Lecture: SMT, Cache Hierarchies

• Topics: SMT processors, cache access basics and innovations (Sections B.1-B.3, 2.1)
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) or multi-core
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/LSQ (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!
Resource Sharing

Thread-1

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

Instr Fetch

Instr Rename

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

Thread-2

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

Instr Fetch

Instr Rename

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

Register File

Issue Queue

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

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
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 & sixtrack worst combination: swim & art
- Static partitioning ensures low interference – worst slowdown is 0.9

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Best Speedup</th>
<th>Worst Speedup</th>
<th>Avg Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>1.48</td>
<td>1.14</td>
<td>1.24</td>
</tr>
<tr>
<td>vpr</td>
<td>1.43</td>
<td>1.04</td>
<td>1.17</td>
</tr>
<tr>
<td>gcc</td>
<td>1.44</td>
<td>1.00</td>
<td>1.11</td>
</tr>
<tr>
<td>mcf</td>
<td>1.57</td>
<td>1.01</td>
<td>1.21</td>
</tr>
<tr>
<td>crafty</td>
<td>1.40</td>
<td>0.99</td>
<td>1.17</td>
</tr>
<tr>
<td>parser</td>
<td>1.44</td>
<td>1.09</td>
<td>1.18</td>
</tr>
<tr>
<td>eon</td>
<td>1.42</td>
<td>1.07</td>
<td>1.25</td>
</tr>
<tr>
<td>perl-bmk</td>
<td>1.40</td>
<td>1.07</td>
<td>1.20</td>
</tr>
<tr>
<td>gap</td>
<td>1.43</td>
<td>1.17</td>
<td>1.25</td>
</tr>
<tr>
<td>vortex</td>
<td>1.41</td>
<td>1.01</td>
<td>1.13</td>
</tr>
<tr>
<td>bzip2</td>
<td>1.47</td>
<td>1.15</td>
<td>1.24</td>
</tr>
<tr>
<td>twolf</td>
<td>1.48</td>
<td>1.02</td>
<td>1.16</td>
</tr>
<tr>
<td>wupwise</td>
<td>1.33</td>
<td>1.12</td>
<td>1.24</td>
</tr>
<tr>
<td>swim</td>
<td>1.58</td>
<td>0.90</td>
<td>1.13</td>
</tr>
<tr>
<td>mgrid</td>
<td>1.28</td>
<td>0.94</td>
<td>1.10</td>
</tr>
<tr>
<td>applu</td>
<td>1.37</td>
<td>1.02</td>
<td>1.16</td>
</tr>
<tr>
<td>mesa</td>
<td>1.39</td>
<td>1.11</td>
<td>1.22</td>
</tr>
<tr>
<td>galgel</td>
<td>1.47</td>
<td>1.05</td>
<td>1.25</td>
</tr>
<tr>
<td>art</td>
<td>1.55</td>
<td>0.90</td>
<td>1.13</td>
</tr>
<tr>
<td>equake</td>
<td>1.48</td>
<td>1.02</td>
<td>1.21</td>
</tr>
<tr>
<td>facerec</td>
<td>1.39</td>
<td>1.16</td>
<td>1.25</td>
</tr>
<tr>
<td>ammp</td>
<td>1.40</td>
<td>1.09</td>
<td>1.21</td>
</tr>
<tr>
<td>lucas</td>
<td>1.36</td>
<td>0.97</td>
<td>1.13</td>
</tr>
<tr>
<td>fma3d</td>
<td>1.34</td>
<td>1.13</td>
<td>1.20</td>
</tr>
<tr>
<td>sixtrack</td>
<td>1.58</td>
<td>1.28</td>
<td>1.42</td>
</tr>
<tr>
<td>apsi</td>
<td>1.40</td>
<td>1.14</td>
<td>1.23</td>
</tr>
<tr>
<td>Overall</td>
<td>1.58</td>
<td>0.90</td>
<td>1.20</td>
</tr>
</tbody>
</table>
The Cache Hierarchy

- Core
- L1
- L2
- L3
- Off-chip memory
Problem 1

- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?
Problem 1

- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?

With L3: $1000 + 10 \times 20 + 30 \times 10 + 300 \times 5 = 3000$
Without L3: $1000 + 10 \times 20 + 10 \times 300 = 4200$
Accessing the Cache

Direct-mapped cache: each address maps to a unique address

Data array

Sets

8-byte words

Offset

Byte address

101000

8 words: 3 index bits
The Tag Array

Byte address

101000

Tag

Compare

Tag array

Data array

8-byte words

Direct-mapped cache: each address maps to a unique address
Increasing Line Size

A large cache line size → smaller tag array, fewer misses because of spatial locality

32-byte cache line size or block size

Tag array

Data array

Tag

Offset

Byte address

10100000
**Associativity**

- **Byte address**: 10100000
- **Tag array**
- **Compare**
- **Tag**: 10100000
- **Set associativity**: 
  - Fewer conflicts; wasted power because multiple data and tags are read
- **Way-1**
- **Way-2**
- **Data array**
Problem 2

• Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 1, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
Problem 2

• Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 1, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
M MH MM H MM HM HMM M M M
Problem 3

• Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
Problem 3

• Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

```
A B B E C C A D B F A E G C G A
M MH M MH MM HM HMM M H M
```
Problem 4

- 64 KB 16-way set-associative data cache array with 64 byte line sizes, assume a 40-bit address
  - How many sets?
  - How many index bits, offset bits, tag bits?
  - How large is the tag array?
Problem 4

- 64 KB 16-way set-associative data cache array with 64 byte line sizes, assume a 40-bit address

- How many sets? 64

- How many index bits (6), offset bits (6), tag bits (28)?

- How large is the tag array (28 Kb)?
Title

• Bullet