# Introduction to CUDA Parallel Programming CUDA平行計算導論

https://ceiba.ntu.edu.tw/1092Phys8061_CUDA

Professor Ting-Wai Chiu (趙挺偉)
Email: twchiu@phys.ntu.edu.tw
Physics Department
National Taiwan University

# This lecture will cover:

- How to Perform Parallel Computation of Monte Carlo Simulation of 2D Ising Model ?
- Parallel Updating with Even-Odd (Checkerboard) Scheme
- GPU Implementation using Global Memory

# Ising Model

https://en.wikipedia.org/wiki/Ising_model

Consider spins on a $D$-dimensional lattice (torus) with energy

$$E(\sigma) = -J \sum_{<i,j>} \sigma_i \sigma_j - B \sum_i \sigma_i$$

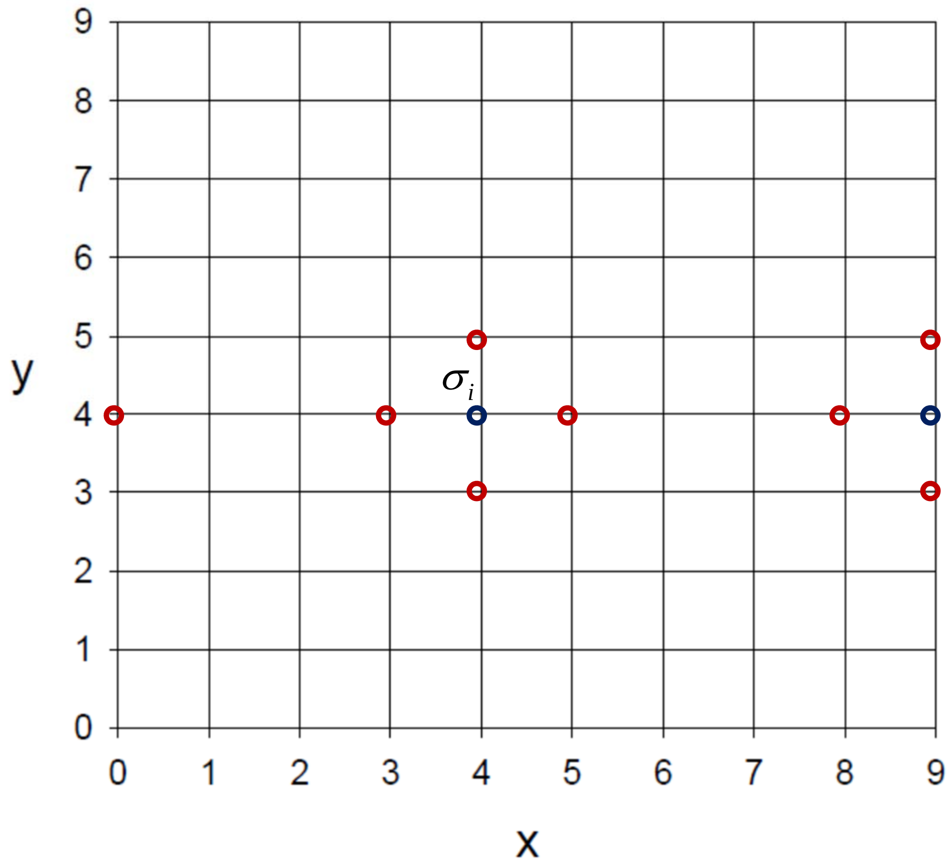
$$\sigma_i = \begin{cases} +1 & \text{spin up} \quad \uparrow \\ -1 & \text{spin down} \quad \downarrow \end{cases}$$

$J$ : coupling strength

$B$ : external magnetic field

where $\sigma_i$ denotes the spin variable at the site $i = (i_1, i_2, \cdots, i_D)$, $B$ is the external magnetic field, $J$ is the coupling strength between nearest neighboring spins, and $< i, j >$ denotes that $i$ and $j$ are nearest neighboring sites.

