Set 5 – Metropolis Algorithm

HPCSE I Fall 2015

Evgeni Todorov 15-909-211

Code description

Serial (disks_serial.cpp)

The serial implementation is contained in 'disks_serial.cpp'. The structure of the program is simple. An object `Problem` is created, containing all necessary functions and parameters. The state of the particles is stored in a 2D array of type `double`, `particles`, the inner dimension being (x,y), the coordinates of the centre of the particle, and the outer going over all particles. Function `init` sets up the problem with initial conditions as described in the assignment. Functions `distance` and `BC`, respectively measure the distance and correct the coordinates of the particles outside of the boundaries, taking into account the periodic boundary conditions. Function `overlap` test if the the given particles overlaps any other particle. Function `sweep` is the main core of the algorithm. At each call it goes in a loop, where it selects one of the particles with equal probability, chooses a new location for its centre from a uniformly distributed circle around the old centre, and relocates the particle unless collision is detected. If collision is detected, the particle stays at its location and next iteration of the loop begins. The number of attempts to relocate a particle is exactly equal to the number of particles, no matter how many of the attempts are successful. On average, there is an attempt to relocate every particle once per call to the sweep function. The `measure` function obtains the histogram of inter part distances at each call, and the `finalize` function averages this measurements to represent a final result. In the `main` function of the script, the `sweep` function is executed the number of times necessary for equilibrating the system in a `for` loop. Then a new `for` loop, where a measurement is performed after every sweep is executed and finally the measurement are averaged.

Parallel (disks_parS.cpp)

An attempt was made to parallelize the program, using the method of giving each thread a separate program to compute after equilibration. Few changes in the structure of the `Problem` were made. A constructor accepting a predetermined `particles` array was added to initialize a separate instance (`p_private`) of `Problem` for each thread to the equilibrated array. Secondly, the random number generators `gen_a` and `gen_p`, used to choose new displacement and particle number, respectively, are now also seeded separately for each instance. The structure of `main` is as follows: first, the system is equilibrated serially. Secondly, in an *`omp parallel`* directive each random number generator is seeded to a different value. A sweep followed by a measure is executed S number of times in total for all threads, with the S/#threads executions in a single thread dependent on each other and the programs on each thread completely separate, using an *`omp for`*. After the execution is finished, the results are accumulated in an *`omp critical`*. The outputs of all threads are averaged outside of the *`parallel`*.

Results

The following chart represents the measurements on the Euler cluster.

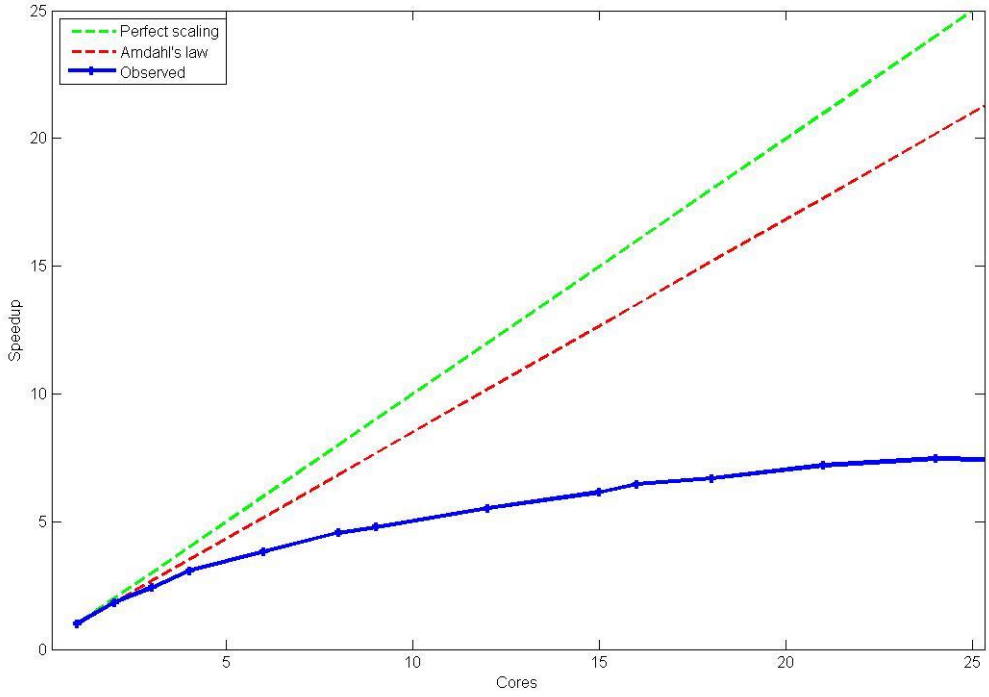

Figure 1 **Measurement executed on the Euler cluster with:**
$bsub -n 48 < disks_script -o disks_gcc
Disks was compiled with:
$g++ -std=c++11 -fopenmp -O3 disks_parS.cpp -o disks
Amdahl's law assuming: 1/6 of the code is serial and 5/6 parallel (Seq = 100, S = 500)

Since the equilibrating portion of the code is serial and included in the timing, no perfect speedup is possible, however a result close to the predicted by Amdahl's law was expected. However, a very limited speed-up, peaking at about 6 was observed. This suggests that the code might not work as intended, i.e. a racing condition might be present (although I couldn't recognize any possibility for it, since both the for loop and the random number generation *should* be completely independent). The total running time of the algorithm (disks_gcc for reference) was also quite slow. I however have no basis for comparison and cannot judge whether it is the result of inefficient coding or large problem size. The disks_serial program performs identically to the disks_parS with 1 core.