CSCI 210: Computer Architecture
Lecture 37: Faster Caches

Stephen Checkoway
Oberlin College
Jan. 12, 2022
Slides from Cynthia Taylor

Announcements

- Problem Set 12 due Friday at midnight
- Cache Lab (final project) due on the final exam day
- Course Evals!
  - Extra credit for 90% response rate
  - Currently at 62% (as of 2022-01-12 at 09:00)
- Office Hours Friday 13:30 – 14:30
  - On Zoom

```
for(int i = 0; i<10,000,000;i++)
    sum+=A[i];
```

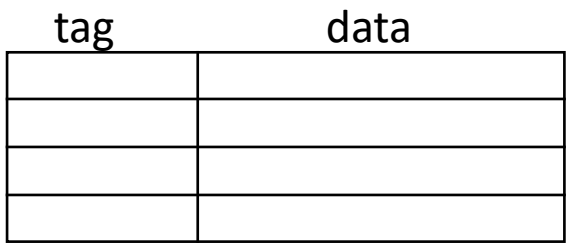
Assume each element of A is 4 bytes and sum is kept in a register. Assume a direct-mapped 32 kB cache with 32 byte blocks. Which changes would help the hit rate of the above code?

Selection	Change
A	Increase to 2-way set associativity
B	Increase block size to 64 bytes
C	Increase cache size to 64 KB
D	A and C combined
E	A, B, and C combined

Three types of cache misses

- Compulsory (or cold-start) misses
  - first access to the data.
- Capacity misses
  - we missed only because the cache isn't big enough.
- Conflict misses
  - we missed because the data maps to the same index as other data that forced it out of the cache.

address:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



DM cache

Cache Miss Type

Suppose you experience a cache miss on a block (let's call it block A). You have accessed block A in the past. There have been precisely 1027 different blocks accessed between your last access to block A and your current miss. Your block size is 32-bytes and you have a 64 kB cache (recall a kB = 1024 bytes). What kind of miss was this?

Selection	Cache Miss
A	Compulsory
B	Capacity
C	Conflict
D	Both Capacity and Conflict
E	None of the above

Multilevel Caches

- Primary (or level-1) cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- L-3 cache usually services multiple CPUs
- L-3 misses go to main memory

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 1 cycle/.25 ns
  - Miss rate/instruction = 2%
  - Main memory access time = 100 ns
- With just primary (L-1) cache
  - Miss penalty = 100 ns/(0.25 ns/cycle) = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

- Now add L-2 cache
  - Access time = 5 ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5 ns/(0.25 ns/cycle) = 20 cycles
- Main memory still 400 cycle penalty, L1 miss rate of 2%
- The Total CPI will be

A. $1 + 2 \times 20 + 5 \times 400$

B. $1 + 0.02 \times 20 + 0.005 \times 400$

C. $1 + 0.02 \times 20 \times 0.005 \times 400$

D. $1 + 0.0195 \times 20 + 0.005 \times 400$

Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size
  - L-1 less associative than L-2

Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue

Prefetching

- Hardware Prefetching
  - suppose you are walking through a single element in an array of large objects
  - hardware determines the “stride” and starts grabbing values early
- Software Prefetching
  - Compiler adds extra instructions to load data before it is needed

Which data structure will have better memory access times assuming you have a prefetcher?

A. ArrayList

B. Linked List

C. There will not be any difference

Writing Cache-Aware Code

- Focus on your working set
- If your “working set” fits in L1 it will be vastly better than a “working set” that fits only on disk.
- If you have a large data set – do processing on it in chunks.
- Think about regularity in data structures (can a prefetcher guess where you are going – or are you pointer chasing)

Cache Simulator

- Take in a datatrace of load/stores from a real program
- Simulate running the program on a given cache
- Calculate how well a given cache would perform for that trace

Cache Parameters

- Always: Write-allocate, write-back, LRU replacement

- Change:
  - Cache size
  - Block size
  - Associativity
  - Miss penalty

Address Trace

Load/Store Address InstructionCount

```
# 0 7fffed80 1
# 0 10010000 10
# 0 10010060 3
# 1 10010030 4
# 0 10010004 6
# 0 10010064 3
# 1 10010034 4
```

L/S: 0 for load, 1 for store

Simulation Results

```
Simulation results:
    execution time                 52268708 cycles
    instructions                    5136716
    memory accesses                 1957764
    overall miss rate                  0.79
    load miss rate                     0.88
    CPI                               10.18
    average memory access time        24.07 cycles
    dirty evictions                  225876
    load_misses                     1525974
    store_misses                      30034
    load_hits                        205909
    store_hits                       195847
```

What do you need to do?

- Create data structures that emulate a cache
- For each instruction, find where it would go in the cache, check if it’s already there
- Calculate number of miss penalty cycles, load misses, store misses, instructions, etc

Reading

- Next lecture: Class Wrap Up