Assignment 4

Due November 7, 12:59pm

NOTE: Late submissions will **NOT** be accepted. Please put your solutions in the CENG 355 **drop-box** (ELW, second floor) – they will be collected at **13:00**.

1. [5 points] Consider the code portion of the matrix-vector product computation as shown below: **(float) 128x128** matrix **A** is multiplied by **(float) 128x1** vector **X**, producing **(float) 128x1** result **Y** (initially all 0's).

```
for (i = 0; i < 128; i++) {
      for (j = 0; j < 128; j++) {
            Y[i] = Y[i] + A[i][j]*X[j];
      }
}
```

Storing **X**, **Y**, and **A** (each `float` array element is a 4-byte number) requires 128*4 + 128*4 + 128*128*4 = **65KB** of memory. If the cache (assume fully associative) is smaller than 65KB, the above code will yield many misses, considerably slowing down program execution. Alternatively, one can perform blocked computation: partition **A** into smaller blocks and perform the product computation block-by-block. If block data can fit into the cache, such blocked computation may significantly outperform the original code.

Rewrite the code fragment above using blocked computation and letting matrix **A**'s blocks be of size **64x64** (i.e., 4 blocks total). Assuming that such blocking yields the best performance, what can you say about the size of the cache?

2. [10 points] Consider a C code fragment below, modifying a given square matrix **float X[N][N]** (stored row by row, i.e., in the row-major order), where **N = 256**:

```
for (i = 0; i < N; i++) {
      average = 0;
      for (j = 0; j < N; j++) {
            average = average + X[i][j];   /* sum row elements */
      }
      average = average/N;             /* row average */
      for (j = 0; j < N; j++) {
            dev = X[i][j] – average; /* deviation from average */
            X[i][j] = dev*dev;       /* deviation squared */
      }
}
```

Determine the **X**-related page fault rate in the following two cases: (1) the main memory uses **1-KB** paging with four pages allocated for **X**, and (2) the main memory uses **4-KB** paging with only one page allocated for **X**. Initially, no part of **X** is in the main memory.

3. [5 points] Consider the **Good** and **Bad** code examples shown on **Slide 55** of the **"Memory"** lecture notes, where only one **4KB**-page was allocated for the array **int a[1024][1024]**. What would be the page fault rate in each example, if we allocate 512 **4KB**-pages for the array? What would be the page fault rate in each example, if we allocate only one **4KB**-page for the array, but let **M** = **N** = **512**?

4. [5 points] A computer with **4-GB** virtual memory has **1 GB** of physical memory and uses **1-MB** paging. It also has L1 and L2 caches for both instructions and data. The cache access times are $\mathbf{C_1 = 1\tau}$ (L1 hit) and $\mathbf{C_2 = 4\tau}$ (L1 miss, L2 hit). The main memory access time is $\mathbf{M = 16\tau}$ (L1 and L2 miss), and the page fault service time is $\mathbf{D = 10{,}000\tau}$.

(a) How many entries are there in the page table?

(b) Given that the hit rates are $\mathbf{h_1 = 95\%}$ (for L1) and $\mathbf{h_2 = 90\%}$ (for L2), and the page fault rate $\mathbf{p = 0\%}$ (no page faults), what is the average access time $\mathbf{T_{ave}}$?

(c) What is the average access time $\mathbf{T_{ave}}$, if $\mathbf{h_1 = 0\%}$, $\mathbf{h_2 = 90\%}$, $\mathbf{p = 0.01\%}$?

(d) What is the average access time $\mathbf{T_{ave}}$, if $\mathbf{h_1 = 95\%}$, $\mathbf{h_2 = 90\%}$, $\mathbf{p = 0.01\%}$?