Programming Assignment 3
Due Tuesday, February 21st

Objectives
Learn about and practice using: small-scale vector programing using SSE.

Prerequisites (to be covered in class or through examples in on-line documentation)
HW – Vector processing
SW – SSE instructions and their use
Programming – alignment, use of intrinsics, mapping C to vector instructions (e.g., with struct/union)
Note: For some intrinsics you need to enable AVX. Use the '-mavx2' option to make it work.

Assignment

Part 1 -- SSE extensions using C structs and union
Reading: B&O web extension "Achieving Greater Parallelism with SIMD Instructions."
Given: test_combine8.c, test_dot.c

- Read the reading and the code. You will notice that solutions to B&O (web extension) practice problems 1, 3, and 4 have been implemented in the two **.c** files.
- Compile and run test_combine8.c using float and "+". Plot the results and get the CPE. Justify the vector results (also comparing with the scalar results).
- Currently test_combine8.c has a function that does vector unrolling using 4 accumulators. Write code for two more functions, with 2 and 8 accumulators, respectively. Plot the results, get the CPEs, and justify.
- Recompile using double rather than float. Does having 8 accumulators still help?
- Compile and run test_dot8.c using float. Plot the results and get the CPE. Justify the vector results (also comparing with the scalar results).
- Currently test_dot8.c has vector unrolling using 2 accumulators. Write code for a new functions with 4 and 8 accumulators. Plot the results, get the CPEs, and justify.

Hand in: results, code, and explanations of results. Explain why the CPEs are different for dot and combine and the various unrollings.

Part 2 -- SSE extensions using intrinsics.
Reading: Alex Fr "Introduction to SSE Programming"
Given: test_intrinsics.c

- Read the reading and the code.
- Compile and run test_intrinsics.c. Plot the results and get the CPEs. Is this what you expected?
- Create two simple functions to get execution time baselines: element-wise add and multiply (float only). What is the CPE? Can vectorized these functions to make your code throughput optimal?
- Create a vectorized dot product function using intrinsics, in particular, using the dot product primitive (in the class notes last section or find a description on line). Plot results and get CPE. Compare this dot product with the vector approach from Part 2.
- Answer the following question: How does this approach compare with that in Part 2? What are the specific differences (performance, programmability, etc.) and when do they matter? (Please note – unlike most other questions in these assignments, this one is mostly qualitative.)

Hand in: modified code, description, results, answers to questions.

Part 3 -- A simple SSE application from scratch: Transpose

Given: test_transpose.c -- optimized from Lab 1

- Create the fastest transpose you can. Try using the SSE transpose intrinsic. Try combining the transpose intrinsic with blocking (from Lab 1).
- Compile test_transpose.c with -O2 and -O3 options and compare with your version.

Hand in: modified code, description, results, and analysis.