Computer Architecture Due on Monday, January 26, 2015

Assignment 1

CS3350B *University of Western Ontario*

PROBBLEM 1. [20 points] Download MatTran1.c and MatTran2.c and compile the two programs using the commands in the comments at the beginning of each file.

1.1 Tune the MatTran1 program by varying the value of the THRESHOLD parameter to get the best timing on your machine. What is the best value of the THRESHOLD?

1.2 What do the two programs do? Which approach does each program take?

Matrix transpose: MatTran2.c *using the naive for-loop method and* MatTran1.c *using the divide and conquer method.*

1.3 Show the CPU info of the machine that your measurement is undertaken.

1.4 Choose some proper performance metrics and use perf to measure them for both programs. Which program is faster? Explain briefly why one is faster than the other.

PROBBLEM 2. [15 points] In this exercise we look at memory locality properties of matrix computation. The following code is written in C. The A, B and D are integer matrices where elements within the same row are stored contiguously. Assume each word is a 32-bit integer.

Algorithm 1: Matrix multiplication

```
for (i = 0; i < 100; ++i) do
    for (j = 0; j < 100; ++j) do
        for (k = 0; k < 100; ++k) do
            $D[i][j] += A[i][k] * B[k][j]$;
```

2.1 How many 16-byte cache lines are needed to store all 32-bit matrix elements of A being referenced? Elaborate the calculation steps.

*16 byte * 8 / 32 bit = 4 elements, that is, 4 elements fit into 1 cache line. Matrix A has 100 * 100 =* 10^4 *elements, such that we need* 10^4 */ 4 = 2500 cache lines.*

2.2 References to which variables exhibit temporal locality? Explain the reasons.

$D[i][j]$*, repeatedly referenced during the time of* k *changing from 0 to 99.*

2.3 References to which variables exhibit spatial locality? Explain the reasons.

$A[i][k]$*, stride-1 reference pattern, access elements successively.*

PROBBLEM 3. [25 points] Media applications that play audio or video files are part of a class of workloads called "streaming" workloads; i.e., they bring in large amounts of data but do not reuse much of it. Consider a video streaming workload that accesses a 512 KB working set sequentially with the following address stream:

$$0, 2, 4, 6, 8, 10, 12, 14, 16, \ldots$$

In general, cache access time is proportional to capacity. Assume that main memory accesses take 60 ns and that memory accesses are 32% of all instructions. Consider a 64 KB direct-mapped L1 cache with a 32-byte cache line. This L1 cache is attached to a processor and its hit time is 0.85 ns.

3.1 What is the miss rate for the address stream above? (Explain the reasons.) How is this miss rate sensitive to the size of the cache or the working set?

*The cache has 64 * 1024 / 32 = 2048 cache lines. When byte 0 is accessed, we get a miss but we cache the entire 32-byte cache line, which implies that, all bytes from byte 0 till byte 31 (block 0) are cached. Since the cache is direct-mapped, this block is cached at entry 0. So far we got 1 miss and 15 hits. The same trend continues for all 512 * 1024 / 32 = 16384 blocks of the 512 KB working set (except that starting from block 2048, we will have to start replacing blocks we had cached earlier).*

The miss rate is 1/16 = 6.25%. This miss rate is totally insensitive to the size of the cache and the size of the working set. The only factor that affects the hit rate is the cache line size. All the misses experienced by this workload are cold cache misses.

3.2 What is the Average Memory Access Time for this processor?

*AMAT = <L1 Hit Time> + <L1 Miss Rate> * <L1 Miss Penalty>, such that 0.85 + 6.25% * 60 = 4.6 ns.*

3.3 Assuming a base CPI of 1.0 without any memory stalls and the L1 hit time determines the cycle time, what is the total CPI for this processor? What is the time per instruction for this processor?

Given that memory accesses are 32% of all instructions, we can deduce: Number of memory accesses / instruction = 0.32.

Given that main memory accesses take 60 ns and <cycle time> = <L1 hit time>, we can deduce: L1 miss penalty in cycles = 60 / 0.85 = 70.59.

*Therefore, we can compute <Average memory-stall cycles> = <Number of memory accesses / instruction> * <L1 miss rate> * <L1 miss penalty in cycles>, and thereby* CPI_{stall} *= <*CPI_{ideal} *> + <average memory-stall cycles>.*

CPI_{stall} *= 1 + 0.32 * 6.25% * 70.59 = 2.41.*

*Time per instruction = 2.41 * 0.85 ns = 2.05 ns.*

3.4 Consider a 16 MB direct-mapped L2 cache with 90% miss rate and 14.5 ns hit time. Is it better or worse to attach this L2 cache to the processor? (Assume a base CPI of 1.0 without any memory stalls.) Explain the reasons.

