Structured Grids and Sparse Matrix Vector Multiplication on the Cell Processor

Sam Williams
Lawrence Berkeley National Lab
One Cyclotron Rd.
MS:50A-1148
Berkeley, CA 94720
SWWilliams@lbl.gov

November 1, 2006

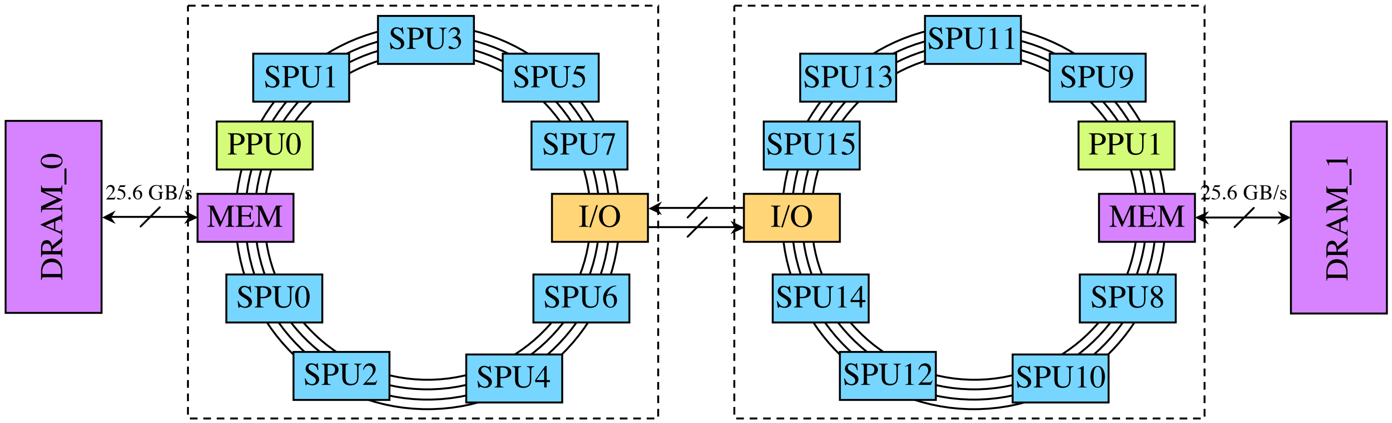
Outline

- Cell Architecture
- Programming Cell
- Benchmarks & Performance
  - Stencils on Structured Grids
  - Sparse Matrix-Vector Multiplication
- Summary

