Introduction to Parallel Programing Techniques

Deferred Assessment

Professor: Stylianos Sygletos

Nestor Edgar Sandoval Alaguna
200243856

Assignment 1

Exercise – 1 [2 points]

Suppose we toss darts randomly at a square dartboard, whose bullseye is at the origin, and whose sides are
0.6m in length. Suppose also that there's a circle inscribed in the square dartboard. The radius of the circle is 0.3m, and its area is 0.093π square meters. If the points that are hit by the darts are uniformly distributed (and we always hit the square), then the number of darts that hit inside the circle should approximately satisfy the equation:

number in circle / total number of tosses = π/4
since the ratio of the area of the circle of the square is π/4. We can use this formula to estimate the value π with a random number generator:
number in circle = 0;
for (toss = 0; toss < number_of_tosses; toss++) {
x = random double between −1 and 1;
y = random double between −1 and 1;
distance_squared = x∗x + y∗y;
if (distance_squared <= 1) number_in_circle++;
}
pi_estimate = 4∗number_in_circle/((double) number_of_tosses);

This is called a "Monte Carlo" method, since it uses randomness (the dart tosses). Write an MPI program that uses a Monte Carlo method to estimate π. Process 0 should read in the total number of tosses and broadcast it to the other processes. Use MPI Reduce to find the global sum of the local variable number_in_circle, and have process 0 print the result. You may want to use **long long int**s for the number of hits in the circle and the number of tosses, since both may have to be very large to get a reasonable estimate of π. Demonstrate with an example the correct operation of your programme.

Answer:

First the program asks for the number of tosses or points that will be used to estimate the value of pi. After this the distance of these points will be calculated and following this the ratio of the number of points in the circle with the total number of points will be calculated. The processes 0 will get the input and broadcast it to other process that calculate the ratio of number of points and then this root process will calculate the sum of all these ratios.

Code:

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

const int MAX_STRING = 100;
double gen_max = RAND_MAX;


long long int calculate_points(int my_rank, int n)
```

```
{
int counter = 0;
double x,y,x_gen,y_gen,distance;

srand(time(NULL)+rand()%5+ my_rank);

for (int i = 0; i < n; i++)
{
x_gen = rand();
x = 2*( x_gen/gen_max ) - 1;
y_gen = rand();
y = 2*( y_gen/gen_max ) - 1;
distance = x*x + y*y;
if ( distance <= 1 )
{
counter++;
}
}
return counter;
}

void get_input(int my_rank, int comm_sz, int* n_p)
{
setbuf(stdout,NULL);
if (my_rank==0)
{
printf("Enter number of tosses: \n");
scanf("%d", n_p);
if(*n_p<comm_sz)
{
exit_with_error("Number of points has to be larger than the number of processes");
}
}
MPI_Bcast(n_p, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}

int main(int argc, char** argv)
{
int comm_sz, my_rank, n, local_n;
long long int local_points, global_points;
double global_est;
MPI_Init(NULL,NULL);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
get_input(my_rank, comm_sz, &n);
local_n = n/comm_sz;
local_points = calculate_points(my_rank, local_n);
MPI_Reduce(&local_points,&global_points,1,MPI_LONG_LONG_INT,MPI_SUM,0,MPI_COMM_WORLD);
global_est = 4*global_points/((double) n);
if (my_rank ==0)
{
printf("the pi estimation is %f \n",n, global_est);
}
MPI_Finalize();
return 0;
}
```

```
nestor@nestor-PC:/media/nestor/CA005B8A005B7BFF/Users/Nestor/Downloads/Parallel pr
ogramming/assignment1$ mpicc -g -Wall ex11.c -o ex11
nestor@nestor-PC:/media/nestor/CA005B8A005B7BFF/Users/Nestor/Downloads/Parallel pr
ogramming/assignment1$ mpiexec -n 4 ex11
Please specify number of tosses
10000000
Estimated value of pi = 3.141788800000000
nestor@nestor-PC:/media/nestor/CA005B8A005B7BFF/Users/Nestor/Downloads/Parallel pr
ogramming/assignment1$
```

Exercise – 2 [4 points]

Write two MPI programs that perform the computation of a global sum; the first using tree-structured computation and the second using a butterfly structure. First, write your programs for the special case in which comm_sz is a power of two. Subsequently, modify your program so that it will handle any number of
processes. Identify and run examples that demonstrate the correct implementation of your code and comment
on the results.

Answer:

Code:

```
#include <assert.h>
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
double generate_max = RAND_MAX;
void exit_with_error(const char *message) {
printf("ERROR: %s.\n", message);
MPI_Finalize();
exit(1);
}
void input_points(int current_rank, int processes_number, int *points_number) {
setbuf(stdout, NULL);
if (current_rank == 0) {
printf("Enter number of points to estimate: \n");
scanf("%d", points_number);
if (*points_number < processes_number) {
exit_with_error(
"Please specify number of tosses");
}
}
MPI_Bcast(points_number, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
int get_points(int current_rank, int n) {
int c = 0;
srand(time(NULL) + rand() % 5 + current_rank);
for (int i = 0; i < n; i++) {
double generate_x = rand();
double x = 2 * (generate_x / generate_max) - 1;
double generate_y = rand();
```

```
double y = 2 * (generate_y / generate_max) - 1;
double l2 = x * x + y * y;
if (l2 <= 1) {
c++;
}
}
return c;
}
long long int calculate_global_sum(int local_value, int current_rank,
int common_size, MPI_Comm common) {
int partner_value;
int partner_rank;
for (int gap = 2; gap <= common_size; gap *= 2) {
if (current_rank % gap < gap / 2) {
partner_rank = current_rank + gap / 2;
} else {
partner_rank = current_rank - gap / 2;
}
MPI_Sendrecv(&local_value, 1, MPI_INT, partner_rank, 0, &partner_value, 1,
MPI_INT, partner_rank, 0, common, MPI_STATUS_IGNORE);
local_value += partner_value;
}
return local_value;
}
int main() {
int common_size, current_rank;
int local_value, global_sum;
long long int local_points, global_points;
double global_estimation;
int n, local_n;
MPI_Comm common;
common = MPI_COMM_WORLD;
MPI_Init(NULL, NULL);
MPI_Comm_rank(MPI_COMM_WORLD, &current_rank);
MPI_Comm_size(MPI_COMM_WORLD, &common_size);
input_points(current_rank, common_size, &n);
local_n = n / common_size;
local_points = get_points(current_rank, local_n);
local_value = current_rank + 1;
global_points =
calculate_global_sum(local_points, current_rank, common_size, common);
global_sum =
calculate_global_sum(local_value, current_rank, common_size, common);
global_estimation = 4 * global_points / ((double)n);
if (current_rank == 0) {
printf("Estimation of pi: %f.\n", global_estimation);
}
MPI_Finalize();
return 0;
}
```

Code result for butterfly implementation running with 8 processes

```
Enter number of points to estimate:
1000
Estimation of pi: 3.124000.
```

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

```
const int MAX_STRING = 100;
double generate_max = RAND_MAX;
void exit_with_error(const char *message) {
printf("ERROR: %s.\n", message);
MPI_Finalize();
exit(1);
}
void get_input(int current_rank, int processes_number, int *points_number) {
setbuf(stdout, NULL);
if (current_rank == 0) {
printf("Enter number of points to estimate: \n");
scanf("%d", points_number);
if (*points_number < processes_number) {
exit_with_error(
"Please specify number of tosses");
}
}
MPI_Bcast(points_number, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
int get_points(int current_rank, int n) {
int c = 0;
srand(time(NULL) + rand() % 5 + current_rank);
for (int i = 0; i < n; i++) {
double generate_x = rand();
double x = 2 * (generate_x / generate_max) - 1;
double generate_y = rand();
double y = 2 * (generate_y / generate_max) - 1;
double l2 = x * x + y * y;
if (l2 <= 1) {
c++;
}
}
return c;
}
long long int calculate_global_sum(int local_value, int current_rank,
int common_size, MPI_Comm common) {
int partner_value;
int partner_rank;
for (int reduce_common_size = common_size; reduce_common_size > 1;
reduce_common_size /= 2) {
if (current_rank < reduce_common_size) {
if (current_rank >= reduce_common_size / 2) {
partner_rank = current_rank - reduce_common_size / 2;
MPI_Send(&local_value, 1, MPI_INT, partner_rank, 0, common);
} else {
partner_rank = current_rank + reduce_common_size / 2;
MPI_Recv(&partner_value, 1, MPI_INT, partner_rank, 0, common,
MPI_STATUS_IGNORE);
local_value += partner_value;
}
}
}
return local_value;
}
int main() {
int common_size, current_rank;
int local_value, global_sum;
long long int local_points, global_points;
double global_estimation;
int n, local_n;
MPI_Comm common;
common = MPI_COMM_WORLD;
MPI_Init(NULL, NULL);
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &current_rank);
MPI_Comm_size(MPI_COMM_WORLD, &common_size);
get_input(current_rank, common_size, &n);
local_n = n / common_size;
local_points = get_points(current_rank, local_n);
local_value = current_rank + 1;
global_points =
calculate_global_sum(local_points, current_rank, common_size, common);
global_sum =
calculate_global_sum(local_value, current_rank, common_size, common);
global_estimation = 4 * global_points / ((double)n);
if (current_rank == 0) {
printf("Estimation of pi %f \n", global_estimation);
}
MPI_Finalize();
return 0;
}
```

Code result for tree implementation running with 8 processes

