Solutions for Efficient Memory Access for Cubic Lattices and Random Number Algorithms

SAPIENZA UNIVERSITÀ DI ROMA

UNIVERSITEIT TWENTE.

TU/e Technische Universiteit Eindhoven University of Technology

GTC 2015

Speaker: Dr. Matteo Lulli
Prof. M. Bernaschi and Prof. G. Parisi

March the 19th, 2015

Outlook

Motivations

Phase Transitions in disordered systems

Motivations

Phase Transitions in disordered systems

- Equilibrium Monte Carlo analysis works well for non-disordered systems
- Disordered systems are very hard to equilibrate

Motivations

Phase Transitions in disordered systems

- Equilibrium Monte Carlo analysis works well for non-disordered systems
- Disordered systems are very hard to equilibrate
- A large number of disorder realizations, **samples**, is required
- 3D Ising spin glass at most $L = 40$ equilibrated so far (Janus, FPGA dedicated machine)

Motivations

Phase Transitions in disordered systems

- Equilibrium Monte Carlo analysis works well for non-disordered systems
- Disordered systems are very hard to equilibrate
- A large number of disorder realizations, **samples**, is required
- 3D Ising spin glass at most $L = 40$ equilibrated so far (Janus, FPGA dedicated machine)
- Robust out-of-equilibrium methods for the study of phase transitions would be very useful

Motivations

Phase Transitions in disordered systems

- Equilibrium Monte Carlo analysis works well for non-disordered systems
- Disordered systems are very hard to equilibrate
- A large number of disorder realizations, **samples**, is required
- 3D Ising spin glass at most $L = 40$ equilibrated so far (Janus, FPGA dedicated machine)
- Robust out-of-equilibrium methods for the study of phase transitions would be very useful

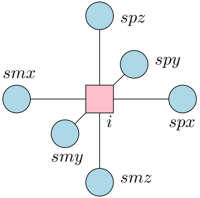
Why GPUs

Even with out-of-equilibrium methods usual CPUs are not sufficiently powerful in order to obtain good estimates

Outline for section 1

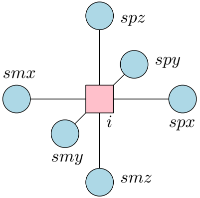
Standard checkerboard pattern

- Nearest-neighbours based problems in 3D



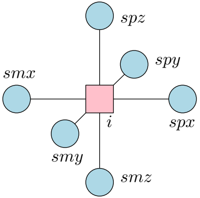
Standard checkerboard pattern

- Nearest-neighbours based problems in 3D
- Cubic lattice of linear size $L = 2n$: Checkerboard colouring
- Each lattice site has nearest neighbours of the other colour



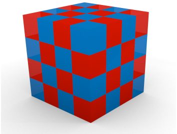
Standard checkerboard pattern

- Nearest-neighbours based problems in 3D
- Cubic lattice of linear size $L = 2n$: Checkerboard colouring
- Each lattice site has nearest neighbours of the other colour



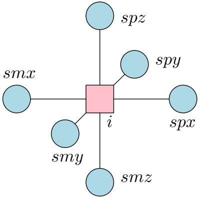
Allocation choices

- Unified allocation: one array



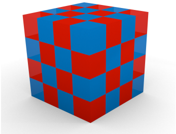
Standard checkerboard pattern

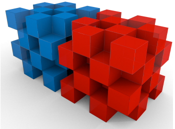
- Nearest-neighbours based problems in 3D
- Cubic lattice of linear size $L = 2n$: Checkerboard colouring
- Each lattice site has nearest neighbours of the other colour



Allocation choices

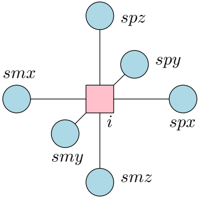
- Unified allocation: one array
- Separated allocation: two arrays





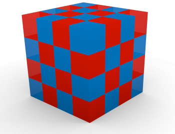
Standard checkerboard pattern

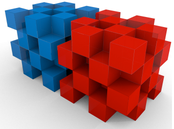
- Nearest-neighbours based problems in 3D
- Cubic lattice of linear size $L = 2n$: Checkerboard colouring
- Each lattice site has nearest neighbours of the other colour



Allocation choices

- Unified allocation: one array
- Separated allocation: two arrays





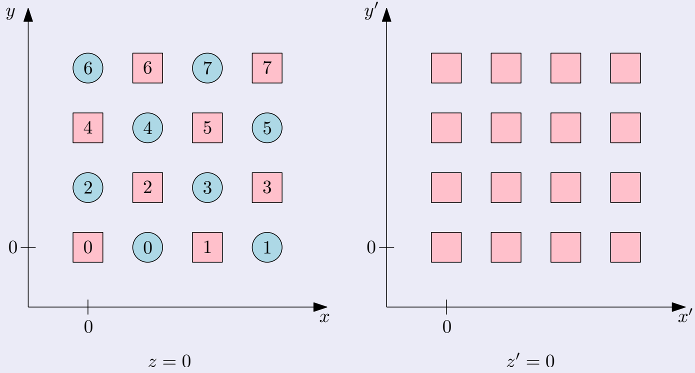
The parity, $(-1)^{x_i+y_i+z_i}$, of each lattice site has to be taken into account

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



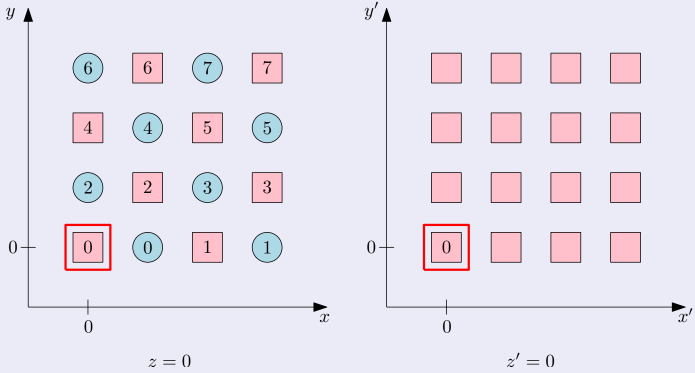
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



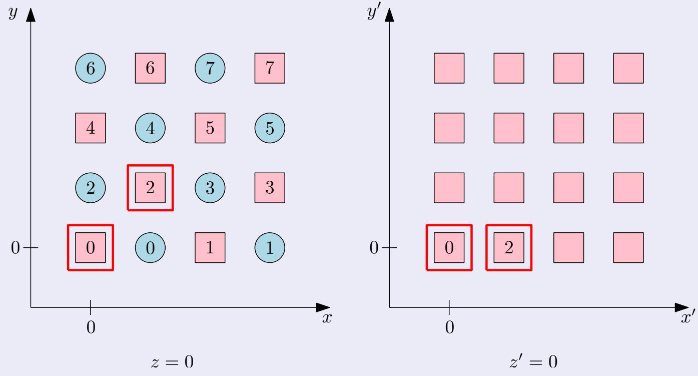
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



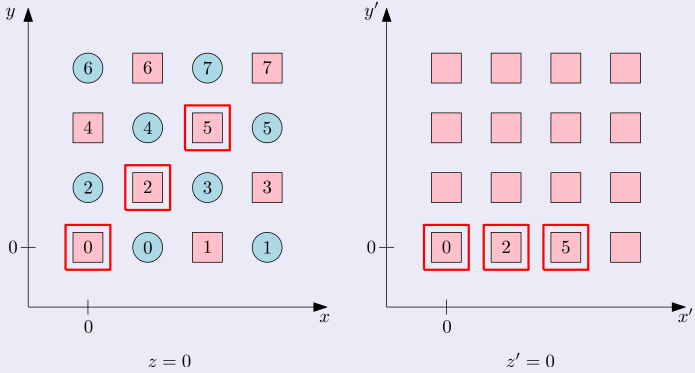
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



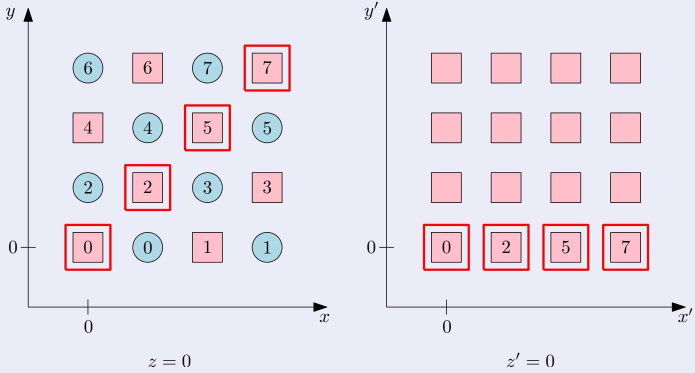
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



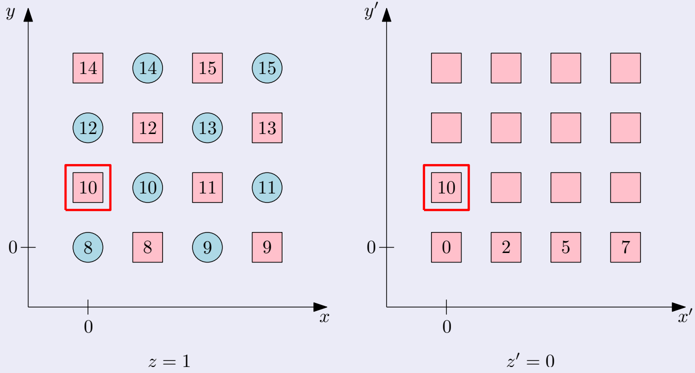
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



