PARALLEL AND DISTRIBUTED COMPUTING
L – 19,20
LAB PROGRAMS
FACULTY : PROF. GAYATHRI R.
DHRUBANKA DUTTA, 17BCE1019

1. OpenMP basic programs

1A. HELLO WORLD

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

int main (int argc, char *argv[])
{
int nthreads, tid;

/*Fork a team of threads giving them their own copies of variables*/
#pragma omp parallel private (nthreads, tid)
{
/*Obtain thread number*/
tid = omp_get_thread_num();
printf("Hello world from thread = %d\n", tid);
/*Only master thread does this*/
if (tid==0)
{
nthreads = omp_get_num_threads();
printf("Number of threads = %d\n",  nthreads);
}
}
/*All threads join the master and disband*/
}
```

```
gcc -fopenmp pdclabHelloWorld.c && time ./a.out
Hello world from thread = 0
Number of threads = 4
Hello world from thread = 2
Hello world from thread = 1
Hello world from thread = 3

real	0m0.003s
user	0m0.005s
sys	0m0.000s
```

1B. ADDITION

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

int main (int argc, char *argv[])
{
int nthreads, a=4, b=5, c, tid;

/*Fork a team of threads giving them their own copies of variables*/
#pragma omp parallel private (nthreads, tid)
{
/*Obtain thread number*/
tid = omp_get_thread_num();
c = a+b;
printf("Result of addition from thread = %d\n", tid);
printf("Result of addition = %d\n", c);
/*Only master thread does this*/
if (tid==0)
{
nthreads = omp_get_num_threads();
printf("Number of threads = %d\n",  nthreads);
}
}
/*All threads join the master and disband*/
}
```

```
gcc -fopenmp pdclabSum.c && time ./a.out
Result of addition from thread = 0
Result of addition = 9
Number of threads = 4
Result of addition from thread = 3
Result of addition = 9
Result of addition from thread = 2
Result of addition = 9
Result of addition from thread = 1
Result of addition = 9

real    0m0.006s
user    0m0.010s
sys     0m0.000s
```

1C. SUBTRACTION

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

```
int main (int argc, char *argv[])
{
int nthreads, a=4, b=5, c, tid;

/*Fork a team of threads giving them their own copies of variables*/
#pragma omp parallel private (nthreads, tid)
{
/*Obtain thread number*/
tid = omp_get_thread_num();
c = a-b;
printf("Result of addition from thread = %d\n", tid);
printf("Result of addition = %d\n", c);
/*Only master thread does this*/
if (tid==0)
{
nthreads = omp_get_num_threads();
printf("Number of threads = %d\n",  nthreads);
}
}
/*All threads join the master and disband*/
}
```

```
gcc -fopenmp pdclabDifference.c && time ./a.out
Result of addition from thread = 1
Result of addition = -1
Result of addition from thread = 3
Result of addition = -1
Result of addition from thread = 0
Result of addition = -1
Number of threads = 4
Result of addition from thread = 2
Result of addition = -1

real	0m0.004s
user	0m0.010s
sys	0m0.000s
```

2. Private and shared constructs

2A. Find the sum of ‘n’ integers using the omp barrier.

CODE:

```
/*Exercise - Find the sum of 'n' integers using the omp barrier*/

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

int cum_sum (int i, int k) {
int indi_sum = 0;
for (int j=i;j<=k;j++)
indi_sum+=j;
return indi_sum;
}
int main () {
int sum = 0, n, sum1, sum2, sum3, sum4;
printf("Enter the number: ");
scanf("%d", &n);
#pragma omp parallel num_threads(4) shared(n) shared(sum) shared(sum1) shared(sum2)
shared(sum3) shared(sum4)
{
if (omp_get_thread_num() == 0)
sum1 = cum_sum(1,n/4);
else if (omp_get_thread_num() == 1)
sum2 = cum_sum(n/4+1, n/2);
else if (omp_get_thread_num() == 2)
sum3 = cum_sum(n/2+1, 3*n/4);
else if (omp_get_thread_num() == 3)
sum4 = cum_sum(3*n/4+1, n);

#pragma omp barrier
if (omp_get_thread_num() == 0) {
sum = sum1+sum2+sum3+sum4;
printf("The sum of the first n integers is %d\n", sum);
}
}
return 0;
}
```

OUTPUT :

```
[dhrubanka@dhrubanka-pc 17BCE1019_LAB2_PDC_18JULY]$ gcc -fopenmp sum_of_n_integers.c
&& ./a.out
Enter the number: 10
The sum of the first n integers is 55
```

```
[dhrubanka@dhrubanka-pc 17BCE1019_LAB2_PDC_18JULY]$ gcc -fopenmp sum_of_n_integers.c
&& ./a.out
Enter the number: 4
The sum of the first n integers is 10
[dhrubanka@dhrubanka-pc 17BCE1019_LAB2_PDC_18JULY]$ gcc -fopenmp sum_of_n_integers.c
&& ./a.out
Enter the number: 5
The sum of the first n integers is 15
[dhrubanka@dhrubanka-pc 17BCE1019_LAB2_PDC_18JULY]$ gcc -fopenmp sum_of_n_integers.c
&& ./a.out6
bash: ./a.out6: No such file or directory
[dhrubanka@dhrubanka-pc 17BCE1019_LAB2_PDC_18JULY]$ gcc -fopenmp sum_of_n_integers.c
&& ./a.out
Enter the number: 6
The sum of the first n integers is 21
```

2B. Create 4 threads for performing the below matrix operations using omp barrier.
i. Addition
ii. Subtraction
iii. Multiplication
iv. Division

CODE:

```
/*
2. Create 4 threads for performing the below matrix operations using omp barrier.
i. Addition
ii. Subtraction
iii. Multiplication
iv. Division
*/

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

int results1[3][3], results2[3][3], results3[3][3], results4[3][3];

void addition(int a[3][3], int n) {
for (int i=0;i<3;i++)
for (int j=0;j<3;j++)
results1[i][j] = a[i][j]+n;
}

void subtraction(int a[3][3], int n) {
for (int i=0;i<3;i++)
for (int j=0;j<3;j++)
results2[i][j] = a[i][j]-n;
}
```

```
void multiplication(int a[3][3], int n) {
for (int i=0;i<3;i++)
for (int j=0;j<3;j++)
results3[i][j] = a[i][j]*n;
}

void division(int a[3][3], int n) {
for (int i=0;i<3;i++)
for (int j=0;j<3;j++)
results4[i][j] = a[i][j]/n;
}

void print_matrix (int a[3][3]) {
for (int i=0;i<3;i++) {
for (int j=0;j<3;j++)
printf("%d ", a[i][j]);
printf("\n");
}

}
int main () {
int a[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}}, n=2;
#pragma omp parallel num_threads(4) shared(a,n,results1,results2,results3,results4)
{
if (omp_get_thread_num() == 0)
addition(a,n);
else if (omp_get_thread_num() == 1)
subtraction(a,n);
else if (omp_get_thread_num() == 2)
multiplication(a,n);
else if (omp_get_thread_num() == 3)
division(a,n);

#pragma omp barrier
if ( omp_get_thread_num() == 0 ) {
printf("Matrix after addition: \n");
print_matrix(results1);
}
else if ( omp_get_thread_num() == 1 ) {
printf("Matrix after subtraction: \n");
print_matrix(results2);
}
else if ( omp_get_thread_num() == 2 ) {
printf("Matrix after multiplication: \n");
print_matrix(results3);
}
else if ( omp_get_thread_num() == 3 ) {
printf("Matrix after division: \n");
print_matrix(results4);
}
}

return 0;
}
```

OUTPUT:

```
[dhrubanka@dhrubanka-pc 17BCE1019_LAB2_PDC_18JULY]$ gcc -fopenmp matrix_ops.c && ./a.out
Matrix after addition:
3 4 5
6 7 8
9 10 11
Matrix after subtraction:
-1 0 1
2 3 4
5 6 7
Matrix after division:
0 1 1
2 2 3
3 4 4
Matrix after multiplication:
2 4 6
8 10 12
14 16 18
```

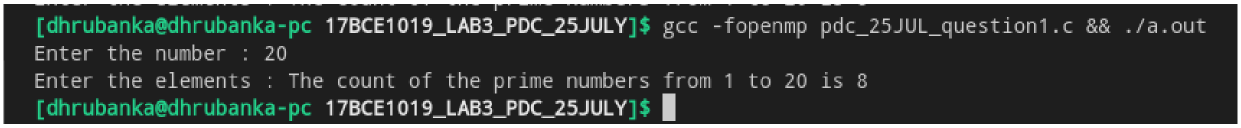
3. work-sharing loop constructs

3A. Write a OpenMP program to find the count of prime numbers from the given input using appropriate constraint

CODE :

```
/* QUESTION  -
Write a OpenMP program to find the count of prime numbers from the given
input using appropriate constraint
 */

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

int check_prime (int n) {
    int i, flag = 1;
    for(i = 2; i <= n/2; ++i)
    {
        if(n%i == 0)
        {
            flag = 0;
            break;
        }
    }
    return flag;

}

int main () {
    const int N = 10;
    int a[N], n, local_count, total_count = 0;

    printf("Enter the number : ");
    scanf("%d",&n);
    printf("Enter the elements : ");

    #pragma omp parallel private(local_count) shared(total_count)
    {
        local_count = 0;
        #pragma omp for
            for (int i=2;i<=n;i++) {
                if (check_prime(i))
                    local_count+=1;
            }
        #pragma omp critical
            total_count += local_count;
    }
    printf("The count of the prime numbers from 1 to %d is %d\n",n, total_count);
    return 0 ;
}
```

