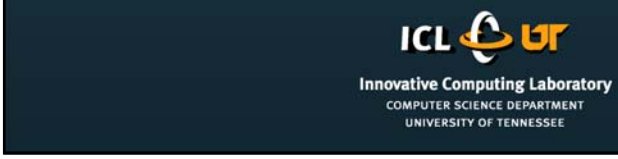
Sony/Toshiba/IBM (STI) CELL Processor

Scientific Computing for Engineers: Spring 2007



Innovative Computing Laboratory
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE

Three Performance-Limiting Walls

- **Power Wall**

  Increasingly, microprocessor performance is limited by achievable power dissipation rather than by the number of available integrated-circuit resources (transistors and wires). Thus, the only way to significantly increase the performance of microprocessors is to improve power efficiency at about the same rate as the performance increase.

- **Frequency Wall**

  Conventional processors require increasingly deeper instruction pipelines to achieve higher operating frequencies. This technique has reached a point of diminishing returns, and even negative returns if power is taken into account.

- **Memory Wall**

  On multi-gigahertz symmetric multiprocessors – even those with integrated memory controllers – latency to DRAM memory is currently approaching 1,000 cycles. As a result, program performance is dominated by the activity of moving data between main storage (the effective-address space that includes main memory) and the processor.

The Memory Wall

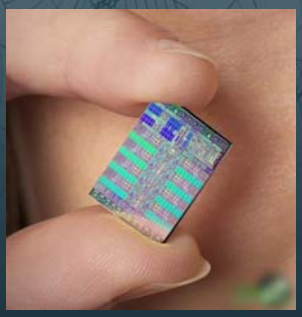
"When a sequential program on a conventional architecture performs a load instruction that misses in the caches, program execution now comes to a halt for several hundred cycles. [...] Even with deep and costly speculation, conventional processors manage to get at best a handful of independent memory accesses in flight. The result can be compared to a bucket brigade in which a hundred people are required to cover the distance to the water needed to put the fire out, but only a few buckets are available."

H. Peter Hofstee
"Cell Broadband Engine Architecture from 20,000 feet"
http://www-128.ibm.com/developerworks/power/library/pa-cbea.html

Their (multicore) low cost does not guarantee their effective use in HPC. This relates back to the data-intensive nature of most HPC applications and the sharing of already limited bandwidth to memory. The stream benchmark performance of Intel's new Woodcrest dual core processor illustrates this point. [...] Much effort was put into improving Woodcrest's memory subsystem, which offers a total of over 21 GBs/sec on nodes with two sockets and four cores. Yet, four-threaded runs of the memory intensive Stream benchmark on such nodes that I have seen extract no more than 35 percent of the available bandwidth from the Woodcrest's memory subsystem."

Richard B. Walsh
"New Processor Options for HPC"
http://www.hpcwire.com/hpc/906849.html

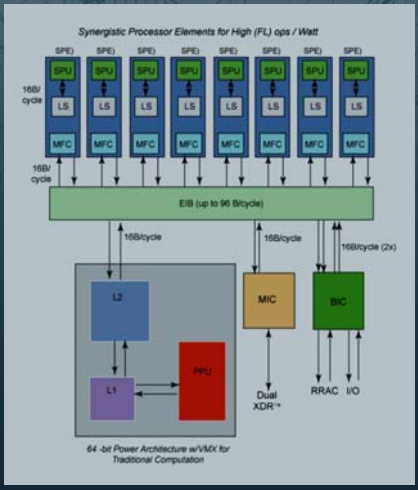
CELL Overview



- $400 million over 5 years
- Sony/Toshiba/IBM alliance known as STI
- STI Design Center – Austin, Texas – March 2001
- Mercury Computer Systems Inc. – dual CELL blades
- Cell Broadband Engine Architecture / CBEA / CELL BE

