Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms

Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, James Demmel

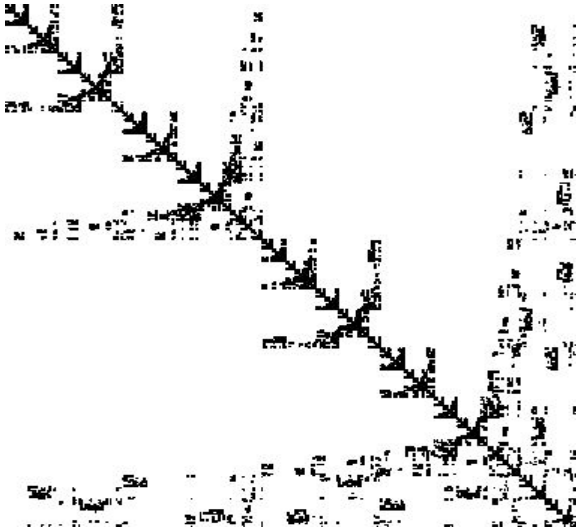
Presented by Bryan Youse

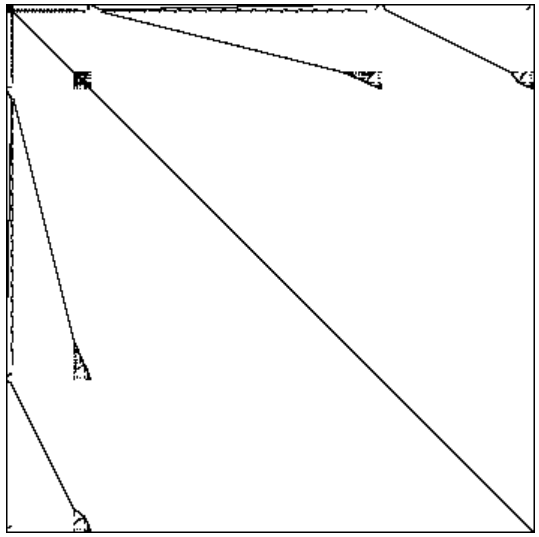
Department of Computer & Information Sciences

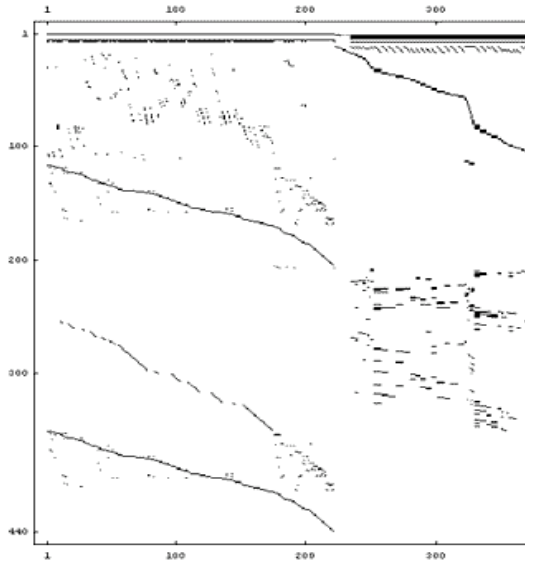
University of Delaware

Sparsity

- Dense Matrix – "common" matrix
- Sparse Matrix – few non-zero entries
- Sparsity – typically expressed as the number o non-zero entries per row







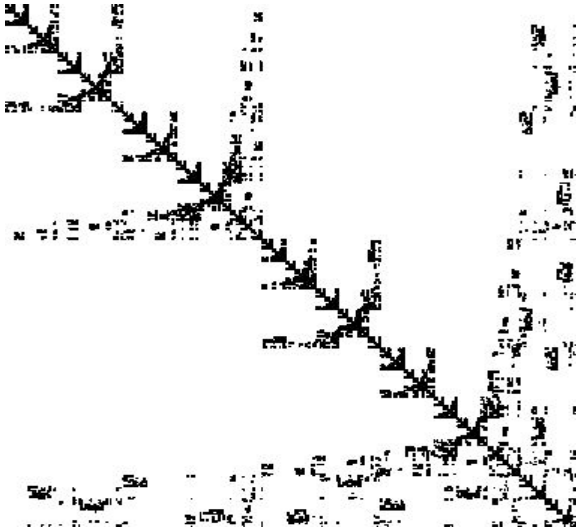
Why care?

