BIPS

COMPUTATIONAL RESEARCH DIVISION

Tuning Sparse Matrix Vector Multiplication for multi-core SMPs

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

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

samw@cs.berkeley.edu

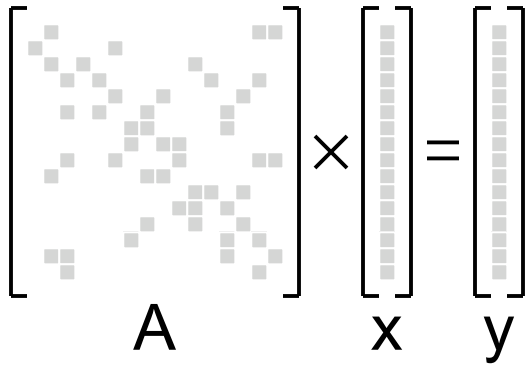
Office of Science
U.S. DEPARTMENT OF ENERGY

Overview

- Multicore is the de facto performance solution for the next decade
- Examined Sparse Matrix Vector Multiplication (SpMV) kernel
  - Important HPC kernel
  - Memory intensive
  - Challenging for multicore
- Present two autotuned threaded implementations:
  - Pthread, cache-based implementation
  - Cell local store-based implementation
- Benchmarked performance across 4 diverse multicore architectures
  - Intel Xeon (Clovertown)
  - AMD Opteron
  - Sun Niagara2
  - IBM Cell Broadband Engine
- Compare with leading MPI implementation(PETSc) with an autotuned serial kernel (OSKI)

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)**



Test Suite

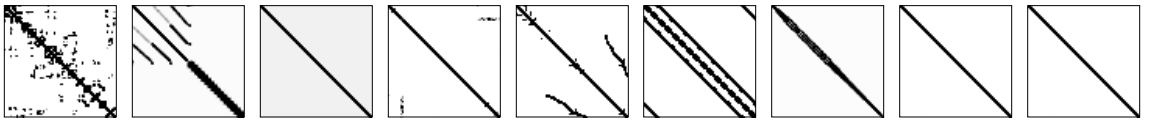
- Dataset (Matrices)
- Multicore SMPs

Matrices Used

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

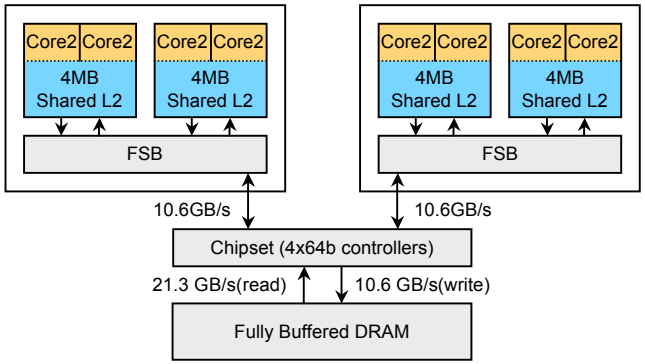
FEM / Accelerator, Circuit, webbase

Extreme Aspect Ratio (linear programming)

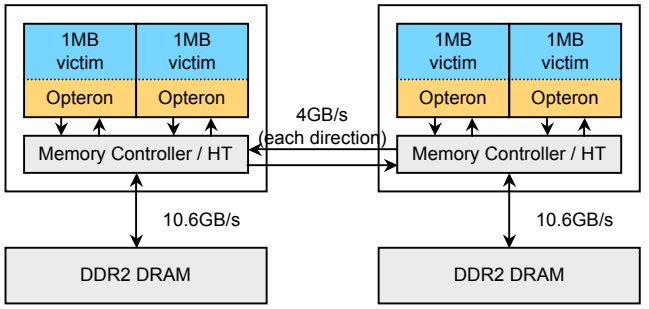
LP

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

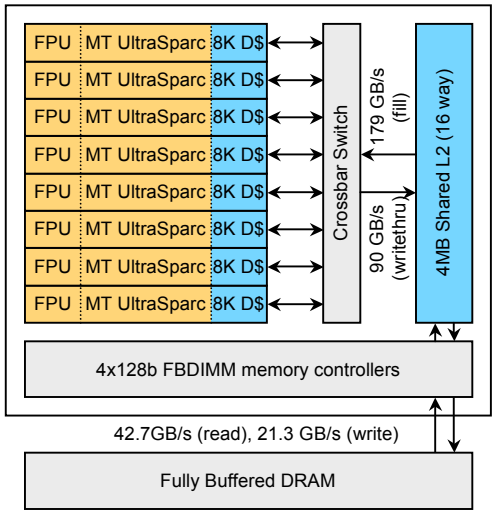
Multicore SMP Systems



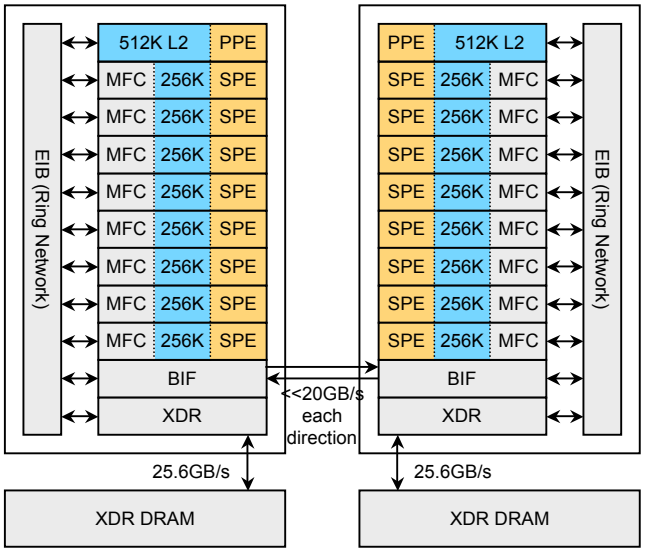
Intel Clovertown



AMD Opteron



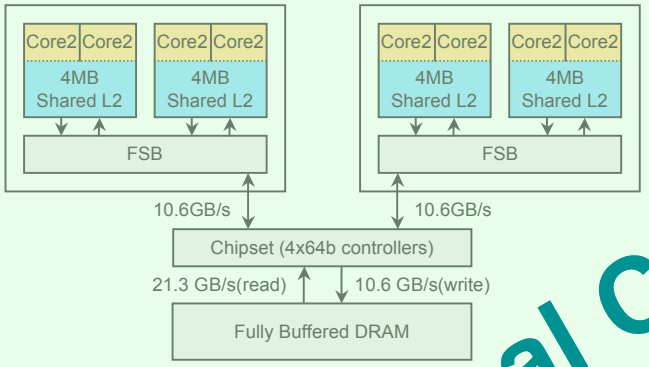
Sun Niagara2



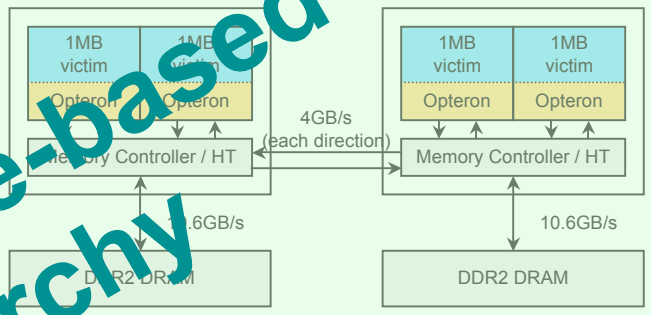
IBM Cell Blade

Office of Science
U.S. DEPARTMENT OF ENERGY

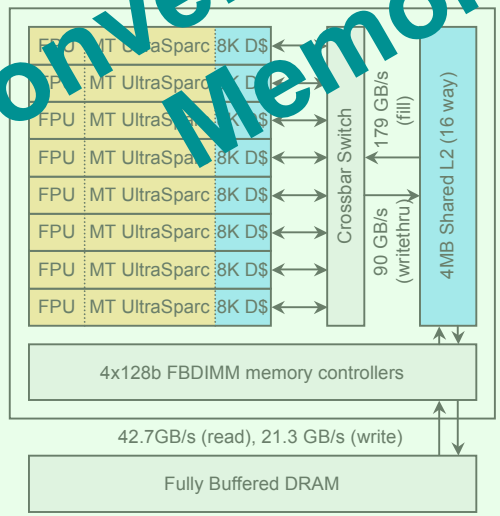
Multicore SMP Systems
(memory hierarchy)



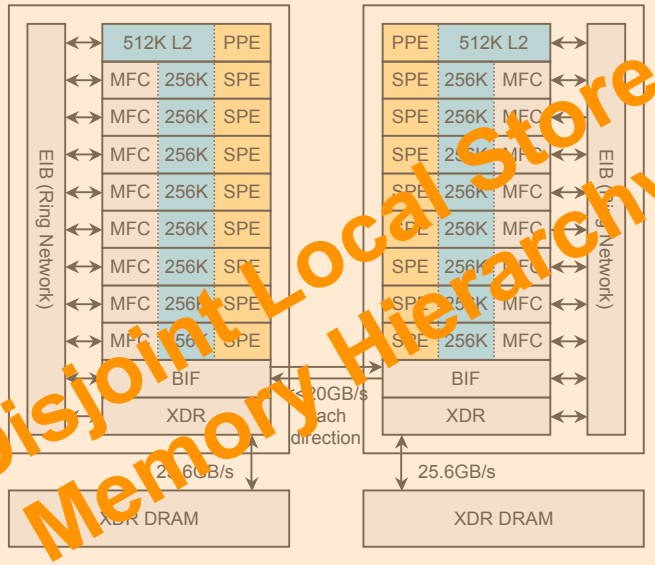
Intel Clovertown



AMD Opteron



Sun Niagara2

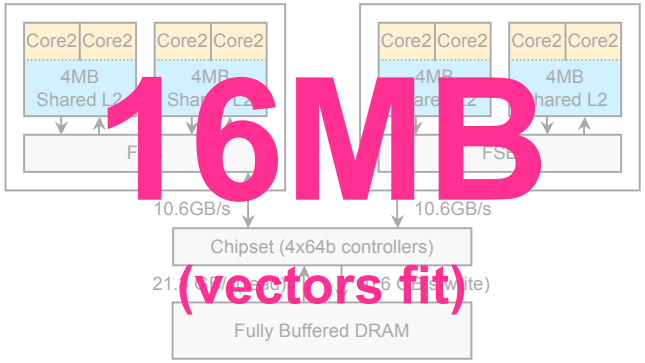


IBM Cell Blade

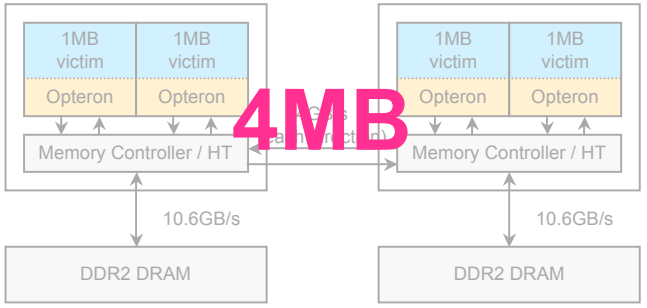
Conventional Cache-based Memory Hierarchy

Disjoint Local Store Memory Hierarchy

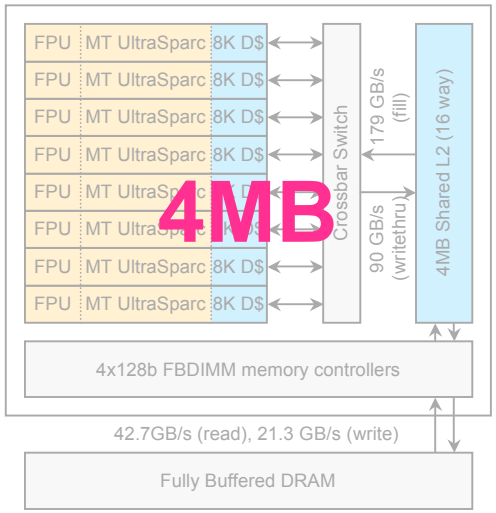
Multicore SMP Systems
(cache)