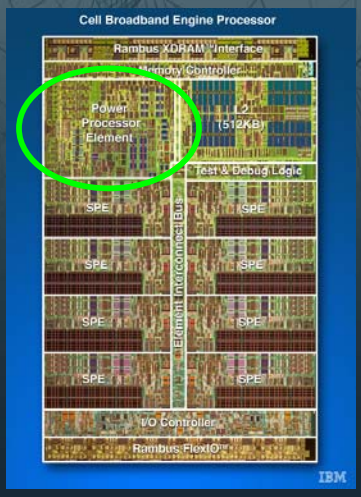
- Playstation3, dual CELL blades, PCI accelerator cards
- 3.2 GHz, 90nm OSI,
- 234 million transistors
  - 165 million – Xbox 360
  - 220 million – Itanium 2 (2002)
  - 1,700 million – Dual-Core Itanium 2 (2006)

CELL Architecture



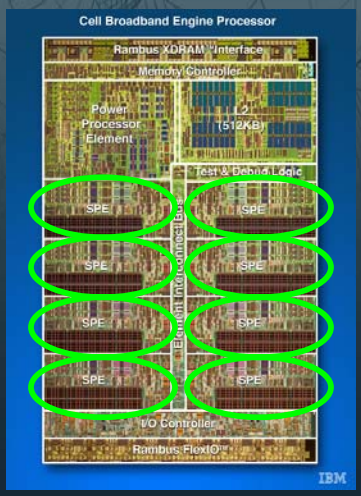
- PPU – PowerPC 970 core
- SPE – Synergistic Processing Element
  - SPU – Synergistic Processing Unit
  - LS – Local Store
  - MFC – Memory Flow Controller
- EIB – Element Interconnection Bus
- MIC – Memory Interface Controller

Power Processing Element



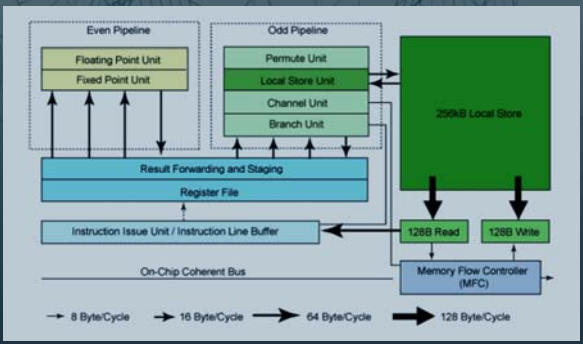
- Power Processing Element (PPE)
  - Power 970 architecture compliant
  - 2-way Symmetric Multithreading (SMT)
  - 32KB Level 1 instruction cache
  - 32KB level 1 data cache
  - 512KB level 2 cache
  - VMX (AltiVec) with 32 128-bit vector registers
  - standard FPU
    - fully pipelined DP with FMA
    - 6.4 Gflop/s DP at 3.2 GHz
  - AltiVec
    - no DP
    - 4-way fully pipelined SP with FMA
    - 25.6 Gflop/s SP at 3.2 GHz

Synergistic Processing Elements



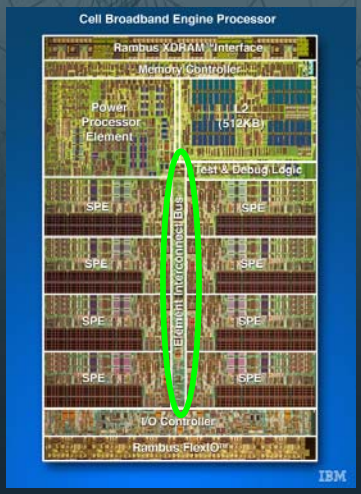
- Synergistic Processing Elements (SPEs)
  - 128-bit SIMD
  - 128 vector registers
  - 256KB instruction and data local memory
  - Memory Flow Controller (MFC)
  - 16-way SIMD (8-bit integer)
  - 8-way SIMD (16-bit integer)
  - 4-way SIMD (32-bit integer, single prec. FP)
  - 2-way SIMD (64-bit double prec. FP)
  - 25.6 Gflop/s SP at 3.2 Ghz (fully pipelined)
  - 1.8 Gflop/s DP at 3.2 Ghz (7 cycle latency)

