Introduction to CUDA Parallel Programming
CUDA平行計算導論

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

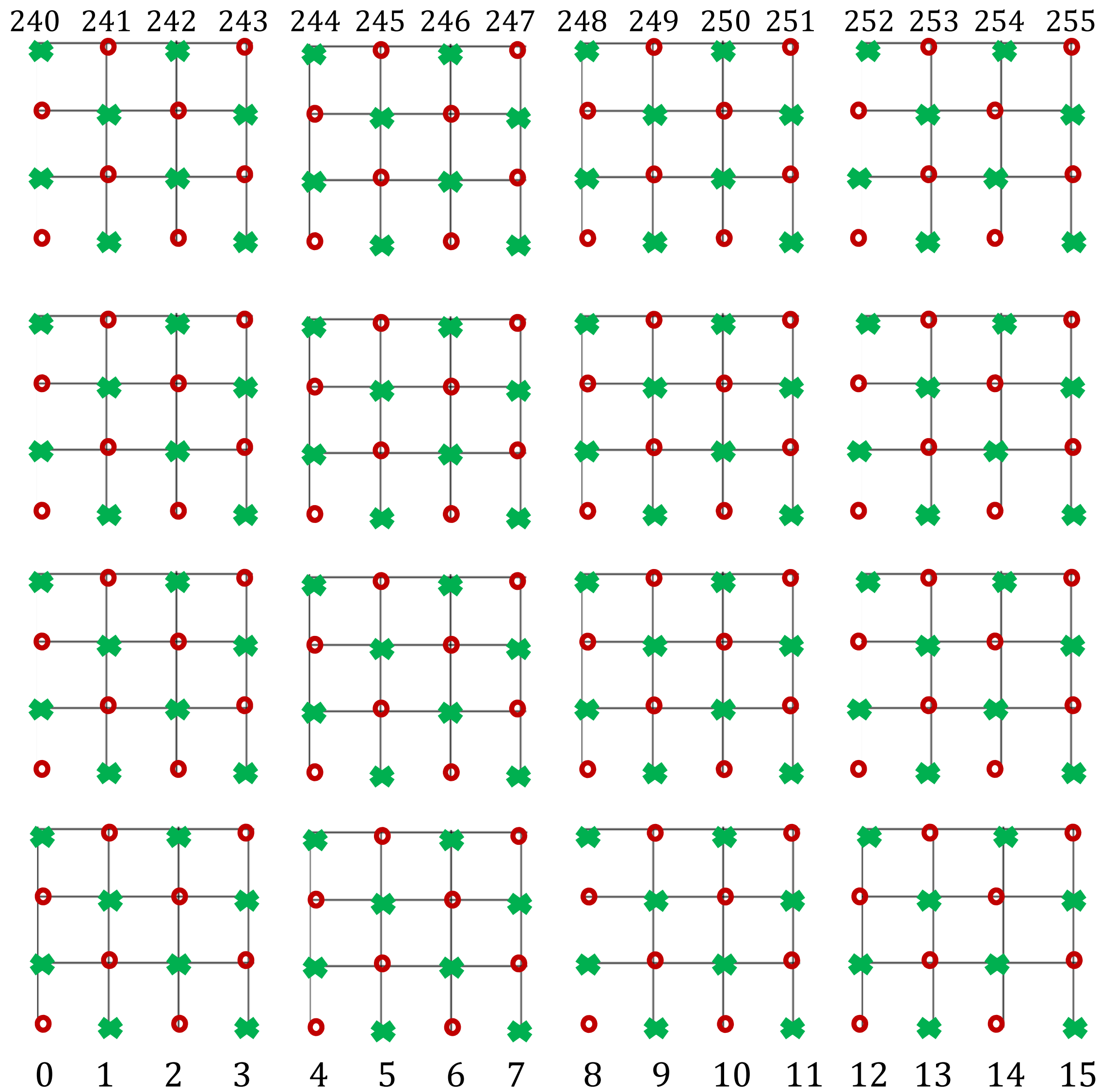
Professor Ting-Wai Chiu (趙挺偉)
Email: twchiu@phys.ntu.edu.tw
Physics Department
National Taiwan University

This lecture will cover:

