Lattice Boltzmann Simulation Optimization on Leading Multicore Platforms

Samuel Williams[1,2], Jonathan Carter[2],
Leonid Oliker[1,2], John Shalf[2], Katherine Yelick[1,2]

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

samw@eecs.berkeley.edu

Office of Science
U.S. DEPARTMENT OF ENERGY

Motivation

- Multicore is the de facto solution for improving peak performance for the next decade
- How do we ensure this applies to sustained performance as well ?
- Processor architectures are extremely diverse and compilers can rarely fully exploit them
- Require a HW/SW solution that guarantees performance without completely sacrificing productivity

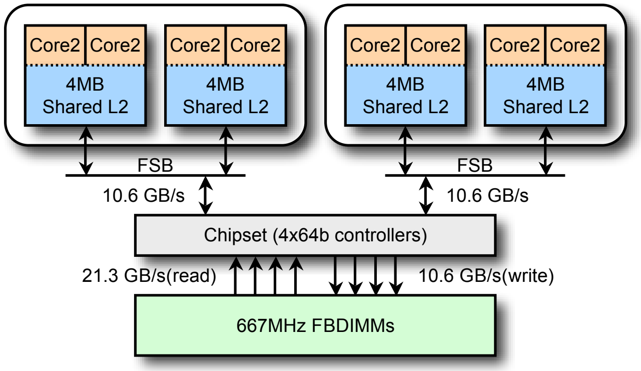
Overview

- Examined the Lattice-Boltzmann Magneto-hydrodynamic (**LBMHD**) application
- Present and analyze two threaded & auto-tuned implementations
- Benchmarked performance across 5 diverse multicore microarchitectures
  - Intel Xeon (Clovertown)
  - AMD Opteron (rev.F)
  - Sun Niagara2 (Huron)
  - IBM QS20 Cell Blade (PPEs)
  - IBM QS20 Cell Blade (SPEs)
- We show
  - Auto-tuning can significantly improve application performance
  - Cell consistently delivers good performance and efficiency
  - Niagara2 delivers good performance and productivity

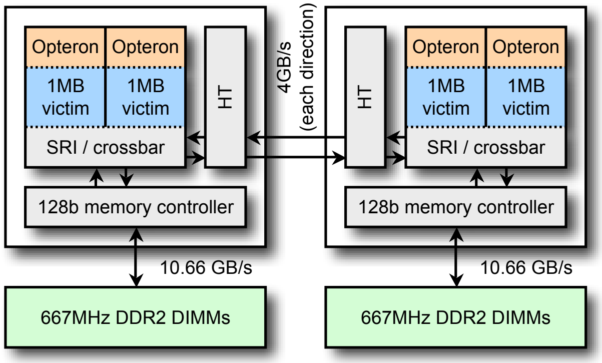
Multicore SMPs used

Multicore SMP Systems

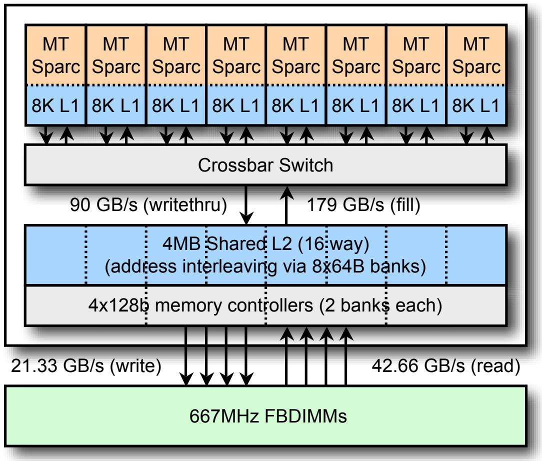
Intel Xeon (Clovertown)



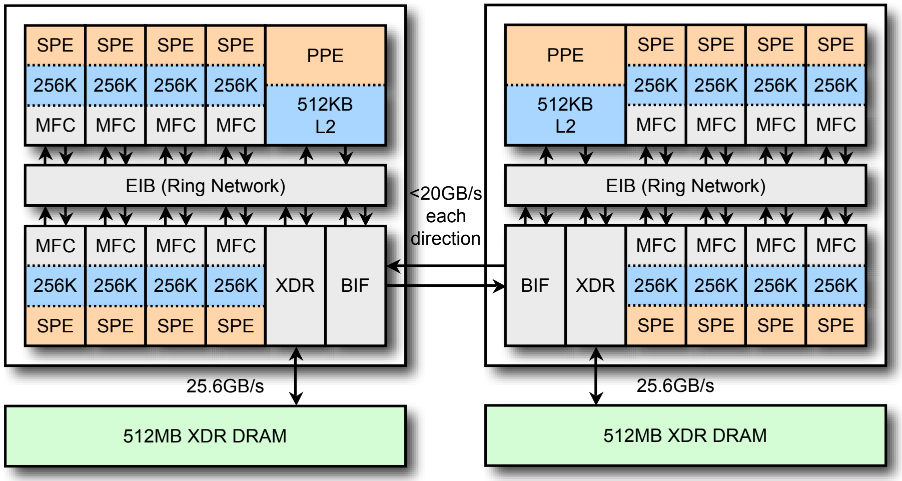
AMD Opteron (rev.F)



Sun Niagara2 (Huron)



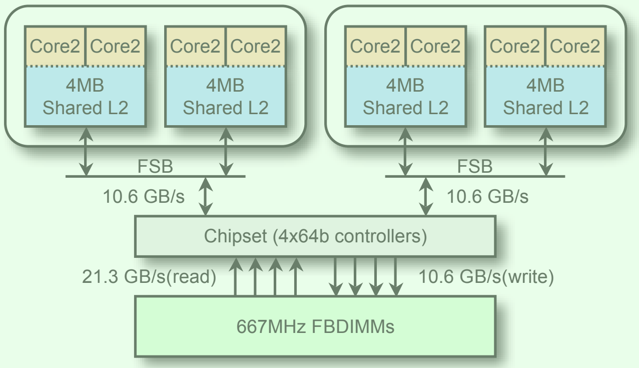
IBM QS20 Cell Blade



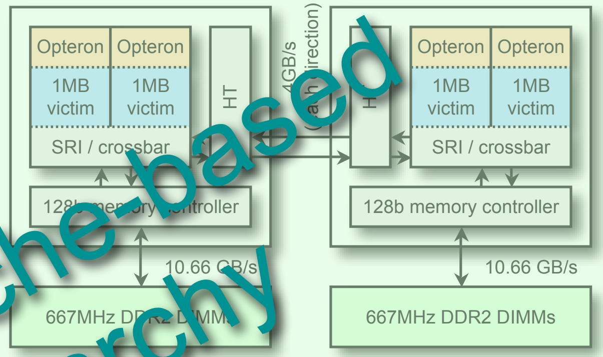
Multicore SMP Systems

(memory hierarchy)

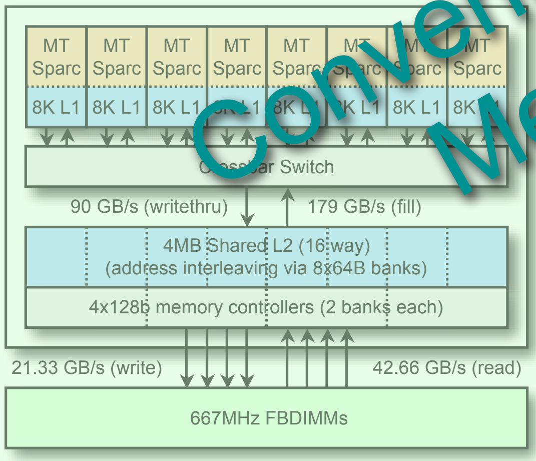
Intel Xeon (Clovertown)



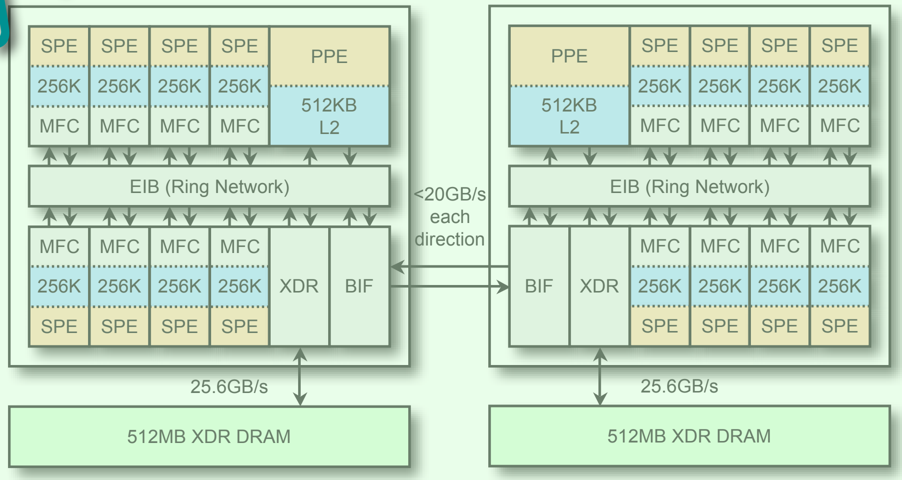
AMD Opteron (rev.F)



Sun Niagara2 (Huron)



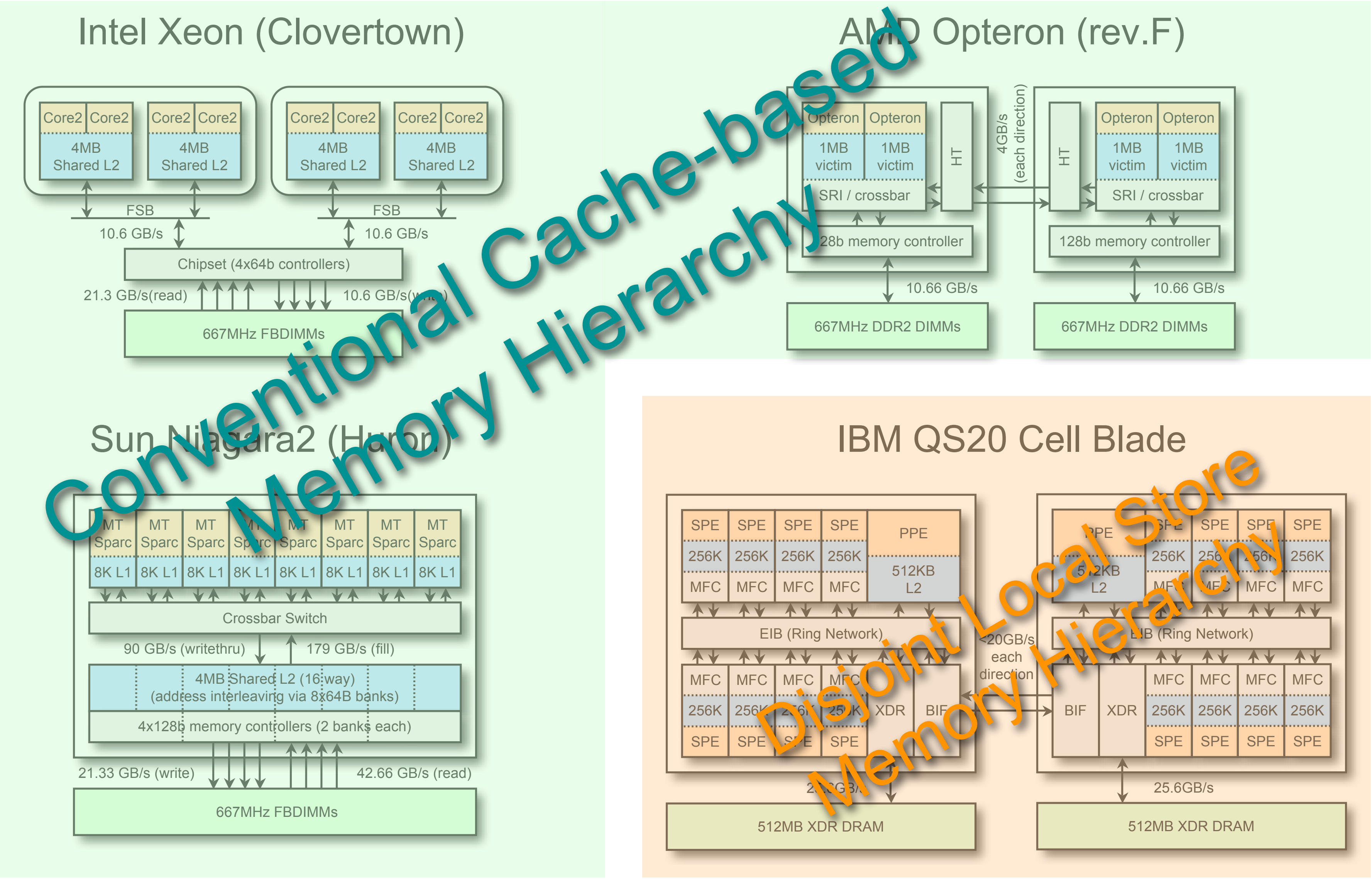
IBM QS20 Cell Blade



Conventional Cache-based Memory Hierarchy

Multicore SMP Systems

(memory hierarchy)



Conventional Cache-based Memory Hierarchy

Disjoint Local Store Memory Hierarchy

Intel Xeon (Clovertown)

Core2 Core2 4MB Shared L2 | Core2 Core2 4MB Shared L2 | Core2 Core2 4MB Shared L2 | Core2 Core2 4MB Shared L2

FSB 10.6 GB/s | FSB 10.6 GB/s

Chipset (4x64b controllers)

21.3 GB/s(read) | 10.6 GB/s(write)

667MHz FBDIMMs

AMD Opteron (rev.F)

Opteron Opteron 1MB victim 1MB victim SRI / crossbar HT 128b memory controller

4GB/s (each direction)

HT Opteron Opteron 1MB victim 1MB victim SRI / crossbar 128b memory controller

10.66 GB/s | 10.66 GB/s

667MHz DDR2 DIMMs | 667MHz DDR2 DIMMs

Sun Niagara2 (Huron)

MT Sparc 8K L1 (×8)

Crossbar Switch

90 GB/s (writethru) | 179 GB/s (fill)

4MB Shared L2 (16 way) (address interleaving via 8x64B banks)

4x128b memory controllers (2 banks each)

21.33 GB/s (write) | 42.66 GB/s (read)

667MHz FBDIMMs

IBM QS20 Cell Blade

SPE 256K MFC (×4) | PPE 512KB L2 | EIB (Ring Network) | MFC 256K SPE (×4) | XDR | BIF

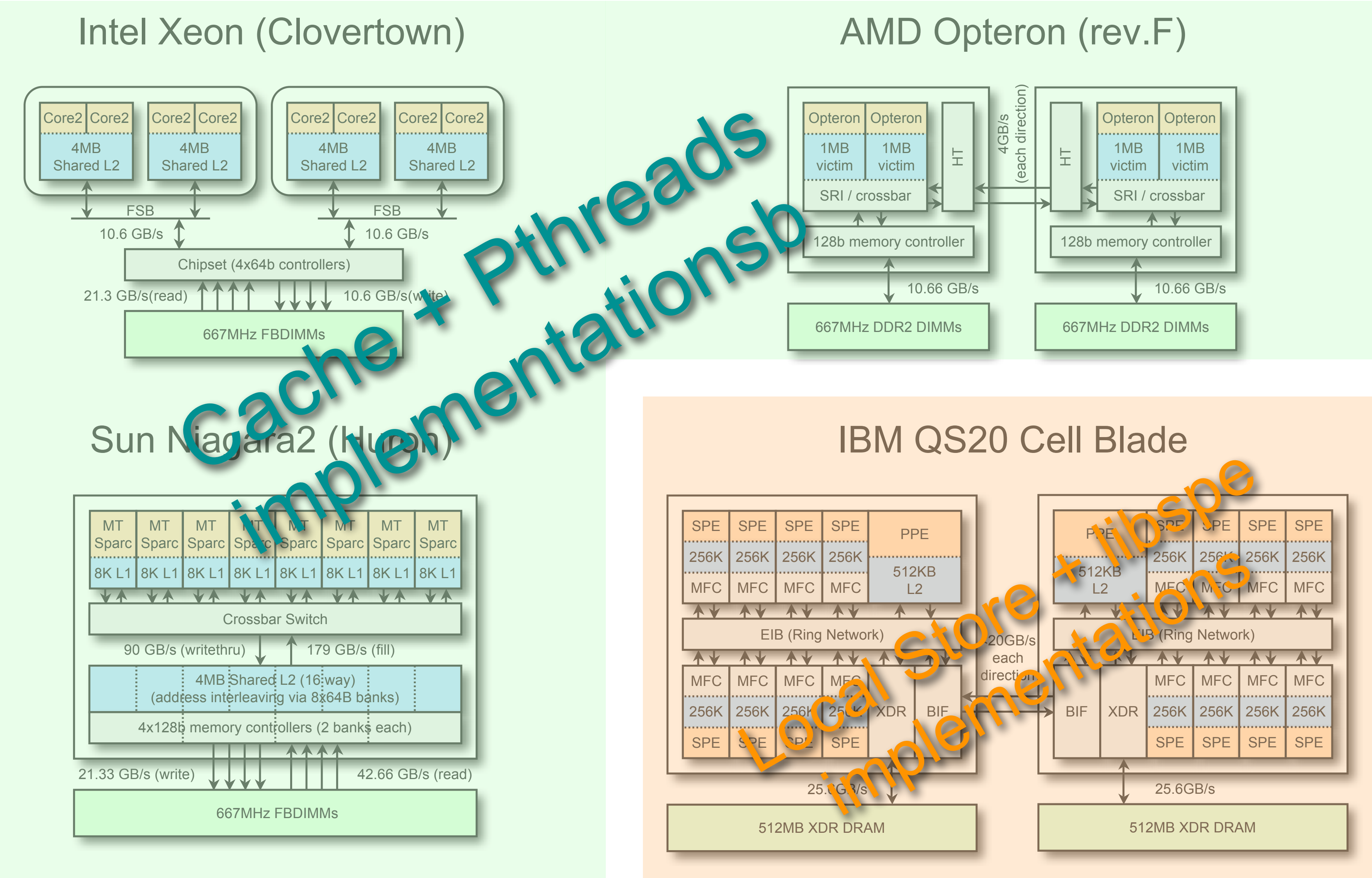
<20GB/s each direction

PPE 512KB L2 | SPE 256K MFC (×4) | EIB (Ring Network) | BIF | XDR | MFC 256K SPE (×4)

25.6GB/s | 25.6GB/s

512MB XDR DRAM | 512MB XDR DRAM

Multicore SMP Systems
(memory hierarchy)



Intel Xeon (Clovertown)

AMD Opteron (rev.F)

Sun Niagara2 (Huron)

IBM QS20 Cell Blade

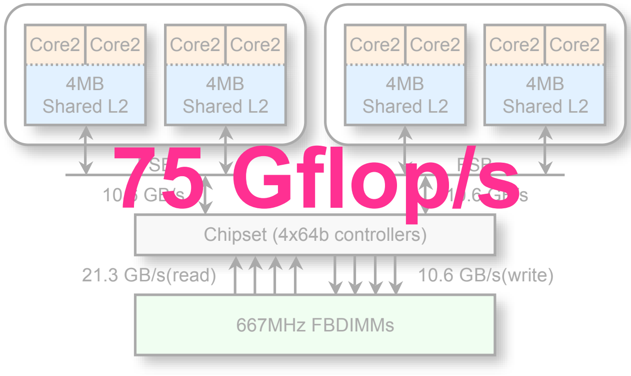
Cache + Pthreads implementationsb

Local Store + libspe implementations

Multicore SMP Systems

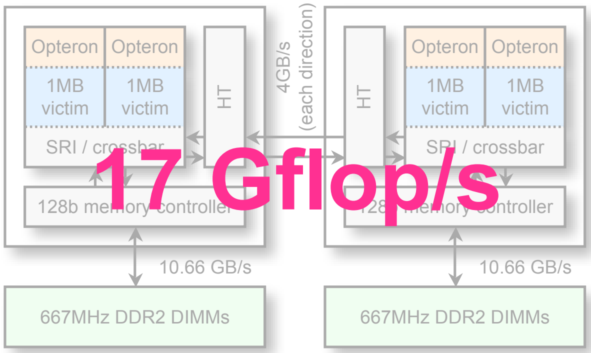
(peak flops)

Intel Xeon (Clovertown)



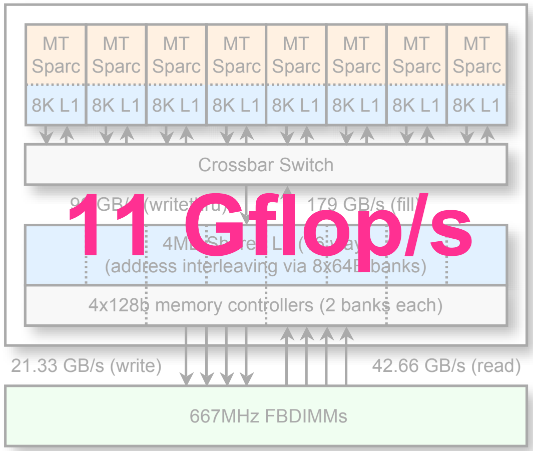
75 Gflop/s

AMD Opteron (rev.F)



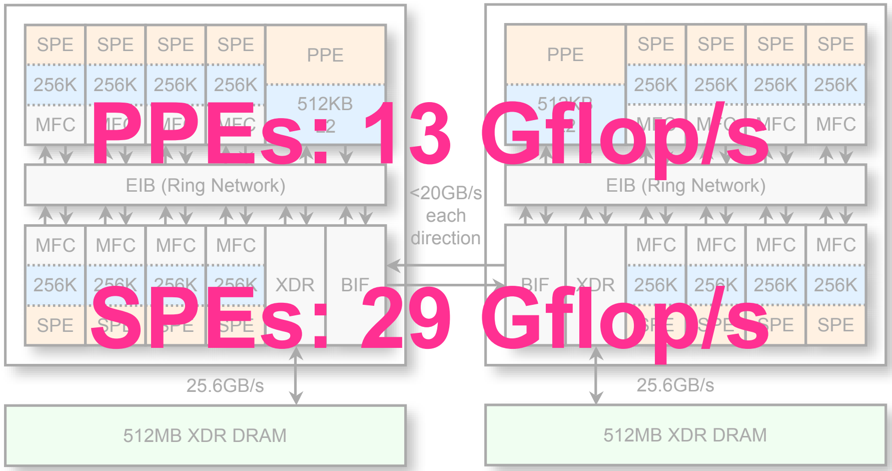
17 Gflop/s

Sun Niagara2 (Huron)



11 Gflop/s

IBM QS20 Cell Blade



PPEs: 13 Gflop/s

SPEs: 29 Gflop/s

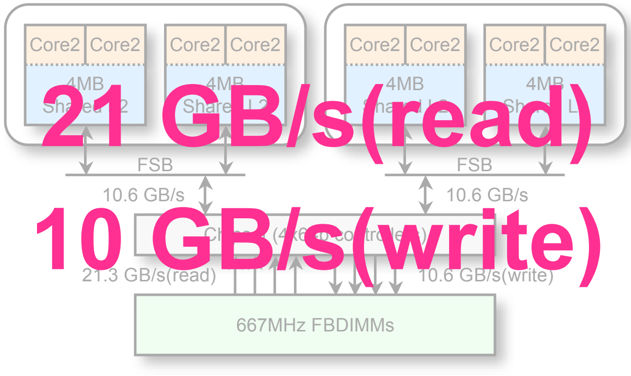
BIPS

Multicore SMP Systems

(peak DRAM bandwidth)

BERKELEY LAB

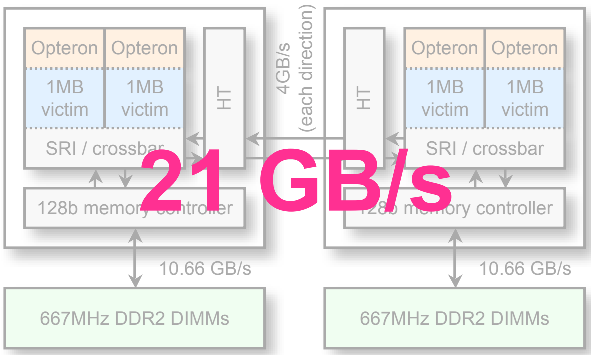
Intel Xeon (Clovertown)



21 GB/s(read)

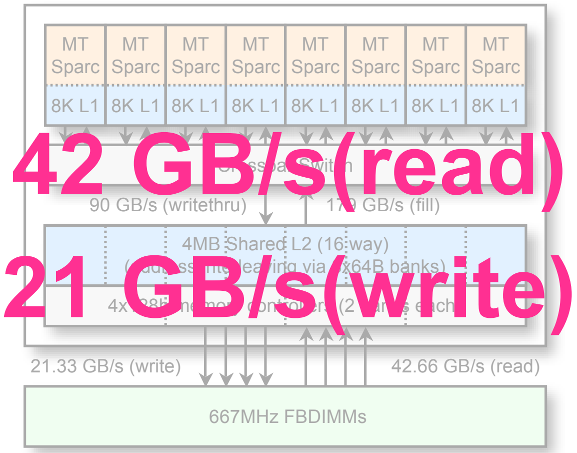
10 GB/s(write)

AMD Opteron (rev.F)



21 GB/s

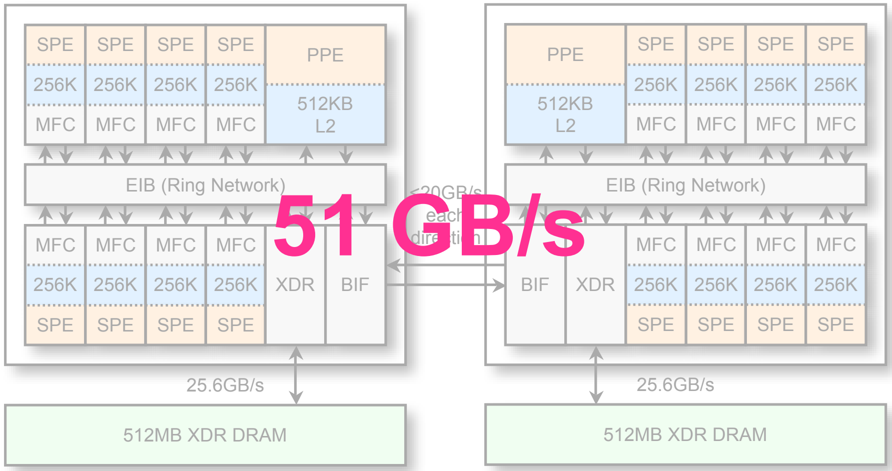
Sun Niagara2 (Huron)



42 GB/s(read)

21 GB/s(write)

IBM QS20 Cell Blade



51 GB/s

Auto-tuning

Auto-tuning

- Hand optimizing each architecture/dataset combination is not feasible

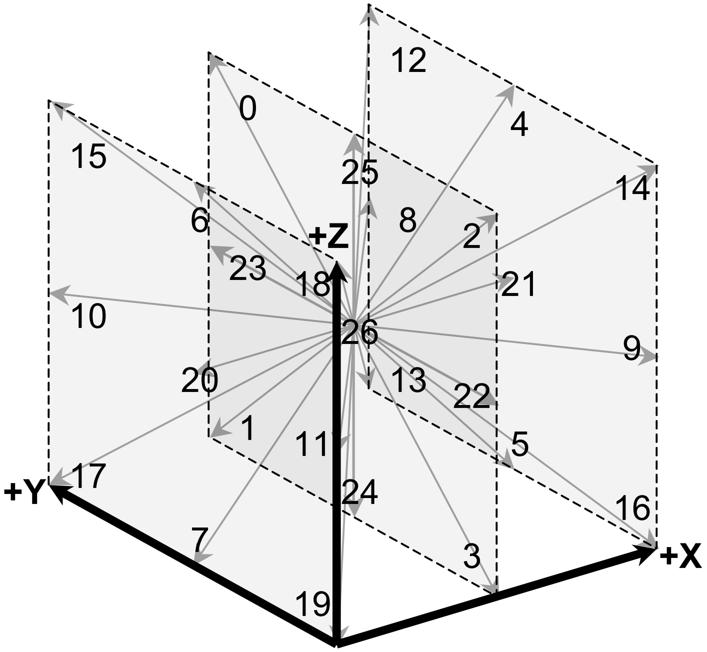
- Our auto-tuning approach finds a good performance solution by a combination of heuristics and exhaustive search
  - Perl script generates many possible kernels
  - (Generate SIMD optimized kernels)
  - Auto-tuning benchmark examines kernels and reports back with the best one for the current architecture/dataset/compiler/…
  - Performance depends on the optimizations generated
  - Heuristics are often desirable when the search space isn't tractable

- Proven value in Dense Linear Algebra(ATLAS), Spectral(FFTW,SPIRAL), and Sparse Methods(OSKI)

Introduction to LBMHD

Introduction to Lattice Methods

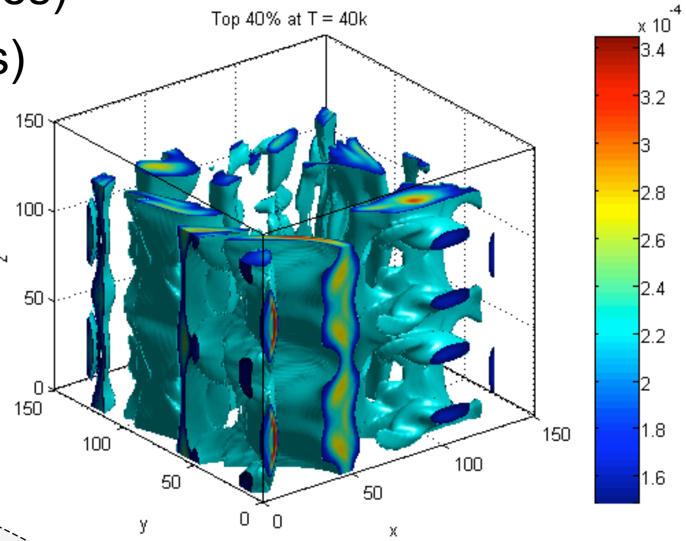
- Structured grid code, with a series of time steps
- Popular in CFD (allows for complex boundary conditions)
- Overlay a higher dimensional phase space
  - Simplified kinetic model that maintains the macroscopic quantities
  - Distribution functions (e.g. 5-27 velocities per point in space) are used to reconstruct macroscopic quantities
  - Significant Memory capacity requirements

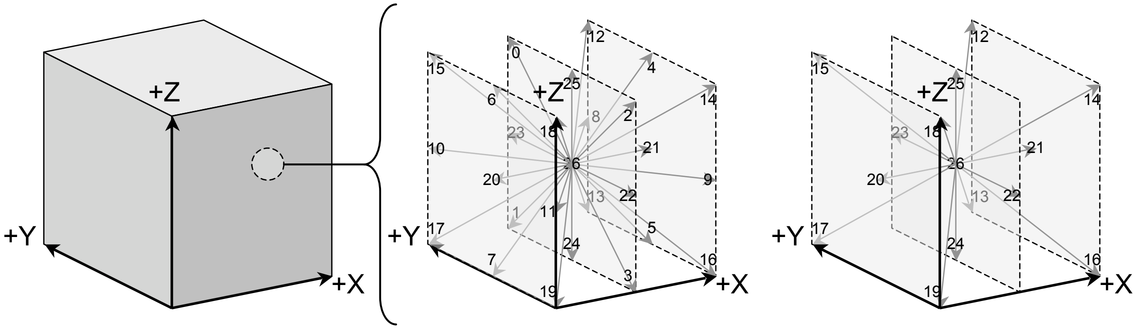


LBMHD

(general characteristics)

- Plasma turbulence simulation
- Couples CFD with Maxwell's equations
- Two distributions:
  - momentum distribution (27 scalar velocities)
  - magnetic distribution (15 vector velocities)
- Three macroscopic quantities:
  - Density
  - Momentum (vector)
  - Magnetic Field (vector)





macroscopic variables

momentum distribution

magnetic distribution

LBMHD
(flops and bytes)

- Must read 73 doubles, and update 79 doubles per point in space (minimum 1200 bytes)
- Requires about 1300 floating point operations per point in space

- Flop:Byte ratio
  - 0.71 (write allocate architectures)
  - 1.07 (ideal)

- Rule of thumb for LBMHD:
  - Architectures with more flops than bandwidth are likely memory bound (e.g. Clovertown)

LBMHD

(implementation details)

- Data Structure choices:
  - **Array of Structures**: no spatial locality, strided access
  - **Structure of Arrays:** huge number of memory streams per thread, but guarantees spatial locality, unit-stride, and vectorizes well

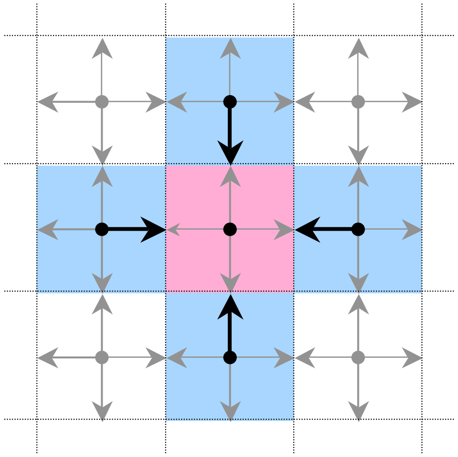
- Parallelization
  - Fortran version used MPI to communicate between tasks.

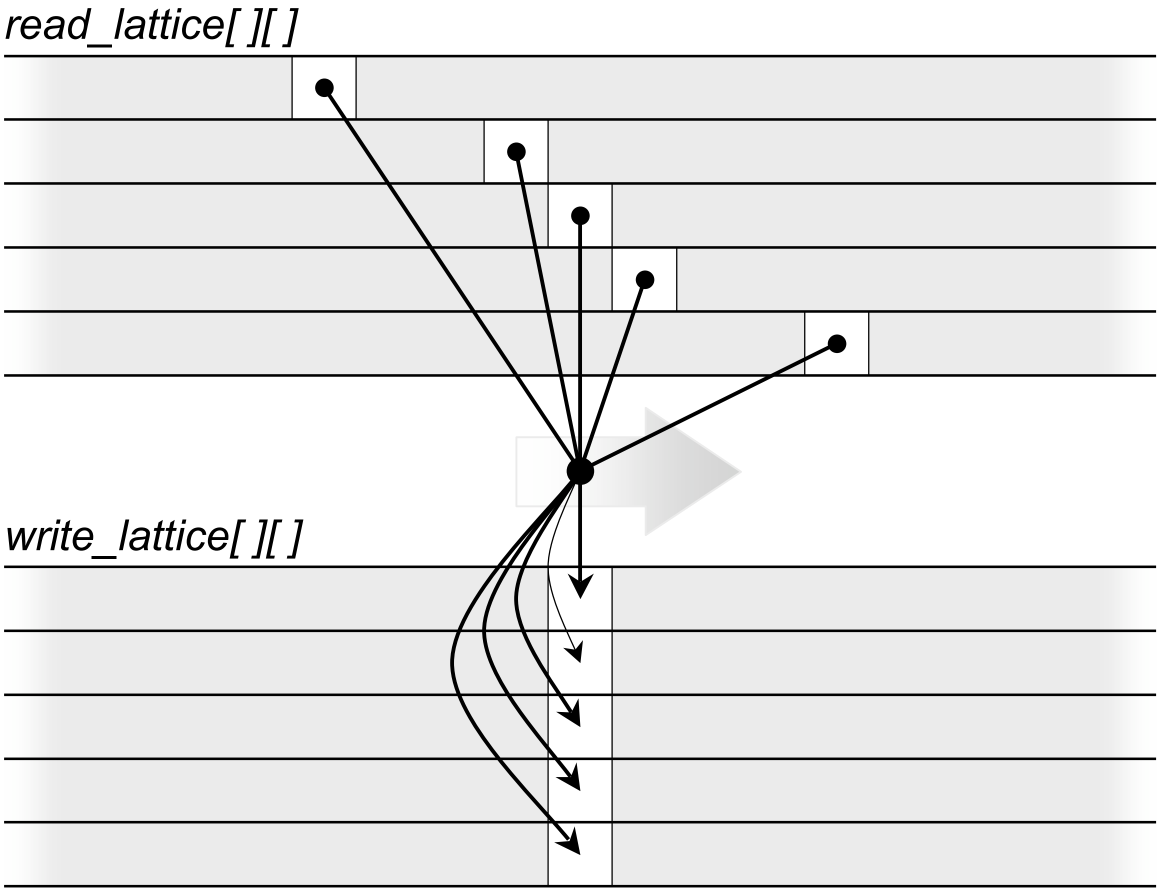
    = bad match for multicore
  - The version in this work uses pthreads for multicore, and MPI for inter-node
  - MPI is not used when auto-tuning

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

Stencil for Lattice Methods

- Very different the canonical heat equation stencil
  - There are multiple read and write arrays
  - There is no reuse





Side Note on Performance Graphs

- Threads are mapped first to cores, then sockets.