OUTPUT :



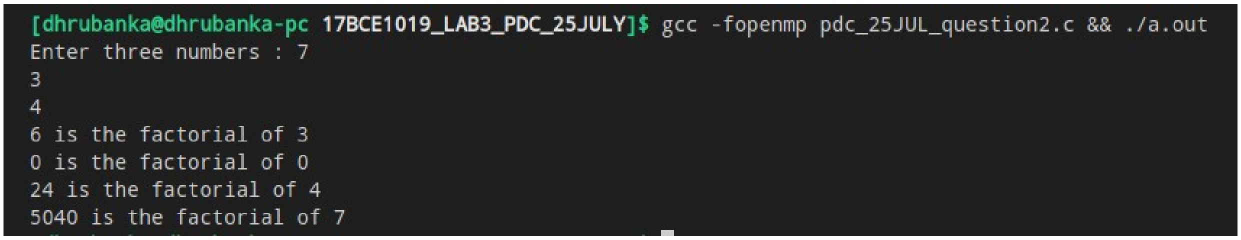
```
[dhrubanka@dhrubanka-pc 17BCE1019_LAB3_PDC_25JULY]$ gcc -fopenmp pdc_25JUL_question1.c && ./a.out
Enter the number : 20
Enter the elements : The count of the prime numbers from 1 to 20 is 8
[dhrubanka@dhrubanka-pc 17BCE1019_LAB3_PDC_25JULY]$ 
```

3B. Write a OpenMP program to find factorial of a 3 different numbers in separate code block. The output of all 3 factorial values should share a same variable through the critical section constraint.

CODE :

```
/* QUESTION -
Write a OpenMP program to find factorial of a 3 different numbers in separate
code block. The output of all 3 factorial values should share a same variable
through the critical section constraint.
*/

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

int factorial (int n) {
    if (n==0)
        return 1;
    return n*factorial(n-1);
}
int main () {
    int local_fact, inputs[3], number;
    //int *final_fact;
    printf("Enter three numbers : ");
    for (int i=0;i<3;i++) {
        scanf("%d", &inputs[i]);
    }
    #pragma omp parallel private(local_fact, number)
    {
        #pragma omp for
            for (int i=0;i<3;i++) {
                number = inputs[i];
                local_fact = factorial(number);
            }
        #pragma omp critical
            printf("%d is the factorial of %d\n", local_fact, number);
    }
    return 0;
}
```

OUTPUT :



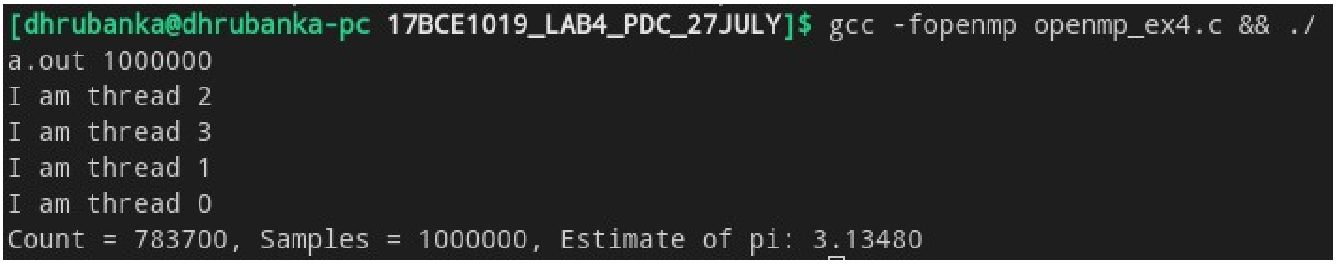
```
[dhrubanka@dhrubanka-pc 17BCE1019_LAB3_PDC_25JULY]$ gcc -fopenmp pdc_25JUL_question2.c && ./a.out
Enter three numbers : 7
3
4
6 is the factorial of 3
0 is the factorial of 0
24 is the factorial of 4
5040 is the factorial of 7
```

4. Reduction

CODE :

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

int main (int argc, char *argv[]) {
    int i , count; // points inside the unit quarter circle
    unsigned short xi[3]; // random  number seed
    int samples; // Samples number of points to generate
    double x,y; // coordinates of points
    double pi; // Estimate of pi

    samples = atoi(argv[1]);

    #pragma omp parallel
    {
        xi[0] = 1;
        xi[1] = 1;
        xi[2] = omp_get_thread_num();
        count = 0;
        printf("I am thread %d\n",xi[2]);
        #pragma omp for firstprivate(xi) private(x,y) reduction(+:count)
        for (i=0;i<samples;i++) {
            x = erand48(xi);
            y = erand48(xi);
            if(x*x+y*y <= 1.0)
                count++;
        }
    }
    pi = 4.0*(double)count / (double)samples;
        printf("Count = %d, Samples = %d, Estimate of pi: %7.5f\n", count, samples, pi);
}
```

OUTPUT :



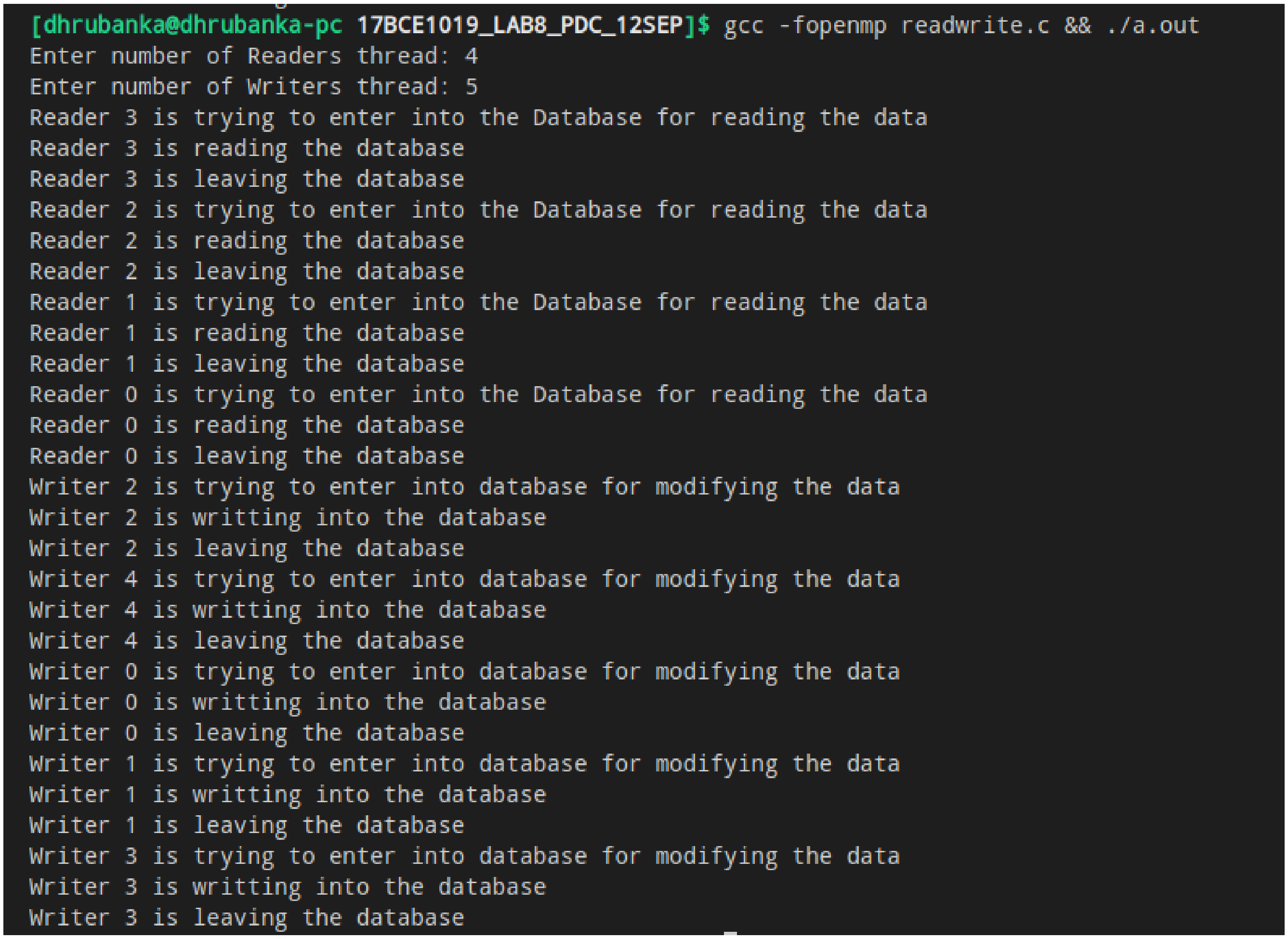
```
[dhrubanka@dhrubanka-pc 17BCE1019_LAB4_PDC_27JULY]$ gcc -fopenmp openmp_ex4.c && ./
a.out 1000000
I am thread 2
I am thread 3
I am thread 1
I am thread 0
Count = 783700, Samples = 1000000, Estimate of pi: 3.13480
```

5. Search engine master-worker pattern

CODE :

