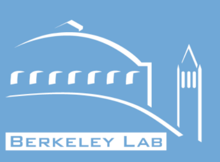


Tuning Sparse Matrix Vector Multiplication for multi-core SMPs

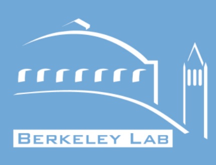
(details in paper at SC07)

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

Overview



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

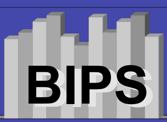
- Examined Sparse Matrix Vector Multiplication (SpMV) kernel
  - Important HPC kernel
  - Memory intensive
  - Challenging for multicore

- Present two auto-tuned threaded implementations:
  - Pthread, cache-based implementation
  - Cell local store-based implementation

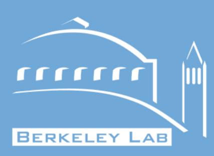
- Benchmarked performance across 4 diverse multicore architectures
  - Intel Xeon (Clovertown)
  - AMD Opteron
  - Sun Niagara2
  - IBM Cell Broadband Engine

- Compare with leading MPI implementation(PETSc) with an auto-tuned serial kernel (OSKI)

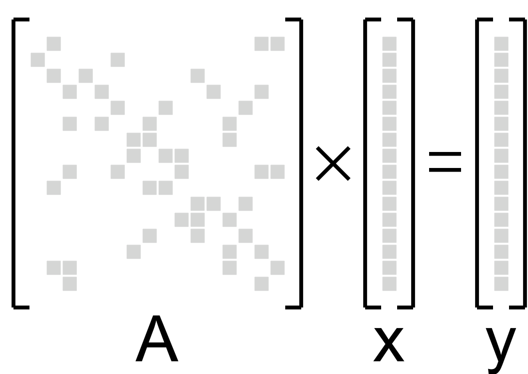
- Show Cell delivers good performance and efficiency, while Niagara2 delivers good performance and productivity

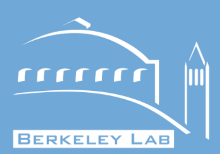


Sparse Matrix Vector Multiplication



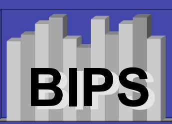
- Sparse Matrix
  - Most entries are 0.0
  - Performance advantage in only storing/operating on the nonzeros
  - Requires significant meta data
- Evaluate y=Ax
  - A is a sparse matrix
  - x & y are dense vectors
- Challenges
  - Difficult to exploit ILP(bad for superscalar),
  - Difficult to exploit DLP(bad for SIMD)
  - Irregular memory access to source vector
  - Difficult to load balance
  - **Very low computational intensity (often >6 bytes/flop)**



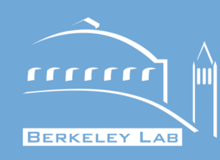


Test Suite

- Dataset (Matrices)
- Multicore SMPs



Matrices Used

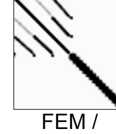


2K x 2K Dense matrix stored in sparse format

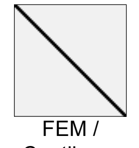
Dense

Well Structured (sorted by nonzeros/row)

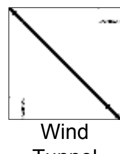
Protein



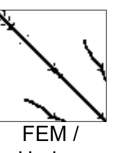
FEM / Spheres



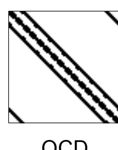
FEM / Cantilever



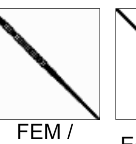
Wind Tunnel



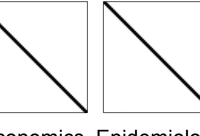
FEM / Harbor



QCD

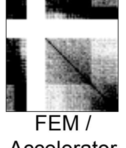


FEM / Ship

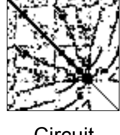


Economics Epidemiology

Poorly Structured hodgepodge



FEM / Accelerator



Circuit

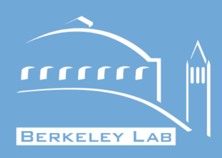
webbase

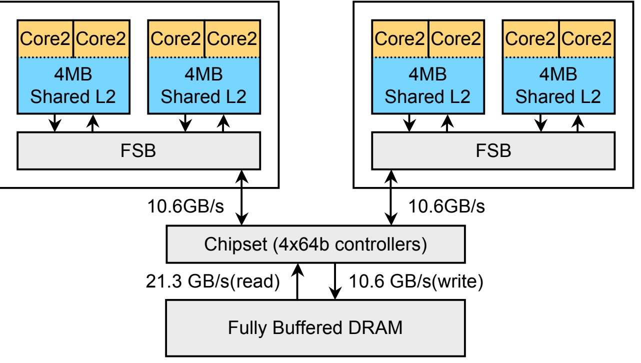
Extreme Aspect Ratio (linear programming)

LP

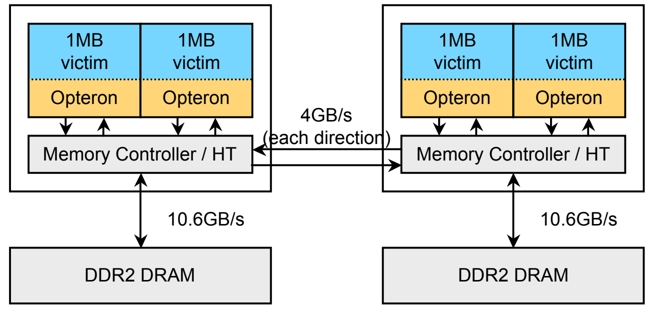
- Pruned original SPARSITY suite down to 14
- none should fit in cache
- Subdivided them into 4 categories
- Rank ranges from 2K to 1M

Multicore SMP Systems

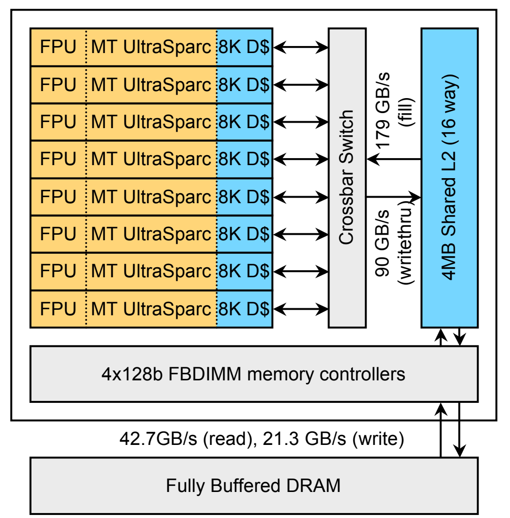




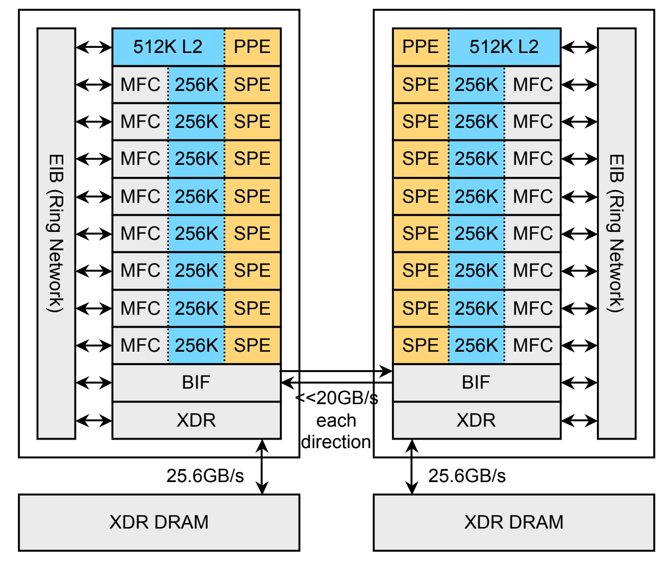
Intel Clovertown



AMD Opteron



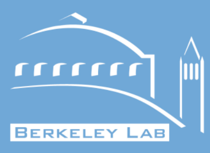
Sun Niagara2

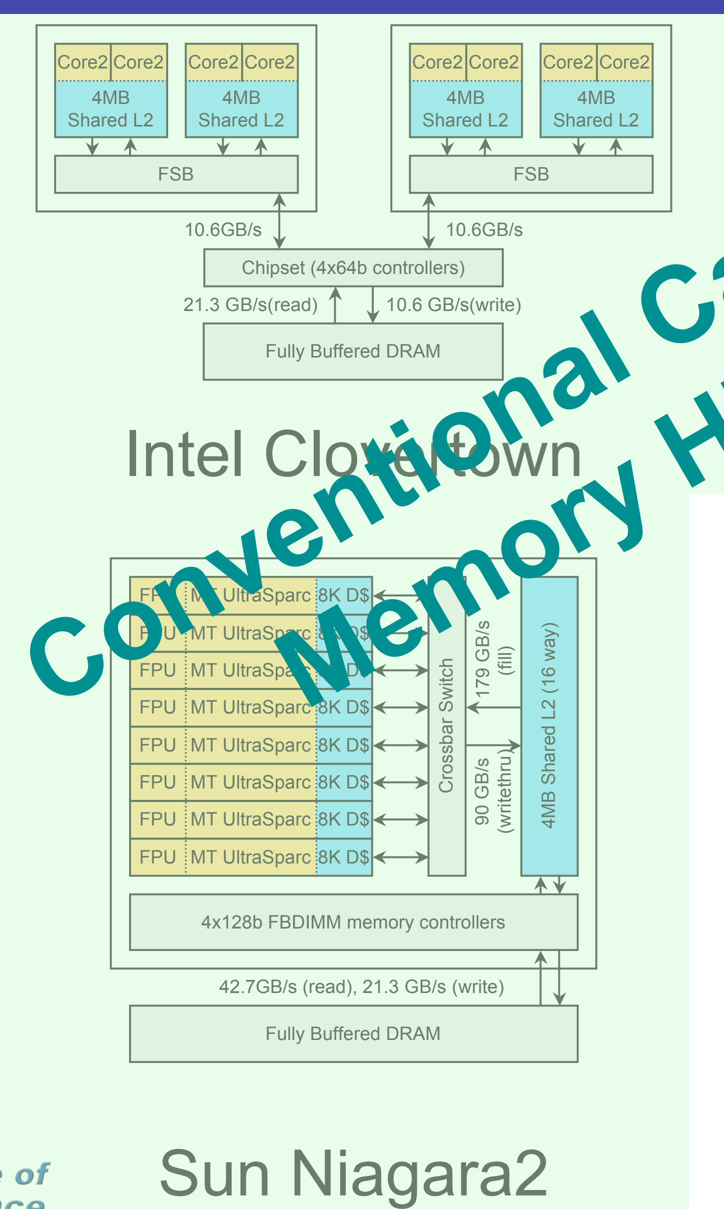


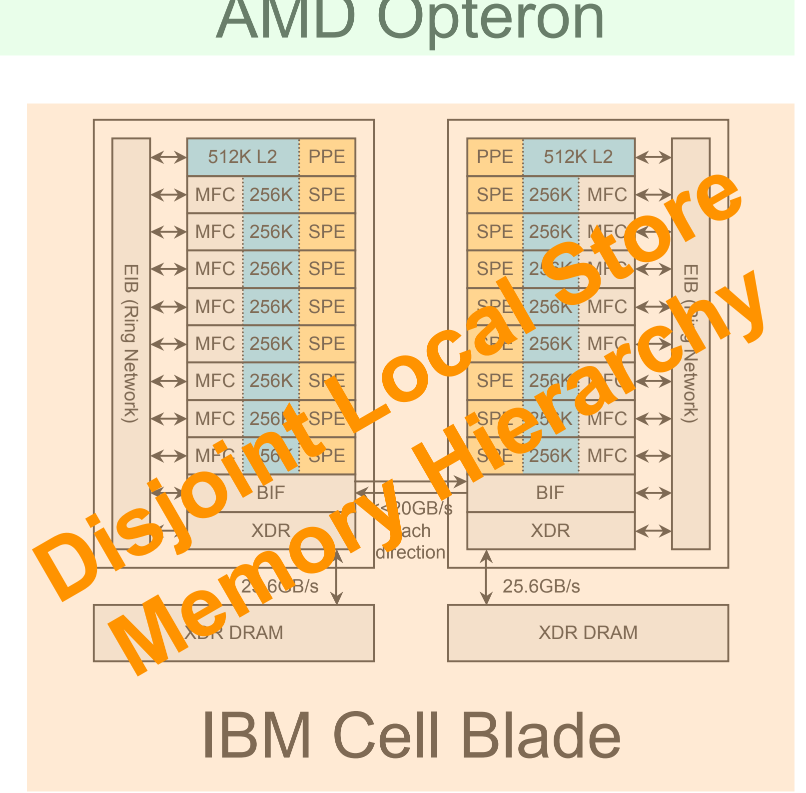
IBM Cell Blade

Multicore SMP Systems

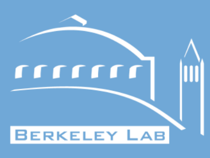
(memory hierarchy)

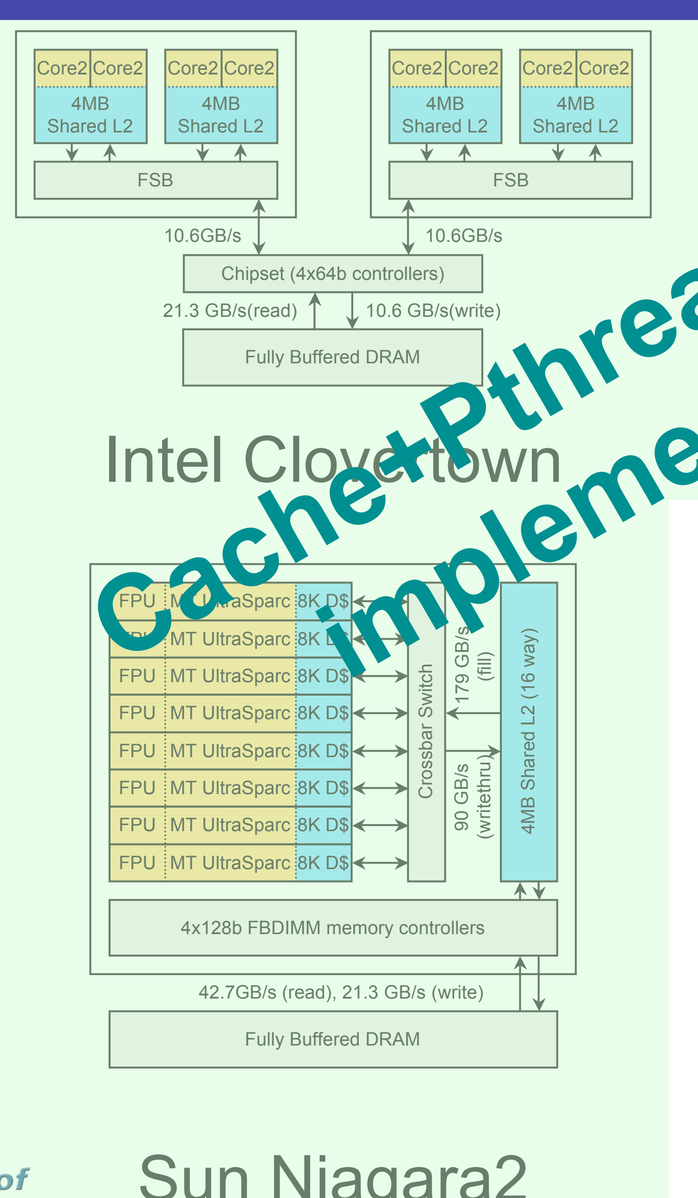


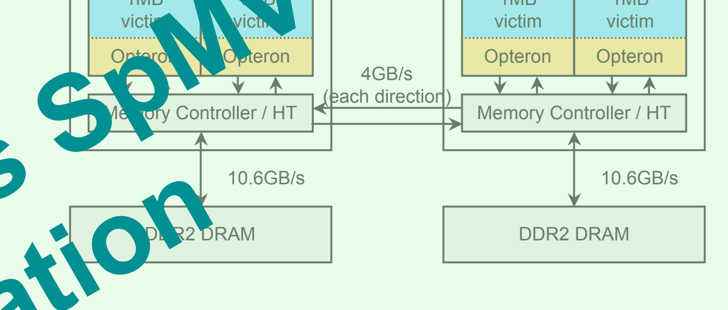


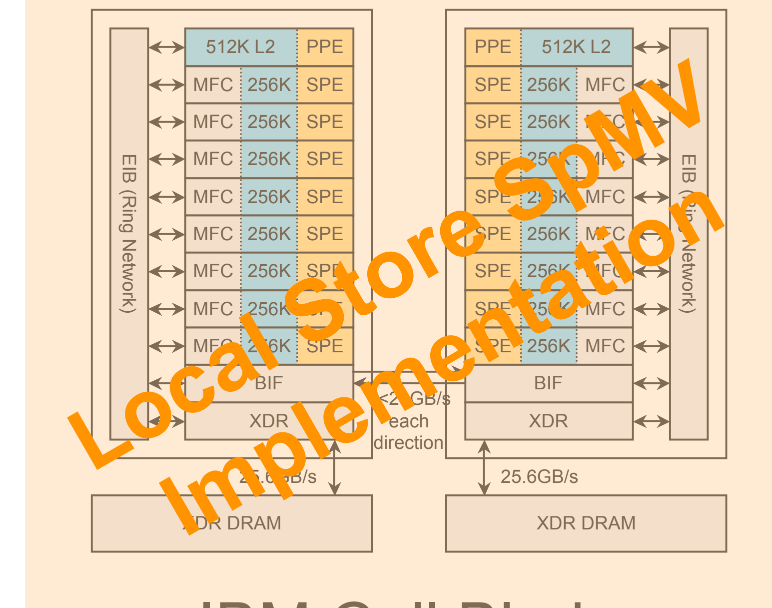


Multicore SMP Systems
(2 implementations)