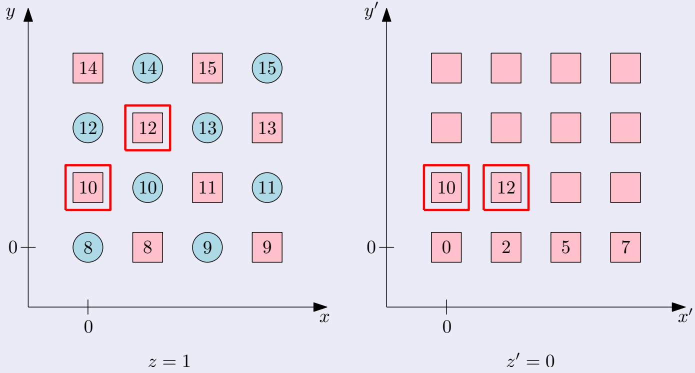
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



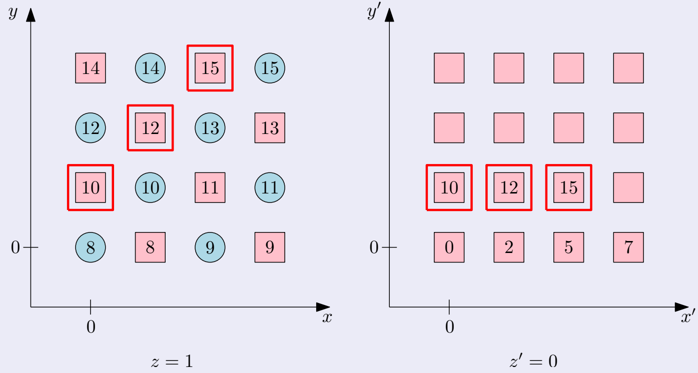
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



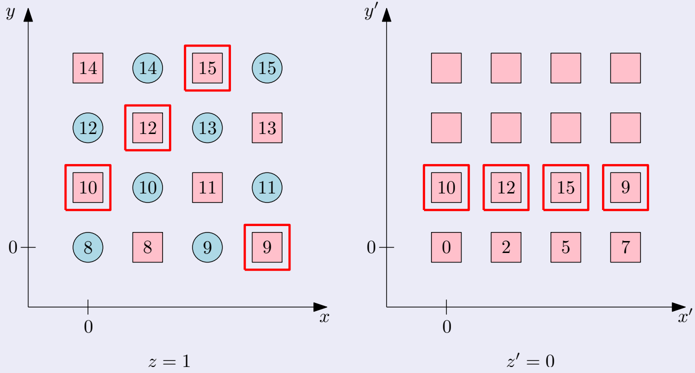
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



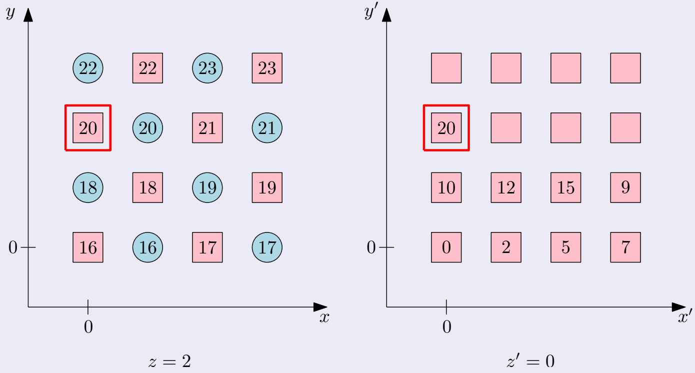
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



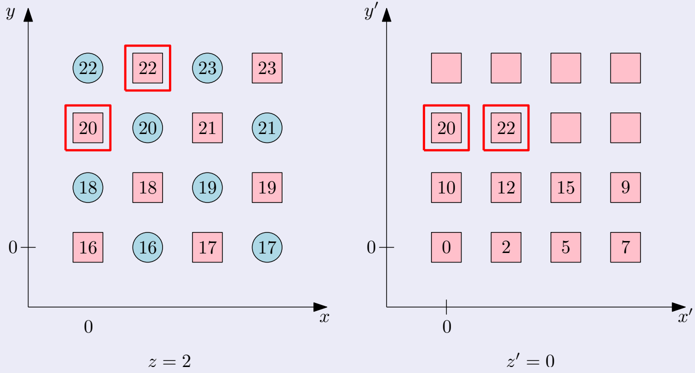
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



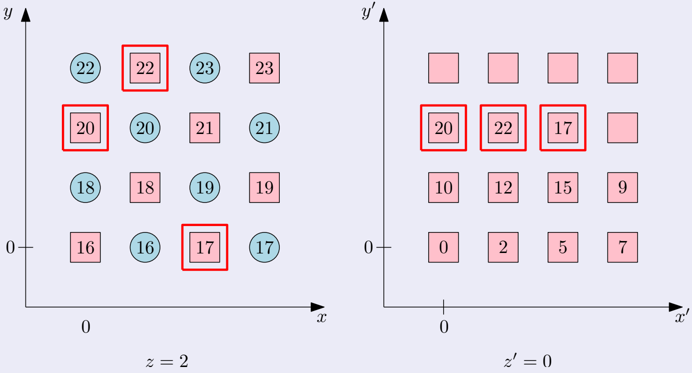
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



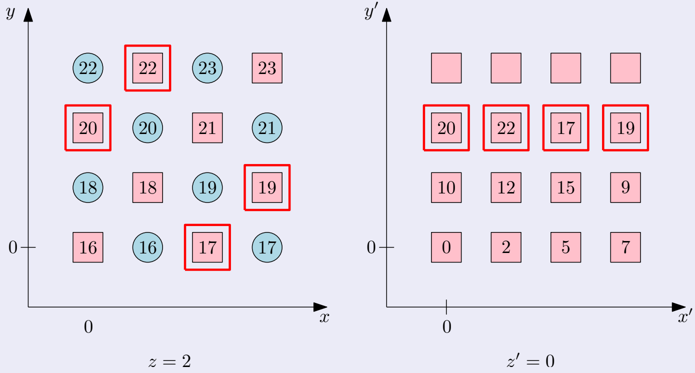
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



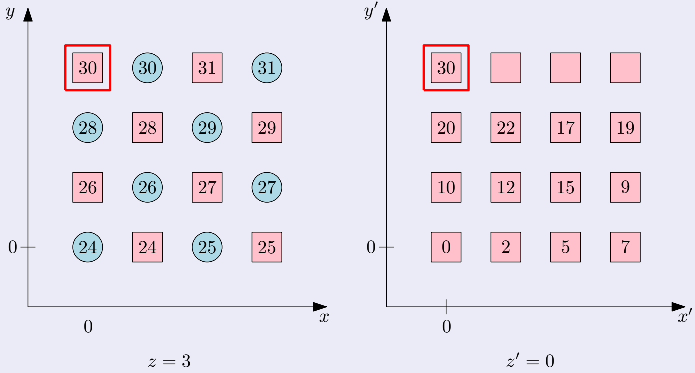
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



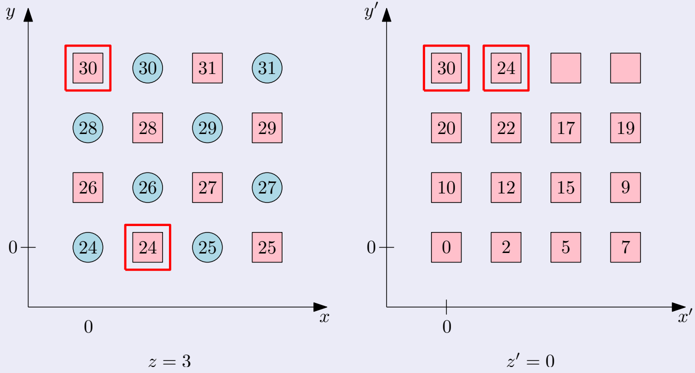
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



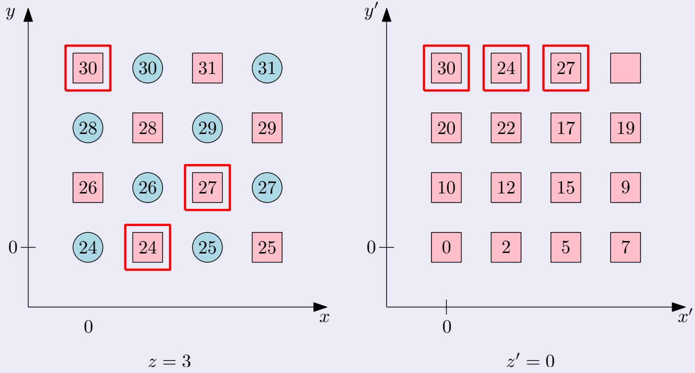
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

Sliced scheme



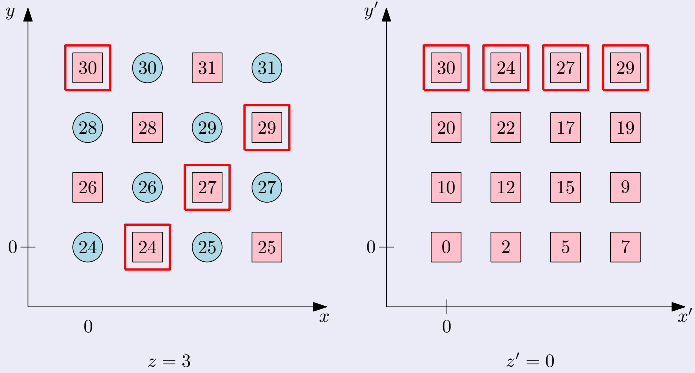
Periodic boundary conditions are necessary

