Fall ’18 CIS 314 Assignment 6 – 100/100 points – Due Friday, 11/16, 11:59 PM

Please submit individual source files for coding exercises (see naming conventions below) and a single solution document for non-coding exercises (.txt or .pdf only). Your code and answers need to be documented to the point that the graders can understand your thought process. Full credit will not be awarded if sufficient work is not shown.

1. [20] Consider the following C code:

```
#define N 4
typedef int array_t[N][N];

int dim() {
    return N;
}

void f(array_t a, int x, int y) {
    for (int i = 0; i < dim(); ++i) {
        for (int j = 0; j < dim(); ++j) {
           a[i][j] = i * x * y + j;
        }
    }
}
```

Rewrite the above procedure *f* to minimize unnecessary function calls and multiplications.

Note that you are **not** required to replace the a[i][j] lookups with pointer arithmetic; focus on the *i * x * y + j* equation with respect to removing unnecessary multiplications.

Also write a main() function to test your procedure. Name your source file 6-1.c.

2. [80] Suppose we’ve got a procedure that computes the inner product of two arrays u and v. Consider the following C code:

```
void inner(float *u, float *v, int length, float *dest) {
    int i;
    float sum = 0.0f;
    for (i = 0; i < length; ++i) {
        sum += u[i] * v[i];
    }
    *dest = sum;
}
```

The x86-64 assembly code for the inner loop is as follows:

```
# u in %rbx, v in %rax, length in %rcx, i in %rdx, sum in %xmm1
.L87:
    movss (%rbx, %rdx, 4), %xmm0 # Get u[i]
```

```
mulss (%rax, %rdx, 4), %xmm0 # Multiply by v[i]
adds %xmm0, %xmm1            # Add to sum
addq $1, %rdx                # Increment i
cmpq %rcx, %rdx              # Compare i to length
jl .L87                      # If <, keep looping
```

a. (20) Diagram how this instruction sequence would be decoded into operations and show the data dependencies between them. Use Figure 5.14 as a guide. Include your diagram in your solutions document.

b. (20) Which operation(s) in the loop can NOT be pipelined? Why? What are the latencies of these operations? Based on this, what is the lower latency bound (in terms of CPE) of the procedure? Assume that *float* addition has a latency of 3, *float* multiplication has a latency of 5, and all integer operations have a latency of 1. Hint: think about which operation(s) depend on a result from the previous loop iteration. Write your answers in your solutions document.

c. (20) Implement a procedure *inner2* that is functionally equivalent to *inner* but uses four-way loop unrolling with four parallel accumulators. Also implement a main() function to test your procedure. Name your source file 6-2.c.

d. (20) Using your code from part c, collect data on the execution times of *inner* and *inner2* with varying array lengths. Summarize your findings and argue whether *inner* or *inner2* is more efficient than the other (or not). Create a graph using appropriate data points to support your argument. Include your summary and graph in your solutions document.

Zip the source files and solution document (if applicable), name the .zip file <Your Full Name>Assignment6.zip (e.g., EricWillsAssignment6.zip), and upload the .zip file to Canvas (see Assignments section for submission link).