8 BONUS: Caching vs. Processing-in-Memory [80 points]

We are given the following piece of code that makes accesses to integer arrays A and B. The size of each element in both A and B is 4 bytes. The base address of array A is $0x00001000$, and the base address of B is $0x00008000$.

```
movi R1, #0x1000 // Store the base address of A in R1
movi R2, #0x8000 // Store the base address of B in R2
movi R3, #0

Outer_Loop:
    movi R4, #0
    movi R7, #0
    Inner_Loop:
        add R5, R3, R4  // R5 = R3 + R4
        // load 4 bytes from memory address R1+R5
        ld R5, [R1, R5] // R5 = Memory[R1 + R5],
        ld R6, [R2, R4] // R6 = Memory[R2 + R4]
        mul R5, R5, R6  // R5 = R5 * R6
        add R7, R7, R5  // R7 += R5
        inc R4          // R4++
        bne R4, #2, Inner_Loop // If R4 != 2, jump to Inner_Loop

    //store the data of R7 in memory address R1+R3
    st [R1, R3], R7     // Memory[R1 + R3] = R7,
    inc R3              // R3++
    bne R3, #16, Outer_Loop // If R3 != 16, jump to Outer_Loop
```

You are running the above code on a single-core processor. For now, assume that the processor *does not* have caches. Therefore, all load/store instructions access the main memory, which has a fixed 50-cycle latency, for both read and write operations. Assume that all load/store operations are serialized, i.e., the latency of multiple memory requests *cannot* be overlapped. Also assume that the execution time of a non-memory-access instruction is zero (i.e., we ignore its execution time).

(a) [15 points] What is the execution time of the above piece of code in cycles?

4000 cycles.

Explanation: There are 5 memory accesses for each outer loop iteration. The outer loop iterates 16 times, and each memory access takes 50 cycles.
$16 * 5 * 50 = 4000$ cycles

(b) [25 points] Assume that a 128-byte private cache is added to the processor core in the next-generation processor. The cache block size is 8-byte. The cache is direct-mapped. On a hit, the cache services both read and write requests in 5 cycles. On a miss, the main memory is accessed and the access fills an 8-byte cache line in 50 cycles. Assuming that the cache is initially empty, what is the new execution time on this processor with the described cache? Show your work.

900 cycles.

Explanation.

At the beginning A and B conflict in the first two cache lines. Then the elements of A and B go to different cache lines. The total execution time is 1910 cycles.

Here is the access pattern for the first outer loop iteration:

$0 - A[0], B[0], A[1], B[1], A[0]$

The first 4 references are loads, the last (A[0]) is a store. The cache is initially empty. We have a cache miss for A[0]. A[0] and A[1] is fetched to 0th index in the cache. Then, B[0] is a miss, and it is conflicting with A[0]. So, A[0] and A[1] are evicted. Similarly, all cache blocks in the first iteration are conflicting with each other. Since we have only cache misses, the latency for those 5 references is $5 * 50 = 250$ cycles

The status of the cache after making those seven references is:

Cache Index	Cache Block
0	~~A(0,1)~~, ~~B(0,1)~~, ~~A(0,1)~~, ~~B(0,1)~~, A(0,1)

Second iteration on the outer loop:
$1 - A[1], B[0], A[2], B[1], A[1]$

Cache hits/misses in the order of the references:
H, M, M, H, M
$Latency = 2 * 5 + 3 * 50 = 165$ cycles

Cache Status:
- A(0,1) is in set 0
- A(2,3) is in set 1
- the rest of the cache is empty

$2 - A[2], B[0], A[3], B[1], A[2]$

Cache hits/misses:
H, M, H, H, H
$Latency : 4 * 5 + 1 * 50 = 70$ cycles

Cache Status:
- B(0,1) is in set 0
- A(2,3) is in set 1
- the rest of the cache is empty

$3 - A[3], B[0], A[4], B[1], A[3]$

Cache hits/misses:
H, H, M, H, H
$Latency : 4 * 5 + 1 * 50 = 70$ cycles

Cache Status:
- B(0,1) is in set 0
- A(2,3) is in set 1
- A(4,5) is in set 2
- the rest of the cache is empty

$4 - A[4], B[0], A[5], B[1], A[4]$

Cache hits/misses:
H, H, H, H, H
$Latency : 5 * 5 = 25$ cycles

Cache Status:
- B(0,1) is in set 0
- B(2,3) is in set 1
- A(4,5) is in set 2
- the rest of the cache is empty

After this point, single-miss and zero-miss (all hits) iterations are interleaved until the 16th iteration.

Overall Latency:
$165 + 70 + (70 + 25) * 7 = 900$ cycles

(c) [15 points] You are not satisfied with the performance after implementing the described cache. To do better, you consider utilizing a processing unit that is available *close to the main memory*. This processing unit can directly interface to the main memory with a *10-cycle* latency, for both read and write operations. How many cycles does it take to execute the same program using the in-memory processing units? (Assume that the in-memory processing unit does not have a cache, and the memory accesses are serialized like in the processor core. The latency of the non-memory-access operations is ignored.)

800 cycles.

Explanation: Same as for the processor core without a cache, but the memory access latency is 10 cycles instead of 50. $16 * 5 * 10 = 800$

(d) [15 points] You friend now suggests that, by changing the cache capacity of the single-core processor (in part (b)), she could provide as good performance as the system that utilizes the memory processing unit (in part (c)).

Is she correct? What is the minimum capacity required for the cache of the single-core processor to match the performance of the program running on the memory processing unit?

No, she is not correct.

Explanation: Increasing the cache capacity does not help because doing so cannot eliminate the conflicts to Set 0 in the first two iterations of the outer loop.

(e) [10 points] What other changes could be made to the cache design to improve the performance of the single-core processor on this program?

Increasing the associativity of the cache.

Explanation: Although there is enough cache capacity to exploit the locality of the accesses, the fact that in the first two iterations the accessed data map to the same set causes conflicts. To improve the hit rate and the performance, we can change the address-to-set mapping policy. For example, we can change the cache design to be set-associative or fully-associative.