CS 475/575 -- Spring Quarter 2021

Project #4

Vectorized Array Multiplication/Reduction using SSE

60 Points

Due: May 12

This page was last updated: March 28, 2021

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 usinf 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 intrinsics 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 something 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 S = Psse/Pnon-sse = Tnon-sse/Tsse (P = Performance, T = Elapsed Time).

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 (one curve only)
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 this code in the file: simd.intrinsics.cpp.

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

- 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.

+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 #19 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**