Sliced checkerboard pattern: definition

It is always possible to **remap** the lattice sites

- Yavors'kii *et al.*, Heisenberg spin glass, 'snake-like' pattern, unified
- Ferrero *et al.*, Potts 2D, separated

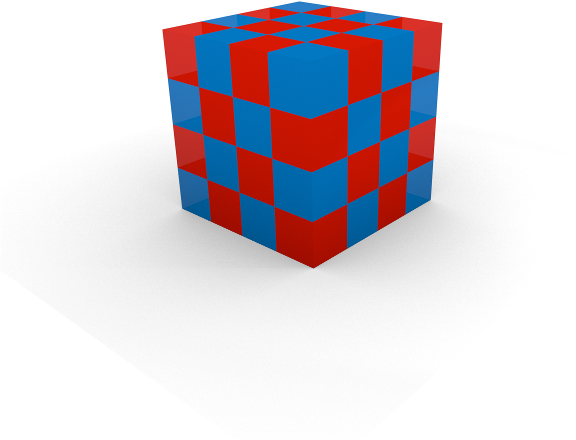
Sliced scheme



Periodic boundary conditions are necessary

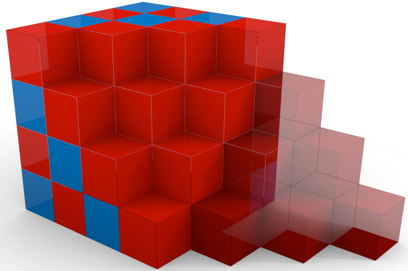
Sliced checkerboard pattern: geometric interpretation

Sites of planes orthogonal to $\vec{n} = (+1, -1, +1)$ are one-coloured.



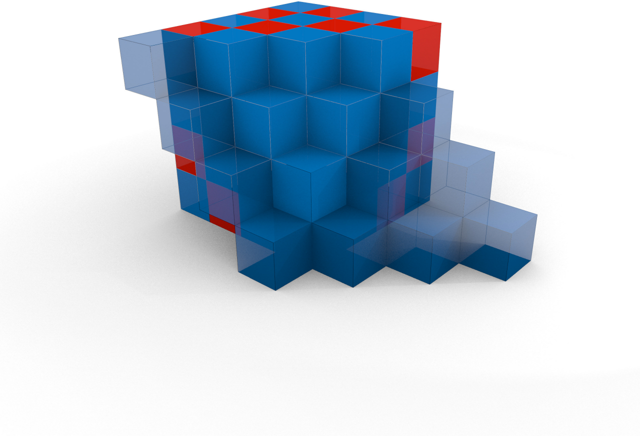
Sliced checkerboard pattern: geometric interpretation

Sites of planes orthogonal to $\vec{n} = (+1, -1, +1)$ are one-coloured.



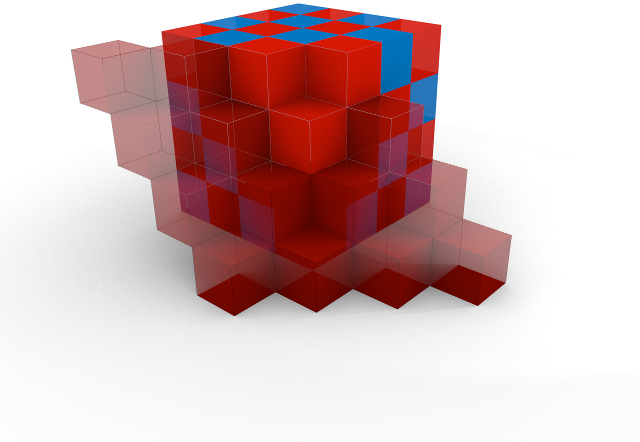
Sliced checkerboard pattern: geometric interpretation

Sites of planes orthogonal to $\vec{n} = (+1, -1, +1)$ are one-coloured.



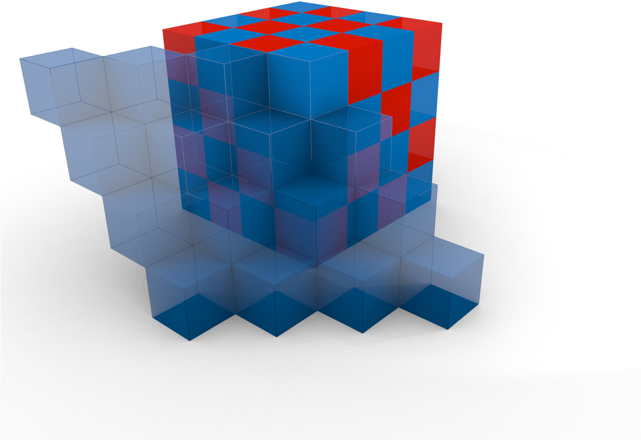
Sliced checkerboard pattern: geometric interpretation

Sites of planes orthogonal to $\vec{n} = (+1, -1, +1)$ are one-coloured.



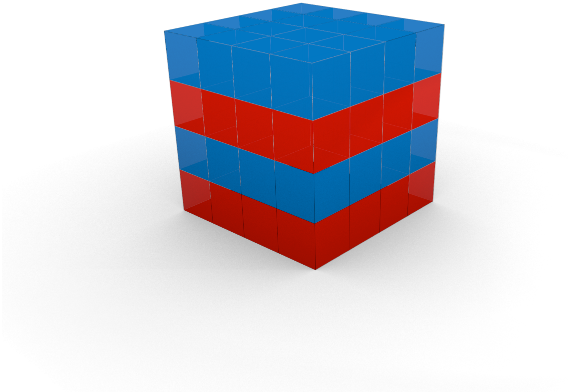
Sliced checkerboard pattern: geometric interpretation

Sites of planes orthogonal to $\vec{n} = (+1, -1, +1)$ are one-coloured.



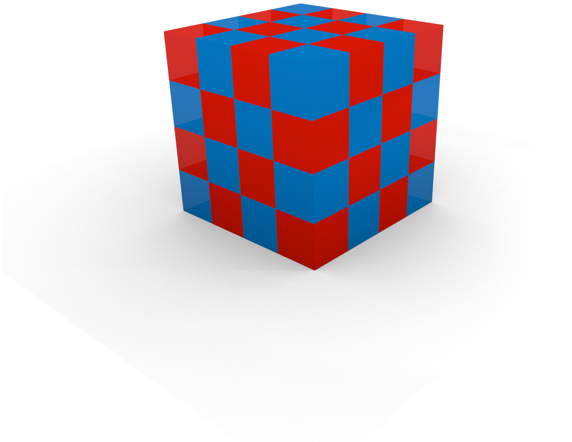
Sliced checkerboard pattern: geometric interpretation

Sites of planes orthogonal to $\vec{n} = (+1, -1, +1)$ are one-coloured.

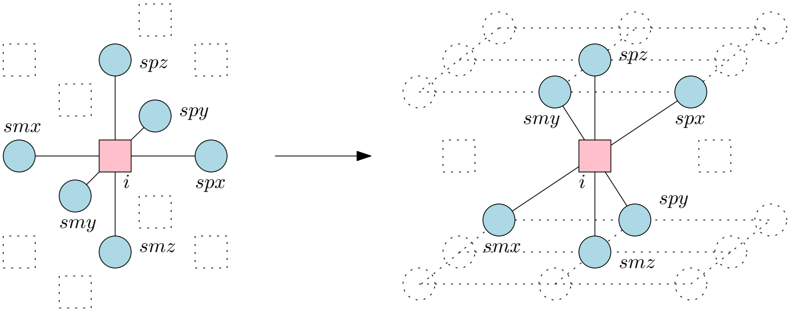


Sliced checkerboard pattern: geometric interpretation

Sites of planes orthogonal to $\vec{n} = (+1, -1, +1)$ are one-coloured.

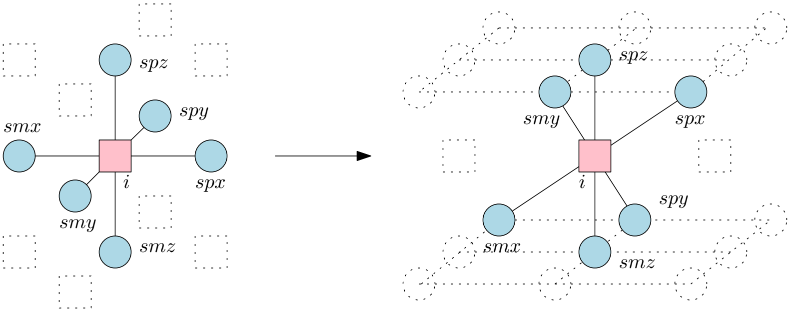


Sliced checkerboard pattern: cubic stencils



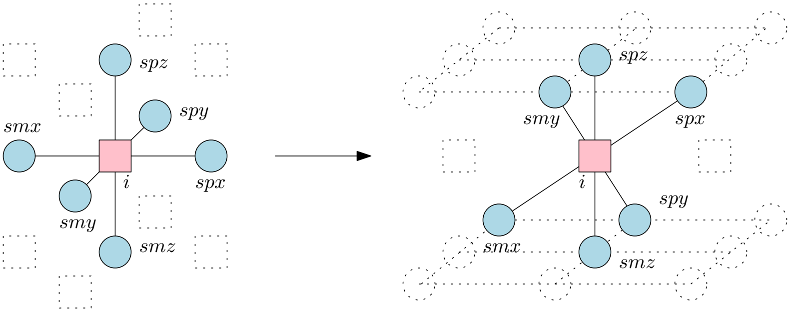
- Variables belonging to one slice are **decoupled**
- No sites parity involved

