Eric Tashji

Homework 4

Problem 1: In this exercise we look at memory locality properties of a matrix computation. The following code is written in C, where elements within the same row are stored contiguously. Assume each word is a 64-bit integer. Answer the following questions based on section 5.1 of the textbook.

```
For (j=0; j<100; j++)
        for(i=0; i<300; i++)
                A[i][j] = B[i][0] + C[i][j]
```

- How many 64-bit integers can be stored in a 16-byte cache block.

  Solution: 2 64-bit integers can be stored in a 16-byte cache block. This is because a byte contains 8 bits. Therefore, the cache block contains a total of 16 * 8 = 128 bits, and 128/64 = 2 integers.

- Which variable references exhibit temporal locality?

  Solution: The i and j variable references exhibit temporal locality because they are part of a loop. Because there is a high likelihood that they are going to be referenced again as part of the loop, they exhibit temporal locality.

- Which variable references exhibit spatial locality? Locality is affected by both the reference order and data layout.

  Solution: The indexes of the arrays A, B, and C exhibit spatial locality because there is a high likelihood that nearby elements within the array will be accessed as the loops iterate through the arrays.

Problem 2: Below is a list of 8-bit memory address references, given as word addresses. Answer the following questions based on Section 5.3 of the textbook.

0x43, 0xc4, 0x2b, 0x42, 0xc5, 0x28, 0xbe, 0x05, 0x92, 0x2a, 0xba, 0xbd

- For each of these references, identify the binary word address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list whether each reference is a hit or a miss, assuming the cache is initially empty.

  Solution: All

  1. 0x43:
     a. Binary word address: 01000011

   b. Tag: 0100
   c. Index: 0011
   d. Miss: A valid bit is not set/initial entry at the index.
2. 0xc4:
   a. Binary word address: 11000100
   b. Tag: 1100
   c. Index: 0100
   d. Miss: A valid bit is not set/initial entry at the index.
3. 0x2b:
   a. Binary word address: 00101011
   b. Tag: 0010
   c. Index: 1011
   d. Miss: A valid bit is not set/initial entry at the index.
4. 0x42:
   a. Binary word address: 01000010
   b. Tag: 0100
   c. Index: 0010
   d. Miss: A valid bit is not set/initial entry at the index.
5. 0xc5:
   a. Binary word address: 11000101
   b. Tag: 1100
   c. Index: 0101
   d. Miss: A valid bit is not set/initial entry at the index.
6. 0x28:
   a. Binary word address: 00101000
   b. Tag: 0010
   c. Index: 1000
   d. Miss: A valid bit is not set/initial entry at the index.
7. 0xbe:
   a. Binary word address: 10111110
   b. Tag: 1011
   c. Index: 1110
   d. Miss: A valid bit is not set/initial entry at the index.
8. 0x05:
   a. Binary word address:00000101
   b. Tag: 0000
   c. Index: 0101
   d. Miss: A valid bit is set, but the tags do not match.
9. 0x92:
   a. Binary word address: 10010010
   b. Tag: 1001
   c. Index: 0010
   d. Miss: A valid bit is set, but the tags do not match.
10. 0x2a:

    a. Binary word address: 00101010
    b. Tag: 0010
    c. Index: 1010
    d. Miss: A valid bit is not set/initial entry at the index.
11. 0xba:
    a. Binary word address: 10111010
    b. Tag: 1011
    c. Index: 1010
    d. Miss: A valid bit is set, but the tags do not match.
12. 0xbd:
    a. Binary word address: 10111101
    b. Tag: 1011
    c. Index: 1101
    d. Miss: A valid bit is not set/initial entry at the index.

