Hsiang Lo

Project #4

Objective

Part 1 – Test array multiplication, SIMD and non-SIMD
Part 2 – Test array multiplication and reduction, SIMD and non-SIMD

Requirements

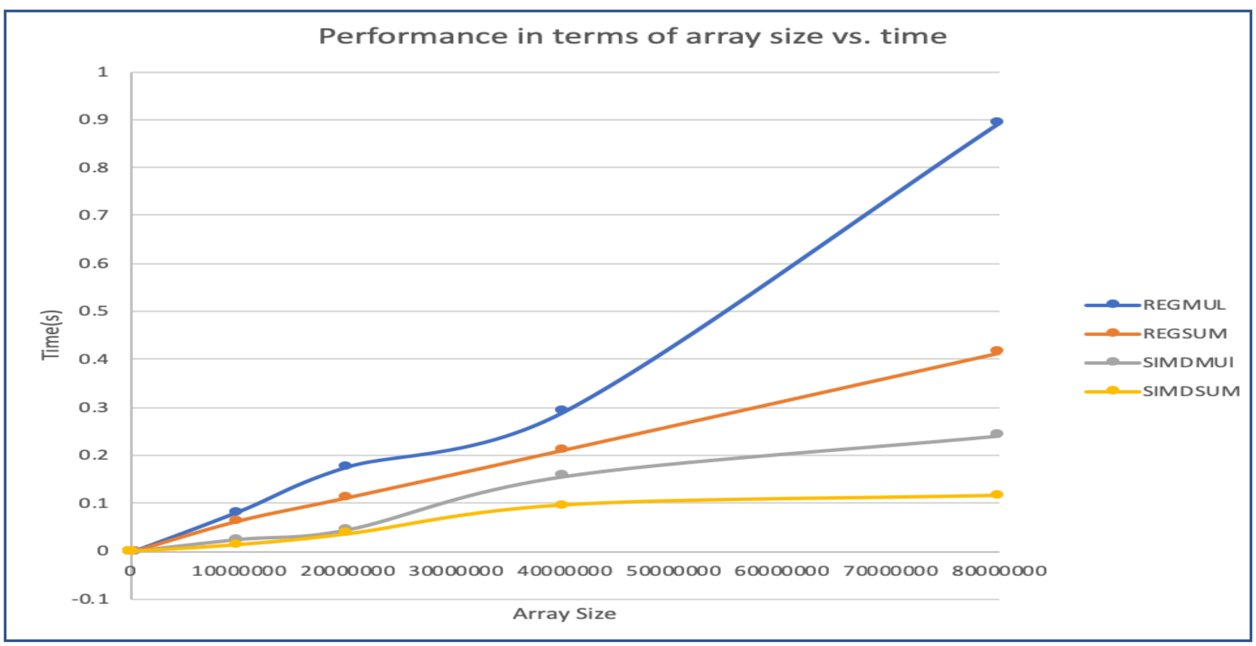
- Use different array sizes from 1K to 32M
- Run each experience a certain number of trials. Use the peak value for record.
- Create a table and a graph showing SSE/Non-SSE speed-up as a function of array size. Speedup will be S = Psse/Pnon-sse = Tnon-sse/Tsse (P = Performance, T = time)

Notes

- _asm = You're no longer in C, but assembly
- Link openmp library for timing using -fopenmp
- Do not use intel, any optimization flags and do not use **-O3**

1. **What machine you ran this on?** My project was ran on my MacBook pro 2017, same as my previous projects in order to main consistency. The project was then ran on school's server, flip.

2. **Show the table and graph**



	Size	wTime
REGMUL	1000	0.000009
REGMUL	2000	0.000014
REGMUL	4000	0.000034
REGMUL	8000	0.000071
REGMUL	16000	0.000144
REGMUL	32000	0.000288
REGMUL	64000	0.000599
REGMUL	128000	0.001203
REGMUL	256000	0.002378
REGMUL	512000	0.004810
REGMUL	1E+07	0.0813
REGMUL	2E+07	0.17451
REGMUL	4E+07	0.29061
REGMUL	8E+07	0.89285
REGSUM	1000	0.000005
REGSUM	2000	0.000010
REGSUM	4000	0.000021
REGSUM	8000	0.000042
REGSUM	16000	0.000078
REGSUM	32000	0.000156
REGSUM	64000	0.000382
REGSUM	128000	0.000665
REGSUM	256000	0.001335
REGSUM	512000	0.002701
REGSUM	1E+07	0.06232
REGSUM	2E+07	0.11087
REGSUM	4E+07	0.21093
REGSUM	8E+07	0.4136
SIMDMUl	1000	0.000002
SIMDMUl	2000	0.000002
SIMDMUl	4000	0.000003
SIMDMUl	8000	0.000006
SIMDMUl	16000	0.000013
SIMDMUl	32000	0.000028
SIMDMUl	64000	0.000061
SIMDMUl	128000	0.000120
SIMDMUl	256000	0.000283
SIMDMUl	512000	0.000811
SIMDMUl	1E+07	0.02418
SIMDMUl	2E+07	0.04445
SIMDMUl	4E+07	0.15649
SIMDMUl	8E+07	0.2417
SIMDSUM	1000	0.000001
SIMDSUM	2000	0.000001
SIMDSUM	4000	0.000003
SIMDSUM	8000	0.000005
SIMDSUM	16000	0.000010
SIMDSUM	32000	0.000020
SIMDSUM	64000	0.000048
SIMDSUM	128000	0.000118
SIMDSUM	256000	0.000204
SIMDSUM	1E+07	0.01364
SIMDSUM	2E+07	0.03582
SIMDSUM	4E+07	0.09541
SIMDSUM	8E+07	0.1152

3. **What patterns are you seeing in the speedups?**

   Starting at around 40,000,000 array sizes, the speedup from using SIMD is very clear from opposed to regular methods without using SIMD. This is most noticeable between SIMDMUL and REGMUL.

4. **Are they consistent across a variety of array sizes?**

   Yes, the array sizes appear to graph consistently from one another. The regular functions shows an extremely high increases in time as array size gets bigger. SIMD functions while also shows increases in time as data sizes get bigger, but not by much in comparison to regular functions.

5. **Why or why not, do you think?**

   All functions behave this way is due to the fact that I tried to increase the data sizes at the multiple of 4 for all four functions, and due to the same usage of variables, the graph remain constant as a result.

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-multiplication?**

   This Is due to the fact that SSE SIMD code provided to us is in assembly language. Using assembly language provides faster compute time for computer as there is less communication time, or decomposing reading level code such as C to assembly which is necessarily for computer to understand and compute. Further, since the time result is extremely small to begin with, the difference in using machine language is more noticeable.

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?**

   This Is due to the fact that SSE SIMD code provided to us is in assembly language. Using assembly language provides faster compute time for computer as there is less communication time, or decomposing reading level code such as C to assembly which is necessarily for computer to understand and compute. Further, since the time result is extremely small to begin with, the difference in using machine language is more noticeable.