Array Sum

	Sequential	1 thread	2 threads	5 threads	7 threads	9 threads
Time	1.066					
Pthread		1.354	0.997	0.982	0.873	0.835
OpenMP		0.704	0.581	0.500	0.536	0.507

Matrix Multiplication

	Sequential	1 thread	2 threads	5 threads	7 threads	9 threads
Time	5.181					
Pthread		2.606	1.713	0.952	0.769	0.801
OpenMP		4.735	3.640	2.269	2.094	2.145

Trapezoidal Rule Integration

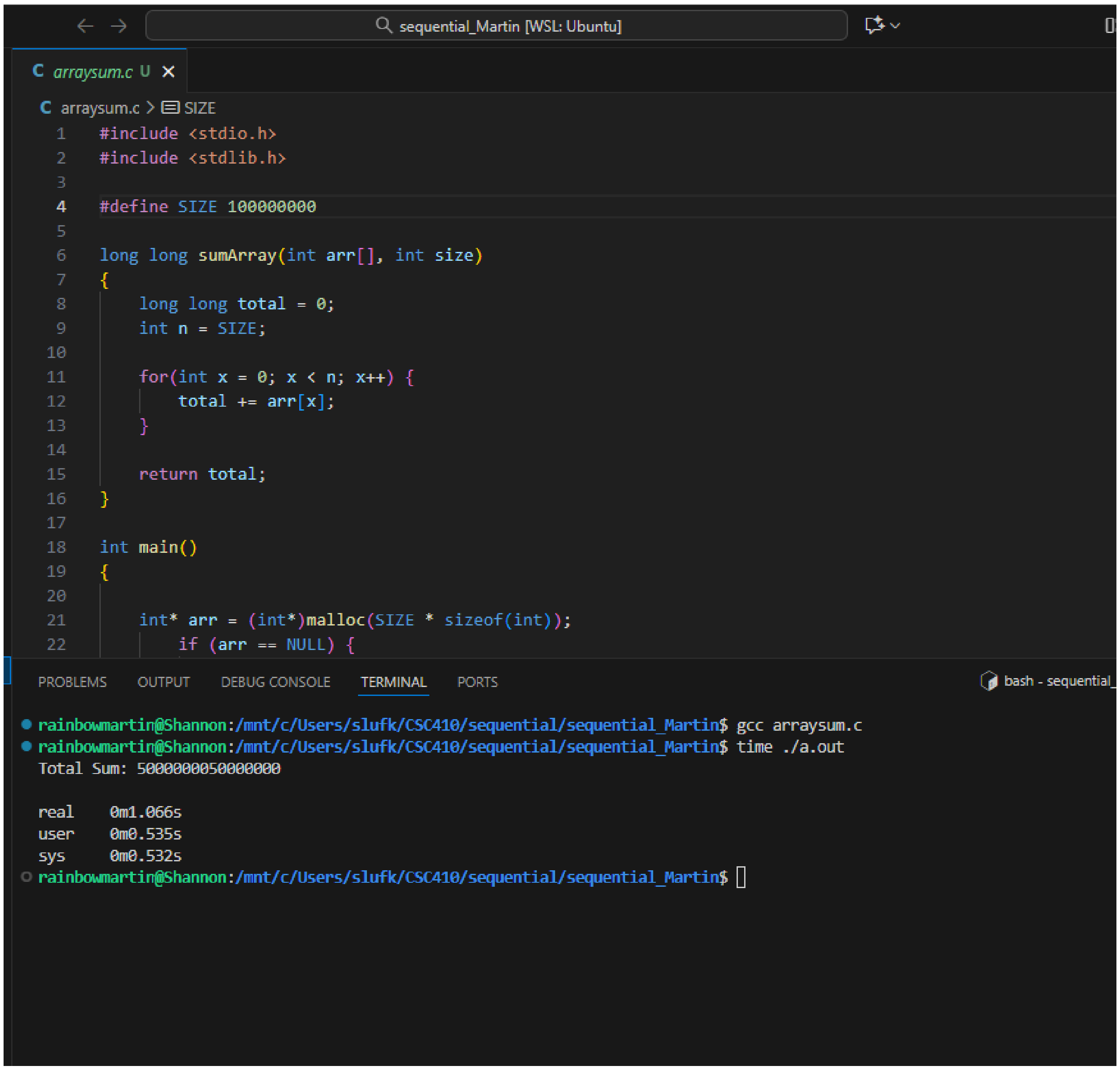
	Sequential	1 thread	2 threads	5 threads	7 threads	9 threads
Time	2.781					
Pthread		2.534	1.654	0.960	0.792	0.828
OpenMP		2.726	1.823	1.057	0.922	0.945

N Queens

	Sequential	1 thread	2 threads	5 threads	7 threads	9 threads
Time	2m15.690					
Pthread		48.580	31.661	15.069	13.172	13.312
OpenMP		50.524	31.670	19.314	15.970	13.060

Array Sum

Sequential



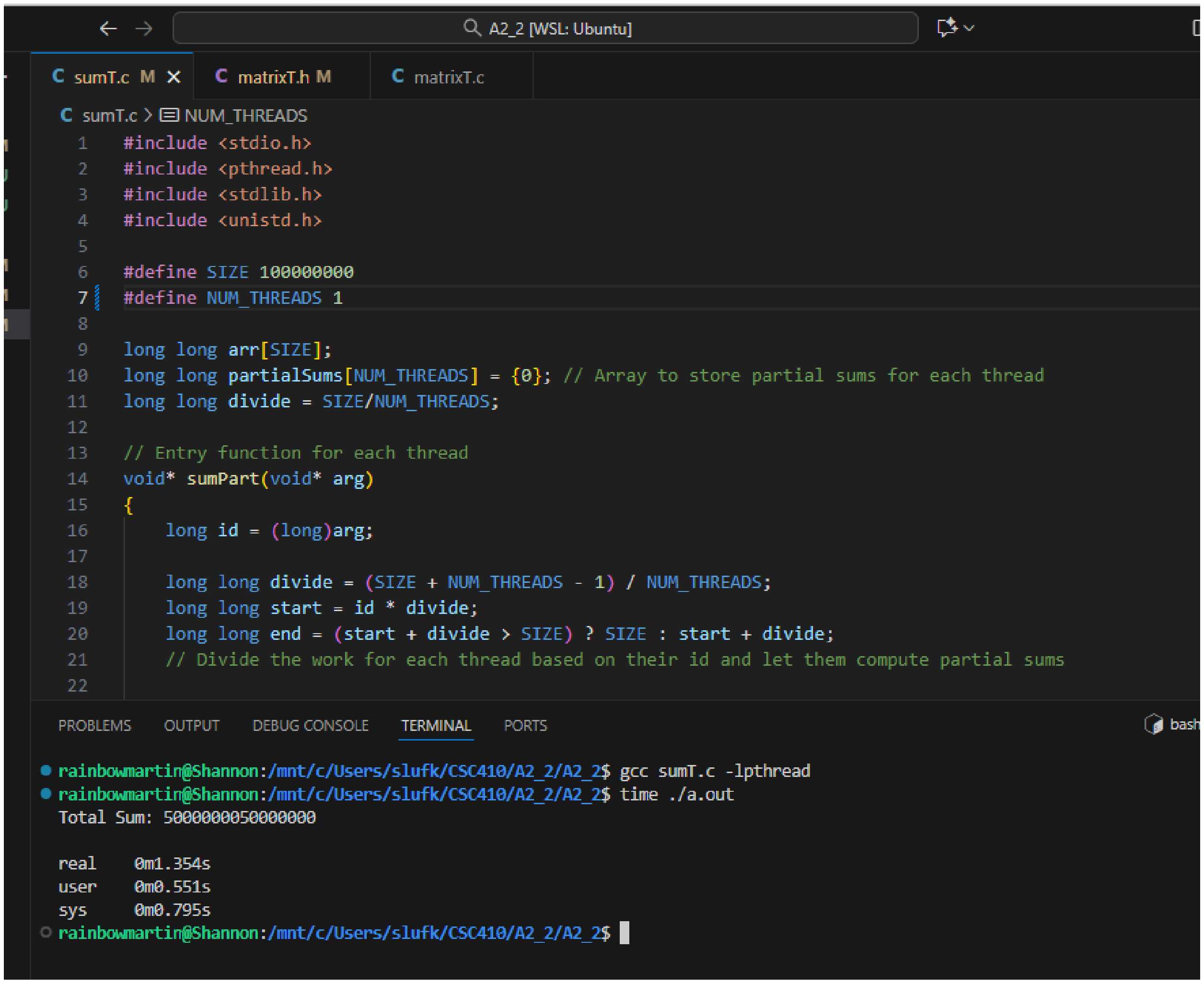
sequential_Martin [WSL: Ubuntu]

arraysum.c > SIZE

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

#define SIZE 100000000

long long sumArray(int arr[], int size)
{
    long long total = 0;
    int n = SIZE;

    for(int x = 0; x < n; x++) {
        total += arr[x];
    }

    return total;
}

int main()
{

    int* arr = (int*)malloc(SIZE * sizeof(int));
        if (arr == NULL) {
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/sequential/sequential_Martin$ gcc arraysum.c
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/sequential/sequential_Martin$ time ./a.out
Total Sum: 5000000050000000

real    0m1.066s
user    0m0.535s
sys     0m0.532s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/sequential/sequential_Martin$
```

Pthread – 1 thread



A2_2 [WSL: Ubuntu]

sumT.c M × | matrixT.h M | matrixT.c

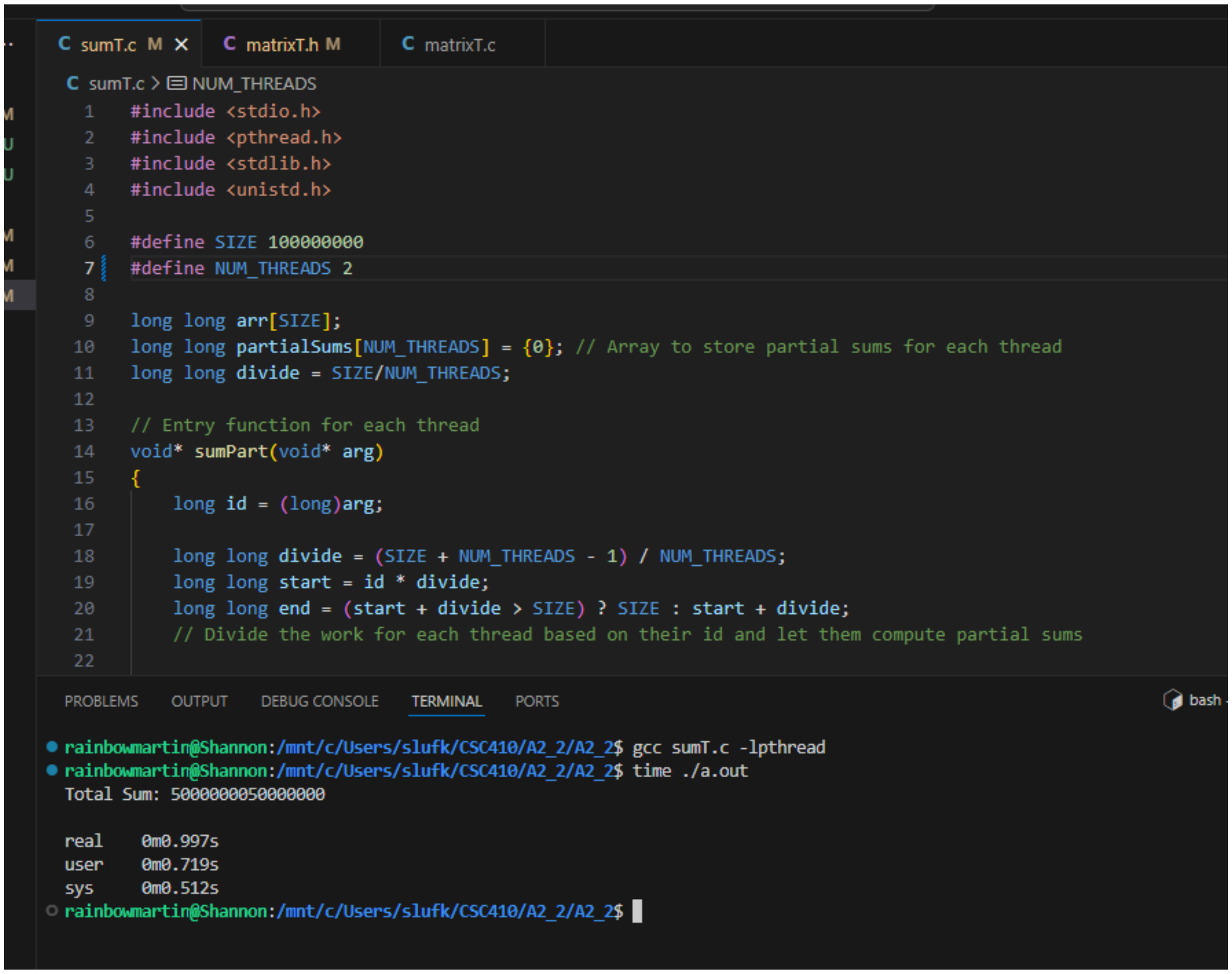
sumT.c > NUM_THREADS

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

#define SIZE 100000000
#define NUM_THREADS 1

long long arr[SIZE];
long long partialSums[NUM_THREADS] = {0}; // Array to store partial sums for each thread
long long divide = SIZE/NUM_THREADS;

// Entry function for each thread
void* sumPart(void* arg)
{
    long id = (long)arg;

    long long divide = (SIZE + NUM_THREADS - 1) / NUM_THREADS;
    long long start = id * divide;
    long long end = (start + divide > SIZE) ? SIZE : start + divide;
    // Divide the work for each thread based on their id and let them compute partial sums

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ gcc sumT.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ time ./a.out
Total Sum: 5000000050000000

real    0m1.354s
user    0m0.551s
sys     0m0.795s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$
```

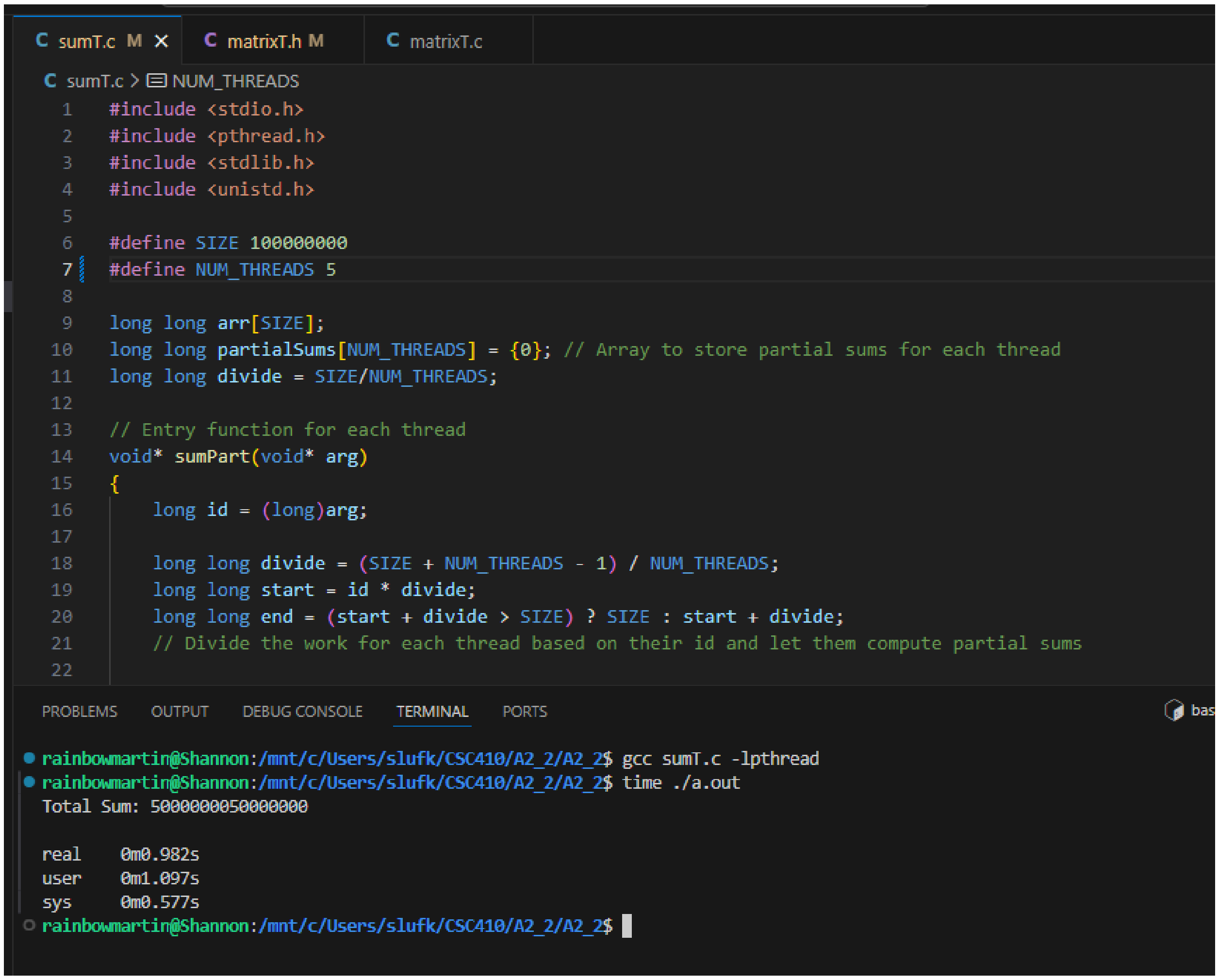
Pthread – 2 threads



```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

#define SIZE 100000000
#define NUM_THREADS 2

long long arr[SIZE];
long long partialSums[NUM_THREADS] = {0}; // Array to store partial sums for each thread
long long divide = SIZE/NUM_THREADS;

// Entry function for each thread
void* sumPart(void* arg)
{
    long id = (long)arg;

    long long divide = (SIZE + NUM_THREADS - 1) / NUM_THREADS;
    long long start = id * divide;
    long long end = (start + divide > SIZE) ? SIZE : start + divide;
    // Divide the work for each thread based on their id and let them compute partial sums

```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ gcc sumT.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ time ./a.out
Total Sum: 5000000050000000

real    0m0.997s
user    0m0.719s
sys     0m0.512s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$
```

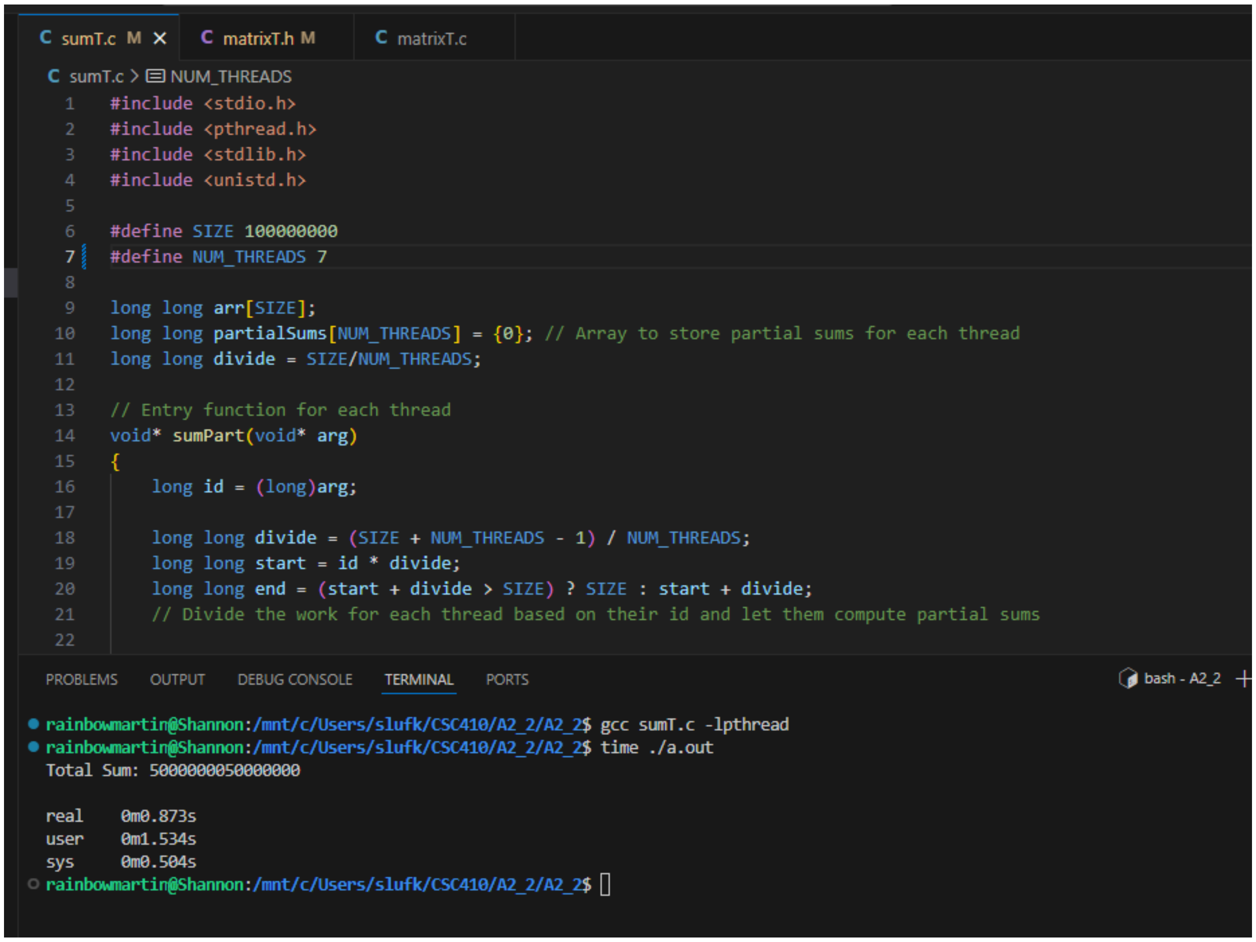
Pthread – 5 threads



```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

#define SIZE 100000000
#define NUM_THREADS 5

long long arr[SIZE];
long long partialSums[NUM_THREADS] = {0}; // Array to store partial sums for each thread
long long divide = SIZE/NUM_THREADS;

// Entry function for each thread
void* sumPart(void* arg)
{
    long id = (long)arg;

    long long divide = (SIZE + NUM_THREADS - 1) / NUM_THREADS;
    long long start = id * divide;
    long long end = (start + divide > SIZE) ? SIZE : start + divide;
    // Divide the work for each thread based on their id and let them compute partial sums

```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ gcc sumT.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ time ./a.out
Total Sum: 5000000050000000

real    0m0.982s
user    0m1.097s
sys     0m0.577s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$
```

