# Simulations in Statistical Physics

## Course for MSc physics students

Janos Török

Department of Theoretical Physics

October 20, 2014

# Parallelization

- Why?
  - The speed of one core processor is limited
  - Larger system sizes
  - Multi-core processors
  - On multi-core system inter-processor data change is fast
- Why not?
  - Computing power is lost
  - Much more code development
  - Very often ensemble average is needed
  - Inter-computer communication is terribly slow

RAM $\rightarrow$ $\sim$15GB/s, Ethernet 125MB/s, Infiniband $\sim$1GB/s

# Simple parallelization

- Multi-threading:
  - Code if copied to multiple processors
  - Memory is shared → no need to copy data between processors
  - Using semaphores to pretect data overwrite
  - Easy to do but unusable on clusters
- E.g. BOOST:
  - Simple parallelization of loops
  - No history dependence
  - Example:

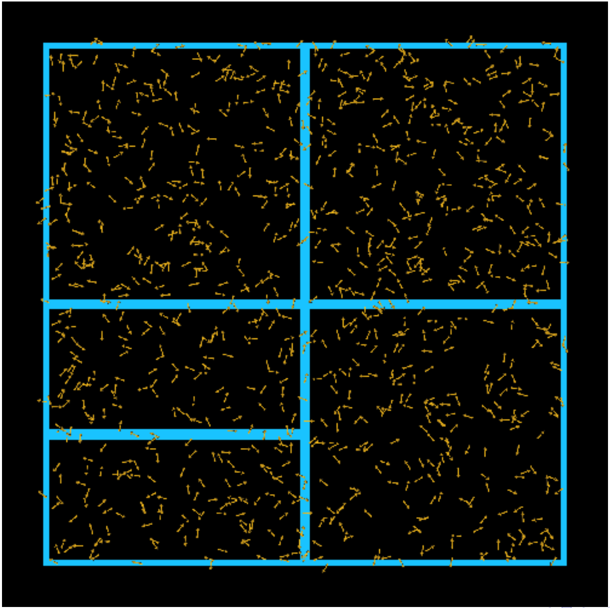
```
#include <string>
#include <iostream>
#include <boost/foreach.hpp>

int main()
{
    std::string hello( "Hello, world!" );

    BOOST_FOREACH( char ch, hello )
    {
        std::cout << ch;
    }

    return 0;
}
```

# Message passing interface MPI

- A given number of copies of the code across processors and machines
- All processors know their id and the total number of processors
- Point-to-point communication: synchron and acynchron
- Gathering data
- Master-slaves, or real parallel, sharing only parts of the system

```
sprintf(buff, "Hello %d! ", i);
MPI_Send(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD);

MPI_Recv(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD, &stat);

MPI_Allreduce(&locf,&sumf,6,MPI_DOUBLE,MPI_SUM,world);
```

# Parallelization (Bird flocking model)

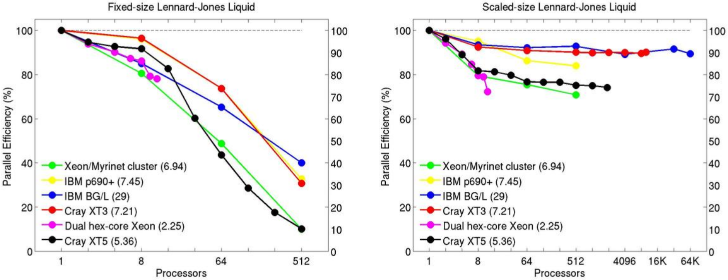


# Parallelization

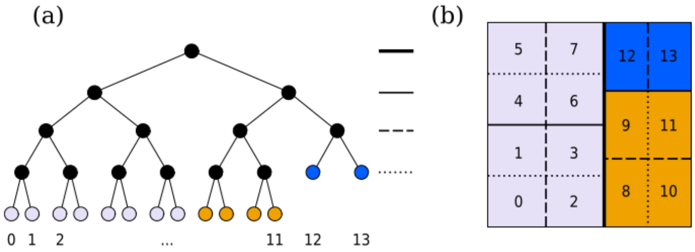
- Molecular dynamics
  - Short range interactions: Box must be duplicated, Verlet in parallel
  - Long range: Parallel fast Fourier transformation
- Contact dynamics
  - Short range interactions: Box must be duplicated
  - Iteration in parallel
- Event Driven Dynamics
  - List must be global, no way!
- Kinetic Monte Carlo
  - List must be global, no way!

