COSC 530 – Fall 2020
Programming Assignment 3
Exploiting the Streaming SIMD Extension (SSE)
DUE: 11:59pm on 12/4

Your assignment is to make an efficient dot product (sum of products) function. Below is a conventional dot production function implementation in C.

```
float dotprod(float x[], float y[], int n) {
    int i; float sum;
    sum = 0.0;
    for (i = 0; i < n; i++) {
        sum += x[i] * y[i];
    }
    return sum;
}
```

You are required to implement other versions of the dotprod function, where each function is placed in a separate le. The versions are (1) loop is unrolled by a factor of 4 (`ur4_dotprod.c`) where you can assume n is an integer multiple of 4, (2) accumulator expansion is applied on the unrolled loop (`ae4_dotprod.c`), and (3) SSE intrinsics are used to process 4 elements of the arrays for each operation within each iteration of the loop (`sse_dotprod.c`). You may find the following SSE intrinsics useful when implementing your version of `sse_dotprod.c`.

`__m128`

A typedef used to declare a variable that can hold four single-precision floating-point values. `__m128` variables can be assigned values from other `__m128` variables, functions that return an `__m128` value, or from a four element float array initializer.

`__m128 _mm_load_ps(float *p);`

Is the prototype for a function that loads four single-precision oating-point values from the address specied in `p` and returns the four values as a `__m128` value. `p` must be aligned on a 16 byte boundary.

`void _mm_store_ps(float *p, __m128 v);`

Is the prototype for a function that stores the `__m128` value in `v` at the address specified in `p`, which must be aligned on a 16 byte boundary.

`__m128 _mm_add_ps(__m128 a, __m128 b);`

Is the prototype for a function that returns the sum of the __m128 values in `a` and `b`.

`__m128 _mm_mul_ps(__m128 a, __m128 b);`

Is the prototype for a function that returns the product of the `__m128` values in `a` and `b`.

Use the following C preprocessor command to obtain access to the typedef and the prototypes for the intrinsics.

```
#include <emmintrin.h>
```

You can precede an array declaration by the following specification to align the array on a 16 byte boundary.

```
__attribute__ ((aligned(16)))
```

You should test the efficiency of your solutions on one of the hydra machines using the `test.c` and run scripts that are in the starter package. You should test each dot product implementation with the array sizes of 32768, 65536, 131072, and 262144. You should also plot the timing results as a bar graph, including the results with the regular implementation of dot product solution shown above and included in `reg_dotprod.c` in the starter package.

For your submission, you should create a gzipped tar file of your project directory with the completed `ur4_dotprod.c`, `ae4_dotprod.c`, and `sse_dotprod.c` implementations, as well as a pdf file containing the timing results of your experiments. Upload this tar file to canvas before due date and time specified at the top of this assignment. Partial credits will be given for incomplete efforts. However, a program that does not compile or run will get 0 points. Point breakdown is below:

- `ur4_dotprod.c` (15)
- `ae4_dotprod.c` (20)
- `sse_dotprod.c` (50)
- results with bar graph (15)