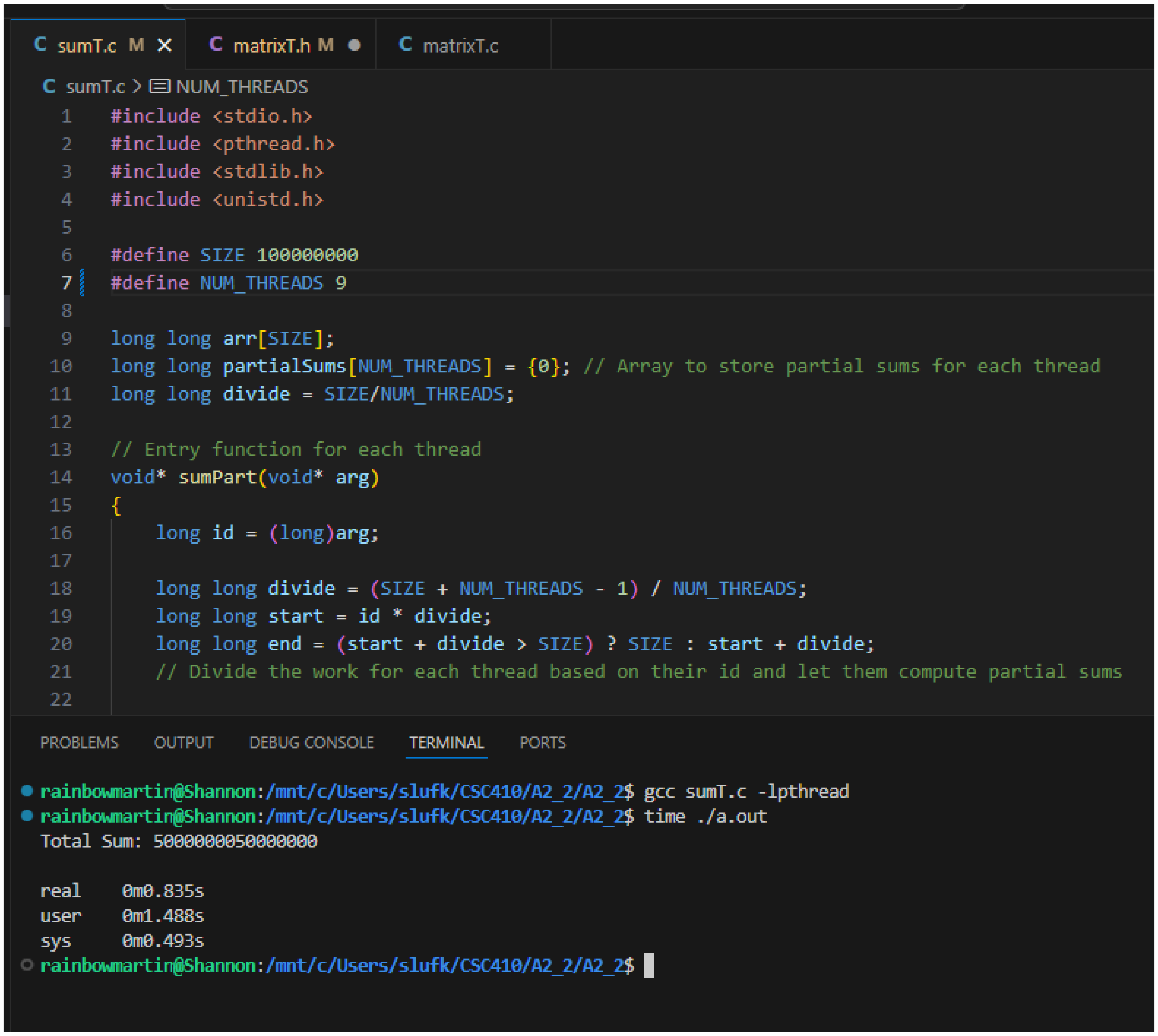
Pthread – 7 threads



```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

#define SIZE 100000000
#define NUM_THREADS 7

long long arr[SIZE];
long long partialSums[NUM_THREADS] = {0}; // Array to store partial sums for each thread
long long divide = SIZE/NUM_THREADS;

// Entry function for each thread
void* sumPart(void* arg)
{
    long id = (long)arg;

    long long divide = (SIZE + NUM_THREADS - 1) / NUM_THREADS;
    long long start = id * divide;
    long long end = (start + divide > SIZE) ? SIZE : start + divide;
    // Divide the work for each thread based on their id and let them compute partial sums

```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ gcc sumT.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ time ./a.out
Total Sum: 5000000050000000

real    0m0.873s
user    0m1.534s
sys     0m0.504s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$
```

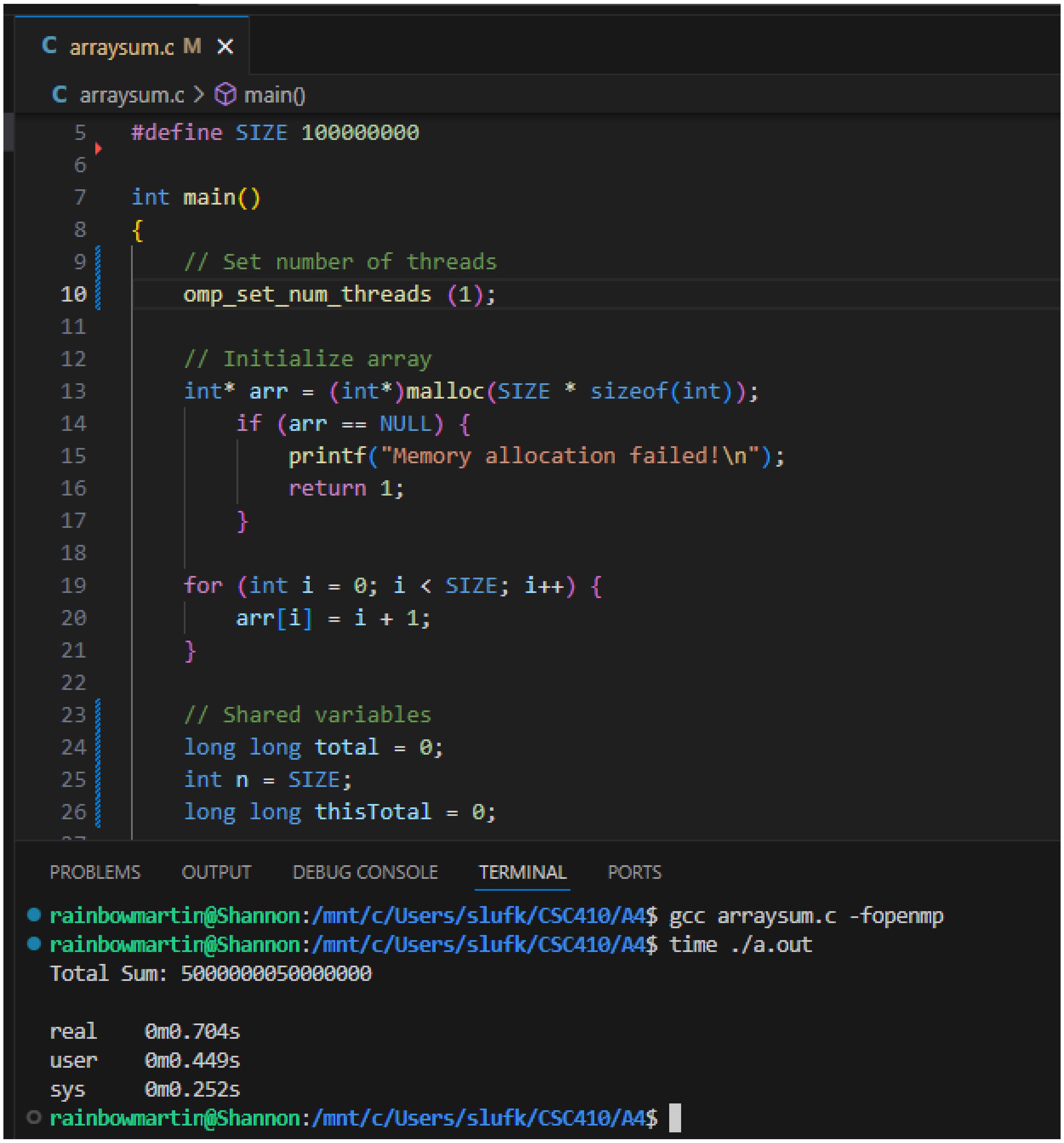
Pthread – 9 threads



```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

#define SIZE 100000000
#define NUM_THREADS 9

long long arr[SIZE];
long long partialSums[NUM_THREADS] = {0}; // Array to store partial sums for each thread
long long divide = SIZE/NUM_THREADS;

// Entry function for each thread
void* sumPart(void* arg)
{
    long id = (long)arg;

    long long divide = (SIZE + NUM_THREADS - 1) / NUM_THREADS;
    long long start = id * divide;
    long long end = (start + divide > SIZE) ? SIZE : start + divide;
    // Divide the work for each thread based on their id and let them compute partial sums

```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ gcc sumT.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ time ./a.out
Total Sum: 5000000050000000

real    0m0.835s
user    0m1.488s
sys     0m0.493s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$
```

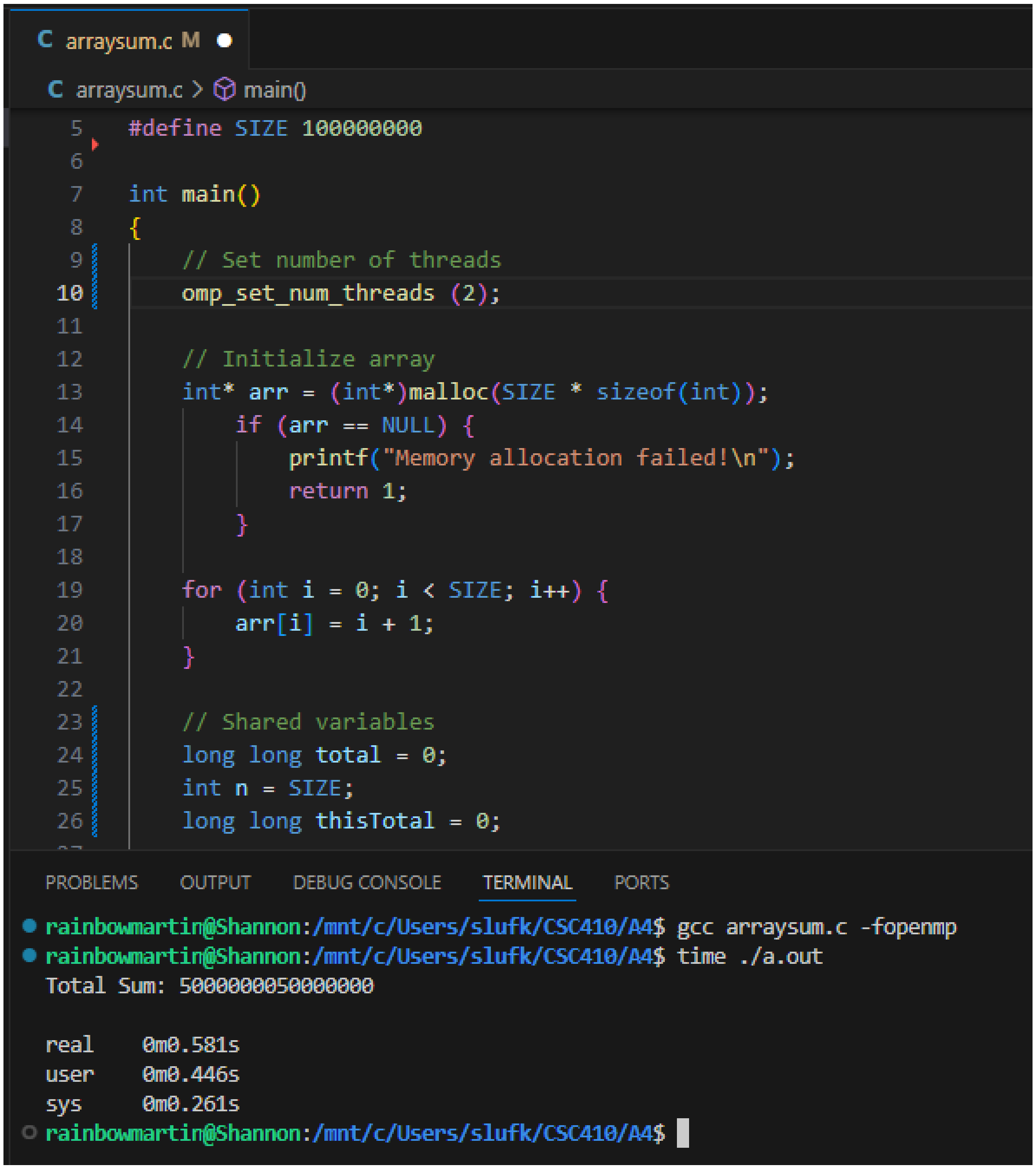
Omp – 1 thread



```c
#define SIZE 100000000

int main()
{
    // Set number of threads
    omp_set_num_threads (1);

    // Initialize array
    int* arr = (int*)malloc(SIZE * sizeof(int));
        if (arr == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }

    // Shared variables
    long long total = 0;
    int n = SIZE;
    long long thisTotal = 0;
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc arraysum.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Total Sum: 5000000050000000

real    0m0.704s
user    0m0.449s
sys     0m0.252s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$
```

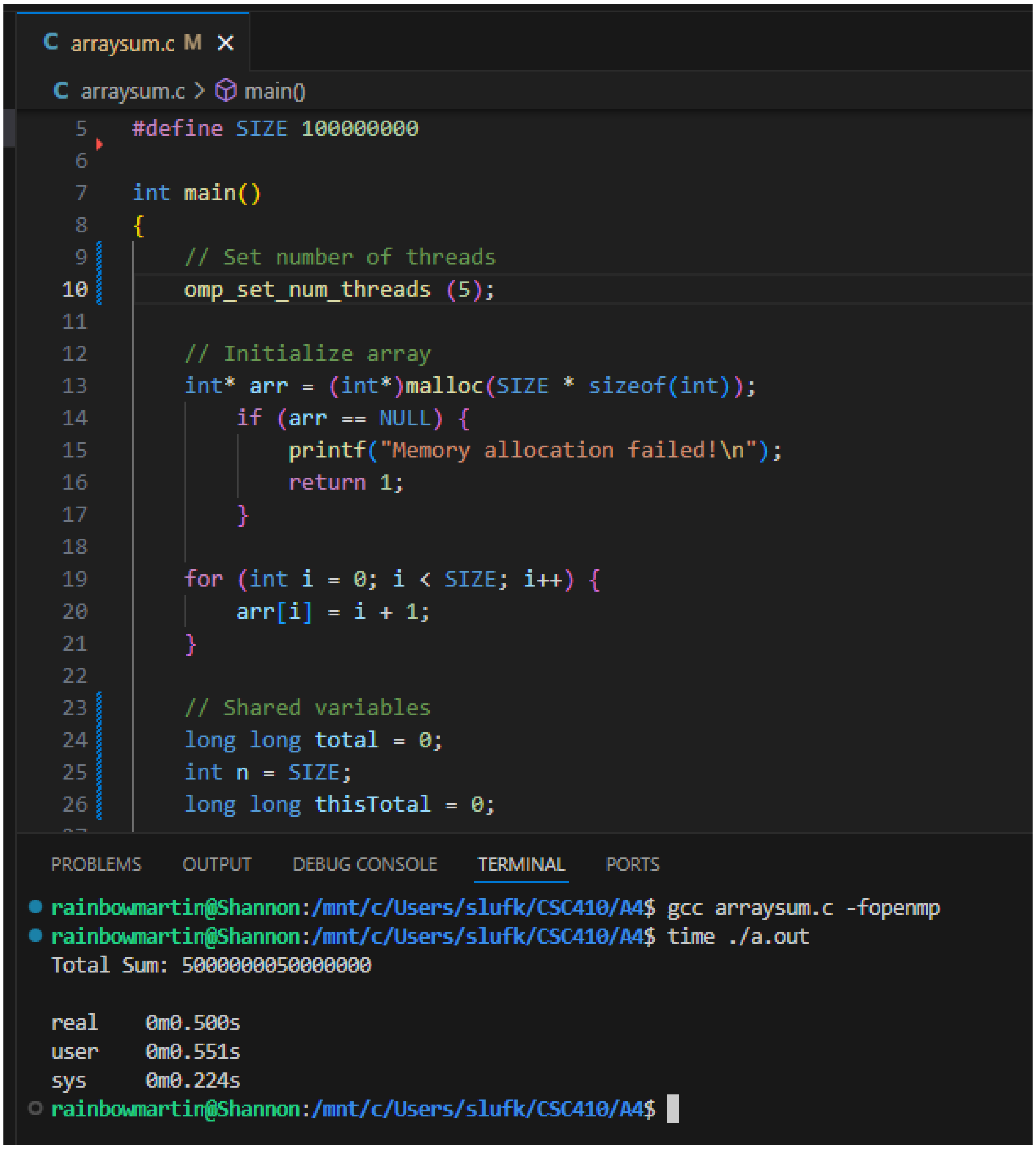
Omp – 2 threads



```c
#define SIZE 100000000

int main()
{
    // Set number of threads
    omp_set_num_threads (2);

    // Initialize array
    int* arr = (int*)malloc(SIZE * sizeof(int));
        if (arr == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }

    // Shared variables
    long long total = 0;
    int n = SIZE;
    long long thisTotal = 0;
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc arraysum.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Total Sum: 5000000050000000

real    0m0.581s
user    0m0.446s
sys     0m0.261s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$
```

Omp – 5 threads



arraysum.c M ×

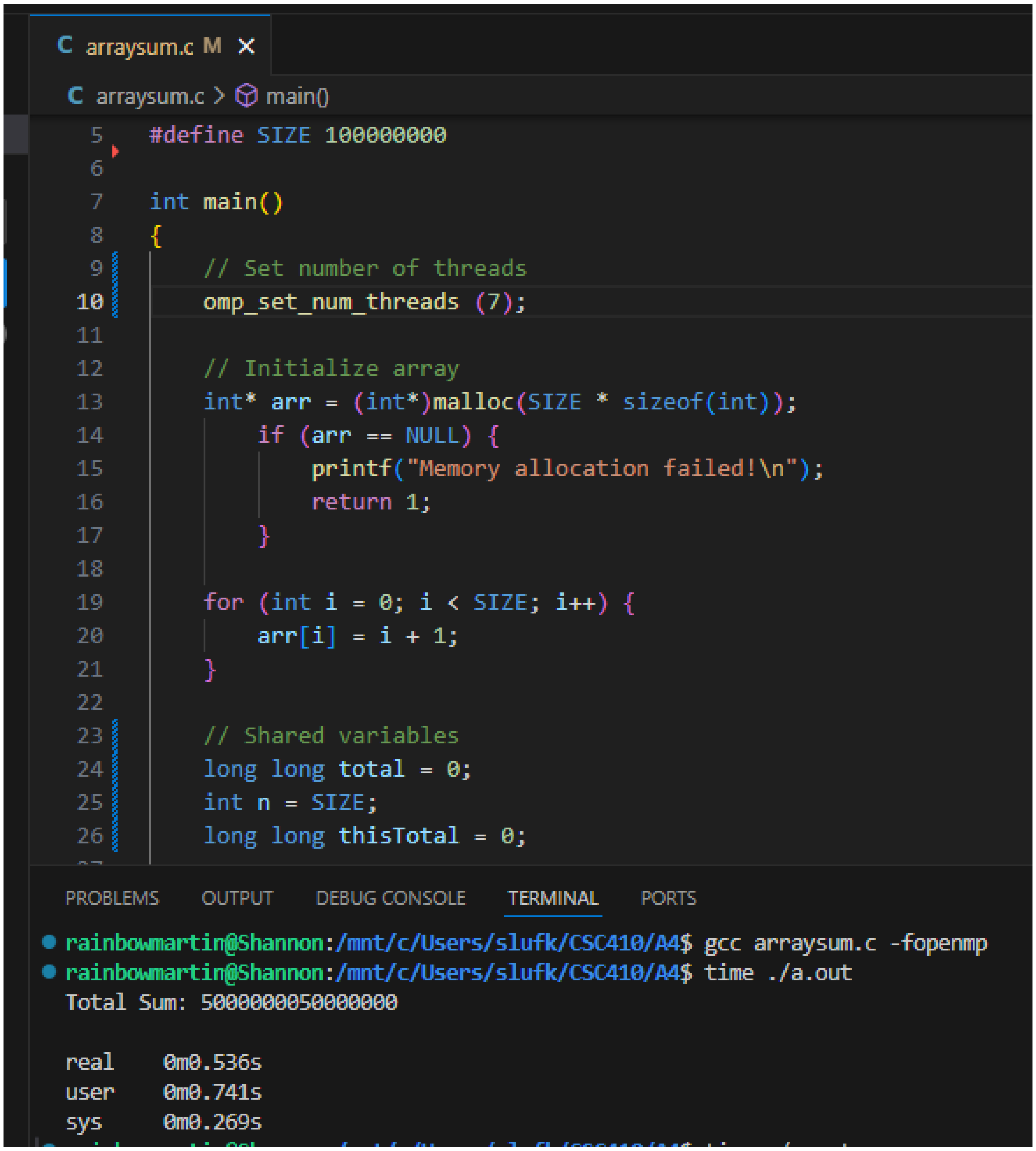
arraysum.c > main()

```c
#define SIZE 100000000

int main()
{
    // Set number of threads
    omp_set_num_threads (5);

    // Initialize array
    int* arr = (int*)malloc(SIZE * sizeof(int));
        if (arr == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }

    // Shared variables
    long long total = 0;
    int n = SIZE;
    long long thisTotal = 0;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc arraysum.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Total Sum: 5000000050000000

real    0m0.500s
user    0m0.551s
sys     0m0.224s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$
```

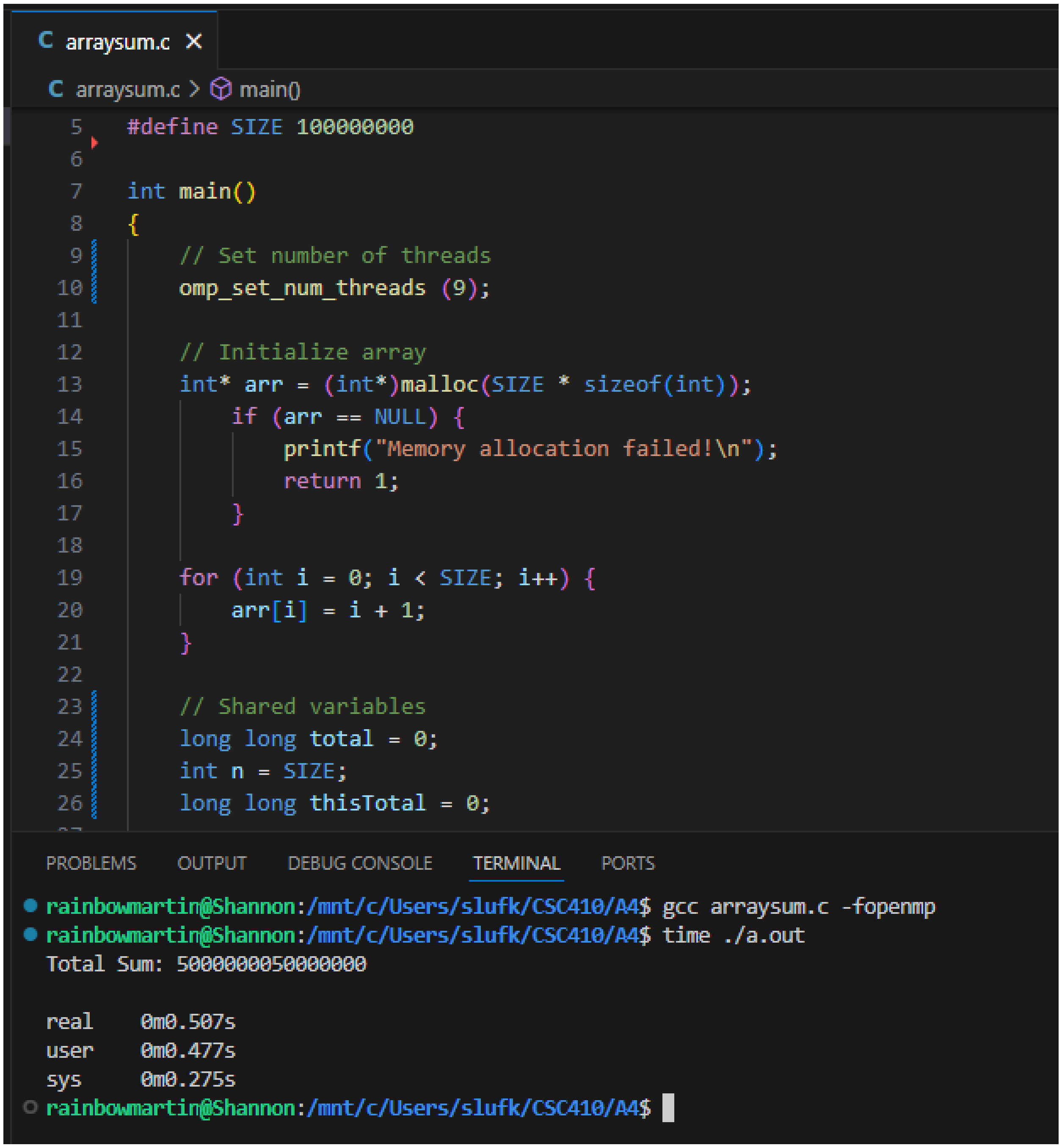
Omp – 7 threads



```c
#define SIZE 100000000

int main()
{
    // Set number of threads
    omp_set_num_threads (7);

    // Initialize array
    int* arr = (int*)malloc(SIZE * sizeof(int));
        if (arr == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }

    // Shared variables
    long long total = 0;
    int n = SIZE;
    long long thisTotal = 0;
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc arraysum.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Total Sum: 5000000050000000

real    0m0.536s
user    0m0.741s
sys     0m0.269s
```

