CUDA Parallel Programming Problem Set 9

R08244002 Shin-Rong, Tsai

- **GPU accelerated Monte Carlo simulation of 2D Ising Model on a torus**

I. Result

Slightly change the code *ising2d_1gpu_gmem_v2.cu* mentioned in course, and referred to the section Laplace 2D with N GPU, and stored it in file *ising2d_Ngpu_gmem_v2.cu.*

```
r08244002@twqcd135:~/PS9$ nvcc -arch=compute_52 -code=sm_52 -O3 -m64 --compiler-options -
fopenmp ising2d_Ngpu_gmem_v2.cu -lgsl -lgslcblas -lcurand -lcudadevrt
r08244002@twqcd135:~/PS9$ ./a.out
  Enter the number of GPUs (NGx, NGy): 1 2
1 2
  * Enter the GPU ID (0/1/...): 0
0
  * Enter the GPU ID (0/1/...): 1
1
Ising Model on 2D Square Lattice with p.b.c.
============================================
Initialize the RNG...
Enter the seed: 123
123
The RNG has been initialized.
Enter the number of sites in one dimension (<= 1000): 200
200
Enter the # of sweeps for thermalization
```

II. Discussion

A. Results in 1-GPU, 2-GPU, and CPU

<u>The settings for the tests for:</u>

$\text{lattice size} = 200 \times 200$

$\text{thermalization steps} = 10000$

$\text{measurements number} = 1024$

$\text{interval between measurements} = 10$

$\text{T} = 2.0 \text{ and } \text{B} = 0.0$

And with cold start, this is quite important!! For hot start, the result doesn't always match the exact solution, they sometimes go to the other local minimum inside the region.

And use the Binning Method (file *binning.c*) to calculate the mean value and error.

	Exact	CPU	1GPU	2GPU
$\langle E \rangle$	-1.74556	$-1.74360 \pm 2.19736 \times 10^{-3}$	$-1.74338 \pm 1.76453 \times 10^{-3}$	$-1.74322 \pm 1.91896 \times 10^{-3}$
$\langle M \rangle$	9.11319×10^{-1}	$9.10233 \times 10^{-1} \pm 1.19775 \times 10^{-3}$	$9.10190 \times 10^{-1} \pm 9.15763 \times 10^{-4}$	$9.10098 \times 10^{-1} \pm 9.41262 \times 10^{-4}$

The results are quite nice!

B. Determining the optimal block size

Since the grid size is constrained by the block size and lattice size, I only dealt with some block size.

The settings for the test are the same in section A. The time used in the code is calculate from the start of the thermalization, to the last measurement.

Time Used (ms)	Block Size (tx, ty)				
	(2,4)	(4,8)	(5,10)	(10, 20)	(20,40)
1-GPU	10244.9	10240.5	10222.8	10225.0	10246.2
2-GPU Top/Down	9133.7		9114.4	9094.5	
2-GPU Left/Right	10093.3		10065.2	10037.6	
CPU	37513.0				

We can see that using GPU is way much faster than using CPU. And using 2-GPU is generally better than just only 1-GPU. But the way we sliced the lattice matters!

We have to copy the lattice block to individual GPUs. The way we store the lattice is x-direction dominant, so the we need to copy it line by line. Split the lattice into left and right block needs more time to copy from host memory to device memory in each GPUs.

The optimal block size for 1-GPU is around (5,10). For 2-GPU Top/Down is (10, 20).

C. $\mathbf{B = 0.0}$ at $\mathbf{T = 2.0 \sim 2.5}$

Since the results from 1-GPU and 2-GPU are basically the same, I used the result from 2-GPU Left/Right only. And calculate the mean and error using Binning Method.

Energy:

	Temperature T					
	2.0	2.1	2.2	2.3	2.4	2.5
$\langle E \rangle$ Metropolis	$-1.74322 \pm 1.919 \times 10^{-3}$	$-1.66037 \pm 1.791 \times 10^{-3}$	$-1.54545 \pm 1.802 \times 10^{-3}$	$-1.34000 \pm 1.985 \times 10^{-3}$	$-1.20171 \pm 1.456 \times 10^{-3}$	$-1.10447 \pm 1.204 \times 10^{-3}$
$\langle E \rangle$ Exact	-1.74556	-1.66208	-1.54649	-1.34287	-1.20397	-1.10608

Magnetization:

	Temperature T					
	2.0	2.1	2.2	2.3	2.4	2.5
$\langle M \rangle$ Metropolis	$9.10098 \times 10^{-1} \pm 9.413 \times 10^{-4}$	$8.68070 \times 10^{-1} \pm 9.521 \times 10^{-4}$	$7.85448 \times 10^{-1} \pm 1.449 \times 10^{-3}$	$1.14330 \times 10^{-1} \pm 6.097 \times 10^{-2}$	$1.36586 \times 10^{-2} \pm 7.307 \times 10^{-3}$	$-5.33525 \times 10^{-3} \pm 2.398 \times 10^{-3}$
$\langle M \rangle$ Exact	0.911319	0.868748	0.784755	0	0	0

We can see that the results in $\langle E \rangle$ matches the exact solution, but $\langle M \rangle$ doesn't. I guess that there is a transition point between $2.2 \sim 2.3$, so that the simulation cannot resolve this step properly. (Plot by *plot.py*)