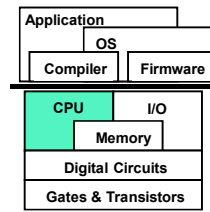


CSE 560 Computer Systems Architecture

Hardware Multithreading

1

This Unit: Multithreading (MT)



- Why multithreading (MT)?
 - Utilization vs. performance
- Three implementations
 - Coarse-grained MT
 - Fine-grained MT
 - Simultaneous MT (SMT)

2

Performance And Utilization

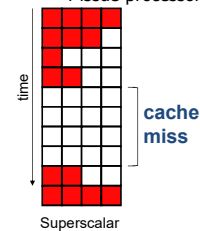
- Performance (IPC) important
- Utilization (actual IPC / peak IPC) important too
- Even moderate superscalars (*e.g.*, 4-way) not fully utilized
 - Average sustained IPC: 1.5–2 → < 50% utilization
 - Mis-predicted branches
 - Cache misses, especially L2
 - Data dependences
- **Multi-threading (MT)**
 - Improve utilization by multi-plexing multiple threads on single CPU
 - One thread cannot fully utilize CPU? Maybe 2, 4 (or 100) can

3

3

Simple Multithreading

- Time evolution of issue slot
 - 4-issue processor



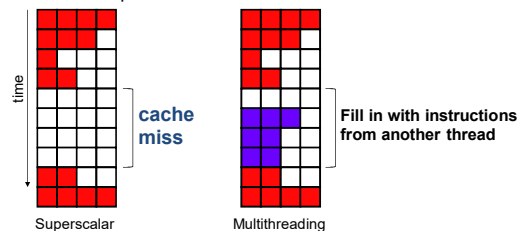
Superscalar

4

4

Simple Multithreading

- Time evolution of issue slot
 - 4-issue processor



- Where does it find a thread? Same problem as multi-core
 - Same shared-memory abstraction

5

5

Latency vs Throughput

- **MT trades (single-thread) latency for throughput**
 - Sharing processor degrades latency of individual threads
 - + But improves aggregate latency of both threads
 - + Improves utilization
- Example
 - Thread A: individual latency=10s, latency with thread B=15s
 - Thread B: individual latency=20s, latency with thread A=25s
 - Sequential latency (first A then B or vice versa): 30s
 - Parallel latency (A and B simultaneously): 25s
 - MT slows each thread by 5s
 - + But improves total latency by 5s
- **Different workloads have different parallelism**
 - SpecFP has lots of ILP (can use an 8-wide machine)
 - Server workloads have TLP (can use multiple threads)

6

6

MT Implementations: Similarities

- How do multiple threads share a single processor?
 - Different sharing mechanisms for different kinds of structures
 - Depend on what kind of state structure stores
- No state:** ALUs
 - Dynamically shared
- Persistent hard state (aka "context"):** PC, registers
 - Replicated
- Persistent soft state:** caches, bpred
 - Dynamically partitioned (like multi-program uni-processor)
 - TLBs need thread ids, caches/bpred tables don't
 - Exception: **ordered "soft" state** (BHR, RAS) is replicated
- Transient state:** pipeline latches, ROB, RS
 - Partitioned ... somehow

7

7

MT Implementations: Differences

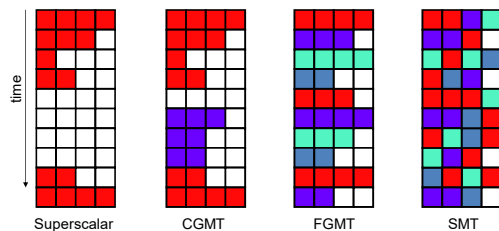
- Main question: **thread scheduling policy**
 - When to switch from one thread to another?
- Related question: **pipeline partitioning**
 - How exactly do threads share the pipeline itself?
- Depends on
 - What kind of latencies (specifically, length) you want to tolerate
 - How much single thread performance you are willing to sacrifice
- Three designs
 - Coarse-grain multithreading (CGMT)
 - Fine-grain multithreading (FGMT)
 - Simultaneous multithreading (SMT)

8

8

The Standard Multithreading Picture

- Time evolution of issue slots
 - Color = thread



9

9

Coarse-Grain Multithreading (CGMT)

