Lecture: SMT, Cache Hierarchies

- Topics: SMT processors, cache access basics/examples
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 throughput 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.
The Alpha 21264 Out-of-Order Implementation

Branch prediction and instr fetch

Instr Fetch Queue

R1 ← R1+R2
R2 ← R1+R3
BEQZ R2
R3 ← R1+R2
R1 ← R3+R2

Decode & Rename

Speculative Reg Map
R1 → P36
R2 → P34

Instr 1
Instr 2
Instr 3
Instr 4
Instr 5
Instr 6

Instr 1
Instr 2
Instr 3
Instr 4
Instr 5
Instr 6

Committed
Reg Map
R1 → P1
R2 → P2

Register File
P1-P64

Reorder Buffer (ROB)

P33 ← P1+P2
P34 ← P33+P3
BEQZ P34
P35 ← P33+P34
P36 ← P35+P34

Issue Queue (IQ)

ALU

Results written to regfile and tags broadcast to IQ

ALU

ALU

ALU
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!
Pipeline Structure

- Front End
- Front End
- Front End
- Front End
- Execution Engine
- Private/Shared Front-end
- Private Front-end
- Rename
- ROB
- I-Cache
- Bpred
- Regs
- IQ
- DCache
- FUs
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
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
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
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

- Byte address
- Offset
- Data array
- Sets

8-byte words

8 words: 3 index bits

101000
The Tag Array

- Byte address
- Tag
- Compare
- 101000
- 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 $\rightarrow$ smaller tag array, fewer misses because of spatial locality

- 32-byte cache line size or block size
**Associativity**

Set associativity $\rightarrow$ fewer conflicts; wasted power because multiple data and tags are read.
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 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 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 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 MM H MM HM HMM M 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)?
Problem 5

• 8 KB fully-associative data cache array with 64 byte line sizes, assume a 40-bit address

• How many sets? How many ways?

• How many index bits, offset bits, tag bits?

• How large is the tag array?
Problem 5

• 8 KB fully-associative data cache array with 64 byte line sizes, assume a 40-bit address

• How many sets (1)? How many ways (128)?

• How many index bits (0), offset bits (6), tag bits (34)?

• How large is the tag array (544 bytes)?
Types of Cache Misses

• Compulsory misses: happens the first time a memory word is accessed – the misses for an infinite cache

• Capacity misses: happens because the program touched many other words before re-touching the same word – the misses for a fully-associative cache

• Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache

• Sidenote: can a fully-associative cache have more misses than a direct-mapped cache of the same size?