ECE 571 – Advanced Microprocessor-Based Design Lecture 11

Vince Weaver

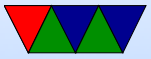
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

4 October 2022

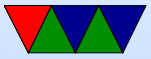
Announcements

- HW#5 was posted, caches

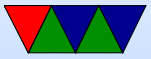


HW#4 (brpred) – Cache Ratio

- Haswell-EP
  - 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)

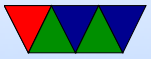


- ARM64 bzip2 branch ratio:
  instr=18,668M, branches=2,674M, 14% (1:7)
  Note there are a lot of branch events avail on ARM64



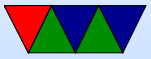
HW#4 (brpred) – Branch Miss

- 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%
- haswell-ep september 2014, epyc 7251 June 2017



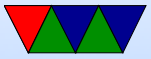
HW#4 (brpred) – Speculative Execution

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



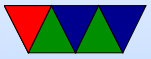
HW#4 – notes on ARM events

- 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]

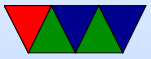


HW#4 Questions – Why Branch Ratio Differ?

- 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.

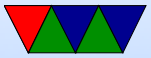


- How could you determine the cause?



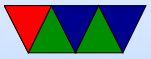
HW#4 Questions – Arch Branch Ratio Differ?

- 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)



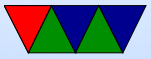
HW#4 Questions – Miss rates differ

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



HW#4 Questions – Speculative Execution

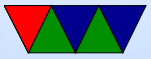
- Retired instructions.
- bzip2: roughly 74% retired,
- equake_l: roughly 54% retired
- So in theory bzip2 is more power efficient



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
```

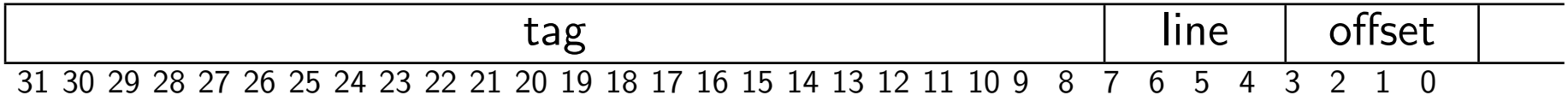


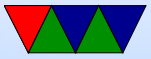
Cache Example Two

512 Byte cache, 2-Way Set Associative, with 16 byte lines, LRU replacement.

write-back, write-allocate

24-bit tag, 16 lines (4 bits), 4-bit offset.

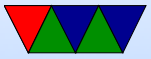




Cache Example 2

Note for LRU: least-recently used

- LRU = 0, least recently used is false, so is most recent
- LRU = 1, least recently used i true, so is oldest

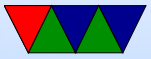


Cache Example – Instruction 1

```
ldrb r1, 0x00000000
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

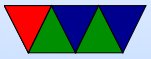


Cache Example – Instruction 2

```
str r1, 0x00000001
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	1	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Hit (updates the dirty bit)

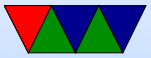


Cache Example – Instruction 3

```
strb r1, 0x00000105
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	1	1	0000 00	1	1	0	0000 01
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold (bring in as it's write allocate)

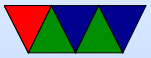


Cache Example – Instruction 4

```
ldr r1, 0x00000206
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 02	1	1	1	0000 01
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

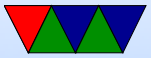


Cache Example – Instruction 5

```
ldb r1, 0x00000000
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 02	1	0	0	0000 00
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Conflict

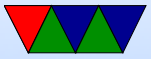


Cache Example – Instruction 6

```
ldb r1, 0x00000030
```

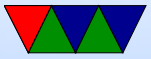
	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 02	1	0	0	0000 00
1	0				0			
2	0				0			
3	1	0	0	0000 00	0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

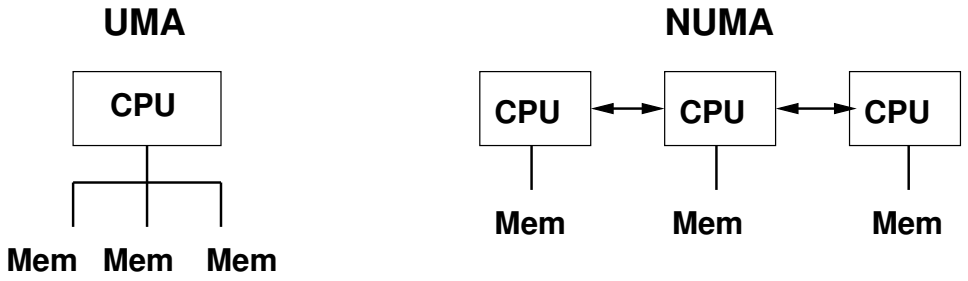


CMP Issues

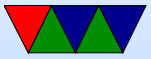
Things were complex enough, what happens when we allow multiple cores to share memory but also have caches.