- + Sacrifices little single thread performance (of 1 thread)
- Tolerates only long latencies (*e.g.*, L2 misses)
- Thread scheduling policy
 - Designate a "preferred" thread (*e.g.*, thread A)
 - Switch to thread B on thread A L2 miss
 - Switch back to A when A L2 miss returns
- Pipeline partitioning
 - None, flush on switch
 - Can't tolerate latencies shorter than 2x pipeline depth
 - Need short in-order pipeline for good performance
- Example: IBM Northstar/Pulsar

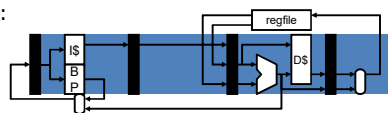


10

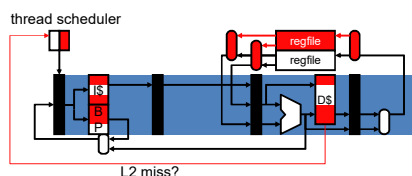
10

CGMT

Original:



CGMT:



11

11

Fine-Grain Multithreading (FGMT)

- Sacrifices *significant* single thread performance
- + Tolerates latencies (*e.g.*, L2 misses, mispredicted branches, *etc.*)
- Thread scheduling policy
 - Switch threads every cycle (round-robin), L2 miss or no
- Pipeline partitioning
 - Dynamic, no flushing
 - Length of pipeline doesn't matter so much
- Need a lot of threads



12

12

Fine-Grain Multithreading (FGMT)

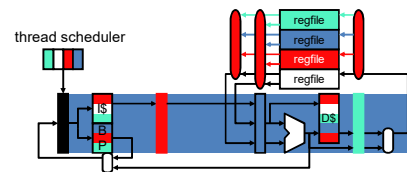
- Extreme example: Denelcor HEP
 - So many threads (100+), it didn't even need caches
 - Failed commercially (or so we thought!)
- Not popular today (in traditional processors)
 - Many threads → many register files
 - One commercial example is Cray Urika (with historical ties to Denelcor HEP, Burton Smith architected both)
- Is popular today (in GPUs)
 - SIMT (single instruction, multiple threads)
 - Data parallel, in-order execution
 - Pipeline isn't the same as what we've been studying, but it does use FGMT

13

Fine-Grain Multithreading

FGMT:

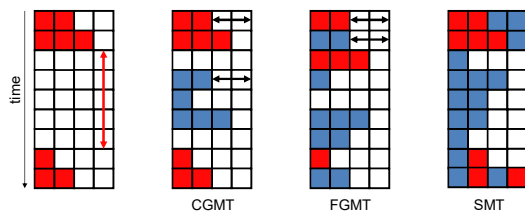
- **Multiple threads in pipeline at once**
- (Many) more threads



14

Vertical and Horizontal Under-Utilization

- FGMT and CGMT reduce **vertical under-utilization**
 - *Nothing* issues in a given cycle
- Do not help with **horizontal under-utilization**
 - *Not all issue slots* issue in a given cycle (for superscalar)



15

Simultaneous Multithreading (SMT)

What can issue insns from multiple threads in one cycle?

- Same thing that issues insns from multiple parts of same program...

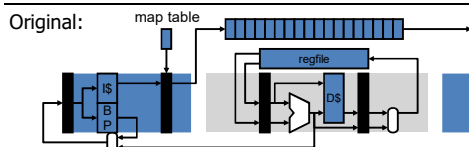
...out-of-order execution

Simultaneous multithreading (SMT): OOO + FGMT

- Aka "**hyper-threading**"
- Observation: once insns are renamed, scheduler doesn't care which thread they come from (well, for non-loads at least)
- Some examples
 - IBM Power5: 4-way issue, 2 threads
 - Intel Pentium4: 3-way issue, 2 threads
 - Intel Core i7: 4-way issue, 2 threads
 - Alpha 21464: 8-way issue, 4 threads (canceled)
- Notice a pattern? #threads (T) x 2 = # issue width (N)