- GPU-Accelerated MC Simulation of 2D Ising Model with Global Memory
- Constant Memory
- Error Estimation in Monte Carlo Simulation

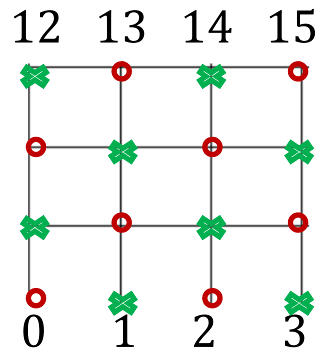
GPU Accelerated Simulation with Global Memory

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



Mapping between the threadId and the site index

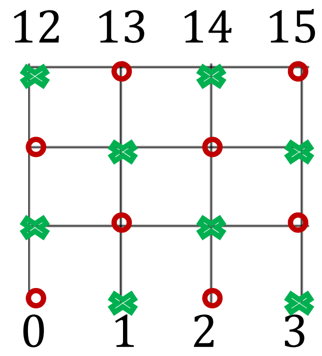
```
// thread index in a block of size (tx,ty)
// corresponds to the index ie/io of the
// lattice with size (2*tx,ty)=(Nx,Ny).
// ie/io = threadIdx.x + threadIdx.y*blockDim.x

    int Nx = 2*blockDim.x;
    int nx = 2*blockDim.x*gridDim.x;

    // first, go over the even sites
    ie = threadIdx.x + threadIdx.y*blockDim.x;
    x = (2*ie)%Nx;
    y = ((2*ie)/Nx)%Nx;
    parity=(x+y)%2;
    x = x + parity;

   // add the offsets to get its position in the full lattice

    x += Nx*blockIdx.x;
    y += blockDim.y*blockIdx.y;
    i = x + y*nx;

   // parallel updating the spins at all even sites
```



blockDim = (2, 4)

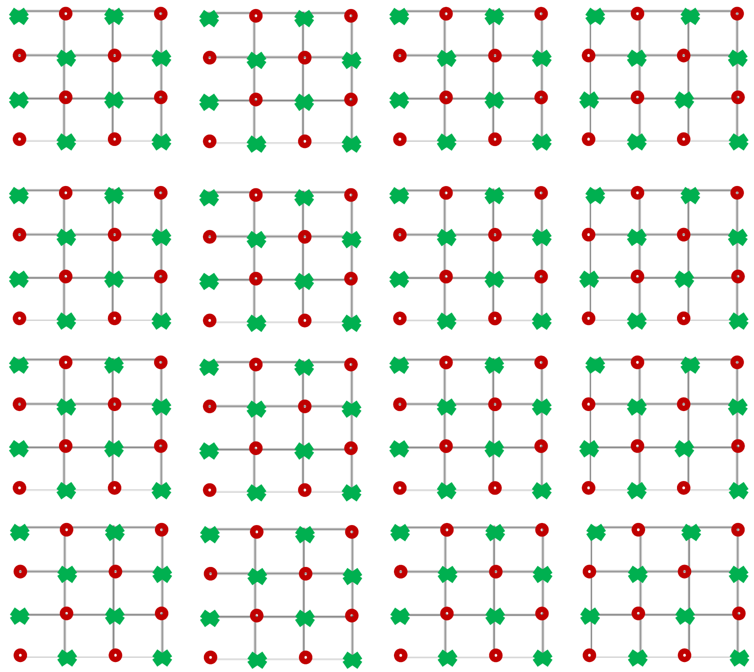
Mapping between the threadId and the site index

```
// thread index in a block of size (tx,ty)
// corresponds to the index ie/io of the
// lattice with size (2*tx,ty)=(Nx,Ny).
// ie/io = threadIdx.x + threadIdx.y*blockDim.x

    int Nx = 2*blockDim.x;
    int nx = 2*blockDim.x*gridDim.x;

    // next, go over the odd sites
    ie = threadIdx.x + threadIdx.y*blockDim.x;
    x = (2*ie)%Nx;
    y = ((2*ie)/Nx)%Nx;
    parity=(x+y+1)%2;
    x = x + parity;

   // add the offsets to get its position in the full lattice

    x += Nx*blockIdx.x;
    y += blockDim.y*blockIdx.y;
    i = x + y*nx;

   // parallel updating the spins at all odd sites
```



blockDim = (2, 4)