- For each of these references, identify the binary word address, the tag, the index, and the offset given a direct-mapped cache with two-word blocks and a total size of eight blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

  Solution:
  1. 0x43:
     a. Binary word address: 01000011
     b. Tag: 0100
     c. Index: 001
     d. Miss: A valid bit is not set/initial entry at the index.
  2. 0xc4:
     a. Binary word address: 11000100
     b. Tag: 1100
     c. Index: 010
     d. Miss: A valid bit is not set/initial entry at the index.
  3. 0x2b:
     a. Binary word address: 00101011
     b. Tag: 0010
     c. Index: 101
     d. Miss: A valid bit is not set/initial entry at the index.
  4. 0x42:
     a. Binary word address: 01000010
     b. Tag: 0100
     c. Index: 001
     d. Hit: The valid bit has been set and the tags match (0x43).
  5. 0xc5:
     a. Binary word address: 11000101
     b. Tag: 1100
     c. Index: 010

    d. MIss: A valid bit has been set, but the tags do not match.
6. 0x28:
    a. Binary word address: 00101000
    b. Tag: 0010
    c. Index: 100
    d. Miss: A valid bit is not set/initial entry at the index.
7. 0xbe:
    a. Binary word address: 10111110
    b. Tag: 1011
    c. Index: 111
    d. Miss: A valid bit is not set/initial entry at the index.
8. 0x05:
    a. Binary word address: 00000101
    b. Tag: 0000
    c. Index: 010
    d. Miss: A valid bit has been set, but the tags do not match.
9. 0x92:
    a. Binary word address: 10010010
    b. Tag: 1001
    c. Index: 001
    d. Miss: A valid bit has been set, but the tags do not match.
10. 0x2a:
    a. Binary word address: 00101010
    b. Tag: 0010
    c. Index: 101
    d. Hit: A valid bit has been set and the tags match.
11. 0xba:
    a. Binary word address: 10111010
    b. Tag: 1011
    c. Index: 101
    d. Miss: A valid bit has been set, but the tags do not match.
12. 0xbd:
    a. Binary word address: 10111101
    b. Tag: 1011
    c. Index: 110
    d. Miss: A valid bit has not been set.

- You are asked to optimize a cache design for the given references. There are three direct-mapped cache designs possible, all with a total of 16 words of data. Which one is better for the above sequence of accesses?

    C1 has 1-word blocks,

    C2 has 2-word blocks, and

C3 has 4-word blocks.

Solution: The cache with the highest performance will be the cache with the least number of misses. For the above sequence of accesses, C1 has 12 misses and C2 has 10 misses. This can be shown through the previous two questions, as the first question describes a cache that functions the same as C1, and the second question describes a cache that functions the same as C2. The sequence for C3 can be shown below:

1. 0x43:
    a. Binary word address: 01000011
    b. Tag: 0100
    c. Index: 00
    d. Miss: A valid bit is not set/initial entry at the index.
2. 0xc4:
    a. Binary word address: 11000100
    b. Tag: 1100
    c. Index: 01
    d. Miss: A valid bit is not set/initial entry at the index.
3. 0x2b:
    a. Binary word address: 00101011
    b. Tag: 0010
    c. Index: 10
    d. Miss: A valid bit is not set/initial entry at the index.
4. 0x42:
    a. Binary word address: 01000010
    b. Tag: 0100
    c. Index: 00
    d. Hit: A valid bit has been set and the tags match.
5. 0xc5:
    a. Binary word address: 11000101
    b. Tag: 1100
    c. Index: 01
    d. Hit: A valid bit has been set and the tags match.
6. 0x28:
    a. Binary word address: 00101000
    b. Tag: 0010
    c. Index: 10
    d. Hit: A valid bit has been set and the tags match.
7. 0xbe:
    a. Binary word address: 10111110
    b. Tag: 1011
    c. Index: 11
    d. Miss: A valid bit has not been set.
8. 0x05:

a. Binary word address: 00000101
b. Tag: 0000
c. Index: 01
d. Miss: A valid bit has been set, but the tags do not match.

