Multi-GPU accelerated multi-spin Monte Carlo simulations of the 2D Ising model

$$\begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \downarrow \\ \uparrow & \uparrow & \downarrow & \uparrow & \downarrow \\ \uparrow & \downarrow & \uparrow & \uparrow & \downarrow \end{array} \quad +$$

$=$

What was done?

- **Ising model**

$$H = -J \sum_{<i,j>} S_i S_j$$

- **Metropolis algorithm:**
  Efficient sampling

$$p_a = e^{-\frac{E}{k_B T}}$$

What was done?

- **Ising model**

$$H = -J \sum_{<i,j>} S_i S_j$$

- **Metropolis algorithm:**
  Efficient sampling

$$p_a = e^{-\frac{E}{k_B T}}$$

Single core CPU

What was done?

- **Ising model**

$$H = -J \sum_{<i,j>} S_i S_j$$

- **Metropolis algorithm:**
  Efficient sampling

$$p_a = e^{-\frac{E}{k_B T}}$$

Single core CPU

Single GPU

What was done?

- **Ising model**

$$H = -J \sum_{<i,j>} S_i S_j$$

- **Metropolis algorithm:**
  Efficient sampling

$$p_a = e^{-\frac{E}{k_B T}}$$

Single core CPU

Single GPU

Mutliple GPUs

What was done?

- **Ising model**

$$H = -J \sum_{<i,j>} S_i S_j$$

- **Metropolis algorithm:**
  Efficient sampling

$$p_a = e^{-\frac{E}{k_B T}}$$

Single core CPU

Single GPU

Mutliple GPUs

How does it scale? Is it worth the effort?

Ising model I

- Formula:

$$H = -J \sum_{<i,j>} S_i S_j$$

Ising model I

- Formula:

$$H = -J \sum_{<i,j>} S_i S_j$$

- A standard model of statistical physics

Ising model I

- Formula:

$$H = -J \sum_{<i,j>} S_i S_j$$

- A standard model of statistical physics
- classical model

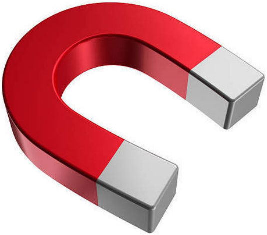
Ising model I

- Formula:

$$H = -J \sum_{<i,j>} S_i S_j$$

- A standard model of statistical physics
- classical model
- Describes magnets:

$$\begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \downarrow \\ \uparrow & \uparrow & \downarrow & \uparrow & \downarrow \\ \uparrow & \downarrow & \uparrow & \uparrow & \downarrow \end{array} \quad \widehat{=}$$



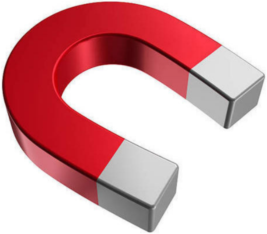
Ising model I

- Formula:

$$H = -J \sum_{<i,j>} S_i S_j$$

- A standard model of statistical physics
- classical model
- Describes magnets:

$$\begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \downarrow \\ \uparrow & \uparrow & \downarrow & \uparrow & \downarrow \\ \uparrow & \downarrow & \uparrow & \uparrow & \downarrow \end{array} \quad \widehat{=}$$



- System sizes: $100'000 \times 100'000$

Ising model II

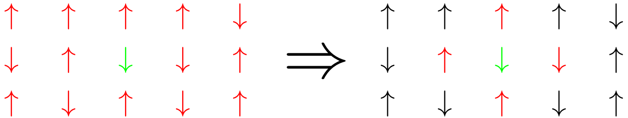
Nearest neighbour interactions only!

$$H = -J \sum_{<i,j>} S_i S_j$$

Ising model II

Nearest neighbour interactions only!

$$H = -J \sum_{<i,j>} S_i S_j$$



$$\begin{array}{ccccc} \uparrow & \uparrow & \uparrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \uparrow \\ \uparrow & \downarrow & \uparrow & \downarrow & \uparrow \end{array} \Longrightarrow \begin{array}{ccccc} \uparrow & \uparrow & \uparrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \uparrow \\ \uparrow & \downarrow & \uparrow & \downarrow & \uparrow \end{array}$$

Ising model II

Nearest neighbour interactions only!

$$H = -J \sum_{<i,j>} S_i S_j$$

$$\begin{array}{ccccc} \uparrow & \uparrow & \uparrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \uparrow \\ \uparrow & \downarrow & \uparrow & \downarrow & \uparrow \end{array} \Longrightarrow \begin{array}{ccccc} \uparrow & \uparrow & \uparrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \uparrow \\ \uparrow & \downarrow & \uparrow & \downarrow & \uparrow \end{array}$$

