GPU Programming 2019/20

Assignment 2

Due: 1/12/2018, 23:59

In this assignment we implement a simple long term weather simulation. As dynamic model we use a transition matrix $P \in \mathbb{R}^{n \times n}$ whose entry P_{ij} describes the influence of the j^{th} grid point on the i^{th} one. The grid points form a regular grid (θ_i, ϕ_i) in spherical coordinates with N_θ and N_ϕ points in the two coordinate directions, see Fig. 1. To obtain the matrix P that describes the transitions between all points, the two-dimensional grid is linearized (analogous to how a two dimensional array is stored linearly in memory), i.e. $N = N_\theta \cdot N_\phi$.

The long term normalized weather (e.g. the temperature) on the grid points is given by the steady state distribution $\pi \in \mathbb{R}^n$ of P. It is mathematically defined by

$$\pi = \pi \, P \,, \tag{1}$$

i.e. π is a left eigenvector of P with eigenvalue 1 (which is the largest eigenvalue of P).

To obtain π we will use power iteration, which is one of the simplest algorithms to compute an eigenvector. It iterates the following two steps

$$x'_{k+1} = x_k \, P \tag{2a}$$

$$x_{k+1} = \frac{x'_{k+1}}{\|x'_{k+1}\|} \tag{2b}$$

where k indexes the iterations. Eq. 2 is performed until either a given number of iterations have been completed or a suitable convergence criterion is met (for us the former suffices, concretely we will use 50 iterations, but you are free to think about a convergence criterion). The initial vector x_0 at step $k = 0$ is a random vector with $(x_0)_i \propto \mathcal{U}([0,1])$ (i.e. the i^{th} entry of the initial vector at iteration $k = 0$ is a uniformly distributed number in $[0,1]$) and then it can be shown that the algorithm converges to the eigenvector associated with the largest eigenvalue as $k \to \infty$.

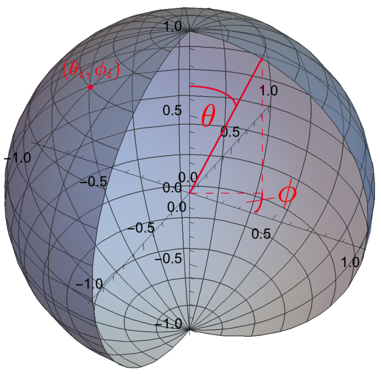

Figure 1: Regular grid in spherical (θ, ϕ) coordinates.

Your tasks are as follows:

1.) Download the skeleton code and generate the build system using `cmake`. The data files containing example transition matrices are also available for download. These can be loaded using `readMat()` in the skeleton code.

2.) Implement a serial version of the power iteration algorithm to compute the stationary distribution π for the given transition matrices. (1 point)

3.) Use `verifyStationary()` to verify that your solution is correct. Implement a suitable threshold for the return value of `verifyStationary()` so that a warning or error is generated when the accuracy goal is not met. (1 point)

4.) Parallelize your serial implementation by computing the matrix-vector product in each iteration of power iteration in parallel. (3 points)

5.) Implement a class `Barrier` that uses `std::mutex` and `std::condition_variable` to realize a barrier for `n` threads. For the interface of the class you can follow the barrier proposed for the C++ standard. (3 points)

6.) Parallelize your serial implementation using the task pool pattern and so that the threads do not have to be started again in each iteration. Use your barrier to realize the required synchronization in the iterations. (3 points)

7.) Generate a graph with the execution times of your serial and your two parallel implementations as a function of the grid resolution. Ensure that your measurements are accurate and meaningful to gauge the effectiveness of your implementations. (1 point)

Please submit your implementation as well as the graph before the deadline to gpucourse@isg.cs.ovgu.de. Your code has to compile and run with the given CMake file on the machines in G29-426.