Autotuning Sparse Matrix and Structured Grid Kernels

Samuel Williams[1,2], Richard Vuduc[3], Leonid Oliker[1,2], John Shalf[2], Katherine Yelick[1,2], James Demmel[1,2], Jonathan Carter[2], David Patterson[1,2]

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

samw@cs.berkeley.edu

Overview

- Multicore is the de facto performance solution for the next decade

- Examined Sparse Matrix Vector Multiplication (SpMV) kernel
  - Important, common, memory intensive, HPC kernel
  - Present 2 autotuned threaded implementations
  - Compare with leading MPI implementation(PETSc) with an autotuned serial kernel (OSKI)

- Examined Lattice-Boltzmann Magneto-hydrodynamic (LBMHD) application
  - memory intensive HPC application (structured grid)
  - Present 2 autotuned threaded implementations

- Benchmarked performance across 4 diverse multicore architectures
  - Intel Xeon (Clovertown)
  - AMD Opteron
  - Sun Niagara2 (Huron)
  - IBM QS20 Cell Blade

- We show
  - Cell consistently delivers good performance and efficiency
  - Niagara2 delivers good performance and productivity

Autotuning

Autotuning

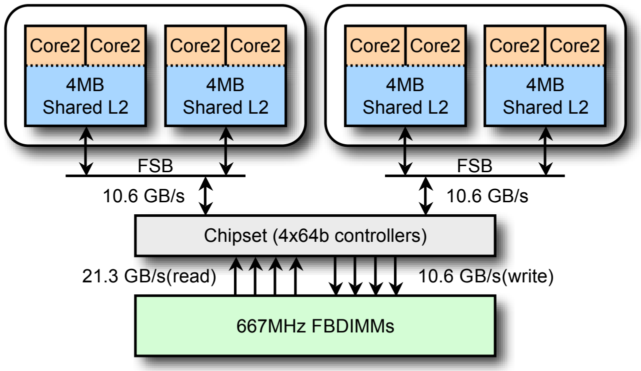
- Hand optimizing each architecture/dataset combination is not feasible
- Autotuning finds a good performance solution be heuristics or exhaustive search
  - Perl script generates many possible kernels
  - Generate SSE optimized kernels
  - Autotuning benchmark examines kernels and reports back with the best one for the current architecture/dataset/compiler/…
  - Performance depends on the optimizations generated
  - Heuristics are often desirable when the search space isn’t tractable

Autotuning
Multicore SMPs
HPC Kernels
SpMV
LBMHD
Summary

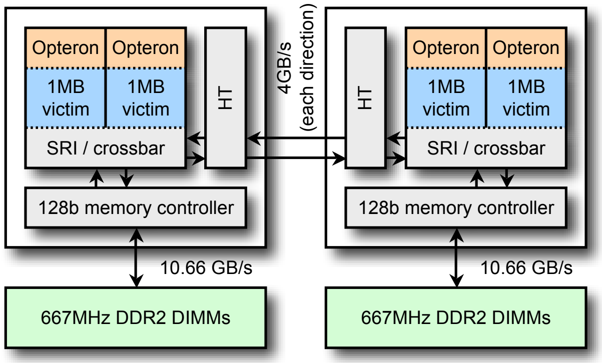
Multicore SMPs used

Multicore SMP Systems

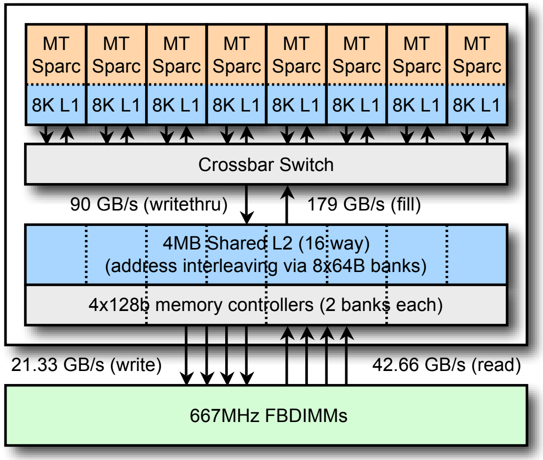
Intel Clovertown



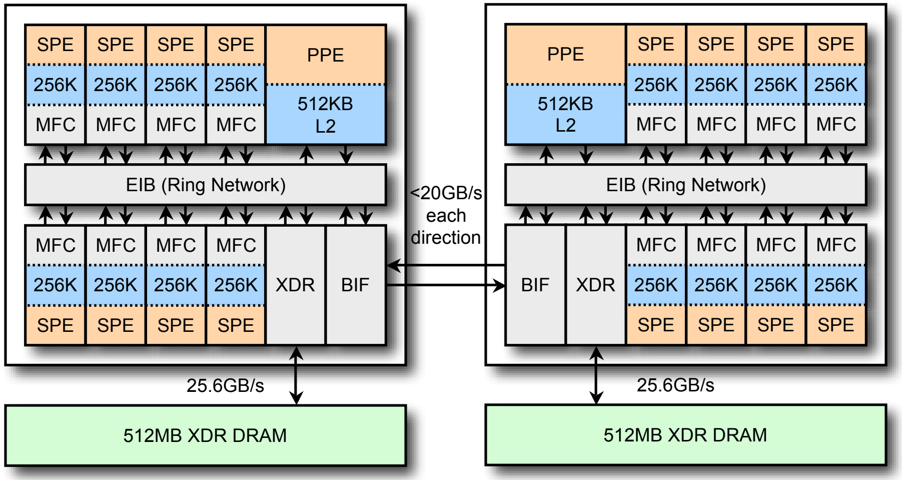
AMD Opteron



Sun Niagara2 (Huron)

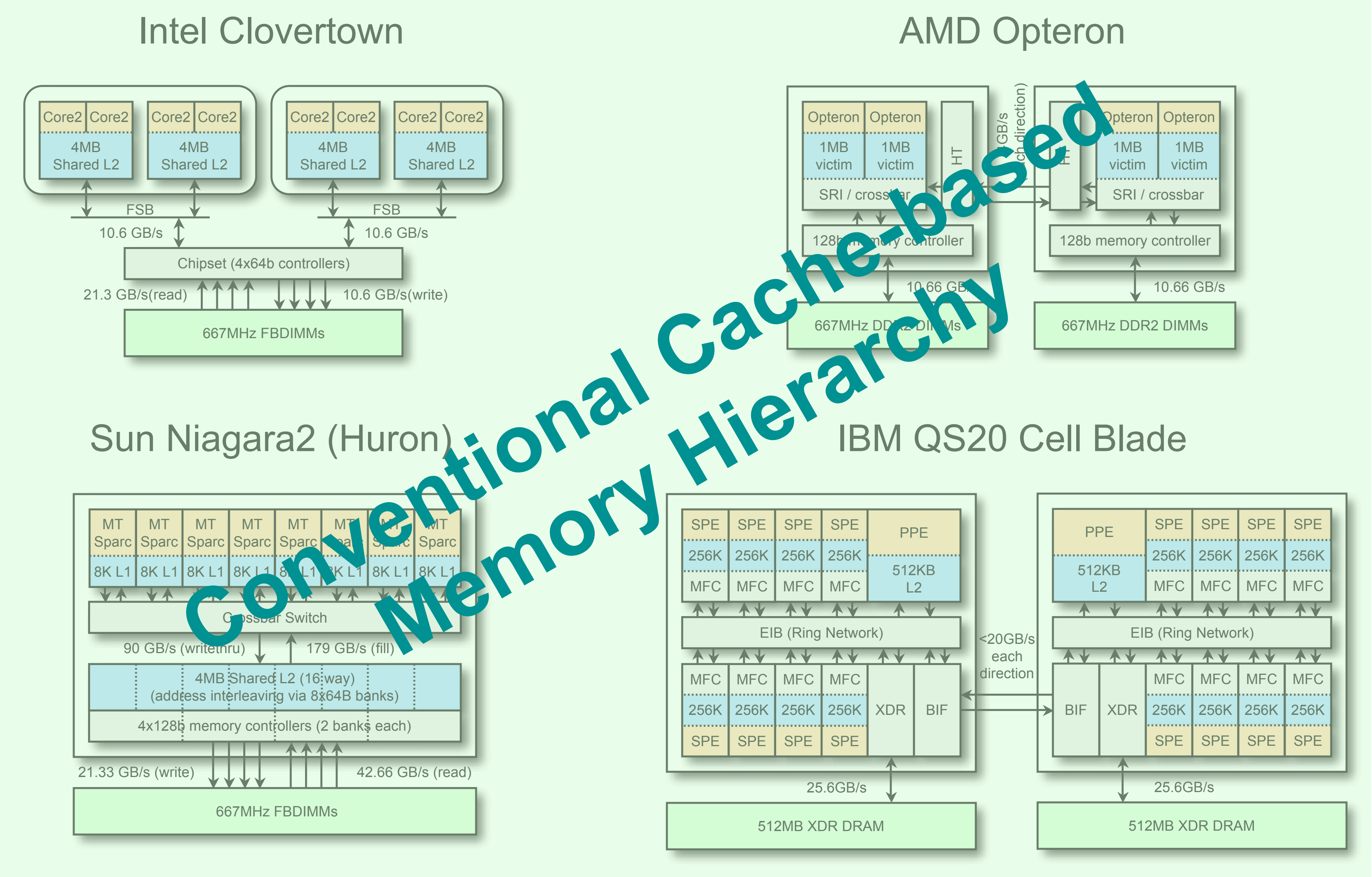


IBM QS20 Cell Blade



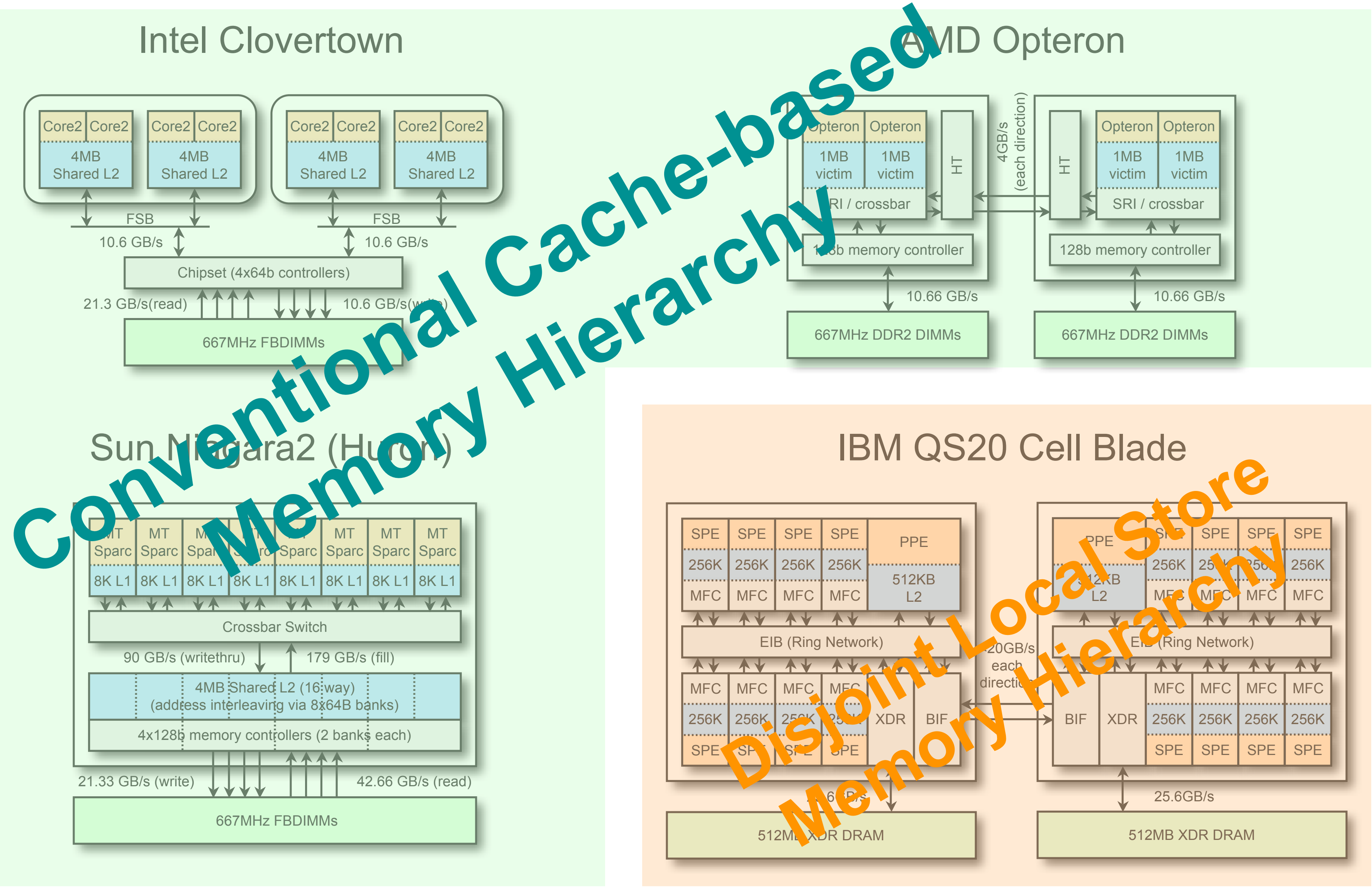
Multicore SMP Systems

(memory hierarchy)



Conventional Cache-based Memory Hierarchy

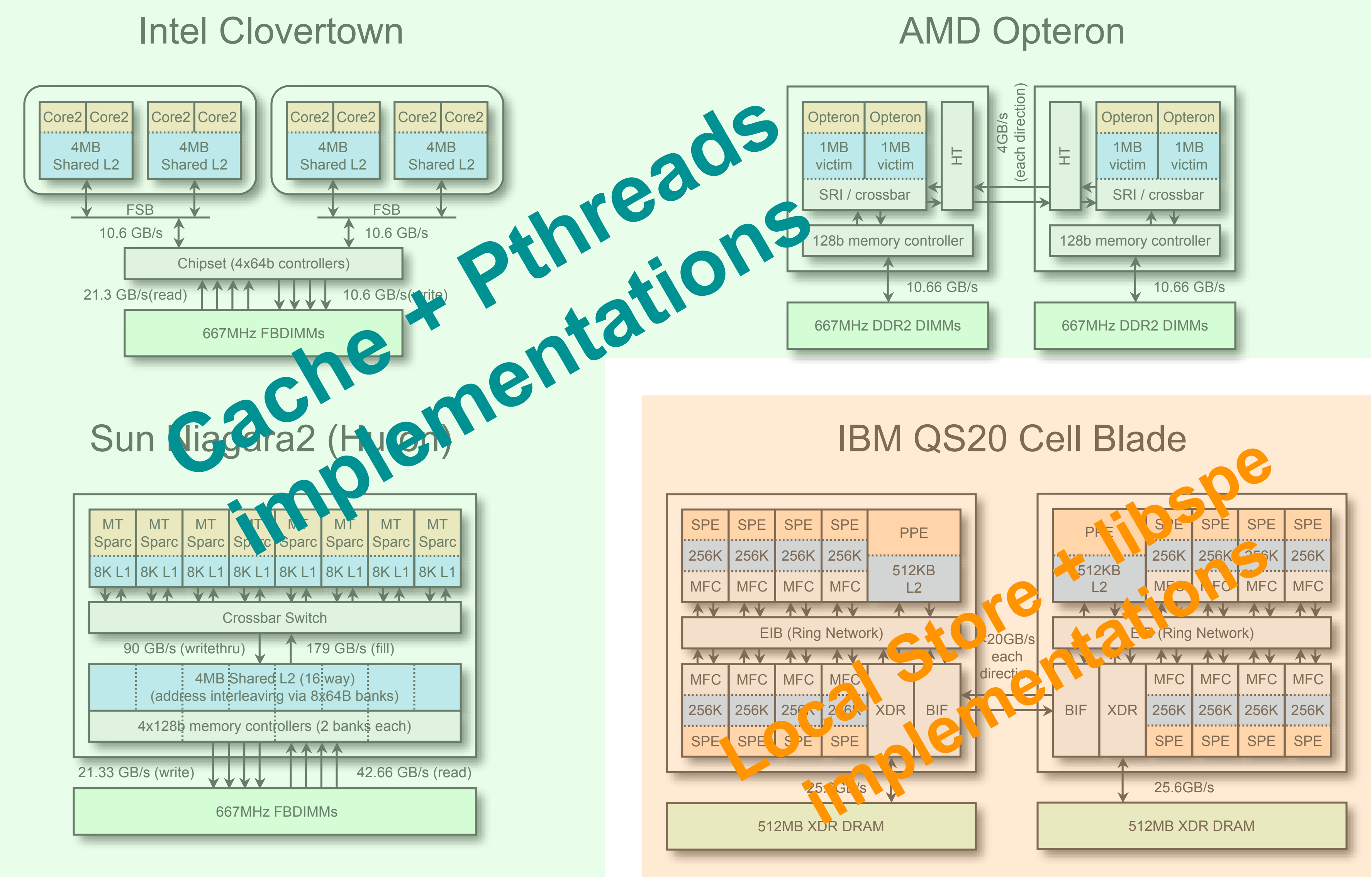
Multicore SMP Systems
(memory hierarchy)



Conventional Cache-based Memory Hierarchy

Disjoint Local Store Memory Hierarchy

Multicore SMP Systems
(memory hierarchy)



Intel Clovertown

AMD Opteron

Sun Niagara2 (Huron)

IBM QS20 Cell Blade

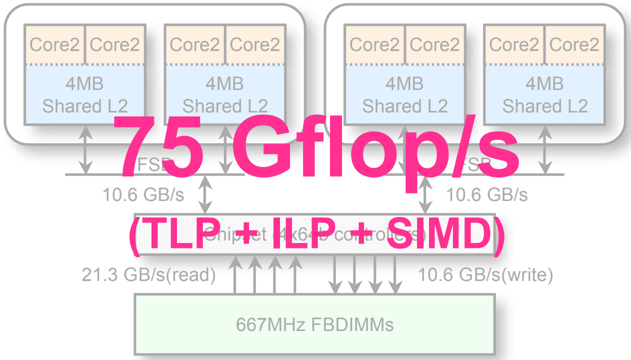
Cache + Pthreads implementations

Local Store + libspe implementations

Multicore SMP Systems

(peak flops)

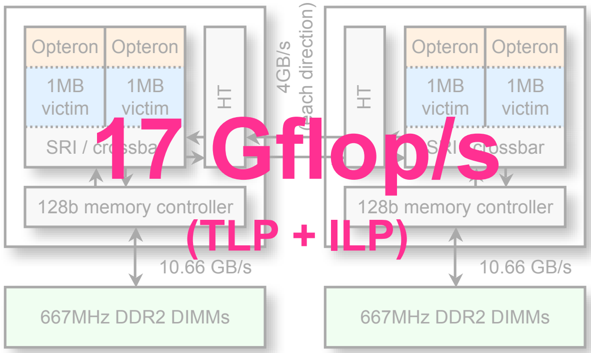
Intel Clovertown



75 Gflop/s

(TLP + ILP + SIMD)