# Efficiency of parallelization

- Large systems are needed
- Boundary must be minimal



# Efficiency of parallelization



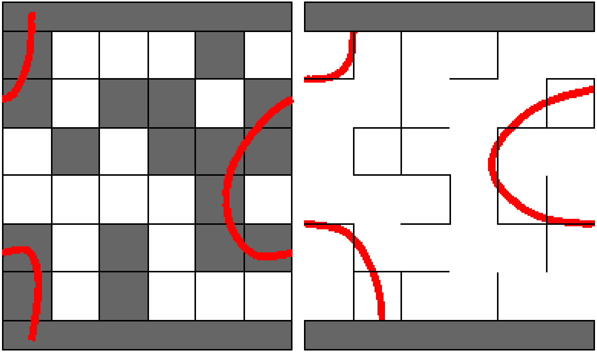

- Calculate time spent in a branch
- Calculate $\sigma_T = \sqrt{\langle T^2 \rangle - \langle T \rangle^2} / \langle T \rangle$
- Move line if necessary ($\sigma_T > \sigma_T^*$)
- Lower in tree (up in Fig), larger the mass of the border
- Only rarely, data transfer is expensive

# Percolation

# Percolation

Behavior of connected cluster

- Site percolation
- Bond percolation



# Percolation theory

Questions (in infinite systems):

1. Is there an infinite cluster in infinite systems?
2. How many infinite clusters are there?
3. Mean cluster size (without the inifinte one)?
4. Cluster size distribution

Answers:

1. Above a critical density with probability 1 below it with probability 0
2. Only 1!
3. Decreases as a power low avay from the critical density
4. Power law

# Percolation theory

Questions (in infinite systems):

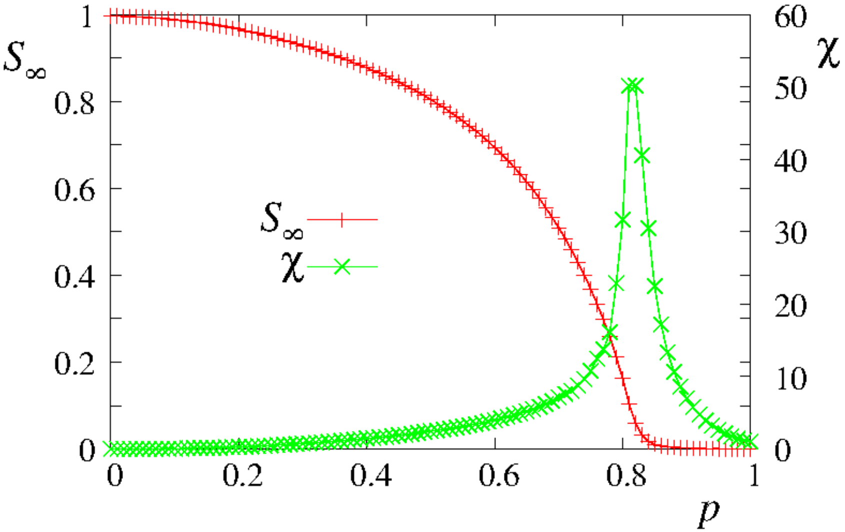
1. Is there an infinite cluster in infinite systems?
2. How many infinite clusters are there?
3. Cluster size distribution ($n_s$)
4. Mean cluster size (without the inifinte one)? ($S = \sum_s s^2 n_s$)

Answers:

1. if $p > p_c$ then yes, otherwise no
2. Only 1!
3. $n_s \sim s^{-\tau}$
4. $S \sim |p - p_c|^{-\gamma}$

Like a second order phase transition in a geometric system!

