The Potential of the Cell Processor for Scientific Computing

Sam Williams
samw@cs.berkeley.edu

Computational Research Division

Future Technologies Group

March 3, 2006

Outline

- Introduction to Cell Architecture
- Programming the Cell Processor
- Benchmarks & Performance
  - Matrix-Matrix Multiplication
  - Sparse Matrix-Vector Multiplication
  - Stencils on Structured Grids
  - FFTs
- Summary

Introduction to Cell

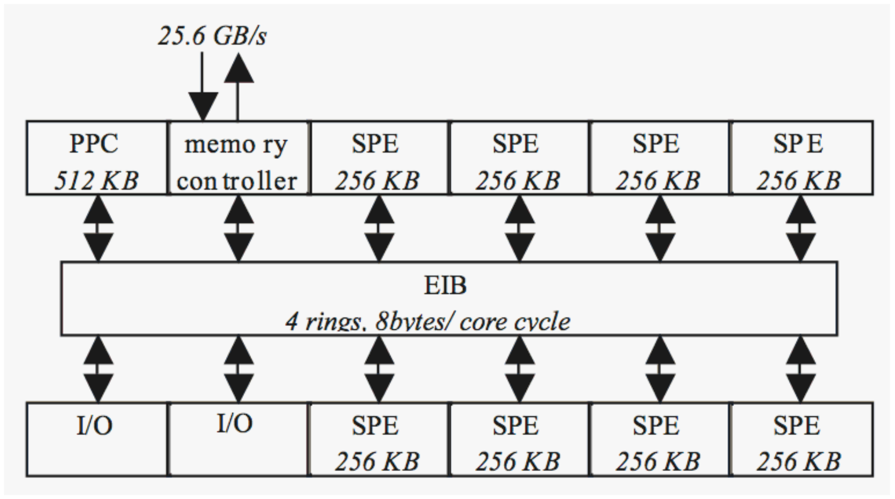
- Will be used in the PS3
- radical departure from the conventional designs including the XBOX 360's Xenon

Cell/PS3	XBOX 360
Heterogeneous	Homogeneous
PPC + 8 SIMD cores	3 x PPC
Software-controlled memory architecture	Conventional Cache-based memory hierarchy (1MB)
221mm^2 (+30%)	168mm^2

- Software controlled memory makes it a very interesting alternative to cache based processors

Cell Architecture

Cell Processor Architecture



- All units connected via EIB
  - 4 x 128b rings @ 1.6GHz
- PPC core @ 3.2GHz
- 8 x SPE's (128b SIMD core)
  - Off-load engine
    Between a μP and a coP
  - 256KB Local store
    - Private address space
    - access to global store via DMA
    - No unaligned access (only via permutes)
  - Dual SIMD issue (private PC)
    - one arithmetic/float/etc…
    - one load/store/permute/branch/channel
  - Statically scheduled, in-order 7 cycle pipelines
  - 3W @ 3.2 GHz
- Memory controller (25.6GB/s dual XDR)
- Perhaps 40W total if PPC is idle

Micro-architectural Issues

- PPE (PowerPC Processing Element)
  - 512KB cache (coherent with DMAs, not LS)
  - Dual thread, dual issue, in order
  - VMX unit + scalar DP FMA
- SPE (Synergistic Processing Element)
  - 7 cycle in order dual SIMD pipelines
  - Single Precision
    - 4 FMA datapaths, 6 cycle latency = 25.6 GFLOP/s
  - Double Precision
    - 1 FMA datapath, 13 cycle latency
    - 13 cycle pipeline doesn't fit in a 7 cycle forwarding network, so they just stall after issuing for correctness = 1.83 GFLOP/s
    - Prohibit dual issuing DP instructions = 1.6 GFLOP/s
  - Software managed BTB (branch hints)

Cell Programming

Programming Model - SCM

- Local store appears to be the SPU's entire memory space.
- However, with DMAs, it can be programmed as a software (user) controlled memory.
- For algorithms whose address stream is expressible as a list of addresses & sizes, a series of DMAs can be issued to transfer data from global store (DRAM) to local store (SRAM) ~ remote get
- Analogous to vector loads from DRAM to a large SRAM (vector register file)
- Local store has constant 6 cycle latency (no cache misses)

- Note: LS's are aliased to the global address space, so it is possible for one SPU to DMA data directly from another SPU

Programming Model - Distributed Memory

- Conceptualize Cell as a distributed memory machine with:
  - One slow processor(PPE) with lots(1000x) of memory
  - Eight fast processors(SPEs) with very little memory
  - 4 high bandwidth rings to connect them

Programming Model - DMA

- Granularity (alignment/length) is quadword (max=16KB)
- Very low level intrinsics (no error checking)
- Non-blocking (wait on tag mask)
- Single Stanza (MFC_GET, MFC_PUT)
  - Ideal for single long unit stride accesses
  - spu_mfcdma32(LSA,GSA,stanzaSize,tag,MFC_GET_CMD);
- Multiple Stanzas (MFC_GETL or MFC_PUTL)
  - Stanzas are then gathered from GS, and packed into LS (tiling in column major matrix)
  - In general could be used to gather individual QW's or stanzas of independent length
  - Specify an array(list) in LS of {GSA:Length}
  - spu_mfcdma32(LSA,&list,listSize,tag,MFC_GETL_CMD);

Programming Model - Double Buffering

- MFC (channel) commands are non blocking
- exploit the parallelism (SPU & MFC) via double buffering
- Total time is the max of computational time and communication time
- Startup/finish penalties

