Math 4610 – HW 7

Jake Daniels A02307117

November 16, 2022

Task 1:

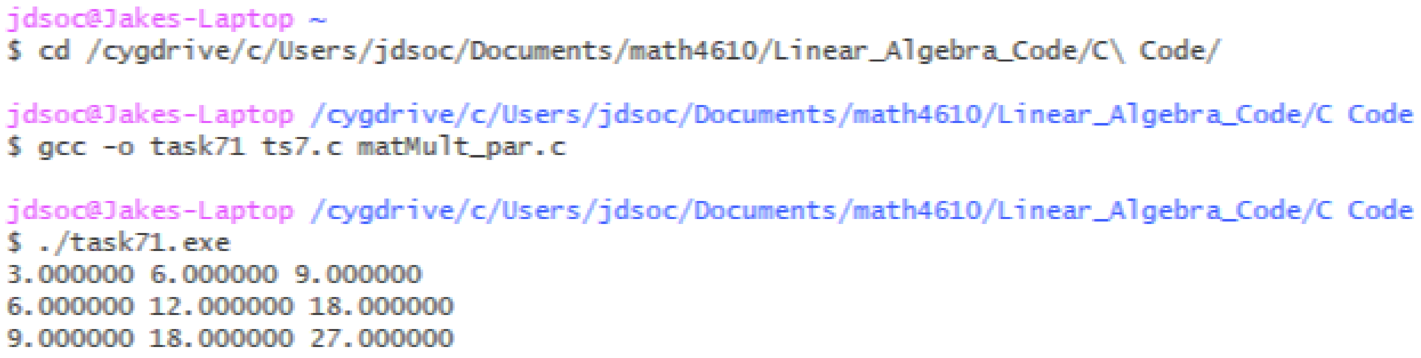
For this task I started by writing code to do matrix multiplication not in parallel:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void matMult_par();

void matMult_par(int n, int m, int k, double A[n][m], double B[m][k], double C[n][k])
{
    for (int l = 0; l < n; l++)
    {
        for (int d = 0; d < k; d++)
        {
            C[l][d] = 0;
        }
    }
        int h, i, j, nthrds, id;
        for (i = 0; i < n; i++)
            {
                for (j = 0; j < k; j++)
                {
                    for (h = 0; h < m; h++)
                    {
                        C[i][j] += A[i][h] * B[h][j];
                    }
                }
            }
}
```

I then wrote the following test code to make sure it works:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

void matMult_par(int n, int m, int k, double A[n][m], double B[m][k], double C[n][k]);

void main()
{
    double A[3][3], B[3][3], C[3][3];
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            A[i][j] = i + 1.0;
            B[i][j] = j + 1.0;
        }
    }
    matMult_par(3, 3, 3, A, B, C);
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            printf("%f ", C[i][j]);
        }
        printf("\n");
    }
}
```

Here is a screenshot of me compiling and running the above code:

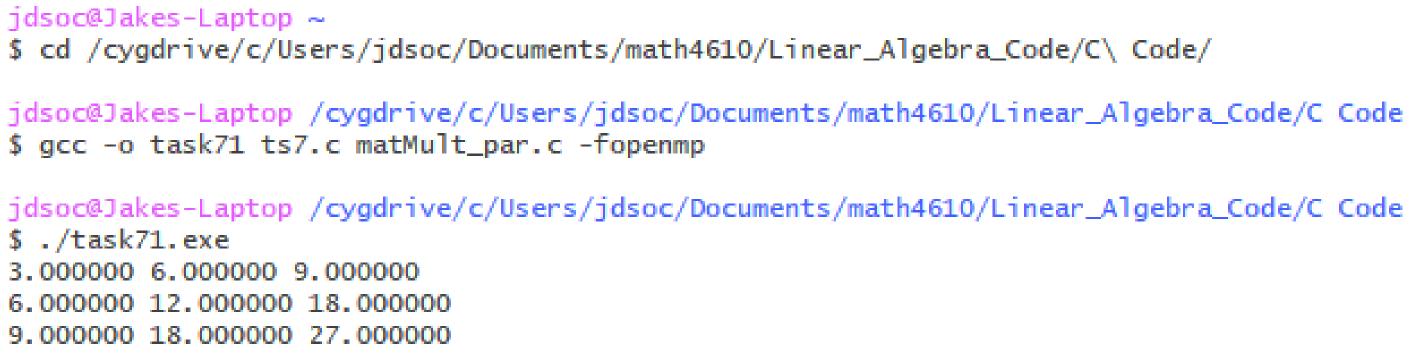


```
jdsoc@Jakes-Laptop ~
$ cd /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C\ Code/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task71 ts7.c matMult_par.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task71.exe
3.000000 6.000000 9.000000
6.000000 12.000000 18.000000
9.000000 18.000000 27.000000
```