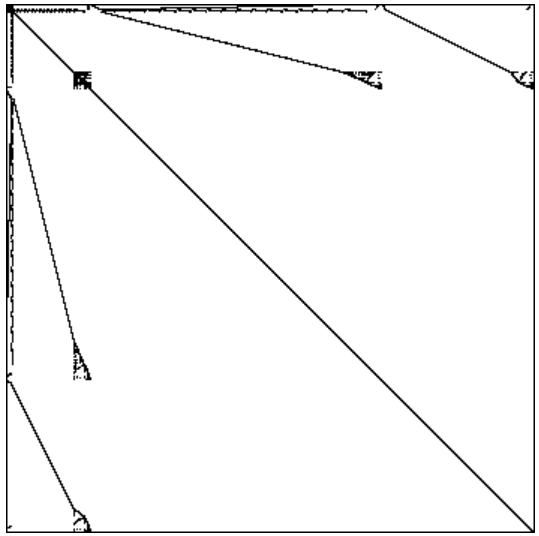
- SpMV – one of the most heavily used kernels in scientific computing
- Other applications
  - Economic modeling
  - Information retrieval
  - Network theory
- Historically (currently!) terrible performance:
  - Current algorithms run at **10% or less** of machine peak performance on a single-core, cache based processor
  - Dense matrix kernels >>> similar Sparse kernels

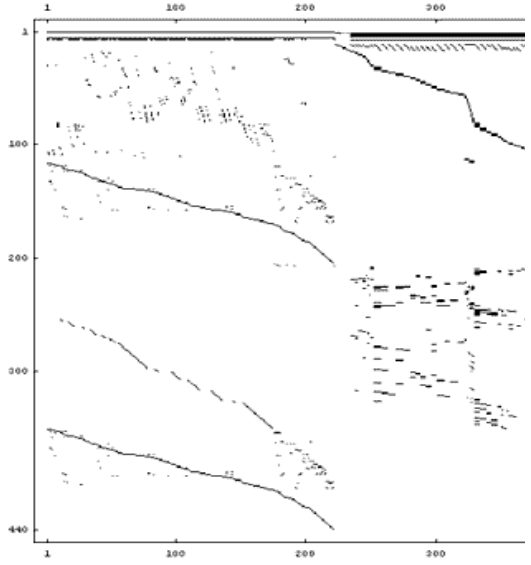
What's the hold up?

Problems with sparse kernels

- data structure woes; needs to both:
  - exploit properties of the sparse matrix
  - fit machine architecture well
- run-time information needed
  - this is the major disadvantage to the dense kernels

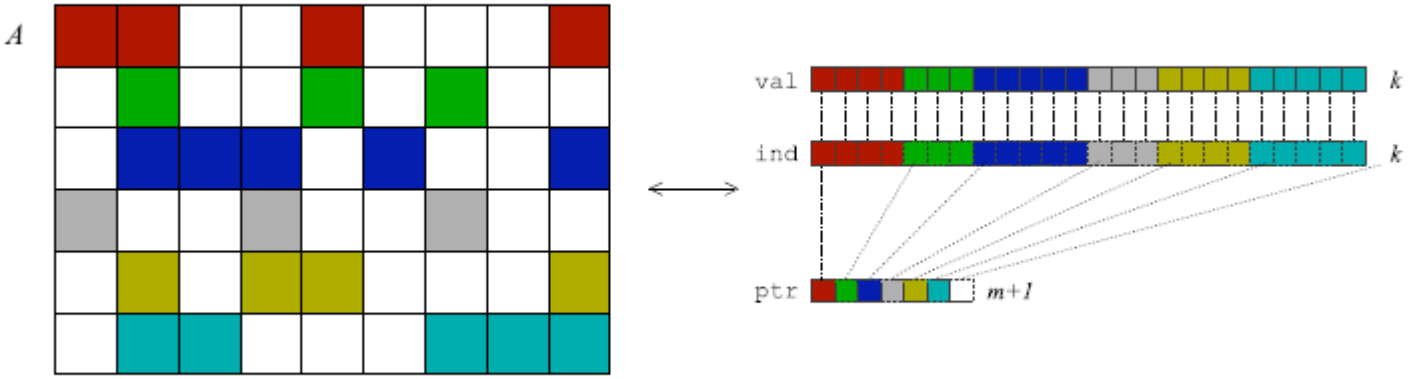






Data Structure

Compressed Sparse Row (CSR) is the standard



- Many optimization tricks can be used to exploit t format

SpMV Optimizations

- 3 categories of optimizations:
  - Low-level code optimizations
  - Data structure optimizations
    - this includes the requisite code changes
  - Parallelization optimizations
- Note that the first two largely affect single core performance
- Goal: As much auto-tuning as possible