# Percolation model



# Percolation theory: Importance

- COFFEE!!!!
- Non-equilibrium statistical physics
- Image analysis
- Percolation on networks: Phase transitions
- Percolation on networks: robustness, fragility
- Flodings

# Percolation model

Bond [site] percolation

- Let us have a lattice (network)
- Each bond [site] is occupied with probability $p$
- (unoccupied with probability $1 - p$)
- A cluster is a set of sites connected by occupied bonds
  [A cluster is a set of occupied sites]

# Hoshen-Kopelman Algorithm

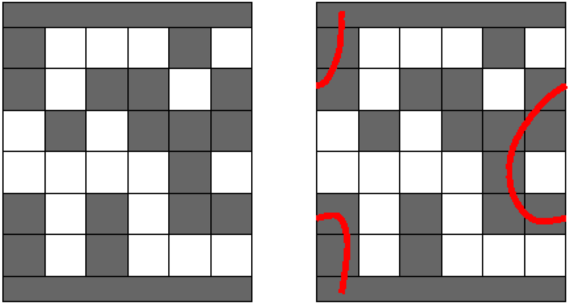
- Numerical task: find clusters
- Identify clusters
- Visit all sites
- Mark them with numbers

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

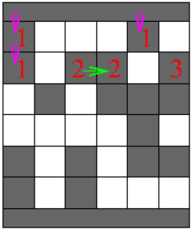
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 2 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 0 |
| 0 | 0 | 2 | 0 | 0 | 4 | 0 | 0 | 3 | 3 | 3 | 0 | 0 | 0 | 5 |
| 0 | 2 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 3 | 0 | 5 | 5 | 5 | 5 |
| 6 | 0 | 2 | 2 | 0 | 7 | 0 | 0 | 0 | 0 | 5 | 5 | 0 | 0 | 5 |
| 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 8 | 0 | 5 |
| 9 | 0 | 2 | 0 | 10 | 0 | 0 | 0 | 5 | 0 | 0 | 5 | 0 | 5 | 5 |
| 9 | 9 | 0 | 0 | 10 | 0 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 0 |
| 9 | 0 | 0 | 10 | 10 | 10 | 0 | 0 | 5 | 5 | 5 | 0 | 0 | 5 | 5 |
| 9 | 9 | 0 | 10 | 10 | 10 | 0 | 11 | 0 | 5 | 0 | 0 | 12 | 0 | 0 |
| 0 | 0 | 13 | 0 | 0 | 10 | 10 | 0 | 5 | 5 | 0 | 12 | 12 | 0 | 0 |

# Hoshen-Kopelman Algorithm

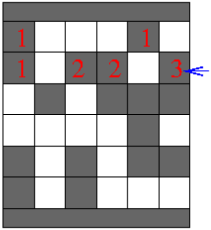
- Site percolation
- Helical boundary conditions
- Go through site in typewriter style
- Check left and above



