IBM Systems and Technology Group

# Cell Broadband Engine Architecture Overview

Luke Browning (lukebrowning@us.im.com)

IBM Brazil Linux Technology Center (Hortolandia)

July 17, 2006

## Discussion topics

- CBE HW architecture
- Target Markets
- CBE performance
- Programming models
- Programming hints
- Components and Ecosystem
- History and Present Status

## Linux on Cell

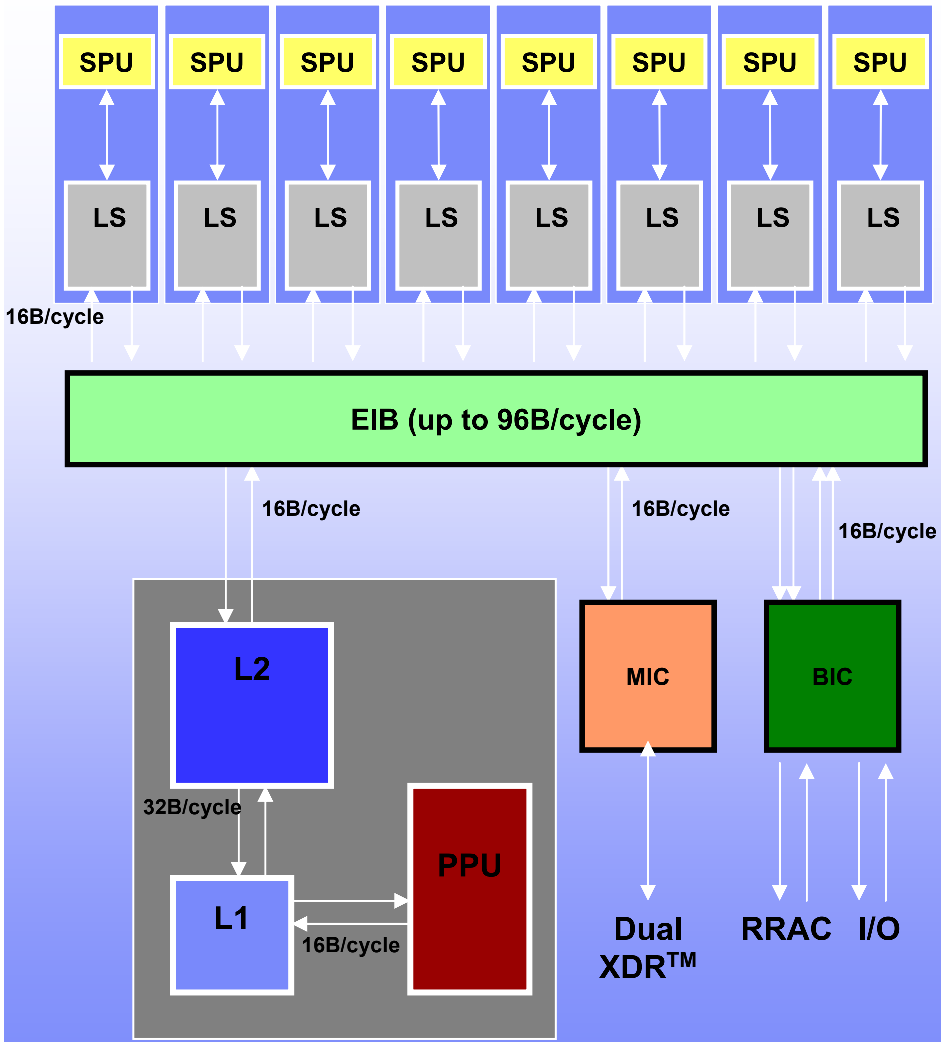
# CBE HW architecture

# CBE HW Review

Cell includes 1 Power Processor core + 8SPEs

- provides more than 8x compute capability than traditional processors
  - decoupled SIMD engines for growth and scalability
- 1 64-bit Power Processor core micro-architecture
  - less complexity with in-order execution
  - minimal chip area / power budget
  - dual issue
  - dual thread SMT
  - VMX
- 8 SPE SIMD engines provide tremendous compute power
  - dual-issue
  - dedicated resources
    - 128 128-bit registers
    - 256KB local store
    - 2x16B/cycle DMA, etc.
  - up to 16-way SIMD for exploiting data parallelism
- Data ring for intra-processor and external communication
  - 96B/cycle peak bandwidth
  - 16B/cycle memory b/w
  - 2 X 16B/cycle BIF and IO