Programming Model - Intrinsics

- We did not benchmark the compiler's ability to SIMDize
- For all critical sections, wrote code with SIMD/quadword intrinsics.
- Ideal performance, somewhat more work than C, far less work than assembly

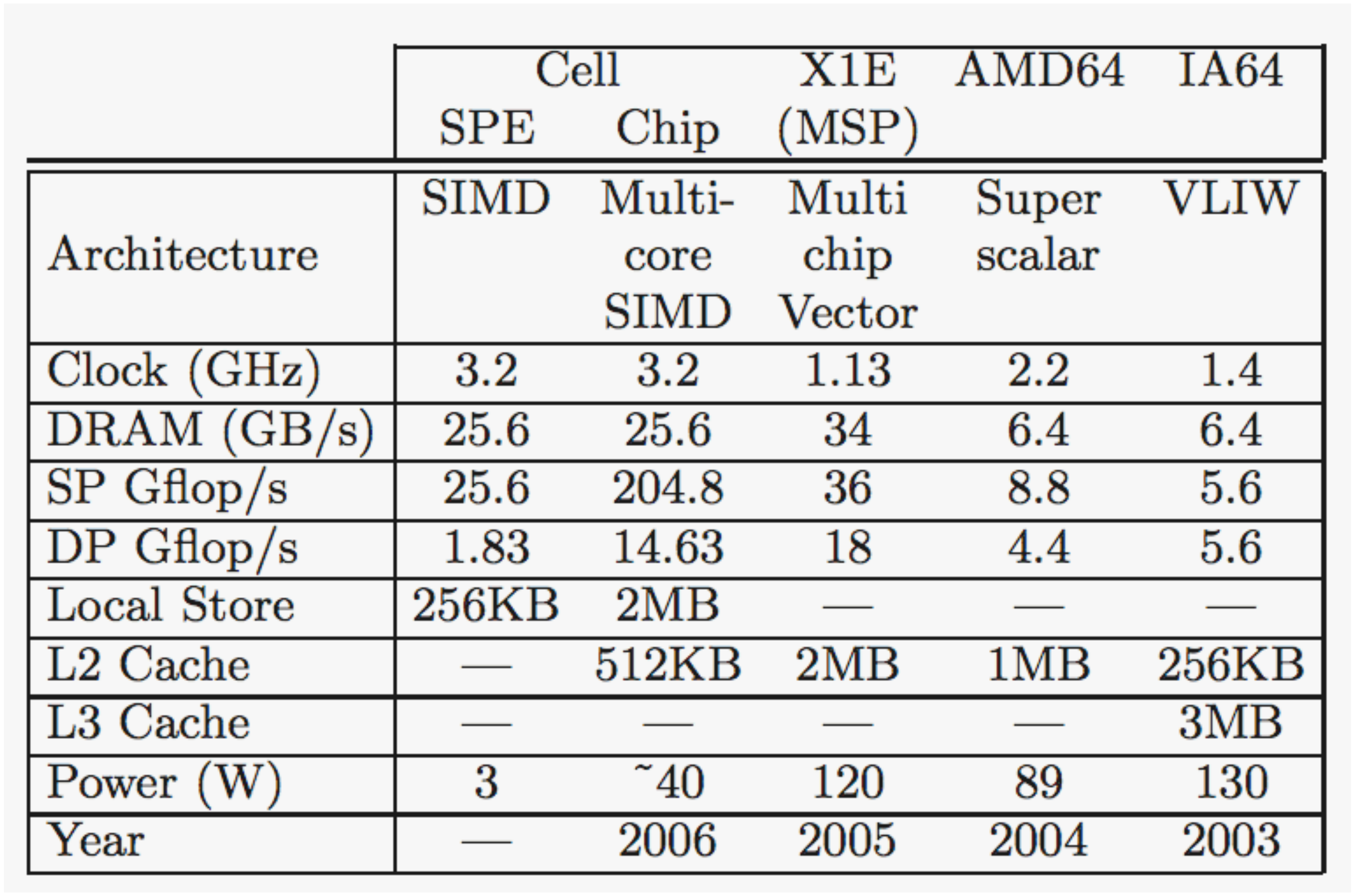
Programming Model - Threading

- Programmed as hierarchical SPMD
  - PPE is used to partition and load balance
  - PPE is not used for any computation
- Implementation
  - PPE creates 8 SPE threads
    - SPE executable is embedded within a PPE program
  - passes pointers to key data structures
  - periodically communicates via mailboxes
  - waits for SPEs to finish

Estimation, Simulation and Exploration

- Modeling
  - Double buffered + long DMAs + in order machine
  - use static timing analysis + memory traffic modeling
  - For regular data structures, spreadsheet modeling works
  - SpMV requires more advanced modeling
- Full System Simulator
  - based on mambo, cycle accurate, includes static timing analyzer, compilers, etc…
- Cell+
  - How severely does DP throughput of 1 SIMD instruction every 7 or 8 cycles impair performance?
  - Cell+ model fully utilizes the DP datapath
  - 1 SIMD instruction every 2 cycles
  - Allows dual issuing of DP instructions with loads/stores/permutes/branch

Comparing Processors



	Cell		X1E	AMD64	IA64
	SPE	Chip	(MSP)		
Architecture	SIMD	Multi-core SIMD	Multi chip Vector	Super scalar	VLIW
Clock (GHz)	3.2	3.2	1.13	2.2	1.4
DRAM (GB/s)	25.6	25.6	34	6.4	6.4
SP Gflop/s	25.6	204.8	36	8.8	5.6
DP Gflop/s	1.83	14.63	18	4.4	5.6
Local Store	256KB	2MB	—	—	—
L2 Cache	—	512KB	2MB	1MB	256KB
L3 Cache	—	—	—	—	3MB
Power (W)	3	˜40	120	89	130
Year	—	2006	2005	2004	2003

