Fourier-pseudospectral method for Cahn-Hilliard Equation on GPU

Kangping Zhu

Courant Institute of Mathematical Sciences

kangping@cims.nyu.edu

December 18, 2012

Background

Cahn-Hilliard & Allen-Cahn Equation

Modelling phase separation of binary fluid or material during polymer formation and etc. It can be viewed as gradient descent of Modica-Mortola energy under different Soblev norm.

$$E_\epsilon(u) = \int \frac{\epsilon}{2}|\nabla u|^2 + \frac{(u^2-1)^2}{\epsilon} \tag{1}$$

$$\frac{\partial u_\epsilon}{\partial t} = -(-\Delta)^\alpha(-\Delta u + u^3 - u) \tag{2}$$

Motivation

Cahn-Hilliard Equation

Coarsening rate of the flow. i.e. How fast or slow will the binary polymer formation finish?
For Cahn-Hilliard equation. Length scale will behave like $t^{\frac{1}{3}}$.
For Allen-Cahn equation maybe $t^{\frac{1}{2}}$?

Numerical Project

Fractional Cahn-Hilliard Equation

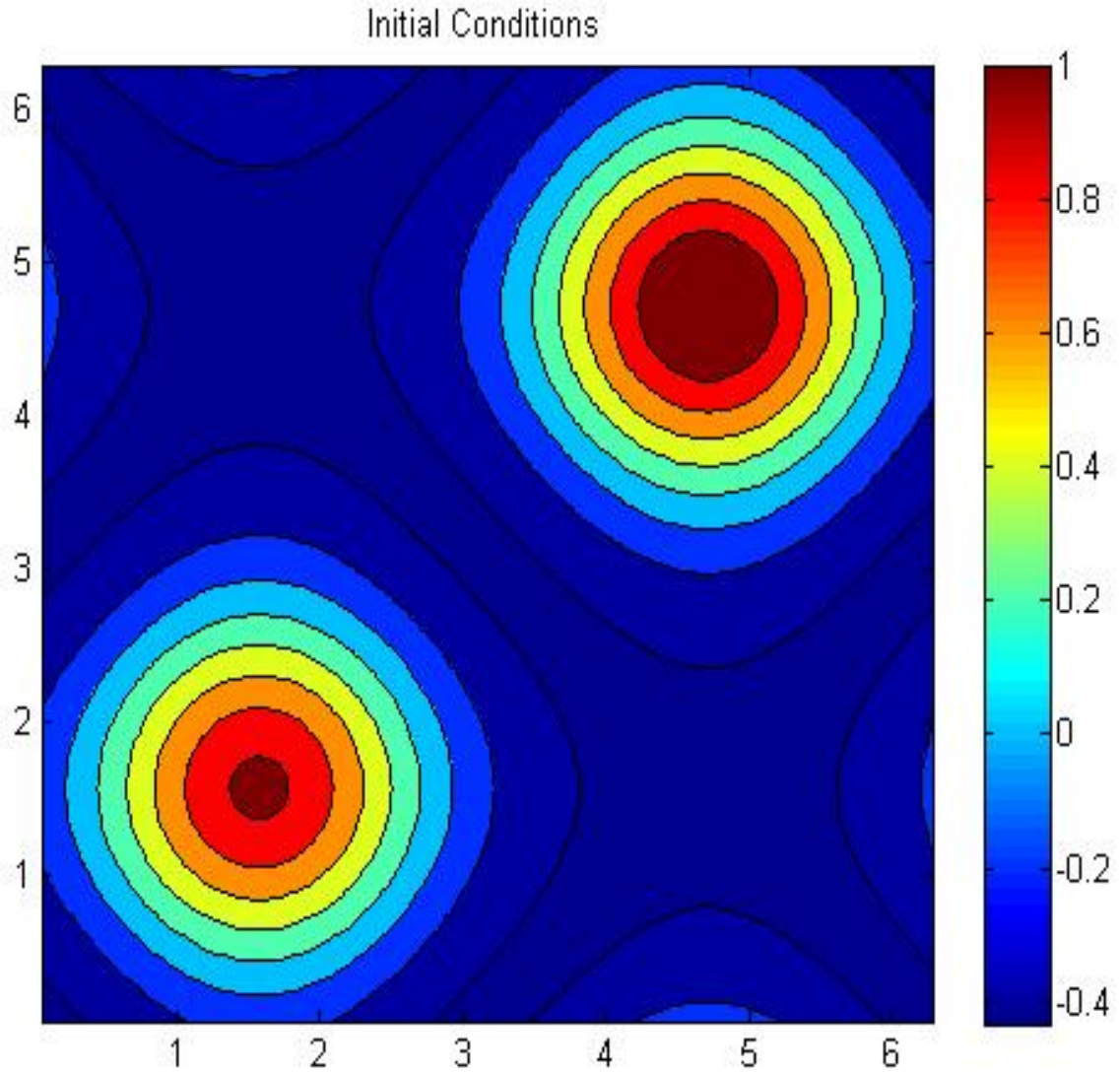
Consider Fractional Cahn-Hilliard equation i.e fractional α
$\alpha = \frac{1}{2}$ means binary separation on 2D surface of 3D material.
$\alpha = \frac{1}{2}$ is critical point that behaviour of the PDE changed.
This numerical project is aiming to find the right time scale for $\alpha < \frac{1}{2}$.

