE
- 512KB L2
- EIB (ring network)
- XDR memory controllers
- 512MB XDR DRAM

**Sun Victoria Falls**
- MT SPARC
- Crossbar
- 179 GB/s
- 190 GB/s
- 4MB Shared L2 (16 way) (64b interleaved)
- 4 Coherency Hubs
- 2x128b controllers

**NVIDIA G80**
- Thread Cluster
- Thread Cluster
- Thread Cluster
- Thread Cluster
- 192KB L2 (Textures only)
- 24 ROPs
- 6 x 64b memory controllers
- 768MB 900MHz GDDR3 Device DRAM

---

One Thread / Core

Multithreaded Cores
Four Architectures

**AMD Barcelona**
- Opteron 512KB victim
- 2MB Shared quasi-victim (32 way)
- 2x64b memory controllers
- HyperTransport
- 667MHz DDR2 DIMMs

**Sun Victoria Falls**
- Opteron 512KB victim
- 2MB Shared quasi-victim (32 way)
- 2x64b memory controllers
- HyperTransport
- 667MHz FBDIMMs

**IBM Cell Blade**
- VMT PPE 512K L2
- EIB (ring network)
- XDR memory controllers
- 512MB XDR DRAM

**NVIDIA G80**
- Thread Cluster
- 192KB L2 (Textures only)
- 24 ROPs
- 6 x 64b memory controllers
- 768MB 900MHz GDDR3 Device DRAM
32b Rooflines for the Four
(in-core parallelism)

- Single Precision Roofline models for the SMPs used in this work.
- Based on micro-benchmarks, experience, and manuals.
- Ceilings = in-core parallelism
- Can the compiler find all this parallelism?
- NOTE:
  - log-log scale
  - Assumes perfect SPMD
32b Rooflines for the Four
(diverged threads)

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

- **Sun Victoria Falls**
  - G80 dynamically finds DLP (shared instruction fetch)
  - SIMT
  - If threads of a warp diverge from SIMD execution, performance is limited by instruction issue bandwidth
  - Ceilings on G80 = number of unique PCs when threads diverge
Some kernels have large numbers of non FP instructions

Saps instruction issue bandwidth

Ceilings = FP fraction of dynamic instruction mix

NOTE:
- Assumes perfect in-core parallelism
Some architectures have drastically different ridge points
VF may be compute bound on many kernels
Clovertown has $\frac{1}{3}$ the BW of Barcelona = ridge point to the right
Using Roofline when Auto-tuning HPC Kernels
Multicore SMPs Used

Intel Xeon E5345 (Clovertown)

AMD Opteron 2356 (Barcelona)

Sun T2+ T5140 (Victoria Falls)

IBM QS20 Cell Blade
Sparse Matrix-Vector Multiplication (SpMV)

Sparse Matrix Vector Multiplication

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

- Evaluate $y = Ax$
  - $A$ is a sparse matrix
  - $x$ & $y$ are dense vectors

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

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

Clovertown

- peak DP
- w/out SIMD
- w/out ILP
- mull/add imbalance
- fits within snoop filter

Barcelona

- peak DP
- w/out SIMD
- w/out ILP
- mull/add imbalance

Victoria Falls

- peak DP
- 25% FP
- 12% FP

Cell Blade

- peak DP
- w/out SIMD
- w/out ILP
- w/out FMA

No naïve Cell implementation
Roofline model for SpMV

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

- **Barcelona**
  - Mul/add imbalance
  - Fits within snoop filter

- **Victoria Falls**
  - 25% FP
  - 12% FP

- **Cell Blade**
  - No naïve Cell implementation

- **Cell Blade**
  - No SW prefetch
  - No NUMA
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
Lattice-Boltzmann Magneto-Hydrodynamics (LBMHD)


Best Paper, Application Track
Plasma turbulence simulation via Lattice Boltzmann Method

Two distributions:
- momentum distribution (27 scalar components)
- magnetic distribution (15 vector components)

Three macroscopic quantities:
- Density
- Momentum (vector)
- Magnetic Field (vector)

Must read 73 doubles, and update 79 doubles per point in space
Requires about 1300 floating point operations per point in space
Just over 1.0 flops/byte (ideal)
**Roofline model for LBMHD**

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

- **Barcelona**
  - Fits within snoop filter

- **Victoria Falls**
  - peak DP
  - mul/add imbalance
  - w/out SIMD
  - w/out ILP
  - w/out NUMA
  - w/out SW prefetch
  - w/out unaligned DMA
  - 25% FP
  - 12% FP

- **Cell Blade**
  - peak DP
  - mul/add imbalance
  - w/out SIMD
  - w/out ILP
  - w/out NUMA
  - w/out SW prefetch
  - w/out NUMA
  - No naïve Cell implementation
Roofline model for LBMHD
(out-of-the-box code)

- Huge datasets
- NUMA allocation/access
- Little ILP
- No DLP
- Far more adds than multiplies (imbalance)
- Essentially random access to memory
- Flop:byte ratio ~0.7
- High conflict misses
- Peak VF performance with 64 threads (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
- Clovrtown pinned to lower BW ceiling