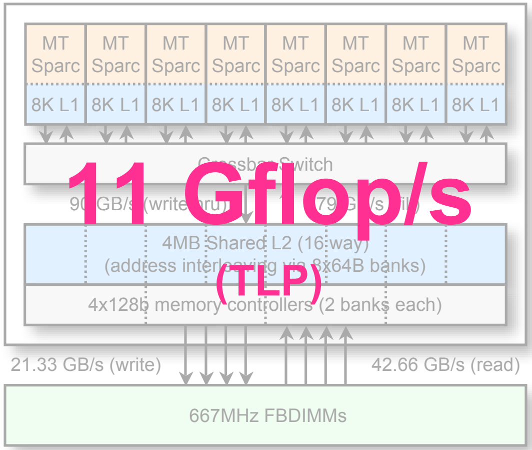
AMD Opteron



17 Gflop/s

(TLP + ILP)

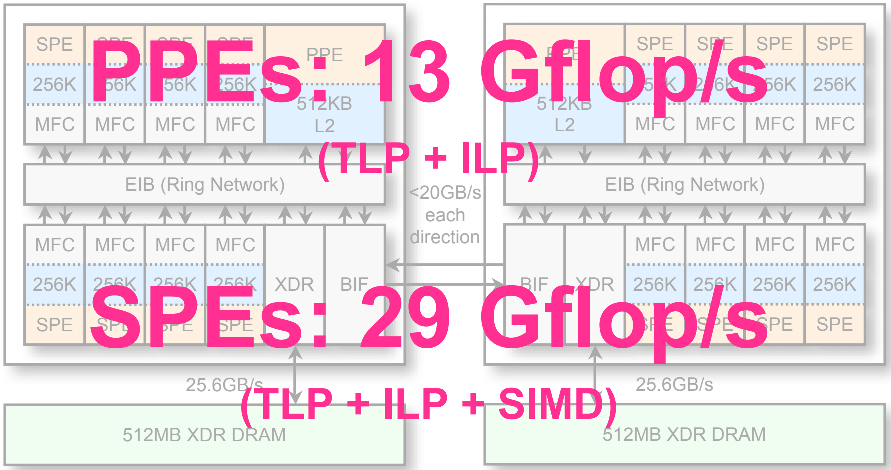
Sun Niagara2 (Huron)



11 Gflop/s

(TLP)

IBM QS20 Cell Blade



PPEs: 13 Gflop/s

(TLP + ILP)

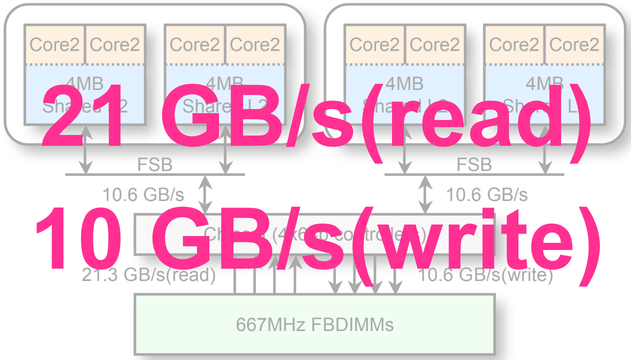
SPEs: 29 Gflop/s

(TLP + ILP + SIMD)

Multicore SMP Systems

(peak DRAM bandwidth)

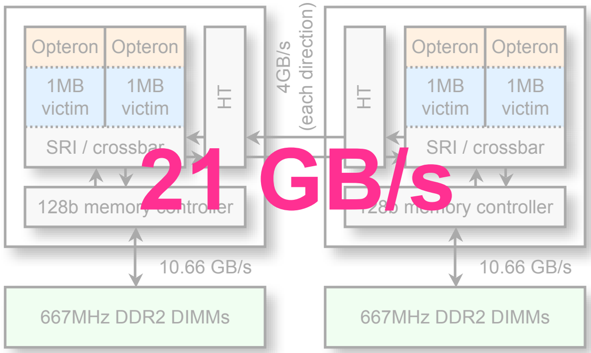
Intel Clovertown



21 GB/s(read)

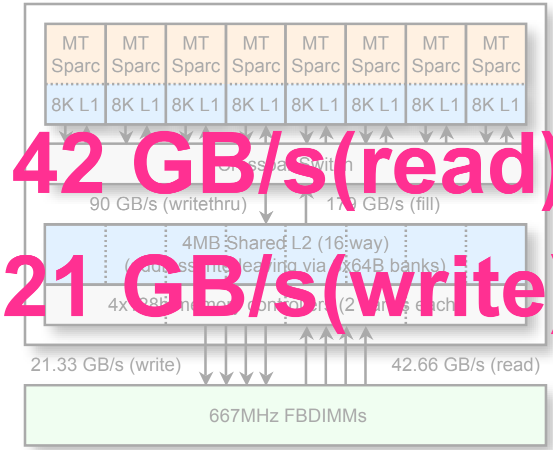
10 GB/s(write)

AMD Opteron



21 GB/s

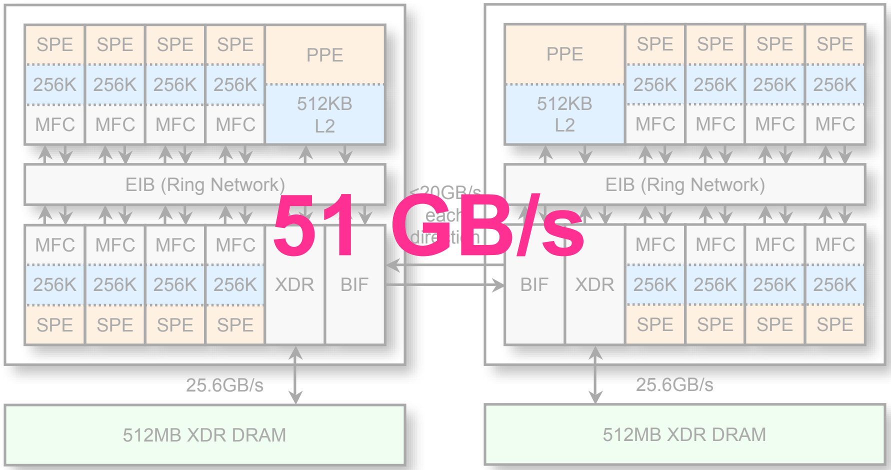
Sun Niagara2 (Huron)



42 GB/s(read)

21 GB/s(write)

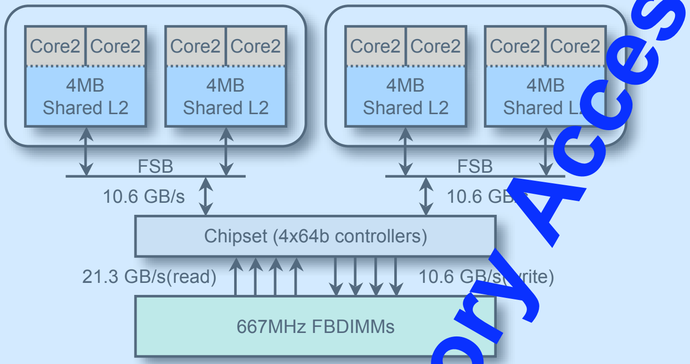
IBM QS20 Cell Blade



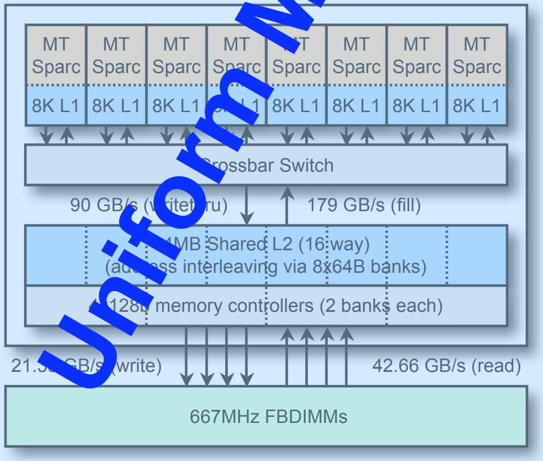
51 GB/s

Multicore SMP Systems

Intel Clovertown

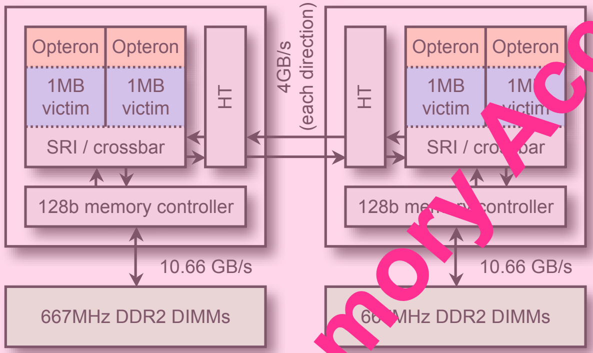


Sun Niagara2 (Huron)

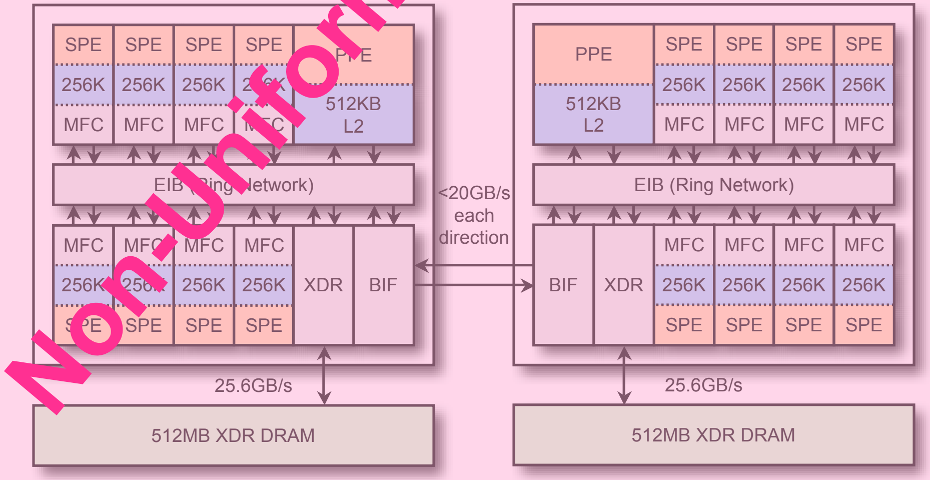


Uniform Memory Access

AMD Opteron



IBM QS20 Cell Blade

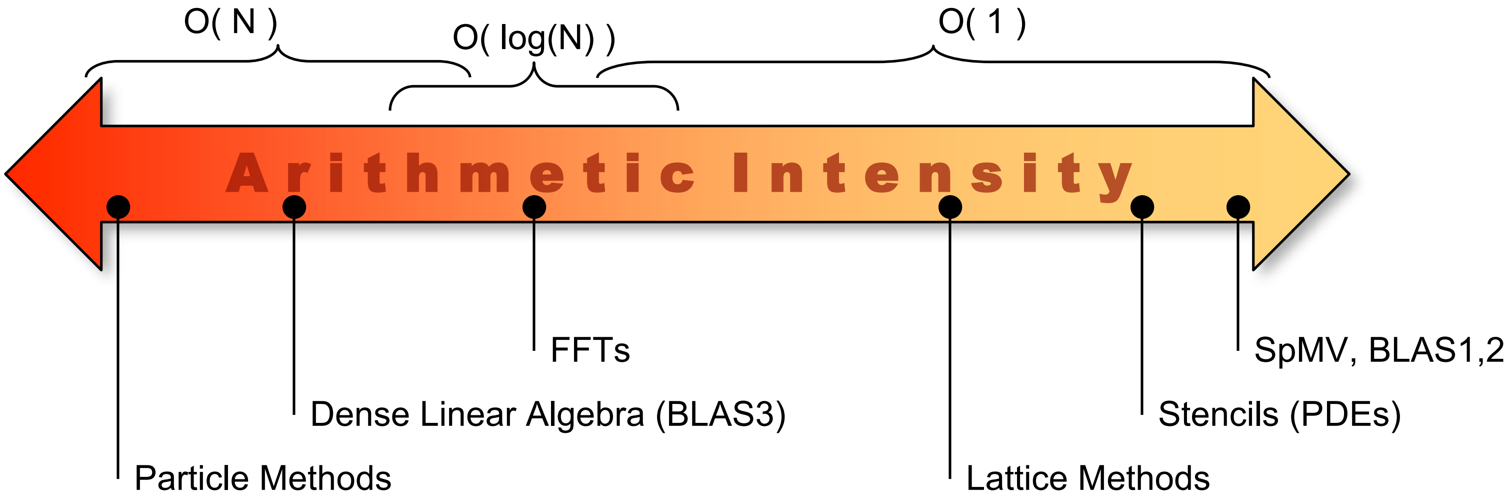


Non-Uniform Memory Access

Autotuning
Multicore SMPs
HPC Kernels
SpMV
LBMHD
Summary

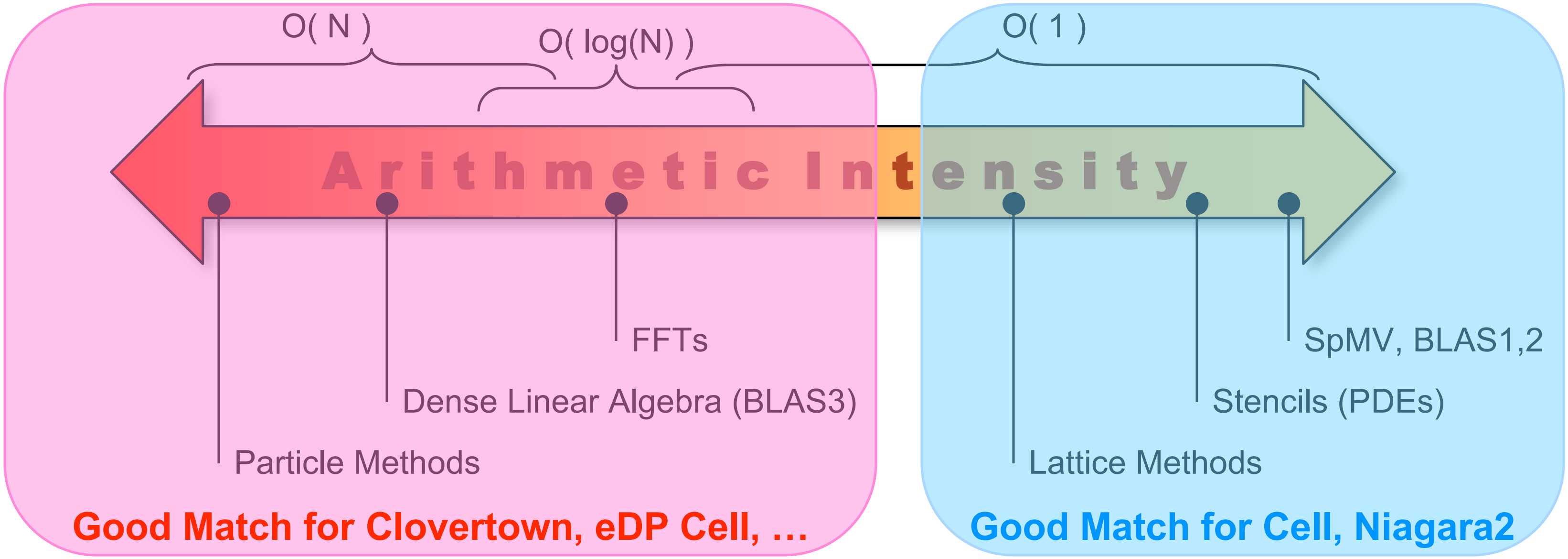
HPC Kernels

Arithmetic Intensity



- Arithmetic Intensity ~ Total Compulsory Flops / Total Compulsory Bytes
- Many HPC kernels have an arithmetic intensity that scales with with problem size (increasing temporal locality)
- But there are many important and interesting kernels that don’t
- Low arithmetic intensity kernels are likely to be memory bound
- High arithmetic intensity kernels are likely to be processor bound
- Ignores memory addressing complexity

Arithmetic Intensity



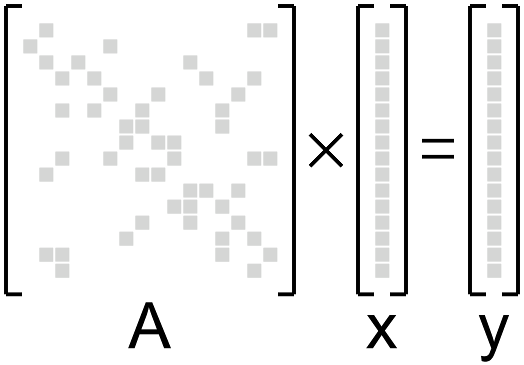
- Arithmetic Intensity ~ Total Compulsory Flops / Total Compulsory Bytes
- Many HPC kernels have an arithmetic intensity that scales with with problem size (increasing temporal locality)
- But there are many important and interesting kernels that don't
- Low arithmetic intensity kernels are likely to be memory bound
- High arithmetic intensity kernels are likely to be processor bound
- Ignores memory addressing complexity

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.

