**Parallel and Distributed Computing Week 10**

**Shiva Manickam M**

**18BCE1272**

Write a program in MPI to create two processes in two different machines. Process 0 pings Process 1 and awaits for return ping using Non-blocking message passing routines.

# CODE:

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

int main(){

MPI\_Init(NULL, NULL);

int ntasks,rank; MPI\_Comm\_size(MPI\_COMM\_WORLD, &ntasks); MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Request request; MPI\_Status status;

int rec[1]={0},s[1] = {1}; if(rank == 0){

MPI\_Isend(s, 1, MPI\_INT, 1 , 10, MPI\_COMM\_WORLD,

&request);

printf("Message sent to P1\n"); sleep(1);

printf("Waiting for response from P1...\n"); int \*flag;

do{

&request);

}

MPI\_Irecv(rec, 1, MPI\_INT, 1 , 22, MPI\_COMM\_WORLD,

MPI\_Test(&request,flag,&status);

} while(!&flag); printf("Response recieved\n");

else if(rank == 1){ s[0] = 2;

sleep(3);

printf("\tWaiting for message from P0\n");

&request);

&request);

MPI\_Irecv(rec, 1, MPI\_INT, 0 , 10, MPI\_COMM\_WORLD,

MPI\_Wait(&request, &status); printf("\tMessage recieved from P0\n"); sleep(1);

MPI\_Isend(s, 1, MPI\_INT, 1 , 22, MPI\_COMM\_WORLD,

printf("\tResponse sent to P0\n");

}

printf("P: %d, S: %d, R: %d\n",rank,s[0],rec[0]); MPI\_Finalize();

return 0;

}

# OUTPUT:

**QUESTION 2:**

Write a program in MPI to create 10 tasks. Construct a ring topology to exchange message to its nearest neighbour in the ring using blocking message passing routines.

# CODE:

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

int main(){

MPI\_Init(NULL, NULL);

int ntasks,rank; MPI\_Comm\_size(MPI\_COMM\_WORLD, &ntasks); MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

int token;

if (rank != 0){

MPI\_Recv(&token, 1, MPI\_INT, rank - 1 , 10, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Node %d Received token %d from Node

%d\n",rank,token,rank-1);

}

else{

token = 1;

}

MPI\_Send(&token, 1, MPI\_INT, (rank + 1) % ntasks , 10, MPI\_COMM\_WORLD);

if(rank == 0){

MPI\_Recv(&token, 1, MPI\_INT, rank - 1, 10, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Node %d Received token %d from Node

%d\n",rank,token,ntasks-1);

}

MPI\_Finalize(); return 0;

}

# OUTPUT:

**QUESTION 3:**

Write an MPI program to compute PI using the “dartboard” technique for 500 rounds by using blocking point-to-point communication routines.

# CODE:

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

void srandom (unsigned seed); double dboard (int darts);

#define DARTS 50000 /\* number of throws at dartboard \*/ #define ROUNDS 500 /\* number of times "darts" is iterated \*/ #define MASTER 0 /\* task ID of master task \*/

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

{

double homepi, pisum, pi, avepi; int taskid, numtasks, rc, i; MPI\_Status status;

/\* Obtain number of tasks and task ID \*/ MPI\_Init(&argc,&argv); MPI\_Comm\_size(MPI\_COMM\_WORLD,&numtasks); MPI\_Comm\_rank(MPI\_COMM\_WORLD,&taskid);

printf ("MPI task %d has started...\n", taskid); srandom (taskid);

avepi = 0;

double t = MPI\_Wtime();

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

homepi = dboard(DARTS);

rc = MPI\_Reduce(&homepi, &pisum, 1, MPI\_DOUBLE, MPI\_SUM, MASTER, MPI\_COMM\_WORLD);

if (taskid == MASTER) { pi = pisum/numtasks;

avepi = ((avepi \* i) + pi)/(i + 1);

}

}

if (taskid == MASTER) {

printf("After %8d throws, average value of pi =

%10.8f\n",(DARTS \* ROUNDS \* numtasks),avepi);

printf ("Real value of PI: 3.1415926535897 \n");