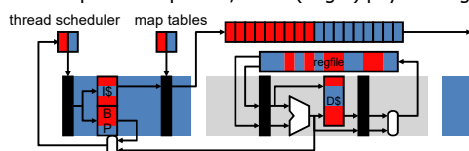
16

Simultaneous Multithreading (SMT)



SMT:

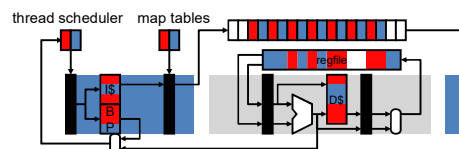
- Replicate map table, share (larger) physical register file



17

SMT Resource Partitioning

- **Physical regfile** and **insn buffer entries** shared at fine-grain
 - Physically unordered and so fine-grain sharing is possible
- How are **physically ordered** structures (ROB/LSQ) shared?
 - Fine-grain sharing (below) entangles commit (and squash)
- Allowing threads to commit independently is important



18

17

18

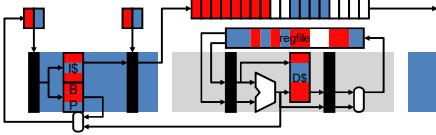
Static & Dynamic Resource Partitioning

Static partitioning (below)

- T equal-sized contiguous partitions
- ± No starvation, sub-optimal utilization (fragmentation)

Dynamic partitioning

- $P > T$ partitions, available partitions assigned on need basis
 - ± Better utilization, possible starvation
 - ICOUNT: fetch policy prefers thread with fewest in-flight insns
- Couple both with larger ROBs/LSQs



19

19

Multithreading Issues

Shared soft state (caches, branch predictors, TLBs, etc.)

Key example: **cache interference**

- General concern for all MT variants
- Can the working sets of multiple threads fit in the caches?
- Shared memory threads help: **Single Program Multiple Data (SPMD)**
 - + Same insns → share I\$
 - + Shared data → less D\$ contention
- MT is good for workloads with shared insn/data
- To keep miss rates low, SMT might need a larger L2 (which is OK)
 - Out-of-order tolerates L1 misses

Large physical register file (and map table)

- physical registers = (**#threads** × #arch-regs) + #in-flight insns
- map table entries = (**#threads** × #arch-regs)

20

20

Subtleties Of Sharing Soft State

What needs a thread ID?

- Caches
- TLBs
- BTB (branch target buffer)
- BHT (branch history table)

21

21

Necessity Of Sharing Soft State

Caches are shared naturally...

- Physically-tagged: address translation distinguishes different threads

TLBs need explicit thread IDs to be shared

- Virtually-tagged: entries of different threads indistinguishable
- Thread IDs are only a few bits: enough to identify on-chip contexts

22

22

Costs Of Sharing Soft State

BTB: Thread IDs make sense

- entries are already large, a few extra bits / entry won't matter
- Different thread's target prediction → definite mis-prediction

BHT: make less sense

- entries are small, a few extra bits / entry is huge overhead
- Different thread's direction prediction → possible mis-prediction

Ordered soft-state should be replicated

- Examples: Branch History Register (BHR*), Return Address Stack (RAS)
- Otherwise they become meaningless... Fortunately, it is typically small

23

23

Multithreading vs. Multicore

If you wanted to run multiple threads would you build a...

- A multicore: multiple separate pipelines?
- A multithreaded processor: a single larger pipeline?

Both will get you throughput on multiple threads

- Multicore core will be simpler, possibly faster clock
- SMT will get you better performance (IPC) on a single thread
 - SMT is basically an ILP engine that converts TLP to ILP
 - Multicore is mainly a TLP (thread-level parallelism) engine

Do both

- Sun's Niagara (UltraSPARC T1)
- 8 processors, each with 4-threads (non-SMT threading)
- 1GHz clock, in-order, short pipeline (6 stages or so)
- Designed for power-efficient "throughput computing"

24

24

Multithreading Summary

- **Latency vs. throughput**
- Partitioning different processor resources
- Three multithreading variants
 - Coarse-grain: no single-thread degradation, but long latencies only
 - Fine-grain: other end of the trade-off
 - Simultaneous: fine-grain with out-of-order
- Multithreading vs. chip multiprocessing

28