Clovrtown:
- Fits within snoop filter
- w/out SIMD
- w/out ILP
- peak DP
- mul/add imbalance
- attainable Gflop/s
- flop:DRAM byte ratio

Barcelona:
- Fits within snoop filter
- w/out SIMD
- w/out ILP
- peak DP
- mul/add imbalance
- attainable Gflop/s
- flop:DRAM byte ratio

Victoria Falls:
- 25% FP
- 12% FP
- w/out SW prefetch
- w/out NUMA

Cell Blade:
- No naïve Cell implementation
- w/out SIMD
- w/out ILP
- mul/add imbalance
- attainable Gflop/s
- flop:DRAM byte ratio

flop:DRAM byte ratio
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
The Heat Equation Stencil

The Heat Equation Stencil

- Explicit Heat equation on a regular grid
- Jacobi
- One double per point in space
- 7-point nearest neighbor stencil
- Must:
  - read every point from DRAM
  - perform 8 flops (linear combination)
  - write every point back to DRAM
- Just over 0.5 flops/byte (ideal)
- Cache locality is important
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 $\frac{1}{3}$
- High locality requirements
- Capacity and conflict misses will severely impair flop:byte ratio

Clovertown

- peak DP
- mul/add imbalance
- w/out SIMD
- w/out ILP
- fits within snoop filter

Barcelona

- peak DP
- mul/add imbalance
- w/out SIMD
- w/out ILP
- w/out NUMA

Victoria Falls

- peak DP
- 25% FP
- 12% FP
- w/out SW prefetch
- w/out NUMA

Cell Blade

- peak DP
- w/out SIMD
- w/out ILP
- mul/add imbalance
- w/out SW prefetch
- w/out NUMA

No naive Cell implementation
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 $\frac{1}{3}$
- High locality requirements
- Capacity and conflict misses will severely impair flop:byte ratio

Clovertown

- peak DP
- mul/add imbalance
- w/out SIMD
- w/out ILP
- fits within snoop filter

Barcelona

- peak DP
- mul/add imbalance
- w/out SIMD
- w/out ILP

Victoria Falls

- peak DP
- 25% FP
- w/out SW prefetch
- w/out NUMA

Cell Blade

- peak DP
- 12% FP
- w/out ILP
- w/out SIMD

No naïve Cell implementation
Roofline model for Stencil
(NUMA, cache blocking, unrolling, prefetch, …)

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

Clovertown

-peak DP
-mul/add imbalance
-w/out SIMD
-w/out ILP

fits within snoop filter

Barcelona

-peak DP
-mul/add imbalance
-w/out SIMD
-w/out ILP

Cell Blade

No naïve Cell implementation

Victoria Falls

-peak DP
-25% FP
-12% FP

fits within snoop filter

Cell Blade

-peak DP
-w/out SIMD
-w/out ILP

w/out NUMA
-unaligned DMA

w/out SIMD
-w/out ILP

mul/add imbalance

w/out SW prefetch
-w/out NUMA

w/out SW prefetch
-w/out NUMA

w/out NUMA
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
Refining the Roofline
Other performance metrics

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

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

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

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

\[
\{\text{Gflop/s, GIPS, crypto, ...}\} \times \{\text{L2, DRAM, PCIe, Network}\}
\]
Conclusions

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

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

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

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

Graph showing trends in performance metrics, including:
- Flop:DRAM byte ratio
- % of peak Gflop/s
- Mul/add imbalance
- Without ILP and without SIMD
- Peak stream bandwidth with memory optimizations

Key points:
- ILP is decreasing
- SIMD is becoming wider
- Bandwidth isn’t keeping up with #cores
- Application flop:byte is decreasing
- Cache/Local Store management is becoming critical
Either of the axes could be swapped to display the formalism of the graph.

- Same formalism can be plotted on different axis.
- Difficult to plot AI.
No Overlap

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

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

AMD Opteron (rev.F)

peak DP

mul / add imbalance

peak stream bandwidth
w/out SW prefetching

w/out NUMA optimizations

w/out ILP or SIMD

attainable Gflop/s

flop:DRAM byte ratio
Auto-tuned Performance
(+Cell/SPE version)

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

Intel Xeon (Clovertown)

AMD Opteron (rev.F)

Sun Niagara2 (Huron)

IBM Cell Blade (SPEs)

- More DIMMs(opteron), FW fix, array padding(N2), etc...
- Cache/TLB Blocking
- Compression
- SW Prefetching
- NUMA/Affinity
- Naïve Pthreads
- Naïve
Auto-tuned Performance
(Local Store Implementation)

- First attempt at cell implementation.
- VL, unrolling, reordering fixed
- No NUMA
- Exploits DMA and double buffering to load vectors
- Straight to SIMD intrinsics.
- Despite the relative performance, Cell’s DP implementation severely impairs performance
### Where do GPUs fit in?

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

<table>
<thead>
<tr>
<th>Instruction Level Parallelism</th>
<th>Expressed at compile time</th>
<th>Discovered at run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLIW</td>
<td>superscalar, SMT, etc…</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Level Parallelism</th>
<th></th>
<th>G80</th>
</tr>
</thead>
</table>