Sliced checkerboard pattern: cubic stencils



- Variables belonging to one slice are **decoupled**
- No sites parity involved
- The spz, smy, spx nearest neighbours belong to the **upper** slice
- The smz, spy, smx nearest neighbours belong to the **lower** slice

Sliced checkerboard pattern: cubic stencils



- Variables belonging to one slice are **decoupled**
- No sites parity involved
- The spz, smy, spx nearest neighbours belong to the **upper** slice
- The smz, spy, smx nearest neighbours belong to the **lower** slice
- The method can be generalized to any dimension

Outline for section 2

Linear Congruential PRNG

- Lehmer-Park-Miller MINSTD

$$R_{n+1} = (16807\, R_n)\text{mod}(2^{31} - 1),$$

- D. Carta implementation: no modulus and overflow handled with 32 bit integers only

MINSTD overflow handling

No 64 bits variables and no 'mod' involved

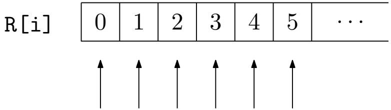
```
RNGT lo = 16807*(seed&0xffff);
RNGT hi = 16807*(seed>>16);
lo += (hi&0x7fff)<<16;
lo += hi>>15;
if(lo > 0x7fffffff) lo -= 0x7fffffff;
return lo;
```

Linear Congruential PRNG

- Lehmer-Park-Miller MINSTD

$$R_{n+1} = (16807\, R_n)\text{mod}(2^{31} - 1),$$

- D. Carta implementation: no modulus and overflow handled with 32 bit integers only
- Easy implementation: one word state
- Coalescence is obtained by definition



MINSTD overflow handling

No 64 bits variables and no 'mod' involved

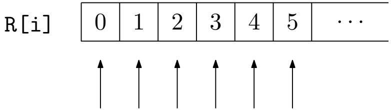
```
RNGT lo = 16807*(seed&0xffff);
RNGT hi = 16807*(seed>>16);
lo += (hi&0x7fff)<<16;
lo += hi>>15;
if(lo > 0x7fffffff) lo -= 0x7fffffff;
return lo;
```

Linear Congruential PRNG

- Lehmer-Park-Miller MINSTD

$$R_{n+1} = (16807\, R_n)\text{mod}(2^{31} - 1),$$

- D. Carta implementation: no modulus and overflow handled with 32 bit integers only
- Easy implementation: one word state
- Coalescence is obtained by definition



- More than one instance: read once, store once
- Low quality random numbers, but still useful in some cases

MINSTD overflow handling

No 64 bits variables and no 'mod' involved

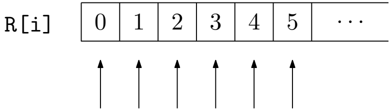
```
RNGT lo = 16807*(seed&0xffff);
RNGT hi = 16807*(seed>>16);
lo += (hi&0x7fff)<<16;
lo += hi>>15;
if(lo > 0x7fffffff) lo -= 0x7fffffff;
return lo;
```

Linear Congruential PRNG

- Lehmer-Park-Miller MINSTD

$$R_{n+1} = (16807\, R_n)\text{mod}(2^{31} - 1),$$

- D. Carta implementation: no modulus and overflow handled with 32 bit integers only
- Easy implementation: one word state
- Coalescence is obtained by definition



- More than one instance: read once, store once
- Low quality random numbers, but still useful in some cases

MINSTD overflow handling

No 64 bits variables and no 'mod' involved

```
RNGT lo = 16807*(seed&0xffff);
RNGT hi = 16807*(seed>>16);
lo += (hi&0x7fff)<<16;
lo += hi>>15;
if(lo > 0x7fffffff) lo -= 0x7fffffff;
return lo;
```

Avoiding 'if' statement

We need to avoid branching

```
lo -= ((-((lo&0x80000000)>>31))&0x7fffffff);
```

Lagged-Fibonacci-like PRNGs: Parisi-Rapuano

Definition

- Modified lagged-Fibonacci PRNG

```
ira[i] = ira[i - 24] + ira[i - 55];
     R = ira[i]^ira[i - 61];
```

- At least 62 entries state

Lagged-Fibonacci-like PRNGs: Parisi-Rapuano

Definition

- Modified lagged-Fibonacci PRNG

```
ira[i] = ira[i - 24] + ira[i - 55];
       R = ira[i]^ira[i - 61];
```

- At least 62 entries state

GPU implementation

- Common practice: load one or more states in Shared Memory. Lags can be used for threads to work together.
- However, lags not suitable (Weigel 2012)
- Good trade-off between efficiency and quality

Lagged-Fibonacci-like PRNGs: Parisi-Rapuano

Definition

- Modified lagged-Fibonacci PRNG

```
ira[i] = ira[i - 24] + ira[i - 55];
      R = ira[i]^ira[i - 61];
```

- At least 62 entries state

GPU implementation

- Common practice: load one or more states in Shared Memory. Lags can be used for threads to work together.
- However, lags not suitable (Weigel 2012)
- Good trade-off between efficiency and quality

Lagged-Fibonacci-like PRNGs: new memory access

- One state per thread with coalescing

```
seed = ira[(i - 24)*threads + threadId]
     + ira[(i - 55)*threads + threadId];

ira[i*threads + threadId] = seed;

seed ^= ira[(i - 61)*threads + threadId];
```

- The method is general and it can be applied to the well-known Mersenne-Twister (MT19937)
- Huge memory consumption $\propto N_{threads} \times N_{state}$
- Parisi-Rapuano: $N_{PR} = 62$
- Mersenne Twister MT19937 $N_{MT} = 624$

Benchmarks: host API

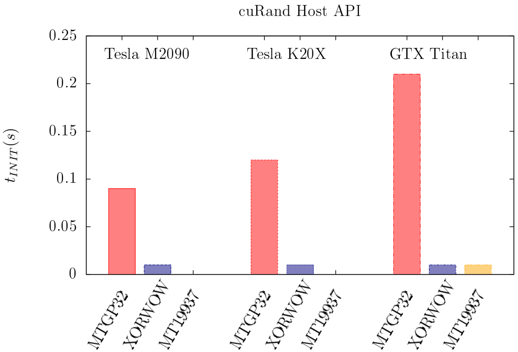
- PRAND benchmark results using cuRand host API
- Filling an array of 2^{29} single-precision floating point variables

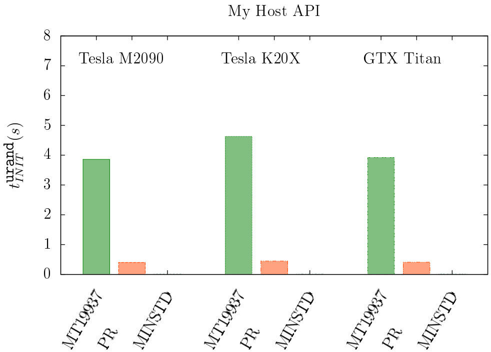
Benchmarks: host API

- PRAND benchmark results using cuRand host API
- Filling an array of 2^{29} single-precision floating point variables
- t_{INIT}: time needed to initialize the PRNG
- $t_{INIT}^{\texttt{urand}}$: time needed to initialize the PRNG using `/dev/urandom`
- t_{GEN}: is the generation time

Benchmarks: host API

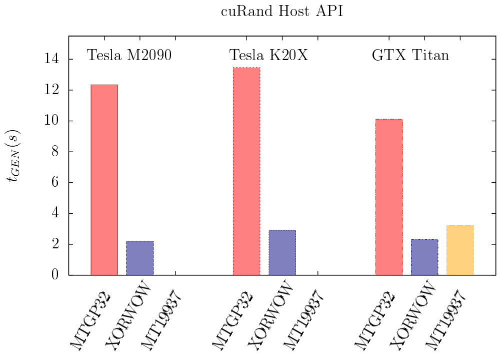
- PRAND benchmark results using cuRand host API
- Filling an array of 2^{29} single-precision floating point variables
- t_{INIT}: time needed to initialize the PRNG
- $t_{INIT}^{\texttt{urand}}$: time needed to initialize the PRNG using `/dev/urandom`
- t_{GEN}: is the generation time

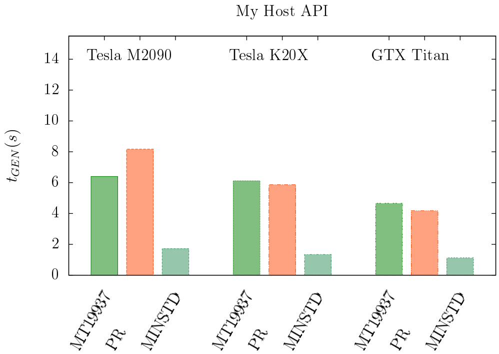




Benchmarks: host API

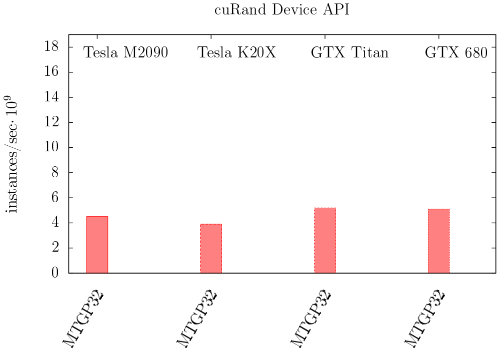
- PRAND benchmark results using cuRand host API
- Filling an array of 2^{29} single-precision floating point variables
- t_{INIT}: time needed to initialize the PRNG
- $t_{INIT}^{\texttt{urand}}$: time needed to initialize the PRNG using `/dev/urandom`
- t_{GEN}: is the generation time

