# Assignment 5

## Due November 22, 14:59

**NOTE:** Late submissions will **NOT** be accepted. Please put your solutions in the ECE 355 drop-box (ELW, second floor) – they will be collected at **15:00**.

**1.** [5 points] Consider the code portion of the matrix-vector product computation as shown below: `(double)` **128x128** matrix **A** is multiplied by `(double)` **128x1** vector **X**, producing `(double)` **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 `double` array element is 8 bytes in size) requires 128*8 + 128*8 + 128*128*8 = **130KB** of memory. If the cache (assume fully associative) is smaller than 130KB, the above code will cause many misses, considerably slowing down the program execution. Alternatively, one can perform blocked computation: partition **A** into smaller blocks and perform the product computation block-by-block. If our data blocks 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, working on a given square matrix `double X[N][N]` (stored row by row, i.e., in the row-major order), where `N = 256`:

```
for (i = 0; i < N; i++) {
      double sum = 0;
      for (j = 0; j < N; j++) {
            sum = sum + X[i][j];
      }
      for (j = 0; j < N; j++) {
            X[i][j] = X[i][j]/sum;
      }
}
```

Determine the **x**-related page fault rate in the following two cases: 1) the main memory uses **2-KB** paging with two 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. **Note:** Type `double` is 64 bits (8 bytes).

**3.** [10 points] Consider a C code fragment below, modifying a given square matrix **`int X[N][N]`** (stored row by row, i.e., in the row-major order), where **`N = 256`**:

```
int average, dev;
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. **Note:** Type **`int`** is 32 bits (4 bytes).