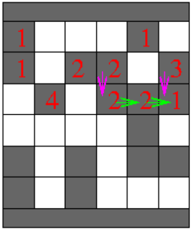
# Hoshen-Kopelman Algorithm, Helical BC



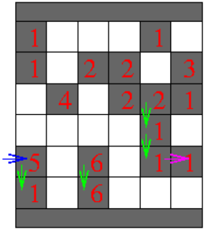
link[1]=1
link[2]=2
link[3]=3



link[1]=1
link[2]=2
link[3]=1



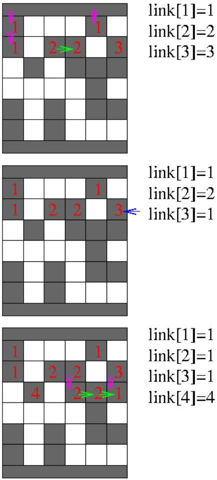
link[1]=1
link[2]=1
link[3]=1
link[4]=4



link[1]=1
link[2]=1
link[3]=1
link[4]=4
link[5]=1
link[6]=6

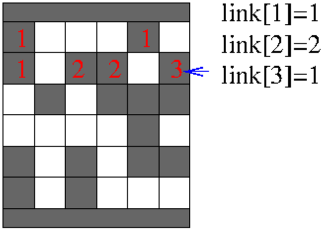
# Hoshen-Kopelman Algorithm

```
largest_label = 0;
for (y = 0;y < n_rows;y++) {
  for (x = 0;x < n_columns;x++) {
    if (occupied[x][y]) {
      left = occupied[x-1][y];
      above = occupied[x][y-1];
      if (left == 0) && (above == 0) {
        largest_label ++;
        label[x][y] = largest_label;

      } else if (left != 0) && (above == 0) {
        label[x,y] = find(left);

      } else if (left == 0) && (above != 0) {
        label[x,y] = find(above);

      } else {
        label[x,y] = union(left,above);
      }
    }
  }
  /* Helical boundary conditions */
  if (occupied[n_columns-1][y]) && (occupied[0][y]) {
    union(occupied[n_columns-1][y],occupied[0][y])
  }
}
```



# Hoshen-Kopelman Algorithm

```
largest_label = 0;
for x in 0 to n_columns {
 for y in 0 to n_rows {
   if occupied[x,y] then
     left = occupied[x-1,y];
     above = occupied[x,y-1];
     if (left == 0) and (above == 0) then
       largest_label = largest_label + 1;
       label[x,y] = largest_label;
     else {
       if (left != 0) {
         if (right != 0)
           UNION(left,above);
         label[x,y] = FIND(above);
       } else
       label[x,y] = FIND(right);
     }
   }
 }
```




```
int link[N];

int find(int x) {
 while (link[x] != x)
   x = link[x];
 return x;
}
```

```
int union(int x, int y) {
  int fx = find(x);
  int fy = find(y);
  if (fx < fy) {
    link[fy] = fx;
    return (fx);
  } else {
    link[fx] = fy;
    return (fy);
  }
}
```

# Hoshen-Kopelman Algorithm

- Go through lattice as typewriter
- Check neighbors
- Resolve conflicts by linking clusters together
- Original trick: use link[] array for cluster size measure
  - link[] positive: number of sites in the cluster
  - link[] negative: cluster is linked to on other cluster
  - Not necessary faster than a seperate arrey for size

# Percolation on networks (graphs)

- Network is defined by nodes and links
- Two arrays:
  - node[] list of nodes
  - link[i][] list of links of node i
  - link[i][j] is a link between i and j
- Cluster: nodes connected with links
- Links can be directed link[i][j] is a link from i $\rightarrow$ j

# Stack (Verem – Hole/Pitfall)

- Last in forst out (LIFO)
- Code:

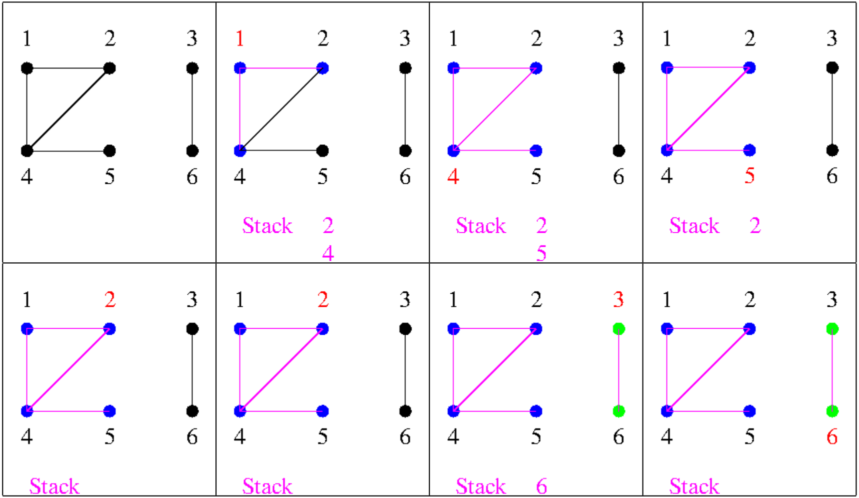
```
int Stack_size = Hopefully_large_enough_number;
int stack[Stack_size];
int sp=0;

void push(int item) {
  stack[sp++] = item;
  if (sp == Stack_size) enlarge_array(stack);
}

int pop() {
  return(stack[--sp]);
}
```

- Error handling?
- Size of the stack?

# Algorithm percolation on networks (graphs)

1. Go through each node
2. Put node in the stack
3. Get a node from the stack
4. Go through each unmarked link of the node
5. Put other end of links in the stack if it is not marked
6. Mark nodes
7. If the stack not empty Go to 3.
8. If the stack empty Go to 1.

# Algorithm percolation on networks (graphs)

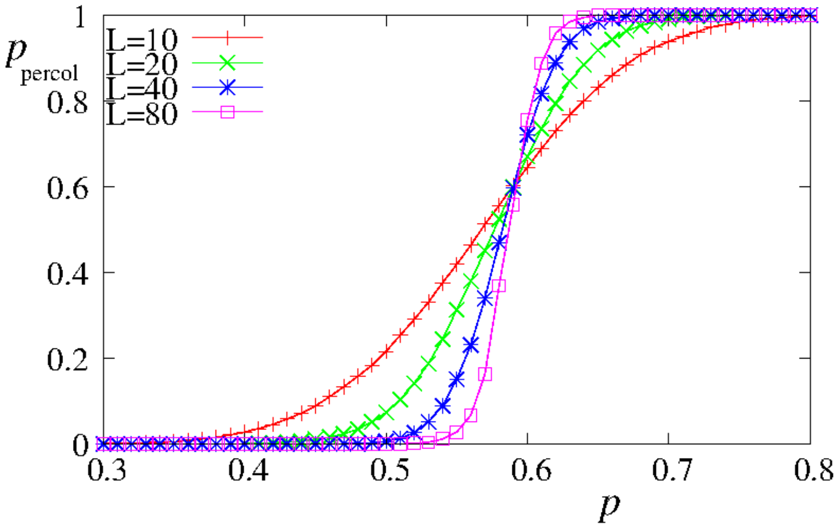


# Algorithm percolation on networks (graphs)

```
int node[N];
int nlink[N];
int link[N][N];
int stack[N]
int sp;

void percol() {
  int a,b,i;
  int cluster;

  sp = 0;
  cluster = 1;
  for (a = 0; a < N; a++) node[N]=0;
  for (a = 0; a < N; a++) {
    if (node[a] == 0) {
      stack[sp++] = a;
      node[a] = cluster++;
    }
    while (sp > 0) {
      i = stack[--sp];
      for (b = 0; b < nlink[i]; b++) {
        if (node[b] == 0) {
          stack[sp++] = b;
          node[b] = node[a];
        }
      }
    }
  }
}
```

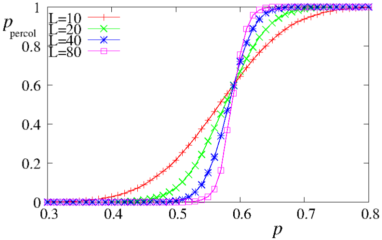
1. Go through each node
2. Put node in the stack
3. Get a node from the stack
4. Go through each unmarked link of the node
5. Put other end of links in the stack if it is not
6. Mark nodes
7. if the stack not empty Go to 3.
8. if the stack empty Go to 1.