SPE Architecture



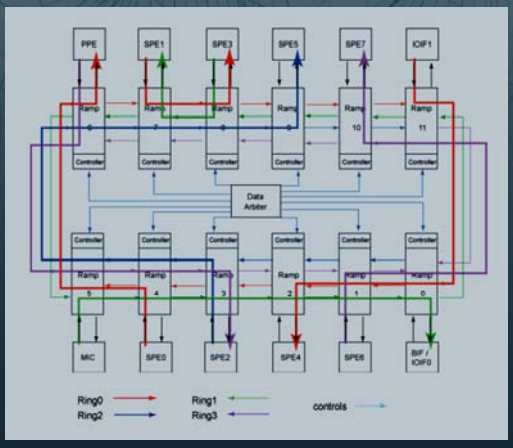
- Dual issue (in order) pipeline
  - Even – arithmetic
    - integer
    - floating point
  - Odd – data motion
    - permutations
    - local store
    - branches
    - channel

Element Interconnection Bus



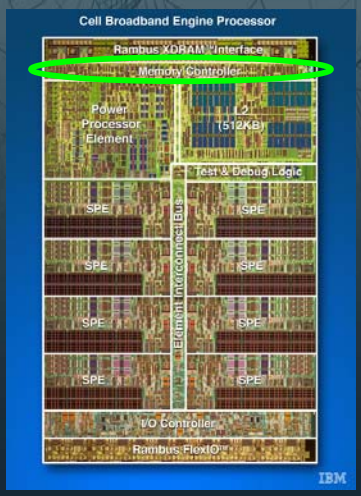
- Element Interconnection Bus (EIB)
  - 4 16B-wide unidirectional channels
  - half the system clock (1.6GHz)
  - 204.8 GB/s bandwidth (arbitration)

Element Interconnection Bus



- Element Interconnection Bus (EIB)
  - 4 16B-wide unidirectional channels
  - half the system clock (1.6GHz)
  - 204.8 GB/s bandwidth (arbitration)

Main Memory System



- Memory Interface Controller (MIC)
  - external dual XDR,
    - 3.2 Ghz max effective frequency, (max 400 MHz, Octal Data Rate),
    - each: 8 banks → max 256 MB,
    - total: 16 banks → max 512 MB,
  - 25.6 GB/s.

CELL Performance – Double Precision

In **double precision**

- every seven cycles each SPE can:
  - process a two element vector,
  - perform two operations on each element.

- in one cycle the FPU on the PPE can:
  - process one element,
  - perform two operations on the element.

8 x 2 x 2 x 3.2 GHz / 7 = *14.63* Gflop/s
2 x 3.2 GHz = *6.4* Gflop/s

21.03 Gflop/s

CELL Performance – Single Precision

In **single precision**

- in one cycle each SPE can:
  - process a four element vector,
  - perform two operations on each element.

- in one cycle the VMX on the PPE can:
  - process a four element vector,
  - perform two operations on each element.

8 x 4 x 2 x 3.2 GHz = *204.8* Gflop/s
4 x 2 x 3.2 GHz = *25.6* Gflop/s

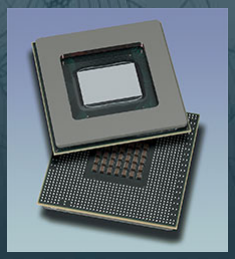
230.4 Gflop/s

CELL Performance – Bandwidth

Bandwidth:

- 3.2 GHz clock:
  - each SPU – **25.6** GB/s, (compare to **25.6** Gflop/s per SPU)
  - Main memory – **25.6** GB/s,
  - EIB – **204.8** GB/s. (compare to **204.8** Gflop/s – 8 SPUs)

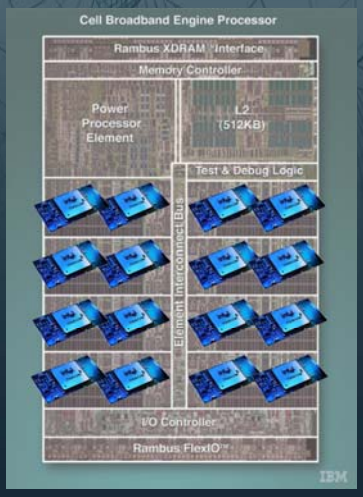
Performance Comparison – Double Precision



