Multithreading

cs9244

Outline

- Why multithreading (MT)
  - Performance and utilization
- Three approaches
  - Fine-grained multithreading
  - Coarse-grained multithreading
  - Simultaneous multithreading
- Crosscutting issues
  - Resource contention
  - SMT vs. CMP
  - Speculative multithreading (SpMT)

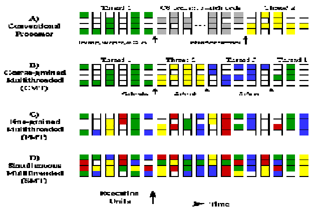
Why multithreading

- Performance
  - Throughput (IPC)
  - Speedup for individual threads
- Utilization
  - Average sustained IPC: 1.5-2 on a moderate superscalar (e.g. 4-way) ➔ < 50%
  - Switch between multiple threads to overlap stalls

Three MT approaches

- Fine-grained multithreading
- Coarse-grained multithreading
- Simultaneous multithreading

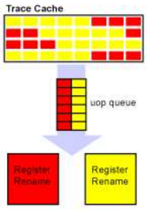
Three MT approaches (cont)



Similarities in MT implementations

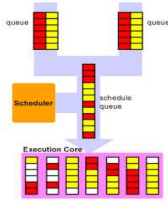
- How do multiple threads share a single processor?
  - Different mechanism for different structures
  - Depend on the context of the structure
- Three sharing mechanism
  - Replicate: PC, Architectural register
  - Partition: re-order buffer, Load/store buffer, queues
    - Statically partitioning vs. dynamically partitioning
  - Share: caches, physical register, execution units
    - The more resources that can be shared, the more efficient MT can be

Two MT resource partitioning categories



- Statically partitioning
  - Fixed partitioning
  - Decomposed equally
- Fairness
  - Ensures that low-IPC threads don't starve high-IPC threads

Two MT resource partitioning categories (cont)



- Dynamically partitioning
  - Has same effect as fixed partitioning
  - Confines each thread the number of entries they can use
  - Can use any entry

Differences in MT implementations

- Thread scheduling policy
  - When to switch from one thread to another
    - Switch every fixed number of cycles
    - Switch when stalls with long latency
- Pipeline sharing
  - How exactly threads share the pipeline
    - Dynamically sharing
    - Varying interleaved instructions from multiple threads vs. instructions from one thread

Fine-grained multithreading

- Switch on a fixed fine-grained schedule (usually on every cycle, in round-robin fashion)
- Dynamically sharing pipelining
- Advantage:
  - Tolerate all latencies
- Disadvantage:
  - Sacrifice the performance of individual threads
  - Need a lot of threads to hide stalls
  - Many threads means many register files
- Example: Denelcor HEP, Tera MTA

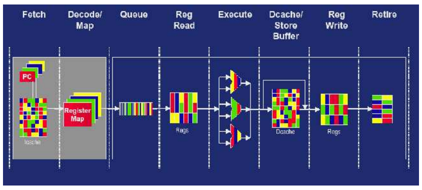
Coarse-grained multithreading

- Switch when reaches certain situations (e.g. L2 misses)
- Thread-switch penalty
- No pipeline sharing
- Advantages:
  - Sacrifice very little individual thread performance
- Disadvantages:
  - Need short in-order pipeline to gain performance
  - Cannot tolerate short latency
- Example: Northstar, Pulsar Power PC from IBM

Simultaneous multithreading

- Fine-grained, dynamically share the pipeline
- Can multithread an out-of-order processor
- Advantages:
  - Tolerate all latencies
  - Higher utilization
  - Sacrifice some individual threads' performance
- Example: Pentium 4 Xeon (5 issues, 2 threads)

Pipeline supporting SMT

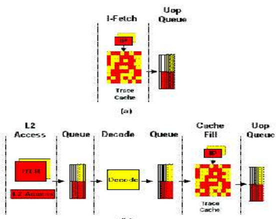


- An earlier version from UCSB called DISC **[Nemirovsky et al. '91]**

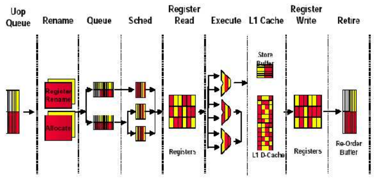
Xeon: case study of implementing SMT

- Adding Hyper-threading to Xeon precessor adds only 5% die area
- Experience 30% gain in performance

Xeon's front-end detailed pipeline



Xeon's out-of-order execution engine detailed pipeline



Crosscutting issues

- Resource contention
- SMT vs. CMP
- Speculative multithreading (SpMT)

Resource contention

- Cache contention
- No cache coherency problems as in SMP
- Cache can be monopolized by one thread
- May increase cache conflicts ➔ may degrade performance seriously

SMT vs. CMP

- Chip Multiprocessor (CMP)
  - Integrate multiple processor cores on a single chip
  - Less sensitive to poor data layout and poor inter-core communication
  - Simple core ➔ short cycle time
  - Wasted resources when lack of TLP
- SMT
  - Multiple “logical” processors
  - More flexible
  - Increasing die area & require longer cycle time

Speculative multithreading

- Relax threads execution order from semantic order
- Changes
  - How to detect mis-speculation?
  - How to rollback: fully or partially?
  - How to identify effective threads?
  - How to weight benefits?
    - Thread start-up overhead
    - Mis-speculation cost