Lecture 18: Cell/B.E. Introduction: Software Development

Kamesh Madduri
Lecture Outline

• Cell software development environment
  – Standards
    • Language extensions, SPU ABI, and Linux reference ABI
  – Development Environment
    • Code development tools, debug tools, performance tools, and other miscellaneous tools
  – Execution Environment
    • Linux and CBE extensions, SPE management library, SPE optimized function libraries, system level simulator, samples, workloads, and demos

• Cell Application Affinity
Lecture Sources

• Cell BE Training Series Track 1: Cell Software Development Overview
  – http://www.power.org/resources/devcorner/cellcorner/CellTraining_Track1
  – Barry Minor, “Cell Application Affinity”

• Georgia Tech Cell/B.E. Programming workshop slides
  – http://www.cc.gatech.edu/~bader/CellProgramming.html
  – Hema Reddy, Cell development tutorials
Cell BE Resources

- developerWorks: Cell BE resource center (SDK, IBM whitepapers, reports, forum)
- Power.org: Cell developer corner (links)
  - [http://www.power.org/resources/devcorner/cellcorner/](http://www.power.org/resources/devcorner/cellcorner/)
- Georgia Tech (Sony-Toshiba-IBM Cell Center of Competence) [http://sti.cc.gatech.edu/](http://sti.cc.gatech.edu/)
  - wiki (technical papers)
    - [http://wiki.cc.gatech.edu/cellbuzz](http://wiki.cc.gatech.edu/cellbuzz)
  - cellbuzz cluster (QS20 Blades)
    - apply for accounts!
Cell Alpha Software Development Environment

Distributed on IBM alphaworks & Barcelona Super Computer sites

- Documentation
- Code Dev Tools
- Samples Workloads Demos
- Debug Tools
- SPE Optimized Function Libraries
- Performance Tools
- SPE Management Lib
- Miscellaneous Tools
- Linux PPC64 with CBE Extensions
- Standards: Language extensions
- SPU ABI, Linux Reference ABI

Development Environment

Execution Environment
CBE Standards

- **Application Binary Interface Specifications**
  - Defines such things as data types, register usage, calling conventions, and object formats to ensure compatibility of code generators and portability of code.
    - SPE ABI
    - Linux for CBE Reference Implementation ABI

- **SPE C/C++ Language Extensions**
  - Defines standardized data types, compiler directives, and language intrinsics used to exploit SIMD capabilities in the core.
  - Data types and Intrinsics styled to be similar to Altivec/VMX.

- **SPE Assembly Language Specification**
System Level Simulator - systemsim

- **Cell BE – full system simulator**
  - Uni-Cell and multi-Cell simulation
  - User Interfaces – TCL and GUI
  - Cycle accurate SPU simulation (pipeline mode)
  - Emitter facility for tracing and viewing simulation events
SW Stack in Simulation

- Application Source Code
- Programming Tools
  - Programming Model
  - OpenMP
  - MPI
- Compilers
- Executables
- Runtime and libraries
- System Software: Hypervisor, Linux/PPC or K42

**CellSim:** Simulation of hardware

**Traces**
Cell Simulator Debugging Environment
Linux on CBE

- Provided as patched to the 2.6.15 PPC64 Kernel
  - Added heterogeneous lwp/thread model
    - SPE thread API created (similar to pthreads library)
    - User mode direct and indirect SPE access models
    - Full pre-emptive SPE context management
    - spe_ptrace() added for gdb support
    - spe_schedule() for thread to physical SPE assignment
      - currently FIFO – run to completion
  - SPE threads share address space with parent PPE process (through DMA)
    - Demand paging for SPE accesses
    - Shared hardware page table with PPE
  - PPE proxy thread allocated for each SPE thread to:
    - Provide a single namespace for both PPE and SPE threads
    - Assist in SPE initiated C99 and POSIX-1 library services
  - SPE Error, Event and Signal handling directed to parent PPE thread
  - SPE elf objects wrapped into PPE shared objects with extended gld
  - All patches for Cell in architecture dependent layer (subtree of PPC64)
**Operating System Runtime Strategy**

- **Heterogeneous Multi-Threading Model**
  - PPE Threads, SPE Threads
  - SPE DMA EA = PPE Process EA Space
  - OS supports Create/Destroy SPE tasks
  - Atomic Update Primitives used for Mutex
  - SPE Context Fully Managed
    - Context Save/Restore for Debug
    - Virtualization Mode (indirect access)
    - Direct Access Mode (realtime)
  - OS assignment of SPE threads to SPEs
    - Programmer directed using affinity mask
  - SPE Compilers use SPE Management Lib.