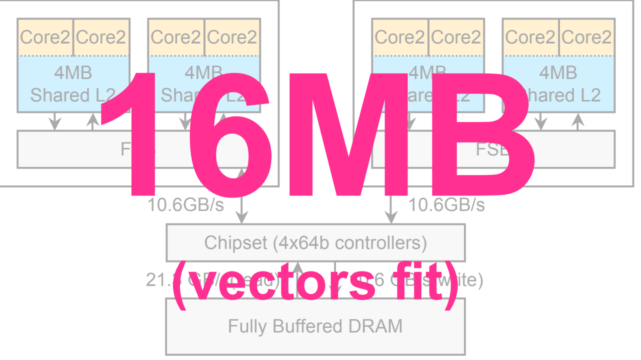




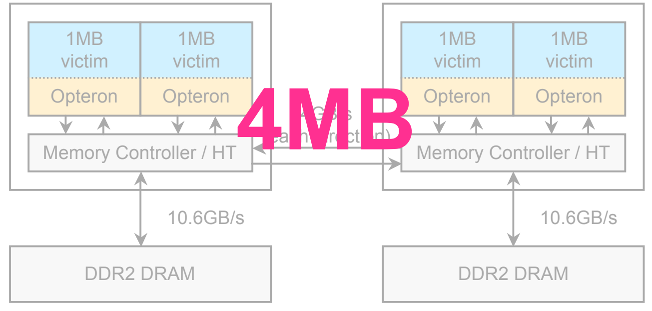




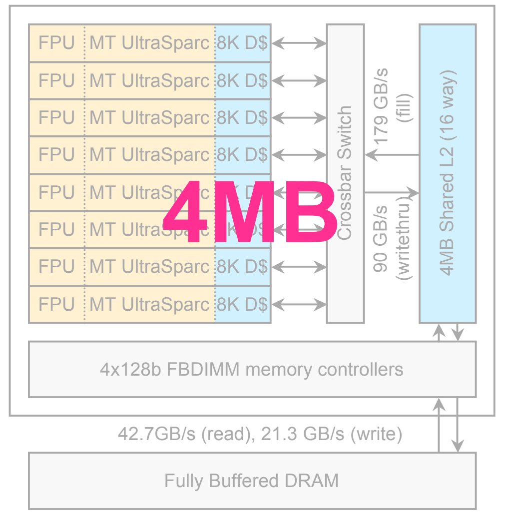
Multicore SMP Systems
(cache)



Intel Clovertown



AMD Opteron

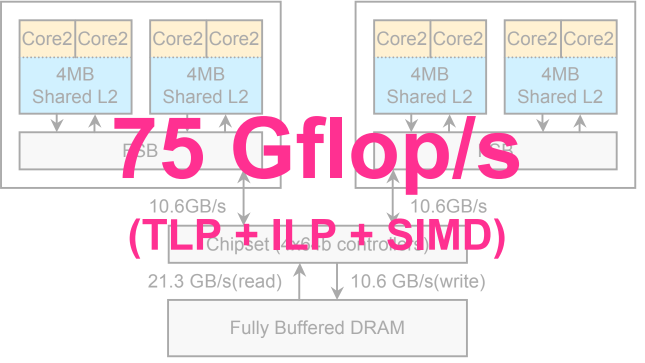


Sun Niagara2

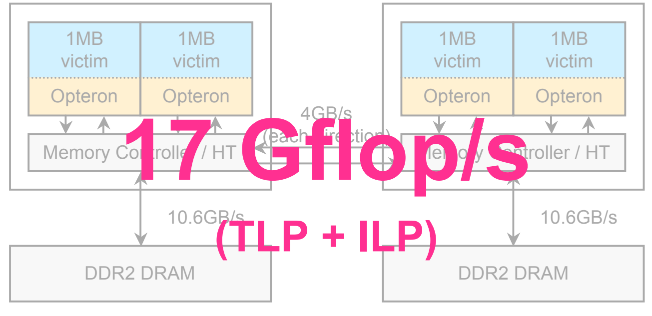
IBM Cell Blade

Multicore SMP Systems

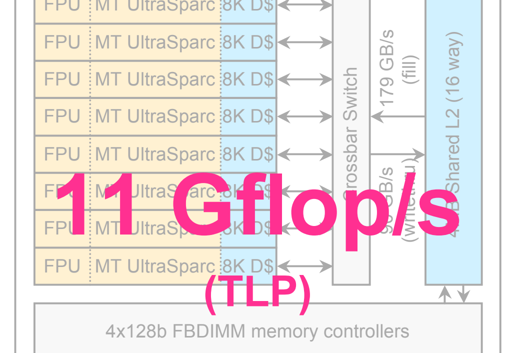
(peak flops)



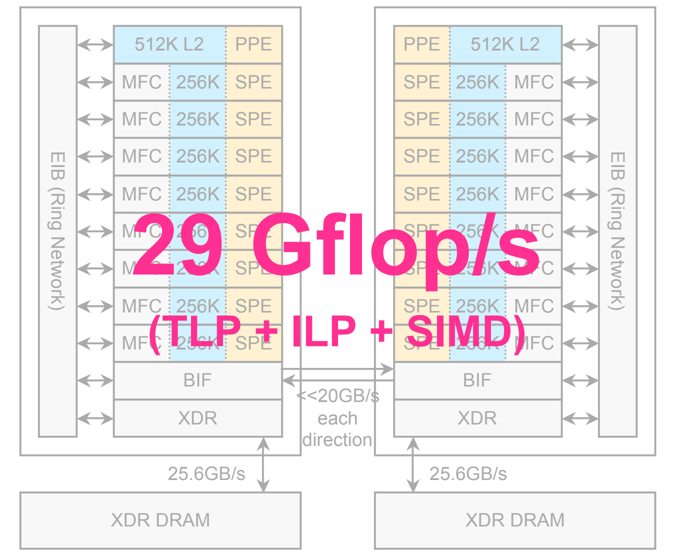
Intel Clovertown



AMD Opteron

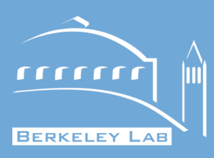


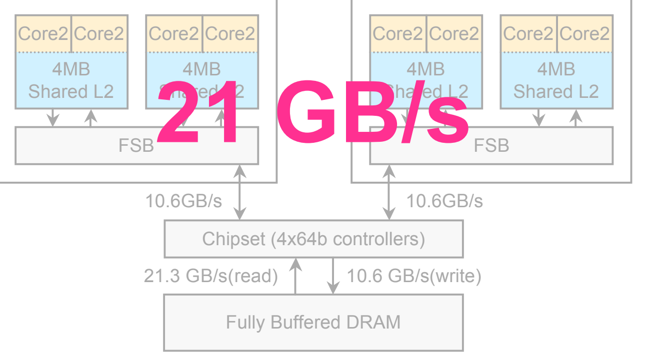
Sun Niagara2



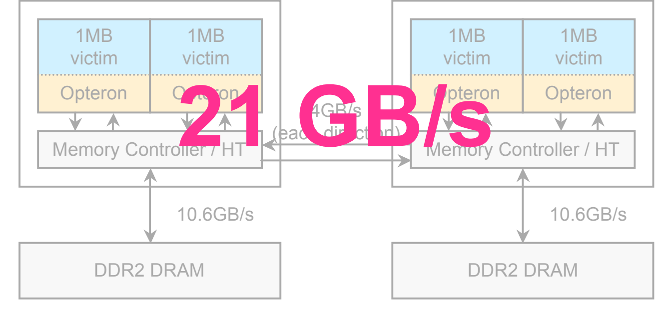
IBM Cell Blade

Multicore SMP Systems
(peak read bandwidth)

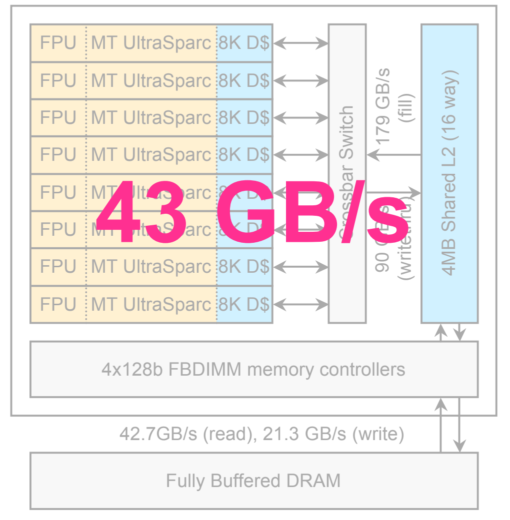




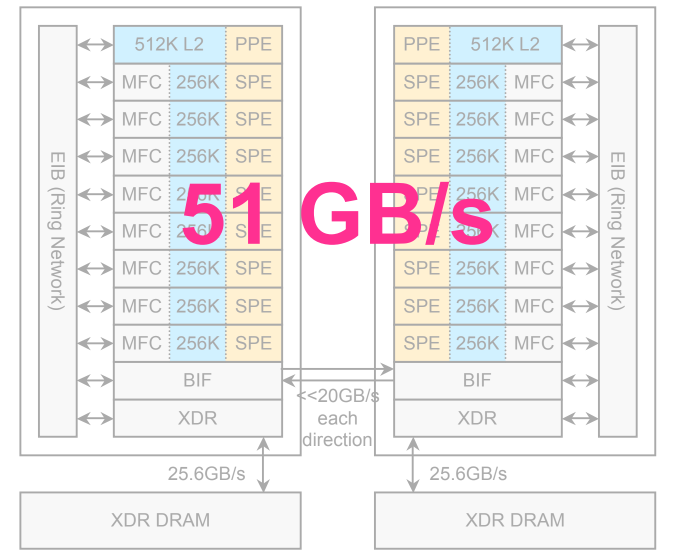
Intel Clovertown



AMD Opteron

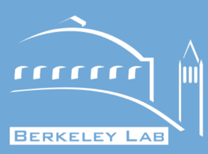


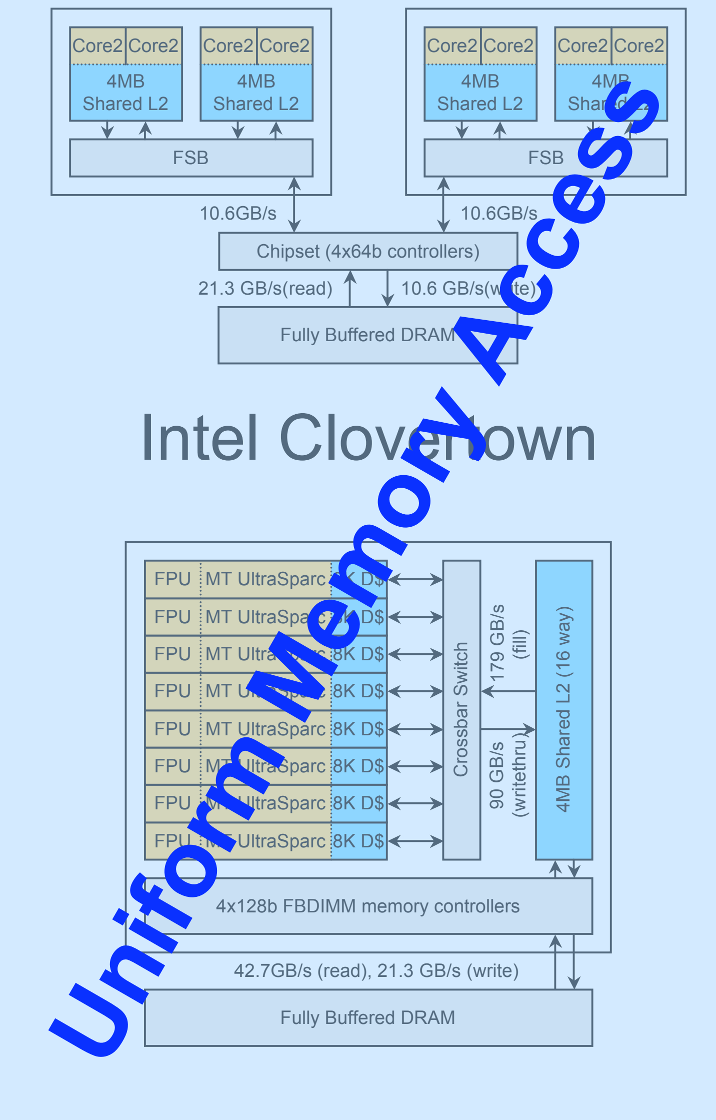
Sun Niagara2



IBM Cell Blade

Multicore SMP Systems
(NUMA)

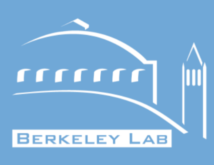


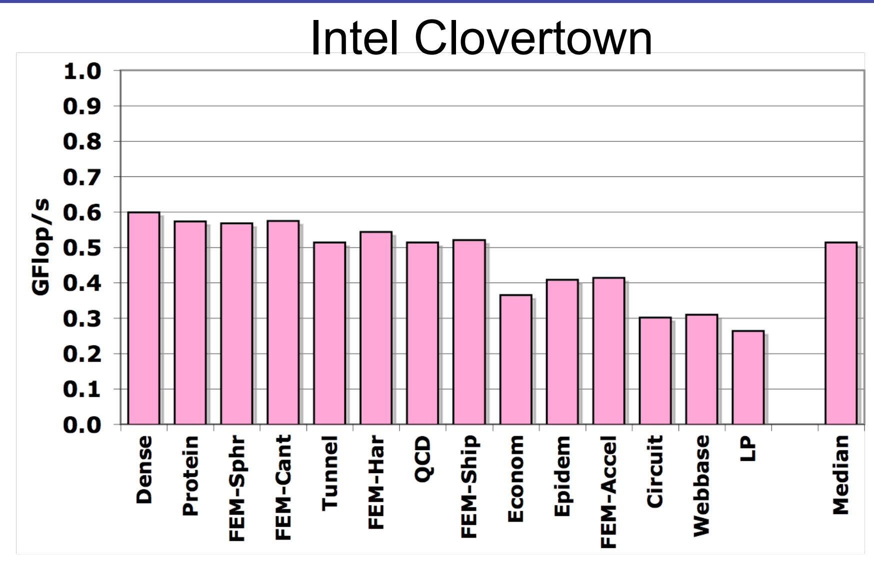


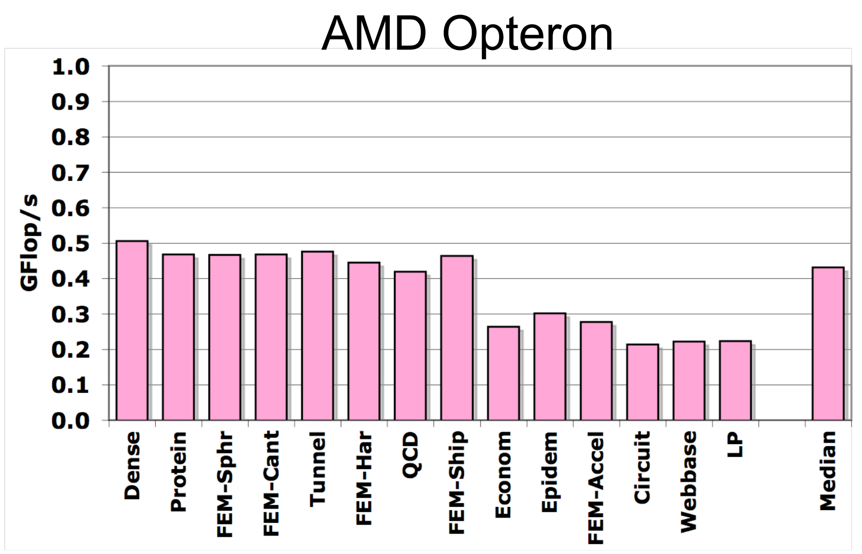
Naïve Implementation for Cache-based Machines

