Dynamic Issue & HW Speculation

Today's topics:
Superscalar pipelines
Dynamic issue
Scoreboarding: control centric approach
Tomasulo: data centric approach

Raising the IPC Ceiling

- w/ single-issue IPC_{max} = 1
  - schedule as hard as you want and it's still the asymptote
    - keeping things in order = lots of stalls
  - XU's finish out of order anyway
  - when the transistor budget is high enough
    - just go with multiple issue
      - >= 4 issue common today for superscalar machines

Superscalar issues: issue_{max} = n
- need n way capability in all pipeline stages
  - fetch n = no worries fetch cache line of instructions/cycle
  - decode n
  - get register values = problems?
  - execute n
  - problems?
  - mem n
  - problems? wait for order completion?
  - WB n
  - problems w/ out of order completion?

Fix OOO Completion Problem First

- Enter the ROB (re-order buffer)
  - basic idea for now
  - issue instructions in-order
  - retire/commit Instructions in order
  - use an intermediate buffer to hold results
    - since destructive action to register file or memory must happen in order

- Other ROB niceties
  - helps w/ speculation
  - nullification
  - exceptions
  - but first a simple example

Reorder Buffer In Action

See any problems?
Several Issues

• WB stage is now the commit stage
  • ROB values move to the register file
    » whoops if tags are in the issue queue
      • those values need to be renamed to the register name
      • seems complex – can you think of a better way?

• ROB values move to the register file
  » whoops if tags are in the issue queue
    • those values need to be renamed to the register name
    • seems complex – can you think of a better way?

• IQ contains both register and tag fields
  » with 1 bit to select which is valid
    • initially tag is selected
    • when tag is retired
      – broadcast to IQ and invert selector on a match
    • what about tag values in the pipe
      – only need to worry about entry into EX stage
      – compare needed there as well
      – ROB is WB stage so that’s not a problem
      – MEM isn’t a problem either

• Key observation
  • all destructive operations are done by the ROB commit/retire

Nullification & Exceptions

• If an exception happens
  • exception type is written to the ROB field
    » note that one instruction could generate an exception in multiple stages
    • only care about the first one so no overwrite is allowed

• If some instruction is speculative
  • then predicate is written to the ROB field
    • note: predicate covers branch delay slots and effectively supports nullification

• WB stage in reality
  • try to retire n instructions per cycle
    » If none have pending predicates or exceptions then retire
      • in order retire 1st member of n-instruction bundle w/problem
        – retire the instructions before
        – modify whatever is next in the bundle
        – take the exception and hold the rest

• Key observation
  • all destructive operations are done by the ROB commit/retire

Decode Complexity

• ROB complicates ID significantly
  • operand fetch now has two sources
    » register file or ROB field
    • hence an additional mux is required
  • rename takes some time
    • structural issue requirements will help mitigate the performance penalty

• Bottom line
  • ID will no longer be a single cycle stage
  • For register poor ISA’s like x86
    • ROB slots effectively provides a renamed register pool
      » actually it’s not the right choice
        • Why?
        • remember the front-end back-end x86 thing
ROB Hazard Removal

- RAW
  - nothing changes here
  - no way you can use a value before it's computed
  - unless the value is predicted and predicated
  - only some academic papers think this is a reasonable idea
  - hence instruction scheduling is required

- Wax
  - ROB renaming effectively removes this problem
  - as long as enough ROB slots exist
  - if not
    - the instruction can't be issued and a NOP is injected in the pipe

- Note
  - stalling pipelines @ GHz frequencies is a problem
  - hence NOPs are dynamically generated and pushed through the pipe
  - any issues here?

EX Stages XU's

- Typical separation of XU's
  - ALU (Int +/-, shift, logical (AND, OR, XOR, NOT)
  - Int-multiply
  - Int-divide
  - FP ops can be 32 or 64-bit (typically implement 64-bit)
    - FP-add-sub
    - FP-multiply
    - FP-divide or FP-invert (1/x)
    - FP-sqrt or FP-isqrt?