*Synergistic Processor Elements for High (Fl)ops / Watt*

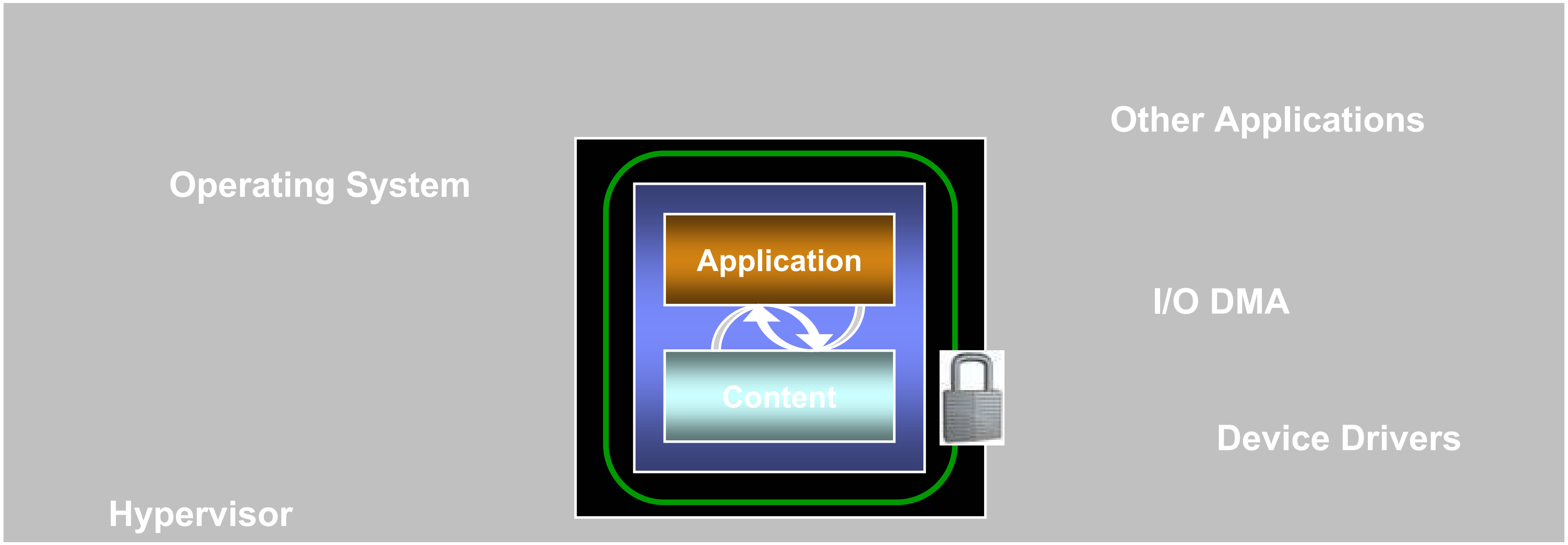


*64-bit Power Architecture w/VMX for Traditional Computation*

# Cell Processor Real Time Features

- Resource Reservation system for reserving bandwidth on shared units such as system memory, I/O interfaces
- L2 Cache Locking system based on Effective or Real Address ranges - supports both locking for Streaming, and locking for High Reuse
- TLB Locking system based on Effective or Real Address ranges or DMA class.
- Fully pre-emptible context switching capability for each SPE
- Privileged Attention Event to SPE for use in contractual light weight context switching
- Multiple concurrent large page support in the PPE and SPE to minimize real-time impact due to TLB misses
- Up to 4 service classes (software controlled) for DMA commands (improves parallelism)
- Large page I/O Translation facility for I/O devices, graphics subsystems, etc - minimizes I/O translation cache misses
- SPE Event Handling facilities for high priority task notification
- PPE SMT Thread priority controls for Low, Medium and High Priority Instruction dispatch

## Cell/SPE Security – Secure Processing Vault



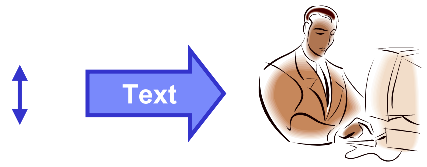
- Resistant to Software Security Hacks
- Does not rely on the security of the Operating System/Hypervisor
  - **Even if the Operating System is hacked, Application and Data remain secure**
- Root of security/trust is with the application and not the Operating System/Hypervisor
- A Processor Core and Memory becomes isolated from the rest of the cores and the system to become a **Secure Processing Vault**
- The Application and Content can execute safely within a Secure Processing Vault

## Linux on Cell

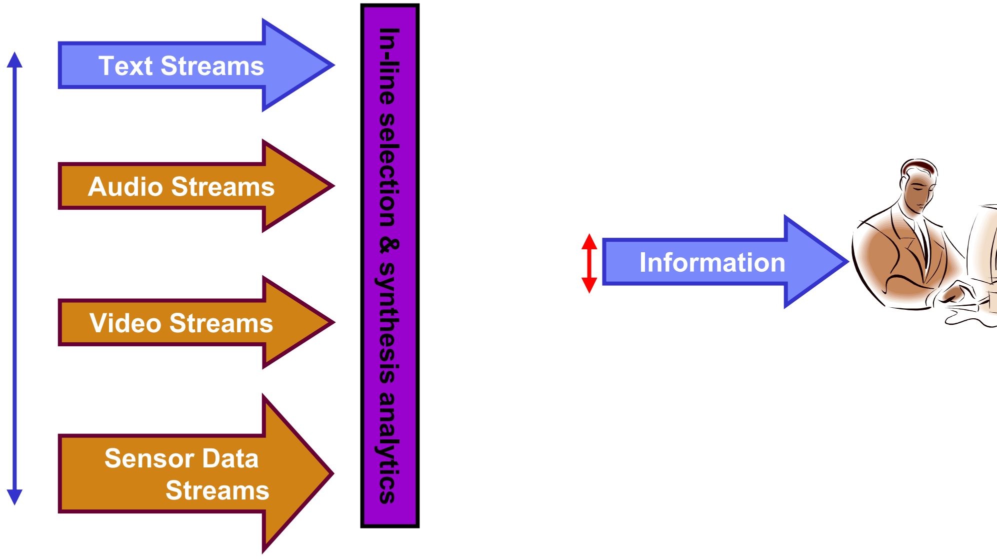
Target Markets

## Next Generation Workloads: A Data centric view

***Then***



***Now***

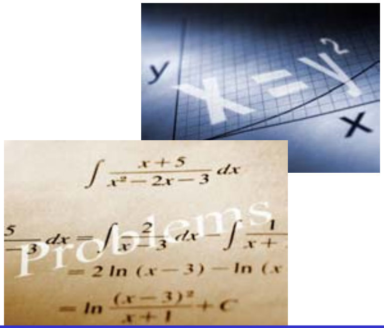


**System synthesizes & filters data into information**

***The Cell system architecture enables the selection, synthesis and presentation of relevant information for human consumption →***

***Real-Time Information Interaction***

# Next Generation Workloads



**Software Stack**

Applications

*ISV, Universities, Labs, etc*

Sector Specific Libraries

*ISVs, Universities, Labs, Open Source, etc.*

Cluster and Scale out Systems

*Global operating systems, cluster file systems and protocols, et*

***Digital Media*** ***Information Based Medicine*** ***Financial Services Sector*** ***Home Media/ Consumer Electronics***

**Market Segment Specific**

**Ubiquitous for all Markets**

Application Tooling and Environment

*Programming model/APIs for accelerators and next generation cluster*

**Compilers**

**C, C++, Fortran, etc**

**Base Libraries**

**e.g. SPE intrinsic, etc**

**Operating Systems such as Linux**

**e.g. SPE exploitation, BE awareness**

**Device Drivers**

**Firmware**

**e.g. Blades, Development platforms, etc**

**HyperVisors**

***Opportunities for Cell technology arise where the rate of data produced far outpaces the rate at which humans can digest the data, interpret as information, and apply to knowledge based decisions … all in real time…***

