GEBZE TECHNICAL UNIVERSITY

COMPUTER SCIENCE ENGINEERING DEPARTMENT

CSE 344- SYSTEMS PROGRAMMING

HW-5 REPORT

ESRA NUR ARICAN

161044028

OBJECTIVE

In the final homework, our goal was to read 2^n x 2^n characters from the files given by the user, convert them to ascii values and create two square matrixes, then use these matrices with the help of threads, find the product matrices and create the fourier transform matrix. Generally the idea is to use POSIX threads to parallelize couple of simple mathematical tasks.

SOLUTION

First, I created necesarry variables such as signal flag for SIGINT signal, mutex and conditions variables for threads, 2D arrays for matrixes.

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

pthread_mutex_t mutex;
pthread_cond_t conditionVar;
int arrived=0;
int m;
pthread_t *threadSig=NULL;
sig_atomic_t sigintFlag=0;
int size;
int size2;

uint8_t** matrixA;
uint8_t** matrixB;
uint64_t** matrixC;
complexNumber** DFT;
```

Then, function declarations were made before main function.

```
//function declarations
void print_error(char error_message[]);
void convert(uint8_t** matrix, char* input, int number);
void print(uint8_t** matrix, int number);
void print_string(char string[]);
void* threadFunction(void * ind);
void printMatrix64(uint64_t **M, int side);
void printDFT(complexNumber** matrix ,size_t size);
void freeResources(int size);
void signalHandler(int sig);
```

Main function: In the main function, to calculate total time spend, ı put start and end times at the beginning and the end. Necesarry commandline arguments are assigned to variables with getopt() funciton.

After assigning variables, ı made some controls about arguments. n should be bigger than 2, m should be an even number. If user enters missing or more arguments, program prints usage rules. Also, there is a case that m might be bigger number than the column number of matrixes. For example, our column number can be 16 and user can enter 100 m number as 100 threads. Since our aim is to calculate columns with threads, i made m=column number in this case. Because we need maximum 16 threads in this case, every thread

calculates 1 column. Remaining 100-16=84 threads can not calculate any column.

```
//- n > 2 (integer) -m:even number
if(n<=2){
    errno=EINVAL;
    print_error ("n must be bigger than 2!\n");
    exit (EXIT_FAILURE);
}

if(m%2 !=0){
    errno=EINVAL;
    print_error ("m must be even number!\n");
    exit (EXIT_FAILURE);
}

if((optind!=11)){
    errno=EINVAL;
    print_error("Wrong or missing commandline arguments! You should enter:/hw5 -i filePath1 -j filePath2 -o output -n 4 -m 2\n");
    exit(EXIT_FAILURE);
}

//reading two input files into memory
size= pow(2,n);
size2=size*size;

// if number of threads given by user is bigger than column number, make them as column number and every thread calculates 1 column
if(m>size){
    m=size;
}
```

After argument check, i created signal handler and open input files

```
    // signal handling
    struct sigaction sa = {0};
    sa.sa_handler = signalHandler;
    sa.sa_flags = 0;
    sigemptyset(&sa.sa_mask);
    sigaction(SIGINT, &sa, NULL);

    // Open necesarry files
    int fdInput1,fdInput2,fdOut;
    if ((fdInput1 = open(inputPath1, O_RDONLY)) == -1){
        perror("Failed to open input file in main.\n");
        exit(EXIT_FAILURE);
    }

    if ((fdInput2 = open(inputPath2, O_RDONLY)) == -1){
        perror("Failed to open input file in main.\n");
        exit(EXIT_FAILURE);
    }

    if ((fdOut = open(outputPath, O_WRONLY | O_CREAT | O_TRUNC, 0644)) == -1){
        perror("Failed to open output file in main.\n");
        exit(EXIT_FAILURE);
    }
```

Later, i read given 2 input files and created A and B matrixes. Fırst, all characters readen in one dimensional array and then created 2D array. Arrays are dynamically allocated and free statements are used after using them to avoid memory leaks. If given input file has insufficient characters, program gives error and exits. Convert() converts given char to equivalent Ascii integer.