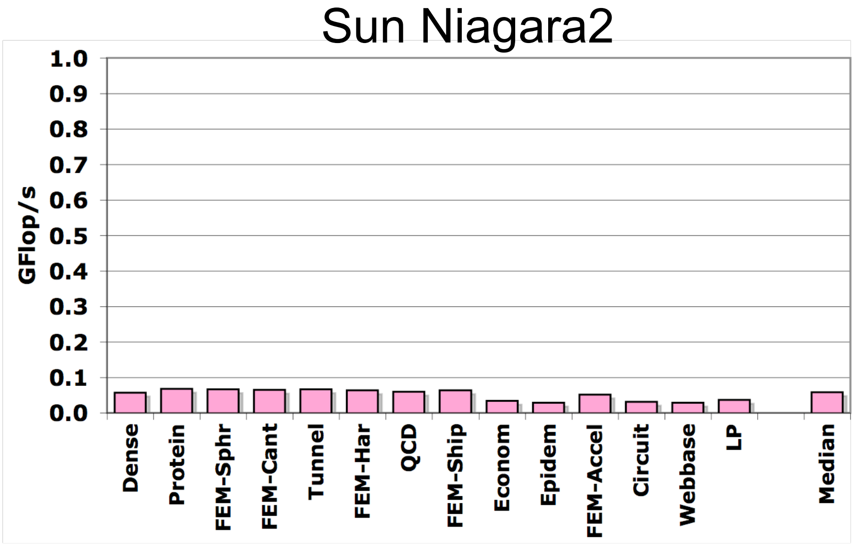
- Performance across suite
- Also, included a median performance number

vanilla C Performance

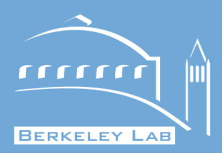








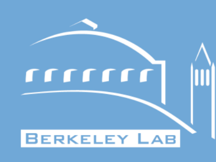
- Vanilla C implementation
- Matrix stored in CSR (compressed sparse row)
- Explored compiler options - only the best is presented here
- x86 core delivers > 10x performance of Niagara2 thread

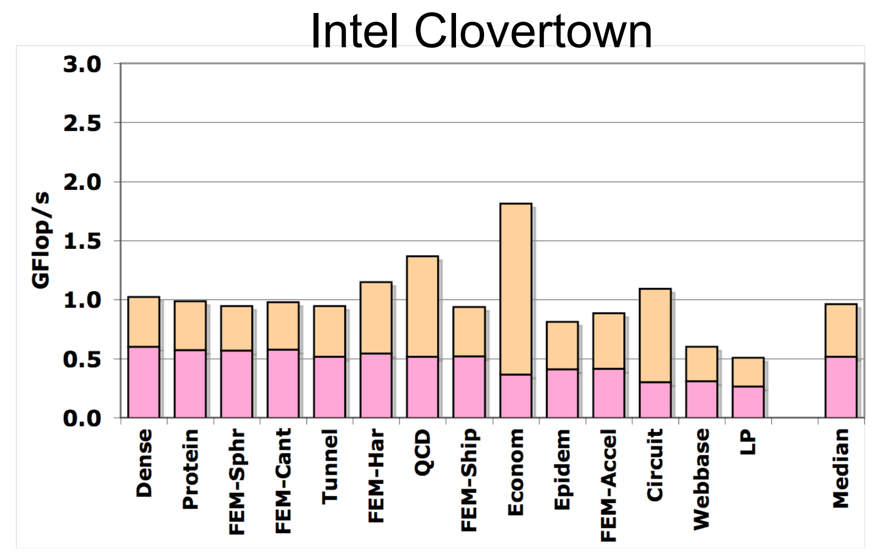


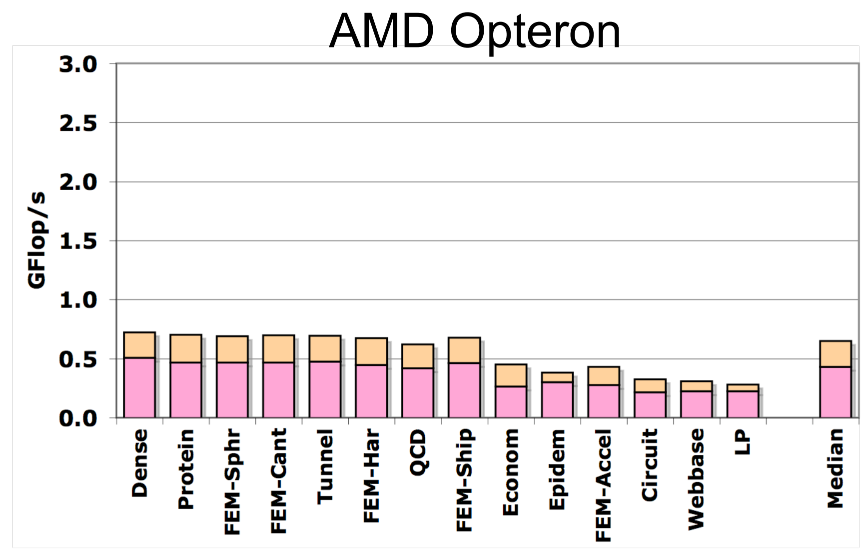
Pthread Implementation for Cache-Based Machines

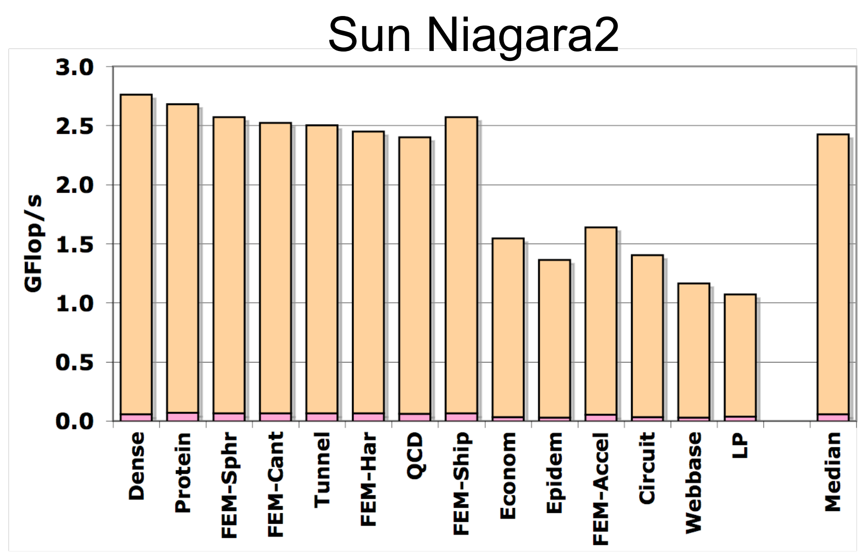
- SPMD, strong scaling
- Optimized for multicore/threading
- Variety of shared memory programming models are acceptable(not just Pthreads)
- More colors = more optimizations = more work

Naïve Parallel Performance

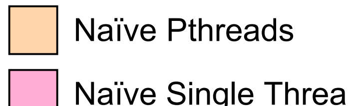


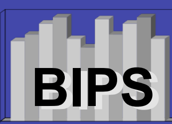




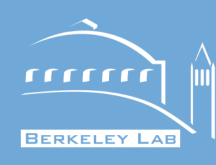


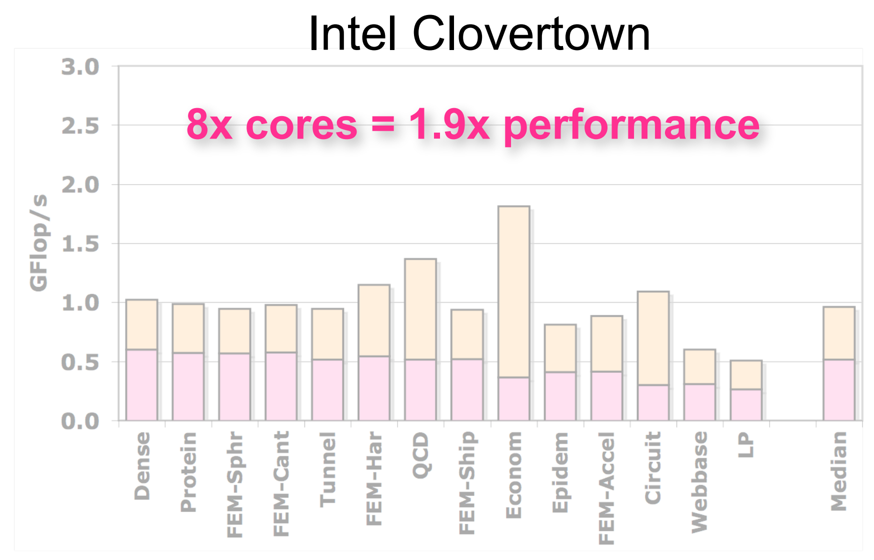
- Simple row parallelization
- Pthreads (didn't have to be)
- 1P Niagara2 > 2x5x 2P x86 machines

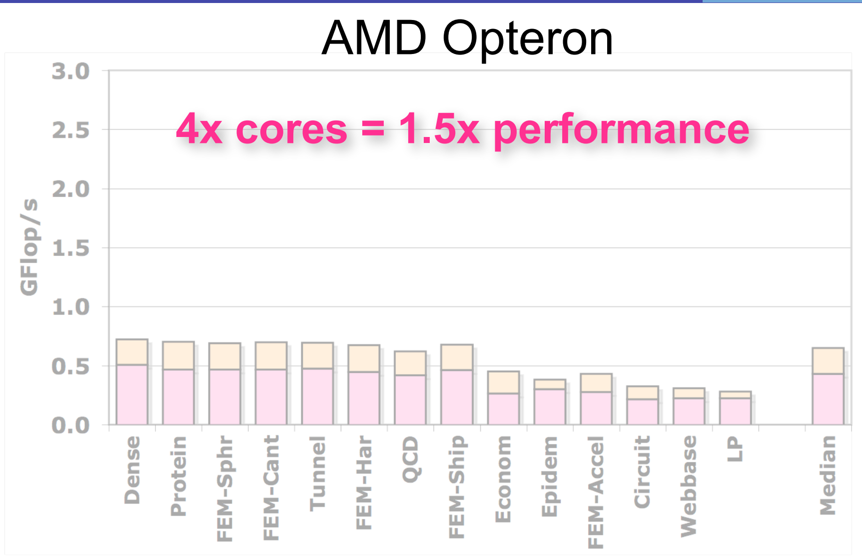


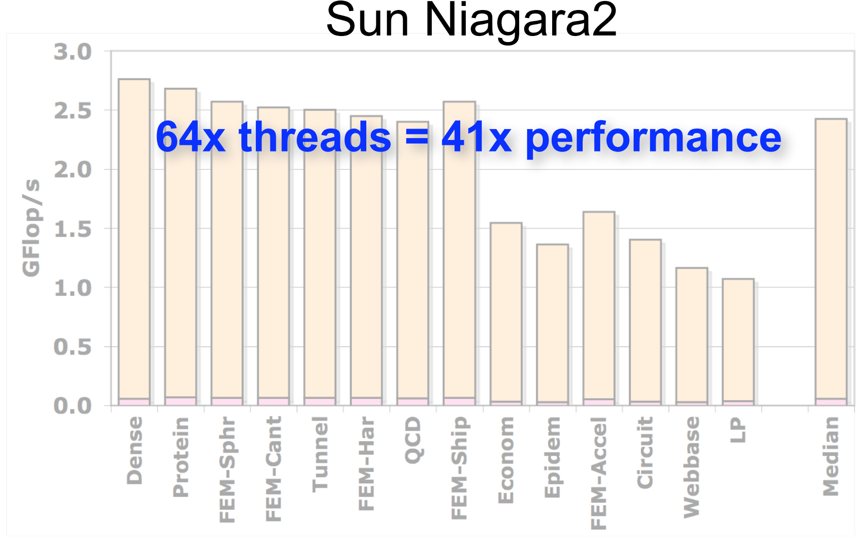


Naïve Parallel Performance

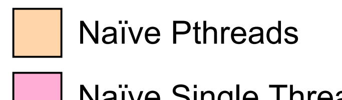


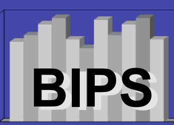




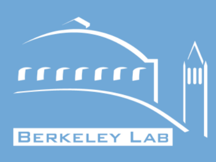


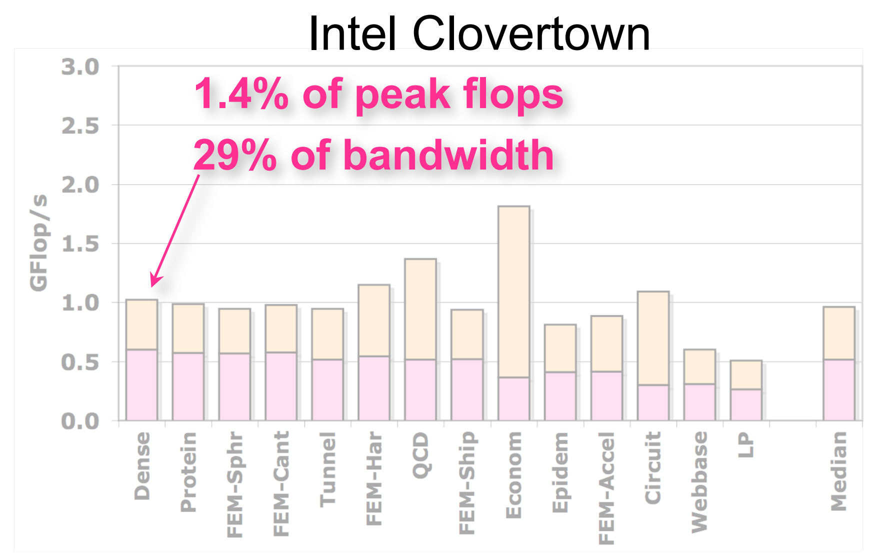
- Simple row parallelization
- Pthreads (didn't have to be)
- 1P Niagara2 > 2x5x 2P x86 machines

