Lecture 11: ILP Innovations and SMT

• Today: out-of-order example, ILP innovations, SMT (Sections 3.5 and supplementary notes)
• Assumptions same as HW 4, except there are 36 physical registers and 32 logical registers

• Estimate the issue time, completion time, and commit time for the sample code
OOO Example

Original code
ADD   R1, R2, R3
LD    R2, 8(R1)
ADD   R2, R2, 8
ST    R1, (R3)
SUB   R1, R1, R5
LD    R1, 8(R2)
ADD   R1, R1, R2

Renamed code
ADD   P33, P2, P3
LD    P34, 8(P33)
ADD   P35, P34, 8
ST    P33, (P3)
SUB   P36, P33, P5

Must wait
### OOO Example

<table>
<thead>
<tr>
<th>Original code</th>
<th>Renamed code</th>
<th>InQ</th>
<th>Iss</th>
<th>Comp</th>
<th>Comm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td>ADD P33, P2, P3</td>
<td>i</td>
<td>i+1</td>
<td>i+6</td>
<td>i+6</td>
</tr>
<tr>
<td>LD R2, 8(R1)</td>
<td>LD P34, 8(P33)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+8</td>
</tr>
<tr>
<td>ADD R2, R2, 8</td>
<td>ADD P35, P34, 8</td>
<td>i</td>
<td>i+4</td>
<td>i+9</td>
<td>i+9</td>
</tr>
<tr>
<td>ST R1, (R3)</td>
<td>ST P33, (P3)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+9</td>
</tr>
<tr>
<td>SUB R1, R1, R5</td>
<td>SUB P36, P33, P5</td>
<td>i+1</td>
<td>i+2</td>
<td>i+7</td>
<td>i+9</td>
</tr>
<tr>
<td>LD R1, 8(R2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R1, R1, R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### OOO Example

<table>
<thead>
<tr>
<th>Original code</th>
<th>Renamed code</th>
<th>InQ</th>
<th>Iss</th>
<th>Comp</th>
<th>Comm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td>ADD P33, P2, P3</td>
<td>i</td>
<td>i+1</td>
<td>i+6</td>
<td>i+6</td>
</tr>
<tr>
<td>LD R2, 8(R1)</td>
<td>LD P34, 8(P33)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+8</td>
</tr>
<tr>
<td>ADD R2, R2, 8</td>
<td>ADD P35, P34, 8</td>
<td>i</td>
<td>i+4</td>
<td>i+9</td>
<td>i+9</td>
</tr>
<tr>
<td>ST R1, (R3)</td>
<td>ST P33, (P3)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+9</td>
</tr>
<tr>
<td>SUB R1, R1, R5</td>
<td>SUB P36, P33, P5</td>
<td>i+1</td>
<td>i+2</td>
<td>i+7</td>
<td>i+9</td>
</tr>
<tr>
<td>LD R1, 8(R2)</td>
<td>LD P1, 8(P35)</td>
<td>i+7</td>
<td>i+8</td>
<td>i+14</td>
<td>i+14</td>
</tr>
<tr>
<td>ADD R1, R1, R2</td>
<td>ADD P2, P1, P35</td>
<td>i+9</td>
<td>i+10</td>
<td>i+15</td>
<td>i+15</td>
</tr>
</tbody>
</table>
Reducing Stalls in Fetch

- Better branch prediction
  - novel ways to index/update and avoid aliasing
  - cascading branch predictors

- Trace cache
  - stores instructions in the common order of execution, not in sequential order
  - in Intel processors, the trace cache stores pre-decoded instructions
Reducing Stalls in Rename/Regfile

- Larger ROB/register file/issue queue

- Virtual physical registers: assign virtual register names to instructions, but assign a physical register only when the value is made available

- Runahead: while a long instruction waits, let a thread run ahead to prefetch (this thread can deallocate resources more aggressively than a processor supporting precise execution)

