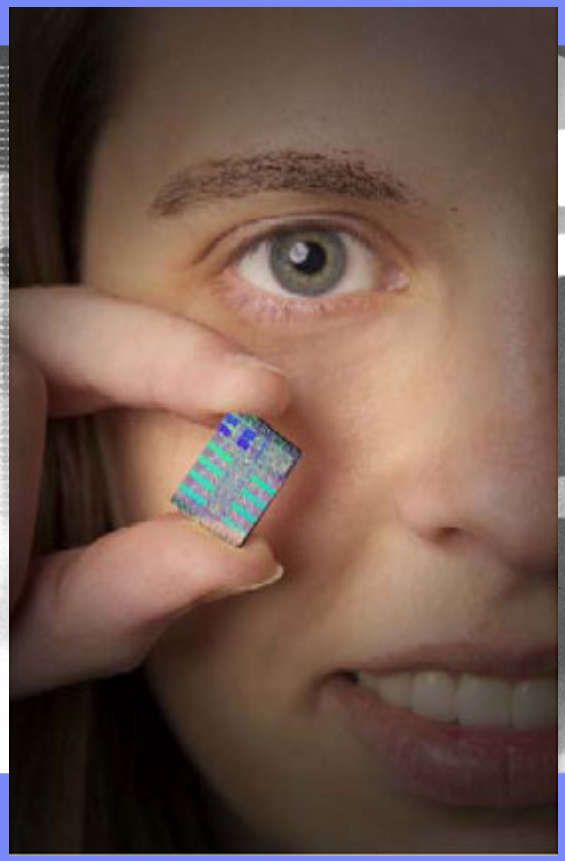
IBM Systems & Technology Group

An Introduction to the Cell Broadband Engine Architecture



Owen Callanan
IBM Ireland, Dublin Software Lab.
Email: owen.callanan@ie.ibm.com

Agenda

- **About IBM Ireland development**
- **Cell Introduction**
- **Cell Architecture**
- **Programming the Cell**
  - SPE programming
  - Controlling data movement
  - Tips and tricks

Introduction - Development in IBM Ireland

- **Lotus Software Research**
- **Tivoli**
- **Industry Assets & Models**
- **RFID Center of Competence**
- **High Performance Computing**

Introduction – HPC in IBM Dublin Lab

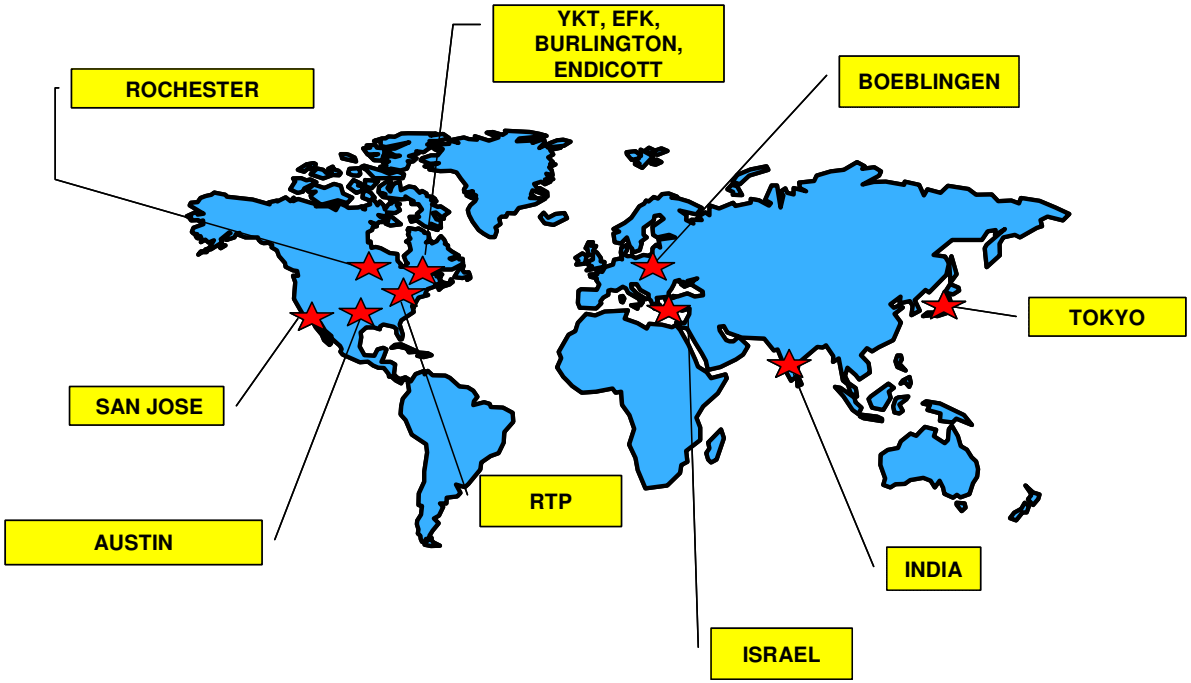
- **Blue Gene team**
- **Deep Computing Visualization (DCV)**
- **Dynamic Application Virtualization (DAV)**
- **Other areas also**
  - E.g. RDMA communications systems

Cell Introduction

Cell Broadband Engine History

- **IBM, SCEI/Sony, Toshiba Alliance formed in 2000**
- **Design Center opens March 2001**
- **~$400M Investment, 5 years, 600 people**
- **February 7, 2005: First technical disclosures**
- **January 12, 2006: Alliance extended 5 additional years**

SONY COMPUTER ENTERTAINMENT



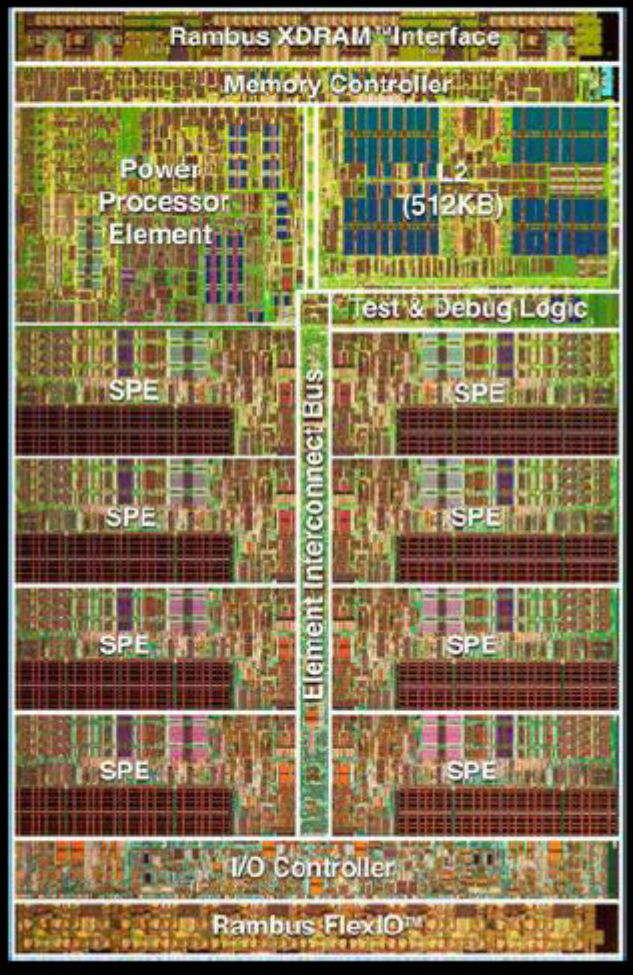
TOSHIBA

IBM

First implementation of Cell B.E. Architecture

Highlights (3.2 GHz)

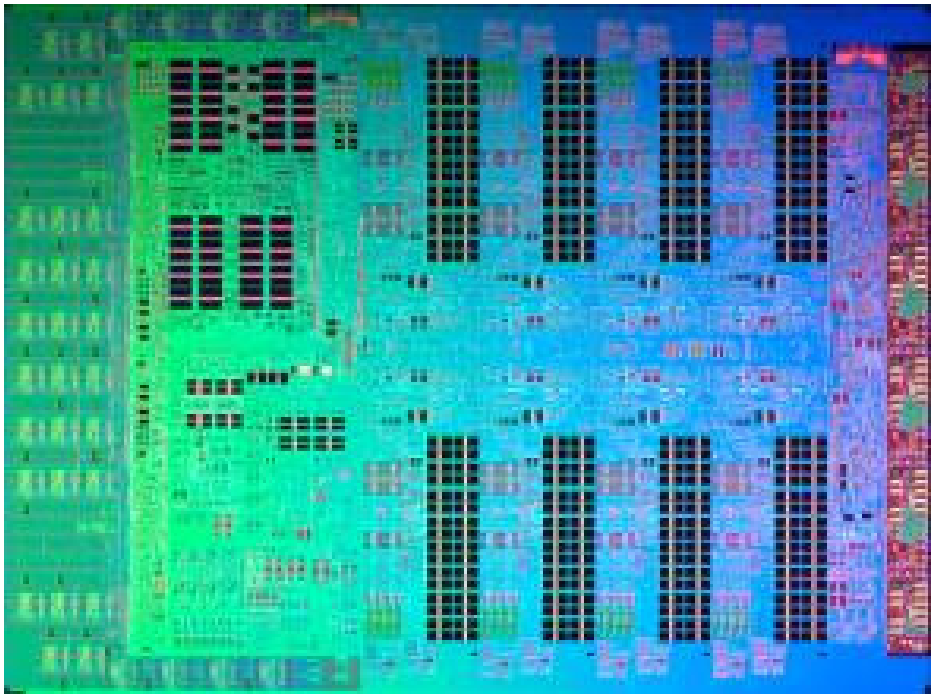
- 241M transistors
- 235mm2
- 9 cores, 10 threads
- >200 GFlops (SP)
- >20 GFlops (DP)
- Up to 25 GB/s memory B/W
- Up to 75 GB/s I/O B/W
- >300 GB/s EIB
- Top frequency >4GHz
  (observed in lab)



