CSI 4360/5360: CONCURRENT AND MULTI-CORE PROGRAMMING
Winter 2020, CRN 13284/ 13302, Oakland University
Homework Assignment #1 – Complex Matrix-Vector Multiplication

Due: Sunday, January 26, 2020 11:59PM
100 points, amounts to 5% of the final grade

Description

A complex matrix is defined as a matrix whose elements may contain complex numbers. Formally, a complex Matrix **M,** is defined as a combination of two real matrices : $\mathbf{M^r}$ **and** $\mathbf{M^i}$, the imaginary part of **M**. A Complex vector is simply a complex Matrix containing a single column.

Similar to real-valued Matrices, a complex matrix may be multiplied by another complex matrix/vector as long as the number of rows in the first matrix matches the number of columns in the second matrix/vector. Given an M X N complex matrix and an N X 1 complex vector, the matrix product of the two will be an M X 1 complex vector. For example, the product of a matrix-vector multiplication, M=3, N=2 is shown below. (Note that the multiplication operation between any two complex values: (a+ib)and (c+id) is calculated as : ac + i (ad+bc) +bd*i^2 or, (ac-bd) + i (ad+bc))

$$\begin{bmatrix} x_{11}+iy_{11} & x_{12}+iy_{12} \\ x_{21}+iy_{21} & x_{22}+iy_{22} \\ x_{31}+iy_{31} & x_{32}+iy_{32} \end{bmatrix} \begin{bmatrix} u_1+iv_1 \\ u_2+iv_2 \end{bmatrix} = \begin{bmatrix} R_1+iI_1 \\ R_2+iI_2 \\ R_3+iI_3 \end{bmatrix}$$

Where,

$$R_1 = x_{11}u_1 + x_{12}u_2 - y_{11}v_1 - y_{12}v_2$$
$$R_2 = x_{21}u_1 + x_{22}u_2 - y_{21}v_1 - y_{22}v_2$$
$$R_3 = x_{31}u_1 + x_{32}u_2 - y_{31}v_1 - y_{32}v_2$$
$$I_1 = x_{11}v_1 + y_{11}u_1 + x_{12}v_2 + y_{12}u_2$$
$$I_2 = x_{21}v_1 + y_{21}u_1 + x_{22}v_2 + y_{22}u_2$$
$$I_3 = x_{31}v_1 + y_{31}u_1 + x_{32}v_2 + y_{32}u_2$$

In this assignment, you will write a program that can correctly perform matrix-vector multiplication for complex numbers with floating-point real and imaginary components. There are several approaches to store complex matrix/vector values into a data structure. Two possible strategies are listed below:

1. Using a single Array:
   To implement the Matrix-Vector Multiplication operation using arrays only, the input M-by-N Matrix may be stored into an allocation of single-dimensional array of (M*N*2) size where all real values are first stored contiguously, followed by all imaginary values. That way, the real component of a complex matrix A at the i'th row and j'th column may be referenced as:

   ```
   A[ i*N +j ]
   ```

   To make a reference to the imaginary component of the same matrix element, you can simply add an (M*N) offset to the index value:

   ```
   A[ M*N + i*N + j ]
   ```

2. Use an Array-of-Structures (AoS): Alternatively, you may allocate an Array of structure types where each structure represents a single complex number by storing the real and imaginary part into two struct members:

   ```
   typedef struct complex {
     float real;
     float imaginary;
   } complex;
   ```

   That way, an M -by N complex matrix can be fit into an array of type `complex` and size M*N.

Your first step is to implement both above strategies into separate functions. The corresponding prototypes of the two functions are given below:

```
void  complex_matvecmul_simplearray(int M, int N, float *A, float *B, float *C);
void  complex_matvecmul_aos(int M, int N, complex *A, complex *B, complex *C);
```

Where,

A : M-by-N complex matrix, input
B: N-by-1 complex vector, input
C: M-by-1 complex vector, output

In the second step, you will write a `main`() function that takes the dimension sizes (M and N) as input and dynamically allocates memory to store the input matrices and vectors. Then, it will initialize the matrices and vectors with randomly generated values between 0.0 and +1.0. The matrices and vectors used as input to both functions must contain identical values (although organized differently into the memory). After that, the `main`() function will call both multiplication functions and print/measure i) the time taken by each function to compute the product vector, in milliseconds, and, ii) the performance of each function in MFLOPS.

Lastly, you will compile and build your program at one of the SECS Linux servers. Use the following flags during compilation: "`-O0 -Wall -Wextra -std=gnu99`". Upon successful build, execute your program for input values: M=N=64, M=N=128, M=N=256, M=N=512, M=N=1024, M=N=2048 and collect the performance data for each run.

Finally, write a report containing:

i) A table of data reporting the time of execution and MFLOPS performance of both multiplication functions for each input combination.
ii) 2 plots showing the trend of "time of execution" of the functions over input size; and the trend of MFLOPS performance over input size;
iii) A short description of your implementation and meaningful explanation of results (e.g. speculate why one function performs better than the other). In the report, clearly mention the name of the server that was used to run your program.

You will need to turn in your report and the source code of the complex matrix-vector multiplication program as a single archive file(.zip or .tar) and upload at the Moodle submission portal.

Grading:

Implementation of functions: 50%

Report: 50%