Advanced Computational Methods in Condensed Matter Physics - PHYS 790A
Spring 2026

Department of Physics - Northern Illinois University
Prof. Andreas Glatz
www.aglatz.net/home/teaching/compphys_S2026

Project 1

due *2026-03-03*

I. COMPLEX GINZBURG-LANDAU EQUATION ON GPUS

"The cubic complex Ginzburg-Landau equation is one of the most-studied nonlinear equations in the physics community. It describes a vast variety of phenomena from nonlinear waves to second-order phase transitions, from superconductivity, superfluidity, and Bose-Einstein condensation to liquid crystals and strings in field theory." [Rev. Mod. Phys. **74**, 99 (2002)].
Here we are going to study the one- and two-dimensional time-dependent complex Ginzburg-Landau equation (CGLE)

$$\partial_t A = A + (1+\imath b)\Delta A - (1+\imath c)|A|^2 A\,,$$

where $A = A(x;t)$ (1D) or $A = A(x,y;t)$ (2D) is a complex function. The two real parameters b and c characterize the linear and non-linear dispersion.
The CGLE shall be solved numerically on a grid with $N_x \times N_y$ equidistant mesh points (in 2D, N_x in 1D) on a GPU using a *quasi-spectral split-step* method and an iterative, parallel solver. Boundary conditions are to be periodic in space.
For the physical dimension of the simulation grid, we introduce the dimensionless length scales L_x and L_y, such that the space is discretized as $h_x = L_x/N_x$ and $h_y = L_y/N_y$.
The time is discretized as h_t and the CGLE is integrated for N_t time steps.
Therefore the simulation parameters are:

- real values: b, c, h_t, L_x, L_y
- integers: N_x, N_y, N_t

($N_y = 1$ can be used to indicate the 1D case, or you would need to specify the dimensionality as an additional parameter.)
For stability and accuracy use $h_t/h_{x,y}^2 < 1/2$.

A. Quasi-spectral split-step method

This methods integrates the CGLE effectively in higher order in real space and at the same time provides a way to calculate the Laplacian in Fourier space using a Fast-Fourier-Transformation, which uses higher order polynomials (the Fourier series), i.e., a global representation - in contrast to local finite difference methods and therefore is, in principle, more accurate. We can use this here directly due to periodic boundary conditions.
In the first, real space step, one calculates:

$$B(x,y) \equiv e^{\Delta t[1-(1+\imath c)|A|^2]} A(x,y;t)\,.$$

Then $\hat{B}(k_x,k_y) = \mathrm{FFT}(B)$, $\hat{C}(k_x,k_y) = \exp[-\Delta t(1+\imath b)\mathbf{k}^2]\hat{B}(k_x,k_y)/\mathcal{N}$ and finally $A(x,y;t+\Delta t) = \mathrm{FFT}^{-1}(\hat{C})$

B. Jacobi Iteration solver

Using the Crank-Nicolson scheme, discretize the equation and solve the linearized equation system with Jacobi iterations. For the nonlinear $|A|^2$ term use the current time solution as estimator or an explicit Euler step in the Crank Nicolson scheme, i.e., for the time discretization $|A|^2 \to (|A|^2(t) + |A|^2_{\mathrm{est}}(t+h_t))/2$, where $|A|^2_{\mathrm{est}}$ is calculated by an explicit Euler step, which can be implemented in the initialization kernel function for the iteration matrix and rhs elements.

C. Tasks

A template code with compile options is provided! This template code includes FFT wrappers, image output functions, and iterative convergence check code.

a) Implement the quasi-spectral split-step solver for the CGLE in CUDA. Please use type `REAL` for real values and `COMPLEX` for complex numbers.

- implement all complex operations for real and imaginary parts separately
- use the parameter `MAXT` for the maximum number of threads per block, calculate the number of blocks accordingly based on this and N_x and N_y
- the Fast Fourier Transform requires manual normalization – multiply $\hat{A}(k_x, k_y; t)$ by $1/N$, $N = N_x \times N_y$
- in order to determine the "k"-index in Fourier space use the macro `k_INDEX(i,`N_x`)` due to the way the FFT is implemented

b) Discretize the CGLE using a implicit Crank Nicolson scheme for the time and central difference in space, derive the iteration matrix elements and the rhs of the linearized equation. Implement the iterative solver. This requires to implement two kernel function to the code (initialization and iteration).

c) In order to "see" the evolution of A, we need to output it. We do not want to do that every time step, but only every N_{step} time steps. Therefore your time loop should check if $n \bmod N_{\text{step}} = 0$, where n is the current time step and then "pull" A from the GPU and write it to a file (use a frame counter for file name generation in the 2D case). Use the supplied routine `writeBM_real(string fn,REAL *a,REAL m,REAL M,int nx,int ny,int cgrad)` for the image output - you should be able to output either $|A|^2$ and $\arg(A)$ (`atan2` function) or $\mathrm{Re}(A)$ and $\mathrm{Im}(A)$. In 1D, the y-axis is the time. The individual 2D images can be combined into a movie using, e.g., `ffmpeg`. Suggestion: Write a simple CUDA kernel, which "splits" A into the two respective components as `REAL` arrays and copy those (or do this on the CPU).

Below are suggested parameters to solve the CGLE, taken from the above cited review paper, so you can compare your results to images shown there. Use both solvers for each parameter set.

d) Solve the CGLE in 1D. In the output image, the y-coordinate of your final image should be the output time steps specified by N_{step}: $N_x = 1024$, $L_x = 500$, $N_t = 7680$, $\Delta t = 0.1$, $N_{\text{step}} = 10$, $b = 0.6$, $c = 1.4$.

e) Solve the CGLE in 2D for $N_x = N_y = 256$ (use powers of 2 for optimal performance) and $L_x = L_y = 100$ [you can double these for nicer outputs], $N_t = 2000$ and $\Delta t = 0.1$ (smaller for finite differences), $b = 0$, $c = 1$, and $N_{\text{step}} = 10$ (i.e. 200 frames). Initialize A with random numbers in $[-0.1, 0.1]$ (both real and imaginary parts).

f) 2D: $c = 0.7$, $b = -2$, $N_x = N_y = 1024$, $L_x = L_y = 500$

g) 2D: $c = 0.75$, $b = 2$, $N_x = N_y = 1024$, $L_x = L_y = 500$

Remarks: For d)-g) use both solvers, if you prefer to solve the CGLE in 3D (instead of 1D and 2D), you can do so and pick appropriate parameters (In that case you need to use external visualization tools). You are also free to use different sets of parameters from the above.

D. Presentation

The presentation of the above results should be about 15mins on the due date day, it should contain:

- Discretization scheme and method for both solvers. Point out important details of each solver.
- Your results and briefly explain the behavior.
- For the results of your simulation show differences between solvers - if any, and explain the (potential) reason and how to avoid these.
- Some brief performance analysis of each solver: Show differences in run-time and, e.g., how many iterations are needed for the Jacobi solver.