CSCE 692 – Chapter 2 & Appendix B Homework Problems

2nd Lt David Crow – 12 January 2019

(100 Points)

Due: NLT 1100 Thursday, 29 January 2019

**Problems:**

**B.1 (a, b, c, d)** [5/5/5/7]

**B.2 (a, b)** [7/7]

**B.5 (a, b, c, d)** [12/12/12/8]

**2.17 (a, b, c)** [5/5/10]

**Instructions:**

* Write your name and a page number on each page
* Please show your work (in some cases, no work = no credit)
* Clearly indicate your answers
* Explain assumptions and provide references (sources) for any additional information you had to research in order to complete any problem.

**Notes:**

1. Answer all sub questions.
2. B.2, To simplify grading, adapt the general format of Fig B.30.
3. B.5, Problem Modification: Write buffer eliminates stalls for 97% of writes.
4. CACTI,
   1. Use CACTI inputs from handout
   2. Let assumed CPU cycle time be random cycle time (ps) for each case

B.1 [5/5/5/7] <B.1> You are trying to appreciate how important the principle of locality is in justifying the use of a cache memory, so you experiment with a computer having an L1 data cache and a main memory (you exclusively focus on data accesses). The latencies (in CPU cycles) of the different kinds of accesses are as follows: cache hit, 1 cycle; cache miss, 110 cycles; main memory access with cache disabled, 105 cycles.

1. [5] <B.1> When you run a program with an overall miss rate of 3%, what will the average memory access time (in CPU cycles) be?

According to the textbook, the average number of cycles for a cache hit is equal to . In plugging in the given numbers, we see that .

1. [5] <B.1> Next, you run a program specifically designed to produce completely random data addresses with no locality. Toward that end, you use an array of size 1 GB (all of which fits in the main memory). Accesses to random elements of this array are continuously made (using a uniform random number generator to generate the elements’ indices). If your data cache size is 64 KB, what will the average memory access time be?

We can use the same equation here. However, we are randomly accessing elements of the array, so the probability of getting a hit is thus the ratio of cache size to main memory size. Thus,

1. [5] <B.1> If you compare the result obtained in part (b) with the main memory access time when the cache is disabled, what can you conclude about the role of locality in justifying the use of cache memory?

Because , it’s clear that one is better off disabling the cache entirely than using the cache when the hit rate is low. In other words, when the hit rate is low enough, the overhead for using a cache negates any benefits gained from cache hits.

When the hit rate is low, we know that we cannot depend on the principle of locality. We can restate the above conclusion in this way: If one cannot depend on the principle of locality, one should not employ a cache. It’s simply not worth it.

1. [7] <B.1> You observed that a cache hit produces a gain of 104 cycles (1 cycle vs. 105), but it produces a loss of 5 cycles in the case of a miss (110 cycles vs. 105). In the general case, we can express these two quantities as G (gain) and L (loss). Using these two quantities (G and L), create a general formula to identify the highest miss rate after which the cache use would be disadvantageous.

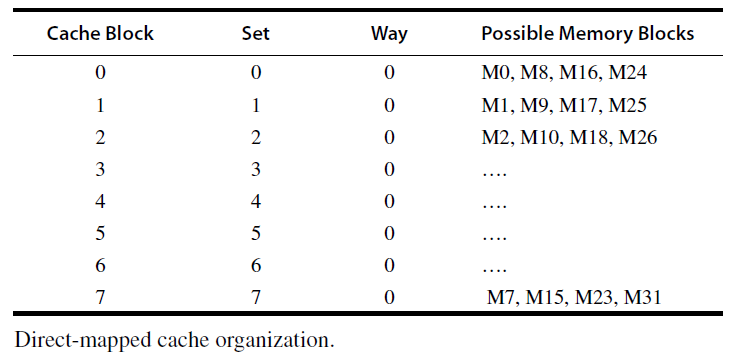
We know that . We can represent the average number of cycles when the cache is enabled as . Similarly, we can represent the average number of cycles when the cache is disabled as .

We now say that and . Thus,

.

If , we shouldn’t use a cache. We can simplify the equation to , which further simplifies to . In other words, if the miss rate is greater than or equal to , a cache is useless.

B.2 [7/7] For the purpose of this exercise, we assume that we have a 512-byte cache with 64-byte blocks. We will also assume that the main memory is 2 KB large. We can regard the memory as an array of 64-byte blocks: M0, M1, ..., M31. Figure B.30 sketches the memory blocks that can reside in different cache blocks if the cache was direct-mapped.