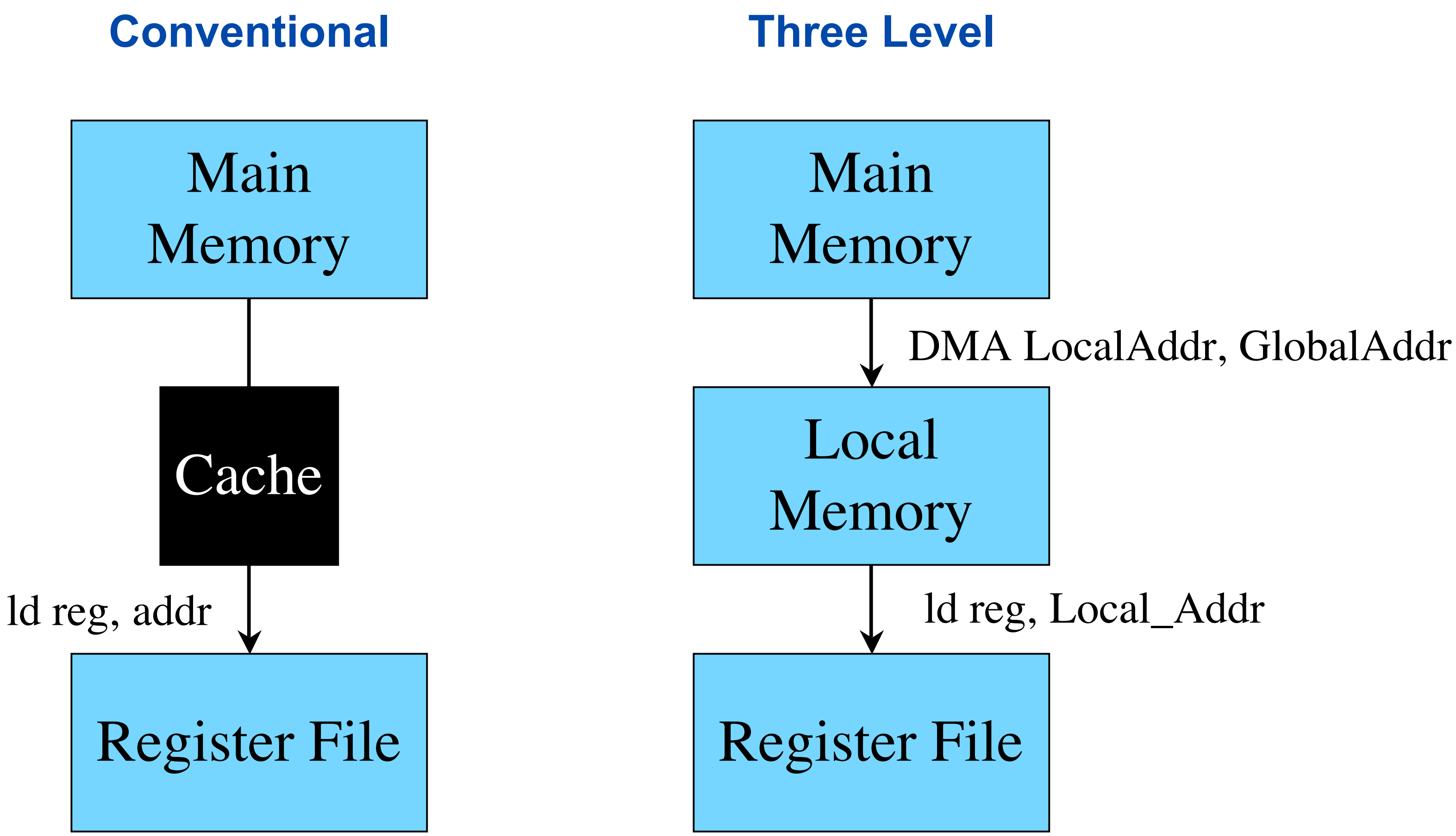
Cell Architecture

Cell Architecture

- **3.2GHz, 9 Core SMP**
  - **One core is a conventional cache based PPC**
  - **The other 8 are local memory based SIMD processors (SPEs)**
- **25.6GB/s memory bandwidth (128b @ 1.6GHz) to XDR**
- **2 chip (16 SPE) SMP blades**



Memory Architectures



SPE notes

- SIMD
- Limited dual issue
  - 1 float/ALU + 1 load/store/permute/etc… per cycle
- 4 FMA single precision datapaths, 6 cycle latency
- 1 FMA double precision datapath, 13 cycle latency
- Double precision instructions are not dual issued, and will stall all subsequent instruction issues by 7 cycles.
- 128b aligned loads/stores (local store to/from register file)
  - Must rotate to access unaligned data
  - Must permute to operate on scalar granularities

Cell Programming

- Modified SPMD (Single Program Multiple Data)
  - Dual Program Multiple Data (control + computation)
  - Data access similar to MPI, but data is shared like pthreads
- Power core is used to:
  - Load/initialize data structures
  - Spawn SPE threads
  - Parallelize data structures
  - Pass pointers to SPEs
  - Synchronize SPE threads
  - Communicate with other processors
  - Perform other I/O operations

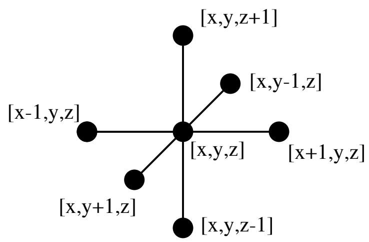
Processors Evaluated

	Cell SPE	X1E SSP	Power5	Opteron	Itanium2
Architecture	SIMD	Vector	Super Scalar	Super Scalar	VLIW
Frequency	3.2 GHz	1.13 GHz	1.9 GHz	2.2 GHz	1.4 GHz
GFLOP/s (double)	1.83	4.52	7.6	4.4	5.6
Cores used	8	4 (MSP)	1	1	1
Aggregate:					
L2 Cache	-	2MB	1.9MB	1MB	256KB
L3 Cache	-	-	36MB	-	3MB
DRAM Bandwidth	25.6 GB/s	34 GB/s	10+5 GB/s	6.4 GB/s	6.4 GB/s
GFLOP/s (double)	14.6	18	7.6	4.4	5.6

Stencil Operations on Structured Grids

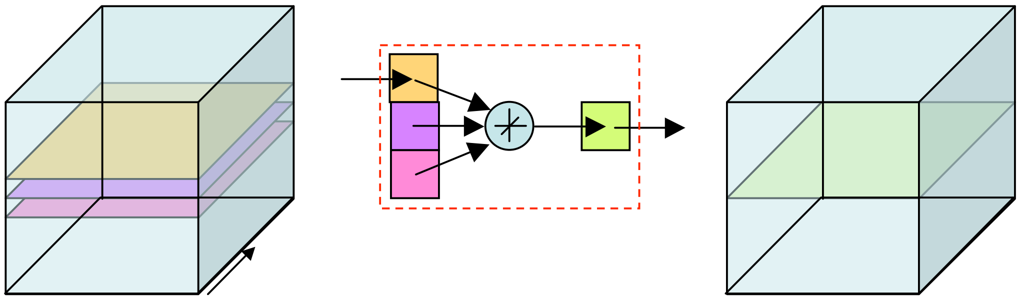
Stencil Operations

- Simple Example - The Heat Equation
  - $dT/dt = k\nabla^2 T$
  - Parabolic PDE on 3D discretized scalar domain
- Jacobi Style (read from current grid, write to next grid)
  - 8 FLOPs per point, typically double precision
  - Next[x,y,z] = Alpha*Current[x,y,z] +
    Beta*(Current[x-1,y,z] + Current[x+1,y,z] +
    Current[x,y-1,z] + Current[x,y+1,z] +
    Current[x,y,z-1] + Current[x,y,z+1])
  - Doesn’t exploit the FMA pipeline well
  - Basically 6 streams presented to the memory subsystem
- Explicit ghost zones bound grid