```
#include<stdio.h>
#include <time.h>
#include <unistd.h>
#include <omp.h>

int main()
{
 int i=0,NumberofReaderThread=0,NumberofWriterThread;

omp_lock_t writelock;
omp_init_lock(&writelock);

int readCount=0;

 printf("Enter number of Readers thread: ");
 scanf("%d",&NumberofReaderThread);
 printf("Enter number of Writers thread: ");
 scanf("%d",&NumberofWriterThread);

int tid=0;
#pragma omp parallel
#pragma omp for

 for(i=0;i<NumberofReaderThread;i++)
 {
    printf("Reader %d is trying to enter into the Database for reading the data\n",i);


    omp_set_lock(&writelock);
    readCount++;
    if(readCount==1)
    {

      printf("Reader %d is reading the database\n",i);
    }

    omp_unset_lock(&writelock);
    readCount--;
    if(readCount==0)
    {
      printf("Reader %d is leaving the database\n",i);
    }
 }
#pragma omp parallel shared(tid)// Specifies that one or more variables should be shared among all threads.
#pragma omp for nowait     //If there are multiple independent loops within a parallel region
 for(i=0;i<NumberofWriterThread;i++)
 {
    printf("Writer %d is trying to enter into database for modifying the data\n",i);

    omp_set_lock(&writelock);

    printf("Writer %d is writting into the database\n",i);
    printf("Writer %d is leaving the database\n",i);

    omp_unset_lock(&writelock);
 }

  omp_destroy_lock(&writelock);
 return 0;
}
```

OUTPUT :



```
[dhrubanka@dhrubanka-pc 17BCE1019_LAB8_PDC_12SEP]$ gcc -fopenmp readwrite.c && ./a.out
Enter number of Readers thread: 4
Enter number of Writers thread: 5
Reader 3 is trying to enter into the Database for reading the data
Reader 3 is reading the database
Reader 3 is leaving the database
Reader 2 is trying to enter into the Database for reading the data
Reader 2 is reading the database
Reader 2 is leaving the database
Reader 1 is trying to enter into the Database for reading the data
Reader 1 is reading the database
Reader 1 is leaving the database
Reader 0 is trying to enter into the Database for reading the data
Reader 0 is reading the database
Reader 0 is leaving the database
Writer 2 is trying to enter into database for modifying the data
Writer 2 is writting into the database
Writer 2 is leaving the database
Writer 4 is trying to enter into database for modifying the data
Writer 4 is writting into the database
Writer 4 is leaving the database
Writer 0 is trying to enter into database for modifying the data
Writer 0 is writting into the database
Writer 0 is leaving the database
Writer 1 is trying to enter into database for modifying the data
Writer 1 is writting into the database
Writer 1 is leaving the database
Writer 3 is trying to enter into database for modifying the data
Writer 3 is writting into the database
Writer 3 is leaving the database
```

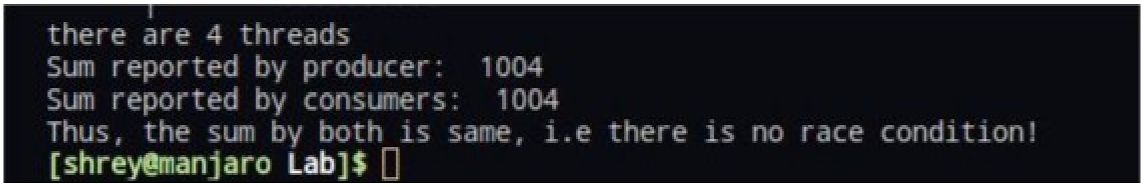
6. OpenMP producer consumer

CODE :

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


#define MAXWORK 40

int work[MAXWORK],  // work to be done
    nwork=0,  // number of items in the queue
    nextput=0,  // producer will place number # at work[nextput]
    nextget=-1,  // consumer will obtain next # at work[nextget]
    breaksum,  // sum after which everyone stops
    done = 0,  // value 1 signals producer exceeded breaksum
    psum,csum,  // sums found by producer, consumers
    pwork,  // work done by producer
    *cwork,  // work done by the consumers
    nth,  // number of threads
    debugflag;  // 1 if debug

void next(int *m)
{  (*m)++;
   if (*m >= MAXWORK) *m = 0;
}

void putwork(int k)
{  int put = 0;
   while (!put)  {
      if (nwork < MAXWORK)  {
         #pragma omp critical
         {  work[nextput] = k;
            if (nwork == 0) nextget = nextput;
            next(&nextput);
            nwork++;
            put = 1;
         }
      }
      else sched_yield();
   }
}

int getwork()
```

```
{  int k,get=0;
   while (!get)  {
      if (done && nwork == 0) return -1;
      if (nwork > 0)  {
         #pragma omp critical
         {
            if (nwork > 0)  {
               k = work[nextget];
               next(&nextget);
               nwork--;
               if (nwork == 0) nextget = -1;
               get = 1;
            }
         }
      }
      else sched_yield();
   }
   return k;
}

void dowork()
{
   #pragma omp parallel
   {  int me = omp_get_thread_num(),
          num;
      #pragma omp single
      {  int i;
         nth = omp_get_num_threads();
         printf("there are %d threads\n",nth);
         cwork = (int *) malloc(nth*sizeof(int));
         for (i = 1; i < nth; i++) cwork[i]=0;
      }
      if (me == 0 && debugflag) {int wait=0; while (!wait) ; }
      #pragma omp barrier
      if (me == 0)  {  // I'm the producer
         pwork = 0;
         while (1)  {
            num = rand() % 100;
            putwork(num);
            psum += num;
            pwork++;
            if (psum > breaksum)  {
               done = 1;
               return;
            }
         }
      }
      else  {  // I'm a consumer
         while (1)  {
            num = getwork();
            if (num == -1) return;
            cwork[me]++;
            #pragma omp atomic
            csum += num;
         }
      }
   }
}

int main(int argc, char **argv)
{  int i;
   breaksum = atoi(argv[1]);
   debugflag = atoi(argv[2]);
   dowork();
   printf("sum reported by producer:  %d\n",psum);
   printf("sum reported by consumers:  %d\n",csum);
   printf("work done by producer:  %d\n",pwork);
   printf("work done by consumers:\n");
   for (i = 1; i < nth; i++)
      printf("%d\n",cwork[i]);
}
```

OUTPUT :



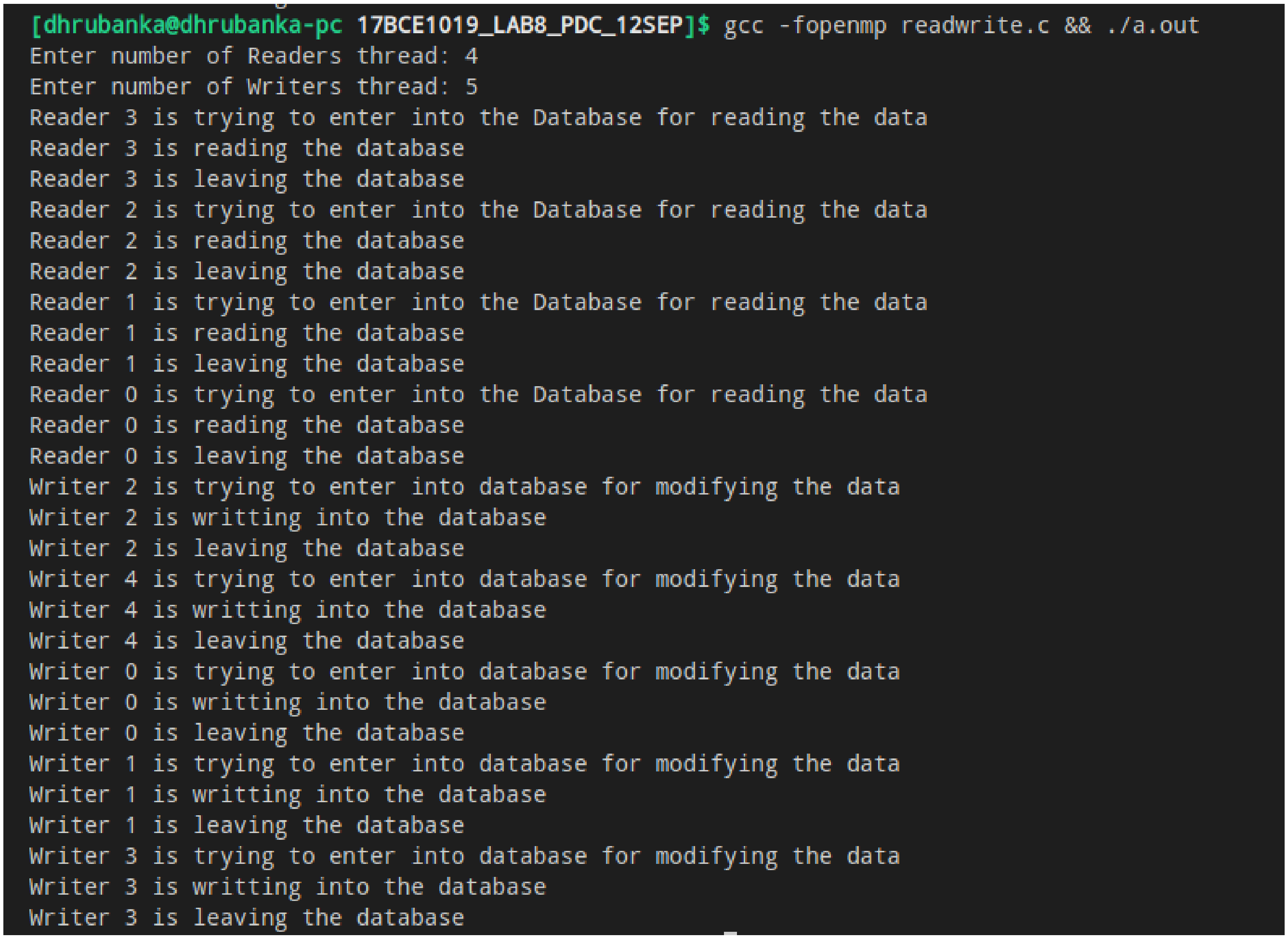
```
there are 4 threads
Sum reported by producer:  1004
Sum reported by consumers:  1004
Thus, the sum by both is same, i.e there is no race condition!
[shrey@manjaro Lab]$ 
```

