Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta High Performance Computing Lab
Batch: B4
Practical no – 8

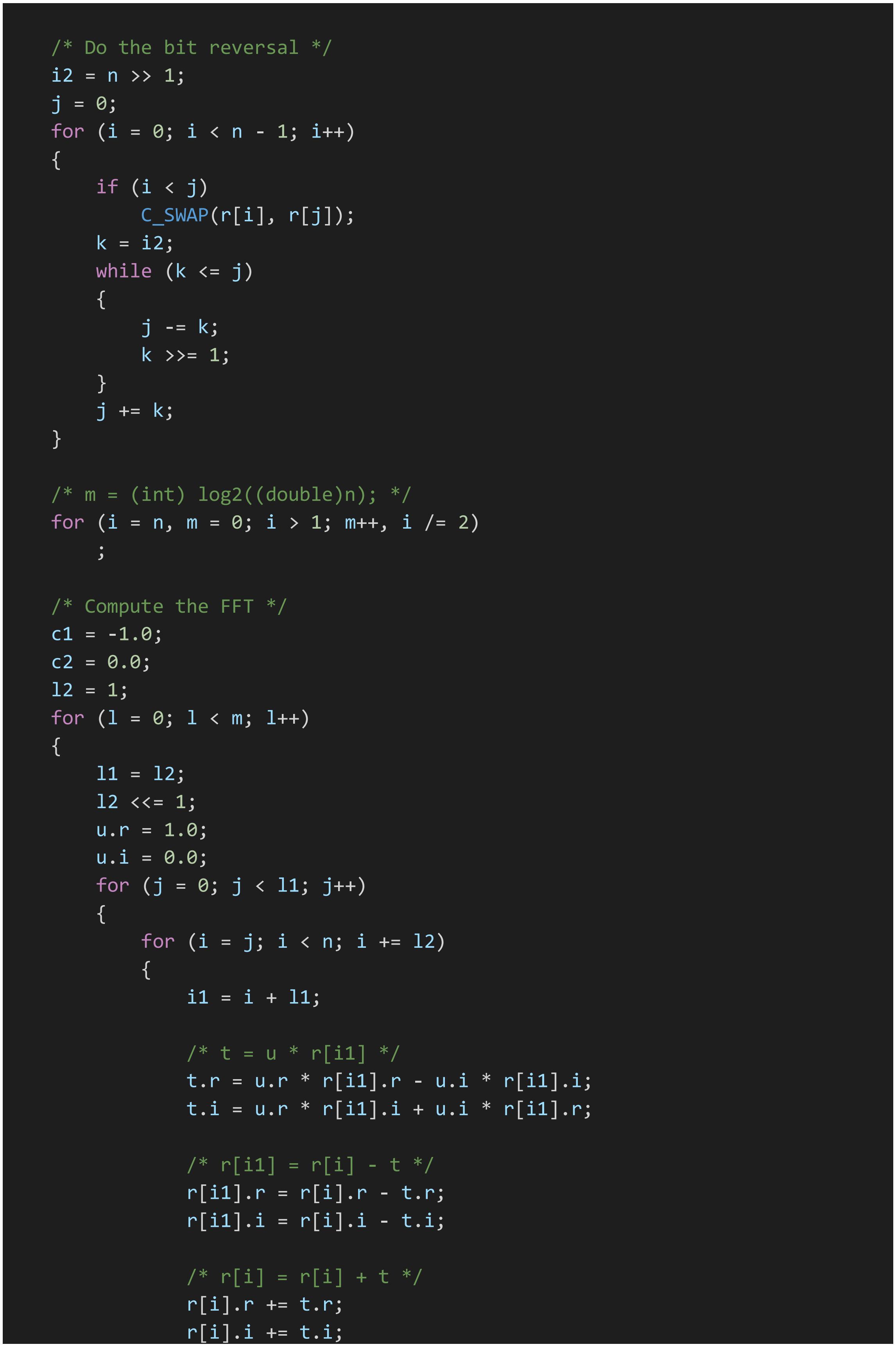
Github Link for Code - https://github.com/Aashita06/HPC_Practicals

Q.1) Study and implement 2D Convolution using MPI. Use different number of processes and analyze the performance.

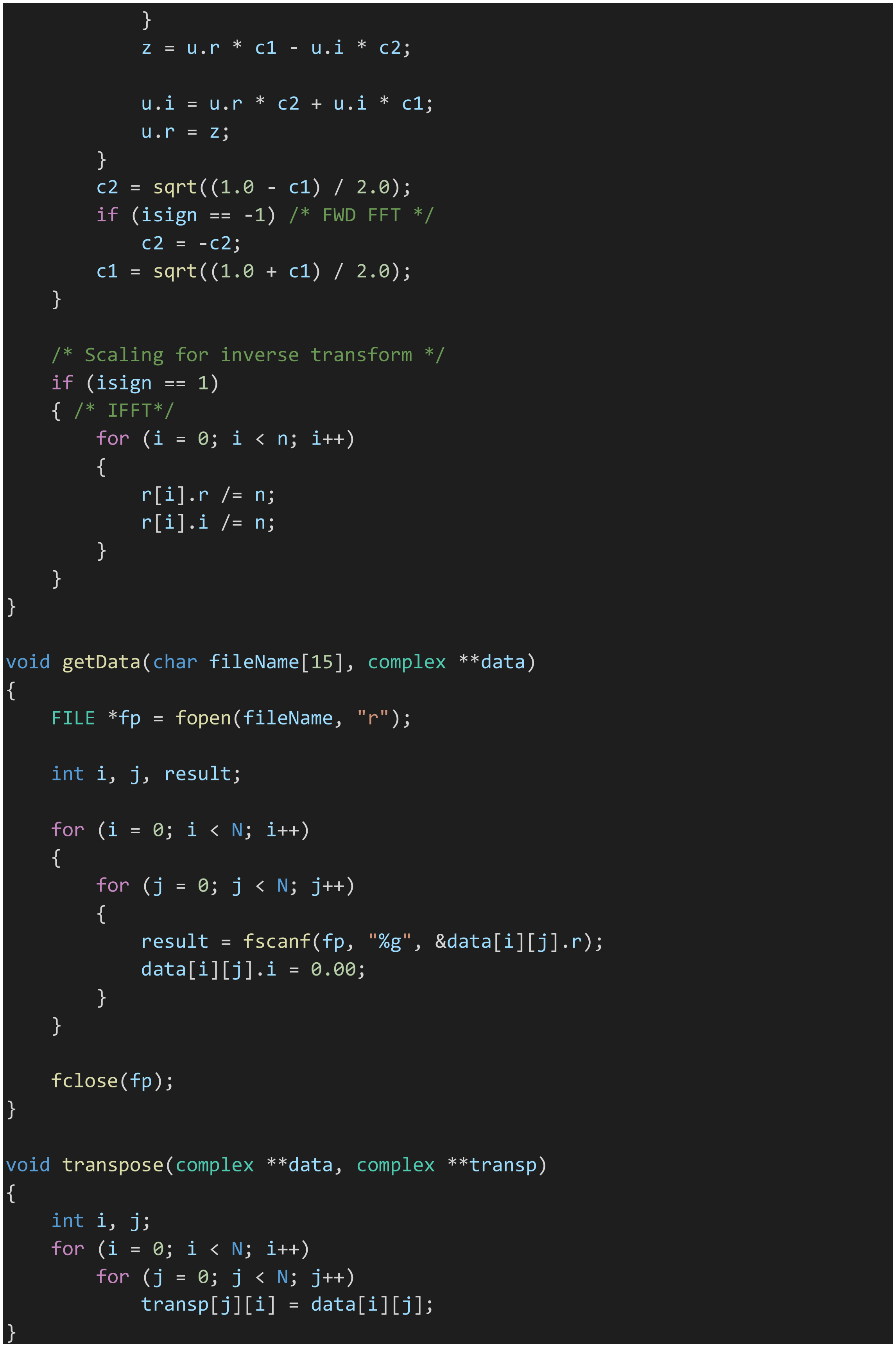
→

Code:

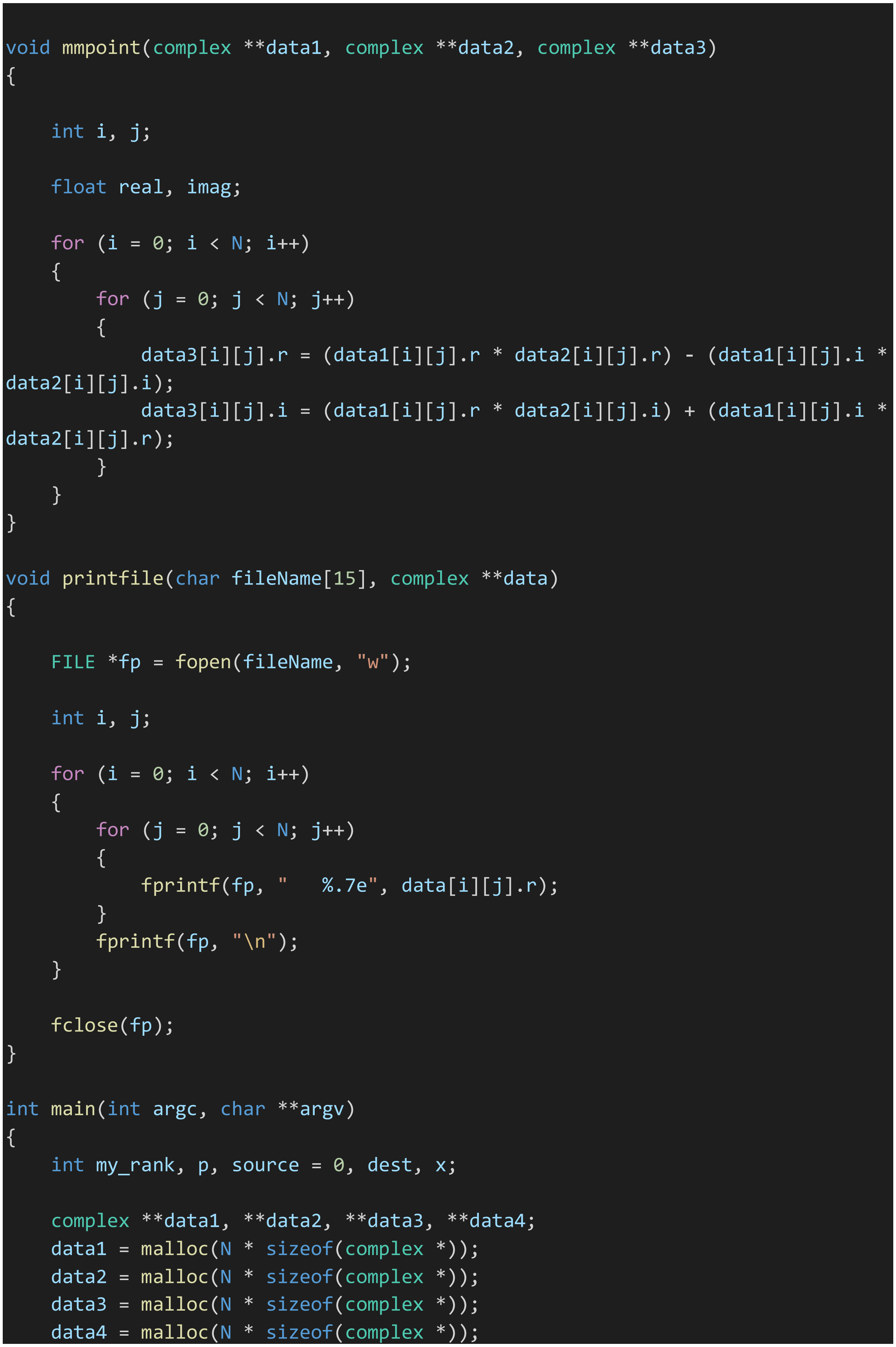
```
#include <assert.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

typedef struct
{
    float r;
    float i;
} complex;
static complex ctmp;

#define C_SWAP(a, b) \
    {                \
        ctmp = (a);  \
        (a) = (b);   \
        (b) = ctmp;  \
    }

#define N 512

void c_fft1d(complex *r, int n, int isign)
{
    int m, i, i1, j, k, i2, l, l1, l2;
    float c1, c2, z;
    complex t, u;

    if (isign == 0)
        return;
```