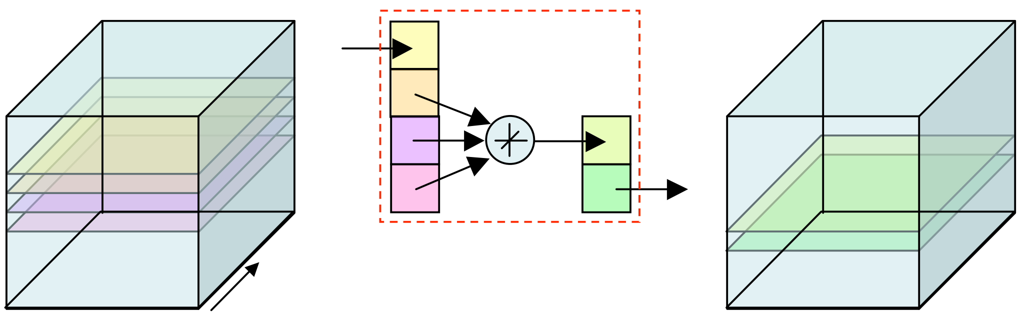
Optimization - Planes

- Naïve approach (cacheless vector machine) is to load 5 streams and store one.
- This is 8 flops per 48 bytes
  - memory limits performance to 4.2 GFLOP/s
- A better approach is to make each DMA the size of a plane
  - cache the 3 most recent planes (z-1, z, z+1)
  - there are only two streams (one load, one store)
  - memory now limits performance to 12.8 GFLOP/s
- Still must compute on each plane after it is loaded
  - e.g. forall Current_local[x,y] update Next_local[x,y]
  - Note: computation can severely limit performance



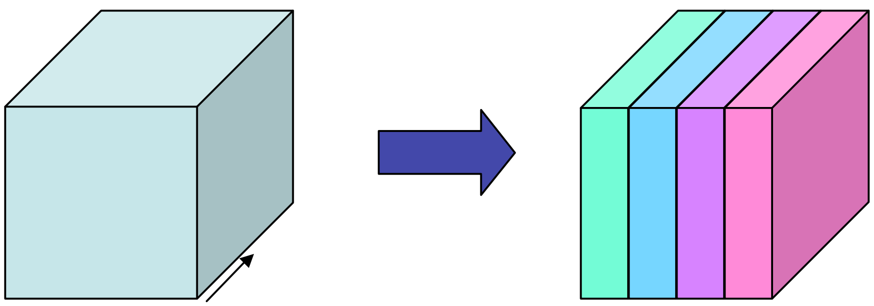
Optimization - Double Buffering

- Add a input stream buffer and and output stream buffer (keep 6 planes in local store)
- Two phases (transfer & compute) are now overlapped
- Thus it is possible to hide the faster of DMA transfer time and computation time



Optimization - Cache Blocking

- Domains can be quite large (~1GB)
- A single plane, let alone 6, might not fit in the local store
- Partition domain into cache blocked slabs so that 6 cache blocked planes can fit in the local store
- Partitioning in the Y dimension maintains good spatial and temporal locality
- Has the added benefit that cache blocks are independent and thus can be parallelized across multiple SPEs
- Memory efficiency can be diminished as an intra grid ghost zone is implicitly created.