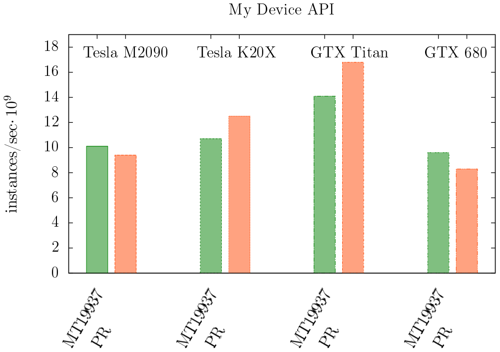




Benchmarks: device API

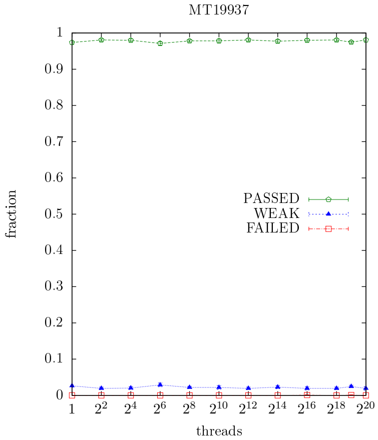
- Number of instances per second
- Launch configuration: 64 blocks of 256 threads
- Each thread produces 2^{15} instances, repeated 10 times





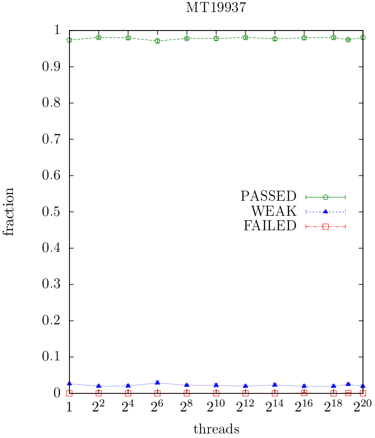
Dieharder Tests

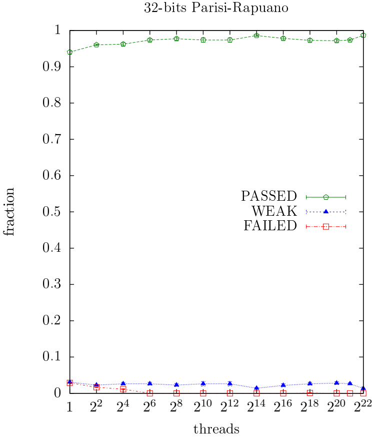
- Let us test the generators initialized via `/dev/urandom` against **Dieharder**



Dieharder Tests

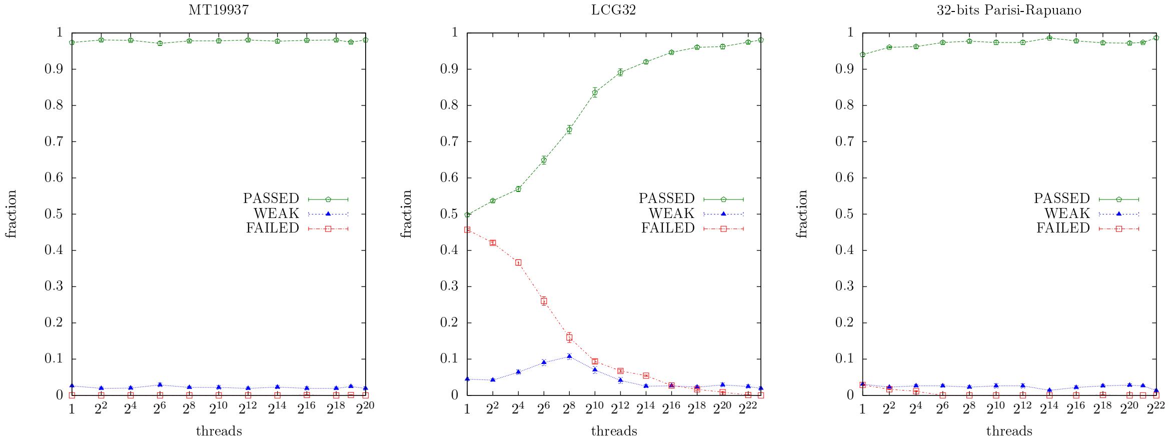
- Let us test the generators initialized via `/dev/urandom` against **Dieharder**





Diehard Tests

- Let us test the generators initialized via `/dev/urandom` against **Dieharder**



Outline for section 3

CUDA streams

- Separated update: all **reds** and then all **blues**
- Communication handled by MPI
- z direction slicing of the system: bulk + boundary

CUDA streams

- Separated update: all **reds** and then all **blues**
- Communication handled by MPI
- z direction slicing of the system: bulk + boundary

Strategy

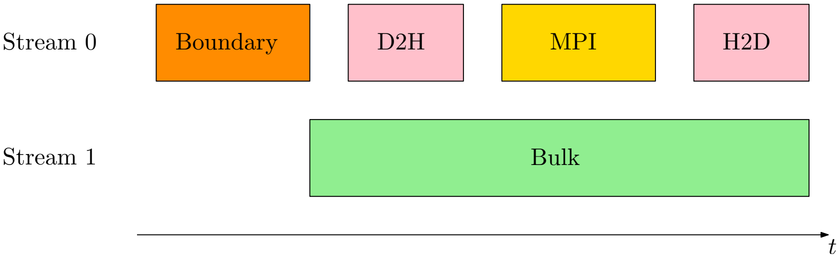
- Overlapping the update of the boundaries and their transfer with the update of the bulk
- One stream each

CUDA streams

- Separated update: all **reds** and then all **blues**
- Communication handled by MPI
- z direction slicing of the system: bulk + boundary

Strategy

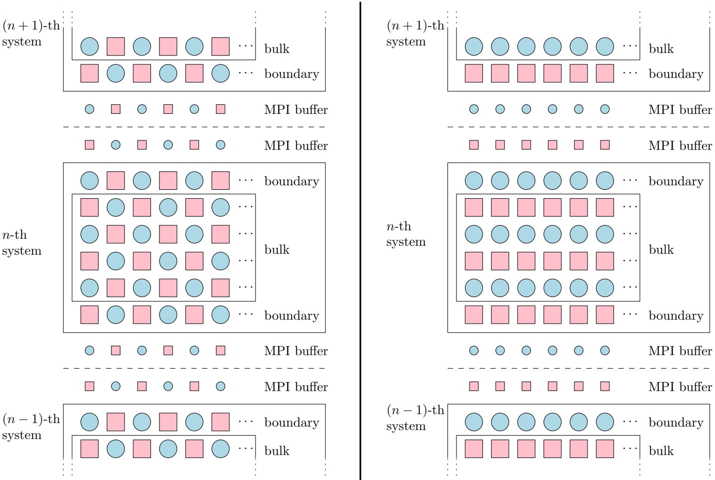
- Overlapping the update of the boundaries and their transfer with the update of the bulk
- One stream each



Sliced checkerboard and MPI

- **Standard** checkerboard boundaries are **two-coloured**
- **Sliced** checkerboard boundaries are **one-coloured**

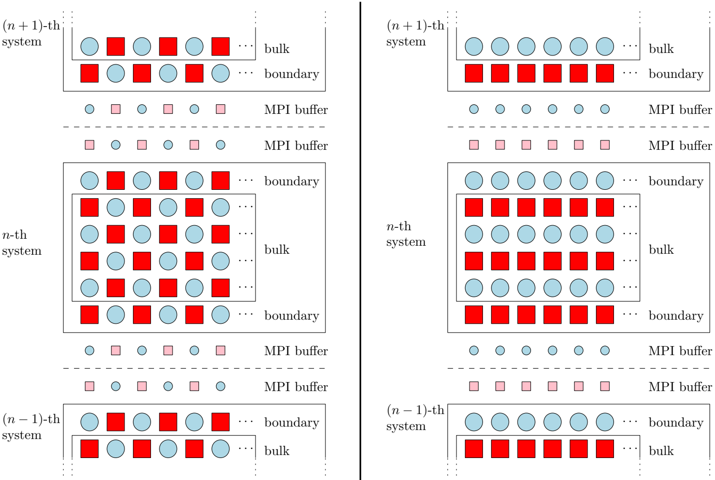
Sliced scheme: communication between nodes is **one-directional only**



Sliced checkerboard and MPI

- **Standard** checkerboard boundaries are **two-coloured**
- **Sliced** checkerboard boundaries are **one-coloured**

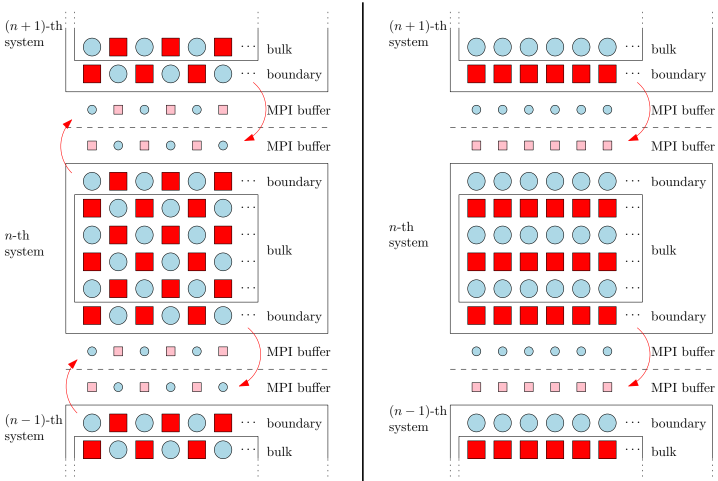
Sliced scheme: communication between nodes is **one-directional only**