Which the above is the correct matrix. Now, in order to change the above code to be able to run in parallel, I changed it to be the following:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
#define NUM_THREADS 8

void matMult_par();

void matMult_par(int n, int m, int k, double A[n][m], double B[m][k], double C[n][k])
{
    for (int l = 0; l < n; l++)
    {
        for (int d = 0; d < k; d++)
        {
            C[l][d] = 0;
        }
    }
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
    {
        int i, j, h, id, nthrds;
        id = omp_get_thread_num();
        nthrds = omp_get_num_threads();
        for (i = id; i < n; i += nthrds)
        {
            for (j = 0; j < k; j++)
            {
                for (h = 0; h < m; h++)
                {
                    C[i][j] += A[i][h] * B[h][j];
                }
            }
        }
    }
}
```

Which when compiled and ran with the same test code above, I get the following:



```
jdsoc@Jakes-Laptop ~
$ cd /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C\ Code/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task71 ts7.c matMult_par.c -fopenmp

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task71.exe
3.000000 6.000000 9.000000
6.000000 12.000000 18.000000
9.000000 18.000000 27.000000
```

Which after hours and hours of it not running correctly finally is working as it is supposed to and outputting the correct matrix.

Task 2:

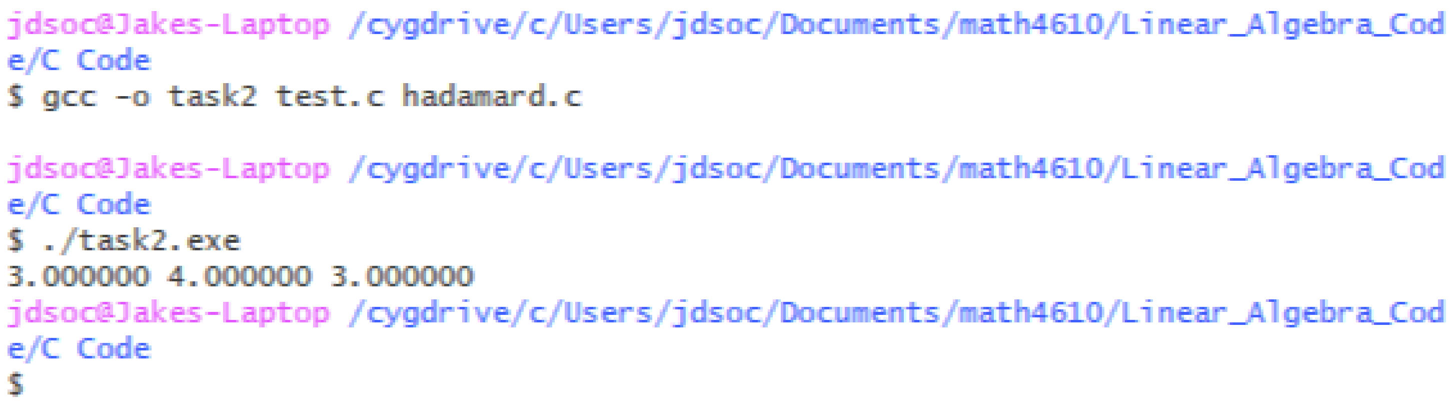
For this task I wrote the following code in C for the Hadamard product of two vectors of the same size:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void hadamard();

void hadamard(int n, double A[n], double b[n], double C[n])
{
    int i;
    for (i = 0; i < n; i++)
    {
        C[i] = A[i] * b[i];
```