# Ising Model on the 2D Lattice with periodic b.c.




## 2D Lattice with periodic b.c. = Torus

# Metropolis Algorithm for MC of Ising Model

(a) The new spin $\sigma_i' = -\sigma_i$ is selected with prob. distribution satisfying $\mathrm{T}(\sigma_\mathrm{i} \to \sigma_i') = \mathrm{T}(\sigma_i' \to \sigma_\mathrm{i})$

(b) Compute the change in energy $\Delta E = E(\sigma_\mathrm{i}') - E(\sigma_\mathrm{i})$

(c) If $\Delta E \le 0$, then $\sigma_i'$ is accepted, otherwise generate a random number $r \in (0,1)$ with uniform deviate. If $r \le \exp(-\Delta E / kT)$, then $\sigma_i'$ is accepted.

$$A(\sigma_i \to \sigma_i') = \min\left( 1, \ \frac{\exp[-E(\sigma')/kT]}{\exp[-E(\sigma)/kT]} \right)$$

(d) Repeat the same procedure for another spin.

# How to Perform Parallel Updating (Simulation) with Metropolis Algorithm ?

# How to Perform Parallel Updating (Simulation) with Metropolis Algorithm ?

One way is to use Even-Odd (Checkerboard) Scheme

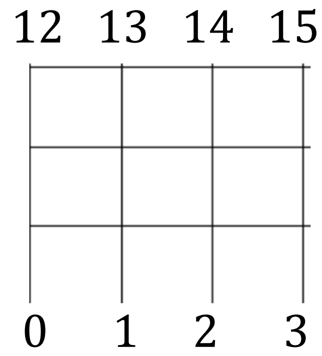
# How to Perform Parallel Updating (Simulation) with Metropolis Algorithm ?

One way is to use Even-Odd (Checkerboard) Scheme

Group the lattice sites into 2 sets, even sites and odd sites, the neighboring sites of any even site are odd sites, and vice versa. Then all even/odd sites can be updated in parallel. Thus the update can be first performed on all even sites in parallel, then go on to update all odd sites in parallel. This completes one sweep. (Note that even-odd scheme only works for models with the nearest neighbor coupling.)