Note: Cell performance does not include the Power core

Benchmark Kernels

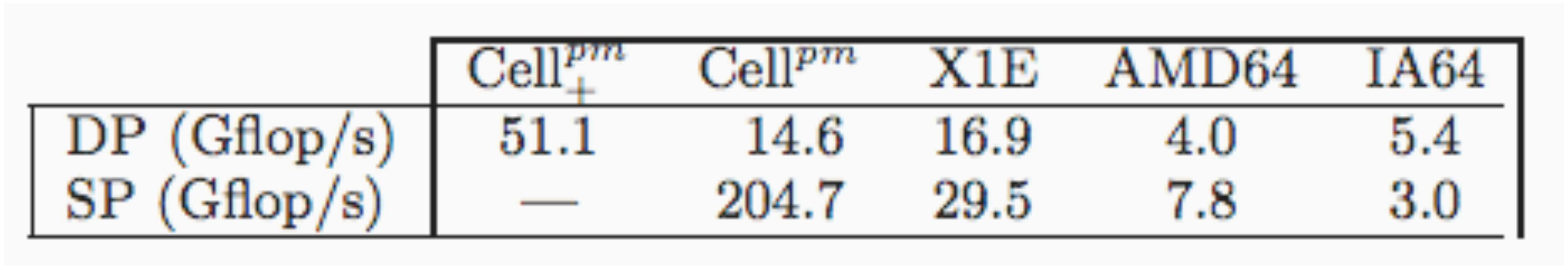
- Matrix-Matrix Multiplication
- Sparse Matrix-Vector Multiplication
- Stencil Computations on Structured Grids
- 1D FFTs
- 2D FFTs

Matrix-Matrix Multiplication

Dense Matrix-Matrix Multiplication

- Blocking
  - Explicit (BDL) or implicit blocking (gather stanzas)
  - Hybrid method would be to convert and store to DRAM on the fly
  - Choose a block size so that kernel is computationally bound
    - $\geq 64^2$ in single precision
    - much easier in double precision (14x computational time, 2x transfer time)
- Parallelization
  - Partition A & C among SPUs
  - Future work - cannon’s algorithm

GEMM - Results



	Cell_{+}^{pm}	Cell^{pm}	X1E	AMD64	IA64
DP (Gflop/s)	51.1	14.6	16.9	4.0	5.4
SP (Gflop/s)	—	204.7	29.5	7.8	3.0

Notes:

Cell^{pm} = Performance Model

IBM's hardware numbers come very close to these

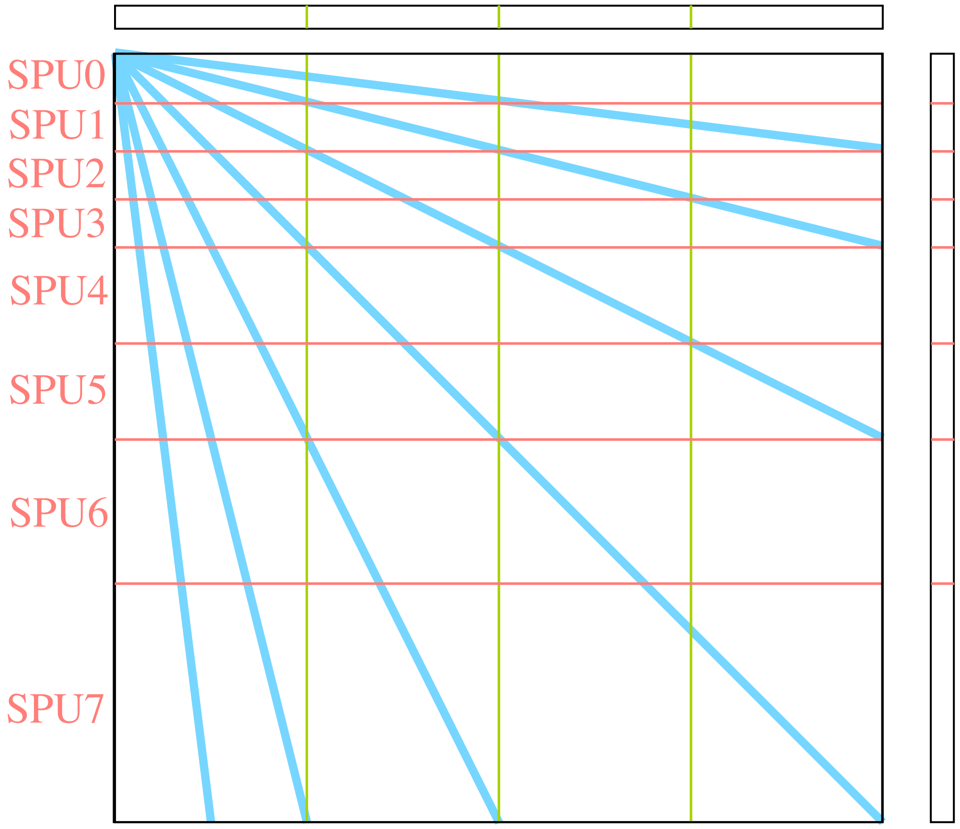
Sparse Matrix-Vector Multiplication

Sparse Matrix-Vector Multiplication

- Use ~CSR or BCSR(performance model only)
- SIMDization
  - require all row lengths to be a multiple of 4
  - Nonzero values are quadword aligned in SP
- Explicitly cache block columns
  - Exploit spatial locality within the local store
- Implicitly cache block the rows
- Matrix “cache block”
  - sub Matrix for the corresponding column and row blocks
- Cache block Parallelization strategies:
  - partition by rows
  - partition by nonzeros.
- Double buffer nonzeros
  - Overlaps computation and communication
  - requires restarting in the middle of a row