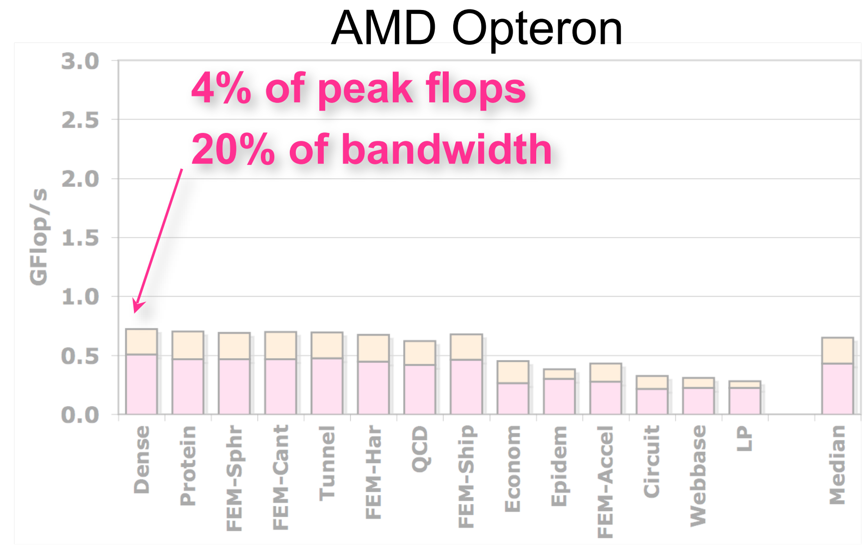


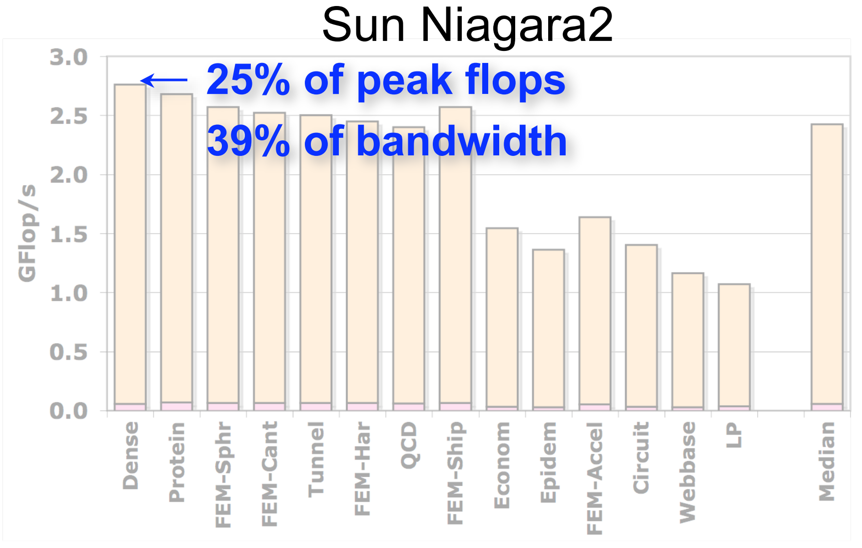


Naïve Parallel Performance

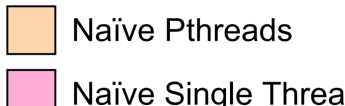




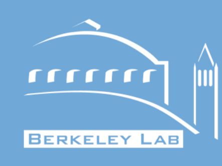




- Simple row parallelization
- Pthreads (didn't have to be)
- 1P Niagara2 > 2x5x 2P x86 machines



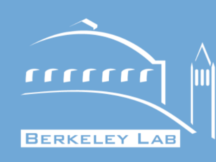
Case for Auto-tuning

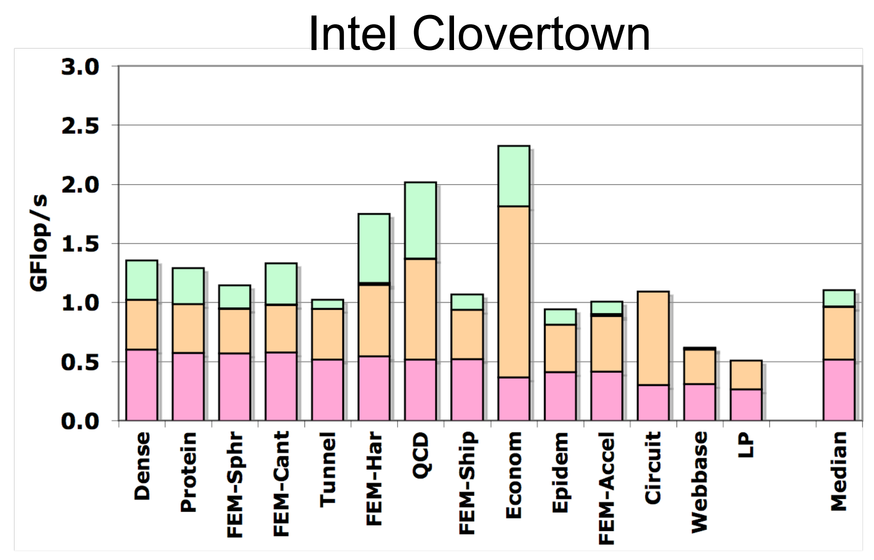


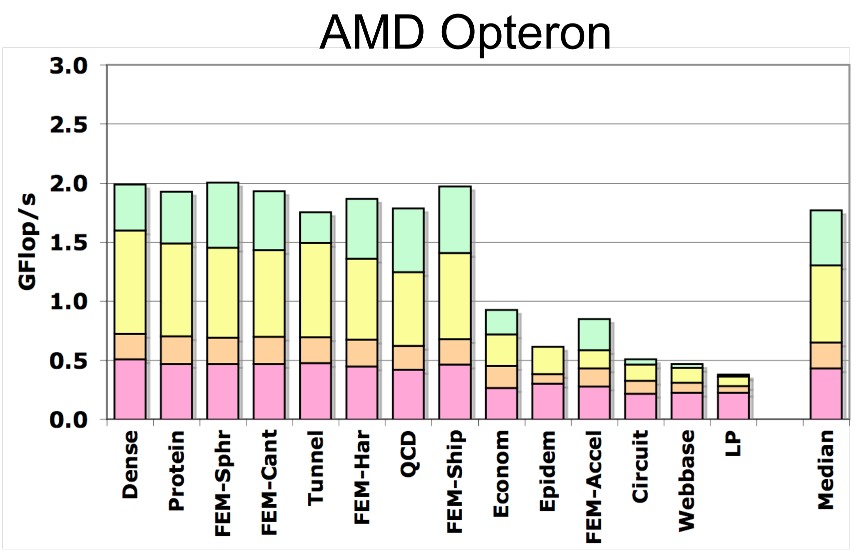
- How do we deliver good performance across all these architectures, across all matrices without exhaustively optimizing every combination

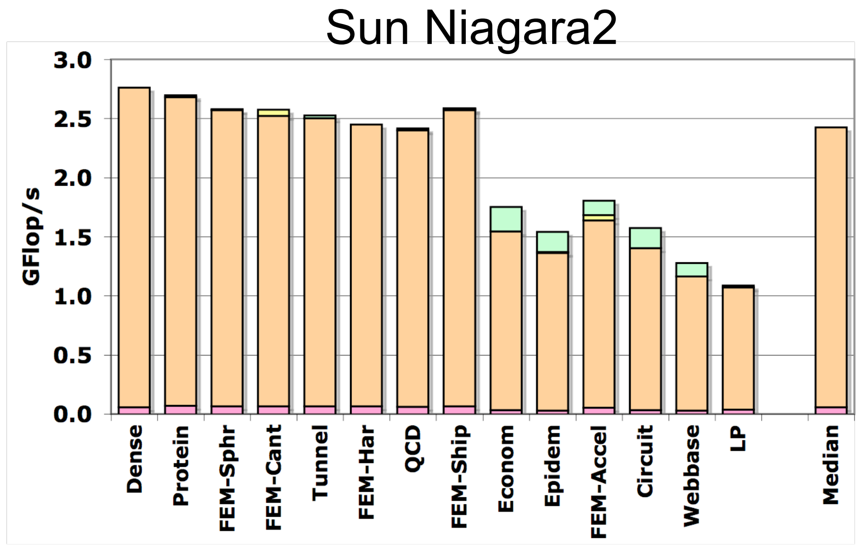
- Auto-tuning (in general)
  - Write a Perl script that generates all possible optimizations
  - Heuristically, or exhaustively searches the optimizations
  - Existing SpMV solution: OSKI (developed at UCB)
- This work:
  - Tuning geared toward multi-core/-threading
  - generates SSE/SIMD intrinsics, prefetching, loop transformations, alternate data structures, etc…
  - "prototype for parallel OSKI"

Performance
(+NUMA & SW Prefetching)





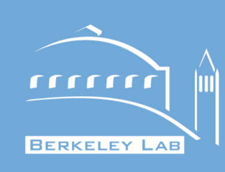




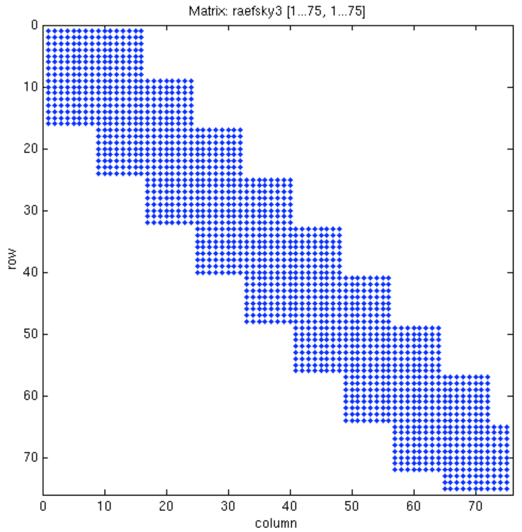
- NUMA aware allocation
- Tag prefetches with the appropriate temporal locality
- Tune for the optimal distance

+Software Prefetching
+NUMA/Affinity
Naïve Pthreads
Naïve Single Thread

Matrix Compression

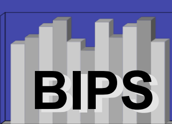


- For memory bound kernels, minimizing memory traffic should maximize performance
- Compress the meta data
  - Exploit structure to eliminate meta data

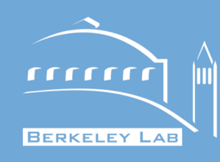


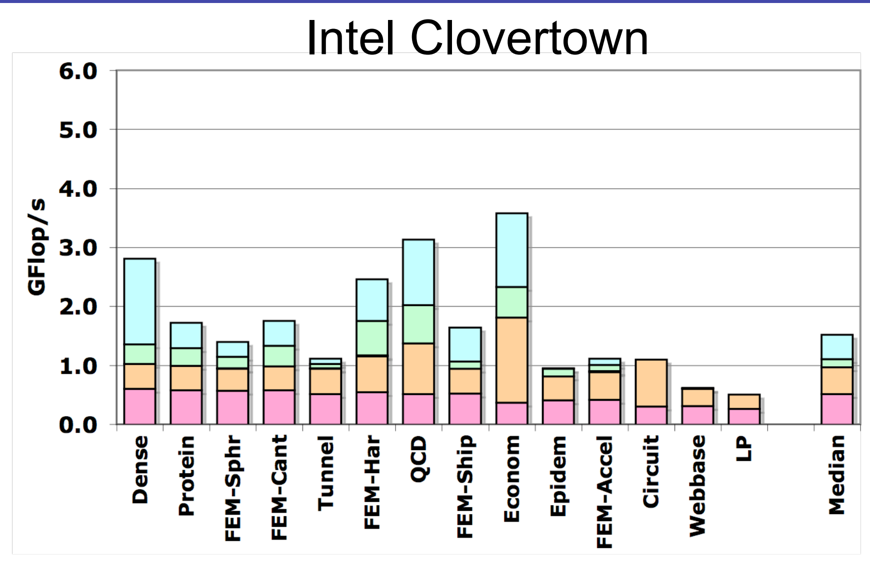
- **Heuristic**: select the compression that minimizes the matrix size:
  - power of 2 register blocking
  - CSR/COO format
  - 16b/32b indices
  - etc…

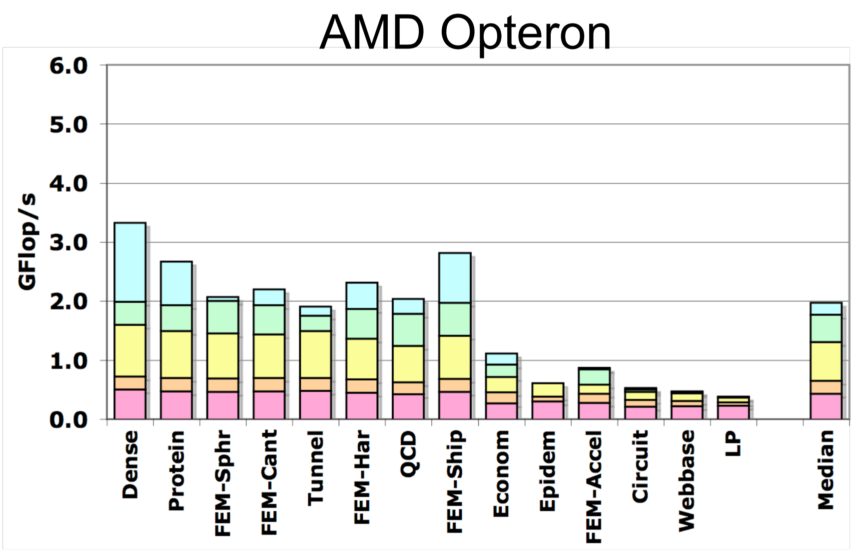
- **Side effect**: matrix may be minimized to the point where it fits entirely in cache

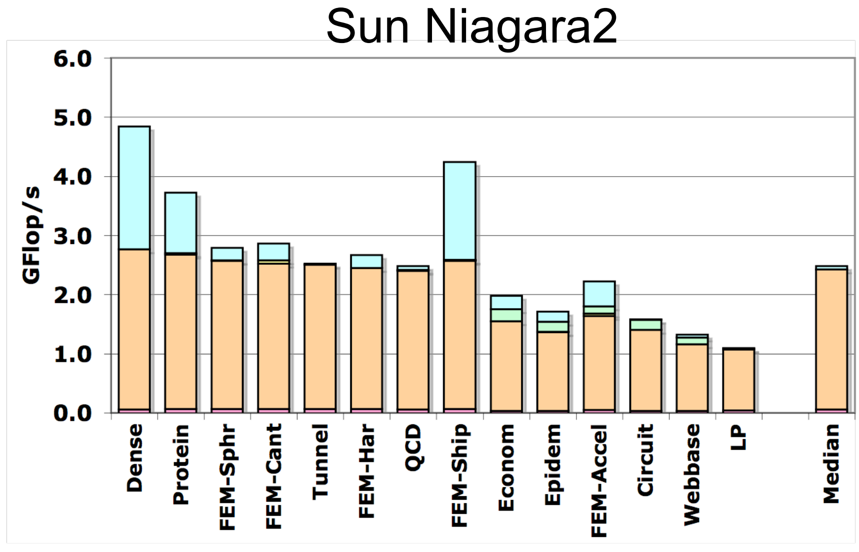


Performance
(+matrix compression)

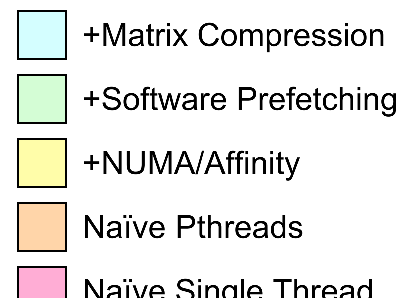




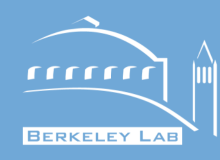


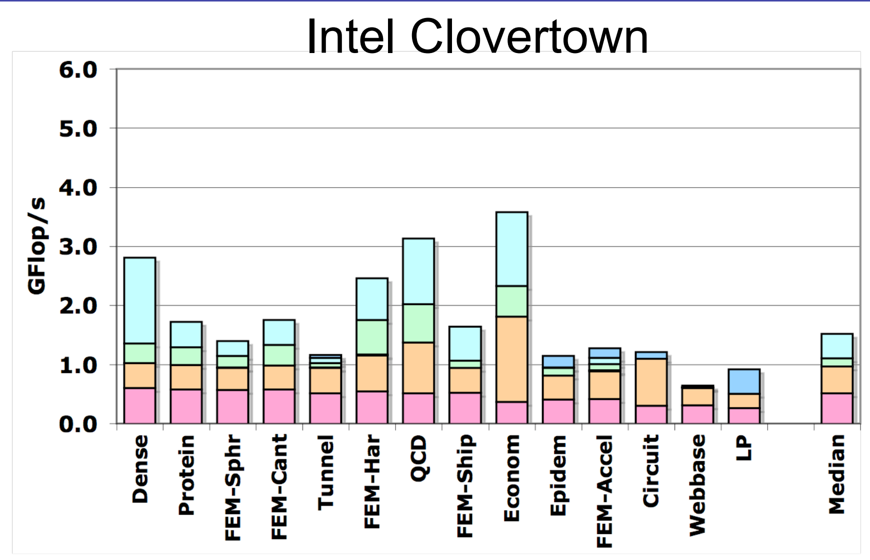


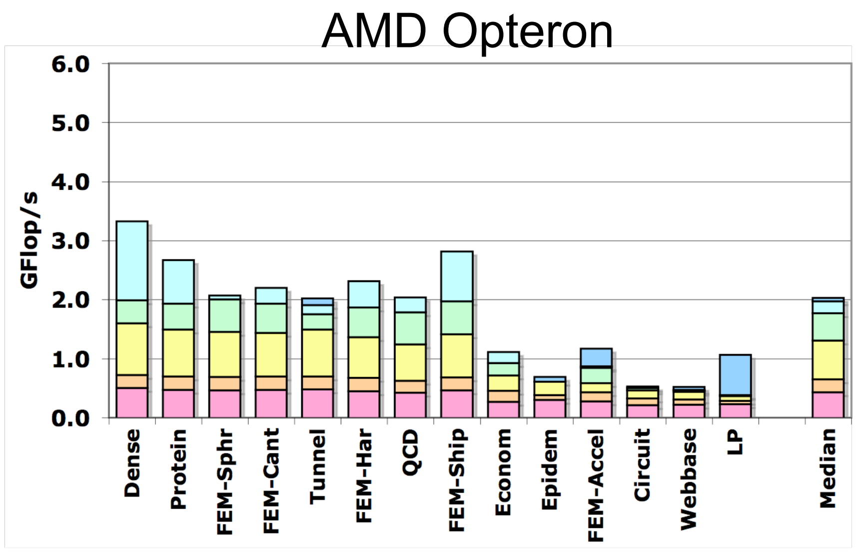
- Very significant on some
- missed the boat on others