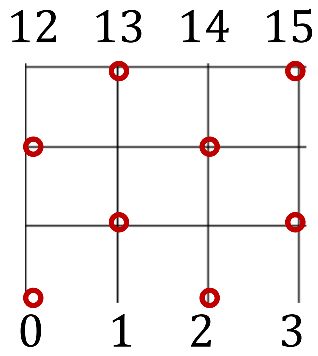
# Even-Odd (Checkerboard) Scheme

Consider the 4x4 lattice



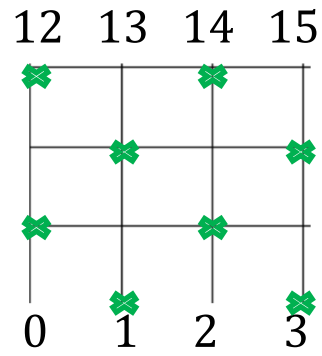
# Even-Odd (Checkerboard) Scheme

Consider the 4x4 lattice



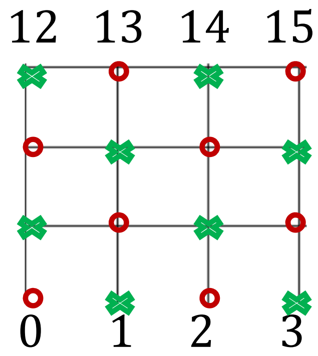
# Even-Odd (Checkerboard) Scheme

Consider the 4x4 lattice



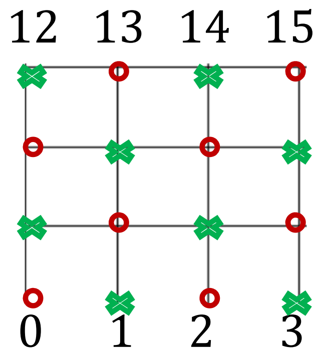
# Even-Odd (Checkerboard) Scheme

Consider the 4x4 lattice



# Even-Odd (Checkerboard) Scheme

Consider the 4x4 lattice




12 13 14 15
08 09 10 11
04 05 06 07
00 01 02 03

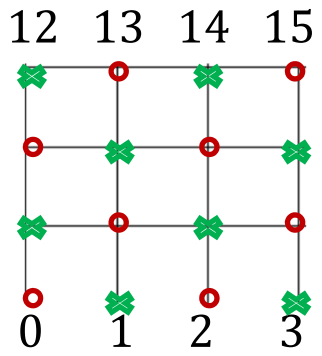
**Even sites** 08 10 13 15
00 02 05 07

**From even space to full space**

**x = (2*iE)%Nx; y = ((2*iE)/Nx )%Nx;**
**parity = (x+y)%2;**
**x = x+parity;**
**i = x+Nx*y;**

# Even-Odd (Checkerboard) Scheme

Consider the 4x4 lattice




**12 13 14 15**
**08 09 10 11**
**04 05 06 07**
**00 01 02 03**

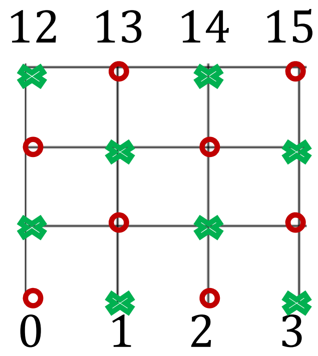
**Odd sites**

**09 11 12 14**
**01 03 04 06**

**From odd space to full space**
**x = (2*iO)%Nx; y = ((2*iO)/Nx )%Nx;**
**parity = (x+y+1)%2;**
**x = x+parity;**
**i = x+Nx*y;**

# Even-Odd (Checkerboard) Scheme

Consider the 4x4 lattice




| | | | |
|---|---|---|---|
| **12 13 14 15** | → | **Odd sites** | **09 11 12 14** |
| **08 09 10 11** | | | **01 03 04 06** |
| **04 05 06 07** | → | **Even sites** | **08 10 13 15** |
| **00 01 02 03** | | | **00 02 05 07** |

**From even space to full space**

**x = (2*iE)%Nx; y = ((2*iE)/Nx )%Nx;**
**parity = (x+y)%2;**
**x = x+parity;**
**i = x+Nx*y;**

**From odd space to full space**

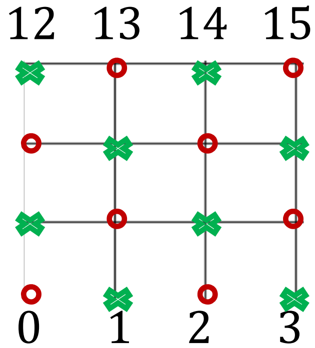
**x = (2*iO)%Nx; y = ((2*iO)/Nx )%Nx;**
**parity = (x+y+1)%2;**
**x = x+parity;**
**i = x+Nx*y;**

# Even-Odd (Checkerboard) Scheme

```c
#include <stdio.h>

void main()
{
  int Nx=4;
  int x,y,ie,i,parity;
  int sites=Nx*Nx;
  int half_sites = sites/2;

  for(ie=0; ie<half_sites; ie++) {
    x=(2*ie)%Nx;
    y=((2*ie)/Nx)%Nx;
    parity=(x+y)%2;
    x = x + parity;
    i = x + y*Nx;
    printf("%2d %2d %2d %2d\n",ie,x,y,i);
  }
  for(io=0; io<half_sites; io++) {
    x=(2*io)%Nx;
    y=((2*io)/Nx)%Nx;
    parity=(x+y+1)%2;
    x = x + parity;
    i = x + y*Nx;
    printf("%2d %2d %2d %2d\n",io,x,y,i);
  }
}
```