Optimizations: Blocking

- Thread Blocking
  - Thread-level parallelism – split matrix up (by rows, cols, or blocks)
- Cache Blocking
  - Problem: Very large matrices cannot fit the entire source or answer vectors in cache
  - Solution: Split matrix up into cache-sized tiles (~1K x 1K is common)

Optimizations: Blocking (2)

- TLB Blocking
  - TLB misses can vary by an order of magnitude depending on the blocking strategy
- Register Blocking
  - Group adjacent non-zeros into rectangular tiles
- Key point to take from blocking: SpMV is a *memory-bound* application; reducing memory footprint is more important than anything else

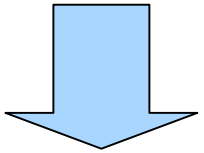
More Optimizations

- Index Size Selection
  - 16b integers to reduce memory traffic
- SIMDization
- Software Prefetching
  - Get the data we know we'll need soon in cache
- Architecture Specific Kernels
  - through auto-tuning

Last Optimization, I Swear

- Loop Optimizations, of course!
  - CSR format enables the core kernel loop to go fron

Old & Busted: Nested loop with two loop variables



New Hotness: Remove inner loop variable

Code Optimization	x86	N2	Cell
SIMDization	✓	N/A	✓
Software Pipelining			✓
Branchless	✓[10]		✓
Pointer Arithmetic		✓[10]	
PF/DMA[1] Values & Indices	✓	✓	✓
PF/DMA[1] Pointers/Vectors			✓
inter-SpMV data structure caching	✓	✓	

Data Structure Optimization	x86	N2	Cell
BCSR	✓	✓	
BCOO	✓	✓	✓
16-bit indices	✓	✓	✓
32-bit indices	✓	✓	N/A
Register Blocking	✓	✓	✓[9]
Cache Blocking	✓[2]	✓[2]	✓[3]
TLB Blocking	✓	✓	

Parallelization Optimizati	x86	N2
Row Threading	✓[4]	✓[4]
Process Affinity	✓[6]	✓[8]
Memory Affinity	✓[6]	N/A

What about the "emerging multicore platforms" part?

- Several leading multicore platforms were tested

Name	Clovertwn	Opteron	Cell	Niagara 2
Chips*Cores	2*4 = 8	2*2 = 4	1*8 = 8	1*8 = 8
Architecture	4-/3-issue, SSE3, OOO, caches		2-VLIW, SIMD,RAM	1-issue, MT,cache
Clock Rate	2.3 GHz	2.2 GHz	3.2 GHz	1.4 GHz
Peak MemBW	21 GB/s	21 GB/s	26 GB/s	41 GB/s
Peak GFLOPS	74.6 GF	17.6 GF	14.6 GF	11.2 GF

Testing Suite

spyplot	Name	Dimensions	Nonzeros (nnz/row)	Description
	Dense	2K x 2K	4.0M (2K)	Dense matrix in sparse format
	Protein	36K x 36K	4.3M (119)	Protein data bank 1HYS
	FEM / Spheres	83K x 83K	6.0M (72)	FEM concentric spheres
	FEM / Cantilever	62K x 62K	4.0M (65)	FEM cantilever
	Wind Tunnel	218K x 218K	11.6M (53)	Pressurized wind tunnel
	FEM / Harbor	47K x 47K	2.37M (50)	3D CFD of Charleston harbor
	QCD	49K x 49K	1.90M (39)	Quark propagators (QCD/LGT)
	FEM/Ship	141K x 141K	3.98M (28)	FEM Ship section/detail
	Economics	207K x 207K	1.27M (6)	Macroeconomic model
	Epidemiology	526K x 526K	2.1M (4)	2D Markov model of epidemic
	FEM / Accelerator	121K x 121K	2.62M (22)	Accelerator cavity design
	Circuit	171K x 171K	959K (6)	Motorola circuit simulation
	webbase	1M x 1M	3.1M (3)	Web connectivity matrix
	LP	4K x 1.1M	11.3M (2825)	Railways set cover Constraint matrix