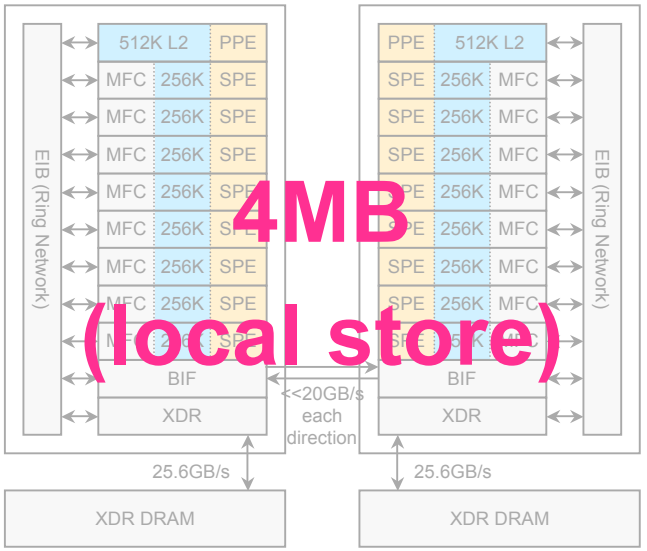
Intel Clovertown



AMD Opteron

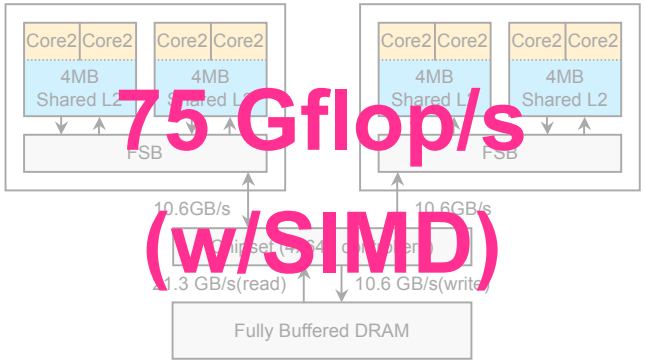


Sun Niagara2



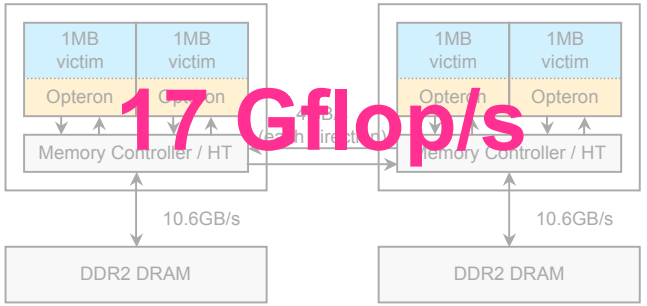
IBM Cell Blade

Multicore SMP Systems
(peak flops)



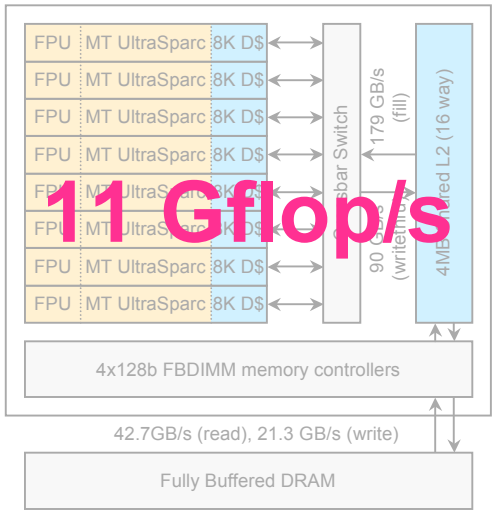
75 Gflop/s
(w/SIMD)

Intel Clovertown



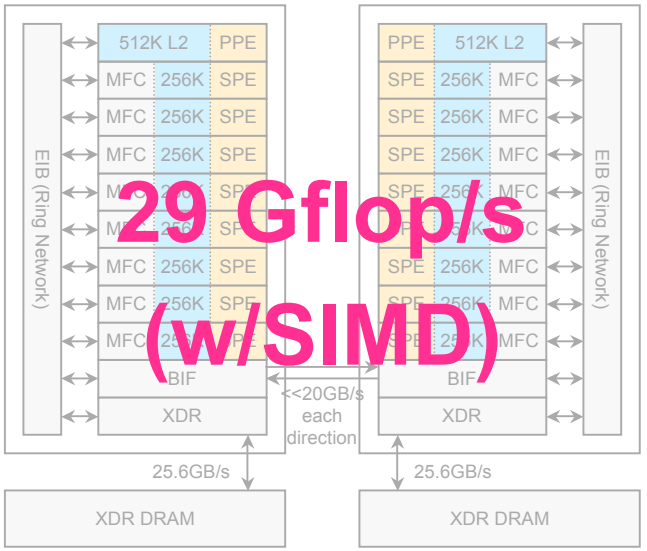
17 Gflop/s

AMD Opteron



11 Gflop/s

Sun Niagara2

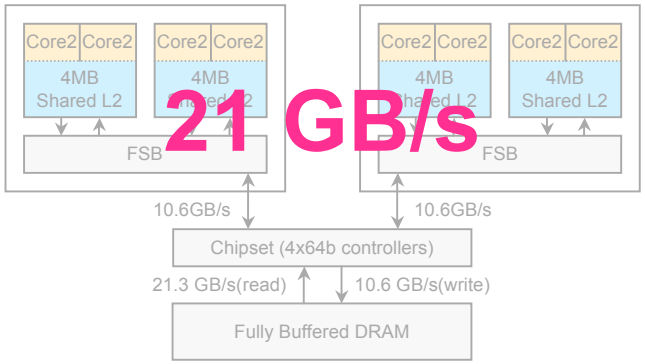


29 Gflop/s
(w/SIMD)

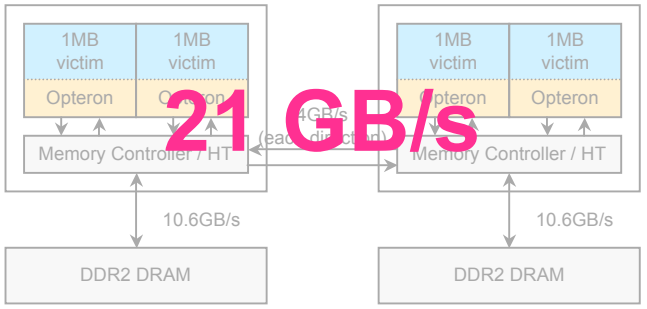
IBM Cell Blade

Office of Science
U.S. DEPARTMENT OF ENERGY

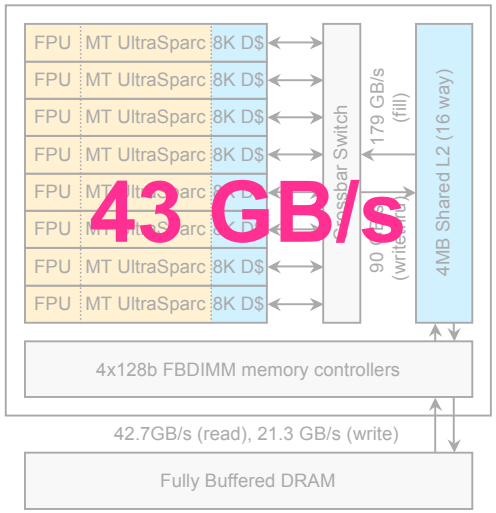
Multicore SMP Systems
(peak read bandwidth)



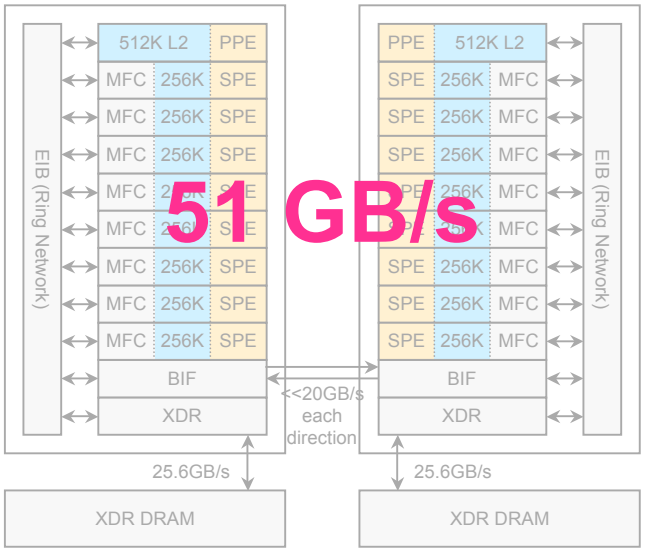
Intel Clovertown



AMD Opteron



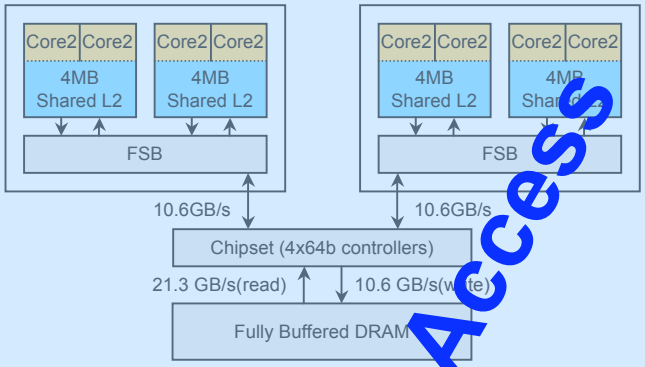
Sun Niagara2



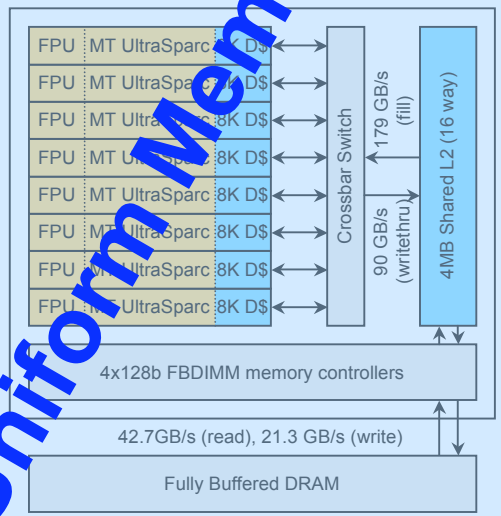
IBM Cell Blade

Office of Science
U.S. DEPARTMENT OF ENERGY

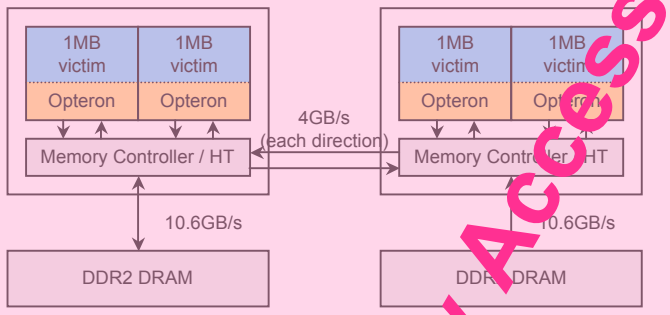
Multicore SMP Systems
(NUMA)



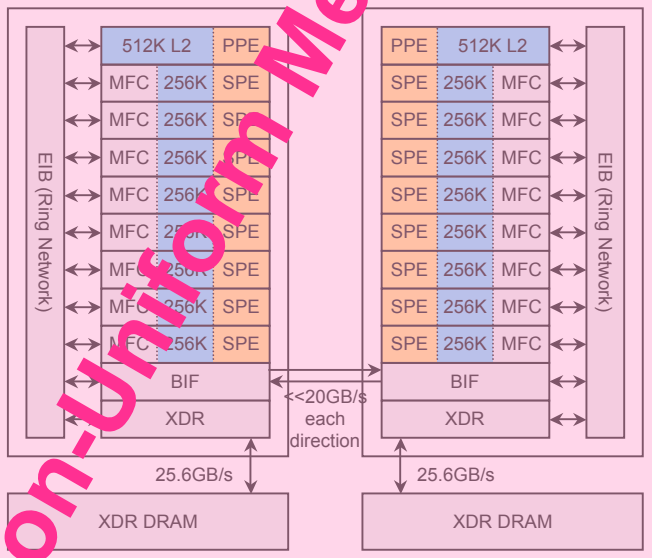
Intel Clovertown



Sun Niagara2



AMD Opteron



IBM Cell Blade

Uniform Memory Access

Non-Uniform Memory Access

BIPS

Berkeley Lab

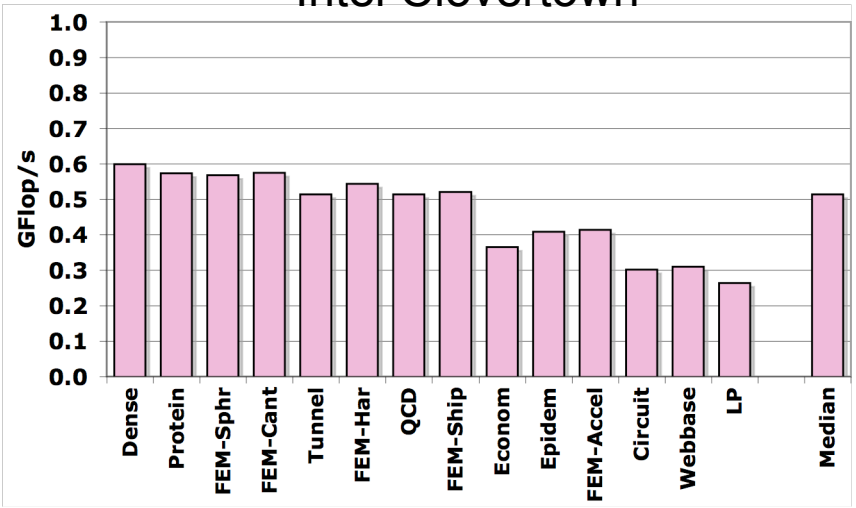
Office of Science
U.S. Department of Energy