IBM PowerXCell™ 8i – QS22

PowerXCell 8i processor

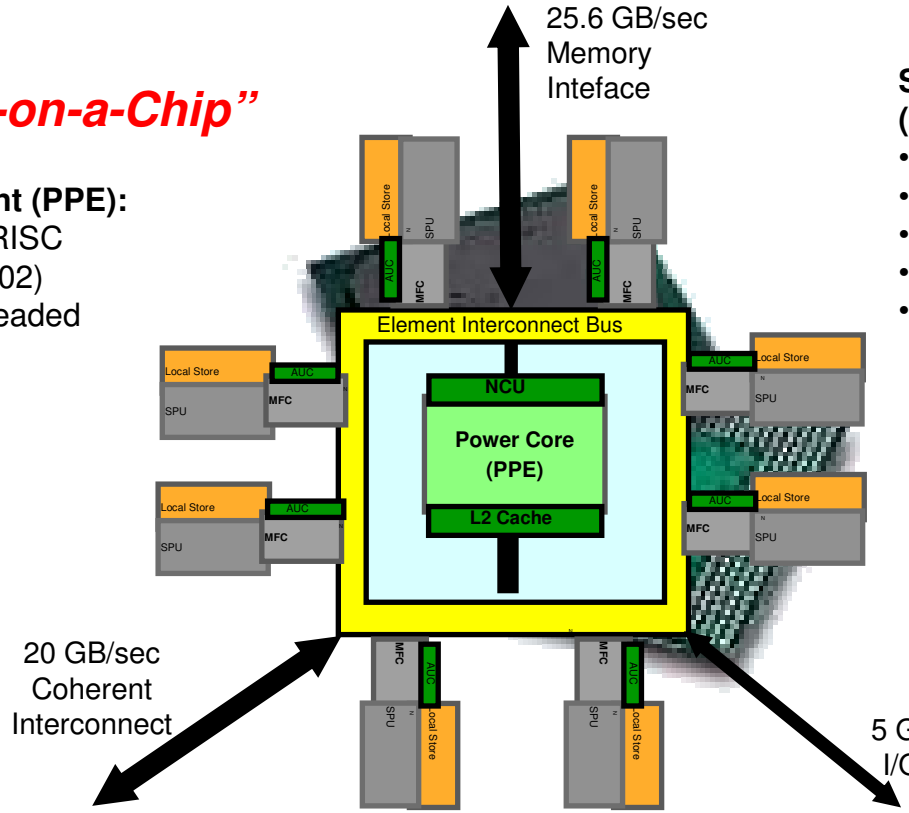
- **65 nm**
- **9 cores, 10 threads**
- **230.4 GFlops peak (SP) at 3.2GHz**
- **108.8 GFlops peak (DP) at 3.2GHz**
- **Up to 25 GB/s memory bandwidth**
- **Up to 75 GB/s I/O bandwidth**



Cell Processor

"Supercomputer-on-a-Chip"

Power Processor Element (PPE):
- General Purpose, 64-bit RISC Processor (PowerPC 2.02)
- 2-Way Hardware Multithreaded
- L1 : 32KB I ; 32KB D
- L2 : 512KB
- Coherent load/store
- VMX
- 3.2 GHz



Synergistic Processor Elements (SPE):
- 8 per chip
- 128-bit wide SIMD Units
- Integer *and* Floating Point capable
- 256KB Local Store
- Up to 25.6 GF/s per SPE --- 200GF/s total *

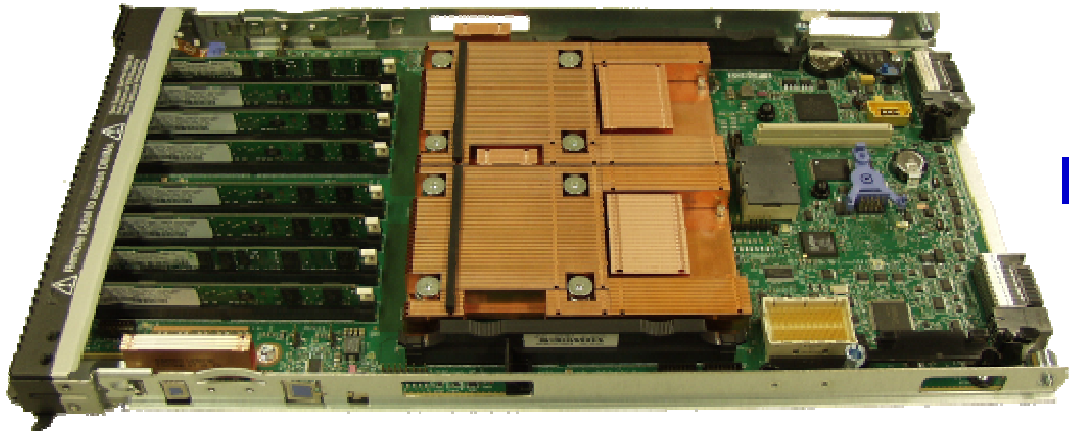
Internal Interconnect:
- Coherent ring structure
- 300+ GB/s total internal interconnect bandwidth
- DMA control to/from SPEs supports >100 outstanding memory requests

External Interconnects:
- 25.6 GB/sec BW memory interface
- 2 Configurable I/O Interfaces
  - Coherent interface (SMP)
  - Normal I/O interface (I/O & Graphics)
  - Total BW configurable between interfaces
  - Up to 35 GB/s out
  - Up to 25 GB/s in

Memory Management & Mapping
- SPE Local Store aliased into PPE system memory
- MFC/MMU controls SPE DMA accesses
  - Compatible with PowerPC Virtual Memory architecture
  - S/W controllable from PPE MMIO
- Hardware or Software TLB management
- SPE DMA access protected by MFC/MMU

IBM BladeCenter QS21

- **Announcement: August 28, 2007**

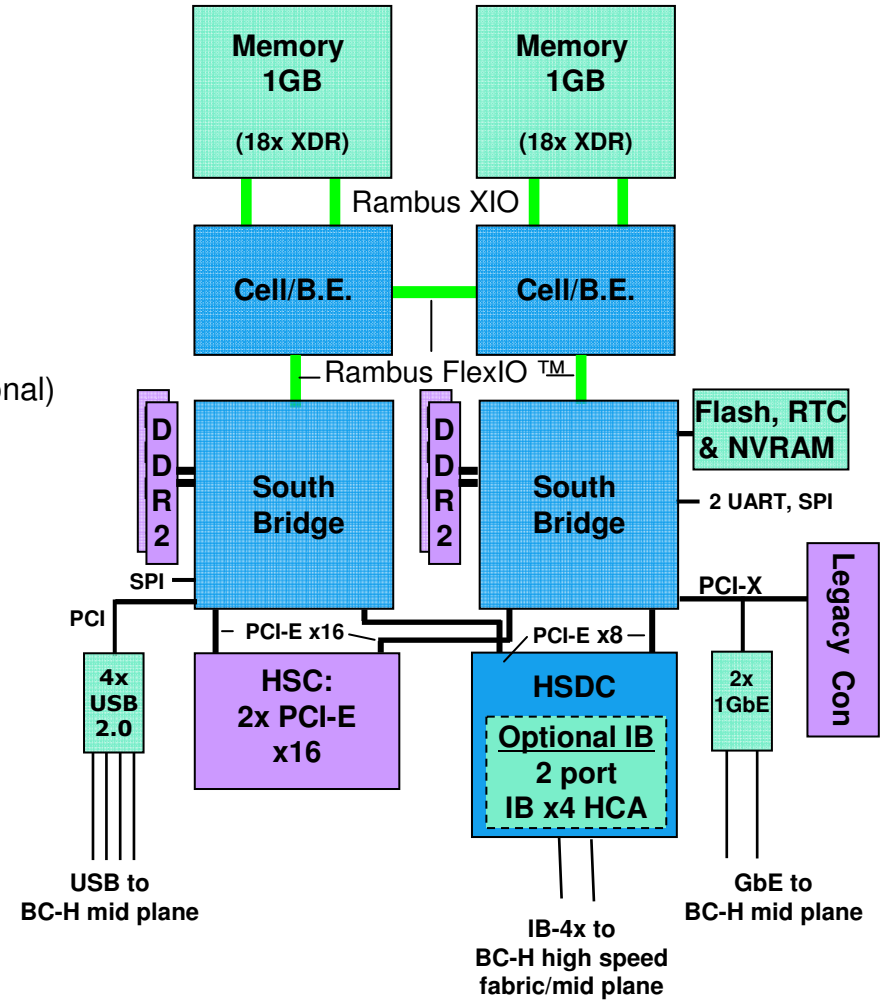


IBM BladeCenter QS22

- **Announcement: May 13, 2008**

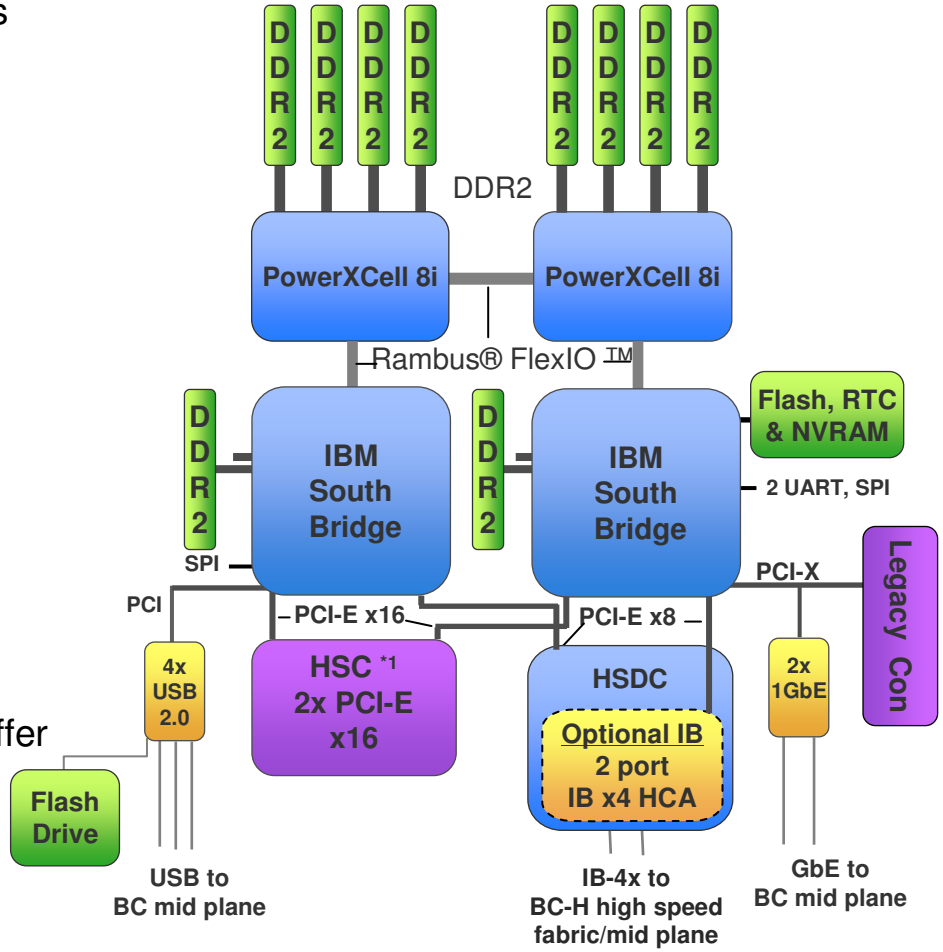
IBM BladeCenter QS21

- **Core Electronics**
  - Dual 3.2GHz Cell/B.E. Processor Configuration
  - 2GB XDRAM (1GB per processor)
  - Dual Gigabit Ethernet (GbE) controllers
  - Single-wide blade (uses 1 BladeCenter H slot)
  - Infiniband 4x channel adapters / (optional)
    - Cisco Systems 4X InfiniBand HCA Expansion Card for BladeCenter (32R1760)
  - Serial Attached SCSI (SAS) daughter card (39Y9190) / (optional)
- **BC Chassis Configuration**
  - Standard IBM BladeCenter H
  - Max. 14 QS21 per chassis
  - 2 Gigabit Ethernet switches
  - External IB switches required for IB option
    - Cisco Systems 4X InfiniBand Switch Module for BladeCenter (32R1756)
- **Peak Performance**
  - Up to 460 GFLOPS per blade
  - Up to 6.4 TFLOPS (peak) in a single BladeCenter H chassis
  - Up to 25.8 TFLOPS in a standard 42U rack



