Problem 1

Consider the following C code that adds two matrices. Assume a float takes 4 bytes to store. Also assume that for all parts of this problem, the addresses of arrays \( a, b \) and \( c \) are stored in registers \( Ra, Rb \) and \( Rc \), respectively.

register int i, j; /* i, j are in the processor registers */

float a[256][256], b[256][256], c[256][256];

for (int i = 0; i < 256; i++) {
    for (int j = 0; j < 256; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}

Part A – MIPS

Convert the given code to MIPS assembly in a straightforward way; i.e. do not worry about optimizations like instruction reordering, loop unrolling, software pipelining etc. How many dynamic instructions does the program take?

Solution:

```
ADDI R1, R0, #256 /* 1 */

LOOp1: ADDI R2, R0, #256 /* 2 */

LOOp2: LD F1, 0(Ra) /* 3 */
        LD F2, 0(Rb) /* 4 */
        ADD.D F3, F1, F2 /* 5 */
        SD F3, 0(Rc) /* 6 */
        ADDI Ra, Ra, #4 /* 7 */
        ADDI Rb, Rb, #4 /* 8 */
        ADDI Rc, Rc, #4 /* 9 */
        SUBI R2, R2, #1 /* 10 */
        BNEZ R2, LOOp2 /* 11 */
```
The 9 instructions in the inner most loop (3-11) are executed 256*256 = 65536 times. Instructions 2, 12 and 13 are executed 256 times and instruction 1 is only executed once. Total = 9*65536 + 3*256 + 1 = 590593 dynamic instructions.

**Part B – VMIPS**

Convert the given code to VMIPS assembly, again, in a straightforward way. You should use the vector ISA provided in Figure 4.3 of the book. Assume a maximum vector length (MVL) of 64. How many dynamic instructions does the code take in this case?

**Solution:**

```assembly
ADDI R1, R0, #64 // 1
MTC1 VLR, R1 // 2
ADDI R1, R0, #256 // 3
LOOP1:
ADDI R2, R0, #256 // 4
LOOP2:
LV V1, Ra // 5
LV V2, Rb // 6
ADDVV.D V3, V1, V2 // 7
SV V3, Rc // 8
ADDI Ra, Ra, #256 // 9
ADDI Rb, Rb, #256 // 10
ADDI Rc, Rc, #256 // 11
SUBI R2, R2, #64 // 12
BNEZ R2, LOOP2 // 13
SUBI R1, R1, #1 // 14
BNEZ R1, LOOP1 // 15
```

The 9 instructions in the inner most loop (5-13) are executed 4*256 = 1024 times. Instructions 4, 14 and 15 are executed 256 times and instructions 1, 2 and 3 are only executed once. Total = 9*1024 + 3*256 + 3 = 9987 dynamic instructions.
Part C – SIMD
Convert the given code to MIPS assembly augmented with 256-bit SIMD multimedia instructions. You can use SIMD instructions like those given on page 285 of the book or you can create your own. How many dynamic instructions does the code take now?

Solution:

```
ADDI R1, R0, #256 /* 1 */
LOOP1: ADDI R2, R0, #256 /* 2 */
LOOP2: L.8D F0, 0(Ra) /* 3 */
L.8D F8, 0(Rb) /* 4 */
ADD.8D F16, F8, F0 /* 5 */
S.8D F16, 0(Rc) /* 6 */
ADDI Ra, Ra, #32 /* 7 */
ADDI Rb, Rb, #32 /* 8 */
ADDI Rc, Rc, #32 /* 9 */
SUBI R2, R2, #8 /* 10 */
BNEZ R2, LOOP2 /* 11 */
SUBI R1, R1, #1 /* 12 */
BNEZ R1, LOOP1 /* 13 */
```

The 9 instructions in the inner most loop (3-11) are executed 32*256 = 8192 times. Instructions 2, 12 and 13 are executed 256 times and instruction 1 is only executed once. Total = 9*8192 + 3*256 + 1 = 74497 dynamic instructions.

Part D – CUDA
Implement the C code given above using CUDA. A skeleton of the kernel has been provided. Assume that there is one thread per matrix element and threads belong to 2D thread blocks which are part of a 2D grid. Note that arrays are linearized in CUDA.

```
__global__ add_matrix(float *a, float *b, float *c, int N) {

    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    int index = x + y * N;
    if (x < N && y < N)
        c[index] = a[index] + b[index];

}
```