1.6 GHz Dual-Core Itanium 2
- 1.6 x 4 x 2 = **12.8** Gflop/s

3.2 GHz CELL BE (SPEs only)
- 3.2 x 8 x 8 = **14.6** Gflop/s

Performance Comparison – Single Precision



1.6 GHz Dual-Core Itanium 2
- 1.6 x 4 x 2 = **12.8** Gflop/s

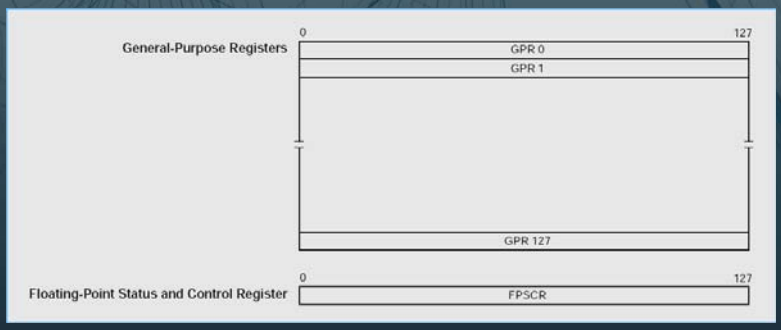
3.2 GHz SPE
- 3.2 x 8 = **25.6** Gflop/s
- One SPE = 2 Dual-Core Itaniums 2

3.2 GHz CELL BE (SPEs only)
- 3.2 x 8 x 8 = **204.8** Gflop/s
- One CBE = 16 Dual-Core Itaniums 2

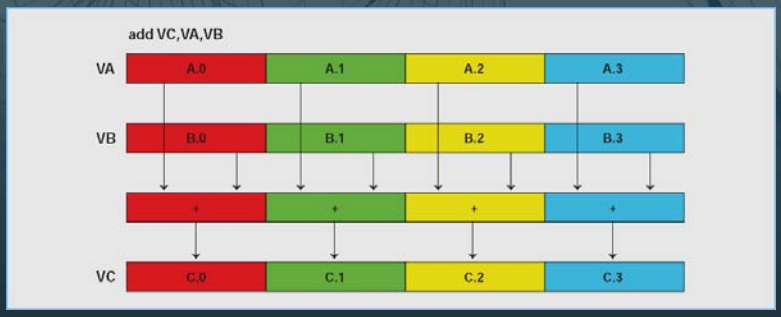
CELL Programming Basics

- Programming the SPUs
  - SIMD'ization (vectorization)
- Communication
  - DMAs
  - Mailboxes
- Measuring Performance
  - SPU decrementer

