ECE 571 – Advanced Microprocessor-Based Design Lecture 16

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

9 October 2024

Announcements

- Don't forget HW#5 (caches)
- Useful reading: "When prefetching works, when it doesn't and why" paper by Lee et al.

HW#4 (brpred) – Haswell-EP Cache Ratio

- Bzip2
  instr=19,698M, branches=3,049M,conditional=2,582M
  branch:instr 15% (1:6),conditional 13% (1:7)%
  If way too low, entering command wrong (no file error low counts)
- equake_l
  instr=1,424B,branches=153B,conditional=145B
  branch:instr 11% (1:9), conditional 10% (1:10)

HW#4 (brpred) – ARM64 Cache Ratio

- Note there are a lot of branch events avail on ARM64
- bzip2 branch ratio:
  instr=18,668M, branches=2,674M, 14% (1:7)
- equake_l branch ratio:
  instr=1,719B, branches=295B, 17% (1:6)

HW#4 – notes on ARM events

- br_immed_spec [Branch speculatively executed, immediate branch]
- br_indirect_spec [Branch speculatively executed, indirect branch]
- br_mis_pred [Branch mispredicted]
- br_pred [Predictable branch]
- br_return_spec [Branch speculatively executed, procedure return]
- btb_mis_pred [BTB misprediction]

HW#4 (brpred) – Branch Miss Rates

- Haswell-EP
  bzip2 = 6.50%, equake_l = 0.51%
- AMD EPYC
  bzip2 = 6.4%, equake_l = 0.44%
- ARM64
  bzip2=4.8%, equake_l = 0.6%
- Ages of machines:
  haswell-ep september 2014, epyc 7251 June 2017, ampere 2018

HW#4 (brpred) – Speculative Execution

- Speculative execution Haswell-EP
  - bzip2: roughly 74% retired
  - equake_l: roughly 56% retired

HW#4 Questions – Why Branch Ratio Differ?

- Compiler being stupid? These are SPEC benchmarks so you can bet that these benchmarks are being optimized as completely as possible.
- Floating point vs Integer code. Floating point, like equake, tends to have lots of regular loops over big blocks of calculations. Integer code like compilers and compression reads user data and makes decisions, so many more if/then loops on irregular data.

- How could you determine the cause?

HW#4 Questions – Arch Branch Ratio Differ?

- BR ratio on bzip Haswell/arm64?
- Actually about the same
- On ARM32 it's more ARM32 is 1:16 (why? probably conditional execution. How could you tell?) (run 32-bit executable on 64-bit or vice-versa)

HW#4 Questions – Miss rates differ

- equake vs bzip
- FP vs Int program.
- FP has more loops, Loops easier to predict.

HW#4 Questions – vs Raptor Lake

- Raptor Lake machine 10 years newer than Haswell-EP
- Branch predictor doesn't seem to have improved much in that time

HW#4 Questions – Speculative Execution

- Retired instructions.
- bzip2: roughly 74% retired,
- equake_l: roughly 54% retired
- So in theory bzip2 is more power efficient

HW#4 Writing a program to give 50%

- What kind of benchmark?
- Random number generation.
- How many branches in random()? Divide-based pseudo-number gen?
- Results below on ivybridge

```
branch-mul (pseudo-random, 2 branches per loop)
 5000138 4999862
        20,123,251      branches                    #  228.926 M/sec
         5,004,391      branch-misses               #   24.87% of all branches
branch-rand (rand(), 17 branches per loop)
 170,143,753      branches                    #  726.443 M/sec
        10,358,447      branch-misses               #    6.09% of all branches
branch-random (random(), 15 branches per loop)
      150,139,161      branches                    #  719.563 M/sec
        10,622,205      branch-misses               #    7.07% of all branches
```

Hardware Prefetching

HW Prefetch Strategies – icache

- Bring in two cache lines
- Branch predictor can provide hints, targets
- Bring in both targets of a branch