Sliced checkerboard and MPI

- **Standard** checkerboard boundaries are **two-coloured**
- **Sliced** checkerboard boundaries are **one-coloured**

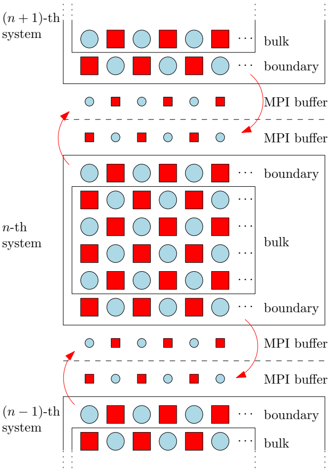
Sliced scheme: communication between nodes is **one-directional only**

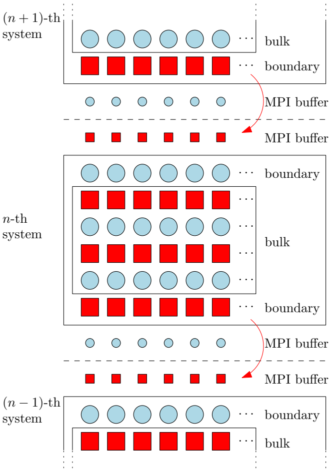


Sliced checkerboard and MPI

- **Standard** checkerboard boundaries are **two-coloured**
- **Sliced** checkerboard boundaries are **one-coloured**

Sliced scheme: communication between nodes is **one-directional only**





Outline for section 4

Three-dimensional Edwards-Anderson model

- Three-dimensional Spin Glass model

$$H = -\sum_{\langle ik \rangle} J_{ik}\sigma_i\sigma_k, \qquad \sigma_i \in \{-1,+1\}, \qquad J_{ik} \in \{-1,+1\}$$

- Bimodal disorder

$$P(J_{ik}) = \frac{1}{2}\left[\delta_K(J_{ik}+1) + \delta_K(J_{ik}-1)\right]$$

- Two-valued variables: one variable-one bit
- Asynchronous MultiSpin-Coding (AMSC) avoiding '`if`' statements

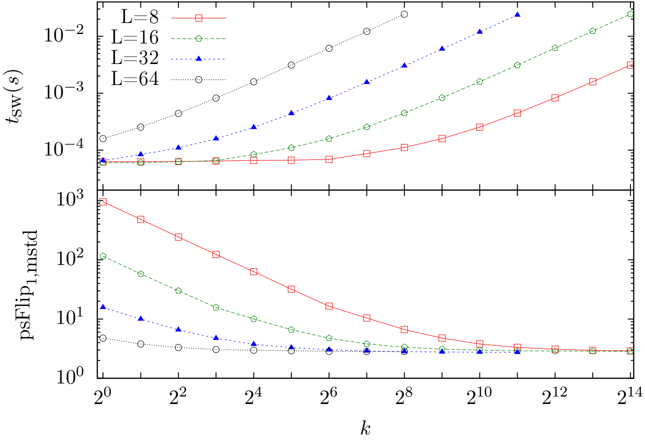
$$\begin{aligned} J_{ik} &= -1 \rightarrow 1 \\ J_{ik} &= +1 \rightarrow 0 \\ \sigma_i &= +1 \rightarrow 1 \\ \sigma_i &= -1 \rightarrow 0 \\ \mathrm{e}_{ik} &= \sigma_i \hat{}\, \sigma_k \hat{}\, \mathrm{J}_{ik} \end{aligned}$$

$$\begin{array}{|c|}\hline \mathrm{e}_{ik0} \\ \hline \mathrm{e}_{ik1} \\ \hline \mathrm{e}_{ik2} \\ \hline \vdots \\ \hline \end{array} = \begin{array}{|c|}\hline \sigma_{i0} \\ \hline \sigma_{i1} \\ \hline \sigma_{i2} \\ \hline \vdots \\ \hline \end{array} \hat{} \begin{array}{|c|}\hline \sigma_{k0} \\ \hline \sigma_{k1} \\ \hline \sigma_{k2} \\ \hline \vdots \\ \hline \end{array} \hat{} \begin{array}{|c|}\hline \mathrm{J}_{ik0} \\ \hline \mathrm{J}_{ik1} \\ \hline \mathrm{J}_{ik2} \\ \hline \vdots \\ \hline \end{array} \qquad \sum_{\langle ik \rangle} \begin{array}{|c|}\hline \mathrm{e}_{ik0} \\ \hline \mathrm{e}_{ik1} \\ \hline \mathrm{e}_{ik2} \\ \hline \vdots \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline \mathrm{s2}_{i0} & \mathrm{s1}_{i0} & \mathrm{s0}_{i0} \\ \hline \mathrm{s2}_{i1} & \mathrm{s1}_{i1} & \mathrm{s0}_{i1} \\ \hline \mathrm{s2}_{i2} & \mathrm{s1}_{i2} & \mathrm{s0}_{i2} \\ \hline \vdots & \vdots & \vdots \\ \hline \end{array}$$

Single- & Multi-GPU Results

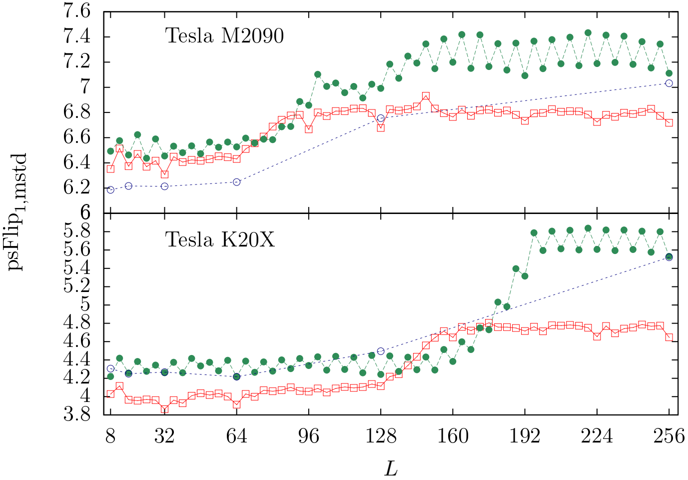
- The smaller the lattice the more disorder realizations are needed to attain saturation
- Let k be the number of coded disorder realizations and $\mathsf{psFlip}_{n,\mathsf{mstd}}$

$$\mathsf{psFlip}_{n,\mathsf{mstd}}(L,k) = t_{\mathsf{sw}} \cdot n \cdot \left(32 \cdot k \cdot 4 \cdot L^3\right)^{-1},$$



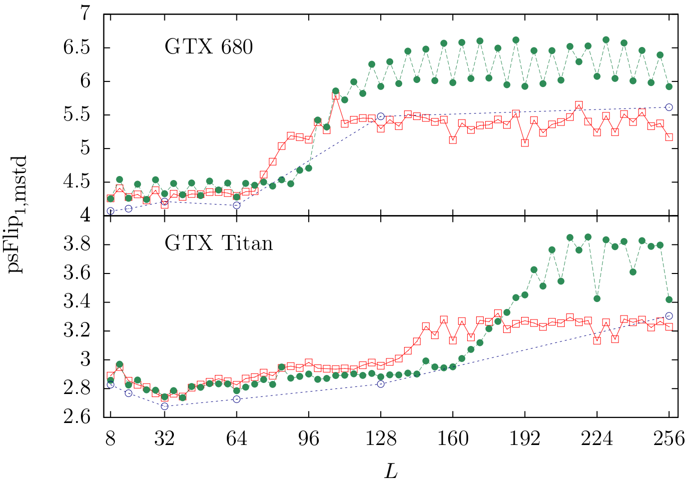
Single- & Multi-GPU Results

- Values for $\mathrm{psFlip}_{1,\mathrm{mstd}}$ for different values of $L = 4k$
- Sliced (red), Standard (green), Standard-Bitwise (blue)



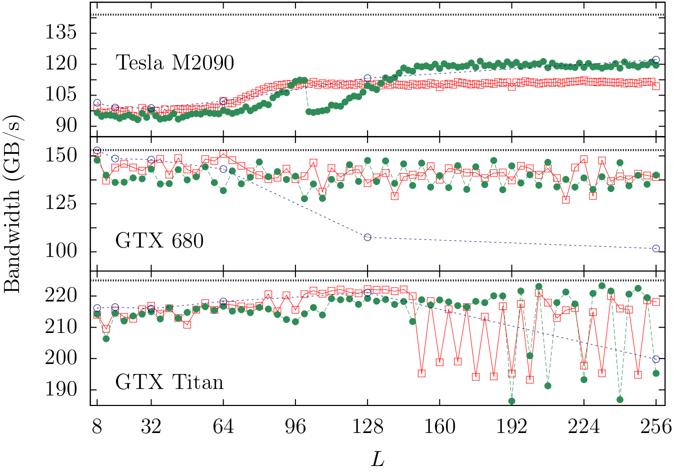
Single- & Multi-GPU Results

- Values for $\text{psFlip}_{1,\text{mstd}}$ for different values of $L = 4k$
- Sliced (red), Standard (green), Standard-Bitwise (blue)