7. Remote Method Invocation

CODE :

```
#include<stdio.h>
#include <time.h>
#include <unistd.h>
#include <omp.h>

int main()
{
 int i=0,NumberofReaderThread=0,NumberofWriterThread;

omp_lock_t writelock;
omp_init_lock(&writelock);

int readCount=0;

 printf("Enter number of Readers thread: ");
 scanf("%d",&NumberofReaderThread);
 printf("Enter number of Writers thread: ");
 scanf("%d",&NumberofWriterThread);

int tid=0;
#pragma omp parallel
#pragma omp for

 for(i=0;i<NumberofReaderThread;i++)
 {
    printf("Reader %d is trying to enter into the Database for reading the data\n",i);


    omp_set_lock(&writelock);
    readCount++;
    if(readCount==1)
    {

      printf("Reader %d is reading the database\n",i);
    }

    omp_unset_lock(&writelock);
    readCount--;
    if(readCount==0)
    {
      printf("Reader %d is leaving the database\n",i);
    }
 }
#pragma omp parallel shared(tid)// Specifies that one or more variables should be shared among all threads.
#pragma omp for nowait     //If there are multiple independent loops within a parallel region
 for(i=0;i<NumberofWriterThread;i++)
 {
    printf("Writer %d is trying to enter into database for modifying the data\n",i);

    omp_set_lock(&writelock);

    printf("Writer %d is writting into the database\n",i);
    printf("Writer %d is leaving the database\n",i);

    omp_unset_lock(&writelock);
 }

  omp_destroy_lock(&writelock);
 return 0;
}
```

OUTPUT :



```
[dhrubanka@dhrubanka-pc 17BCE1019_LAB8_PDC_12SEP]$ gcc -fopenmp readwrite.c && ./a.out
Enter number of Readers thread: 4
Enter number of Writers thread: 5
Reader 3 is trying to enter into the Database for reading the data
Reader 3 is reading the database
Reader 3 is leaving the database
Reader 2 is trying to enter into the Database for reading the data
Reader 2 is reading the database
Reader 2 is leaving the database
Reader 1 is trying to enter into the Database for reading the data
Reader 1 is reading the database
Reader 1 is leaving the database
Reader 0 is trying to enter into the Database for reading the data
Reader 0 is reading the database
Reader 0 is leaving the database
Writer 2 is trying to enter into database for modifying the data
Writer 2 is writting into the database
Writer 2 is leaving the database
Writer 4 is trying to enter into database for modifying the data
Writer 4 is writting into the database
Writer 4 is leaving the database
Writer 0 is trying to enter into database for modifying the data
Writer 0 is writting into the database
Writer 0 is leaving the database
Writer 1 is trying to enter into database for modifying the data
Writer 1 is writting into the database
Writer 1 is leaving the database
Writer 3 is trying to enter into database for modifying the data
Writer 3 is writting into the database
Writer 3 is leaving the database
```