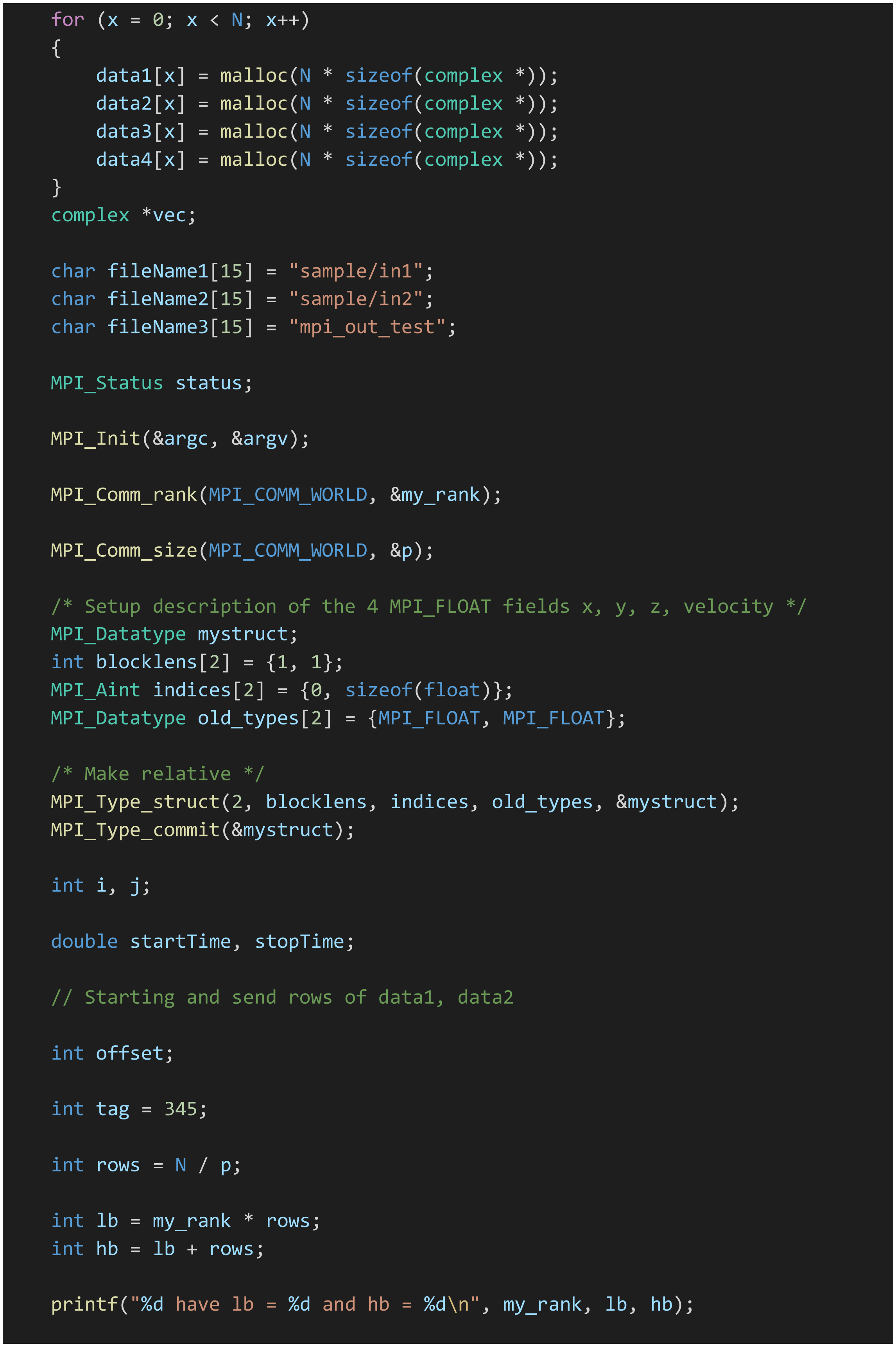
```
/* Do the bit reversal */
i2 = n >> 1;
j = 0;
for (i = 0; i < n - 1; i++)
{
    if (i < j)
        C_SWAP(r[i], r[j]);
    k = i2;
    while (k <= j)
    {
        j -= k;
        k >>= 1;
    }
    j += k;
}

/* m = (int) log2((double)n); */
for (i = n, m = 0; i > 1; m++, i /= 2)
    ;

/* Compute the FFT */
c1 = -1.0;
c2 = 0.0;
l2 = 1;
for (l = 0; l < m; l++)
{
    l1 = l2;
    l2 <<= 1;
    u.r = 1.0;
    u.i = 0.0;
    for (j = 0; j < l1; j++)
    {
        for (i = j; i < n; i += l2)
        {
            i1 = i + l1;

            /* t = u * r[i1] */
            t.r = u.r * r[i1].r - u.i * r[i1].i;
            t.i = u.r * r[i1].i + u.i * r[i1].r;

            /* r[i1] = r[i] - t */
            r[i1].r = r[i].r - t.r;
            r[i1].i = r[i].i - t.i;

            /* r[i] = r[i] + t */
            r[i].r += t.r;
            r[i].i += t.i;
```



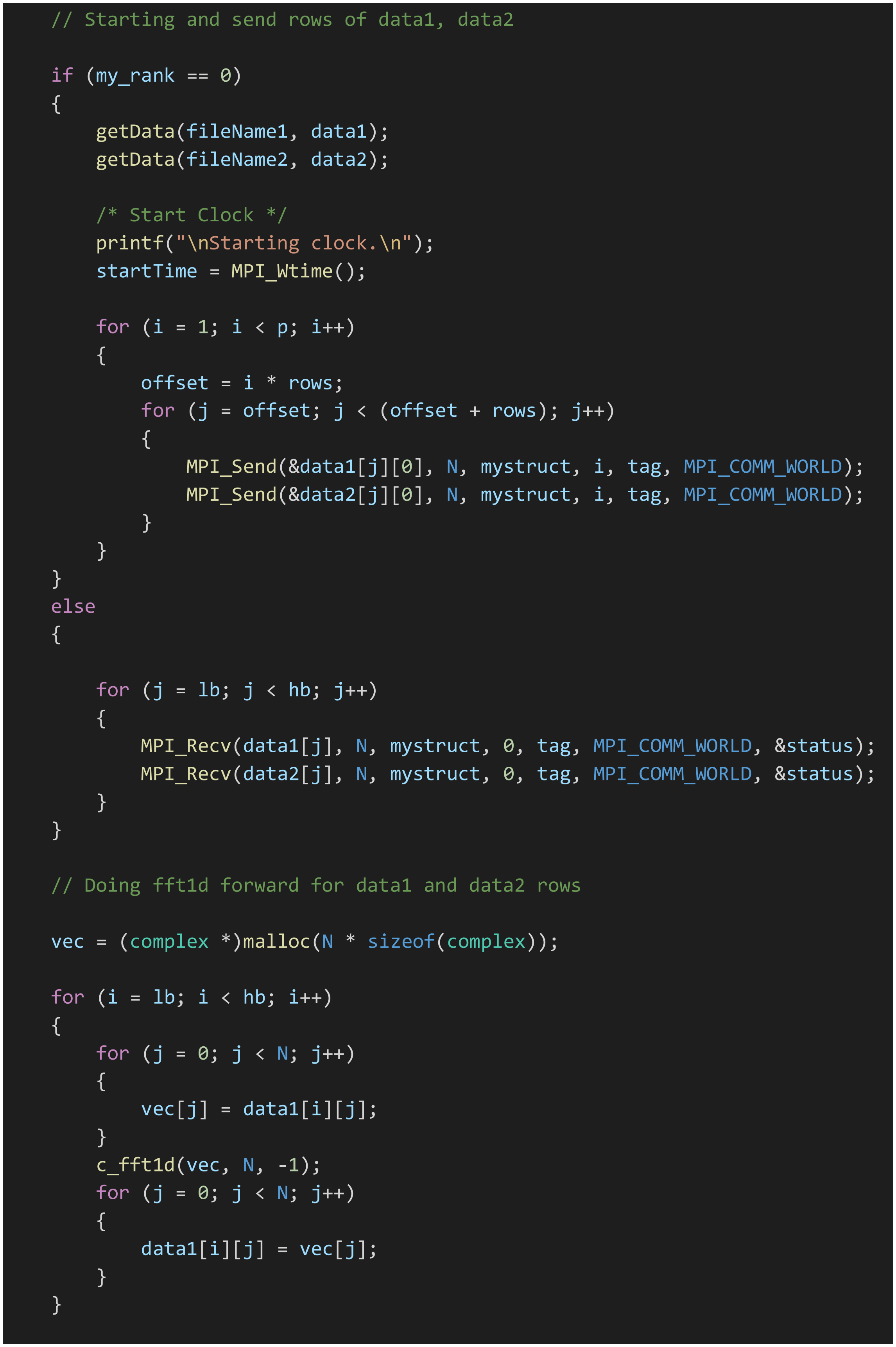
```
            }
            z = u.r * c1 - u.i * c2;

            u.i = u.r * c2 + u.i * c1;
            u.r = z;
        }
        c2 = sqrt((1.0 - c1) / 2.0);
        if (isign == -1) /* FWD FFT */
            c2 = -c2;
        c1 = sqrt((1.0 + c1) / 2.0);
    }

    /* Scaling for inverse transform */
    if (isign == 1)
    { /* IFFT*/
        for (i = 0; i < n; i++)
        {
            r[i].r /= n;
            r[i].i /= n;
        }
    }
}

void getData(char fileName[15], complex **data)
{
    FILE *fp = fopen(fileName, "r");

    int i, j, result;

    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            result = fscanf(fp, "%g", &data[i][j].r);
            data[i][j].i = 0.00;
        }
    }

    fclose(fp);
}

void transpose(complex **data, complex **transp)
{
    int i, j;
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            transp[j][i] = data[i][j];
}
```



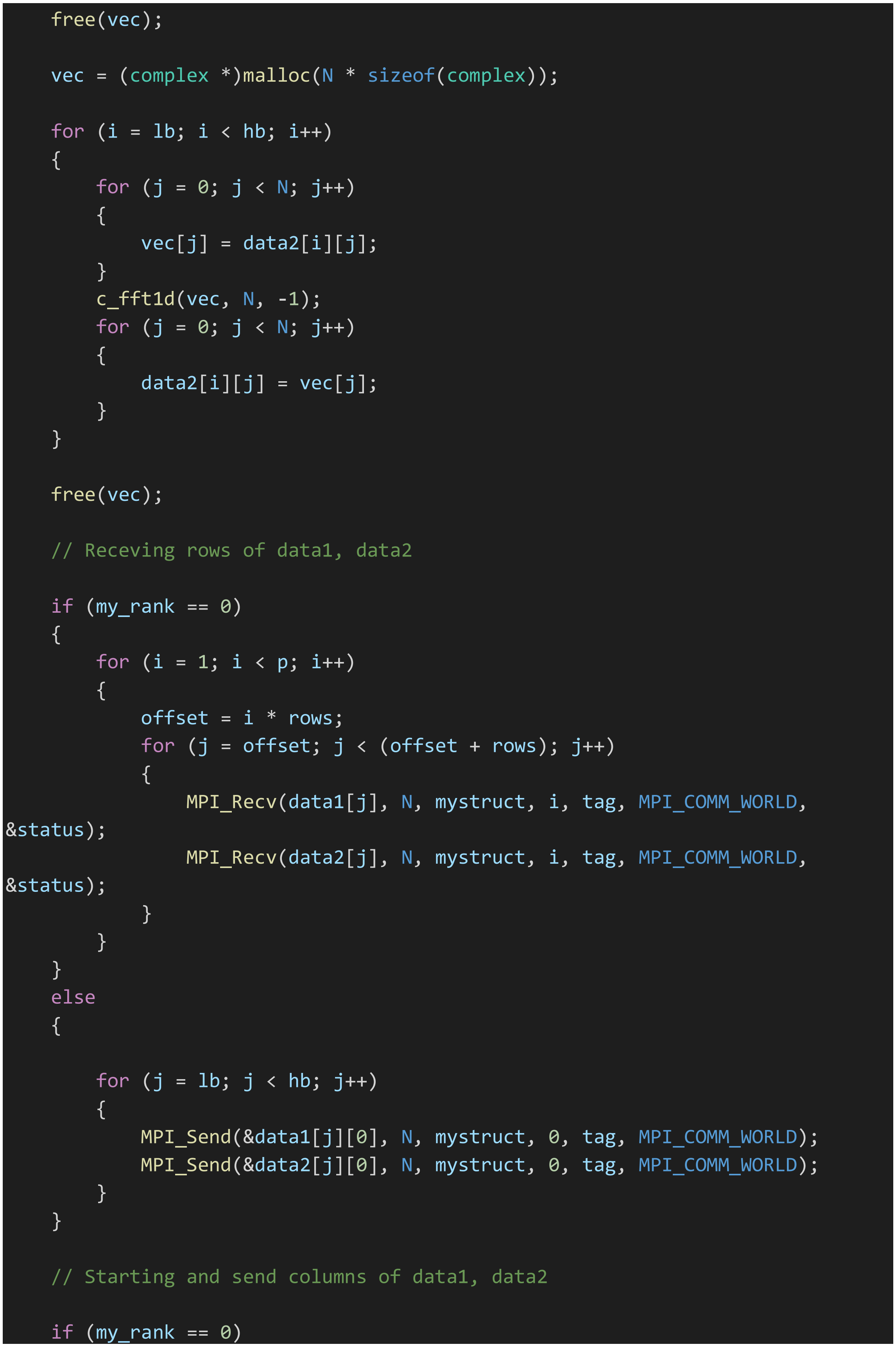
```
void mmpoint(complex **data1, complex **data2, complex **data3)
{

    int i, j;

    float real, imag;

    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            data3[i][j].r = (data1[i][j].r * data2[i][j].r) - (data1[i][j].i * data2[i][j].i);
            data3[i][j].i = (data1[i][j].r * data2[i][j].i) + (data1[i][j].i * data2[i][j].r);
        }
    }
}

void printfile(char fileName[15], complex **data)
{

    FILE *fp = fopen(fileName, "w");

    int i, j;

    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            fprintf(fp, "   %.7e", data[i][j].r);
        }
        fprintf(fp, "\n");
    }

    fclose(fp);
}

int main(int argc, char **argv)
{
    int my_rank, p, source = 0, dest, x;

    complex **data1, **data2, **data3, **data4;
    data1 = malloc(N * sizeof(complex *));
    data2 = malloc(N * sizeof(complex *));
    data3 = malloc(N * sizeof(complex *));
    data4 = malloc(N * sizeof(complex *));
```



