2D Gray-Scott Model in CUDA

Uroš Lotrič, Davor Sluga

April 2025

1 Gray-Scott Model of a Reaction-Diffusion System

The Gray-Scott model [1] is a reaction-diffusion system that simulates the behaviour of two species, U and V, interacting over time and space. It models the following interaction:

$$U + 2V \rightarrow 3V$$

The reaction consumes U and produces V. Consequently, the amount of both species needs to be controlled to maintain the reaction.

The following partial differential equations govern the reaction:

$$\begin{aligned} \frac{\partial U}{\partial t} &= -UV^2 + F(1-U) + D_u \nabla^2 U \quad , \\ \frac{\partial V}{\partial t} &= UV^2 - (F+k)V + D_v \nabla^2 V \quad . \end{aligned}$$

Variables U and V are the concentrations of the two reacting species. The two equations describe the rate of change of U and V. The first term on the right side of the equation describes the reaction between the two species. As U gets consumed by the reaction to produce V, the term has a negative sign in the first equation and a positive sign in the second. The second term of the first equation describes the rate at which U is being replenished externally. The replenishment rate is governed by the feed factor F. In the second equation, we have the opposite situation, where the second term determines the removal rate of species V and is governed by the kill rate k. The last term in both equations describes the diffusion of each species, where D_u and D_v are diffusion coefficients. This system generates complex and often visually appealing patterns as the two species diffuse and react (figure 1).

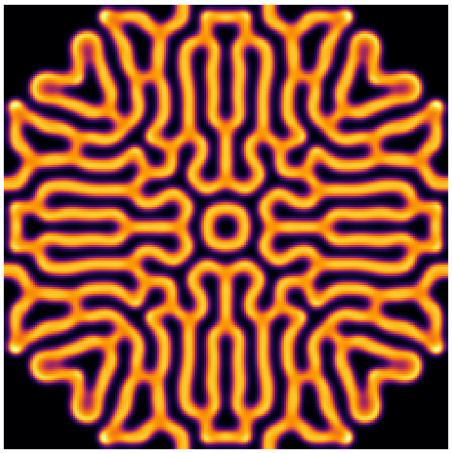

Figure 1: Example of a pattern generated by the Gray-Scott model. This image visualises the concentration of species V after several thousand iterations.

2 Solving Gray-Scott Model using Finite Differences method

To numerically solve the Gray-Scott equations, we discretise both space and time. Let $U_{i,j}^n$ and $V_{i,j}^n$ represent the concentrations of U and V at position (i, j) and time step n.

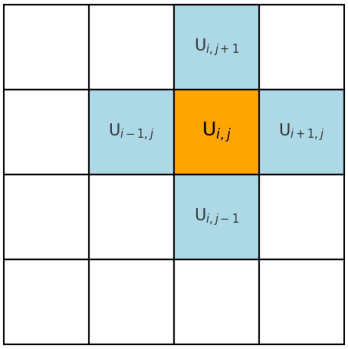

Figure 2: Representation of the 5-point stencil used to compute the Laplacian for grid U.

We approximate the Laplacian using a standard 5-point stencil2,

$$\nabla^2 U^n_{i,j} \approx U^n_{i+1,j} + U^n_{i-1,j} + U^n_{i,j+1} + U^n_{i,j-1} - 4U^n_{i,j} \quad ,$$
$$\nabla^2 V^n_{i,j} \approx V^n_{i+1,j} + V^n_{i-1,j} + V^n_{i,j+1} + V^n_{i,j-1} - 4V^n_{i,j} \quad .$$

Substituting these into the original equations and discretising time using the forward Euler method, we obtain the update rules:

$$U^{n+1}_{i,j} = U^n_{i,j} + \Delta t \left(-U^n_{i,j}(V^n_{i,j})^2 + F(1 - U^n_{i,j}) + D_u \nabla^2 U^n_{i,j}\right) \quad ,$$
$$V^{n+1}_{i,j} = V^n_{i,j} + \Delta t \left(+U^n_{i,j}(V^n_{i,j})^2 - (F + k)V^n_{i,j} + D_v \nabla^2 V^n_{i,j}\right) \quad .$$

These equations are applied iteratively at each grid point for a number of time steps, allowing the simulation to evolve.

Algorithm 1 2D Gray-Scott Solver Pseudocode

```
procedure GrayScottSolver(U, V, D_u, D_v, F, k, Δt, steps)
    for n = 1 to steps do
        for each grid cell (i, j) do
            ∇²U ← U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} − 4 · U_{i,j}
            ∇²V ← V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} − 4 · V_{i,j}
            UV² ← U_{i,j} · V²_{i,j}
            U^new_{i,j} ← U_{i,j} + Δt · (−UV² + F · (1 − U_{i,j}) + D_u · ∇²U)
            V^new_{i,j} ← V_{i,j} + Δt · (+UV² − (F + k) · V_{i,j} + D_v · ∇²V)
        end for
        for each grid cell (i, j) do
            U_{i,j} = U^new_{i,j}
            V_{i,j} = V^new_{i,j}
        end for
    end for
end procedure
```

When the stencil reaches over the edge of the simulation grid, wrap the indices to the opposite side of the grid. E.g.:

$$U_{i+1,j} \to U_{((i+1) \bmod N),j}$$
$$U_{i-1,j} \to U_{((i-1+N) \bmod N),j} \quad ,$$

where N is the height of the grid.

3 Parallel Gray-Scott Solver

The Gray-Scott model can be easily parallelised. For both concentration grids, new values in each cell can be computed independently and, thus, in parallel, based on the values of the neighbouring cells in the previous iteration. Of course,

there exists a dependence between iterations, so all of the computations in the previous iteration need to finish before we proceed to the next iteration.