Naïve Implementation

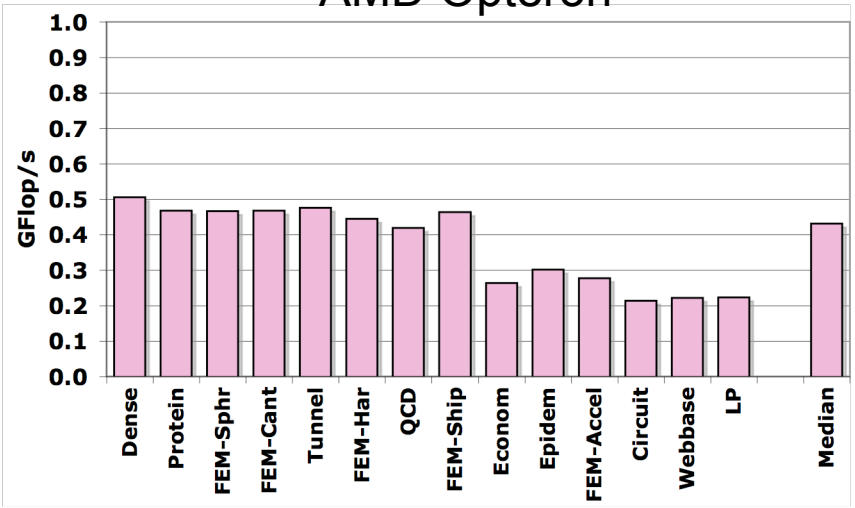
- For cache-based machines
- Included a median performance number

vanilla C Performance

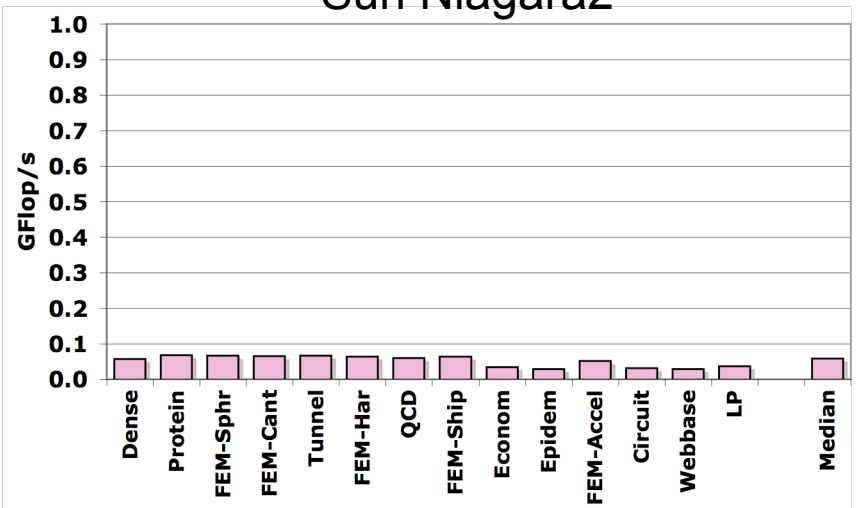
Intel Clovertown



AMD Opteron



Sun Niagara2



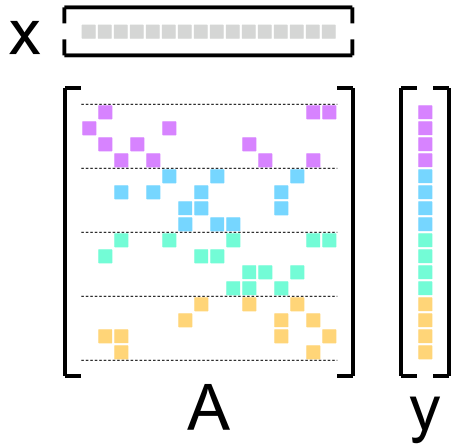
- Vanilla C implementation
- Matrix stored in CSR (compressed sparse row)
- Explored compiler options - only the best is presented here

Pthread Implementation

- Optimized for multicore/threading
- Variety of shared memory programming models are acceptable(not just Pthreads)
- More colors = more optimizations = more work

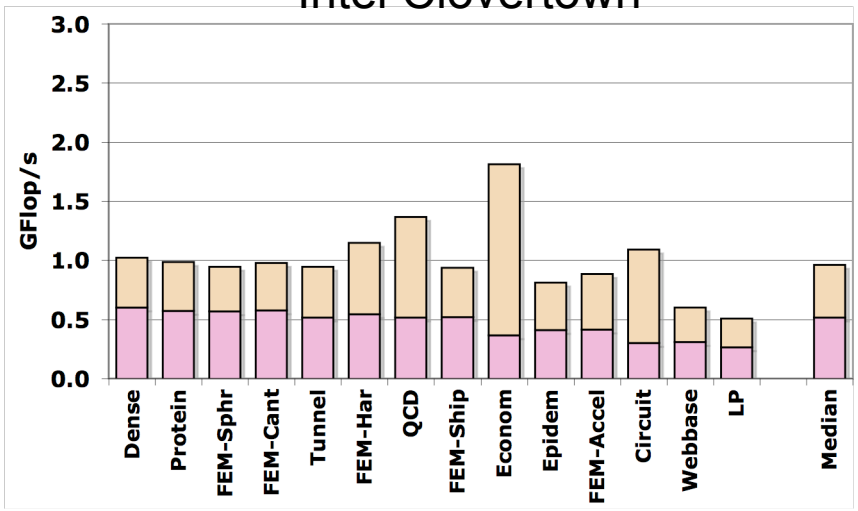
Parallelization

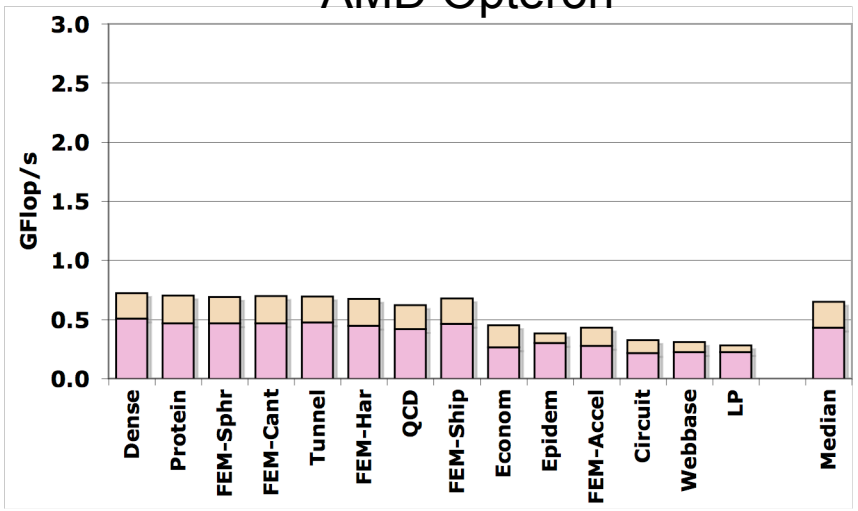
- Matrix partitioned by rows and balanced by the number of nonzeros
- SPMD like approach
- A barrier() is called before and after the SpMV kernel
- Each sub matrix stored separately in CSR
- Load balancing can be challenging

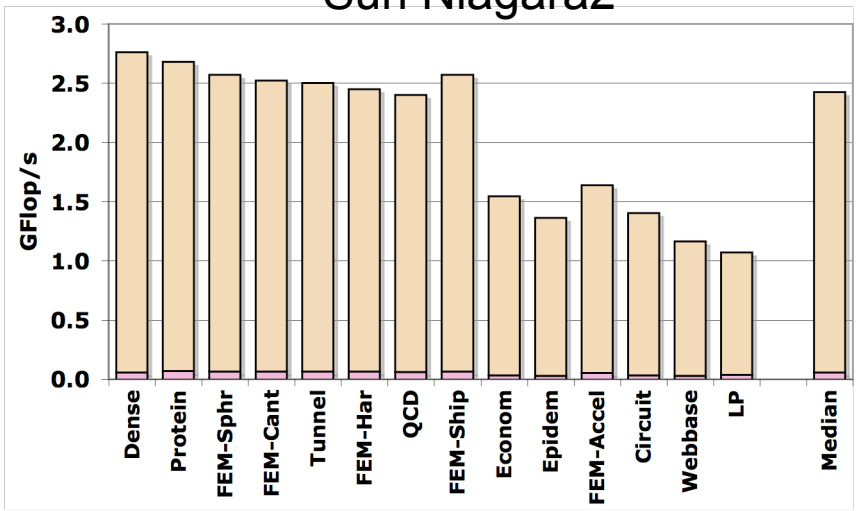


- # of threads explored in powers of 2 (in paper)