# Even-Odd (Checkerboard) Scheme

```
#include <stdio.h>

void main()
{
  int Nx=4;
  int x,y,ie,i,parity;
  int sites=Nx*Nx;
  int half_sites = sites/2;

  for(ie=0; ie<half_sites; ie++) {
    x=(2*ie)%Nx;
    y=((2*ie)/Nx)%Nx;
    parity=(x+y)%2;
    x = x + parity;
    i = x + y*Nx;
    printf("%2d %2d %2d %2d\n",ie,x,y,i);
  }
  for(io=0; io<half_sites; io++) {
    x=(2*io)%Nx;
    y=((2*io)/Nx)%Nx;
    parity=(x+y+1)%2;
    x = x + parity;
    i = x + y*Nx;
    printf("%2d %2d %2d %2d\n",io,x,y,i);
  }
} // The complete code at twqcd80:/home/cuda_lecture_2021/Ising2D/index_eo.c
```




| ie | x | y | i |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 2 | 0 | 2 |
| 2 | 1 | 1 | 5 |
| 3 | 3 | 1 | 7 |
| 4 | 0 | 2 | 8 |
| 5 | 2 | 2 | 10 |
| 6 | 1 | 3 | 13 |
| 7 | 3 | 3 | 15 |

| io | x | y | i |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 1 | 3 | 0 | 3 |
| 2 | 0 | 1 | 4 |
| 3 | 2 | 1 | 6 |
| 4 | 1 | 2 | 9 |
| 5 | 3 | 2 | 11 |
| 6 | 0 | 3 | 12 |
| 7 | 2 | 3 | 14 |

# Parallel Updating (Simulation) with Even-Odd (Checkerboard) Scheme

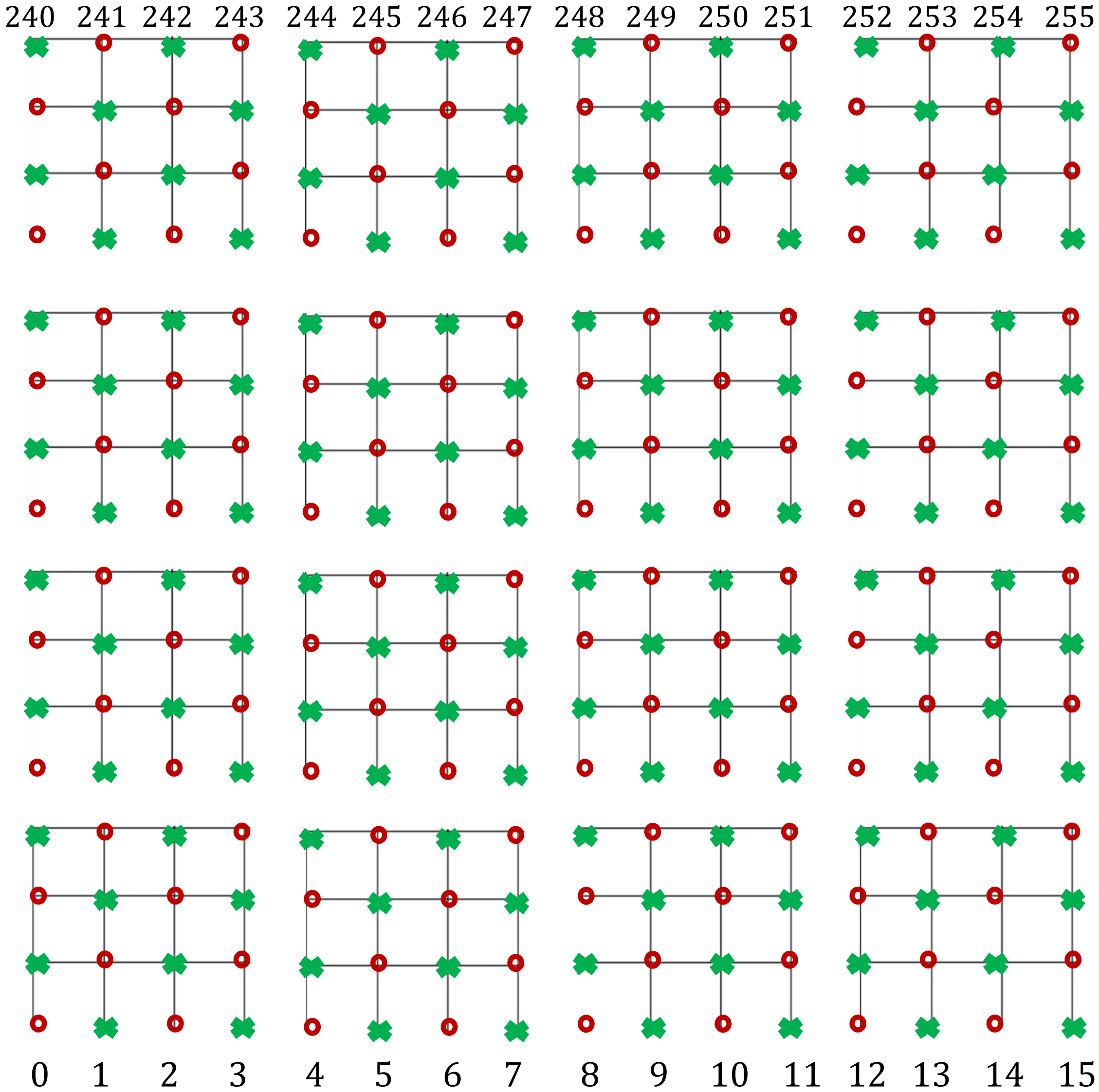
```
 for(int ie=0; ie<ns/2; ie++) {       // update even sites
   x = (2*ie)%nx;
   y = ((2*ie)/nx)%nx;
   parity=(x+y)%2;
   x = x + parity;
   i = x + y*nx;
   old_spin = spin[i];
   new_spin = -old_spin;
   k1 = fw[x] + y*nx;     // right
   k2 = x + fw[y]*nx;     // top
   k3 = bw[x] + y*nx;     // left
   k4 = x + bw[y]*nx;     // bottom
   spins = spin[k1] + spin[k2] + spin[k3] + spin[k4];
   de = -(new_spin - old_spin)*(spins + B);
   if((de <= 0.0) || (gsl_rng_uniform(rng) < exp(-de/T))) {
// if (new energy <= old energy) or (r < exp(-dE/kT))  accept the new spin
     spin[i] = new_spin;         // accept the new spin;
   }
 }
```