```
Enter number of points to estimate:
1000
Estimation of pi 3.196000
```

Exercise – 3 [4 points]

A ping-pong is a communication in which two messages are sent, first from process A to process B (ping) and then from process B back to process A (pong). Timing blocks of repeated ping-pongs is a common way
to estimate the cost of sending messages. Time the ping-pong program using the C clock function on your system.
a) How long does the code have to run before clock gives a non-zero run-time? How do the times you got with the clock function compare to times taken with MPI_Wtime? [2 points]
b) What happens on your system when the count argument is 0? Can you explain why you get a nonzero elapsed time when you send a zero-byte message?
Identify and run appropriate examples to support your answers.

Answer:
Since the clock function calculates the CPU time, it gives 0 as run time for first 500 ping-pongs, this happens due to that the ping and pong goes with the tick of the CPUs clock. In order to see an impact, the ping-pongs should be greater than 500 as in the code.
C clock only calculates the CPU time, it doesn't not have in count the idle time of the cores.

Due to the overhead time of communication between processes the runtime is non zero even with the count equal to 0.

Code:

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

int main(void) {
```

```c
int ping = 1;int pong = 2;int count = 1000;
int comm_sz, my_rank;
double str, fin, loc_elap, elap;
double str_c, fin_c, loc_elap_c, elap_c;


MPI_Comm comm;
MPI_Init(NULL, NULL);
comm = MPI_COMM_WORLD;
MPI_Comm_size(comm, &comm_sz);
MPI_Comm_rank(comm, &my_rank);
MPI_Barrier(comm);
str = MPI_Wtime();
str_c = (double)clock();

for (int i = 0; i < count; i++) {
if (my_rank == 1) {
MPI_Send(&pong, 1, MPI_INT, 0, 0, comm);
MPI_Recv(&ping, 1, MPI_INT, 0, 0, comm, MPI_STATUSES_IGNORE);
}
if (my_rank == 0) {
MPI_Send(&ping, 1, MPI_INT, 1, 0, comm);
MPI_Recv(&pong, 1, MPI_INT, 1, 0, comm, MPI_STATUSES_IGNORE);
}
}

fin = MPI_Wtime();
fin_c = (double)clock();
loc_elap = fin - str;
loc_elap_c = (fin_c - str_c)/ (double)CLOCKS_PER_SEC;
MPI_Reduce(&loc_elap, &elap, 1, MPI_DOUBLE, MPI_MAX, 0, comm);
MPI_Reduce(&loc_elap_c, &elap_c, 1, MPI_DOUBLE, MPI_MAX, 0, comm);
loc_elap_c = (fin_c - str_c)/ (double)CLOCKS_PER_SEC;
if (my_rank == 0) {
printf("Elapsed time with MPI_Wtime is: %f\n", elap);
printf("Elapsed time with C clock function is: %f\n", elap_c);
}

MPI_Finalize();
return 0;
}
```

```
nestor@nestor-PC:/media/nestor/CA005B8A005B7BFF/Users/Nestor/Downloads/Parallel programming/assignment1$ mpicc -g -Wall ex13.c -o ex13
nestor@nestor-PC:/media/nestor/CA005B8A005B7BFF/Users/Nestor/Downloads/Parallel programming/assignment1$ mpiexec -n 2 ex13
Elapsed time with MPI_Wtime is: 0.000217
Elapsed time with C clock function is: 0.000216
nestor@nestor-PC:/media/nestor/CA005B8A005B7BFF/Users/Nestor/Downloads/Parallel programming/assignment1$ mpiexec -n 2 ex13
Elapsed time with MPI_Wtime is: 0.000290
Elapsed time with C clock function is: 0.000288
```