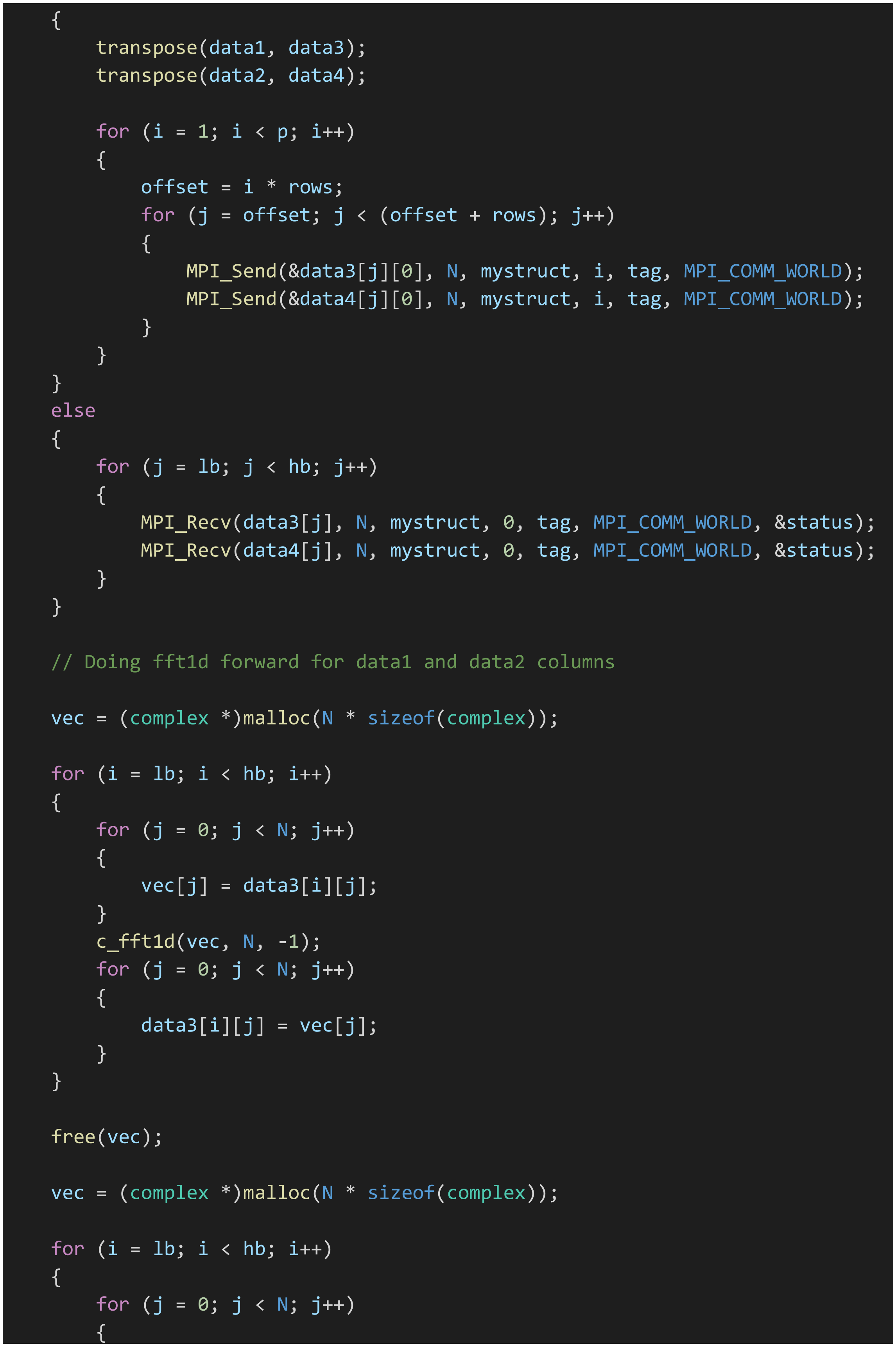
```c
for (x = 0; x < N; x++)
{
    data1[x] = malloc(N * sizeof(complex *));
    data2[x] = malloc(N * sizeof(complex *));
    data3[x] = malloc(N * sizeof(complex *));
    data4[x] = malloc(N * sizeof(complex *));
}
complex *vec;

char fileName1[15] = "sample/in1";
char fileName2[15] = "sample/in2";
char fileName3[15] = "mpi_out_test";

MPI_Status status;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

MPI_Comm_size(MPI_COMM_WORLD, &p);

/* Setup description of the 4 MPI_FLOAT fields x, y, z, velocity */
MPI_Datatype mystruct;
int blocklens[2] = {1, 1};
MPI_Aint indices[2] = {0, sizeof(float)};
MPI_Datatype old_types[2] = {MPI_FLOAT, MPI_FLOAT};

/* Make relative */
MPI_Type_struct(2, blocklens, indices, old_types, &mystruct);
MPI_Type_commit(&mystruct);

int i, j;

double startTime, stopTime;

// Starting and send rows of data1, data2

int offset;

int tag = 345;

int rows = N / p;

int lb = my_rank * rows;
int hb = lb + rows;

printf("%d have lb = %d and hb = %d\n", my_rank, lb, hb);
```



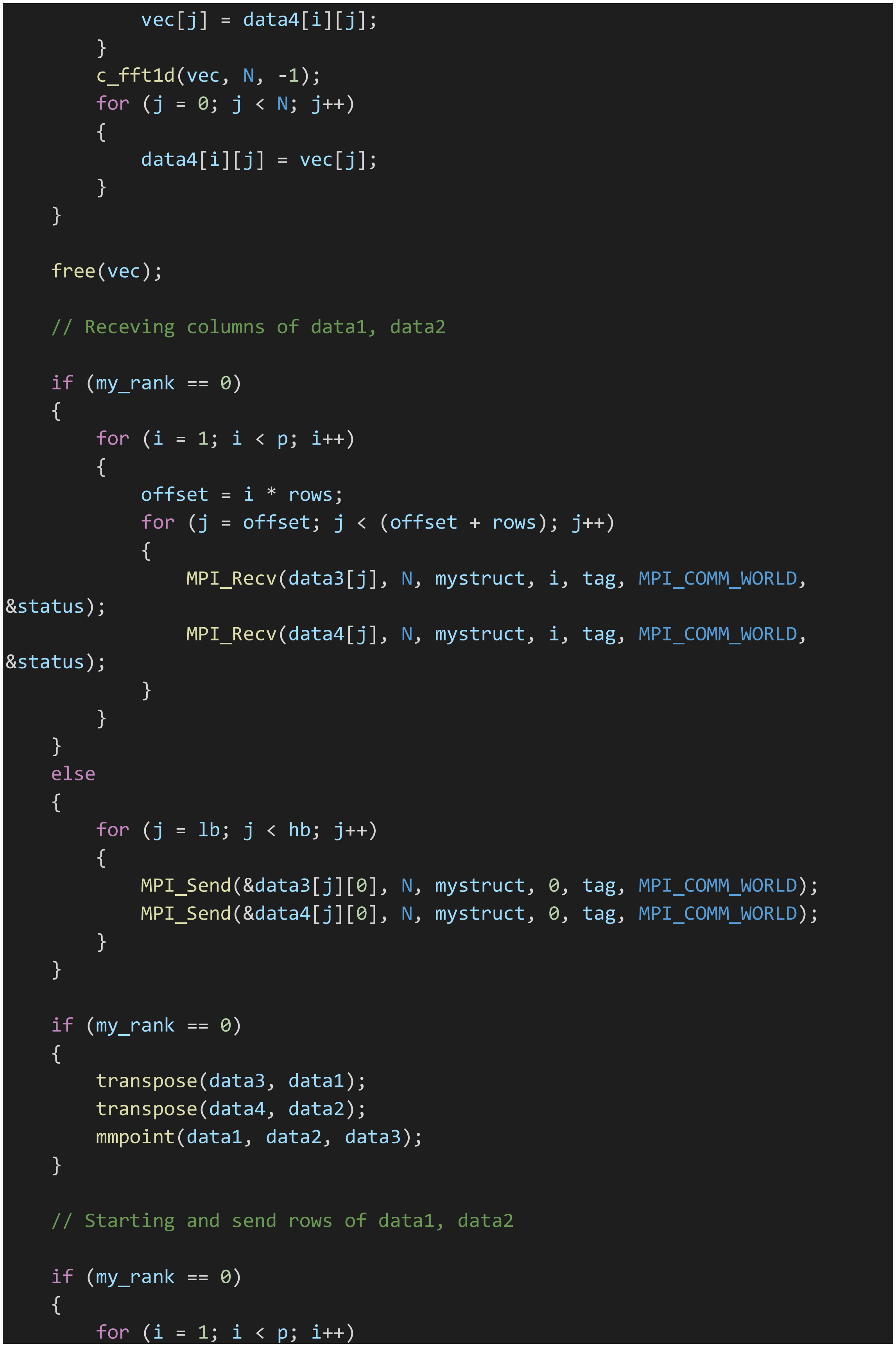
```
    // Starting and send rows of data1, data2

    if (my_rank == 0)
    {
        getData(fileName1, data1);
        getData(fileName2, data2);

        /* Start Clock */
        printf("\nStarting clock.\n");
        startTime = MPI_Wtime();

        for (i = 1; i < p; i++)
        {
            offset = i * rows;
            for (j = offset; j < (offset + rows); j++)
            {
                MPI_Send(&data1[j][0], N, mystruct, i, tag, MPI_COMM_WORLD);
                MPI_Send(&data2[j][0], N, mystruct, i, tag, MPI_COMM_WORLD);
            }
        }
    }
    else
    {

        for (j = lb; j < hb; j++)
        {
            MPI_Recv(data1[j], N, mystruct, 0, tag, MPI_COMM_WORLD, &status);
            MPI_Recv(data2[j], N, mystruct, 0, tag, MPI_COMM_WORLD, &status);
        }
    }

    // Doing fft1d forward for data1 and data2 rows

    vec = (complex *)malloc(N * sizeof(complex));

    for (i = lb; i < hb; i++)
    {
        for (j = 0; j < N; j++)
        {
            vec[j] = data1[i][j];
        }
        c_fft1d(vec, N, -1);
        for (j = 0; j < N; j++)
        {
            data1[i][j] = vec[j];
        }
    }
```