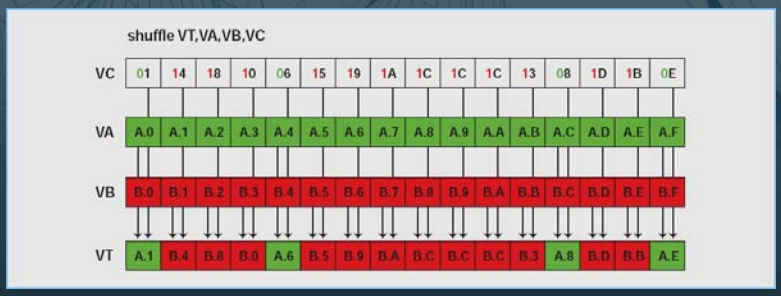
SPE Register File



SPE SIMD Arithmetic



SPE SIMD Data Motion



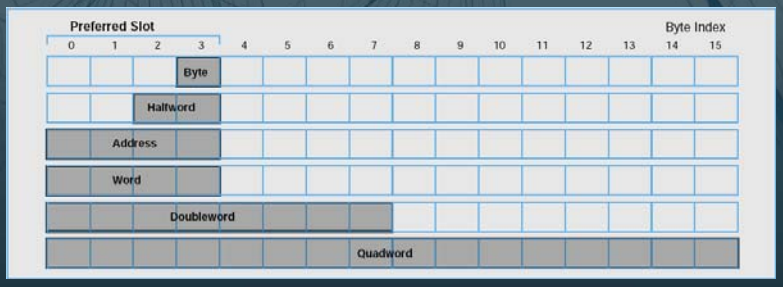
SPE SIMD Vector Data Types

Vector Data Type	Content
vector unsigned char	Sixteen 8-bit unsigned chars
vector signed char	Sixteen 8-bit signed chars
vector unsigned short	Eight 16-bit unsigned halfwords
vector signed short	Eight 16-bit signed halfwords
vector unsigned int	Four 32-bit unsigned words
vector signed int	Four 32-bit signed words
vector unsigned long long	Two 64-bit unsigned doublewords
vector signed long long	Two 64-bit signed doublewords
vector float	Four 32-bit single-precision floats
vector double	Two 64-bit double precision floats
qword	quadword (16-byte)

SPE SIMD Arithmetic Intrinsics

Arithmetic Intrinsics	
d = spu_add(a, b)	Vector add
d = spu_addx(a, b, c)	Vector add extended
d = spu_genb(a, b)	Vector generate borrow
d = spu_genbx(a, b, c)	Vector generate borrow extended
d = spu_genc(a, b)	Vector generate carry
d = spu_gencx(a, b, c)	Vector generate carry extended
d = spu_madd(a, b, c)	Vector multiply and add
d = spu_mhhadd(a, b, c)	Vector multiply high high and add
d = spu_msub(a, b, c)	Vector multiply and subtract

SPE Scalar Processing



SPE Static Code Analysis

```
                                                    .L19:
0D                                            78      a       $49,$8,$10
1D 012                                        789     lqx     $51,$6,$9
0D                                             89     ila     $47,66051
1D 0123                                        89     lqx     $52,$6,$11
0  0                                            9     ai      $7,$7,-1
0  ----456789                                         fma     $50,$51,$12,$52
1       -----012345                                   stqx    $50,$6,$11
1            123456                                   lqx     $48,$8,$10
0D            23                                      ai      $8,$8,4
1D            234567                                  lqa     $44,ctx+16
1              345678                                 lqx     $43,$6,$9
1               ---7890                               rotqby  $46,$48,$49
1                  ---1234                            shufb   $45,$46,$46,$47
0                     ---567890                       fm      $42,$12,$45
0d                       -----123456                  fma     $41,$42,$44,$43
1d                            ------789012             stqx    $41,$6,$9
0D                                  89                ai      $6,$6,16
                                                    .L39:
1D                                  8901                brnz    $7,.L19
```

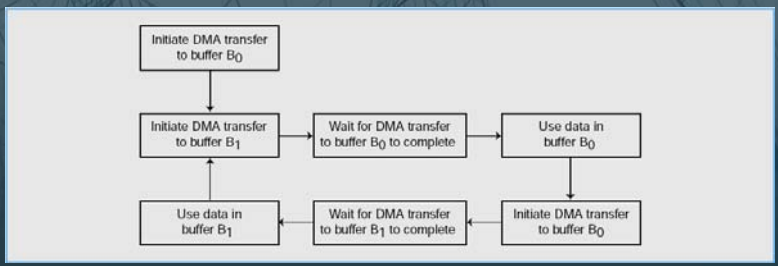
SPE DMA Commands

DMA Commands	
mfc_put(ls, ea, size, tag, tid, rid)	Move data from local storage to effective address
mfc_putb(ls, ea, size, tag, tid, rid)	Move data from local storage to effective address with barrier
mfc_putf(ls, ea, size, tag, tid, rid)	Move data from local storage to effective address with fence
mfc_get(ls, ea, size, tag, tid, rid)	Move data from effective address to local storage
mfc_getb(ls, ea, size, tag, tid, rid)	Move data from effective address to local storage with barrier
mfc_getf(ls, ea, size, tag, tid, rid)	Move data from effective address to local storage with fence
List DMA Commands	
mfc_putl(ls, ea, list, list_size, tag, tid, rid)	Move data from local storage to effective address using MFC list
mfc_putlb(ls, ea, list, list_size, tag, tid, rid)	Move data from local storage to effective address using MFC list with barrier
mfc_putlf(ls, ea, list, list_size, tag, tid, rid)	Move data from local storage to effective address listing MFC list with fence
mfc_getl(ls, ea, list, list_size, tag, tid, rid)	Move data from effective address to local storage using MFC list
mfc_getlb(ls, ea, list, list_size, tag, tid, rid)	Move data from effective address to local storage using MFC list with barrier
mfc_getlf(ls, ea, list, list_size, tag, tid, rid)	Move data from effective address to local storage using MFC list with fence