}

t = MPI\_Wtime() - t;

printf("Time taken in Node %d is %f\n", taskid , t); MPI\_Finalize();

return 0;

}

double dboard(int darts)

{

#define sqr(x) ((x)\*(x)) long random(void);

double x\_coord, y\_coord, pi, r; int score, n;

unsigned int cconst;

if (sizeof(cconst) != 4) {

printf("Wrong data size for cconst variable in dboard routine!\n");

printf("See comments in source file. Quitting.\n"); exit(1);

}

cconst = 2 << (31 - 1);

score = 0;

for (n = 1; n <= darts; n++) {

r = (double)random()/cconst; x\_coord = (2.0 \* r) - 1.0;

r = (double)random()/cconst; y\_coord = (2.0 \* r) - 1.0;

if ((sqr(x\_coord) + sqr(y\_coord)) <= 1.0) score++;

}

pi = 4.0 \* (double)score/(double)darts; return(pi);

}

# OUTPUT:

**QUESTION 1:**

**Parallel and Distributed Computing Week 11**

**Shiva Manickam M**

**18BCE1272**

Write a program in MPI to generate ‘n’ random float numbers and send’ k’ of those to each node and make them compute the average and send it back to the master which computes the average of those averages.

# CODE:

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

// size of array #define n 10000

// Temporary array for slave process float arr[n];

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

{

int pid, np, k, nRec;

MPI\_Status status; MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &pid); MPI\_Comm\_size(MPI\_COMM\_WORLD, &np);

if (pid == 0) {

srand(time(0)); float a[n];

for (int i = 0; i < n;i++) a[i] = rand()%1000;

int index, i; k = n /(np-1);

if (np > 1) {

for (i = 1; i < np - 1; i++) {

MPI\_COMM\_WORLD); MPI\_COMM\_WORLD);

}

index = (i-1) \* k;

MPI\_Send(&k, 1, MPI\_INT, i, 0, MPI\_Send(&a[index], k, MPI\_FLOAT, i, 0,

MPI\_COMM\_WORLD);

}

index = (i-1) \* k; int rem = n - index;

MPI\_Send(&rem, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

MPI\_Send(&a[index], rem, MPI\_FLOAT, i, 0,

float sum = 0; float tmp = 0;

for (i = 1; i < np; i++) {

MPI\_Recv(&tmp, 1, MPI\_FLOAT, MPI\_ANY\_SOURCE, 0,

MPI\_COMM\_WORLD, &status);

sum += tmp;

}

sum /= n;

printf("Average of array is : %f\n", sum);

}

else {

MPI\_Recv(&nRec, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,

&status); &status);

MPI\_Recv(&arr, nRec, MPI\_FLOAT, 0, 0, MPI\_COMM\_WORLD,

float pavg = 0;

for (int i = 0; i < nRec; i++) pavg += arr[i];

MPI\_Send(&pavg, 1, MPI\_FLOAT, 0, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}

# OUTPUT:

**QUESTION 2:**

Write an MPI program to compute PI using the “dartboard” technique for 1000 rounds by using reduction collective computation.

# CODE:

#include "mpi.h" #include <stdio.h> #include <stdlib.h> #include <time.h>

double dboard (int darts); #define DARTS 50000

#define ROUNDS 1000

#define MASTER 0

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

{

double homepi, pisum, pi, avepi; int taskid, numtasks, rc, i; MPI\_Status status;

/\* Obtain number of tasks and task ID \*/ MPI\_Init(&argc,&argv); MPI\_Comm\_size(MPI\_COMM\_WORLD,&numtasks); MPI\_Comm\_rank(MPI\_COMM\_WORLD,&taskid);

printf ("MPI task %d has started...\n", taskid); srand(time(0)\*taskid);

avepi = 0;

double t = MPI\_Wtime();