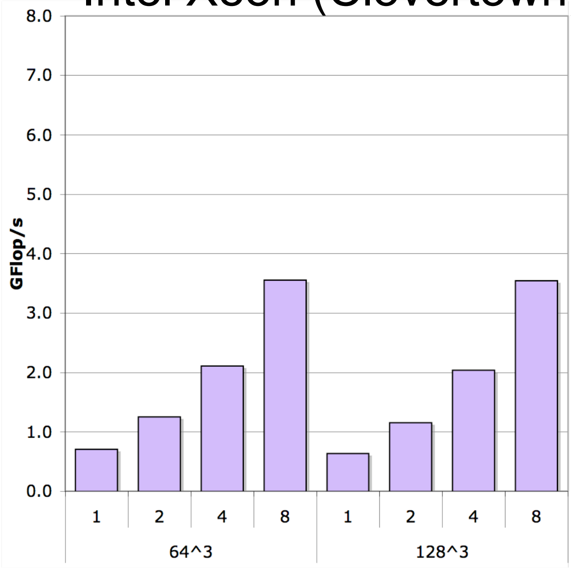
  i.e. multithreading, then multicore, then multisocket
- Niagara2 always used 8 threads/core.
- Show two problem sizes

- We'll step through performance as optimizations/features are enabled within the auto-tuner
- **More colors implies more optimizations were necessary**
- This allows us to compare architecture performance while keeping programmer effort(productivity) constant

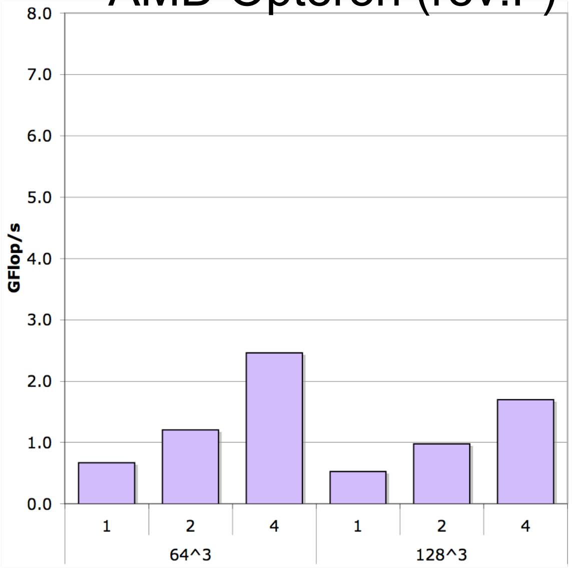
Performance and Analysis of Pthreads Implementation

Pthread Implementation

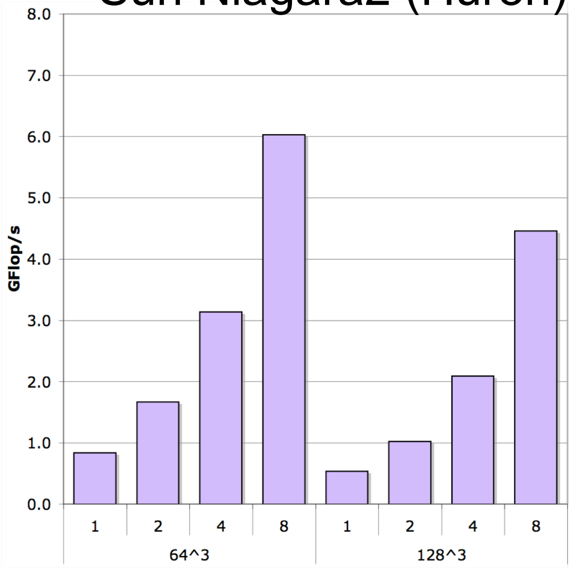
Intel Xeon (Clovertown)



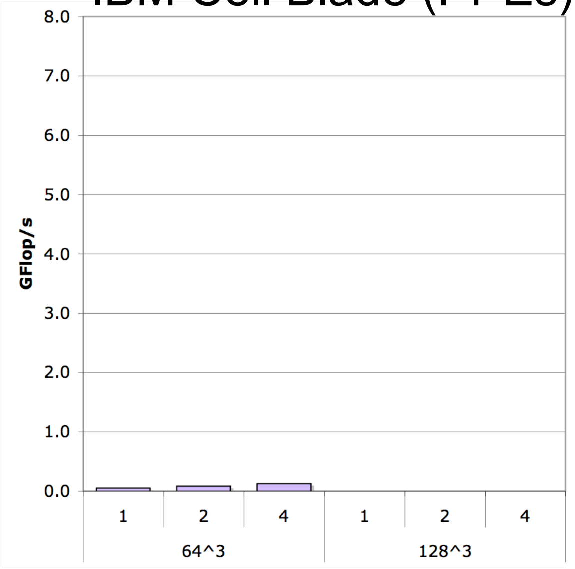
AMD Opteron (rev.F)



Sun Niagara2 (Huron)



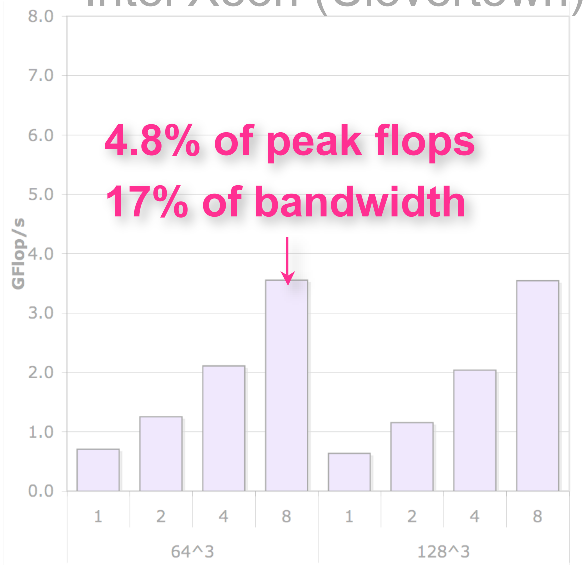
IBM Cell Blade (PPEs)



- Not naïve
  - fully unrolled loops
  - NUMA-aware
  - 1D parallelization
- Always used 8 threads per core on Niagara2
- 1P Niagara2 is faster than 2P x86 machines

Pthread Implementation

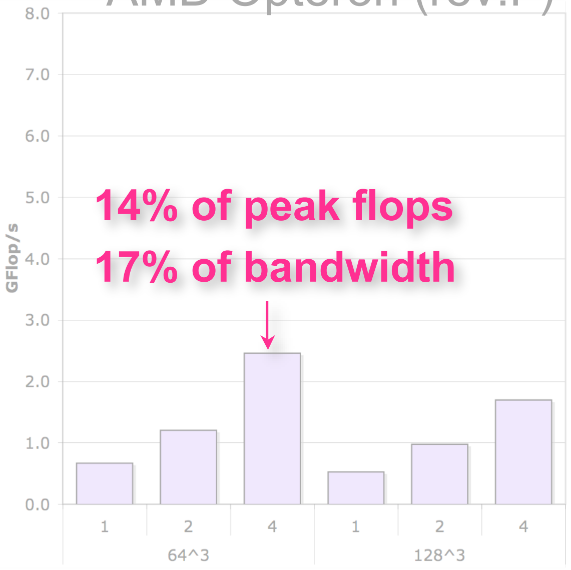
Intel Xeon (Clovertown)



4.8% of peak flops

17% of bandwidth

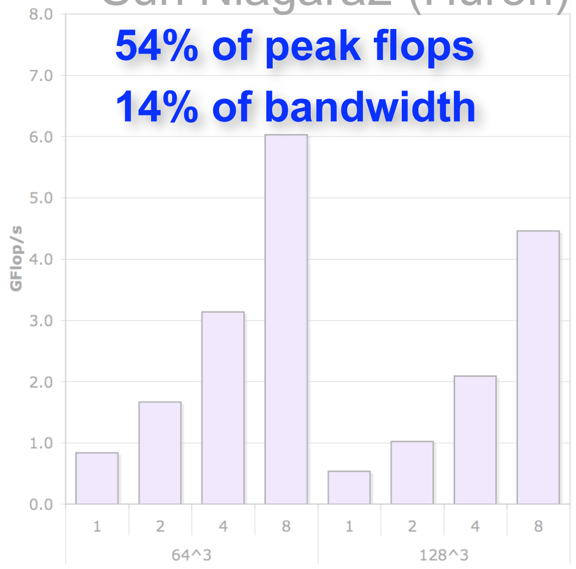
AMD Opteron (rev.F)



14% of peak flops

17% of bandwidth

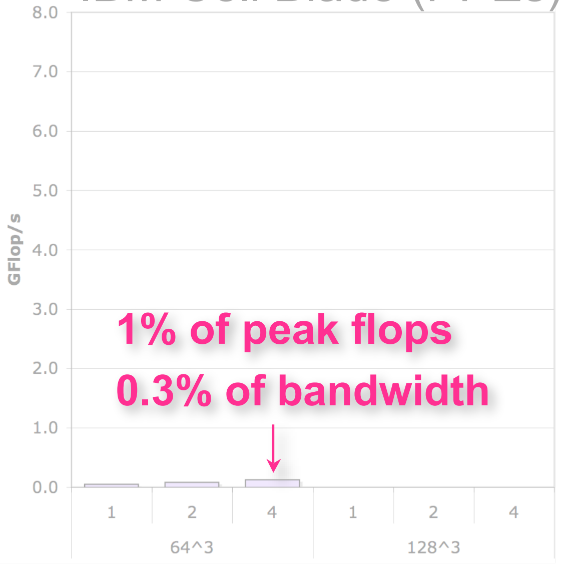
Sun Niagara2 (Huron)



54% of peak flops

14% of bandwidth

IBM Cell Blade (PPEs)

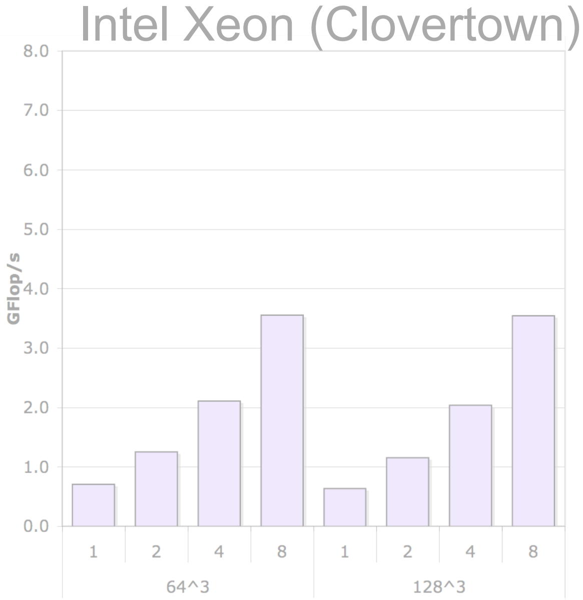


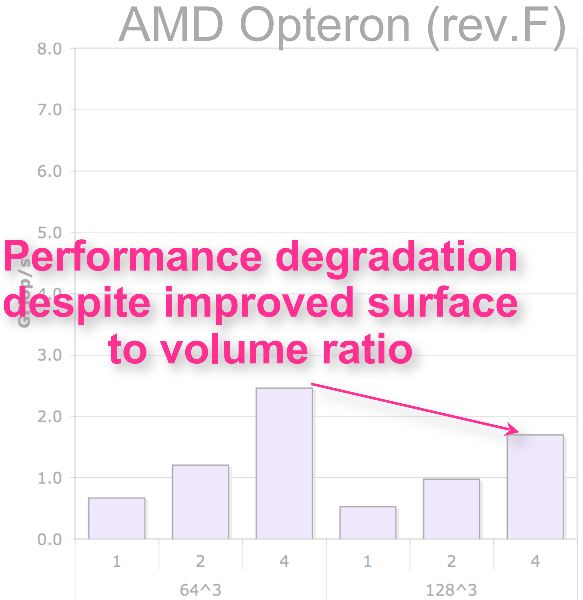
1% of peak flops

0.3% of bandwidth

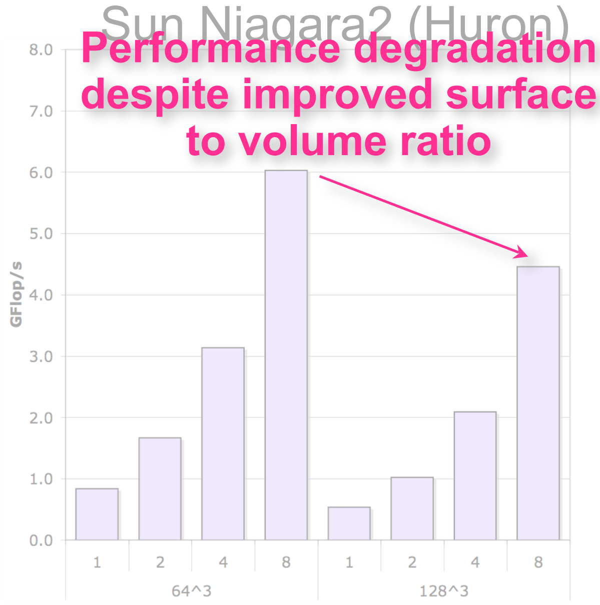
- Not naïve
  - fully unrolled loops
  - NUMA-aware
  - 1D parallelization
- Always used 8 threads per core on Niagara2
- 1P Niagara2 is faster than 2P x86 machines

Initial Pthread Implementation

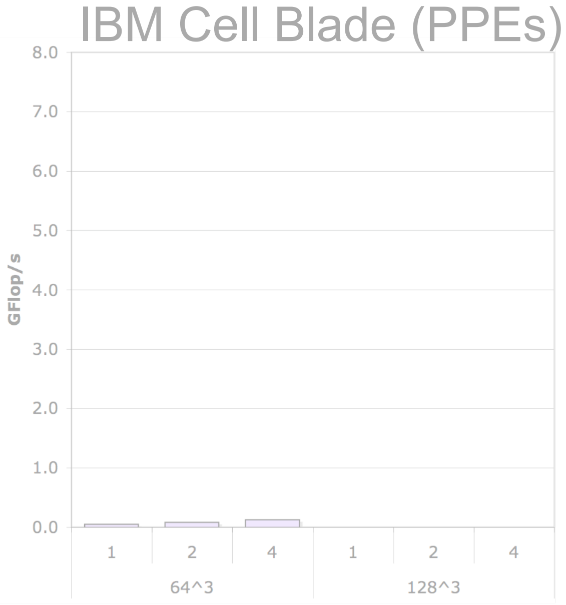




Performance degradation despite improved surface to volume ratio



Performance degradation despite improved surface to volume ratio



- Not naïve
  - fully unrolled loops
  - NUMA-aware
  - 1D parallelization
- Always used 8 threads per core on Niagara2
- 1P Niagara2 is faster than 2P x86 machines

Cache effects

- Want to maintain a working set of velocities in the L1 cache
- 150 arrays each trying to keep at least 1 cache line.