Omp – 9 threads

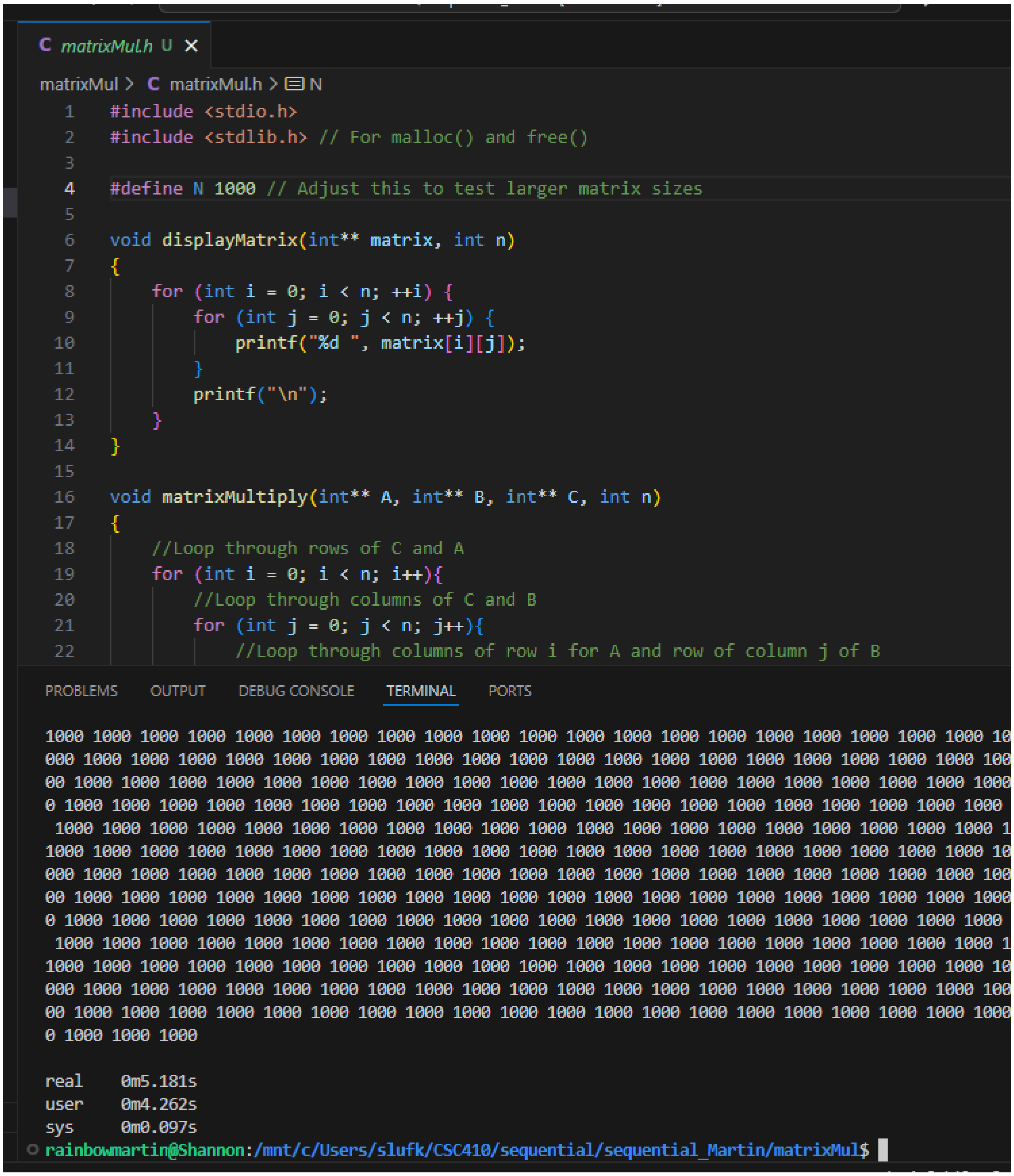


```c
#define SIZE 100000000

int main()
{
    // Set number of threads
    omp_set_num_threads (9);

    // Initialize array
    int* arr = (int*)malloc(SIZE * sizeof(int));
        if (arr == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }

    // Shared variables
    long long total = 0;
    int n = SIZE;
    long long thisTotal = 0;
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc arraysum.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Total Sum: 5000000050000000

real    0m0.507s
user    0m0.477s
sys     0m0.275s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$
```

Matrix Multiplication

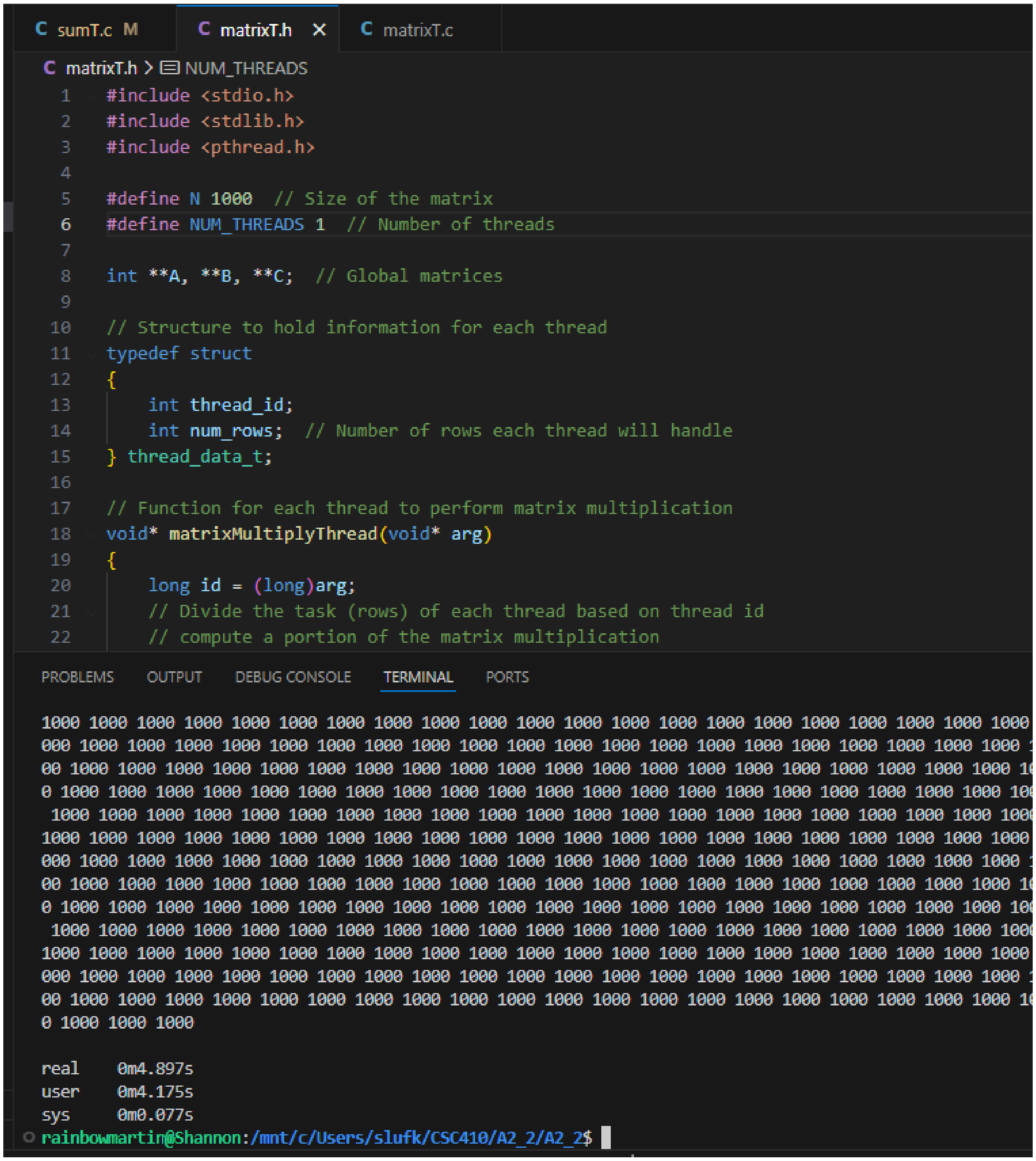
Sequential



```c
#include <stdio.h>
#include <stdlib.h> // For malloc() and free()

#define N 1000 // Adjust this to test larger matrix sizes

void displayMatrix(int** matrix, int n)
{
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

void matrixMultiply(int** A, int** B, int** C, int n)
{
    //Loop through rows of C and A
    for (int i = 0; i < n; i++){
        //Loop through columns of C and B
        for (int j = 0; j < n; j++){
            //Loop through columns of row i for A and row of column j of B
```

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000

real    0m5.181s
user    0m4.262s
sys     0m0.097s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/sequential/sequential_Martin/matrixMul$ 
```

Pthread – 1 thread



sumT.c M | matrixT.h | matrixT.c

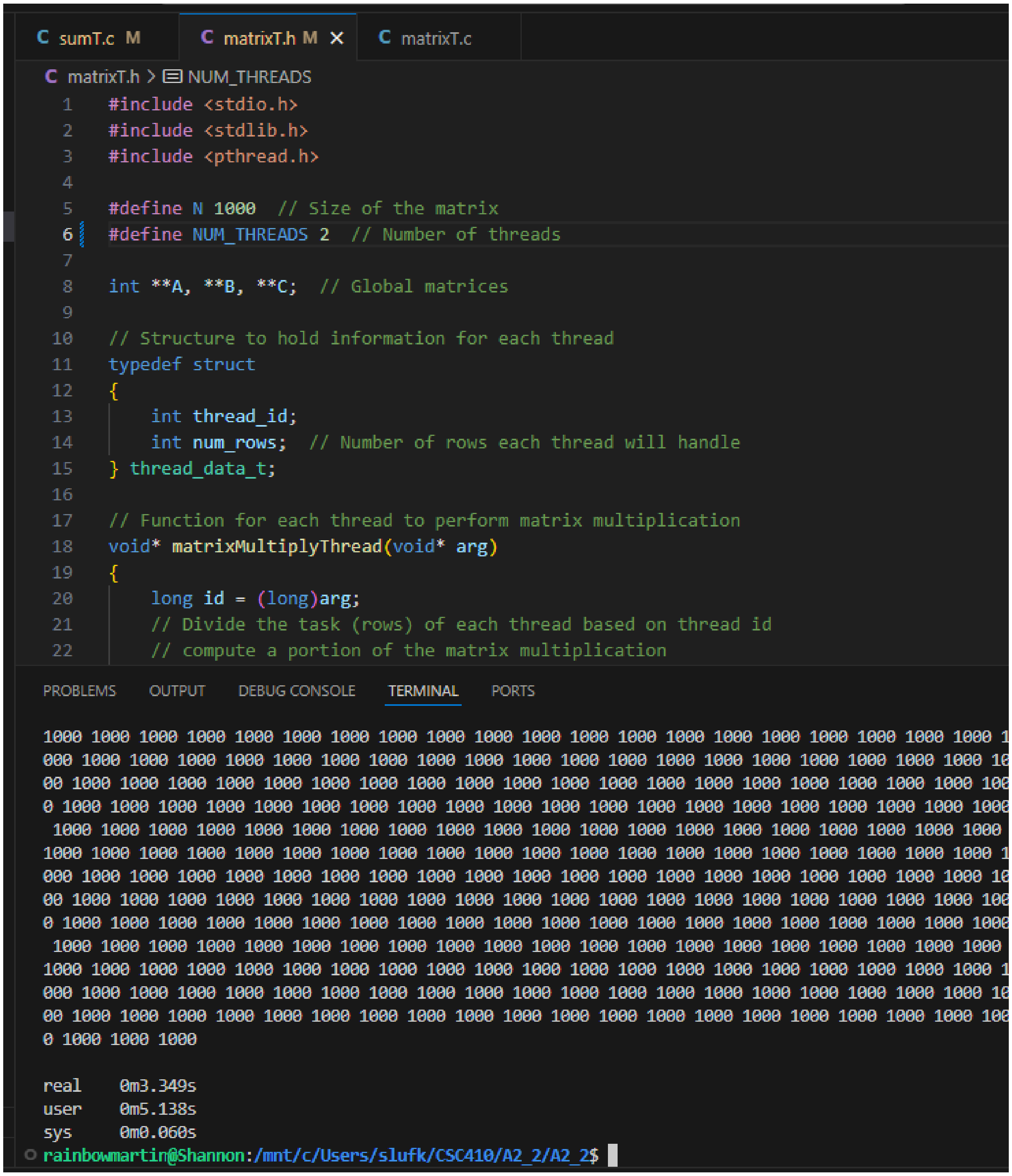
matrixT.h > NUM_THREADS

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

#define N 1000  // Size of the matrix
#define NUM_THREADS 1  // Number of threads

int **A, **B, **C;  // Global matrices

// Structure to hold information for each thread
typedef struct
{
    int thread_id;
    int num_rows;  // Number of rows each thread will handle
} thread_data_t;

// Function for each thread to perform matrix multiplication
void* matrixMultiplyThread(void* arg)
{
    long id = (long)arg;
    // Divide the task (rows) of each thread based on thread id
    // compute a portion of the matrix multiplication
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
0 1000 1000 1000

real    0m4.897s
user    0m4.175s
sys     0m0.077s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$
```

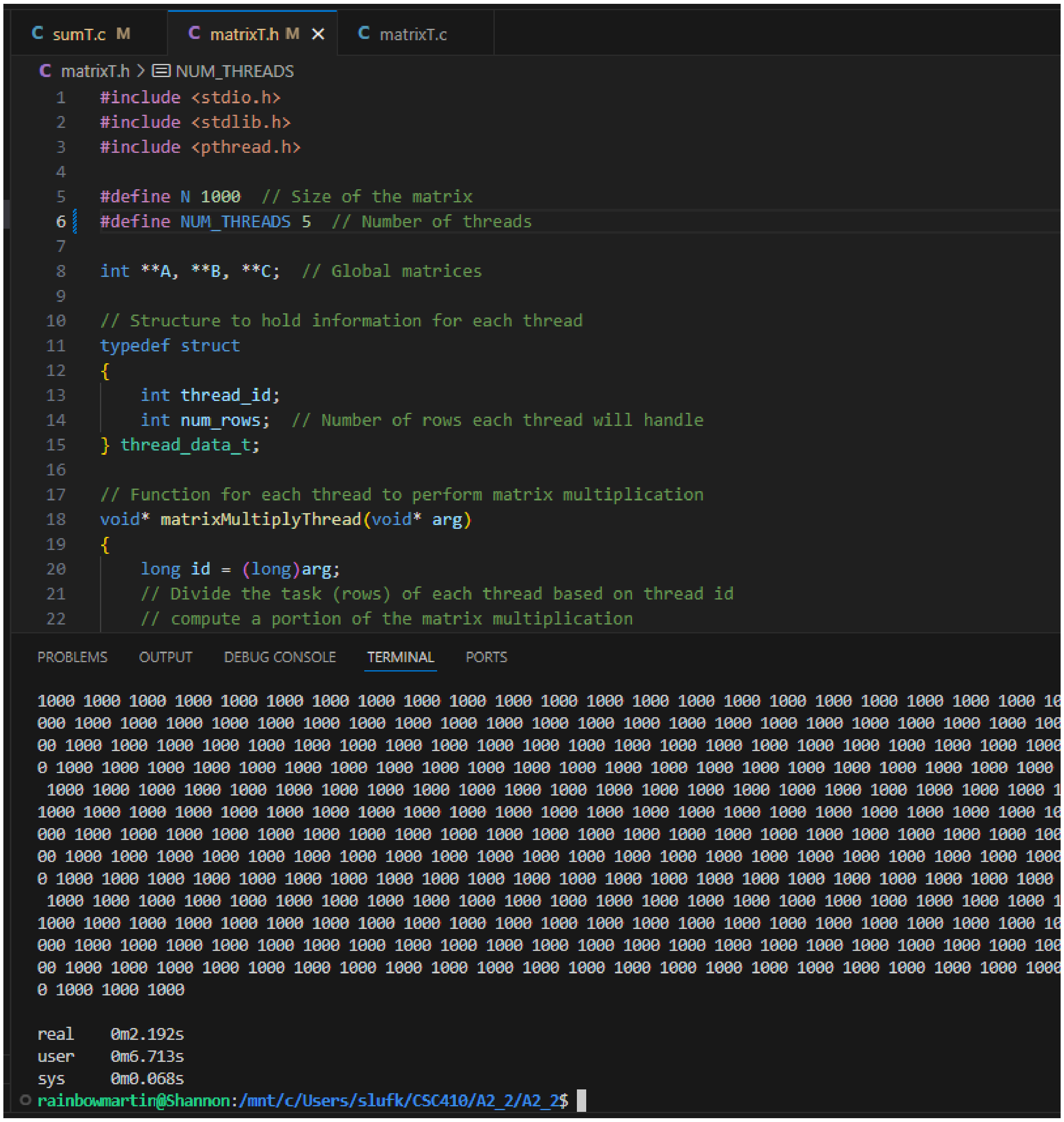
Pthread – 2 threads



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

#define N 1000  // Size of the matrix
#define NUM_THREADS 2  // Number of threads

int **A, **B, **C;  // Global matrices

// Structure to hold information for each thread
typedef struct
{
    int thread_id;
    int num_rows;  // Number of rows each thread will handle
} thread_data_t;

// Function for each thread to perform matrix multiplication
void* matrixMultiplyThread(void* arg)
{
    long id = (long)arg;
    // Divide the task (rows) of each thread based on thread id
    // compute a portion of the matrix multiplication
```

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
0 1000 1000 1000

real    0m3.349s
user    0m5.138s
sys     0m0.060s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$
```

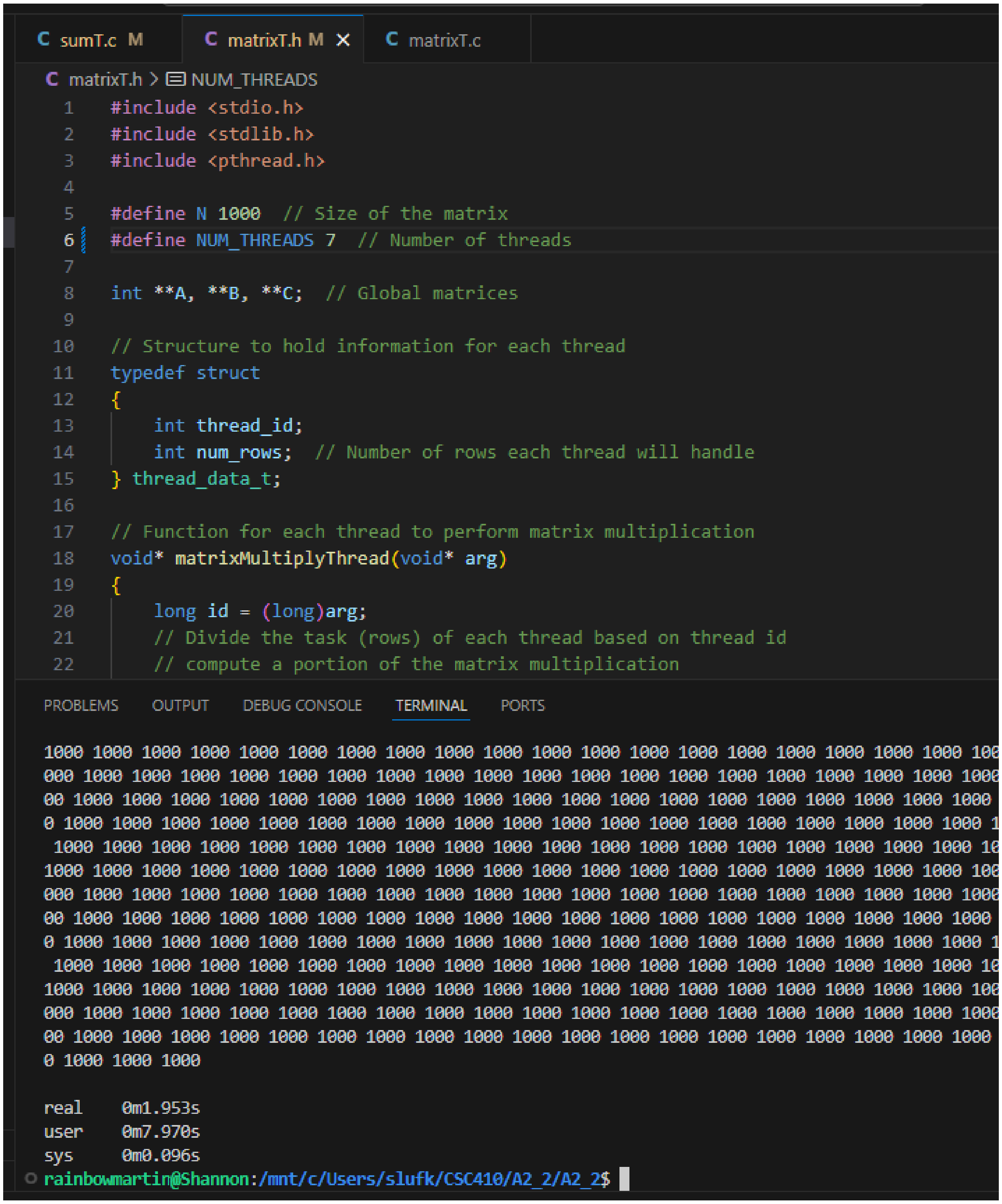
Pthread – 5 threads



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

#define N 1000  // Size of the matrix
#define NUM_THREADS 5  // Number of threads

int **A, **B, **C;  // Global matrices

// Structure to hold information for each thread
typedef struct
{
    int thread_id;
    int num_rows;  // Number of rows each thread will handle
} thread_data_t;

// Function for each thread to perform matrix multiplication
void* matrixMultiplyThread(void* arg)
{
    long id = (long)arg;
    // Divide the task (rows) of each thread based on thread id
    // compute a portion of the matrix multiplication
```

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000

real    0m2.192s
user    0m6.713s
sys     0m0.068s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$
```

Pthread – 7 threads



sumT.c M   matrixT.h M ×   matrixT.c

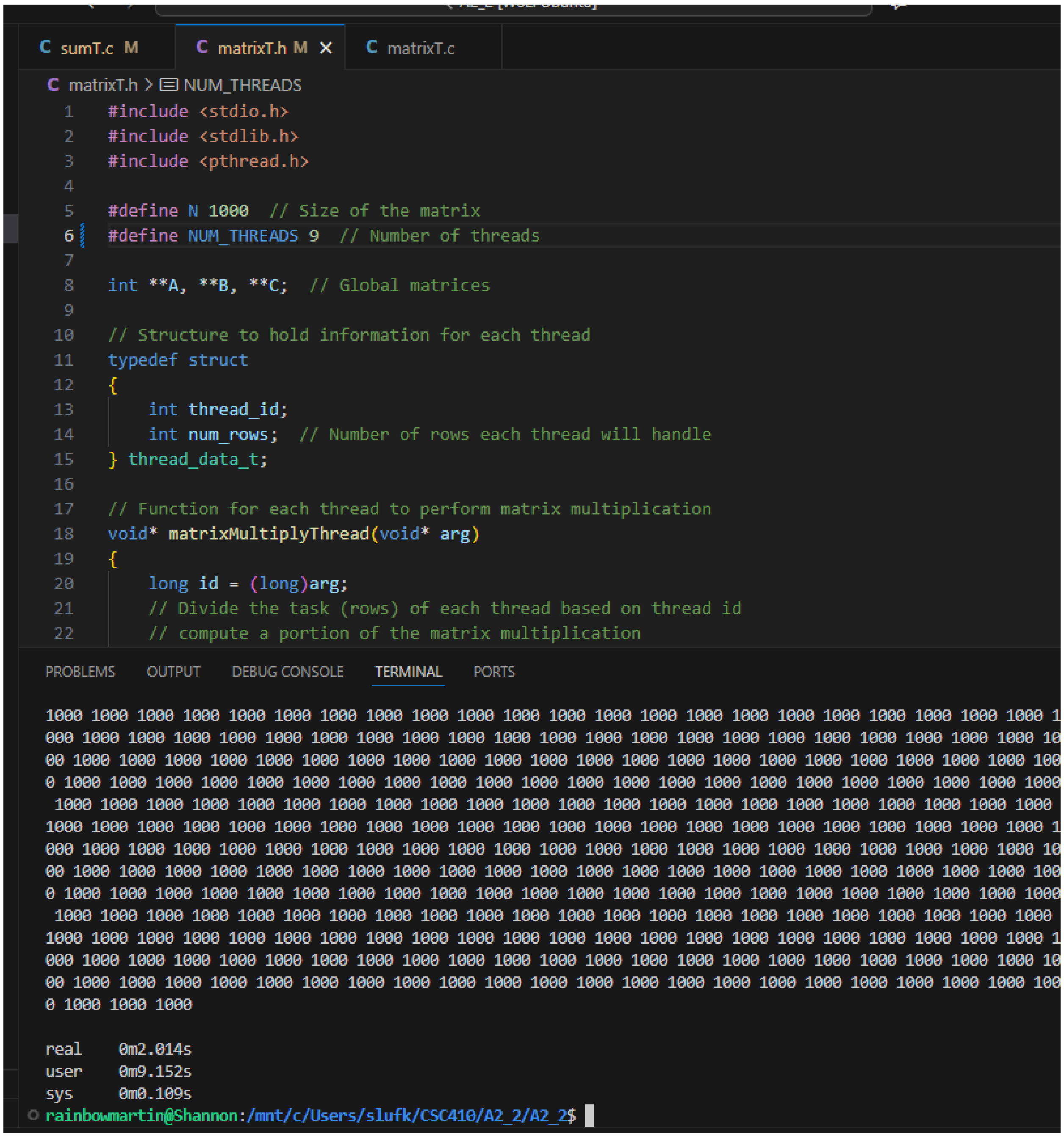
matrixT.h > NUM_THREADS

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

#define N 1000  // Size of the matrix
#define NUM_THREADS 7  // Number of threads

int **A, **B, **C;  // Global matrices

// Structure to hold information for each thread
typedef struct
{
    int thread_id;
    int num_rows;  // Number of rows each thread will handle
} thread_data_t;