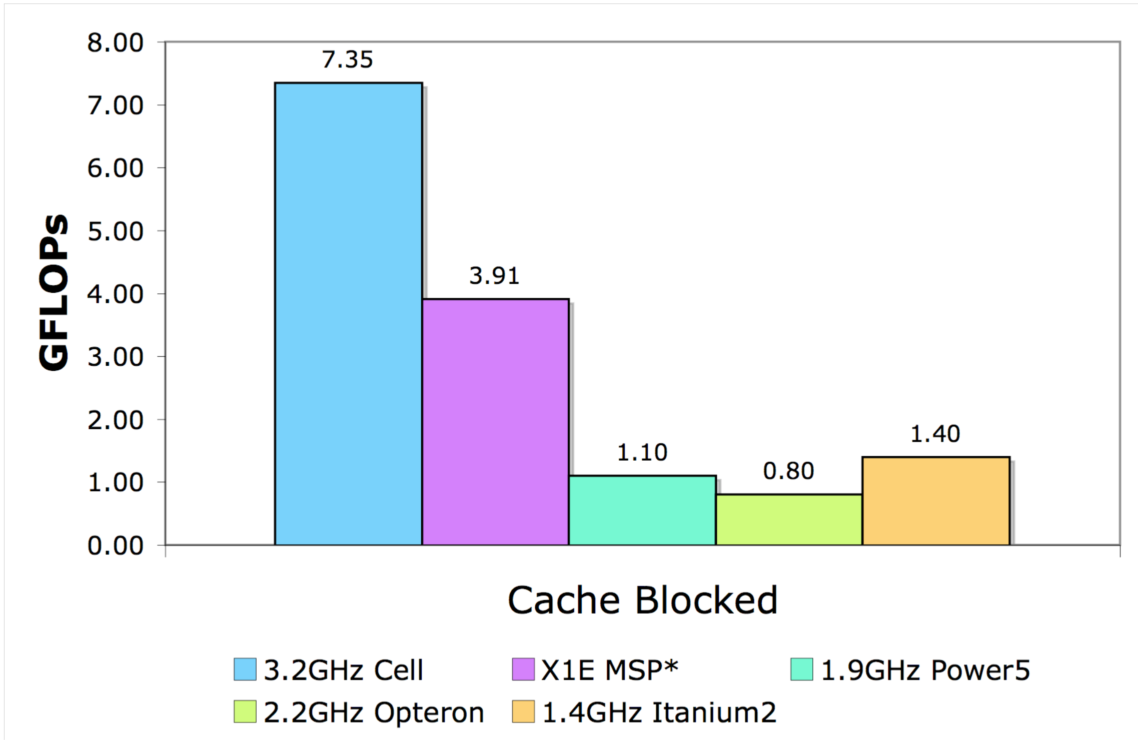


Optimization - Register Blocking

- Instead of computing on pencils, compute on ribbons (4x2)
- Hides functional unit & local store latency
- Minimizes local store memory traffic
- Minimizes loop overhead

- May not be beneficial / noticeable for cache based machines

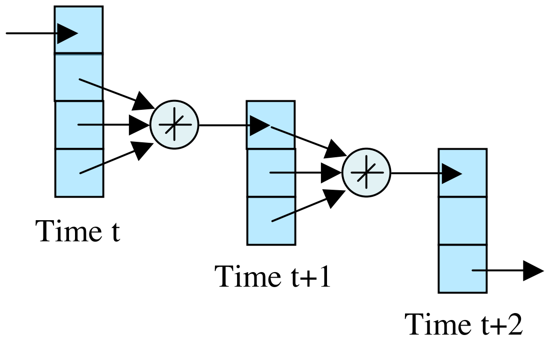
Double Precision Results



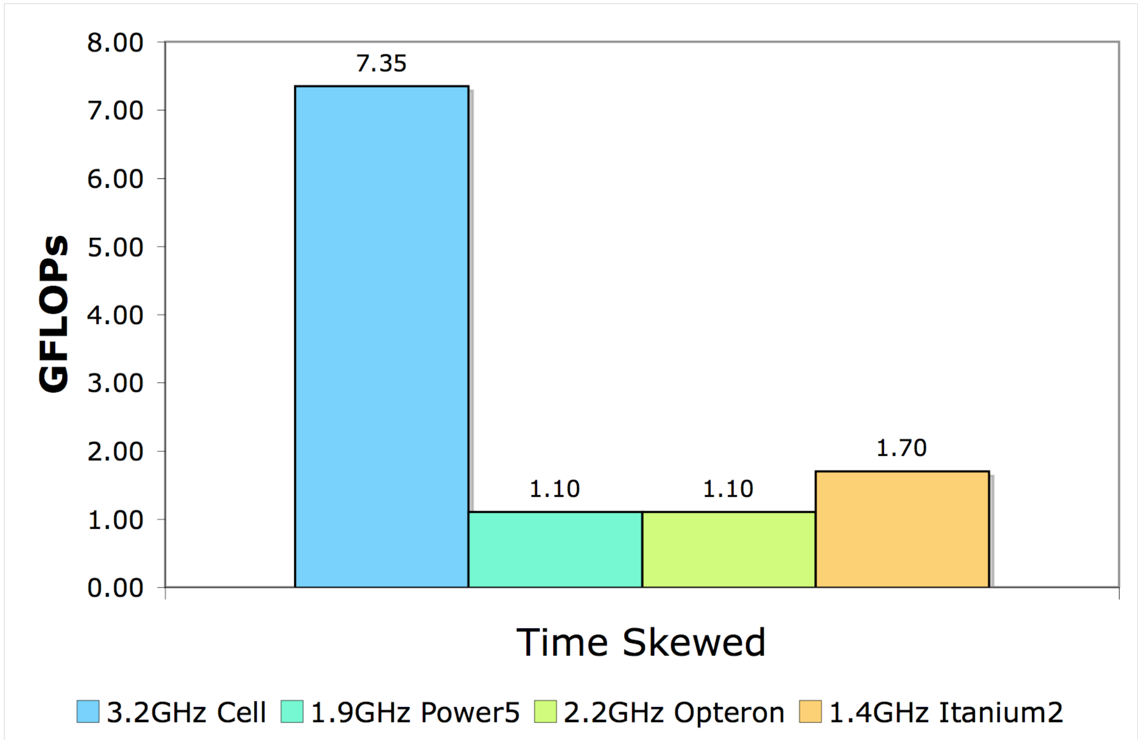
- ~256^3 problem
- Performance model matches well with hardware
- 5-9x performance of Opteron/Itanium/Power5
- X1E ran a slight variant (beta hard coded to be 1.0, not cache blocked)

Optimization - Temporal Blocking

- If the application allows it, perform block the outer (temporal) loop
- Only appropriate on memory bound implementations
  - Improves computational intensity
  - Cell SP or Cell with faster DP
- Simple approach
  - Overlapping trapezoids in time-space plot
  - Can be inefficient due to duplicated work
  - If performing n steps, the local store must hold 3(n+1) planes
- Time Skewing is algorithmically superior, but harder to parallelize.
- Cache Oblivious is similar but implemented recursively.