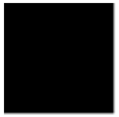
Sparse Matrix Vector Multiplication

- 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 computational intensity (often >6 bytes/flop) = likely memory bound**

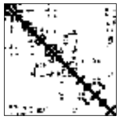


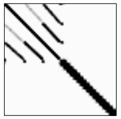
Dataset (Matrices)

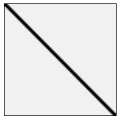
2K x 2K Dense matrix
stored in sparse format

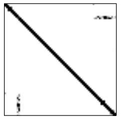

Dense

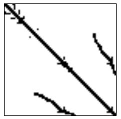
Well Structured
(sorted by nonzeros/row)

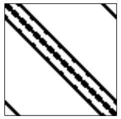

Protein

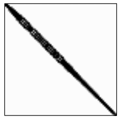

FEM / Spheres

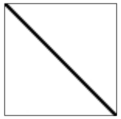

FEM / Cantilever

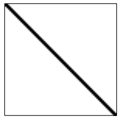

Wind Tunnel


FEM / Harbor

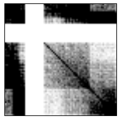

QCD

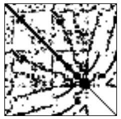

FEM / Ship

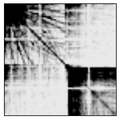

Economics


Epidemiology

Poorly Structured
hodgepodge


FEM / Accelerator


Circuit


webbase

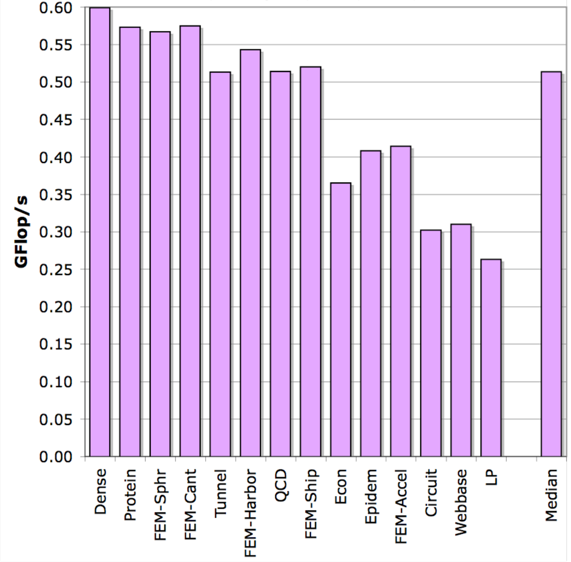
Extreme Aspect Ratio
(linear programming)

LP

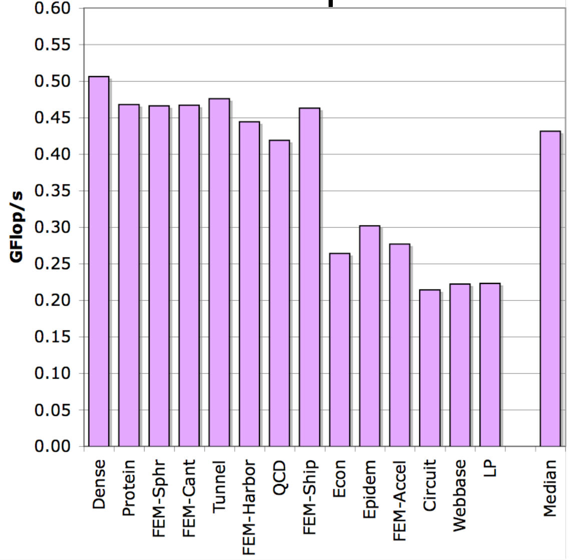
- Pruned original SPARSITY suite down to 14
- none should fit in cache
- Subdivided them into 4 categories
- Rank ranges from 2K to 1M

Naïve Serial Implementation

Intel Clovertown

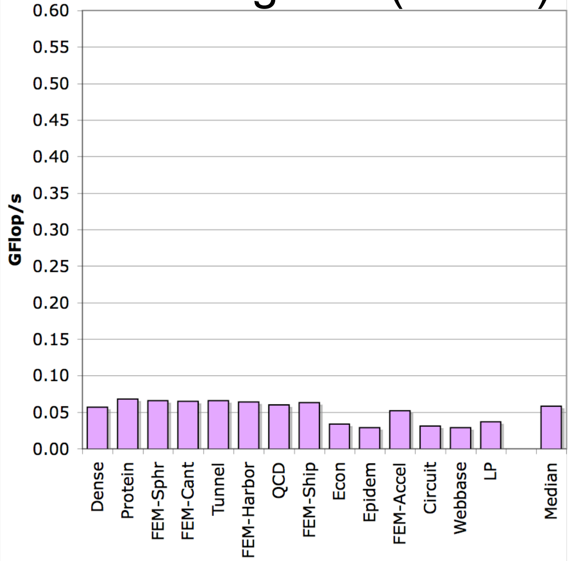


AMD Opteron

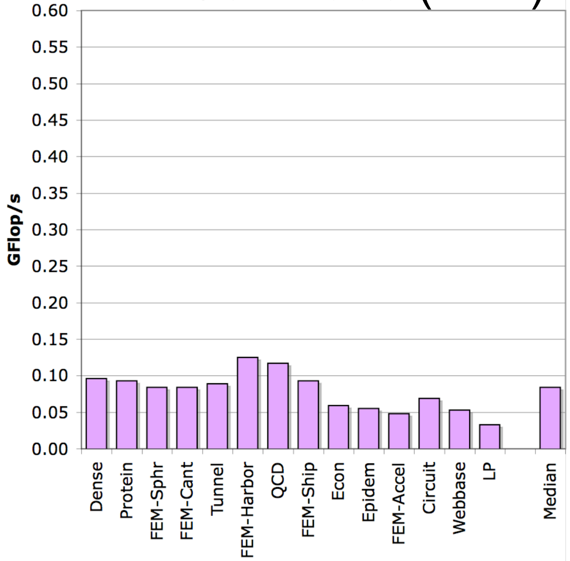


- Vanilla C implementation
- Matrix stored in CSR (compressed sparse row)
- Explored compiler options, but only the best is presented here
- x86 core delivers > 10x the performance of a Niagara2 thread

Sun Niagara2 (Huron)

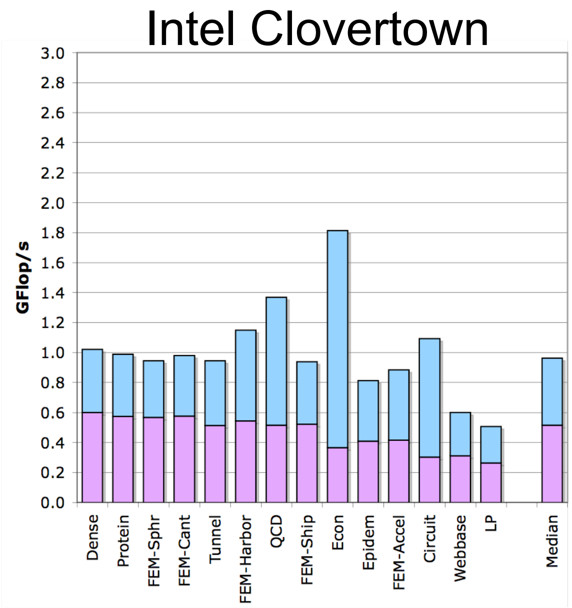


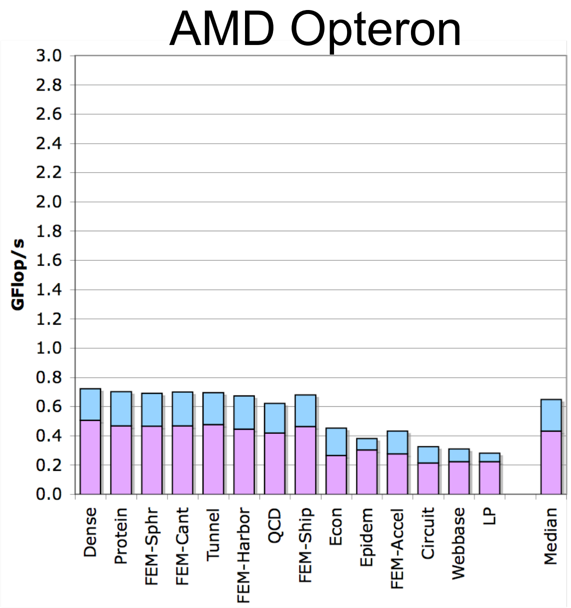
IBM Cell Blade (PPE)

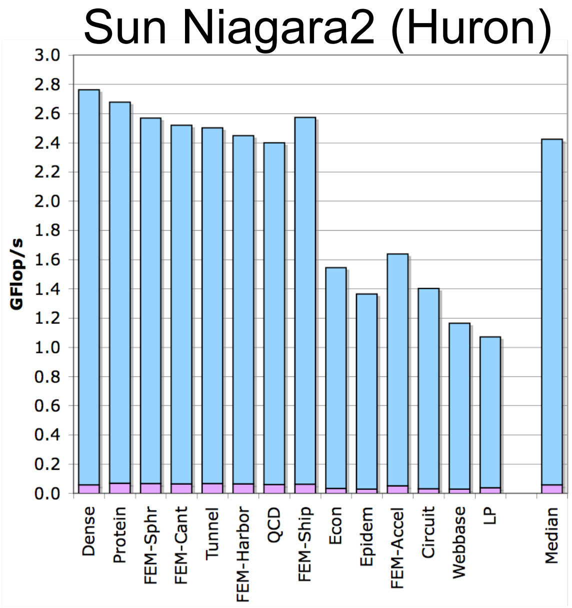


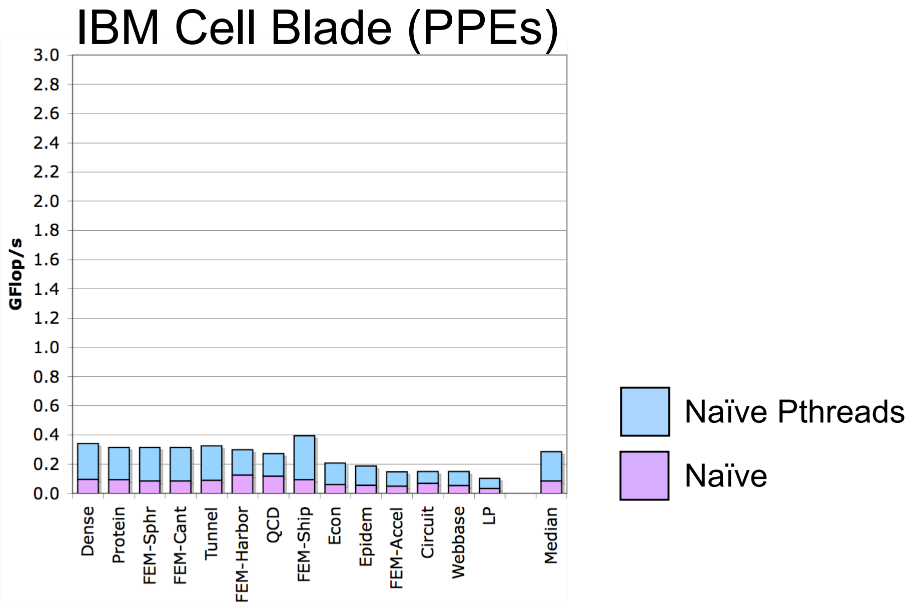
Naïve Parallel Implementation

- SPMD style
- Partition by rows
- Load balance by nonzeros
- N2 ~ 2.5x x86 machine



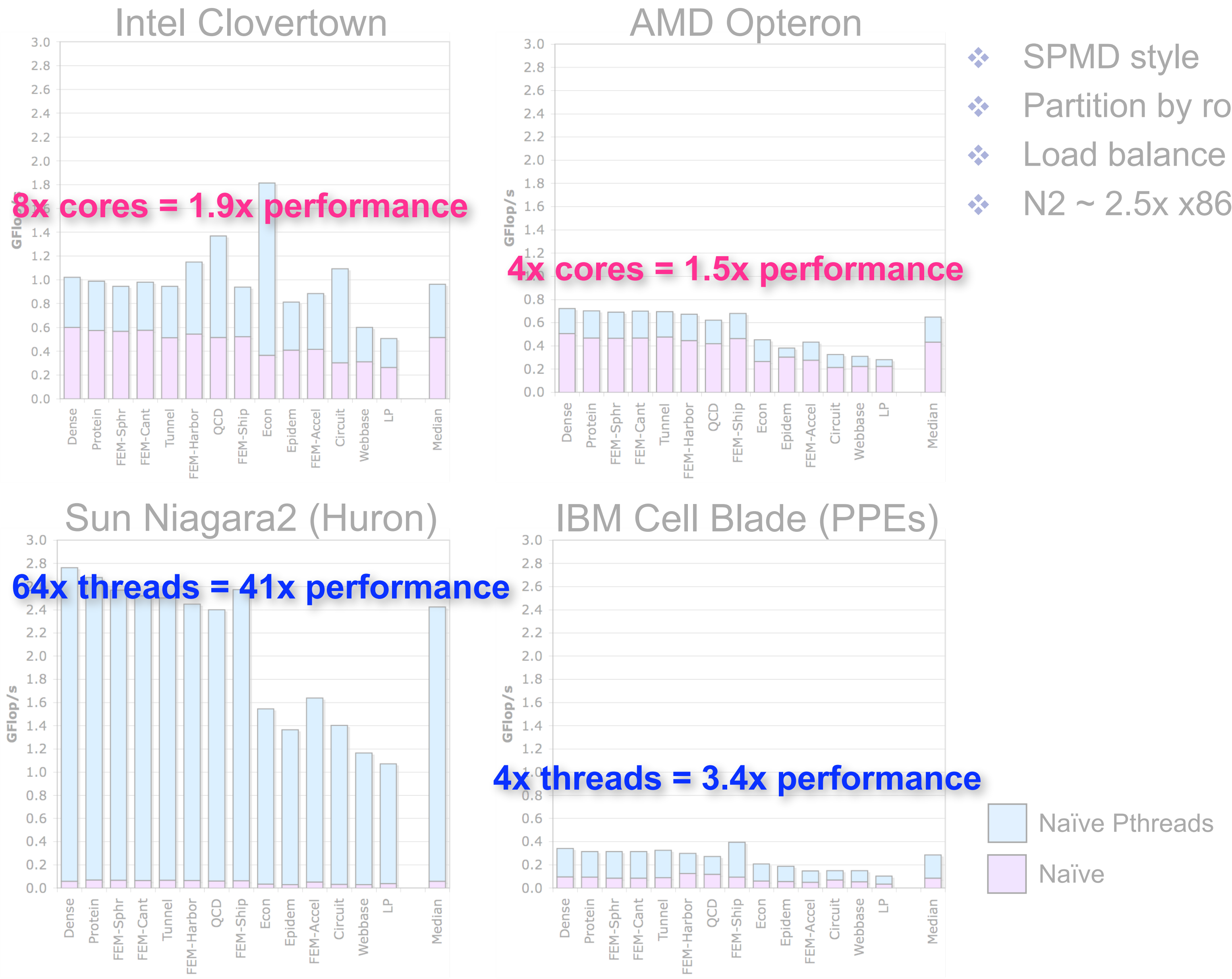






Naïve Parallel Implementation

- SPMD style
- Partition by rows
- Load balance by nonzeros
- N2 ~ 2.5x x86 machine


Intel Clovertown

8x cores = 1.9x performance

AMD Opteron

4x cores = 1.5x performance

Sun Niagara2 (Huron)

64x threads = 41x performance

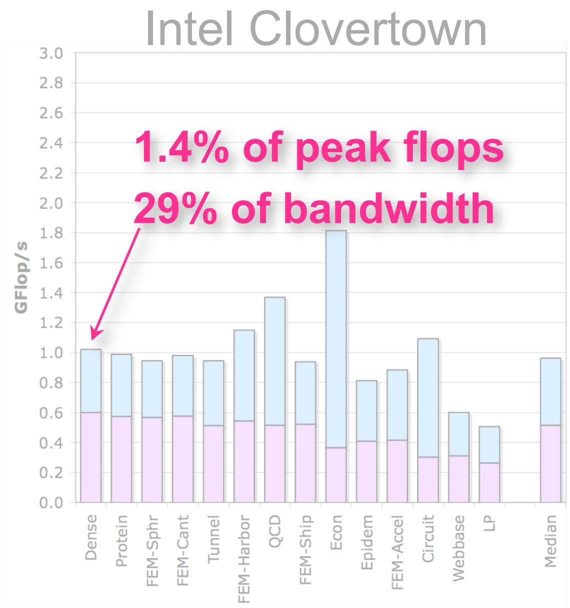
IBM Cell Blade (PPEs)

4x threads = 3.4x performance

EECS
Electrical Engineering and
Computer Sciences

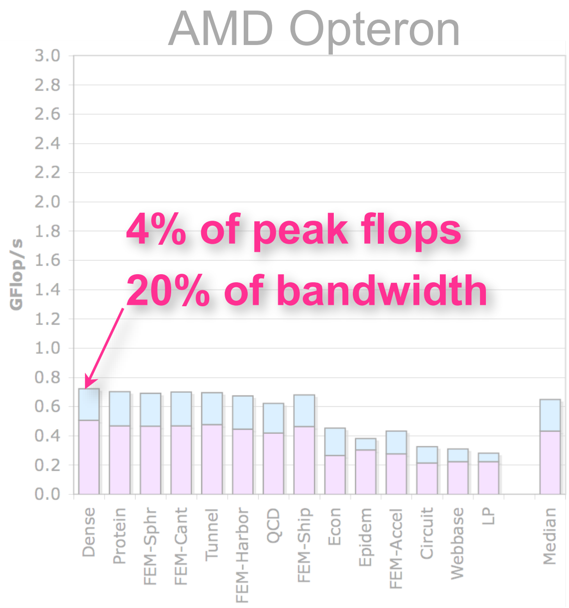
Naïve Parallel Implementation

BERKELEY PAR LAB


Intel Clovertown

1.4% of peak flops

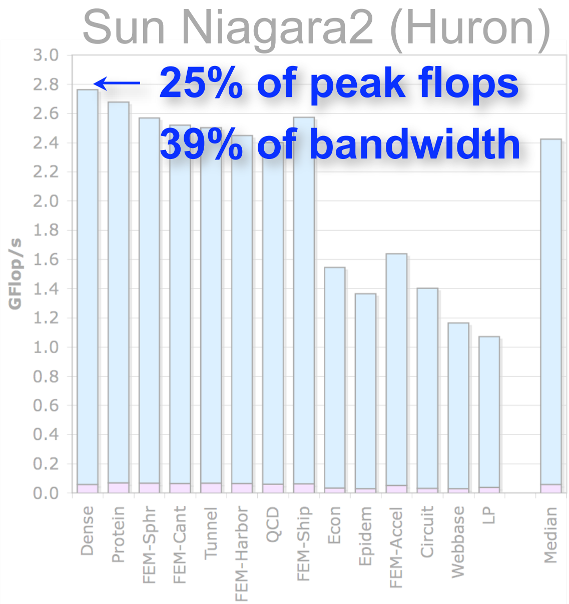
29% of bandwidth


AMD Opteron

4% of peak flops

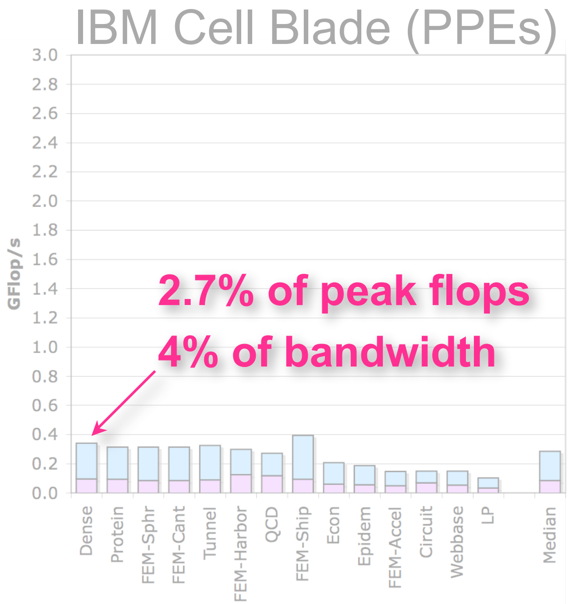
20% of bandwidth

- SPMD style
- Partition by rows
- Load balance by nonzeros
- N2 ~ 2.5x x86 machine


Sun Niagara2 (Huron)

25% of peak flops

39% of bandwidth


IBM Cell Blade (PPEs)