Calculation of energy: $\mathcal{O}\left(n^2\right) \Rightarrow \mathcal{O}\left(n\right)$

Metropolis algorithm

- **Goal:** Sample phase space

$$\begin{matrix} \uparrow & \downarrow & \downarrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \downarrow \\ \uparrow & \uparrow & \downarrow & \uparrow & \downarrow \\ \uparrow & \downarrow & \uparrow & \uparrow & \downarrow \end{matrix} \quad \overset{!}{\sim} \quad e^{-\frac{E}{k_B T}}$$

Metropolis algorithm

- **Goal:** Sample phase space

$$\begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \downarrow \\ \uparrow & \uparrow & \downarrow & \uparrow & \downarrow \\ \uparrow & \downarrow & \uparrow & \uparrow & \downarrow \end{array} \quad \overset{!}{\sim} \quad e^{-\frac{E}{k_B T}}$$

- **Algorithm:**

Metropolis algorithm

- **Goal:** Sample phase space

$$\begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \downarrow \\ \uparrow & \uparrow & \downarrow & \uparrow & \downarrow \\ \uparrow & \downarrow & \uparrow & \uparrow & \downarrow \end{array} \quad \overset{!}{\sim} \quad e^{-\frac{E}{k_B T}}$$

- **Algorithm:**
  1.) Propose new state: Random spin flips!

$$\begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \uparrow \\ \downarrow & \uparrow & \downarrow & \uparrow & \uparrow \\ \uparrow & \uparrow & \uparrow & \downarrow & \downarrow \end{array} \quad \Rightarrow \quad \begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \uparrow \\ \downarrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \uparrow & \uparrow & \uparrow & \downarrow & \downarrow \end{array}$$

Metropolis algorithm

- **Goal:** Sample phase space

$$\begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \downarrow \\ \downarrow & \uparrow & \downarrow & \downarrow & \downarrow \\ \uparrow & \uparrow & \downarrow & \uparrow & \downarrow \\ \uparrow & \downarrow & \uparrow & \uparrow & \downarrow \end{array} \quad \overset{!}{\sim} \quad e^{-\frac{E}{k_B T}}$$

- **Algorithm:**

  1.) Propose new state: Random spin flips!

$$\begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \uparrow \\ \downarrow & \uparrow & \downarrow & \uparrow & \uparrow \\ \uparrow & \uparrow & \uparrow & \downarrow & \downarrow \end{array} \quad \Longrightarrow \quad \begin{array}{ccccc} \uparrow & \downarrow & \downarrow & \uparrow & \uparrow \\ \downarrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \uparrow & \uparrow & \uparrow & \downarrow & \downarrow \end{array}$$

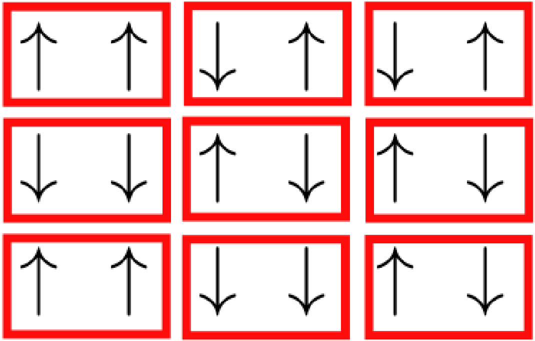
  2.) Accept the new configuration: $p_a = e^{-\frac{\Delta E}{k_B T}}$

Single core CPU: Data structure

- Multi-spin coding: 1 spin $\widehat{=}$ 1 bit

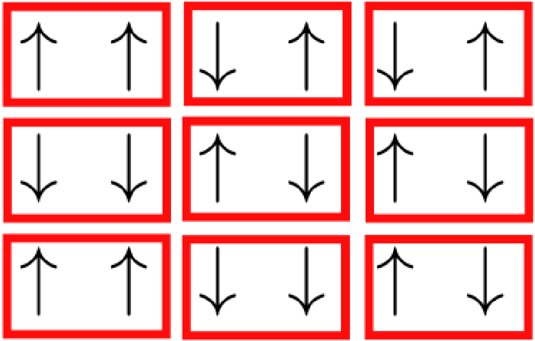
Single core CPU: Data structure

- Multi-spin coding: 1 spin $\widehat{=}$ 1 bit
- Group spins in groups of size 32 (int)