# Result

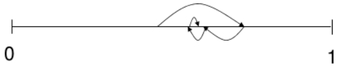


# Determine $p_c$

- From order parameter:

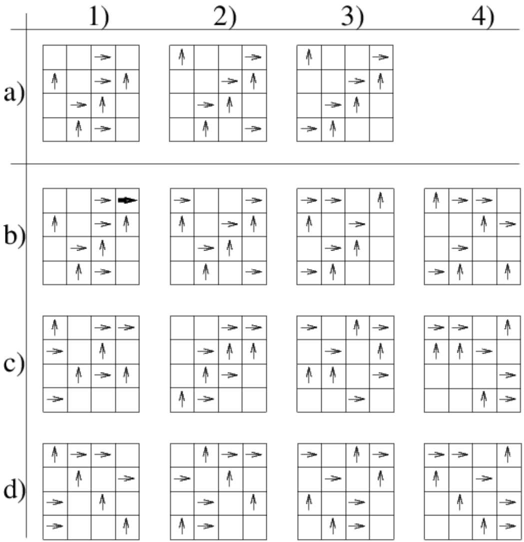



- Increase and decrease $p$ by $p/2$ to converge to $p_c$
- Use the monotonity of the percolation
- Same random number sequence can be generated!



# Monotonity

Not always true!




9. ábra: Az a/1 helyen található konfigurációból kiindulva blokkolt határciklushoz jutunk (a/3). A b/1 helyen az a/1 konfigurációhoz hozzávettük még a vastagon kihúzott nyilat, így a b/1-ben a sűrűség nagyobb lett, mint az a/1-ben. Innét indítva a modellt Szabadon mozgó fázishoz jut (d/4).

# Ising-model

- Spins
  - Interact with extrenal field $h_i$
  - Interact with neighbors with coeff. $J_{ij}$
- The Hamiltonian:

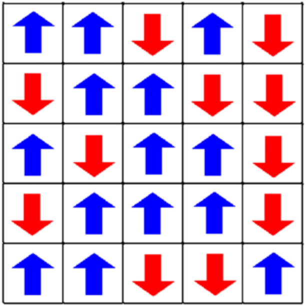
$$H(\sigma) = -\sum_{\langle i\ j \rangle} J_{ij}\sigma_i\sigma_j - \mu \sum_i h_i\sigma_i$$

- Order parameter magnetization

$$M = \sum_i \sigma_i$$

# 2D Ising-model

- 2 dimensions
- Homogeneous interaction: $J_{ij} = J$
- No external field (for the time being) $h = 0$



# Importance sampling

- Given a Hamiltonian $H(\mathbf{q}, \mathbf{p})$
- We ask for the time average of a dynamics quantity at temperature $T$

$$\bar{A} = \int A(\mathbf{q}, \mathbf{p}) P^{eq}(\mathbf{q}, \mathbf{p}, T) d\mathbf{q} d\mathbf{p}$$

- In the canonical ensemble

$$P^{eq}(\mathbf{q}, \mathbf{p}, T) = \frac{1}{Z} e^{-\beta H(\mathbf{q}, \mathbf{p})}$$

- If $A$ depends only on the energy (often the case):

$$\bar{A} = \int A(E) \omega(E) P^{eq}(E, T) dE$$

Importance sampling is needed!

# Importance sampling

- $\omega(E)P^{eq}(E,T)$ has a very sharp peak (for large $N$)
- System spends most of its time *in equilibrium*
- Importance sampling:

Generate configurations with the equilibrium probability

- if configurations are chosen accordingly, the for $K$ measurements:

$$\bar{A} \simeq \frac{1}{K}\sum_{i=1}^{K} A_i$$

How togenerate equilibrium configurations?