- Impossible with Niagara's 1KB/thread L1 working set = **capacity misses**

- On other architectures, the combination of:
  - Low associativity L1 caches (2 way on opteron)
  - Large numbers or arrays
  - Near power of 2 problem sizes

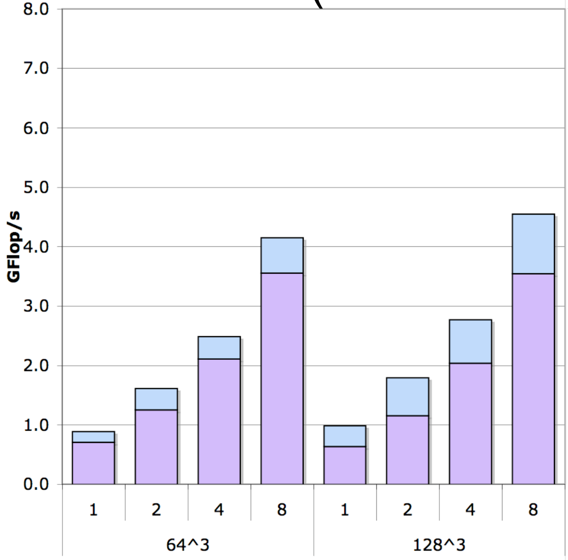
  can result in large numbers of **conflict misses**

- **Solution:** apply a lattice (offset) aware padding heuristic to the velocity arrays to avoid/minimize conflict misses

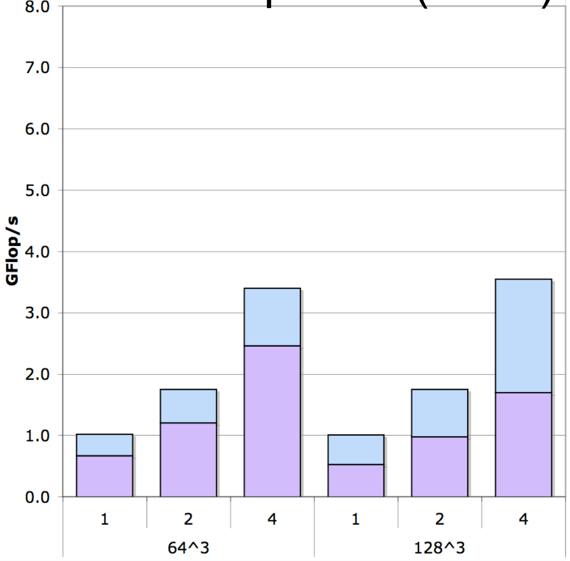
Auto-tuned Performance

(+Stencil-aware Padding)

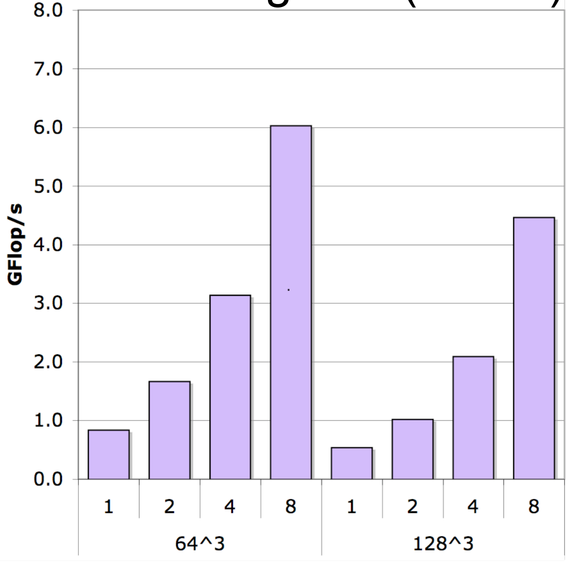
Intel Xeon (Clovertown)



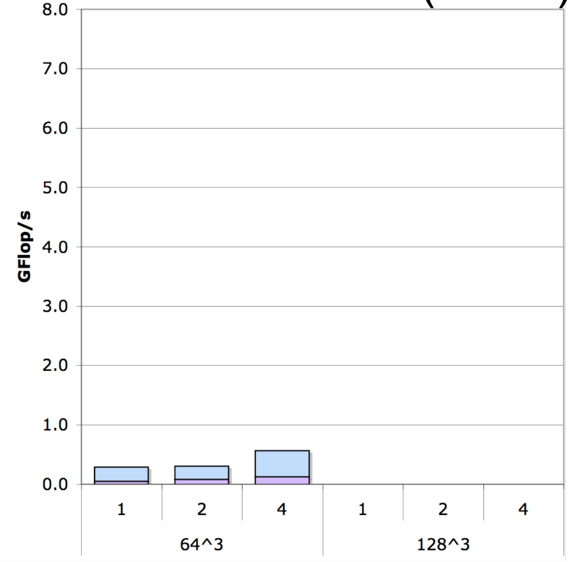
AMD Opteron (rev.F)



Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



- 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

Blocking for the TLB

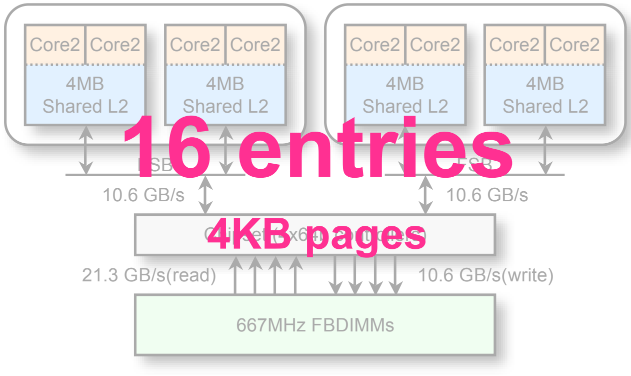
- Touching 150 different arrays will thrash TLBs with less than 128 entries.
- Try to maximize TLB page locality
- **Solution:** borrow a technique from compilers for vector machines:
  - Fuse spatial loops
  - Strip mine into vectors of size vector length (VL)
  - Interchange spatial and velocity loops
- Can be generalized by varying:
  - The number of velocities simultaneously accessed
  - The number of macroscopics / velocities simultaneously updated

- Has the side benefit expressing more ILP and DLP (SIMDization) and cleaner loop structure at the cost of increased L1 cache traffic

Multicore SMP Systems

(TLB organization)

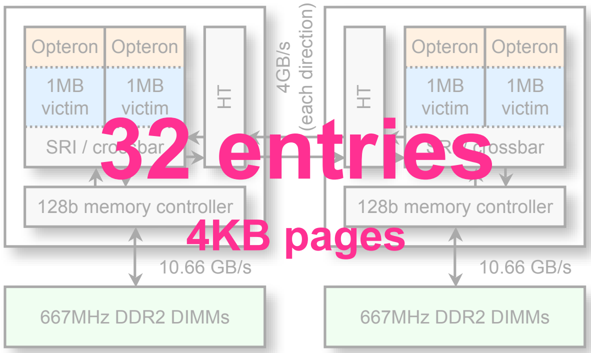
Intel Xeon (Clovertown)



16 entries

4KB pages

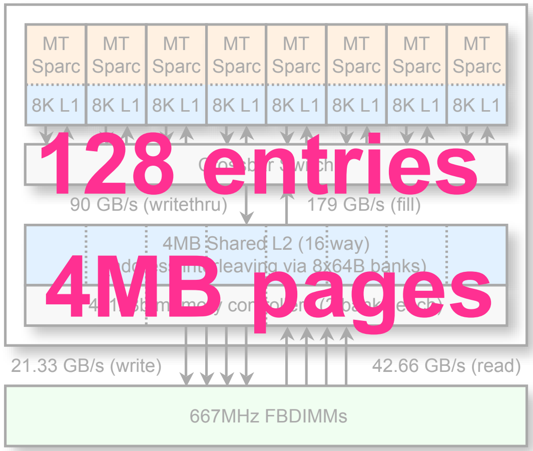
AMD Opteron (rev.F)



32 entries

4KB pages

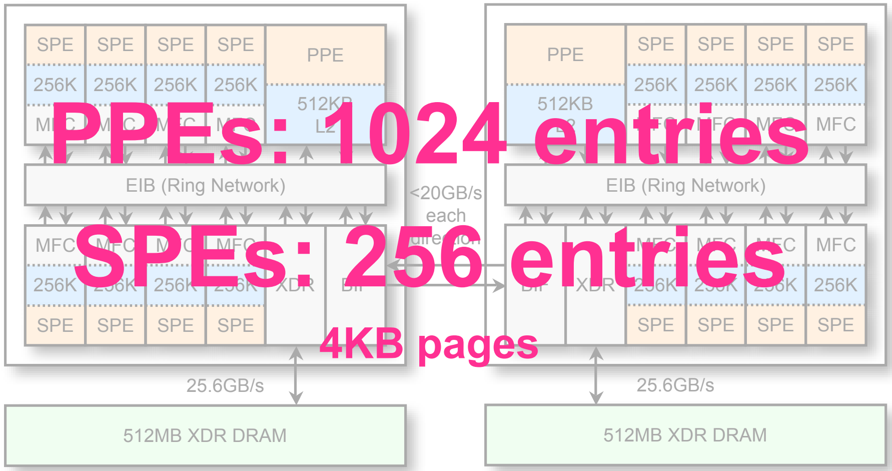
Sun Niagara2 (Huron)



128 entries

4MB pages

IBM QS20 Cell Blade



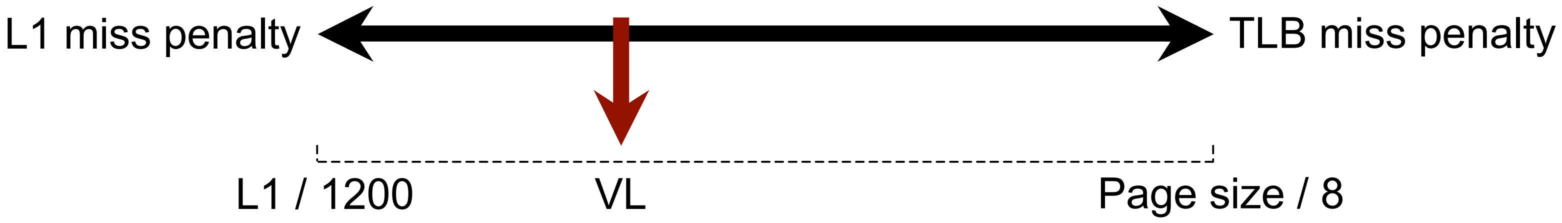
PPEs: 1024 entries

SPEs: 256 entries

4KB pages

Cache / TLB Tug-of-War

- For cache locality we want small VL
- For TLB page locality we want large VL
- Each architecture has a different balance between these two forces

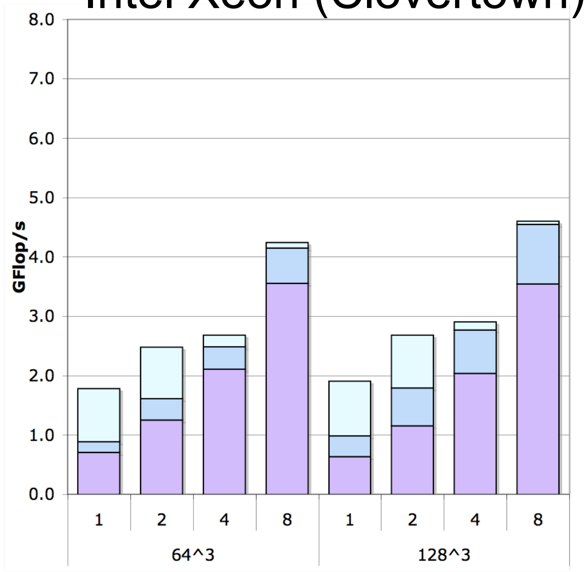


- **Solution**: auto-tune to find the optimal VL

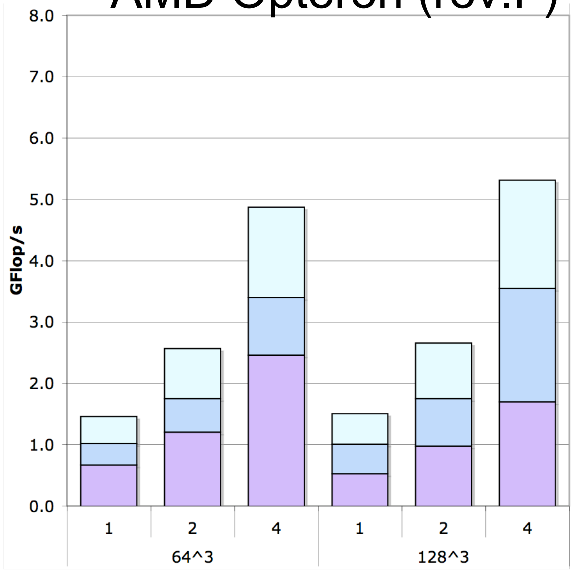
Auto-tuned Performance

(+Vectorization)

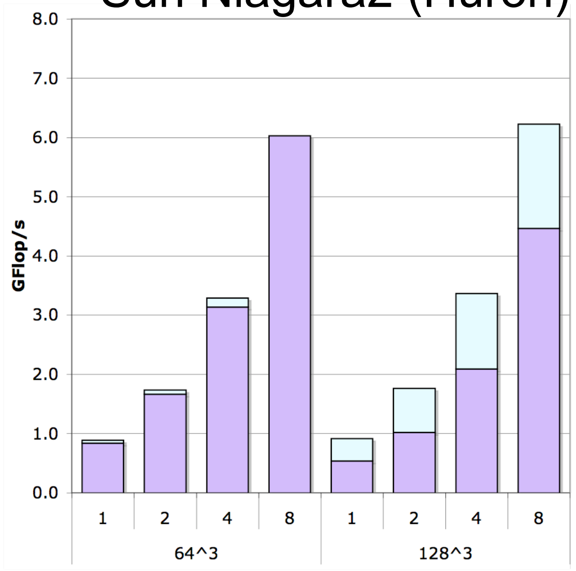
Intel Xeon (Clovertown)



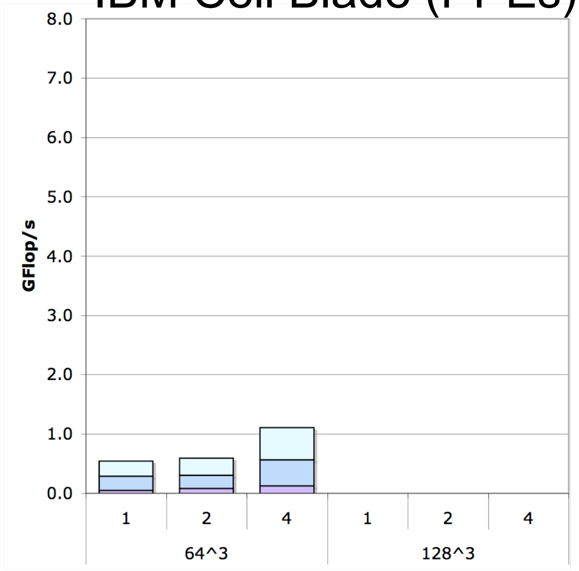
AMD Opteron (rev.F)



Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



- 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

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

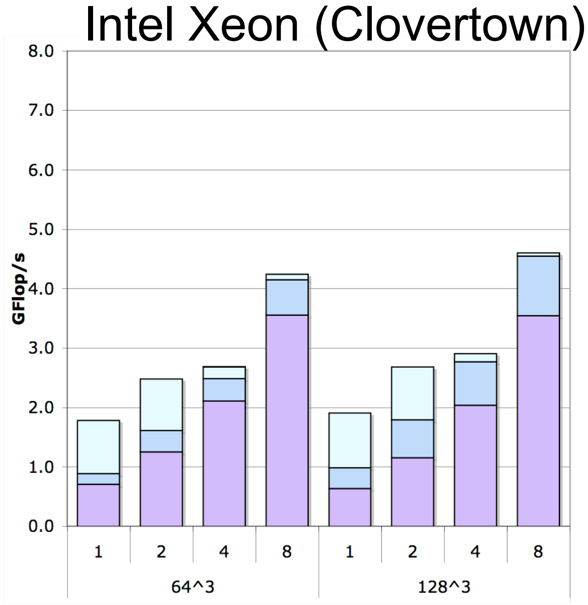
□ +Vectorization

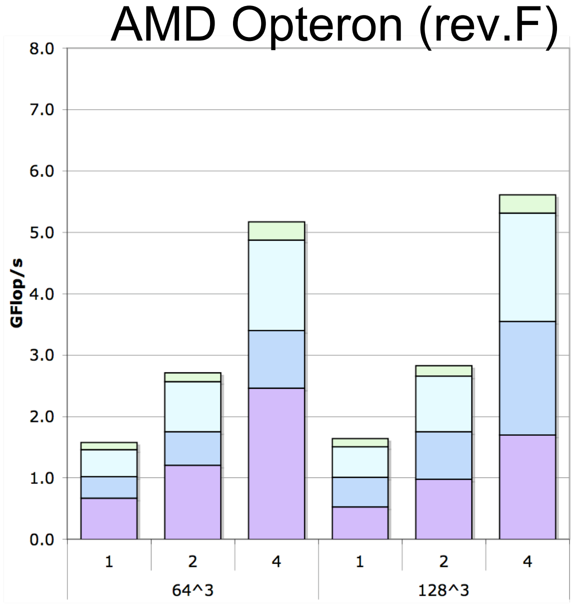
■ +Padding

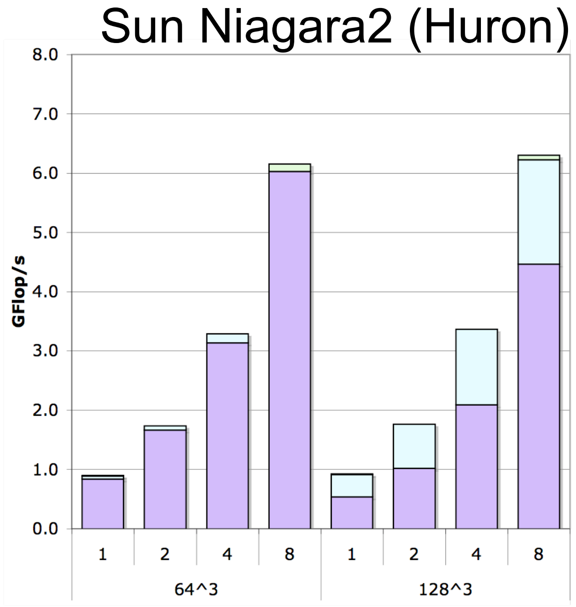
■ Naïve+NUMA

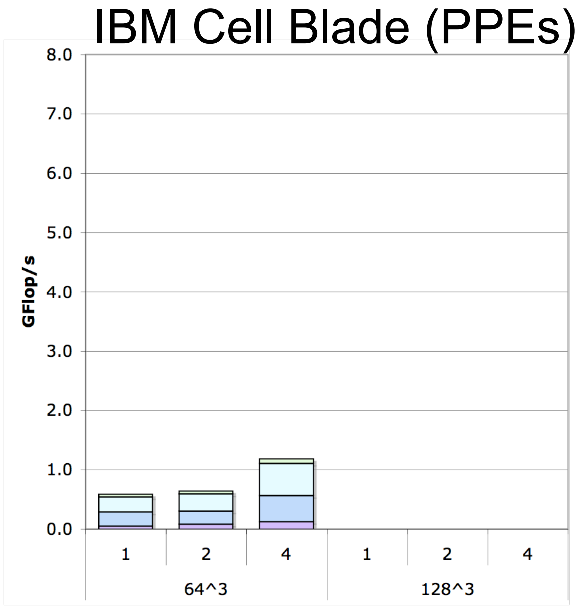
Auto-tuned Performance

(+Explicit Unrolling/Reordering)