BladeCenter® QS22 – PowerXCell 8i

- Core Electronics
  - Two 3.2GHz PowerXCell 8i Processors
  - SP: 460 GFlops peak per blade
  - DP: 217 GFlops peak per blade
  - Up to 32GB DDR2 800MHz
  - Standard blade form factor
  - Support BladeCenter H chassis
- Integrated features
  - Dual 1Gb Ethernet (BCM5704)
  - Serial/Console port, 4x USB on PCI
- Optional
  - Pair 1GB DDR2 VLP DIMMs as I/O buffer (2GB total) (46C0501)
  - 4x SDR InfiniBand adapter (32R1760)
  - SAS expansion card (39Y9190)
  - 8GB Flash Drive (43W3934)



*The HSC interface is not enabled on the standard products. This interface can be enabled on "custom" system implementations for clients by working with the Cell services organization in IBM Industry Systems.

IBM to Build World's First Cell Broadband Engine™ Based Supercomputer

Revolutionary Hybrid Supercomputer at Los Alamos National Laboratory Will Harness Cell PowerXCell 8i and AMD Opteron™ Technology

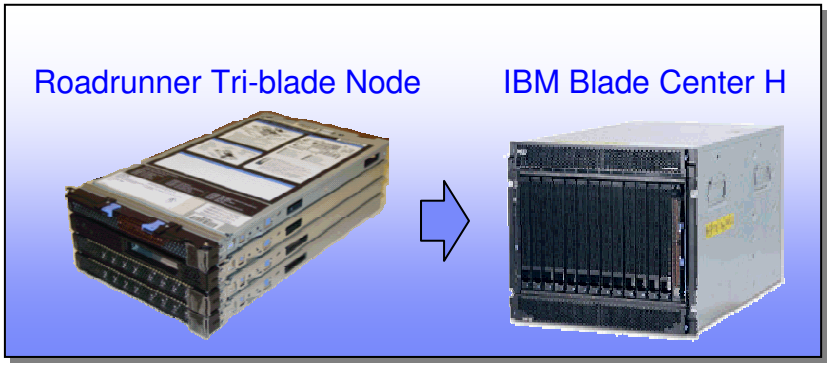
x86 Linux Master Nodes
AMD Opteron™
6,912 Dual Core chips
~50 TFlop/s
55 TB Memory

PowerXCell 8i Accelerators
12,960 PowerXCell 8i chips
~1,330 TFlop/s
52 TB Memory

- **Goal 1 PetaFlop Double Precision Floating Point *Sustained***
  - *1.4 PetaFlop Peak DP Floating point (2.8 SP)*
  - *216 GB/s sustained File System I/O:*
  - *107 TB of Memory*
  - *296 server racks that take up around 5,000 square feet- 3.9 MW power*
  - *Hybrid of Opteron X64 AMD processors (System x3755 servers) and Cell BE Blade Servers connected via high speed network*
  - *Modular Approach means the Master Cluster could be made up of Any Type System – including Power, Intel*

Los Alamos NATIONAL LABORATORY

Roadrunner Building Blocks



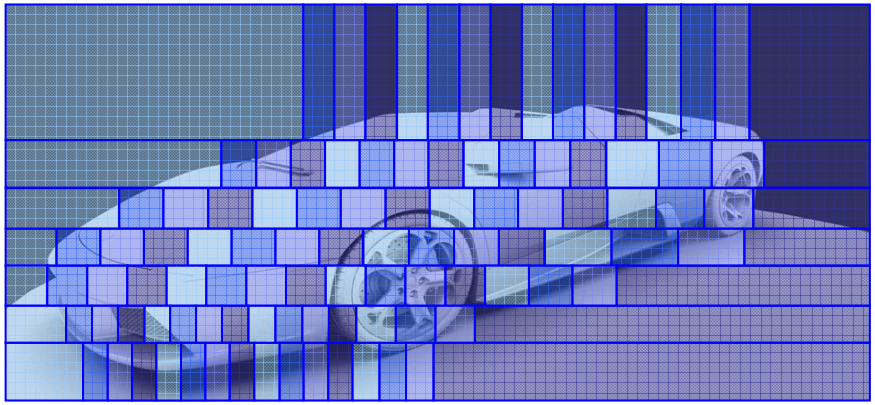
Roadrunner At a Glance

- **Cluster of 18 Connected Units**
  - 6,912 AMD dual-core Opterons
  - 12,960 IBM Cell eDP accelerators
  - 49.8 Teraflops peak (Opteron)
  - 1.33 Petaflops peak (Cell eDP)
  - 1PF sustained Linpack
- **InfiniBand 4x DDR fabric**
  - 2-stage fat-tree; all-optical cables
  - Full bi-section BW within each CU
    - 384 GB/s (bi-directional)
  - Half bi-section BW among CUs
    - 3.45 TB/s (bi-directional)
  - Non-disruptive expansion to 24 CUs
- **80 TB aggregate memory**
  - 28 TB Opteron
  - 52 TB Cell
- **216 GB/s sustained File System I/O**
  - 216x2 10G Ethernets to Panasas
- **RHEL & Fedora Linux**
- **SDK for Multicore Acceleration**
- **xCAT Cluster Management**
  - System-wide GigEnet network
- **3.9 MW Power**
  - 0.35 GF/Watt
- **Area**
  - 296 racks
  - 5500 ft^2

IBM iRT - Interactive Ray-tracer

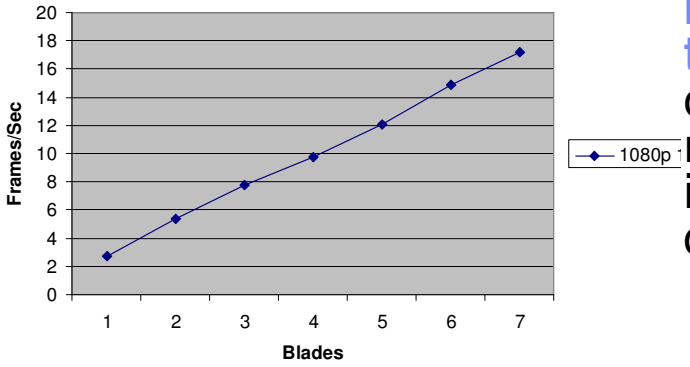
Ray Trace Images in Interactive Time

SPE Partitions



GDC 07 Configuration, Seven QS20 Blades, 3.2 TFLOPS

iRT Blade Performance Scaling



Composite image including **reflection, transparency, detailed shadows, ambient occlusion, and 4x4 jittered multi-sampling, totaling up to 288 rays per pixel.** The car model is comprised of 1.6 million polygons and the image resolution is 1080p hi-def.

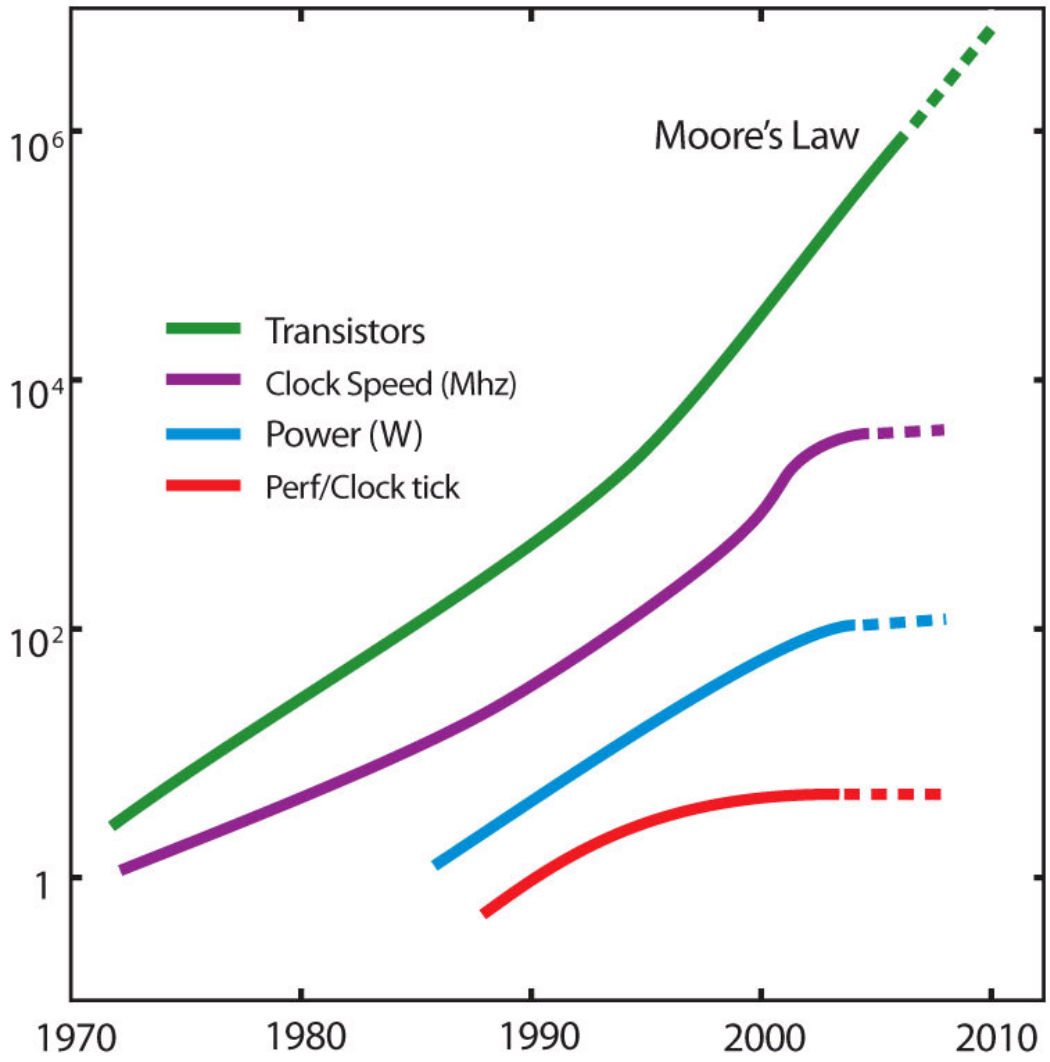
Demo

Also Shown at GDC March 2007

- **PS3 head node Leveraging seven QS20 blades**
- **3 Million Triangle Urban Landscape**
- **>100 Textures**
- **1080p HDTV**

Cell Architecture

The new reality of microprocessors



Moore's law, the doubling of transistors on a chip every 18 months (top curve), continues unabated. Three other metrics impacting computer performance (bottom three curves) have leveled off since 2002: the maximum electric power a microprocessor can use, the clock speed, and performance (operations) per clock tick.