- Overlaps
  - Branch and Mem ops can be handled with an ALU
  - Int mul or div can be handled by the FP equivalent
    - a common choice is to have an Int-mul but not an Int-div
    - why?
  - actual choice influences structural issue rules

Structural Issue Rules

- Clearly vary by machine
- Example for a 6 issue machine
  - 2 ALU
  - 1 Branch
  - 1 Int Mul or Divide
  - 1 FP Add or Sub
  - 1 Mem
- Why does this make sense?
  - e.g. justification

- Clearly vary by machine
- Example for a 6 issue machine
  - 2 ALU or 1 ALU and 1 Int-Mul
  - 1 Branch
  - 1 FP Mul or Divide
  - 1 FP Add or Sub
  - 1 Mem
- Why does this make sense?
  - Look at instruction frequency and common effort
    - Branch average about every 6 instructions so need that
    - LD + ST about every 6 as well
    - seldom need FP Mul & Divide on same cycle
    - FP Add/Sub share exponent normalization
    - Int-Divide is done on the FP-Div unit
Dynamic Issue

- Until Now
  - Instructions have been issued in order
    » Compiler thinks the world is sequential
    » HW must fulfill that contract
  - e.g. Issue Queue
- Dynamic Issue
  - Basics
    » Use instruction window/buffer rather than a Q
    » Choose the <= n instructions to issue
      - Such that dependencies are satisfied
      - And structural rules are not violated
  - 2 Methods
    » Control-centric: Scoreboarding
    » Data-centric: Tomasulo (text focus)

Dynamic Issue Context

- Less viable in multi-core land
  - Single thread performance is not longer the Holy Grail
  - Power wall is the fundamental constraint
    » Dynamic issue consumes a lot of power
    » All the OOO/ROB stuff consumes a lot of power
  - Thermal wall is also an issue
    » Frequency derating is common
    » Affects reliability & cost in a major way
  - With billion transistor chips
    » If they're all active than the chip melts
    » Interesting stat in a recent talk
      - C0 state is in play a very small percentage of the time
  - Hence
    » I previously spent a lot of time on this issue
      » This term we'll look at the conceptual side
      » And skip the minutiae

Trends

Core Comparison

- Source: Presentation by John Shalf @ NERSC

Each core operates at 1/3 to 1/10th efficiency of largest chip, but you can pack 100x more cores onto a chip and consume 1/20 the power.
Another Viewpoint

- source: John Shalf

<table>
<thead>
<tr>
<th></th>
<th>Traditional Core</th>
<th>Throughput Core</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out of Order</td>
<td>In Order</td>
</tr>
<tr>
<td>Weight</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>Power</td>
<td>37.5</td>
<td>8.25</td>
</tr>
<tr>
<td>Freq</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Threads</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>SPEC Intcad</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>Throughput Capacity</td>
<td>32</td>
<td>296</td>
</tr>
<tr>
<td>Power</td>
<td>0.4</td>
<td>13</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Note: these numbers are a bit optimistic but the trend is correct

Scoreboarding

- Introduced by Seymour Cray in the CDC 6600 circa 1964
  - 4 FPUs, 5 MMUs, 7 IUs
  - centralized control knows all
  - RISC like instruction set
  - 60% performance gain from dynamic reordering
  - Inflated cost by 60% - good thing at $1.2M
  - Not a chip
    - So chip heat and cooling was for the room, not the chip
  - Later MIPS, IBM, & HP bring it back in single chip guise
    - Later changed to more decentralized approach due to long wire phenomena
    - Alpha was the last to convert to dynamic issue but was short lived
      - DEC dies and Intel buys the part that is Alpha
    - And then squashes it

Scoreboard Idea

- Simple in concept, hairy in practice

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Window</th>
<th>Scoreboard</th>
<th>EX</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: normal ID stage now split in 2:
1. decode
2. issue