2.7% of peak flops

4% of bandwidth

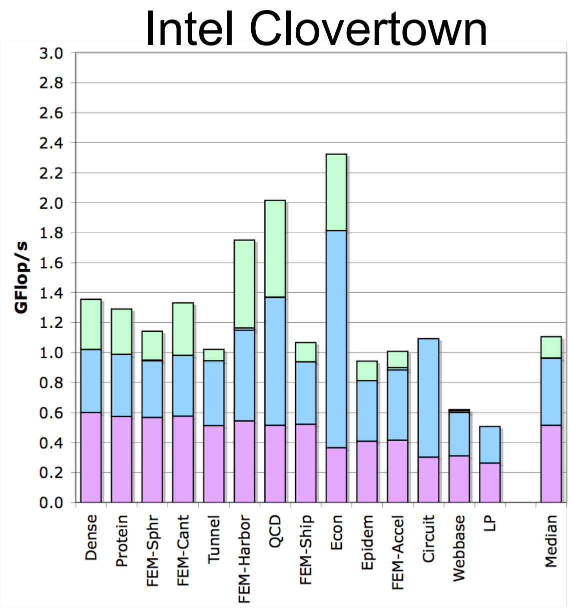
□ Naïve Pthreads

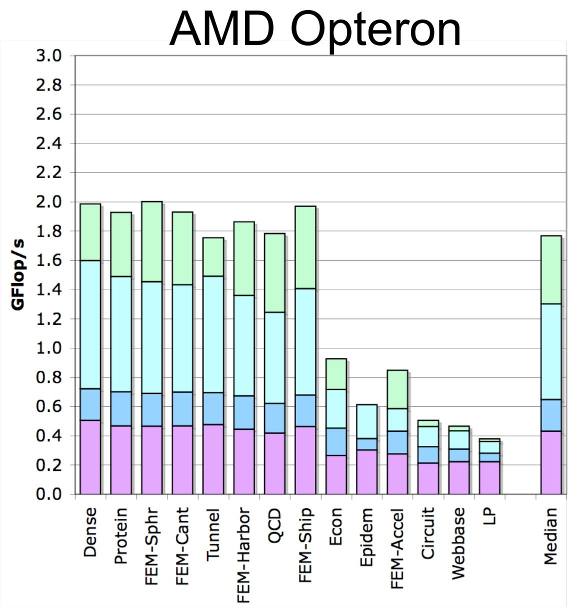
□ Naïve

EECS
Electrical Engineering and Computer Sciences

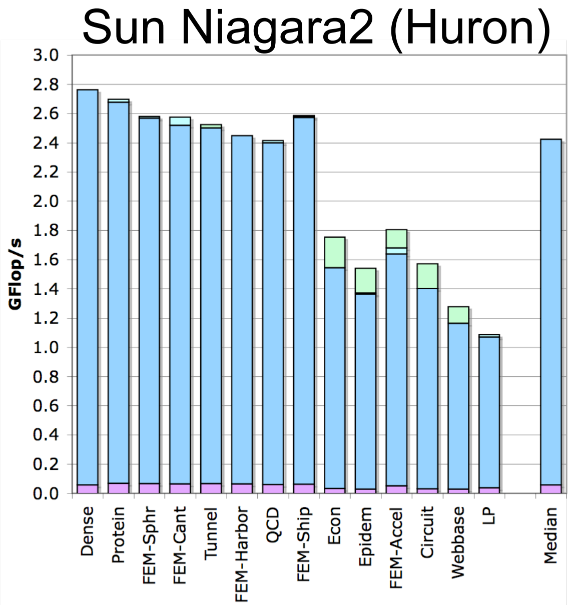
Autotuned Performance
(+NUMA & SW Prefetching)

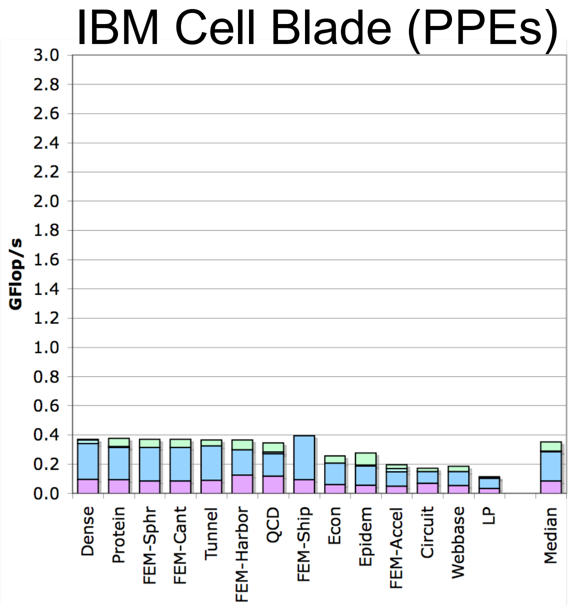
BERKELEY PAR LAB





- Use first touch, or libnuma to exploit NUMA.
- Also includes process affinity.
- Tag prefetches with temporal locality
- **Autotune**: search for the optimal prefetch distances





🟩 +SW Prefetching

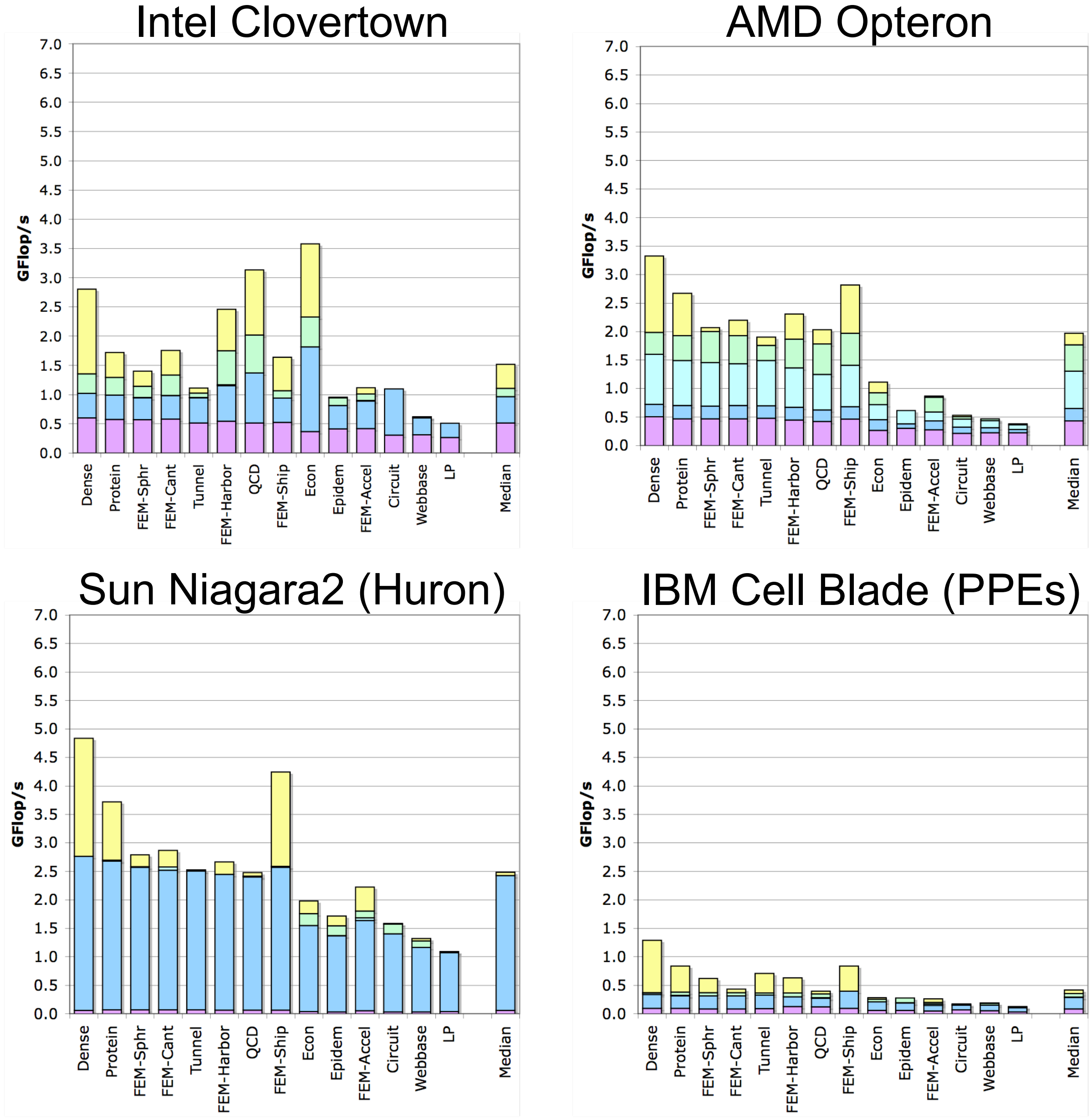
▢ +NUMA/Affinity

🟦 Naïve Pthreads

🟪 Naïve

Autotuned Performance

(+Matrix Compression)



Intel Clovertown

AMD Opteron

Sun Niagara2 (Huron)

IBM Cell Blade (PPEs)

- ❖ If memory bound, only hope is minimizing memory traffic
- ❖ Heuristically compress the parallelized matrix to minimize it
- ❖ Implemented with SSE
- ❖ Benefit of prefetching is hidden by requirement of register blocking
- ❖ Options: register blocking, index size, format, etc…

🟨 +Compression

🟩 +SW Prefetching

□ +NUMA/Affinity

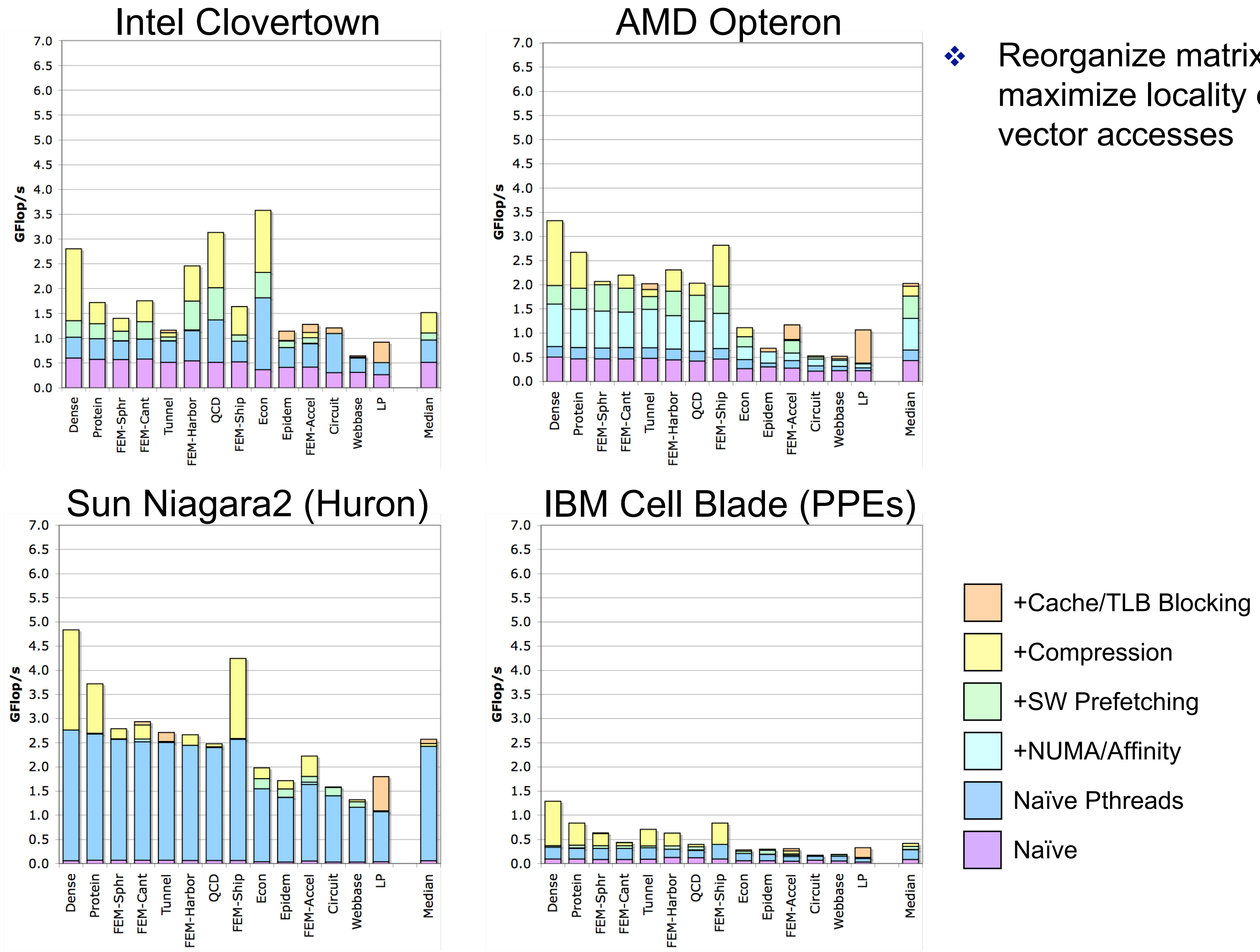
🟦 Naïve Pthreads

🟪 Naïve

Autotuned Performance

(+Cache/TLB Blocking)

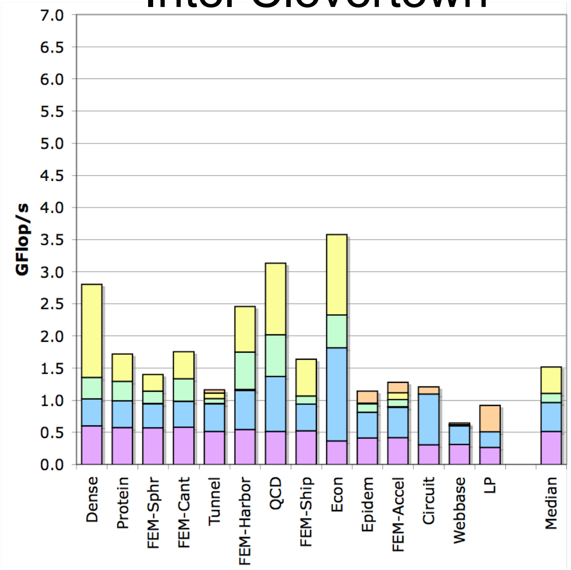
- Reorganize matrix to maximize locality of source vector accesses



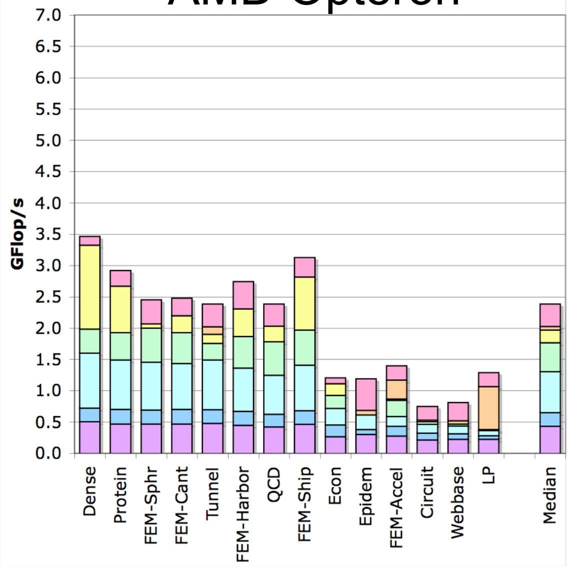
Autotuned Performance

(+DIMMs, Firmware, Padding)

Intel Clovertown

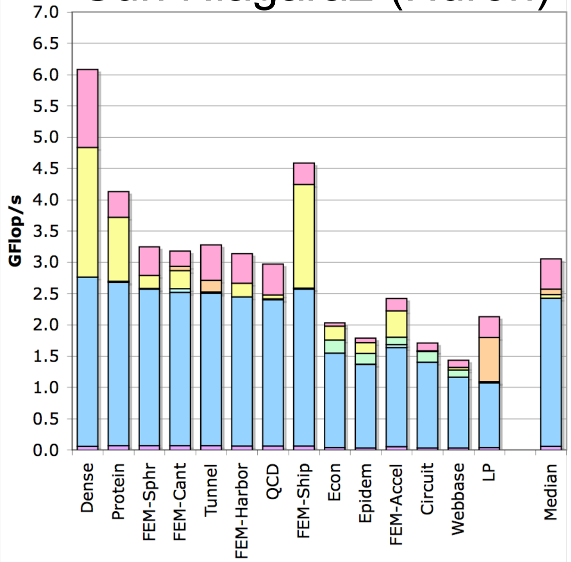


AMD Opteron

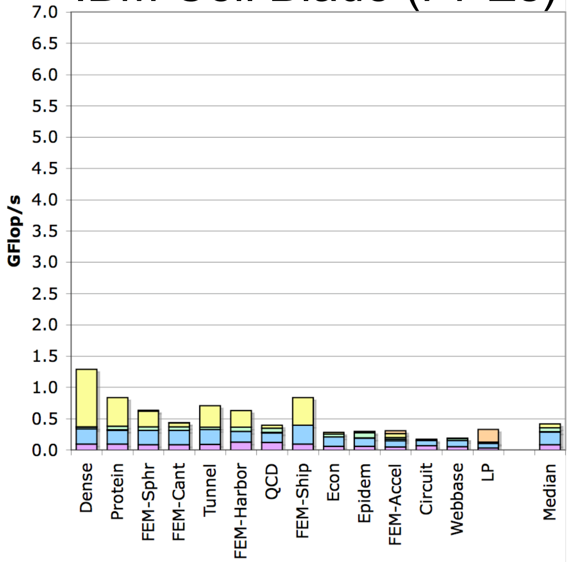


- ❖ Clovertown was already fully populated with DIMMs
- ❖ Gave Opteron as many DIMMs as Clovertown
- ❖ Firmware update for Niagara2
- ❖ Array padding to avoid inter-thread conflict misses
- ❖ PPE’s use ~1/3 of Cell chip area

Sun Niagara2 (Huron)



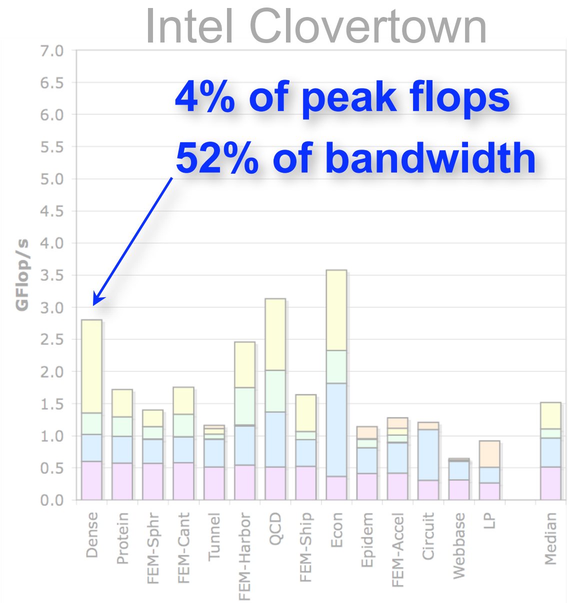
IBM Cell Blade (PPEs)



- 🟥 +More DIMMs(opteron), +FW fix, array padding(N2), etc…
- 🟧 +Cache/TLB Blocking
- 🟨 +Compression
- 🟩 +SW Prefetching
- ⬜ +NUMA/Affinity
- 🟦 +Naïve Pthreads
- 🟪 Naïve