Single core CPU: Data structure

- Multi-spin coding: 1 spin $\widehat{=}$ 1 bit
- Group spins in groups of size 32 (int)



- 100’000 × 100’000 lattice: $\approx$ 1.2 GB

Single core CPU: Algorithm I

Type:	**# opposed**	**ΔE caused by flip of ↑**
$\begin{array}{ccc} & \uparrow & \\ \uparrow & \uparrow & \uparrow \\ & \uparrow & \end{array}$	0	$+8J$
$\begin{array}{ccc} & \downarrow & \\ \uparrow & \uparrow & \uparrow \\ & \uparrow & \end{array}$	1	$+4J$
$\begin{array}{ccc} & \downarrow & \\ \uparrow & \uparrow & \downarrow \\ & \uparrow & \end{array}$	2	0
$\begin{array}{ccc} & \downarrow & \\ \downarrow & \uparrow & \downarrow \\ & \uparrow & \end{array}$	3	$-4J$
$\begin{array}{ccc} & \downarrow & \\ \downarrow & \uparrow & \downarrow \\ & \downarrow & \end{array}$	4	$-8J$

Single core CPU: Algorithm I

Type:	# opposed	ΔE caused by flip of ↑
↑ ↑ ↑ ↑ ↑	0	$+8J$
↓ ↑ ↑ ↑ ↑	1	$+4J$
↓ ↑ ↑ ↓ ↑	2	0
↓ ↓ ↑ ↓ ↑	3	$-4J$
↓ ↓ ↑ ↓ ↓	4	$-8J$

$$H = -J \sum_{<i,j>} S_i S_j$$

Single core CPU: Algorithm I

Type:	# opposed	ΔE caused by flip of ↑
↑ ↑ ↑ ↑ ↑	0	$+8J$
↓ ↑ ↑ ↑ ↑	1	$+4J$
↓ ↑ ↑ ↓ ↑	2	0
↓ ↓ ↑ ↓ ↑	3	$-4J$
↓ ↓ ↑ ↓ ↓	4	$-8J$

$$H = -J \sum_{<i,j>} S_i S_j$$

$$p_a = e^{-\frac{\Delta E}{k_B T}}$$

Single core CPU: Algorithm I

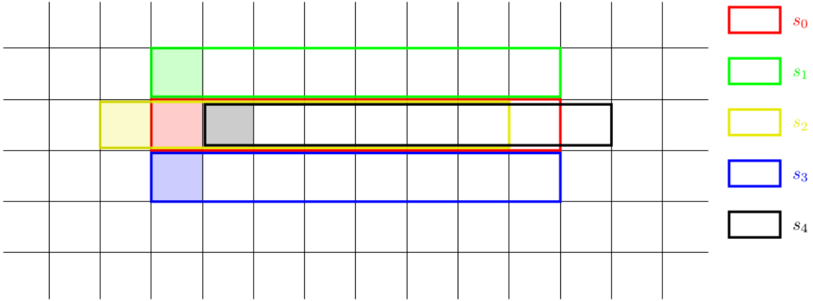
Type:	# opposed	ΔE caused by flip of $\uparrow$
$\begin{array}{ccc} & \uparrow & \\ \uparrow & \uparrow & \uparrow \\ & \uparrow & \end{array}$	0	$+8J$
$\begin{array}{ccc} & \downarrow & \\ \uparrow & \uparrow & \uparrow \\ & \uparrow & \end{array}$	1	$+4J$
$\begin{array}{ccc} & \downarrow & \\ \uparrow & \uparrow & \downarrow \\ & \uparrow & \end{array}$	2	0
$\begin{array}{ccc} & \downarrow & \\ \downarrow & \uparrow & \downarrow \\ & \uparrow & \end{array}$	3	$-4J$
$\begin{array}{ccc} & \downarrow & \\ \downarrow & \uparrow & \downarrow \\ & \downarrow & \end{array}$	4	$-8J$

$$H = -J \sum_{<i,j>} S_i S_j$$

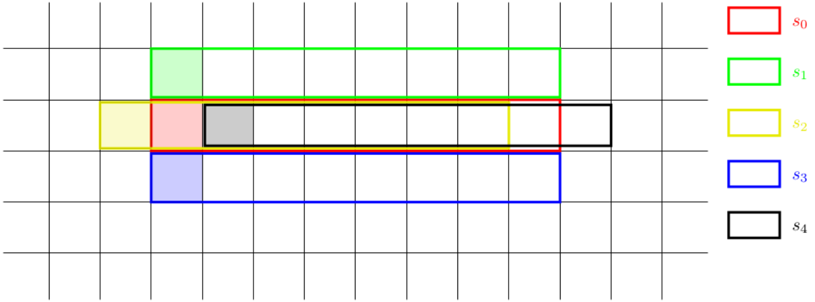
$$p_a = e^{-\frac{\Delta E}{k_B T}}$$

Count #opposed spins!

