Homework Assignment 3-Example Solution

CS/ECE 6810: Computer Architecture
October 31,2018
Name:
UID:

OoO and Cache Optimization

Due Date: November 11, 2018.
120 points

Important Notes:

- Solutions turned in must be your own. Please, mention references (if any) at the end of each question. Please refrain from cheating.
- All solutions must be accompanied by the equations used/logic/intermediate steps. Writing only the final answer will receive **zero** credit.
- All units must be mentioned wherever required.
- Late submissions **(after 11:59 pm on 11/11/2018)** will not be accepted.
- All submissions must be in PDF only. Scanned copies of handwritten solutions will not be accepted as a valid submission.
- write down any extra assumptions which are not mentioned in the questions.

1. **OoO Processor and Cache Parameters.** Please specify each of the following statement is True or False and explain why.

   i Instructions read all source operands from the register file in the Tomasulos algorithm.**(2 points)**
   False (1 point). The source operands can be provided either from the register file, common data bus, or load buffers (1 point).

   ii In a processor with hardware speculation, a store can always be committed as soon as its effective address and source value are available in the load-store queue.**(2 points)**
   False (1 point). The instruction are committed in order. If the earlier instruction cannot be committed for any reason, the later one has to wait even if all the operands and effective address are available (1 point).

   iii If cache block size remains same, but the number of cache blocks doubles, the compulsory misses remain the same.**(2 points)**
   True (1 point). Compulsory misses may be reduced by prefetching; However, as the block sizes remains the same, the compulsory misses will not change (1 point).

iv If the cache capacity is fixed but the block size is increased, the miss rate will not change.**(2 points)**

False(1 point). The miss rate may decrease, increase or remain same. The number of cache blocks will decrease and conflict misses may increase. Also, due to block size increase, the compulsory misses may decrease (i.e. for application with high spatial locality) (1 point).

v Write-through caches use write allocate mechanism to prevent writing dirty blocks to lower memory levels.**(2 points)**

False (1 point). A write-through cache does not prevent any writes to the lower layers (1 point).

2. **Memory Hierarchy.** Consider a processor with the following memory organization: L1 Cache, L2 Cache, L3 Cache and main memory. Each cache stores both tags and data. The data and tag arrays are going to be accessed with the same index value. Assume that the processor does serial tag/data look-up (first tag lookup and then data access) for L2 and L3 caches and parallel tag/data look-up for L1 cache. The table given below provides the time take to access the tag and data arrays in CPU cycles. Find the number of cycles required to complete 2000 load instructions accessing this hierarchy.**(15 points)**

Level	Tag Access	Data Access	Hit Rate
L1 (32 KB)	1	-	40%
L2 (1 MB)	4	12	50%
L3 (8 MB)	25	90	80%
Main Memory	-	500	-

We should go level by level ; In each new level, we suppose that we had miss in higher level; Number of required cycles:

L1: 2000*0.4*1 (2 points)

L2: 2000* 0.6 *0.50 * (1+4+12) (3 points)

L3: 2000 * 0.6 * 0.50* 0.80 (1+4+25+90) (3 points)

Main Memory: 2000 *0.6 * 0.50 * 0.20* (1+4+25+500) (3 points)

Number of required cycles : (2000*0.4*1) + (2000* 0.6 *0.50 * (1+4+12))+ (2000 * 0.6 * 0.50* 0.80 (1+4+25+90)) + (2000 *0.6 * 0.50 * 0.20* (1+4+25+500)) (2 point) = 132200 cycles (2 points)

3. **Cache Hit Rate.** Consider a 256KB main memory system with a direct mapped cache that stores up to 4 blocks. Each cache block comprises 4 words. At the beginning, cache is empty. Compute the hit rate for the following stream of addresses generated by the processor.(All memory addresses are in decimal format and each address refer to a word).**(10 points)**

```
170 , 257, 168, 246, 176, 175, 176, 177, 175, 176, 177, 175, 176, 177, 176,
175, 174, 173, 172, 171, 170, 169, 168, 167, 168, 165, 164
```