Cell AwareOS (Linux)
SPE Virtualization / Scheduling Layer (m->n SPE threads)
Existing PPE tasks/threads
New SPE tasks/threads

Application Source & Libraries

PPE object files
SPE object files

Physical PPE

MT1 MT2
Physical SPEs

PPE

SPE
SPE
SPE
SPE
SPE
SPE
SPE
SPE
SPE
SPE

© 2006 IBM Corporation
CBE Extensions to Linux

**PPC32 Apps.**  
**Cell32 Workloads**  
- SPE Management Library – spe tasks
  - spe create group, create thread
  - spe get/set affinity, get/set context
  - spe get event, get group, get_ls, get_ps_area, get_threads
  - spe get / set priority, get policy
  - spe group defaults, group max
  - spe kill / wait
  - spe open / close image
  - spe write signal, read_in_mbox, write_out_mbox, read_mbox status
  - ppe initiated spe DMAs

**Cell64 Workloads**  
- SPUFS Filesystem /spu/thread#/  
  - open, read, write, close
  - mem – problem state access to Local Storage
  - signal1 – direct application access to Signal 1
  - signal2 – direct application access to Signal 2
  - ctrl – direct application access to SPE controls, DMA Queues, mailboxes

**PPC64 Apps.**  
- SPUFS Filesystem /spu/thread#/  
  - open, mmmap, close
  - mem – problem state access to Local Storage
  - signal1 – direct application access to Signal 1
  - signal2 – direct application access to Signal 2
  - ctrl – direct application access to SPE controls, DMA Queues, mailboxes

**64-bit Linux Kernel**  
- Cell BE Architecture Specific Code
  - Multi-large page, SPE event & fault handling, IIC & IOMMU support

**Firmware / Hypervisor**  
- Cell Reference System Hardware
SPE Management Library

- SPEs are exposed as threads
  - SPE thread model interface is similar to POSIX threads.
  - SPE thread consists of the local store, register file, program counter, and MFC-DMA queue.
  - Associated with a single Linux task.
  - Features include:
    - **Threads** - create, groups, wait, kill, set affinity, set context
    - **Thread Queries** - get local store pointer, get problem state area pointer, get affinity, get context
    - **Groups** - create, set group defaults, destroy, memory map/unmap, madvise.
    - **Group Queries** - get priority, get policy, get threads, get max threads per group, get events.
    - **SPE image files** - opening and closing

- SPE Executable
  - Standalone SPE program managed by a PPE executive.
  - Executive responsible for loading and executing SPE program. It also services assisted requests for I/O (eg, fopen, fwrite, fprintf) and memory requests (eg, mmap, shmat, ...).
Optimized SPE and Multimedia Extension Libraries

- Standard SPE C library subset
  - optimized SPE C99 functions including stdlib c lib, math and etc.
  - subset of POSIX.1 Functions – PPE assisted
- Audio resample - resampling audio signals
- FFT - 1D and 2D fft functions
- gmath - mathematic functions optimized for gaming environment
- image - convolution functions
- intrinsics - generic intrinsic conversion functions
- large-matrix - functions performing large matrix operations
- matrix - basic matrix operations
- mpm - multi-precision math functions
- noise - noise generation functions
- oscillator - basic sound generation functions
- sim – simulator only function including print, profile checkpoint, socket I/O, etc …
- surface - a set of bezier curve and surface functions
- sync - synchronization library
- vector - vector operation functions
Code Development Tools

- **GNU based binutils**
  - From Sony Computer Entertainment
  - gas SPE assembler
  - gld SPE ELF object linker
    - ppu-embedspu script for embedding SPE object modules in PPE executables
  - misc bin utils (ar, nm, ...) targeting SPE modules

- **GNU based C/C++ compiler targeting SPE**
  - From Sony Computer Entertainment
  - retargeted compiler to SPE
  - Supports common SPE Language Extensions and ABI (ELF/Dwarf2)

- **Cell Broadband Engine Optimizing Compiler (executable)**
  - IBM XLC C/C++ for PowerPC (Tobey)
  - IBM XLC C retargeted to SPE assembler (including vector intrinsics) - highly optimizing
  - Prototype CBE Programmer Productivity Aids
    - Auto-Vectorization (auto-SIMD) for SPE and PPE Multimedia Extension code
    - spu_timing Timing Analysis Tool
Debug Tools