## Linux on Cell

# CBE Performance

## Single-SPE MatrixMultiply Performance (Single Precision)

| | # of Cycles | # of Insts. | CPI | Dual Issue | Channel Stalls | Other Stalls | # of Used Registers | GFLOPs | Effic'y |
|---|---|---|---|---|---|---|---|---|---|
| **Original (scalar)** | 258.9M | 247.1M | 1.05 | 26.1% | 11.4% | 26.3% | 47 | 0.42 | 1.6% |
| **SIMD optimized** | 9.78M | 13.8M | 0.711 | 40.3% | 3.0% | 9.8% | 60 | 10.96 | 42.8% |
| **SIMD + dbl buf** | 9.68M | 13.6M | 0.711 | 41.4% | 2.6% | 10.2% | 65 | 11.12 | 43.4% |
| **Optimized code** | 4.27M | 8.42M | 0.508 | 80.1% | 0.2% | 0.4% | 69 | 25.12 | 98.1% |

- The original scalar-version of MatrixMultiply on SPE achieved only 0.42 GFlops.
- Performance improved significantly with optimizations and tunings by
  - taking advantage of data level parallelism using SIMD
  - double buffering for concurrent data transfers and computation
  - optimizing dual issue rate, instruction scheduling, etc.

# Cell Performance Comparison

- BE's performance is about an order of magnitude better than traditional GPPs for media and other applications that can take advantage of its SIMD capability
- BE can outperform a P4/SSE2 at same clock rate by 3 to 18x (assuming linear scaling) in various types of application workloads

| Type | Algorithm | 3.2 GHz GPP | 3.2 GHz Cell | Perf Advantage |
|---|---|---|---|---|
| HPC | **Matrix Multiplication (S.P.)** | 25.6 Gflops* (w/SIMD) | 200 GFlops (8SPEs) | 8x (8SPEs) |
| | **Linpack (S.P.) 4k x 4k** | 25.6 GFlops* (w/SIMD) | 156 GFlops (8SPEs) | 6x (8SPEs) |
| | **Linpack (D.P.) 1k x 1k** | 7.2 GFlops (3.6GHz IA32/SSE3) | 9.67 GFLops (8SPEs) | 1.3x (8SPEs) |
| graphics | **TRE** | .85 fps (2.7GHz G5/VMX) | 30 fps (Cell) | 35x (Cell) |
| | **transform-light** | 128 MVPS (2.7GHz G5/VMX) | 217 MVPS (one SPE) | 1.7x (one SPE) |
| security | **AES ECB encryp. 128b key** | 1.03 Gbps | 2.06Gbps (one SPE) | 2x (one SPE) |
| | **AES ECB decryp. 128b key** | 1.04 Gbps | 1.5Gbps (one SPE) | 1.4x (one SPE) |
| | **TDES ECB encryp.** | 0.13 Gbps | 0.17 Gbps (one SPE) | 1.3x (one SPE) |
| | **DES ECB encryp.** | 0.43 Gbps | 0.49 Gbps (one SPE) | 1.1x (one SPE) |
| | **SHA-1** | 0.9 Gbps | 2.12 Gbps (one SPE) | 2.3x (one SPE) |
| video processing | **mpeg2 decoder (sdtv)** | 354 fps (w/SIMD) | 329 fps (one SPE) | 0.9x (one SPE) |

*assuming 100% compute efficiency, achieving theoretical peak of 25.6GLOPS, in its single precision MatrixMultiply & Linpack implementation

## Cell Performance Summary

- **Cell's performance is about an order of magnitude better than GPP for media and other applications that can take advantage of its SIMD capability**
  - performance of its simple PPE is comparable to a traditional GPP performance
  - each SPE is able to perform mostly the same as, or better than, a GPP with SIMD running at the same frequency
  - key performance advantage comes from its 8 de-coupled SPE SIMD engines with dedicated resources including large register files and DMA channels
- **BE can cover a wide range of application space with its capabilities in**
  - floating point operations
  - integer operations
  - data streaming / throughput support
  - real-time support
- **BE microarchitecture features are exposed to not only its compilers but also its applications**
  - performance gains from tuning compilers and applications can be significant
  - tools/simulators are provided to assist in performance optimization efforts

## Linux on Cell

programming models

## Software Exploitable Parallelism on Cell BE

- **Data-level parallelism – SIMD**
  - SPE SIMD architecture
  - VMX unit of PPE
- **Task-level parallelism – 8 SPEs + 2 PPE SMT**
- **Data Transfer via SPE DMA engines (MFCs)**
  - Demo (TRE)
- **SMP Cell BE system / cluster level**