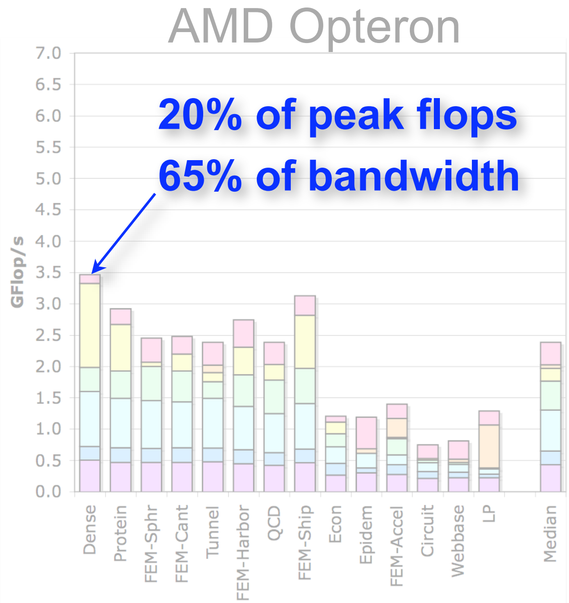
Autotuned Performance

(+DIMMs, Firmware, Padding)


Intel Clovertown

4% of peak flops

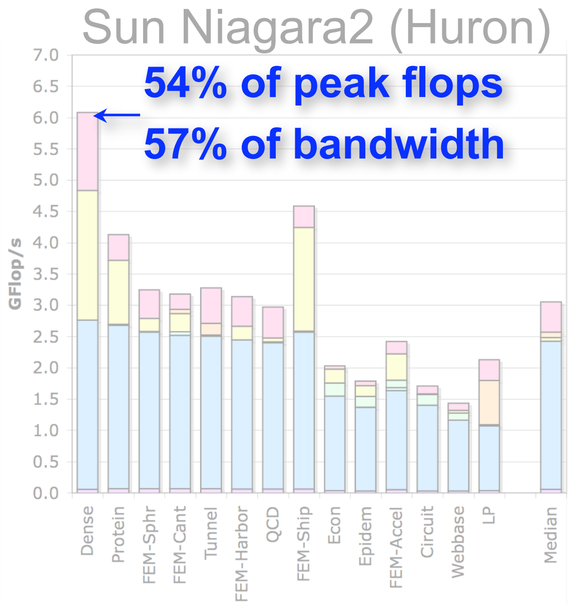
52% of bandwidth


AMD Opteron

20% of peak flops

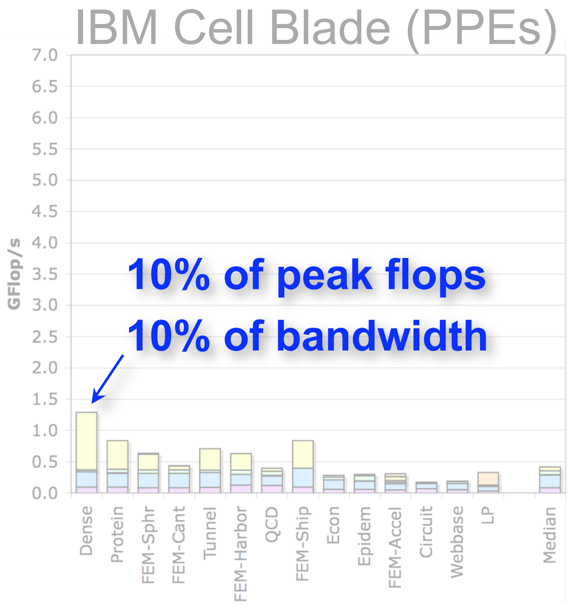
65% of bandwidth

- Clovertown was already fully populated with DIMMs
- Gave Opteron as many DIMMs as Clovertown
- Firmware update for Niagara2
- Array padding to avoid inter-thread conflict misses
- PPE's use ~1/3 of Cell chip area


Sun Niagara2 (Huron)

54% of peak flops

57% of bandwidth


IBM Cell Blade (PPEs)

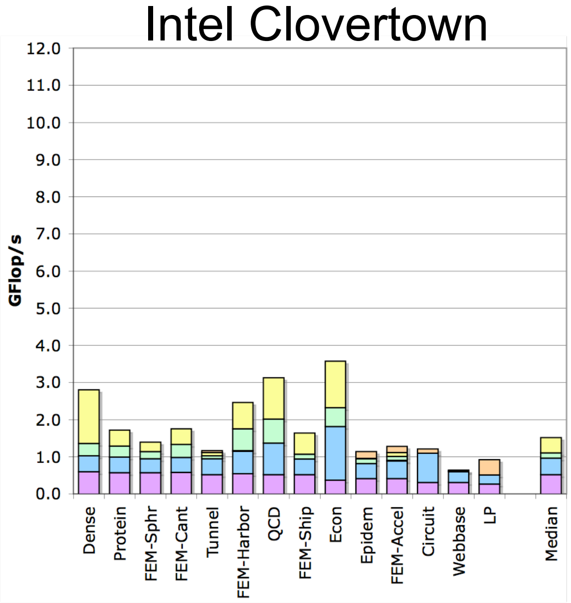
10% of peak flops

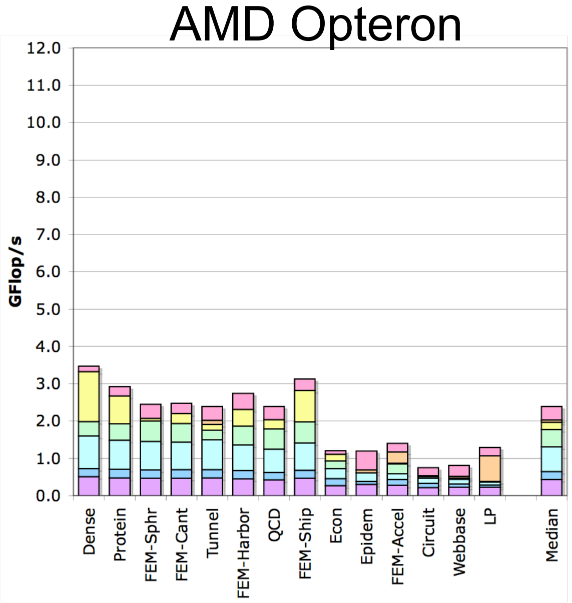
10% of bandwidth

- 🟪 +More DIMMs(opteron), +FW fix, array padding(N2), etc…
- 🟧 +Cache/TLB Blocking
- 🟨 +Compression
- 🟩 +SW Prefetching
- ⬜ +NUMA/Affinity
- 🟦 Naïve Pthreads
- 🟪 Naïve

Autotuned Performance

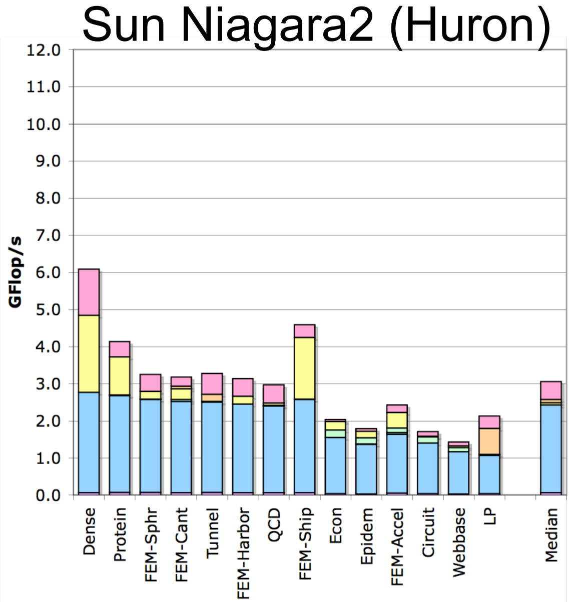
(+Cell/SPE version)

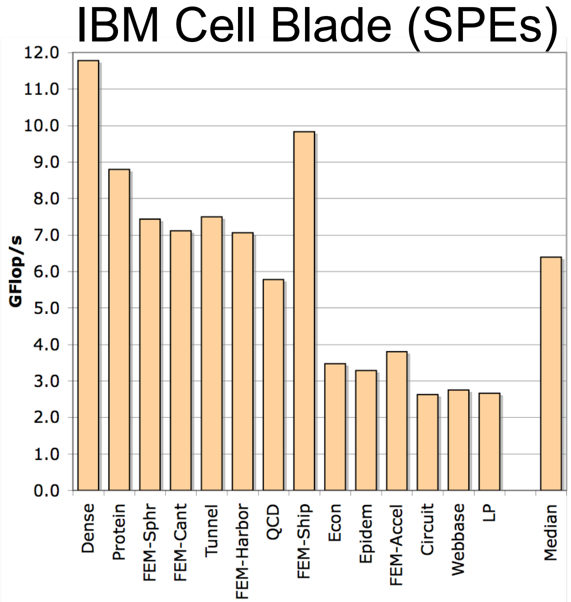

Intel Clovertown


AMD Opteron

- Wrote a double precision Cell/SPE version
- DMA, local store blocked, NUMA aware, etc…
- Only 2x1 and larger BCOO
- Only the SpMV-proper routine changed

- About 12x faster (median) than using the PPEs alone.


Sun Niagara2 (Huron)


IBM Cell Blade (SPEs)

■ +More DIMMs(opteron), +FW fix, array padding(N2), etc…

■ +Cache/TLB Blocking

■ +Compression

■ +SW Prefetching

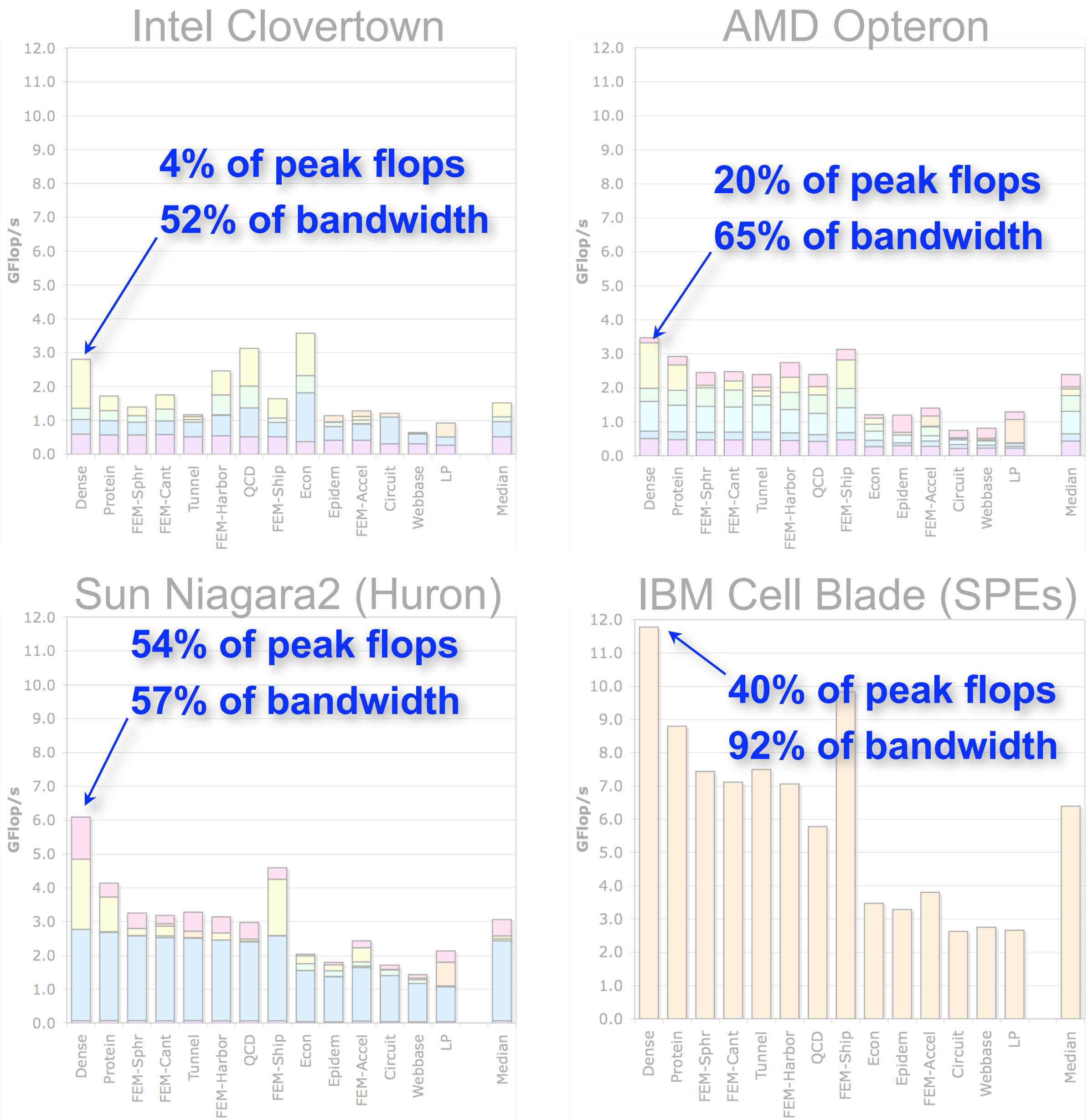
■ +NUMA/Affinity

■ Naïve Pthreads

■ Naïve

Autotuned Performance

(+Cell/SPE version)



Intel Clovertown: 4% of peak flops, 52% of bandwidth

AMD Opteron: 20% of peak flops, 65% of bandwidth

Sun Niagara2 (Huron): 54% of peak flops, 57% of bandwidth

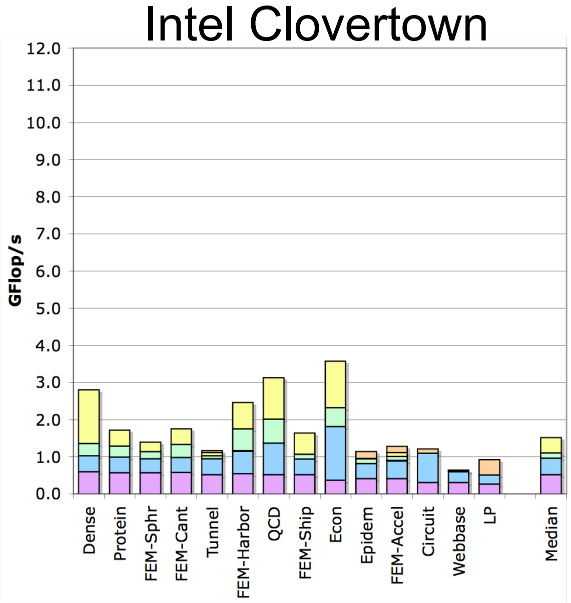
IBM Cell Blade (SPEs): 40% of peak flops, 92% of bandwidth

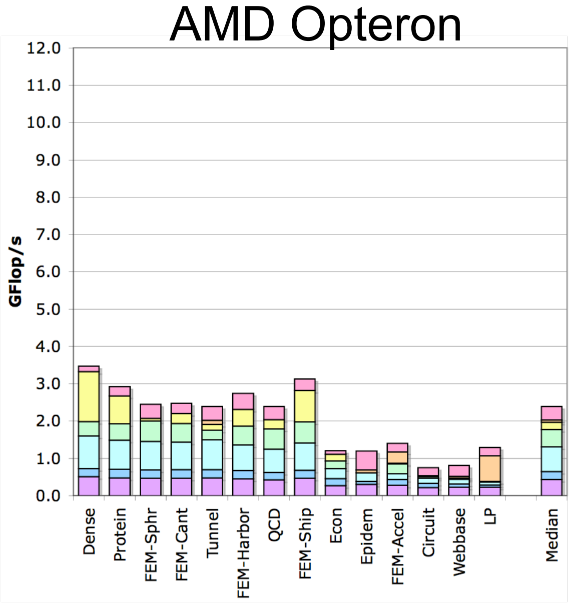
- Wrote a double precision Cell/SPE version
- DMA, local store blocked, NUMA aware, etc…
- Only 2x1 and larger BCOO
- Only the SpMV-proper routine changed
- About 12x faster than using the PPEs alone.

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

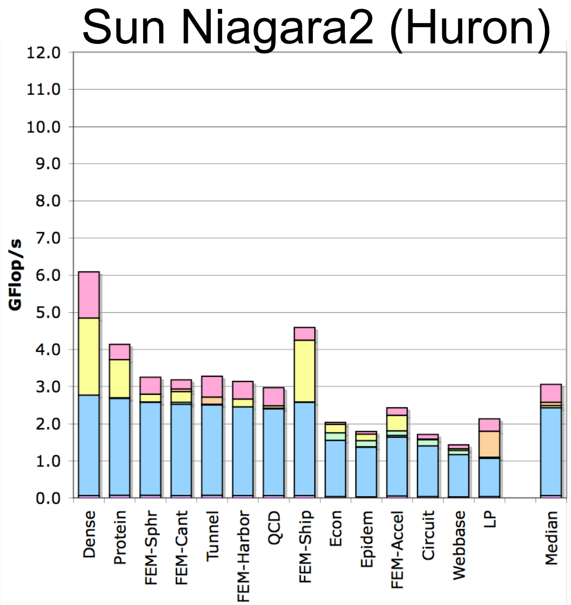
Autotuned Performance

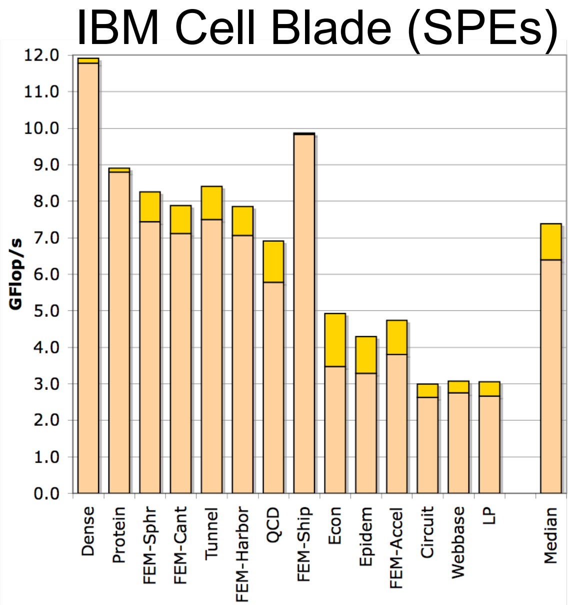
(How much did double precision and 2x1 blocking hurt)


