T17 Advanced Processors: Multithreaded Processors

School of Electrical and Computer Engineering
Cornell University

revision: 2016-12-02-09-26

1 Multithreading Overview 2

2 Vertical Multithreading 3

3 Simultaneous Multithreading 6
1. Multithreading Overview

- TLP from multiprogramming (multiple applications)
- TLP from multithreaded applications
  - Run one application faster with multiple threads
  - pthreads, Cilk, TBB, OpenMP
2. Vertical Multithreading

**Multicore vs Coarse-Grain Multithreading**

**Fine-Grain Multithreading**

Hardware support to enable interleaving multiple threads on a single core at a very fine granularity.

We will discuss two variants of fine-grain multithreading:

- **Vertical Multithreading**
- **Simultaneous Multithreading (SMT)**

**Vertical Multithreading**

Switch between threads at a cycle-by-cycle granularity.

State for all thread kept in dedicated hardware.

Thread scheduling handled by hardware.
2. Vertical Multithreading

**AT CODE EXAMPLE**

```plaintext
j = thread_id;
start = j * (n/threads);
for (i = start; i < n/threads; i++)
```

**VERTICAL MULTITHREADING MICRO-ARCHITECTURE**

[Diagram showing parallel processing and thread scheduling logic]

- **Duplicate ARCH State**
- **Extra Thread Resuming Logic**

**Examples of Code Execution**

- `to: (w r1, r0(r2))`
- `to: mul r3, r1, r4`
- `to: add r2, r1, r2`
- `to: add r3, r3, r4`
- `to: add r4, r2, r4`
- `to: add r5, r3, r5`
- `to: cpy r0, r0`
- `to: add r6, r0, r6`

**Comments**

- Completely hide load/use delay latency
- Hard hide multiple raw delay latency
- Could potentially remove X -> Z bypass path!
- Hard hide branch-resolution delay latency
2. Vertical Multithreading

SCHEDULING POLICIES

1. STATIC FIXED INTERLEAVING
   - Each of N threads execute one instruction every N cycles
   - If thread is not ready to go can either:
     - Stall entire front-end
     - Insert dummy, but do not stall front-end
   - Can potentially eliminate interlocking + bypass network

2. DYNAMIC INTERLEAVING
   - Hardware keeps track of which threads are ready
   - Picks next thread to execute based on priority scheme

3. COARSE-GRANULAR HARDWARE INTERLEAVING
   - Use threads to hide occasional cache miss latency

   1. [2012-2012-2012-2012-2012]
   2. [2012-2012-2012-2012-2012]
   3. [111111111111111111111111111111111111111111111111111111111111] Cache miss
3. Simultaneous Multithreading

**Simultaneous Multithreading (SMT)**

- **OOO**
  - Quad Issue
  - Wasted Cycles

- **OOO + VMT**
  - Quad Issue
  - Wasted Issue Slots
  - IPC < 4

- **OOO + SMT**
  - Quad Issue
  - Second Thread Interleaved Cycle by Cycle

On this cycle, we are issuing four instructions from three threads at the same time.

SMT uses the fine-grained control already present in an OOO superscalar processor to allow instructions from different threads to issue at the same time.

Add multiple fetch engines to enable fetching and decoding instructions from different threads.

SQ does not know about threads, simply finds instructions that are ready to issue. These instructions may or may not be from different threads.

* SMT adapts to parallelism type
  - For applications with high TLP but no TLP, app can use entire width of the machine.
  - For applications with high TLP but less TLP, the width of the machine is shared across threads.
3. Simultaneous Multithreading

**SMT Microarchitecture**

Duplicate Arch State

IQ, RT, FL usually dynamically hard partitioned

ROB

LSQ

Pref dynamically shared

IQ, Pref, LSQ

As with vertical MT, architectural state must be duplicated

Microarchitectural state can either be

- Duplicated at design time
- Hard partitioned at boot time
- Dynamically shared at execution time

Usually need to increase size of shared data structures IQ, Pref, LSQ

Thread Scheduling

Fetch from thread with the least instructions in flight
Draw a pipeline diagram for the assembly loop to the right executing on a dual-issue IO2L microarchitecture with register renaming, memory disambiguation, perfect branch prediction, and *two SMT threads*. Draw the diagram to illustrate how both threads simultaneously execute the first iteration of the loop.

<table>
<thead>
<tr>
<th>Assembly Instruction</th>
<th>Register(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lw x1, 0(x2)</code></td>
<td></td>
</tr>
<tr>
<td><code>mul x3, x1, x4</code></td>
<td></td>
</tr>
<tr>
<td><code>sw x3, 0(x5)</code></td>
<td></td>
</tr>
<tr>
<td><code>addi x2, x2, 4</code></td>
<td></td>
</tr>
<tr>
<td><code>addi x5, x5, 4</code></td>
<td></td>
</tr>
<tr>
<td><code>addi x7, x7, -1</code></td>
<td></td>
</tr>
<tr>
<td><code>bgtz x7, loop</code></td>
<td></td>
</tr>
</tbody>
</table>