The OPELL Partition/Overlay Manager

University of Delaware
CAPSL Group
Contributors

• Main Team
  – Joseph Manzano
  – Yi Jiang
  – Ge Gan
  – Ziang Hu
  – Andrew Russo
  – Guang R Gao

• Special Thanks to
  – EARTH Technologies International
  – The members of the CAPSL Research Group
Outline

• **Objective**
  • Partition Manager: the why, the what and how?
  • Cache Like Schemes: Modulus approach
  • Cache Like Schemes: LRU-like approach
  • Testbed, results, conclusions and future work
Objective

• A GNU based OpenMP CBE based
Outline

• Objective

• **Partition Manager: the why, the what and how?**
  • Cache Like Schemes: Modulus approach
  • Cache Like Schemes: LRU-like approach
  • Testbed, results, conclusions and future work
Partition Manager
Why?

• Many-core architectures are on the rise
• Old Problems, new solutions
  – Explicit memory hierarchies
  – Heterogeneous computing environments
  – More dependency on System software than hardware solutions
• System software
  – Extendable and portable solutions applied to Many-core architectures
  – Partition Manager being part of a group of system software for many core architectures
The Partition Manager

What?

A Ghost Function
- Do not [explicitly] use the stack
- The call is not visible to the programmer
- It cleans anything that it has done before and after

Function
- Loads partitions when code is needed

The Partition
- The unit in which the on-demand code will be loaded. It is composed of user functions.

A Typical Call
- int foo()
  
  ```c
  {
    bar();
  }
  ```

- int bar()
  ```c
  {
    printf("Hello");
  }
  ```

The Partition Manager Runtime Framework

Implemented in the CELL B.E. Toolchain v 1.1
Partition Manager

How?

Compiler Options: `-fcode-partition` and `-fcode-partition-manu`

Language Pragmas: `#pragma partitions` and `#pragma keep_function`

[SPU] Compiler

- Create Partitions based on size and number of (static) calls.
- Insert New Assembly Directives

[SPU] Assembler

- Generate code according to the new directives
- Calls to Partition Manager for `.pmcall` and `.funcpointer` calls

[SPU] Linker

- Create the partitions, helper structures and combine them into their respective sections
- Embed partition information in function offsets
- Partition List
- 6-bit partition id on each function offset
- Insertion of linker script parameters (buffer size and location)
Partition Manager

How?

Partition Stack
Partition List
PE’s Memory

Processing Element
Partition / Overlay Buffer
PM

Main / External Storage
Partitions
Partition Manager
How?

Function State is saved
Partition State is saved
Symbol address is decoded

Symbol Address

Original contents of PM parameter registers are saved.
PM parameters are saved
PM is called
Partition Manager

How?

3. Search the stack and send a request to load

4. Find the external Address

5. Load the partition by a DMA transfer

6. Restore function state and call the function
Partition Manager

How?

7 Return from function
   Save returning function state

8 Restore partition state
   Load partition if needed
   Restore function state

---

Main / External Storage

Processing Element

PE's Memory

P0 state

PM

... ori $8, $2, 0
    pmcall main
    brsl $1r, test_p
...

9 Return to caller
Partition Manager
How?

Processing Element

PE's Memory

Main / External Storage

ori $8, $2, 0
.pmcall main
brsl $lr, testTc
How to manage this component during runtime so that the number of DMA transfers is reduced?
Outline

• Objective
• Partition Manager: the why, the what and how?
  • **Cache Like Schemes: Modulus approach**
  • Cache Like Schemes: LRU-like approach
  • Testbed, results, conclusions and future work
Cache-Like Schemes

- **Single Buffer**
  - Trivial Case
  - One Buffer is equal to one active partition
  - Replacement Policy: Trivial $\Rightarrow$ always replace

- **Double Buffer**
  - Double buffer is a restricted case of N Buffers
  - Two buffers
  - Replacement Policy: Modular
# Dynamic Partition Types