```c
    int bytesread = 0;

    while(((bytesread = read(fdInput1, inputA, size2)) == -1) && (errno == EINTR));


    // If there are not sufficient characters in the files then it will print an error message and exit gracefully.
    if(bytesread != size2){

        fprintf (stderr, "\nerrno = %d: %s there are not sufficient characters in the inputPathA file\n\n", errno, strerror (errno));
        exit(1);
    }

    // These characters will be converted into its ASCII code integer equivalent in 2D array.
    matrixA = (uint8_t **)calloc(size, sizeof(uint8_t *));
    for(int i = 0; i < size; i++){
        matrixA[i] = (uint8_t *)calloc(size, sizeof(uint8_t));
        memset(matrixA[i],0,sizeof(uint8_t));
    }
    // Convert operation
    convert(matrixA, inputA, size);
    free(inputA);
```

After reading and creating A and B matrices, C matris as multiplication result and DFT array which is struct type of complex number are allocated dynamically and created.

Since we dont need input files after that, input files are closed.

In the main function, other important parts are about threads. To be able sycnhronize threads and wait all threads to first part and then move to the second part, we need to use mutex and condition variables. So i made mutex and conditions variable initialization settings from our textbook.

After initializing the mutex and condition variables, I created the number of threads given by the user with pthread_create and then I did the join operation.

The function I sent while creating the threads is the function that does the mathematical operations. Also, since I will use the thread index in both print statements and calculations, I sent this number to the function.

```c
    for(i = 0; i < m; i++) {
        threadIndexs[i]=i;

        if(pthread_create(&m_threads[i], NULL, threadFunction, &threadIndexs[i]) != 0) {
          perror("ERROR pthread_create ");
          freeResources(size);
          close(fdInput1);close(fdInput2);close(fdOut);
          exit(EXIT_FAILURE);
        }
    }

    threadSig=m_threads;

    if(sigintFlag==1){
        freeResources(size);
        close(fdInput1);close(fdInput2);close(fdOut);
        exit(EXIT_FAILURE);
    }

    for(i = 0; i < m; i++) {
        if(pthread_join(m_threads[i],NULL) != 0) {
          perror("ERROR pthread_join");
          freeResources(size);
          exit(EXIT_FAILURE);
        }
```

The main function takes the resulting DFT matrix after the threads have finished their work and prints this matrix to the output file in csv format. It then closes the open files, prints the total time spent on the screen and ends.

Also, the if condition, which is often used in the main function, was written to control the ctrl-c signal. If the flag holding the SIGINT signal is changed, all files are closed and the memory is freed. Since it can be inconvenient to do too many operations in the signal handler, I only check the signal in the signal handler and do the cleaning in the if condition.

```
    // Write result DFT matrix to output file
    for (int i = 0; i<size; i++){
        for (int j = 0; j< size; j++){

            sprintf(message," %.3f + %.3f i ,",DFT[i][j].real,DFT[i][j].imaginary);
            write(fdOut,message,strlen(message));

        }
    write(fdOut,"\n",sizeof("\n"));
    }

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    if(sigintFlag==1){
        freeResources(size);
        close(fdInput1);close(fdInput2);close(fdOut);
        exit(EXIT_FAILURE);
    }

    sprintf(message,"[%.19s] The process has written the output file. The total time spent is %f seconds.\n",ctime(&t),cpu_time_used);
    print_string(message);


    close(fdOut);

    freeResources(size);
    pthread_exit(0);
```

void* threadFunction(void * ind): In this function, after each thread is created, it is provided to come and perform its own mathematical operations. First of all, the first thing I noticed was that each thread will create the C matrix by calculating a certain number of columns. Afterwards, the generation of the C matrix had to be finished before all threads could calculate the DFT coefficient. In other words, no thread should pass to the second part before all threads have finished the first part.

In the method I use for the synchronization barrier, I lock the mutex first. If all threads reached here, I check with if, then I unlock. There are matrix C calculation operations on this piece of code, and DFT calculation operations below it.