8. Solving Sudoku puzzle

CODE :

```
#include "sudoku.h"
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int empty_cells=0;

struct pair_t{
	int x,y;
};
typedef struct pair_t pair;
pair empty_cells_list[SIZE*SIZE];

struct queue_element{
	int board[SIZE][SIZE];
	int forward_map[SIZE][SIZE][SIZE];
};

struct queue
{
	struct queue_element **list;
	int size,length;
	int head,tail;
};

int forward_map[SIZE][SIZE][SIZE];
int found = 0;
int **final_board;
int reverse_map_row[SIZE][SIZE][SIZE],reverse_map_column[SIZE][SIZE][SIZE];
int reverse_map_box[SIZE][SIZE][MINIGRIDSIZE][MINIGRIDSIZE];

void init_queue(struct queue *q, int length)
{
	q->list = malloc(sizeof(struct queue_element*)*length);
	q->head=0;
	q->tail=0;
	q->length=0;
	q->size=length;
}

void push(struct queue *q, struct queue_element *elem)
{
    if((q->tail+1)%q->size==(q->head))
    {
        printf("Queue full!!! PAAANIIIC\n");
        exit(0);
    }
    q->list[q->tail] = elem;
    q->tail = ((q->tail)+1)%(q->size);
    q->length++;
}

struct queue_element* pop(struct queue *q)
{
    if(q->head == q->tail)
    {
        printf("Queue empty!!! PAAANIIIC\n");
        exit(0);
    }
    struct queue_element* ret = q->list[q->head];
    q->head = ((q->head)+1)%(q->size);
    q->length--;
}

void populate_f(int x, int y, int val, int map[SIZE][SIZE][SIZE])
{
    if(val!=0)
    {
        int j,k;
        for(j=0;j<SIZE;j++)
        {
            if(j!=x)
                map[j][y][val-1]=1;
            if(j!=y)
                map[x][j][val-1]=1;
            if(j!=val-1)
                map[x][y][j]=1;
        }
        j=x-x%MINIGRIDSIZE;
        k=y-y%MINIGRIDSIZE;

        int jj,kk;
        for(jj=j;jj<j+MINIGRIDSIZE;jj++)
            for(kk=k;kk<k+MINIGRIDSIZE;kk++)
                if(jj!=x || kk!=y)
                    map[jj][kk][val-1]=1;

    }
}

void dfs_populate_f(int x, int y, int val, int map[SIZE][SIZE][SIZE])
```

```
{
    if(val!=0)
    {
        int j,k;
        for(j=0;j<SIZE;j++)
        {
            if(j!=x)
                map[j][y][val-1]=1;
            if(j!=y)
                map[x][j][val-1]=1;
            // if(j!=val-1)
            //  map[x][y][j]=1;
        }
        j=x-x%MINIGRIDSIZE;
        k=y-y%MINIGRIDSIZE;

        int jj,kk;
        for(jj=j;jj<j+MINIGRIDSIZE;jj++)
            for(kk=k;kk<k+MINIGRIDSIZE;kk++)
                if(jj!=x || kk!=y)
                    map[jj][kk][val-1]=1;

    }
}

int findPosition(int x, int y)
{
    int i,single=0,idx=-1;
    for(i=0; i<SIZE; i++)
    {
        if(single)
        {
            if(forward_map[x][y][i] == 0)
                return -1;
        }
        else
        {
            if(forward_map[x][y][i] == 0)
            {
                single = 1;
                idx = i+1;
            }
        }
    }
    return idx;
}

int findCol(int x, int y, int **inp)
{
    int i,single=0,idx=-1;
    for(i=0; i<SIZE; i++)
    {
        if(single)
        {
            if(forward_map[x][i][y] == 0 && inp[x][i]==0)
                return -1;
        }
        else
        {
            if(forward_map[x][i][y] == 0 && inp[x][i]==0)
            {
                single = 1;
                idx = i;
            }
        }
    }
    return idx;
}

int findRow(int x, int y, int **inp)
{
    int i,single=0,idx=-1;
    for(i=0; i<SIZE; i++)
    {
        if(single)
        {
            if(forward_map[i][x][y] == 0 && inp[i][x]==0)
                return -1;
        }
        else
        {
            if(forward_map[i][x][y] == 0 && inp[i][x]==0)
            {
                single = 1;
                idx = i;
            }
        }
    }
    return idx;
}

pair findCell(int x, int y, int val, int **inp)
{
    int i,j,single=0;
    pair idx;
    idx.x=-1;
    for(i=x; i<x+MINIGRIDSIZE; i++)
    {
        for(j=y; j<y+MINIGRIDSIZE; j++)
        {
            if(single)
            {
                if(forward_map[i][j][val] == 0 && inp[i][j]==0)
                {
                    idx.x=-1;
                    idx.y=-1;
                    return idx;
                }
            }
            else
            {
                if(forward_map[i][j][val] == 0 && inp[i][j]==0)
                {
                    single = 1;
                    idx.x = i;
                    idx.y = j;
                }
            }
        }
    }
    return idx;
}

void dfs(int board[SIZE][SIZE], int forward_map[SIZE][SIZE][SIZE], int idx)
{
    // printf("idx=%d\n", idx);
    if(idx==empty_cells)
    {
        printf("found!!\n");
        found = 1;
        int i,j;
        for(i=0;i<SIZE;i++)
        {
            for(j=0;j<SIZE;j++)
                printf("%d ",board[i][j]);
            printf("\n");
        }
        // printf("sizeof(board)=%lx\n",sizeof(board));
        // memcpy(final_board,board,sizeof(board));
        for(i=0;i<SIZE;i++)
            for(j=0;j<SIZE;j++)
                final_board[i][j]=board[i][j];
        // printf("returning\n");
        return;
    }
    else if(found)
        return;
    else
    {
        int val;
        for(val=0; val<SIZE; val++)
        {
            int x = empty_cells_list[idx].x;
            int y = empty_cells_list[idx].y;
            if(forward_map[x][y][val]==0)
            {
                int row_bkp[SIZE], col_bkp[SIZE], box_bkp[MINIGRIDSIZE][MINIGRIDSIZE], val_bkp[SIZE];

                int j,k;
                for(j=0;j<SIZE;j++)
                {
                    if(j!=x)
                        col_bkp[j]=forward_map[j][y][val];
                    if(j!=y)
                        row_bkp[j]=forward_map[x][j][val];
                }
                j=x-x%MINIGRIDSIZE;
                k=y-y%MINIGRIDSIZE;

                int jj,kk;
                for(jj=j;jj<j+MINIGRIDSIZE;jj++)
                    for(kk=k;kk<k+MINIGRIDSIZE;kk++)
                        if(jj!=x || kk!=y)
                            box_bkp[jj-j][kk-k]=forward_map[jj][kk][val];

                board[x][y]=val+1;
                dfs_populate_f(x,y,val+1,forward_map);
                dfs(board,forward_map,idx+1);

                for(j=0;j<SIZE;j++)
                {
                    if(j!=x)
                        forward_map[j][y][val]=col_bkp[j];
                    if(j!=y)
                        forward_map[x][j][val]=row_bkp[j];
                }
                j=x-x%MINIGRIDSIZE;
                k=y-y%MINIGRIDSIZE;

                for(jj=j;jj<j+MINIGRIDSIZE;jj++)
                    for(kk=k;kk<k+MINIGRIDSIZE;kk++)
                        if(jj!=x || kk!=y)
                            forward_map[jj][kk][val]=box_bkp[jj-j][kk-k];

                board[x][y]=0;
```

```
			}
		}
	}
}

int **solveSudoku(int ** inp)
{
	final_board=malloc(sizeof(int*)*SIZE);
	int i,j;
	for(i=0;i<SIZE;i++)
		final_board[i]=malloc(sizeof(int)*SIZE);
	memset(forward_map,0,sizeof(forward_map));
	memset(reverse_map_row,0,sizeof(reverse_map_row));
	memset(reverse_map_column,0,sizeof(reverse_map_column));
	memset(reverse_map_box,0,sizeof(reverse_map_box));


	//#pragma omp parallel for
	for(i=0;i<SIZE*SIZE;i++)
	{
		// printf("i=%d\n", i);
		int x=i%SIZE;
		int y=i/SIZE;
		int val=inp[x][y];
		if(val>0)
			populate_f(x,y,val,forward_map);
	}
	int changed_outer=1;
	while(changed_outer)
	{
		int changed=1;
		changed_outer=0;
		while(changed)
		{
			changed=0;
			// #pragma omp parallel
			{
	//ELIMINATION
				// #pragma omp for
				for(j=0;j<SIZE*SIZE;j++)
				{
					int x=j%SIZE, y=j/SIZE;
					if(inp[x][y]!=0)
					{
						// printf("x=%d,y=%d\n", x,y);
						continue;
					}
					int pos = findPosition(x,y);
					if(pos>0)
```

```
                    {
                        inp[x][y]=pos;
                        printf("Elimination at x=%d, y=%d, val=%d\n", x,y,pos);
                        populate_f(x,y,pos,forward_map);
                        changed=1;
                    }
                }
                // #pragma omp for
                for(j=0;j<SIZE;j++)
                {
                    int p;
                    for(p=0;p<SIZE;p++)
                    {
    //LONE-RANGER-ROW
                        int pos = findCol(j,p,inp);
                        if(pos>=0)
                        {
                            inp[j][pos]=p+1;
                            printf("Lone ranger 1 at x=%d, y=%d, val=%d\n",
j,pos,p+1);

                            populate_f(j,pos,p+1,forward_map);
                            changed=1;
                        }

    //LONE-RANGER-COL
                        pos = findRow(j,p,inp);
                        if(pos>=0)
                        {
                            inp[pos][j]=p+1;
                            printf("Lone ranger 2 at x=%d, y=%d, val=%d\n",
pos,j,p+1);

                            populate_f(pos,j,p+1,forward_map);
                            changed=1;
                        }

    //LONE_RANGER-BOX
                        int x=(j%MINIGRIDSIZE)*MINIGRIDSIZE,
y=(j/MINIGRIDSIZE)*MINIGRIDSIZE;
                        pair p1 = findCell(x,y,p,inp);
                        if(p1.x>=0)
                        {
                            inp[p1.x][p1.y]=p+1;
                            printf("Lone ranger 3 at x=%d, y=%d, val=%d\n",
p1.x,p1.y,p+1);

                            populate_f(p1.x,p1.y,p+1,forward_map);
                            changed=1;
                        }
                    }
                }
```

```
            }
        }
// TWINS ROWS
        int k;
        for(k=0;k<SIZE;k++)
        {
            for(i=0;i<SIZE;i++)
            {
                for(j=i+1;j<SIZE;j++)
                {
                    int l,cnt=0,first=0,second=0,position[2];
                    for(l=0;l<SIZE;l++)
                    {
                        if(forward_map[k][l][i]==0){
                            first|=1<<l;
                            if(cnt<=1)
                                position[cnt]=l;
                            else
                            {
                                cnt=0;
                                break;
                            }
                            cnt++;

                        }
                        if(forward_map[k][l][j]==0)
                            second|=1<<l;
                    }
                    if(cnt==2 && second==first)
                    {
                        for(l=0;l<SIZE;l++)
                        {
                            forward_map[k][position[0]][l]=1;
                            forward_map[k][position[1]][l]=1;
                        }
                        forward_map[k][position[0]][j]=0;
                        forward_map[k][position[0]][i]=0;
                        forward_map[k][position[1]][j]=0;
                        forward_map[k][position[1]][i]=0;
                    }
                }
            }
        }
// TWINS COLUMNS
        for(k=0;k<SIZE;k++)
        {
            for(i=0;i<SIZE;i++)
            {
                for(j=i+1;j<SIZE;j++)
                {
                    int l,cnt=0,first=0,second=0,position[2];
                    for(l=0;l<SIZE;l++)
                    {
                        if(forward_map[l][k][i]==0){
                            first|=1<<l;
                            if(cnt<=1)
                                position[cnt]=l;
                            else
                            {
                                cnt=0;
                                break;
                            }
                            cnt++;

                        }
                        if(forward_map[l][k][j]==0)
                            second|=1<<l;
                    }
                    if(cnt==2 && second==first)
                    {
                        for(l=0;l<SIZE;l++)
                        {
                            forward_map[position[0]][k][l]=1;
                            forward_map[position[1]][k][l]=1;
                        }
                        forward_map[position[0]][k][j]=0;
                        forward_map[position[0]][k][i]=0;
                        forward_map[position[1]][k][j]=0;
                        forward_map[position[1]][k][i]=0;
                    }
                }
            }
        }
        // TWINS GRID

        for(k=0;k<SIZE;k++)
        {
            int x=(k%MINIGRIDSIZE)*MINIGRIDSIZE,
y=(k/MINIGRIDSIZE)*MINIGRIDSIZE;
            for(i=0;i<SIZE;i++)
            {
                for(j=i+1;j<SIZE;j++)
                {
                    int l1,l2,cnt=0,first=0,second=0,positionx[2],positiony[2];
                    for(l1=x;l1<x+MINIGRIDSIZE;l1++)
                        for(l2=y;l2<y+MINIGRIDSIZE;l2++)
                        {
```