Parallel Updating the Spins on All Even/Odd Sites



```
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) || (ranf[i] < exp(-de/T))) {
  spin[i] = new_spin;   // accept the new spin;
 }
```

// See twqcd80:/home/cuda_lecture_2021/Ising2D/ising2d_1gpu_gmen_v1.cu

The Metropolis Kernel (v1)

```
__global__ void metro_gmem(int* spin, float *ranf, const float B, const float T)
{
    ...
    // first, parallel updating spins on all even sites

    ie = threadIdx.x + threadIdx.y*blockDim.x;
    x = (2*ie)%Nx;
    y = ((2*ie)/Nx)%Nx;
    parity=(x+y)%2;
    x = x + parity;
    x += Nx*blockIdx.x;
    y += blockDim.y*blockIdx.y;
    old_spin = spin[i];
    new_spin = -old_spin;
    k1 = fw[x] + y*nx;
    k2 = x + fw[y]*nx;
    k3 = bw[x] + y*nx;
    k4 = x + bw[y]*nx;
    spins = spin[k1] + spin[k2] + spin[k3] + spin[k4];
    de = -(new_spin - old_spin)*(spins + B);
    if((de <= 0.0) || (ranf[i] < exp(-de/T))) spin[i] = new_spin;
    __syncthreads();  // See the discussions in the next page

    // next, parallel updating spins on all odd sites
    ...
} // See twqcd80:/home/cuda_lecture_2021/Ising2D/ising2d_1gpu_gmem_v1.cu
```

The Metropolis Kernel (v1)

Note that syncthreads() only applies to all threads in a block, but NOT all threads in the entire grid. In other words, not all blocks are synchronized. Thus, even if two simulations start with the same RNG seed, they do not necessarily get exactly the same result. One way to avoid this problem is to update even sites and odd sites with two different kernels, as in v2. Another way is to use the method of cooperative groups, which is only available for GPUs with compute capability > sm_60 and compile with CUDA 9.1, as shown in v3.
For details, see CUDA Programming Guide, Appendix C, Cooperative Group
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#grid-synchronization-cg

The Metropolis Kernel (v2)

```
__global__
void metro_gmem_even(int* spin, float *ranf, const float B, const float T)
{
    ...
    // parallel updating spins on all even sites
}

__global__
void metro_gmem_odd(int* spin, float *ranf, const float B, const float T)
{
    ...
    // parallel updating spins on all odd sites
}

int main(void)
{
   ...
   for(int swp=0; swp<nt; swp++) {     // thermalization
     rng_MT(h_rng, ns);
     cudaMemcpy(d_rng, h_rng, ns*sizeof(float), cudaMemcpyHostToDevice);
     metro_gmem_even<<<blocks,threads>>>(d_spin, d_rng, B, T); // update even sites
     metro_gmem_odd<<<blocks,threads>>>(d_spin, d_rng, B, T);  // update odd sites
   }
   ...
} // See twqcd80:/home/cuda_lecture_2021/Ising2D/ising2d_1gpu_gmem_v2.cu
```

The Metropolis Kernel (v3)

```
#include <cooperative_groups.h>
...
__global__ void metro_gmem(int* spin, float *ranf, const float B, const float T)
{   ...
    gsizeX = gridDim.x*blockDim.x;
    gsizeY = gridDim.y*blockDim.y;
    namespace cg = cooperative_groups;
    cg::grid_group g = cg::this_grid();
    ith = g.thread_rank();                  // thread index of the thread group
    x0  = (2*ith) % (2*gsizeX);
    y0  = (ith / gsizeX) % gsizeY;

    for(y1=y0; y1 < nx; y1 += gsizeY) {
    for(x1=x0; x1 < nx; x1 += (gsizeX*2)) {
       // first, updating spins on all even sites
    }}
    cg::sync(g);  // synchronize all threads in the grid

    for(y1=y0; y1 < nx; y1 += gsizeY) {
    for(x1=x0; x1 < nx; x1 += (gsizeX*2)) {
      // next, updating spins on all odd sites
    }}
}

// See twqcd80:/home/cuda_lecture/Ising2D/ising2d_1gpu_gmem_v3.cu
```

The Metropolis Kernel (v3)

Remarks

- A code segment is added to check whether the grid size is valid for the cooperative groups operation. For lattice size 256 x 256, (bx, by) = (4, 8) is OK for v2 code, but its resulting grid size is too large for cooperative groups operation. Hence the block size needs to be enlarged.