```
    //synchronization barrier
    pthread_mutex_lock(&mutex);
    ++arrived;
    while(arrived < m){
        pthread_cond_wait(&conditionVar,&mutex);
    }

    pthread_cond_broadcast(&conditionVar);
    pthread_mutex_unlock(&mutex);

    // c artık hazır
```

To calculate C matrix, i used simple for loops and made matrix multiplication.

The only issue is to decide every thread's start and end indexes to calculate correct columns. Start index is 2^n/m * thread index and end index is start index + 2^n/m .

```
    clock_t start1, end1;
    double cpu_time_used1;

     start1 = clock();
     /* Do the work. */
     // her thread:a b nin hesaplayıp c ye atılması
    for (int i=0; i < size; i++) {
        for (int j=startInner; j <endInner; j++) {

            for (int k = 0; k < size; k++) {
                matrixC[i][j] += matrixA[i][k] * matrixB[k][j];
            }
        }
    }
    end1 = clock();
    cpu_time_used1 = ((double) (end1 - start1)) / CLOCKS_PER_SEC;


    sprintf(message,"[%.19s] Thread %d has reached the rendezvous point in %f second.\n",ctime(&t),index,cpu_time_used1);
    print_string(message);
```

After all threads reach the rendezvous point, the fourier transform process is started. Each thread does its calculations by using the column that it has to do in the C matrix that is formed. Here again, start end indexes are used. While performing the operations according to the formula in the Fourier transform, I only used the math.h library, the complex.h library was not used. Since our result is a complex number, the DFT array consists of a struct called complex number. I kept real and imaginary parts of float type in this struct.

```
// calculating fourier transform
for(int i=startInner;i<endInner;i++){
    for(int j=0;j<size;j++){
        float ak=0;
        float bk=0;
        for(int ii=0;ii<size;ii++){
            for(int jj=0;jj<size;jj++){

                float x=-2.0*M_PI*i*ii/(float)size;
                float y=-2.0*M_PI*j*jj/(float)size;
                ak+=matrixC[ii][jj]*cos(x+y);
                bk+=matrixC[ii][jj]*1.0*sin(x+y);
            }
        }
        DFT[i][j].real=ak;
        DFT[i][j].imaginary=bk;

    }
}
```

I would like to mention another point that I paid attention to here. Suppose we have n numbers of 4, so we have a matrix of 16 columns. If the number m is entered as 10, each thread cannot calculate an integer column. In this case, each thread calculated 1 column and the last thread calculated the remaining columns. Similarly, if the number of threads is 6, 16/6=2 each thread calculates exactly 2 columns, while the last thread calculates the remaining columns. Like this, in cases where m is not exactly divided by the number of columns, I arrange my start end index variables at the beginning of the function in order to perform the calculation correctly. For the case I gave for example, if the number of columns is 16, and n gets numbers like 4, 8, all threads do their normal calculations as integer number of columns. In any case, the resulting matrix is correct.

```
if(size%m != 0 && index==m-1){
    startInner=(size/m)*index;
    endInner=size;
}
```

After calculating DFT, all threads prints their calculation times and finishes.

Now, i will explain helper functions that i wrote for the homework;

void print_string(char string[]): Prints given string to STDOUT

void print_error(char error_message[]): Prints error to STDERR

void convert(uint8_t matrix, char* input, int number):** Converts characters to its equivalent ASCII code integer value.

void print(uint8_t matrix, int number):** Prints A and B matrixes to console

void printMatrix64(uint64_t **M, int side): Prints C matrix to console

void printDFT(complexNumber matrix,size_t size):** Prints DFT array to console, since DFT array is type of complex number, it has its own print method.

void freeResources(int size): Frees all memories allocated dynamically, matrixA, matrixB, matrixC, DFT.

void signalHandler(int sig): Signal handler for SIGINT signal.

COMPILING and OUTPUTS

Compiling with wall, by makefile no warnings and errors:

```
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$ make
gcc -c -o hw5.o hw5.c -Wall -pedantic-errors -std=gnu99 -pthread -lrt -lm
gcc -o hw5 hw5.o -Wall -pedantic-errors -std=gnu99 -pthread -lrt -lm
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$
```

Running code with invalid commandline arguments:

```
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$  ./hw5 -i a.txt -j b.txt -o sout.csv -n 3
Wrong or missing commandline arguments! You should enter:/hw5 -i filePath1 -j filePath2 -o output -n 4 -m 2
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$  ./hw5 -j b.txt -o sout.csv -n 3
Wrong or missing commandline arguments! You should enter:/hw5 -i filePath1 -j filePath2 -o output -n 4 -m 2
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$ ./hw5 -i 1.txt -j none.txt -o sout.csv -n 5 -m 8
Failed to open input file in main.
: No such file or directory
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$
```

Running with valid inputs and valgrind with input files:

Input file 1 consists:
sadgsgsaGsadGsdgsdGa'!^4!'^5%'^5!'352saDGsdag2ttewrrtcwetwb545ba

Input file 2 consists:

asdfdsgasgsdgsadgsadgsdagasdgsdgasdgasdg324ipo5432ijh5t234io5fc1

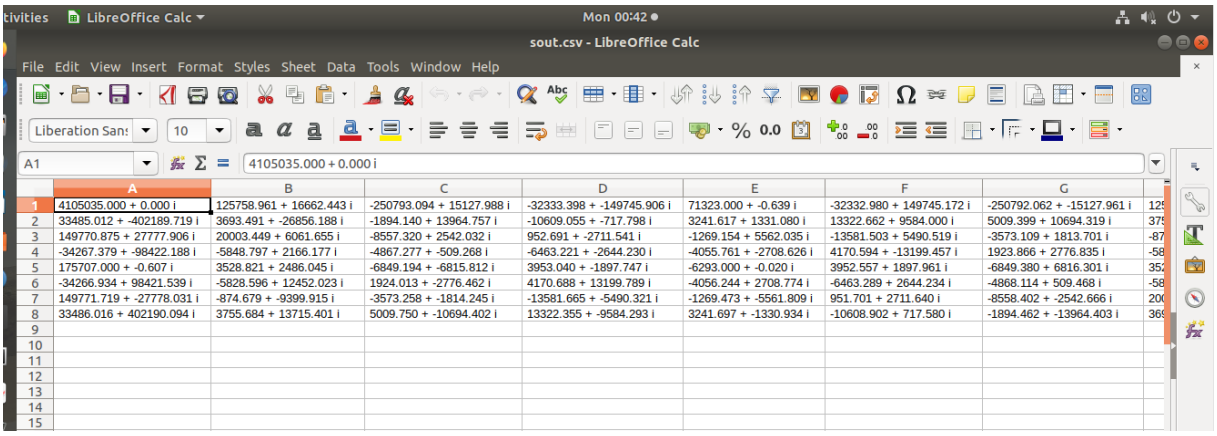
First i entered n number bigger than the file contents, this should be error and program should terminate:

```
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$ valgrind --leak-check=full --show-leak-kinds=all ./hw5 -i a.txt -j b.txt -o output1.csv -n 5 -m 4

==11802== Memcheck, a memory error detector
==11802== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==11802== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==11802== Command: ./hw5 -i a.txt -j b.txt -o output1.csv -n 5 -m 4
==11802==

errno = 0: Success there are not sufficient characters in the inputPathA file
```

Now, entering n = 3 since file includes 64 characters:

```
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$ valgrind --leak-check=full --show-leak-kinds=all ./hw5 -i a.txt -j b.txt -o sout.csv -n 3 -m 4
==11819== Memcheck, a memory error detector
==11819== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==11819== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==11819== Command: ./hw5 -i a.txt -j b.txt -o sout.csv -n 3 -m 4
==11819==
[Mon May 23 00:40:16] Two matrices of size 8x8 have been read. The number of threads is 4
[Mon May 23 00:40:16] Thread 1 has reached the rendezvous point in 0.000014 second.
[Mon May 23 00:40:16] Thread 0 has reached the rendezvous point in 0.000910 second.
[Mon May 23 00:40:16] Thread 2 has reached the rendezvous point in 0.000014 second.
[Mon May 23 00:40:16] Thread 3 has reached the rendezvous point in 0.000015 second.
[Mon May 23 00:40:16] Thread 1 is advancing to the second part
[Mon May 23 00:40:16] Thread 0 is advancing to the second part
[Mon May 23 00:40:16] Thread 2 is advancing to the second part
[Mon May 23 00:40:16] Thread 3 is advancing to the second part
[Mon May 23 00:40:16] Thread 2 has finished the second part in 0.024380 second.
[Mon May 23 00:40:16] Thread 1 has finished the second part in 0.004215 second.
[Mon May 23 00:40:16] Thread 0 has finished the second part in 0.004008 second.
[Mon May 23 00:40:16] Thread 3 has finished the second part in 0.004034 second.
[Mon May 23 00:40:16] The process has written the output file. The total time spent is 0.314467 seconds.
==11819==
==11819== HEAP SUMMARY:
==11819==     in use at exit: 0 bytes in 0 blocks
==11819==   total heap usage: 71 allocs, 71 frees, 10,644 bytes allocated
==11819==
==11819== All heap blocks were freed -- no leaks are possible
==11819==
==11819== For counts of detected and suppressed errors, rerun with: -v
==11819== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$
```

Output csv file:



	A	B	C	D	E	F	G
1	4105035.000 + 0.000 i	125758.961 + 16662.443 i	-250793.094 + 15127.988 i	-32333.398 + -149745.906 i	71323.000 + -0.639 i	-32332.980 + 149745.172 i	-250792.062 + -15127.961 i
2	33485.012 + -402189.719 i	3693.491 + -26856.188 i	-1894.140 + 13964.757 i	-10609.055 + -717.798 i	3241.617 + 1331.080 i	13322.662 + 9584.000 i	5009.399 + 10694.319 i
3	149770.875 + 27777.906 i	20003.449 + 6061.655 i	-8557.320 + 2542.032 i	952.691 + -2711.541 i	-1269.154 + 5562.035 i	-13581.503 + 5490.519 i	-3573.109 + 1813.701 i
4	-34267.379 + -98422.188 i	-5848.797 + 2166.177 i	-4867.277 + -509.268 i	-6463.221 + -2644.230 i	-4055.761 + -2708.626 i	4170.594 + -13199.457 i	1923.866 + 2776.835 i
5	175707.000 + -0.607 i	3528.821 + 2486.045 i	-6849.194 + -6815.812 i	3953.040 + -1897.747 i	-6293.000 + -0.020 i	3952.557 + 1897.961 i	-6849.380 + 6816.301 i
6	-34266.934 + 98421.539 i	-5828.596 + 12452.023 i	1924.013 + -2776.462 i	4170.688 + 13199.789 i	-4056.244 + 2708.774 i	-6463.289 + 2644.234 i	-4868.114 + 509.468 i
7	149771.719 + -27778.031 i	-874.679 + -9399.915 i	-3573.258 + -1814.245 i	-13581.665 + -5490.321 i	-1269.473 + -5561.809 i	951.701 + 2711.640 i	-8558.402 + -2542.666 i
8	33486.016 + 402190.094 i	3755.684 + 13715.401 i	5009.750 + -10694.402 i	13322.355 + -9584.293 i	3241.697 + -1330.934 i	-10608.902 + 717.580 i	-1894.462 + -13964.403 i

Trying same files with different m number:

```
[Mon May 23 00:43:06] Two matrices of size 8x8 have been read. The number of threads is 8
[Mon May 23 00:43:06] Thread 1 has reached the rendezvous point in 0.000009 second.
[Mon May 23 00:43:06] Thread 0 has reached the rendezvous point in 0.000850 second.
[Mon May 23 00:43:06] Thread 2 has reached the rendezvous point in 0.000010 second.
[Mon May 23 00:43:06] Thread 3 has reached the rendezvous point in 0.000010 second.
[Mon May 23 00:43:06] Thread 4 has reached the rendezvous point in 0.000039 second.
[Mon May 23 00:43:07] Thread 5 has reached the rendezvous point in 0.000034 second.
[Mon May 23 00:43:07] Thread 6 has reached the rendezvous point in 0.000034 second.
[Mon May 23 00:43:07] Thread 7 has reached the rendezvous point in 0.000011 second.
[Mon May 23 00:43:06] Thread 1 is advancing to the second part
[Mon May 23 00:43:06] Thread 0 is advancing to the second part
[Mon May 23 00:43:06] Thread 2 is advancing to the second part
[Mon May 23 00:43:06] Thread 3 is advancing to the second part
[Mon May 23 00:43:06] Thread 4 is advancing to the second part
[Mon May 23 00:43:07] Thread 5 is advancing to the second part
[Mon May 23 00:43:07] Thread 6 is advancing to the second part
[Mon May 23 00:43:07] Thread 7 is advancing to the second part
[Mon May 23 00:43:07] Thread 6 has finished the second part in 0.001888 second.
[Mon May 23 00:43:07] Thread 5 has finished the second part in 0.001890 second.
[Mon May 23 00:43:06] Thread 2 has finished the second part in 0.001924 second.
[Mon May 23 00:43:06] Thread 0 has finished the second part in 0.001570 second.
[Mon May 23 00:43:07] Thread 7 has finished the second part in 0.002445 second.
[Mon May 23 00:43:06] Thread 1 has finished the second part in 0.002853 second.
[Mon May 23 00:43:06] Thread 4 has finished the second part in 0.023146 second.
[Mon May 23 00:43:06] Thread 3 has finished the second part in 0.003077 second.
[Mon May 23 00:43:06] The process has written the output file. The total time spent is 0.508673 seconds.
==11890==
==11890== HEAP SUMMARY:
==11890==     in use at exit: 0 bytes in 0 blocks
==11890==   total heap usage: 87 allocs, 87 frees, 11,912 bytes allocated
==11890==
==11890== All heap blocks were freed -- no leaks are possible
==11890==
==11890== For counts of detected and suppressed errors, rerun with: -v
==11890== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$
```

Output file producing same result:

A1 | 4105035.000 + 0.000 i

	A	B	C	D	E	F	G	
1	4105035.000 + 0.000 i	125758.961 + 16662.443 i	-250793.094 + 15127.988 i	-32333.398 + -149745.906 i	71323.000 + -0.639 i	-32332.980 + 149745.172 i	-250792.062 + -15127.961 i	125
2	33485.012 + -402189.719 i	3693.491 + -26856.188 i	-1894.140 + 13964.757 i	-10609.055 + -717.798 i	3241.617 + 1331.080 i	13322.662 + 9584.000 i	5009.399 + 10694.319 i	375
3	149770.875 + 27777.906 i	20003.449 + 6061.655 i	-8557.320 + 2542.032 i	952.691 + -2711.541 i	-1269.154 + 5562.035 i	-13581.503 + 5490.519 i	-3573.109 + 1813.701 i	-874
4	-34267.379 + -98422.188 i	-5848.797 + 2166.177 i	-4867.277 + -509.268 i	-6463.221 + -2644.230 i	-4055.761 + -2708.626 i	4170.594 + -13199.457 i	1923.866 + 2776.835 i	-582
5	175707.000 + -0.607 i	3528.821 + 2486.045 i	-6849.194 + -6815.812 i	3953.040 + -1897.747 i	-6293.000 + -0.020 i	3952.557 + 1897.961 i	-6849.380 + 6816.301 i	352
6	-34266.934 + 98421.539 i	-5828.596 + 12452.023 i	1924.013 + -2776.462 i	4170.688 + 13199.789 i	-4056.244 + 2708.774 i	-6463.289 + 2644.234 i	-4868.114 + 509.468 i	-584
7	149771.719 + -27778.031 i	-874.679 + -9399.915 i	-3573.258 + -1814.245 i	-13581.665 + -5490.321 i	-1269.473 + -5561.809 i	951.701 + 2711.640 i	-8558.402 + -2542.666 i	200
8	33486.016 + 402190.094 i	3755.684 + 13715.401 i	5009.750 + -10694.402 i	13322.355 + -9584.293 i	3241.697 + -1330.934 i	-10608.902 + 717.580 i	-1894.462 + -13964.403 i	369
9								
10								
11								
12								
13								

Also program is tested for bigger and different files.

No errors and leaks with valgrind.

Finally, as the number of n increases, the computation times increase visibly, and when the number of m increases, I expected the thread times to decrease, but I did not observe a very obvious difference in this part. In some test cases, the time decreased as the number of threads increased, while in some cases it remained almost the same.

Result for bigger n value: n is given 6 in this case and m is given 16;

Since 2^6*2^6 is big number, especially calculations at the second part took really long, but first part completed shorter time like other smaller sizes.