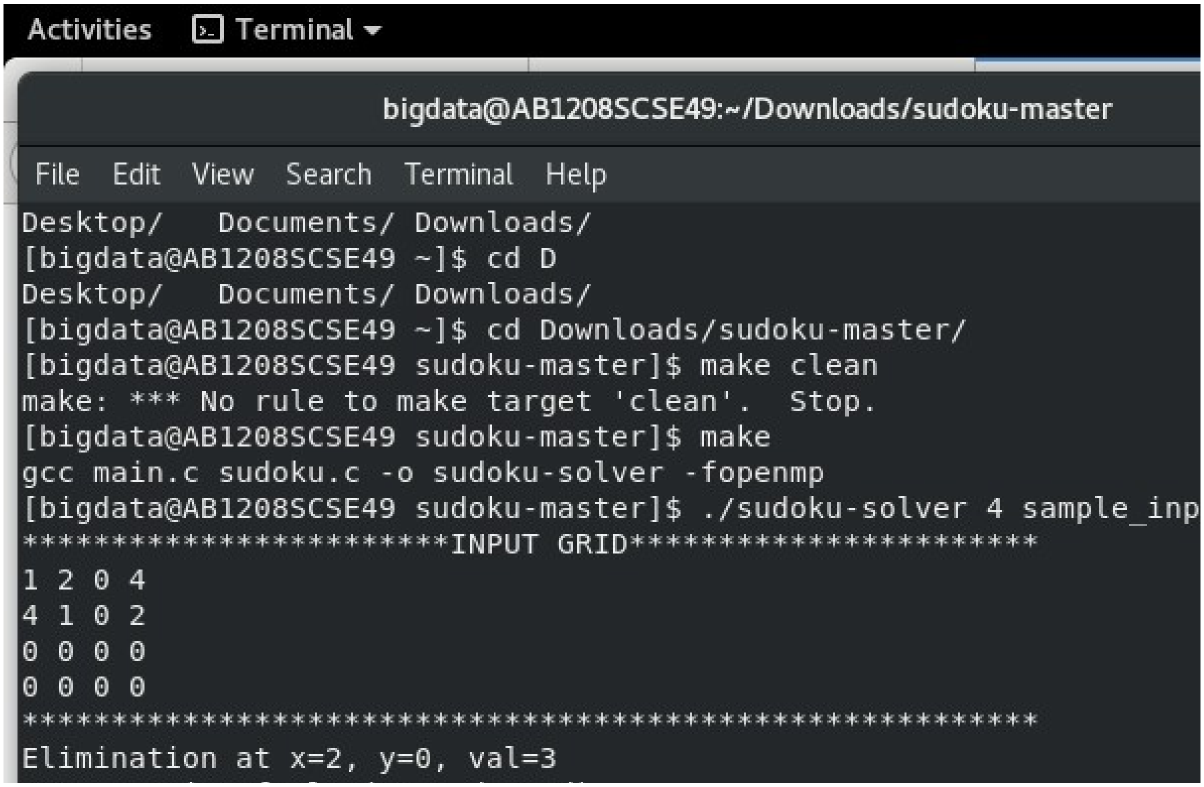
```
                        if(forward_map[l1][l2][i]==0){
                            first|=1<<(l1*MINIGRIDSIZE+l2);
                            if(cnt<=1)
                            {
                                positionx[cnt]=l1;
                                positiony[cnt]=l2;
                            }
                            else
                            {
                                cnt=0;
                                break;
                            }
                            cnt++;

                        }
                        if(forward_map[l1][l2][j]==0)
                            second|=1<<(l1*MINIGRIDSIZE+l2);
                    }
                if(cnt==2 && second==first)
                {
                    for(l1=0;l1<SIZE;l1++)
                    {
                        forward_map[positionx[0]][positiony[0]][l1]=1;
                        forward_map[positionx[1]][positiony[1]][l1]=1;
                    }
                    forward_map[positionx[0]][positiony[0]][j]=0;
                    forward_map[positionx[0]][positiony[0]][i]=0;
                    forward_map[positionx[1]][positiony[1]][j]=0;
                    forward_map[positionx[1]][positiony[1]][i]=0;
                }
            }
        }
    }
}
final_board=inp;
for(i=0;i<SIZE;i++)
{
    for(j=0;j<SIZE;j++)
        printf("%d ",inp[i][j]);
    printf("\n");
}

// #pragma omp parallel for
for(i=0; i<SIZE*SIZE; i++)
{
    int x=i%SIZE, y=i/SIZE;
    if(inp[x][y]==0)
    {
        int ind;
```

```
                // #pragma omp critical
                ind=empty_cells++;
            empty_cells_list[ind].x=x;
            empty_cells_list[ind].y=y;
        }
    }
    printf("empty cells %d\n",empty_cells);
    struct queue *q = malloc(sizeof(struct queue));
    int num_threads;
    int idx=0;
    #pragma omp parallel shared(q)
    {

        int k;
        #pragma omp single
        {
            num_threads = omp_get_num_threads();
            init_queue(q, num_threads*SIZE);
            struct queue_element *elem;
            elem = malloc(sizeof(struct queue_element));
            // memcpy((elem->board),(inp),sizeof(inp));
            for(i=0;i<SIZE;i++)
                for(j=0;j<SIZE;j++)
                    elem->board[i][j]=inp[i][j];

            // memcpy(elem->forward_map,forward_map,sizeof(forward_map));
            for(i=0;i<SIZE;i++)
                for(j=0;j<SIZE;j++)
                    for(k=0;k<SIZE;k++)
                        elem->forward_map[i][j][k]=forward_map[i][j][k];

            push(q,elem);
            while(q->length < num_threads && idx<empty_cells)
            {
                int l=q->length;
                int i;
                struct queue *tmp=malloc(sizeof(struct queue));
                init_queue(tmp,l*SIZE);
                for(i=0;i<l;i++)
                {
                    int j;
                    for(j=0;j<SIZE;j++)
                    {
                        if(forward_map[empty_cells_list[idx].x]
[empty_cells_list[idx].y][j]==0)
                        {
                            int a,b,c;
                            struct queue_element * temp=malloc(sizeof(struct
queue_element));
```

```
                            // memcpy(temp->board,q->list[i]-
>board,sizeof(temp->board));
                            for(a=0;a<SIZE;a++)
                                for(b=0;b<SIZE;b++)
                                    temp->board[a][b]=q->list[i]-
>board[a][b];


                            // memcpy(temp->forward_map,q->list[i]-
>forward_map,sizeof(temp->forward_map));
                            for(a=0;a<SIZE;a++)
                                for(b=0;b<SIZE;b++)
                                    for(c=0;c<SIZE;c++)
                                        temp->forward_map[a][b]
[c]=q->list[i]->forward_map[a][b][c];

                            temp->board[empty_cells_list[idx].x]
[empty_cells_list[idx].y]=j+1;

    populate_f(empty_cells_list[idx].x,empty_cells_list[idx].y,j+1,temp->forward_map);
                            push(tmp,temp);
                        }
                    }
                }
                idx++;
                free(q);
                q=tmp;
            }

        }
        // init_queue(&q, );
        #pragma omp for schedule(dynamic,1)
        for(i=0; i<q->length;i++)
        {
            printf("dfs i=%d\n", i);
            // for(k=0;k<SIZE;k++)
            // {
            //      for(j=0;j<SIZE;j++)
            //          printf("%d ",q->list[i]->board[k][j]);
            //      printf("\n");
            // }

            dfs(q->list[i]->board,q->list[i]->forward_map,idx);
        }
    }
//BRUTE-FORCE
    for(i=0;i<SIZE;i++)
    {
        for(j=0;j<SIZE;j++)
```

```
                        printf("%d ",final_board[i][j]);
                printf("\n");
        }
        return final_board;
}
```

OUTPUT :



```
Desktop/   Documents/ Downloads/
[bigdata@AB1208SCSE49 ~]$ cd D
Desktop/   Documents/ Downloads/
[bigdata@AB1208SCSE49 ~]$ cd Downloads/sudoku-master/
[bigdata@AB1208SCSE49 sudoku-master]$ make clean
make: *** No rule to make target 'clean'.  Stop.
[bigdata@AB1208SCSE49 sudoku-master]$ make
gcc main.c sudoku.c -o sudoku-solver -fopenmp
[bigdata@AB1208SCSE49 sudoku-master]$ ./sudoku-solver 4 sample_inp
************************INPUT GRID***********************
1 2 0 4
4 1 0 2
0 0 0 0
0 0 0 0
*********************************************************
Elimination at x=2, y=0, val=3
```

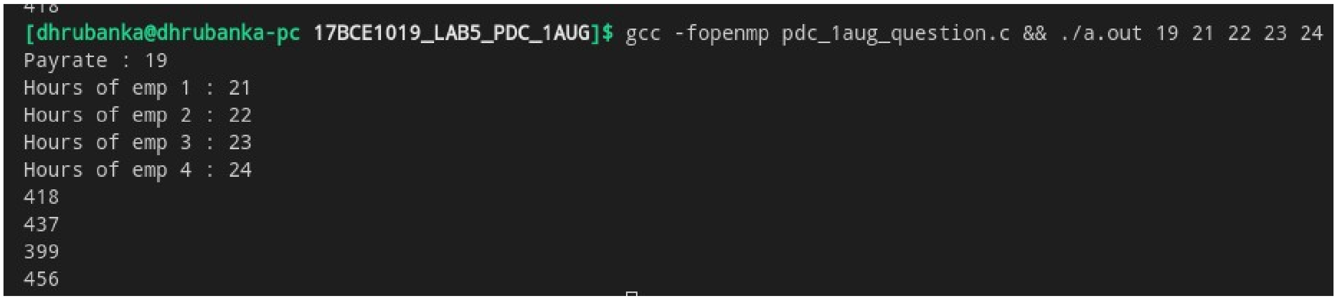
9. MPI basic programs

Exercise 1:Execute simple MPI program for finding the employee payroll processing by providing no.of Hours worked and pay rate as input. Assume the number of employees to be 4. Observe the time taken for calculating payroll of each employee.

CODE :

```
/*Execute simple MPI program for finding the employee payroll
processing by providing no.of Hours worked and pay rate as input.
Assume the number of employees to be 4. Observe the time taken for
calculating payroll of each employee. */

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

float return_sal(int hours, float rate) {
    float sal;
    sal = rate*hours;
}

int main(int argc, char** argv) {
    int emps[4], payrate = atoi(argv[1]), salary;
    printf("Payrate : %d\n", payrate);
    for (int i=1;i<argc;i++)
        emps[i-1] = atoi(argv[i]);
    for(int i=1;i<argc-1;i++)
        printf("Hours of emp %d : %d\n",i,emps[i]);

    #pragma omp parallel private(salary)
    {
        #pragma omp for
            for (int i=1;i<argc-1;i++)
                salary = return_sal(emps[i], payrate);

        #pragma omp critical
            printf("%d \n",salary);
    }

}
```

OUTPUT :



```
418
[dhrubanka@dhrubanka-pc 17BCE1019_LAB5_PDC_1AUG]$ gcc -fopenmp pdc_1aug_question.c && ./a.out 19 21 22 23 24
Payrate : 19
Hours of emp 1 : 21
Hours of emp 2 : 22
Hours of emp 3 : 23
Hours of emp 4 : 24
418
437
399
456
```

10. MPI send and receive primitives

CODE :

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

int main(int argc, char** argv) {
  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // We are assuming at least 2 processes for this task
  if (world_size < 2) {
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  int number;
  if (world_rank == 0) {
    // If we are rank 0, set the number to -1 and send it to process 1
    number = -1;
    MPI_Send(
      /* data         = */ &number,
      /* count        = */ 1,
      /* datatype     = */ MPI_INT,
      /* destination  = */ 1,
      /* tag          = */ 0,
      /* communicator = */ MPI_COMM_WORLD);
  } else if (world_rank == 1) {
    MPI_Recv(
      /* data         = */ &number,
      /* count        = */ 1,
      /* datatype     = */ MPI_INT,
      /* source       = */ 0,
      /* tag          = */ 0,
      /* communicator = */ MPI_COMM_WORLD,
      /* status       = */ MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n", number);
  }
  MPI_Finalize();
}
```

OUTPUT :