I then wrote the following code to test my code above:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void hadamard(int n, double A[n], double b[n], double C[n]);

void main()
{
    double A[3] = {1, 2, 3};
    double b[3] = {3, 2, 1};
    double C[3];
    hadamard(3, A, b, C);
    for (int i = 0; i < 3; i++)
    {
        printf("%f ", C[i]);
    }
}
```

Here is a screenshot of me compiling and running the above code, along with the correct output for the above vectors:



```
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Cod
e/C Code
$ gcc -o task2 test.c hadamard.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Cod
e/C Code
$ ./task2.exe
3.000000 4.000000 3.000000
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Cod
e/C Code
$
```

Task 3:

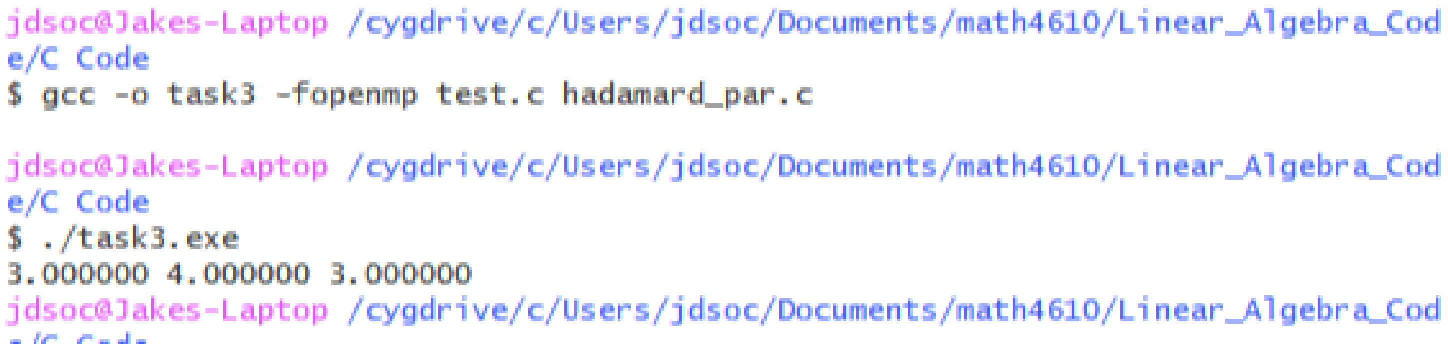
For this task I wrote a new code changing the one above to run in parallel:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
#define NUM_THREADS 8

void hadamard_par();

void hadamard_par(int n, double A[n], double b[n], double C[n])
{
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
    {
        int i, id, nthreads;
        id = omp_get_thread_num();
        nthreads = omp_get_num_threads();
        for (i = id; i < n; i = i + nthreads)
        {
            C[i] = A[i] * b[i];
        }
    }
}
```

I wrote the following code to test it, which is basically the same as the test code for Task 2:

```c
void main()
{
    double A[3] = {1, 2, 3};
    double b[3] = {3, 2, 1};
    double C[3];
    hadamard_par(3, A, b, C);
    for (int i = 0; i < 3; i++)
    {
        printf("%f ", C[i]);
    }
}
```

Here is a screenshot of me and compiling and running the code and getting the correct output:



```
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Cod
e/C Code
$ gcc -o task3 -fopenmp test.c hadamard_par.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Cod
e/C Code
$ ./task3.exe
3.000000 4.000000 3.000000
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Cod
```

Task 4:

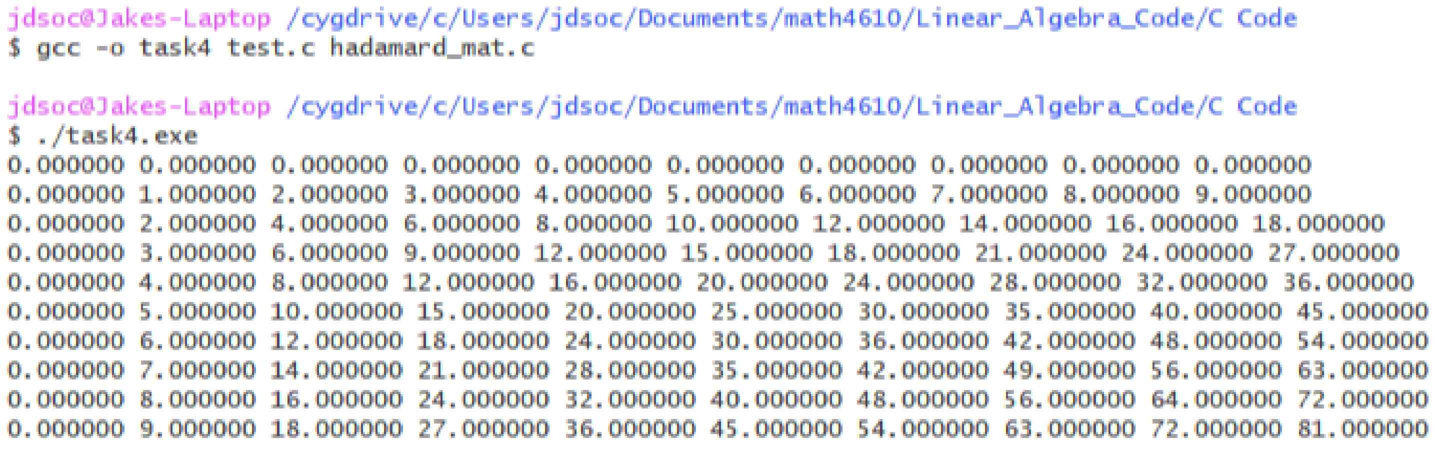
For this code I wrote code to calculate the Hadamard product in general for matrices:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void hadamard_mat();

void hadamard_mat(int n, int m, double A[n][m], double B[n][m], double C[n][m])
{
    int i, j;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            C[i][j] = A[i][j] * B[i][j];
        }
    }
}
```

I then wrote the following code to test it, I chose two 10x10 matrices to show it works but it can obviously be extended higher, however the larger it gets the harder it is to fit it nicely in this write-up:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void hadamard_mat(int n, int m, double A[n][m], double B[n][m], double C[n][m]);

void main()
{
    double A[10][10], B[10][10], C[10][10];
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            A[i][j] = i;
            B[i][j] = j;
        }
    }
    hadamard_mat(10, 10, A, B, C);
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            printf("%f ", C[i][j]);
        }
        printf("\n");
    }
}
```

Here is the screenshot of me compiling the above code and running it:



```
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task4 test.c hadamard_mat.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task4.exe
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 14.000000 16.000000 18.000000
0.000000 3.000000 6.000000 9.000000 12.000000 15.000000 18.000000 21.000000 24.000000 27.000000
0.000000 4.000000 8.000000 12.000000 16.000000 20.000000 24.000000 28.000000 32.000000 36.000000
0.000000 5.000000 10.000000 15.000000 20.000000 25.000000 30.000000 35.000000 40.000000 45.000000
0.000000 6.000000 12.000000 18.000000 24.000000 30.000000 36.000000 42.000000 48.000000 54.000000
0.000000 7.000000 14.000000 21.000000 28.000000 35.000000 42.000000 49.000000 56.000000 63.000000
0.000000 8.000000 16.000000 24.000000 32.000000 40.000000 48.000000 56.000000 64.000000 72.000000
0.000000 9.000000 18.000000 27.000000 36.000000 45.000000 54.000000 63.000000 72.000000 81.000000
```

Task 5:

For this section I first did some research for what we define to be an outer (wedge) product. It is defined as the following:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

Tensor product of $\mathbf{v}$ and $\mathbf{w}$ is given by

$$\mathbf{v} \otimes \mathbf{w} = \begin{bmatrix} v_1 w_1 & v_1 w_2 & \cdots & v_1 w_m \\ v_2 w_1 & v_2 w_2 & \cdots & v_2 w_m \\ \vdots & \vdots & \ddots & \vdots \\ v_n w_1 & v_n w_2 & \cdots & v_n w_m \end{bmatrix}$$