<table>
<thead>
<tr>
<th>Partition Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Partition in which code is being executed</td>
</tr>
<tr>
<td>Inactive</td>
<td>Partition in which code was or will be executed and resides in one of the sub-buffers</td>
</tr>
<tr>
<td>Evicted</td>
<td>Partition which may or may not execute and it is not loaded yet. All partitions are of this type at the beginning</td>
</tr>
<tr>
<td>EWOR</td>
<td>Evicted With Opportunity of Reuse: Partition which was executed and it will be required in the future but was replaced by another partition.</td>
</tr>
</tbody>
</table>
Cache-Like Schemes

• N Buffers
  – Buffer is divided into N sub-buffers
  – Each sub-buffer is managed by the runtime
  – Replacement policies:
    • Modulus Approach
    • LRU-like Approach
Cache Like Schemes

• Modulus approach
  – FIFO Approach: Oldest partition gets replaced
  – Helpful with long chains of functions
  – Relocation problem
    • What happens when a returning partition gets reloaded?
    • Associated buffer
      – An EWOR partition will be loaded in the buffer which was first loaded in.
      – Prevents relocation problems but negates replacement policies for returning partitions
    – Formula: Next loading buffer \( \Rightarrow \) \( \text{tail} = (\text{tail} + 1) \mod N \)
```c
#include <stdio.h>

void f1() { f2(); }
void f2() { f3(); }
void f3() { f4(); }
void f4() { printf("-P"); }

typedef unsigned long long __ea_t;
int main(__ea_t spuid, __ea_t argp, __ea_t envp) {
    f1();
    return 0;
}
```
Side Note: Helper Structures

Partition Stack

Partition ID

Associated Buffer

Next Action

Currently in the SPE memory

Tail index

Legend:
- Partition ID
- Next Action
- Associated Buffer
- Tail Index
- Currently in the SPE memory
An Example

main calls to f1

Calling

- Partition ID
- Next Action
- Associated Buffer
- Tail Index
- Currently in the SPE memory
An Example

f1 calls f2

Calling

0  X

Partition ID  Next Action
Associated Buffer  Tail Index
Currently in the SPE memory
An Example

f2 calls f3

<table>
<thead>
<tr>
<th>Partition ID</th>
<th>Associated Buffer</th>
<th>Next Action</th>
<th>Tail Index</th>
<th>Currently in the SPE memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Calling
An Example

<table>
<thead>
<tr>
<th>Partition ID</th>
<th>Associated Buffer</th>
<th>Tail Index</th>
<th>Currently in the SPE memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
An Example

f4 returns to f3

<table>
<thead>
<tr>
<th>Partition ID</th>
<th>Associated Buffer</th>
<th>Next Action</th>
<th>Tail Index</th>
<th>Currently in the SPE memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Example

<table>
<thead>
<tr>
<th>Partition ID</th>
<th>Associated Buffer</th>
<th>Next Action</th>
<th>Tail Index</th>
<th>Currently in the SPE memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

f3 returns to f2
**An Example**

- **f2 returns to f1**
  - Partition ID: 3
  - Associated Buffer: 2
  - Next Action: 1
  - Tail Index: 1

- **currently in the SPE memory**

- **Returning**

- **EWOR**

<table>
<thead>
<tr>
<th>Partition ID</th>
<th>Associated Buffer</th>
<th>Next Action</th>
<th>Tail Index</th>
<th>Currently in the SPE memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
An Example

f1 returns to main

Evicted

Returning
• Saturation Point
  – In chemistry, when adding more solvent will not result in a higher concentration solution since all solvent molecules have been bonded
  – In our case, when sub dividing the buffer will not achieve a reduction of DMA transfers since the partitions will not fill them.
Outline

• Objective
• Partition Manager: the why, the what and how?
• Cache Like Schemes: Modulus approach
• **Cache Like Schemes: LRU-like approach**
• Testbed, results, conclusions and future work
Cache-Like Schemes

