Exercise 3

Parallel & Distributed Computer Systems

Deadline: January 28, 2024

Implement in `CUDA` the evolution of an Ising model in two dimensions for a given number of steps k.

The Ising model, named after the physicist Ernst Ising, is a mathematical model of ferromagnetism in statistical mechanics. The model consists of discrete magnetic dipole moments of atomic "spins" that can be in one of two states (+1 or -1). The spins are arranged in a square 2D lattice with periodic boundary conditions, allowing each spin to interact with its four immediate neighbors. The dipole moments update in discrete time steps according to the majority of the spins among the four neighbors of each lattice point. The edge lattice points wrap around to the other side (known as toroidal or periodic boundary conditions), i.e. if side length is n, and grid coordinates are $0 : n-1$, then the previous to node 0 is node $n-1$ and vice versa .

The magnetic moment gets the value of the majority of the spins of its neighbors and itself:

```
sign(G[i-1,j] + G[i,j-1] + G[i,j] + G[i+1,j] + G[i,j+1])
```

The Ising model belongs to the general family of Cellular Automata that may exhibit very complex behaviors even with simple update rules and only local interactions. This is the main idea behind the book by Stephen Wolfram, A New Kind of Science.

V0. Sequential

Write the sequential version, in `C` to simulate an Ising model of size `nxn` for `k` iterations, staring from a uniform random initial state. The size `n` and number of iterations `k` are given. Make sure to avoid `if` statements for the boundaries. Use two arrays, read from one and write to the other, then swap the pointers for the next iteration. Do not check if changes are made. Persuade yourself and us that your code works correctly with reasonable arguments and tests!

V1. GPU with one thread per moment

Modify the sequential code to make a kernel function and call the kernel with a grid that matches the Ising model and one thread per moment. Confirm that works correctly by matching it to the result of the sequential run.

V2. GPU with one thread computing a block of moments

Assign more work per thread, but in preparation for the next request, make each thread to compute a block of moments. Explore what block size is fastest. Confirm that your code works correctly by matching it to the result of the sequential and V1 runs.

V3. GPU with multiple thread sharing common input moments

Use shared memory so that threads do not read multiple times from the main memory but from the shared memory. Confirm that works correctly by matching it to the result of the sequential V1, and V2 runs.

What to submit

- A 4 page report in PDF format. Report execution times and speedup of your implementations for different n and k. Start from a random initial state.
- Upload the source code on GitHub, BitBucket, Dropbox, Google Drive, etc. and add a link in your report.
- Argue about the validity and effectiveness of your code. You can also include any other information you consider essential for evaluating your work.
- Cite any external sources you may have used.