Naïve Parallel Performance





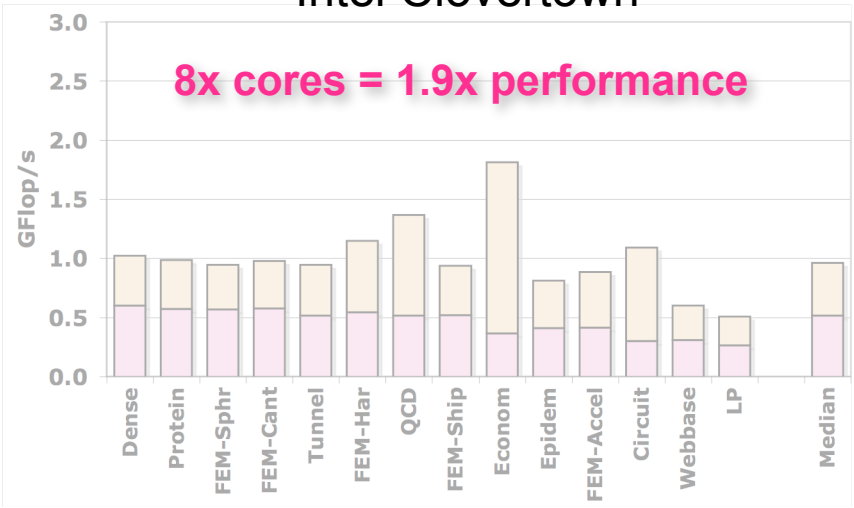


Naïve Pthreads

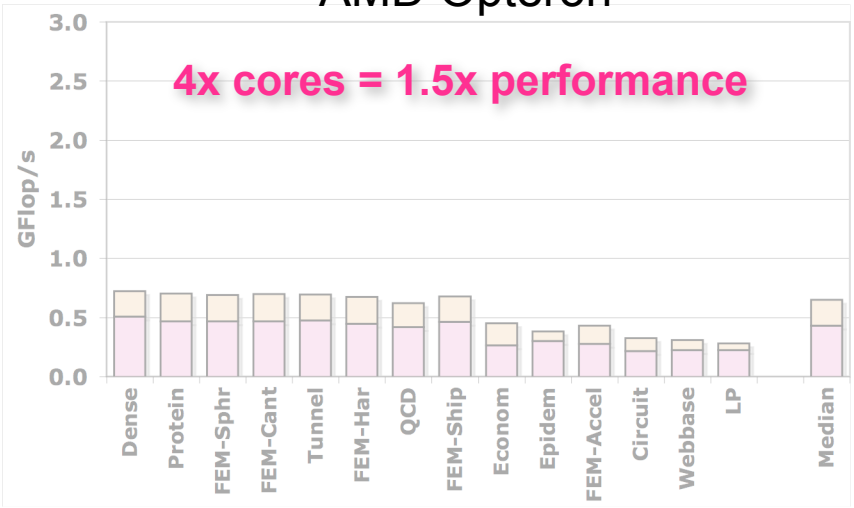
Naïve Single Thread

Naïve Parallel Performance

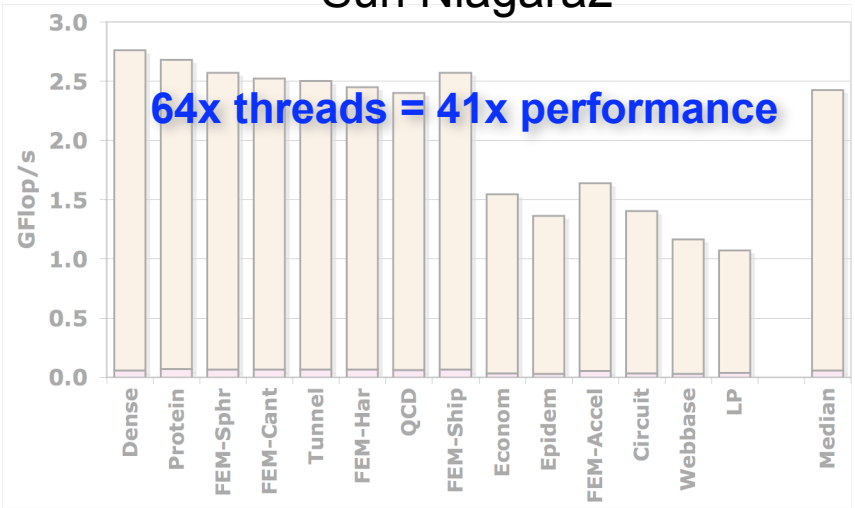
Intel Clovertown



AMD Opteron



Sun Niagara2



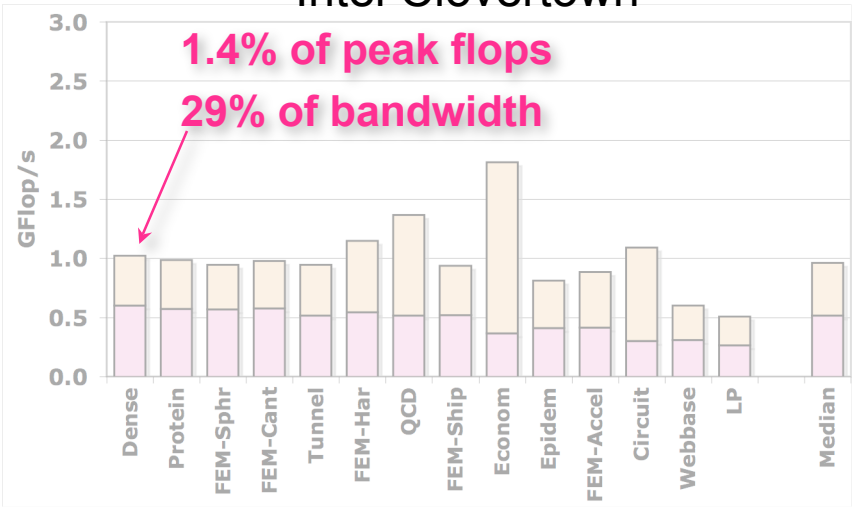
Naïve Pthreads

Naïve Single Thread

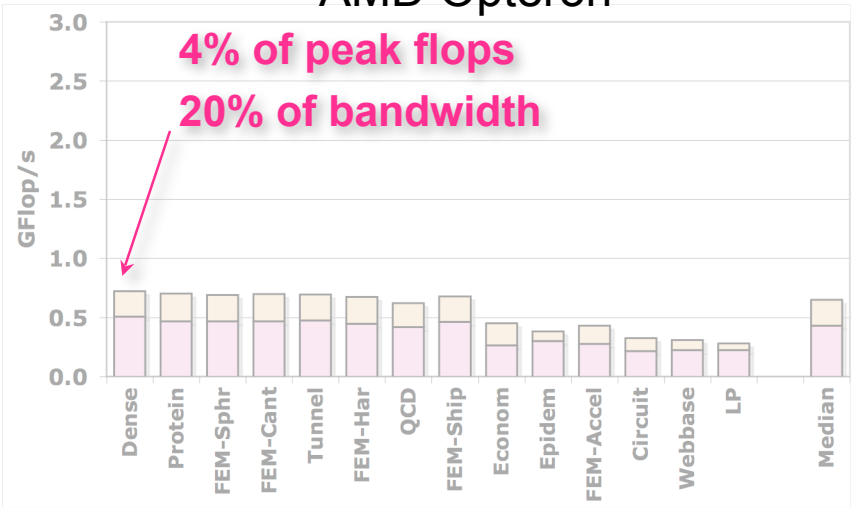
Office of Science
U.S. DEPARTMENT OF ENERGY

Naïve Parallel Performance

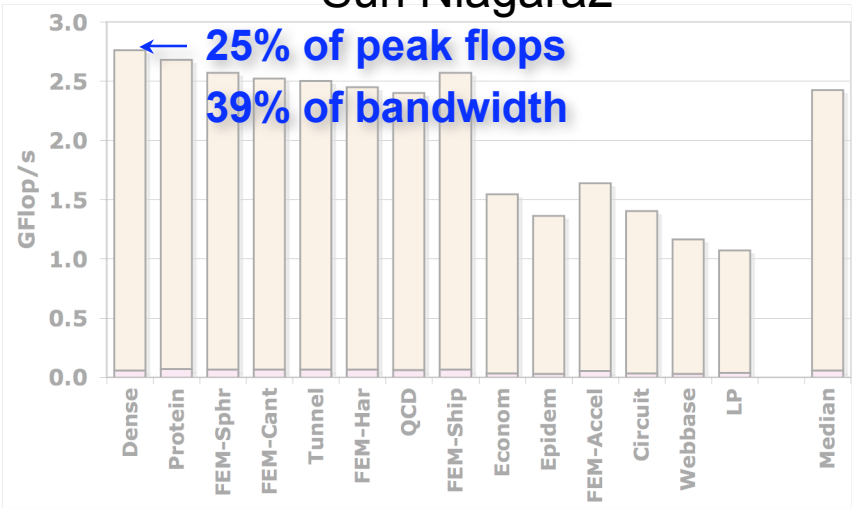
Intel Clovertown



AMD Opteron



Sun Niagara2



Naïve Pthreads

Naïve Single Thread

Office of Science
U.S. DEPARTMENT OF ENERGY

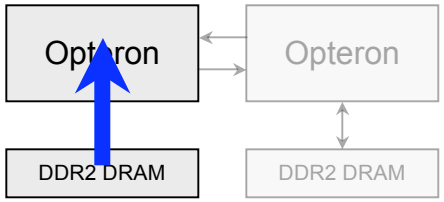
Case for Autotuning

- How do we deliver good performance across all these architectures, across all matrices without exhaustively optimizing every combination

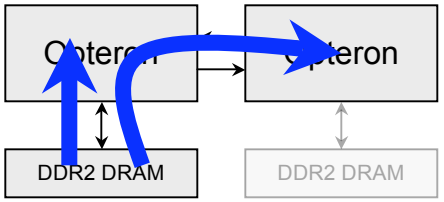
- Autotuning
  - Write a Perl script that generates all possible optimizations
  - Heuristically, or exhaustively search the optimizations
  - Existing SpMV solution: OSKI (developed at UCB)
- This work:
  - Optimizations geared for multi-core/-threading
  - generates SSE/SIMD intrinsics, prefetching, loop transformations, alternate data structures, etc…
  - “prototype for parallel OSKI”

Exploiting NUMA, Affinity

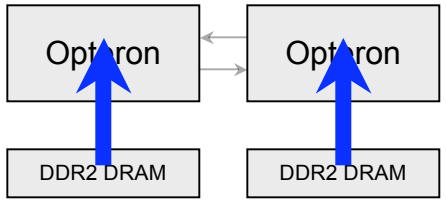
- Bandwidth on the Opteron(and Cell) can vary substantially based on placement of data



Single Thread



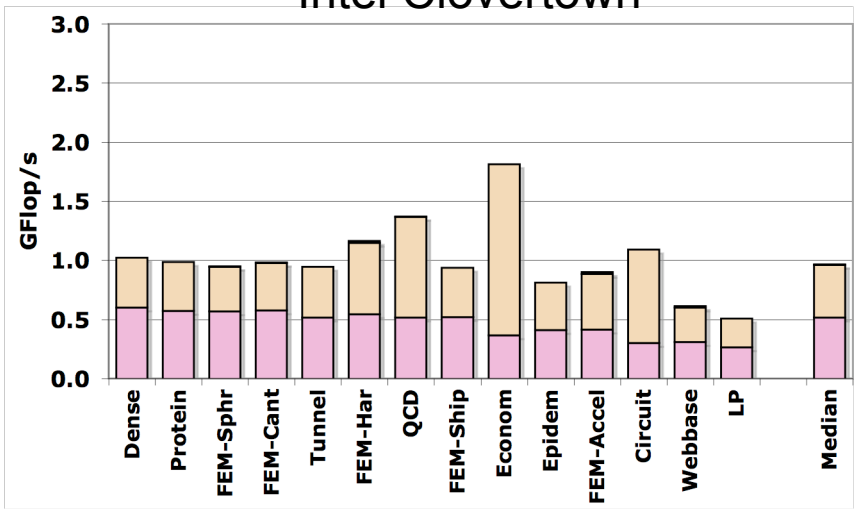
Multiple Threads,
One memory controller

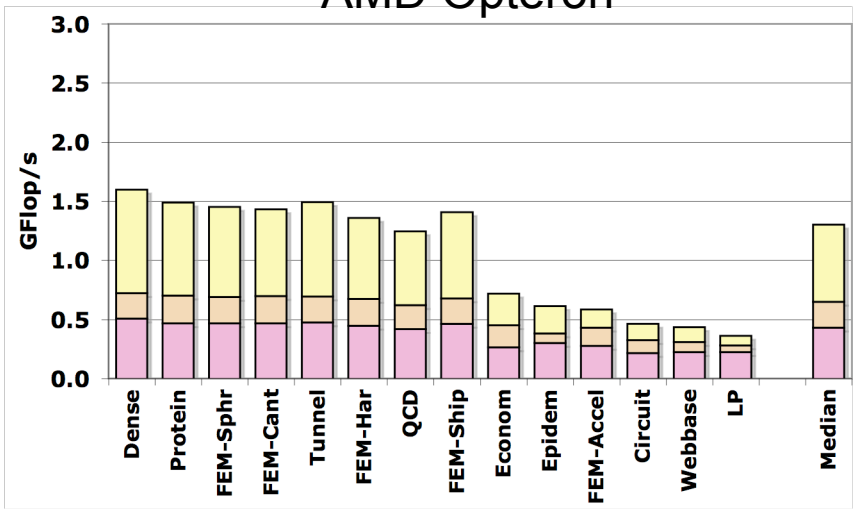


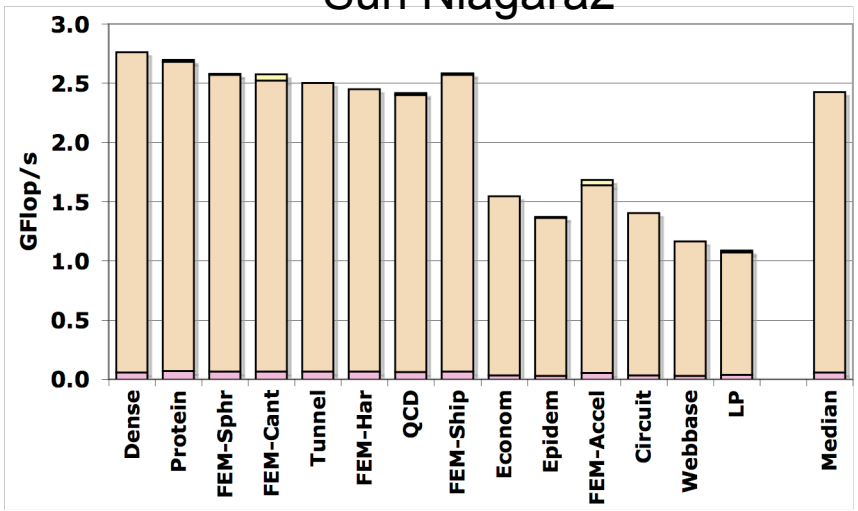
Multiple Threads,
Both memory controllers

- Bind each sub matrix and the thread to process it together
- Explored libnuma, Linux, and Solaris routines
- Adjacent blocks bound to adjacent cores

Performance (+NUMA)







