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

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define _USE_MATH_DEFINES
#include <math.h>

int main() {
    long n = 700000000;
    long i, count = 0;
    double x, y, pi;
    double startTime, endTime;

    startTime = omp_get_wtime();

    srand((unsigned) time(NULL));
```

```
    for (i = 0; i < n; i++) {
        x = (double) rand() / RAND_MAX;
        y = (double) rand() / RAND_MAX;

        if (x * x + y * y <= 1) count++;
    }

    endTime = omp_get_wtime();

    pi = 4.0 * count / n;
    if(pi<0.99*M_PI || 1.01*M_PI<pi) {
        fprintf(stderr, "Error: estimated value deviates significantly: %f\n", pi);
        return 1;
    }
        printf("%2.4f\n", endTime-startTime);
    return 0;
}
```

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>


int main() {
    long n = 700000000;
    long i, count = 0;
    double x, y, pi;
    double startTime, endTime;

    startTime = omp_get_wtime();

    srand((unsigned) time(NULL));
    #pragma omp parallel for private(x, y)
    for (i = 0; i < n; i++) {
        x = (double) rand() / RAND_MAX;
        y = (double) rand() / RAND_MAX;

```

```
23          #pragma omp critical
24          {
25          if (x * x + y * y <= 1) count++;
26          }
27      }
28
29      endTime = omp_get_wtime();
30
31      pi = 4.0 * count / n;
32
33      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);
38      return 0;
39  }
```

1.1.3 Parallel Implementation using an Atomic Statement

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <time.h>
4   #include <omp.h>
5   #define _USE_MATH_DEFINES
6   #include <math.h>
7
8   int main() {
9       long n = 700000000;
10      long i, count = 0;
11      double x, y, pi;
12      double startTime, endTime;
13
14      startTime = omp_get_wtime();
15
16      srand((unsigned) time(NULL));
17      #pragma omp parallel for private(x,y)
18      for (i = 0; i < n; i++) {
19          x = (double) rand() / RAND_MAX;
20          y = (double) rand() / RAND_MAX;
21
22          if (x * x + y * y <= 1) {
```

```
            #pragma omp atomic
            count++;
        }
    }

    endTime = omp_get_wtime();

    pi = 4.0 * count / n;
    if(pi<0.99*M_PI|| 1.01*M_PI<pi) {
        fprintf(stderr, "Error: estimated value deviates significantly: %f\n", pi);
        return 1;
    }
        printf("%2.4f\n", endTime-startTime);
    return 0;
}
```

1.1.4 Parallel Implementation using a reduction clause

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define _USE_MATH_DEFINES
#include <math.h>

int main() {
    long n = 700000000;
    long i, count = 0;
    double x, y, pi;
    double startTime, endTime;

    startTime = omp_get_wtime();

    srand((unsigned) time(NULL));
    #pragma omp parallel for reduction(+: count) private(x,y)
    for (i = 0; i < n; i++) {
        x = (double) rand() / RAND_MAX;
        y = (double) rand() / RAND_MAX;

        if (x * x + y * y <= 1) count++;
    }

```

```
25      endTime = omp_get_wtime();
26
27      pi = 4.0 * count / n;
28      if(pi<0.99*M_PI|| 1.01*M_PI<pi) {
29          fprintf(stderr, "Error: estimated value deviates significantly: %f\n", pi);
30          return 1;
31      }
32          printf("%2.4f\n", endTime-startTime);
33      return 0;
34  }
```

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

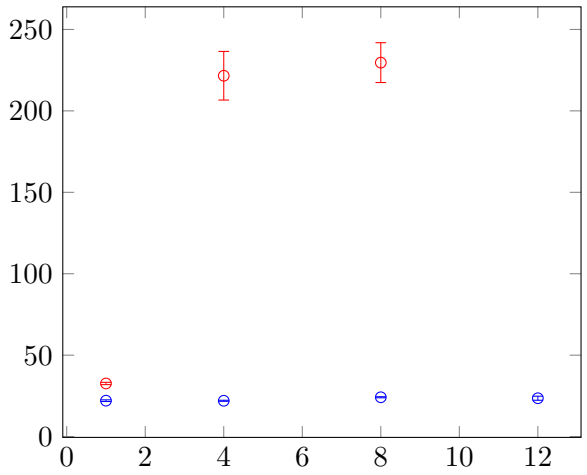
```
1   #!/bin/bash
2   # usage: sbatch [slurm_options] <executable> <number_of_measurements>
3
4   # Execute job in the partition "lva" unless you have special requirements.
5   #SBATCH --partition=lva
6   # Name your job to be able to identify it later
7   #SBATCH --job-name csba4017
8   # Redirect output stream to this file
9   #SBATCH --output=output.log
10  # Maximum number of tasks (=processes) to start in total
11  #SBATCH --ntasks=1
12  # Maximum number of tasks (=processes) to start per node
13  #SBATCH --ntasks-per-node=1
14  # Enforce exclusive node allocation, do not share with other jobs
15  #SBATCH --exclusive
16
17  ./main.sh $1 $2
```

1.2.2 Main Script

```
#!/bin/bash
# Usage: ./main.sh <executable> <number_of_measurements>

results=$1".dat"
echo "x y ey" > $results
make
for i in {1,4,8,12}
do
        measurements=$i"_"$1".log"
        export OMP_NUM_THREADS=$i
        for j in $(seq 1 $2)
        do
                ./$1 >> $measurements
        done
        ./toTable $measurements $results $2 $i
        rm $measurements
done
make clean
```

1.3 Experiment Results



1.4 Discussion