## HW aspects influencing viable Programming Models

PowerPC 64 compliant

High speed coherent interconnect

Limited local store size

Coherent shared memory

Signal Notification Registers

Multiple execution units

Atomic operations

Aliased LS memory

Mailboxes

Heterogeneous

Multi-threading

SIMD

SPE Events

SW managed DMA engines

Bandwidth Reservations

VM address translation and protection

High speed EIB

DMA list supporting scatter / gather

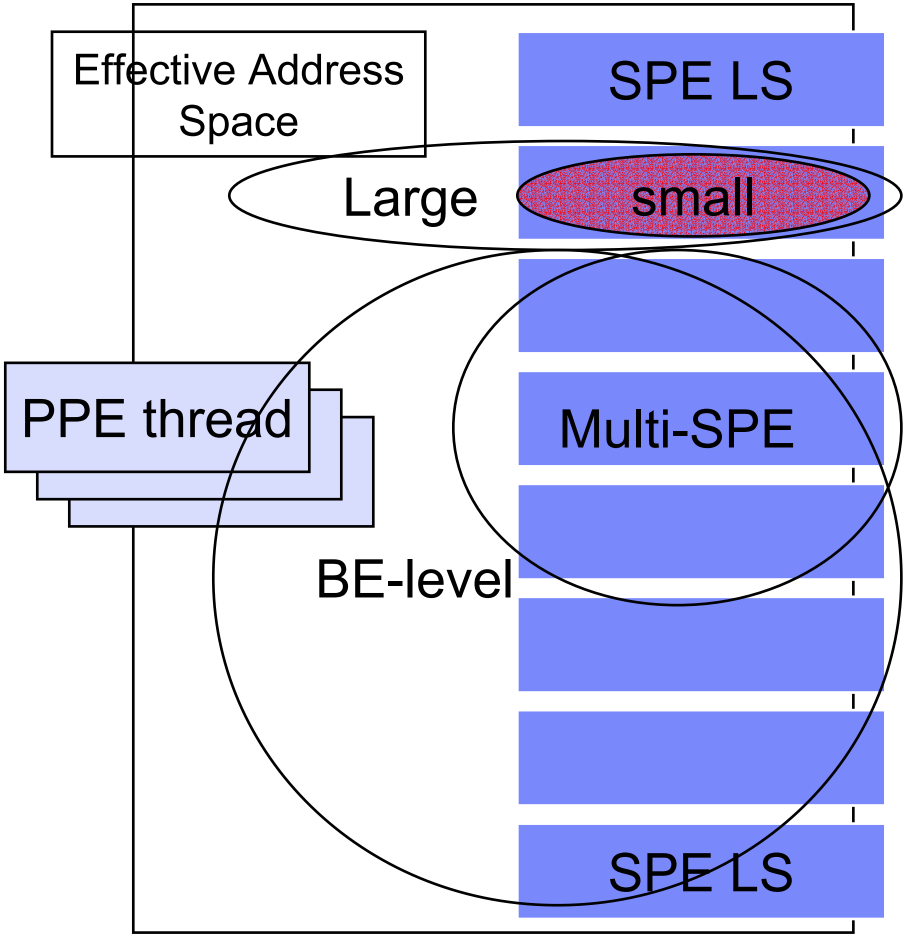
Direct problem state mapping

Resource Management Tables

DMA alignment & size restrictions

# Programming models in a single Cell BE

- **PPE programming models**
- **SPE Programming models**
  - Small single-SPE models
  - Large single-SPE models
  - Multi-SPE parallel programming models
- **Integrated Object Format -Cell BE Embedded SPE Object Format (CESOF)**
- **Multi-tasking SPEs**
  - Local Store resident multi-tasking
  - Self-managed multi-tasking
  - Kernel-managed SPE scheduling and virtualization



## PPE programming models

- **PPE is a 64-bit PowerPC core, hosting operating systems and hypervisor**
- **PPE program inherits traditional programming models**
- **Cell BE environment: a PPE program serves as a controller or facilitator**
  - CESOF object format and runtime provides SPE image handles to a PPE program
  - PPE program establishes a runtime environment for SPE programs
    - e.g. memory mapping, exception handling,
  - PPE program starts and stops SPE programs
  - It allocates and manages Cell BE system resources
    - SPE scheduling, hypervisor CBEA resource management
  - It provides OS services to SPE programs
    - e.g. printf, file I/O

## Small single-SPE models

- **Single tasked environment**
- **Small enough to fit into a 256KB- local store**
- **Sufficient for many dedicated workloads**
- **Separated SPE and PPE address spaces – LS / EA**
- **Explicit input and output of the SPE program**
  - Program arguments and exit code per SPE ABI
  - DMA
  - Mailboxes
  - SPE side system calls
- **Foundation for a function offload model or a synchronous RPC model**
  - Facilitated by interface description language (IDL)

## Small single-SPE models – tools and environment

- **SPE compiler/linker compiles and links an SPE executable**
- **The SPE executable image is embedded as reference-able RO data in the PPE executable (CESOF)**
- **A Cell BE programmer controls an SPE program via a PPE controlling process and its SPE management library**
  - i.e. loads, initializes, starts/stops an SPE program
- **The PPE controlling process, OS/PPE, and runtime/(PPE or SPE) together establish the SPE runtime environment, e.g. argument passing, memory mapping, system call service.**

## Small single-SPE models – a sample

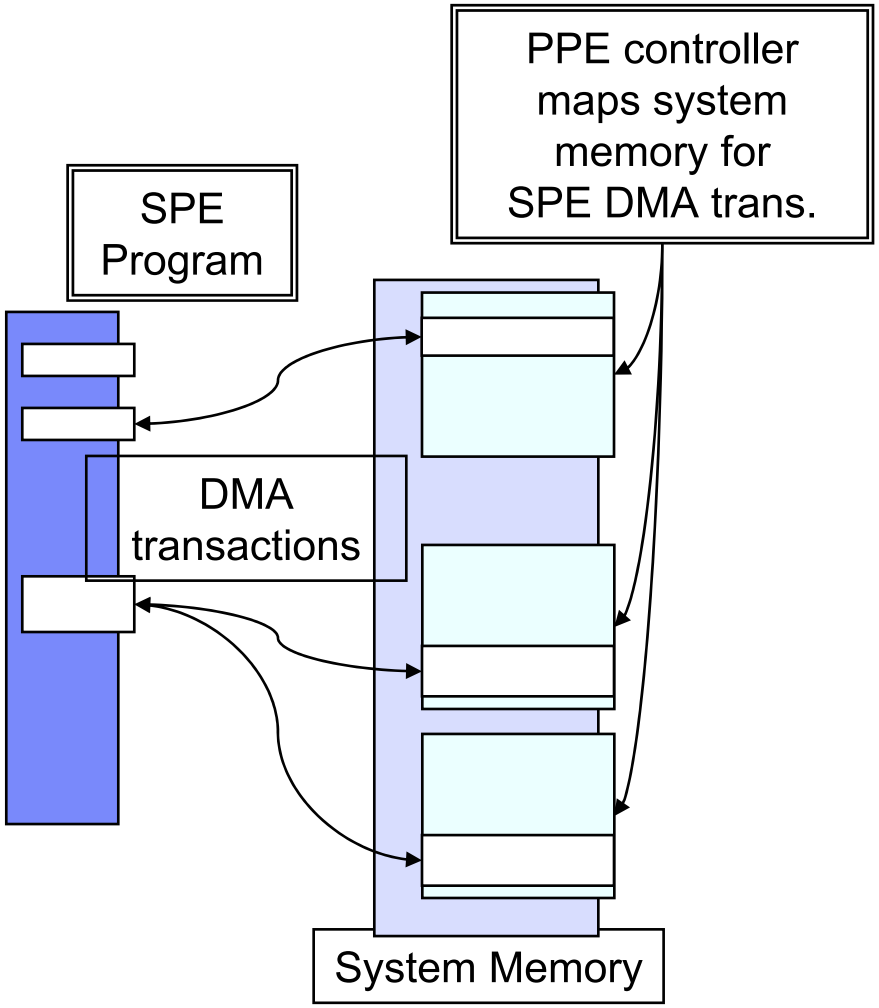
```
/* spe_foo.c:
 * A C program to be compiled into an executable called “spe_foo”
 */

 int main( int speid, addr64 argp, addr64 envp)
 {
           char i;

           /* do something intelligent here */
           i = func_foo (argp);

           printf( “Hello world! my result is %d \n”, i);

           return i;
 }
```