```
mpirun -n 2 ./send_recv
Process 1 received number -1 from process 0
```

11. MPI synchronization

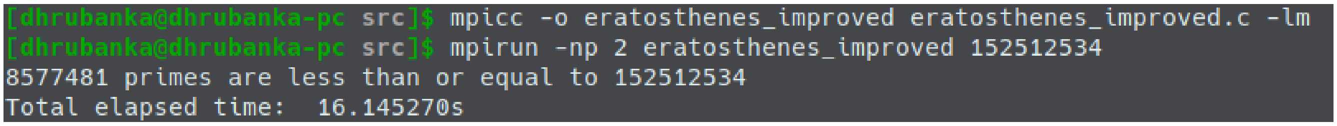
CODE :

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


int main(int argc, char** argv)                {
   int    count;              /* local prime count */
   double  elapsed_time;         /* parallel execution time */
   int    first;              /* index of first multiple */
   int    global_count;        /* global prime count */
   int    high_value;          /* highest value on this proc */
   int    i;
   int    id;                 /* process id number */
   int    index;              /* index of current prime */
   int    low_value;           /* lowest value on this proc */
   int    n;                  /* sieving from 2, ..., n */
   int    p;                  /* number of processes */
   int    proc0_size;          /* size of proc 0's subarray */
   int    prime;              /* current prime */
   int    size;               /* elements in marked string */
   int    first_value_index;
   int    prime_step;
   int    prime_doubled;
   int    sqrt_n;
   int    prime_multiple;
   int    num_per_block;
   int    block_low_value;
   int    block_high_value;
   int    first_index_in_block;
   char*   marked;             /* portion of 2, ..., n */
   char*   primes;

   MPI_Init(&argc, &argv);

   /* start the timer */
   MPI_Barrier(MPI_COMM_WORLD);
   elapsed_time = -MPI_Wtime();

   MPI_Comm_rank(MPI_COMM_WORLD, &id);
   MPI_Comm_size(MPI_COMM_WORLD, &p);

   if (argc != 2)    {
       if (id == 0) /* parent process */
           printf("Command line: %s <m>\n", argv[0]);
       MPI_Finalize();
       exit(1);
   } /* if (argc != 2) */

   n = atoi(argv[1]);

   /*
    * Figure out this process's share of the array, as well as the
    * integers represented by the first and last array elements
    */
   low_value  = BLOCK_FIRST + BLOCK_LOW(id, p, n - 1)  * BLOCK_STEP;
   high_value = BLOCK_FIRST + BLOCK_HIGH(id, p, n - 1) * BLOCK_STEP;
   size       = BLOCK_SIZE(id, p, n - 1);

   /*
    * bail out if all the primes used for sieving are not all
    * help by process 0
    */
   proc0_size = (n - 1) / p;

   if ((2 + proc0_size) < (int)sqrt((double)n))    {
       if (id == 0) /* parent process */
           printf("Too many processes\n");
       MPI_Finalize();
       exit(1);
   } /* if */

   // compute primes from 2 to sqrt(n);
   sqrt_n = sqrt(n);
   primes = (char*)calloc(sqrt_n + 1, 1);
   for (prime_multiple = 2;
        prime_multiple <= sqrt_n;
        prime_multiple += 2)    {
       primes[prime_multiple] = 1;
   } /* for */

   for (prime = 3; prime <= sqrt_n; prime += 2)    {
       if (primes[prime] == 1)
           continue;

       for (prime_multiple = prime << 1;
            prime_multiple <= sqrt_n;
            prime_multiple += prime)    {
           primes[prime_multiple] = 1;
       }
   } /* for */
```

```
/*
 * allocate this process' share of the array
 */
marked = (char*)calloc(size * sizeof(char), 1);
if (marked == NULL)    {
    printf("Cannot allocate enough memory\n");
    MPI_Finalize();
    exit(1);
} /* if */

num_per_block    = 1024 * 1024;
block_low_value  = low_value;
block_high_value = MIN(high_value,
                       low_value + num_per_block * BLOCK_STEP);

for (first_index_in_block = 0;
     first_index_in_block < size;
     first_index_in_block += num_per_block)    {
    for (prime = 3; prime <= sqrt_n; prime++)       {
        if (primes[prime] == 1)
            continue;
        if (prime * prime > block_low_value)   {
            first = prime * prime;
        }
       else   {
            if (!(block_low_value % prime))    {
                first = block_low_value;
            }
            else    {
                first = prime - (block_low_value % prime) +
                        block_low_value;
            }
       }

       /*
        * optimization - consider only odd multiples
        *                  of the prime number
        */
       if ((first + prime) & 1) // is odd
          first += prime;

       first_value_index = (first - BLOCK_FIRST) / BLOCK_STEP -
                           BLOCK_LOW(id, p, n - 1);
       prime_doubled     = prime << 1;
       prime_step        = prime_doubled / BLOCK_STEP;
       for (i = first; i <= high_value; i += prime_doubled)   {
           marked[first_value_index] = 1;
           first_value_index += prime_step;
       } /* for */
```

```
        }

        block_low_value += num_per_block * BLOCK_STEP;
        block_high_value = MIN(high_value,
                          block_high_value + num_per_block * BLOCK_STEP);
    } /* for first_index_in_block */


    /*
     * count the number of prime numbers found on this process
     */
    count = 0;
    for (i = 0; i < size; i++)
        if (!marked[i])
            count++;

    MPI_Reduce(&count, &global_count, 1, MPI_INT,
               MPI_SUM, 0, MPI_COMM_WORLD);

    /*
     * stop the timer
     */
    elapsed_time += MPI_Wtime();

    /* print the results */
    if (id == 0)   {
        global_count += 1; /* add first prime, 2 */
        printf("%d primes are less than or equal to %d\n",
               global_count, n);
        printf("Total elapsed time: %10.6fs\n",
               elapsed_time);
    } /* if */

    MPI_Finalize();

    return 0;
}
```

OUTPUT :



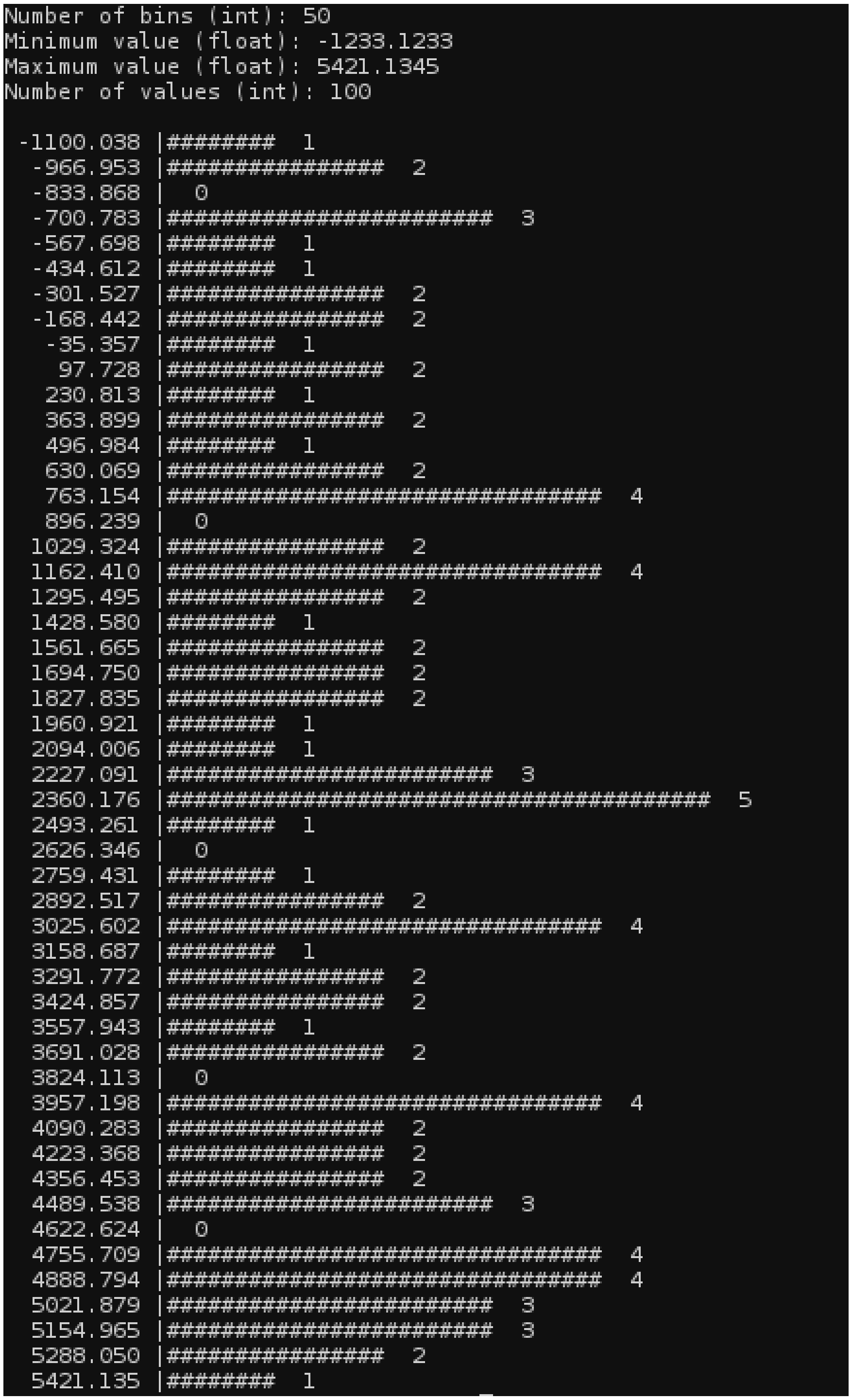
```
[dhrubanka@dhrubanka-pc src]$ mpicc -o eratosthenes_improved eratosthenes_improved.c -lm
[dhrubanka@dhrubanka-pc src]$ mpirun -np 2 eratosthenes_improved 152512534
8577481 primes are less than or equal to 152512534
Total elapsed time:  16.145270s
```