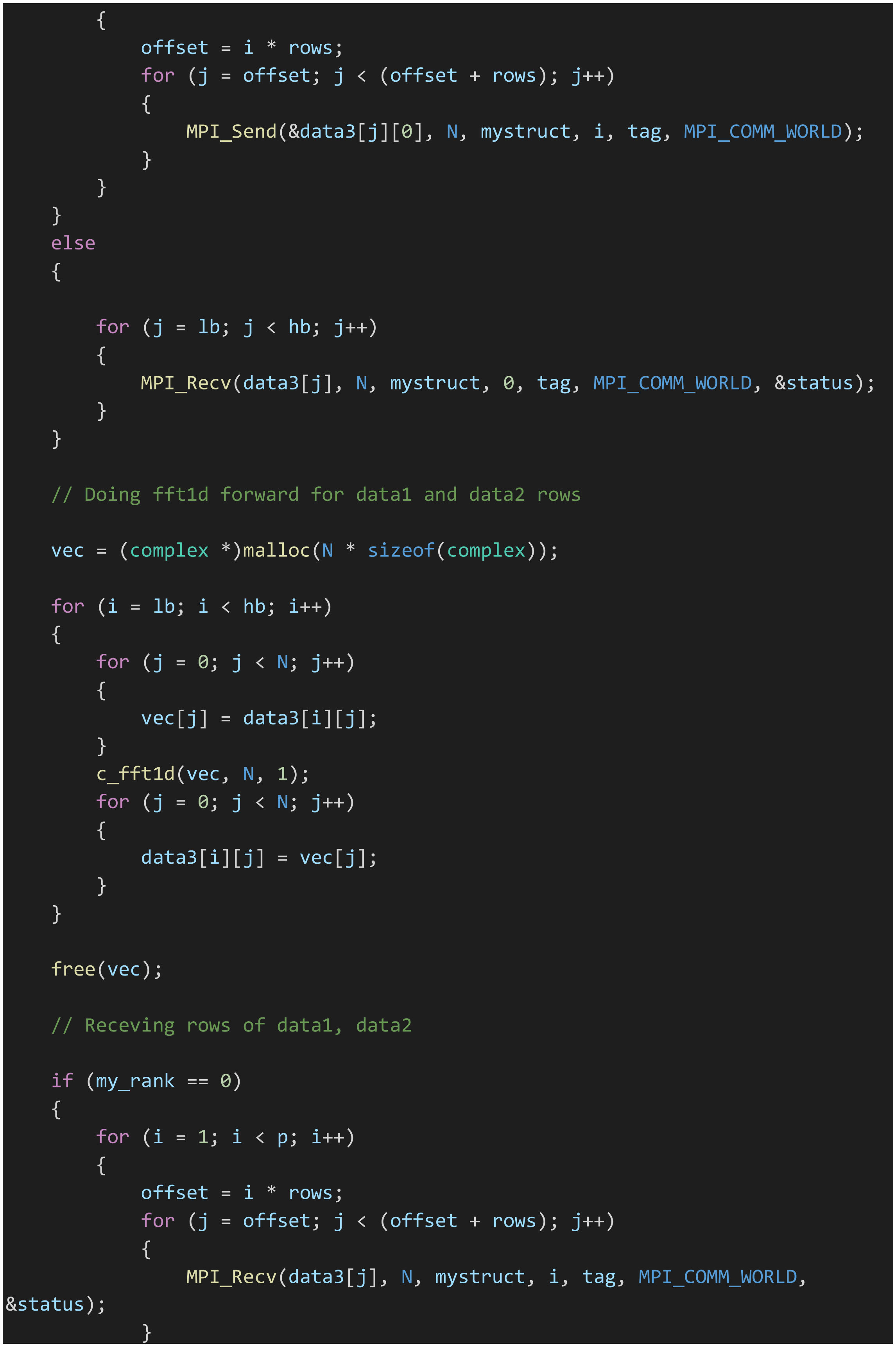
```
    free(vec);

    vec = (complex *)malloc(N * sizeof(complex));

    for (i = lb; i < hb; i++)
    {
        for (j = 0; j < N; j++)
        {
            vec[j] = data2[i][j];
        }
        c_fft1d(vec, N, -1);
        for (j = 0; j < N; j++)
        {
            data2[i][j] = vec[j];
        }
    }

    free(vec);

    // Receving rows of data1, data2

    if (my_rank == 0)
    {
        for (i = 1; i < p; i++)
        {
            offset = i * rows;
            for (j = offset; j < (offset + rows); j++)
            {
                MPI_Recv(data1[j], N, mystruct, i, tag, MPI_COMM_WORLD,
&status);
                MPI_Recv(data2[j], N, mystruct, i, tag, MPI_COMM_WORLD,
&status);
            }
        }
    }
    else
    {

        for (j = lb; j < hb; j++)
        {
            MPI_Send(&data1[j][0], N, mystruct, 0, tag, MPI_COMM_WORLD);
            MPI_Send(&data2[j][0], N, mystruct, 0, tag, MPI_COMM_WORLD);
        }
    }

    // Starting and send columns of data1, data2

    if (my_rank == 0)
```



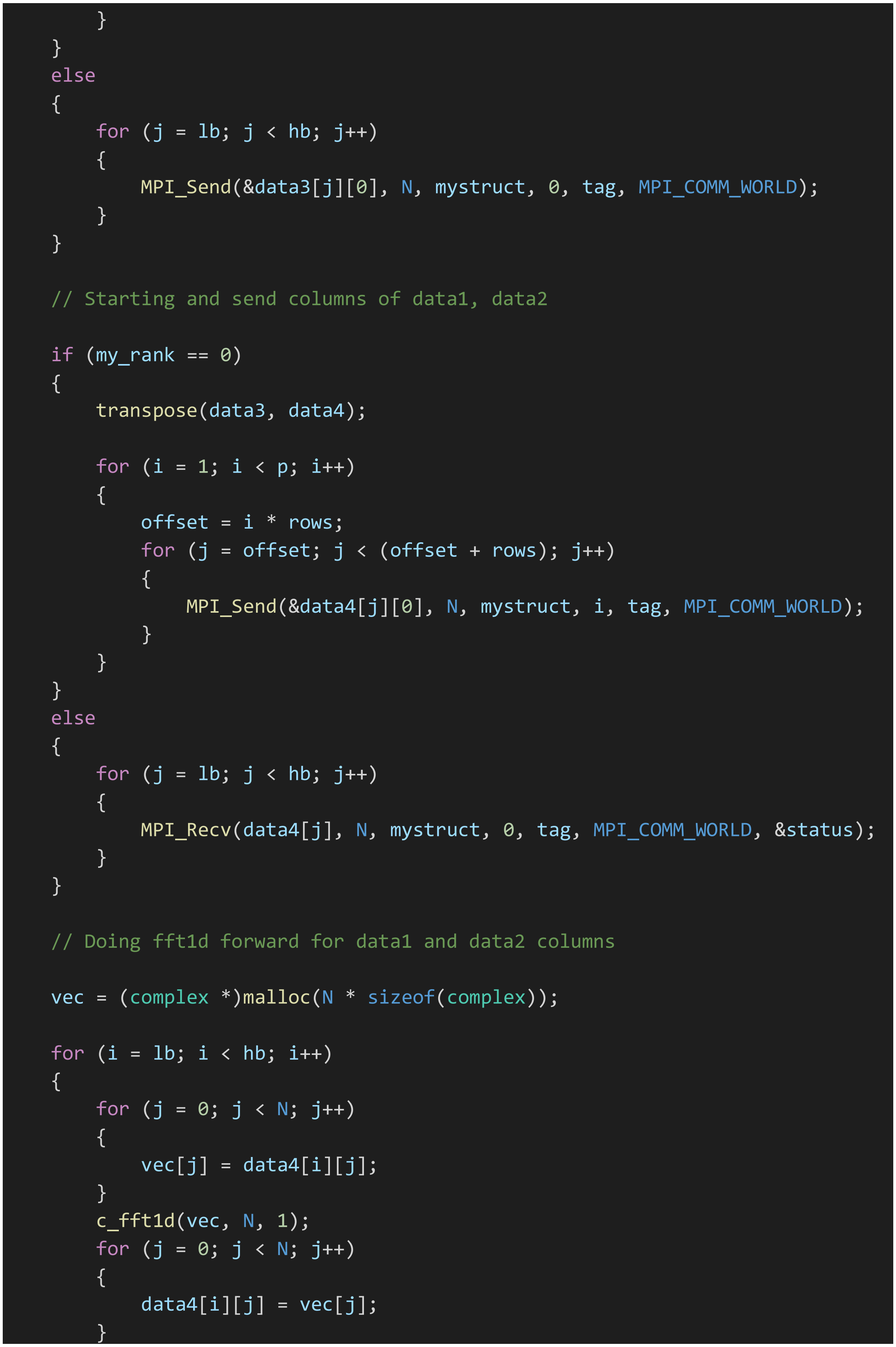
```
    {
        transpose(data1, data3);
        transpose(data2, data4);

        for (i = 1; i < p; i++)
        {
            offset = i * rows;
            for (j = offset; j < (offset + rows); j++)
            {
                MPI_Send(&data3[j][0], N, mystruct, i, tag, MPI_COMM_WORLD);
                MPI_Send(&data4[j][0], N, mystruct, i, tag, MPI_COMM_WORLD);
            }
        }
    }
    else
    {
        for (j = lb; j < hb; j++)
        {
            MPI_Recv(data3[j], N, mystruct, 0, tag, MPI_COMM_WORLD, &status);
            MPI_Recv(data4[j], N, mystruct, 0, tag, MPI_COMM_WORLD, &status);
        }
    }

    // Doing fft1d forward for data1 and data2 columns

    vec = (complex *)malloc(N * sizeof(complex));

    for (i = lb; i < hb; i++)
    {
        for (j = 0; j < N; j++)
        {
            vec[j] = data3[i][j];
        }
        c_fft1d(vec, N, -1);
        for (j = 0; j < N; j++)
        {
            data3[i][j] = vec[j];
        }
    }

    free(vec);

    vec = (complex *)malloc(N * sizeof(complex));

    for (i = lb; i < hb; i++)
    {
        for (j = 0; j < N; j++)
        {
```