# Parallel Updating (Simulation) with Even-Odd (Checkerboard) Scheme

```
 for(int io=0; io<ns/2; ie++) {        // update odd sites
   x = (2*io)%nx;
   y = ((2*io)/nx)%nx;
   parity=(x+y+1)%2;
   x = x + parity;
   i = x + y*nx;
   old_spin = spin[i];
   new_spin = -old_spin;
   k1 = fw[x] + y*nx;     // right
   k2 = x + fw[y]*nx;     // top
   k3 = bw[x] + y*nx;     // left
   k4 = x + bw[y]*nx;     // bottom
   spins = spin[k1] + spin[k2] + spin[k3] + spin[k4];
   de = -(new_spin - old_spin)*(spins + B);
   if((de <= 0.0) || (gsl_rng_uniform(rng) < exp(-de/T))) {
// if (new energy <= old energy) or (r < exp(-dE/kT))  accept the new spin
     spin[i] = new_spin;         // accept the new spin;
   }
 } //Complete code at twqcd80:/home/cuda_lecture_2021/Ising2D/ising2d_cpu_eo.c
```

# GPU Implementation with Global Memory

Lattice size: 16 x 16
gridDim = (4, 4)
blockDim = (2, 4)



# GPU Implementation with Global Memory

Some considerations:

- GPU kernel can perform several sweeps at each call
- To use CPU or GPU to generate RNG ?
- To put the entire simulation in the GPU ?
- Instead of computing $\exp(-\Delta E/kT)$, its value can be looked up from a table which is prepared before the start of the simulation, since the number of possible values of $\exp(-\Delta E/kT)$ are limited, due to the discreteness of the spin.