// Function for each thread to perform matrix multiplication
void* matrixMultiplyThread(void* arg)
{
    long id = (long)arg;
    // Divide the task (rows) of each thread based on thread id
    // compute a portion of the matrix multiplication
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

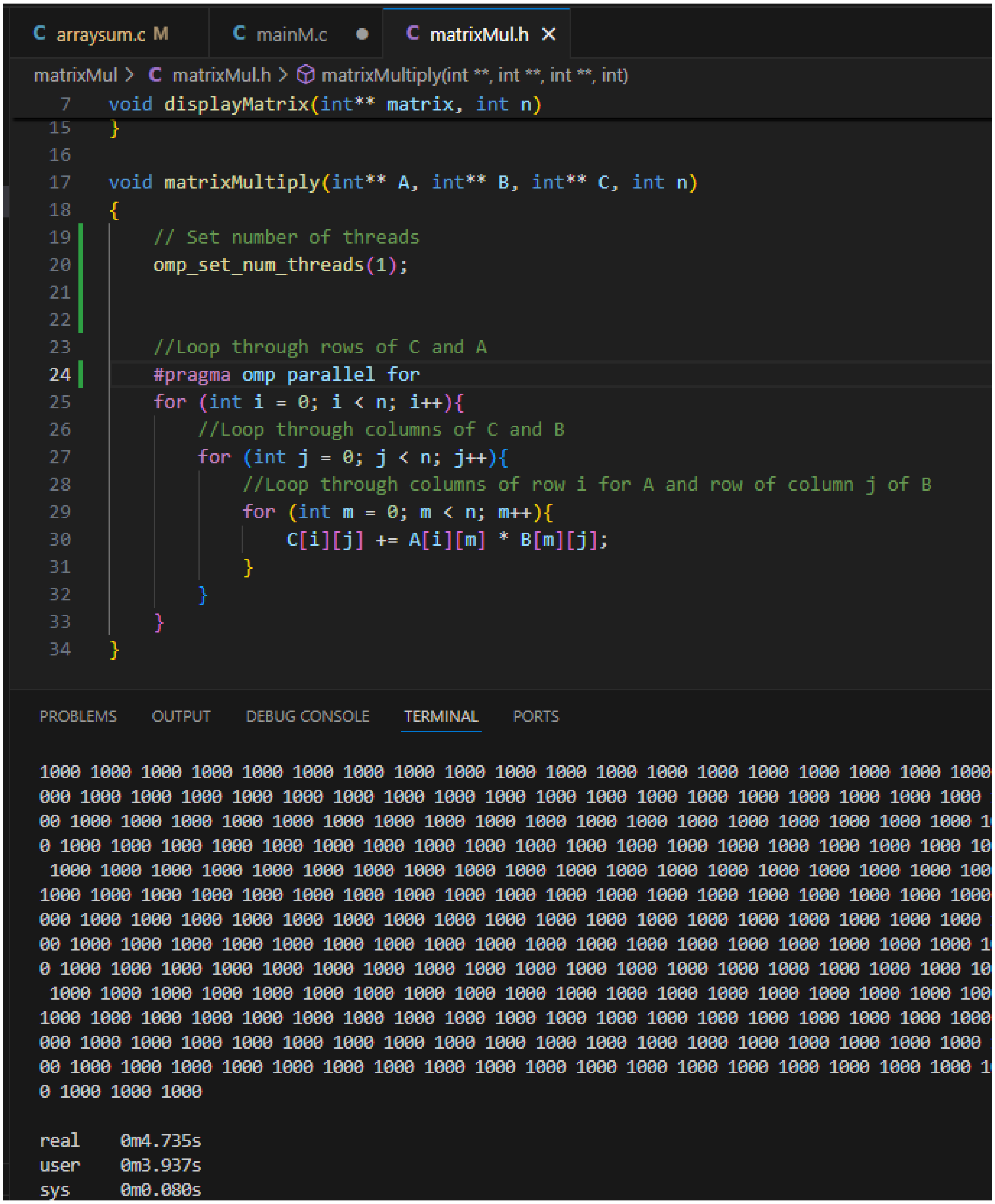
```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000

real    0m1.953s
user    0m7.970s
sys     0m0.096s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$ 
```

Pthread – 9 threads


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

#define N 1000  // Size of the matrix
#define NUM_THREADS 9  // Number of threads

int **A, **B, **C;  // Global matrices

// Structure to hold information for each thread
typedef struct
{
    int thread_id;
    int num_rows;  // Number of rows each thread will handle
} thread_data_t;

// Function for each thread to perform matrix multiplication
void* matrixMultiplyThread(void* arg)
{
    long id = (long)arg;
    // Divide the task (rows) of each thread based on thread id
    // compute a portion of the matrix multiplication
```

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
0 1000 1000 1000

real    0m2.014s
user    0m9.152s
sys     0m0.109s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A2_2/A2_2$
```

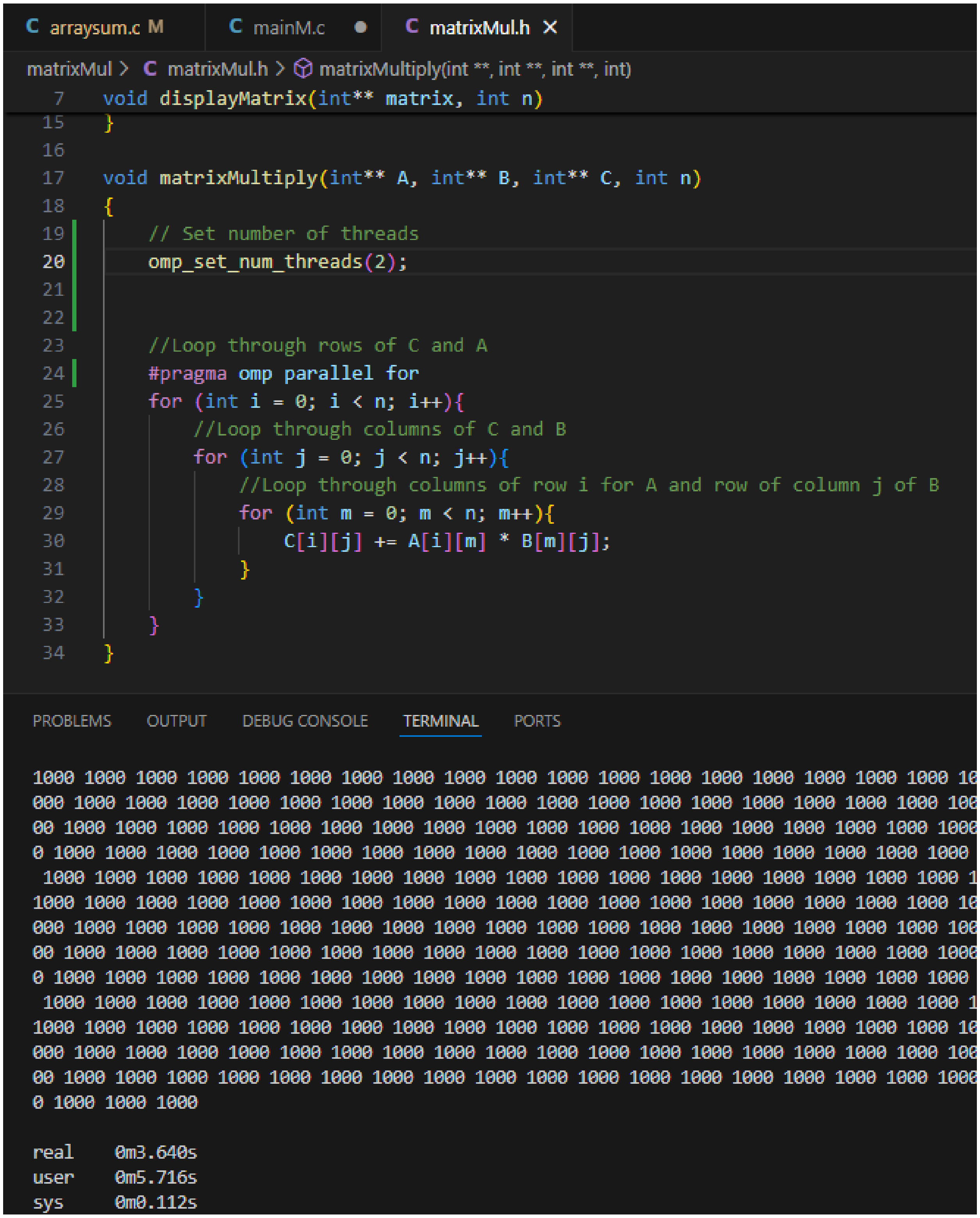
Omp – 1 thread


arraysum.c M
mainM.c
matrixMul.h
matrixMul > matrixMul.h > matrixMultiply(int **, int **, int **, int)

```c
void displayMatrix(int** matrix, int n)
}

void matrixMultiply(int** A, int** B, int** C, int n)
{
    // Set number of threads
    omp_set_num_threads(1);


    //Loop through rows of C and A
    #pragma omp parallel for
    for (int i = 0; i < n; i++){
        //Loop through columns of C and B
        for (int j = 0; j < n; j++){
            //Loop through columns of row i for A and row of column j of B
            for (int m = 0; m < n; m++){
                C[i][j] += A[i][m] * B[m][j];
            }
        }
    }
}
```

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
0 1000 1000 1000

real    0m4.735s
user    0m3.937s
sys     0m0.080s
```

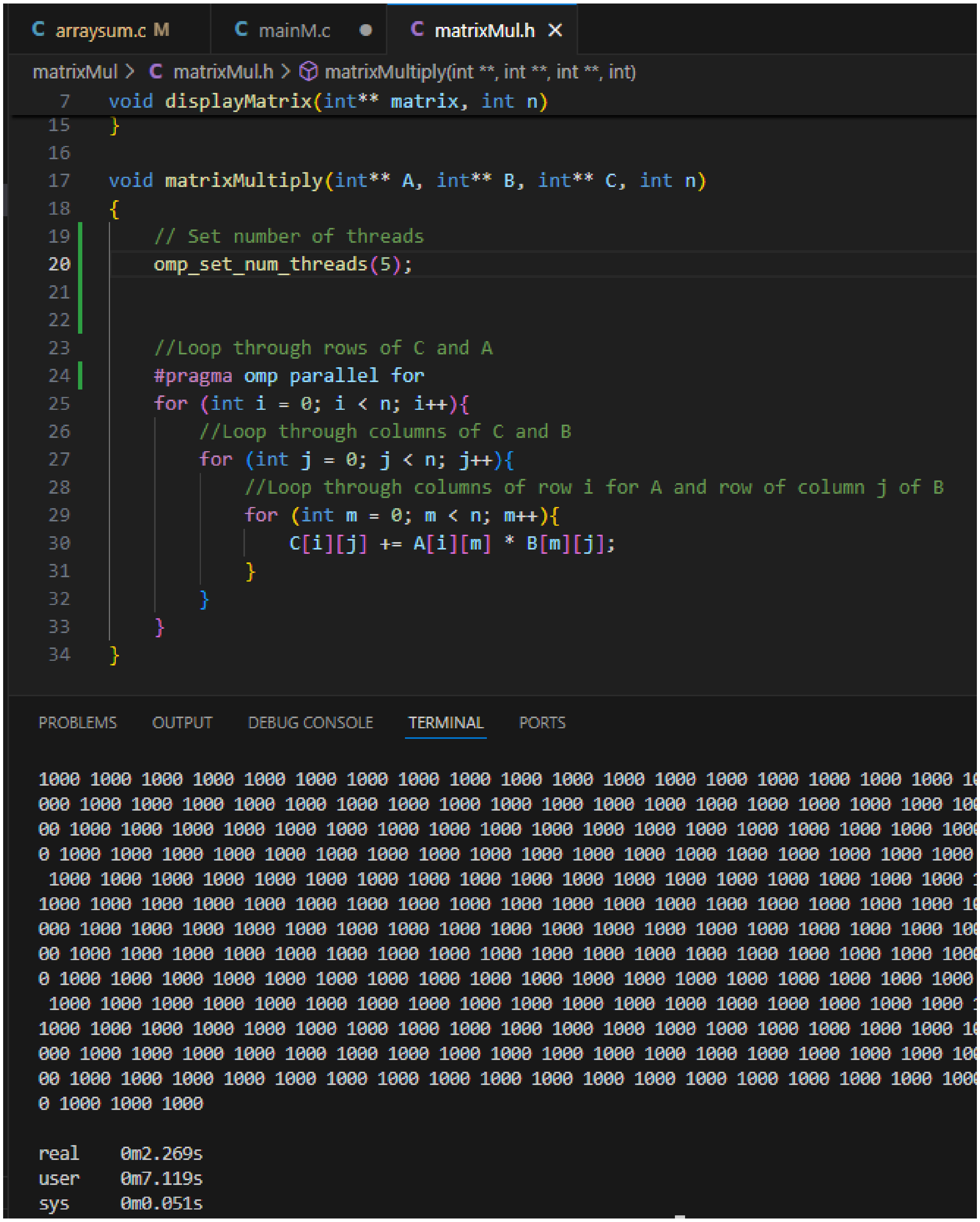
Omp – 2 threads



```c
void displayMatrix(int** matrix, int n)
}

void matrixMultiply(int** A, int** B, int** C, int n)
{
    // Set number of threads
    omp_set_num_threads(2);


    //Loop through rows of C and A
    #pragma omp parallel for
    for (int i = 0; i < n; i++){
        //Loop through columns of C and B
        for (int j = 0; j < n; j++){
            //Loop through columns of row i for A and row of column j of B
            for (int m = 0; m < n; m++){
                C[i][j] += A[i][m] * B[m][j];
            }
        }
    }
}
```

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000

real    0m3.640s
user    0m5.716s
sys     0m0.112s
```

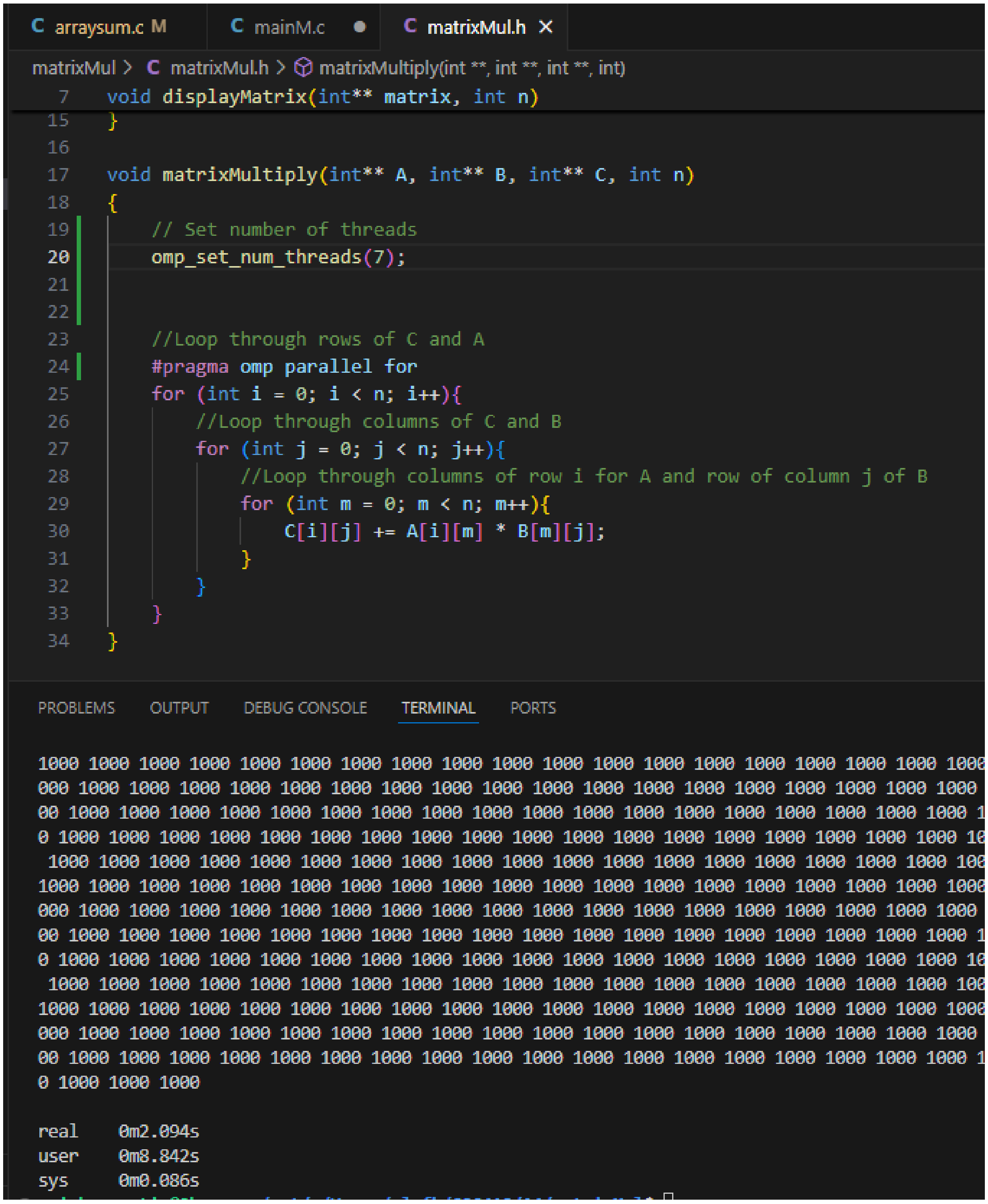
Omp – 5 threads



```c
void displayMatrix(int** matrix, int n)
}

void matrixMultiply(int** A, int** B, int** C, int n)
{
    // Set number of threads
    omp_set_num_threads(5);


    //Loop through rows of C and A
    #pragma omp parallel for
    for (int i = 0; i < n; i++){
        //Loop through columns of C and B
        for (int j = 0; j < n; j++){
            //Loop through columns of row i for A and row of column j of B
            for (int m = 0; m < n; m++){
                C[i][j] += A[i][m] * B[m][j];
            }
        }
    }
}
```

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
0 1000 1000 1000

real    0m2.269s
user    0m7.119s
sys     0m0.051s
```

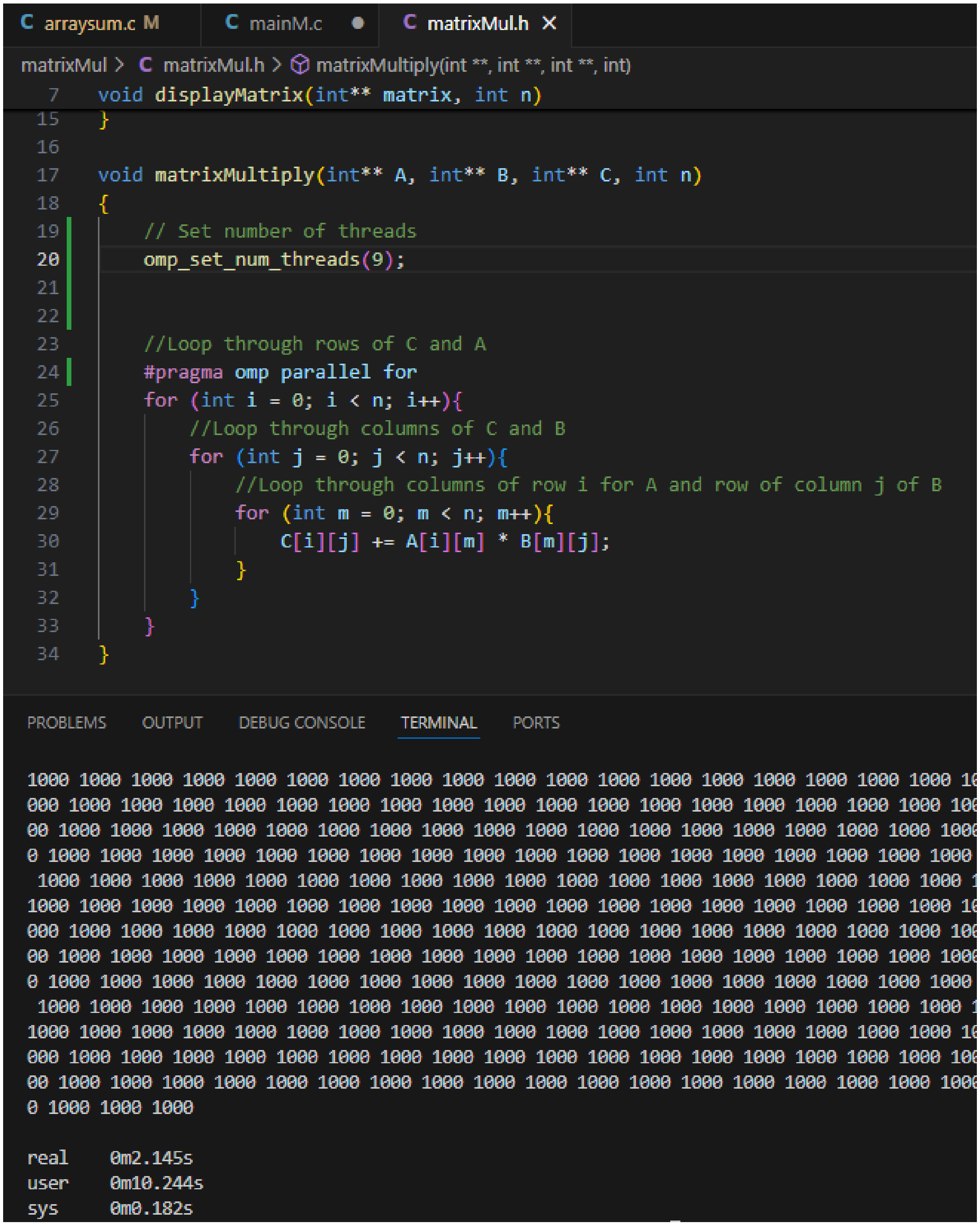
Omp – 7 threads



```c
void displayMatrix(int** matrix, int n)
}

void matrixMultiply(int** A, int** B, int** C, int n)
{
    // Set number of threads
    omp_set_num_threads(7);


    //Loop through rows of C and A
    #pragma omp parallel for
    for (int i = 0; i < n; i++){
        //Loop through columns of C and B
        for (int j = 0; j < n; j++){
            //Loop through columns of row i for A and row of column j of B
            for (int m = 0; m < n; m++){
                C[i][j] += A[i][m] * B[m][j];
            }
        }
    }
}
```

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
0 1000 1000 1000

real    0m2.094s
user    0m8.842s
sys     0m0.086s
```

