CS 475/575 -- Spring Quarter 2022

Project #4

Vectorized Array Multiplication/Reduction using SSE

60 Points

Due: May 11

This page was last updated: May 11, 2022

Introduction

There are many problems in scientific and engineering computing where you want to multiply arrays of numbers together and add up all the multiplies to produce a single sum (Fourier transformation, convolution, autocorrelation, etc.): sum = ΣA[i]*B[i]

This project is to test array multiplication/reduction using SIMD and non-SIMD.

For the "control groups" benchmarks, do not use OpenMP parallel for-loops. Just use straight C/C++ for-loops. In this project, we are only using OpenMP for the timing.

Requirements

1. Use the supplied SIMD SSE assembly language 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.

2. Use different array sizes from 1K to 8M. The choice of in-between values is up to you, but pick values that will make for a good graph.

3. Run each array-size test a certain number of trials. Use the peak value for the performance you record.

4. Create a table and a graph showing SSE/Non-SSE speed-up as a function of array size. Speedup in this case will be (P = Performance, T = Elapsed Time):

   S = Psse/Pnon-sse = Tnon-sse/Tsse

5. **Note: this is not a multithreading assignment, so you don't need to worry about a NUMT. Don't use any OpenMP-isms except for getting the timing.**

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

7. Parallel Fraction doesn't apply to SIMD parallelism, so don't compute one.

8. Your commentary write-up (turned in as a separate PDF file) should tell:
    1. What machine you ran this on
    2. Show the table of performances for each array size and the corresponding speedups
    3. Show the graph of SIMD/non-SIMD speedup versus array size (either one graph with two curves, or two graphs each with one curve)
    4. What patterns are you seeing in the speedups?
    5. Are they consistent across a variety of array sizes?
    6. Why or why not, do you think?

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.

  Find starter code in the file: all04.cpp.

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

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

- 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 this code. It jumbles up the use of the registers.

+5 points Extra Credit

Combine multithreading and SIMD in one test. In this case, you will vary *both* the array size and the number of threads (NUMT). Show your table of performances. Produce a graph similar to the one on Slide #20 of the *SIMD Vector* notes, using your numbers. Add a brief discussion of what your curves are showing and why you think it is working this way.

Grading:

Feature	Points
Array Multiply/Reduction performances and speedups	20
Array Multiply/Reduction speedup curve	20
Commentary	20
Extra Credit	+5
Potential Total	**65**