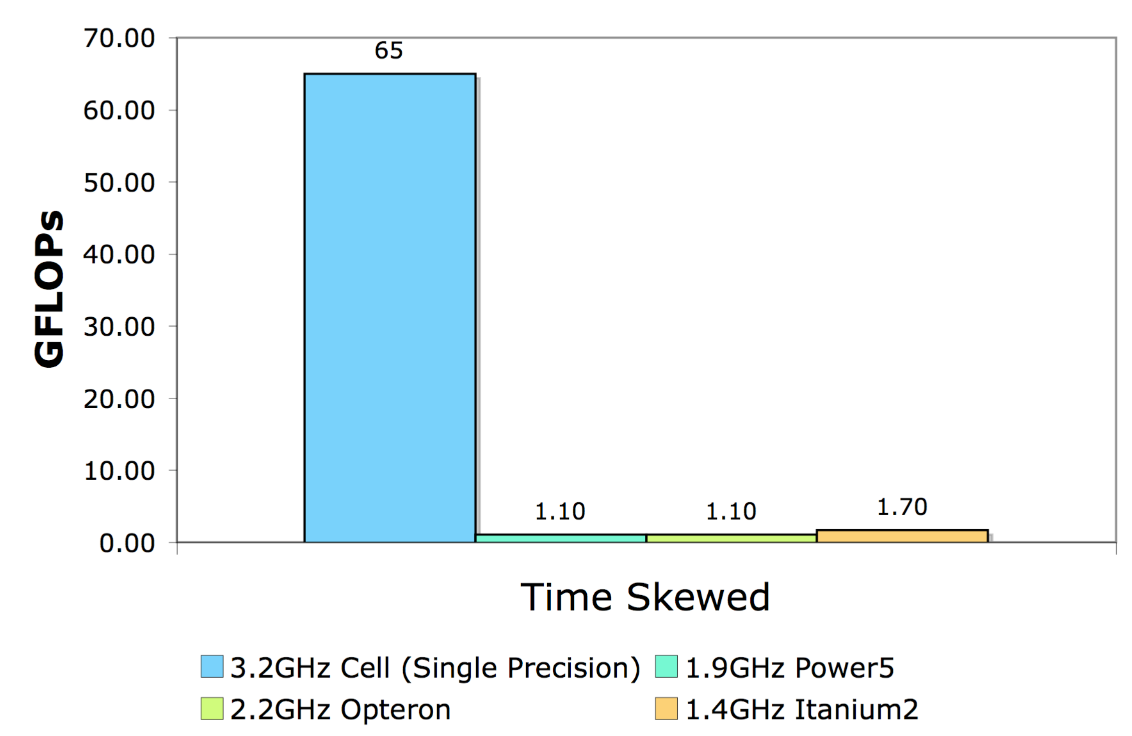


Temporal Blocking Results



- **Cell is computationally bound in double precision (no benefit in temporal blocking, so only cache blocked shown)**
- **Cache machines show the average of 4 steps of time skewing**

Temporal Blocking Results (2)

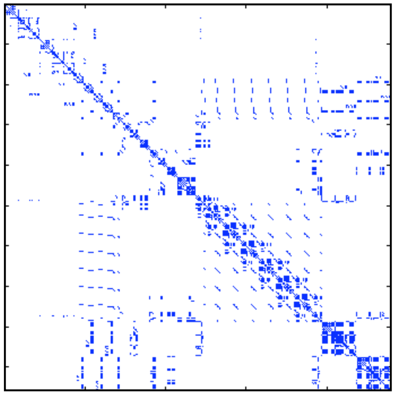


- **Temporal blocking on Cell was implemented in single precision (four step average)**
- **Others still use double precision**

Sparse Matrix-Vector Multiplication

Sparse Matrix Vector Multiplication

- Most of the matrix entries are zeros, thus the non zero entries are sparsely distributed
- Dense methods compute on all the data, sparse methods only compute the nonzeros (as only they compute to the result)
- Can be used for unstructured grid problems
- Issues
  - Like DGEMM, can exploit a FMA well
  - Very low computational intensity (1 FMA for every 12+ bytes)
  - Non FP instructions can dominate
  - Can be very irregular
  - Row lengths can be unique and very short



Compressed Sparse Row

- Compressed Sparse Row (CSR) is the standard format
  - Array of nonzero values
  - Array of corresponding column for each nonzero value
  - Array of row starts containing index (in the above arrays) of first nonzero in the row

Optimization - Double Buffer Nonzeros

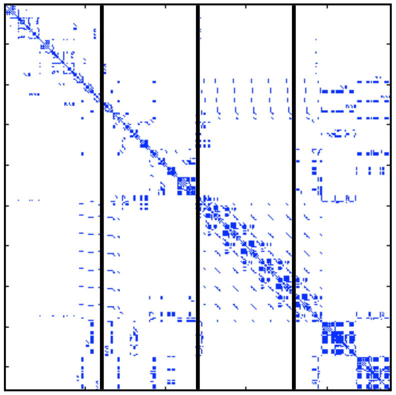
- Computation and Communication are approximately equally expensive
- While operating on the current set of nonzeros, load the next (~1K nonzero buffers)
- Need complex (thought) code to stop and restart a row between buffers
- Can nearly double performance