Intel Clovertown


AMD Opteron

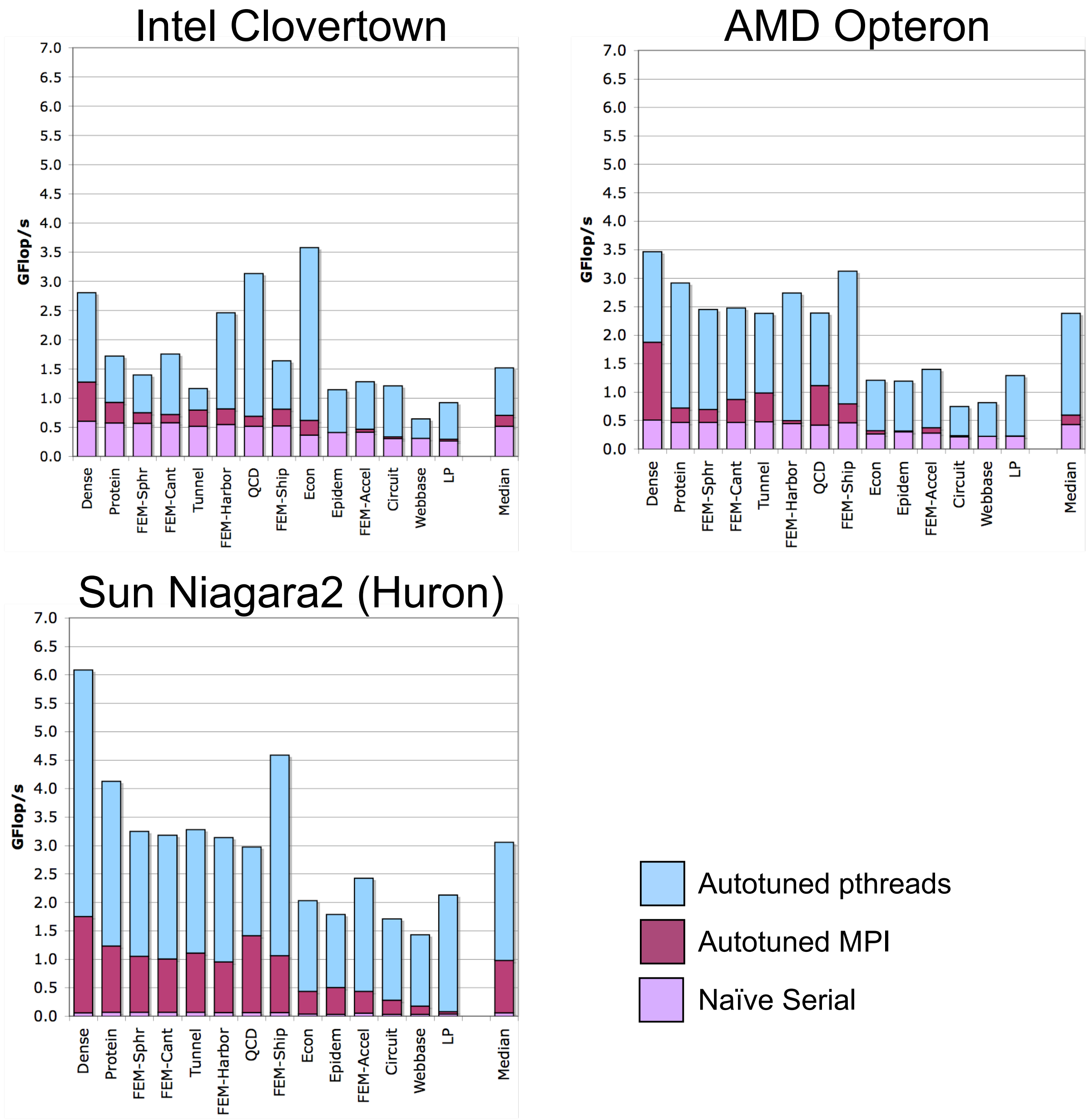
- ❖ Model faster cores by commenting out the inner kernel calls, but still performing all DMAs
- ❖ Enabled 1x1 BCOO
- ❖ ~16% improvement


Sun Niagara2 (Huron)


IBM Cell Blade (SPEs)

- ■ +better Cell implementation
- ■ +More DIMMs(opteron), +FW fix, array padding(N2), etc…
- ■ +Cache/TLB Blocking
- ■ +Compression
- ■ +SW Prefetching
- ■ +NUMA/Affinity
- ■ Naïve Pthreads
- ■ Naïve

MPI vs. Threads



- ❖ On x86 machines, autotuned(OSKI) shared memory MPICH implementation rarely scales beyond 2 threads
- ❖ Still debugging MPI issues on Niagara2, but so far, it rarely scales beyond 8 threads.

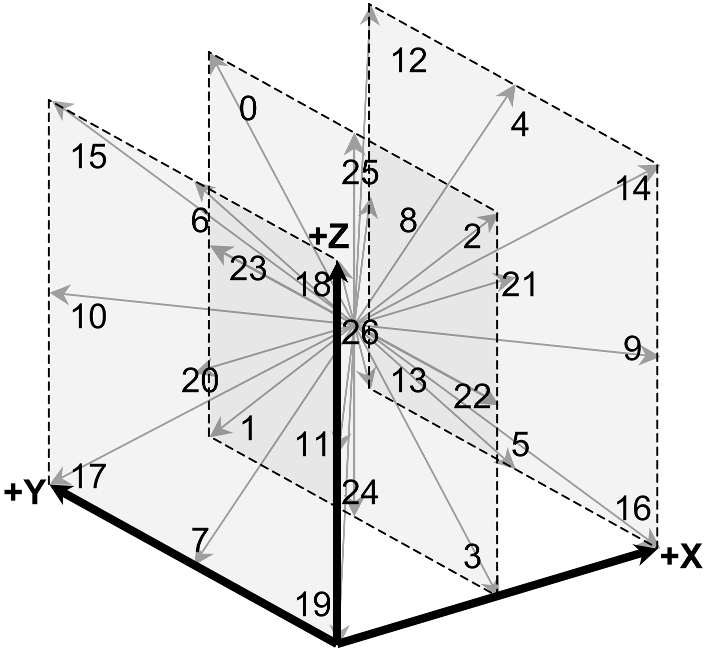
Lattice-Boltzmann Magneto-Hydrodynamics (LBMHD)

❖Preliminary results

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

Lattice Methods

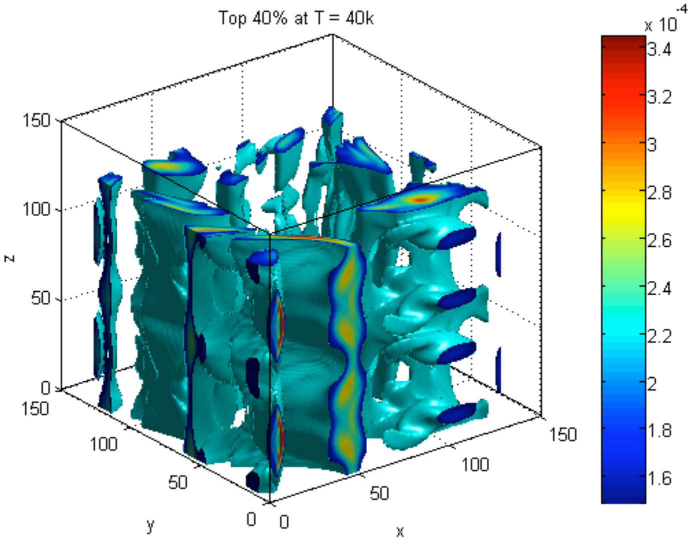
- Structured grid code, with a series of time steps
- Popular in CFD
- Allows for complex boundary conditions
- Higher dimensional phase space
  - Simplified kinetic model that maintains the macroscopic quantities
  - Distribution functions (e.g. 27 velocities per point in space) are used to reconstruct macroscopic quantities
  - Significant Memory capacity requirements



LBMHD

(general characteristics)

- Plasma turbulence simulation
- Two distributions:
  - momentum distribution (27 components)
  - magnetic distribution (15 vector compone
- Three macroscopic quantities:
  - Density
  - Momentum (vector)
  - Magnetic Field (vector)



- Must read 73 doubles, and update(write) 79 doubles per point in space
- Requires about 1300 floating point operations per point in space
- Just over 1.0 flops/byte (ideal)
- No temporal locality between points in space within one time step

LBMHD
(implementation details)

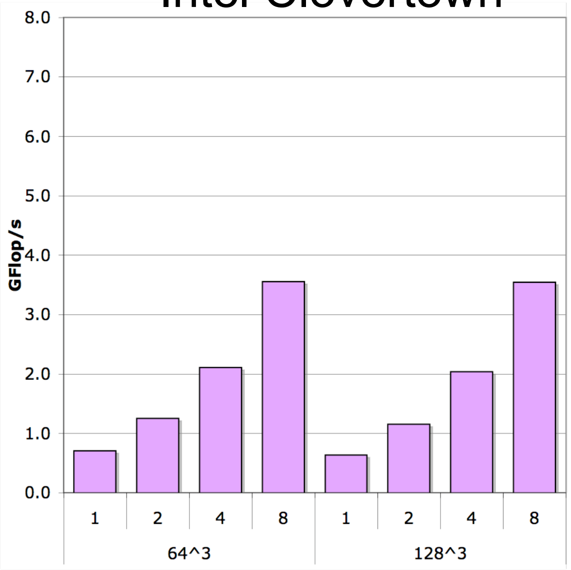
- Data Structure choices:
  - **Array of Structures**: lacks spatial locality
  - **Structure of Arrays:** huge number of memory streams per thread, but vectorizes well

- Parallelization
  - Fortran version used MPI to communicate between nodes.
    = bad match for multicore
  - This version uses pthreads for multicore, and MPI for inter-node
  - MPI is not used when autotuning

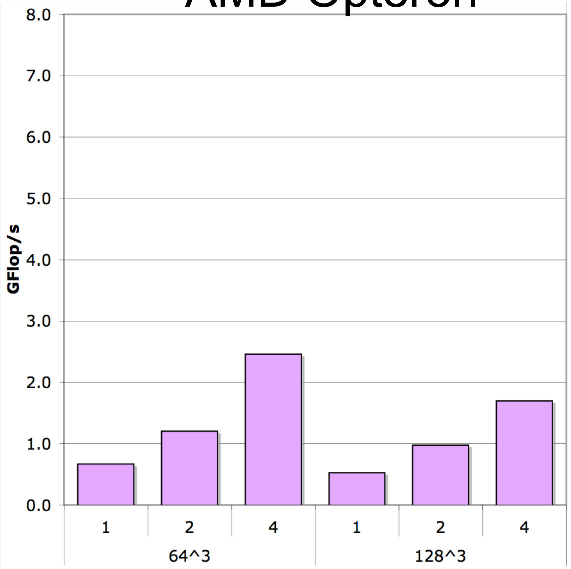
- Two problem sizes:
  - 64^3 (~330MB)
  - 128^3 (~2.5GB)

Pthread Implementation

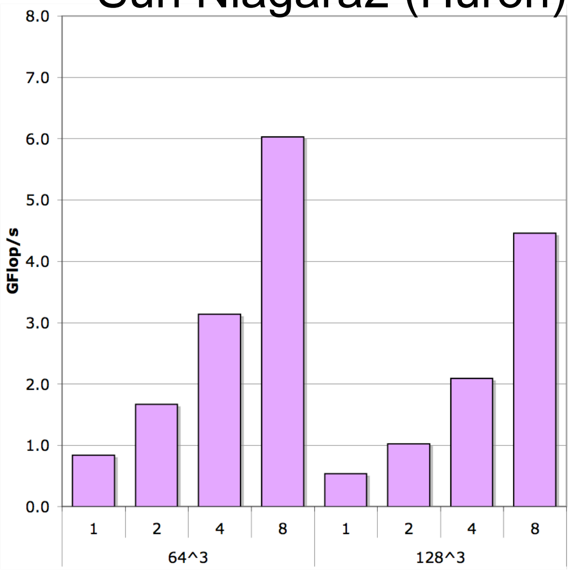
Intel Clovertown



AMD Opteron



Sun Niagara2 (Huron)

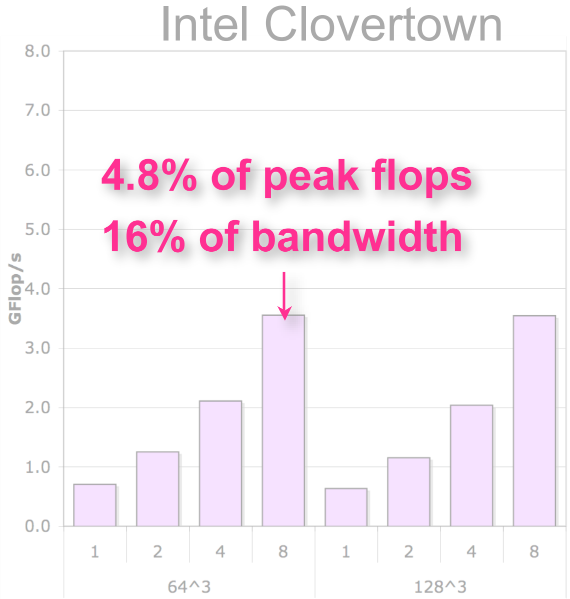


IBM Cell Blade

Cell version was not autotuned

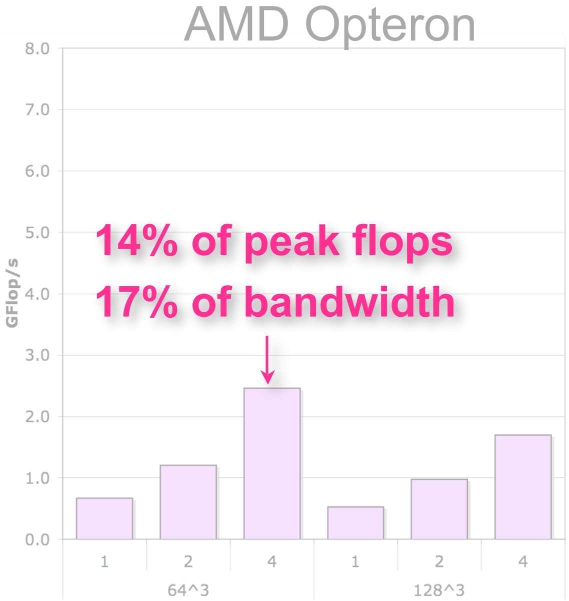
- ❖ Not naïve
  - fully unrolled loops
  - NUMA-aware
  - 1D parallelization
- ❖ Always used 8 threads per core on Niagara2

Pthread Implementation


Intel Clovertown

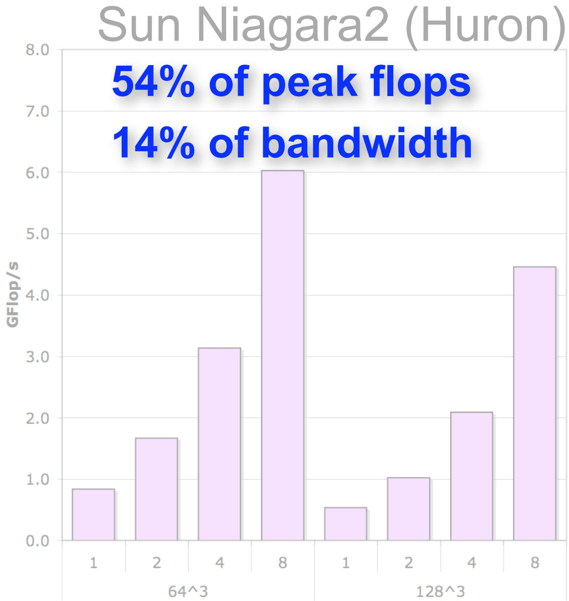
4.8% of peak flops

16% of bandwidth


AMD Opteron

14% of peak flops

17% of bandwidth


Sun Niagara2 (Huron)

54% of peak flops

14% of bandwidth

IBM Cell Blade

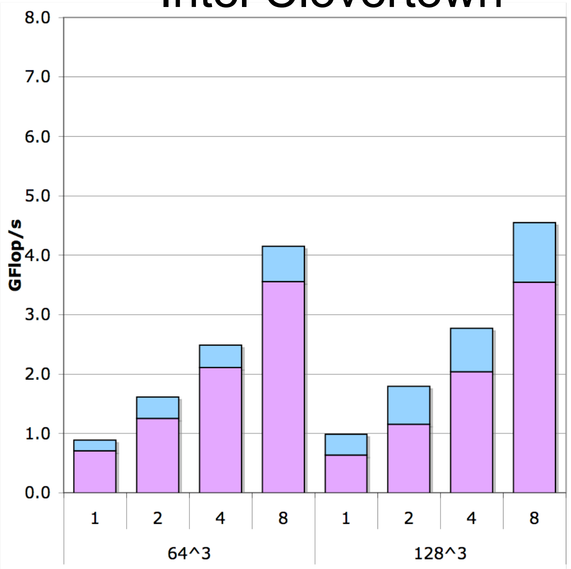
Cell version was not autotuned

- Not naïve
  - fully unrolled loops
  - NUMA-aware
  - 1D parallelization
- Always used 8 threads per core on Niagara2

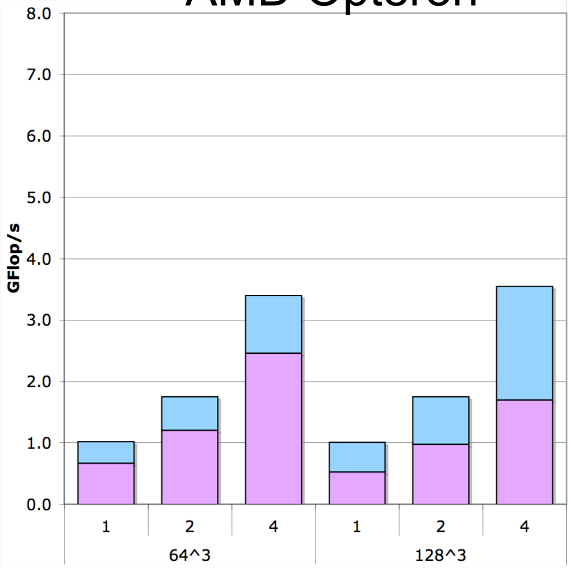
Autotuned Performance

(+Stencil-aware Padding)

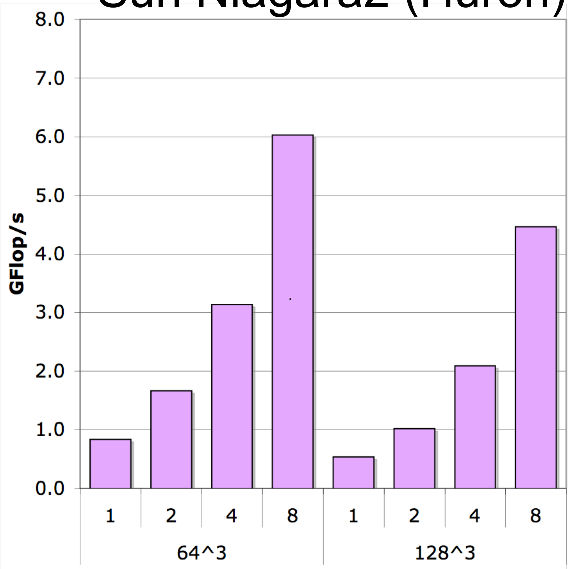
Intel Clovertown



AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade

Cell version was not autotuned

- This lattice method is essentially 79 simultaneous 72-point stencils
- Can cause conflict misses even with highly associative L1 caches (not to mention opteron's 2 way)
- **Solution**: pad each component so that when accessed with the corresponding stencil(spatial) offset, the components are uniformly distributed in the cache

