LAB 2 - USING SSE FOR ACCELERATING VECTOR MATRIX MULTIPLICATION

List of files:

- *matrix.c*: containing c source code for the matrix multiplication application
- *run.sh*: executing script
- *lab2.pdf*: this file - the description file

During the second lab, you are learning/practising how to accelerate an application with Streaming SIMD Extensions (SSE). The application in this lab is vector matrix multiplication. Given a vector v with n elements, a matrix M with $n \times n$ elements, the application computes the product vector $r[n] = v[n] \times M[n \times n]$ using both the normal execution and the SSE execution. Please note that, in the current version, the size of the input matrix and vector has to be a multiple of 4 ($n = m * 4, \forall m \in N$). In this version, the element of the vector and the matrix is single-precision floating-point ($float$ data type).

How to use the script file? The script file automates the following steps: it compiles, and executes the application. The output is the execution time for both the normal execution and the SSE execution. To execute the application, you need to specify the size of the matrix and the vector (value n). Below is an example in which n is equal to 1000.

$\$ > ./run\_sse.sh$ 1000

Here is the example output:

SEQUENTIAL EXECUTION: 0.009507 (sec)
PARALLEL EXECUTION: 0.003652 (sec)

This means that the execution time of the normal execution is $9.507ms$ and of SSE is $3.652ms$.

What will you do during the lab:

1. **Task 1**: Sketch the speed-up of SSE compared to the normal execution diagram with different input matrix and vector size.
2. **Task 2**: Modify the current SSE version of the application to have a new application which can accept an arbitrary size, and sketch the speed-up.
3. **Task 3**: Modify the application to calculate with double-precision floating-point (*double* data type) and sketch the speed-up .
4. **Task 4**: Modify the application to calculate the product of two matrices with SSE.
5. **Task 5 - bonus**: Accelerate/Modify the application with other techniques (e.g. AVX) to acheive a higher speed-up than the current SSE implementation.

Write a short report describing your results. Please add appropriate discussion and explanation.