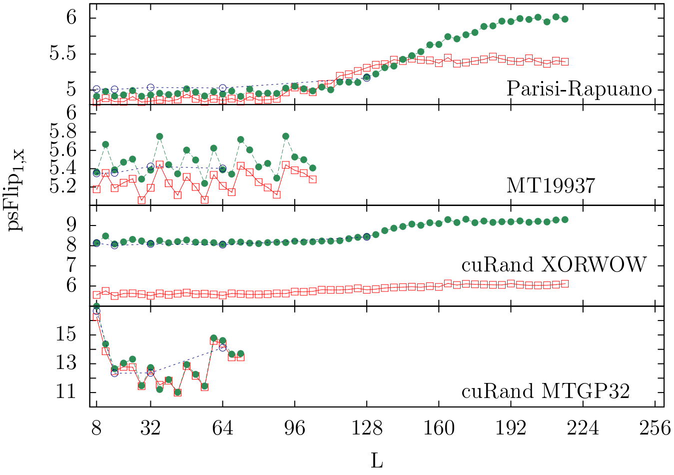
Single- & Multi-GPU Results

- Bandwidth for different values of $L = 4k$
- Sliced (red), Standard (green), Standard-Bitwise (blue)



Single- & Multi-GPU Results

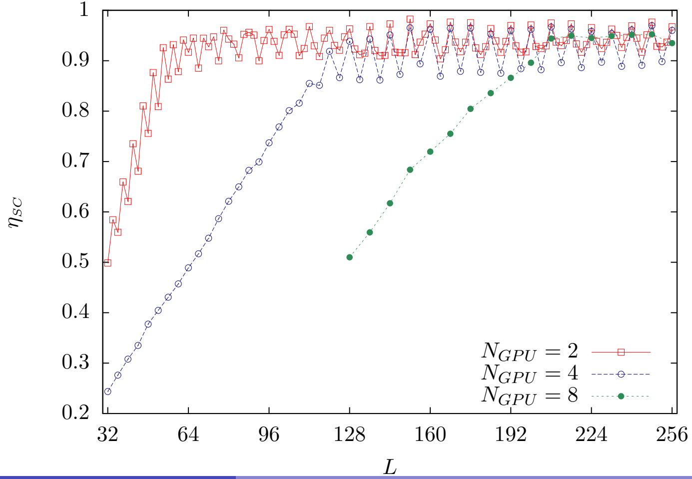
- Values for $\text{psFlip}_{1,x}$ for different random number generators and $L = 4k$
- Sliced (red), Standard (green), Standard-Bitwise (blue)



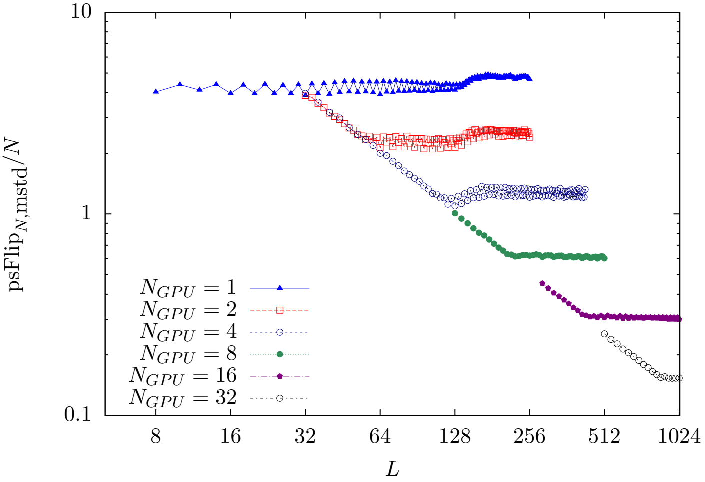
Single- & Multi-GPU Results (CSCS Piz Daint)

- strong-scaling efficiency

$$\eta_{SC} = \frac{\mathsf{psFlip}_{1,\mathsf{x}}}{\mathsf{psFlip}_{N,\mathsf{x}}},$$



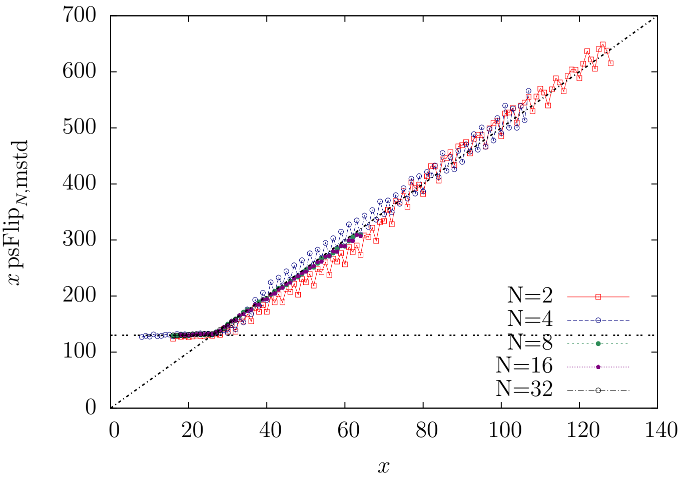
Single- & Multi-GPU Results (CSCS Piz Daint)



Single- & Multi-GPU Results (CSCS Piz Daint)

- Data scaling

$$\frac{L}{N}\,\mathsf{psFlip}_{N,\mathsf{x}}(L,k) \propto t_{\mathsf{sw}}\left(\frac{L}{N}\right)^{-2}, \qquad x = \frac{L}{N}.$$



Physics results

- This GPU implementation together with a **new** theoretical approach allowed us to obtain **cutting-edge** estimations for the critical parameters for the 3D Ising Spin Glass phase transition
- 3.1 years of one GTX Titan
- ~ 20 faster than high-end CPUs implementations and comparable with FPGA (Janus 2, 2013)

	T_c	ν	η	ω	z	Years
Janus 2013	$1.1019(29)$	$2.562(42)$	$-0.3900(36)$	$1.12(10)$		75 (FPGA)
Our Work[1]	$1.099(5)$	$2.47(10)$	$-0.39(1)$	$1.3(2)$	$6.80(15)$	3.1 (GPU)
Haus. et al. 2008	$1.109(10)$	$2.45(15)$	$-0.375(10)$	$1.0(1)$		40 (CPU)

The use of GPUs was necessary and unavoidable in order to prove the validity and effectiveness of the new theoretic approach

[1] M. Lulli, G. Parisi & A. Pelissetto in preparation

Outline for section 5

Conclusions & Perspectives

Conclusions

- A new **sliced** cubic stencil access pattern which in some cases performs better than highly tuned solutions
- A new access pattern for lagged-Fibonacci-like PRNGs which performs better than analogues in CURAND ($\sim 2\times$)
- 3D Ising Spin Glass single GPU: performances are stable for a large interval (the largest so far) of L; the MT19937 PRNG performs **only** $\sim 10\%$ slower than the PR
- The **sliced** multi-GPU version shows a very good strong scaling efficiency and competitive speeds for dynamic studies
- Together with a new off-equilibrium finite-size scaling approach we obtained the **2nd** most precise estimates for EA3D critical parameters so far

Perspectives

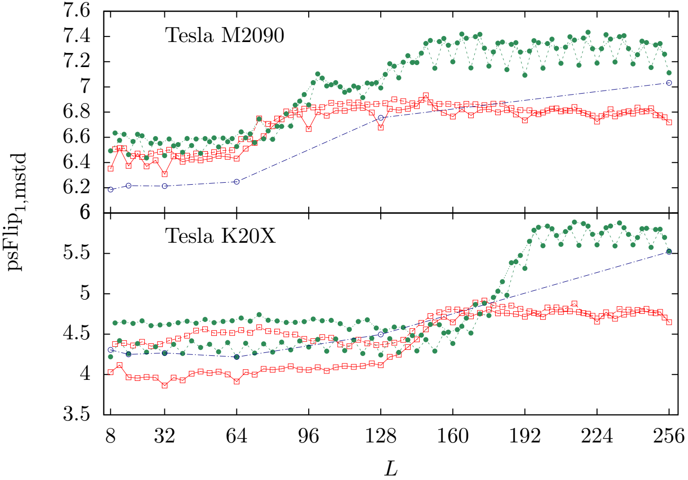
- Implement Parallel Tempering dynamics for equilibrium simulations

matteo.lulli@gmail.com

...and please give your feedback!

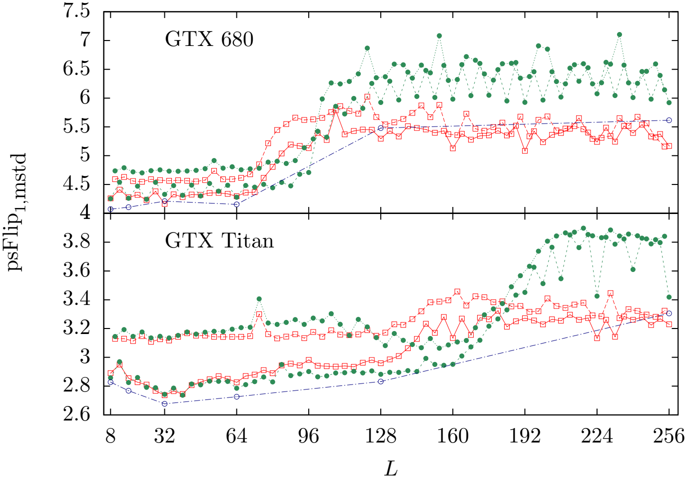
Even and Odd subsequences for $L = 2n$

- Values for $\text{psFlip}_{1,\text{mstd}}$ for different values of $\mathbf{L = 2n}$
- Sliced (red), Standard (green), Standard-Bitwise (blue)



Even and Odd subsequences for $L = 2n$

- Values for $\text{psFlip}_{1,\text{mstd}}$ for different values of $\mathbf{L = 2n}$
- Sliced (red), Standard (green), Standard-Bitwise (blue)



