Advanced Microarchitecture

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

Lecture notes based on notes by Ilhyun Kim
Updated by Mikko Lipasti

Outline

• Instruction scheduling overview
  – Scheduling atomicity
  – Speculative scheduling
  – Scheduling recovery
• Complexity-effective instruction scheduling techniques
• Building large instruction windows
  – Runahead, CFP, ICFP
• Scalable load/store handling
• Control Independence

Readings

• Read on your own:
  – Shen & Lipasti Chapter 10 on Advanced Register Data Flow – skim

• To be discussed in class:

Instruction scheduling

• A process of mapping a series of instructions into execution resources
  – Decides when and where an instruction is executed
    • Data dependence graph
    • Mapped to two FUs

• A set of wakeup and select operations
  – Wakeup
    • Broadcasts the tags of parent instructions selected
    • Dependent instruction gets matching tags, determines if source operands are ready
    • Resolves true data dependences
  – Select
    • Picks instructions to issue among a pool of ready instructions
    • Resolves resource conflicts
      – Issue bandwidth
      – Limited number of functional units / memory ports
Advanced instruction scheduling
Ilhyun Kim & Mikko H. Lipasti

Scheduling

- Basic wakeup and select operations

Select logic

Wakeup logic

Scheduling Atomicity

- Operations in the scheduling loop must occur within a single clock cycle
  - For back-to-back execution of dependent instructions

Implication of scheduling atomicity

- Pipelining is a standard way to improve clock frequency
- Hard to pipeline instruction scheduling logic without losing ILP
  - ~10% IPC loss in 2-cycle scheduling
  - ~19% IPC loss in 3-cycle scheduling
- A major obstacle to building high-frequency microprocessors

Scheduler Designs

- Data-Capture Scheduler
  - Keep the most recent register value in reservation stations
  - Data forwarding and wakeup are combined

Scheduler Designs

- Non-Data-Capture Scheduler
  - Keep the most recent register value in RF (physical registers)
  - Data forwarding and wakeup are decoupled
  - Complexity benefits
    - Simpler scheduler / data / wakeup path
Advanced instruction scheduling
Ilhyun Kim & Mikko H. Lipasti

Mapping to pipeline stages
• AMD K7 (data-capture)

Speculative Scheduling
• Speculatively wakeup dependent instructions even before the parent instruction starts execution
  – Keep the scheduling loop within a single clock cycle
• But, nobody knows what will happen in the future
• Source of uncertainty in instruction scheduling: loads
  – Cache hit / miss
  – Store-to-load aliasing
  – eventually affects timing decisions
• Scheduler assumes that all types of instructions have pre-determined fixed latencies
  – Load instructions are assumed to have a common case (over 90% in general)
    – DL1 hit latency
  – If incorrect, subsequent (dependent) instructions are replayed

Scheduling atomicity
& non-data-capture scheduler
• Multi-cycle scheduling loop

Scheduling replay
• Speculation needs verification / recovery
  – There’s no free lunch
• If the actual load latency is longer (i.e. cache miss) than what was speculated
  – Best solution (disregarding complexity): replay data-dependent instructions issued under load shadow

Wavefront propagation
• Speculative execution wavefront
  – speculative image of execution (from scheduler’s perspective)
• Both wavefront propagates along dependence edges at the same rate (1 level / cycle)
  – the real wavefront runs behind the speculative wavefront
• The load resolution loop delay complicates the recovery process
  – scheduling miss is notified a couple of clock cycles later after issue
Load resolution feedback delay in instruction scheduling

- Scheduling runs multiple clock cycles ahead of execution
  - But, instructions can keep track of only one level of dependence at a time (using source operand identifiers)

Requirements of scheduling replay

- Propagation of recovery status should be faster than speculative wavefront propagation
- Recovery should be performed on the transitive closure of dependent instructions

- Conditions for ideal scheduling replay
  - All mis-scheduled dependent instructions are invalidated instantly
  - Independent instructions are unaffected

- Multiple levels of dependence tracking are needed
  - e.g. Am I dependent on the current cache miss?
  - Longer load resolution loop delay → tracking more levels

Scheduling replay schemes

- Alpha 21264: Non-selective replay
  - Replays all dependent and independent instructions issued under load shadow
  - Analogous to squashing recovery in branch misprediction
  - Simple but high performance penalty
    - Independent instructions are unnecessarily replayed