- Since there are limits on the block size and the grid size for cooperative groups operation, a very large lattice may need more than just one grid. Hence, in the kernel, for loops are introduced.

Constant Memory

```
Device 0: "GeForce GTX 1060 6GB"
  CUDA Driver Version / Runtime Version          10.2 / 10.2
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 6078 MBytes (6373179392 bytes)
  (10) Multiprocessors, (128) CUDA Cores/MP:     1280 CUDA Cores
  GPU Max Clock rate:                            1759 MHz (1.76 GHz)
  Memory Clock rate:                             4004 Mhz
  Memory Bus Width:                              192-bit
  L2 Cache Size:                                 1572864 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536),
                                                 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  ...
  ...
```

Constant Memory in the Ising Model

```
__constant__ int fw[1000],bw[1000];  // declare constant memory

__global__ void metro_gmem(int* spin, float *ranf, float B, float T)
{    ...
     k1 = fw[x] + y*nx;      // right
     k2 = x + fw[y]*nx;      // top
     k3 = bw[x] + y*nx;      // left
     k4 = x + bw[y]*nx;      // bottom
     ...
}

int main(void)
{  ...
   ffw = (int*)malloc(nx*sizeof(int));
   bbw = (int*)malloc(nx*sizeof(int));
   for(int i=0; i<nx; i++) {
     ffw[i]=(i+1)%nx;
     bbw[i]=(i-1+nx)%nx;
   }
   cudaMemcpyToSymbol(fw, ffw, nx*sizeof(int));//copy data to const. mem.
   cudaMemcpyToSymbol(bw, bbw, nx*sizeof(int));
   ...
}
```

Error Estimation in Monte Carlo Simulation

In Monte Carlo simulation of the Ising model, the expectation value of any observable can be measured by averaging over the configurations.

$$\langle O \rangle = \frac{1}{Z} \sum_{\{C\}} O(C)\, e^{-E(C)/kT} \approx \frac{1}{N} \sum_{k=1}^{N} O(C_k) \pm E(\langle O \rangle)$$

where $E(\langle O \rangle)$ is the error of the mean.

According to the central limit theorem, the distribution of $\langle O \rangle$ obeys the Gaussian distribution in the limit $N \to \infty$, with the variance

$$\sigma_{\langle O \rangle}^2 = \frac{1}{N} \left[\frac{1}{N} \sum_{k=1}^{N} O_k^2 - \left(\frac{1}{N} \sum_{k=1}^{N} O_k \right)^2 \right] = \frac{1}{N} \sigma_O^2$$

If all C_k are statistically independent, then $E(\langle O \rangle) = \frac{1}{\sqrt{N}} \sigma_O$

Error Estimation in Monte Carlo Simulation (cont)

However, in the Monte Carlo simulation, $\{C_k\}$ are not statistically independent, the error of the mean is corrected as follows.

$$E(\langle O \rangle) = \frac{1}{\sqrt{N}} \sigma_O [1 + 2C(1) + 2C(2) + \cdots]$$

$$C(m) = \frac{\langle O_i O_{i+m} \rangle - \langle O_i \rangle^2}{\langle O_i^2 \rangle - \langle O_i \rangle^2} \quad \text{autocorrelation function}$$

In practice, $\langle O \rangle$ is calculated using configurations separated by a fixed interval in the simulation sequence. An appropriate sampling interval can be estimated from the value of m for which $C(m)$ becomes small.