**Figure B.30 Memory blocks distributed to direct-mapped cache**

1. [7] Show the contents of the table if the cache is organized as a fully-associative cache.

Each block can reside in any of the cache blocks. We treat the entire cache as a single set. We can also call this an -way cache, where is the number of cache blocks. Here’s the table:

|  |  |  |  |
| --- | --- | --- | --- |
| Cache Block | Set | Way | Possible Memory Blocks |
| 0 | 0 | 0 | M0, M1, M2, …, M31 |
| 1 | 0 | 1 | M0, M1, M2, …, M31 |
| 2 | 0 | 2 | … |
| 3 | 0 | 3 | … |
| 4 | 0 | 4 | … |
| 5 | 0 | 5 | … |
| 6 | 0 | 6 | … |
| 7 | 0 | 7 | … |

1. [7] Repeat part (a) with the cache organized as a four-way set associative cache.

In an eight-block cache, four-way set associative means that we divide our cache into two sets of four ways each. Because we have two sets, each block can go into set .

|  |  |  |  |
| --- | --- | --- | --- |
| Cache Block | Set | Way | Possible Memory Blocks |
| 0 | 0 | 0 | M0, M2, M4, …, M30 |
| 1 | 0 | 1 | M0, M2, M4, …, M30 |
| 2 | 0 | 2 | … |
| 3 | 0 | 3 | … |
| 4 | 1 | 0 | M1, M3, M5, …, M31 |
| 5 | 1 | 1 | M1, M3, M5, …, M31 |
| 6 | 1 | 2 | … |
| 7 | 1 | 3 | … |

B.5 [12/12/12/8] You are building a system around a processor with in-order execution that runs at 1.1 GHz and has a CPI of 1.35 excluding memory accesses. The only instructions that read or write data from memory are loads (20% of all instructions) and stores (10% of all instructions).

The memory system for this computer is composed of a split L1 cache that imposes no penalty on hits. Both the I-cache and D-cache are direct mapped and hold 32 KB each. The I-cache has a 2% miss rate and 32-byte blocks, and the D-cache is write-through (no-write allocate) with a 5% read miss rate and 16-byte blocks. There is a write buffer on the D-cache that eliminates stalls for 97% of all writes.

The 512 KB write-back (write allocate), unified L2 cache has 64-byte blocks and an access time of 15 ns. It is connected to the L1 cache by a 128-bit data bus that runs at 266 MHz and can transfer one 128-bit word per bus cycle. Of all memory references sent to the L2 cache in this system, 80% are satisfied without going to main memory. Also, 50% of all blocks replaced are dirty.

The 128-bit-wide main memory has an access latency of 60 ns, after which any number of bus words may be transferred at the rate of one per cycle on the 128-bit-wide 133 MHz main memory bus.

*A useful tool for solving this type of problem is to extract all of the available information from the problem description. It is possible that not all of the information will be necessary to solve the problem but having it in summary form makes it easier to think about.*

1. [12] What is the average memory access time for instruction accesses?

We are told that .

It should be clear that . We’re told that .

We require to move a data block from the L2-cache to the I-cache. Because each transfer requires , we know .

Additionally, . We’re told that .

We require to move a data block from main memory to the L2-cache. Because each transfer requires , we know . Note that this is also the penalty we pay for dirty blocks.

Now we can just plug-and-play.

1. [12] What is the average memory access time for data reads?

We require to move a data block from the L2-cache to the D-cache. We calculated above that this takes .

This tells us that

Now we can evaluate.

1. [12] What is the average memory access time for data writes?

We know that . We must stall until the buffer is done writing to the L1-cache, so . We’re told that the L1-cache is write-through, so we’re always going to write to the L2-cache when writing to the L1-cache. In other words, and .

So far, then, we know that .

Now, . This is (we calculated this earlier).

Thus, .

1. [8] What is the overall CPI, including memory accesses?

The total CPI is equal to . The comes from the loads, which constitute 20% of all instructions. Similarly, the comes from the stores, which constitute 10% of all instructions.

That is, .

2.17 [5/5/10] <2.2> The following questions investigate the impact of small and simple caches using CACTI and assume a 32 nm (0.032 μm) technology.