*AMAT = <L1 Hit Time> + <L1 Miss Rate> * <L1 Miss Penalty>*

*<L1 Miss Penalty> = <L2 Hit Time> + <L2 Miss Rate> * <L2 Miss Penalty>*

*AMAT = 0.85 ns + 6.25% * (14.5 ns + 90% * 60 ns) = 5.13 ns, which is worse to attach this L2 cache.*

PROBBLEM 4. [30 points] For a direct-mapped cache design with a 16-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
15-10	9-4	3-0

4.1 What is the cache line size (in words)?

Since the offset is 3-0, that is 4 bits, it implies 2^4 *bytes = 16 bytes = 4 words.*

4.2 How many entries (i.e. cache lines, or cache blocks) does the cache have?

Since the index is 9-4, that is 6 bits, for this direct-mapped cache, it implies 2^6 *sets = 64 entries.*

4.3 What is the ratio between total bits required for such a cache implementation over the data storage bits?

Total bits = 64 entries × (1 valid bit + 6 tag bits + 16 × 8 data bits) = 64 × 135.

Data bits = 64 entries × 16 × 8 data bits = 64 × 128.

Ratio = 64 × 135 / (64 × 128) = 1.0546875.

Starting from power on, the following byte-addressed cache references are recorded:
4, 182, 46, 6, 196, 94, 197, 22, 190, 54, 197, 265.

4.4 List the final state of the cache, with each valid entry represented as a record of <index, tag, data> in binary.

The solution is shown in Table 2.

Reference	Tag	Index	Offset
4	000000	000000	0100
182	000000	001011	0110
46	000000	000010	1110
6	000000	000000	0110
196	000000	001100	0100
94	000000	000101	1110
197	000000	001100	0101
22	000000	000001	0110
190	000000	001011	1110
54	000000	000011	0110
265	000000	010000	1001

Table 1: References in the binary representation

		Data block (4 words)			
Index	Tag	3	2	1	0
000000	0	Memory(7)	Memory(6)^{hit}	Memory(5)	Memory(4)
000001	0	Memory(23)	Memory(22)	Memory(21)	Memory(20)
000010	0	Memory(47)	Memory(46)	Memory(45)	Memory(44)
000011	0	Memory(55)	Memory(54)	Memory(53)	Memory(52)
...	-	-	-	-	-
000101	0	Memory(95)	Memory(94)	Memory(93)	Memory(92)
...	-	-	-	-	-
001011	0	Memory(191)	Memory(190)	Memory(189)	Memory(188)
001100	0	Memory(199)	Memory(198)	Memory(197)^{hit}	Memory(196)
...	-	-	-	-	-
010000	0	Memory(267)	Memory(266)	Memory(265)	Memory(264)
...	-	-	-	-	-

Table 2: The final state of the cache

4.5 What is the hit ratio? Elaborate the calculation steps.

Reference 6, 197 and 197 are the hits, such that the hit ratio is 3 / 12 = 1 / 4.

4.6 [Bonus: 10 points] If this cache is 2-way-set-associative, will it improve the hit ratio regarding to those recorded cache references? Explain the reasons.

No, the hit ratio will be the same, see Table 3.

		Data block (4 words)			
Index	Tag	3	2	1	0
000000	0	Memory(7)	Memory(6)^{hit}	Memory(5)	Memory(4)
000000	-	-	-	-	-
000001	0	Memory(23)	Memory(22)	Memory(21)	Memory(20)
000001	-	-	-	-	-
000010	0	Memory(47)	Memory(46)	Memory(45)	Memory(44)
000010	-	-	-	-	-
000011	0	Memory(55)	Memory(54)	Memory(53)	Memory(52)
000011	-	-	-	-	-
000101	0	Memory(95)	Memory(94)	Memory(93)	Memory(92)
000101	-	-	-	-	-
001011	0	Memory(183)	Memory(182)	Memory(181)	Memory(180)
001011	0	Memory(191)	Memory(190)	Memory(189)	Memory(188)
001100	0	Memory(199)	Memory(198)	Memory(197)^{hit}	Memory(196)
001100	-	-	-	-	-
010000	0	Memory(267)	Memory(266)	Memory(265)	Memory(264)
010000	-	-	-	-	-

Table 3: The final state of the 2-way-set-associative cache

PROBBLEM 5. [10 points] Recall that we have two write policies and write allocation policies, and their combinations can be implemented either in L1 or L2 cache. Assume the following choices for L1 and L2 caches:

L1	L2
Write through, non-write allocate	Write back, write allocate

5.1 Buffers are employed between different levels of memory hierarchy to reduce access latency. For this given configuration, list the possible buffers needed between L1 and L2 caches, as well as L2 cache and memory.

Between L1 and L2 caches, one write buffer is required. When the miss occurs, we directly update the portion of the block into the buffer, which will be waiting to be written into L2 cache, while the processor doesn't need to stall if the buffer is not full.

Between L2 cache and the memory, we require write and store buffers. When we have a cache miss, we must first write the block back to memory if the data in the cache is modified. In this situation, a write buffer is required to hold that data, such that the processor can continue the execution while that data is waiting to be written to the memory. In the meanwhile, a store buffer is used, such that the processor places the new data in the store buffer. Then when a cache hit occurs, this new data is written from the store buffer into the cache.

5.2 Describe the procedure of handling an L2 write-miss, considering the component involved and the possibility of replacing a dirty block.

First we check whether the block is dirty. If it is, then we write the dirty block to memory. Next, we retrieve the target block from memory (overwriting the block that is in our way). Finally we write to our L2 block.