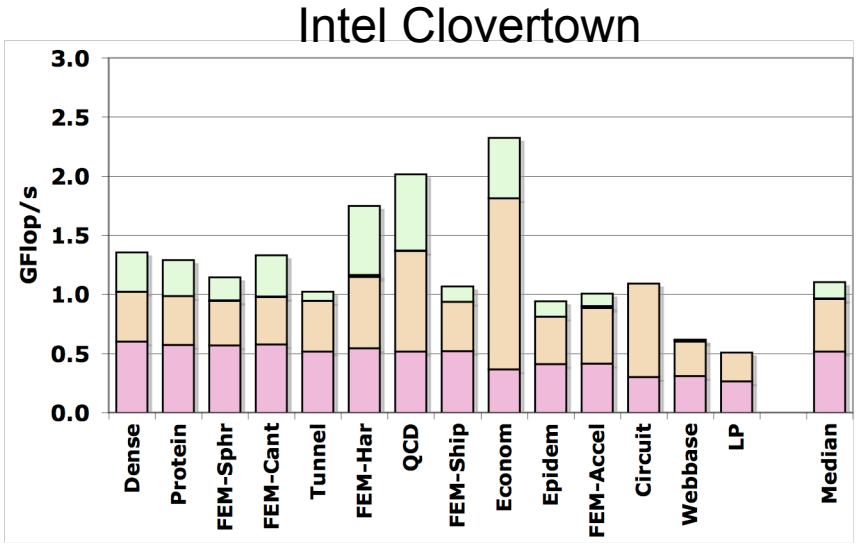
+NUMA/Affinity

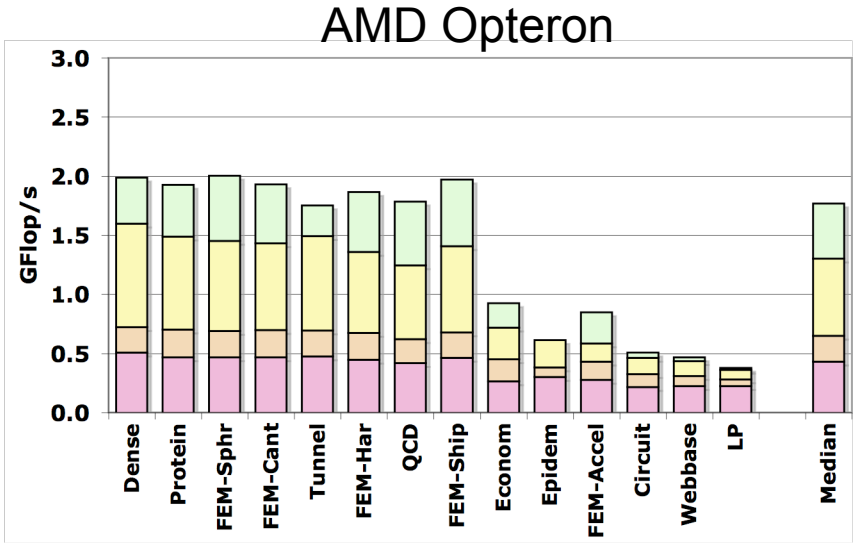
Naïve Pthreads

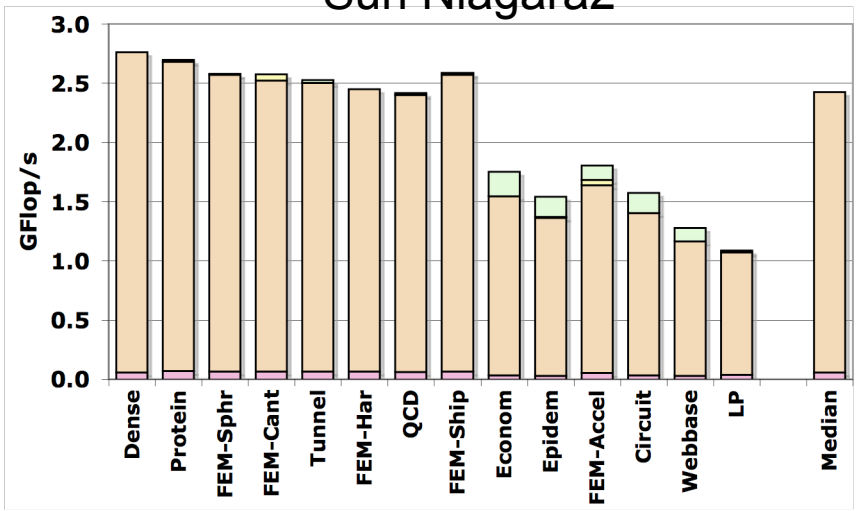
Naïve Single Thread

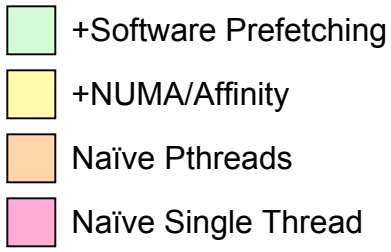
Office of Science
U.S. DEPARTMENT OF ENERGY

Performance (+SW Prefetching)



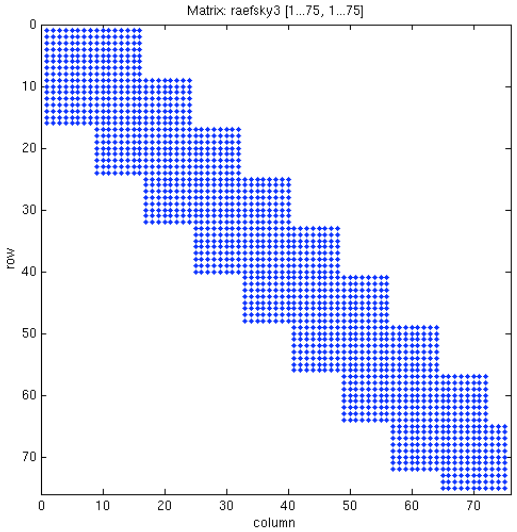






Matrix Compression

- For memory bound kernels, minimizing memory traffic should maximize performance
- Compress the meta data
  - Exploit structure to eliminate meta data

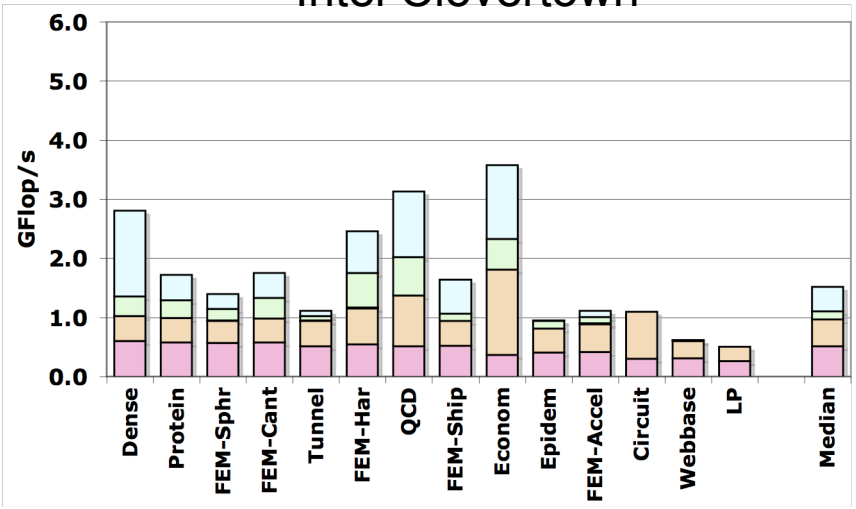


- **Heuristic**: select the compression that minimizes the matrix size:
  - power of 2 register blocking
  - CSR/COO format
  - 16b/32b indices
  - etc…

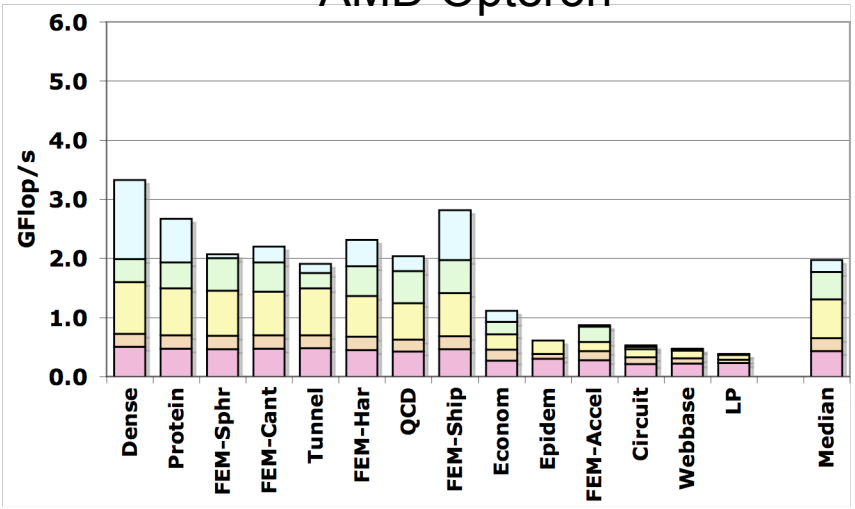
- **Side effect**: matrix may be minimized to the point where it fits entirely in cache

Performance (+matrix compression)

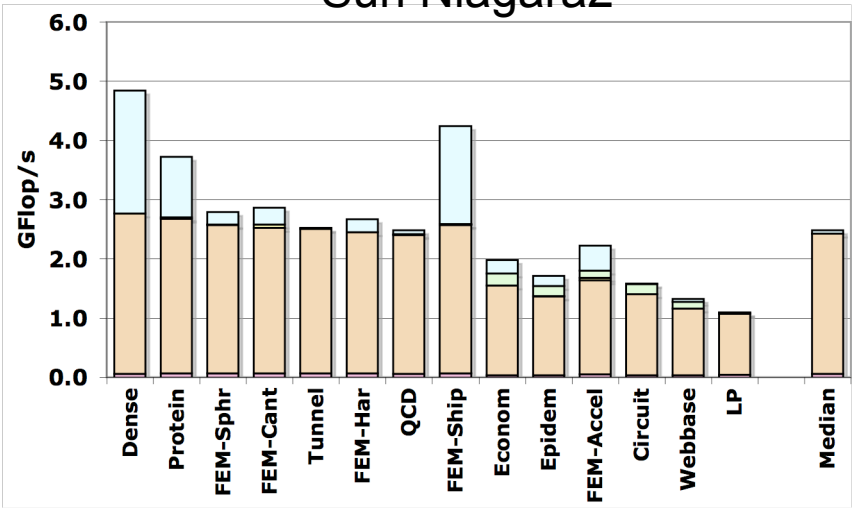
Intel Clovertown

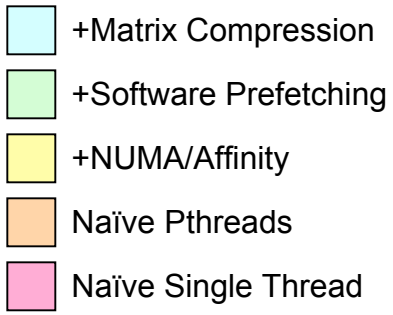


AMD Opteron



Sun Niagara2





Office of Science
U.S. DEPARTMENT OF ENERGY

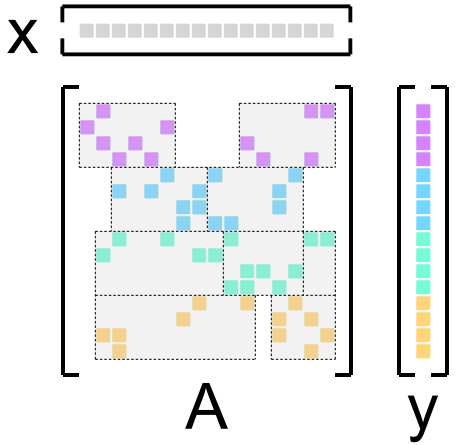
Cache and TLB Blocking

- Accesses to the matrix and destination vector are streaming
- But, access to the source vector can be random
- Reorganize matrix (and thus access pattern) to maximize reuse.
- Applies equally to TLB blocking (caching PTEs)

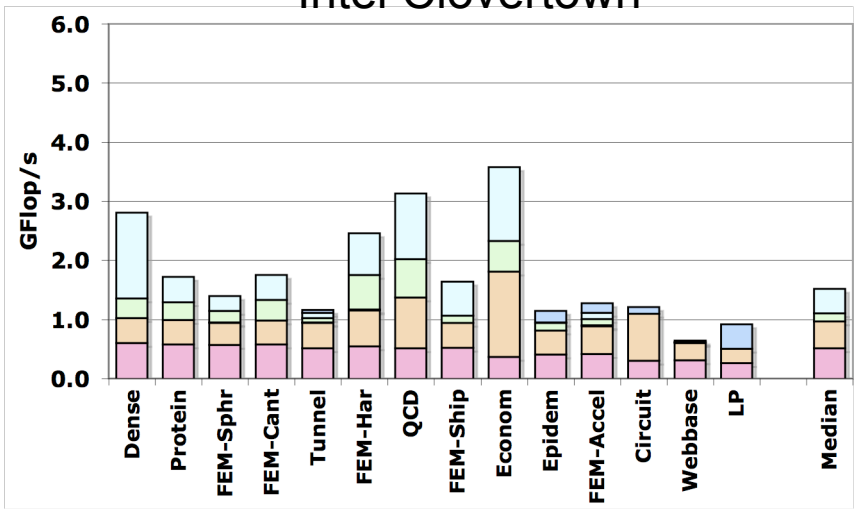
- **Heuristic**: block destination, then keep adding more columns as long as the number of source vector cache lines(or pages) touched is less than the cache(or TLB). Apply all previous optimizations individually to each cache block.