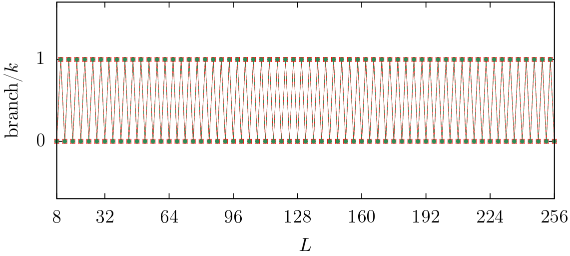
Even and Odd subsequences for $L = 2n$

What is happening?

$$L_0 = 2(2m), \qquad \frac{V_0}{2} = \frac{(4m)^3}{2} = 32m^3, \qquad (32m^3)\text{mod}(32) = 0, \quad \forall\, m \in \mathbb{N},$$

$$L_1 = 2(2m+1), \qquad \frac{V_1}{2} = \frac{[2(2m+1)]^3}{2} = 4(2m+1)^3, \qquad [4(2m+1)^3]\text{mod}(32) \neq 0, \quad \forall\, m \in \mathbb{N}.$$

- The **odd** subsequence is never commensurate to the actual warp size.
- For **every** sample there is **one** warp branching



More on MultiSpin-Coding

- Two-valued variables: one variable-one bit

More on MultiSpin-Coding

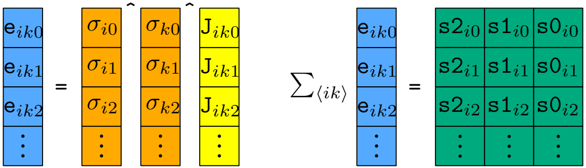
- Two-valued variables: one variable-one bit
- Asynchronous MultiSpin-Coding (AMSC)

More on MultiSpin-Coding

- Two-valued variables: one variable-one bit
- Asynchronous MultiSpin-Coding (AMSC)

$$J_{ik} = -1 \rightarrow 1$$
$$J_{ik} = +1 \rightarrow 0$$
$$\sigma_i = +1 \rightarrow 1$$
$$\sigma_i = -1 \rightarrow 0$$

$$\mathrm{e}_{ik} = \sigma_i \hat{}\, \sigma_k \hat{}\, \mathrm{J}_{ik}$$



More on MultiSpin-Coding

- Two-valued variables: one variable-one bit
- Asynchronous MultiSpin-Coding (AMSC)

$$
\begin{aligned}
J_{ik} &= -1 \to 1 \\
J_{ik} &= +1 \to 0 \\
\sigma_i &= +1 \to 1 \\
\sigma_i &= -1 \to 0 \\
\mathrm{e}_{ik} &= \sigma_i \,\hat{}\, \sigma_k \,\hat{}\, \mathrm{J}_{ik}
\end{aligned}
$$

$$
\begin{bmatrix} \mathrm{e}_{ik0} \\ \mathrm{e}_{ik1} \\ \mathrm{e}_{ik2} \\ \vdots \end{bmatrix}
=
\begin{bmatrix} \sigma_{i0} \\ \sigma_{i1} \\ \sigma_{i2} \\ \vdots \end{bmatrix}
\hat{}
\begin{bmatrix} \sigma_{k0} \\ \sigma_{k1} \\ \sigma_{k2} \\ \vdots \end{bmatrix}
\hat{}
\begin{bmatrix} \mathrm{J}_{ik0} \\ \mathrm{J}_{ik1} \\ \mathrm{J}_{ik2} \\ \vdots \end{bmatrix}
\qquad
\sum_{\langle ik \rangle}
\begin{bmatrix} \mathrm{e}_{ik0} \\ \mathrm{e}_{ik1} \\ \mathrm{e}_{ik2} \\ \vdots \end{bmatrix}
=
\begin{bmatrix} \mathrm{s2}_{i0} & \mathrm{s1}_{i0} & \mathrm{s0}_{i0} \\ \mathrm{s2}_{i1} & \mathrm{s1}_{i1} & \mathrm{s0}_{i1} \\ \mathrm{s2}_{i2} & \mathrm{s1}_{i2} & \mathrm{s0}_{i2} \\ \vdots & \vdots & \vdots \end{bmatrix}
$$

Metropolis Dynamics - Avoiding 'if' statements

$$\Delta E = H[\{\sigma_{i\neq a}, -\sigma_a\}] - H[\{\sigma_{i\neq a}, \sigma_a\}] = -12, -8, -4, 0, 4, 8, 12.$$

The acceptance probability

$$P_{\text{flip}}(\Delta E) = \begin{cases} 1, & \Delta E \leq 0 \\ e^{-\beta \Delta E}, & \Delta E > 0 \end{cases}$$

More on MultiSpin-Coding

- Two-valued variables: one variable-one bit
- Asynchronous MultiSpin-Coding (AMSC)

$$
\begin{aligned}
J_{ik} &= -1 \to 1 \\
J_{ik} &= +1 \to 0 \\
\sigma_i &= +1 \to 1 \\
\sigma_i &= -1 \to 0
\end{aligned}
$$

$$\mathsf{e}_{ik} = \sigma_i \,\hat{}\, \sigma_k \,\hat{}\, \mathsf{J}_{ik}$$

$$
\begin{array}{|c|}\hline \mathsf{e}_{ik0} \\ \hline \mathsf{e}_{ik1} \\ \hline \mathsf{e}_{ik2} \\ \hline \vdots \\ \hline \end{array}
=
\begin{array}{|c|}\hline \sigma_{i0} \\ \hline \sigma_{i1} \\ \hline \sigma_{i2} \\ \hline \vdots \\ \hline \end{array}
\,\hat{}\,
\begin{array}{|c|}\hline \sigma_{k0} \\ \hline \sigma_{k1} \\ \hline \sigma_{k2} \\ \hline \vdots \\ \hline \end{array}
\,\hat{}\,
\begin{array}{|c|}\hline \mathsf{J}_{ik0} \\ \hline \mathsf{J}_{ik1} \\ \hline \mathsf{J}_{ik2} \\ \hline \vdots \\ \hline \end{array}
\qquad
\sum_{\langle ik \rangle}
\begin{array}{|c|}\hline \mathsf{e}_{ik0} \\ \hline \mathsf{e}_{ik1} \\ \hline \mathsf{e}_{ik2} \\ \hline \vdots \\ \hline \end{array}
=
\begin{array}{|c|c|c|}\hline \mathsf{s2}_{i0} & \mathsf{s1}_{i0} & \mathsf{s0}_{i0} \\ \hline \mathsf{s2}_{i1} & \mathsf{s1}_{i1} & \mathsf{s0}_{i1} \\ \hline \mathsf{s2}_{i2} & \mathsf{s1}_{i2} & \mathsf{s0}_{i2} \\ \hline \vdots & \vdots & \vdots \\ \hline \end{array}
$$

Metropolis Dynamics - Avoiding 'if' statements

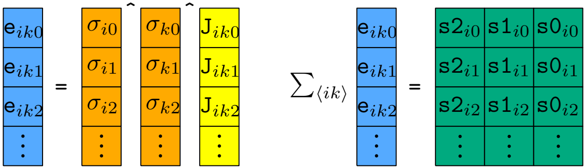
$$
\begin{array}{lclllcl}
(0,\ 0,\ 0) = 0 & \to & \Delta E = -12 & \quad & (1,\ 0,\ 0) = 4 & \to & \Delta E = 4 \\
(0,\ 0,\ 1) = 1 & \to & \Delta E = -8 & & (1,\ 0,\ 1) = 5 & \to & \Delta E = 8 \\
(0,\ 1,\ 0) = 2 & \to & \Delta E = -4 & & (1,\ 1,\ 0) = 6 & \to & \Delta E = 12 \\
(0,\ 1,\ 1) = 3 & \to & \Delta E = 0 & & & &
\end{array}
$$

More on MultiSpin-Coding

- Two-valued variables: one variable-one bit
- Asynchronous MultiSpin-Coding (AMSC)

$$J_{ik} = -1 \to 1$$
$$J_{ik} = +1 \to 0$$
$$\sigma_i = +1 \to 1$$
$$\sigma_i = -1 \to 0$$

$$\mathtt{e}_{ik} = \sigma_i \hat{} \sigma_k \hat{} \mathtt{J}_{ik}$$



Metropolis Dynamics - Avoiding 'if' statements

Normalized transition probabilities & random number comparison

$$R_{max} \exp(-\beta \Delta E) = \mathtt{EXP12}, \mathtt{EXP8}, \mathtt{EXP4}$$

```
cond12 = -(R < EXP12);  cond8 = -(R < EXP8);  cond4 = -(R < EXP4);
```

Spin flip: `spin = mask^spin;`

```
mask = cond12 | (~sum2)
       | ((sum2 & (sum2^sum1)) & (cond8 | (cond4 & (~sum0))));
```

Tiny differences - Calligraphy

Sliced Addressing

```
x = i%d_L; y = (i/d_L)%d_L; z = i/d_A;


smz = i + (SM(z - 1, d_hL) - z)*d_A;
spy = smz + (SP(y + 1, d_L) - y)*d_L;
smy = i + (SM(y - 1, d_L) - y)*d_L;
smx = spy - x + SM(x - 1, d_L);
spx = smy - x + SP(x + 1, d_L);
```

Standard Addressing

```
x = i%d_hL; y = (i/d_hL)%d_L; z = i/d_hA;
par = (y^z)&1;

spx = i - x + SP(x + 1 - (par^1), d_hL);
smx = i - x + SM(x - 1 + par, d_hL);
spy = i + (SP(y + 1, d_L) - y)*d_hL;
smy = i + (SM(y - 1, d_L) - y)*d_hL;
spz = i + (SP(z + 1, d_L) - z)*d_hA;
smz = i + (SM(z - 1, d_L) - z)*d_hA;
```

For the sliced scheme `spz = i`, and **parity** is not involved