Review

14 January 2019 20:34

Pipeline Hazards:

- Structural
    - Hardware cannot support a combination of instr
    - **Bubbles**
- Data
    - Instr depends on result from a instr still in the pipeline
    - **Forwarding**
    - **Stall 'Slots' - Compiler scheduling**
- Control
    - Delay between fetching instr and executing branches/jumps
    - **Early branch determination**
    - **SMT.**
        - **Get a free cycle to decode branch**
        - **While executing other thread**

Evaluating Pipeline designs:

- Influenced by
    - Compiler scheduling
    - Compiler instruction selection
    - Application behavior
- Influences
    - Number of instructions
    - Number of registers needed
    - Number of stalls
    - Clock time

Temporal Locality:

- if an item is referenced, it will tend to be referenced again soon

Spatial Locality:

- If an item is referenced, items whose addresses are close by tend to be referenced soon

Cache:

- Tag, Index, Block Offset
- DM, N-Way set, fully associative
- LRU vs ran replacement
- Write Through vs Write Back
    - WT buffer to L2$
    - WB - dirty bit

Caches

18 January 2019 15:56

$$AMAT = HitTime + MissRate * MissPenalty$$

Reducing Miss Rate:

Misses:

- **Compulsory**
- **Capacity**
- **Conflict**

For same Capacity:

- **Block size**
  - initial improvement due to spatial locality
  - too large a block size wastes cache capacity - lower temporal
- **Associativity**
  - will increase hit rate, but increase hit time
    - higher AMAT as cache size increases
  - Solution
    - Way speculation
    - Victim Cache
      - For DM cache
      - access same time as L1
      - **very small**

Skewed-associative caches:

- use different indices in each way
  - **hash** index bits
- costs
  - latency of hash
  - LRU implementation

Hardware Prefetching

- Predict next data to be fetched
- Stream (FIFO) in parallel with cache
- On miss
  - check prefetch stream
  - avoid pollution
  - needs extra mem BW

Compiler:

- **software prefetching**
  - load to register or cache
  - cost of issuing
  - superscalar has more issue BW
  - often, HW just as effective
- **storage layout** transformations
  - Merging arrays
  - Multidimension array - match array layout to traversal order
- **iteration space** transformations
  - Loop interchange (nested order)
  - Loop fusion
  - Blocking - blocks of data vs rows/columns

Reducing Miss Penalty:

Write-Through - all the way

Write Back - only cache

Write Miss

- Allocate new cache line
  - read to fill block first
- Non-Allocate
  - Send to level data currently stored at

Large blocks - early restart / critical word first

Non-blocking cache:

- continue to supply hits

Add a second-level Cache:

- $global_{miss\_rate} = L1_{miss\_rate} * L2_{miss\_rate}$
- different multi-level inclusion strategies

Reduce Cache Hit time:

- avoid address translation for L1$
- virtually-indexed, physically-tagged
- limits a direct mapped cache to page-size
  - increase via associativity

Dynamic Scheduling

18 January 2019 15:56

Advantages:

- Handles cases when dependencies unknown
- Simplifies the compiler
- HW can run code intended for another pipeline

Issue Stage:

- **In-order**
- Begins Execution
- Split ID stage of 5-stage pipeline
  - Issue Stage
    - Decode
    - Check for structural hazards
  - Read Operands
    - Wait until no data hazards
    - Then read operands

Execution

- **Out-of-order Execution**
- Out-of-order **Completion**

Data Dependence:

- **True Dependence**
  - Flow of data
  - RAW
- Name Dependence
  - Can solve by ***renaming***
  - No flow of data
  - **Anti-Dependence**
    - WAR
  - **Output Dependence**
    - WAW

Tomasulo's Algorithm:

- Goal
  - Increase effective number of registers
    - Renaming in hardware

- Associate register with a **changeable tag**

1. Issue - In Order
   a. Feed **Reservation Station**
      i. Source Operands
         1) Either Register Value
         2) Or Tag
      ii. Free
         1) Avoid Structural Hazards
2. Execution - Out of Order
   a. on Functional Units
   b. Might have to wait for value from CDB
3. Write Back
   a. May complete out-of-order
   b. **Broadcast** result on **Common Data Bus**
      i. With Tag
      ii. Goes to RS and/or Registers

Precise Interrupts: - **ROB**

- Tomsula's out-of-order completion
- Fix this with commit/retire stage

- **Reorder buffer**
  - Instruction, tag, value
  - In order
  - Commits to **commit-side registers**