"The biggest reason for the leveling off is the heat dissipation problem. With transistors at 65-nanometer sizes, the heating rate would increase 8 times whenever the clock speed was doubled, outstripping our ability to carry the heat away by standard air cooling." -Ken Koch

Three major limiters to processor performance

- **Frequency Wall**
  - Diminishing returns from deeper pipelines
- **Memory Wall**
  - Processor frequency vs. DRAM memory latency
  - Latency introduced by multiple levels of memory
- **Power Wall**
  - Limits in CMOS technology
  - Hard limit to acceptable system power

Techniques for Better Efficiency

- **Chip level multi-processors**
  - Go back to simpler core designs and use the extra available chip area for multiple cores.
- **Vector Units/SIMD**
  - Have the same instruction execution logic operate on multiple pieces of data concurrently; allows for more throughput with little increase in overhead.
- **Rethink memory organization**
  - At the register level hidden registers with register renaming is transparent to the end user but expensive to implement. Can a compiler do better with more instruction set architected registers?
  - Caches are automatic but not necessarily the most efficient use of chip area. Temporal and spatial locality mean little when processing streams of data.

Cell BE Design Highlights

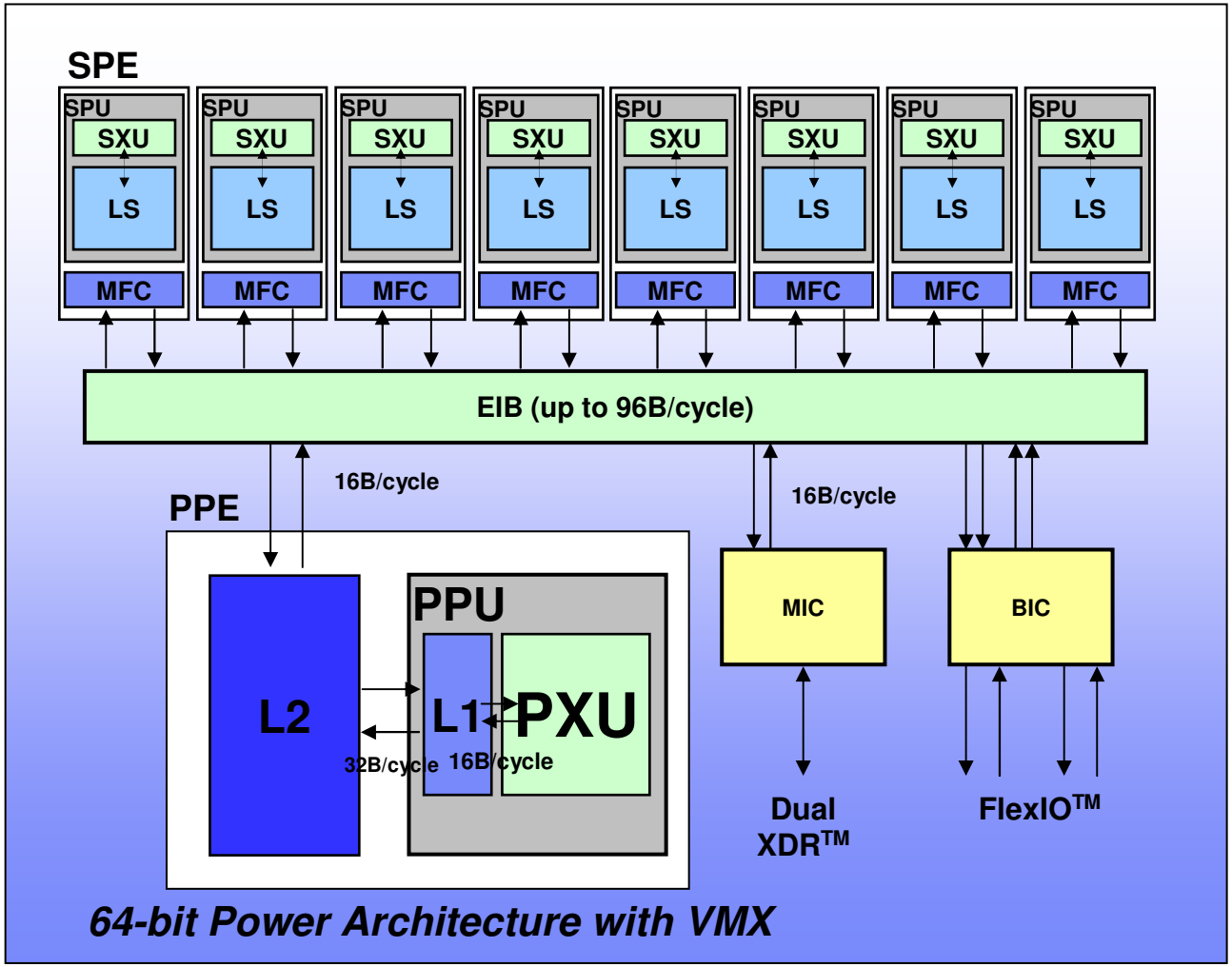
- **Increase concurrency**
  - Heterogeneous multiple cores processor
  - SIMD/Vector operations in a core

    ➔ allow for high frequencies
  - Start memory movement early so that memory is available when needed
- **Increase efficiency**
  - Simpler cores devote more resources to actual computation
  - Programmer managed memory is more efficient than dragging data through caches
  - Large register files give the compiler more flexibility and eliminate transistors needed for register renaming
  - Specialize processor cores for specific tasks

    ➔ efficient support of various class of applications

Cell BE Architecture

Cell BE Block Diagram



PPE

- **Dual-thread PowerPC processor**
  - Not a dual core, bit like a hyperthreaded Pentium 4
- **Conventional cache memory system**
  - 64 kByte level 1, 512 kByte level 2
- **PPE is not very powerful**
  - It runs the OS and manages the calculation
  - Must use the SPEs to get good performance

The SPEs

- **VLIW-style vector processors**
  - Simple and small so lots fit on the chip
  - Relies on a good compiler for good performance
  - Optimised for performance on certain workloads
    - Pure SIMD instruction set
- **Explicit main memory access with DMAs**
  - Calculation can continue while DMA completes
  - SPEs have 256k local store & lots of registers
- **SPEs synchronise using mailboxes**

Element Interconnect Bus

- **Actually 4 counter directional rings**
  - 25.6 Gbytes bandwidth on each
- **Best with large data transfers (>4kBytes)**
  - Lots of small transfers can saturate the bus
- **4 times more on chip bandwidth than memory bandwidth**
  - Can improve performance by reusing data between SPEs
  - Nearest neighbour best due to ring communication system

SIMD Improves Throughput and Efficiency

- **SPE Registers**
  - 128 registers, each 128 bits wide (16 bytes)
  - Unified register file: All registers handle integer and floating point data types.
  - Registers are wide enough for:
    - Integer types: 16 bytes, 8 short ints (16 bit), 4 ints (32 bit), 2 long longs (64 bit)
    - FP types: 4 floats (32 bits) or 2 doubles (64 bits)
- **SPEs are designed to work on vectors within a register**
  - A single instruction can generate multiple results within a register. For example, a register filled with 4 floats can be multiplied by another register with 4 floats, generating 4 results with a single instruction.
  - Vector units are a useful addition to general purpose architectures. Here, the SPE is designed from the ground up to be a vector unit.
  - Vector operations on registers are extremely useful when there is a lot of parallelism inherent in the data.