Intel Xeon (Clovertown)


AMD Opteron (rev.F)


Sun Niagara2 (Huron)


IBM Cell Blade (PPEs)

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

🟩 +Unrolling

🟦 +Vectorization

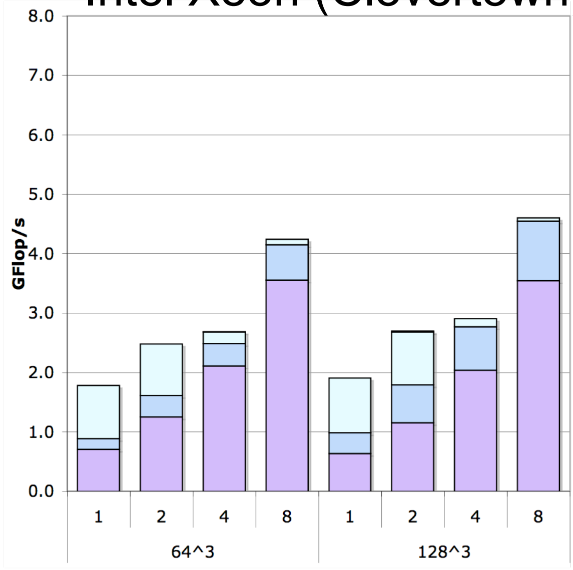
🟦 +Padding

🟪 Naïve+NUMA

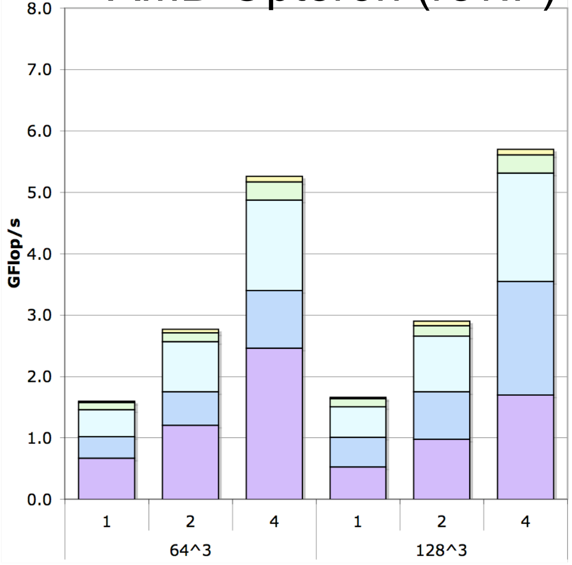
Auto-tuned Performance

(+Software prefetching)

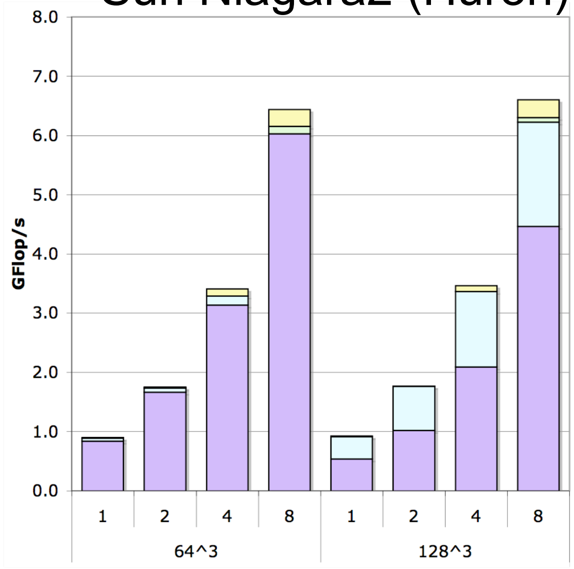
Intel Xeon (Clovertown)



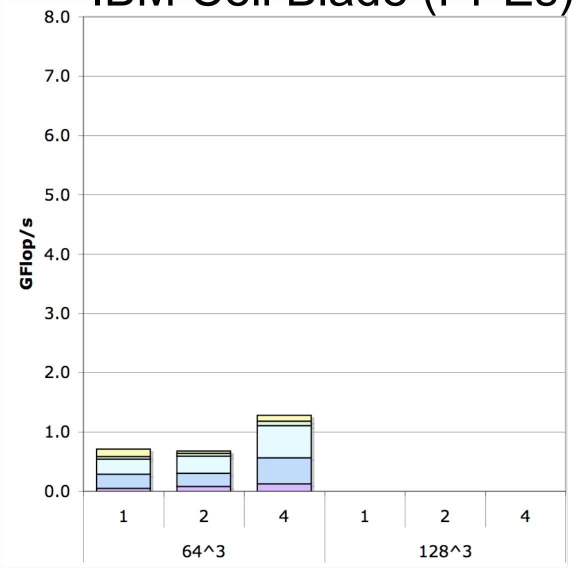
AMD Opteron (rev.F)



Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



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

🟨 +SW Prefetching

🟩 +Unrolling

⬜ +Vectorization

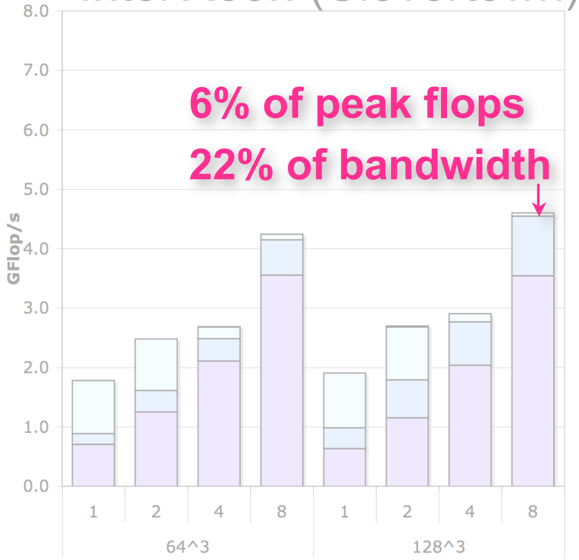
🟦 +Padding

🟪 Naïve+NUMA

Auto-tuned Performance

(+Software prefetching)

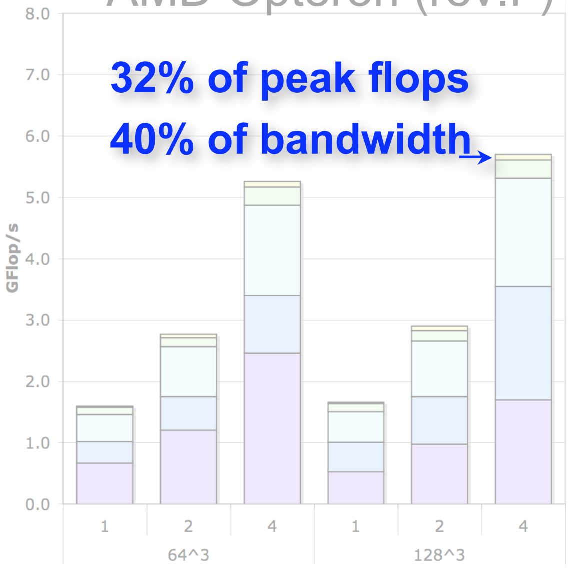
Intel Xeon (Clovertown)



6% of peak flops

22% of bandwidth

AMD Opteron (rev.F)

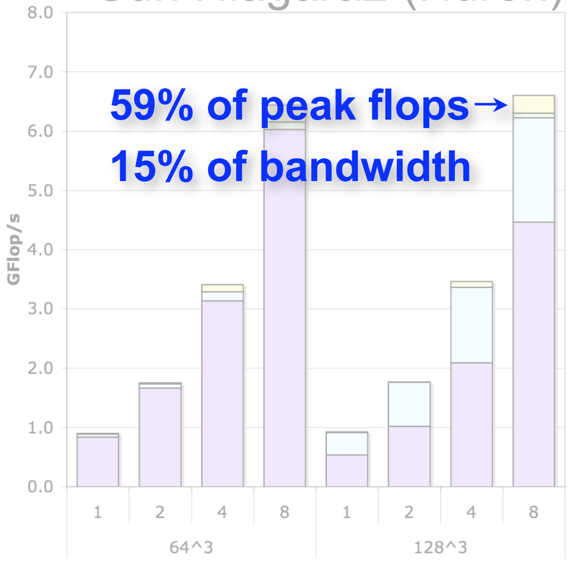


32% of peak flops

40% of bandwidth

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

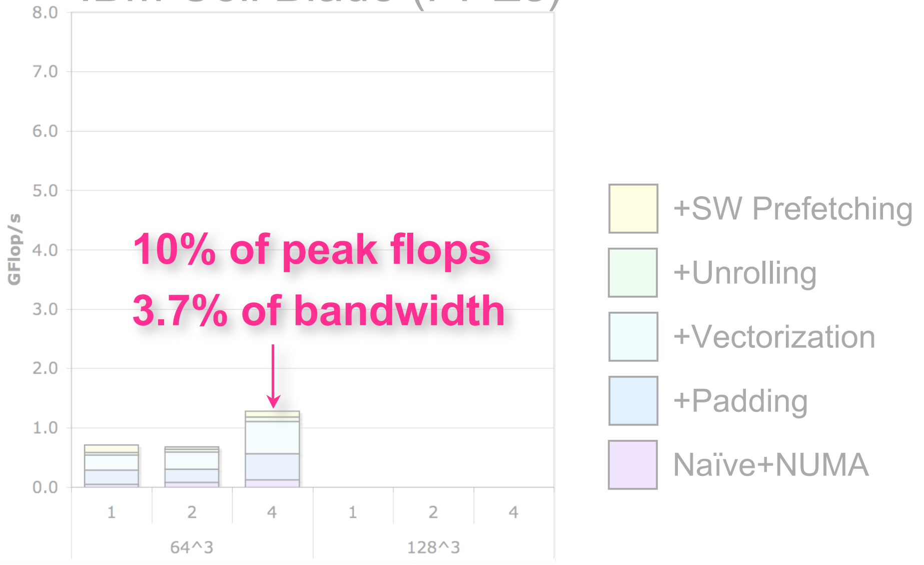
Sun Niagara2 (Huron)



59% of peak flops

15% of bandwidth

IBM Cell Blade (PPEs)



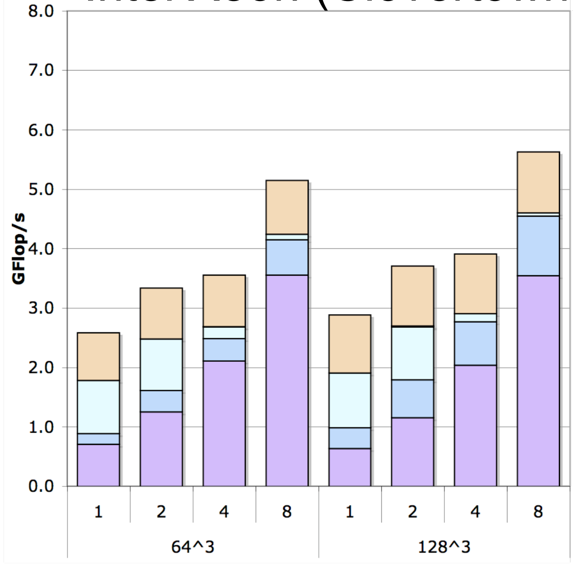
10% of peak flops

3.7% of bandwidth

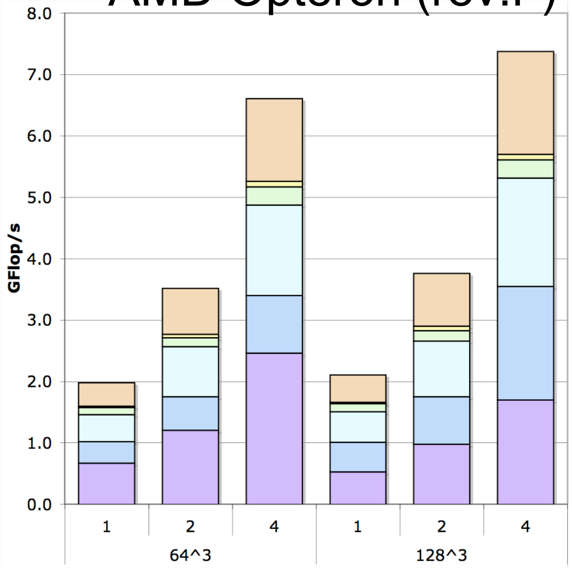
Auto-tuned Performance

(+SIMDization, including non-temporal stores)

Intel Xeon (Clovertown)

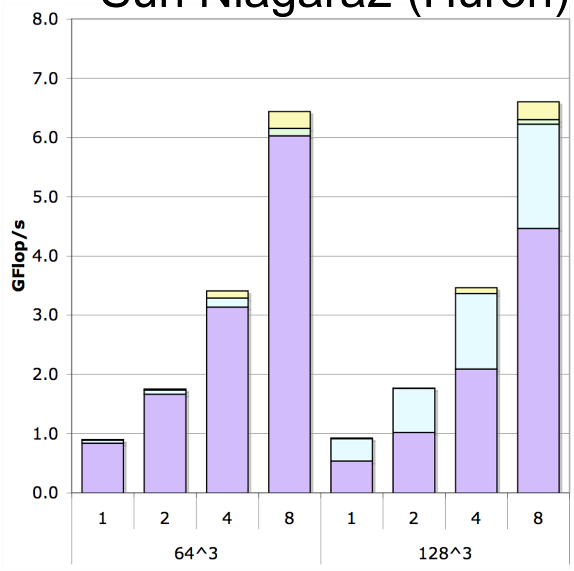


AMD Opteron (rev.F)

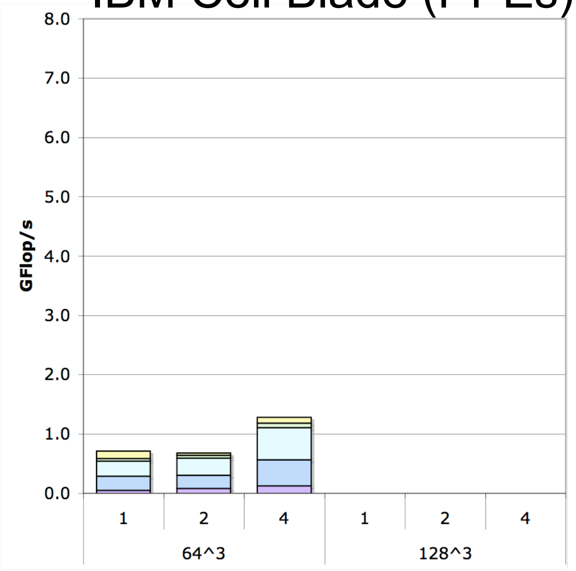


- Compilers(gcc & icc) failed at exploiting SIMD.
- Expanded the code generator to use SIMD intrinsics.
- Explicit unrolling/reordering was extremely valuable here.
- Exploited ***movntpd*** to minimize memory traffic (only hope if memory bound)
- Significant benefit for significant work

Sun Niagara2 (Huron)



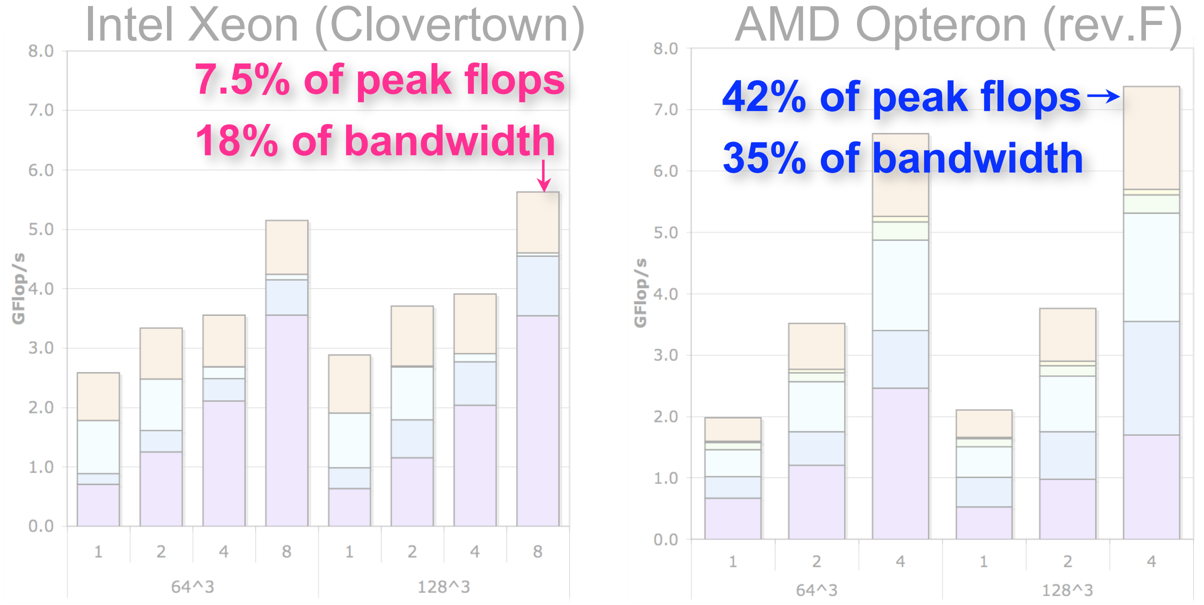
IBM Cell Blade (PPEs)