SpMV - example figure

- Explicitly choose column blocking via cost function
  - cache block perimeter is fixed (LS)
  - What is optimal r x c?
- Parallelize across SPUs
  - Cost function of execution time
  - αRows + βNZ
- Partially double buffer row pointers to find structure
  - Completely eliminate empty blocks
  - Prune empty rows

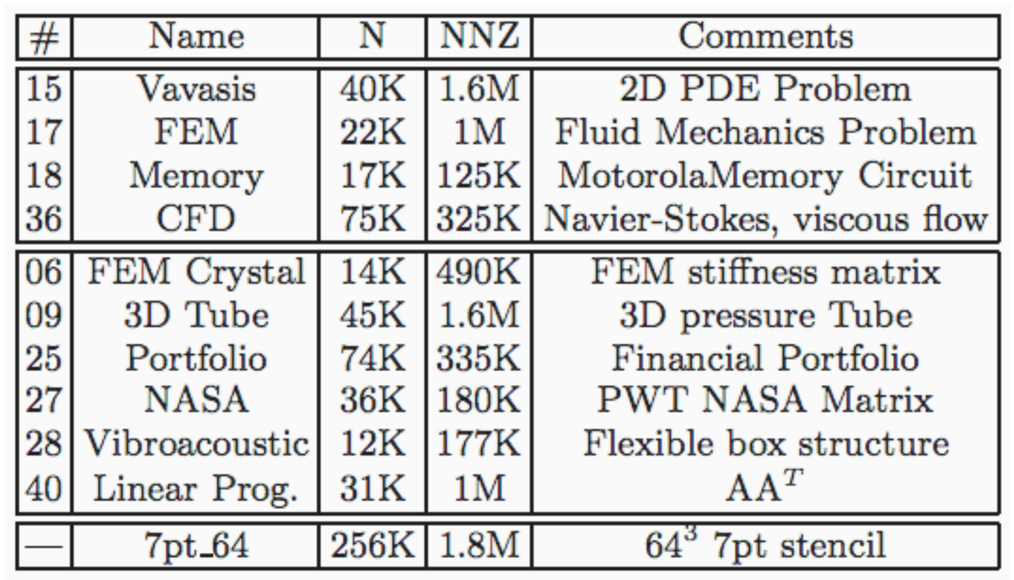


SpMV - implementation

- Use Performance estimates to guide actual implementation
  - Double precision / Row lengths must be even (QW aligned), no BCSR(yet)
  - Parallelization
    - costFunction(rows,NZs) ~ execution time
  - Runtime blocking
    - cost function based
  - LS=256KB=32K doubles, max column block = 32K
    - no need to transfer 32b absolute column index, when we only need a 15b relative index)
  - Runtime search for structure
    - empty cache blocks
    - search for first non empty row

SpMV - matrices

- 4 nonsymmetric SPARSITY matrices
- 6 symmetric SPARSITY matrices
- 7pt Heat equation matrix