```
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$ valgrind --leak-check=full ./hw5 -i in12.txt -j in12_2.txt -o out.csv -n 6 -m 16
==12136== Memcheck, a memory error detector
==12136== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==12136== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==12136== Command: ./hw5 -i in12.txt -j in12_2.txt -o out.csv -n 6 -m 16
==12136==
[Mon May 23 00:56:45] Two matrices of size 64x64 have been read. The number of threads is 16
[Mon May 23 00:56:46] Thread 1 has reached the rendezvous point in 0.001729 second.
[Mon May 23 00:56:46] Thread 0 has reached the rendezvous point in 0.002043 second.
[Mon May 23 00:56:46] Thread 2 has reached the rendezvous point in 0.002077 second.
[Mon May 23 00:56:46] Thread 3 has reached the rendezvous point in 0.001332 second.
[Mon May 23 00:56:46] Thread 4 has reached the rendezvous point in 0.001657 second.
[Mon May 23 00:56:46] Thread 5 has reached the rendezvous point in 0.001195 second.
[Mon May 23 00:56:46] Thread 6 has reached the rendezvous point in 0.001264 second.
[Mon May 23 00:56:46] Thread 7 has reached the rendezvous point in 0.001405 second.
[Mon May 23 00:56:46] Thread 8 has reached the rendezvous point in 0.001194 second.
[Mon May 23 00:56:46] Thread 9 has reached the rendezvous point in 0.001368 second.
[Mon May 23 00:56:46] Thread 10 has reached the rendezvous point in 0.001694 second.
[Mon May 23 00:56:46] Thread 11 has reached the rendezvous point in 0.001274 second.
[Mon May 23 00:56:46] Thread 12 has reached the rendezvous point in 0.001479 second.
[Mon May 23 00:56:46] Thread 13 has reached the rendezvous point in 0.001661 second.
[Mon May 23 00:56:46] Thread 14 has reached the rendezvous point in 0.001228 second.
[Mon May 23 00:56:46] Thread 15 has reached the rendezvous point in 0.001112 second.
[Mon May 23 00:56:46] Thread 14 is advancing to the second part
[Mon May 23 00:56:46] Thread 9 is advancing to the second part
[Mon May 23 00:56:46] Thread 13 is advancing to the second part
[Mon May 23 00:56:46] Thread 11 is advancing to the second part
[Mon May 23 00:56:46] Thread 7 is advancing to the second part
[Mon May 23 00:56:46] Thread 3 is advancing to the second part
```