Omp – 9 threads

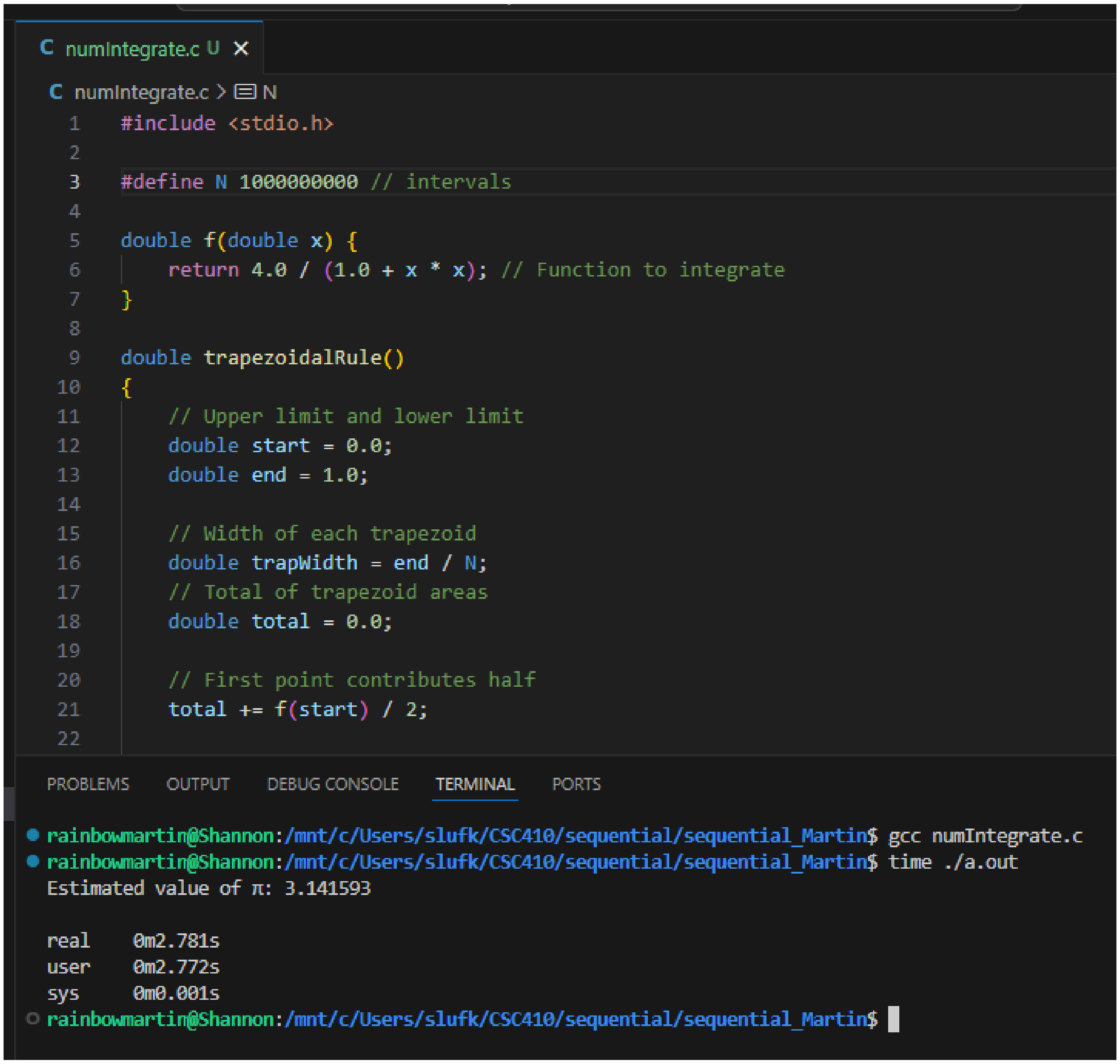


```c
void displayMatrix(int** matrix, int n)
}

void matrixMultiply(int** A, int** B, int** C, int n)
{
    // Set number of threads
    omp_set_num_threads(9);


    //Loop through rows of C and A
    #pragma omp parallel for
    for (int i = 0; i < n; i++){
        //Loop through columns of C and B
        for (int j = 0; j < n; j++){
            //Loop through columns of row i for A and row of column j of B
            for (int m = 0; m < n; m++){
                C[i][j] += A[i][m] * B[m][j];
            }
        }
    }
}
```

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 10
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 100
00 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
0 1000 1000 1000

real    0m2.145s
user    0m10.244s
sys     0m0.182s
```

Trapezoidal Rule Integration

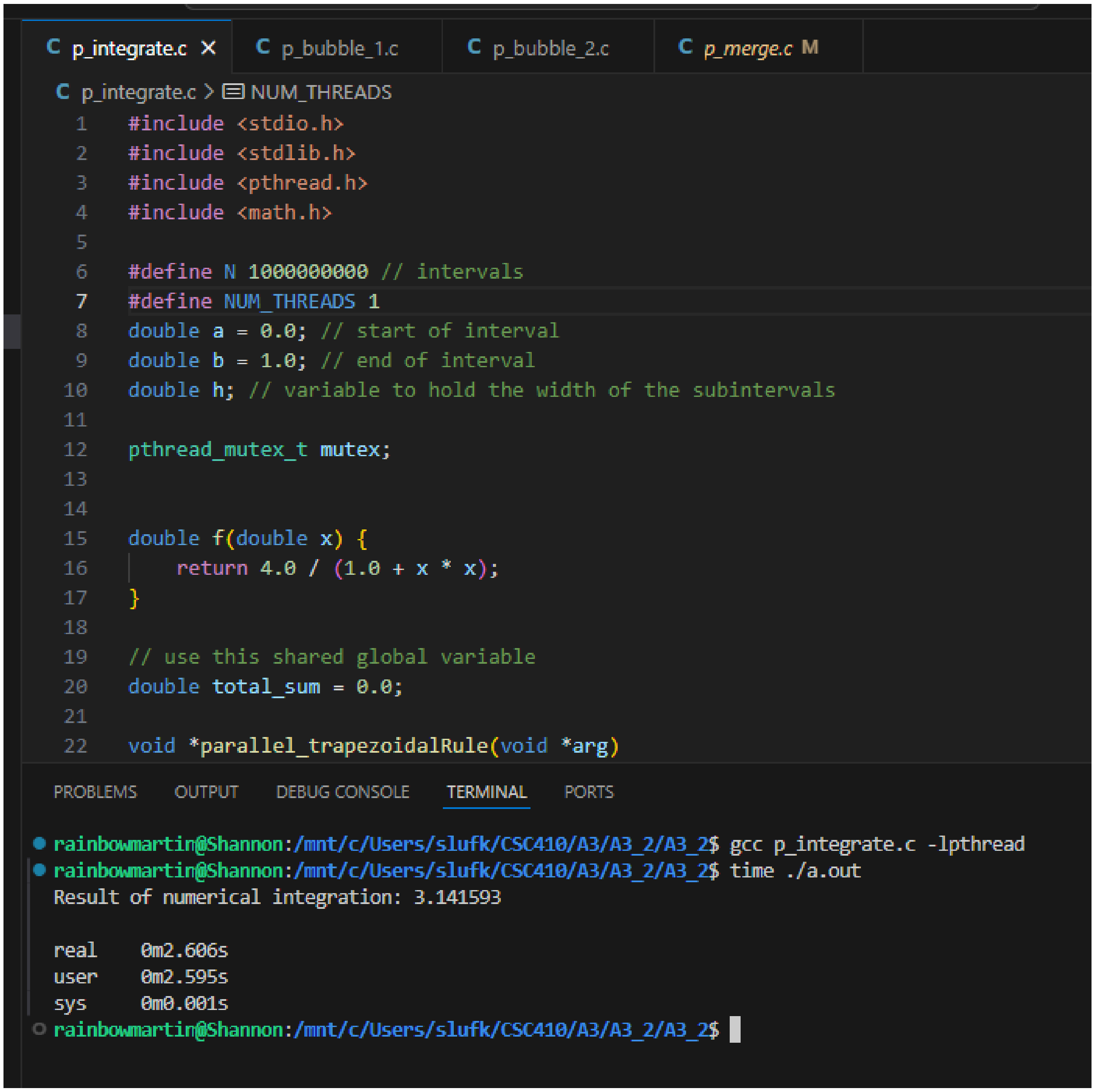
Sequential



```c
#include <stdio.h>

#define N 1000000000 // intervals

double f(double x) {
    return 4.0 / (1.0 + x * x); // Function to integrate
}

double trapezoidalRule()
{
    // Upper limit and lower limit
    double start = 0.0;
    double end = 1.0;

    // Width of each trapezoid
    double trapWidth = end / N;
    // Total of trapezoid areas
    double total = 0.0;

    // First point contributes half
    total += f(start) / 2;

```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/sequential/sequential_Martin$ gcc numIntegrate.c
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/sequential/sequential_Martin$ time ./a.out
Estimated value of π: 3.141593

real    0m2.781s
user    0m2.772s
sys     0m0.001s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/sequential/sequential_Martin$
```

Pthread – 1 thread



p_integrate.c ✕ | p_bubble_1.c | p_bubble_2.c | p_merge.c M

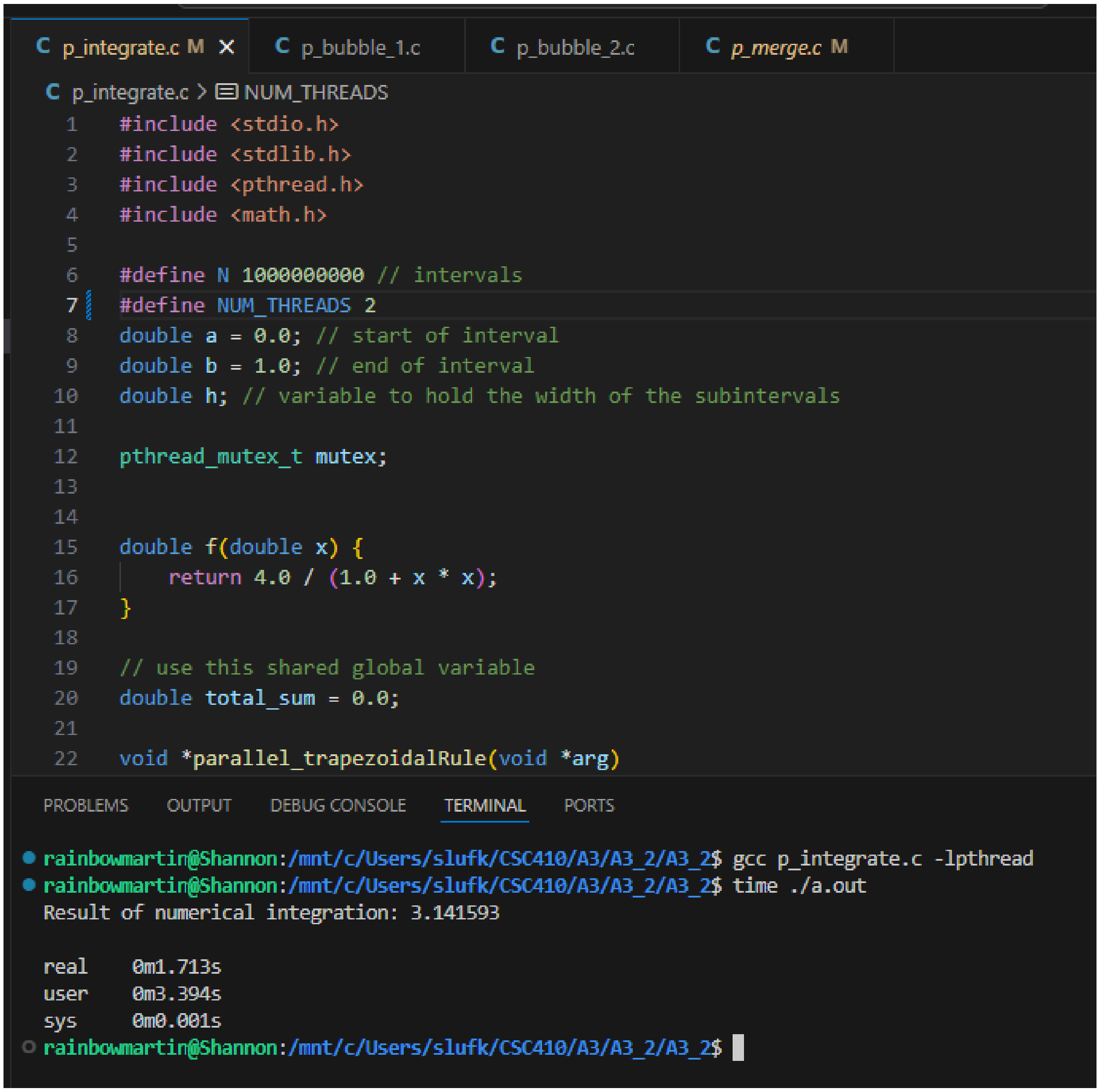
p_integrate.c > NUM_THREADS

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

#define N 1000000000 // intervals
#define NUM_THREADS 1
double a = 0.0; // start of interval
double b = 1.0; // end of interval
double h; // variable to hold the width of the subintervals

pthread_mutex_t mutex;


double f(double x) {
    return 4.0 / (1.0 + x * x);
}

// use this shared global variable
double total_sum = 0.0;

void *parallel_trapezoidalRule(void *arg)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ gcc p_integrate.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ time ./a.out
Result of numerical integration: 3.141593

real    0m2.606s
user    0m2.595s
sys     0m0.001s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$
```

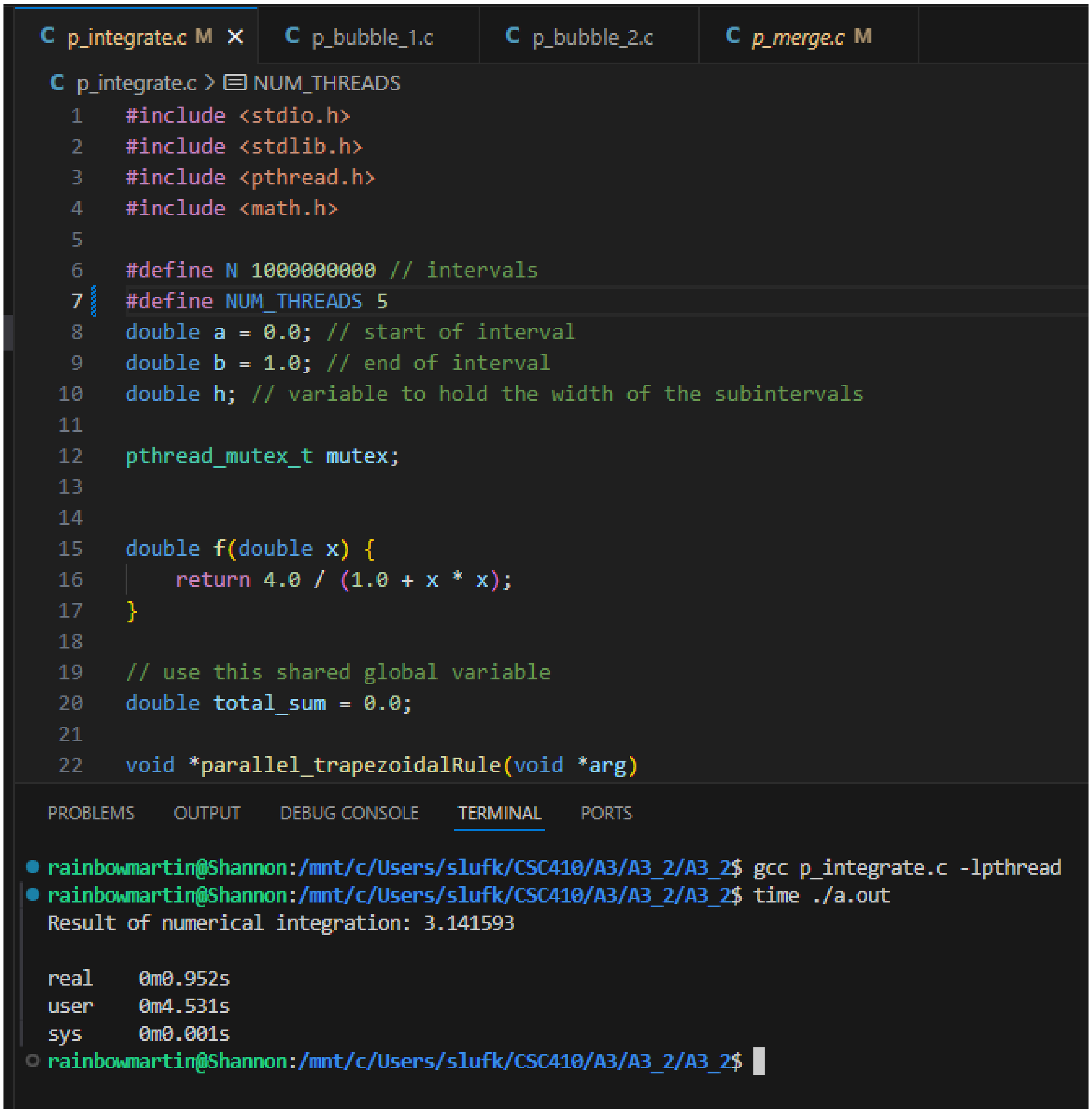
Pthread – 2 threads



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

#define N 1000000000 // intervals
#define NUM_THREADS 2
double a = 0.0; // start of interval
double b = 1.0; // end of interval
double h; // variable to hold the width of the subintervals

pthread_mutex_t mutex;


double f(double x) {
    return 4.0 / (1.0 + x * x);
}

// use this shared global variable
double total_sum = 0.0;

void *parallel_trapezoidalRule(void *arg)
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ gcc p_integrate.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ time ./a.out
Result of numerical integration: 3.141593

real    0m1.713s
user    0m3.394s
sys     0m0.001s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$
```

Pthread – 5 threads



p_integrate.c M × | p_bubble_1.c | p_bubble_2.c | p_merge.c M

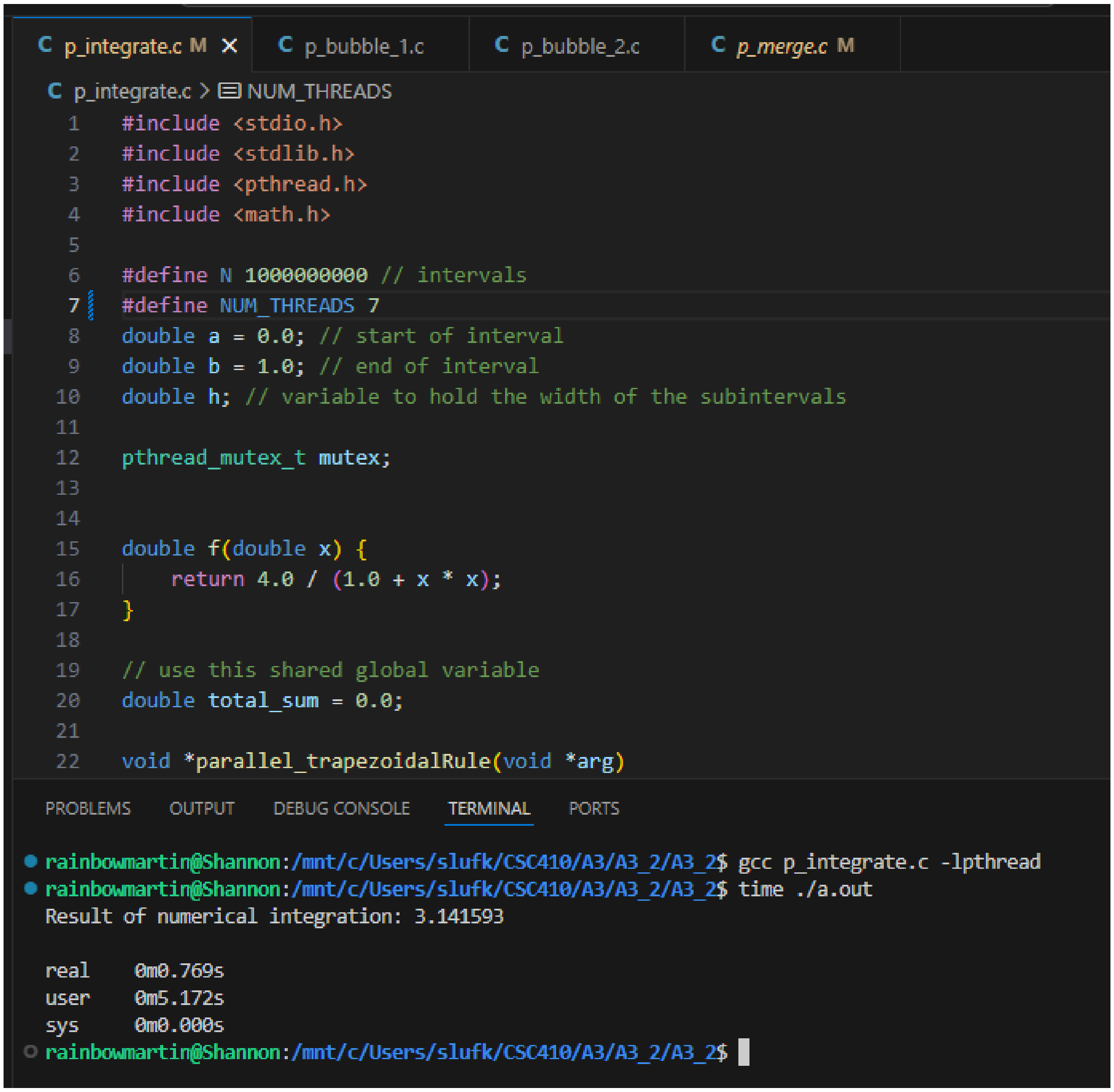
p_integrate.c > NUM_THREADS

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

#define N 1000000000 // intervals
#define NUM_THREADS 5
double a = 0.0; // start of interval
double b = 1.0; // end of interval
double h; // variable to hold the width of the subintervals

pthread_mutex_t mutex;


double f(double x) {
    return 4.0 / (1.0 + x * x);
}

// use this shared global variable
double total_sum = 0.0;

void *parallel_trapezoidalRule(void *arg)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ gcc p_integrate.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ time ./a.out
Result of numerical integration: 3.141593

real    0m0.952s
user    0m4.531s
sys     0m0.001s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$
```

Pthread - 7 threads



p_integrate.c M × | p_bubble_1.c | p_bubble_2.c | p_merge.c M

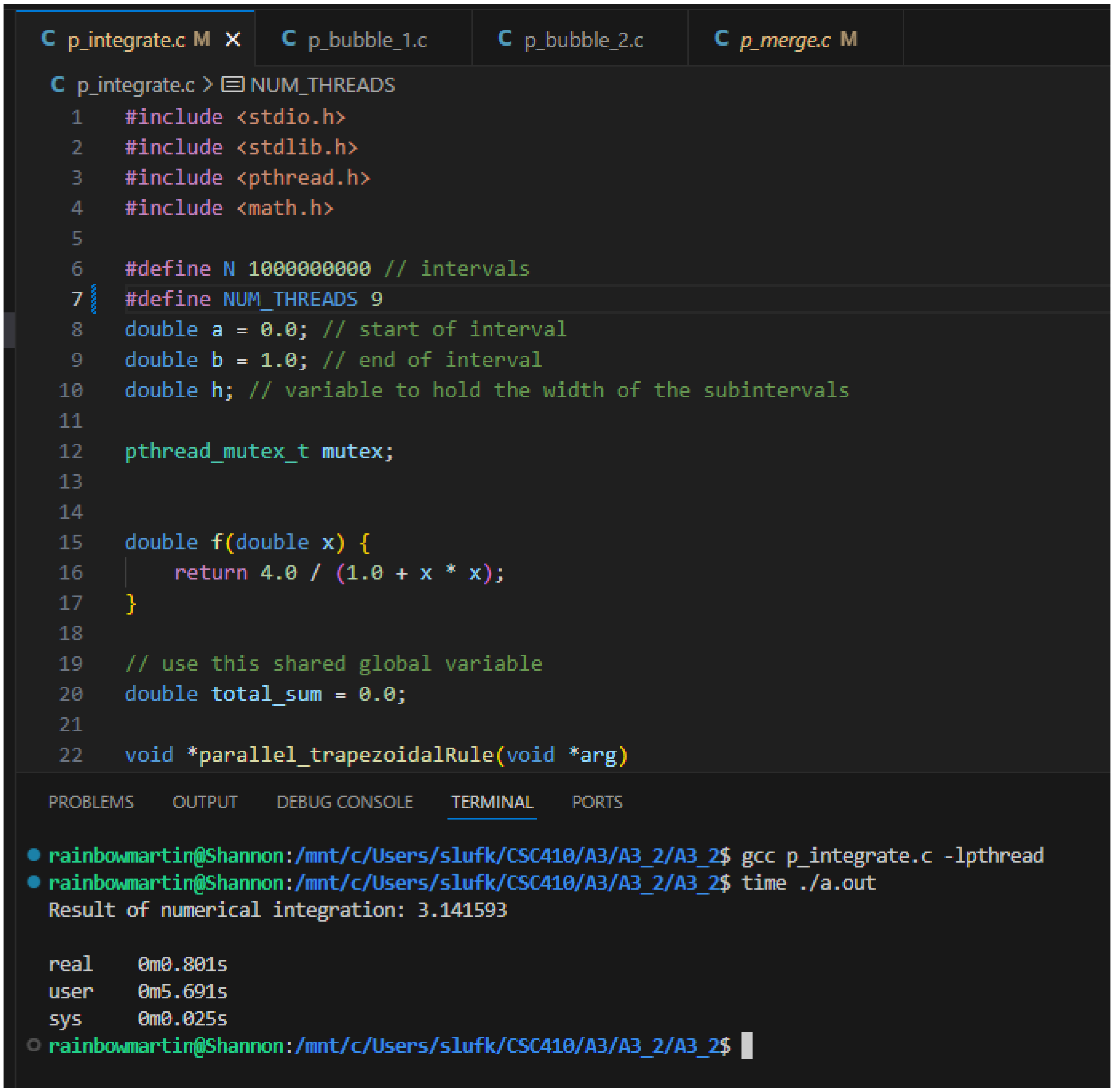
p_integrate.c > NUM_THREADS

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

#define N 1000000000 // intervals
#define NUM_THREADS 7
double a = 0.0; // start of interval
double b = 1.0; // end of interval
double h; // variable to hold the width of the subintervals

pthread_mutex_t mutex;


double f(double x) {
    return 4.0 / (1.0 + x * x);
}

// use this shared global variable
double total_sum = 0.0;

void *parallel_trapezoidalRule(void *arg)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ gcc p_integrate.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ time ./a.out
Result of numerical integration: 3.141593

real    0m0.769s
user    0m5.172s
sys     0m0.000s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$
```

