CS4803DGC Design and Programming of Game Console
Spring 2011
Prof. Hyesoon Kim
Playstation 3

- PowerPC-base Core @3.2GHz
- 1 VMX vector unit per core
- 512KB L2 cache
- 7 x SPE @3.2GHz
- 7 x 128b 128 SIMD
- GPRs 7 x 256KB SRAM for SPE
- 1 of 8 SPEs reserved for redundancy total floating point performance: 218 GFLOPS
- **GPU**: RSX @550MHz
- 1.8 TFLOPS floating point performance
- Full HD (up to 1080p) x 2 channels
- **Sound**: Dolby 5.1ch, DTS, LPCM, etc. (Cell-base processing)
- **Memory**:
  - 256MB XDR Main RAM @3.2GHz, 256MB GDDR3 VRAM @700MHz
• Sony, IBM, Toshiba Collaborative effort
• Start from 2000
• 100 x speedup of Playstaion 2
• 90-nm process with low-k dielectrics, copper interconnections
• Overcoming Memory walls and Power walls
Cell Broadband Engine Processor

L2 (512KB)  Test & Debug Logic  SPE  SPE  SPE  SPE  SPE

Power Processor Element  Element Interconnect Bus

Rambus XDRAM™ Interface  Memory Controller  I/O Controller  Rambus FlexIO™
Key Attributes

• High design frequency → low voltage and low power
• Power architecture compatibility to utilize IBM software infrastructure & experiences
• SPE: SIMD architecture. Support media/game applications
• A power & area efficient PPE
Cell Processor Block Diagram

On-chip coherent bus (up to 96 bytes per cycle)

- SXU
- LS
- DMA
- L2
- Power core
- Memory controller
- Dual Rambus XDR**
- Bus interface controller
- Rambus FlexIO**
PPE Major Units

Threads alternate fetch and dispatch cycles.
SPE Major Units

Floating-point unit
  Fixed-point unit

Permute unit
  Load/store unit
  Branch unit
  Channel unit

Result forwarding and staging
  Register file

Instruction issue unit/instruction line buffer
  On-chip coherent bus

Local store (256 KB)
  Single-port SRAM

128B read
128B write

DMA unit

14.5mm² (90nm SOI)
Announcement

- Lab #5 should be turn in T-squire.
PPE (POWER PROCESSOR ELEMENT)
PPE

- Pipeline depth: 23 stages
- Dual in-order issue
  - Almost possible combinations
  - Not the same FU
  - Not two instructions from simple vector, completed vector, vector FP, scalar FP
- 2way - SMT (issue 2 instructions from 2 threads)
- 1st level: 32KB 2nd level: 512KB
PPE

• IU (Instruction unit): instruction fetch, decode, branch, issue and completion
  – Fetch 4 instructions per cycle per thread
  – 4KB branch predictor (global + local)
    • Branch predictor (4KB 2-bit branch history + 6-bits of global history per thread)
• XU (Fixed point unit): 32-bit, 64-bit general purpose register file per thread
• VSU (A vector scalar unit): vector scalar and floating point : Have decoupled queues
SPE (SYNERGISTIC PROCESSOR ELEMENTS)
SPE Major Units

Floating-point unit
  Fixed-point unit

Permute unit
  Load/store unit
  Branch unit
  Channel unit

Result forwarding and staging
  Register file

Instruction issue unit/instruction line buffer

On-chip coherent bus

Local store (256 KB)
  Single-port SRAM

128B read
128B write

DMA unit

14.5mm² (90nm SOI)
SPE Pipeline

SPE pipeline front end

SPE pipeline back end

Branch instruction

Permute instruction

Load/store instruction

Fixed-point instruction

Floating-point instruction

IF  Instruction fetch
IB  Instruction buffer
ID  Instruction decode
IS  Instruction issue
RF  Register file access
EX  Execution
WB  Write back
SPE Local Store (LS)

- SPE load and store instructions are performed within a local address space.
- Local address space is untranslated, unguarded and noncoherent with respect to the system address space.
- LS: private memory, not a cache
  - Access time is fixed, predictable real-time behavior.
Load and Store in SPE