```
[Mon May 23 00:56:46] Thread 4 is advancing to the second part
[Mon May 23 00:56:46] Thread 1 is advancing to the second part
[Mon May 23 00:56:46] Thread 15 is advancing to the second part
[Mon May 23 00:56:46] Thread 9 has finished the second part in 8.972382 second.
[Mon May 23 00:56:46] Thread 14 has finished the second part in 8.922938 second.
[Mon May 23 00:56:46] Thread 6 has finished the second part in 8.991481 second.
[Mon May 23 00:56:46] Thread 15 has finished the second part in 8.986116 second.
[Mon May 23 00:56:46] Thread 10 has finished the second part in 10.304963 second.
[Mon May 23 00:56:46] Thread 12 has finished the second part in 10.316375 second.
[Mon May 23 00:56:46] Thread 13 has finished the second part in 9.571176 second.
[Mon May 23 00:56:46] Thread 2 has finished the second part in 9.568822 second.
[Mon May 23 00:56:46] Thread 5 has finished the second part in 9.605920 second.
[Mon May 23 00:56:46] Thread 4 has finished the second part in 9.611817 second.
[Mon May 23 00:56:46] Thread 11 has finished the second part in 9.226439 second.
[Mon May 23 00:56:46] Thread 1 has finished the second part in 9.225261 second.
[Mon May 23 00:56:46] Thread 0 has finished the second part in 8.884680 second.
[Mon May 23 00:56:46] Thread 8 has finished the second part in 8.896226 second.
[Mon May 23 00:56:46] Thread 3 has finished the second part in 9.170684 second.
[Mon May 23 00:56:46] Thread 7 has finished the second part in 9.164243 second.
[Mon May 23 00:56:45] The process has written the output file. The total time spent is 76.197844 seconds.
==12136==
==12136== HEAP SUMMARY:
==12136==     in use at exit: 0 bytes in 0 blocks
==12136==   total heap usage: 343 allocs, 343 frees, 100,912 bytes allocated
==12136==
==12136== All heap blocks were freed -- no leaks are possible
==12136==
==12136== For counts of detected and suppressed errors, rerun with: -v
==12136== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Same input files are tested with 32 threads this time, and total time spend decreased only 3 seconds.

```
[Mon May 23 01:01:11] Thread 31 has finished the second part in 4.427573 second.
[Mon May 23 01:01:10] Thread 24 has finished the second part in 4.475023 second.
[Mon May 23 01:01:10] Thread 21 has finished the second part in 4.255712 second.
[Mon May 23 01:01:10] Thread 23 has finished the second part in 4.255615 second.
[Mon May 23 01:01:10] Thread 22 has finished the second part in 4.393642 second.
[Mon May 23 01:01:10] Thread 17 has finished the second part in 4.369536 second.
[Mon May 23 01:01:09] Thread 3 has finished the second part in 4.183470 second.
[Mon May 23 01:01:10] Thread 18 has finished the second part in 4.183849 second.
[Mon May 23 01:01:09] Thread 12 has finished the second part in 4.166480 second.
[Mon May 23 01:01:10] Thread 19 has finished the second part in 4.159713 second.
[Mon May 23 01:01:09] Thread 11 has finished the second part in 4.201479 second.
[Mon May 23 01:01:09] Thread 13 has finished the second part in 4.203519 second.
[Mon May 23 01:01:09] Thread 10 has finished the second part in 4.612671 second.
[Mon May 23 01:01:10] Thread 20 has finished the second part in 4.618985 second.
[Mon May 23 01:01:09] Thread 9 has finished the second part in 4.279356 second.
[Mon May 23 01:01:09] Thread 0 has finished the second part in 4.290756 second.
[Mon May 23 01:01:10] Thread 14 has finished the second part in 4.477390 second.
[Mon May 23 01:01:09] Thread 2 has finished the second part in 4.483717 second.
[Mon May 23 01:01:09] The process has written the output file. The total time spent is 73.054934 seconds.
==12163==
==12163== HEAP SUMMARY:
==12163==     in use at exit: 0 bytes in 0 blocks
==12163==   total heap usage: 407 allocs, 407 frees, 105,984 bytes allocated
==12163==
==12163== All heap blocks were freed -- no leaks are possible
==12163==
==12163== For counts of detected and suppressed errors, rerun with: -v
==12163== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
esra@ubuntu:~/Desktop/SystProgramming2022/hw5$
```