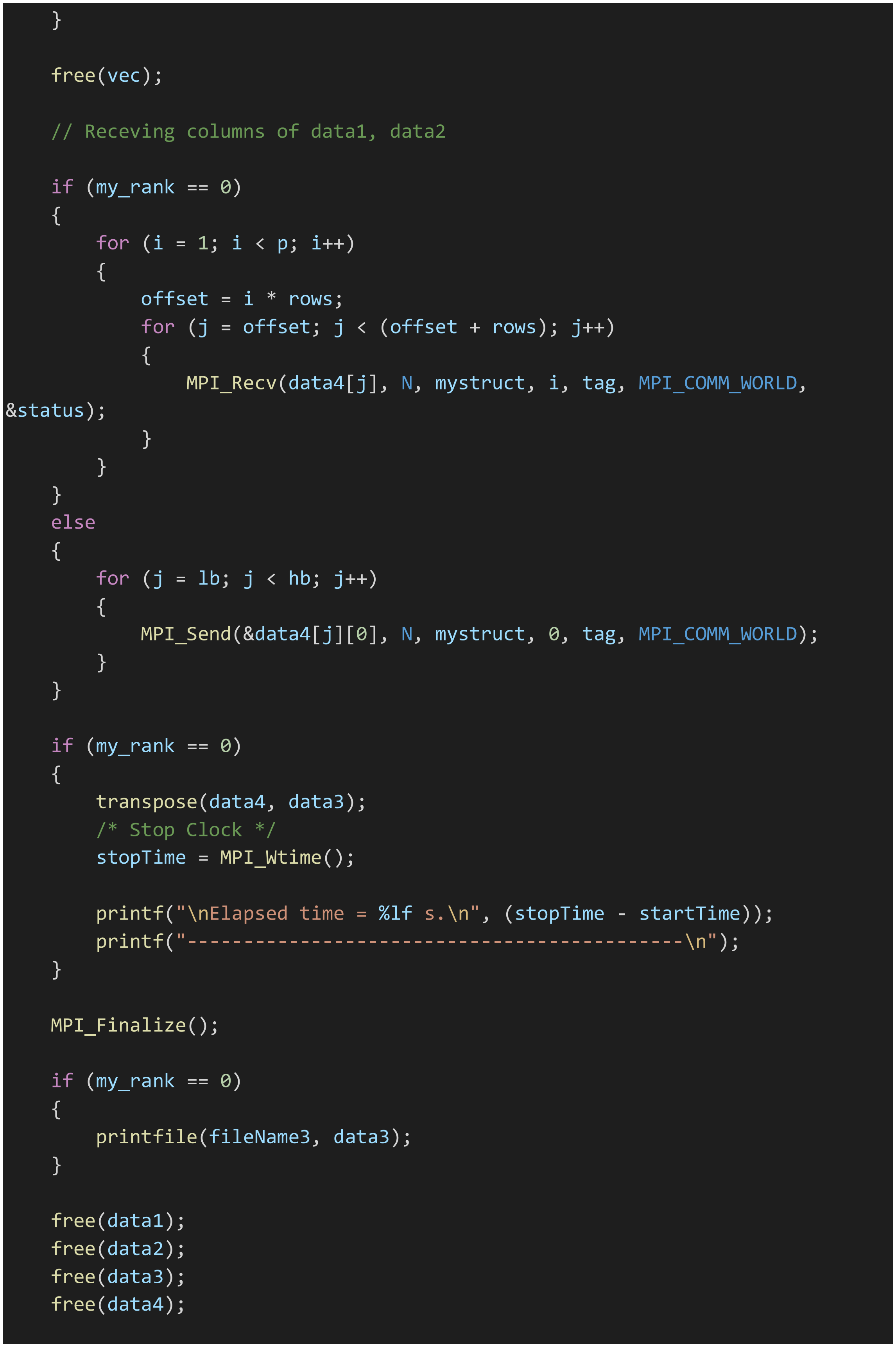
```
            vec[j] = data4[i][j];
        }
        c_fft1d(vec, N, -1);
        for (j = 0; j < N; j++)
        {
            data4[i][j] = vec[j];
        }
    }

    free(vec);

    // Receving columns of data1, data2

    if (my_rank == 0)
    {
        for (i = 1; i < p; i++)
        {
            offset = i * rows;
            for (j = offset; j < (offset + rows); j++)
            {
                MPI_Recv(data3[j], N, mystruct, i, tag, MPI_COMM_WORLD,
&status);
                MPI_Recv(data4[j], N, mystruct, i, tag, MPI_COMM_WORLD,
&status);
            }
        }
    }
    else
    {
        for (j = lb; j < hb; j++)
        {
            MPI_Send(&data3[j][0], N, mystruct, 0, tag, MPI_COMM_WORLD);
            MPI_Send(&data4[j][0], N, mystruct, 0, tag, MPI_COMM_WORLD);
        }
    }

    if (my_rank == 0)
    {
        transpose(data3, data1);
        transpose(data4, data2);
        mmpoint(data1, data2, data3);
    }

    // Starting and send rows of data1, data2

    if (my_rank == 0)
    {
        for (i = 1; i < p; i++)
```



```
        {
            offset = i * rows;
            for (j = offset; j < (offset + rows); j++)
            {
                MPI_Send(&data3[j][0], N, mystruct, i, tag, MPI_COMM_WORLD);
            }
        }
    }
    else
    {

        for (j = lb; j < hb; j++)
        {
            MPI_Recv(data3[j], N, mystruct, 0, tag, MPI_COMM_WORLD, &status);
        }
    }

    // Doing fft1d forward for data1 and data2 rows

    vec = (complex *)malloc(N * sizeof(complex));

    for (i = lb; i < hb; i++)
    {
        for (j = 0; j < N; j++)
        {
            vec[j] = data3[i][j];
        }
        c_fft1d(vec, N, 1);
        for (j = 0; j < N; j++)
        {
            data3[i][j] = vec[j];
        }
    }

    free(vec);

    // Receving rows of data1, data2

    if (my_rank == 0)
    {
        for (i = 1; i < p; i++)
        {
            offset = i * rows;
            for (j = offset; j < (offset + rows); j++)
            {
                MPI_Recv(data3[j], N, mystruct, i, tag, MPI_COMM_WORLD,
&status);
            }
```



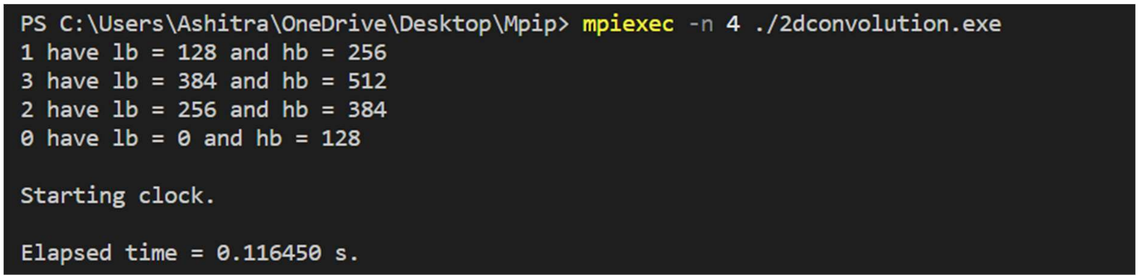
```
        }
    }
    else
    {
        for (j = lb; j < hb; j++)
        {
            MPI_Send(&data3[j][0], N, mystruct, 0, tag, MPI_COMM_WORLD);
        }
    }

    // Starting and send columns of data1, data2

    if (my_rank == 0)
    {
        transpose(data3, data4);

        for (i = 1; i < p; i++)
        {
            offset = i * rows;
            for (j = offset; j < (offset + rows); j++)
            {
                MPI_Send(&data4[j][0], N, mystruct, i, tag, MPI_COMM_WORLD);
            }
        }
    }
    else
    {
        for (j = lb; j < hb; j++)
        {
            MPI_Recv(data4[j], N, mystruct, 0, tag, MPI_COMM_WORLD, &status);
        }
    }

    // Doing fft1d forward for data1 and data2 columns

    vec = (complex *)malloc(N * sizeof(complex));

    for (i = lb; i < hb; i++)
    {
        for (j = 0; j < N; j++)
        {
            vec[j] = data4[i][j];
        }
        c_fft1d(vec, N, 1);
        for (j = 0; j < N; j++)
        {
            data4[i][j] = vec[j];
        }
```



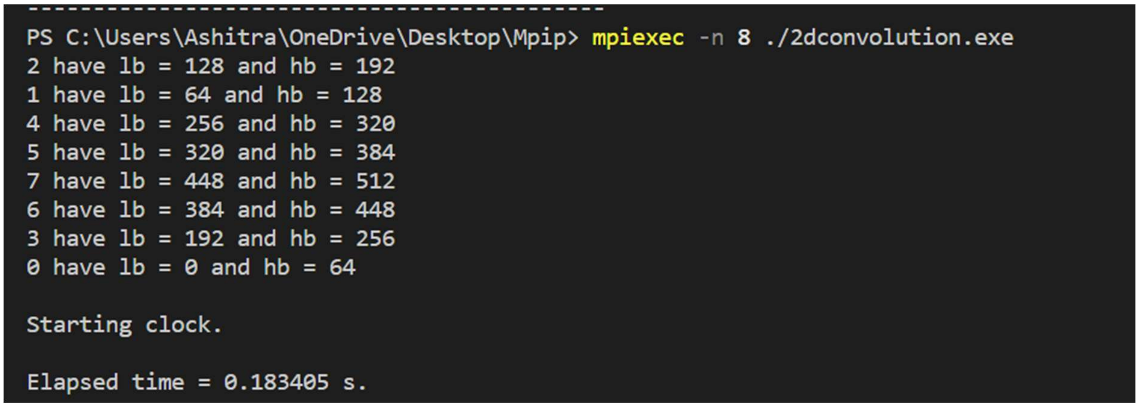
```
    }

    free(vec);

    // Receving columns of data1, data2

    if (my_rank == 0)
    {
        for (i = 1; i < p; i++)
        {
            offset = i * rows;
            for (j = offset; j < (offset + rows); j++)
            {
                MPI_Recv(data4[j], N, mystruct, i, tag, MPI_COMM_WORLD,
&status);
            }
        }
    }
    else
    {
        for (j = lb; j < hb; j++)
        {
            MPI_Send(&data4[j][0], N, mystruct, 0, tag, MPI_COMM_WORLD);
        }
    }

    if (my_rank == 0)
    {
        transpose(data4, data3);
        /* Stop Clock */
        stopTime = MPI_Wtime();

        printf("\nElapsed time = %lf s.\n", (stopTime - startTime));
        printf("--------------------------------------------\n");
    }

    MPI_Finalize();

    if (my_rank == 0)
    {
        printfile(fileName3, data3);
    }

    free(data1);
    free(data2);
    free(data3);
    free(data4);
```

```
    return 0;
}
```

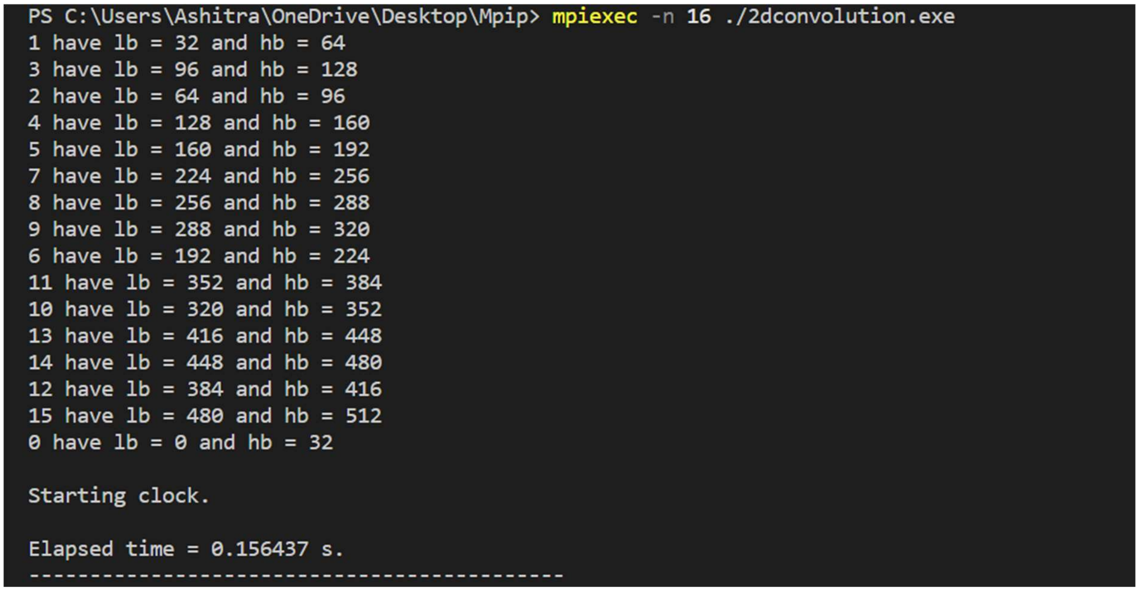
Output:



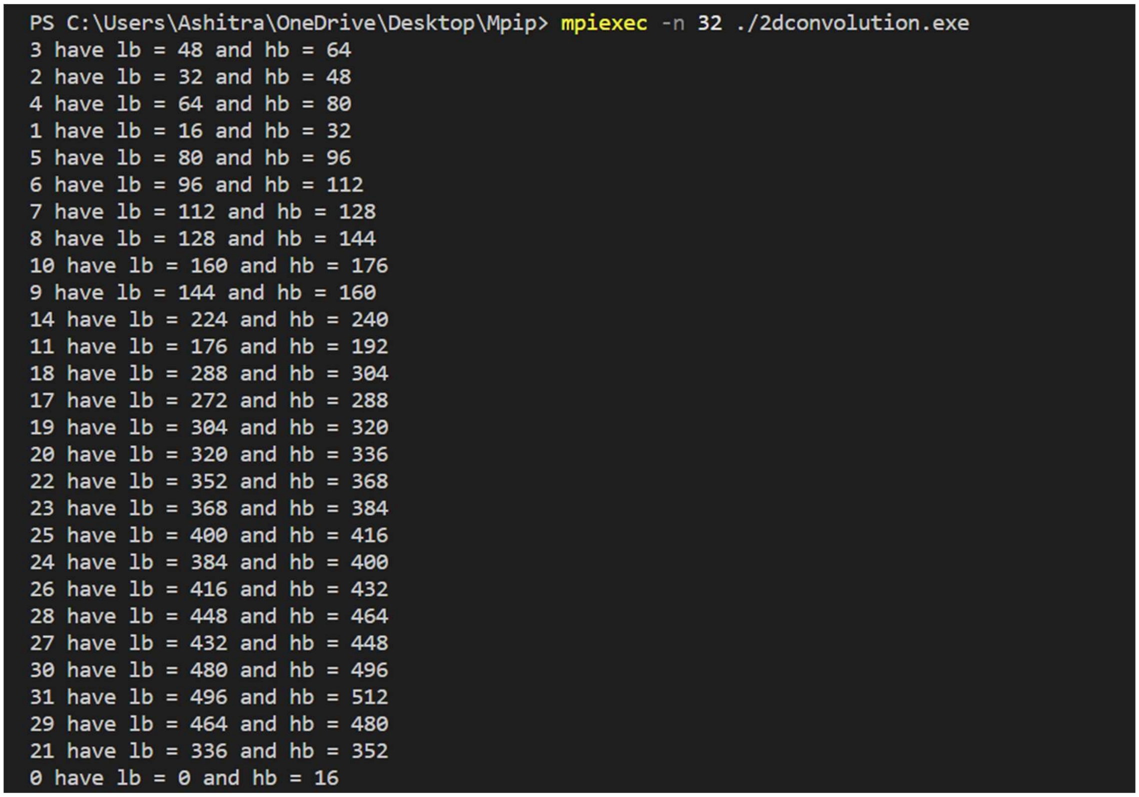
```
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 4 ./2dconvolution.exe
1 have lb = 128 and hb = 256
3 have lb = 384 and hb = 512
2 have lb = 256 and hb = 384
0 have lb = 0 and hb = 128

Starting clock.

Elapsed time = 0.116450 s.
```