Since there is a direct mapped cache with 4 blocks (16 words) we know where an address is mapped. We just need to find the reminder of each address to 16. However, since the granularity is block, in the case of a miss, 1 block (4 words) are placed into cache. For instance, the reminder of 170 of 16 is 10. So, it has to be located in third block of cache; In this block the addresses which reminders of 16 are 8,9, 10, 11 should be held. Hence, at beginning, a block that contains 168, 169, 170, 171 will be placed in the third

block of cache. The same procure will be applied to remaining addressees. in the case of address conflict, an old cache block is replaced with new one. (3 points) Hence we have the following results:

`170->miss , 257->miss, 168->->hit, 246->miss, 176->miss, 175->miss, 176->hit, 177->hit, 175->hit, 176->hit, 177->hit, 175->hit, 176->hit, 177->hit, 176->hit, 175->hit, 174->hit, 173->hit, 172->hit, 171->hit, 170->hit, 169->hit, 168->hit, 167->miss, 168->hit, 165->hit, 164->hit` (6 points)

Hit rate is: $\frac{21}{27} = 77.7\%$ (1 point)

4. **Cache Performance.** Consider the following pseudo code; suppose that **X** and **Y** are allocated as contiguous integer arrays in memory and are aligned to the 4KB boundaries. Assume **k** and **l** are allocated in the register file with no need for memory accesses. We execute the code on a machine where the integer type (**int**) is 4 bytes wide.

```
#define M 1024
int X[M*M];
int Y[M*M];
int k, l;
  for (k = 0; k < M; k++) do
    for (l = 0; l < M; l++) do
      Y[l*M+k] = X[k*M+l];
    end for
  end for
```

i Consider a 4KB 2-way set-associative cache architecture while cache block size is 32-byte (8-word) and the replacement policy is LRU. Compute the hit rate of data accesses generated by `load`s and `store`s to **X** and **Y**?**(15 points)**

Question assumes that **X** and **Y** are allocated as contiguous integer arrays in memory. Hence, access to X (load)and Y (store) will not lead to conflict with each other in the cache. and the store and load hit rates can computed independently.

Load: when a block of X is loaded into cache, all words inside blocks are accessed successively (l is incremented by 1). So, in each 8 accesses, we have one compulsory miss for and 7 hits (6 points). The hit rate is $\frac{7}{8}$ (2 points)

Store: For store, l is multiplied by 1024 each time. The first access, is Y[0] and the second one is Y[1024] and so on. since the cache block size is 32-byte , store cannot exploit spatial locality. All stores during the first outer-loop iteration incur compulsory misses. After that, there are capacity misses (5 points). The store hit rate is 0% (2 points).

ii Consider a 4KB fully associative cache architecture with 32-byte blocks. The replacement policy is LRU. Rewrite the code to remove all of the non-compulsory misses. (You need to ensure the new code generate the exact same output in the main memory.You are allowed to add a nested for loop to the code if necessary.) Please provide explanation on how the new code can remove those misses.**(15 points)**

Based on the above explanation, we are allowed to rewrite the code such that that it can exploit spatial locality. Stores are not able to exploit the spatial locality in the original code. However, tiling (blocking) can help to increase spatial locality. For tiling, we have to break large blocks into sub-blocks that fit in the cache. Since the cache is fully associative, the location of the block is not important

Store: Consider the following code. We break the for loops by introducing tiling and adding one nested for-loop. When a word in a block is accessed, all other words are also accessed in next iterations of the third loop (when the t is increased) (3 points).

Load: At the same time, load can exploit spatial locality. 128 blocks can be placed in the cache (i.e. 64 blocks for loads and 64 blocks for stores). At the first iteration of the third loop, X[0], X[1024], ...,X[7168] are accessed. In the second iteration, X[1], X[1025],...,X[71669] are accessed and the same procedure happens in next iterations. As it can be observed, when a block is placed into cache, all words inside that are used. Hence, we have just compulsory misses (2 points).

Please trace the following pseudo code to understand better (10 points).