Pthread - 9 threads



p_integrate.c M × | p_bubble_1.c | p_bubble_2.c | p_merge.c M

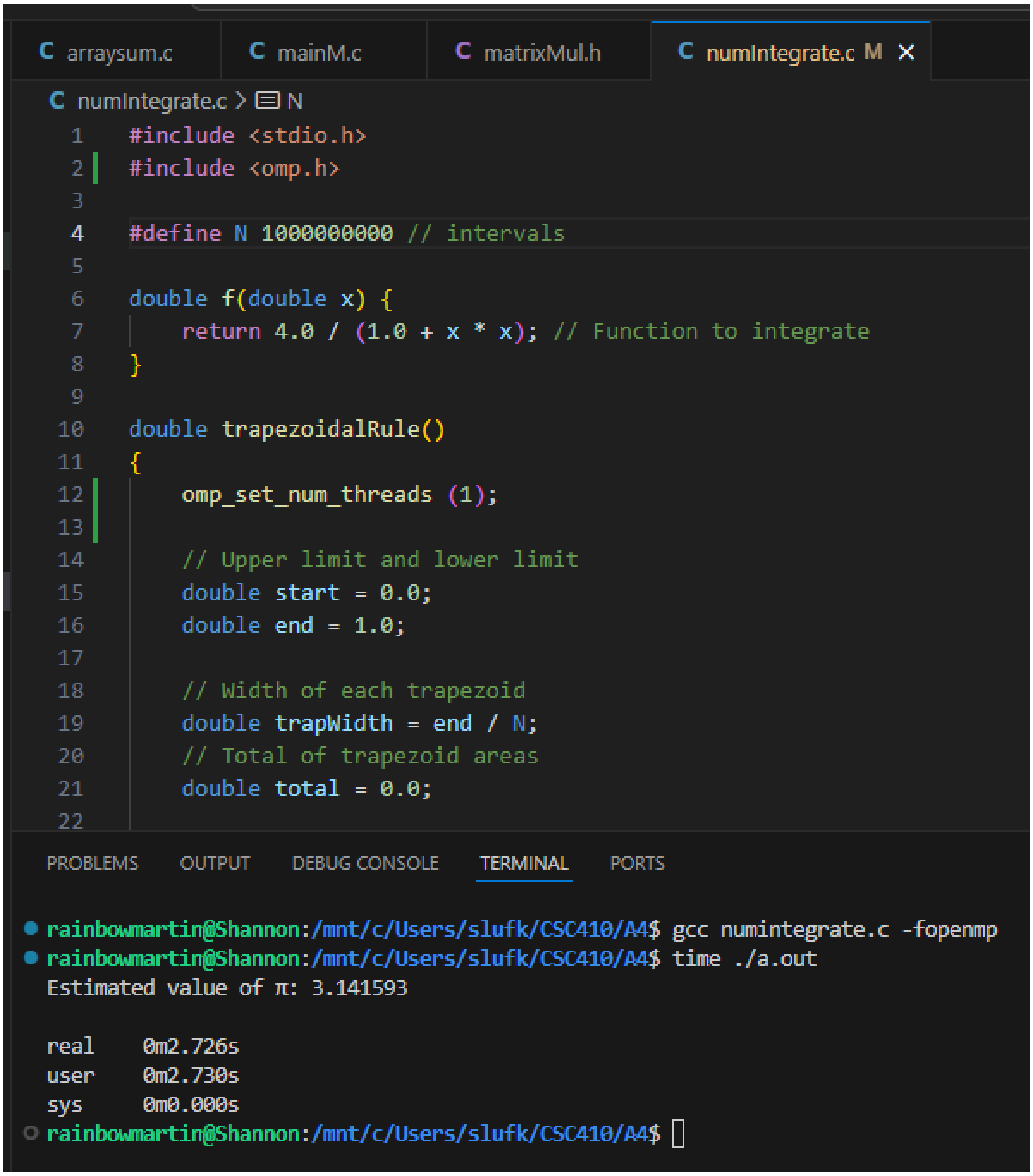
p_integrate.c > NUM_THREADS

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

#define N 1000000000 // intervals
#define NUM_THREADS 9
double a = 0.0; // start of interval
double b = 1.0; // end of interval
double h; // variable to hold the width of the subintervals

pthread_mutex_t mutex;


double f(double x) {
    return 4.0 / (1.0 + x * x);
}

// use this shared global variable
double total_sum = 0.0;

void *parallel_trapezoidalRule(void *arg)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ gcc p_integrate.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ time ./a.out
Result of numerical integration: 3.141593

real    0m0.801s
user    0m5.691s
sys     0m0.025s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A3/A3_2/A3_2$ 
```

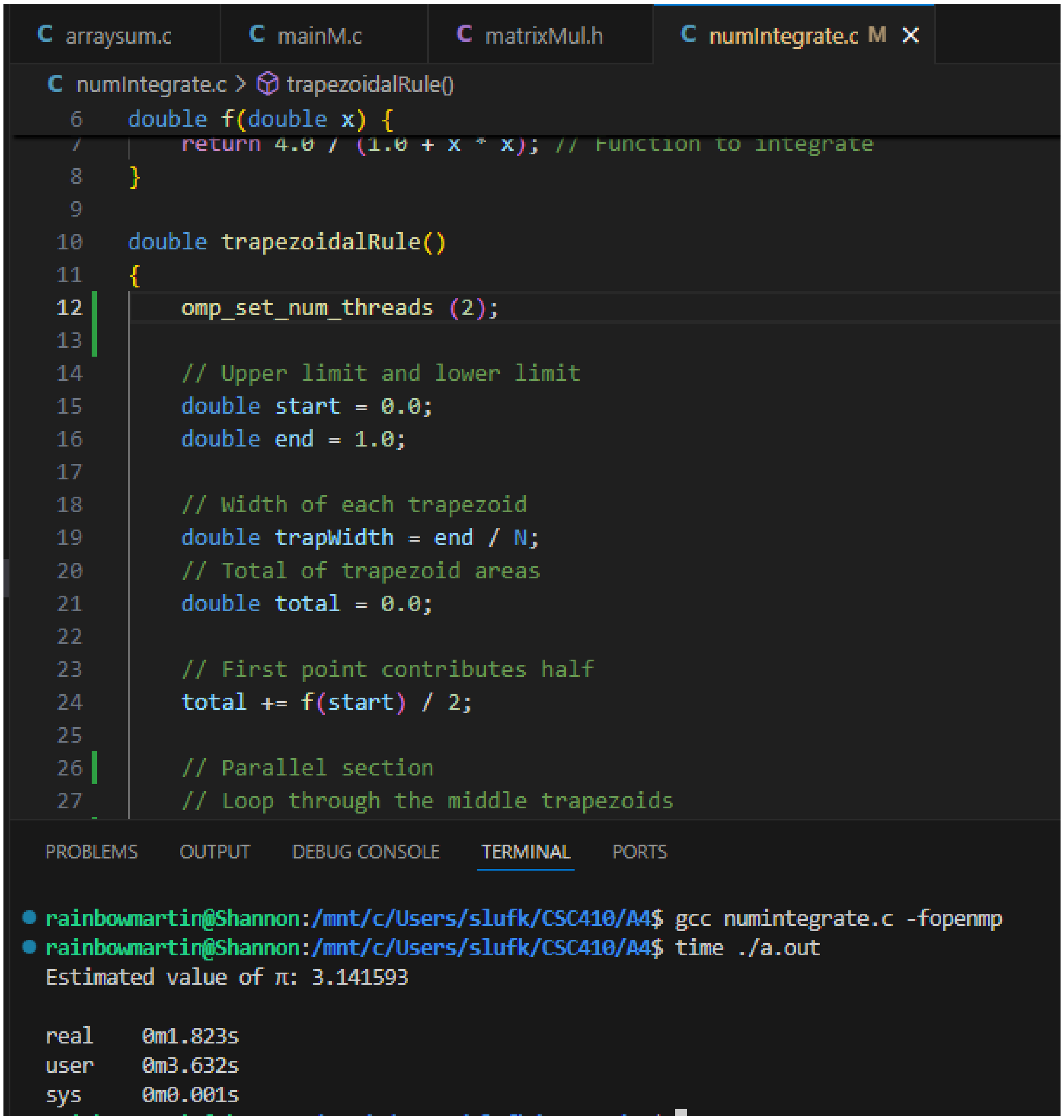
Omp – 1 thread



```c
#include <stdio.h>
#include <omp.h>

#define N 1000000000 // intervals

double f(double x) {
    return 4.0 / (1.0 + x * x); // Function to integrate
}

double trapezoidalRule()
{
    omp_set_num_threads (1);

    // Upper limit and lower limit
    double start = 0.0;
    double end = 1.0;

    // Width of each trapezoid
    double trapWidth = end / N;
    // Total of trapezoid areas
    double total = 0.0;

```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc numintegrate.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Estimated value of π: 3.141593

real    0m2.726s
user    0m2.730s
sys     0m0.000s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$
```

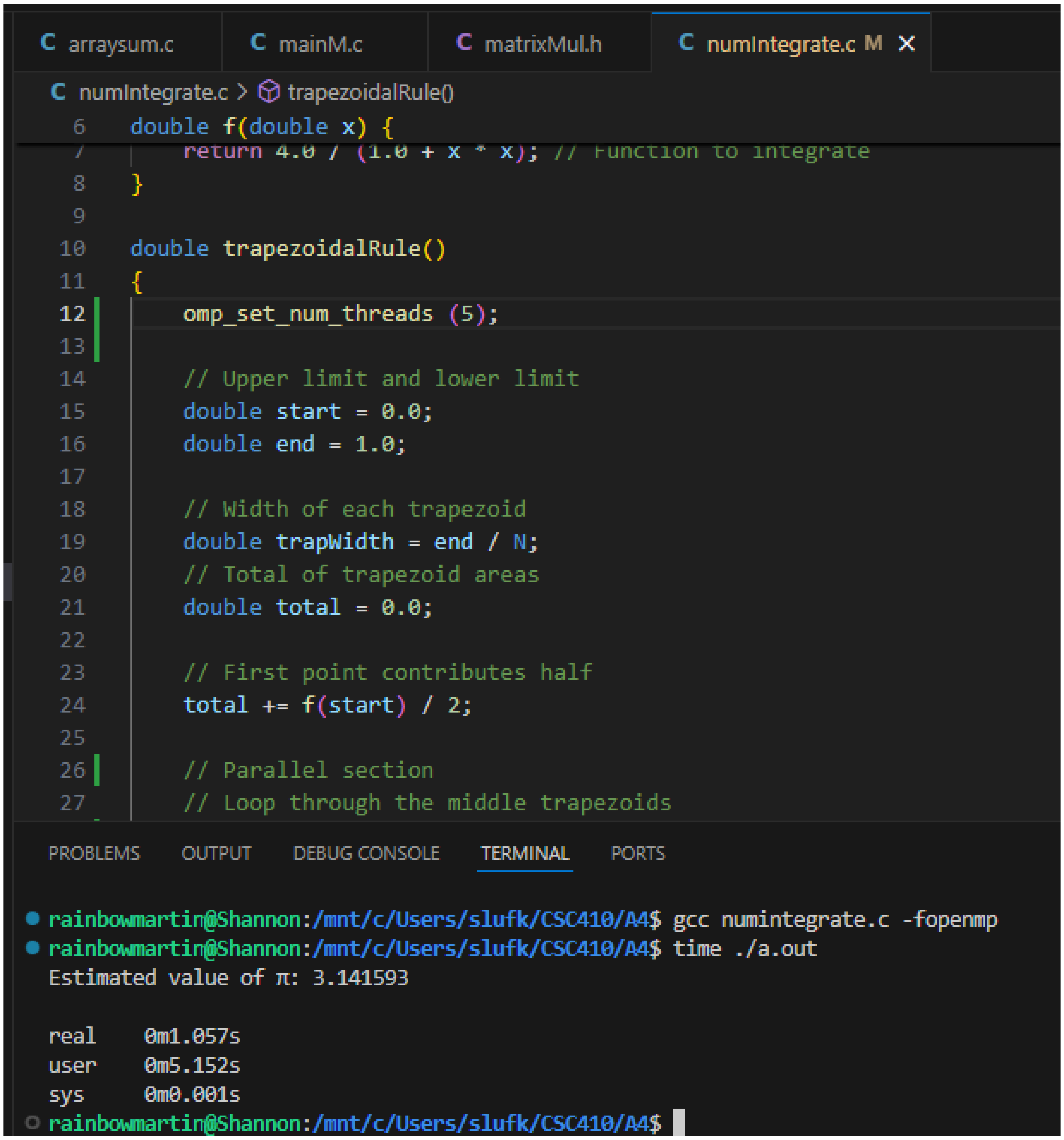
Omp – 2 threads



```c
double f(double x) {
    return 4.0 / (1.0 + x * x); // Function to integrate
}

double trapezoidalRule()
{
    omp_set_num_threads (2);

    // Upper limit and lower limit
    double start = 0.0;
    double end = 1.0;

    // Width of each trapezoid
    double trapWidth = end / N;
    // Total of trapezoid areas
    double total = 0.0;

    // First point contributes half
    total += f(start) / 2;

    // Parallel section
    // Loop through the middle trapezoids
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc numintegrate.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Estimated value of π: 3.141593

real    0m1.823s
user    0m3.632s
sys     0m0.001s
```

Omp – 5 threads



arraysum.c | mainM.c | matrixMul.h | numIntegrate.c M

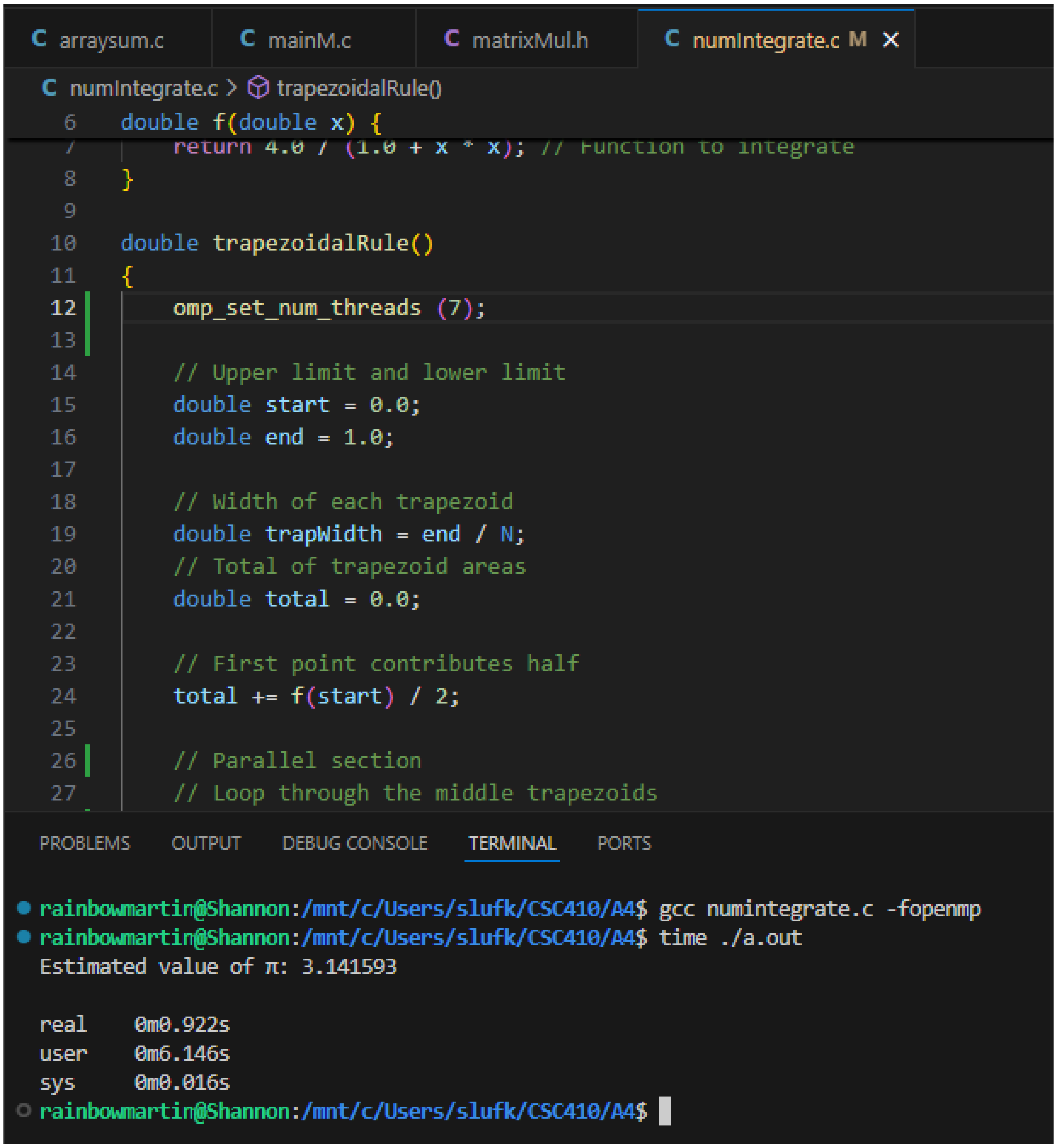
numIntegrate.c > trapezoidalRule()

```c
double f(double x) {
    return 4.0 / (1.0 + x * x); // Function to integrate
}

double trapezoidalRule()
{
    omp_set_num_threads (5);

    // Upper limit and lower limit
    double start = 0.0;
    double end = 1.0;

    // Width of each trapezoid
    double trapWidth = end / N;
    // Total of trapezoid areas
    double total = 0.0;

    // First point contributes half
    total += f(start) / 2;

    // Parallel section
    // Loop through the middle trapezoids
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc numintegrate.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Estimated value of π: 3.141593

real    0m1.057s
user    0m5.152s
sys     0m0.001s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$
```

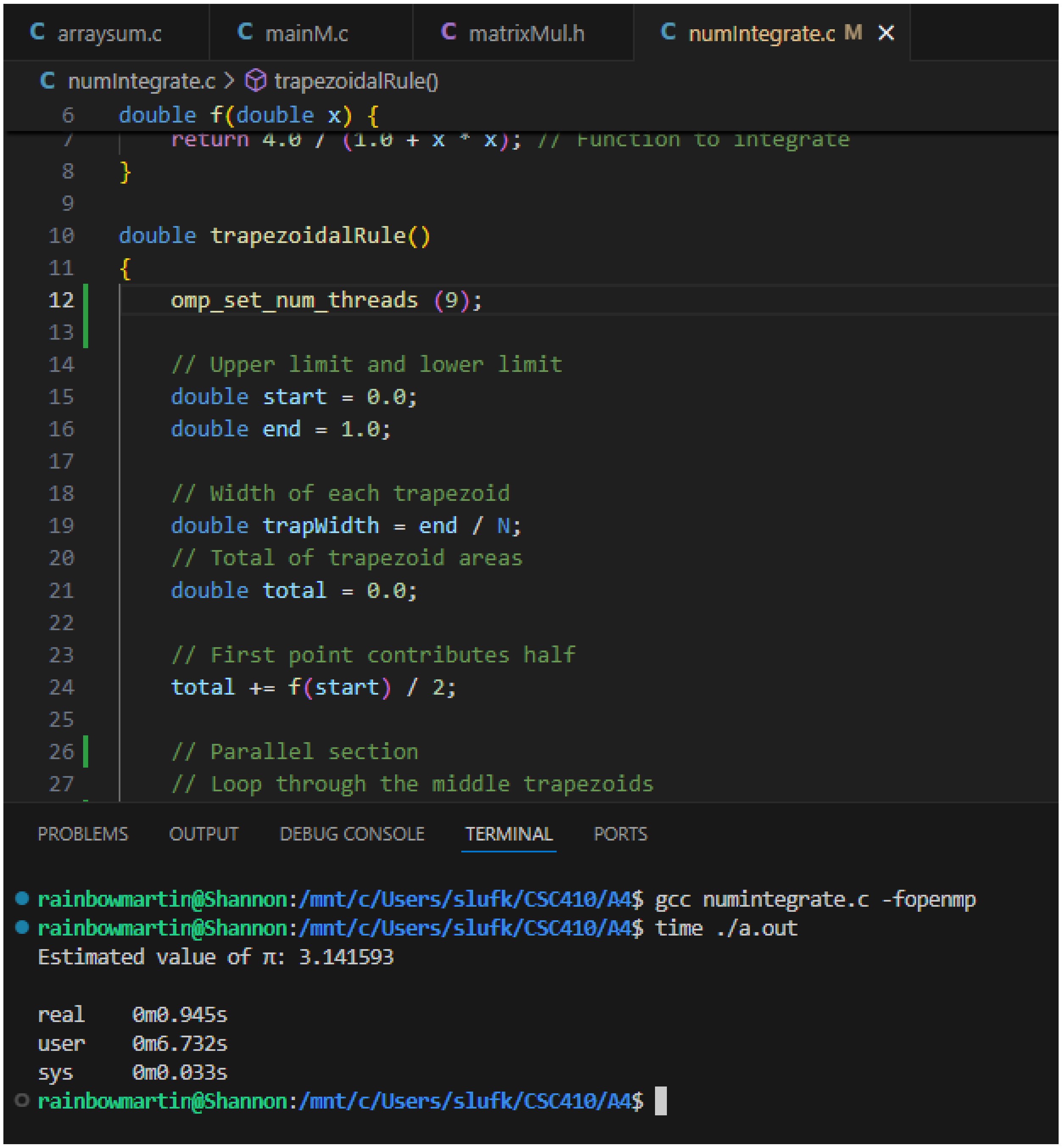
Omp – 7 threads



```c
double f(double x) {
    return 4.0 / (1.0 + x * x); // Function to integrate
}

double trapezoidalRule()
{
    omp_set_num_threads (7);

    // Upper limit and lower limit
    double start = 0.0;
    double end = 1.0;

    // Width of each trapezoid
    double trapWidth = end / N;
    // Total of trapezoid areas
    double total = 0.0;

    // First point contributes half
    total += f(start) / 2;

    // Parallel section
    // Loop through the middle trapezoids
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc numintegrate.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Estimated value of π: 3.141593

real    0m0.922s
user    0m6.146s
sys     0m0.016s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$
```

Omp – 9 threads



C arraysum.c   C mainM.c   C matrixMul.h   C numIntegrate.c M ×

C numIntegrate.c > trapezoidalRule()

