6.823 SPring15 Quiz 4 Review (L20-L24)

TA: Po-An Tsai, Hsin-Jung Yang
Transactional memory

• Atomicity (all or nothing)
  – At commit, all memory writes take effect at once
  – On abort, none of the writes appear to take effect

• Isolation
  – No other code can observe writes before commit

• Serializability
  – Transactions seem to commit in a single serial order
  – The exact order is not guaranteed
1. Eager versioning (undo-log based)
   Update memory location directly
   Maintain undo info in a log
   Fast commits
   Slow aborts

2. Lazy versioning (write-buffer based)
   Buffer data until commit in a write buffer
   Update actual memory locations at commit
   Fast aborts
   Slow commits
Conflic Detection

1. Pessimistic detection
   Check for conflicts during loads or stores

2. Optimistic detection
   Detect conflicts when a transaction attempts to commit
Reliability

• What happens if a bit flip
  – DUE : Detected Unrecoverable Error
  – SDC : Silent Data Corruption

• What structures are important
  – ACE : Architecturally Correct Execution
  – AVF : Architectural Vulnerability Factor
ACE and AVF

• Do the questions to make sure you understand how to calculate or argue for them
**VLIW: Very Long Instruction Word**

- Software (compiler) packs independent instructions into a large instruction
- Deterministic latency for all operations
  - Latency measured in ‘instructions’
- Hardware does not perform dependency checks
  - Software adds explicit NOP instructions
How to get denser packing?

- Loop unrolling
- Software pipelining

How many FP ops/cycle?

1 fadd / 8 cycles = 0.125
Loop Unrolling

- Unroll the inner loop to perform M iterations at once
  - To get more independent instructions
- Need to handle the case where the total iteration is not a multiple of M

Unroll 4 iterations:
How many FP ops/cycle?
4 fadds / 11 cycles = 0.36
Software Pipelining

- Execute different iterations in parallel

Unroll 4 iterations and then do software pipelining:
How many FP ops/cycle? 4 fadds / 4 cycles = 1
**Predicated Execution**

- Eliminate hard to predict branches with predicated execution
- Example: (p1) ADD r1, r2, r3
  - Execute (commit) add if the predicate register p1 is true

Mahlke et al, ISCA95: On average >50% branches removed
Vector Machine

- Single Instruction Multiple Data (SIMD)
  - Data level parallelism (DLP)
  - Single instruction for multiple loop iterations
  - Vector Length Register (VLR)
  - Conditional execution using the vector mask (VM) register

The basic structure of a vector architecture
Vector Arithmetic Execution

• Use deep pipeline (=> fast clock) to execute element operations
• Simplifies control of deep pipeline because elements in vector are independent (=> no hazards!)

V3 <- v1 * v2
Multiple Lanes

Lane 0:
- FP add pipe 0
- Vector registers: elements 0, 4, 8, ...
- FP mul. pipe 0

Lane 1:
- FP add pipe 1
- Vector registers: elements 1, 5, 9, ...
- FP mul. pipe 1

Lane 2:
- FP add pipe 2
- Vector registers: elements 2, 6, 10, ...
- FP mul. pipe 2

Lane 3:
- FP add pipe 3
- Vector registers: elements 3, 7, 11, ...
- FP mul. pipe 3

Vector load-store unit
Vector Instruction Parallelism

- Can overlap the execution of multiple vector instructions

6 SIMD instructions executing on 8 lanes with 32-element vector registers
Vector Chaining

- Allow a vector operation to start as soon as the individual elements of its vector source operand become available

```
LV v1
MULV v3, v1, v2
ADDV v5, v3, v4
```
Graphical Processing Units (GPU)

• GPU consists of many multithreaded SIMD processors

• Terminology
  – **Thread**: a CUDA thread, which is associated with each data element
  – **Warp**: a traditional thread that contains SIMD instructions executed on a SIMD processor. Each SIMD instruction operates with multiple (CUDA) threads (ex: 32 threads/warp).
  – **Programming abstractions**:
    • **Thread block**: A group of threads (ex: 512 threads) mapped to a SIMD processor to execute a vectorized loop.
    • **Grid**: A grid has multiple thread blocks that can be mapped to different SIMD processors and execute independently.
Multithreaded SIMD Processor

4 warps

2 lanes
### Multithreaded SIMD Processor

**Example:**
- 16 physical lanes
- 10s of warps with 32 threads per warp
- Warp scheduler issues an SIMD instruction (for a specific warp) when its elements (threads in the warp) are all ready

#### Instruction Cache

<table>
<thead>
<tr>
<th>Instruction cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warp scheduler</td>
</tr>
<tr>
<td>Scoreboard</td>
</tr>
<tr>
<td>Warp No.</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

#### Instruction Register

<table>
<thead>
<tr>
<th>SIMD Lanes (Thread Processors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg 1K×32</td>
</tr>
<tr>
<td>Reg 1K×32</td>
</tr>
<tr>
<td>Reg 1K×32</td>
</tr>
<tr>
<td>Reg 1K×32</td>
</tr>
<tr>
<td>Reg 1K×32</td>
</tr>
<tr>
<td>Reg 1K×32</td>
</tr>
<tr>
<td>Reg 1K×32</td>
</tr>
<tr>
<td>Reg 1K×32</td>
</tr>
</tbody>
</table>

#### Address Coalescing Unit

<table>
<thead>
<tr>
<th>Local Memory 64 KB</th>
</tr>
</thead>
</table>

#### Interconnection Network

<table>
<thead>
<tr>
<th>To Global Memory</th>
</tr>
</thead>
</table>
GPU vs Vector Machine

• **Similarities:**
  – Works well with data-level parallel problems
  – Mask registers
  – Multiple lanes
  – Large register files

• **Differences:**
  – GPU uses multithreading to hide memory latency
  – GPU has many functional units, as opposed to a few deeply pipelined units like a vector processor
The end

Good luck!! 😊