COMP 429/529- Parallel Programming: Assignment 2

Due: 7 December 2021

Instructor: Didem Unat, TA: Erhan Tezcan

Notes: You may discuss the problems with your peers but the submitted work must be your own work. **The best performing implementation (the highest Gflops rate) will be acknowledged in the class and awarded with a modest gift. Because of this reason, no late assignment will be accepted, no deadline extension will be granted.** Please submit your answers through blackboard. This assignment is worth 20% of your total grade, can be done as a team of 2.

Cardiac Electrophysiology Simulation

In this assignment you'll implement the Aliev-Panfilov heart electrophysiology simulator using CUDA. Along with this project description, studying lectures on stencil computation might be very helpful for the assignment. Since implementing/debugging parallel codes under CUDA can be time consuming, give yourself sufficient time to complete the assignment (I am giving you more than enough time). In addition, this assignment requires you to conduct various performance studies. Please also allow yourself enough time for performance measurements. The performance results will be collected on the KUACC cluster. For instructions about how to compile and run jobs on KUACC, please read the Assignment and Environment sections carefully.

Background

Simulations play an important role in science, medicine, and engineering. For example, a cardiac electrophysiology simulator can be used for clinical diagnostic and therapeutic purposes. Cell simulators entail solving a coupled set of a equations: a system of Ordinary Differential Equations (ODEs) together with Partial Differential Equations (PDEs). We will not be focusing on the underlying numerical issues, but if you are interested in learning more about them please refer to references at the end of this manuscript.

Our simulator models the propagation of electrical signals in the heart, and it incorporates a cell model describing the kinetics of a the membrane of a single cell. The PDE couples multiple cells into a system. There can be different cell models, with varying degrees of complexity. We will use a model known as the Aliev-Panfilov model, that maintains 2 state variables, and solves one PDE. This simple model can account for complex behavior such as how spiral waves break up and form elaborate patterns. Spiral waves can lead to life threatening situations such as ventricular fibrillation (a medical condition when the heart

muscle twitches randomly rather than contracting in a coordinated fashion).

Our simulator models electrical signals in an idealized system in which voltages vary at discrete points in time, called timesteps on discrete positions of a mesh of points. In our case we'll use a uniformly spaced mesh (Irregular meshes are also possible, but are more difficult to parallelize.) At each time step, the simulator updates the voltage according to nearest neighboring positions in space and time. This is done first in space, and next in time. Nearest neighbors in space are defined on the north, south, east and west.

In general, the finer the mesh (and hence the more numerous the points) the more accurate the solution, but for an increased computational cost. In a similar fashion, the smaller the timestep, the more accurate the solution, but at a higher computational cost. To simplify our performance studies, we will run our simulations for a given number of iterations rather than a given amount of simulated time because actual simulation takes too long (several days) on large grids.

Simulator

The simulator keeps track of two state variables that characterize the electrophysiology we are simulating. Both are represented as 2D arrays. The first variable, called the excitation, is stored in the $E[\,][\,]$ array. The second variable, called the recovery variable, is stored in the $R[\,][\,]$ array. Lastly, we store $Eprev$, the voltage at the previous timestep. We need this to advance the voltage over time.
Since we are using the method of finite differences to solve the problem, we discretize the variables E and R by considering the values only at a regularly spaced set of discrete points. Here is the formula for solving the PDE, where the E and $Eprev$ refer to the voltage at current and previous timestep, respectively, and the constant α is defined in the simulator:

```
1
2  E[i,j] =  Eprev[i,j] +  alpha* ( Eprev[i+1,j] + Eprev[i-1,j] +
3                                   Eprev[i,j+1] + Eprev[i,j-1] - 4 * Eprev[i,j])
```

Here is the formula for solving the ODE, where references to E and R correspond to whole arrays. Expressions involving whole arrays are pointwise operations (the value on i,j depends only on the value of i,j), and the constants $kk, a, b, \epsilon, M1$ and $M2$ are defined in the simulator and dt is the time step size.

```
1
2  E = E -  dt*( kk * E * (E - a) * (E - 1) + E * R);
3
4  R = R + dt*(epsilon + M1*R / ( E + M2)) * (-R - kk * E * (E-b-1));
```

Serial Code

You are provided with a working serial simulator that uses the Aliev-Panfilov cell model. The simulator includes a plotting capability (using gnuplot) which you can use to debug your code, and also to observe the simulation dynamics. The plot frequency can be adjusted from command line. Your timings results will be taken **when the plotting is disabled.** The simulator has various options as follows.

```
./cardiacsim

With the arguments
-t <float> Duration of simulation
-n <int> Number of mesh points in the x and y dimensions
-p <int> Plot the solution as the simulator runs, at regular intervals
-x <int> block size in x dim
-y <int> block size in y dim
-v <int> kernel version
Example command line
        ./cardiacsim -n 400 -t 1000  -p 100
```

The example will simulate on a 400 x 400 box, and run to 1000 units of simulated time, plotting the evolving solution every 100 units of simulated time.