- **GNU gdb**
  - Multi-core Application source level debugger supporting PPE multithreading, SPE multithreading, interacting PPE and SPE threads
  - Modes of debugging SPU threads
    - Standalone SPE debugging
    - Attach to SPE thread
      - Thread ID output when SPU_DEBUG_START=1
SPE Performance Tools (executables)

- **Static analysis (spu_timing)**
  - Annotates assembly source with instruction pipeline state

- **Dynamic analysis (CBE System Simulator)**
  - Generates statistical data on SPE execution
    - Cycles, instructions, and CPI
    - Single/Dual issue rates
    - Stall statistics
    - Register usage
    - Instruction histogram

- **pmcount**
  - Tool to access to HW performance counters
  - Not currently public – being ported to oprofile interface standard
Miscellaneous Tools – IDL Compiler

- **PPE application**
- **SPE function**
- **.idl**

**Written by programmer**

**Generated by IDL Compiler**

**PPE Compiler**
- **ppe_stub.c**
- **stub.h**

**SPE Compiler**
- **spe_stub.c**

**Call @ run-time**

**PPE binary**

**SPE binary**
Sample Source

- cesof - the samples for the CBE embedded SPU object format usage
- spu_clean - cleans the SPU register and local store
- spu_entry - sample SPU entry function (crt0)
- spu_interrupt - SPU first level interrupt handler sample
- spulet - direct invocation of a spu program from Linux shell
- sync
- simpleDMA / DMA
- tutorial - example source code from the tutorial
- SDK test suite
  - 57 tests
  - Exercising all packages in the SDK
Workloads

- FFT16M – optimized 16 M point complex FFT
- Oscillator - audio signal generator
- Matrix Multiply – matrix multiplication workload
- VSE_subdiv - variable sharpness subdivision algorithm
Workloads / Demos

- Numerous code samples provided to demonstrate system design constructs
- Complex workloads and demos used to evaluate and demonstrate system performance

Terrain Rendering Engine

Geometry Engine

Physics Simulation

Subdivision Surfaces
Subsystem Sample – Geometry Engine

- OpenGL-like geometry engine
  - Geometry processing is offloaded to compile-time configurable SPE “vertex shader”
  - User Queue communication model consisting of 4KB blocks for SPU command requests with command headers in SPE Mailbox FIFO
  - Not released in public SDK
CBE Software Developers Kit Release

- **alphaWorks site: All ILA and CPL packages**
  - IBM Full System Simulator for the CBE Processor
    - Executable - ILA for early release program
  - IBM XL C Alpha Edition for the CBE Processor
    - Executable - ILA for early release program
  - IBM CBE Software Samples and Libraries
    - Source - CPL v1.0
  - IBM SPU instruction timing tool
    - Executable - ILA for early release program

- **Barcelona Super Computer Center (GPL,LGPL)**
  - gcc and binutils for the Cell Broadband Engine
    - From Sony Computer Entertainment
  - Cell Broadband Engine SPE Management Library
  - Linux Patches for Cell Broadband Engine
  - SDK Installation Script

- **IBM developerWorks (CBE documentation)**
  - Getting Started with CBE – Installing the CBE Programming Environment
  - Cell Broadband Engine Standards and Documentation
    - Linux CBE Reference Implementation ABI Specification
    - CBE Registers
    - CBE Architecture v1.0
    - SPU Instruction Set Architecture v1.0
    - SPU ABI Specification v1.4
    - SPU Assembly Language Specification v1.3
    - SPU C/C++ Language Extension Specification v2.1
    - PowerPC Architecture Books 1-3 v2.02
    - PowerPC Vector/SIMD Multimedia Extension Technology PEM
Testing Environment for Public SDK