## Small single-SPE models – PPE controlling program

```
/* the spe image handle supplied by CESOF layer */
extern spe_program_handle spe_foo;


int main()
{
        int rc, status;
        speid_t spe_id;

        /* load & start the spe_foo program on an allocated spe */
        spe_id = spe_create_thread (0, &spe_foo, 0, NULL, -1, 0);

        /* wait for spe prog. to complete and return final status */
        rc = spe_wait (spe_id, &status, 0);

        return status;
}
```

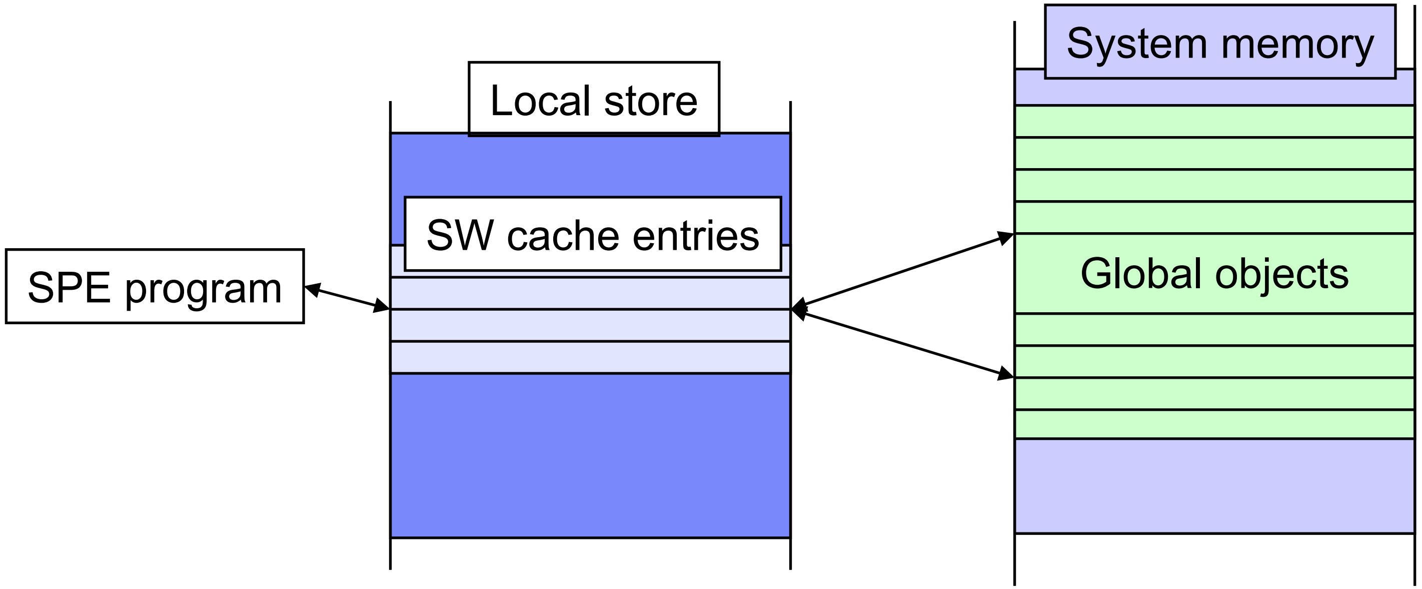
# Large single-SPE programming models

- Data or code working set cannot fit completely into a local store
- The PPE controlling process, kernel, and libspe runtime establish the system memory mapping as SPE's secondary memory store
- The SPE program accesses the secondary memory store via its software-controlled SPE DMA engine - Memory Flow Controller (MFC)



## Large single-SPE programming models – data cache

- **System memory as secondary memory store**
  - Manual management of data buffers
  - Automatic software-managed data cache
    - Software cache framework libraries
    - Compiler runtime support



## Linux on Cell

# Programming Hints

# CBE General Programming Practices

- **Offload as much work onto the SPEs as possible**
  - Use the PPE as the control plane processor
    - Orchestrate and schedule the SPEs
    - Assist SPEs with exceptional events
  - Use SPEs as data plane processors
- **Partitioning and Work allocation strategies**
  - Algorithmic
    - Possible self regulated work allocation
  - Work queues
    - Single – SPE arbitrated

      Works well when the work task are computationally significant and variable.
    - Multiple – PPE distributed

      Works well when time to complete the task is predictable.
  - Consider all domains in which to partition the problem.
    - Ex: Video application
      - Space – partition scan lines or image regions to a different SPE
      - Time – partition each frame to a different SPE

## CBE General Programming Practices (cont)

- Minimize atomic operations and synchronization events
- Accommodate potential data type differences
  - SPE is ILP32 (32-bit integers, longs, and pointers)
  - PPE is either ILP32 or LP64 (64-bit longs and pointers)

# PPE Programming Practices

- Utilize multi-threading capabilities of the PPE
  - When there are lots of L1 and L2 cache misses
    - Pointer chasing
    - Scatterd array / vector accesses
  - When there is poorly pipelined floating-point operations
    - Lots of dependencies
    - Loops can not be effectively unrolled
    - Can not be SW pipelined

- Self manage cache using data cache instructions
  - PPE supports two forms of dcbt instructions
    - Classic (th=0)
      - Prefetches a single cache line from memory into the L2 and L1
    - Enhanced (th=8)
      - Prefetches up to a page of memory into the L2.
  - The VMX's data stream instructions are NoOp'd and should not be used.

## SPE Programming Practices

- SPE programmer managed data transfers
  - Forces the programmer to be aware of all data accesses.
  - Encourages thinking about data access patterns.
  - Example 16 M-point FFT

    **Problem:**

    - Traditional FFT requires n*$\log_2(n)$ passes through the data.
    - Stages must be performed sequentially.

    memory bound

    **Solution**:

    - Utilized a variation of the *stride-by-1* algorithm proposed by David H. Bailey based upon Stockham's self-sorting FFT.
    - Processed 8 butterfly stages at once.
    - Reduced data accesses to 1/8th.