Low-complexity scheduling techniques

- FIFO (Palacharla, Jouppi, Smith, 1996)
  - Replaces conventional scheduling logic with multiple FIFOs
  - Steers logic puts instructions into different FIFOs considering dependencies
  - A FIFO contains a chain of dependent instructions
  - Only the head instructions are considered for issue
Advanced instruction scheduling
Ilhyun Kim & Mikko H. Lipasti

FIFO (cont’d)
- Scheduling example

FIFO (cont’d)
- Performance
  - Comparable performance to the conventional scheduling
  - Reduced scheduling logic complexity
  - Many related papers on clustered microarchitecture
  - Can in-order clusters provide high performance? [Zilles reading]

Memory Dataflow

Key Challenge: MLP
- Tolerate/overlap memory latency
  - Once first miss is encountered, find another one
- Naïve solution
  - Implement a very large ROB, IQ, LSQ
  - Power/area/delay make this infeasible
- Build virtual instruction window

Runahead
- Use poison bits to eliminate miss-dependent load program slice
  - Forward load slice processing is a very old idea
    - Massive Memory Machine [Sarma et al. 84]
    - Datascalar [Burger, Kaxiras, Goodman 97]
  - Runahead proposed by [Dundas, Mudge 97]
- Checkpoint state, keep running
  - When miss completes, return to checkpoint
    - May need runahead cache for store/load communication

Waiting Instruction Buffer
[Lebeck et al. ISCA 2002]
- Capture forward load slice in separate buffer
  - Propagate poison bits to identify slice
- Relieve pressure on issue queue
- Reinsert instructions when load completes
- Very similar to Intel Pentium 4 replay mechanism
  - But not publicly known at the time
Continual Flow Pipelines
[Srinivasan et al. 2004]
- Slice buffer extension of WIB
  - Store operands in slice buffer as well to free up buffer entries on OOO window
  - Relieve pressure on rename/physical registers
- Applicable to
  - data-capture machines (Intel P6) or
  - physical register file machines (Pentium 4)
- Recently extended to in-order machines (iCFP)
- Challenge: how to buffer loads/stores

Scalable Load/Store Queues
- Load queue/store queue
  - Large instruction window: many loads and stores have to be buffered (25%/15% of mix)
  - Expensive searches
    - positional-associative searches in SQ,
    - associative lookups in LQ
      - coherence, speculative load scheduling
    - Power/area/delay are prohibitive

Store Queue/Load Queue Scaling
- Multilevel queues
- Bloom filters (quick check for independence)
- Eliminate associative load queue via replay
  [Cain 2004]
  - Issue loads again at commit, in order
  - Check to see if same value is returned
  - Filter load checks for efficiency:
    - Most loads don’t issue out of order (no speculation)
    - Most loads don’t coincide with coherence traffic

SVW and NoSQ
- Store Vulnerability Window (SVW)
  - Assign sequence numbers to stores
  - Track writes to cache with sequence numbers
  - Efficiently filter out safe loads/stores by only checking against writes in *vulnerability window*
- NoSQ
  - Rely on load/store alias prediction to satisfy dependent pairs
  - Use SVW technique to check

Store/Load Optimizations
- Weakness: predictor still fails
  - Machine should fail gracefully, not fall off a cliff
    - Glass jaw
- Several other concurrent proposals
  - DMDC, Fire-and-forget, ...

Instruction Flow
**Transparent Control Independence**

- Control flow graph convergence
  - Execution reconverges after branches
  - If-then-else constructs, etc.
- Can we fetch/execute instructions beyond convergence point?
- How do we resolve ambiguous register and memory dependences
  - Writes may or may not occur in branch shadow
- TCI employs CFP-like slice buffer to solve these problems
  - Instructions with ambiguous dependences buffered
  - Reinsert them the same way forward load miss slice is reinserted
- "Best" CI proposal to date, but still very complex and expensive, with moderate payback

**Summary of Advanced Microarchitecture**

- Instruction scheduling overview
  - Scheduling atomicity
  - Speculative scheduling
  - Scheduling recovery
- Complexity-effective instruction scheduling techniques
- Building large instruction windows
  - Runahead, CFP, ICFP
- Scalable load/store handling
- Control Independence