- 🟧 +SIMDization
- 🟨 +SW Prefetching
- 🟩 +Unrolling
- ▢ +Vectorization
- 🟦 +Padding
- 🟪 Naïve+NUMA

Auto-tuned Performance

(+SIMDization, including non-temporal stores)



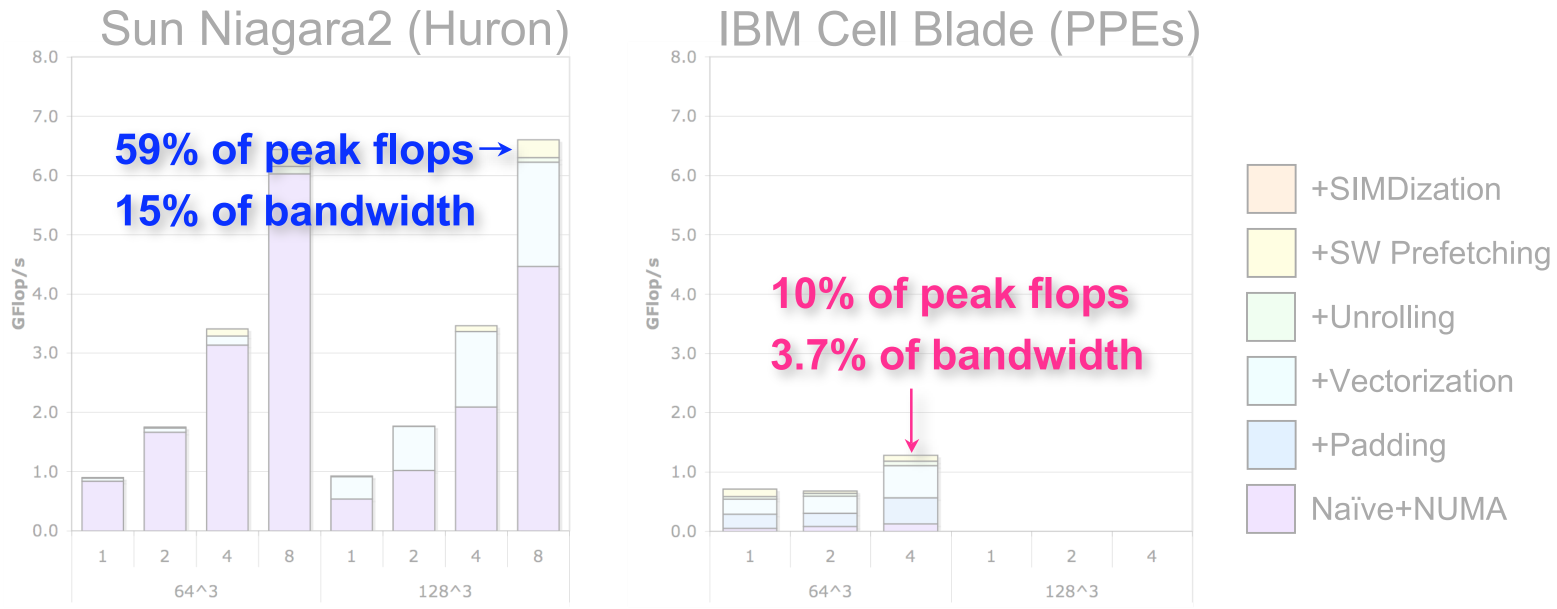
7.5% of peak flops

18% of bandwidth

42% of peak flops→

35% of bandwidth

- 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



59% of peak flops→

15% of bandwidth

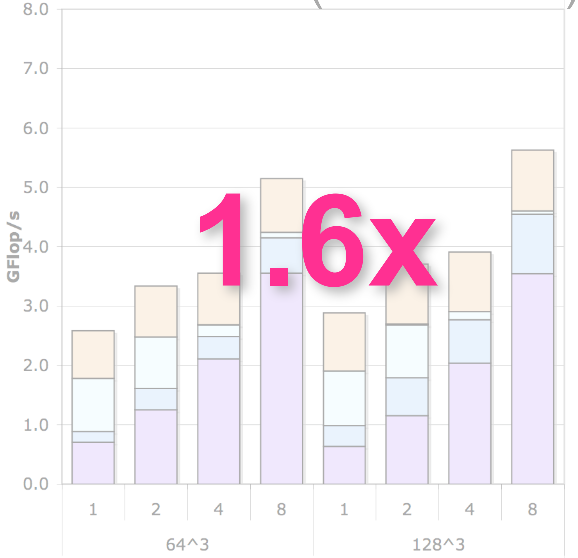
10% of peak flops

3.7% of bandwidth

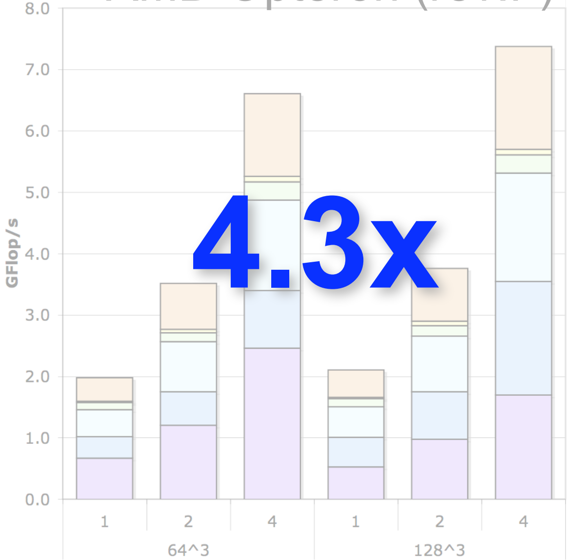
Auto-tuned Performance

(+SIMDization, including non-temporal stores)

Intel Xeon (Clovertown)

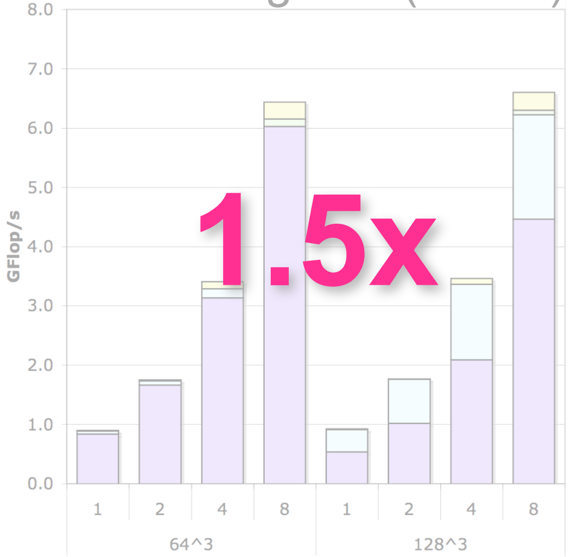


AMD Opteron (rev.F)

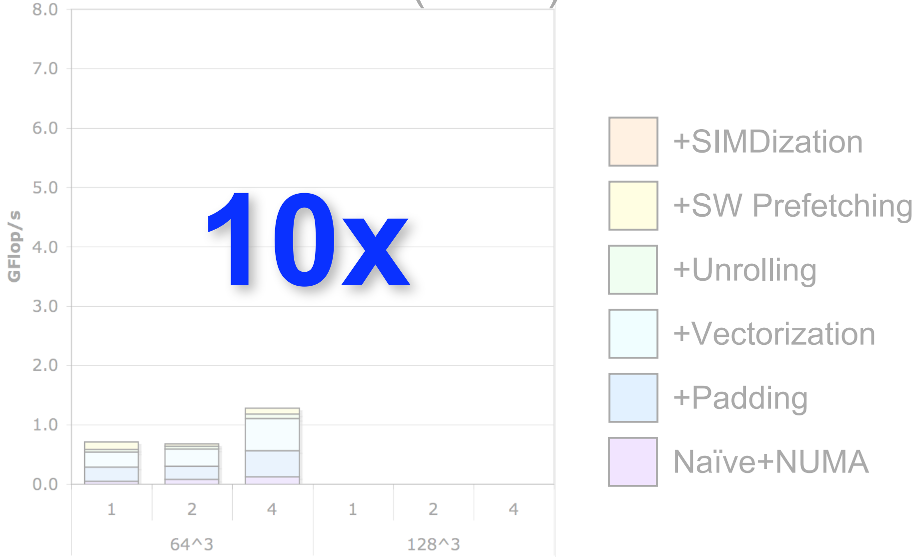


- Compilers(gcc & icc) failed at exploiting SIMD.
- Expanded the code generator to use SIMD intrinsics.
- Explicit unrolling/reordering was extremely valuable here.
- Exploited ***movntpd*** to minimize memory traffic (only hope if memory bound)
- Significant benefit for significant work

Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



Performance and Analysis of Cell Implementation

Office of Science
U.S. DEPARTMENT OF ENERGY

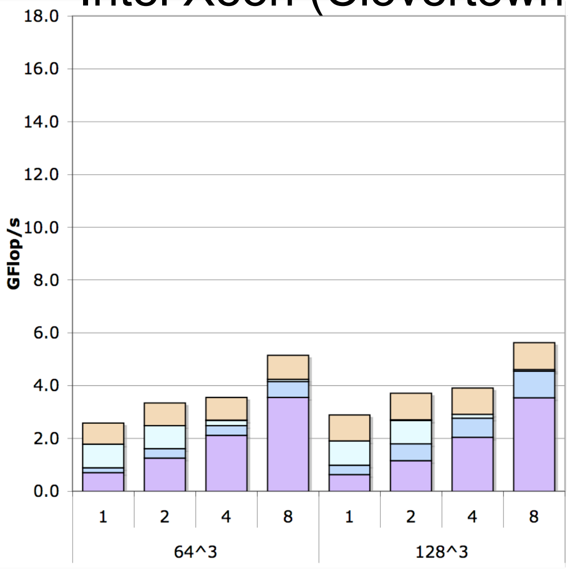
Cell Implementation

- Double precision implementation
  - DP will severely hamper performance
- Vectorized, double buffered, but not auto-tuned
  - No NUMA optimizations
  - No Unrolling
  - VL is fixed
  - Straight to SIMD intrinsics
  - Prefetching replaced by DMA list commands
- Only *collision()* was implemented.

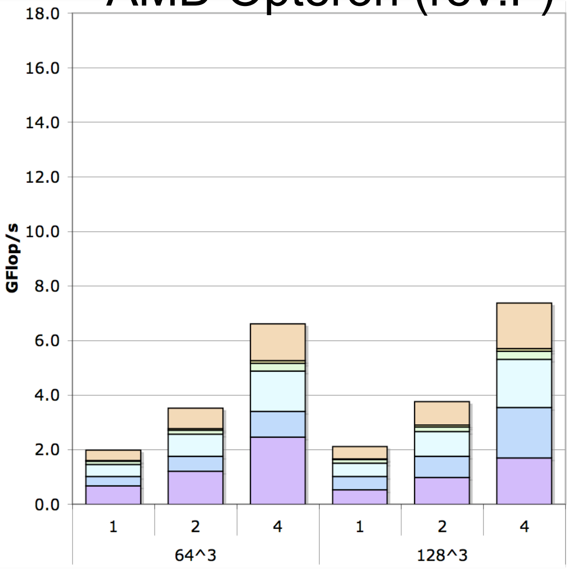
Auto-tuned Performance

(Local Store Implementation)

Intel Xeon (Clovertown)

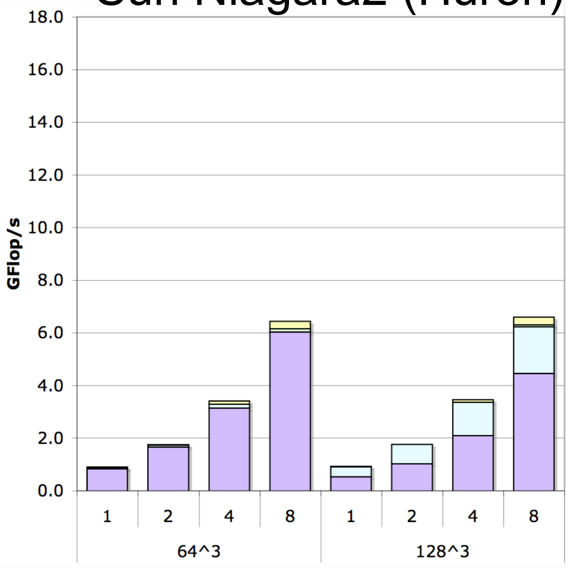


AMD Opteron (rev.F)

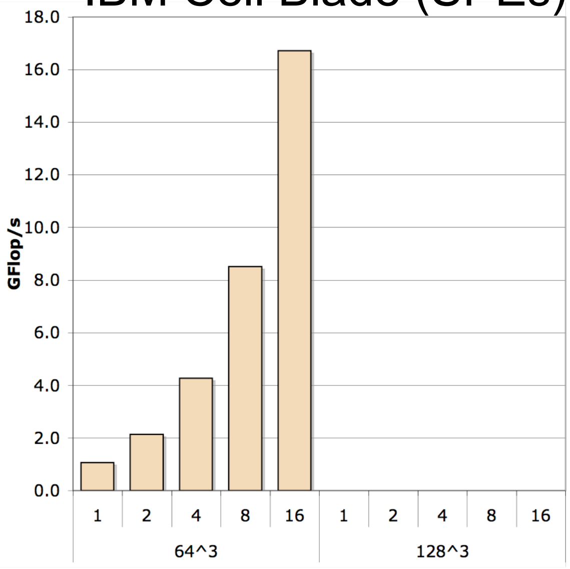


- First attempt at cell implementation.
- VL, unrolling, reordering fixed
- No NUMA
- Exploits DMA and double buffering to load vectors
- Straight to SIMD intrinsics.
- Despite the relative performance, Cell's DP implementation severely impairs performance

Sun Niagara2 (Huron)



IBM Cell Blade (SPEs)*

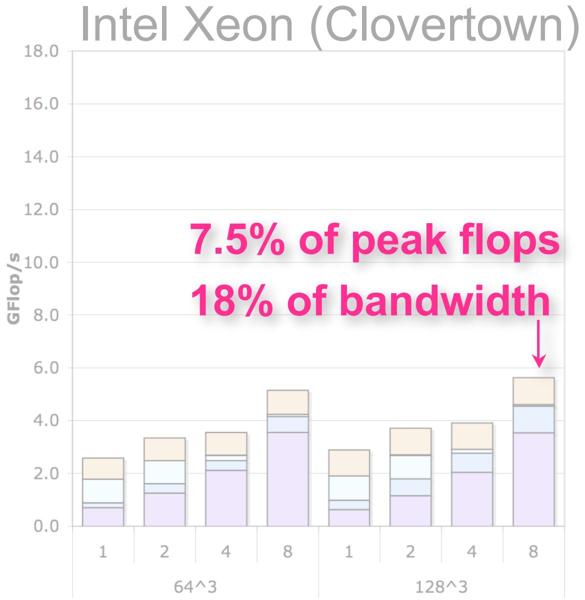


- ▨ +SIMDization
- ▨ +SW Prefetching
- ▨ +Unrolling
- ▨ +Vectorization
- ▨ +Padding
- ▨ Naïve+NUMA

*collision() only

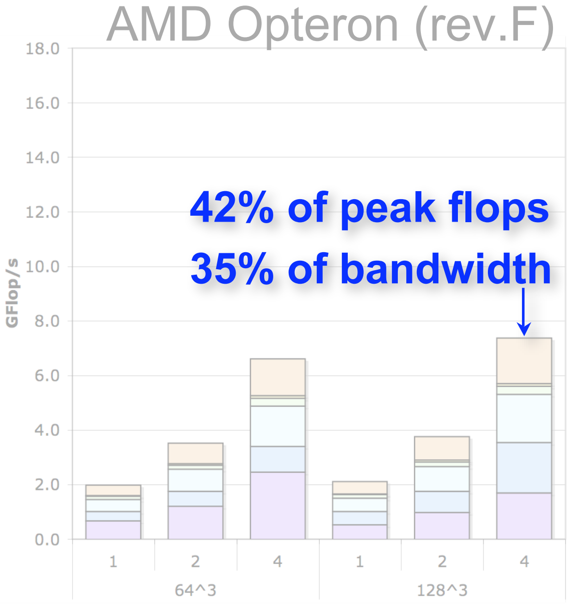
Auto-tuned Performance

(Local Store Implementation)


Intel Xeon (Clovertown)

7.5% of peak flops

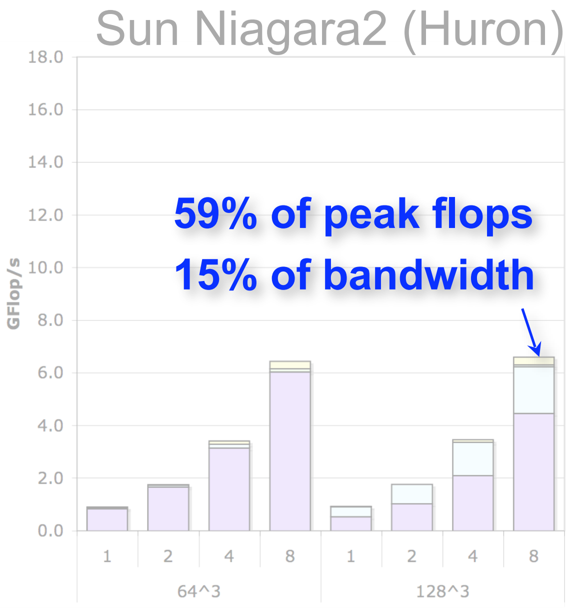
18% of bandwidth


AMD Opteron (rev.F)