Actually, the dense matrix provides the performance upper bound:

- SpMV is limited by memory throughput
- Dense case supports arbitrary register blocks (no added zeros)
- Loops are long running -> more CPU time vs. Memory fetch time

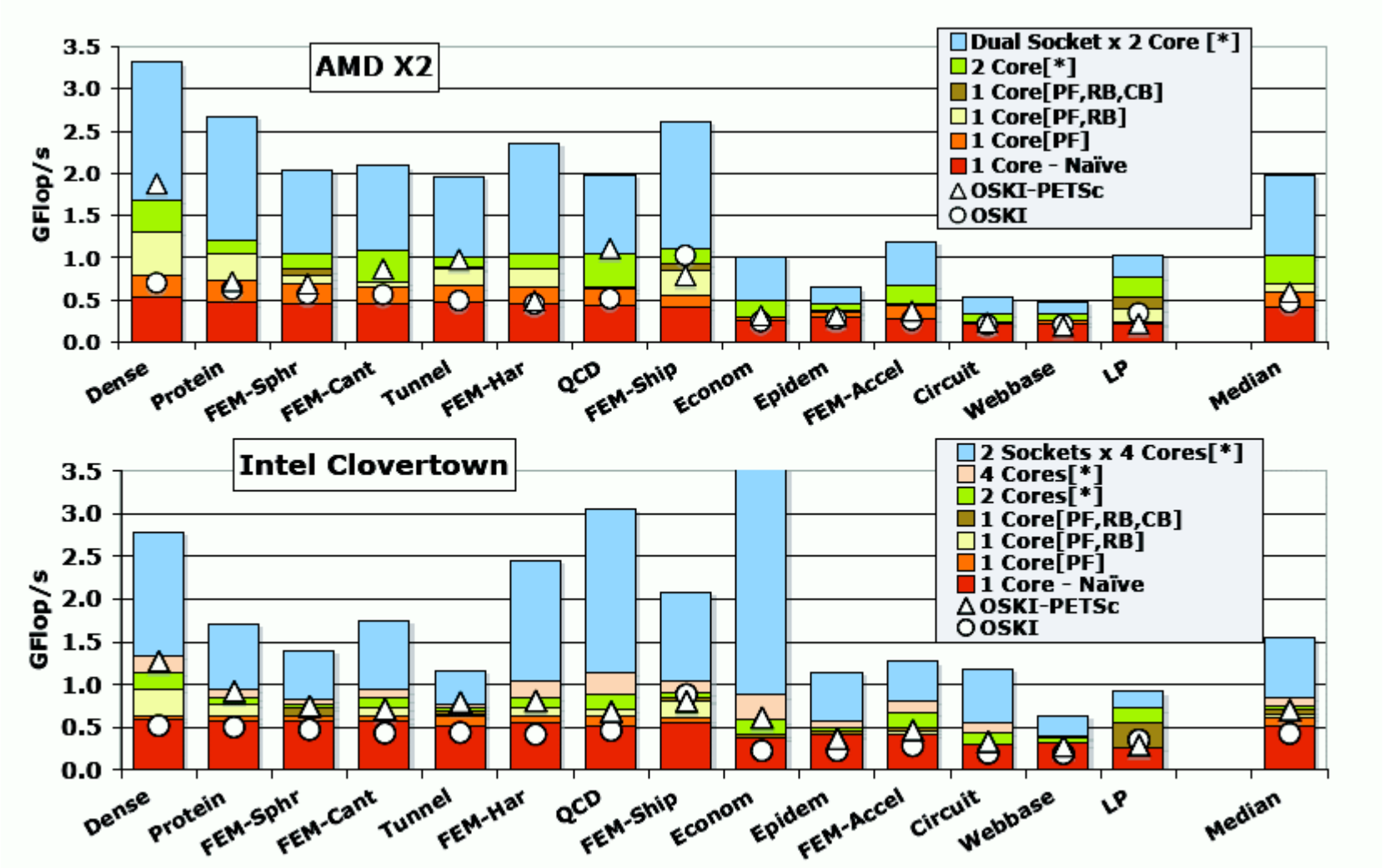
Promising Initial Results

Machine	Sustained Memory Bandwidth in GB/s (% of configuration peak bandwidth) Sustained Performance in GFlop/s (% of configuration peak computation) one socket, one core, one thread	one socket, one core, all threads	one socket, all cores, all threads	all sockets, all cores, all threads
AMD X2	$5.24\,GB/s$ (49.2%) $1.31\,GF/s$ (29.7%)	$5.24\,GB/s$ (49.2%) $1.31\,GF/s$ (29.7%)	$6.73\,GB/s$ (63.0%) $1.68\,GF/s$ (19.1%)	$13.35\,GB/s$ (62.6%) $3.31\,GF/s$ (18.8%)
Clovertown	$3.82\,GB/s$ (35.8%) $0.95\,GF/s$ (10.2%)	$3.82\,GB/s$ (35.8%) $0.95\,GF/s$ (10.2%)	$5.37\,GB/s$ (50.3%) $1.33\,GF/s$ (3.6%)	$11.24\,GB/s$ (52.7%) $2.79\,GF/s$ (3.7%)