UMA and NUMA

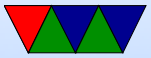


- UMA – Uniform Memory Access
  same speed to access all of memory
- NUMA – Non-Uniform Memory Access
  accesses to memory connected to other CPU can take longer



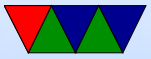
Cache Coherency

- Only gets complicated when you start writing.
- Need some way to know if other core's caches have the address you are accessing
  - Snoopy – "snoop" the bus
  - Directory – have central logic (doesn't scale as well)
- Cache coherency protocols
  - MESI
  - MOESI



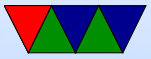
MESI

- State machine: modified, exclusive, shared, invalid
- Invalid – starts out here
- Shared – only has been read, same as memory, can be in multiple caches
- Exclusive – a core is requesting to read, so it gets exclusive access, invalidates all other copies
- Modified – dirty, has been written to. Write back and then can change to S

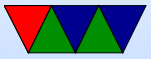


Other Concerns

- What about load/store values in the pipeline that haven't been retired yet?
- What about Memory Model? Can loads pass stores? x86 has strict model, others no so much
  Apple M2 has special "act like x86 mode" to make emulation easier
- Special assembly instructions to flush cache, memory barriers. etc
- Locking in multi-threaded apps

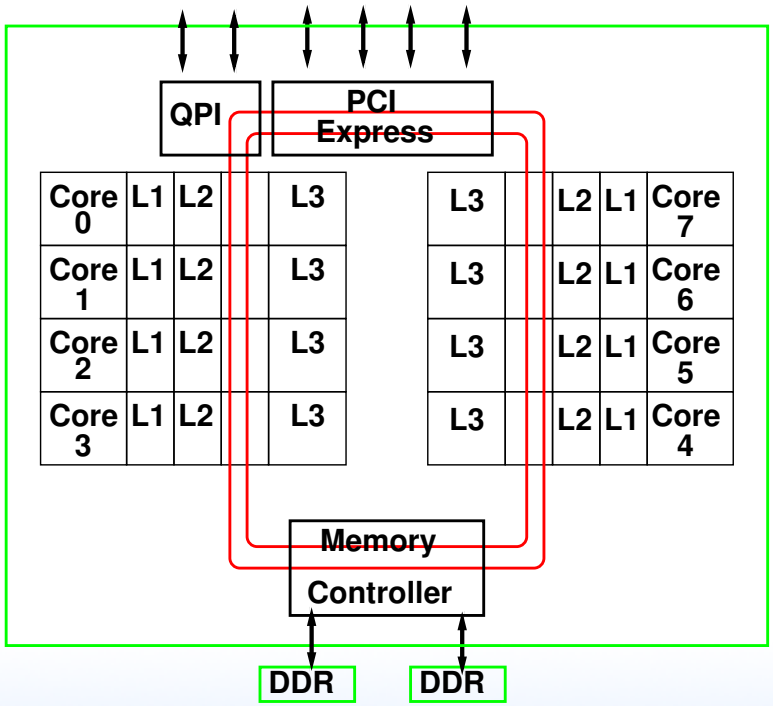


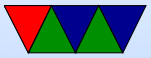
Example Real-World Cache layouts



Haswell-EP Cache Layout

Note: see "Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture" by Molka, Hackenberg, Schöne, Nagel.



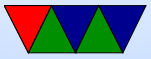


Haswell-EP Cache Parameters

- per core 32kB L1 I/D – 4 clocks
  64B/line, 8-Way
  (shared if hyper-threaded)
  writeback

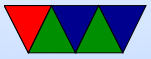
- mOp cache? 1.5K instructions, 8-way, 6Mop/line
  Loop stream detector, can execute w/o accessing icache

- per core 256kB L2 unified – 11 clocks

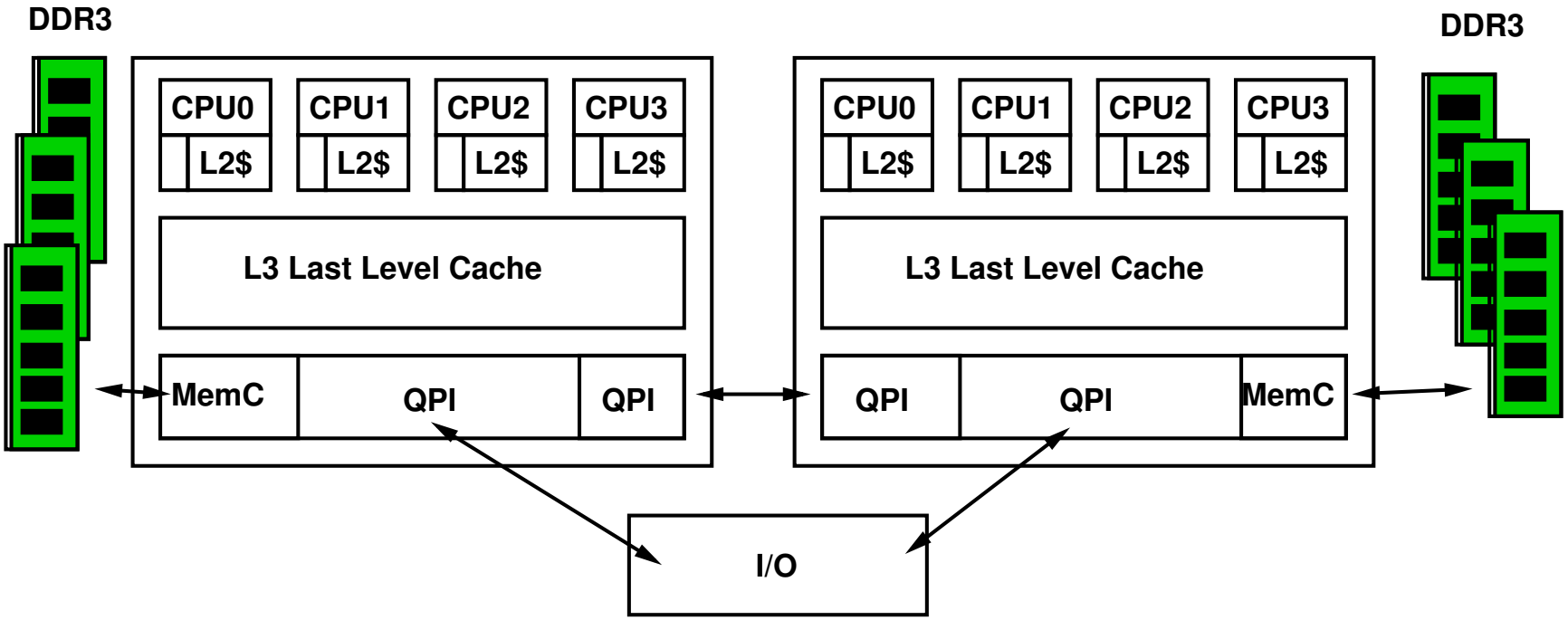


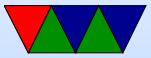
64B/line, 8-way
writeback. non-inclusive

- shared L3 20MB, writeback
- various hw prefetchers operating



SandyBridge Cache Layout



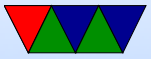


SandyBridge Cache Parameters

- per core 32kB L1 I/D – 4 clocks
  64B/line, 8-Way
  (shared if hyper-threaded)
  writeback

- mOp cache? 1.5K instructions, 8-way, 6Mop/line
  Loop stream detector, can execute w/o accessing icache

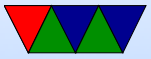
- per core 256kB L2 unified – 12 clocks



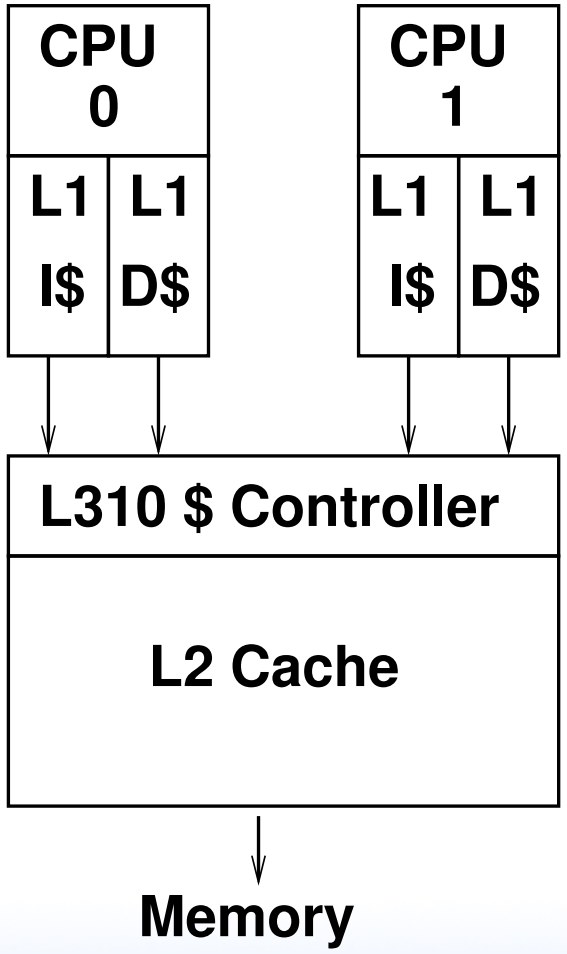
64B/line, 8-way
writeback. non-inclusive

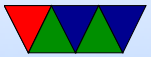
- shared L3 1MB-20MB – 26-31 clocks
  64B/line. 12-way (varies)
  writeback, inclusive

- various hw prefetchers operating



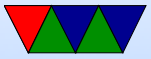
Cortex A9 Cache Layout





Cortex A9 Cache Layout

- OMAP4430 processor
- 32kB 4-way associative, separate L1-I and L1-D
  pseudo-round-robin or pseudo random replacement
  8-word line size (32B)
  critical-word first filling
  instruction: VIPT, data: PIPT
- Optional L2 cache controller
  Pandaboard has L310 L2 cache controller, 1MB 16-way



Optional prefetcher

- data cache reads/writes non-blocking, 4 outstanding misses
  write buffer: 4 64-bit, allowing write combining