• LRU Like Approach
  – Each sub-buffer has a count
  – It decrements when a partition action takes and it does not involve the partition in that sub-buffer.
  – The smallest count is replaced
    • If a set of smallest partitions is encountered, a random one of the set is replaced
```c
#include <stdio.h>

int c(int ff){ return (ff-1); }
int b(int ff){ return ff*8; }
int a(int ff){
    int x;
    for(x = 0; x < 2; ++x){
        ff = b(ff);
    }
    return c(ff);
}

typedef unsigned long long __ea_t;
int main(__ea_t spuid, __ea_t argp, __ea_t envp){
    a();
    return 0;
}
```
An Example

main calls to a

Calling
An Example

a calls b (1)

Partition ID
Associated Buffer
Tail Index
Currently in the SPE memory

Calling

0  X

10  0

0  1

0
An Example

Returns to a (1)

<table>
<thead>
<tr>
<th>Partition ID</th>
<th>Associated Buffer</th>
<th>Tail Index</th>
<th>Currently in the SPE memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returning
An Example

Calling

Partition ID
Associated Buffer
Next Action
Tail Index
Currently in the SPE memory

a calls b (2)

0
X

10
9

1
2

0
1

0
An Example

<table>
<thead>
<tr>
<th>Partition ID</th>
<th>Associated Buffer</th>
<th>Next Action</th>
<th>Tail Index</th>
<th>Currently in the SPE memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b returns to a (2)
An Example

Calling

Partition ID  Associated Buffer  Next Action  Tail Index  Currently in the SPE memory

0  X  10  9  1  2  0

a calls c
An Example

c returns to a

Returning

Partition ID  |  Next Action
Associated Buffer  |  Tail Index
Currently in the SPE memory
An Example

a returns to main

<table>
<thead>
<tr>
<th>Partition ID</th>
<th>Associated Buffer</th>
<th>Next Action</th>
<th>Tail Index</th>
<th>Currently in the SPE memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returning
Outline

• Objective
• Partition Manager: the why, the what and how?
• Cache Like Schemes: Modulus approach
• Cache Like Schemes: LRU-like approach
• Testbed, results, conclusions and future work
Testbed

• A Playstation 3 system running YDL 5
• GCC 4.2 compiler
• CELL B.E. SDK 1.1
• A Set of Micro benchmarks
  – A set of DSP kernels
  – SPEC gzip compression utility
  – A jacobi simulation, a multi grid program, a laplace kernel, and a molecular dynamic simulator
  – A group of seven synthetic benchmarks
Results

Single versus 2- versus 4-Buffers (Modulus)

- **DSP Kernels**
  - Single, Double, Quad

- **GZIP Compression**
  - DMA Counts

- **GZIP De compression**
  - Single, Double, Quad

- **JACOBI**
  - Single, Double, Quad

- **MGRID**
  - DMA Counts

- **MD**
  - Single, Double, Quad

- **LAPLACE**
  - DMA Counts

- **Synthetic Case 7**
  - DMA Counts
Results

4-Buffer-Modulus versus 4-Buffer-LRU

DMA Counts

- GZIP C
- GZIP D
- MGRID

LRU Partition
MODULUS
Conclusions

- Adding software buffers can greatly reduce the number of code DMA in the partition manager.
- The replacement policies can have adverse or beneficial behavior according to the application behavior.
- Applications possess a “saturation” point which can be found to find an optimal N sub-buffering scheme.
Future Work

• Find a heuristic for saturation points [The Critical Path of the Partition Graph]
• Work on the buffering schemes based on size and static frequency calls
• Introduce the concept of outlining for big functions and inlining for small ones
• Investigate how the EWOR partitions might be more effectively managed

• Manzano, J, et. Al, “Towards an Automatic code layout framework.” International Workshop on OpenMP. June 3rd - June 7th, 2007. Tsinghua University, Beijing, China
"But what ... is it good for?"
* Engineer at the Advanced Computing Systems Division of IBM, 1968, commenting on the microchip.

"I'm not dumb, I just have a command of thoroughly useless information."
-- Bill Watterson, cartoonist of Calvin & Hobbes