Introduction to Parallel Programing Techniques

Deferred Assessment

Professor: Stylianos Sygletos

Nestor Edgar Sandoval Alaguna
200243856

Assignment 6

Exercise – 1 [2 points] [OpenMP Implementation]
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 0.093 π 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 $\pi/4$.
We can use this formula to estimate the value π with a random number generator:

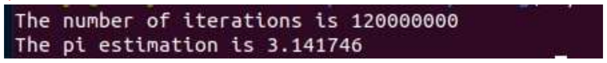
```
number in circle = 0;
for (toss = 0; toss < number_of_tosses; toss++) {
x = random double between − 1 and 1;
y = random double between − 1 and 1;
distance_squared = x∗x + y∗y;
if (distance_squared <= 1) number_in_circle++;
} pi_estimate = 4∗number_in_circle/((double) number_of_tosses);
```

Write a OpenMP program that uses a Monte Carlo method to estimate π. The main thread should read in the total number of tosses and print the estimate. You may want to use long long int 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.

Answer:

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <time.h>
double gen_max = RAND_MAX;
int main(int argc, char *argv[])
{
long thread_count;
long int n, n_c=0;
double pi_estimate;
thread_count = strtol(argv[1],NULL,10);
n = strtol(argv[2], NULL, 10);
# pragma omp parallel for num_threads(thread_count) \
reduction(+: n_c)
for (int i = 0; i < n; i++)
{
double x_gen = rand();
double x = 2*( x_gen/gen_max ) - 1;
double y_gen = rand();
double y = 2*( y_gen/gen_max ) - 1;
double l2 = x*x + y*y;
if ( l2 <= 1 )
{
n_c++;
}
}
pi_estimate = (4.0*n_c)/n;
printf("The number of iterations is %ld \n", n);
printf("The pi estimation is %f \n", pi_estimate);
return 0;
}
```



```
The number of iterations is 120000000
The pi estimation is 3.141746
```

Exercise – 2 [4 points]

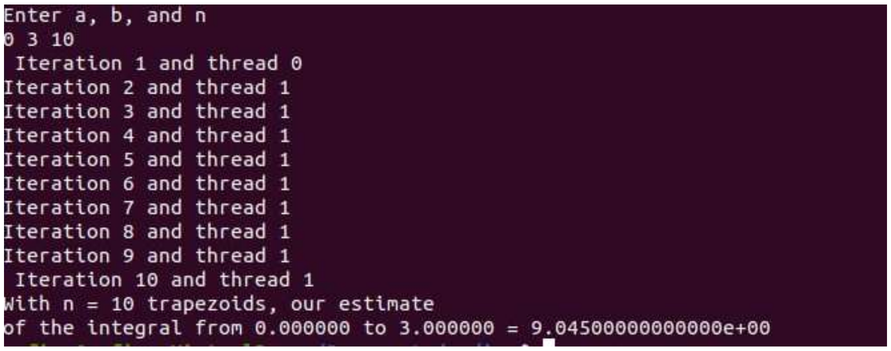
Modify the trapezoidal rule program that uses a parallel for directive (omp trap 3.c, which will be given to you during the lectures) so that the parallel for is modified by a schedule (runtime) clause. Run the program with various assignments to the environment variable OMP_SCHEDULE and determine which iterations are assigned to which thread. This can be done by allocating an array iterations of n ints and in the Trap function assigning omp_get_thread_num() to iterations[i] in the ith iteration of the for loop. What is the default assignment of iterations on your system? How are guided schedules determined?

Answer:

The scheduling clause allows to implement different schemes as it can be seen in this exercise, the run time of this method uses OMP schedule to determine the best schedule for the iterations in a loop.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
void Usage(char* prog_name);
double f(double x); /* Function we're integrating */
double Trap(double a, double b, int n, int thread_count);
int main(int argc, char* argv[]) {
double global_result = 0.0; /* Store result in global_result */
double a, b; /* Left and right endpoints */
int n; /* Total number of trapezoids */
int thread_count;
if (argc != 2) Usage(argv[0]);
thread_count = strtol(argv[1], NULL, 10);
printf("Enter a, b, and n\n");
scanf("%lf %lf %d", &a, &b, &n);
global_result = Trap(a, b, n, thread_count);
printf("With n = %d trapezoids, our estimate\n", n);
printf("of the integral from %f to %f = %.14e\n",
a, b, global_result);
return 0;
}
double f(double x) {
double return_val;
return_val = x*x;
return return_val;
}
double Trap(double a, double b, int n, int thread_count) {
double h, approx;
int i, thread_nu;
int* it = malloc(n*sizeof(int));
h = (b-a)/n;
approx = (f(a) + f(b))/2.0;
# pragma omp parallel for num_threads(thread_count) \
reduction(+: approx) schedule(runtime)
for (i = 1; i <= n-1; i++){
approx += f(a + i*h);
it[i] = omp_get_thread_num();
}
approx = h*approx;
printf(" Iteration 1 and thread %d ", it[0]);
thread_nu = it[0];
for (i = 2; i < n; i++)
if (thread_nu != it[i]) {
printf("\n");
printf("Iteration %d and thread %d ", i, it[i]);
}
if (thread_nu == it[n-1]) {
printf("Iteration%d and thread %d ", n, it[n-1]);
} else {
printf("\n");
printf(" Iteration %d and thread %d ", n, it[n-1]);
}
printf("\n");
free(it);
return approx;
}
void Usage(char* prog_name) {
fprintf(stderr, "usage: %s <number of threads>\n", prog_name);
exit(0);
}
```



```
Enter a, b, and n
0 3 10
 Iteration 1 and thread 0
Iteration 2 and thread 1
Iteration 3 and thread 1
Iteration 4 and thread 1
Iteration 5 and thread 1
Iteration 6 and thread 1
Iteration 7 and thread 1
Iteration 8 and thread 1
Iteration 9 and thread 1
 Iteration 10 and thread 1
With n = 10 trapezoids, our estimate
of the integral from 0.000000 to 3.000000 = 9.04500000000000e+00
```

The program was run with 3 threads.