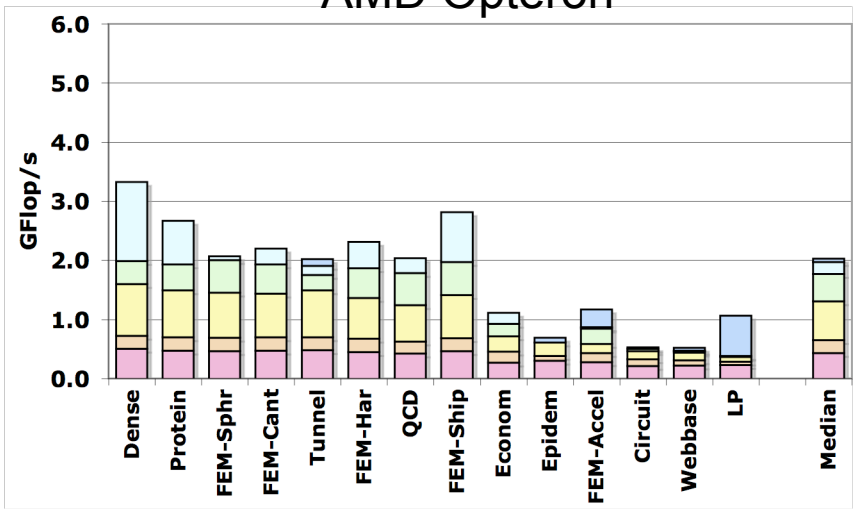
- **Search**: neither, cache, cache&TLB

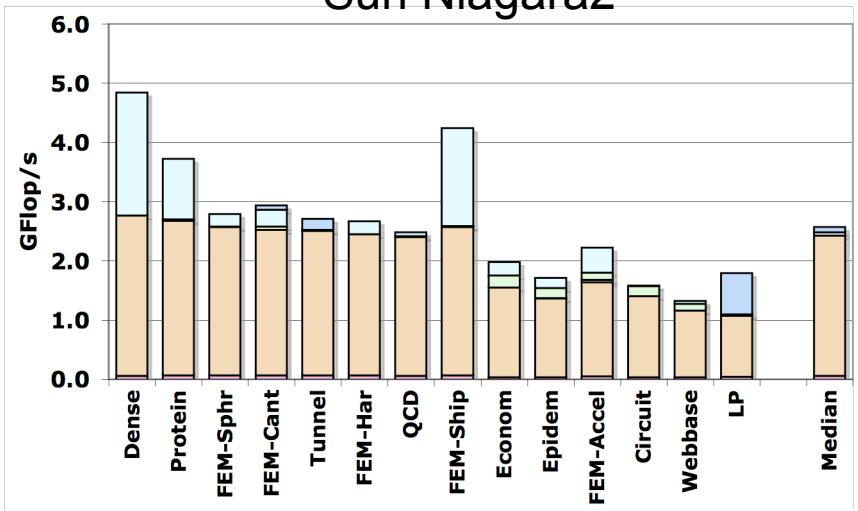
- Better locality at the expense of confusing the hardware prefetchers.

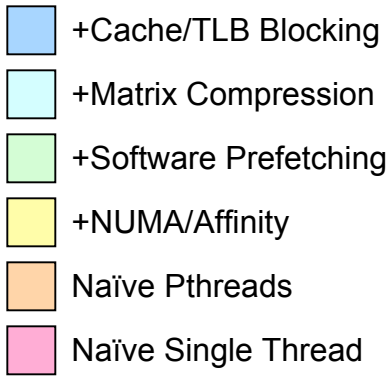


Performance (+cache blocking)









Office of Science
U.S. DEPARTMENT OF ENERGY

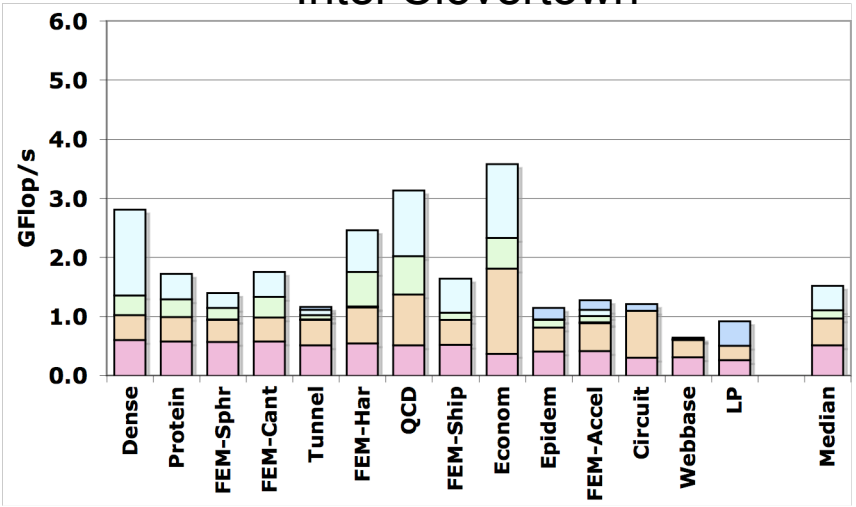
Banks, Ranks, and DIMMs

- In this SPMD approach, as the number of threads increases, so to does the number of concurrent streams to memory.
- Most memory controllers have finite capability to reorder the requests. (DMA can avoid or minimize this)
- Addressing/Bank conflicts become increasingly likely

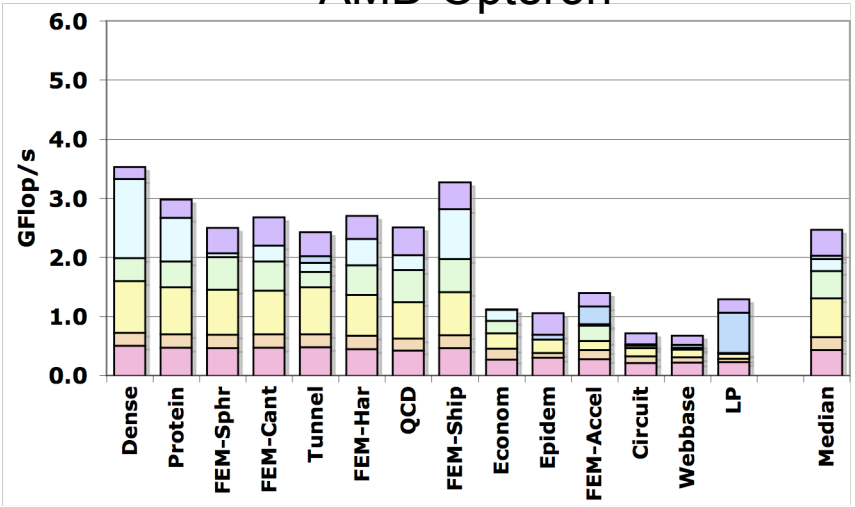
- Add more DIMMs, configuration of ranks can help
- Clovertown system was already fully populated

Performance (more DIMMs, …)

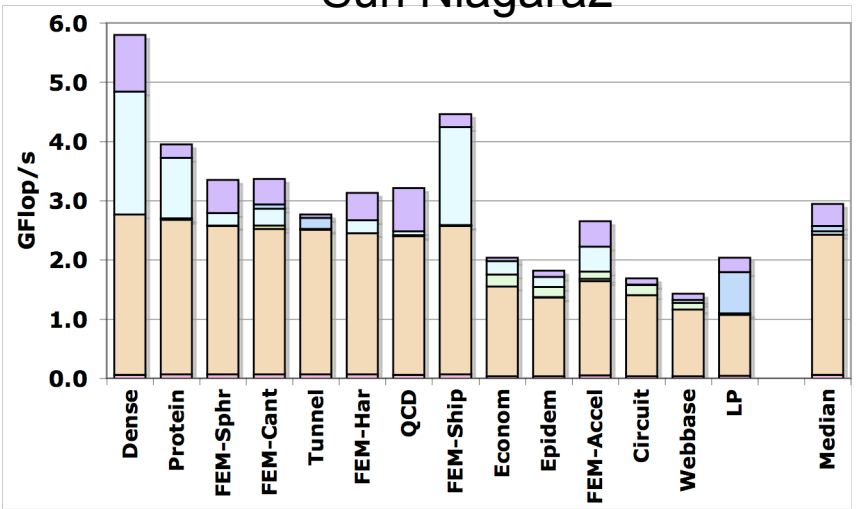
Intel Clovertown

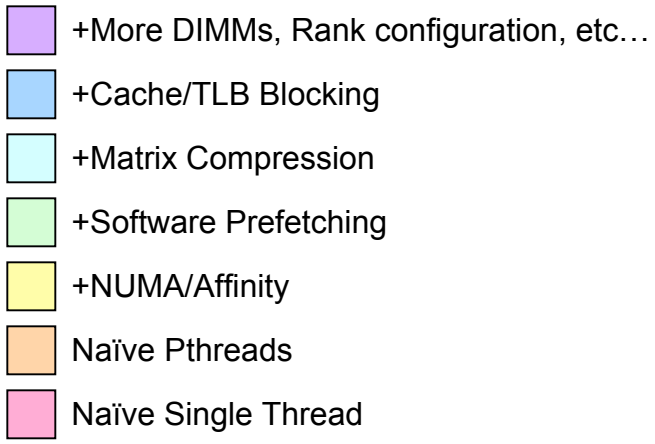


AMD Opteron



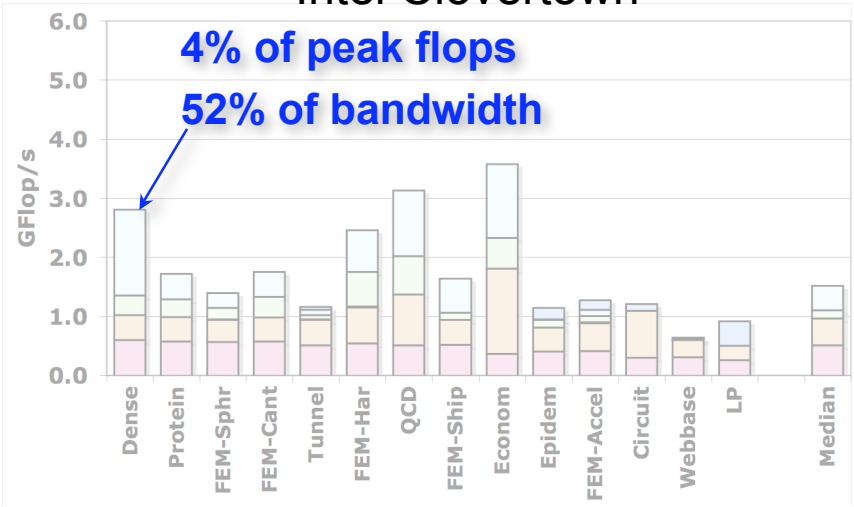
Sun Niagara2



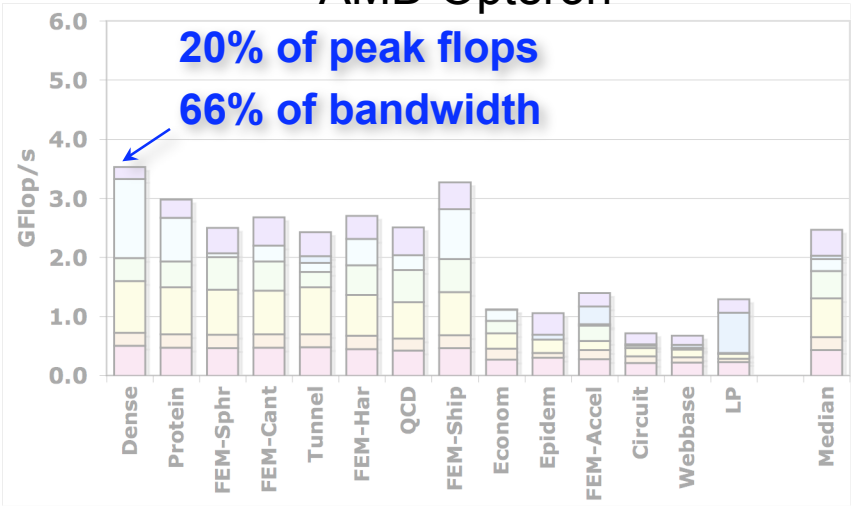


Performance (more DIMMs, …)