Error Estimation in Monte Carlo Simulation (cont)

In Monte Carlo simulation, the samples from the Markov chain are not statistically independent by are rather correlated. Suppose we perform N successive measurements of the observable $A_i,\ i=1,\cdots,N,\ N \gg 1$. Then the error of the mean can be estimated by taking the square root of

$$\left\langle (\delta A)^2 \right\rangle = \left\langle \left[\frac{1}{N} \sum_{i=1}^{N} \left(A_i - \langle A \rangle \right) \right]^2 \right\rangle = \left\langle \left(\frac{1}{N} \sum_{i=1}^{N} A_i - \langle A \rangle \right)^2 \right\rangle$$

$$\approx \frac{1}{N^2} \left\langle \sum_{i=1}^{N} \left(A_i - \langle A \rangle \right)^2 \right\rangle + \frac{2}{N^2} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \left(\langle A_i A_j \rangle - \langle A \rangle^2 \right)$$

(see the derivation in the next page)

Error Estimation in Monte Carlo Simulation (cont)

$$\left(\frac{1}{N}\sum_{i=1}^{N} A_i - \langle A\rangle\right)^2 = \frac{1}{N^2}\left(\sum_i A_i\right)\left(\sum_j A_j\right) - \frac{2}{N}\left(\sum_i A_i\right)\langle A\rangle + \langle A\rangle^2$$

$$= \frac{1}{N^2}\sum_i A_i^2 + \frac{2}{N^2}\sum_i \sum_{j=i+1} A_i A_j - \frac{2}{N}\left(\sum_i A_i\right)\langle A\rangle + \langle A\rangle^2$$

$$= \frac{1}{N^2}\sum_{i=1}^{N}\left(A_i - \langle A\rangle\right)^2 + \frac{2}{N^2}\left(\sum_i A_i\right)\langle A\rangle - \frac{\langle A\rangle^2}{N} + \frac{2}{N^2}\sum_i \sum_{j=i+1} A_i A_j - \frac{2}{N}\left(\sum_i A_i\right)\langle A\rangle + \langle A\rangle^2$$

$$\approx \frac{1}{N^2}\sum_{i=1}^{N}\left(A_i - \langle A\rangle\right)^2 + \frac{2}{N^2}\sum_i \sum_{j=i+1}\left(A_i A_j - \langle A\rangle^2\right)$$

since

$$+\frac{2}{N^2}\left(\sum_i A_i\right)\langle A\rangle \approx +\frac{2}{N}\langle A\rangle^2,$$

$$-\frac{2}{N}\left(\sum_i A_i\right)\langle A\rangle \approx -2\langle A\rangle^2,$$

$$+\frac{2}{N} - \frac{1}{N} - 2 + 1 = \frac{1}{N} - 1 = -\frac{2}{N^2}\frac{N(N-1)}{2}$$

Error Estimation in Monte Carlo Simulation (cont)

$$\approx \frac{1}{N}\left\langle \left(A-\langle A\rangle\right)^2 \right\rangle + \frac{2}{N}\left(\langle A_i\, A_{i+1}\rangle - \langle A\rangle^2\right) + \frac{2}{N}\left(\langle A_i\, A_{i+2}\rangle - \langle A\rangle^2\right) + \cdots$$

$$= \frac{\left\langle \left(A-\langle A\rangle\right)^2 \right\rangle}{N}\left[1 + 2\frac{\langle A_i\, A_{i+1}\rangle - \langle A\rangle^2}{\langle A_i{}^2\rangle - \langle A\rangle^2} + 2\frac{\langle A_i\, A_{i+2}\rangle - \langle A\rangle^2}{\langle A_i{}^2\rangle - \langle A\rangle^2} \right.$$

$$\left. + \cdots + 2\frac{\langle A_i\, A_{i+N-1}\rangle - \langle A\rangle^2}{\langle A_i{}^2\rangle - \langle A\rangle^2}\right]$$

where

$$\frac{\left\langle \left(A-\langle A\rangle\right)^2 \right\rangle}{N} = \frac{\langle A^2\rangle - \langle A\rangle^2}{N}$$

