EE/CSCI 451
Fall 2020
Homework 1

Assigned: August 25, 2020
Due: September 1, 2020, before end of day Any time On Earth (AOE)
Total Points: 100

1 [20 points]

Explain the following terms:

1. Temporal locality
2. Very Long Instruction Word (VLIW) Processor
3. Single instruction multiple data (SIMD)
4. Cache hit ratio
5. Memory bound computation
6. Cache line
7. Instruction-level parallelism
8. Data dependencies
9. Superscalar processor
10. Column major layout

2 [20 points]

Consider a memory system with a single cycle cache and 200 cycle latency DRAM with the processor operating at 1 GHz. Assume that the processor has four multiply-add units and is capable of executing eight instructions in each cycle of 1 ns. Consider the problem of computing the dot product of two vectors on such a platform. A dot-product computation performs one multiply-add (2 FLOPs) on a single pair of vector elements, i.e., each floating point operation requires one data fetch.

1. In each memory cycle, the processor fetches one word (one vector element). What is the sustained performance in the best case (in FLOPS) of a dot product of two vectors? Is it a compute bound process or memory bound process?
2. In each memory cycle, the processor fetches sixteen words. What is sustained performance in the best case of a dot product of two vectors? Is it a compute bound process or a memory bound process?

```
/* dot product loop */
for (i = 0; i < dim; i++)
      dot_product += a[i] * b[i];
```

3 [20 points]

Consider a memory system with a single cycle cache and 100 cycle latency DRAM with the processor operating at 1 GHz. The processor has two multiply-add units and is capable of executing four instructions in each cycle. In each memory cycle, the processor fetches four words. Now consider the problem of multiplying a dense matrix with a vector using a two-loop dot-product formulation. The matrix is of dimension $2K \times 2K$. Thus, each row of the matrix takes 8 KB of storage. Assume the vector is cached, what is the sustained performance in the best case of this technique using a two-loop dot-product based matrix-vector product? (State your assumptions)

```
/* matrix-vector product loop */
for (i = 0; i < dim; i++)
        for (j = 0; j < dim; j++)
                c[i] += a[i][j] * b[j];
```

4 [10 points]

For the same memory system and processor in Problem 2, consider the problem of multiplying two dense matrices of dimension $8K \times 8K$. What is the sustained performance in the best case using a three-loop dot-product based formulation? (Assume that matrices are laid out in a row-major fashion. Also assume that 8 words can be fetched in one memory cycle)

```
/* matrix-matrix product loop */
for (i = 0; i < dim; i++)
        for (j = 0; j < dim; j++)
                for (k = 0; k < dim; k++)
                        c[i][j] += a[i][k] * b[k][j];
```

5 [15 points]

For the same memory system and processor in Problem 2, consider a program which needs to read w words from DRAM and reuse them for k times. The size of the cache line is equal to one word. We assume that the cache is large enough to store the w words and has an ideal cache placement strategy (none of the data items is overwritten by others). The computation itself of the program takes n cycles. What is the total execution time of the program, including the memory read time and computation time? (State your assumptions)

6 [15 points]

Read the slides (Cache Basics.pdf) posted on the Blackboard and solve the following problem. Assuming you execute the bubble sort algorithm listed below to sort a 64-element (N=64) array on a uni-processor; the size of each element is 1 word; the processor has a cache whose size is 8 words; the cache uses direct mapped scheme, first-in-first-out replacement policy, and write through policy; the cache line size is 2 words. Compute the cache hit ratio for read operations when executing the algorithm. Explain.

```
/* bubble sort */
for (i = 0; i < N; i++)
      for (j = 0; j < N-1; j++)
            R1 <- a[j];
            R2 <- a[j+1];
            if (R2> R1)
                  swap (R1, R2);
            end if
            a[j] <- R1;
            a[j+1] <- R2;
      end for
end for
```