Single core CPU: Algorithm II

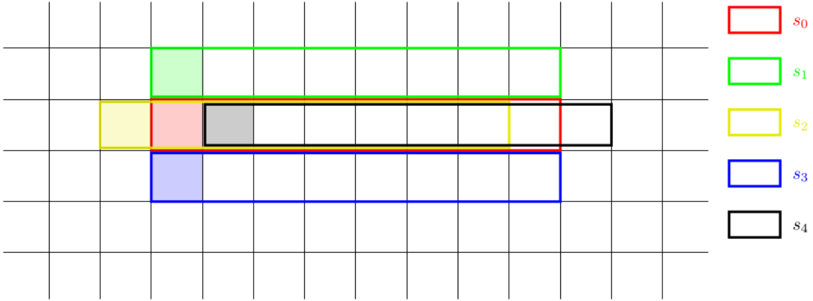


Single core CPU: Algorithm II



🟥 ≠ 🟩 ⇔ 🟥 XOR 🟩

Single core CPU: Algorithm II



🟥 ≠ 🟩 ⇔ 🟥 XOR 🟩

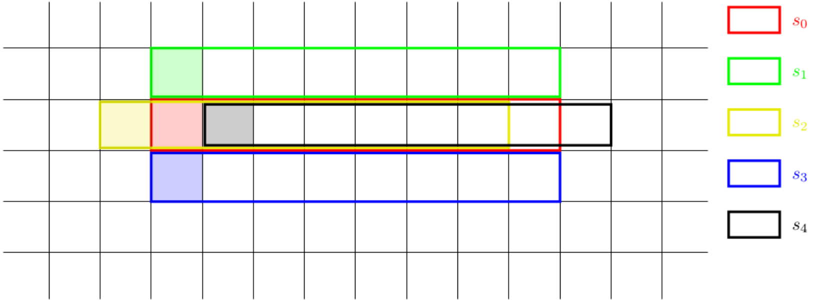
i_1 = 🟥 XOR 🟩

i_2 = 🟥 XOR 🟨

i_3 = 🟥 XOR 🟦

i_4 = 🟥 XOR ▭

Single core CPU: Algorithm II



🟥 $\neq$ 🟩 $\Leftrightarrow$ 🟥 XOR 🟩

i_1 = 🟥 XOR 🟩

i_2 = 🟥 XOR 🟨

i_3 = 🟥 XOR 🟦

i_4 = 🟥 XOR ⬛

Combine with acceptance probability:

$i_1 + i_2 + i_3 + i_4 + 2\mathrm{exp}_8 + \mathrm{exp}_4 \geq 2$

Single GPU implementation I

- Port CPU implementation

Single GPU implementation I

- Port CPU implementation
  ☹ Bad performance!

Single GPU implementation I

- Port CPU implementation
  ☹ Bad performance!
- GPU in a nutshell:

Single GPU implementation I

- Port CPU implementation

  ☹ Bad performance!
- GPU in a nutshell:
  - 100s of cores, blocks of 512

Single GPU implementation I

- Port CPU implementation

  ☹ Bad performance!

- GPU in a nutshell:
  - 100s of cores, blocks of 512
  - Memory:

Single GPU implementation I

- Port CPU implementation
  ☹ Bad performance!
- GPU in a nutshell:
  - 100s of cores, blocks of 512
  - Memory:
    Global memory:

    - big (≈ 4 GB)
    - slow

Single GPU implementation I

- Port CPU implementation
  ☹ Bad performance!
- GPU in a nutshell:
  - 100s of cores, blocks of 512
  - Memory:

Global memory:

- big (≈ 4 GB)
- slow

Shared memory:

- small (≈ 16 kB per block)
- fast

Single GPU implementation I

- Port CPU implementation
  ☹ Bad performance!
- GPU in a nutshell:
  - 100s of cores, blocks of 512
  - Memory:

Global memory:

- big (≈ 4 GB)
- slow

Shared memory:

- small (≈ 16 kB per block)
- fast

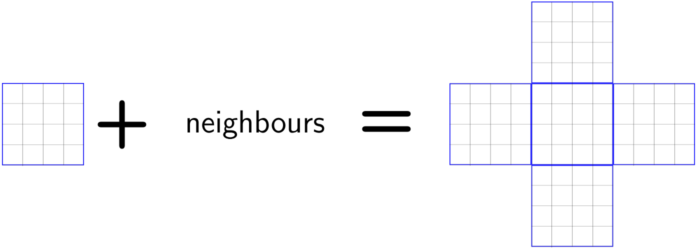
Reduce # accesses to global memory!