Error Estimation in Monte Carlo Simulation (cont)

$$\left\langle (\delta A)^2 \right\rangle \simeq \frac{\left\langle A^2 \right\rangle - \left\langle A \right\rangle^2}{N-1} \left[1 + 2 \sum_{j=1}^{N-1} C(j) \right]$$

$$C(j) \equiv \frac{\left\langle A_i A_{i+j} \right\rangle - \left\langle A \right\rangle^2}{\left\langle A^2 \right\rangle - \left\langle A \right\rangle^2}, \text{ or } \frac{\left\langle A_i A_{i+j} \right\rangle - \left\langle A_i \right\rangle \left\langle A_{i+j} \right\rangle}{\left\langle A_i^2 \right\rangle - \left\langle A_i \right\rangle^2}$$

see the sample code autoT_A.c

see the sample code autoT_B.c

Let $t = j\tau, \quad t_{N-1} = (N-1)\tau$

$$\sum_{j=1}^{N-1} C(j) \to \frac{1}{\tau} \int_0^{t_{N-1}} dt \, C(t)$$

autoT_A.c

```
...
for(t=0;t<tmax1;t++) {   // from t=0 to tmax
  A_cor[t]=0.0;
  count = 0;
  for(i=1; i<N-t+1; i++) {
    A_cor[t] += A[i]*A[i+t];
    count += 1;
  }
  A_cor[t] = A_cor[t]/count - A_ave*A_ave;
}
...
```

autoT_B.c

```
...
for(t=0;t<tmax1;t++) {
   A_cor[t]=0.0;
   count = 0;
   Ai_ave = 0.0; Aj_ave = 0.0;
   for(i=1; i<N-t+1; i++) {
     Ai_ave += A[i];
     Aj_ave += A[i+t];
     A_cor[t] += A[i]*A[i+t];
     count += 1;
   }
   Ai_ave /= count;
   Aj_ave /= count;
   A_cor[t] = A_cor[t]/count - Ai_ave*Aj_ave;
}
...
```

Error Estimation in Monte Carlo Simulation (cont)

$$\left\langle (\delta A)^2 \right\rangle \simeq \frac{\left\langle A^2 \right\rangle - \left\langle A \right\rangle^2}{N-1} \left[1 + \left(\frac{2}{\tau} \right) \int_0^\infty dt\, C(t) \right]$$

$$C(0) = 1, \quad C(\infty) = 0$$

Assume $C(t) \sim \exp(-t/\tau_A)$, $\int_0^\infty dt\, C(t) = \tau_A$ integrated correlation time

$$\left\langle (\delta A)^2 \right\rangle \simeq \frac{\left\langle A^2 \right\rangle - \left\langle A \right\rangle^2}{N-1} \left(1 + \frac{2\tau_A}{\tau} \right) = \frac{\left\langle A^2 \right\rangle - \left\langle A \right\rangle^2}{N-1} 2\tau_{\text{int}},$$

$$\tau_{\text{int}} \equiv \frac{1}{2} + \frac{\tau_A}{\tau} = \frac{1}{2} + \sum_{j=1} C(j) \Big|_{C(j) > 0}$$

$$\left\langle A \right\rangle \approx \frac{1}{N} \sum_{i=1}^{N} A_i \pm \sqrt{\left\langle (\delta A)^2 \right\rangle} = \frac{1}{N} \sum_{i=1}^{N} A_i \pm \sqrt{\frac{\left\langle A^2 \right\rangle - \left\langle A \right\rangle^2}{N-1}} \sqrt{2\tau_{\text{int}}}$$

Binning Method

In practice, $C(t)$ is notoriously difficult to measure.

A way to obtain a proper error estimate for MC is by **blocking** the data. We average n_b successive measurements of A_i to form a block average B_i. If the blocks are large enough, the autocorrelation of successive blocks will be small.

For large $n_b \gg 1$,

$$C(\tau) = \frac{\langle B_i B_{i+\tau} \rangle - \langle B_i \rangle^2}{\langle B_i^{\,2} \rangle - \langle B_i \rangle^2} \simeq \frac{1}{n_b} \ll 1$$

Thus the integrated autocorrelation time can be estimated by increasing n_b until $C(\tau) \ll 1$, provided that the number of blocks $m = N / n_b$ is sufficently large throughout this process.

Binning Method (cont)

Thus we can measure the variance of the block averages

$$(\delta A)^2 \simeq \frac{\langle B_i{}^2 \rangle - \langle B_i \rangle^2}{m-1}$$

for several values of n_b, and extrapolate to get the error estimate, or to increase n_b until δA saturates (~ a plateau). Then the saturated δA is taken as the error of the mean.