- Local store is a private memory
- Load/store instruction to read or write
- DMA (Direct Memory Access) unit transfers data between local store and system memory
- Translation, protection is governed by PPE
SPE Core

- SIMD RISC-style 32 bit fixed length instruction
- 2-issue core (static scheduling)
- 128 General purpose registers (both floating points, integers)
- Most instructions operates on 128bit wide data
  (2 x 64-bit, 4 x 32-bit, 8 x 16-bit, 1638-bit, and 128x1-bit)
- Operations: single precision floating point, integer arithmetic, logical, loads, stores, compares and branches
- 256KB of private memory
Meenderinck and Juurlink, Specialization of the Cell SPE for Media Applications
Even Pipe & Odd Pipe

Static scheduling:
Fetch 2 instructions
Check whether it can be done in parallel or not
If not execute in-order
Memory Space

- No O/S on SPE
- Only user mode
- Fixed delay and without exception, greatly simplifying the core design
• No cache or virtual memory:
  – Only memory SPU can directly access is
  • No Scalar unit
• Two pipeline (odd and even)
  – In general, even pipe handles math and odd pipe
DMA Engine

- Transfers are divided into 128 Bytes packets for the on chip interconnect
- Typical 128B requires 16 processor cycles
- Instruction fetch 128B (reduce the pressure to DMA)
- DMA priority
  - Commands (high) → loads/stores (16B line at a time) → instruction (prefetch), reads eight lines
  - Special instruction to force instruction fetch
LS in SPE

- 256KB,
- 256KB/16B = 2048 lines
- Minimum access size is 16B
  - Byte addressable but last 4 bits are ignored
  - Even scalar occupies all 16B
    - More packing!!

(big endian)
Effective Addresses in SPE

- SPU does not know effective addresses
- All LS is real addresses
- PPUs know what these addresses are
  - map function

Virtual Address Space
How can we launch a thread on SPE?

Listing 7.1 Display an SPU’s ID: spu_basic.c

```
#include <stdio.h>

int main (unsigned long long spe_id,
          unsigned long long argp,
          unsigned long long envp) {
    printf("Hello World! My thread id is %lld\n",
           spe_id);
    return 0;
}
```

- `spe_id`: ids for SPE execution thread
- `argp`: data from PPU
- `envp`: environmental data
- `argp` and `envp` parameters can be any 64-bit integer values, but usually for effective addresses from PPU to SPE
include <stdio.h>
#include <stdlib.h>
#include <libspe2.h>

extern spe_program_handle_t spu_basic; /* Program handle */

int main(int argc, char **argv) {
    spe_context_ptr_t ctx; /* Context */
    unsigned int entry_point; /* Start address */
    int retval; /* Return value */
    spe_stop_info_t stop_info; /* Stop info */

    /* Create the SPE Context */
    ctx = spe_context_create(0, NULL);
    if (!ctx) {
        perror("spe_context_create");
        exit(1);
    }

    /* Load the program handle into the context */
    retval = spe_program_load(ctx, &spu_basic);
    if (retval) {
        perror("spe_program_load");
        exit(1);
    }

    /* Run the program inside the context */
    entry_point = SPE_DEFAULT_ENTRY;
    retval = spe_context_run(ctx, &entry_point, 0,
                             NULL, NULL, &stop_info);
    if (retval < 0) {
        perror("spe_context_run");
        exit(1);
    }

    /* Deallocate the context */
    retval = spe_context_destroy(ctx);
    if (retval) {
        perror("spe_context_destroy");
        exit(1);
    }
    return 0;
}
Memory Synchronization

- Order of local read and write $\rightarrow$ program order
- External access: no ordering
- Weakly consistent
- Intrinsics to help
  - spu_dsync(): forces loads, stores and external accesses to complete before continuing
  - spu_sync(): forces \{spu_dsync()\}, and instruction fetches complete before continuing
  - spu_sync_c(): loads, stores, spu_sync() + channel writes
SPU stack operation

Figure 10.6  SPU stack before and after a function call
Instruction handling in SPU

- Instructions are fetched and stored in instruction buffers (ILBs)
- Store up to 32 instructions
- Reads from LS
- Sends a request to DMA
- DMA also fetches instructions: Branch misprediction, spu_sync()
Branch