- On Branch
  - If prediction correct
    - Continue
  - Else
    - ROB entries trashed
    - Issue-side registers reset from commit-side

- Misprediction penalty
  - Shorter pipeline
- Store-load forwarding in ROB

RUU: Register Update unit

- Combined RS and ROB
  - Tag - slot in RUU
  - Operands can wait on Tags
- Sits Between Dispatch and Scheduler
  - FUs write to RUU via CDB
- Connects to Commit

Branch Prediction

18 January 2019 15:57

<u>1-bit BP Buffer</u>

- lower bits of instruction address used as index
- taken / not-taken
- problem: in loop - 2 mispredictions

<u>n-bit BP Buffer</u>

- add *hysteresis*
- **2^n states** - half take, half not-taken
- increasing number of entries or n will only take us so far
- bad for branches which aren't highly biased to taken or not-taken

<u>Local History:</u>

- instruction address

<u>Global History:</u>

- taken - not-taken history for all previously-executed branches
- Compromise - use **m** entries
    - m-bit Branch History Register
    - shift register

<u>**Correlating** BP Buffer:</u>

- **(m,n) "gselect"**
- use m bits to select one of 2^m n-bit BHTs
- real programs have low correlation

<u>**Tournament** Branch Predictor</u>

- 2 predictors: one global, one local
- combine with a **selector**
    - driven by a third predictor

<u>Return Address Predictors</u>

- function can be called from many locations
- hard to predict which address to return to
- save return address in a stack like buffer
    - 8-16 entries
    - catch jump for entries
    - check if correct prediction on return
        - entries trashes by interrupts/other programs

<u>Warm-up:</u>

- Simple predictors re-learn fast
- Sophisticated predictors re-lean more slowly
- ***Selective*** predictor chose simple until sophisticated warms up

<u>Branch **Target Buffer:**</u>

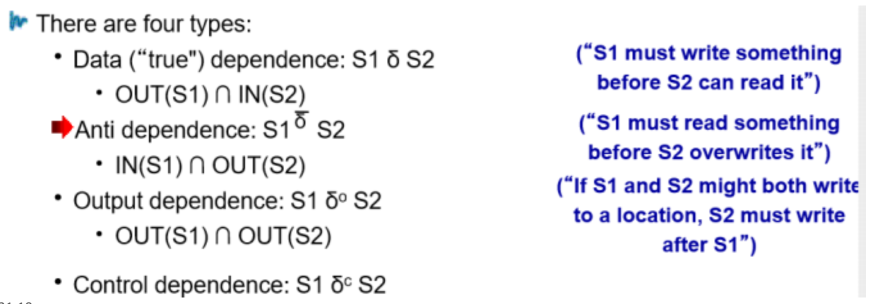
- indexed by lower PC bits
- if tag matches
    - value stored = predicted target address
- BTB can predict if current instruction will be a taken branch
    - without decoding instruction
- When a branch is *committed*
    - update BTB
        - value = target address
        - tag = upper branch instruction address
        - can have extra n-bit predictor for each entry
- if done before commit
    - can help with speculated instructions
- done after
    - BTB won't hold incorrect info

<u>Predicated Execution:</u>

- instead of branch
    - conditionally executed instructions
- reduce number of branches
    - reduce number of *mis-predicted* branches

Dependencies

18 January 2019 15:57



- There are four types:
  - Data ("true") dependence: S1 δ S2
    - OUT(S1) ∩ IN(S2)
  - Anti dependence: S1 $\bar{\delta}$ S2
    - IN(S1) ∩ OUT(S2)
  - Output dependence: S1 δ^o S2
    - OUT(S1) ∩ OUT(S2)
  - Control dependence: S1 δ^c S2

("S1 must write something before S2 can read it")

("S1 must read something before S2 overwrites it")

("If S1 and S2 might both write to a location, S2 must write after S1")

21:10

$Statement \; - \boldsymbol{S_X}$

$\text{-}instance \; S_x^I$

Dependence from earlier iteration: $S_X^I \; \boldsymbol{\delta}_< \; S_X^J$

Loop-carried:

True dep: $\delta_<$

Anti-dep: $\bar{\delta}_<$

Respect order: δ_*

Same iteration of S: $\delta_=$

Parallel: EPIC/VLIW

18 January 2019 15:57

Side-channels / Vulnerabilities

09 March 2019 23:10

Parallel: GPU

18 January 2019 15:57

Vectors

09 March 2019 23:12

Multicore & Clusters

18 January 2019 15:57