Niagara2	$0.66\,GB/s$ (1.5%) $0.16\,GF/s$ (11.7%)	$3.79\,GB/s$ (8.9%) $0.94\,GF/s$ (67.3%)	$23.28\,GB/s$ (54.6%) $5.80\,GF/s$ (51.8%)	$23.28\,GB/s$ (54.6%) $5.80\,GF/s$ (51.8%)
Cell(PS3)	$4.76\,GB/s$ (18.6%) $1.15\,GF/s$ (62.9%)	$4.76\,GB/s$ (18.6%) $1.15\,GF/s$ (62.9%)	$21.16\,GB/s$ (82.6%) $5.12\,GF/s$ (46.6%)	$21.16\,GB/s$ (82.6%) $5.12\,GF/s$ (46.6%)
Cell(Blade)	$4.75\,GB/s$ (18.6%) $1.15\,GF/s$ (62.9%)	$4.75\,GB/s$ (18.6%) $1.15\,GF/s$ (62.9%)	$24.73\,GB/s$ (96.6%) $5.96\,GF/s$ (40.7%)	$47.29\,GB/s$ (92.4%) $11.35\,GF/s$ (38.8%)

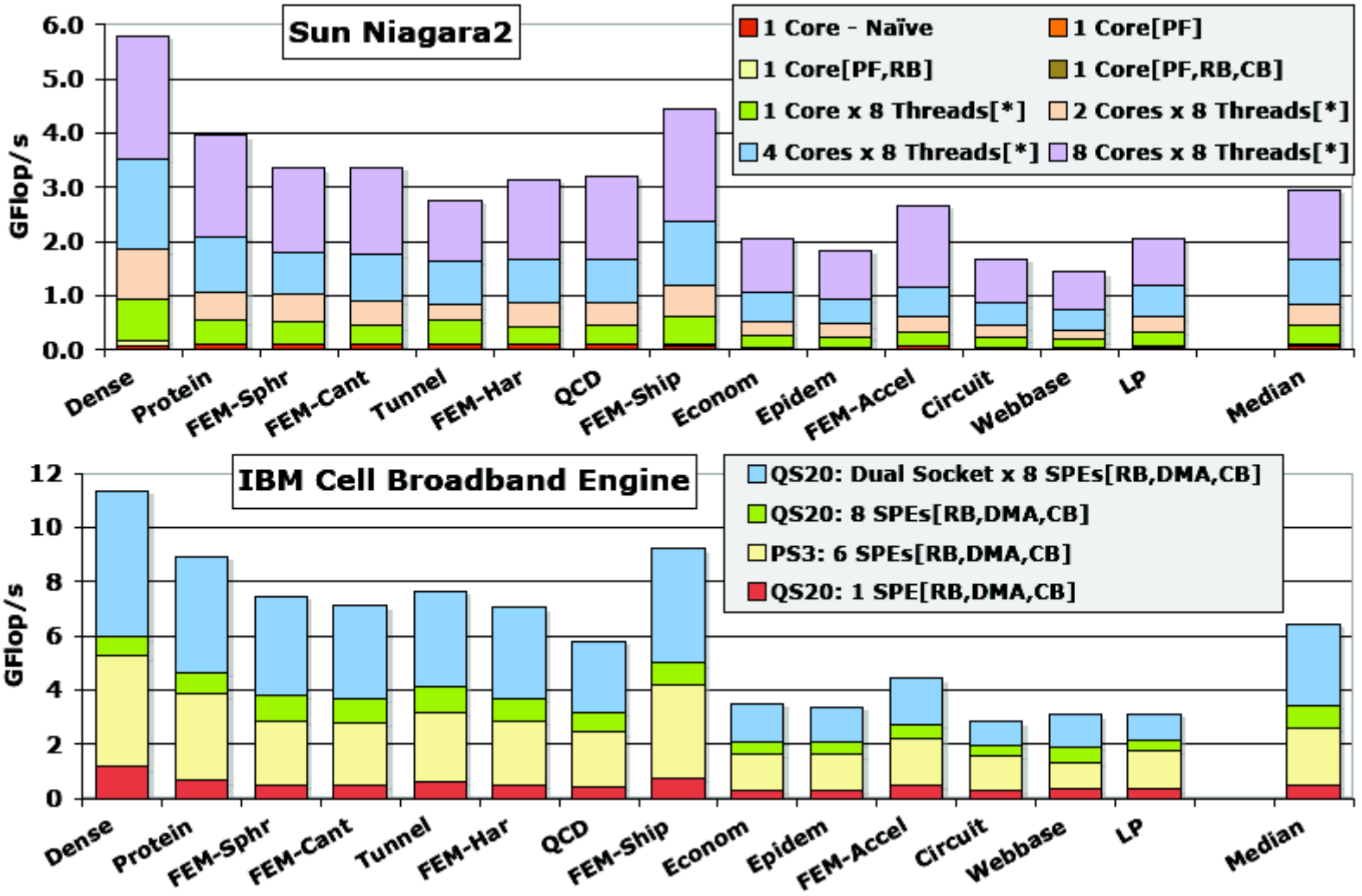
Table 3: Sustained bandwidth and computational rate for a dense matrix stored in sparse format, in GB/s (and percentage configuration's peak bandwidth) and GFlop/s (and percentage of configuration's peak performance).

