# Project #5

Submit Assignment

**Due** Monday by 11:59pm **Points** 60 **Submitting** a file upload
**Available** May 14 at 12am - May 24 at 11:59pm 11 days

**CS 475/575 -- Spring Quarter 2018**

**Project #5**

**Vectorized Array Multiplication**

**60 Points**

**Due: May 21**

## Introduction

There are many problems in scientific and engineering computing where you want to multiply arrays of numbers (matrix manipulation, Fourier transformation, convolution, etc.).

This project is in two parts. The first part is to test array multiplication, SIMD and non-SIMD. The second part is to test array multiplication and reduction, SIMD and non-SIMD.

**Use the gcc or g++ compilers for both parts, but... Because simd.p5.cpp uses assembly language, this code is not portable. I know for sure it works on flip, using gcc/g++ 4.8.5. You are welcome to try it other places, but there are no guarantees. It doesn't work on rabbit. Do not use "-O3".**

# Requirements

1. Use the supplied SIMD SSE code to run an array multiplication timing experiment. Run the same experiment a second time using your own C/C++ array multiplication code.

2. Use the supplied SIMD SSE code to run an array multiplication + reduction timing experiment. Run the same experiment a second time using your own C/C++ array multiplication + reduction code.

3. Use different array sizes from 1K to 32M. The choice of in-between values is up to you, but pick something that will make for a good graph.

4. Feel free to run each array-size test a certain number of trials if you want. Use the peak value for the performance you record. Check peak versus average performance to be sure you are getting consistent answers. Try it again if the peak and average are not within, say, 20% of each other.

5. Create a table and a graph showing SSE/Non-SSE speed-up as a function of array size. Note: this is not a multithreading assignment, so you don't need to worry about a NUMT. Speedup in this case will be S = Psse/Pnon-see = Tnon-sse/Tsse (P = Performance, T = Elapsed Time). Plot both curves on the same set of axes.

6. The Y-axis performance units in this case will be "Speed-Up", i.e., dimensionless.

7. Be sure that the graphs are plotted so that "up" means "faster".

8. Your commentary write-up (turned in as a PDF file) should tell:
    1. What machine you ran this on
    2. Show the table and graph
    3. What patterns are you seeing in the speedups?
    4. Are they consistent across a variety of array sizes?
    5. Why or why not, do you think?
    6. Knowing that SSE SIMD is 4-floats-at-a-time, why could you get a speed-up of < 4.0 or > 4.0 in the array mutiplication?
    7. Knowing that SSE SIMD is 4-floats-at-a-time, why could you get a speed-up of < 4.0 or > 4.0 in the array mutiplication-reduction?

# SSE SIMD code:

- You are certainly welcome to write your own if you want, but we have already written Linux SSE code to help you with this.

  The two files that you want are **simd.p5.h** and **simd.p5.cpp**.

  A Makefile might look like this:

```
simd.p5.o:      simd.p5.h  simd.p5.cpp
                g++  -c  simd.p5.cpp -o simd.p5.o

arraymult:      arraymult.cpp simd.p5.o
                g++  -o arraymult  arraymult.cpp simd.p5.o  -lm  -fopenmp
```

- Note that you are linking in the OpenMP library because we are using it for timing.

- **Because simd.p5.cpp uses assembly language, this code is not portable. I know for sure it works on flip, using gcc/g++ 4.8.5. You are welcome to try it other places, but there are no guarantees. It doesn't work on rabbit. Do not use "-O3".**

- You can run the tests one-at-a-time, or you can script them by making the array size a #define that you set from outside the program.

# Warning!

*Do not use any optimization flags when compiling the simd.p5.cpp code.* It jumbles up the use of the registers.

*Do not use the icc or icpc compilers when compiling the simd.p5.cpp code.* It jumbles up the use of the registers.

# Grading:

| Feature | Points |
|---|---|
| Array Multiply numbers and curve | 20 |
| Array Multiply + Reduction numbers and curve | 20 |
| Commentary | 20 |
| **Potential Total** | **60** |