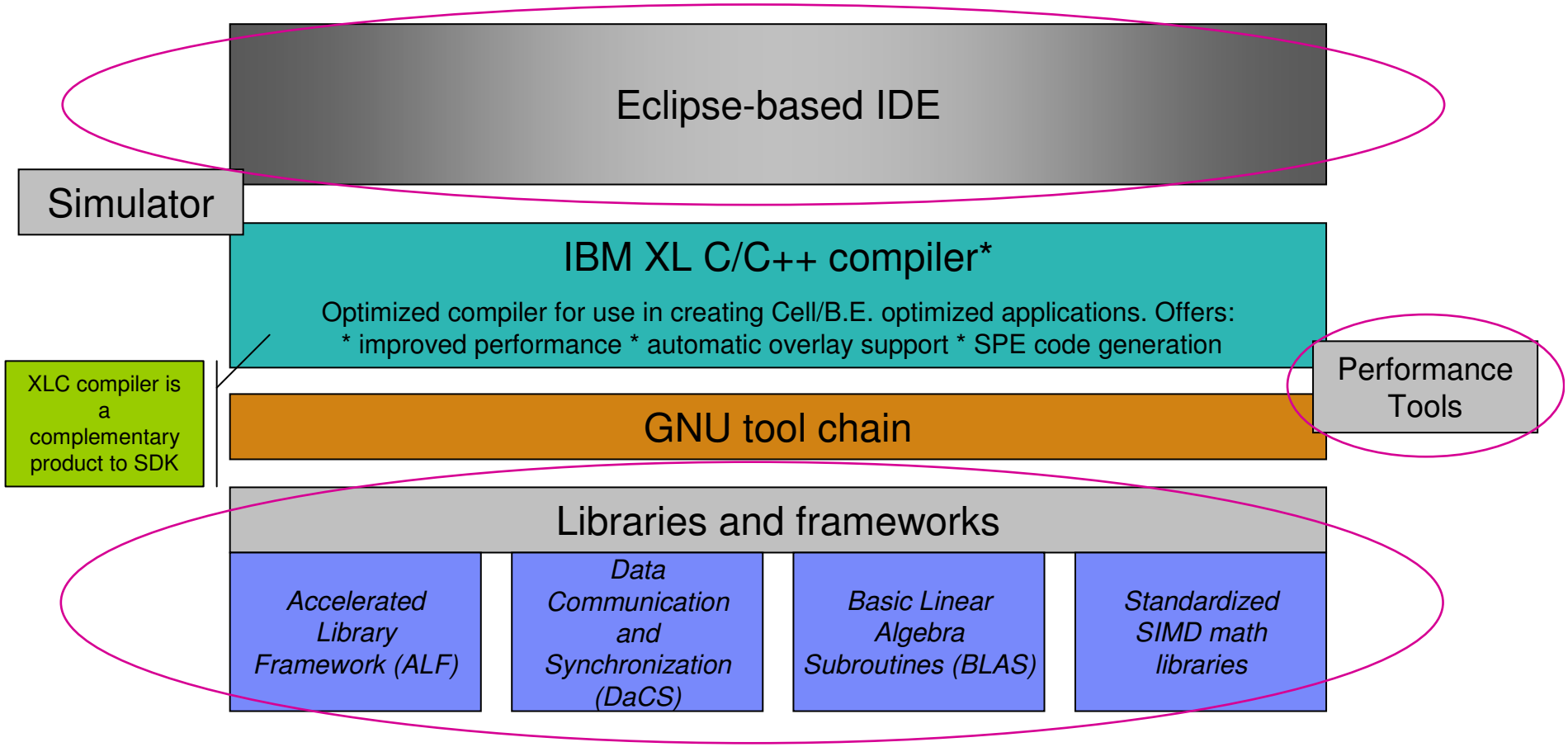
Concurrency with Multiple Cores

- **Each Cell BE has 8 SPUs**
  - Each SPU can operate independently.
  - SPUs in the same process share memory, so they can coordinate.
  - It's like having an 8 node cluster on single chip.
- **The Cell BE PPU**
  - Runs the host Linux operating system.
  - Two hardware threads.
  - Best used as a 'traffic director' for the SPUs; keep computational work to a minimum here so that latency for servicing the SPUs is minimal.

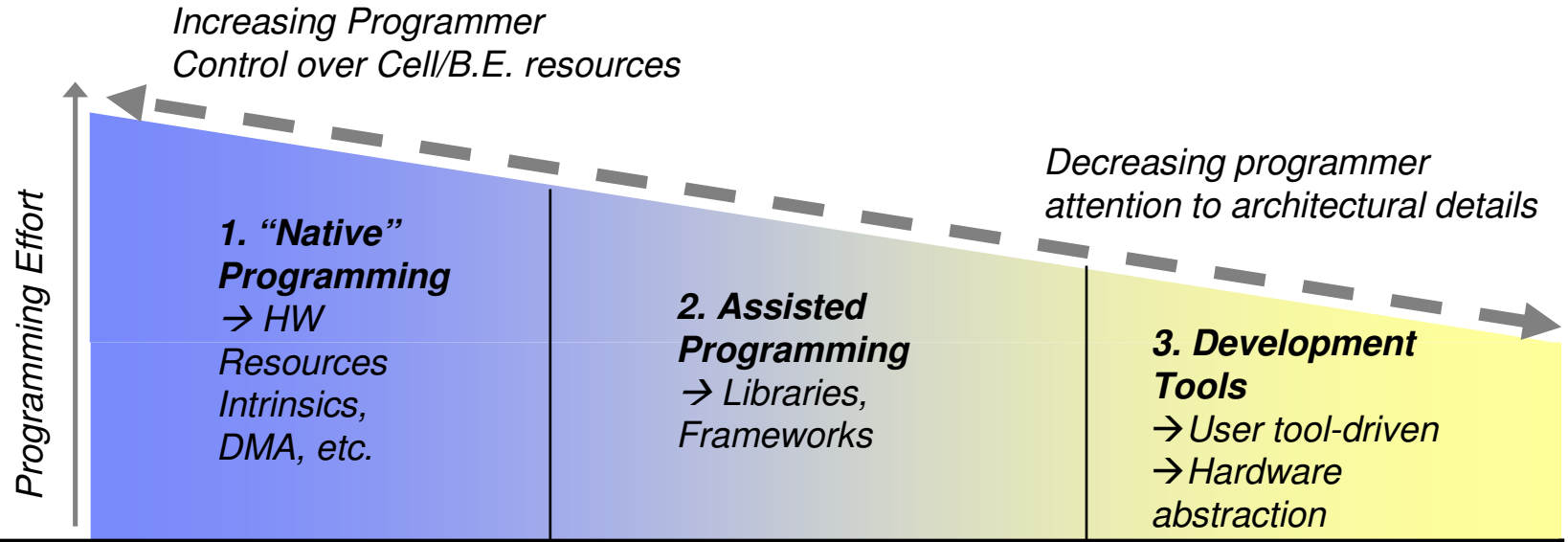
Programming the Cell B.E.

IBM SDK for Multicore Acceleration and related tools

The IBM SDK is a complete tools package that simplifies programming for the Cell Broadband Engine Architecture



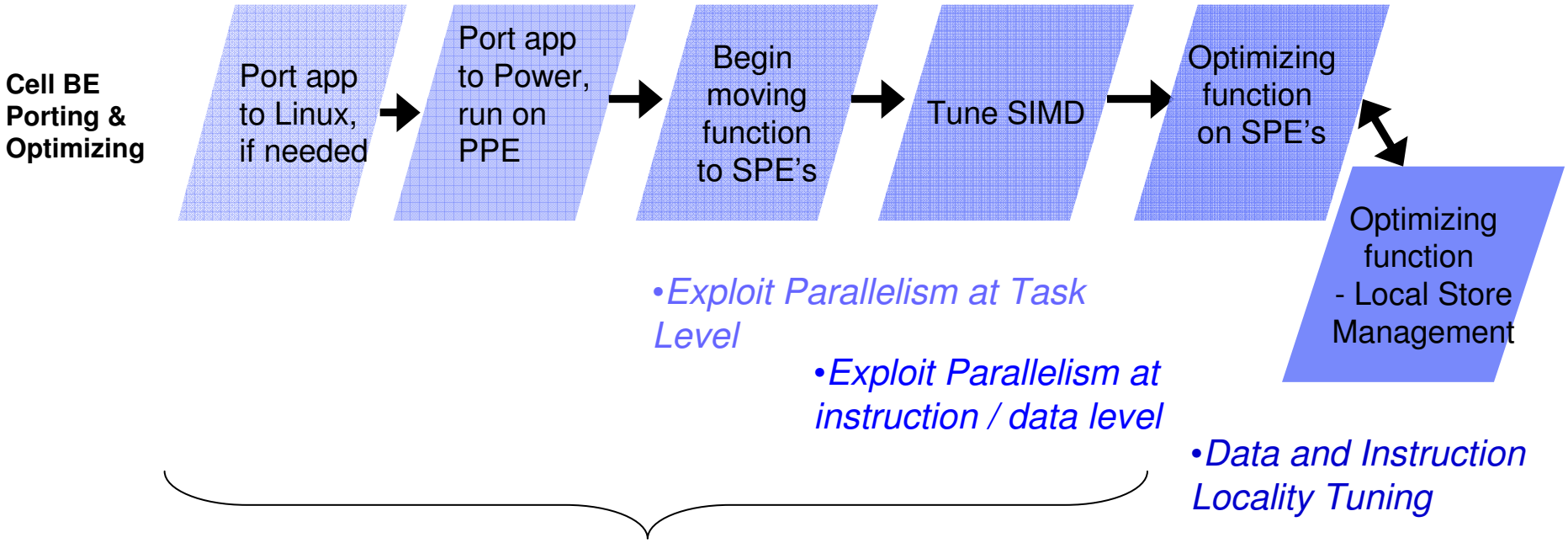
Cell/B.E. Programming Approaches are Fully Customizable!



	1. "Native" Programming	2. Assisted Programming	3. Development Tools
Advantages	▪Best performance possible ▪Best use of "Native" resources	▪Greatly reduced development time over "Native" Programming ▪Still allows some custom use of "Native" resources	▪Minimum development time required ▪Some degree of platform independence
Limitations	▪Requires the most coding work of the three options ▪Requires highest level of "Native" expertise	▪Performance gains may not be as great as with "Native" Programming ▪Confined to limitations of frameworks and libraries chosen	▪Performance gains may not be as great as with "Native" Programming ▪CASE tool determines debugging capabilities and platform support choices
Where it is most useful	▪Embedded Hardware / Real-time Applications ▪Hardware resources / power / space / cost are at a premium	▪Vast Majority of all applications	▪Simultaneous Deployments across multiple Hardware architectures ▪Programmers pool / skill base is restricted to high level skills

Cell Multi-core Programming Effort Roadmap

Requires *mostly the same* effort to port to any multi-core architecture.



Writing for Cell BE speeds up code on all multi-core architectures because it uses the same parallel best practices – Cell architecture just gains *more* from them because of its design.

SPE programming

- **Pure vector processor**
  - Every instruction is like an SSE instruction
  - Pure scalar code can be inefficient
  - Compiler can help, but only on simple code
- **DIY approach is to use intrinsics**
  - Look like function calls; they translate to instructions
  - Example: `vecR = spu_add(vec1, vec2)` translates to a single SPE instruction