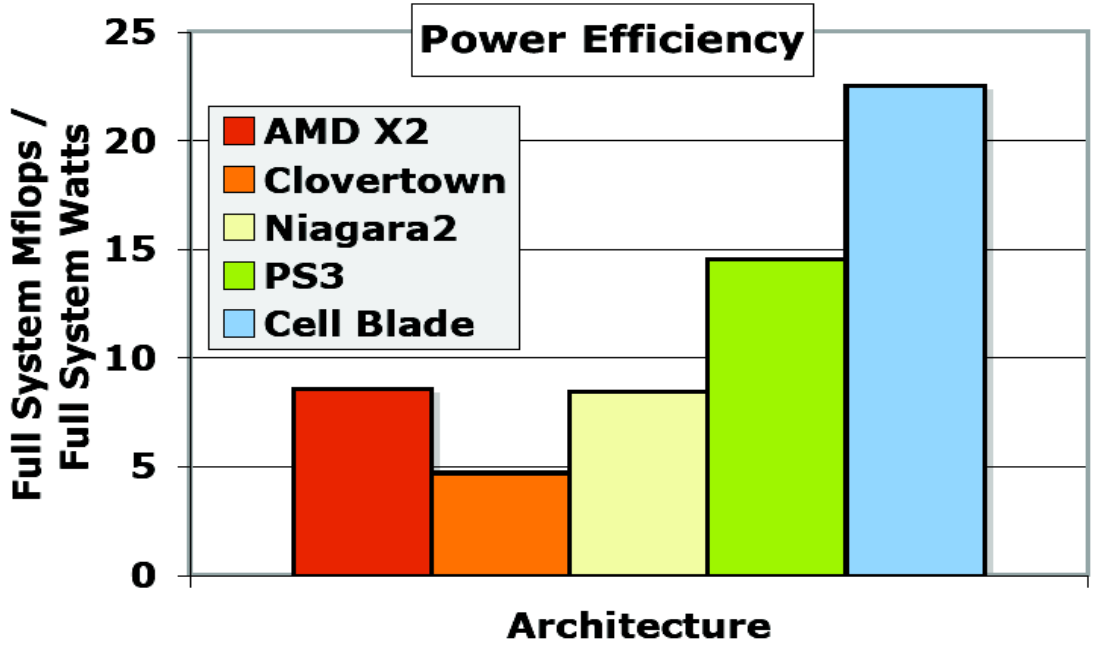
- Remember, outside of this project, we typically expect only 10% of peak performance

More Results



More Results





- Cell's SpMV times are dramatically faster
- All architectures showed good results
  - motivating the need for these optimizations!