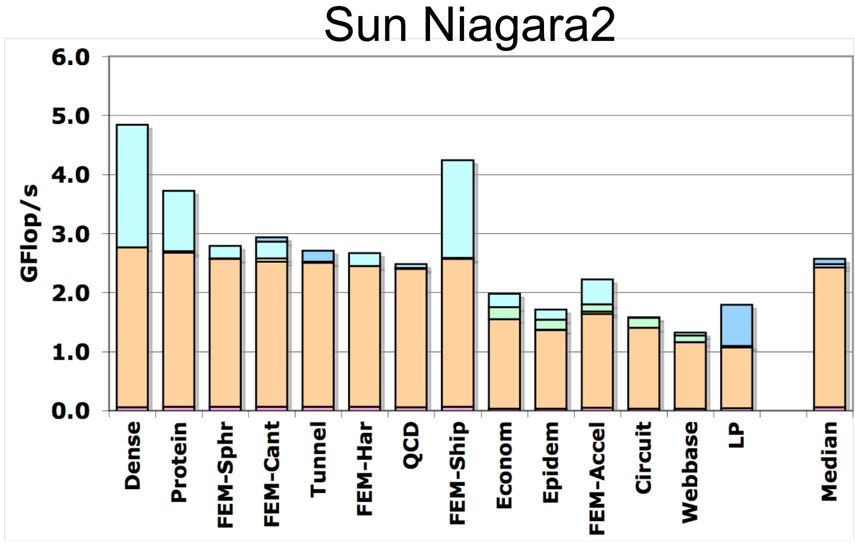


Performance
(+cache & TLB blocking)





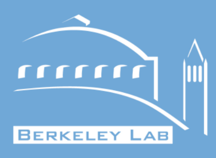


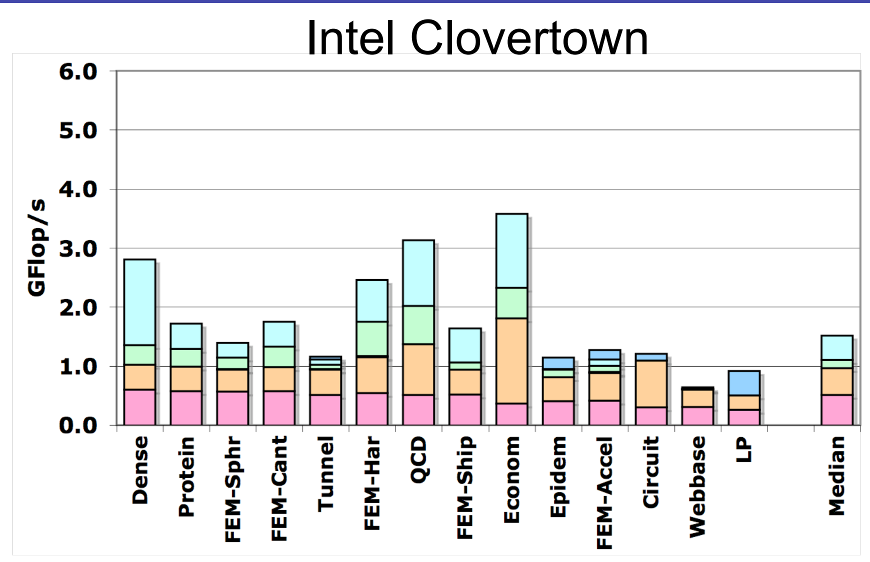


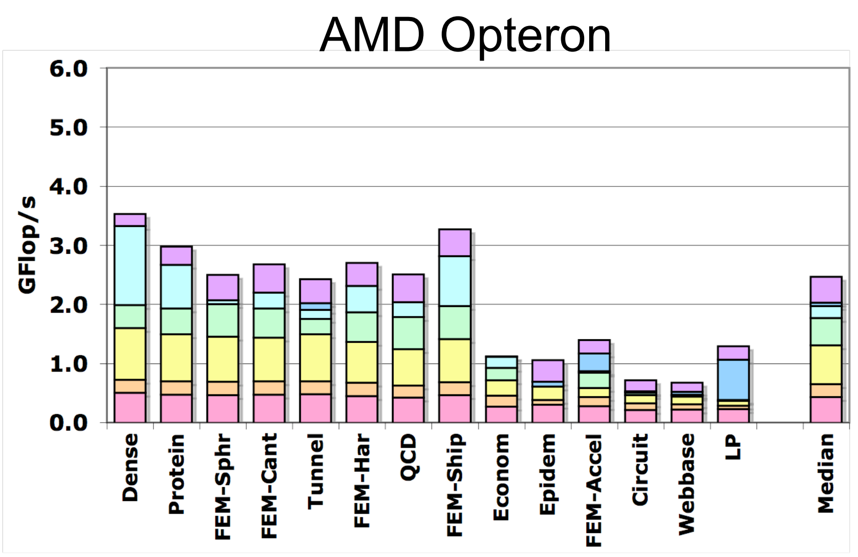
- Cache blocking geared towards sparse accesses

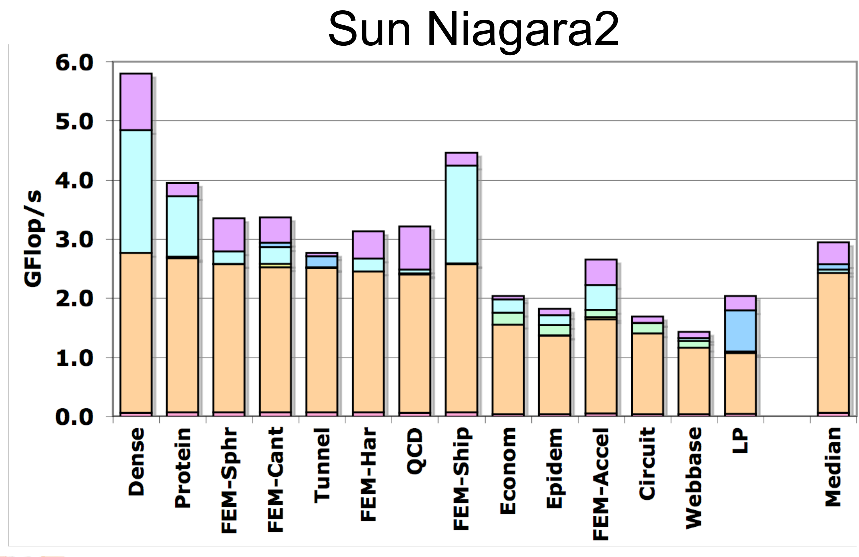
+Cache/TLB Blocking
+Matrix Compression
+Software Prefetching
+NUMA/Affinity
Naïve Pthreads
Naïve Single Thread

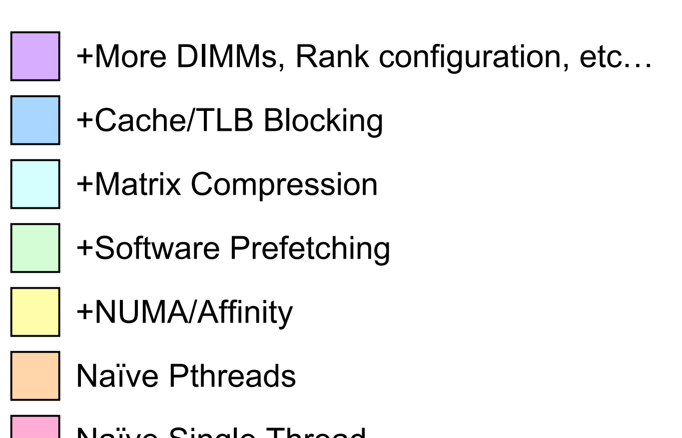
Performance
(more DIMMs,firmware fix …)



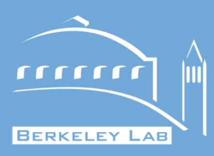


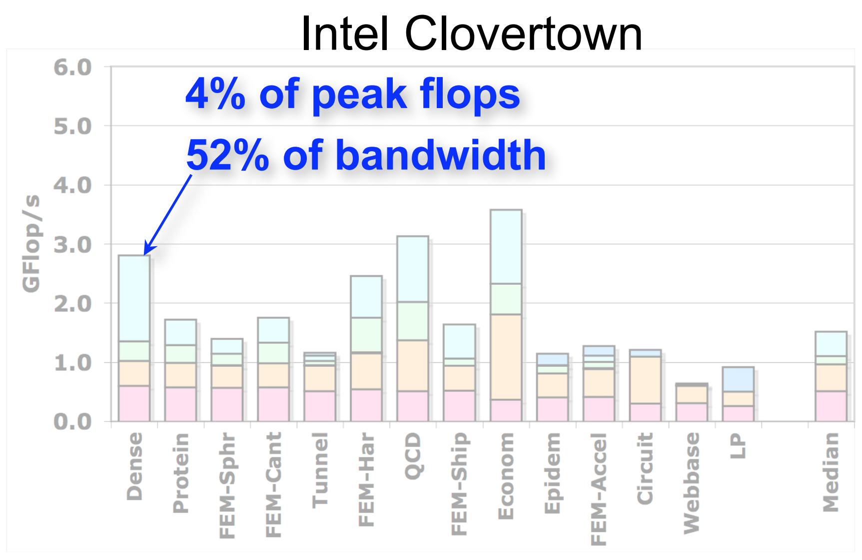


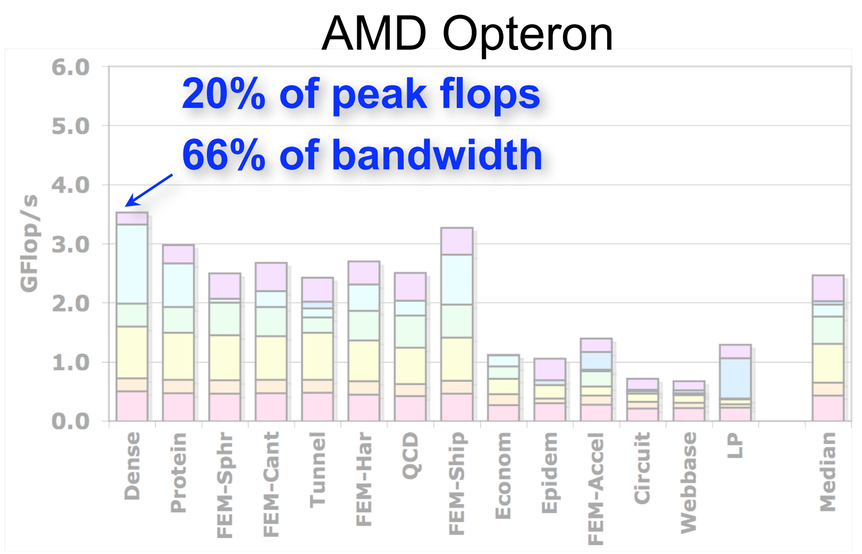


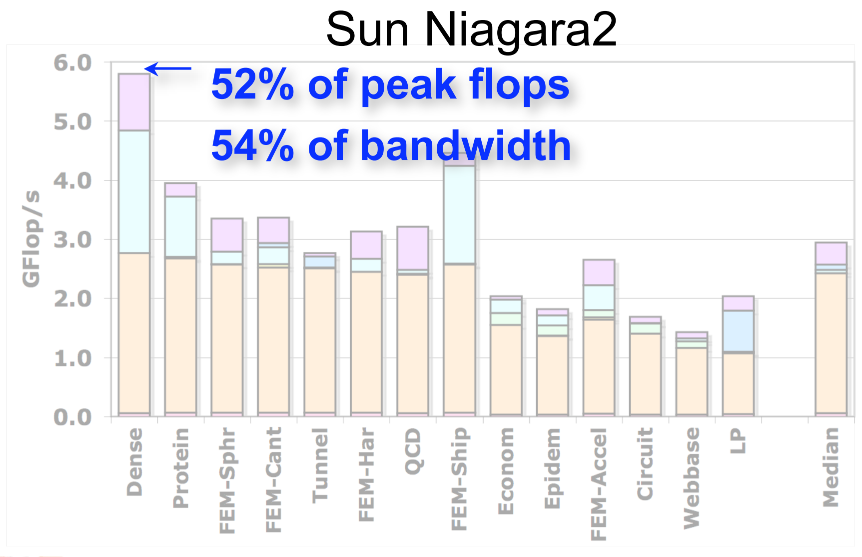


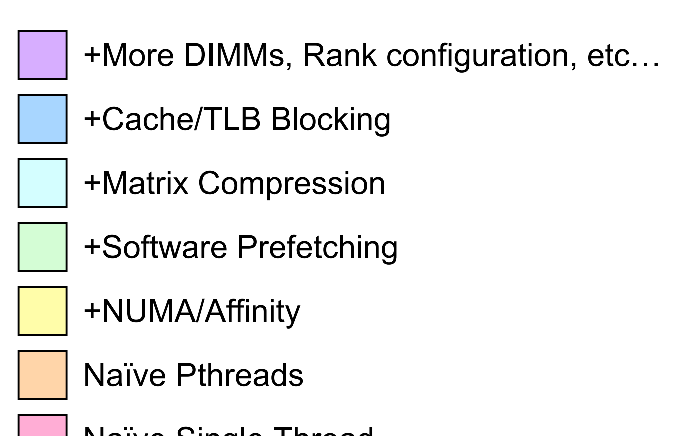
Performance
(more DIMMs,firmware fix …)