12. Histogram implementation using MPI

CODE :

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

void Get_input(   int* bin_count_p,
                  float* min_meas_p,
                  float* max_meas_p,
                  int* data_count_p,
                  int* local_data_count_p,
                  int my_rank,
                  int comm_sz,
                  MPI_Comm comm);

void Gen_data(    float local_data[],
                  int local_data_count,
                  int data_count,
                  float min_meas,
                  float max_meas,
                  int my_rank,
                  MPI_Comm comm);

void Set_bins(    float bin_maxes[],
                  int loc_bin_cts[],
                  float min_meas,
                  float max_meas,
                  int bin_count,
                  int my_rank,
                  MPI_Comm comm);

void Find_bins(   int bin_counts[],
                  float local_data[],
                  int loc_bin_cts[],
                  int local_data_count,
                  float bin_maxes[],
                  int bin_count,
                  float min_meas,
                  MPI_Comm comm);

int Which_bin(    float data,
                  float bin_maxes[],
                  int bin_count,
                  float min_meas);

void Print_histo( float bin_maxes[],
                  int bin_counts[],
                  int bin_count,
                  float min_meas);

void e(          int error);

int main(int argc, char* argv[]) {
  int       bin_count;
  float     min_meas;
  float     max_meas;
  float*    bin_maxes;
  int*      bin_counts;
  int*      loc_bin_cts;
  int       data_count;
  int       local_data_count;
  float*    data;
  float*    local_data;
  int       my_rank;
  int       comm_sz;
  MPI_Comm  comm;

  // Initialize mpi
  e(MPI_Init(&argc, &argv));
  comm = MPI_COMM_WORLD;
  e(MPI_Comm_size(comm, &comm_sz));
  e(MPI_Comm_rank(comm, &my_rank));

  // get user inputs for bin_count, max_meas, min_meas, and data_count
  Get_input(&bin_count, &min_meas, &max_meas, &data_count,
            &local_data_count, my_rank, comm_sz, comm);

  // allocate arrays
  bin_maxes = malloc(bin_count*sizeof(float));
  bin_counts = malloc(bin_count*sizeof(int));
  loc_bin_cts = malloc(bin_count*sizeof(int));
  data = malloc(data_count*sizeof(float));
  local_data = malloc(local_data_count*sizeof(float));

  // insert code below to finish this program
  Set_bins(bin_maxes,loc_bin_cts,min_meas,max_meas,bin_count,my_rank,comm);
  Gen_data(local_data,local_data_count,data_count,min_meas,max_meas,my_rank,comm);

Find_bins(bin_counts,local_data,loc_bin_cts,local_data_count,bin_maxes,bin_count,min_meas,comm)
;
  e(MPI_Reduce(loc_bin_cts,bin_counts,bin_count,MPI_INT,MPI_SUM,0,comm));
  if(my_rank == 0)
    Print_histo(bin_maxes,bin_counts,bin_count,min_meas);

  free(bin_maxes);
  free(bin_counts);
  free(loc_bin_cts);
  free(data);
  free(local_data);
  MPI_Finalize();
  return 0;
} /* main */

void e(int error) {
  if(error != MPI_SUCCESS) {
    fprintf(stderr,"Error starting MPI program, Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD,error);
    MPI_Finalize();
    exit(1);
  }
}

/* Print out the histogram */
void Print_histo(
     float bin_maxes[] /* in */,
     int bin_counts[]  /* in */,
     int bin_count     /* in */,
     float min_meas    /* in */) {

  int width = 40;
  int max = 0;
  int row_width;
  int i;
  int j;

  // get max count
  for(i = 0; i < bin_count; i++) {
    if(bin_counts[i] > max)
      max = bin_counts[i];
  }
  for(i = 0; i < bin_count; i++) {
    printf("%10.3f |",bin_maxes[i]);
    row_width = (float) bin_counts[i] / (float) max * (float) width;
    for(j=0; j < row_width; j++) {
      printf("#");
    }
    printf("  %d\n",bin_counts[i]);
  }
}  /* Print_histo */

/* Find out the appropriate bin for each data in local_data and increase the number of data in this bin  */
void Find_bins(
     int bin_counts[]      /* out */,
     float local_data[]    /* in  */,
     int loc_bin_cts[]     /* out */,
     int local_data_count  /* in  */,
     float bin_maxes[]     /* in  */,
     int bin_count         /* in  */,
     float min_meas        /* in  */,
     MPI_Comm comm){

  int i;
  int bin;

  for(i = 0; i < local_data_count; i++) {
    bin = Which_bin(local_data[i],bin_maxes,bin_count,min_meas);
    loc_bin_cts[bin]++;
  }
}  /* Find_bins */

/* Find out the appropriate bin for each data */
int Which_bin(float data, float bin_maxes[], int bin_count,
     float min_meas) {

  int i;
  for(i = 0; i < bin_count-1; i++) {
    if(data <= bin_maxes[i]) break;
  }
  return i;
}  /* Which_bin */

/* Initialzie each bin */
void Set_bins(
     float bin_maxes[]  /* out */,
     int loc_bin_cts[]  /* out */,
     float min_meas     /* in  */,
     float max_meas     /* in  */,
     int bin_count      /* in  */,
     int my_rank        /* in  */,
     MPI_Comm comm      /* in  */) {

  float range = max_meas - min_meas;
  float interval = range / bin_count;

  int i;
  for(i = 0; i < bin_count; i++) {
    bin_maxes[i] = interval * (float)(i+1) + min_meas;
    loc_bin_cts[i] = 0;
  }
}  /* Set_bins */

/* Generate random data */
void Gen_data(
     float local_data[]    /* out */,
     int local_data_count  /* in  */,
     int data_count        /* in  */,
     float min_meas        /* in  */,
     float max_meas        /* in  */,
     int my_rank           /* in  */,
     MPI_Comm comm         /* in  */) {
  float* data;
  if(my_rank == 0) {
    float range = max_meas - min_meas;
    data = malloc(data_count*sizeof(float));

    int i;
    for(i=0;i<data_count;i++) {
      data[i] = (float) rand() / (float) RAND_MAX * range + min_meas;
    }
  }
  e(MPI_Scatter(data,local_data_count,MPI_FLOAT,local_data,local_data_count,MPI_FLOAT, 0,
comm));
  if(my_rank == 0) free(data);
}  /* Gen_data */

/*  Get user inputs for bin_count, max_meas, min_meas, and data_count */
void Get_input(
     int* bin_count_p,        /* out */
     float* min_meas_p,       /* out */
     float* max_meas_p,       /* out */
     int* data_count_p,       /* out */
     int* local_data_count_p, /* out */
     int my_rank,             /* in  */
     int comm_sz,             /* in  */
     MPI_Comm comm            /* in  */) {

  if(my_rank == 0) {
    printf("Number of bins (int): ");
    scanf("%d",bin_count_p);
    printf("Minimum value (float): ");
    scanf("%f",min_meas_p);
    printf("Maximum value (float): ");
    scanf("%f",max_meas_p);
    // Make sure min < max
    if(*max_meas_p < *min_meas_p) {
      float* temp = max_meas_p;
      max_meas_p = min_meas_p;
      min_meas_p = temp;
    }
    printf("Number of values (int): ");
    scanf("%d",data_count_p);
    // Make sure data_count is a multiple of comm_sz
    *local_data_count_p = *data_count_p / comm_sz;
    *data_count_p = *local_data_count_p * comm_sz;
    printf("\n");
  }
  e(MPI_Bcast(bin_count_p,1,MPI_INT,0,comm));
  e(MPI_Bcast(min_meas_p,1,MPI_FLOAT,0,comm));
  e(MPI_Bcast(max_meas_p,1,MPI_FLOAT,0,comm));
  e(MPI_Bcast(data_count_p,1,MPI_INT,0,comm));
  e(MPI_Bcast(local_data_count_p,1,MPI_INT,0,comm));
}  /* Get_input */
```

OUTPUT :



```
Number of bins (int): 50
Minimum value (float): -1233.1233
Maximum value (float): 5421.1345
Number of values (int): 100

 -1100.038 |########  1
  -966.953 |################  2
  -833.868 |  0
  -700.783 |########################  3
  -567.698 |########  1
  -434.612 |########  1
  -301.527 |################  2
  -168.442 |################  2
   -35.357 |########  1
    97.728 |################  2
   230.813 |########  1
   363.899 |################  2
   496.984 |########  1
   630.069 |################  2
   763.154 |################################  4
   896.239 |  0
  1029.324 |################  2
  1162.410 |################################  4
  1295.495 |################  2
  1428.580 |########  1
  1561.665 |################  2
  1694.750 |################  2
  1827.835 |################  2
  1960.921 |########  1
  2094.006 |########  1
  2227.091 |########################  3
  2360.176 |########################################  5
  2493.261 |########  1
  2626.346 |  0
  2759.431 |########  1
  2892.517 |################  2
  3025.602 |################################  4
  3158.687 |########  1
  3291.772 |################  2
  3424.857 |################  2
  3557.943 |########  1
  3691.028 |################  2
  3824.113 |  0
  3957.198 |################################  4
  4090.283 |################  2
  4223.368 |################  2
  4356.453 |################  2
  4489.538 |########################  3
  4622.624 |  0
  4755.709 |################################  4
  4888.794 |################################  4
  5021.879 |########################  3
  5154.965 |########################  3
  5288.050 |################  2
  5421.135 |########  1
```