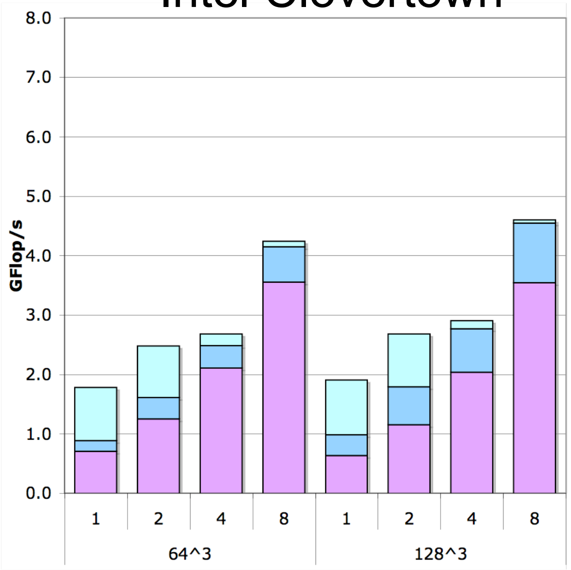
🟦 +Padding

🟪 Naïve+NUMA

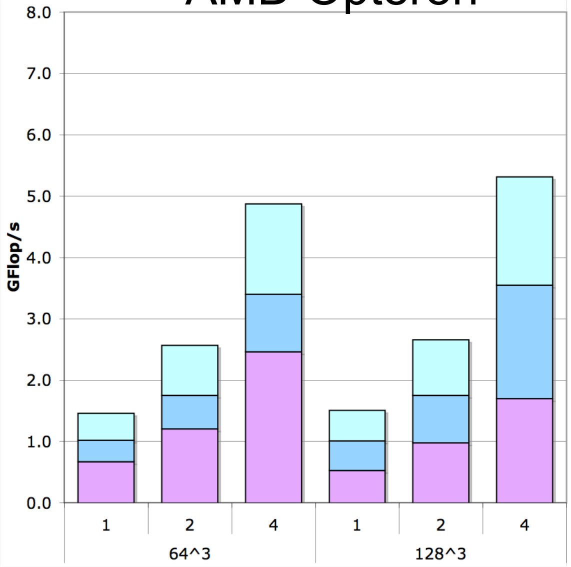
Autotuned Performance

(+Vectorization)

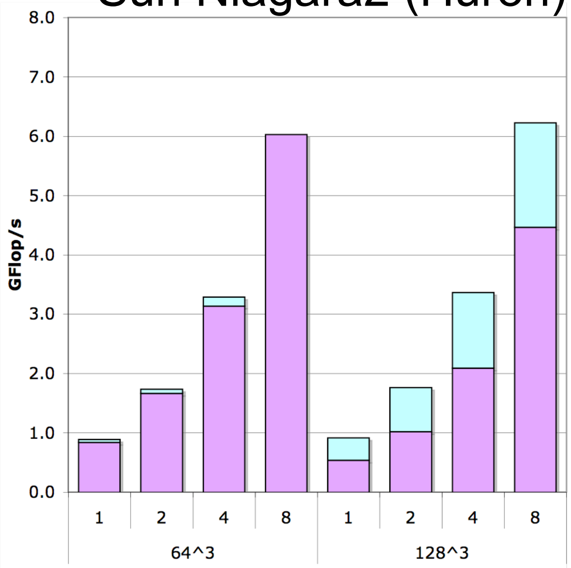
Intel Clovertown



AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade

Cell version was not autotuned

- Each update requires touching ~150 components, each likely to be on a different page
- TLB misses can significantly impact performance

- **Solution**: vectorization
- Fuse spatial loops,

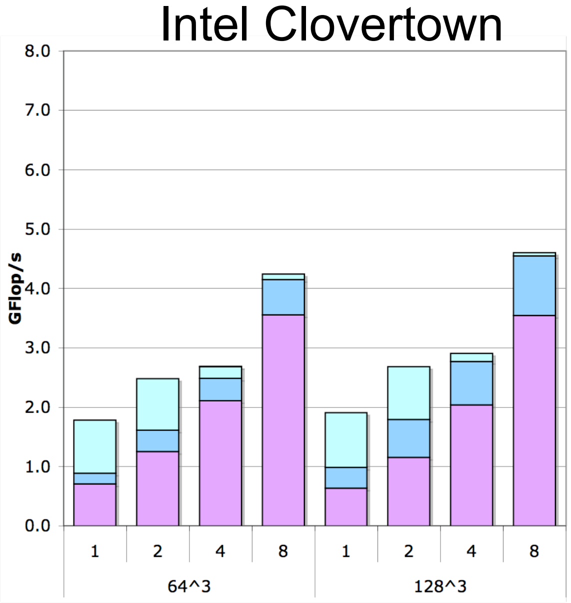
  strip mine into vectors of size VL, and interchange with phase dimensional loops

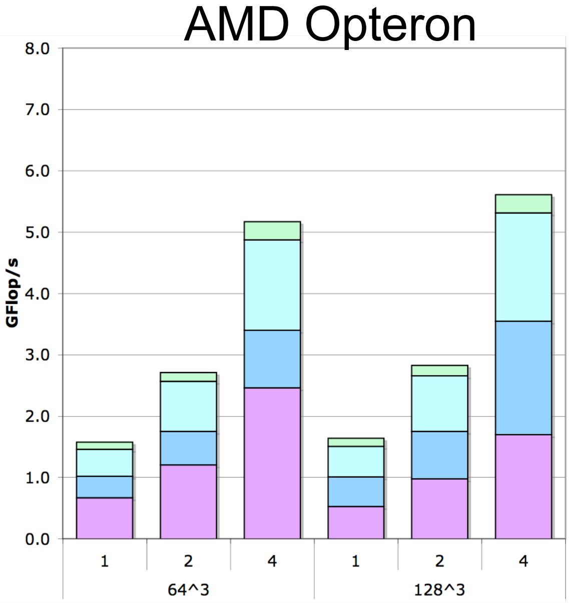
- **Autotune**: search for the optimal vector length
- Significant benefit on some architectures
- Becomes irrelevant when bandwidth dominates performance

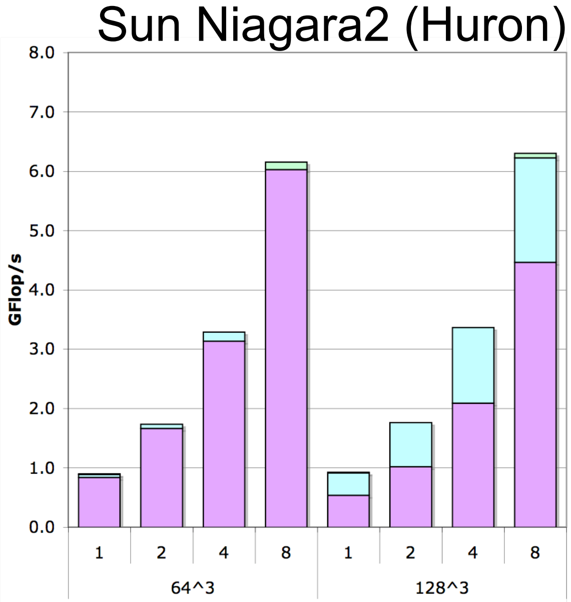
▭	+Vectorization
▭	+Padding
▭	Naïve+NUMA

Autotuned Performance

(+Explicit Unrolling/Reordering)







IBM Cell Blade

Cell version was not autotuned

- Give the compilers a helping hand for the complex loops
- **Code Generator**: Perl script to generate all power of 2 possibilities
- **Autotune**: search for the best unrolling and expression of data level parallelism
- Is essential when using SIMD intrinsics

🟩 +Unrolling

▢ +Vectorization

🟦 +Padding

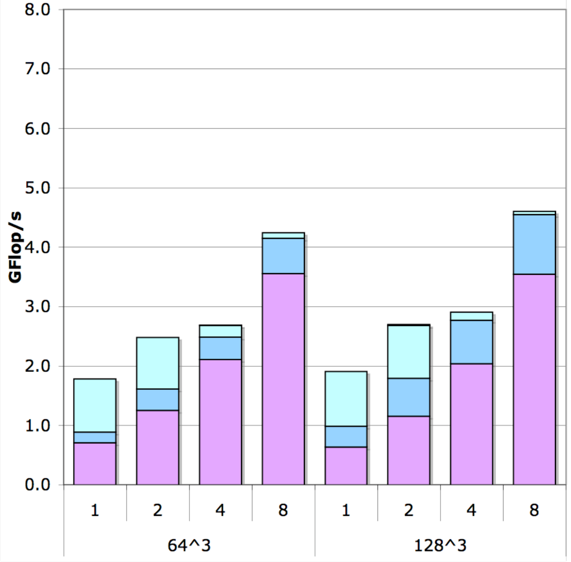
🟪 Naïve+NUMA

EECS
Electrical Engineering and Computer Sciences

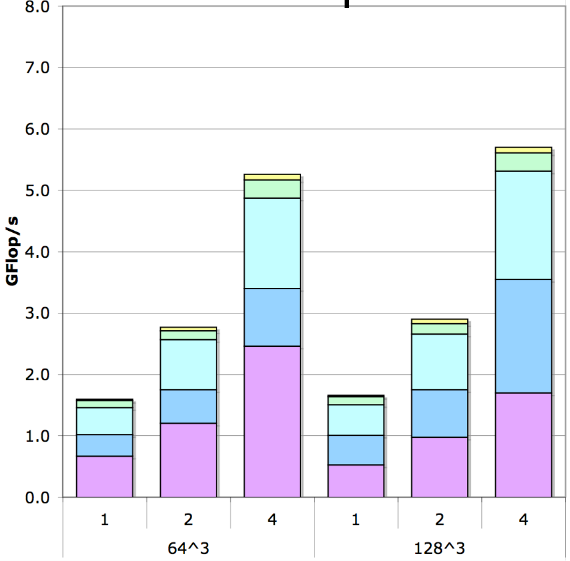
Autotuned Performance
(+Software prefetching)

BERKELEY PAR LAB

Intel Clovertown

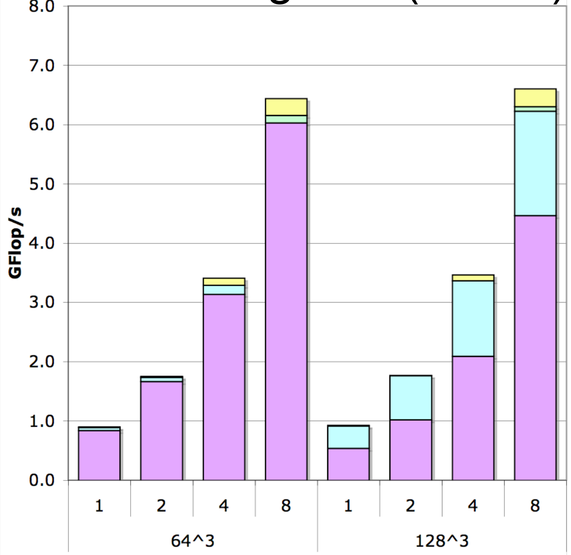


AMD Opteron



- Expanded the code generator to insert software prefetches in case the compiler doesn't.
- **Autotune**:
  - no prefetch
  - prefetch 1 line ahead
  - prefetch 1 vector ahead.
- Relatively little benefit for relatively little work

Sun Niagara2 (Huron)



IBM Cell Blade

Cell version was not autotuned

🟨 +SW Prefetching

🟩 +Unrolling

🟦 +Vectorization

🟦 +Padding

🟪 Naïve+NUMA

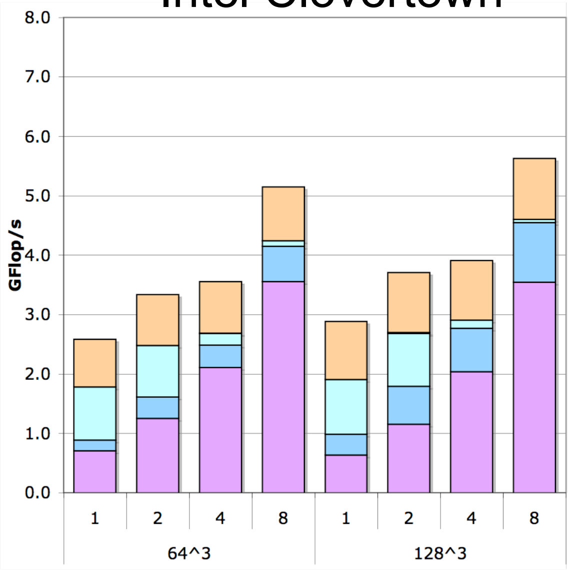
EECS
Electrical Engineering and
Computer Sciences

Autotuned Performance

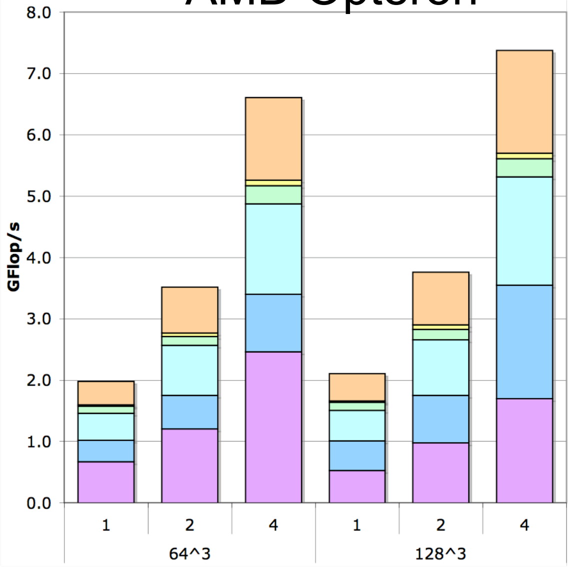
(+SIMDization, including non-temporal stores)

BERKELEY PAR LAB

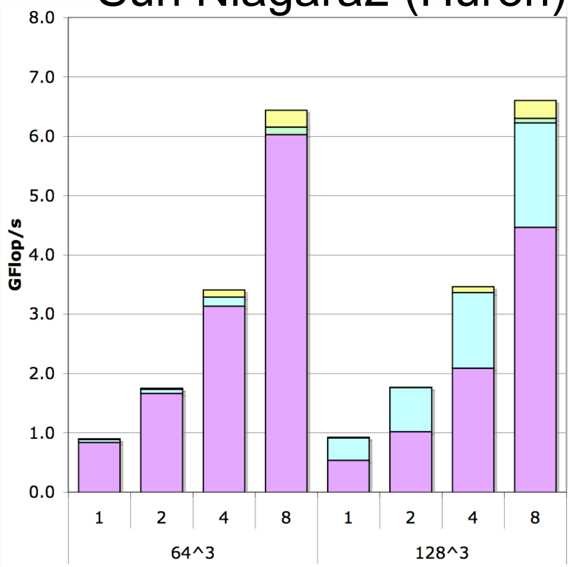
Intel Clovertown



AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade

Cell version was not autotuned

- Compilers(gcc & icc) failed at exploiting SIMD.
- Expanded the code generator to use SIMD intrinsics.
- Explicit unrolling/reordering was extremely valuable here.
- Exploited *movntpd* to minimize memory traffic (only hope if memory bound)
- Significant benefit for significant work

■ +SIMDization

■ +SW Prefetching

■ +Unrolling

■ +Vectorization

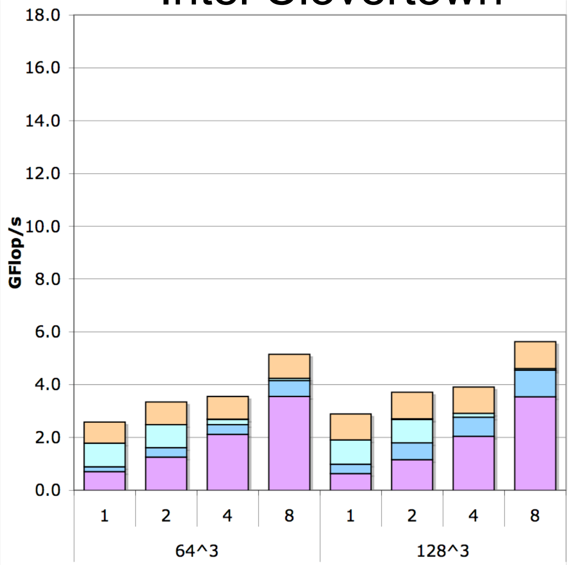
■ +Padding

■ Naïve+NUMA

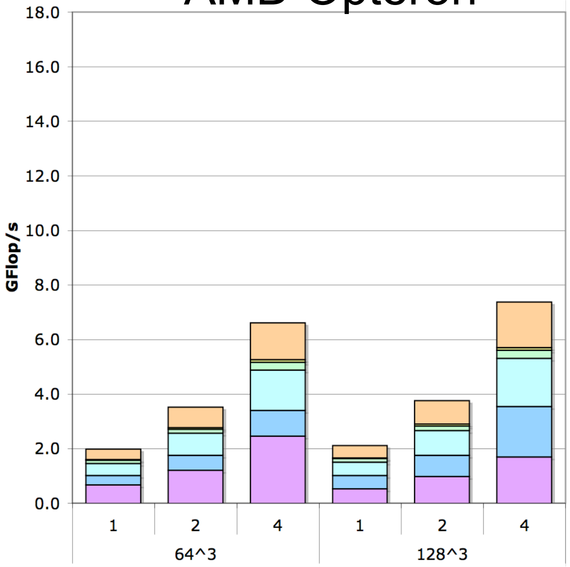
Autotuned Performance

(Cell/SPE version)

Intel Clovertown

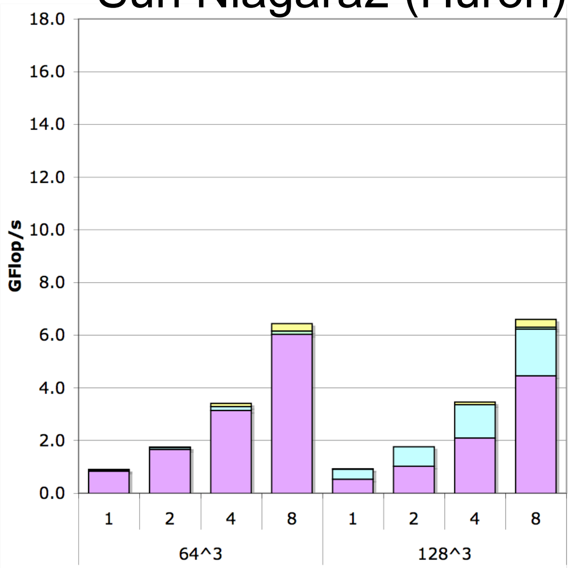


AMD Opteron

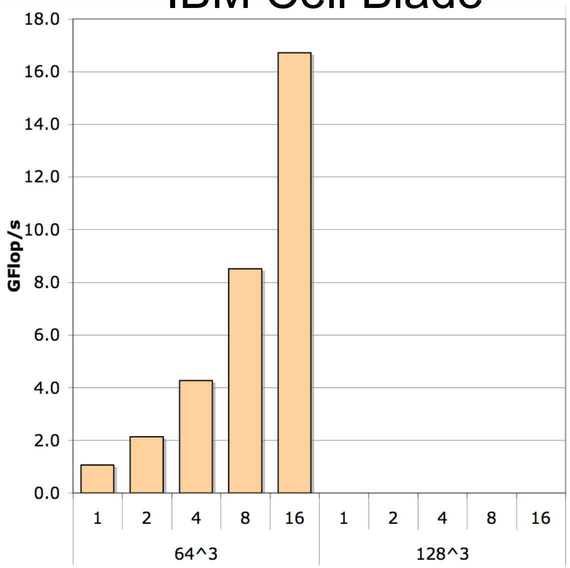


- ❖ First attempt at cell implementation.
- ❖ VL, unrolling, reordering fixed
- ❖ Exploits DMA and double buffering to load vectors
- ❖ Straight to SIMD intrinsics.
- ❖ Despite the relative performance, Cell's DP implementation severely impairs performance