- **Simulator only**
- **Hardware only**
- **Same binary**

<table>
<thead>
<tr>
<th>Native Dev Env.</th>
</tr>
</thead>
<tbody>
<tr>
<td>System tests</td>
</tr>
<tr>
<td>Sample / Libraries / Workloads</td>
</tr>
<tr>
<td>Passthru</td>
</tr>
<tr>
<td>Libspe</td>
</tr>
<tr>
<td>Kernel (2.6.14)</td>
</tr>
<tr>
<td>Simulator Firmware</td>
</tr>
<tr>
<td>Sim. devices + drivers</td>
</tr>
<tr>
<td>Sys. SIM / x86</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Full App Env.</th>
</tr>
</thead>
<tbody>
<tr>
<td>System tests</td>
</tr>
<tr>
<td>Sample / Libraries / Workloads</td>
</tr>
<tr>
<td>(Sys. SIM subset)</td>
</tr>
<tr>
<td>(Sys. SIM subset)</td>
</tr>
<tr>
<td>Full App Env.</td>
</tr>
<tr>
<td>Libspe</td>
</tr>
<tr>
<td>Kernel (2.6.14)</td>
</tr>
<tr>
<td>Cell Blade Firmware</td>
</tr>
<tr>
<td>IDE/Ethernet devices + drivers</td>
</tr>
<tr>
<td>CBE on Cell Blade</td>
</tr>
</tbody>
</table>

**Cell Simulator**

**Native platform**
Cell Broadband Engine

Conventional Power CPU + 8 SPEs
Cell Broadband Engine

Four SPEs fit into the area of one conventional CPU
The SPE Advantage

**SPE**

- 128 Vector Registers
- 6 Cycle latency floating point pipeline
- 16 Outstanding loads
- Single cycle access to working set (Deterministic Local Store)
- Asynchronous DMA access to main memory
- 25 GB/sec to main memory

**Conventional CPU**

- 32 Vector Registers
- 10 Cycle latency floating point pipeline
- 6 Outstanding loads
- Multi cycle access to working set (Nondeterministic Cache)
- Synchronous Load/Store access to main memory
- 9 GB/sec to main memory
Cell Software design considerations

- Two Levels of Parallelism
  - Regular vector data that is SIMD-able
  - Independent tasks that may be executed in parallel

- Computational
  - SIMD engines on 8 SPEs and 1 PPE
  - Parallel sequence to be distributed over 8 SPE / 1 PPE
  - 256KB local store per SPE usage (data + code)

- Communicational
  - DMA and Bus bandwidth
    - DMA granularity – 128 bytes
    - DMA bandwidth among LS and System memory
  - Traffic control
    - Exploit computational complexity and data locality to lower data traffic requirement
  - Shared memory / Message passing abstraction overhead
  - Synchronization
  - DMA latency handling
Typical Cell software development flow

- Algorithm complexity study
- Data layout/locality and Data flow analysis
- Experimental partitioning and mapping of the algorithm and program structure to the architecture
- Develop PPE Control, PPE Scalar code
- Develop PPE Control, partitioned SPE scalar code
  - Communication, synchronization, latency handling
- Transform SPE scalar code to SPE SIMD code
- Re-balance the computation / data movement
- Other optimization considerations
  - PPE SIMD, system bottle-neck, load balance
Things that work extremely well today

- Problem can be re-coded
- Predictable non-trivial memory access pattern
  - Can build scatter-gather lists
- Problem can benefit from SIMD
- Focus on 32b float, or <=32b integer

- Examples:
  - FFTs (best result about 100GFlops)
  - Terrain Rendering Engine
  - Volume rendering

- Typical code is double-buffered gather-compute-scatter
Things that work well today

• Compute bound codes
• Small enough to be rewritten
• Main datatype is 32b float or <= 32b Int
• Benefits from SIMD

• Examples:
  – Crypto codes (RSA, SHA, DES, etc. etc. etc.)
  – Media codes (MPEG 2, MPEG 4, H.264, JPEG)
  – … many many others …
Things likely to work well

• Library .. Device/API based applications
  – Graphics and physics and sound and …
• Scientific codes … library based
  – No rewrite
  – If granularity is ok
Question marks

• Can a compiler based approach, without restructuring code specifically for the SPEs result in a chip-level advantage?
  – About 3-4x more SPEs in same area or power
  – But, have to compiler manage local store

• Interesting benchmarks: SpecFP, MediaBench, EEMBC, etc.
  – New more explicitly parallel benchmarks?

• Would you ever use an SPE for a SpecInt-type workload?
## Cell application affinity

### Cell Broadband Engine
- Non-homogeneous coherent multi-Processor
  - Dual-threaded control-planes processor
  - 8 independent data-plane processors
  - Thread-level parallelism
- SIMD processing architecture
  - 128-entry, 128-bit register files
  - Pipelined execution units
  - Branch hint
  - Data-level parallelism
- Rich integer instruction set
  - Word, halfword, byte, bit
  - Boolean
  - Shuffe
  - Rotate, shift, mask
- Single-precision floating point
- Double-precision floating point
- 256KB SPU local stores
  - Asynchronous DMA/main memory interface
  - Channel interface
  - Single-cycle load/store to/from registers
- High-bandwidth internal bus
  - 96 bytes transferred per clock
  - 100M+ outstanding transfers supported
- Coherent bus interface
  - Up to 30GB/s out, 25 GB/s in
  - Direct attach of another Cell
  - Can be configured as non-coherent
- Non-coherent bus interface
  - Up to 10GB/s out, 10 GB/s in
- 25+ GB/s XDR memory interface