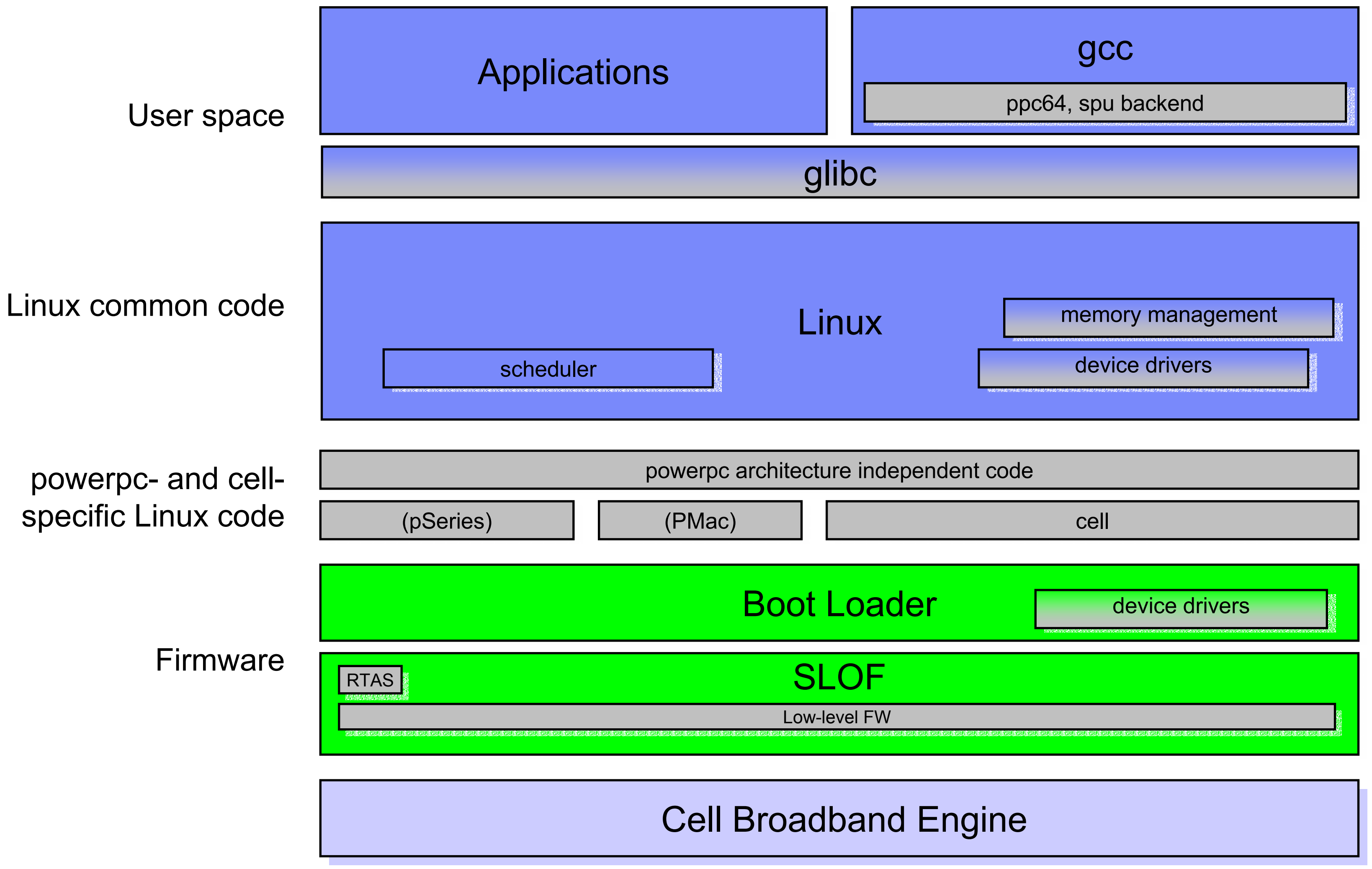
## Performance hints and tips

- Use local memory and local SPEs (via NUMA control and SPE affinity API) whenever possible to avoid performance impact of cache coherence protocol
- Implement communication patterns minimizing contention for EIB resources (ring segments, ramps,...)
- Avoid synchronous access to system memory to avoid contention
- Implement time critical code in assembler
- Use prefetching and double buffering techniques to hide memory latency

## Linux on Cell

Components and Ecosystem

# Cell Software Stack



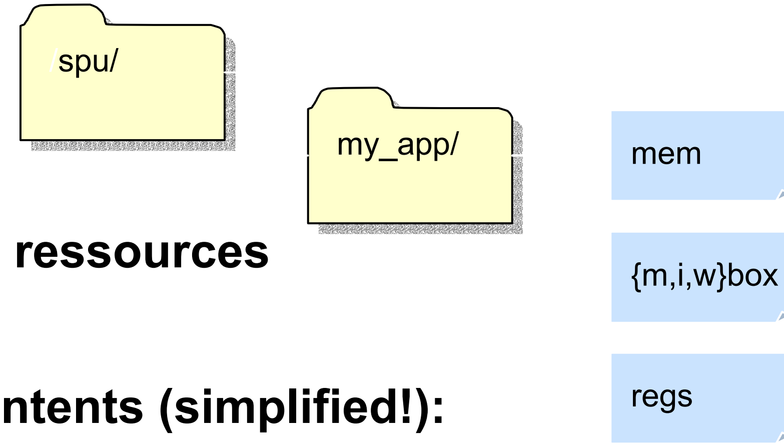
## Linux on Cell – Components

- “Cell“: a new platform in the powerpc architecture
  - As are pSeries, PMac, Maple
  - Running 64 bit
- Development on latest kernel
  - Most of the code is in the kernel since 2.6.14-rc1
- SPE support in virtual file system
- SPE compiler, debugger, runtime environment
- Hardware support
  - Interrupt controller, I/O Memory Management Unit (IOMMU), RTAS, device drivers

## Linux on Cell – Ecosystem

- Specifications of the “Cell Broadband Engine Architecture“
- IBM Full System Simulator
- SDK “Samples and Libraries“
- XLC compiler
- Kernel and GNU toolchain

## SPE support on kernel level



- **Virtual filesystem provides access to SPE ressources**
  - File operations manipulate SPEs
- **Subdirectories represent virtual SPEs. Contents (simplified!):**
  - mem (read/write, mmap, async I/O)
  - mbox, ibox, wbox (read/write, poll)
  - regs (read/write)
- **Hybrid threads: SPE code runs while proxy PPE thread blocks**
  - Memory protection for DMA transfers corresponding to PPE address space
  - SPU system calls executed by PPE proxy thread