HW Prefetch Strategies – dcache

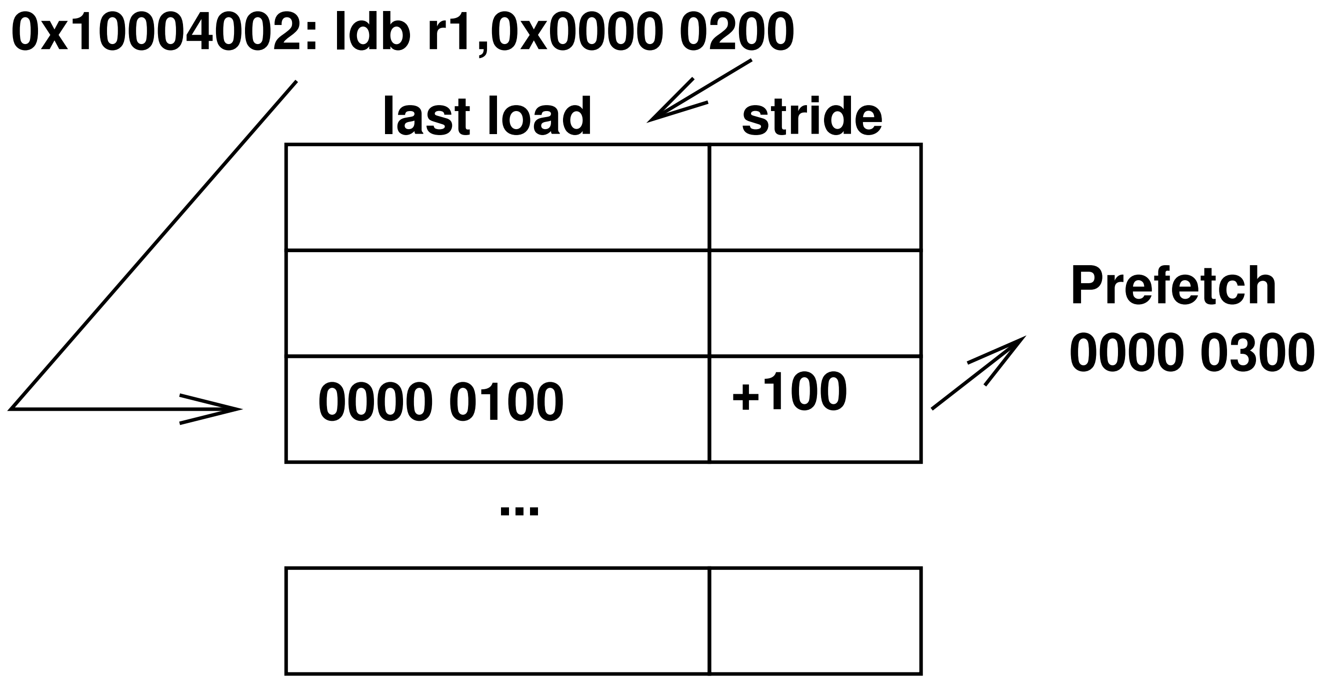
- Bring in next line – on miss bring in N and N+1 (or more?)

- Demand – bring in on miss (every other access a miss with linear access)
  Tagged – bring in N+1 on first access to cache line (no misses with linear access)

Hardware Prefetching – Stride Prefetching

- Stride predictors – like branch predictor, but with load addresses, keep track of stride
- Separate stream buffer?

Stride Predictor



Hardware Prefetching – Correlation/Content-Directed Prefetching

- How to handle things like pointer chasing / linked lists?
- Correlation – records sequence of misses, then when traversing again prefetches in that order
- Content directed – recognize pointers and pre-fetch what they point to

Using 2-bit Counters

- Use 2-bit counter to see if load causing lots of misses, if so automatically treat as streaming load (Rivers)
- Partitioned cache: cache stack, heap, etc, (or little big huge) separately (Lee and Tyson)

SW Prefetch notes from paper

- *When Prefetching Works, When it Doesn't, and Why* by Lee, Kim, and Vuduc (ACM TACO 2012)
- Experiment on some SPEC CPU 2006 benchmarks, some helped, some hurt, some same
- Times SW Prefetch works well
  - Large number of streams (more than available tables)
  - Short streams (takes while to train up HW prefetch)
  - Irregular memory access
  - Hint to bring into L1 (HW often only prefetches to

L3)

  - Loop bounds, SW less likely to go off end of arrays at end of loops

- Times SW works poorly
  - Increases instruction count (both insns, but also a sw prefetch might have extra calcs to construct address)
  - Static behavior, cannot adapt to phase behavior
  - Code changes might be needed (unrolling, etc) to give more calculations between loads
- SW and HW might be antagonistic
  - SW might predict all easy prefetches, leaving HW with

tougher ones and less to learn from

Cortex A9 Prefetch

- PLD – prefetch instruction
  has dedicated instruction unit
- Optional hardware prefetcher. (Disabled on pandaboard)
- Can prefetch 8 data streams, detects ascending and descending with stride of up to 8 cache lines
- Keeps prefetching as long as causing hits
- Stops if: crosses a 4kB page boundary, changes context,

a DSB (barrier) or a PLD instruction executes, or the program does not hit in the prefetched lines.

- PLD requests always take precedence

Quick Look at Haswell Prefetch

- `https://software.intel.com/en-us/articles/disclosure-of-hw-prefetcher-control-on-some-intel-proce`
- 4 prefetches, can independently disable
- L2 hardware prefetcher – fetch data or code into L2
- L2 adjacent cache line prefetcher – bring in 2 cache lines (128B)
- DCU prefetcher – fetch into L1-D cache
- DCU IP prefetcher – use load history to predict what to bring in