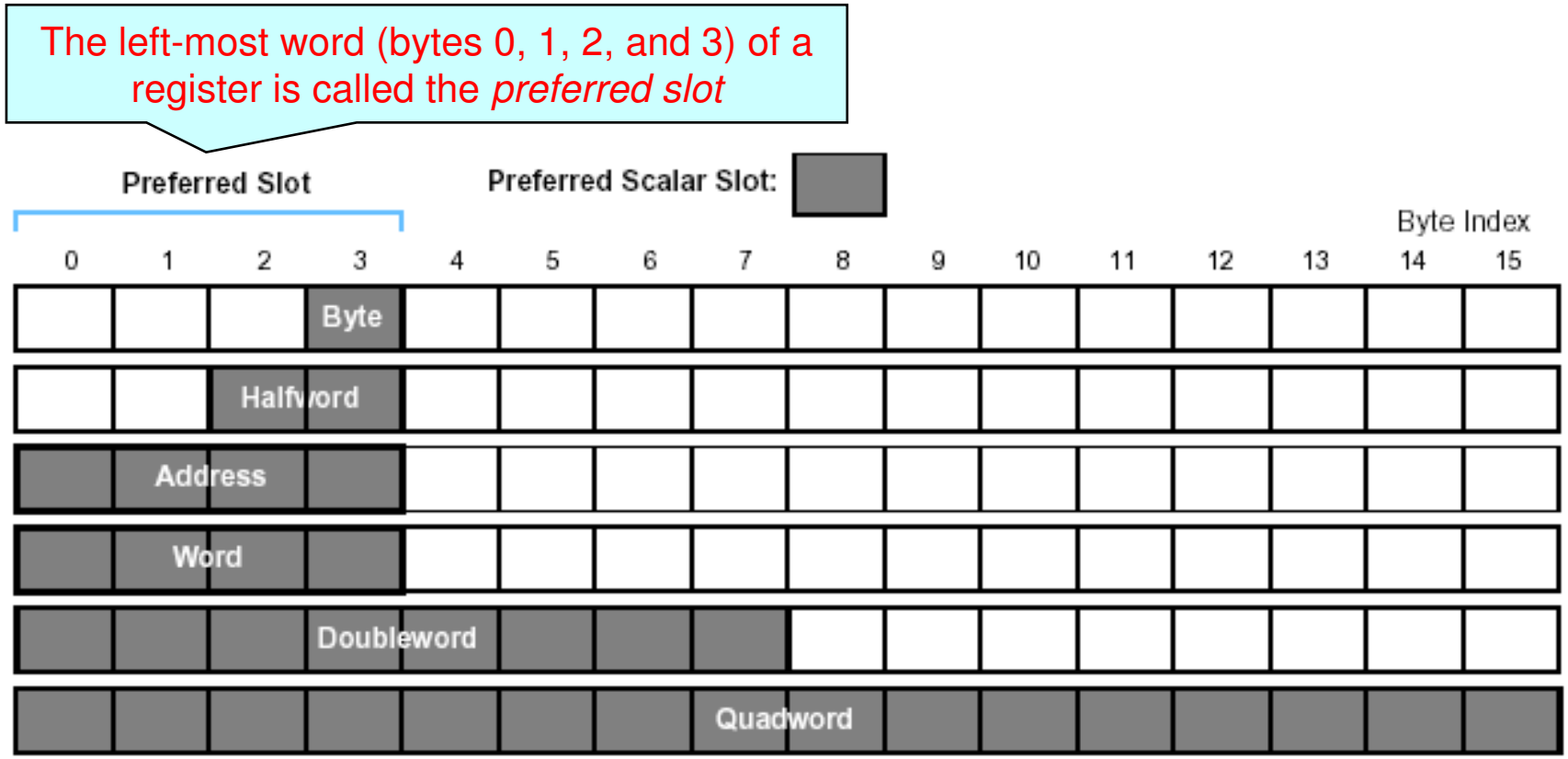
Example of an SPE Program

```
#include <stdio.h>
// Define a type we can look at either as an array of ints or as a vector.
typedef union {
    int iVals[4];
    vector signed int myVec;
} vecVar;
int main()
{
    vecVar v1, v2, vConst; // define variables
    // load the literal value 2 into the 4 positions in vConst,
    vConst.myVec = (vector signed int){2, 2, 2, 2};
    // load 4 values into the 4 element of vector v1
    v1.myVec = (vector signed int){10, 20, 30, 40};
    // call vector add function
    v2.myVec = spu_add( v1.myVec, vConst.myVec );
    // see what we got!
    printf("\nResults:\nv2[0] = %d, v2[1] = %d, v2[2] = %d, v2[3] = %d\n\n",
                v2.iVals[0], v2.iVals[1], v2.iVals[2], v2.iVals[3]);
    return 0;
}
```

SPU intrinsics

Register Layout of Data Types and Preferred Slot

When instructions use or produce scalar operands or addresses, the values are in the preferred scalar slot



Cell BE Communications Programming

Using the SPE memory system

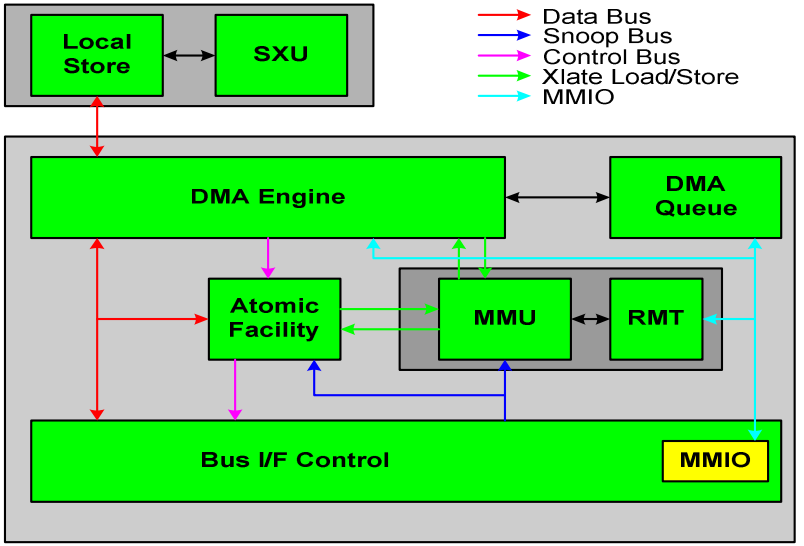
- **Each SPE has a software controlled local store**
  - Explicitly move data using DMA operations
  - Or use a software controlled cache
- **Explicit DMA is harder to program, but gives more control**
  - Good for high-volume predictable data movement patterns
- **IBM XL Single source compiler is available**
  - Can automatically partition calculation across multiple SPEs and PPE

Managing data for performance

- **Pipeline memory access using two buffers**
  - Issue a DMA to fetch data into one buffer
  - Work on pre-fetched data in the other
  - Reduces stalling due to memory access
- **Try to only DMA large chunks of data**
  - DMAs have a fixed overhead; lots of small transfers will saturate the bus
- **Software caches are an alternative**
  - Good for applications with random, small data accesses
  - Not so good for high volume data streams

Cell's Primary Communication Mechanisms

- DMA transfers, mailbox messages, and signal-notification
- All three are implemented and controlled by the SPE's MFC



Mechanism	Description
DMA transfers	Used to move data and instructions between main storage and an LS. SPEs rely on asynchronous DMA transfers to hide memory latency and transfer overhead by moving information in parallel with SPU computation.
Mailboxes	Used for control communication between an SPE and the PPE or other devices. Mailboxes hold 32-bit messages. Each SPE has two mailboxes for sending messages and one mailbox for receiving messages.
Signal notification	Used for control communication from the PPE or other devices. Signal notification (also called *signaling)* uses 32-bit registers that can be configured for one-sender-to-one-receiver signalling or many-senders-to-one-receiver signalling.

MFC Commands

- Main mechanism for SPUs to
  - access main storage (DMA commands)
  - maintain **synchronization** with other processors and devices in the system (Synchronization commands)
- Can be issued either SPU via its MFC by PPE or other device, as follows:
  - Code running on the SPU issues an MFC command by executing a series of writes and/or reads using **channel instructions -** read channel (rdch), write channel (wrch), and read channel count (rchcnt).
  - Code running on the PPE or other devices issues an MFC command by performing a series of stores and/or loads to **memory-mapped I/O** (MMIO) registers in the MFC
- MFC commands are queued in one of two independent MFC command queues:
  - MFC SPU Command Queue — For channel-initiated commands by the associated SPU
  - MFC Proxy Command Queue — For MMIO-initiated commands by the PPE or other device

DMA Commands

- MFC commands that transfer data are referred to as DMA commands
- Transfer direction for DMA commands referenced from the SPE
  - Into an SPE (from main storage to local store) → **get**
  - Out of an SPE (from local store to main storage) → **put**

DMA Get and Put Command (SPU)

- **DMA get from main memory into local store**

  (void) mfc_get(volatile void *ls, uint64_t ea, uint32_t size,
  uint32_t tag, uint32_t tid, uint32_t rid)

- **DMA put into main memory from local store**

  (void) mfc_put(volatile void *ls, uint64_t ea, uint32_t size,
  uint32_t tag, uint32_t tid, uint32_t rid)

lsaddr = target address in SPU local store for fetched data (SPU local address)

ea = effective address from which data is fetched (global address)

size = transfer size in bytes

tag_id = tag-group identifier

tid = transfer-class id

rid = replacement-class id

- To ensure order of DMA request execution:
  - mfc_putf : **fenced** (all commands executed before within the same tag group must finish first, later ones could be before)
  - mfc_putb : **barrier** (the barrier command and all commands issued thereafter are not executed until all previously issued commands in the same tag group have been performed)

Waiting for DMA Completion (SPE)

- **Wait for any tagged DMA:**

  mfc_read_tag_status_any():

  – wait until **any** of the specified tagged DMA commands is completed

- **Wait for all tagged DMA:**

  mfc_read_tag_status_all():

  – wait until **all** of the specified tagged DMA commands are completed

➢ **Specified tagged DMA commands = command specified by current tag mask setting**

DMA Example: Read into Local Store

```
inline void dma_mem_to_ls(unsigned int mem_addr,
         volatile void *ls_addr,unsigned int size)
{
  unsigned int tag = 0;
  unsigned int mask = 1;
  mfc_get(ls_addr,mem_addr,size,tag,0,0);
  mfc_write_tag_mask(mask);
  mfc_read_tag_status_all();
}
```

Read contents of mem_addr into ls_addr

Set tag mask

Wait for all tag DMA completed

DMA Example: Write to Main Memory

```
inline void dma_ls_to_mem(unsigned int mem_addr,volatile
void *ls_addr, unsigned int size)
{
  unsigned int tag = 0;
  unsigned int mask = 1;
  mfc_put(ls_addr,mem_addr,size,tag,0,0);
  mfc_write_tag_mask(mask);
  mfc_read_tag_status_all();
}
```

Write contents of mem_addr into ls_addr

Set tag mask

Wait for all tag DMA completed

SPE – SPE DMA

- Address in the other SPE's local store is represented as a 32-bit effective address (global address)
- SPE issuing the DMA command needs a pointer to the other SPE's local store as a 32-bit effective address (global address)
- PPE code can obtain effective address of an SPE's local store:

```
#include <libspe.h>

speid_t speid;

void *spe_ls_addr;

..

spe_ls_addr = spe_get_ls(speid);
```

- Effective address of an SPE's local store can then be made available to other SPEs (e.g. via DMA or mailbox)

DMA Characteristics

- **DMA transfers**
  - transfer sizes can be 1, 2, 4, 8, and n*16 bytes (n integer)
  - maximum is 16KB per DMA transfer
  - 128B alignment is preferable
- **DMA command queues per SPU**
  - 16-element queue for SPU-initiated requests
  - 8-element queue for PPE-initiated requests
    - SPU-initiated DMA is always preferable
- **DMA tags**
  - each DMA command is tagged with a 5-bit identifier
  - same identifier can be used for multiple commands
  - tags used for polling status or waiting on completion of DMA commands
- **DMA lists**
  - a single DMA command can cause execution of a list of transfer requests (in LS)
  - lists implement scatter-gather functions
  - a list can contain up to 2K transfer requests

Tips to Achieve Peak Bandwidth for DMAs

- The performance of a DMA data transfer is best when the source and destination addresses have the same quadword offsets within a PPE cache line.
- Quadword-offset-aligned data transfers generate full cache-line bus requests for every unrolling, except possibly the first and last unrolling.
- Transfers that start or end in the middle of a cache line transfer a partial cache line (less than 8 quadwords) in the first or last bus request, respectively.