## Exploiting SPEs: task based abstraction

- **PPE proxy thread controls SPE context**
- **PPE and SPE calls for**
  - Mailboxes
  - DMA
  - Events
- **Simple spu runtime environment (newlib)**
- **A lot of library extensions**
  - Encryption, signal processing, math operations

→**APIs provided by user space library**

## SPE exploitation – PPE programming interfaces

- **SPE Runtime Management Library ("libspe")**
  - Thread management interfaces
    - spe_open_image, spe_create_thread, spe_wait, spe_kill, spe_get_event, ...
  - Indirect access to Memory Flow Control (MFC) features
    - spe_mfc_get, spc_read_out_mbox, spe_write_signal, spe_read_tag_status, …
  - Intended to be portable across operating systems
  - On Linux, implemented on top of spufs kernel API
- **Implementation of spe_create_thread**
  - Allocate virtual SPE context in spufs (spu_create)
  - Load SPE application code into context
  - Start PPE thread using pthread_create
  - Within new thread, commence SPE execution (spu_run)

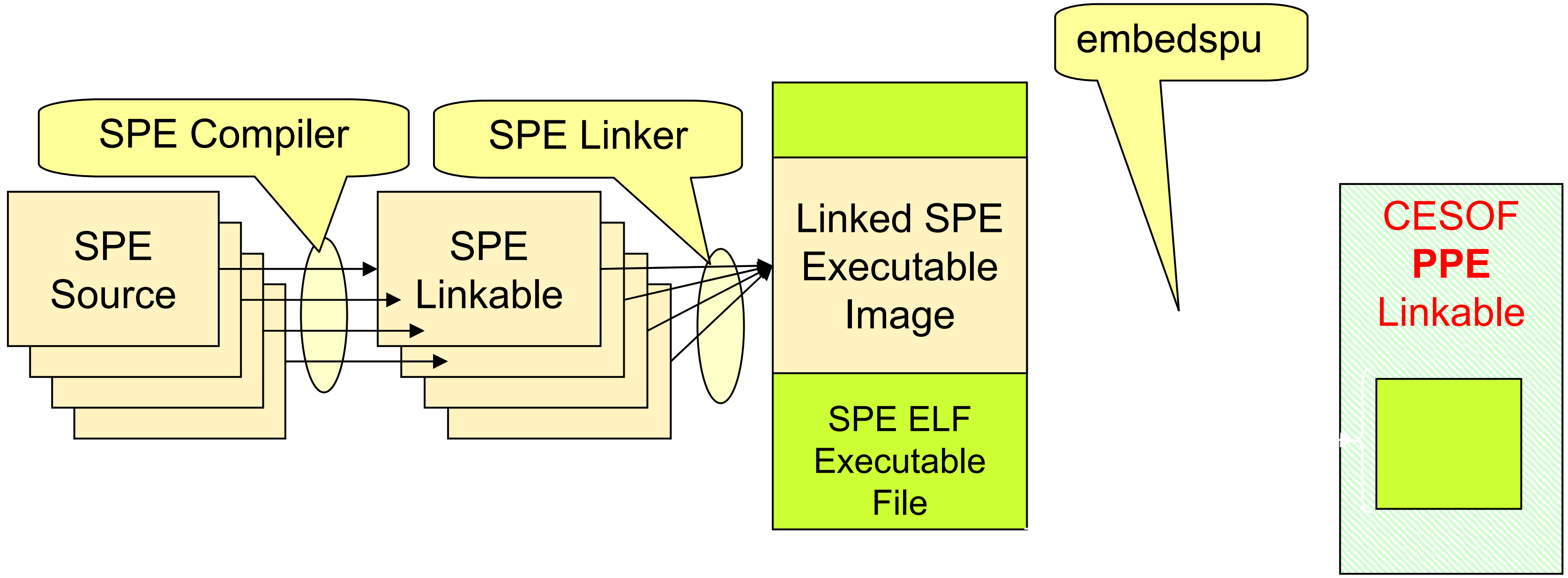
## Exploiting SPEs: direct mapping in problem space

- **SPE Library interface spe_get_ps_area()**
  - SPU registers are memory mapped into user address space of the controlling PPE program
  - Target SPE thread must have been created with SPE_MAP_PS
- **Applications can manipulate processor registers to control and perform MFC operations**
  - Initiate DMA transfers
  - send messages to mailboxes
  - send signals
- **No additional library calls need to be made**
- **Can also be used to perform SPE to SPE communications**