42% of peak flops

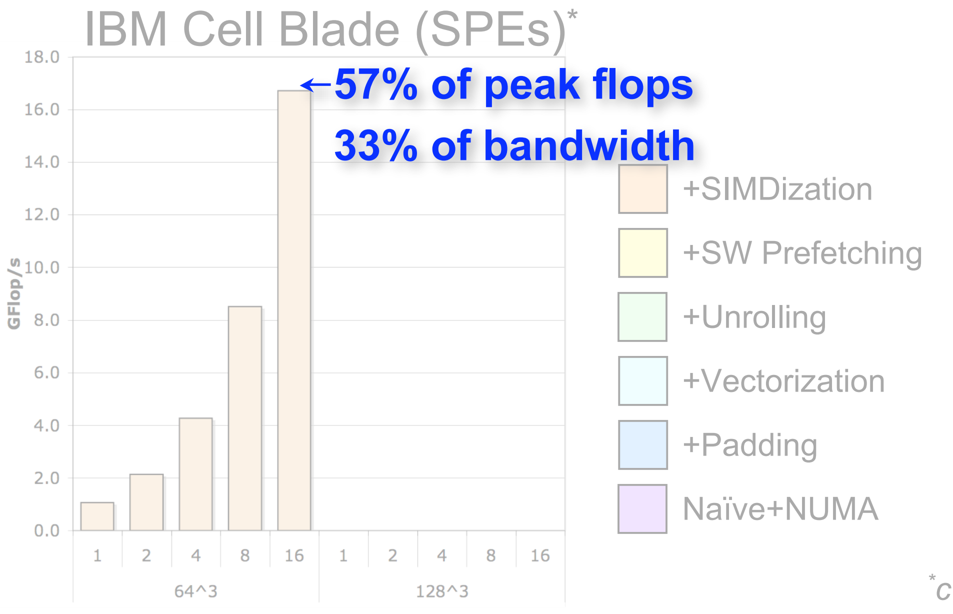
35% of bandwidth

- 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


Sun Niagara2 (Huron)

59% of peak flops

15% of bandwidth


IBM Cell Blade (SPEs)*

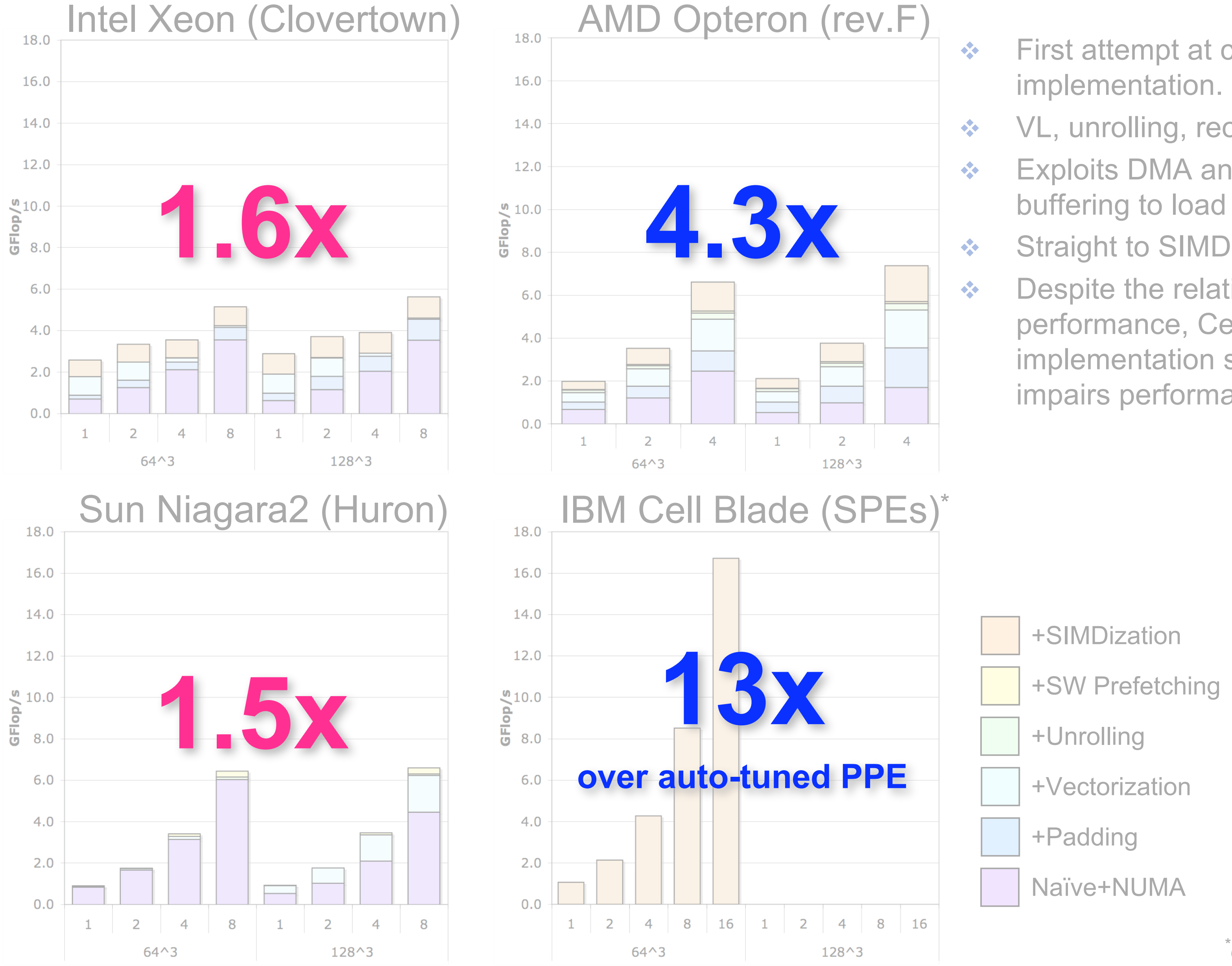
57% of peak flops

33% of bandwidth

*collision() only

Speedup from Heterogeneity

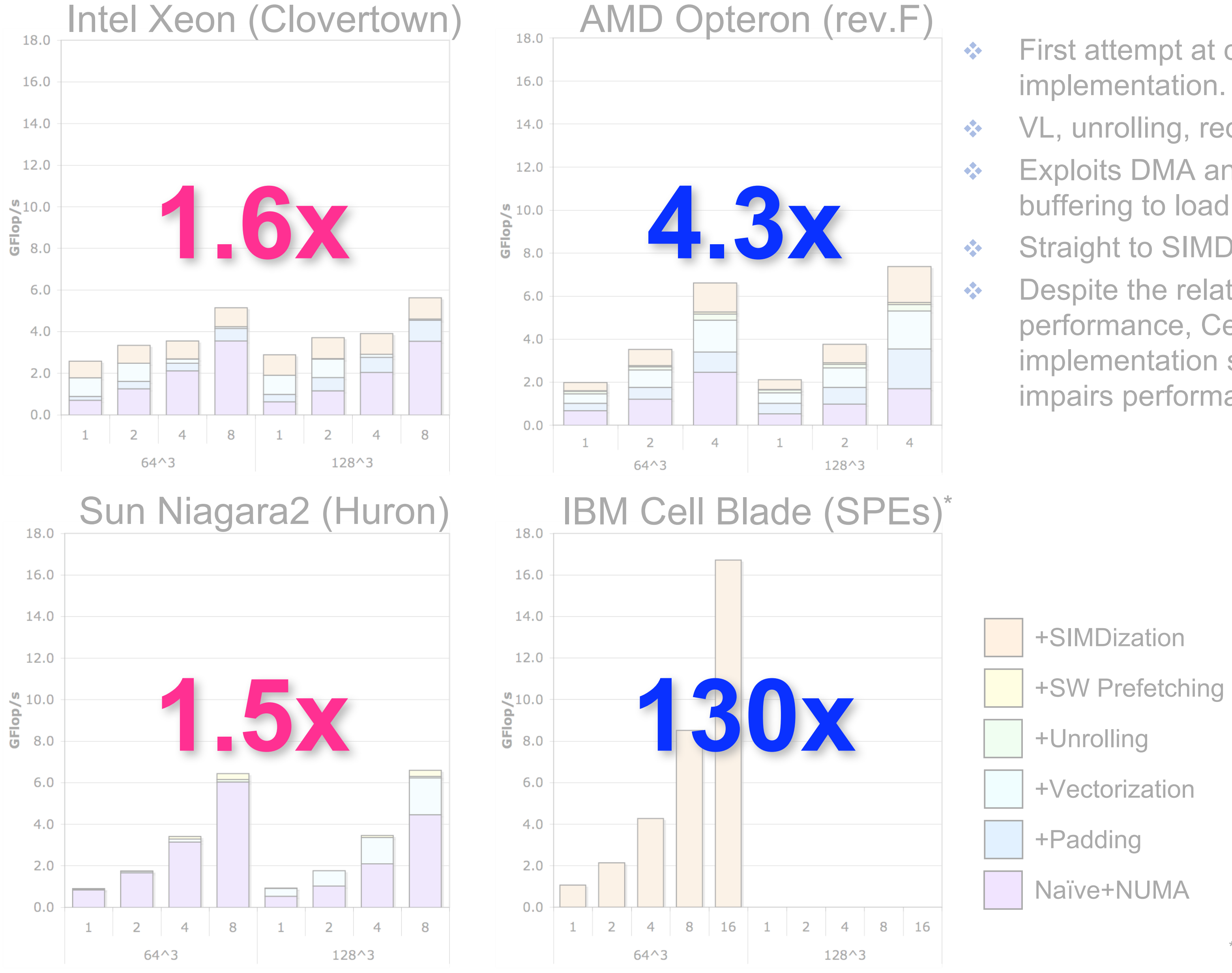
- First attempt at cell implementation.
- VL, unrolling, reordering fixed
- Exploits DMA and double buffering to load vectors
- Straight to SIMD intrinsics.
- Despite the relative performance, Cell's DP implementation severely impairs performance



*collision() only

Speedup over naive

- 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



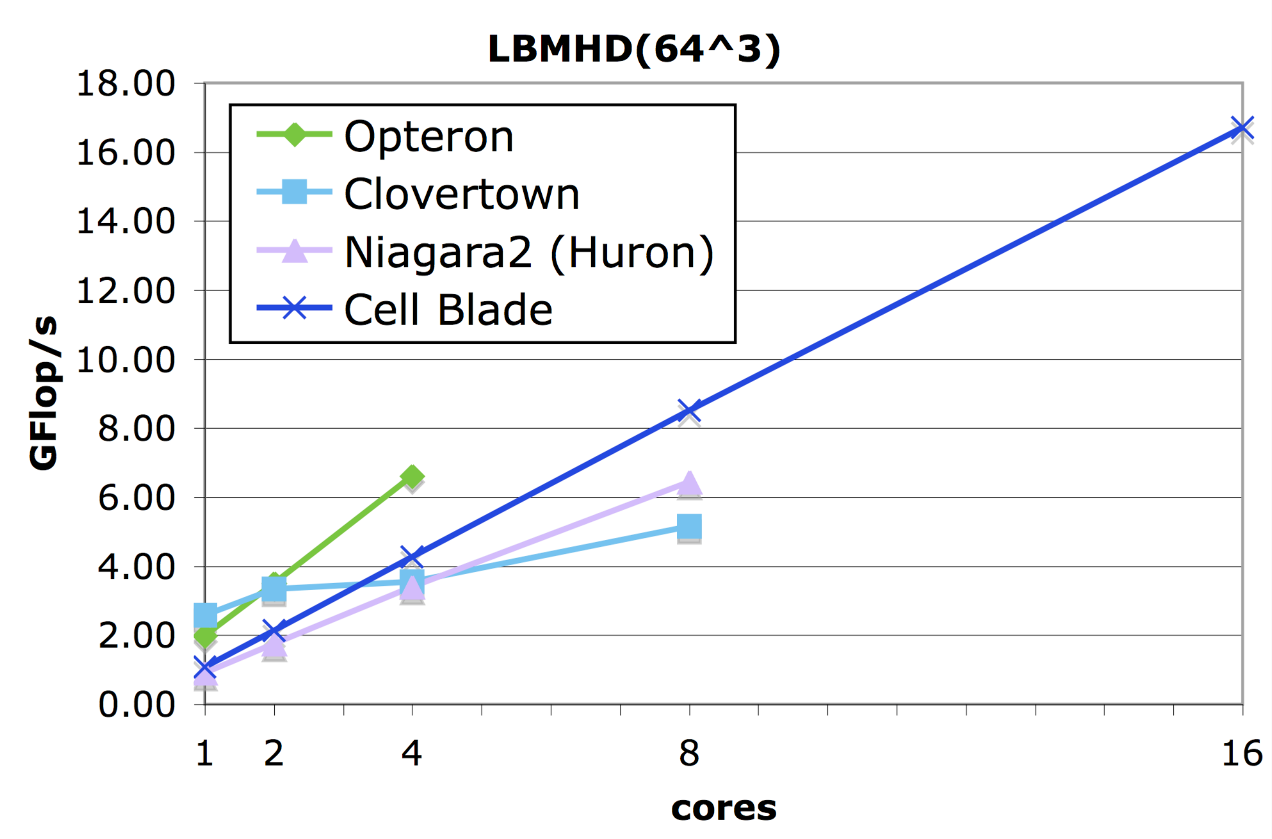
[*] collision() only

Summary

Aggregate Performance

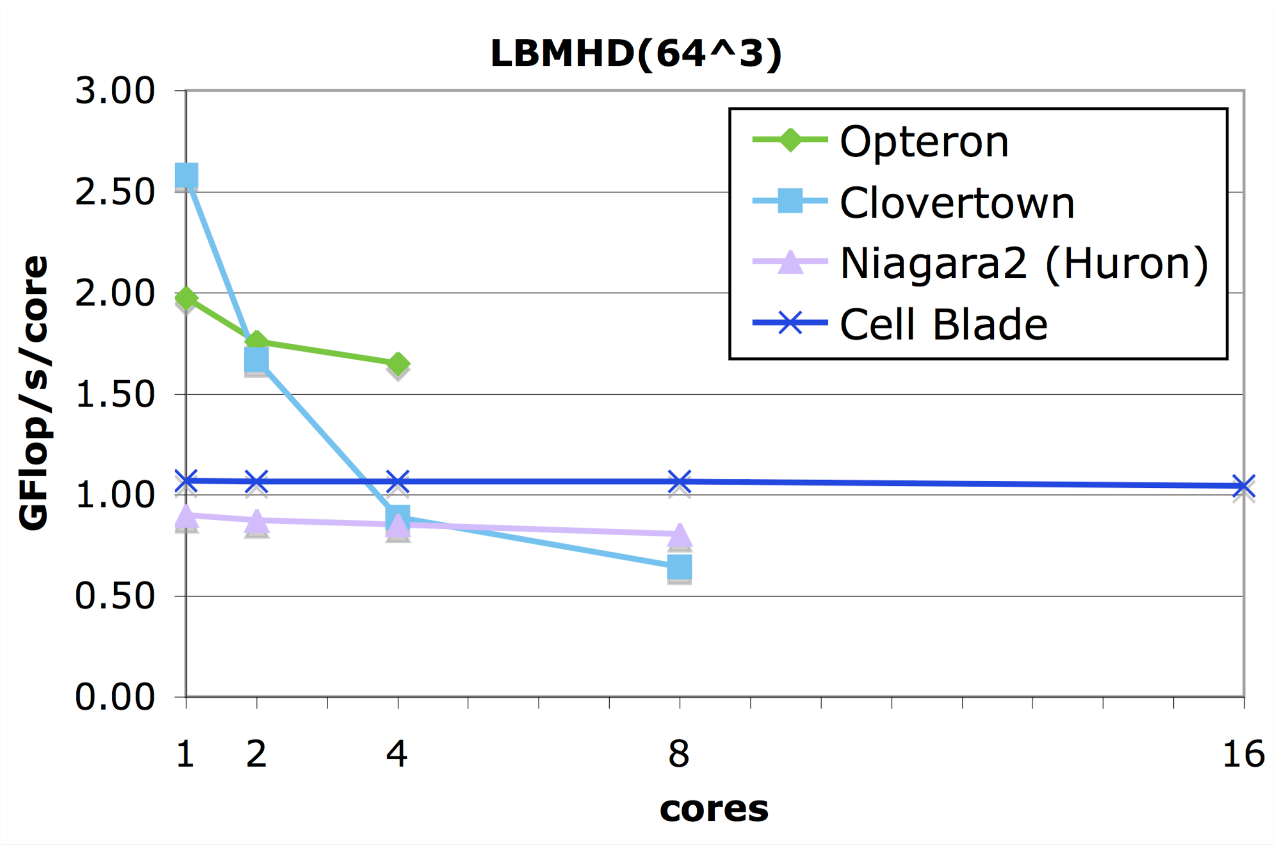
(Fully optimized)

- Cell SPEs deliver the best full system performance
  - Although, Niagara2 delivers near comparable per socket performance
- Dual core Opteron delivers far better performance (bandwidth) than Clovertown
- Clovertown has far too little effective FSB bandwidth



Parallel Efficiency
(average performance per thread, Fully optimized)

- Aggregate Mflop/s / #cores
- Niagara2 & Cell showed very good multicore scaling
- Clovertown showed very poor multicore scaling (FSB limited)