Single GPU implementation II

- Metaspins (4×4)
  $\rightarrow$ 1 metaspin $\widehat{=}$ 2 bytes = 1 unsigned short int
  $\rightarrow$ only 1 read per metaspin

Single GPU implementation II

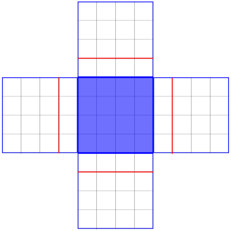
- Metaspins (4 × 4)
  → 1 metaspin $\widehat{=}$ 2 bytes = 1 unsigned short int
  → only 1 read per metaspin
- Read:



+ neighbours =

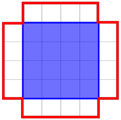
Single GPU implementation II

Global memory:



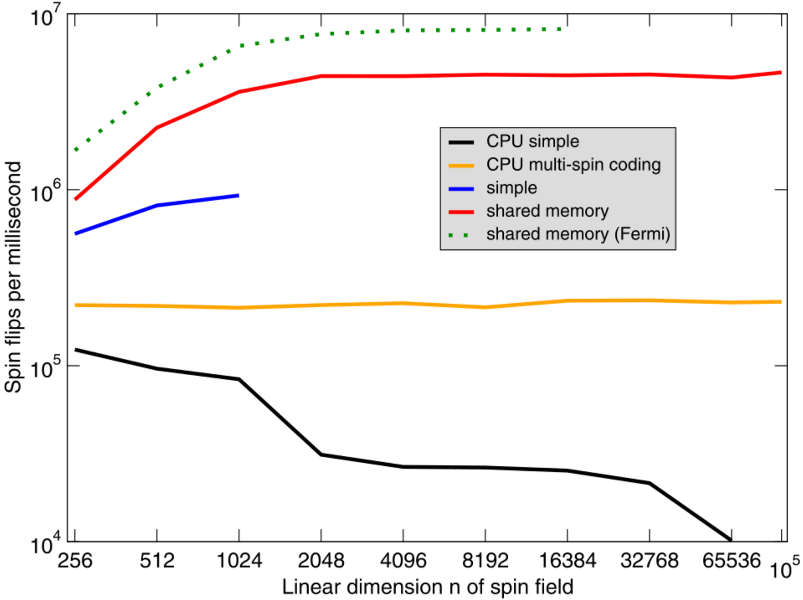
$\xrightarrow{5reads}$

Shared memory:



⇒ 5 reads to flip entire metaspin!

Results: CPU vs. GPU



Multi-GPU approach

- Single GPU:
  fast

Multi-GPU approach

- Single GPU:
  fast
  system size $\leq$ 4 GB ($\widehat{=}$ $100'000 \times 100'000$)

Multi-GPU approach

- Single GPU:
  fast
  system size $\leq$ 4 GB ($\widehat{=}$ $100'000 \times 100'000$)
- Idea: Distributed lattice!

Multi-GPU approach

- Single GPU:
  fast
  system size ≤ 4 GB ($\widehat{=}$ 100′000 $\times$ 100′000)
- Idea: Distributed lattice!
- Algorithm:
  1. **Copy neighbour borders to GPU**

Multi-GPU approach

- Single GPU:

  fast

  system size $\leq$ 4 GB ($\widehat{=}$ $100'000 \times 100'000$)
- Idea: Distributed lattice!
- Algorithm:
  1. **Copy neighbour borders to GPU**
  2. **Update own region on GPU**

Multi-GPU approach

- Single GPU:

  fast

  system size ≤ 4 GB ($\widehat{=} 100'000 \times 100'000$)

- Idea: Distributed lattice!
- Algorithm:
  1. **Copy neighbour borders to GPU**
  2. **Update own region on GPU**
  3. **Copy boundary spins to CPU**

