1 Multithreading Overview

2 Vertical Multithreading

3 Simultaneous Multithreading
1. Multithreading Overview

- **ILP vs DLP vs TLP**
  
  - **ILP**: Instruction-Level Parallelism
  - **DLP**: Data-Level Parallelism
  - **TLP**: Thread-Level Parallelism

- **SIMD** vs **MIMD**

---

TLP from multiprogramming (multiple applications)
TLP from multithreaded applications
- Run one application faster with multiple threads
- pthreads, GIL, 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 threads kept in dedicated hardware.

Thread scheduling handled by hardware.
2. Vertical Multithreading

**Code Example**

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

**Vertical Multithreading Micro-Architecture**

Extra threading sequencing logic

```
<table>
<thead>
<tr>
<th>PC</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>I</td>
</tr>
</tbody>
</table>
```

Count point

Complete hide load use delay latency

```
T0: lw r1, 0(r2)
T1: lw r1, 0(r2)
T2: sw r3, 0(r5)
```

Could potentially remove X->I bypass path!

```
```

New hide branch resolution delay latency

```

```
```
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 interlock and bypass network

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

3. Coarse-Grain Hardware Interleaving
   - Use threads to hide occasional cache miss latency

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>2012</td>
<td>2012</td>
<td>2012</td>
<td>2012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>2012</td>
<td>2012</td>
<td>2012</td>
<td>2012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>1111111111111111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Vertical Multithreading
3. Simultaneous Multithreading

Simultaneous Multithreading (SMT)

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

SMT uses the fine grain 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 ILP, the width of the machine is shared across threads.
3. Simultaneous Multithreading

**SMT Microarchitecture**

- IQ, RT, FL usually dynamically shared
- ROB hard partitioned
- LSQ can be shared (carefully) or hard partitioned

**Duplicate 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, PIF, 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.

```
lw r1, 0(r2)
mul r3, r1, r4
sw r3, 0(r5)
addiu r2, r2, 4
addiu r5, r5, 4
addiu r7, r7, -1
bgtz r7, loop
```