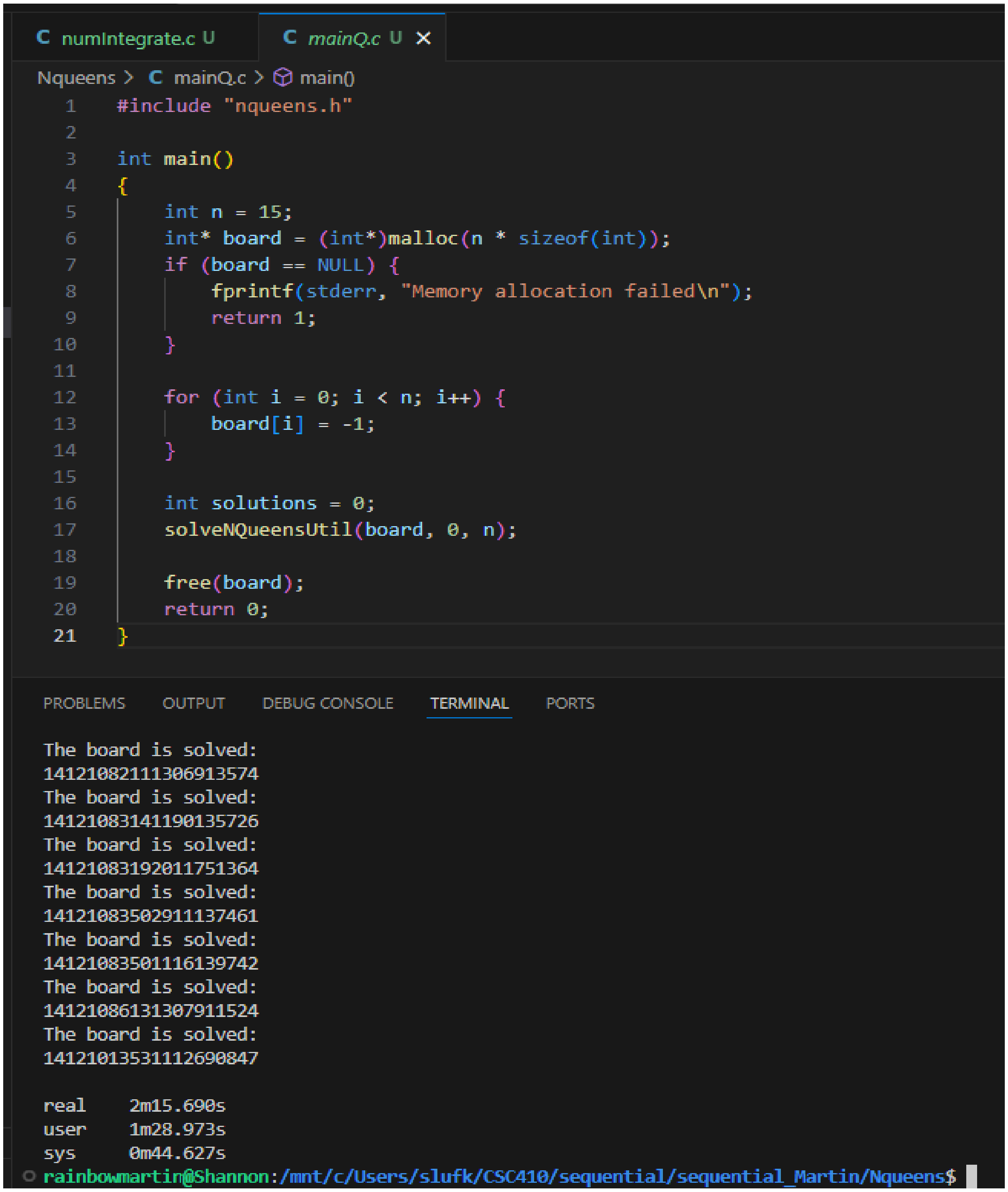
```c
double f(double x) {
    return 4.0 / (1.0 + x * x); // Function to integrate
}

double trapezoidalRule()
{
    omp_set_num_threads (9);

    // Upper limit and lower limit
    double start = 0.0;
    double end = 1.0;

    // Width of each trapezoid
    double trapWidth = end / N;
    // Total of trapezoid areas
    double total = 0.0;

    // First point contributes half
    total += f(start) / 2;

    // Parallel section
    // Loop through the middle trapezoids
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ gcc numintegrate.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$ time ./a.out
Estimated value of π: 3.141593

real    0m0.945s
user    0m6.732s
sys     0m0.033s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4$
```

N Queens

Sequential



numIntegrate.c U | mainQ.c U

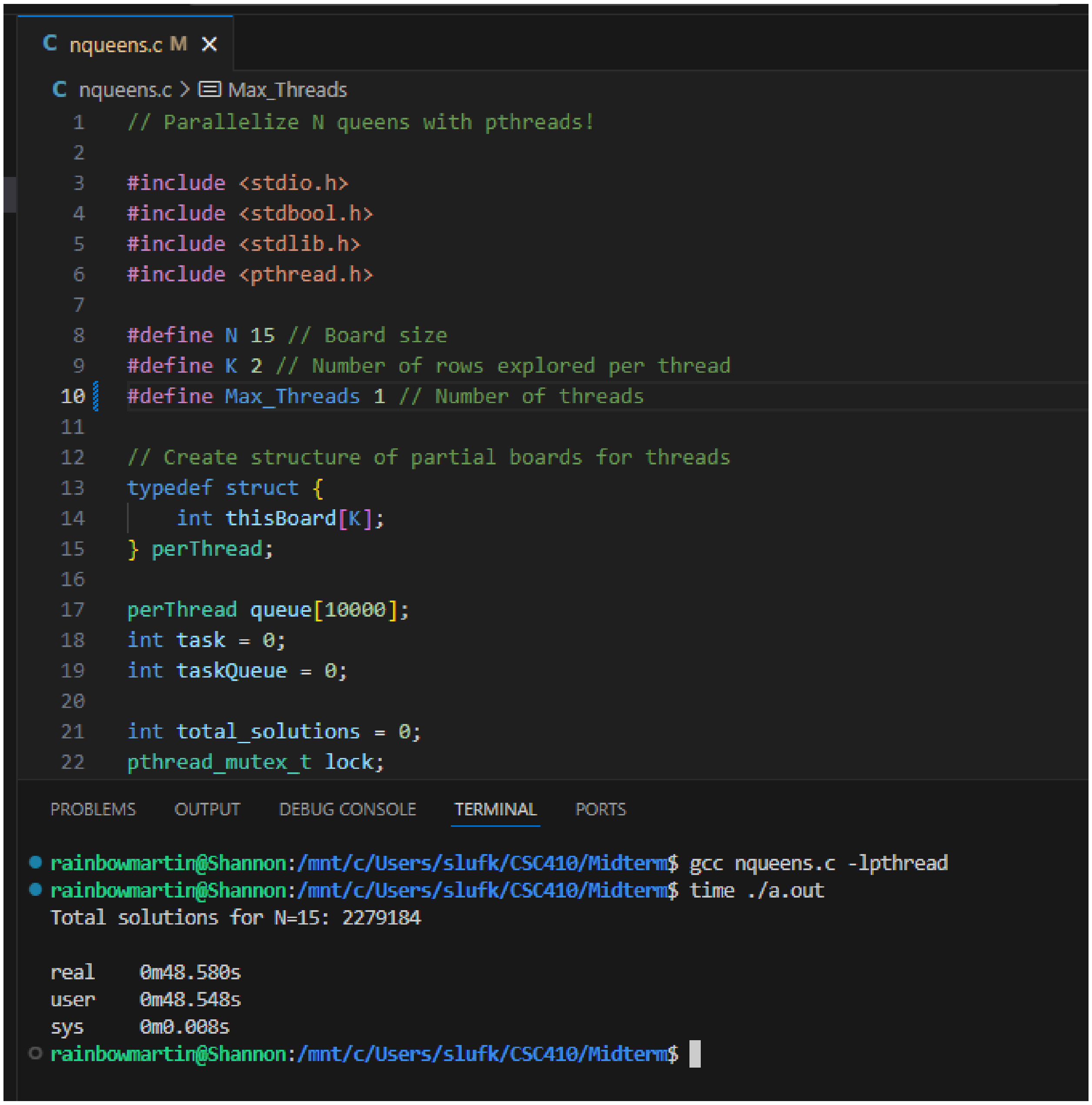
Nqueens > mainQ.c > main()

```c
#include "nqueens.h"

int main()
{
    int n = 15;
    int* board = (int*)malloc(n * sizeof(int));
    if (board == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
    }

    for (int i = 0; i < n; i++) {
        board[i] = -1;
    }

    int solutions = 0;
    solveNQueensUtil(board, 0, n);

    free(board);
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
The board is solved:
1412108211130691357 4
The board is solved:
14121083141190135726
The board is solved:
14121083192011751364
The board is solved:
14121083502911137461
The board is solved:
14121083501116139742
The board is solved:
14121086131307911524
The board is solved:
14121013531112690847

real    2m15.690s
user    1m28.973s
sys     0m44.627s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/sequential/sequential_Martin/Nqueens$
```

Pthread – 1 thread



nqueens.c M ×

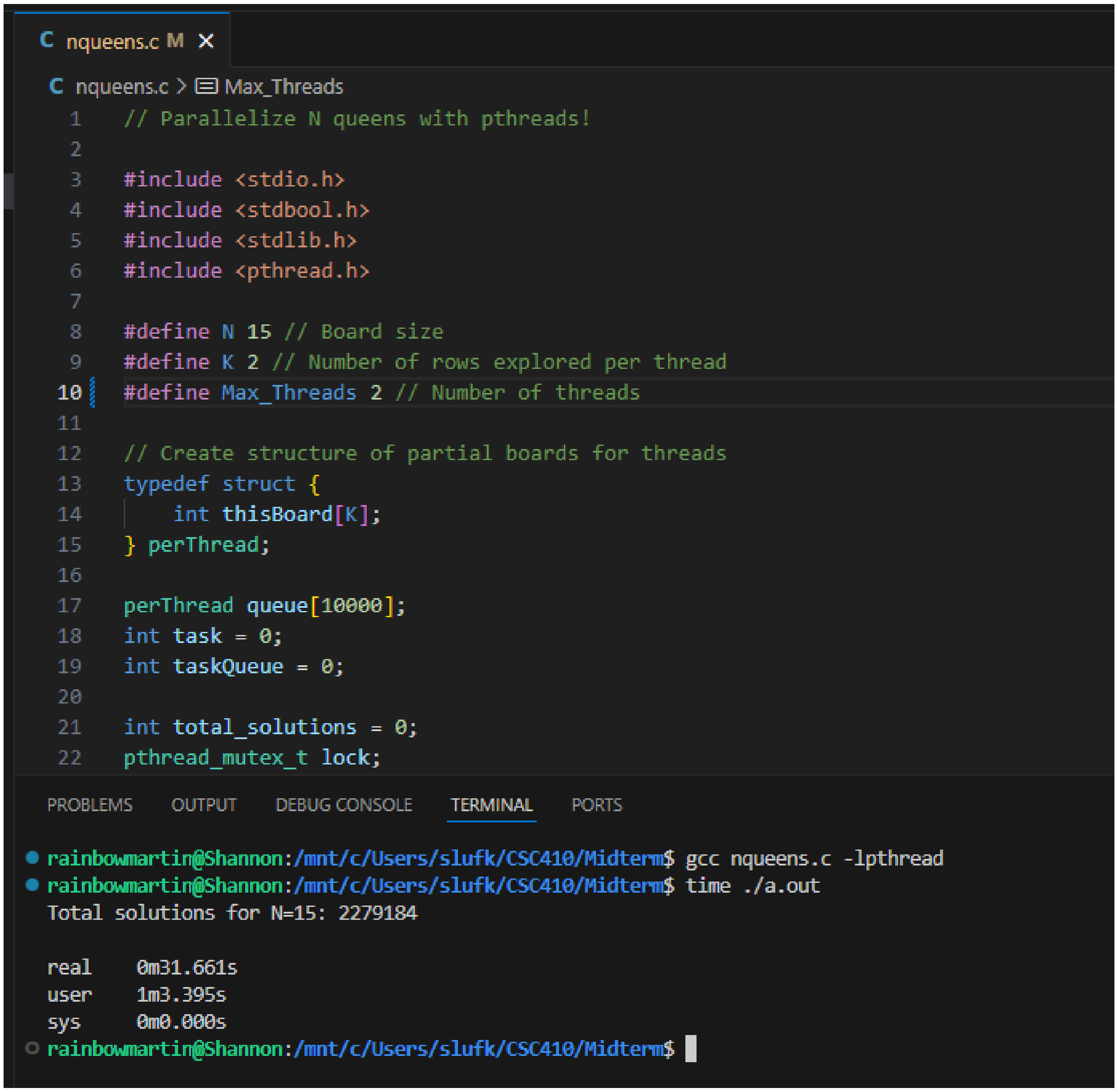
nqueens.c > Max_Threads

```c
// Parallelize N queens with pthreads!

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <pthread.h>

#define N 15 // Board size
#define K 2 // Number of rows explored per thread
#define Max_Threads 1 // Number of threads

// Create structure of partial boards for threads
typedef struct {
    int thisBoard[K];
} perThread;

perThread queue[10000];
int task = 0;
int taskQueue = 0;

int total_solutions = 0;
pthread_mutex_t lock;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ gcc nqueens.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ time ./a.out
Total solutions for N=15: 2279184

real    0m48.580s
user    0m48.548s
sys     0m0.008s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$
```

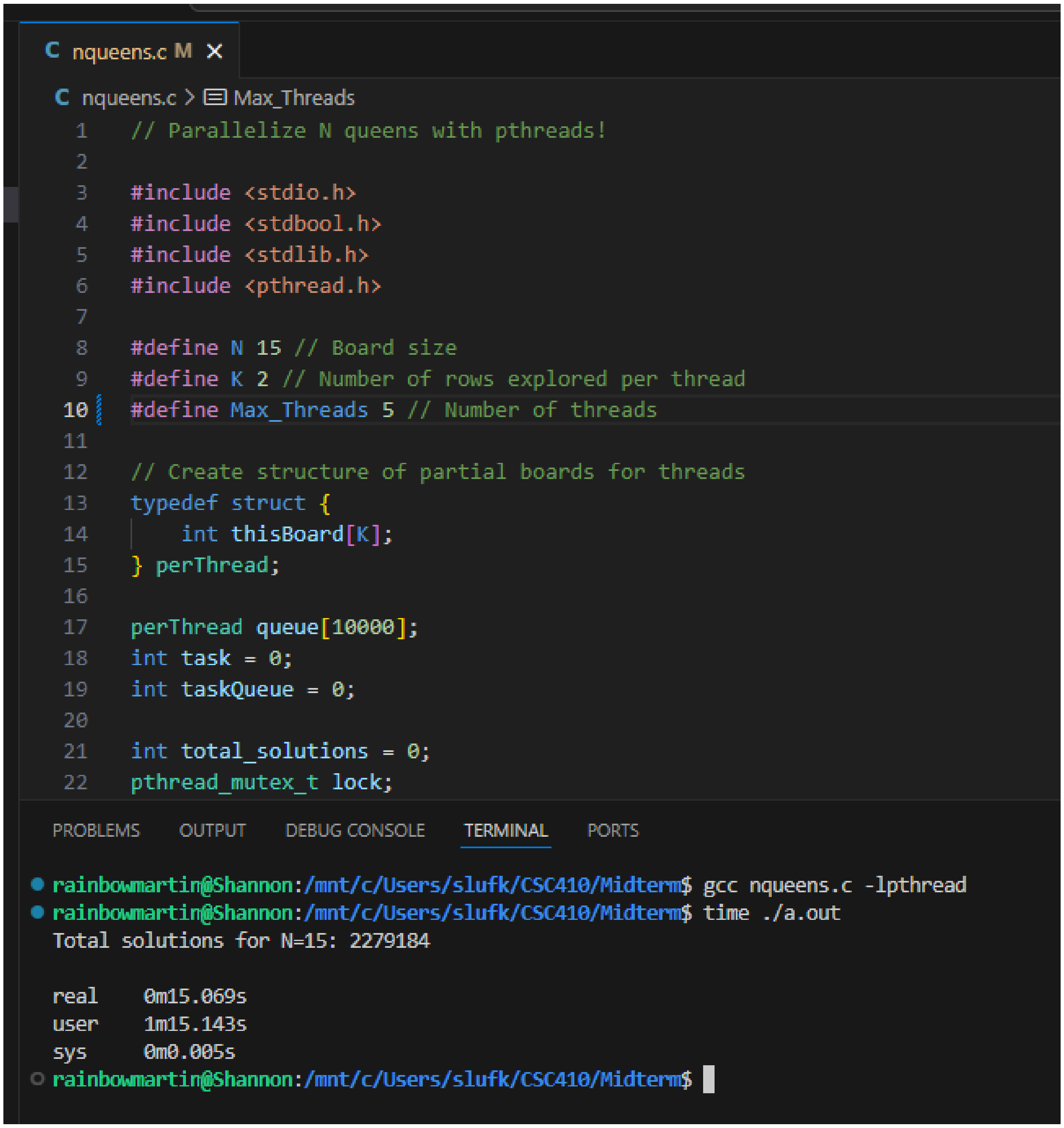
Pthread – 2 threads



```c
// Parallelize N queens with pthreads!

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <pthread.h>

#define N 15 // Board size
#define K 2 // Number of rows explored per thread
#define Max_Threads 2 // Number of threads

// Create structure of partial boards for threads
typedef struct {
    int thisBoard[K];
} perThread;

perThread queue[10000];
int task = 0;
int taskQueue = 0;

int total_solutions = 0;
pthread_mutex_t lock;
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ gcc nqueens.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ time ./a.out
Total solutions for N=15: 2279184

real    0m31.661s
user    1m3.395s
sys     0m0.000s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$
```

pthread – 5 threads



nqueens.c M ×

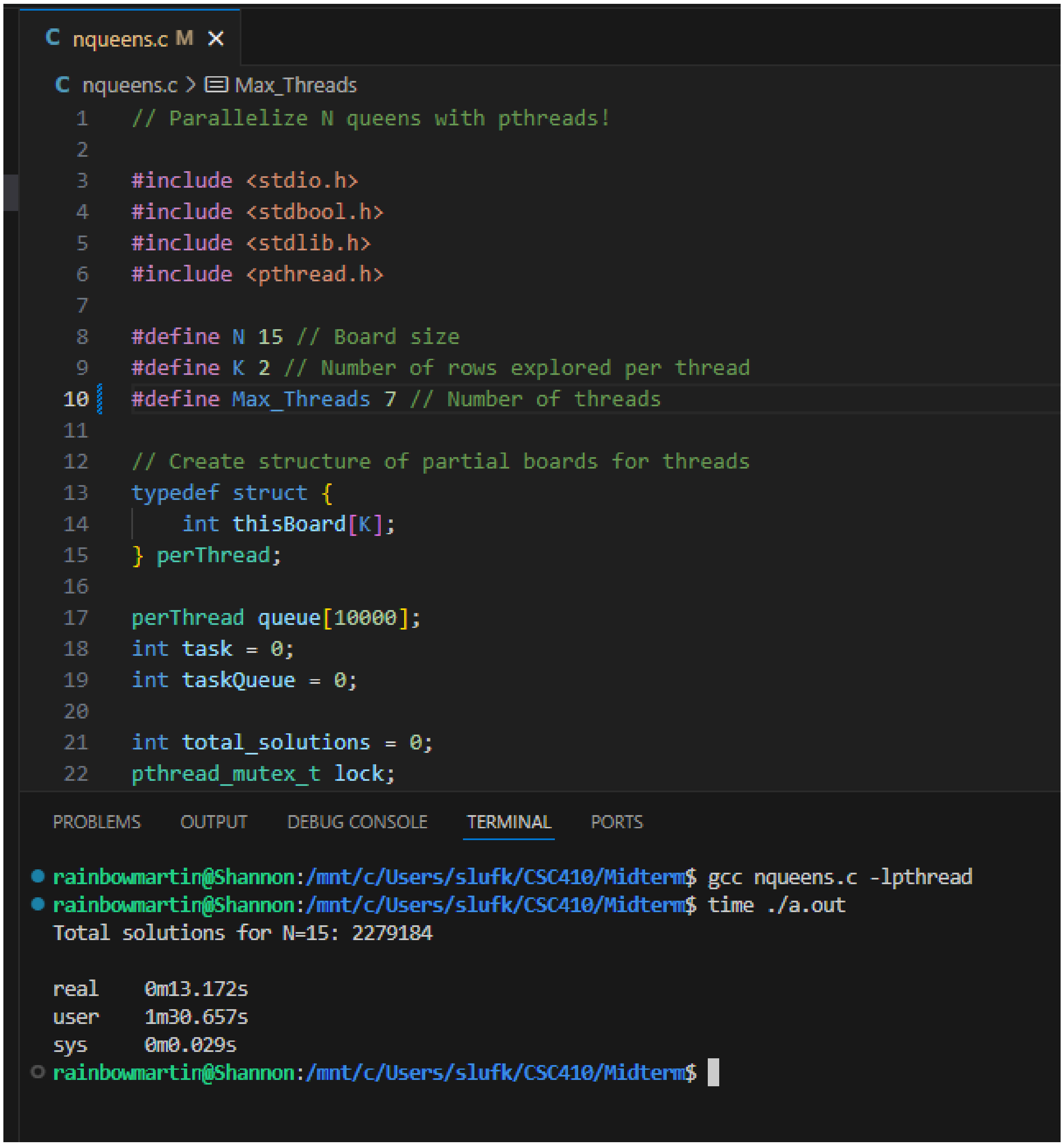
nqueens.c > Max_Threads

```c
// Parallelize N queens with pthreads!

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <pthread.h>

#define N 15 // Board size
#define K 2 // Number of rows explored per thread
#define Max_Threads 5 // Number of threads

// Create structure of partial boards for threads
typedef struct {
    int thisBoard[K];
} perThread;

perThread queue[10000];
int task = 0;
int taskQueue = 0;

int total_solutions = 0;
pthread_mutex_t lock;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ gcc nqueens.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ time ./a.out
Total solutions for N=15: 2279184

real    0m15.069s
user    1m15.143s
sys     0m0.005s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ 
```

Pthread - 7 threads



nqueens.c M ×

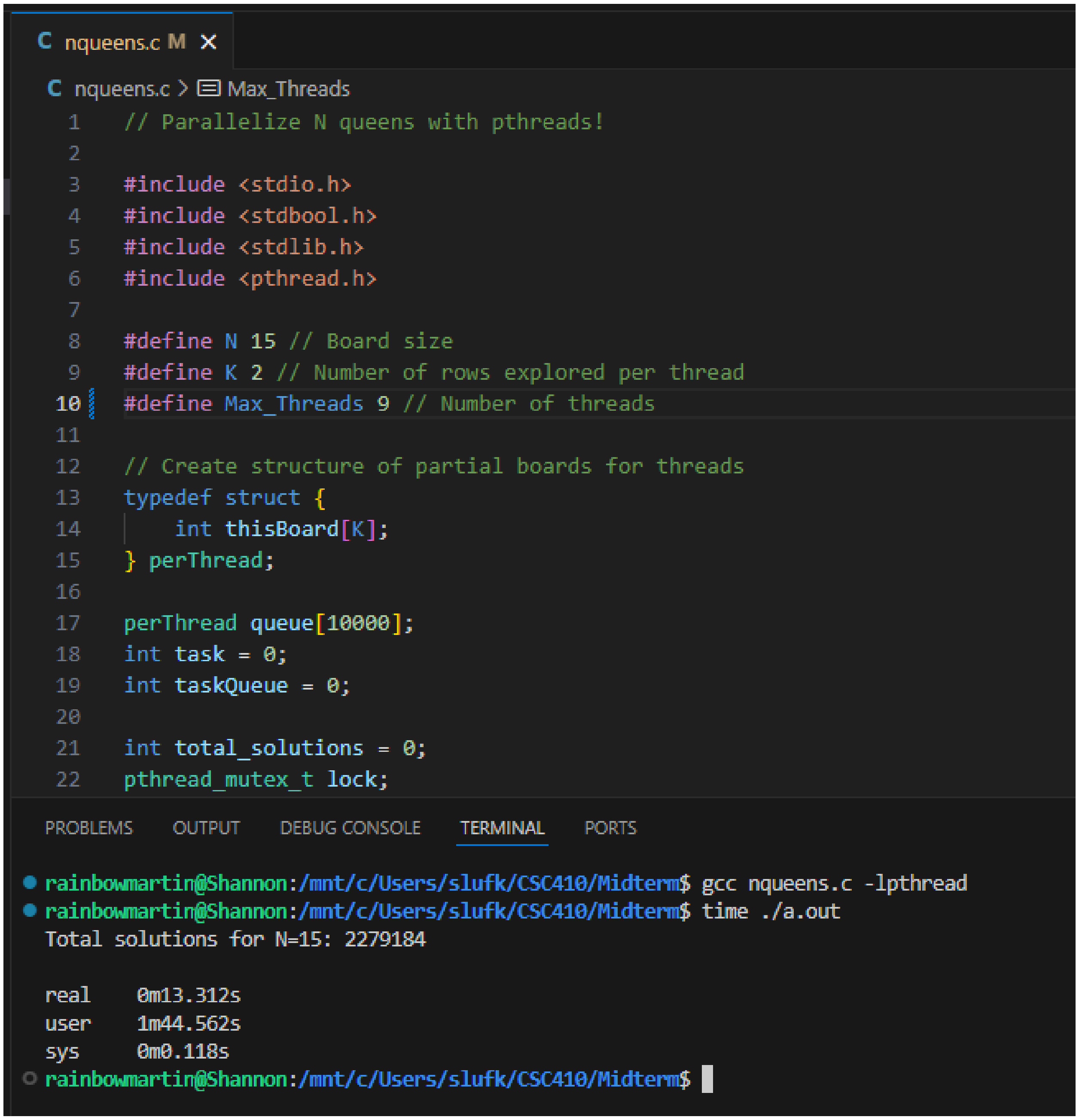
nqueens.c > Max_Threads

```c
// Parallelize N queens with pthreads!

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <pthread.h>

#define N 15 // Board size
#define K 2 // Number of rows explored per thread
#define Max_Threads 7 // Number of threads

// Create structure of partial boards for threads
typedef struct {
    int thisBoard[K];
} perThread;

perThread queue[10000];
int task = 0;
int taskQueue = 0;

int total_solutions = 0;
pthread_mutex_t lock;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ gcc nqueens.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ time ./a.out
Total solutions for N=15: 2279184

real    0m13.172s
user    1m30.657s
sys     0m0.029s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$
```