SPU Programming Tips

SPU Programming Tips

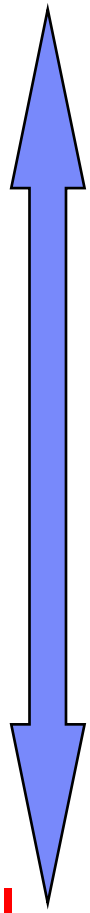
- Level of Programming (Assembler, Intrinsics, Auto-Vectorization)
- Overlap DMA with computation (double, multiple buffering)
- Dual Issue rate (Instruction Scheduling)
- Design for limited local store
- Branch hints or elimination
- Loop unrolling and pipelining
- Integer multiplies (avoid 32-bit integer multiplies)
- Shuffle byte instructions for table look-ups
- Avoid scalar code
- Choose the right SIMD strategy
- Load / Store only by quadword

Programming Levels on Cell BE

- **Expert level**
  - Assembler, high performance, high efforts
- **More ease of programming**
  - C compiler, vector data types, intrinsics, compiler schedules instructions + allocates registers
- **Auto-SIMDization**
  - for scalar loops, user should support by alignment directives, compiler provides feedback about SIMDization
- **Highest degree of ease of use**
  - user-guided parallelization necessary, Cell BE looks like a single processor

Trade-Off

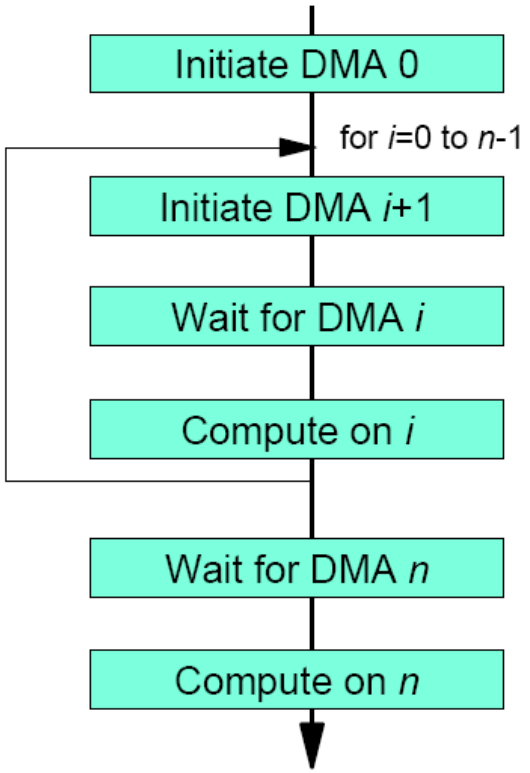
Performance vs. Effort



Requirements for Compiler increasing with each level

Overlap DMA with computation

- **Double or multi-buffer code or (typically) data**
- **Example for double buffering n+1 data blocks:**
  - Use multiple buffers in local store
  - Use unique DMA tag ID for each buffer
  - Use fence commands to order DMAs within a tag group
  - Use barrier commands to order DMAs within a queue



Start DMAs from SPU

- **Use SPE-initiated DMA transfers rather than PPE-initiated DMA transfers, because**
  - there are more SPEs than the one PPE
  - the PPE can enqueue only eight DMA requests whereas each SPE can enqueue 16

Performance on the SPE

- **SPE has two pipelines**
  - One arithmetic pipe and one control pipe
- **Want to keep both busy; How?**
  - Unroll loops; more instructions to schedule
  - Software pipeline loops; eliminates dependencies between instructions, improving issue rates
  - Or go overboard and do both
- **Compilers can auto unroll some loops (the easy option)**
- **Use analysis tools to guide optimization**

Branch Optimizations

- **SPE**
  - Heavily pipelined → high penalty for branch misses (18 cycles)
  - Hardware policy: assume all branches are not taken
- **Advantage**
  - Reduced hardware complexity
  - Faster clock cycles
  - Increased predictability
- **Solution approaches**
  - If-conversions: compare and select operations
  - Predications/code re-org: compiler analysis, user directives
  - Branch hint instruction (hbr, 11 cycles before branch)

Branches

- Eliminate non-predicted branches
  - use feedback directed optimization
  - use __builtin_expect when programmer can explicitly direct branch prediction
    - ex:

```
if (a > b)                          c += 1
else                        c = a+b

if (__builtin_expect(a>b, 0))   c += 1        // predict a is not > b
else                                c = a+b
```

  - utilitze the select bits (spu_sel) instruction.
    - ex:

```
if (a > b)                          c += 1
else                        c = a+b

select = spu_cmpgt(a, b);
c1 = spu_add(c, 1);
ab = spu_add(a, b);
c   = spu_sel(ab, c1, select);
```

Loop Unrolling

- Unroll loops
  - to reduce dependencies
  - increase dual-issue rates
- This exploits the large SPU register file.
- Compiler auto-unrolling is not perfect, but pretty good.

Loop Unrolling - Examples

```
j=N;
For(i=1, i<N, i++) {
   a[i] = (b[i] + b[j]) / 2;
   j = i;
}
```

```
a[1] = (b[1] + b[N]) / 2;
For(i=2, i<N, i++) {
   a[i] = (b[i] + b[i-1]) / 2;
}
```

```
For(i=1, i<100, i++) {
   a[i] = b[i+2] * c[i-1];
}
```

```
For(i=1, i<99, i+=2) {
   a[i] = b[i+2] * c[i-1];
   a[i+1] = b[i+3] * c[i];
}
```

SPU

- Unroll loops to reduce dependencies and increase dual issue rates.
  - exploits large SPU register file
  - Example - xformlight workload (each loop iteration processes 4 vertices)

SW unroll factor	normalized performance	CPI	dual issue	dependency stalls	regs	text size
1 (none)	1.00	1.35	3.3%	27.2%	78	768
2	1.52	0.91	19.8%	5.9%	103	1344
4	1.73	0.76	34.3%	0.9%	128	3076
8	1.66	0.67	35.8%	1.5%	128	5252

  - compiler auto-unrolling is not perfect, but doing pretty good.
  - Results using spuxlc unrolling (unroll_large):

SW unroll factor	normalized performance	CPI	dual issue	dependency stalls	regs	text size
1 (none)	1.57	0.87	21.6%	1.9%	94	1472
2	1.52	0.91	18.4%	5.9%	103	1344
4	1.73	0.76	34.3%	0.9%	128	3076
8	1.66	0.67	35.8%	1.5%	128	5252

Function-Inlining

- Function-inlining eliminates the two branches associated with function-call linkage
- These include the *branch and set link* (such as brasl) for function-call entry, and the *branch indirect* (such as bi) for function-call return
- Notes: Over-aggressive use of inlining and loop unrolling can result in code that reduces the LS space available for data storage or, in the extreme case, is too large to fit in the LS.

Avoid Scalar Code

- Scalar load/store are slow with long latency
  - SPU only supports quadword loads and stores
  - Ex:

```
void add1(int *p) {
        *p += 1;
}

add1:   lqd     $4, 0(p)      # load the qword pointed to by p
        rotqby  $5, $4, p     # move *p to element 0 of reg 5
        ai      $5, $5, 1     # add 1
        cwd     $6, 0(ptr)    # generate a shuffle pattern to insert *p+1 into qword
        shufb   $4, $5, $3    # insert scalar into qword
        stqd    $4, 0(p)      # save qword with updated scalar pointed to by p
```

  - Strategies:
    - consider making scalars qword integer vectors
    - load or store scalar arrays as quadwords and perform your own extraction and insertion to eliminate load/store instructions.
  - SDK example is RC4 encryption - scalar, non-parallelizable algorithm

	instructions	cycles	CPI	speedup
scalar	540120	723245	1.34	-
optimized to eliminate scalar overhead	265794	388457	1.46	1.86

Promoting Scalar Data Types to Vector Data Types

- Use spu_promote and spu_extract to efficiently promote scalars to vectors, or vectors to scalars.

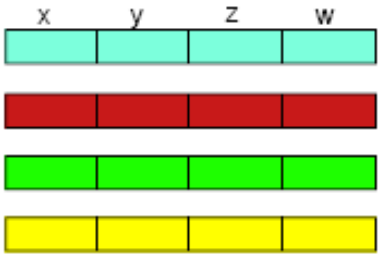
Instruction	Description
d = spu_promote(a, element)	Promote a scalar to a vector.
d = spu_extract(a, element)	Extract a vector element from its vector.

Choose an SIMD strategy appropriate for your algorithm

- Evaluate array-of-structure (AOS) organization
  - For graphics vertices, this organization (also called or vector-across) can have more-efficient code size and simpler DMA needs,
  - but less-efficient computation unless the code is unrolled.
- Evaluate structure-of-arrays (SOA) organization.
  - For graphics vertices, this organization (also called parallel-array) can be easier to SIMDize,
  - but the data must be maintained in separate arrays or the SPU must shuffle AOS data into an SOA form.

Choose SIMD strategy appropriate for algorithm

- **vec-across**
  - More efficient code size
  - Typically less efficient code/computation unless code is unrolled
  - Typically simpler DMA needs



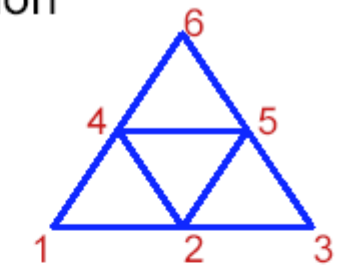
4 independent 4-component vectors (cyan, red, green and yellow)

- **parallel-array**
  - Easy to SIMD – program as if scalar, operating on 4 independent objects at a time
  - Data must be maintained in separate arrays or SPUs must shuffle vec-across data into a parallel array form

- **Consider unrolling affects when picking SIMD strategy**

SIMD Example

- SIMD example - point-normal triangle subdivision
  - problem: compute subdivided vertices for *n* independent triangles



  - solution 1: vec-across
    - evaluate vertices one at a time, unroll in improve performance
  - solution 2: parallel array
    - evaluate 4 subdivision vertices at a time for a single triangle
  - solution 3: parallel array
    - evaluate 1 subdivision point at a time on 4 independent triangles

solution	normalized performance	CPI	dual issue	dependency stalls	regs	text size
1	1.0	1.10	9.1%	14.1%	72	1472
1 (unrolled by 2)	1.26	1.04	12.6%	13.6%	112	2496
1 (unrolled by 4)	1.54	0.90	18.5%	5.8%	127	4480
2	1.34	0.96	11.9%	2.6%	107	1856
3	1.70	0.99	13.6%	4.7%	113	512

Where to get more Cell BE information?