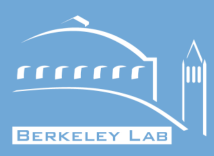




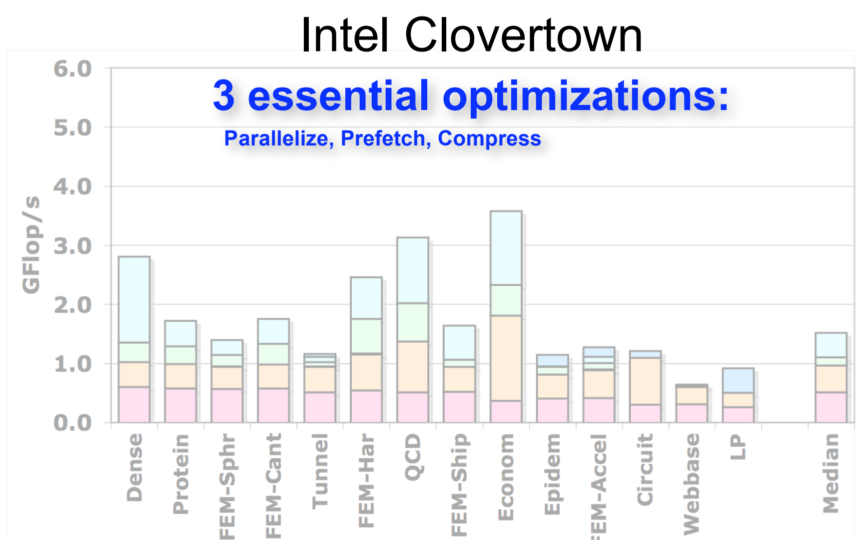




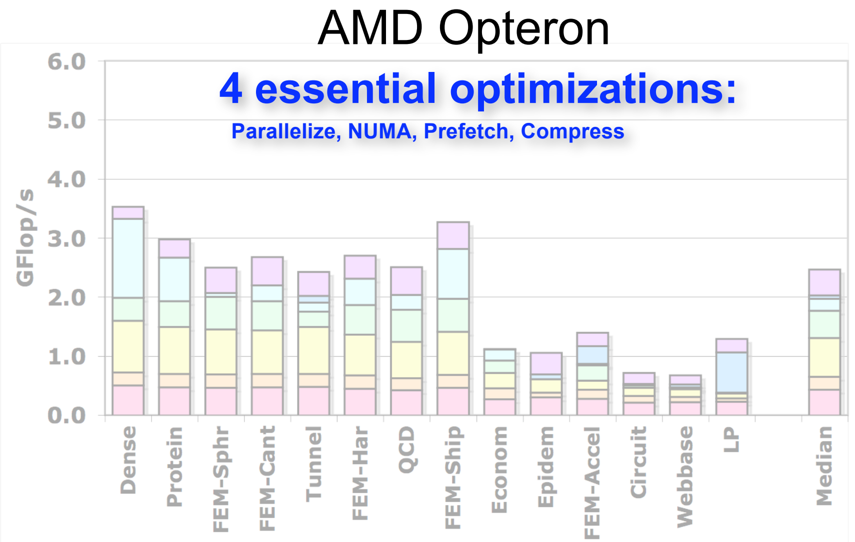
Performance
(more DIMMs,firmware fix …)



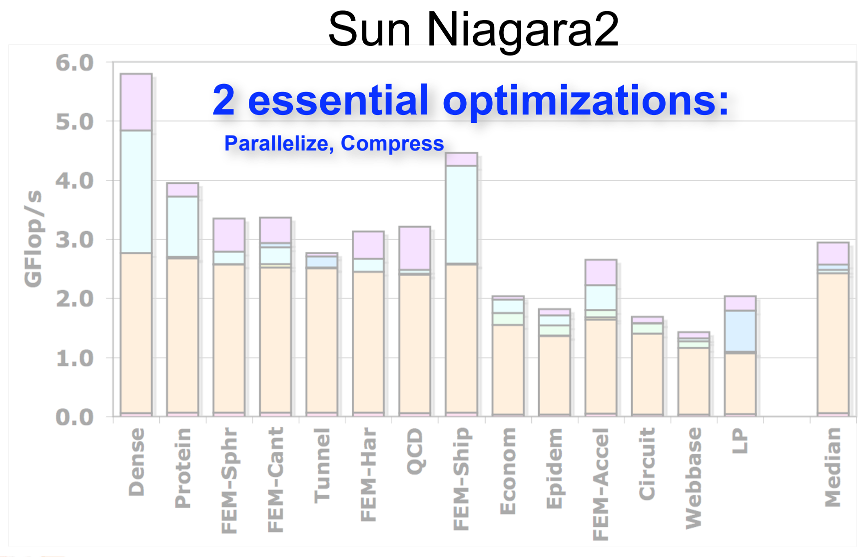
Intel Clovertown

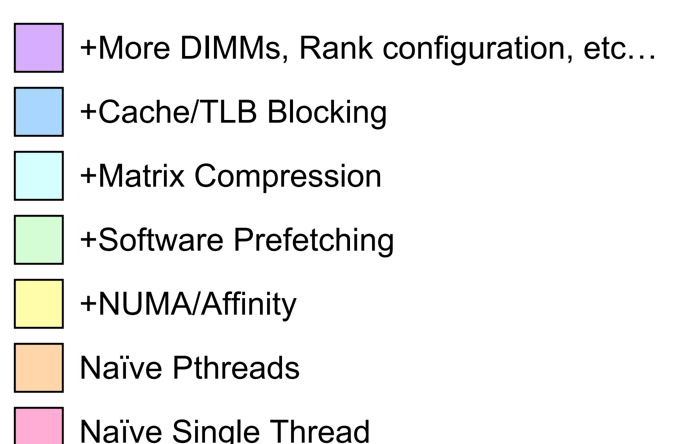


AMD Opteron

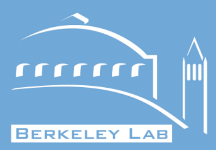


Sun Niagara2





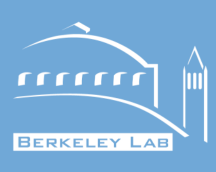
Office of Science
U.S. DEPARTMENT OF ENERGY



Cell Implementation

- Comments
- Performance

Cell Implementation



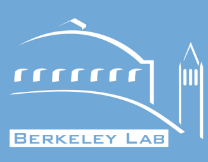
- **No vanilla C implementation (aside from the PPE)**
- In some cases, what were optional optimizations on cache based machines, are requirements for correctness on Cell

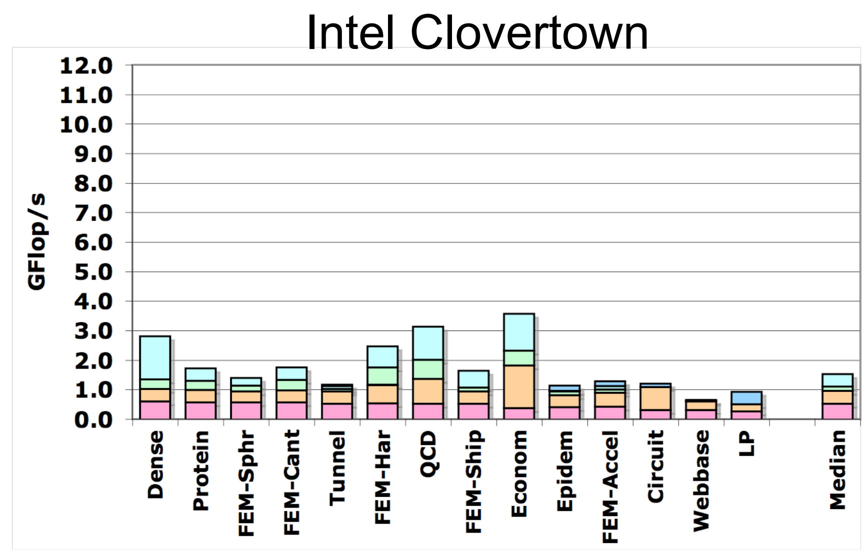
- Even SIMDized double precision is extremely weak
  - Scalar double precision is unbearable
  - Minimum register blocking is 2x1 (SIMDizable)
  - **Can increase memory traffic by 66%**
- Optional Cache blocking is now required local store blocking
  - **Spatial and temporal locality is captured by software when the matrix is optimized**
  - In essence, the high bits of column indices are grouped into DMA lists
- Branchless implementation

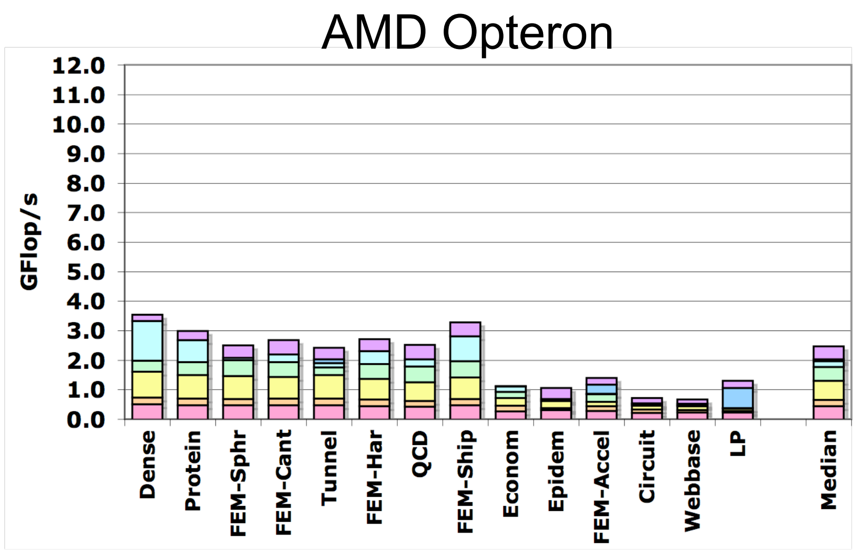
- Despite the following performance numbers, Cell was still handicapped by double precision

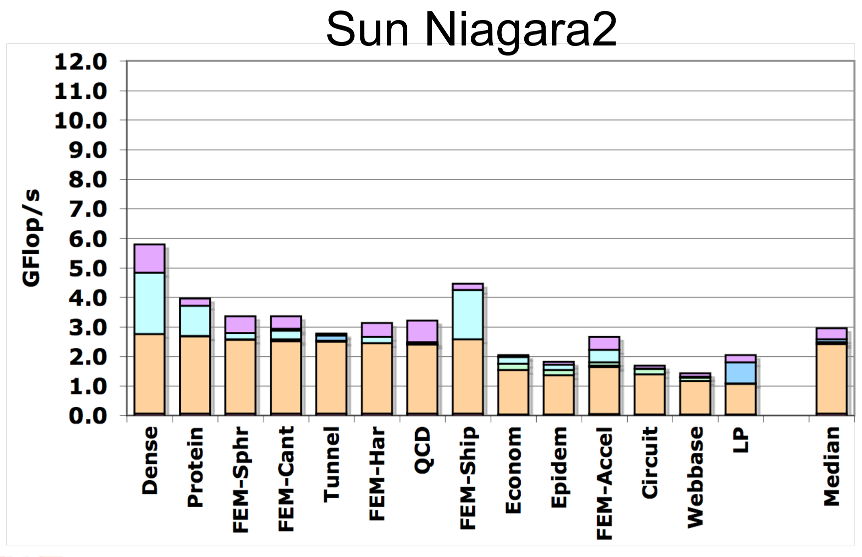
Performance

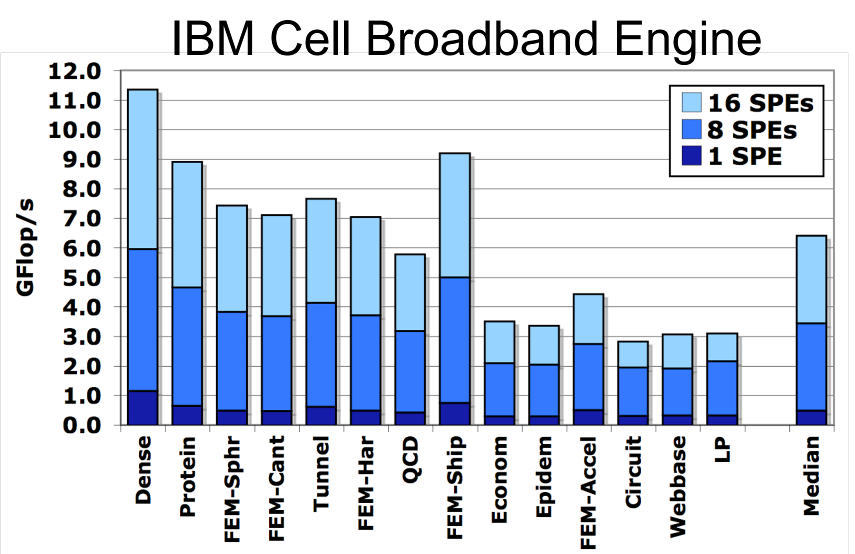
(all)