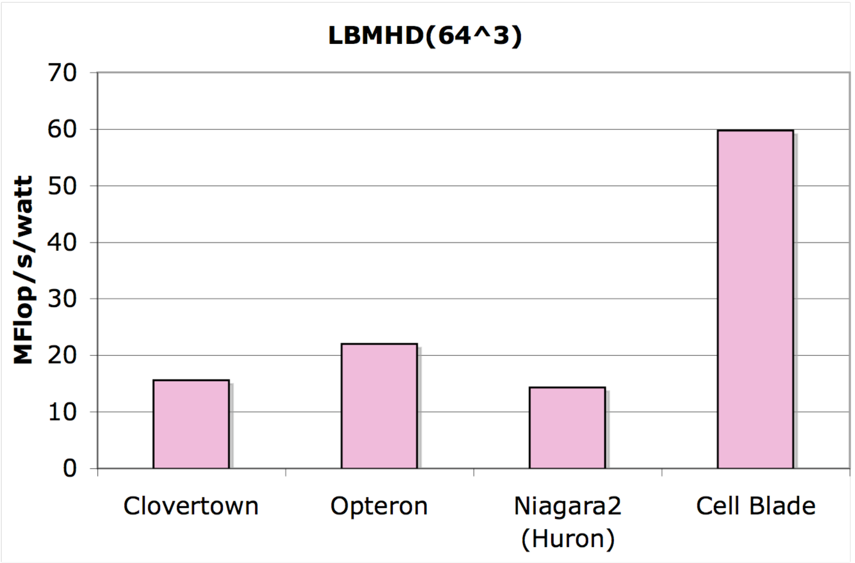
Power Efficiency

(Fully Optimized)

- Used a digital power meter to measure sustained power
- 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 (just vectorization for large problems).
- Clovertown, Opteron, and Cell all required SIMD (hampers productivity) for best performance.
- Cache based machines required search for some optimizations, while Cell relied solely on heuristics (less time to tune)

Summary

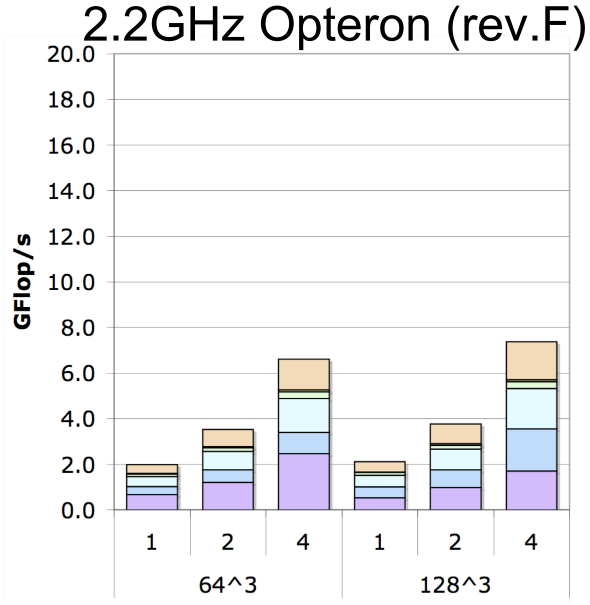
- Niagara2 delivered both very good performance and productivity
- Cell delivered very good performance and efficiency (processor and power)
- On the memory bound Clovertown parallelism wins out over optimization and auto-tuning

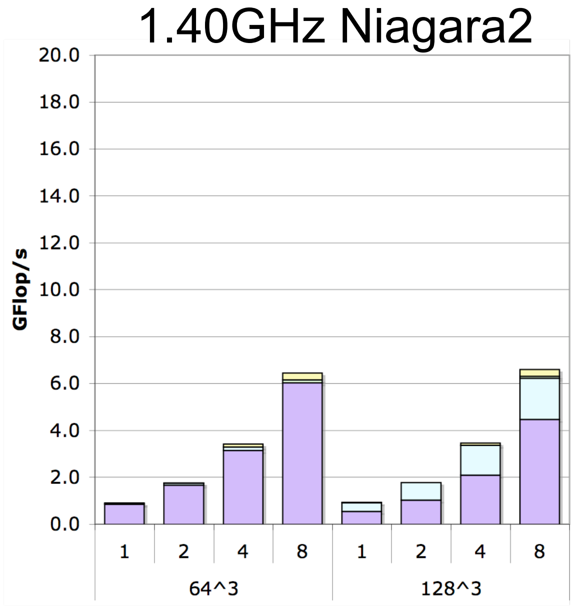
- **Our multicore auto-tuned LBMHD implementation significantly outperformed the already optimized serial implementation**

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

Multi-core arms race

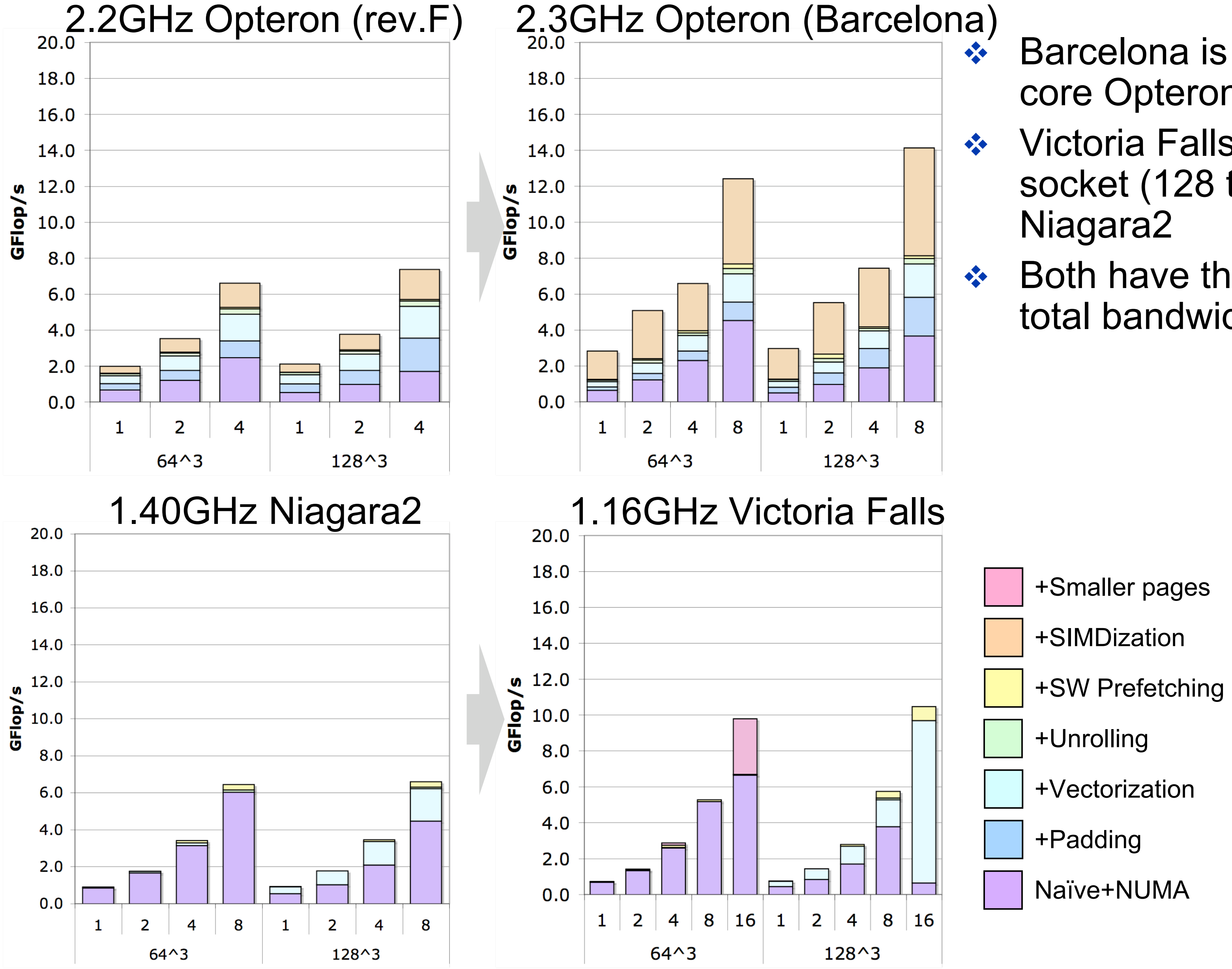
New Multicores


2.2GHz Opteron (rev.F)

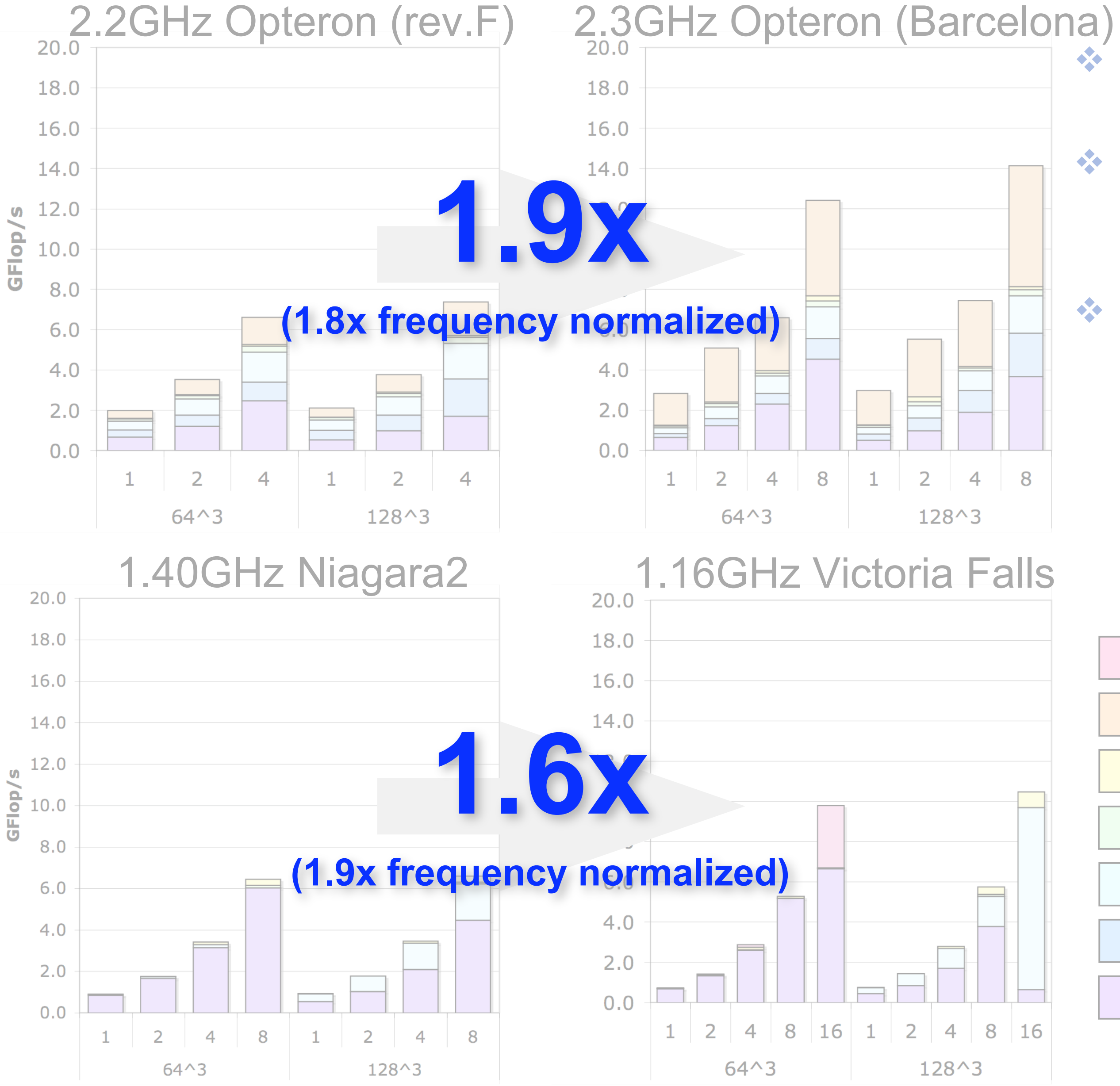

1.40GHz Niagara2

New Multicores

- Barcelona is a quad core Opteron
- Victoria Falls is a dual socket (128 threads) Niagara2
- Both have the same total bandwidth



Speedup from multicore/socket



1.9x

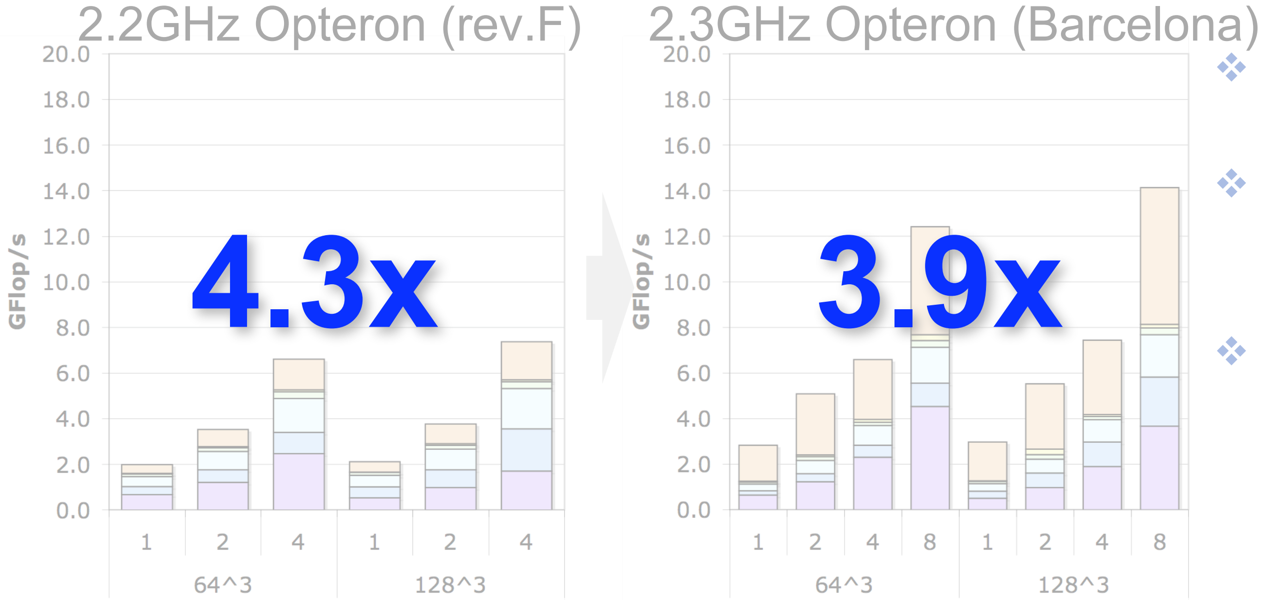
(1.8x frequency normalized)

1.6x

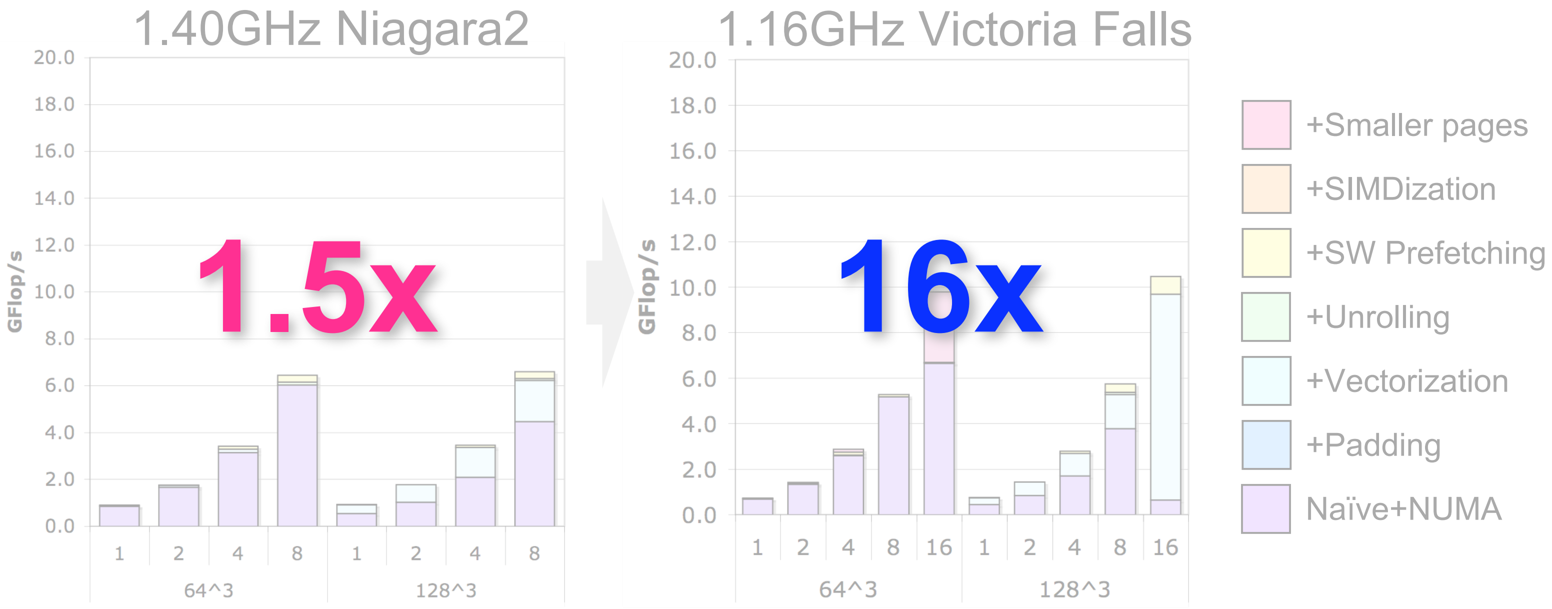
(1.9x frequency normalized)

- Barcelona is a quad core Opteron
- Victoria Falls is a dual socket (128 threads) Niagara2
- Both have the same total bandwidth

Speedup from Auto-tuning



- Barcelona is a quad core Opteron
- Victoria Falls is a dual socket (128 threads) Niagara2
- Both have the same total bandwidth



Questions?

Acknowledgements

- UC Berkeley
  - RADLab Cluster (Opterons)
  - PSI cluster(Clovertowns)
- Sun Microsystems
  - Niagara2 donations
- Forschungszentrum Jülich
  - Cell blade cluster access
- George Vahala, et. al
  - Original version of LBMHD
- ASCR Office in the DOE office of Science
  - contract DE-AC02-05CH11231