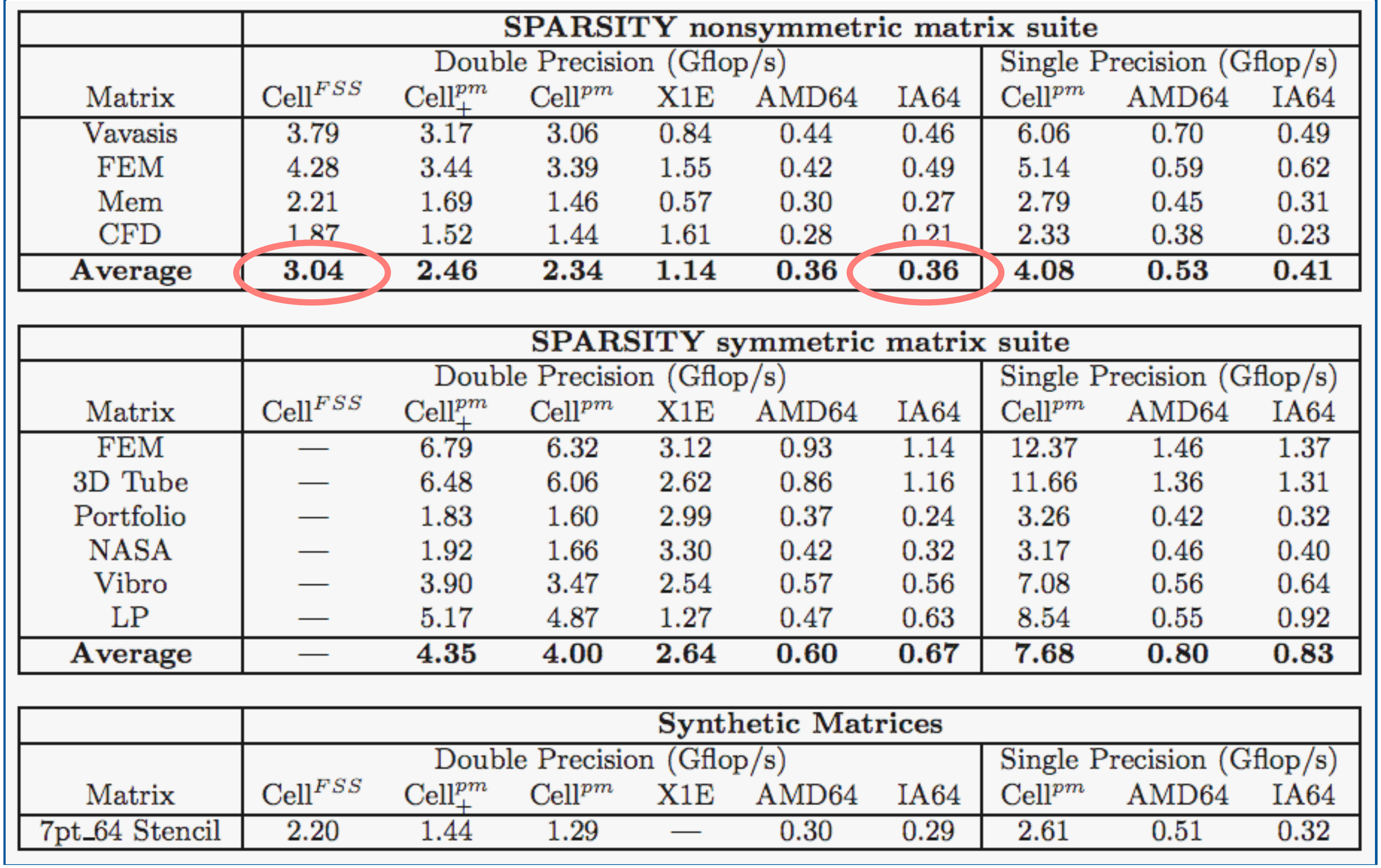


#	Name	N	NNZ	Comments
15	Vavasis	40K	1.6M	2D PDE Problem
17	FEM	22K	1M	Fluid Mechanics Problem
18	Memory	17K	125K	MotorolaMemory Circuit
36	CFD	75K	325K	Navier-Stokes, viscous flow
06	FEM Crystal	14K	490K	FEM stiffness matrix
09	3D Tube	45K	1.6M	3D pressure Tube
25	Portfolio	74K	335K	Financial Portfolio
27	NASA	36K	180K	PWT NASA Matrix
28	Vibroacoustic	12K	177K	Flexible box structure
40	Linear Prog.	31K	1M	AA^T
—	7pt_64	256K	1.8M	64^3 7pt stencil

SpMV - other machines

- 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 and Jagged Diagonal

SpMV - Results



	SPARSITY nonsymmetric matrix suite								
	Double Precision (Gflop/s)						Single Precision (Gflop/s)		
Matrix	Cell^{FSS}	Cell$^{pm}_{+}$	Cell^{pm}	X1E	AMD64	IA64	Cell^{pm}	AMD64	IA64
Vavasis	3.79	3.17	3.06	0.84	0.44	0.46	6.06	0.70	0.49
FEM	4.28	3.44	3.39	1.55	0.42	0.49	5.14	0.59	0.62
Mem	2.21	1.69	1.46	0.57	0.30	0.27	2.79	0.45	0.31
CFD	1.87	1.52	1.44	1.61	0.28	0.21	2.33	0.38	0.23
Average	**3.04**	**2.46**	**2.34**	**1.14**	**0.36**	**0.36**	**4.08**	**0.53**	**0.41**

	SPARSITY symmetric matrix suite								
	Double Precision (Gflop/s)						Single Precision (Gflop/s)		
Matrix	Cell^{FSS}	Cell$^{pm}_{+}$	Cell^{pm}	X1E	AMD64	IA64	Cell^{pm}	AMD64	IA64
FEM	—	6.79	6.32	3.12	0.93	1.14	12.37	1.46	1.37
3D Tube	—	6.48	6.06	2.62	0.86	1.16	11.66	1.36	1.31
Portfolio	—	1.83	1.60	2.99	0.37	0.24	3.26	0.42	0.32
NASA	—	1.92	1.66	3.30	0.42	0.32	3.17	0.46	0.40
Vibro	—	3.90	3.47	2.54	0.57	0.56	7.08	0.56	0.64
LP	—	5.17	4.87	1.27	0.47	0.63	8.54	0.55	0.92
Average	—	**4.35**	**4.00**	**2.64**	**0.60**	**0.67**	**7.68**	**0.80**	**0.83**

	Synthetic Matrices								
	Double Precision (Gflop/s)						Single Precision (Gflop/s)		
Matrix	Cell^{FSS}	Cell$^{pm}_{+}$	Cell^{pm}	X1E	AMD64	IA64	Cell^{pm}	AMD64	IA64
7pt_64 Stencil	2.20	1.44	1.29	—	0.30	0.29	2.61	0.51	0.32

SpMV - Future optimizations

- Auto-tuning
  - other parallelization strategies
  - BCSR (better for SIMD, worse for memory traffic)
  - other storage formats (DIA/JAG/etc…)
- Symmetry (currently only present in the performance model)
  - Easier to exploit in single precision & w/BCSR
  - Cache blocking limits benefit (~50%)
- Segmented Scan
  - Reduces loop overhead at the expense of nonzero processing time
  - Good if NZ/Row (within a cache block) is small
  - single segment(VL=1) would be beneficial
  - Make runtime decision for a given cache block
  - Complicated by presence of empty rows within a cache block