SPE DMA Status

DMA Status	
mfc_stat_ cmd_queue()	Check number of available entries in MFC DMA queue
mfc_write_tag_mask(mask)	Set tag mask to select tag groups to be included in query operation
mfc_read_tag_mask()	Read tag mask indicating groups to be included in query operation
mfc_write_tag_update(ts)	Request the tag status to be updated
mfc_write_tag_update_immediate()	Request that tag status be updated immediately
mfc_write_tag_update_any()	Request that tag status be updated when any tag groups complete
mfc_write_tag_update_all()	Request that tag status be updated when all tag groups complete
mfc_stat_tag_update()	Check availability of tag Update Request Status channel
mfc_read_tag_status()	Wait for an updated tag status
mfc_read_tag_status_immediate()	Wait for the updated tag status of any enabled group
mfc_read_tag_status_any()	Wait for no outstanding operations for any enabled groups
mfc_read_tag_status_all()	Wait for no outstanding operations for all enabled groups
mfc_stat_tag_status()	Check availability of MFC_RdTagStat channel
mfc_read_list_stall_status()	Read list DMA stall-and-notify status
mfc_stat_list_stall_status()	Check availability of List DMA stall-and-notify status
mfc_write_list_stall_ack(tag)	Acknowledge tag group containing stalled DMA list commands
mfc_read_atomic_status()	Check availability of atomic command status

DMA Double Buffering




```
Prologue
Receive tile 1
Receive tile 2
Compute tile 1
Swap buffers

Epilogue
Send tile N-1
Compute tile N
Send tile N1
```

```
Loop body
FOR I=2 TO N-1
    Send tile I-1
    Receive tile I+1
    Compute tile I
    Swap buffers
END FOR
```


SPE Mailboxes

SPU Mailboxes	
spu_read_in_mbox()	Read next data entry in the SPU Inbound Mailbox
spu_stat_in_mbox()	Get the number of data entries in the SPU Inbound Mailbox
spu_write_out_mbox(data)	Send data to the SPU Outbound Mailbox
spu_stat_out_mbox()	Get the available capacity of the SPU Outbound Mailbox
spu_write_out_intr_mbox(data)	Send data to the SPU Outbound Interrupt Mailbox
spu_stat_out_intr_mbox()	Get the available capacity of the SPU Outbound Interrupt Mailbox

- FIFO queues
- 32-bit messages
- Intended for mainly for communication between the PPE and the SPEs

SPE Decrementer

SPU Decrementer	
spu_read_decrementer()	Read the current value of the decrementer
spu_write_decrementer(count)	Load a value into the decrementer

- 14MHz – IBM dual CELL blade
- 80MHz – Sony Playstation3

CELL Basic Coding Tips

- Local Store
  - Keep in mind the Local Store is 256KB in size
  - Use plug-ins to handle larger codes
- DMA Transfers
  - Use SPE-initiated DMA transfers
  - Use double-buffering to hide transfers
  - Use fence and barrier to order transfers
- Loops
  - Unroll loops to reduce dependency stalls, increase dual-issue rate, and exploit SPU large register file
- Branches
  - Eliminate non-predicted branches
- Dual-Issue
  - Choose intrinsics to maximize dual-issue

UT CELL BE Cluster

Ashe.CS.UTK.EDU

UT CELL BE Cluster – Historical Perspective

Connection Machine CM-5 (512 CPUs)

- 512 x 128 = **65** Gflop/s DP

Playstation3 (4 units)

- 4 x 17 = **68** Gflop/s DP