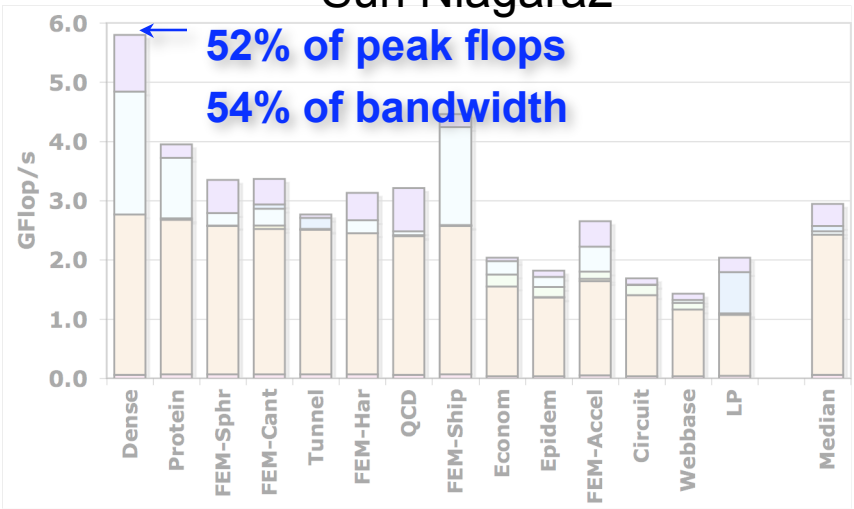
Intel Clovertown

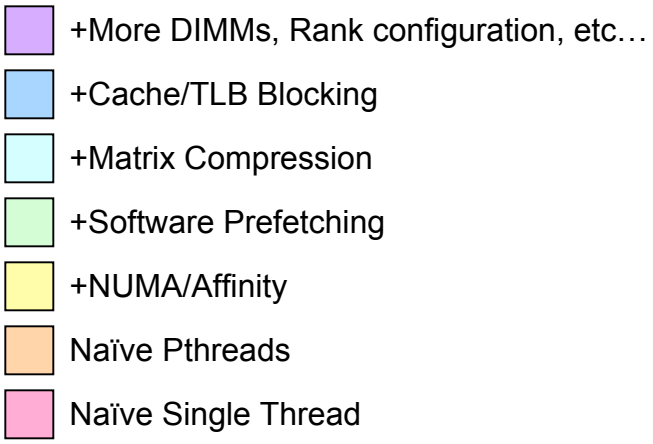


AMD Opteron



Sun Niagara2

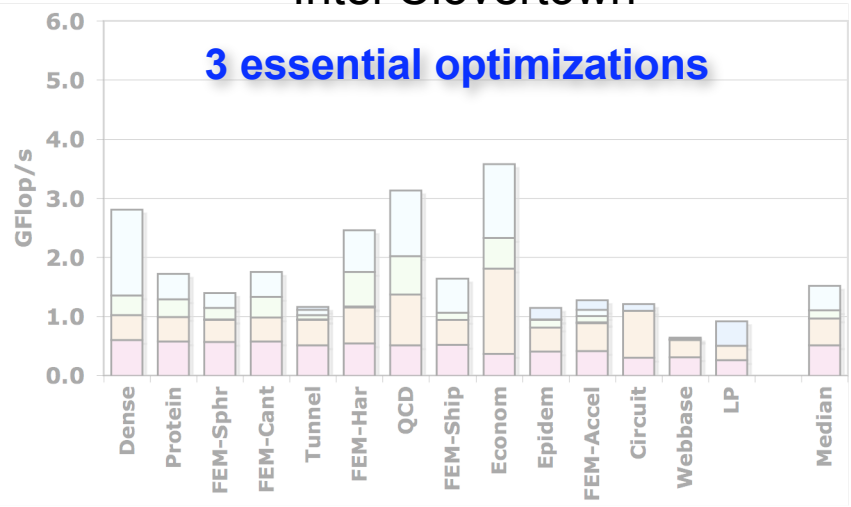




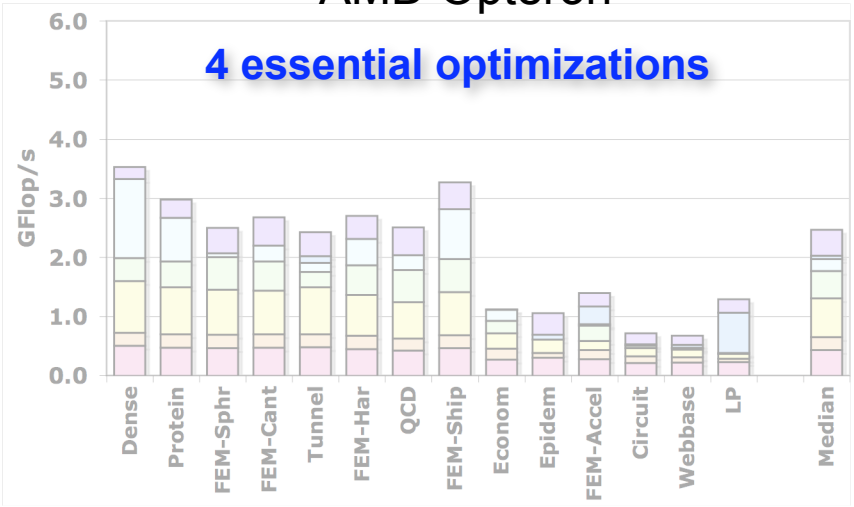
Office of Science
U.S. DEPARTMENT OF ENERGY

Performance (more DIMMs, …)

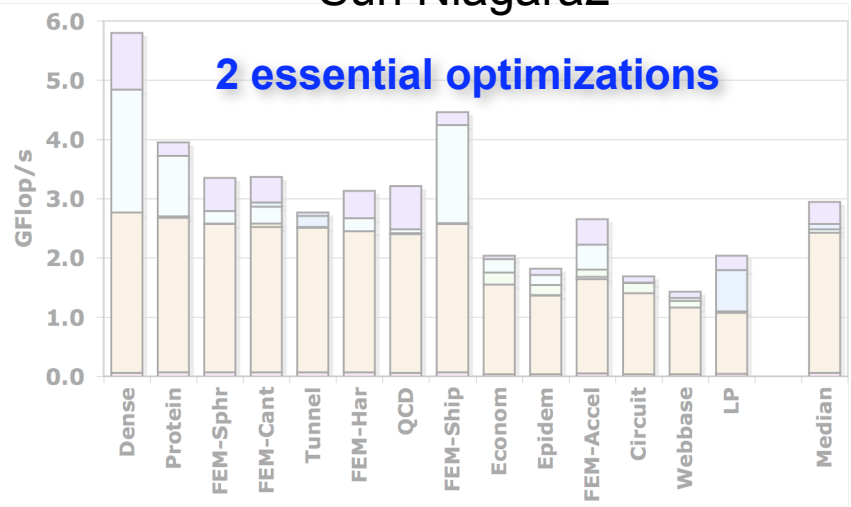
Intel Clovertown



AMD Opteron



Sun Niagara2



- +More DIMMs, Rank configuration, etc…
- +Cache/TLB Blocking
- +Matrix Compression
- +Software Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve Single Thread

Office of Science
U.S. DEPARTMENT OF ENERGY

Cell Implementation

- Comments
- Performance

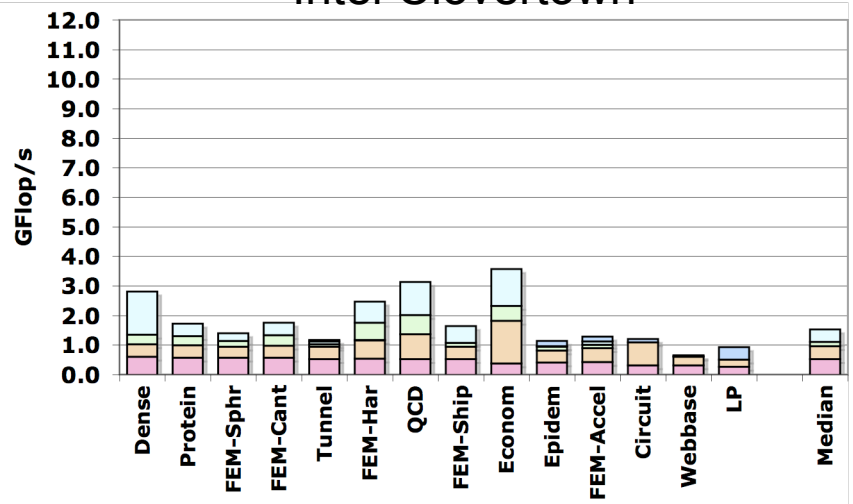
Cell Implementation

- **No vanilla C implementation (aside from the PPE)**
- Even SIMDized double precision is extremely weak
  - Scalar double precision is unbearable
  - Minimum register blocking is 2x1 (SIMDizable)
  - **Can increase memory traffic by 66%**
- Cache blocking optimization is transformed into local store blocking
  - **Spatial and temporal locality is captured by software when the matrix is optimized**
  - In essence, the high bits of column indices are grouped into DMA lists
- No branch prediction
  - Replace branches with conditional operations
- In some cases, what were optional optimizations on cache based machines, are requirements for correctness on Cell

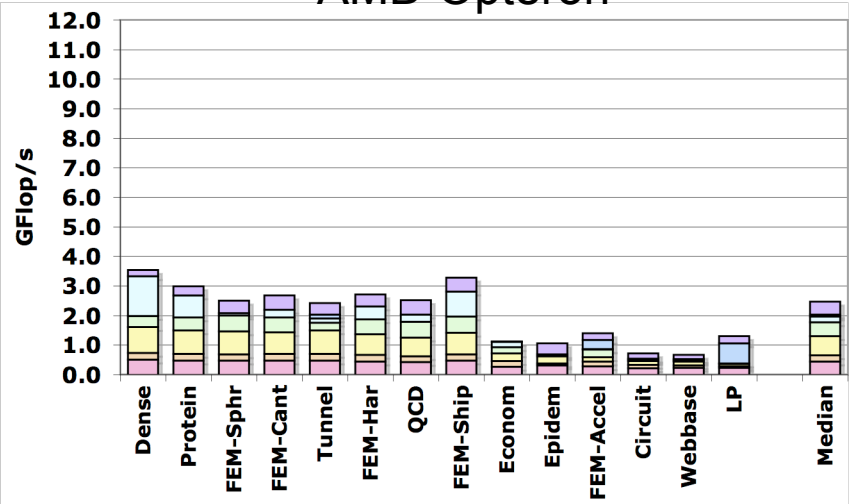
- Despite the performance, Cell is still handicapped by double precision