Numerical Method

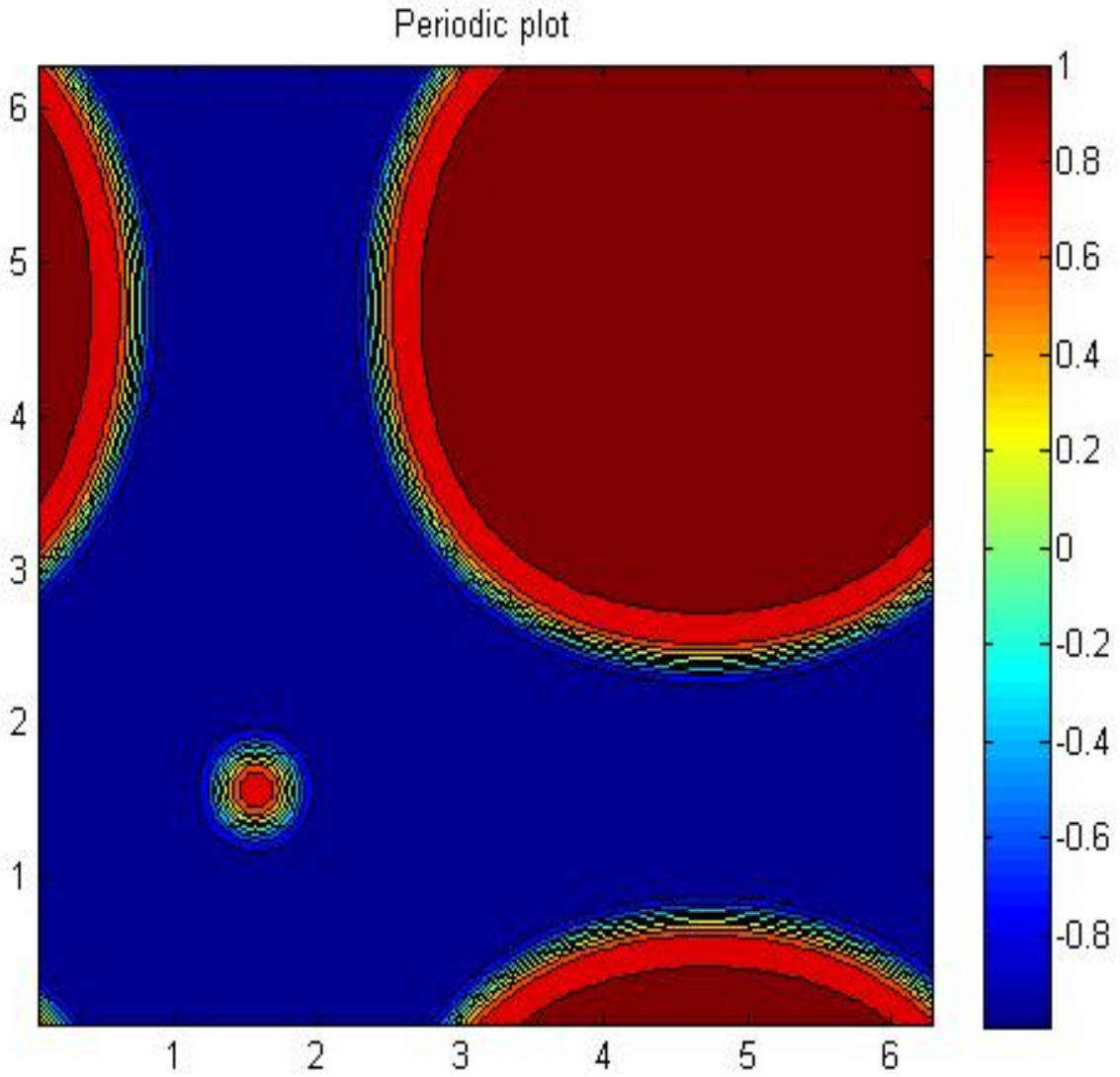
Pseudo-spectral Method

Fraction Laplacian is easy to deal with in Frouier Space
Fourier transform on both side of equation then use inverse Fourier transform
Use implicit time stepping to get accuracy and stability.
Use Pre-conditioned conjugate gradient method to solve each time step

Initial time



Later



Down to earth FINALLY

Numerical task in the project

'FAST!!' Fourier Transform

Why? 10^5 conjugated gradient step in total,10 FFT each step.

Reduction

Matrix entry-wise multiplication(solve by customizing FFT kernel to hide the calculation)

Demo

Demo

Let's see a simple demo of my several version of FFT

Demo

Demo

Let's see a simple demo of my several version of FFT
BTW CuFFT achieves over 300GFLOPs on Tesla Fermi.

Approach to get a FAST FFT

Sequential 1DFFT

Higher Radix usually gives better result.
1. Better complexity, but not much. up to 25% better than original Cooley-Tukey Radix-4
2. Better memory access pattern, less data transfer more real work!
My implementation of Radix-2 to Radix-4 to Radix-8 each gives me a factor of 2 speed up
FFTW usually use radix-16 or radix-32 depends on problem size
PS. UNROLL the loops

Approach to get a FAST FFT

GPU 1DFFT

1. Higher Radix is even better. More work in between memory access.
2. Separate first pass and later pass. factor of 2
3. Exchange data between local work items when possible 30% speed up
4. Put first three pass in a single kernel. (64 point FFT using local sync)

However, this means we have to use hierarchy FFT.

PS. Better to generate the code automatically?

Approach to get a FAST FFT

GPU 2DFFT

1. Don't follow the TEXTBOOK!
2. Multiple different kernels in one for loop is bad
3. Put everything in one kernel if possible. i.e write 2D FFT kernel (Don't be lazy)

Approach to get a FAST FFT

Leftover & future work

1. 3D matrix transposition(Hierarchy FFT)
2. Complex multiplication on GPU.
3. For special size matrix, better hierarchy separation.

References

Brian Wetton et al. (2012)

High accuracy solutions to energy gradient flows from material science models

Journal of Computational Physics submitted.

Naga Govindaraju et al. (2008)

High performance discrete Fourier Transforms on Graphics Processors

Supercomputing

The End