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

homepi = dboard(DARTS);

rc = MPI\_Reduce(&homepi, &pisum, 1, MPI\_DOUBLE, MPI\_SUM, MASTER, MPI\_COMM\_WORLD);

if (taskid == MASTER) { pi = pisum/numtasks;

avepi = ((avepi \* i) + pi)/(i + 1);

}

}

if (taskid == MASTER) {

printf("After %d throws, average value of pi =

%10.15f\n",(DARTS \* ROUNDS \* numtasks),avepi);

printf ("Real value of PI: 3.1415926535897 \n");

}

t = MPI\_Wtime() - t;

printf("Time taken in Node %d is %f\n", taskid , t); MPI\_Finalize();

return 0;

}

double dboard(int darts)

{

#define sqr(x) ((x)\*(x)) long random(void);

double x\_coord, y\_coord, pi, r; int score, n;

unsigned int cconst;

if (sizeof(cconst) != 4) {

printf("Wrong data size for cconst variable in dboard routine!\n");

printf("See comments in source file. Quitting.\n"); exit(1);

}

cconst = 2 << (31 - 1);

score = 0;

for (n = 1; n <= darts; n++) {

r = (double)random()/cconst; x\_coord = (2.0 \* r) - 1.0;

r = (double)random()/cconst; y\_coord = (2.0 \* r) - 1.0;

if ((sqr(x\_coord) + sqr(y\_coord)) <= 1.0) score++;

}

pi = 4.0 \* (double)score/(double)darts; return(pi);

}

# OUTPUT:

**QUESTION 3:**

Write an MPI program to perform matrix multiplication (1000x1000) using scatter and gather routines.

# CODE:

#include "mpi.h" #include <stdio.h> #include <stdlib.h> #include <time.h>

#define N 1000

int main(int argc, char \*argv[]){ srand(time(0));

double t;

int processID, maxProcess, i, j; MPI\_Init (&argc, &argv);

int (\*a)[N] = malloc(sizeof(int[N][N]));

int (\*b)[N] = malloc(sizeof(int[N][N]));

int (\*c)[N] = malloc(sizeof(int[N][N]));

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &processID); MPI\_Comm\_size(MPI\_COMM\_WORLD, &maxProcess);

t = MPI\_Wtime(); MPI\_Barrier(MPI\_COMM\_WORLD); if ( processID == 0)

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

a[i][j] = rand()%10;

b[i][j] = rand()%10;

}

}

/\*

MPI\_Barrier(MPI\_COMM\_WORLD); if ( processID == 0) {

printf("\nMatrix A\n"); for(i = 0;i < N; i++) {

for(j = 0;j < N;j++) printf("%d\t",a[i][j]);

printf("\n");

}

}

MPI\_Barrier(MPI\_COMM\_WORLD); if ( processID == 0) {

printf("\nMatrix B\n"); for(i = 0;i < N; i++) {

for(j = 0;j < N;j++) printf("%d\t",b[i][j]);

printf("\n");

}

}

MPI\_Barrier(MPI\_COMM\_WORLD);

\*/

MPI\_Scatter(a, N\*N/maxProcess, MPI\_INT, a, N\*N/maxProcess,MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(b, N\*N, MPI\_INT, 0,MPI\_COMM\_WORLD);

int k;

for (i = 0; i < N/maxProcess; i++) for (j = 0; j < N; j++) {

c[i][j] = 0;

for (k = 0; k < N; k++)

c[i][j] += a[i][k] \* b[k][j];

}

MPI\_Gather(c, N\*N/maxProcess, MPI\_INT, c, N\*N/maxProcess,MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD); t = MPI\_Wtime() - t;

if (processID == 0) {

printf("Time taken is %0.4fs for %d processes to computer product of 1000x1000 matrix\n", t, maxProcess);

}

/\*

if (processID == 0){ printf("AxB = \n"); for(i = 0;i < N; i++) {

for(j = 0;j < N;j++) printf("%d\t",c[i][j]);

printf("\n");

}

}

\*/ free(a);

free(b);

free(c); MPI\_Finalize();

return 0;

}

# OUTPUT:

