Report on GPU solver for Fourier Pseudo-spectral method on Cahn-Hilliard equation

Kangping Zhu
Courant Institute, NYU
kangping@cims.nyu.edu

December 20, 2012

1 Problem description

This project is about solving gradient descent of H^-s Soblev norm of Modica-Motorla energy

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

So the PDE can be viewed as such

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

Numerical method for Cahn-Hilliard equation has been well-studied, but few can be generalized to fractional Cahn-Hilliard equation. In this project to modified the numerical method proposed by Brian Wetton et al in the paper High accuracy solutions to energy gradient flows from material science models[1]. The method they described in the paper is a pseudo-spectral method suitable for considering fractional laplacian. Mathematically speaking, it is a implicit time stepping method using adaptive time step. At each time step, using conjugate gradient to solve the minimization problem. However, in typical interesting cases like Cahn-Hilliard and Fractional Cahn-Hilliard equation, PDE contains non-linear term, which is not suitable for Fourier method in general. A way to get around this is linearize the non-linear term, use the modified discretization as a pre-conditioner for the conjugate gradient step. The discretization is as follow:

$$U^m - k_m[\epsilon\Delta_h^s\Delta_h U^m + \Delta_h^s\frac{W'(U^m)}{\epsilon}] - U^{m-1} = 0 \tag{3}$$

And the pre-conditioner is

$$U^m - k_m[\epsilon\Delta_h^s\Delta_h U^m + \Delta_h^s\frac{W''(U^{m-1})U^m}{\epsilon}] - U^{m-1} = 0 \tag{4}$$

Where $W(u)$ is the double well potential.
This numerical method is particularly suitable for a GPU implementation, since

the real numerical tasks are 2DFFT, reduction and matrix entry-wise multiplication. In this project, all the matrix entry-wise multiplication can be hide into FFT kernels for speed up, therefore by customized the FFT kernel, we can gain nearly a 2X speed up. That leaves the most important task being a decent FFT kernel.

2 2DFFT

FFT is a well-studied problem. In this project, our most important reference is the paper by Naga K. Govindaraju et al[2] [High performance Discrete Fourier Transforms on Graphics Processors]. And we also referenced to the APPLE_OPENCL_FFT package.[3]
The most naive version of FFT pseudo code is as follows:

Code for Stockham FFT Cite form NK Goveindataju et al[2]
for(Ns = 1;Ns < N; Ns *= 4)

for (int j = 0; j<N/4; j++)

fftiteration(j,N,Ns,a,b);

d = a ;
a = b;
b = d;

void FftIteration(int j, int N, int R , int Ns,
float2* data0, float2*data1)
‘ float2 v[R];
int idxS = j;
float angle = -2 ∗ *M_PI**(j%Ns)/(Ns*R);
for(int r=0 ; r <R; r++)
v[r] = data0[idxS+r*N/R];
v[r] *= (cos(r*angle), sin(r*angle));

FFT< 4 >(v);
int idxD = expand(j,Ns,R);
for(int r=0; r<R; r++)
data1[idxD+r*Ns] = v[r];

This code actually behaves well when the array size is big. Typically because the local synchronization would only benefit the first three to four passes of the algorithm when running on a GPU. However, for the 2D FFT problem we are facing in this project, we only need 1D fft for size 1024. For this particular size,

first three passes take too long to synchronize. For a better performance, we decide to use hierarchy FFT, which we view the 1024 number array as 64*16 matrix, the first kernel will handle 16 size of 64 FFT in one pass, then the second kernel will perform 64 size of 16 FFT. In this way, we take full advantage of the local memory sync to reduce the global memory sync we need. This modification will not help much in 1D especially when N is large, as the transposition is taking too long. But for 2D size of 1024*1024, in our test, this is the fastest way.
The most commonly used existing FFT package on GPU is CuFFT, which has been proved to be very fast(UP to 400 GFLOP/s). However, in our project, using existing package could be sub-optimal as we will lose the fact that we can hide those multiplication into FFT kernels.

3 Scale

In this project, the targeting scale is 1024*1024 floating point matrix. There are two considerations in choosing this particular scale. First, for the numerical simulation to make sense, physically one need to use small ϵ, as it is usually in the microscopic scale. However, to achieve satisfying result, one need at least ten grid points across the microscopic scale, as the interface interaction is crucial in the dynamics. If we choose $\epsilon = 0.01$, then grid size should be 10^{-3}. This gives the lower bound for our simulation. It will hardly give meaningful result for a smaller scale. However, conjugate gradient method in this case is memory consuming, as we have to store the conjugate gradient directions and etc. Typically, taking into consideration that we need to sometimes reject the time stepping, the total memory space for this computation is 10 times the original matrix. In this case the only upper bound for scaling is the Global memory size of GPU. A typical GPU would have several Gb global memory, which made the size 1024*1024 reasonable. Of course the state of art Tesla Fermi would do a little better, but not too much beyond size 1024*1024.

4 Perfomance expectation

Our expectation is that for size 1024*1024 simulation, it can be finished in relatively short time, so that we can use the code to run multiple passes and get numerical evidence for coarsening behaviour in the Fraction Cahn-Hilliard systems. In the whole simulation, we expect to take 10^5 conjugate gradient step, which 10 FFT each step. at the same time one reduction is performed and 4 vector addition is performer. In this regard, taking into account the complexity of FFT is O(NlogN), we expect FFT to be the most time consuming part. However, these numerical tasks are all well-suited for GPU architecture, therefore good for parallel code.

5 Perfomance

The performance of whole project code is hard to measure as lacking of existing benchmark. However, the FFT part of the project can be well-discussed in terms of performance. By using hierarchy FFT of our targeting size 1024*1024. We achieve a rate of $17-18$GFLOP/s on GeForce GTX 590, a factor of 3 speed up is expected by changing into Tesla. For size 256*256, a rate of 13GFLOP/s is achieved on GTX 590. In fact, our original code in Matlab is running so slow that we can not even target size of 1024*1024. Now even using GPU chip on normal desktop, code is expected to finish one 1024 *1024 simulation in less time then Matlab takes to run 256 * 256 simulation.

Size	GTX 590	Tesla
256*256	13GFLOPs	40GFLOPs
1024*1024	17GFLOPs	51GFLOPs

6 Code distribution and building

Codes are available through github
https://github.com/enjoymj/Fractional-Cahn-Hilliard-PDE
It includes various version of FFT kernel(both 1D and 2D). The optimal kernels are specifically targeted towards 1024*1024 matrix. And main function of whole simulation is also available.[Subject to frequent updates] The code can be easily complied on a GNU linux machine with OpenCL library.

References

[1]Brian Wetton et al. (2012)
High accuracy solutions to energy gradient flows from material science models
Journal of Computational Physics submitted.
[2] Naga Govindaraju et al. (2008)
High performance discrete Fourier Transforms on Graphics Processors
Supercomputing
[3] Apple OpenCL FFT
http://developer.apple.com/library/mac/samplecode/OpenCL_FFT
/Introduction/Intro.html