Performance

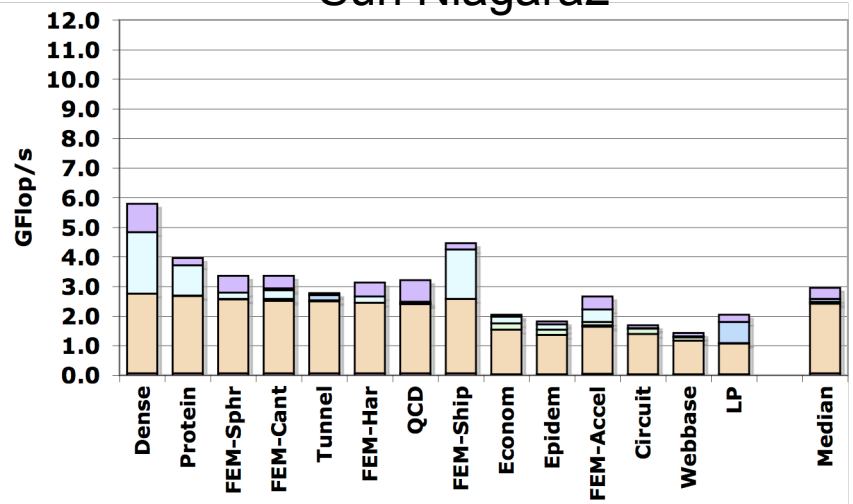
Intel Clovertown



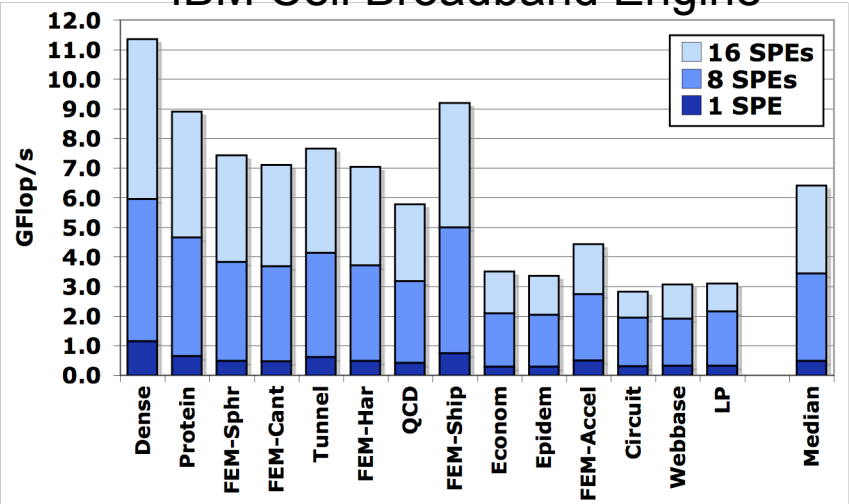
AMD Opteron



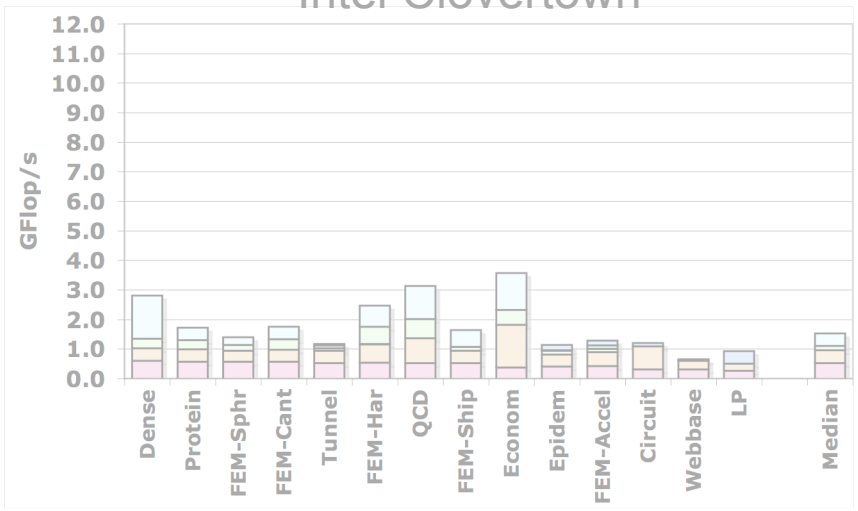
Sun Niagara2

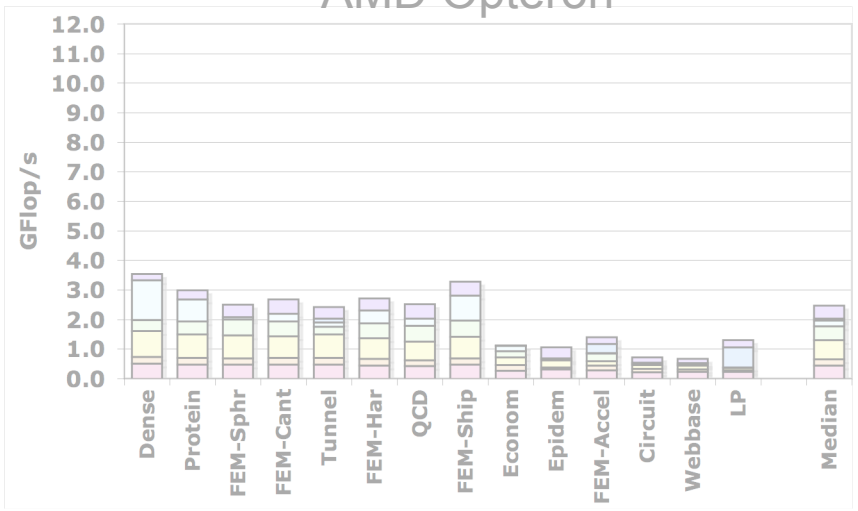


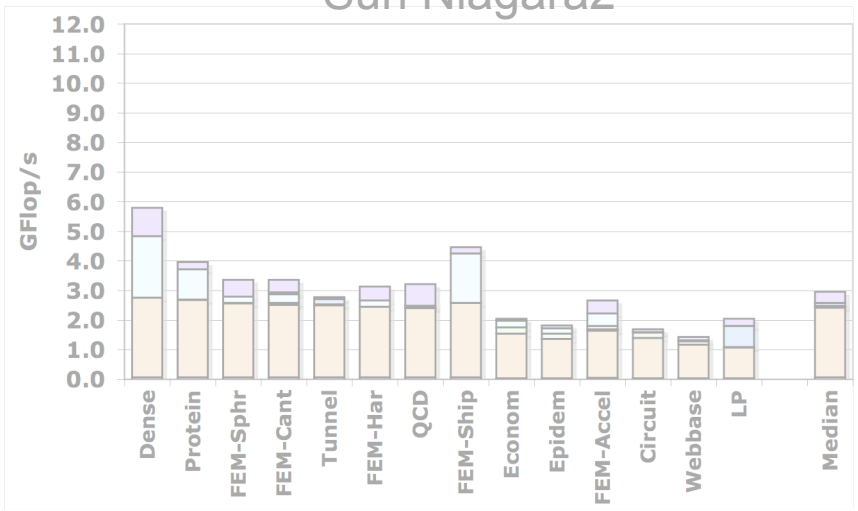
IBM Cell Broadband Engine

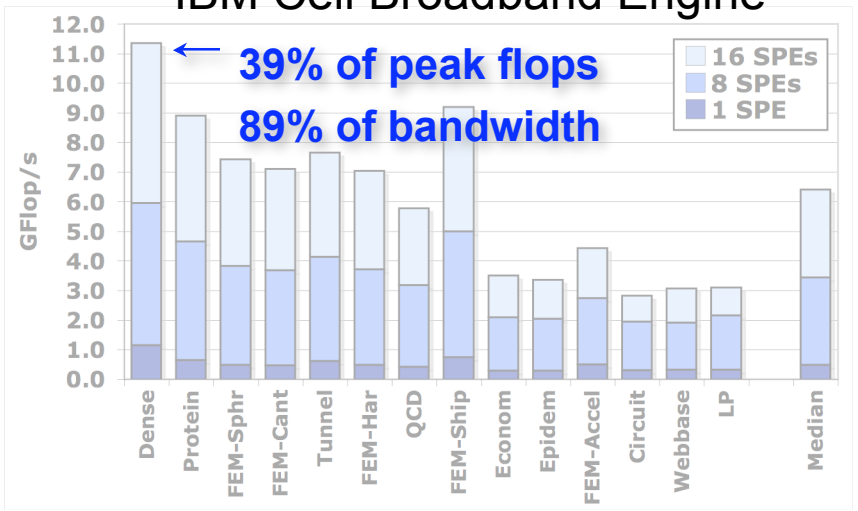


Performance









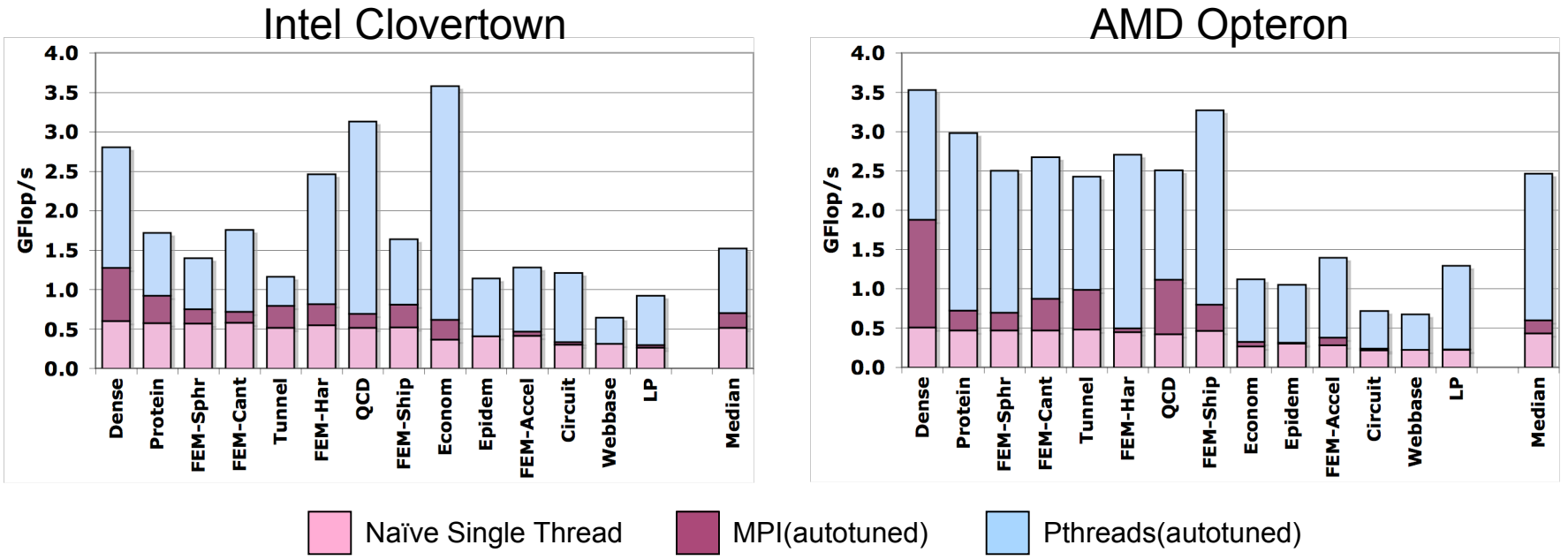
Office of Science
U.S. DEPARTMENT OF ENERGY

Multicore MPI Implementation

- This is the default approach to programming multicore

Multicore MPI Implementation

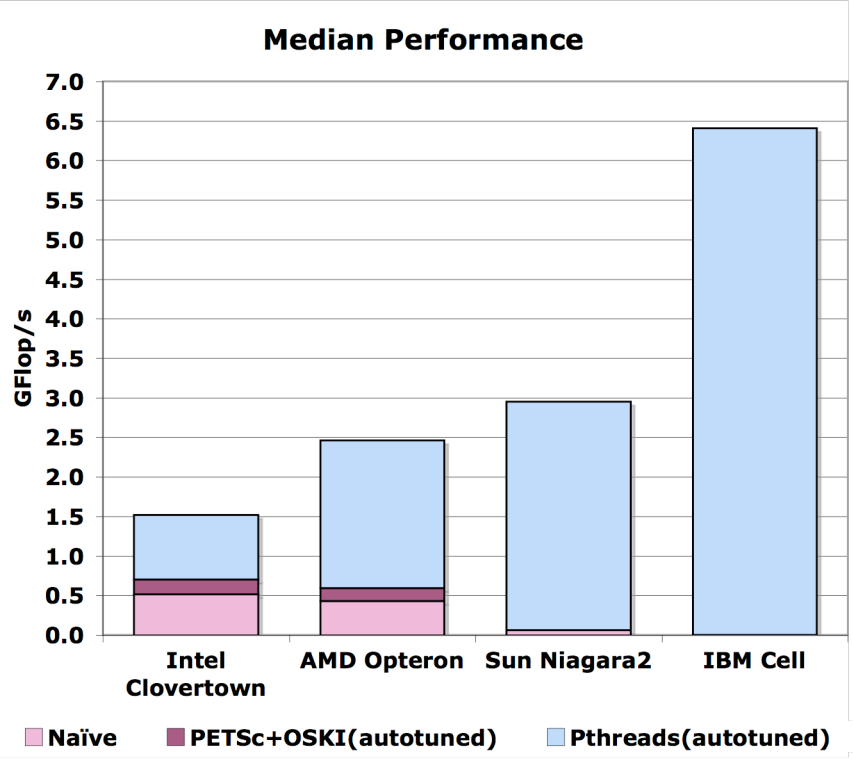
- Used PETSc with shared memory MPICH
- Used OSKI (developed @ UCB) to optimize each thread
- = Highly optimized MPI

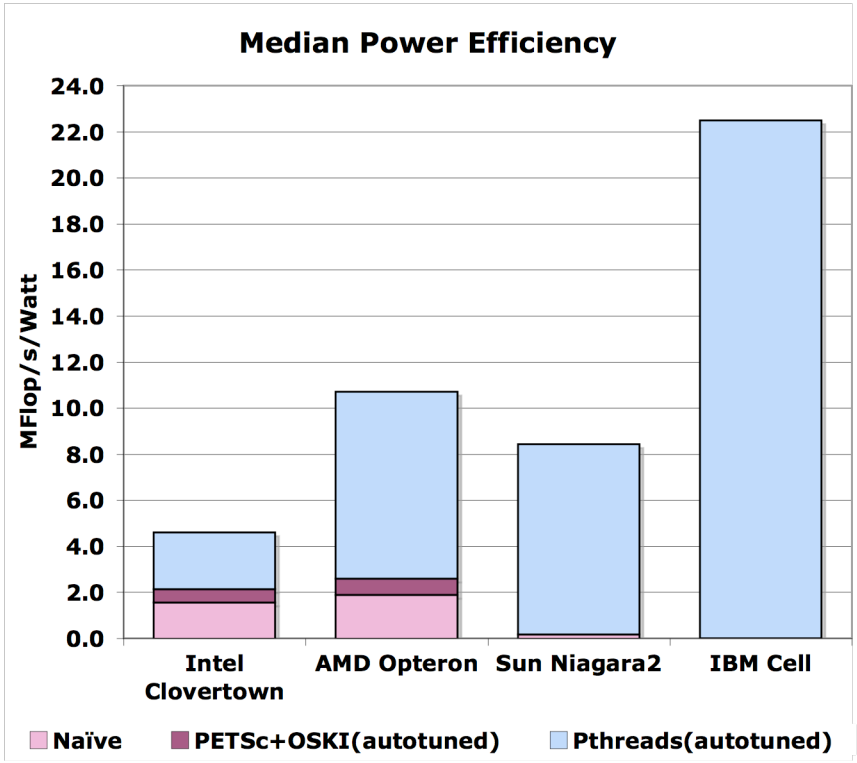


Summary

Median Performance & Efficiency

- Used digital power meter to measure sustained system power
  - FBDIMM drives up Clovertown and Niagara2 power
  - Right: sustained MFlop/s / sustained Watts
- Default approach(MPI) achieves very low performance and efficiency





Summary

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

- Most machines achieved less than 50-60% of DRAM bandwidth
- Niagara2 delivered both very good performance and productivity
- Cell delivered very good performance and efficiency
  - 90% of memory bandwidth
  - High power efficiency
  - Easily understood performance
  - Extra traffic = lower performance (future work can address this)

- **multicore specific autotuned implementation significantly outperformed a state of the art MPI implementation**
  - ✓ Matrix compression geared towards multicore
  - ✓ NUMA
  - ✓ Prefetching

Acknowledgments

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

Questions?