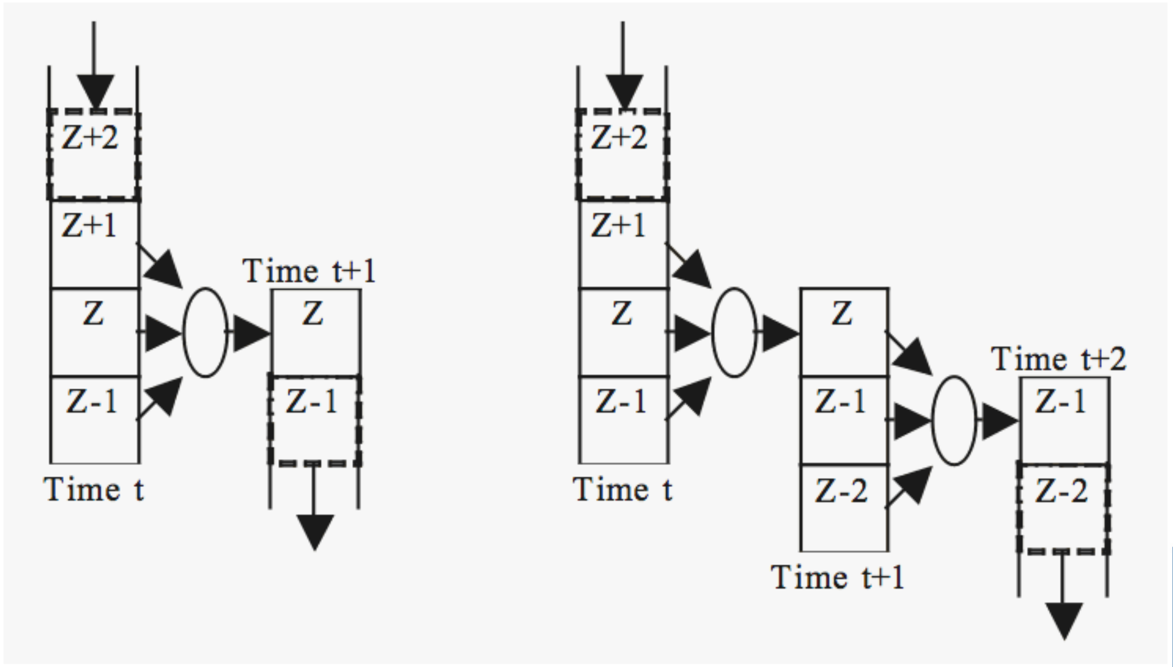
Stencil Computations on Structured Grids

Stencils on Structured Grids

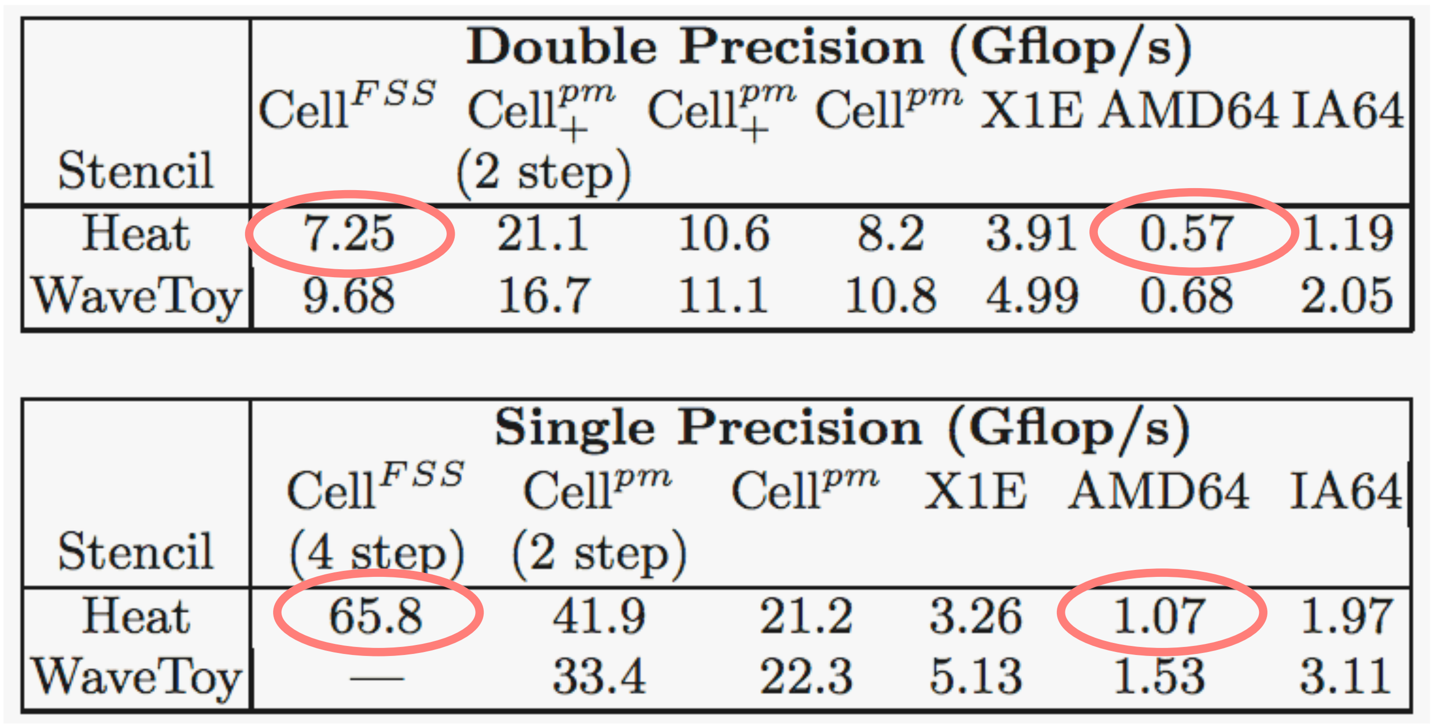
- Keep 4 planes in local store
  - (Z-1,t), (Z,t), (Z+1,t) -> (Z,t+1)
- Double buffer input and output planes
  - (Z+2,t) & (Z-1,t+1)
- Parallelization
  - Break middle loop up among SPEs
  - Maintains long DMAs and double buffering in Z direction
  - Computational intensity drops some
- SIMDization
  - Permutes required to pack left & right neighbors together into a SIMD register
- Each domain is bounded by a ghost zone
- Examined both CHOMBO heattut and CACTUS WaveToy equations

