Prof. Dr. O. Rheinbach
Dipl.-Math. F. Röver
Institut für Numerische Mathematik und Optimierung
TU Bergakademie Freiberg

2nd Exercise in HPC

Exercise 1
The following is a sequential program to approximate π using Monte-Carlo simulation. It works by generating pairs (x, y) of random numbers in $[0, 1]$ and couting as a hit if

$$\sqrt{x^2 + y^2} \leq 1.$$

```
#include <stdlib.h>
#include <stdio.h>

int main()
{
  long i,hit=0;
  double x,y;
  long simulations=4000000;
  for(i=0; i<simulations; i++)
  {
    x = ((double)rand())/RAND_MAX;
    y = ((double)rand())/RAND_MAX;
    if (((x*x)+(y*y))<=1)
    {
      hit++;
    }
  }
  printf("Pi=%16.16f",(4.0*hit)/simulations);
}
```

The same program reads in Fortran

```
program main
  integer i,hit
  double precision x,y
  integer simulations
  hit=0
  simulations=4000000
  call srand(4711)
  do i=1,simulations
    x=rand()
    y=rand()
    if ((x*x + y*y) .le. 1.0d0) then
      hit=hit+1
    endif
```

```
  enddo

  write(*,*) (4.0d0*hit)/simulations
end
```

It shows, as typical for Monte-Carlo simulations, slow convergence but can easily be parallelized. Use OpenMP to parallelize this program and test it for different numbers of cores. What are your results? Try to use the function `rand_r` instead of `rand`!

Exercise 2
We have

$$\int_0^1 \frac{1}{1+x^2} = \arctan(1) - \arctan(0) = \pi/4. \tag{1}$$

Write an OpenMP program using (1) to compute in parallel an approximation of π by numerical integration using the Trapezoid and Simpson's rule (if you do not know them, see `wikipedia`). How much (real) time do you need to compute 14 digits of π using either method? What speedup do you achieve if you use $1, 2, 4, 8, 12$ processor cores?

Exercise 3
Use the following program to understand the use of locking in OpenMP. Change `num_threads(2)` to different numbers, i.e., `num_threads(4)` etc., to obtain a different number of threads.

```
#include <stdio.h>
#include <omp.h>

double d;
omp_lock_t dl;

int main() {
  omp_init_lock(&dl);

  #pragma omp parallel num_threads(2)
  {
    int tid = omp_get_thread_num();
    int i, j;

    for (i = 0; i < 3; ++i)
    {
      omp_set_lock(&dl);    // waits if already locked
      printf("Thread number %d locking, i=%d\n", tid,i);
      d=i+1000*tid;
      printf("Thread number %d setting d=%f\n", tid, d);
      printf("Thread number %d unlocking, i=%d\n", tid,i);
      omp_unset_lock(&dl);
    }
  }
  omp_destroy_lock(&dl);
  printf("Value d=%f",d);
}
```