Access time (ps)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Cache Size | | | | | | | |
| Associativity | 16KB | 32 KB | 64 KB | 128 KB | 1 MB | 2 MB | 4 MB | 8 MB |
| 1 | 301.4 | 387.6 | 477.7 | 731.6 | 1705 | 2014 | 2501 | 3010 |
| 2 | 322.2 | 406.5 | 523.6 | 771.8 | 1736 | 2046 | 2626 | 3046 |
| 4 | 344.2 | 429.1 | 520.6 | 778.3 | 1737 | 2044 | 2625 | 3069 |
| 8 | 561.6 | 574.2 | 663.0 | 917.6 | 1749 | 2053 | 2546 | 3056 |

Cycle time (ps)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Cache Size | | | | | | | |
| Associativity | 16KB | 32 KB | 64 KB | 128 KB | 1 MB | 2 MB | 4 MB | 8 MB |
| 1 | 183.9 | 183.9 | 238.9 | 183.9 | 127.2 | 127.2 | 127.2 | 128.2 |
| 2 | 181.3 | 183.9 | 236.2 | 183.9 | 127.2 | 127.2 | 121.4 | 127.2 |
| 4 | 240.8 | 181.3 | 236.2 | 181.3 | 127.2 | 127.2 | 121.4 | 124.7 |
| 8 | 149.1 | 155.2 | 181.3 | 155.2 | 124.7 | 124.6 | 124.7 | 124.7 |

1. [5] <2.2> Compare the access times of 64 KB caches with 64-byte blocks and a single bank. What are the relative access times of two-way and four-way set associative caches in comparison to a direct mapped organization?

I’ve copied the relevant values from the tables above into the table below. Additionally, I’ve added a column for relative access time, where *relative* means *relative to a direct-mapped cache of 64 KB.*

**Table 2.17a: Access time in picoseconds and relative speedup when varying the associativity of a 64 KB cache**

|  |  |  |
| --- | --- | --- |
| Associativity | Access Time (ps) | Relative Access Time |
| 1 | 477.7 | 1.0000 |
| 2 | 523.6 | 1.0961 |
| 4 | 520.6 | 1.0898 |

It’s clear that both two-way and four-way associativity perform slower than a direct-mapped cache. Interestingly, though, the two-way cache is the slowest of the three. This contradicts a preconceived notion I had of increasing access times as the number of ways increases.

1. [5] <2.2> Compare the access times of four-way set associative caches with 64-byte blocks and a single bank. What are the relative access times of 32 KB and 64 KB caches in comparison to a 16 KB cache?

Again, I’ve copied the relevant values into the table below. I’ve also added a column for relative access time, where *relative* means *relative to a four-way set associative cache of 16 KB.*

**Table 2.17b: Access time in picoseconds and relative speedup when varying the cache size of a four-way set associative cache**

|  |  |  |
| --- | --- | --- |
| Cache size | Access Time (ps) | Relative Access Time (ps) |
| 16 KB | 344.2 | 1.0000 |
| 32 KB | 429.1 | 1.2467 |
| 64 KB | 520.6 | 1.5125 |

We can see that, for the values tested here, smaller caches greatly outperform larger caches. Although the relative speed of the 64 KB cache compared to the 32 KB cache is not shown in the table, a simple calculation shows that, for these three tests, access time increases by more than each time we double the cache size.

1. [10] <2.2> For a 64 KB cache, find the cache associativity between 1 and 8 with the lowest average memory access time given that misses per instruction for a certain workload suite is 0.00664 for direct mapped, 0.00366 for two-way set associative, 0.000987 for four-way set associative, and 0.000266 for eight-way set associative cache. Overall, there are 0.3 data references per instruction. Assume cache misses take 10 ns in all models. To calculate the hit time in cycles, assume, for each associativity, the CPU will be built to have a cycle time equal to the random cycle time found using CACTI, which corresponds to the maximum frequency that cache configuration can operate without any bubbles in the pipeline.

We know that . This is the same as .

Here, we know that . This gives the following values:

* One-way:
* Two-way:
* Four-way:
* Eight-way:

Additionally, .

* One-way:
* Two-way:
* Four-way:
* Eight-way:

Finally, .

* One-way:
* Two-way:
* Four-way:
* Eight-way:

We now have everything we need to compare the average memory access times.

* One-way:
* Two-way:
* Four-way:
* Eight-way:

As we can see, the four-way set associative cache has the lowest average memory access time.