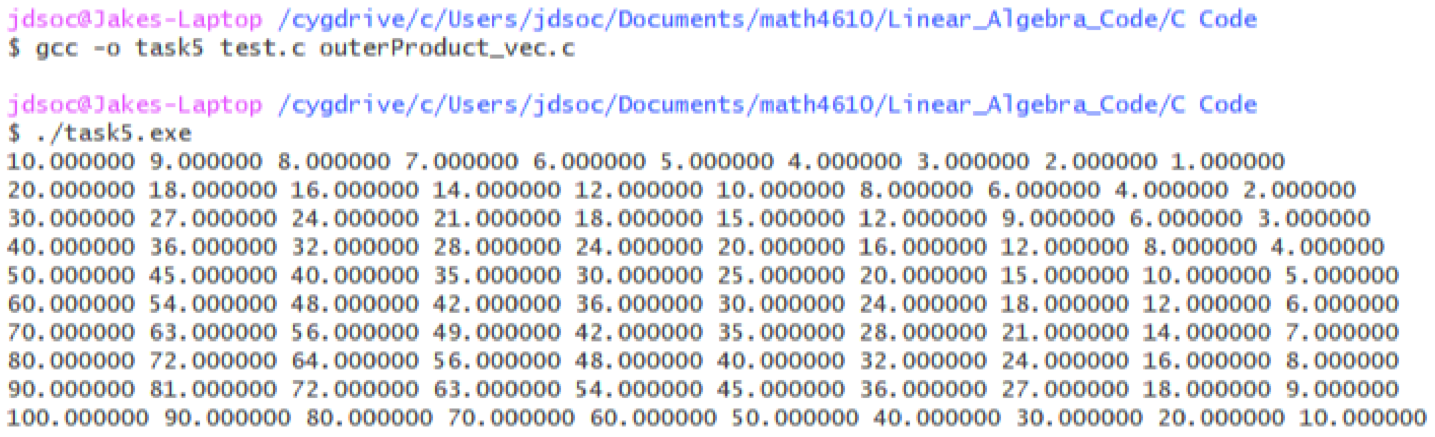
Meaning we get a matrix where $A_{nm} = v_n w_m$. With this knowledge I wrote the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void outerProduct_vec();

void outerProduct_vec(int n, int m, double A[n], double B[m], double C[n][m])
{
    int i, j;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            C[i][j] = A[i] * B[j];
        }
    }
}
```

I then wrote the following test code:

```
void main()
{
    double A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    double B[10] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    double C[10][10];
    outerProduct_vec(10, 10, A, B, C);
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            printf("%f ", C[i][j]);
        }
        printf("\n");
    }
}
```

Here is the screenshot of me compiling and running my code:



```
jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ gcc -o task5 test.c outerProduct_vec.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/Documents/math4610/Linear_Algebra_Code/C Code
$ ./task5.exe
10.000000 9.000000 8.000000 7.000000 6.000000 5.000000 4.000000 3.000000 2.000000 1.000000
20.000000 18.000000 16.000000 14.000000 12.000000 10.000000 8.000000 6.000000 4.000000 2.000000
30.000000 27.000000 24.000000 21.000000 18.000000 15.000000 12.000000 9.000000 6.000000 3.000000
40.000000 36.000000 32.000000 28.000000 24.000000 20.000000 16.000000 12.000000 8.000000 4.000000
50.000000 45.000000 40.000000 35.000000 30.000000 25.000000 20.000000 15.000000 10.000000 5.000000
60.000000 54.000000 48.000000 42.000000 36.000000 30.000000 24.000000 18.000000 12.000000 6.000000
70.000000 63.000000 56.000000 49.000000 42.000000 35.000000 28.000000 21.000000 14.000000 7.000000
80.000000 72.000000 64.000000 56.000000 48.000000 40.000000 32.000000 24.000000 16.000000 8.000000
90.000000 81.000000 72.000000 63.000000 54.000000 45.000000 36.000000 27.000000 18.000000 9.000000
100.000000 90.000000 80.000000 70.000000 60.000000 50.000000 40.000000 30.000000 20.000000 10.000000
```

Yes it can be extended to the matrices and is called the Kronecker product. There are no restrictions on the sizes of the matrices. Also you could extend this method to be in parallel by running the inside loop starting at the thread number and adding the amount of threads each iteration.