Stencils - Time Skewing

- Low computational intensity limits performance to be ~ memory bandwidth
- Time Skewing (with ghost zones) allows much higher performance



Stencils - Results



Stencil	Double Precision (Gflop/s)						
	Cell^{FSS}	Cell$^{pm}_{+}$ (2 step)	Cell$^{pm}_{+}$	Cell^{pm}	X1E	AMD64	IA64
Heat	7.25	21.1	10.6	8.2	3.91	0.57	1.19
WaveToy	9.68	16.7	11.1	10.8	4.99	0.68	2.05

Stencil	Single Precision (Gflop/s)					
	Cell^{FSS} (4 step)	Cell^{pm} (2 step)	Cell^{pm}	X1E	AMD64	IA64
Heat	65.8	41.9	21.2	3.26	1.07	1.97
WaveToy	—	33.4	22.3	5.13	1.53	3.11

Note:

Cell^{FSS}	**= Full System Simulator**
Cell^{pm}	**= Performance model**

(n step) = performs n steps of time skewing

Fast Fourier Transforms

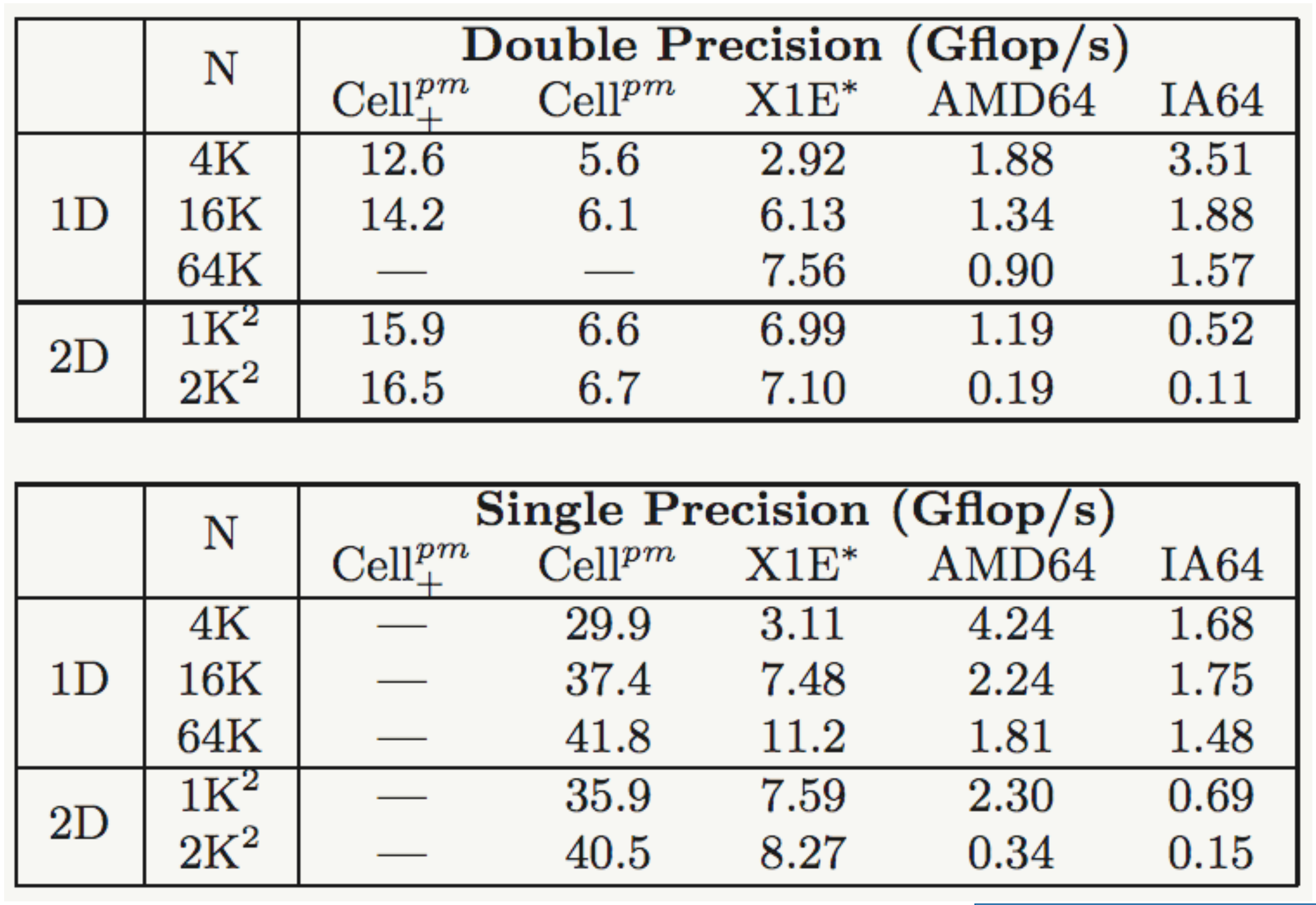
1D Fast Fourier Transforms

- Naïve Algorithm
  - Load roots of unity
  - Load data (cyclic)
  - Local work, on-chip transpose, local work
  - i.e. SPEs cooperate on a single FFT
  - No overlap of communication or computation