- Two-level register files: values being kept around in the register file for precise exceptions can be moved to 2nd level
Stalls in Issue Queue

- Two-level issue queues: 2nd level contains instructions that are less likely to be woken up in the near future

- Value prediction: tries to circumvent RAW hazards

- Memory dependence prediction: allows a load to execute even if there are prior stores with unresolved addresses

- Load hit prediction: instructions are scheduled early, assuming that the load will hit in cache
Functional Units

• Clustering: allows quick bypass among a small group of functional units; FUs can also be associated with a subset of the register file and issue queue
Thread-Level Parallelism

• Motivation:
  ➢ a single thread leaves a processor under-utilized for most of the time
  ➢ by doubling processor area, single thread performance barely improves

• Strategies for thread-level parallelism:
  ➢ multiple threads share the same large processor → reduces under-utilization, efficient resource allocation
    Simultaneous Multi-Threading (SMT)
  ➢ each thread executes on its own mini processor → simple design, low interference between threads
    Chip Multi-Processing (CMP)
How are Resources Shared?

Each box represents an issue slot for a functional unit. Peak thruput is 4 IPC.

- Superscalar processor has high under-utilization – not enough work every cycle, especially when there is a cache miss
- Fine-grained multithreading can only issue instructions from a single thread in a cycle – can not find max work every cycle, but cache misses can be tolerated
- Simultaneous multithreading can issue instructions from any thread every cycle – has the highest probability of finding work for every issue slot
What Resources are Shared?

- Multiple threads are simultaneously active (in other words, a new thread can start without a context switch)

- For correctness, each thread needs its own PC, IFQ, logical regs (and its own mappings from logical to phys regs)

- For performance, each thread could have its own ROB (so that a stall in one thread does not stall commit in other threads), I-cache, branch predictor, D-cache, etc. (for low interference), although note that more sharing → better utilization of resources

- Each additional thread costs a PC, IFQ, rename tables, and ROB – cheap!
Pipeline Structure

What about RAS, LSQ?
Resource Sharing

Thread-1

Instr Fetch

R1 ← R1 + R2
R3 ← R1 + R4
R5 ← R1 + R3

Instr Rename

P65 ← P1 + P2
P66 ← P65 + P4
P67 ← P65 + P66

Thread-2

Instr Fetch

R2 ← R1 + R2
R5 ← R1 + R2
R3 ← R5 + R3

Instr Rename

P76 ← P33 + P34
P77 ← P33 + P76
P78 ← P77 + P35

Issue Queue

P65 ← P1 + P2
P66 ← P65 + P4
P67 ← P65 + P66
P76 ← P33 + P34
P77 ← P33 + P76
P78 ← P77 + P35

Register File

FU

FU

FU

FU
Performance Implications of SMT

- Single thread performance is likely to go down (caches, branch predictors, registers, etc. are shared) – this effect can be mitigated by trying to prioritize one thread.

- While fetching instructions, thread priority can dramatically influence total throughput – a widely accepted heuristic (ICOUNT): fetch such that each thread has an equal share of processor resources.

- With eight threads in a processor with many resources, SMT yields throughput improvements of roughly 2-4.

- Alpha 21464 and Intel Pentium 4 are examples of SMT.
Pentium4 Hyper-Threading

• Two threads – the Linux operating system operates as if it is executing on a two-processor system

• When there is only one available thread, it behaves like a regular single-threaded superscalar processor

• Statically divided resources: ROB, LSQ, issueq -- a slow thread will not cripple thruput (might not scale)

• Dynamically shared: trace cache and decode (fine-grained multi-threaded, round-robin), FUs, data cache, bpred
Multi-Programmed Speedup

- sixtrack and eon do not degrade their partners (small working sets?)
- swim and art degrade their partners (cache contention?)
- Best combination: swim & sixtrackworst combination: swim & art
- Static partitioning ensures low interference – worst slowdown is 0.9
Title

• Bullet