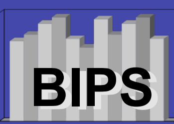




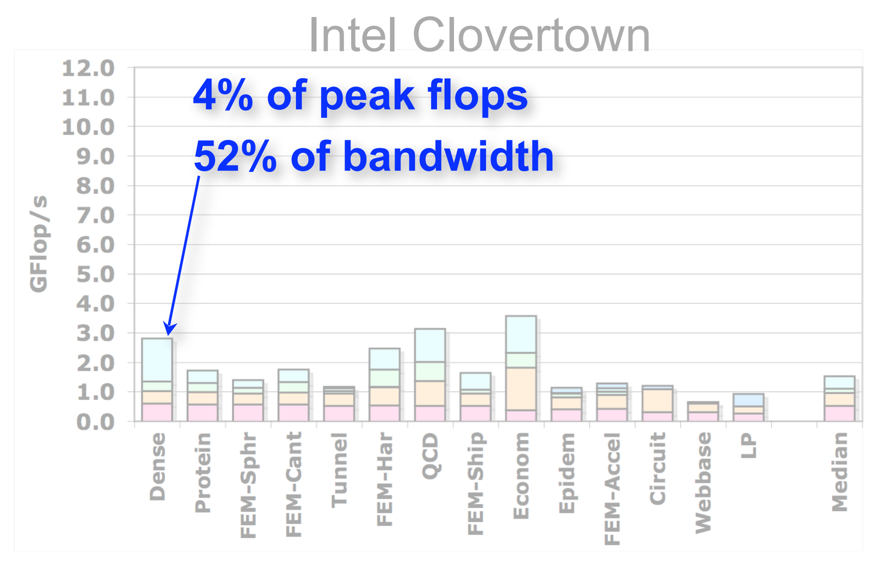


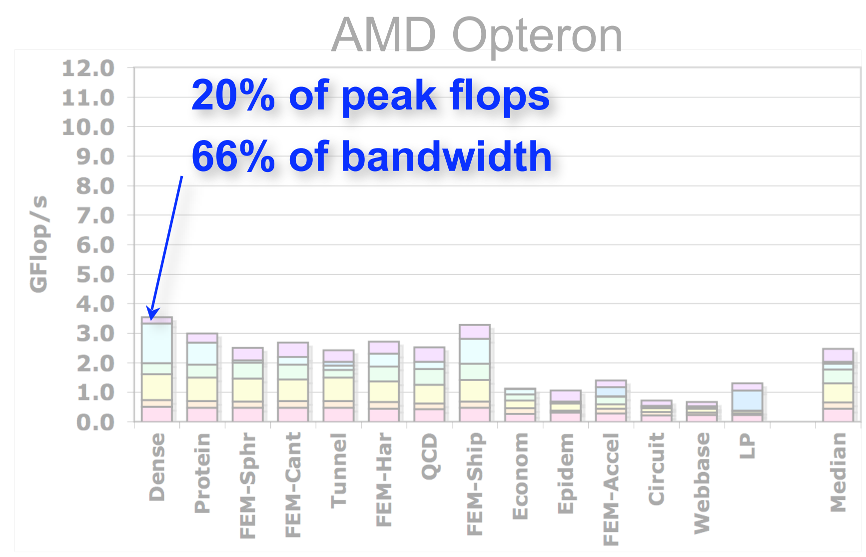


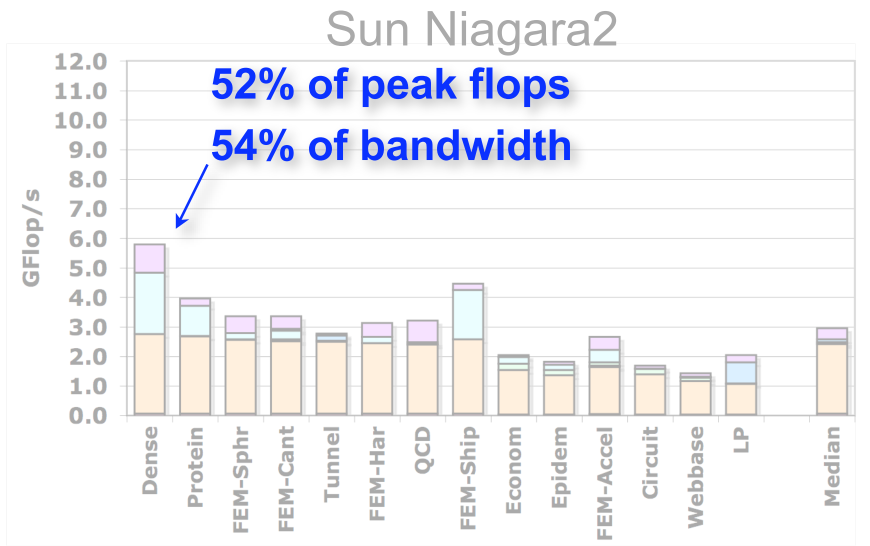


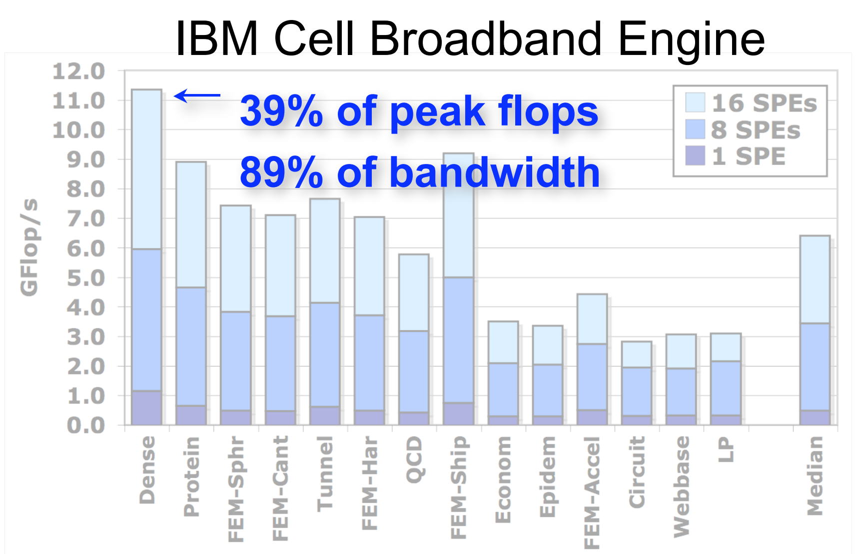


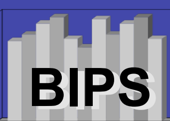
Hardware Efficiency



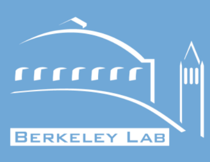


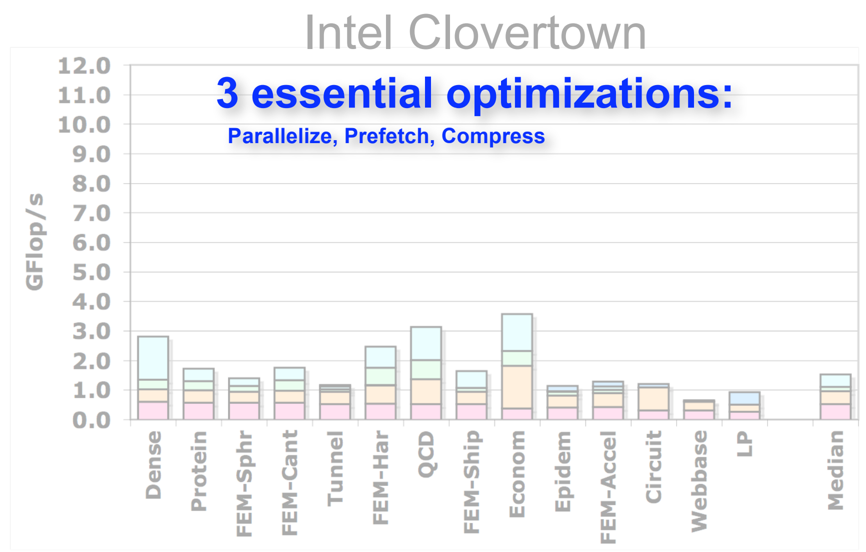


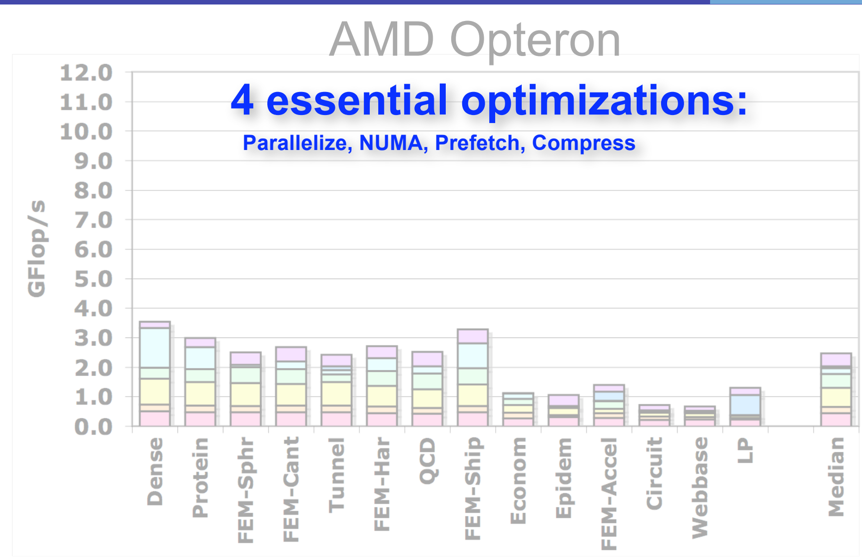


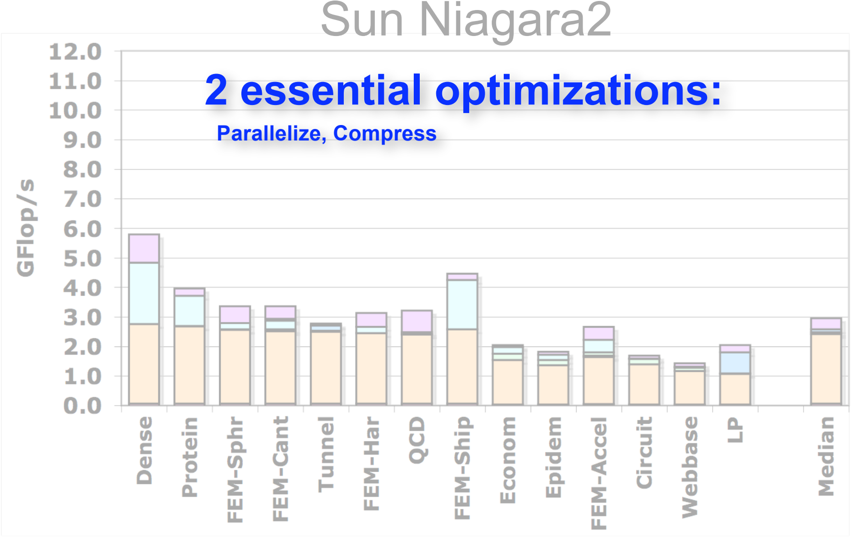


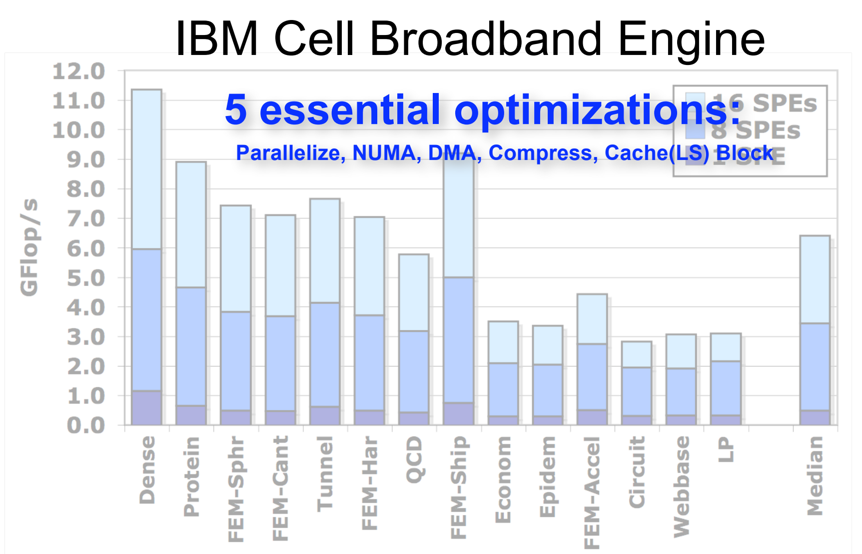
Productivity



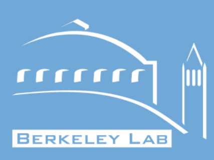


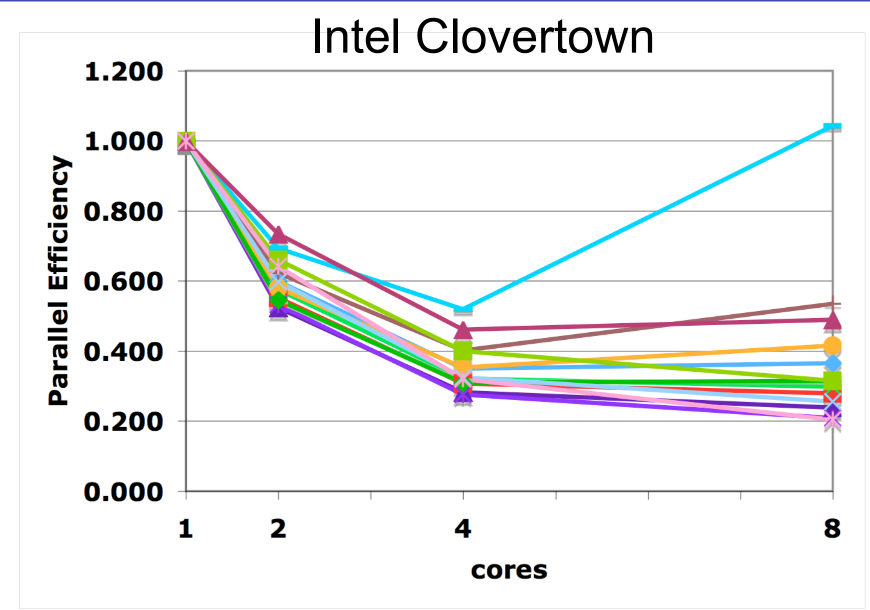


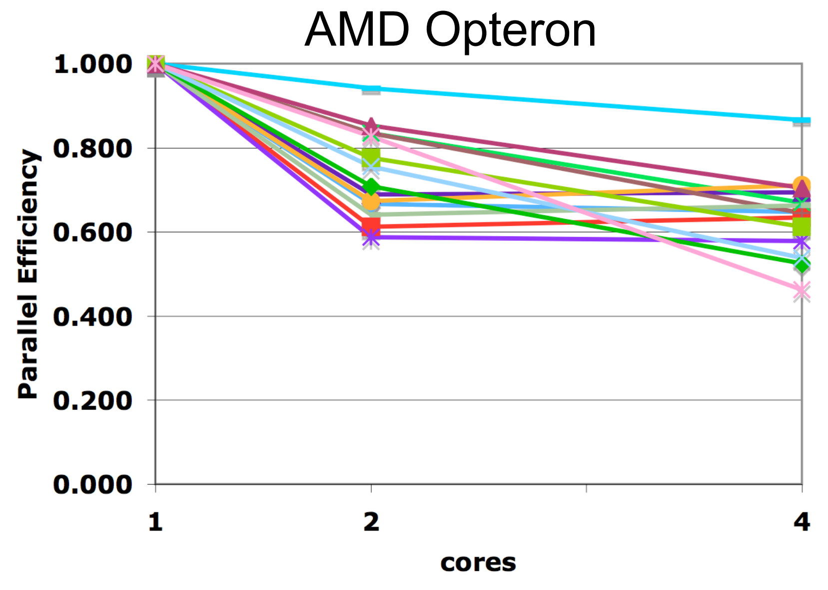


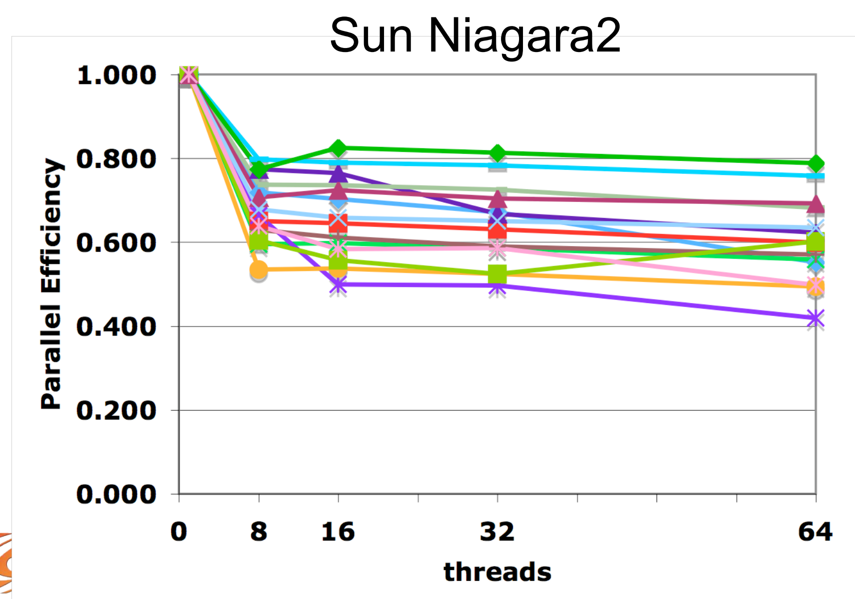


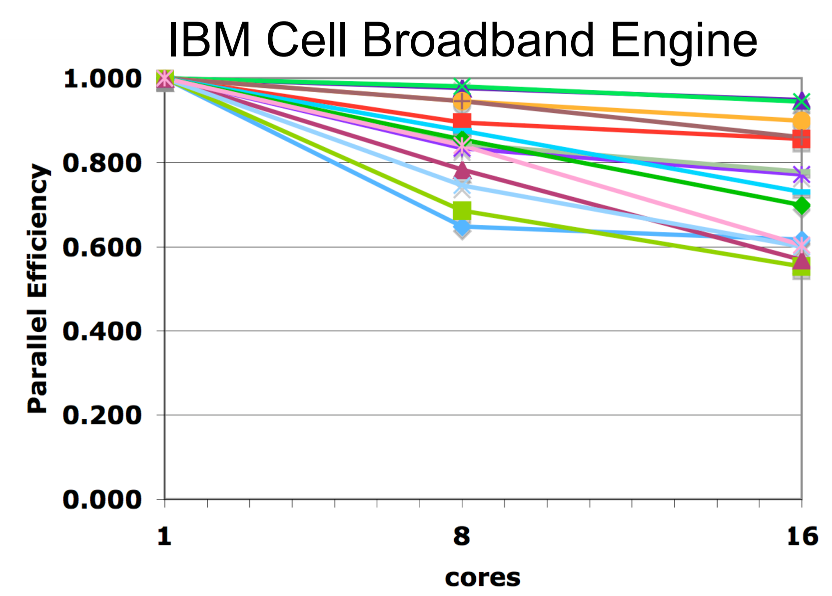
Parallel Efficiency



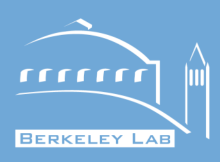


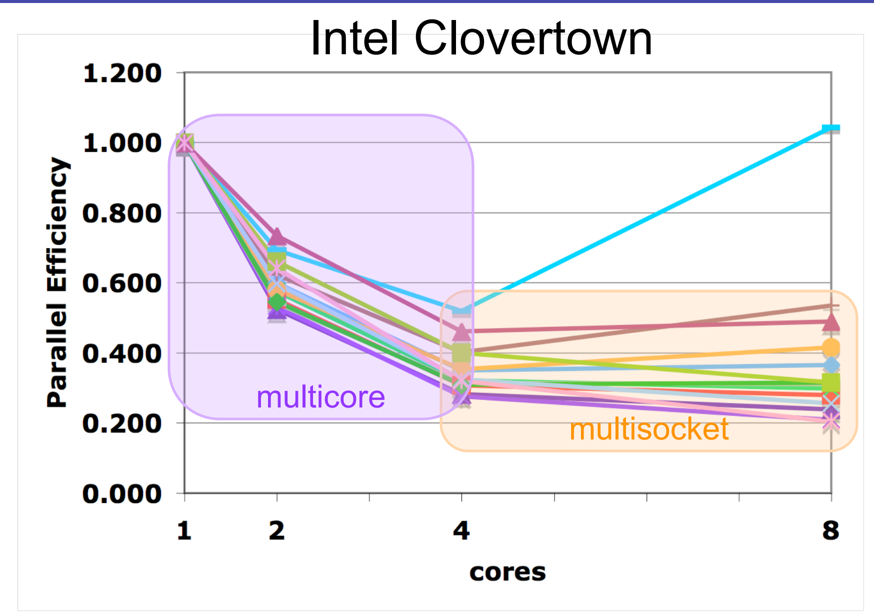


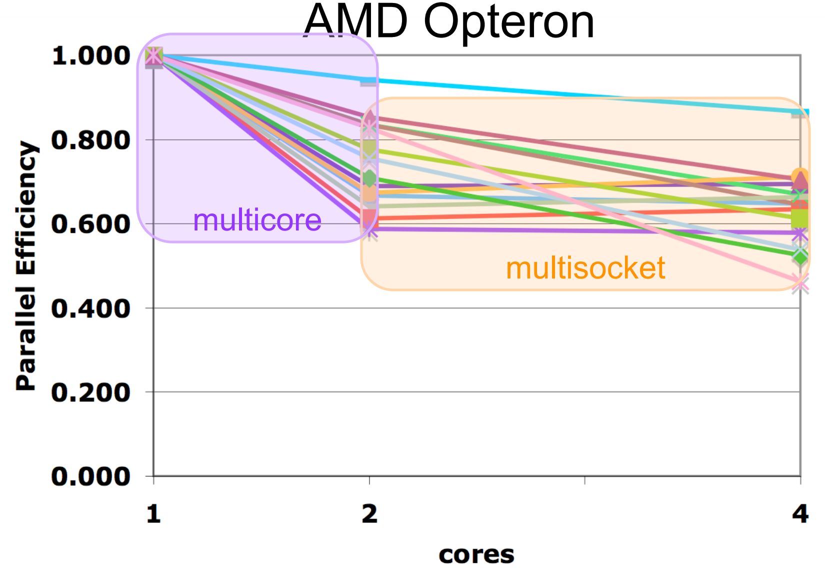


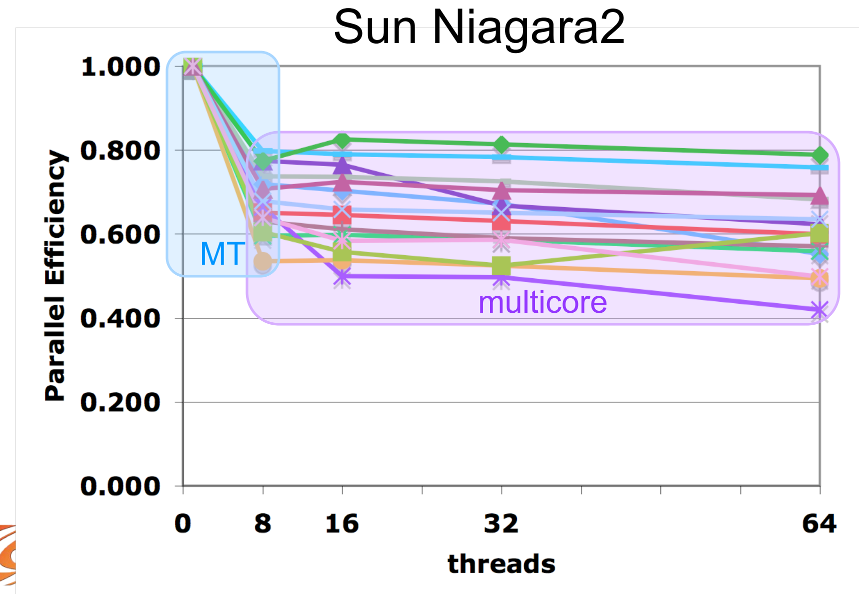


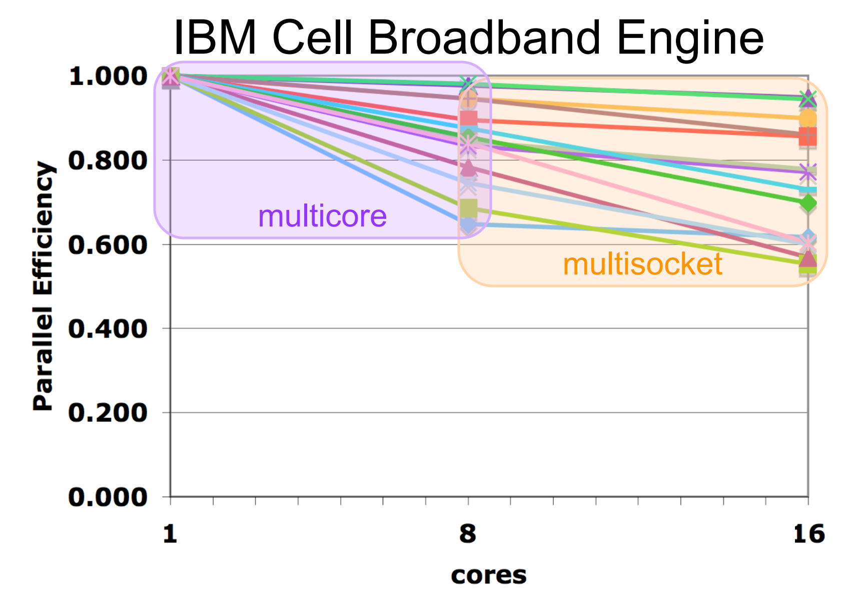
Parallel Efficiency
(breakdown)







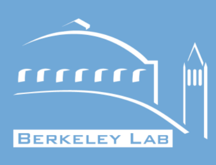




Multicore MPI Implementation

- 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
- = good autotuned shared memory MPI implementation

Intel Clovertown

GFlop/s: 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0

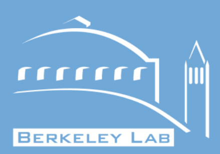
Dense, Protein, FEM-Sphr, FEM-Cant, Tunnel, FEM-Har, QCD, FEM-Ship, Econom, Epidem, FEM-Accel, Circuit, Webbase, LP, Median

AMD Opteron

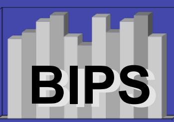
GFlop/s: 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0

Dense, Protein, FEM-Sphr, FEM-Cant, Tunnel, FEM-Har, QCD, FEM-Ship, Econom, Epidem, FEM-Accel, Circuit, Webbase, LP, Median

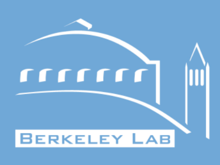
Naïve Single Thread | MPI(autotuned) | Pthreads(autotuned)



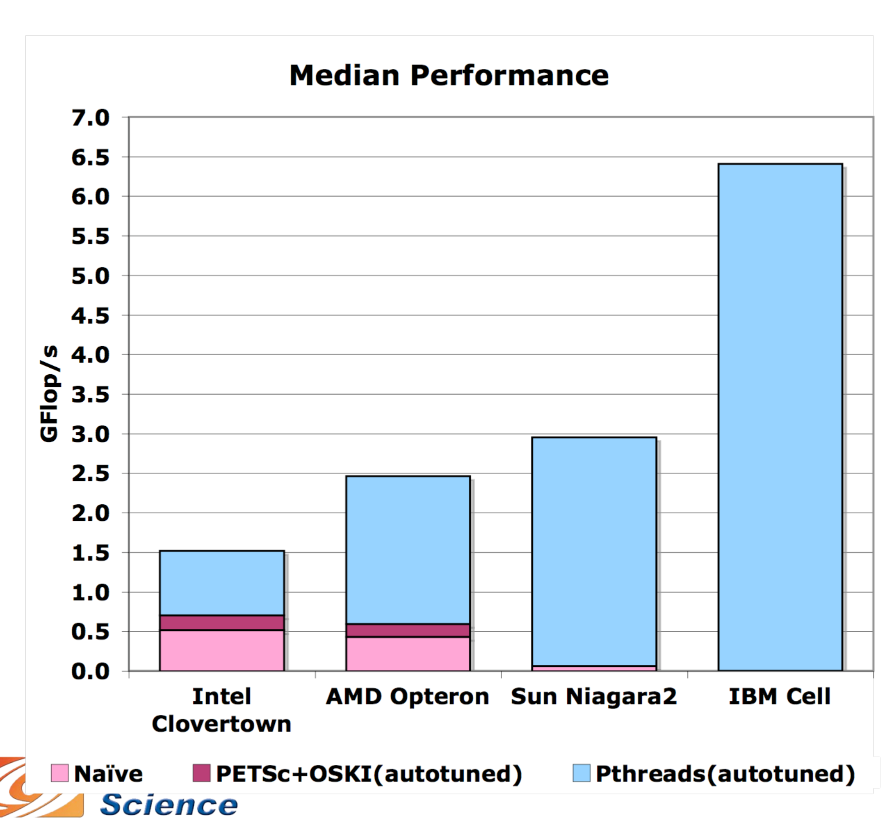
Summary

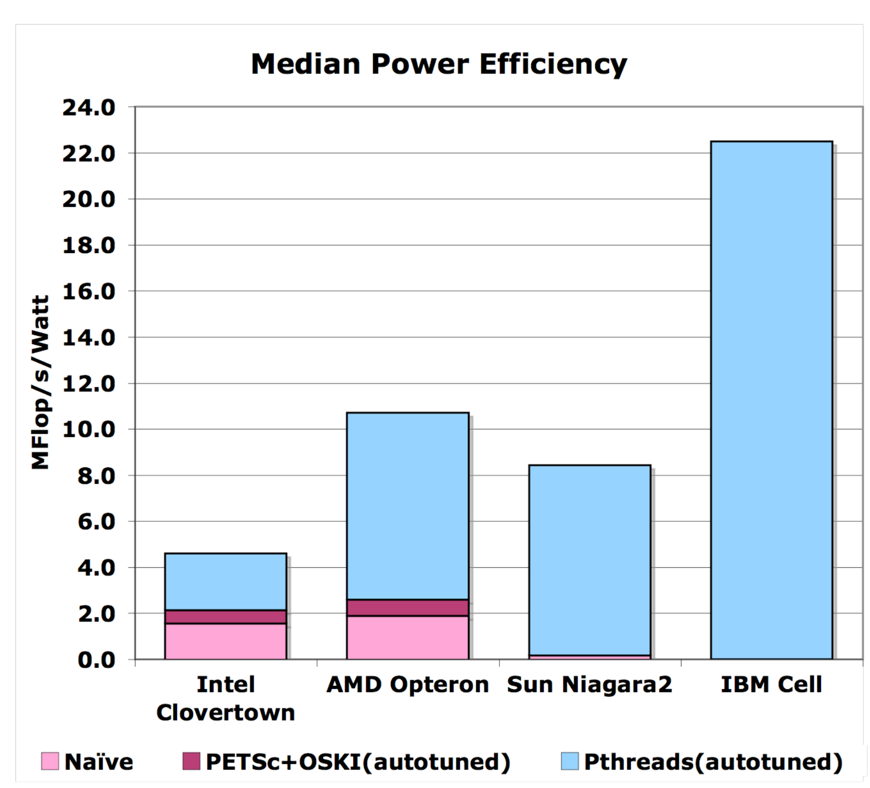


Median Performance & Efficiency

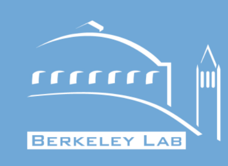


- 1P Niagara2 was consistently better than 2P x86 machines (more potential bandwidth)
- Cell delivered by far the best performance (better utilization)
- Used digital power meter to measure sustained system power
- FBDIMM is power hungry (12W/DIMM)
  - Clovertown(330W)
  - Niagara2 (350W) power





Summary

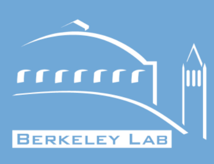