• Compiler/programmer hint
  – An upcoming branch address and branch target, prefetching at least 17 instructions
• 3-source bitwise selection instruction to eliminate branch (similar to predication)
• Multi-path and select instructions
  
```c
if (a > 1) { b = func1(1)}
else {
  b = func2(x);
}
```

  `b1 = func1(x)`
  `b2 = func2(x)`

  `if (a > 1) b = b1`

• SMBTB: software managed BTB, software loads the target address into a register file.
Element interconnect Bus

• Connect DMAs

• Four rings:
  – Separate lines for data and command
  – Two carry data in the clockwise
  – PPE> SPE1> SPE3>SPE5> SPE7> IOIf1> IOIF0> SPE6> SPE4> SPE4>SPE0>MIC
  – Two carry data in the counter-clockwise
EIB

- PPE $\rightarrow$ SPE
- SPE to SPE
  - (need to check with PPE first)
- Point to Point communication

Figure 12.1 Communication infrastructure of the Cell processor

Programming the CELL Processor, Scarpino
DMA data transfer size

• 1, 2, 4, 8 and 16 Bytes and multiples of 16 bytes up to a maximum of 16KB
• Each DMA transfer always takes at least eight bus cycles regardless of data sizes

• Either polling or blocking interfaces to determine when the transfer is complete
  – Use tag id to check
  – Use one tag ID for each DMA request {single copy, block copy}
Memory Flow Controller: Channel

• Channels are unidirectional communication paths
  – Similar to FIFO fixed capacity queues
  – Read-only or write-only from SPU’s view point
Storage Domains

Channels (channel commands) 1 of 8

Local Storage (local-address space) 1 of 8

Main Storage (effective-address space)

SPE

SPU

LS

MFC

DMA Controller

MMIO Registers

Other Devices

I/O

Other SPEs

PPE

DRAM Memory
Input/Output Interface (IOIF)

• IOIF: connects the cell to external peripherals
  – e.g.) Memory interfaces
On-chip network

- Rambus XDR
- 12.8 GB/s per 32-bit memory channel (x2)
- High bandwidth support between cell processors
- IOIF: Input–output interface; BIF: broadband interface
Streaming Models

- Pipeline Programming models between PPE and SPEs
  - If all SPEs do the same amount of work, this will be efficient

Shared memory model

- SPE and PPE assume fully coherent memory space
  - All DMAs are cache coherent

Asymmetric thread runtime model
Multistage pipeline model requires very careful load balancing.

Use mail box (communication channel to the PPE or another SPE) to build producer and consumer model.
Use DMA or special register files.
Playstation 4 Rumors

- 2012? (from www.ps4game.com)
- Abandon the cell processor in the PS4 (from www.ps4game.com)

http://www.ps4playstation4.com/ps4-release-date-countdown-begins

http://www.theps4forums.com/
## Term Projects

- Feedbacks are written in t-square

<table>
<thead>
<tr>
<th>Team Name</th>
<th>Topic</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Balloon shooting</td>
<td>Nintendo DS</td>
</tr>
<tr>
<td>B</td>
<td>SkyRoads (control a ship)</td>
<td>Nintendo DS</td>
</tr>
<tr>
<td>C</td>
<td>Space Shooting game</td>
<td>Nintendo DS</td>
</tr>
<tr>
<td>D</td>
<td>Star-ship combat</td>
<td>Nintendo DS</td>
</tr>
<tr>
<td>E</td>
<td>Morphological Anti-aliasing</td>
<td>?</td>
</tr>
<tr>
<td>F</td>
<td>Classroom assistant tool</td>
<td>Nintendo DS</td>
</tr>
<tr>
<td>G</td>
<td>TETRIS</td>
<td>TEGRA 2</td>
</tr>
</tbody>
</table>

**Notes:**

- Team A: Balloon shooting on Nintendo DS
- Team B: SkyRoads (control a ship) on Nintendo DS
- Team C: Space Shooting game on Nintendo DS
- Team D: Star-ship combat on Nintendo DS
- Team E: Morphological Anti-aliasing
- Team F: Classroom assistant tool on Nintendo DS
- Team G: TETRIS on Tegra 2