Cell Resources

- **Cell resource center at developerWorks**
  - http://www-128.ibm.com/developerworks/power/cell/
- **Cell developer's corner at power.org**
  - http://www.power.org/resources/devcorner/cellcorner/
- **The cell project at IBM Research**
  - http://www.research.ibm.com/cell/
- **Cell BE at IBM Engineering & Technical Services**
  - http://www-03.ibm.com/technology/
- **IBM Power Architecture**
  - http://www-03.ibm.com/chips/power/
- **Linux info at the Barcelona Supercomputing Center website**
  - http://www.bsc.es/projects/deepcomputing/linuxoncell/

Cell Education

- **Online courses at IBM Education Assistant**
  - http://publib.boulder.ibm.com/infocenter/ieduasst/stgv1r0/index.jsp
- **Online courses at IBM Learning**
  - http://ibmlearning.ibm.com/index.html
- **Podcasts at power.org**
  - http://www.power.org
- **Onsite classes at IBM Innovation Center**
  - https://www-304.ibm.com/jct09002c/isv/spc/events/cbea.html
- **IBM Redbook: Programming the Cell Broadband Engine™ Architecture**
  - http://www.redbooks.ibm.com/abstracts/sg247575.html

Cell BE Documentation - Processor

Cell Broadband Engine

- PowerPC User Instruction Set Architecture - Book I
- PowerPC Virtual Environment Architecture - Book II
- PowerPC Operating Environment Architecture - Book III
- Vector/SIMD Multimedia Extension Technology Programming Environments Manual
- Cell Broadband Engine Architecture
- Cell Broadband Engine Registers
- Synergistic Processor Unit Instruction Set Architecture

A Sample of Cell BE Technical Articles

- Real-time Ray Tracing
- Papers from the Fall Processor Forum 2005: Unleashing the Cell Broadband Engine Processor: The Element Interconnect Bus
- Papers from the Fall Processor Forum 2005: Unleashing the power of the Cell Broadband Engine: A programming model approach
- Cell Broadband Engine Architecture and its first implementation
- Introduction to the Cell Broadband Engine
- Introduction to the Cell Multiprocessor
- Maximizing the power of the Cell Broadband Engine processor: 25 tips to optimal application performance
- Terrain Rendering Engine (TRE): Cell Broadband Engine Optimized Real-time Ray-caster
- An Implementation of the Feldkamp Algorithm for Medical Imaging on Cell Broadband Engine
- Cell Broadband Engine Support for Privacy, Security, and Digital Rights Management Applications
- A Programming Example: Large FFT on the Cell Broadband Engine

Notes on Benchmarks and Values

The IBM benchmarks results shown herein were derived using particular, well configured, development-level and generally-available computer systems. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the numbers stated here. Buyers should consult other sources of information to evaluate the performance of systems they are considering buying and should consider conducting application oriented testing. For additional information about the benchmarks, values and systems tested, contact your local IBM office or IBM authorized reseller or access the Web site of the benchmark consortium or benchmark vendor.

- Performance information is provided "AS IS" and no warranties or guarantees are expressed or implied by IBM.
- Latest levels of libraries and directives were used as indicated. Number of SPUs (or equivalently number of SPEs) used as indicated.
- Different application implementations of the algorithms tested in these benchmarks may affect performance results.
- Source for all data is IBM internal benchmark testing as of April 15, 2008.
- For a description of the LINPACK workload refer to http://www.netlib.org/benchmark/performance.pdf
- PFAFFT is part of the CWP (Center for Waveform Phenomena) freeware seismic data processing package from the Colorodo School of Mines: http://www.cwp.mines.edu/
- The SCAMPI workload is a new algorithm based on SNORT: http://www.snort.org/
- Image registration application was completed as a joint project between IBM and Mayo Foundation for Medical Education and Research to port the ITK library to Cell/B.E..

  Refer to ITK - Image Registration Toolkit: http://www.doc.ic.ac.uk/~dr/software/
- For a description of the HMMer application refer to

  http://hmmer.janelia.org/ and http://powerdev.osuosl.org/project/hmmerAltivecGen2mod
- For a description of IBM iRT, refer to
  - http://gametomorrow.com/blog/index.php/2007/11/09/cell-and-the-boeing-777-at-sc07/ and
  - http://gametomorrow.com/blog/index.php/2007/09/05/cell-vs-g80/
- Source 3D data for IBM iRT provided by and used with permission of The Boeing Company
- RHEL 5.2 is officially supported for QS22; in some cases results were obtained with RHEL 5.1;

Revised April 25, 2008

Notes on Benchmarks and Values, cont.

- For a description of BLAS and LAPACK functions refer to http://www.netlib.org/lapack/faq.html ; MKL 10 and SDK 3.0.0.3 were used for BLAS and LAPACK routines.
- Description of financial algorithms can be found in the open source financial library: http://www.quantlib.org/reference/overview.html
- Mersenne-Twister information can be found at http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
- Additional information on OpenMP: Simple, Portable, Scalable SMP Programming can be found at http://www.openmp.org
- Additional information on Monte Carlo methods: N. Metropolis and S. Ulam. 1949. The Monte Carlo method. Journal of the American Statistical Association 44:335-341.
- Additional information on Black-Scholes:
  - Black, Fischer; Myron Scholes (1973). "The Pricing of Options and Corporate Liabilities". Journal of Political Economy 81 (3): 637-654.
  - The origin of Black-Scholes: http://bradley.bradley.edu/~arr/bsm/model.html.
  - Black-Scholes formula is available:
    - http://www.riskglossary.com/link/black_scholes_1973.htm
    - http://www.global-derivatives.com/index.php?option=com_content&task=view&id=52&Itemid=31
- Additional information on Cholesky: André-Louis Cholesky: http://www.gnu.org/software/gsl/manual/html_node/Cholesky-Decomposition.html
- Additional details on digital video surveillance can be found at http://domino.watson.ibm.com/library/CyberDig.nsf/1e4115aea78b6e7c85256b360066f0d4/b45ed52afb06ac378525711f00619b02?OpenDocument
- Additional details on the Los Alamos National Laboratory Roadrunner system can be found at http://www.lanl.gov/orgs/hpc/roadrunner/rrinfo/RR%20webPDFs/Turner_Apps_v6_LA-UR.pdf
- Additional information on medical imaging can be found at
  - Image Registration Toolkit: http://www.doc.ic.ac.uk/~dr/software/
  - http://www-03.ibm.com/technology/cell/pdf/Mayo_1.1.pdf
  - http://icsl.washington.edu/miworkshop/MIWorkShop.pdf
  - http://www-03.ibm.com/technology/cell/pdf/GEposterISMRM.pdf
  - http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/5B1968BDD8D11639872570AB005A3A39/$file/GSPx2005paper%20for%20sdk.pdf

Revised April 25, 2008

Special Notices -- Trademarks

This document was developed for IBM offerings in the United States as of the date of publication. IBM may not make these offerings available in other countries, and the information is subject to change without notice. Consult your local IBM business contact for information on the IBM offerings available in your area. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

Information in this document concerning non-IBM products was obtained from the suppliers of these products or other public sources. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquires, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

The information contained in this document has not been submitted to any formal IBM test and is provided "AS IS" with no warranties or guarantees either expressed or implied.

All examples cited or described in this document are presented as illustrations of the manner in which some IBM products can be used and the results that may be achieved. Actual environmental costs and performance characteristics will vary depending on individual client configurations and conditions.

IBM Global Financing offerings are provided through IBM Credit Corporation in the United States and other IBM subsidiaries and divisions worldwide to qualified commercial and government clients. Rates are based on a client's credit rating, financing terms, offering type, equipment type and options, and may vary by country. Other restrictions may apply. Rates and offerings are subject to change, extension or withdrawal without notice.

IBM is not responsible for printing errors in this document that result in pricing or information inaccuracies.

All prices shown are IBM's United States suggested list prices and are subject to change without notice; reseller prices may vary.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Many of the features described in this document are operating system dependent and may not be available on Linux. For more information, please check: http://www.ibm.com/systems/p/software/whitepapers/linux_overview.html

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Some measurements quoted in this document may have been estimated through extrapolation. Users of this document should verify the applicable data for their specific environment.

Revised January 19, 2006

Special Notices (Cont.) -- Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States and/or other countries: alphaWorks, BladeCenter, Blue Gene, ClusterProven, developerWorks, e business(logo), e(logo)business, e(logo)server, IBM, IBM(logo), ibm.com, IBM Business Partner (logo), IntelliStation, MediaStreamer, Micro Channel, NUMA-Q, PartnerWorld, PowerPC, PowerPC(logo), pSeries, TotalStorage, xSeries; Advanced Micro-Partitioning, eServer, Micro-Partitioning, NUMACenter, On Demand Business logo, OpenPower, POWER, Power Architecture, Power Everywhere, Power Family, Power PC, PowerPC Architecture, POWER5, POWER5+, POWER6, POWER6+, Redbooks, System p, System p5, System Storage, VideoCharger, Virtualization Engine.

A full list of U.S. trademarks owned by IBM may be found at: http://www.ibm.com/legal/copytrade.shtml.

Cell Broadband Engine and Cell Broadband Engine Architecture are trademarks of Sony Computer Entertainment, Inc. in the United States, other countries, or both.
Rambus is a registered trademark of Rambus, Inc.
XDR and FlexIO are trademarks of Rambus, Inc.
UNIX is a registered trademark in the United States, other countries or both.
Linux is a trademark of Linus Torvalds in the United States, other countries or both.
Fedora is a trademark of Redhat, Inc.
Microsoft, Windows, Windows NT and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries or both.
Intel, Intel Xeon, Itanium and Pentium are trademarks or registered trademarks of Intel Corporation in the United States and/or other countries.
AMD Opteron is a trademark of Advanced Micro Devices, Inc.
Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.
TPC-C and TPC-H are trademarks of the Transaction Performance Processing Council (TPPC).
SPECint, SPECfp, SPECjbb, SPECweb, SPECjAppServer, SPEC OMP, SPECviewperf, SPECapc, SPEChpc, SPECjvm, SPECmail, SPECimap and SPECsfs are trademarks of the Standard Performance Evaluation Corp (SPEC).
AltiVec is a trademark of Freescale Semiconductor, Inc.
PCI-X and PCI Express are registered trademarks of PCI SIG.
InfiniBand™ is a trademark the InfiniBand® Trade Association
Other company, product and service names may be trademarks or service marks of others.

Revised July 23, 2006

Special Notices - Copyrights

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both.

IBM IBM Logo Power Architecture

Other company, product and service names may be trademarks or service marks of others.

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury, or catastrophic property damage. The information contained in this document does not affect or change IBM product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6351

The IBM home page is http://www.ibm.com
The IBM Microelectronics Division home page is
http://www.chips.ibm.com