# Metropolis algorithm

(Metropoli-Rosenbluth-Rosenbluth- Teller-Teller=$MR^2T^2$ algorithm)

- Sequence of configurations using a Markov chain
- Configuration is generated from the previous one
- Transition probability: equilibrium probability
- Detailed balance:

$$P(x)P(x \to x') = P(x')P(x' \to x)$$

- Rewritten:

$$\frac{P(x \to x')}{P(x' \to x)} = \frac{P(x')}{P(x)} = e^{-\beta \Delta E}$$

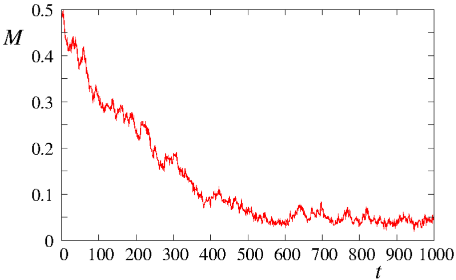
- Only the ration of transition probabilities are fixed

# Characteristic time

- Equilibrium: system is stationary.
- We can measure after relaxation time
- New measurement after correlation time

$$\phi_{EE}(t) = \frac{\langle E(t')E(t'+t)\rangle - \langle E\rangle^2}{\langle E^2\rangle - \langle E\rangle^2}, \quad \tau = \int_0^\infty \phi_{EE}(t)dt$$

- Sample with intervals $\Delta t > \tau$



# Metropolis algorithm

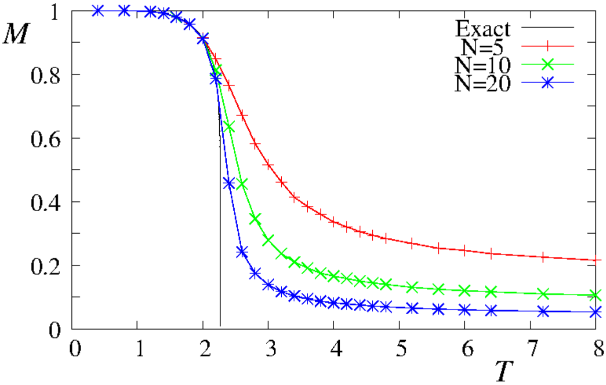
Recipes:

- Choose an elementary step $x \to x'$
- Calculate $\Delta E$
- Calculate $P(x \to x')$
- Generate random number $r \in [0, 1]$
- If $r < P(x \to x')$ then new state is $x'$; otherwise it remains $x$
- Increase time
- Measure what you want
- Restart

# Finite size effects

## Magnetization 2d lattice Ising model

- Determine critical temperature
- Determine critical exponents
- System size dependence???



# Finite size scaling

- Correlation length

$$\xi \propto |T - T_c|^{-\nu}$$

- If $L$ is finite $\xi$ cannot be larger than $L$

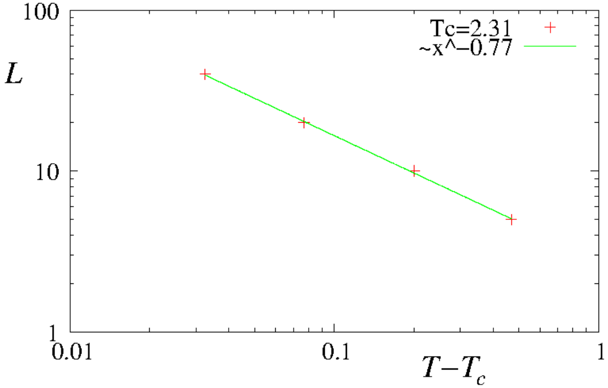
$$L \propto |T(L) - T_c|^{-\nu}$$

- The position and the width of the transition

$$|T(L) - T_c| \propto L^{-1/\nu}$$
$$\sigma(L) \propto L^{-1/\nu}$$

# Three parameter fit: Ising model

- Theory: $\nu = 1$, $T_c \simeq 2.27$



# Finite size scaling: Ising model

- Theory: $\nu = 1$, $T_c \simeq 2.27$