Sun Niagara2 (Huron)



IBM Cell Blade*

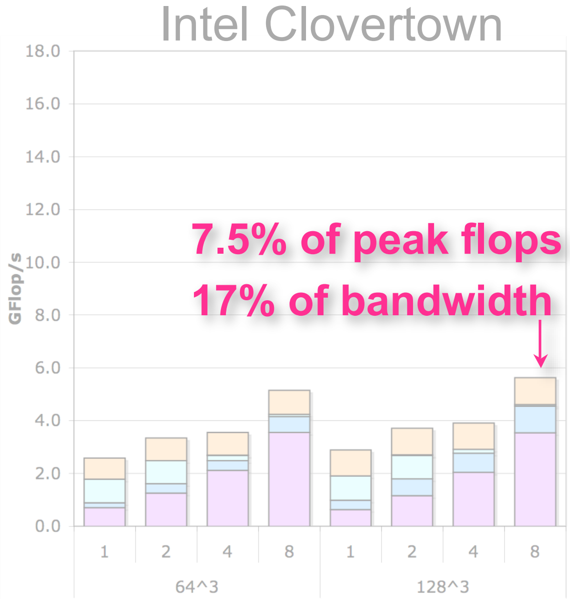


- ▨ +SIMDization
- ▨ +SW Prefetching
- ▨ +Unrolling
- ▨ +Vectorization
- ▨ +Padding
- ▨ Naïve+NUMA

*collision() only

Autotuned Performance

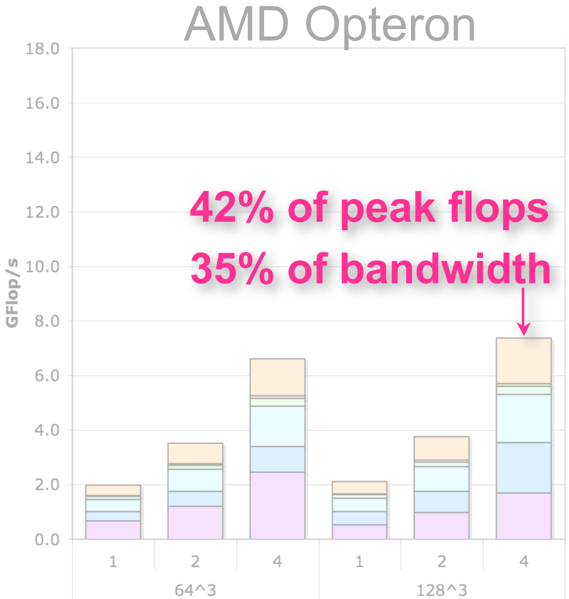
(Cell/SPE version)



Intel Clovertown

7.5% of peak flops

17% of bandwidth

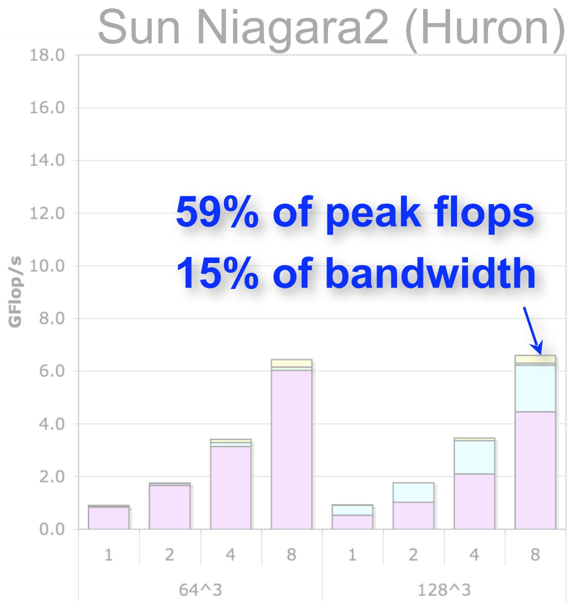


AMD Opteron

42% of peak flops

35% of bandwidth

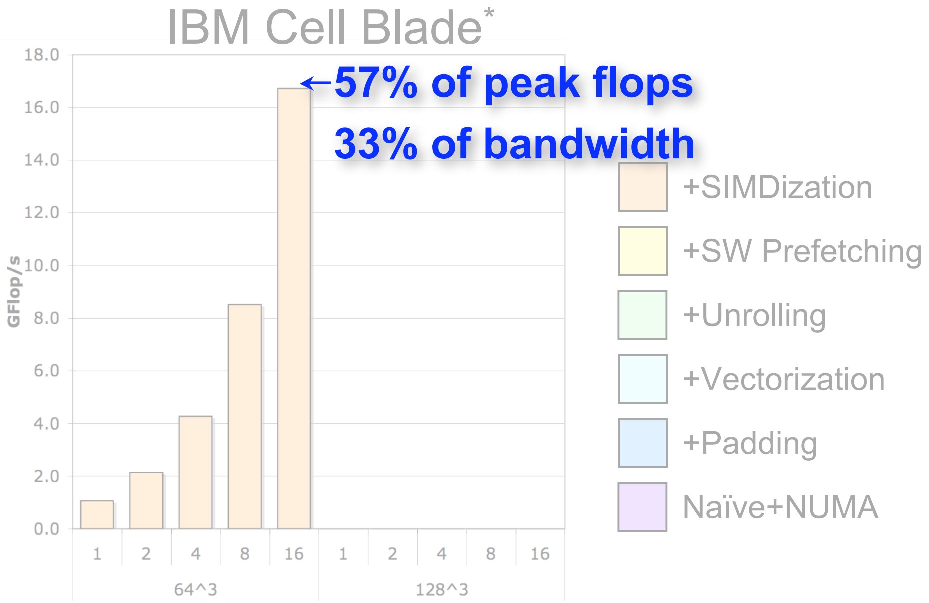
- First attempt at cell implementation.
- VL, unrolling, reordering fixed
- Exploits DMA and double buffering to load vectors
- Straight to SIMD intrinsics.
- Despite the relative performance, Cell's DP implementation severely impairs performance



Sun Niagara2 (Huron)

59% of peak flops

15% of bandwidth



IBM Cell Blade*

57% of peak flops

33% of bandwidth

*collision() only

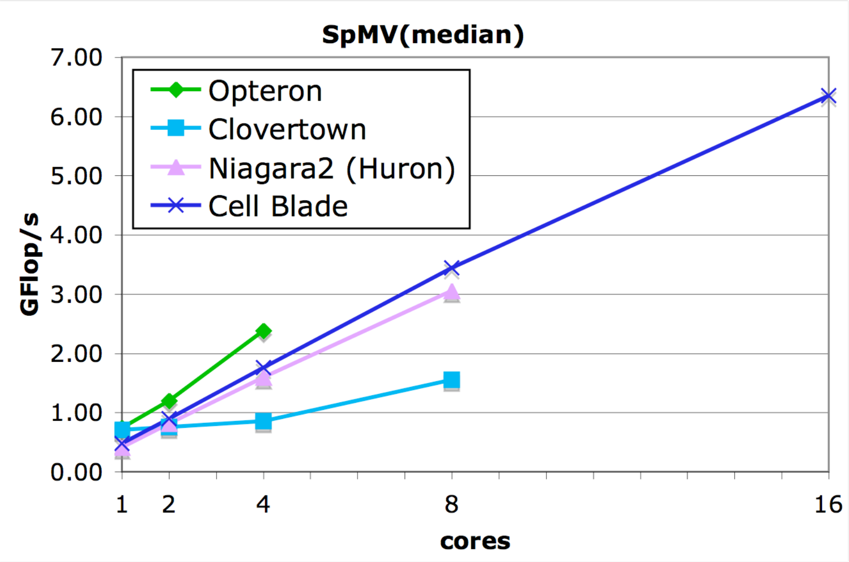
Autotuning
Multicore SMPs
HPC Kernels
SpMV
LBMHD
Summary

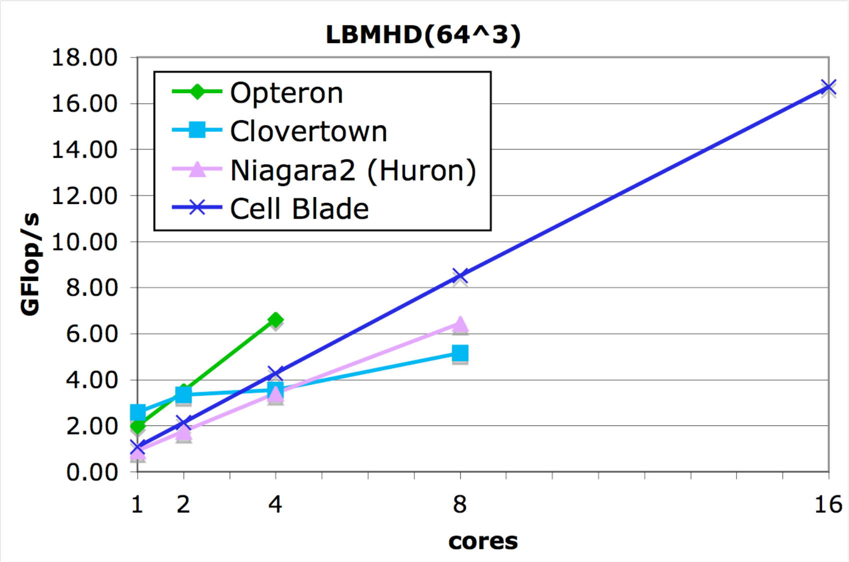
Summary

Aggregate Performance

(Fully optimized)

- Cell consistently delivers the best full system performance
- Niagara2 delivers comparable per socket performance
- Dual core Opteron delivers far better performance (bandwidth) than Clovertown, but as the flop:byte ratio increases its performance advantage decreases.
- Huron has far more bandwidth than it can exploit
  - (too much latency, too few cores)
- Clovertown has far too little effective FSB bandwidth

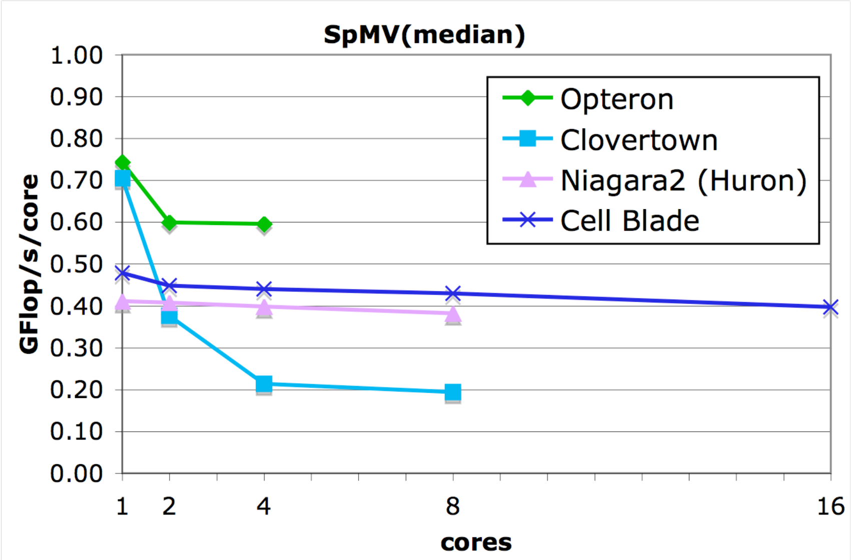


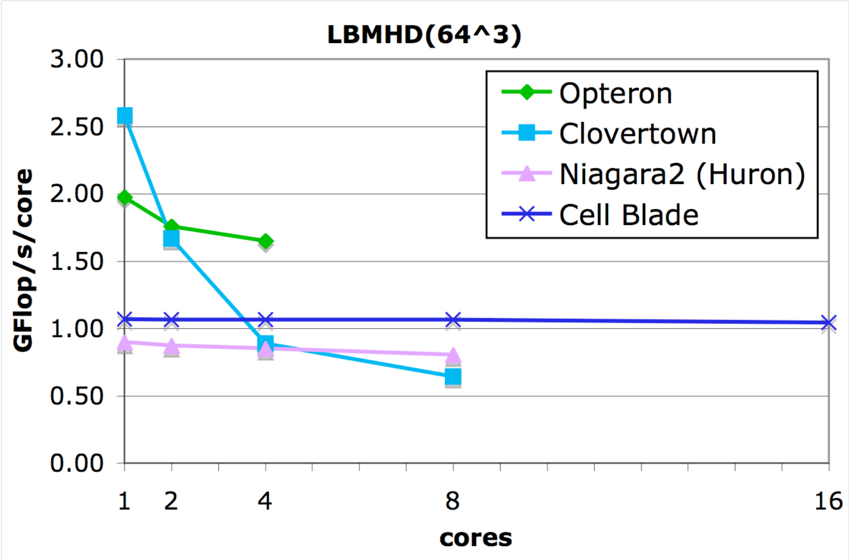


Parallel Efficiency

(average performance per thread, Fully optimized)

- Aggregate Mflop/s / #cores
- Niagara2 & Cell show very good multicore scaling
- Clovertown showed very poor multicore scaling on both applications
- For SpMV, Opteron and Clovertown showed good multisocket scaling
- Clovertown runs into bandwidth limits far short of its theoretical peak even for LBMHD
- Opteron lacks the bandwidth for SpMV, and the FP resources to use its bandwidth for LBMHD





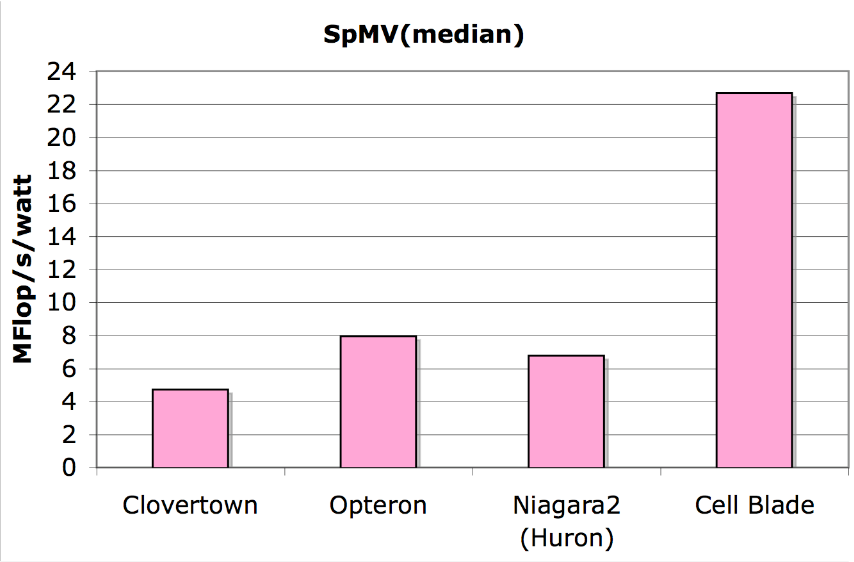
Power Efficiency

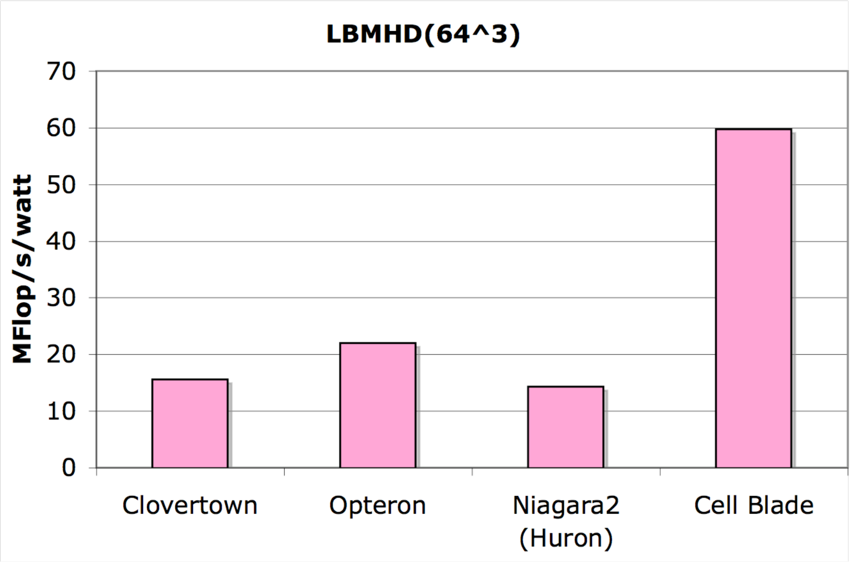
(Fully Optimized)

- Used a digital power meter to measure sustained power under load
- Calculate power efficiency as:

  sustained performance / sustained power

- All cache-based machines delivered similar power efficiency
- FBDIMMs (~12W each) sustained power
  - 8 DIMMs on Clovertown (total of ~330W)
  - 16 DIMMs on N2 machine (total of ~450W)





Productivity

- Niagara2 required significantly less work to deliver good performance.
- For LBMHD, Clovertown, Opteron, and Cell all required SIMD (hampers productivity) for best performance.
- Virtually every optimization was required (sooner or later) for Opteron and Cell.
- Cache based machines required search for some optimizations, while cell always relied on heuristics

Summary

- **Paradoxically**, the most complex/advanced architectures required the most tuning, and delivered the lowest performance.

- Niagara2 delivered both very good performance and productivity
- Cell delivered very good performance and efficiency (processor and power)

- **Our multicore specific autotuned SpMV implementation significantly outperformed an autotuned MPI implementation**
- **Our multicore autotuned LBMHD implementation significantly outperformed the already optimized serial implementation**

- Sustainable memory bandwidth is essential even on kernels with moderate computational intensity (flop:byte ratio)
- Architectural transparency is invaluable in optimizing code

Acknowledgements

- UC Berkeley
  - RADLab Cluster (Opterons)
  - PSI cluster(Clovertowns)
- Sun Microsystems
  - Niagara2 access
- Forschungszentrum Jülich
  - Cell blade cluster access

Questions?

Backup Slides