**Project 8**

**Due Sunday 4/10/2016 by 11:59 PM**

**No delay is accepted**

**For each day late submission 2 points penalty is incurred.**

**Post your answer in the folder: *Project 8* *on blackboard.***

**If you have any question please email me at mbadii@pace.edu**

***Do not delete anything from this file. Just write your answer under the word: Answer.***

Write a parallel program with MPI functions to do the following:

1) Define an array of size 1000.

2) Generate a random integer in the range of: 0 t0 999 for: ***n***.

3) Generate ***n*** random integers in the range of: 0 to 1000 and save them in array.

4) For m (its value comes from the keyboard) processes divide that array to (m-1) segments. All the segments except the last one must be equal (Note that the last segment may get bigger than other segments)

5) Each of process p1, p2, p3,…, pm-1 obtains the biggest number from its segment and send it to the process with rank = 0.

6) Process 0 receives a number from each process: p1, p2, p3,…, pm-1, finds the biggest number and print it.

Note: If ***m -1*** is bigger ***n*** terminate the execution of the program with an appropriate message the shows the value of ***m-1*** and ***n***.

The following is a serial program. It may help.

Note: Remember the variable comm\_sz in the parallel program ***must not get its value in the program or via scanf***.

//File name: a.c

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define atmost 1000

int find(int\* a, int from, int to){

int i, b;

printf("%d----%d\n", from,to);

b = a[from];

for(i = from + 1; i <= to; i++)

if(a[i] > b)

b = a[i];

return b;

}

int main(){

int i, n, a[atmost], comm\_sz, biggest, h, b;

printf("How many segments? ");

scanf("%d", &comm\_sz);

srand((unsigned)time(NULL));

n = rand()%atmost;

printf("n = %d\n", n);

for(i = 0; i < n; i++){

a[i] = rand() % 10001;

printf("a[%d] = %d, ", i, a[i]);

}

printf("\n");

h = n/(comm\_sz-1);

biggest = -1;

printf("h = %d\n", h);

for(i = 0; i < comm\_sz - 1; i++){

if(i == comm\_sz - 2)

b = find(a, i \* h, n-1);

else

b = find(a, i \* h, (i+1) \* h - 1);

if(b > biggest)

biggest = b;

}

printf("The biggest is: %d\n", biggest);

return 0;

}

Note: Since I need to test your project use static array (i.e. Do not use the built-in function: malloc). Name your array: a. Only use one array. Do not sort.

**Answer**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define atmost 1000

int FindLocalMax(int offset,int h,int a[],int my\_rank)

{

int max\_local = -1,i;

for( i=offset; i<offset+h ; i++)

{

if(a[i] > max\_local)

{

max\_local=a[i];

}

}

printf("Local Max = %d from rank %d \n",max\_local,my\_rank);

return max\_local;

}

int main(int argc, char \*argv[])

{

int i, n, a[atmost], comm\_sz,my\_rank,rc=0,dest,offset,tag1,tag2,h,source;

int local\_max,max\_value;

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &comm\_sz);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);

srandom((unsigned)time(NULL));

n = random()%atmost;

h = n/(comm\_sz-1);

if((comm\_sz-1)>n)

{

printf("Number of MPI process cannot be greater than number of elements.Hence terminating!\n");

MPI\_Abort(MPI\_COMM\_WORLD, rc);

exit(0);

}

if(my\_rank == 0)

{

printf("Number of Process entered %d \n",comm\_sz);

printf("The random number %d\n",n);

printf("The rank is %d\n",my\_rank);

for(i = 0; i < n; i++)

{

a[i] = rand()%1001;

}

offset =0;

tag2 = 1;

tag1 = 2;

for (dest=1; dest<comm\_sz; dest++)

{

MPI\_Send(&offset, 1, MPI\_INT, dest, tag1, MPI\_COMM\_WORLD);

MPI\_Send(&a[offset], h, MPI\_FLOAT, dest, tag2, MPI\_COMM\_WORLD);

printf("Process 0 sent %d elements to process%d with offset %d\n",h,dest,offset);

offset = offset + h;

}

printf("Process 0 has %d elements with offset =%d \n",n-offset,offset);

max\_value = FindLocalMax(offset,n-offset,a,my\_rank);

for (i = 1; i < comm\_sz; i++)

{

MPI\_Recv(&local\_max, 1, MPI\_INT, i, tag2, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

if(local\_max > max\_value)

{

max\_value = local\_max;

}

}

printf("\n Final Maximum value = %d\n ",max\_value);

}

if (my\_rank > 0)

{

printf("Rank = %d\n",my\_rank);

source = 0,dest = 0;

MPI\_Recv(&offset, 1, MPI\_INT, source, tag1, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&a[offset], h, MPI\_INT, source, tag2, MPI\_COMM\_WORLD, &status);

local\_max = FindLocalMax(offset,h,a,my\_rank);

MPI\_Send(&local\_max, 1, MPI\_INT,dest, tag2, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}