9. 0x92:
    a. Binary word address: 10010010
    b. Tag: 1001
    c. Index: 00
    d. Miss: A valid bit has been set, but the tags do not match.
10. 0x2a:
    a. Binary word address: 00101010
    b. Tag: 0010
    c. Index: 10
    d. Hit: A valid bit has been set and the tags match.
11. 0xba:
    a. Binary word address: 10111010
    b. Tag: 1011
    c. Index: 10
    d. Miss: A valid bit has been set, but the tags do not match.
12. 0xbd:
    a. Binary word address: 10111101
    b. Tag: 1011
    c. Index: 11
    d. Hit: A valid bit has been set and the tags match.

From the above data, we can see that C3 only has 6 misses. Therefore, we can conclude that C3 has the lowest miss rate, and is therefore the most efficient of the three.

Problem 3: Cache block size (B) can affect both miss rate and miss latency. Assuming a machine with a base CPI of 1, and an average of 1.35 references (both instruction and data) per instruction, find the block size that minimizes the total miss latency given the following miss rates for various block sizes.

Block Size	8	16	32	64	128
Miss Rate	5%	3%	2%	1.5%	1.1%

Answer the following questions based on Section 5.3 of the textbook.

- What is the optimal block size for a miss penalty of 30 x B cycles?

  Solution: We can determine the optimal block size by determining the optimal number of total cycles for each block size, given a program of x instructions. In this case, the

program run will have 1.0e6 instructions. The total number of clock cycles can be determined by adding the number of memory-stall cycles to the number of CPU execution cycles. We also know that the number of memory accesses is 1.35 references per instruction times 1.0e6 instructions, which equals 1.35e6 memory accesses. We also know that the number of memory-stall cycles can be calculated by multiplying the number of memory accesses per program times the miss rate times the miss penalty.

1. Block size: 8
    a. Memory-stall cycles = 1.35e6 / 1 x .05 x 30 x 8
    b. Memory-stall cycles = 1.35e6 x .05 x 240
    c. Memory-stall cycles = 1.62 e7 cycles
    d. Total cycles = 1.62e7 memory-stall cycles + 1.0e6 cpu Execution cycles = **1.72e7 total cycles**.
2. Block size: 16
    a. Memory-stall cycles = 1.35e6 / 1 x .03 x 30 x 16
    b. Memory-stall cycles = 1.35e6 x .03 x 480
    c. Memory-stall cycles = 1.944e7 cycles
    d. Total cycles = 1.944e7 memory-stall cycles + 1.0e6 cpu execution cycles = **2.044e7 total cycles.**
3. Block size: 32
    a. Memory-stall cycles = 1.35e6 / 1 x .02 x 30 x 32
    b. Memory-stall cycles = 1.35e6 x .02 x 960
    c. Memory-stall cycles = 2.592e7 cycles
    d. Total cycles = 2.592e7 memory-stall cycles + 1.0e6 cpu execution cycles = **2.692e7 total cycles.**
4. Block size: 64
    a. Memory-stall cycles = 1.35e6 / 1 x .015 x 30 x 64
    b. Memory-stall cycles = 1.35e6 x .015 x 1920
    c. Memory-stall cycles = 3.888e7 cycles
    d. Total cycles = 3.888e7 memory-stall cycles + 1.0e6 cpu execution cycles = **3.988e7 total cycles.**
5. Block size: 128
    a. Memory-stall cycles = 1.35e6 / 1 x .011 x 30 x 128
    b. Memory-stall cycles = 1.35e6 x .011 x 3840
    c. Memory-stall cycles = 5.7024 e7
    d. Total cycles = 5.7024e7 memory-stall cycles + 1.0e6 cpu execution cycles = **5.8024e7 total cycles.**

Using these calculations, we can determine that in this instance, a block size of 8 would be the optimal block size because it has the lowest number of total cycles, thereby resulting in the lowest total CPU time. **Therefore, 8 is the optimal block size in this instance.**

- If miss latency was constant (independent of B), what is the optimal bock size?

  Solution: We can answer this using the procedure for the previous question. However, the equation must be altered to accommodate the constant latency. Therefore, we will assume a constant penalty rate of 30 cycles, regardless of the block size.