Optimization - SIMDization

- **Row Padding**
  - Pad rows to the nearest multiple of 128b
  - Might requires O(N) explicit zeros
  - Loop overhead still present
  - Generally works better in double precision
- **BCSR**
  - Nonzeros are grouped into dense r x c blocks(sub matrices)
  - O(nnz) explicit zeros are added
  - Choose r&c so that it meshes well with 128b registers
  - Performance can hurt especially in DP as computing on zeros is very wasteful
  - Can hide latency and amortize loop overhead

Optimization - Cache Blocked Vectors

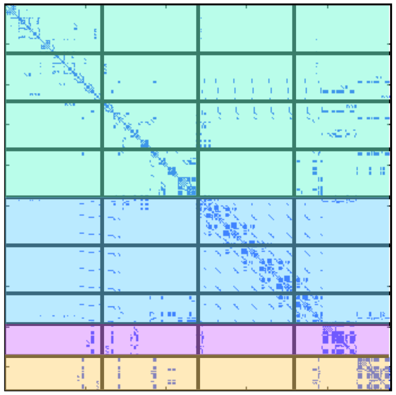
- Doubleword DMA gathers from DRAM can be expensive
- Cache block source and destination vectors
- Finite LS, so what's the best aspect ratio?
- DMA large blocks into local store
- Gather operations into local store
  - ISA vs. memory subsystem inefficiencies
  - Exploits temporal and spatial locality within the SPE
- In effect, the sparse matrix is explicitly blocked into submatrices, and we can skip, or otherwise optimize empty submatrices



- Indices are now relative to the cache block
  - half words
  - reduces memory traffic by 16%

Optimization - Load Balancing

- Potentially irregular problem
- Load imbalance can severely hurt performance
- Partitioning by rows is clearly insufficient
- Partitioning by nonzeros is inefficient when the matrix has few but highly variable nonzeros per row.



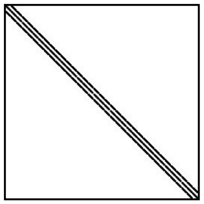
- Define a cost function of number of row starts and number of nonzeros.
- Determine the parameters via static timing analysis or tuning.

Other Approaches

- BeBop / OSKI on the Itanium2 & Opteron
  - uses BCSR
  - auto tunes for optimal r x c blocking
  - Cell implementation is similar
- Cray's routines on the X1E
  - Report best of CSRP, Segmented Scan & Jagged Diagonal

