Sheet 06

PS Parallel Programming

Patrick Wintner

April 27, 2025

1 Approximation of Pi

The execution times of different implementations of the Monte Carlo method are measured.

1.1 Source Code

1.1.1 Serial Implementation

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <omp.h>
5 #define _USE_MATH_DEFINES
  #include <math.h>
   int main() {
       long n = 700000000;
       long i, count = 0;
10
       double x, y, pi;
       double startTime, endTime;
12
13
       startTime = omp_get_wtime();
14
15
       srand((unsigned) time(NULL));
```

```
for (i = 0; i < n; i++) {
17
            x = (double) rand() / RAND_MAX;
            y = (double) rand() / RAND_MAX;
19
20
            if (x * x + y * y \le 1) count++;
21
       }
22
23
       endTime = omp_get_wtime();
25
       pi = 4.0 * count / n;
26
        if(pi<0.99*M_PI || 1.01*M_PI<pi) {
27
            fprintf(stderr, "Error: estimated value deviates significantly: %f\n", pi);
28
            return 1;
29
       }
            printf("%2.4f\n", endTime-startTime);
31
       return 0;
32
   }
33
```

1.1.2 Parallel Implementation using a Critical Section

```
#include <stdio.h>
  #include <stdlib.h>
  #include <time.h>
   #include <omp.h>
   #define _USE_MATH_DEFINES
   #include <math.h>
8
   int main() {
       long n = 700000000;
10
       long i, count = 0;
       double x, y, pi;
12
       double startTime, endTime;
13
14
       startTime = omp_get_wtime();
15
16
       \#pragma\ omp\ parallel\ private(x,\ y)
17
            unsigned seed = (unsigned) time(NULL)+13*omp_get_thread_num();
19
            #pragma omp for schedule (static)
20
            for (i = 0; i < n; i++) {
21
                x = (double) rand_r(&seed) / RAND_MAX;
```

```
y = (double) rand_r(&seed) / RAND_MAX;
23
24
                 #pragma omp critical
25
26
                     if (x * x + y * y \le 1) count++;
27
                }
28
            }
29
        }
31
        endTime = omp_get_wtime();
32
33
        pi = 4.0 * count / n;
34
35
        if(pi<0.99*M_PI|| 1.01*M_PI<pi) {
            fprintf(stderr, "Error: estimated value deviates significantly: %f\n", pi);
37
            return 1;
38
39
            printf("%2.4f\n", endTime-startTime);
40
        return 0;
   }
42
```

1.1.3 Parallel Implementation using an Atomic Statement

```
#include <stdio.h>
2 #include <stdlib.h>
  #include <time.h>
   #include <omp.h>
   #define _USE_MATH_DEFINES
   #include <math.h>
   int main() {
       long n = 700000000;
9
       long i, count = 0;
10
       double x, y, pi;
11
       double startTime, endTime;
12
       startTime = omp_get_wtime();
14
15
       \#pragma\ omp\ parallel\ private(x,y)
16
17
            unsigned seed = (unsigned) time(NULL) + 13*omp_get_thread_num();
18
            #pragma omp for schedule (static)
```

```
for (i = 0; i < n; i++) {
20
                x = (double) rand_r(&seed) / RAND_MAX;
21
                y = (double) rand_r(&seed) / RAND_MAX;
22
23
                if (x * x + y * y \le 1) {
24
                     #pragma omp atomic
25
                     count++;
26
                }
            }
28
       }
29
30
       endTime = omp_get_wtime();
31
32
       pi = 4.0 * count / n;
       if(pi<0.99*M_PI|| 1.01*M_PI<pi) {
34
            fprintf(stderr, "Error: estimated value deviates significantly: %f\n", pi);
35
            return 1;
36
       }
37
            printf("%2.4f\n", endTime-startTime);
       return 0;
   }
40
```

1.1.4 Parallel Implementation using a Reduction Clause

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
  #include <omp.h>
   #define _USE_MATH_DEFINES
   #include <math.h>
   int main() {
       long n = 700000000;
       long i, count = 0;
10
       double x, y, pi;
11
       double startTime, endTime;
12
       startTime = omp_get_wtime();
14
15
       \#pragma\ omp\ parallel\ private(x,y)
16
17
           unsigned seed = (unsigned) time(NULL)+13*omp_get_thread_num();
```

```
#pragma omp for reduction(+: count) schedule (static)
19
            for (i = 0; i < n; i++) {
20
                x = (double) rand_r(&seed) / RAND_MAX;
21
                y = (double) rand_r(&seed) / RAND_MAX;
22
23
                if (x * x + y * y \le 1) count++;
24
            }
25
       }
27
       endTime = omp_get_wtime();
28
29
       pi = 4.0 * count / n;
30
        if(pi<0.99*M_PI|| 1.01*M_PI<pi) {
31
            fprintf(stderr, "Error: estimated value deviates significantly: %f\n", pi);
            return 1;
33
34
            printf("%2.4f\n", endTime-startTime);
35
       return 0;
36
   }
37
```

1.2 Measurement Method

The measurement was done on the LCC3 cluster by calling ./job.sh serial 3, ./job.sh critical 3, ./job.sh atomic 3 and ./job.sh reduction 3.

The following scripts are involved in the experiment.

1.2.1 SLURM Script

```
#!/bin/bash
# usage: sbatch [slurm_options] <executable> <number_of_measurements>

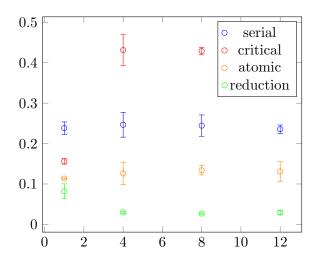
# Execute job in the partition "lva" unless you have special requirements.
# SBATCH --partition=lva
# Name your job to be able to identify it later
# SBATCH --job-name csba4017
# Redirect output stream to this file
# SBATCH --output=output.log
# Maximum number of tasks (=processes) to start in total
# SBATCH --ntasks=1
```

```
# Maximum number of tasks (=processes) to start per node
#SBATCH --ntasks-per-node=1
# Enforce exclusive node allocation, do not share with other jobs
#SBATCH --exclusive
# ./benchmark.sh $1 $2
```

1.2.2 Benchmark Script

```
#!/bin/bash
   # Usage: ./benchmark.sh <executable> <number_of_measurements>
  results=$1".dat"
  echo "x y ey" > $results # create header
  for i in {1,4,8,12} # number of threads
           measurements=$i"_"$1".log"
           export OMP_NUM_THREADS=$i
           for j in $(seq 1 $2) # repeat measurement £2 times
11
12
                    ./$1 >> $measurements # store measurement results in <executable>.log
13
           done
14
           ./toTable $measurements $results $2 $i #store table in <executable>.dat
15
           rm $measurements
17 done
18 make clean
```

1.3 Experiment Results



1.4 Discussion