  1. Bock size: 8
     a. Memory-stall cycles = 1.35e6 / 1 x .05 x 30
     b. Memory-stall cycles = 2.025e6 cycles
     c. Total cycles = 2.025e6 memory-stall cycles + 1.0e6 cpu execution cycles = **3.025e6 total cycles.**
  2. Block size: 16
     a. Memory-stall cycles = 1.35e6 x .03 x 30
     b. Memory-stall cycles = 1.215e6 cycles
     c. Total cycles = 1.215e6 memory-stall cycles + 1.0e6 cpu execution cycles = **2.215e6 total cycles.**
  3. Block size: 32
     a. Memory-stall cycles = 1.35e6 x .02 x 30
     b. Memory-stall cycles = 8.1e5 cycles
     c. Total cycles = 8.1e5 memory-stall cycles + 1.0e6 cpu execution cycles = **1.81e6 total cycles.**
  4. Block size: 64
     a. Memory-stall cycles = 1.35e6 x .015 x 30
     b. Memory-stall cycles = 6.075e5 cycles
     c. Total cycles = 6.075e5 memory-stall cycles + 1.0e6 cpu execution cycles = **1.6075e6 total cycles.**
  5. Block size: 128
     a. Memory-stall cycles = 1.35e6 x .011 x 30
     b. Memory-stall cycles = 4.455e5 cycles
     c. Total cycles = 4.455e5 memory-stall cycles + 1.0e6 cpu execution cycles = **1.4455e6 total cycles.**

     Using these calculations, we can determine that the optimal block size for a constant penalty rate would be **128 blocks**, because it has the lowest number of total cycles, and therefore the lowest CPU time.

Problem 4: Consider a byte addressing architecture with 64-bit memory addresses.

- Which bits of the address would be used in the tag, index, and offset in a direct-mapped cache with 512 1-word blocks.

  Solution: Note that the first 2 bits are ignored because they are used for the byte address.

  1. Offset: First 0 bits (after 2 bits for byte address).
  2. Index: Next 9 bits

3. Tag: Final 53 bits

- Which bits of the address would be used in the tag, index, and offset in a direct-mapped cache with 64 8-word blocks.

  Solution: Note that 2 bits are used for the byte address.
  1. Offset: First 3 bits (after 2 bits for byte address).
  2. Index: Next 6 bits.
  3. Tag: Final 53 bits.

- What is the ratio of bits used for storing data to total bits stored in the cache in each of the above cases?

  Solution: We can determine the total number of bits in a direct-mapped cache by multiplying the number of blocks times the block size plus the tag size plus the valid field size. We can determine the bits used for storing data by multiplying the number of blocks by the tag size + the valid field size

  Case 1:
  1. Total bits = 512 x (32 + 53 + 1) = 44032 bits
  2. Bits for Storing data = 512 x (53 + 1) = 27648
  3. 44032 / 27648 = **For every 1.59 total bits stored, 1 bit is used for storing the data.**

  Case 2:
  1. Total bits = 64 x (8 x 32 + (53 + 1)) = 19840 bits
  2. Bits for storing data = 64 x (53 + 1) = 3456 bits
  3. 19480 / 3456 = **For every 5.74 total bits stored, 1 bit used for storing the data.**

- Which bits of the address would be used in the tag, index and offset in a two-way set associative cache with 1-word blocks and a total capacity of 512 words?

  Solution: We find the total blocks by dividing the number of words per block by the total number of words (this results in 512). Then, we can divide the total number of blocks by the number of blocks per set. In this case, it is 2, so the result is 256. Finally, we do log2(256) to find the number of bits for the index. We can also find the offset by doing log2(1). Also keep in mind that 2 bits are used for the byte address.
  1. Offset: First 0 bits (after 2 bits for byte address).
  2. Index: Next 8 bits.
  3. Tag: Final 54 bits.