You'll notice that the solution arrays are allocated 2 larger than the domain size (n + 2). We pad the boundaries with a cell on each side, in order to properly handle ghost cell updates.

Assignment

You will parallelize the cardiac simulation using CUDA. Starting with the serial implementation provided, the assignment requires 4 versions of the same code:

- Version 1: Parallelize the simulator using single GPU. First implement a naive version that makes all the references to global memory. Make sure that your naive version works correctly before you implement the other three versions. Check if all the data allocations on the GPU, data transfers and synchronisations are implemented correctly.
- Version 2: Fuse all the kernels into one kernel. You can fuse the ODE and PDE loops into a single loop, thus into a single kernel.
- Version 3: Use temporary variables to eliminate global memory references for R and E arrays in the ODEs.
- Version 4: Then optimise your CUDA implementation by using shared memory (on-chip memory) on the GPU by bringing a 2D block into shared memory and sharing it with multiple threads in the same thread block.
- You should implement these optimizations on top of each other (e.g. Version 3 should be implemented on top of Version 2).

- For details on how to implement these optimizations, refer to lectures on memory optimizations for stencil computation.
- Note that these optimisations do not guarantee performance improvement. Implement them and observe how they affect the performance.

For full credit, you have to implement all these versions. However, if you are running for the best implementation competition, then you will need to further optimise the code. You can refer to the Nvidia Best Practices:
http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#abstract

Ghost Cells

Since all the CUDA threads share global memory, there is no need to exchange ghost cells between thread blocks. However, the physical boundaries of the domain need to be updated every iteration using mirror boundary updates. You can do this in a series of separate kernel launches on the GPU, which will be a bit costly, or embed mirror boundary updates into a single kernel launch.

Reporting Performance

- Use N=1024 and t=500 for your studies.
- Conduct a performance study without the plotter is on.
- Observe that data transfer time affects the performance. When you measure the execution time and Gflop/s rates, do not include the data transfer time.
- Run the stream benchmark variant provided by us to measure the device bandwidth. The bandwidth rate measured by the benchmark is the highest bandwidth rate that can be achieved on the device. Then compare it against the bandwidth rate that your implementation achieves. As you optimise the code you will get closer to this peak bandwidth but you will never achieve the same bandwidth, can only achieve a fraction of it.
- Tune the block size for your implementation and draw a performance chart for various block sizes.
- Compare the performance of your best implementation with the CPU version provided to you (serial).
- Document these in your report.

Environment

We will be using the KUACC cluster for the assignment.

- Note that everyone in the class will be using the GPUs on the cluster for the assignment. Make sure to give yourself enough time to develop and test your program before the project deadline.
- Collect performance data on the same GPU device, and on your report indicate which GPU device you used. Note that V100 GPUs are the fastest but they are busiest devices. You can prepare your report using an older generation of GPUs (K20 or K80) but collect the performance of your best implementation on a V100. An another important point is that Tesla T4 GPUs do not support double precision, your simulation results might be incorrect if you test it on T4s.
- If you want to develop and test the application on your local machine and your machine is a CUDA-capable Nvidia GPU, then you need to install CUDA Software Development Kit (SDK). Visit Nvidia's website for details. Pay attention whether it supports double precision. If not, you can change the type declarations to single for the purpose of code development.

Submission

- Submit all four versions of the code, label them properly.
- The first paragraph of your report should clearly state which versions work properly or which version is the best performant one. We will only test the final implementation (version 4) if the report does't guide us.
- In the first paragraph, if you are participating for the competition, then explicitly state that you nominate yourself and list the Gflops rate you achieve for N=1024 and t=500.
- Document your work in a well-written report which discusses your findings. Offer insight into your results and plots.
- Submit both the report and source code electronically through blackboard.
- Please create a parent folder named after your username(s). Your parent folder should include a report in pdf and a subdirectory for the source. Include all the necessary files to compile your code. Be sure to delete all object and executable files before creating a zip file.

Grading

Your grade will depend on 3 factors: performance, correctness and the depth and your explanations of observed performance in your report.

Implementation (80 points): Each version (20 pts each)

Report (20 points): implementation description and any tuning/optimisation you performed (10 pts), performance study (10 pts).

No bonus point for the best implementation.

GOOD LUCK.

References

https://www.simula.no/publications/stability-two-time-integrators-aliev-panfilov-system.

GOOD LUCK.