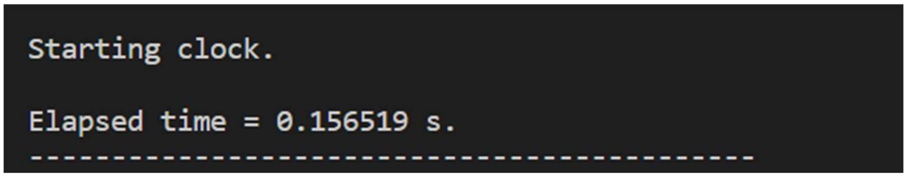
```
--------------------------------------------
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 8 ./2dconvolution.exe
2 have lb = 128 and hb = 192
1 have lb = 64 and hb = 128
4 have lb = 256 and hb = 320
5 have lb = 320 and hb = 384
7 have lb = 448 and hb = 512
6 have lb = 384 and hb = 448
3 have lb = 192 and hb = 256
0 have lb = 0 and hb = 64

Starting clock.

Elapsed time = 0.183405 s.
```



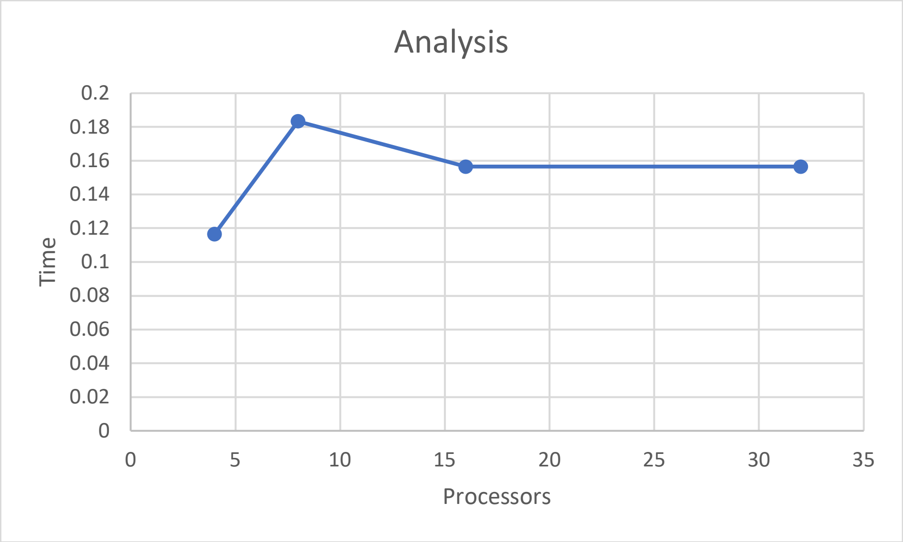
```
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 16 ./2dconvolution.exe
1 have lb = 32 and hb = 64
3 have lb = 96 and hb = 128
2 have lb = 64 and hb = 96
4 have lb = 128 and hb = 160
5 have lb = 160 and hb = 192
7 have lb = 224 and hb = 256
8 have lb = 256 and hb = 288
9 have lb = 288 and hb = 320
6 have lb = 192 and hb = 224
11 have lb = 352 and hb = 384
10 have lb = 320 and hb = 352
13 have lb = 416 and hb = 448
14 have lb = 448 and hb = 480
12 have lb = 384 and hb = 416
15 have lb = 480 and hb = 512
0 have lb = 0 and hb = 32

Starting clock.

Elapsed time = 0.156437 s.
--------------------------------------------
```



```
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 32 ./2dconvolution.exe
3 have lb = 48 and hb = 64
2 have lb = 32 and hb = 48
4 have lb = 64 and hb = 80
1 have lb = 16 and hb = 32
5 have lb = 80 and hb = 96
6 have lb = 96 and hb = 112
7 have lb = 112 and hb = 128
8 have lb = 128 and hb = 144
10 have lb = 160 and hb = 176
9 have lb = 144 and hb = 160
14 have lb = 224 and hb = 240
11 have lb = 176 and hb = 192
18 have lb = 288 and hb = 304
17 have lb = 272 and hb = 288
19 have lb = 304 and hb = 320
20 have lb = 320 and hb = 336
22 have lb = 352 and hb = 368
23 have lb = 368 and hb = 384
25 have lb = 400 and hb = 416
24 have lb = 384 and hb = 400
26 have lb = 416 and hb = 432
28 have lb = 448 and hb = 464
27 have lb = 432 and hb = 448
30 have lb = 480 and hb = 496
31 have lb = 496 and hb = 512
29 have lb = 464 and hb = 480
21 have lb = 336 and hb = 352
0 have lb = 0 and hb = 16
```



```
Starting clock.

Elapsed time = 0.156519 s.
-------------------------------------------
```

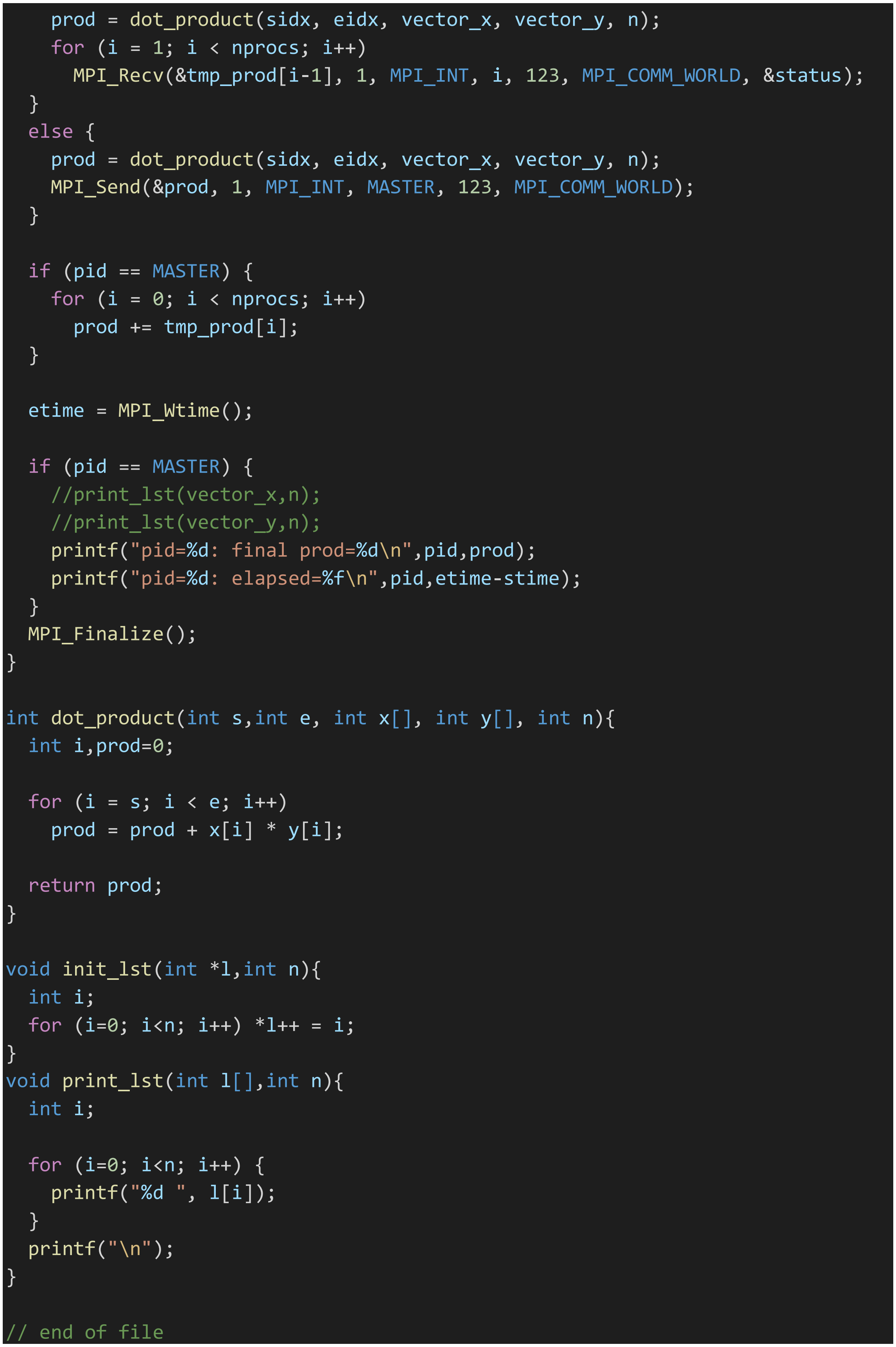


Q.2) Implement dot product using MPI. Use different number of processes and analyze the performance.

→

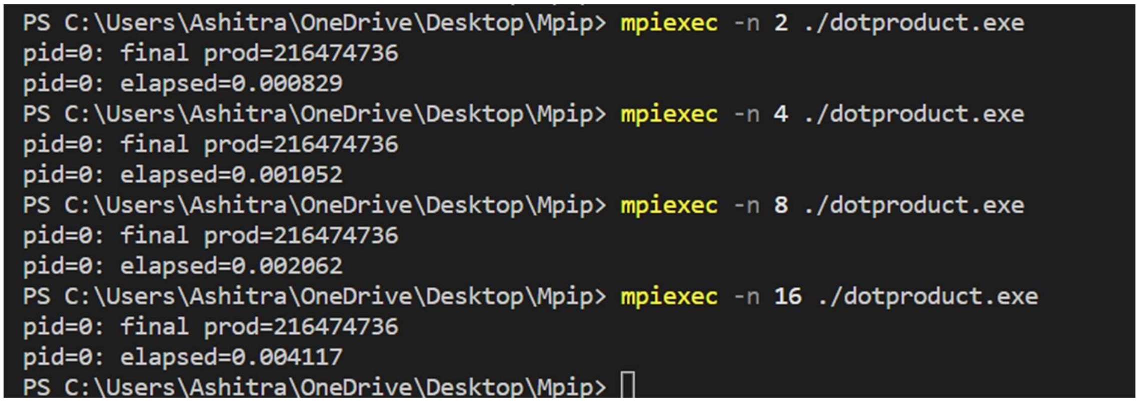
Code:

```
#include <stdio.h>
#include <mpi.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

#define NELMS 100000
#define MASTER 0
#define MAXPROCS 16

int dot_product();
void init_lst();
void print_lst();

int main() {
  int i,n,vector_x[NELMS],vector_y[NELMS];
  int prod,sidx,eidx,size;
  int pid,nprocs, rank;
  double stime,etime;
  MPI_Status status;
  MPI_Comm world;

  n = 100000;
  if (n > NELMS) { printf("n=%d > N=%d\n",n,NELMS); exit(1); }

  MPI_Init(NULL, NULL);
  world = MPI_COMM_WORLD;
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &pid);

  int portion = n / nprocs;
  sidx = pid * portion;
  eidx = sidx + portion;
  init_lst(vector_x, n);
  init_lst(vector_y, n);

  int tmp_prod[nprocs];
  for (i = 0; i < nprocs; i++)
    tmp_prod[i] = 0;

  stime = MPI_Wtime();

  if (pid == MASTER) {
```