Note: The tiling parameters (TILE) may vary from 8 to 64.

```
#define M 1024
#define TILE 8
int X[M*M];
int Y[M*M];
int k, l;
  for (k = 0; k < M; k+ = TILE) do
    for (l = 0; l < M; l + +) do
      for (t = 0; t < TILE; t + +) do
        Y[l*M+(k+t)] = X[(k+t)*M+l];
      end for
    end for
  end for
```

5. **Cache Addressing.** Consider a processor using 3 cache levels. Level-1 cache is a 32KB direct mapped cache with 16B blocks used for both instructions and data. Level-2 is a 256KB, 2-way set associative cache with 64B cache lines. Level-3 is a 1MB, 4-way set associative cache with 64B cache lines. Assume that the processor can address up to 16GB of main memory. Compute the size of the tag arrays in KB for each level.**(15 points)**

   L1: (5 points)

   16 byte cache block; So the offset is 4 bit

   number of blocks = $\frac{32KB}{16B} = \frac{2^5 * 2^{10}}{2^4} = 2^{11}$;

   So, number of index bit is 11

   Main memory is 16GB= 2^{34} byte

   number of Tag bit = 34 - (11+4) = 19 bits

   L1: Data Array size : 32 KB; Tag Array Size = 19* 2^{11} bit = 38912 =4.75 KB

   L2: (5 points)

   64 byte cache block; So the offset is 6 bit

   number of blocks = $\frac{256KB}{64B*2} = 2^{11}$; So, number of index bit is 11

   Main memory is 16GB= 2^{34} byte

   number of Tag bit = 34 - (11+6) = 17 bits

   L2: Data Array size : 256 KB; Tag Array Size = 2*17* 2^{11} bit = 69632 = 8.5 KB

   L3: (5 points)

64 byte cache block; So the offset is 6 bit

number of blocks = $\frac{1MB}{64B*4} = 2^{14}$; So, number of index bit is 12

Main memory is 16GB= 2^{34} byte

number of Tag bit = 34 - (12+6) = 16 bits

L3: Data Array size : 1 MB; Tag Array Size = 4*16* 2^{12} bit = 262144 = 32 KB

6. **Cache and Memory Model using CACTI**

   CACTI (`http://www.cs.utah.edu/~rajeev/cacti7/`) is an integrated cache and memory model for access time, cycle time, area, leakage, and dynamic power. You are asked to use CACTI 7 for investigating the impact of small and simple caches using a 22nm CMOS technology node. Consider a processor using 2 cache levels. Level-1 cache is a 32 KB direct mapped cache with 16B blocks used for both instructions and data. Level-2 is a 1MB, 4-way set associative cache with 64B cache lines. Assume that the processor can address up to 2GB of main memory. Assume that both of the Level-1 and Level-2 caches are single bank.

   i Compare the access time, energy, leakage power, and area of the caches mentioned above.**(10 points)**

   ii Investigate the impact of varying associativity for 1, 2, and 8 on the access time, energy, leakage, and area of the Level-2 cache.**(10 points)**
   We will evaluate your answers based on your reported assumptions, observations, and results.

7. **Bonus Question.** The table below lists a sequence of loads and stores in the load-store queue (LSQ), when their one/two input operands are made available, and their computed effective addresses. Find when the address calculation happens for each load/store and when each load/store accesses the data memory. The processor assumes that loads do not depend on prior stores and issues loads speculatively. If the speculation is incorrect, the load must be re-issued. Assume that there are enough memory ports that you don't have to worry about structural hazards.**(20 points)**

LD/ST	The register for the address calculation is made available	The register that must be stored into memory is made available	The calculated effective addr (hexadecimal)	Addr calculation happens	Data memory accessed
LD	6	-	12345678	6 or 7	7 or 8
ST	9	8	87654321	9 or 10	10 or 11
LD	1	-	87654321	1 or 2	2 or 3
LD	5	-	12345678	5 or 6	6 or 7
ST	23	30	87654321	30 or 31	31 or 32
LD	4	-	87654321	4 or 5	5 or 6

Store commands are issued independently. Loads are issued speculatively. If the store update the locations in memory which loads reads from that location the value should be recovered. Based on the assumptions that how many cycles does it take to access to memory (1 or 2), or when the address calculation can start (the same cycle the register are available or the cycle after that). Please note that, for the case that the load should be recovered, we don't need to refer to the main memory again. It is forwarded from the the LSQ table directly.