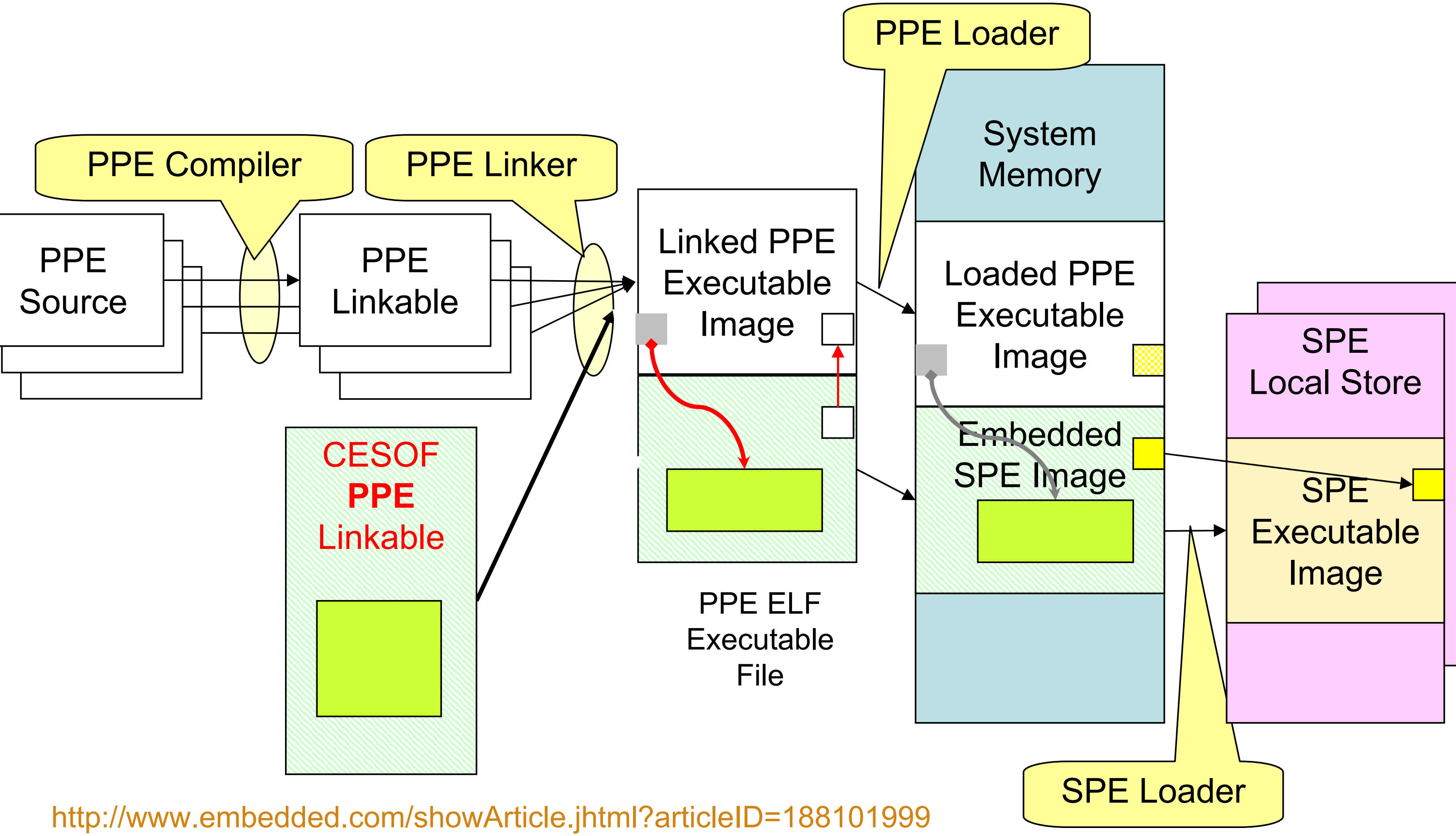
## gcc support

- **PPE: handled by rs6000 back end**
  - Processor-specific tuning, pipeline description
- **SPE: new spu back end**
  - Built as cross-compiler
  - Handles vector data types, intrinsics
  - Middle-end support: branch hints, aggressive if-conversion
  - Future: gcc port exploiting auto-vectorization?
- **cell: no special spu support today**
  - Future: single source mixed-architecture compiler?

## CESOF object



# Combined CBE executable



http://www.embedded.com/showArticle.jhtml?articleID=188101999

## Debugging Cell applications

- **SPE-only debugger**
  - Attach just to single SPE thread of a process
  - Use spufs instead of ptrace to manipulate state
- **GDB child-session support**
  - Master PPE GDB spawns child-GDB sessions for SPEs
  - Unified user interface provides access to all GDBs
- **GDB multi-architecture target**
  - Single GDB session to debug full Cell process
  - Nontrivial implementation issues ...

## Linux on Cell

History and Present Status

# Linux on Cell – initial disclosure and distro: 2005

- Initial disclosure – 2005/04/28

  http://ozlabs.org/pipermail/linuxppc64-dev/2005-April/003878.html

  - new platform (called BPA those days)
  - spufs
  - support for EIC, IIC, IOMMU, PCI
  - drivers for console, NVRAM, watchdog
  - libspe (called libspu those days) – 2005/05/13
  - Gigabit Ethernet – 2005/06/28
- Initial distro – 2005/07/29

  http://www.bsc.es/projects/deepcomputing/linuxoncell/

  - Fedora Core 3-based RPMs
- GNU toolchain from Sony to BSC and BSC to public – 2005/10
  - spu-gcc 3.4.1
  - spu-binutils 2.15

## Linux on Cell – Limited Availability: 2005/09/30

- Linux on Cell – base
  - PPC 64 kernel 2.6.13 with base Cell support
  - Spufs
  - Device support: IOMMU, EIC/IIC, Gigabit Ethernet, console, flash update, NVRAM, PCI, watchdog

    **`http://ozlabs.org/pipermail/linuxppc64-dev/2005-September/005815.html`**
  - API for SPE enablement: Load and execute code on SPE, SPE-initiated DMA, mailboxes, signals → libspe

    **`http://ozlabs.org/pipermail/linuxppc64-dev/2005-October/005860.html`**
- Basic toolchain
  - gcc 3.4.1 for SPE
  - Binutils 2.15
- Packaged for and tested with Fedora Core 3
  - Distribution through Barcelona Supercomputing Center

    **`http://www.bsc.es/projects/deepcomputing/linuxoncell/`**

## Linux on Cell – SDK 1.0: 2005/11/09

- First version of a basic, but fully functional Cell dev't kit
  - Fixes in Linux kernel (2.6.14), libspe and GNU toolchain
  - gdb
  - xlc
  - Full System Simulator
  - C99 environment on SPEs
  - Samples and libraries
  - Fedora Core 4-based RPMs
  - scripts for full development environment on Intel
  - CBEA specification available

## Linux on Cell – SDK 1.1: 2006/07/14

- Linux kernel (2.6.16)
- Dual BE support
- improved GNU (4.0.2) and XLC/C++ tool chains
  - C++ support added to XL C compiler for PPU and SPU applications
- Binutils upgraded (2.16.1)
- Support for GDB server running in both PPEs and SPEs
- NUMA support
- Quaternion Julia Set sample
- Improved installation using revamped process and RPMs
- Single ISO image is available
- Simulator host and target FC5

## Resources

- IBM developerWorks
  - http://www-128.ibm.com/developerworks/power/Cell/
- IBM developerWorks Library
  - http://www-306.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine
- IBM alphaWorks and CBE SDK
  - http://www.alphaworks.ibm.com/topics/Cell
- Architecture Documents
  - http://www-128.ibm.com/developerworks/power/Cell/downloads_doc.html
- Articles
  - http://www-128.ibm.com/developerworks/power/Cell/articles.html
- IBM developerWorks Cell BE forum
  - http://www-128.ibm.com/developerworks/forums/dw_forum.jsp?forum=739&cat=46
- CBE kernel release site
  - http://www.bsc.es/projects/deepcomputing/linuxonCell/?S_TACT=105AGX16&S_CMP=DWPA
- CBE kernel mailing list
  - https://ozlabs.org/mailman/listinfo/cbe-oss-dev