### Accelerated Functions
- Signal processing
- Image processing
- Audio resampling
- Noise generation
- Sound oscillation
- Digital filtering
- Curve and surface evaluation
- FFT
- Matrix mathematics
- Vector mathematics
- Game Physics / Physics simulation
- Video compression / decompression
- Surface subdivision
- Transform-light
- Graphics content creation
- Security encryption / decryption
- Pattern matching
- Language parsing
- TCP/IP offload
- Encoding / decoding
- Parallel processing
- Real time processing
- ...

### Target Applications
- Medical imaging / visualization
- Drug discovery
- Petroleum reservoir modeling
- Seismic analysis
- Avionics
- Air traffic control systems
- Radar systems
- Sonar systems
- Training simulation
- Targeting
- Defense and security IT
- Surveillance
- Secure communications
- LAN/MAN Routers
- Network processing
- XML and SSL acceleration
- Voice and pattern recognition
- Video conferencing
- Computational chemistry
- Climate modeling
- Data mining and analysis
- Media server
- Digital content creation
- Digital content distribution
- ...
Target opportunities for the Cell blade

- Aerospace & Defense
  - Signal & Image Processing
  - Security, Surveillance
  - Simulation & Training, …

- Petroleum Industry
  - Seismic computing
  - Reservoir Modeling, …

- Public Sector / Gov’t & Higher Educ.
  - Signal & Image Processing
  - Computational Chemistry, …

- Finance
  - Trade modeling

- Medical Imaging
  - CT Scan
  - Ultrasound, …

- Industrial
  - Semiconductor / LCD
  - Video Conference

- Consumer / Digital Media
  - Digital Content Creation
  - Media Platform
  - Video Surveillance, …

- Communications Equipment
  - LAN/MAN Routers
  - Access
  - Converged Networks
  - Security, …

- Communications Equipment
  - LAN/MAN Routers
  - Access
  - Converged Networks
  - Security, …
Visualization via Ray-tracing

- **Turner Whitted** – First recursive ray-tracer showing reflections and refractions (1979)
- **Cars** - First Pixar movie to deploy ray-tracing (2006)
- 27 Years later - Computational Power has caught up
- Interactive ray-tracing is around the corner
Interactive Ray-tracing

- Preferred Algorithm for Film and Special Effects
- Renewed interest from Interactive Graphics Community
- Global Illumination
- Rendering time scales logarithmically with scene complexity
- Scales well on multi-core processors
- Mathematically elegant Algorithmically simple
Why Ray-tracing?
Visualization for E Commerce

• Server Based Rendering for E commerce
• Visual Blades
Real-time Visualization of Huge Models
Real-time Visualization of Digital Mock-ups

- 350 M triangle model of Boeing 777
Volume Rendering for Medical
Huge Virtual Worlds

- Server side rendering delivered as compressed image streams to handhelds
Other Commercial Uses

- Engineering Design: Automotive, Aerospace, Consumer, Industrial
- Architecture: Digital Video Animation and Special Effects
- Computer Games
- Movies: Digital Video Animation and Special Effects
- Engineering Design: Computational Science and Engineering
TRE Rendered Output
Terrain Rendering Engine (TRE) System Configurations

Cell Blade System | Network | Clients

Wireless Access
Conclusions

• Cell ushers in a new era of leading edge processors optimized for digital media and entertainment
• Desire for realism is driving a convergence between supercomputing and entertainment
• New levels of performance and power efficiency beyond what is achieved by PC processors
• Responsiveness to the human user and the network are key drivers for Cell
• Cell will enable entirely new classes of applications, even beyond those we contemplate today
Summary

What you should have learnt in this class:

• Cell/B.E. Software Development Environment
• Cell Application Affinity

• Next class: Cell programming kick-off, hands-on demo
  – Download the Cell SDK 2.1 VMware image
    • http://www.cc.gatech.edu/~bader/CellProgramming.html
  – Sign up for a CellBuzz blade cluster account
    • http://wiki.cc.gatech.edu/cellbuzz