- **Paradoxically**, the most complex/advanced architectures required the most tuning, and delivered the lowest performance.

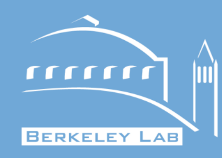
- Niagara2 delivered both very good performance and productivity
- Cell delivered very good performance and efficiency
  - 90% of memory bandwidth (most get ~50%)
  - High power efficiency
  - Easily understood performance
  - Extra traffic = lower performance (future work can address this)

- **Our multicore specific autotuned implementation significantly outperformed an autotuned MPI implementation**
- It exploited:
  - ✓ Matrix compression geared towards multicore, rather than single
  - ✓ NUMA
  - ✓ Prefetching

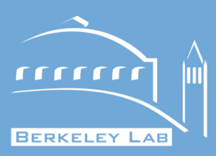
Acknowledgments



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

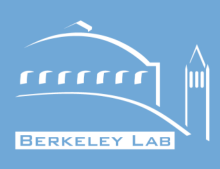


Questions?

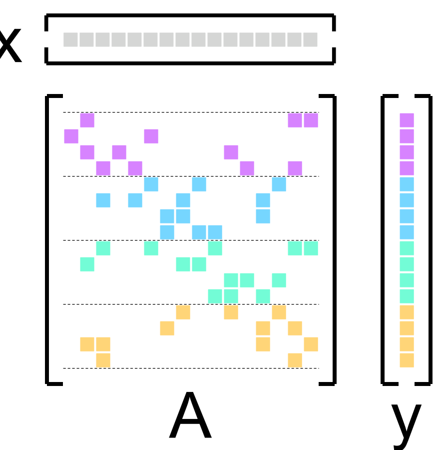


Backup Slides

Parallelization

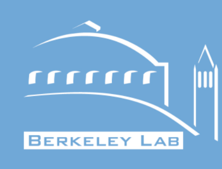


- Matrix partitioned by rows and balanced by the number of nonzeros
- SPMD like approach
- A barrier() is called before and after the SpMV kernel
- Each sub matrix stored separately in CSR
- Load balancing can be challenging

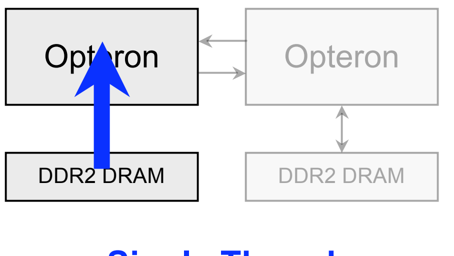


- # of threads explored in powers of 2 (in paper)

Exploiting NUMA, Affinity



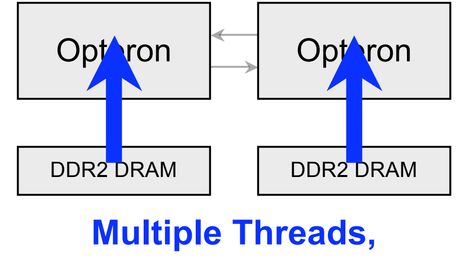
- Bandwidth on the Opteron(and Cell) can vary substantially based on placement of data



Single Thread

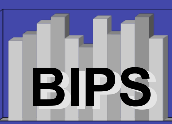
Opteron
Opteron
DDR2 DRAM
DDR2 DRAM

Multiple Threads,
One memory controller

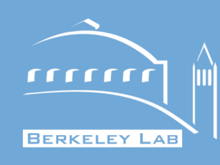


Multiple Threads,
Both memory controllers

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

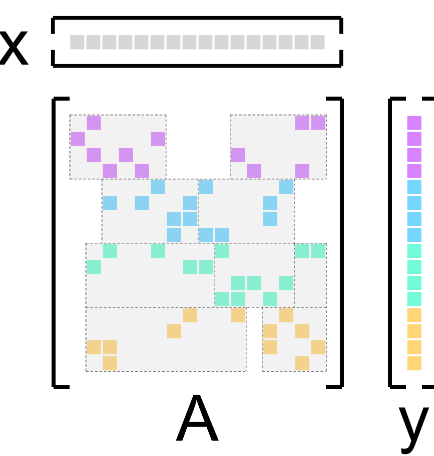


Cache and TLB Blocking



- 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)

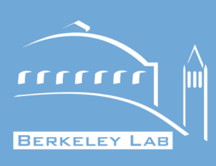
- **Heuristic**: block destination, then keep adding more columns as long as the number of source vector cache lines(or pages) touched is less than the cache(or TLB). Apply all previous optimizations individually to each cache block.



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

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

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