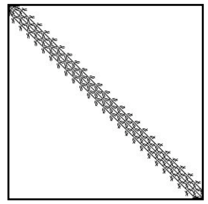
Benchmark Matrices

#06 - FEM Crystal



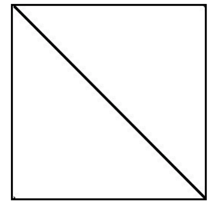
N=14K, NNZ=490K

#09 - 3D Tube



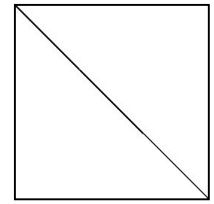
N=45K, NNZ=1.6M

#17 - FEM



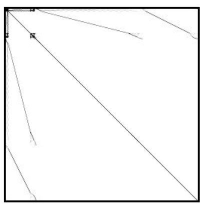
N=22K, NNZ=1M

#36 - CFD



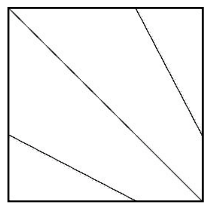
N=75K, NNZ=325K

#18 - Circuit



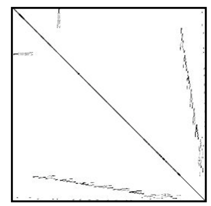
N=17K, NNZ=125K

#25 - Financial



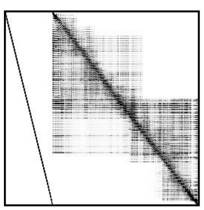
N=74K, NNZ=335K

#27 - NASA



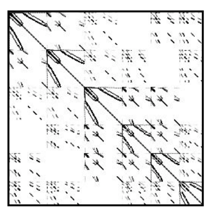
N=36K, NNZ=180K

#15 - PDE



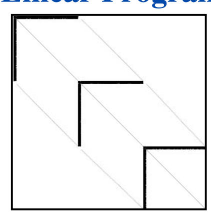
N=40K, NNZ=1.6M

#28 - Vibroacoustic



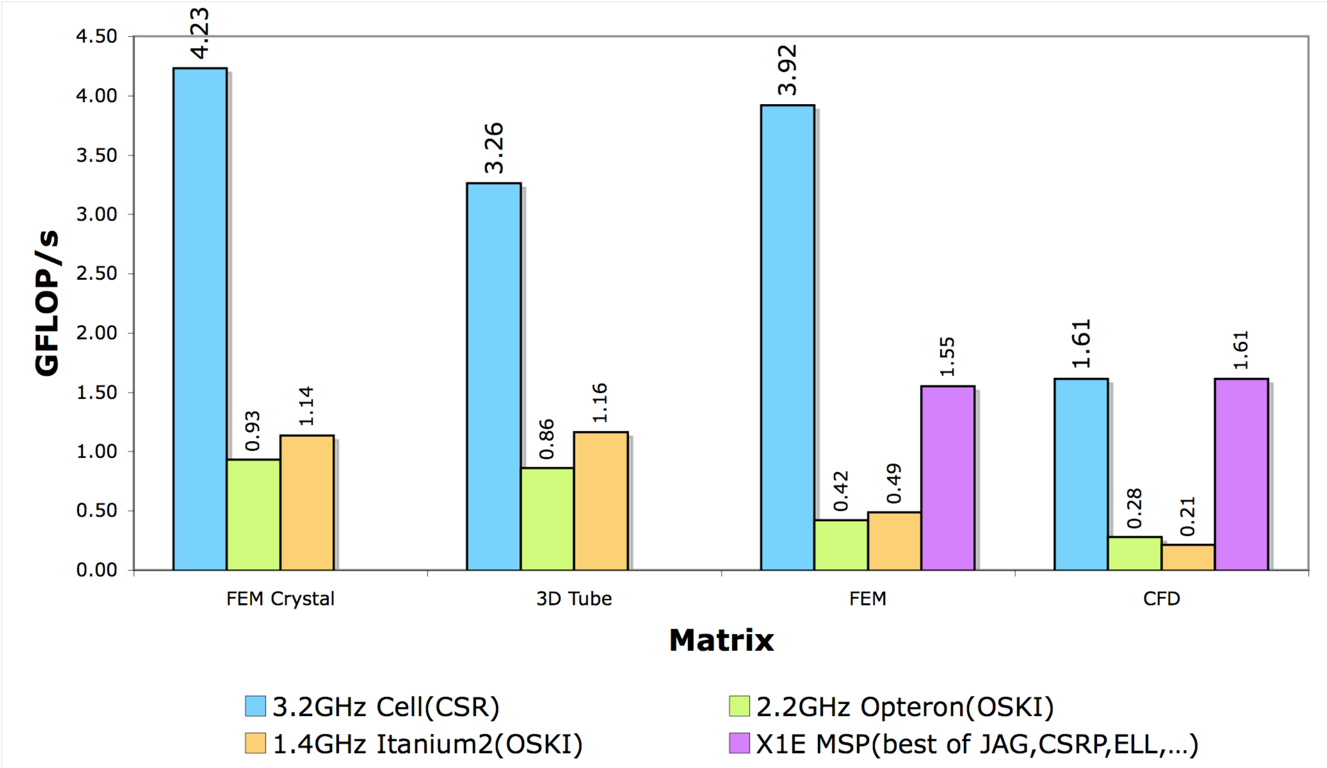
N=12K, NNZ=177K

#40 - Linear Programming



N=31K, NNZ=1M

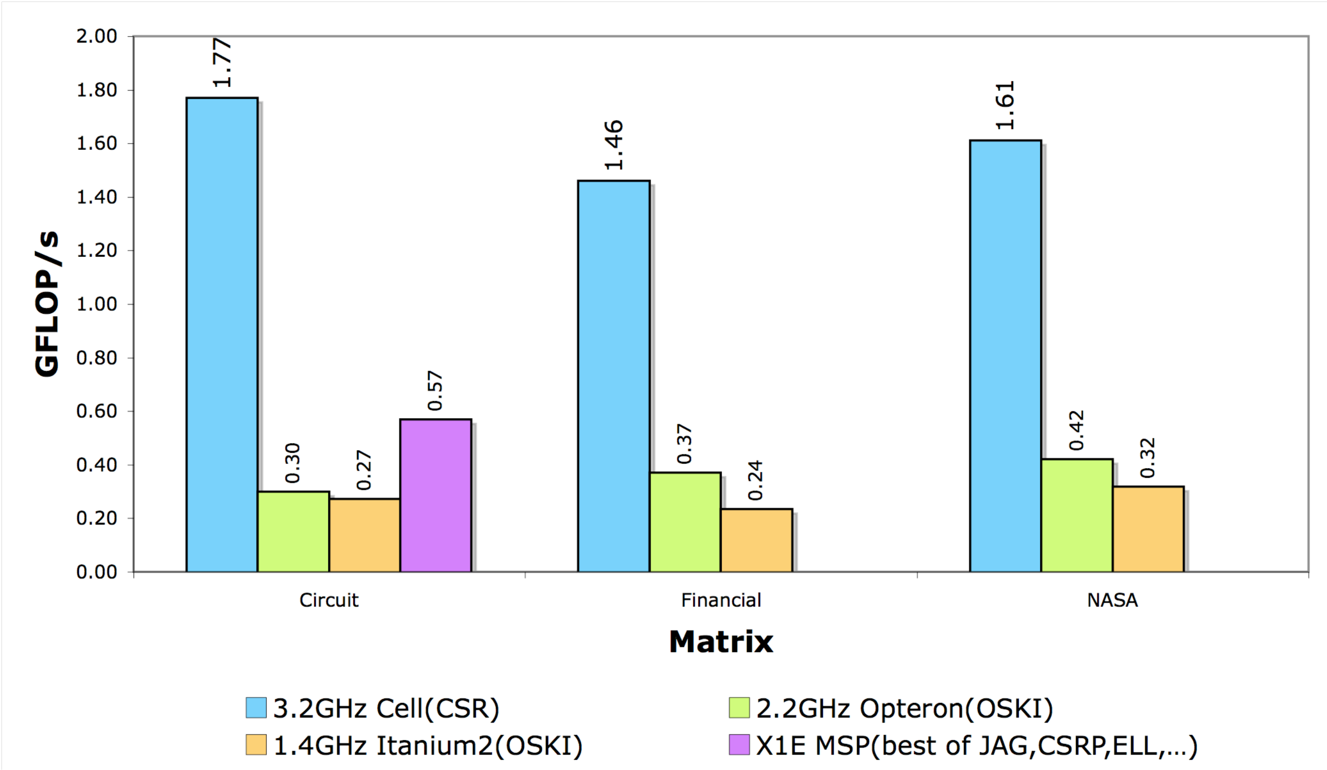
Double Precision SpMV Performance



Notes:

- few nonzeros per row severely limited performance on CFD
- BCSR was clearly exploited on the first 2
- 3-8x faster than Opteron/Itanium

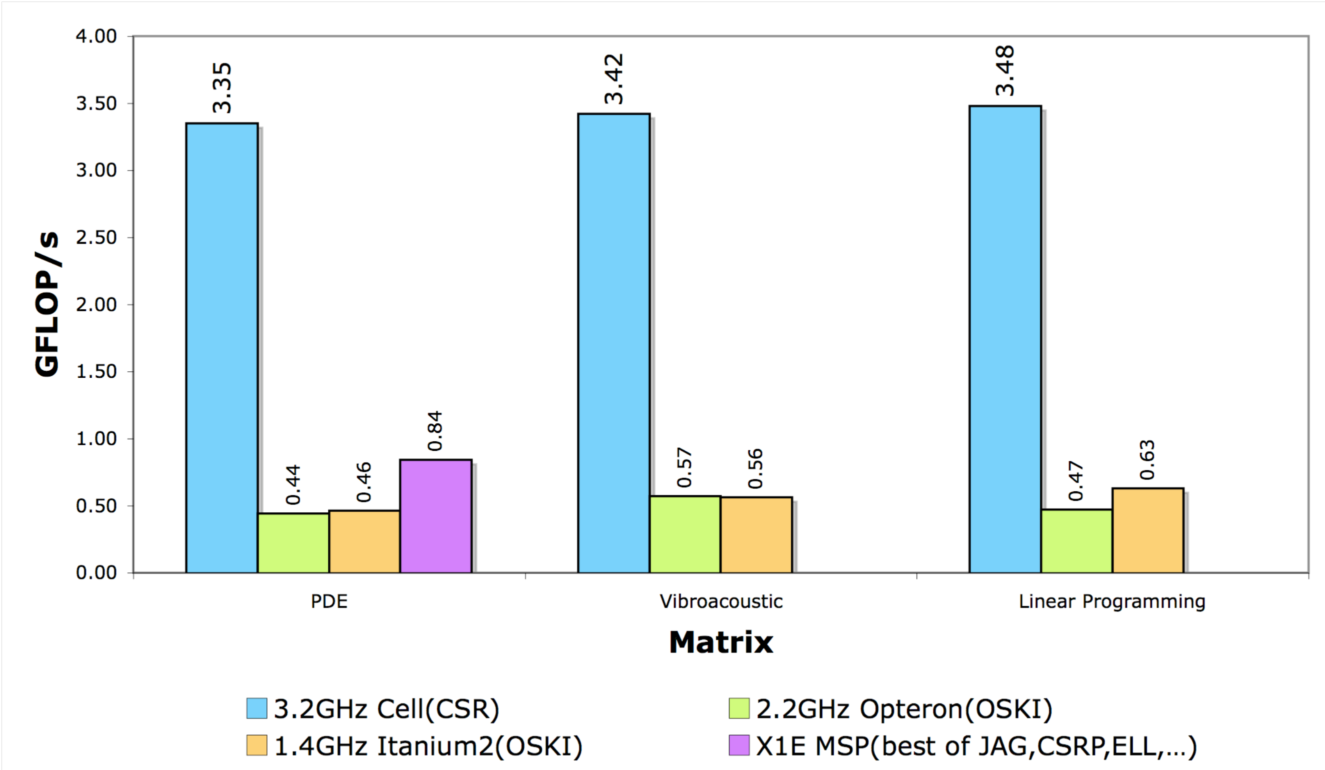
Double Precision SpMV Performance (2)



Notes:

- few nonzeros per row (hurts a lot)
- Not structured as well as previous set
- 4-6x faster than Opteron/Itanium

Double Precision SpMV Performance (3)



Notes:

- **many nonzeros per row**
- **Load imbalance hurt cell's performance by as much as 20%**
- **5-7x faster than Opteron/Itanium**

Conclusions

Conclusions

- Cell performance is far more predictable than conventional OOO machines
- Even in double precision, it obtains much better performance on a surprising variety of codes.
- Cell can eliminate unneeded memory traffic, hide memory latency, and thus achieves a much higher percentage of memory bandwidth.

- Instruction set can be very inefficient for poorly SIMDizable or misaligned codes.
- Loop overheads can heavily dominate performance.

- Programming model could be streamlined

Acknowledgments

- This work is a collaboration with the following LBNL/FTG members:
  - John Shalf, Lenny Oliker, Shoaib Kamil, Parry Husbands, Kaushik Datta and Kathy Yelick
- Additional thanks to
  - Joe Gebis and David Patterson
- X1E FFT numbers provided by:
  - Bracy Elton, and Adrian Tate
- Cell access provided by IBM

Questions?