To ensure that concentrations in each point at time step $n+1$ are obtained from concentrations at time step n, we must use two arrays for each concentration. Suppose a concentration in time step n is stored in the first array. Then we compute the concentration at time step $n+1$ into the second array. In the next iteration, we compute the concentration at time step $n+2$ into the first array using the values from the second array, and so on, in each iteration, swapping the source and target array.

4 Assignment

Implement a sequential and parallel version of the Gray-Scott model using C/C++ and CUDA. The algorithm should work for square grids of arbitrary size. You can find some interesting starting configurations and simulation parameters here.

One example of starting parameters for a grid of size $N \times N$: $\Delta t = 1$, $D_u = 0.16$, $D_v = 0.08$, $F = 0.060$, and $k = 0.062$, with a non-zero $N/4 \times N/4$ square shape ($U_{i,j} = 0.75$, $V_{i,j} = 0.25$) seeded in the center of concentration grids U and V. All other cell values of U should be set to 1.0 and all other values of V should be set to 0.0

Basic tasks (for grades 6-8):

- Parallelise the algorithm using CUDA as efficiently as possible. Avoid unnecessary memory transfers between the host and the device. When dividing the workload, find the optimal number of threads and thread block size.
- Measure the execution time of the algorithm on Arnes cluster for different grid sizes. Use the next grid sizes: 256x256, 512x512, 1024x1024, 2048x2048 and 4096x4096. Benchmark the algorithm on 5000 simulation steps. When measuring time, the data transfers to and from the GPU must be also included.
- Compute the speed-up $S = t_s/t_p$ of your algorithm for each image size; t_s is the execution time of the sequential algorithm on the CPU, and t_p is the execution time of the parallel algorithm on the GPU.
- Visualise the resulting concentration grid V. You can even create a video showing how the concentration of species V evolves through time.
- Write a short report (1-2 pages) summarising your solution and presenting the measurements performed on the cluster. The main focus should be on presenting and explaining the time measurements and speed-ups.

- Hand in your code and the report to ucilnica through the appropriate form by the specified deadline (**13. 5. 2025**) and defend your code and report during labs.

Bonus tasks (for grades 9-10):

- Use shared memory within thread blocks to store local tiles of the grid, allowing faster access to neighbouring cells.
- Experiment with and optimise how the simulation grid is split between thread blocks, e.g. square blocks, stripes, etc.
- Utilise two GPUs to perform the simulation; split the work between them accordingly and exchange the data as necessary.
- Perform additional optimisations as you see fit.

5 HPC Challenge

Implement a parallel version of the 3D Gray-Scott Model of a Reaction-Diffusion System. In this version, concentrations are denoted with 3 indices, $U_{i,j,k}$, and $V_{i,j,k}$. In 3D, Laplacians using the standard 7-point stencil become

$$\nabla^2 U^n_{i,j,k} \approx U^n_{i+1,j,k} + U^n_{i-1,j,k} + U^n_{i,j+1,k} + U^n_{i,j-1,k} + U^n_{i,j,k+1} + U^n_{i,j,k-1} - 6U^n_{i,j,k} \ ,$$
$$\nabla^2 V^n_{i,j,k} \approx V^n_{i+1,j,k} + V^n_{i-1,j,k} + V^n_{i,j+1,k} + V^n_{i,j-1,k} + V^n_{i,j,k+1} + V^n_{i,j,k-1} - 6V^n_{i,j,k} \ .$$

The update equations are then

$$U^{n+1}_{i,j,k} = U^n_{i,j,k} + \Delta t \left(-U^n_{i,j,k} (V^n_{i,j,k})^2 + F(1 - U^n_{i,j,k}) + D_u \nabla^2 U^n_{i,j,k} \right) \quad ,$$
$$V^{n+1}_{i,j,k} = V^n_{i,j,k} + \Delta t \left(+U^n_{i,j,k} (V^n_{i,j,k})^2 - (F + k) V^n_{i,j,k} + D_v \nabla^2 V^n_{i,j,k} \right) \quad .$$

For the challenge, evolve the initial volume for a given number of iterations and output the average concentration of V. The cube-shaped volume will range from $128 \times 128 \times 128$ to $1536 \times 1536 \times 1536$. The number of iterations will be large enough (a few hundred) to reduce the effects of initialisation.

Prepare a solution in C/C++ that supports graphics accelerators using CUDA. You are encouraged to combine CUDA with shared memory systems using OpenMP library and distributed memory systems using OpenMPI in a way to optimize the execution times.

Template code and run scripts are provided on the repository. Your task is to implement the stated problem in the file `gray_scott.cu`. The organizers will only consider the solutions built and executed using the script `run-gray-scott.sh`. Each solution will be tested in an isolated environment consisting of two 12-core nodes, each having two Nvidia V100 GPUS. Submit the solutions through the course web page `https://ucilnica.fri.uni-lj.si/hpc`.

5.1 Rules of the game

- Each student works on his own; any suspicion of plagiarism leads to disqualification.
- The challenge is open until Sunday, June 1st.
- Only solutions which include GPU support qualify for the challenge.
- The organisers will evaluate the submitted solutions for correctness and performance in terms of running time. Tests will be performed on the Arnes cluster in an isolated environment using whole nodes in each run.
- The solutions will be ranked according to the achieved performance.
- Rewards:
  - Winner gets 6/10 perfectly answered questions on the written exam.
  - Any solution which gives correct results in all test runs equals 2/10 perfectly answered questions on the written exam.
  - The organisers reserve the right to appropriately reward other solutions with 2-6/10 perfectly answered questions on the written exam.
- The candidate can use the rewards only on the first written exam they take.

References

[1] John E. Pearson, Complex Patterns in a Simple System. Science 261, 189-192, 1993.