Multi-GPU approach

- Single GPU:
  fast
  system size $\leq$ 4 GB ($\widehat{=}$ $100'000 \times 100'000$)
- Idea: Distributed lattice!
- Algorithm:
  1. **Copy neighbour borders to GPU**
  2. **Update own region on GPU**
  3. **Copy boundary spins to CPU**
  4. **Exchange boundary spins with other nodes**

Multi-GPU approach

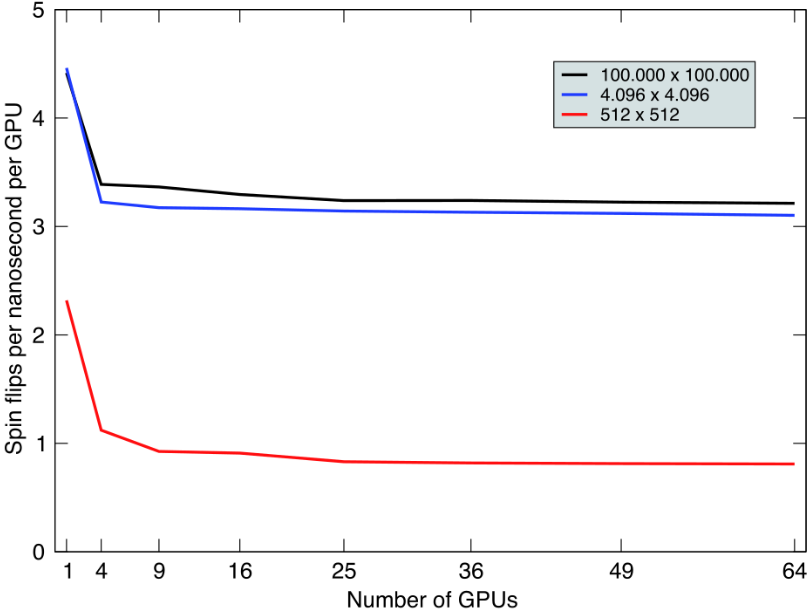
- Single GPU:

  fast

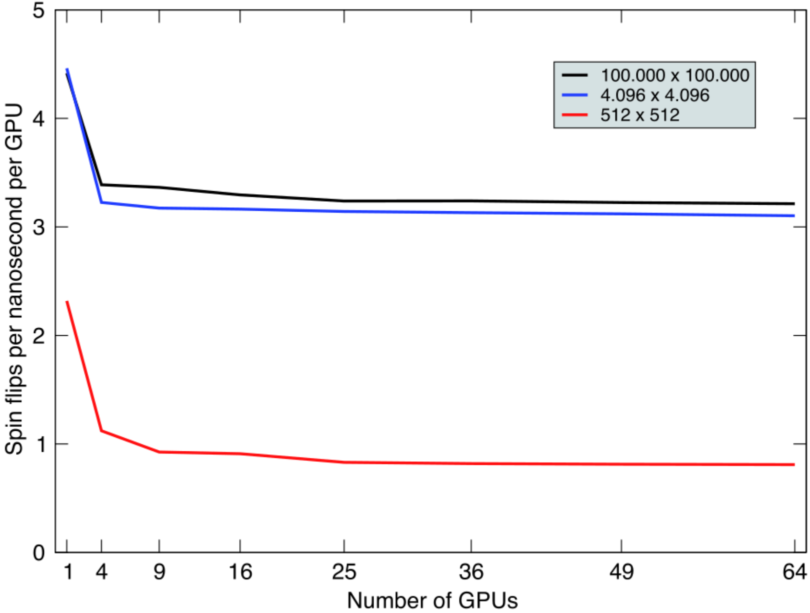
  system size $\leq$ 4 GB ($\hat{=}$ $100'000 \times 100'000$)

- Idea: Distributed lattice!
- Algorithm:
  1. **Copy neighbour borders to GPU**
  2. **Update own region on GPU**
  3. **Copy boundary spins to CPU**
  4. **Exchange boundary spins with other nodes**
  5. Repeat or finish

Multi-GPU: Results



Multi-GPU: Results



64 GPUs, $800'000 \times 800'000$: 3s

Conclusion

Conclusion

- No uncertainties in benchmark plots

Conclusion

- No uncertainties in benchmark plots
- Why this system?

Conclusion

- No uncertainties in benchmark plots
- Why this system?
- Approach generalisable?

Conclusion

- No uncertainties in benchmark plots
- Why this system?
- Approach generalisable?
- Not examined big systems (no weak scaling plot)

Conclusion

- No uncertainties in benchmark plots
- Why this system?
- Approach generalisable?
- Not examined big systems (no weak scaling plot)

Why this paper?!