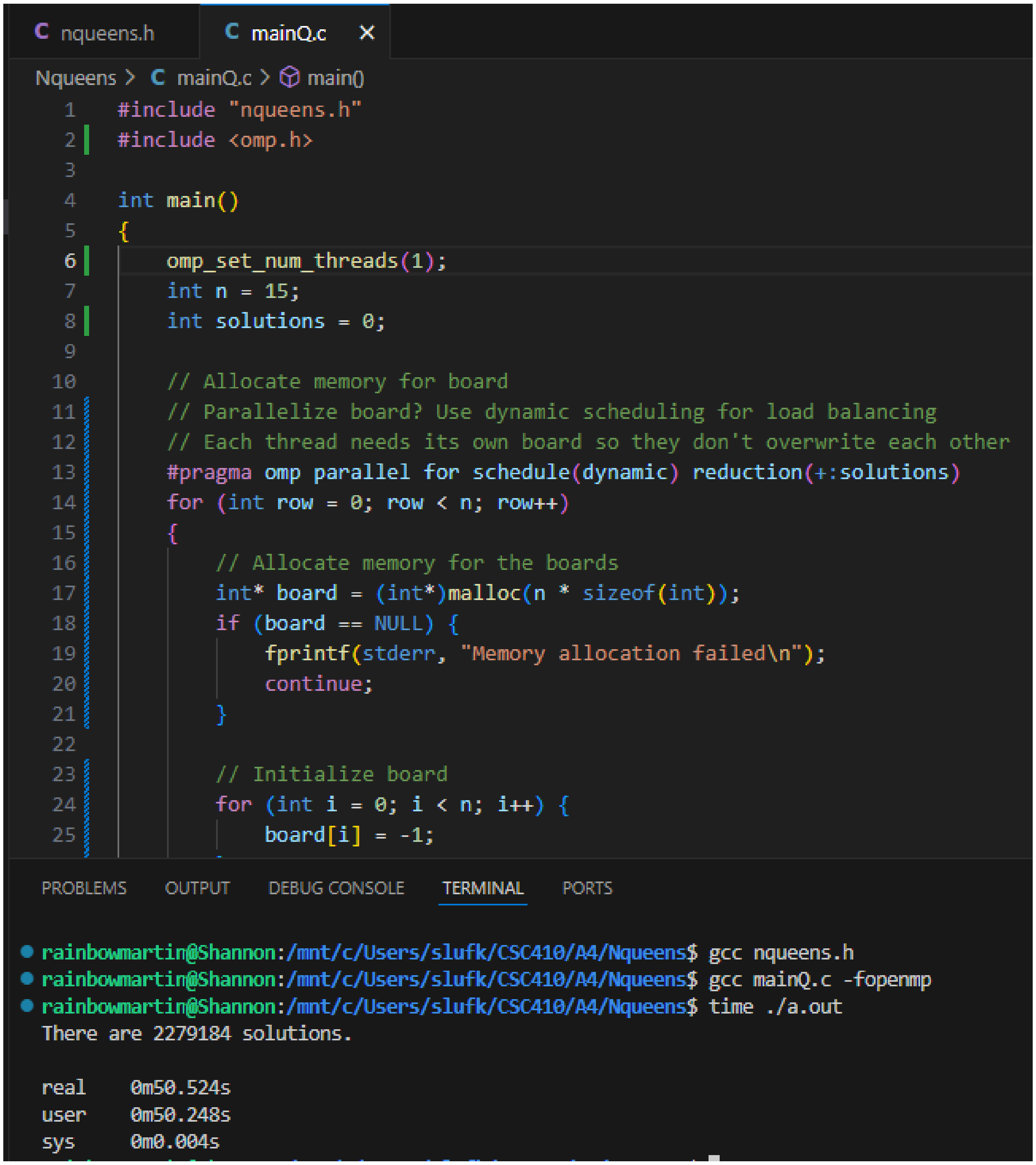
Pthread - 9 threads



```c
// Parallelize N queens with pthreads!

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <pthread.h>

#define N 15 // Board size
#define K 2 // Number of rows explored per thread
#define Max_Threads 9 // Number of threads

// Create structure of partial boards for threads
typedef struct {
    int thisBoard[K];
} perThread;

perThread queue[10000];
int task = 0;
int taskQueue = 0;

int total_solutions = 0;
pthread_mutex_t lock;
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ gcc nqueens.c -lpthread
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$ time ./a.out
Total solutions for N=15: 2279184

real    0m13.312s
user    1m44.562s
sys     0m0.118s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/Midterm$
```

Omp – 1 thread



nqueens.h   mainQ.c

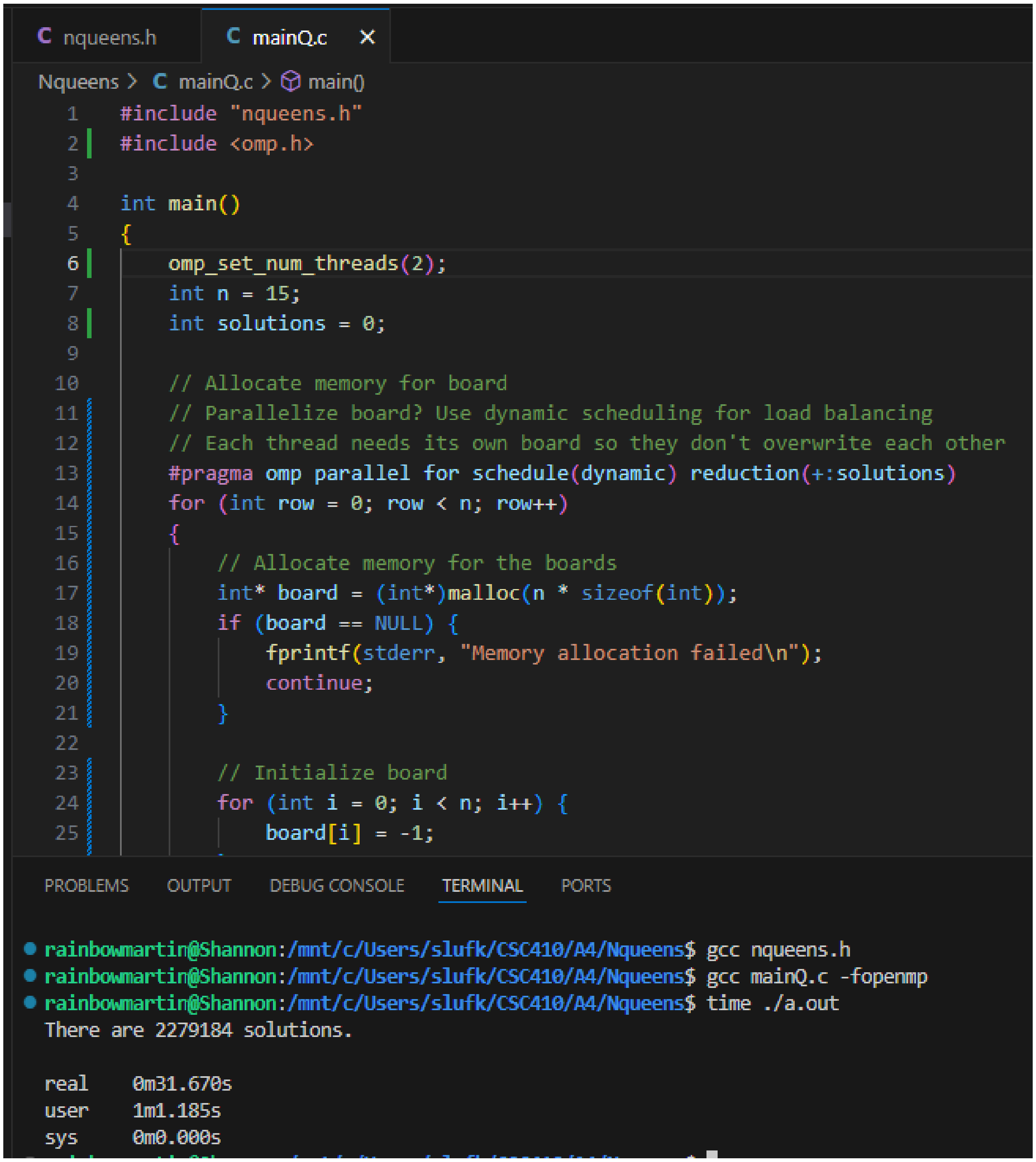
Nqueens > mainQ.c > main()

```c
#include "nqueens.h"
#include <omp.h>

int main()
{
    omp_set_num_threads(1);
    int n = 15;
    int solutions = 0;

    // Allocate memory for board
    // Parallelize board? Use dynamic scheduling for load balancing
    // Each thread needs its own board so they don't overwrite each other
    #pragma omp parallel for schedule(dynamic) reduction(+:solutions)
    for (int row = 0; row < n; row++)
    {
        // Allocate memory for the boards
        int* board = (int*)malloc(n * sizeof(int));
        if (board == NULL) {
            fprintf(stderr, "Memory allocation failed\n");
            continue;
        }

        // Initialize board
        for (int i = 0; i < n; i++) {
            board[i] = -1;
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc nqueens.h
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc mainQ.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ time ./a.out
There are 2279184 solutions.

real    0m50.524s
user    0m50.248s
sys     0m0.004s
```

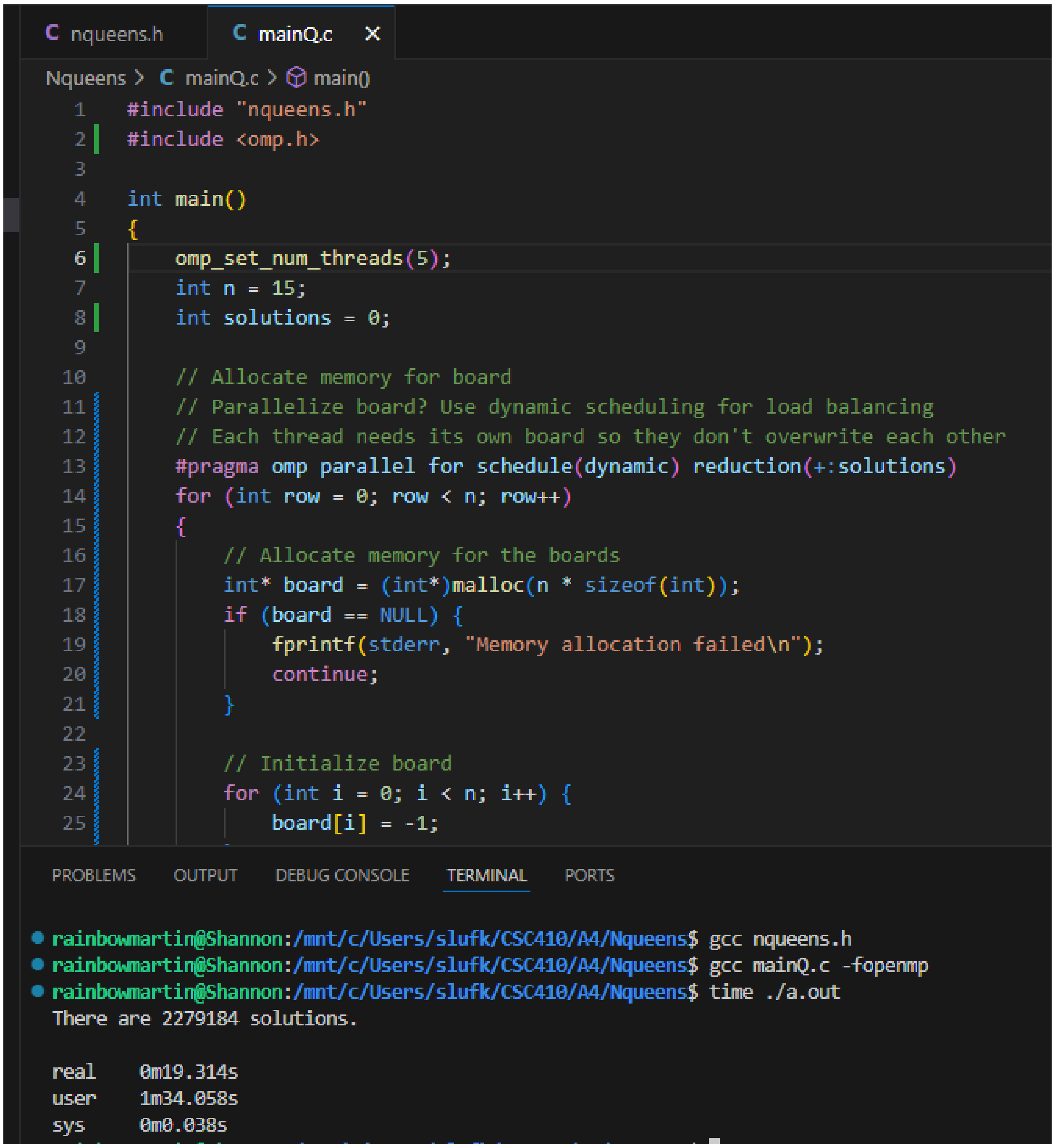
Omp – 2 threads



```c
#include "nqueens.h"
#include <omp.h>

int main()
{
    omp_set_num_threads(2);
    int n = 15;
    int solutions = 0;

    // Allocate memory for board
    // Parallelize board? Use dynamic scheduling for load balancing
    // Each thread needs its own board so they don't overwrite each other
    #pragma omp parallel for schedule(dynamic) reduction(+:solutions)
    for (int row = 0; row < n; row++)
    {
        // Allocate memory for the boards
        int* board = (int*)malloc(n * sizeof(int));
        if (board == NULL) {
            fprintf(stderr, "Memory allocation failed\n");
            continue;
        }

        // Initialize board
        for (int i = 0; i < n; i++) {
            board[i] = -1;
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc nqueens.h
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc mainQ.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ time ./a.out
There are 2279184 solutions.

real    0m31.670s
user    1m1.185s
sys     0m0.000s
```

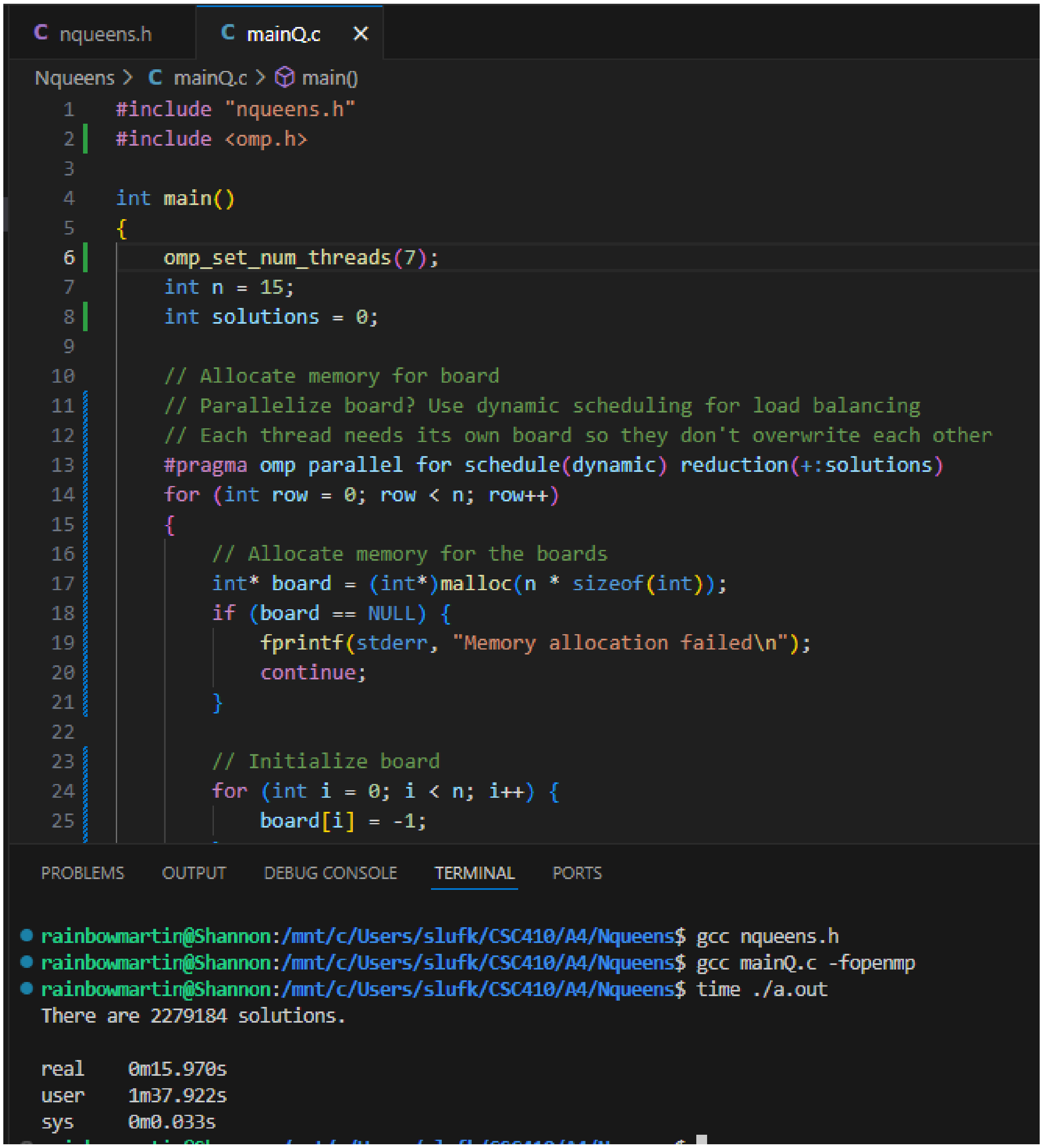
Omp – 5 threads



```c
#include "nqueens.h"
#include <omp.h>

int main()
{
    omp_set_num_threads(5);
    int n = 15;
    int solutions = 0;

    // Allocate memory for board
    // Parallelize board? Use dynamic scheduling for load balancing
    // Each thread needs its own board so they don't overwrite each other
    #pragma omp parallel for schedule(dynamic) reduction(+:solutions)
    for (int row = 0; row < n; row++)
    {
        // Allocate memory for the boards
        int* board = (int*)malloc(n * sizeof(int));
        if (board == NULL) {
            fprintf(stderr, "Memory allocation failed\n");
            continue;
        }

        // Initialize board
        for (int i = 0; i < n; i++) {
            board[i] = -1;
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc nqueens.h
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc mainQ.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ time ./a.out
There are 2279184 solutions.

real    0m19.314s
user    1m34.058s
sys     0m0.038s
```

Omp – 7 threads



nqueens.h   mainQ.c

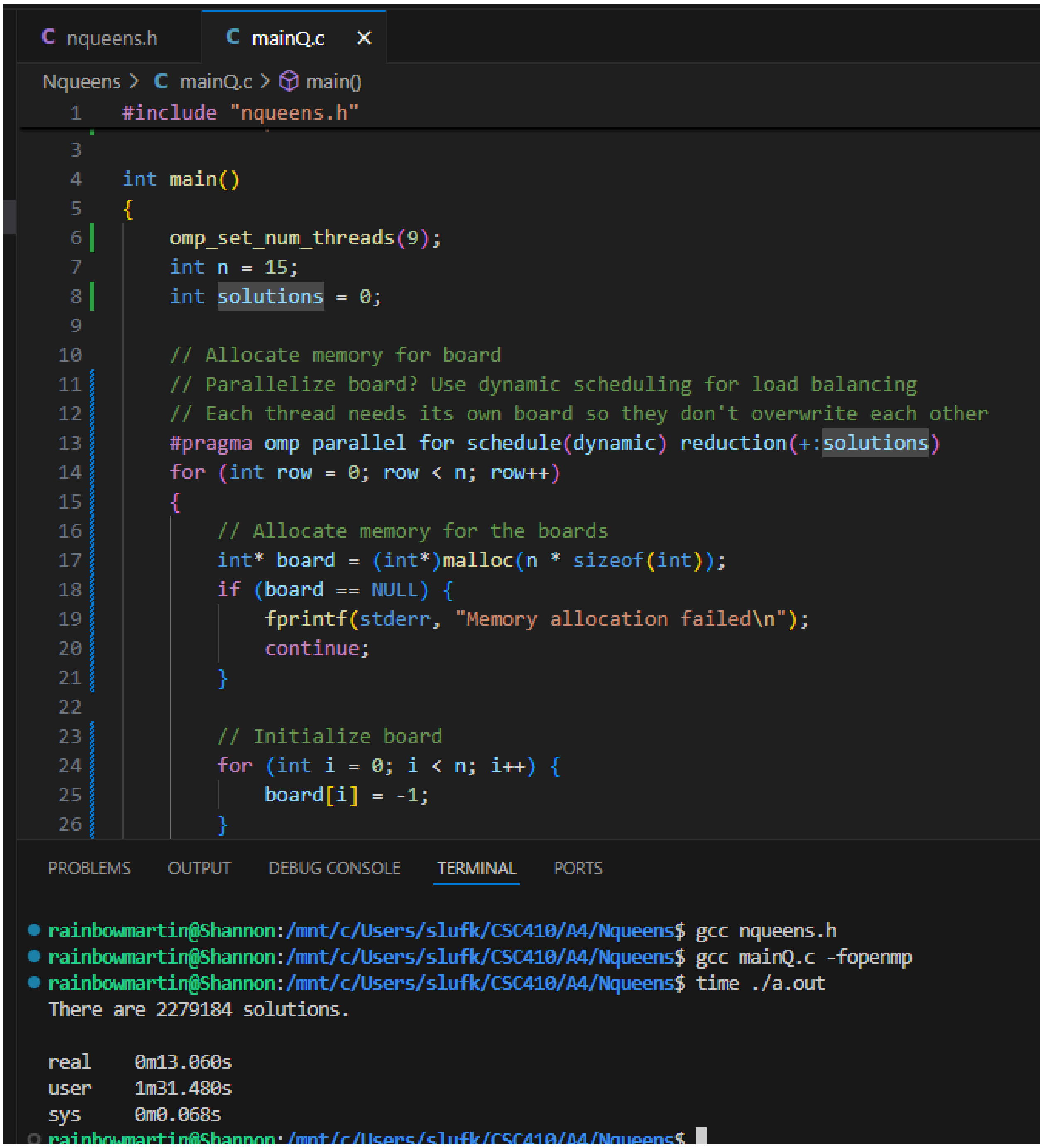
Nqueens > mainQ.c > main()

```c
#include "nqueens.h"
#include <omp.h>

int main()
{
    omp_set_num_threads(7);
    int n = 15;
    int solutions = 0;

    // Allocate memory for board
    // Parallelize board? Use dynamic scheduling for load balancing
    // Each thread needs its own board so they don't overwrite each other
    #pragma omp parallel for schedule(dynamic) reduction(+:solutions)
    for (int row = 0; row < n; row++)
    {
        // Allocate memory for the boards
        int* board = (int*)malloc(n * sizeof(int));
        if (board == NULL) {
            fprintf(stderr, "Memory allocation failed\n");
            continue;
        }

        // Initialize board
        for (int i = 0; i < n; i++) {
            board[i] = -1;
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc nqueens.h
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc mainQ.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ time ./a.out
There are 2279184 solutions.

real    0m15.970s
user    1m37.922s
sys     0m0.033s
```

Omp – 9 threads



```c
#include "nqueens.h"

int main()
{
    omp_set_num_threads(9);
    int n = 15;
    int solutions = 0;

    // Allocate memory for board
    // Parallelize board? Use dynamic scheduling for load balancing
    // Each thread needs its own board so they don't overwrite each other
    #pragma omp parallel for schedule(dynamic) reduction(+:solutions)
    for (int row = 0; row < n; row++)
    {
        // Allocate memory for the boards
        int* board = (int*)malloc(n * sizeof(int));
        if (board == NULL) {
            fprintf(stderr, "Memory allocation failed\n");
            continue;
        }

        // Initialize board
        for (int i = 0; i < n; i++) {
            board[i] = -1;
        }
```

```
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc nqueens.h
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ gcc mainQ.c -fopenmp
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$ time ./a.out
There are 2279184 solutions.

real    0m13.060s
user    1m31.480s
sys     0m0.068s
rainbowmartin@Shannon:/mnt/c/Users/slufk/CSC410/A4/Nqueens$
```