2D Fast Fourier Transforms

- Each SPE performs 2 * (N/8) FFTs
- Double buffer rows
  - overlap communication and computation
  - 2 incoming, 2 outgoing
- Straightforward algorithm (N^2 2D FFT):
  - N simultaneous FFTs, transpose,
    N simultaneous FFTs, transpose.
- Long DMAs necessitate transposes
- transposes represent about 50% of total SP execution time
- SP Simultaneous FFT's run at ~ 75 GFLOP/s

FFT - Results

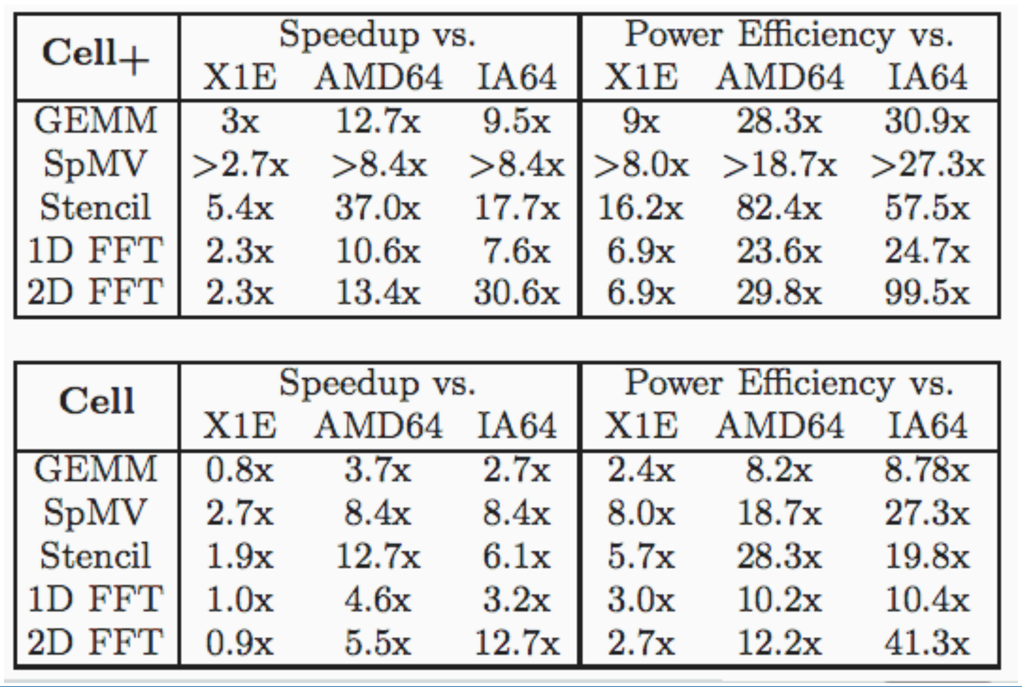


	N	Double Precision (Gflop/s)				
		Cell$^{pm}_{+}$	Cell^{pm}	X1E*	AMD64	IA64
1D	4K	12.6	5.6	2.92	1.88	3.51
	16K	14.2	6.1	6.13	1.34	1.88
	64K	—	—	7.56	0.90	1.57
2D	1K^2	15.9	6.6	6.99	1.19	0.52
	2K^2	16.5	6.7	7.10	0.19	0.11

	N	Single Precision (Gflop/s)				
		Cell$^{pm}_{+}$	Cell^{pm}	X1E*	AMD64	IA64
1D	4K	—	29.9	3.11	4.24	1.68
	16K	—	37.4	7.48	2.24	1.75
	64K	—	41.8	11.2	1.81	1.48
2D	1K^2	—	35.9	7.59	2.30	0.69
	2K^2	—	40.5	8.27	0.34	0.15

DP Performance/Efficiency Summary

- Existing Cell implementation is significantly faster than commodity(Itanium2/Opteron) processors in DP
- Power efficiency is currently more than 10x the commodity processors
- Cell+ would increase performance and efficiency significantly



Cell+	Speedup vs.			Power Efficiency vs.		
	X1E	AMD64	IA64	X1E	AMD64	IA64
GEMM	3x	12.7x	9.5x	9x	28.3x	30.9x
SpMV	>2.7x	>8.4x	>8.4x	>8.0x	>18.7x	>27.3x
Stencil	5.4x	37.0x	17.7x	16.2x	82.4x	57.5x
1D FFT	2.3x	10.6x	7.6x	6.9x	23.6x	24.7x
2D FFT	2.3x	13.4x	30.6x	6.9x	29.8x	99.5x

Cell	Speedup vs.			Power Efficiency vs.		
	X1E	AMD64	IA64	X1E	AMD64	IA64
GEMM	0.8x	3.7x	2.7x	2.4x	8.2x	8.78x
SpMV	2.7x	8.4x	8.4x	8.0x	18.7x	27.3x
Stencil	1.9x	12.7x	6.1x	5.7x	28.3x	19.8x
1D FFT	1.0x	4.6x	3.2x	3.0x	10.2x	10.4x
2D FFT	0.9x	5.5x	12.7x	2.7x	12.2x	41.3x

Acknowledgments

- This work (paper in CF06) is a collaboration with the following FTG members:
  - John Shalf, Lenny Oliker, Shoaib Kamil, Parry Husbands, and Kathy Yelick
- Additional thanks to
  - Joe Gebis and David Patterson
- X1E FFT numbers provided by:
  - Bracy Elton, and Adrian Tate