Multi-XU Scoreboard

- Registers
  - FP Molt
  - FP Molt
  - FP Divide
  - FP Add
  - Integer Unit

Note: this model could support both single or multi-issue

Exception is that one multiply will be issued per cycle

All depends on bus/trunk structure

Scoreboard contains ROB
Not Shown

- Memory ops
  - scoreboard views memory interface as just another XU
- Branches
  - scoreboard tracks branch resolution
    » nullifies any speculative instructions in the branch delay slots
- Details of what the scoreboard entries contain
  - similar to the ROB
  - difference is centralized control
    » gets signals from everywhere and sends enables/selects back
    » round trip over long wires is prohibitive today for single core
    » but it consumes too much power
    » jury still out whether this is a dead tactic or not

Data Centric Dynamic Issue

- Started with the IBM 360/91 circa 1969
  - Tomasulo original idea applied only to the floating point units
    » note:
      - no caches, few registers, no precise exceptions
      - long and variable cycle latencies
      - note w/o caches operands came from registers or main memory
      - memory was based on ferrite cores
      - dinosaurs were a problem in the parking lot
      - results in out of order completion
    » note these characteristics now apply to other pieces of the machine
      - memory hierarchy creates unknown latency returns
      - floating point ops still have variable latencies
- Same basic idea but dataflow based
  - dynamic issue and hazard control is still the goal

Different Control Model

- Multiple XU’s
  - fronted by “reservation stations”
  - when reservation station gets all of it’s operands the Instruction issues into the associated XU
    » out of order issue & out of order completion
    » basically a mechanism for implementing data-flow
    » which is the true semantic contract
- XU’s create results which are tagged with the appropriate reservation station slot ID
  - equivalent of forwarding logic
  - implicitly removes RAW hazards
  - values placed on a “common data bus”
  - reservation station slots are registers
  - implicit renaming
  - removes WAX hazard problem
- Separate load and store Q’s
  - deals with the memory disambiguation issue
  - provides a write buffer (we’ll see more of this later)

New Pipeline Model

- Fetch
  - in order into instruction queue
- Dispatch
  - in-order into an available reservation station
- Issue
  - happens when a res. station slot gets all of it’s operands
    » instruction packet goes into Execute
- Mem & WB are concurrent
  - makes sense since only LD & ST use the MEM stage anyway
  - WB goes to waiting reservation stations, registers, or memory
- Key point
  - in-order fetch and dispatch
  - out of order completion and issue
### Tomasulo Comments

- **CDB is the weak link**
  - needs to be wide enough to hold multiple XU results
  - same laminarity issue with a width wrinkle
    - If you need to execute n Instructions/cycle on average
  - if fetch, dispatch, issue, CDB needs to support n as well

- **Locality**
  - layout has surprisingly local wires
  - no long wire round-trip as per scoreboard approach
  - exception
    - CDB goes EVERYWHERE
      - power hog and a frequency barrier
      - high-C multi-drop bus has signal integrity and delay issues
      - fixed with repeaters but adds delay and power

### Tomasulo Memory Issues

- **Out of order loads and stores possible**
  - OK if addresses don’t match

- **Dynamic memory disambiguation**
  - stall loads if a pending store to the same address
  - OR garner the value from the store unit
  - stall stores when there is a pending load from a previous instruction

- **But what about speculation & exceptions**
  - note exceptions weren’t precise in 1969
    - as far as I can tell nothing was
  - famous Wavy Gravy comment
    - “if you can remember the 60’s you weren’t there”

- **Add the ROB?**
  - It worked before and it will again
Final Comments

- This stuff is important conceptually
  - advise to go through the exercises in the text
    - the mid-term will definitely have something similar
  - BUT
    - don't get too whacked on their particular algorithm
    - variations exist
  - some aspect of these ideas are likely useful in the future
  - for now instructions in a thread are sequential
    - HW dynamics can help exploit ILP
  - key issue
    - socket/chip these days has multiple cores
    - maximizing performance/watt is a critical concern
      - speculation and HW dynamics can be too "watty"
        - but there's no need to go crazy at least right now
        - and in the foreseeable future