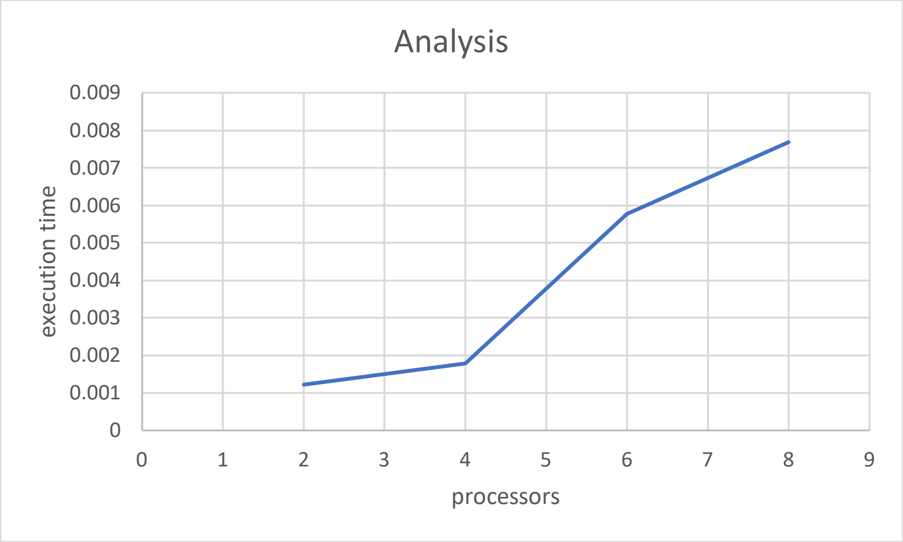


```
    prod = dot_product(sidx, eidx, vector_x, vector_y, n);
    for (i = 1; i < nprocs; i++)
      MPI_Recv(&tmp_prod[i-1], 1, MPI_INT, i, 123, MPI_COMM_WORLD, &status);
  }
  else {
    prod = dot_product(sidx, eidx, vector_x, vector_y, n);
    MPI_Send(&prod, 1, MPI_INT, MASTER, 123, MPI_COMM_WORLD);
  }

  if (pid == MASTER) {
    for (i = 0; i < nprocs; i++)
      prod += tmp_prod[i];
  }

  etime = MPI_Wtime();

  if (pid == MASTER) {
    //print_lst(vector_x,n);
    //print_lst(vector_y,n);
    printf("pid=%d: final prod=%d\n",pid,prod);
    printf("pid=%d: elapsed=%f\n",pid,etime-stime);
  }
  MPI_Finalize();
}

int dot_product(int s,int e, int x[], int y[], int n){
  int i,prod=0;

  for (i = s; i < e; i++)
    prod = prod + x[i] * y[i];

  return prod;
}

void init_lst(int *l,int n){
  int i;
  for (i=0; i<n; i++) *l++ = i;
}
void print_lst(int l[],int n){
  int i;

  for (i=0; i<n; i++) {
    printf("%d ", l[i]);
  }
  printf("\n");
}

// end of file
```

Output:



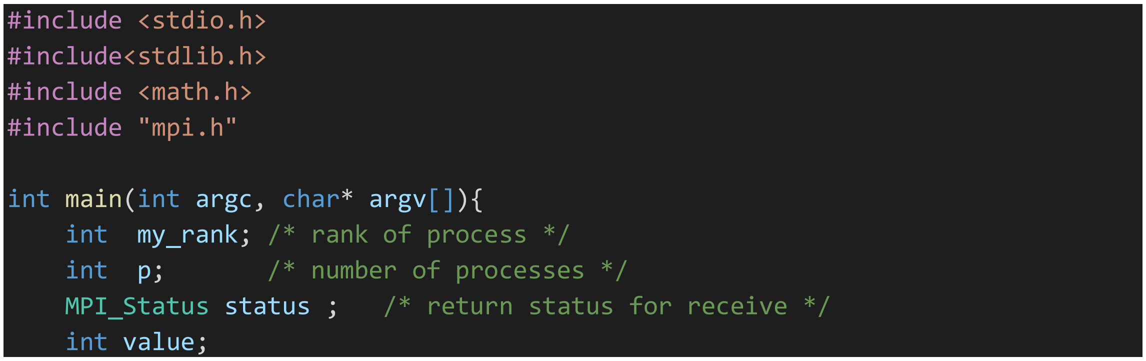
```
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 2 ./dotproduct.exe
pid=0: final prod=216474736
pid=0: elapsed=0.000829
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 4 ./dotproduct.exe
pid=0: final prod=216474736
pid=0: elapsed=0.001052
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 8 ./dotproduct.exe
pid=0: final prod=216474736
pid=0: elapsed=0.002062
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 16 ./dotproduct.exe
pid=0: final prod=216474736
pid=0: elapsed=0.004117
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip>
```



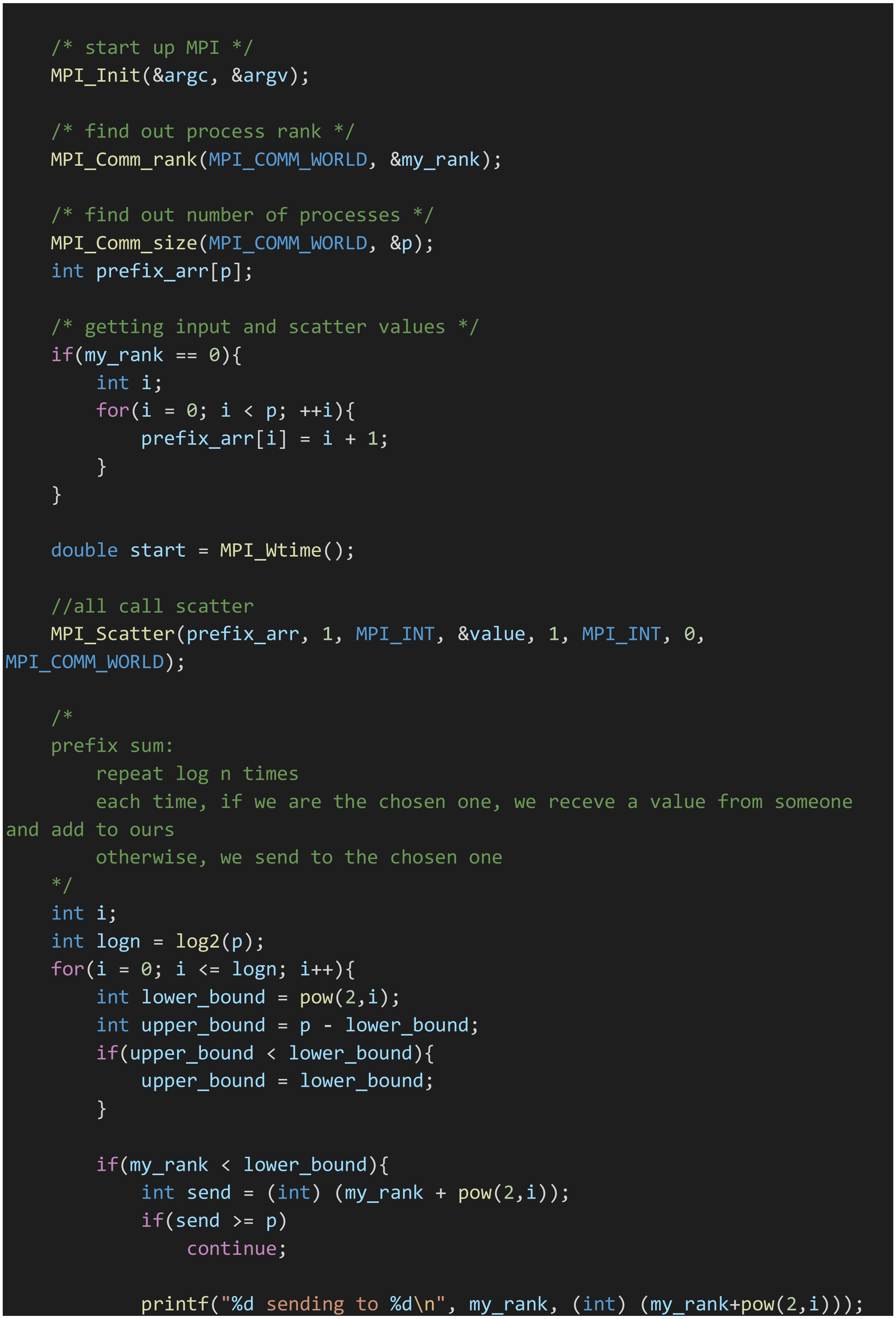
Q.3) Implement Prefix sum using MPI. Use different number of processes and analyze the performance.

→

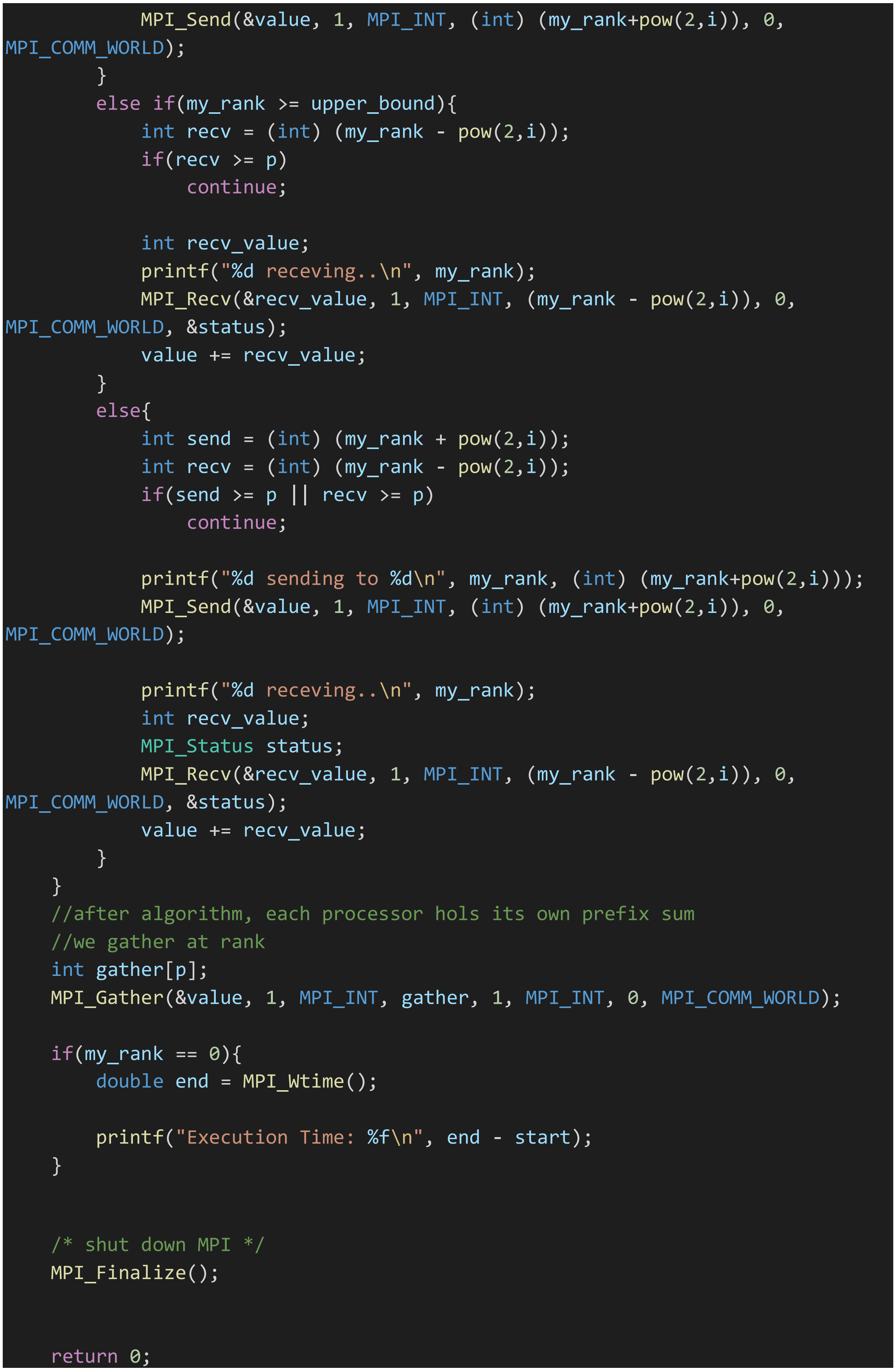
Code:



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

int main(int argc, char* argv[]){
    int  my_rank; /* rank of process */
    int  p;       /* number of processes */
    MPI_Status status ;   /* return status for receive */
    int value;
```



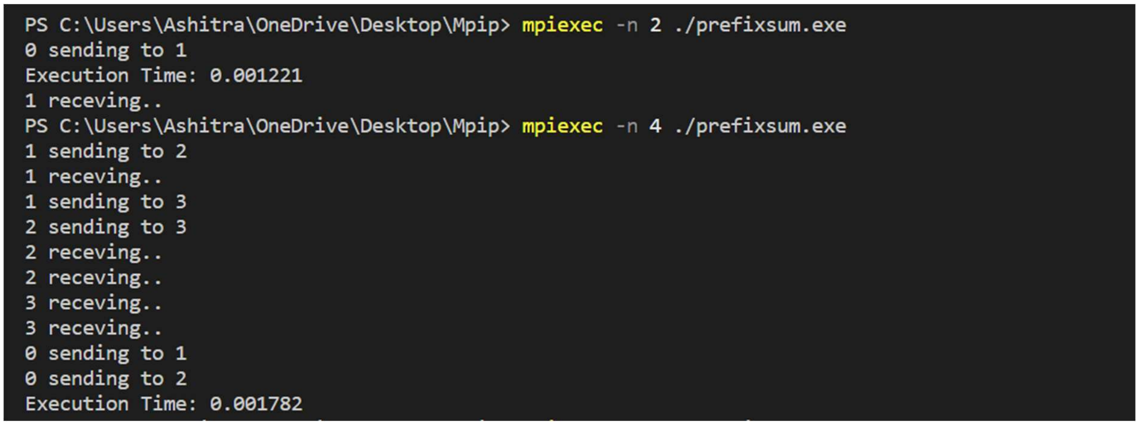
```
    /* start up MPI */
    MPI_Init(&argc, &argv);

    /* find out process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    int prefix_arr[p];

    /* getting input and scatter values */
    if(my_rank == 0){
        int i;
        for(i = 0; i < p; ++i){
            prefix_arr[i] = i + 1;
        }
    }

    double start = MPI_Wtime();

    //all call scatter
    MPI_Scatter(prefix_arr, 1, MPI_INT, &value, 1, MPI_INT, 0,
MPI_COMM_WORLD);

    /*
    prefix sum:
        repeat log n times
        each time, if we are the chosen one, we receve a value from someone
and add to ours
        otherwise, we send to the chosen one
    */
    int i;
    int logn = log2(p);
    for(i = 0; i <= logn; i++){
        int lower_bound = pow(2,i);
        int upper_bound = p - lower_bound;
        if(upper_bound < lower_bound){
            upper_bound = lower_bound;
        }

        if(my_rank < lower_bound){
            int send = (int) (my_rank + pow(2,i));
            if(send >= p)
                continue;

            printf("%d sending to %d\n", my_rank, (int) (my_rank+pow(2,i)));
```



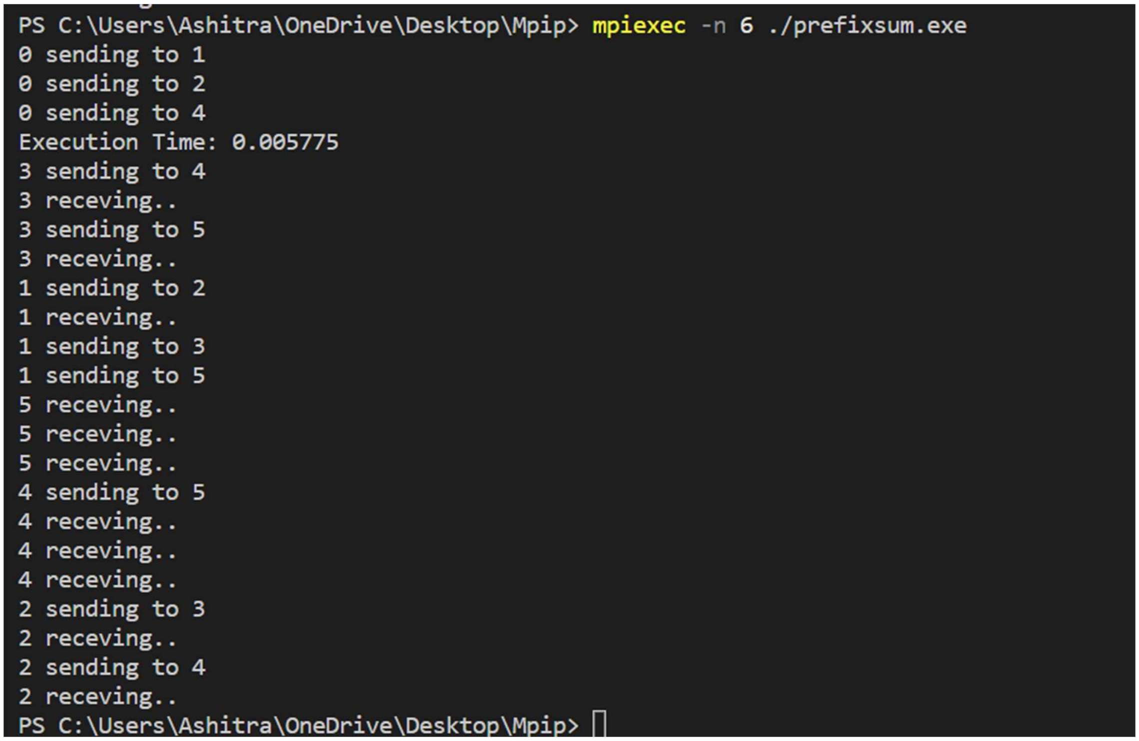
```
            MPI_Send(&value, 1, MPI_INT, (int) (my_rank+pow(2,i)), 0,
MPI_COMM_WORLD);
        }
        else if(my_rank >= upper_bound){
            int recv = (int) (my_rank - pow(2,i));
            if(recv >= p)
                continue;

            int recv_value;
            printf("%d receving..\n", my_rank);
            MPI_Recv(&recv_value, 1, MPI_INT, (my_rank - pow(2,i)), 0,
MPI_COMM_WORLD, &status);
            value += recv_value;
        }
        else{
            int send = (int) (my_rank + pow(2,i));
            int recv = (int) (my_rank - pow(2,i));
            if(send >= p || recv >= p)
                continue;

            printf("%d sending to %d\n", my_rank, (int) (my_rank+pow(2,i)));
            MPI_Send(&value, 1, MPI_INT, (int) (my_rank+pow(2,i)), 0,
MPI_COMM_WORLD);

            printf("%d receving..\n", my_rank);
            int recv_value;
            MPI_Status status;
            MPI_Recv(&recv_value, 1, MPI_INT, (my_rank - pow(2,i)), 0,
MPI_COMM_WORLD, &status);
            value += recv_value;
        }
    }
    //after algorithm, each processor hols its own prefix sum
    //we gather at rank
    int gather[p];
    MPI_Gather(&value, 1, MPI_INT, gather, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if(my_rank == 0){
        double end = MPI_Wtime();

        printf("Execution Time: %f\n", end - start);
    }


    /* shut down MPI */
    MPI_Finalize();


    return 0;
```

```
}
```

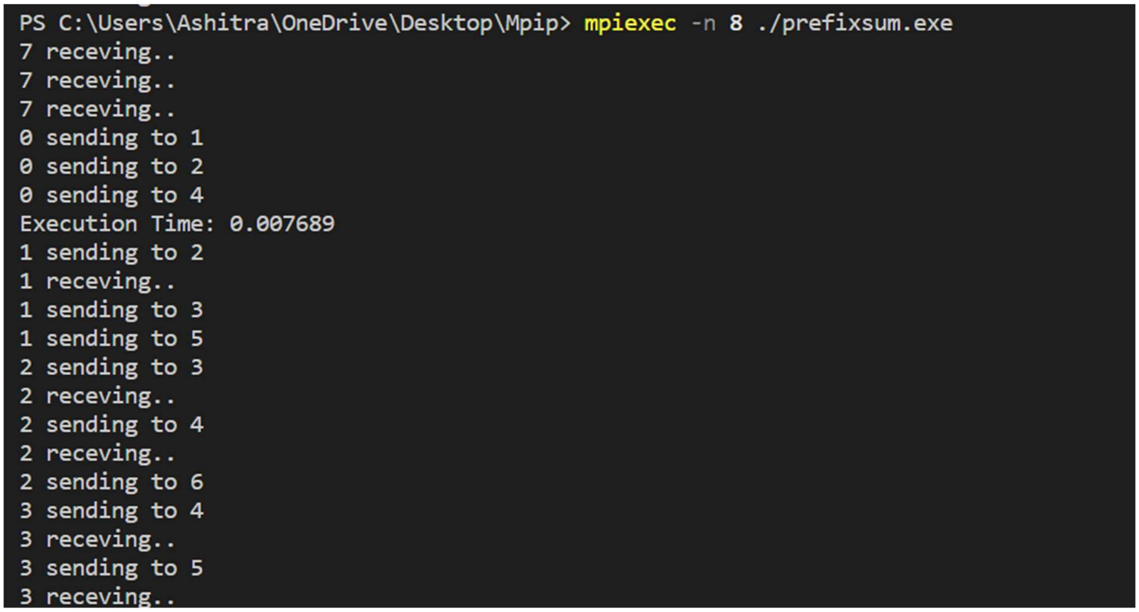
Output:



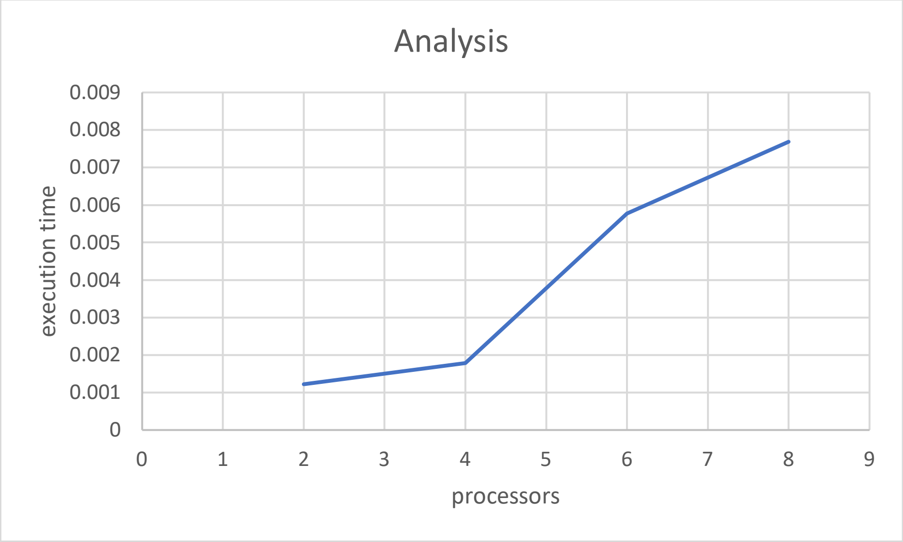
```
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 2 ./prefixsum.exe
0 sending to 1
Execution Time: 0.001221
1 receving..
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 4 ./prefixsum.exe
1 sending to 2
1 receving..
1 sending to 3
2 sending to 3
2 receving..
2 receving..
3 receving..
3 receving..
0 sending to 1
0 sending to 2
Execution Time: 0.001782
```



```
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 6 ./prefixsum.exe
0 sending to 1
0 sending to 2
0 sending to 4
Execution Time: 0.005775
3 sending to 4
3 receving..
3 sending to 5
3 receving..
1 sending to 2
1 receving..
1 sending to 3
1 sending to 5
5 receving..
5 receving..
5 receving..
4 sending to 5
4 receving..
4 receving..
4 receving..
2 sending to 3
2 receving..
2 sending to 4
2 receving..
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip>
```



```
PS C:\Users\Ashitra\OneDrive\Desktop\Mpip> mpiexec -n 8 ./prefixsum.exe
7 receving..
7 receving..
7 receving..
0 sending to 1
0 sending to 2
0 sending to 4
Execution Time: 0.007689
1 sending to 2
1 receving..
1 sending to 3
1 sending to 5
2 sending to 3
2 receving..
2 sending to 4
2 receving..
2 sending to 6
3 sending to 4
3 receving..
3 sending to 5
3 receving..
```


Analysis
execution time
processors