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

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

March the 19th, 2015
Outlook

1. Cubic stencils
2. PRNGs
3. Multi-GPU and MPI
4. Results
5. Conclusions & Perspectives
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
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

1 Cubic stencils

2 PRNGs

3 Multi-GPU and MPI

4 Results

5 Conclusions & Perspectives
Standard checkerboard pattern

- Nearest-neighbours based problems in 3D

![Diagram of a cube with labeled vertices: $spz$, $spy$, $spx$, $smy$, $smx$, $smz$.

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

![Diagram showing sliced checkerboard pattern]

- 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**

![Diagram showing sliced checkerboard pattern]

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**

![Diagram showing sliced checkerboard pattern with periodic boundary conditions](image)

**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**

![Diagram of sliced checkerboard pattern](image)

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**

![Diagram of sliced checkerboard pattern](image)

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**

![Sliced scheme diagram]

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*

![Diagram showing the sliced checkerboard pattern with periodic boundary conditions]

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.
Variables belonging to one slice are **decoupled**

No sites parity involved
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

1. Cubic stencils
2. PRNGs
3. Multi-GPU and MPI
4. Results
5. Conclusions & Perspectives
Linear Congruential PRNG

- Lehmer-Park-Miller MINSTD

\[ R_{n+1} = (16807 \times R_n) \mod (2^{31} - 1), \]

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

**MINSTD overflow handling**

No 64 bits variables and no 'mod' involved

\[
\begin{align*}
\text{RNGT lo} &= 16807 \times (\text{seed} \& 0xffff); \\
\text{RNGT hi} &= 16807 \times (\text{seed} \gg 16); \\
\text{lo} &= (\text{hi} \& 0x7fff) \ll 16; \\
\text{lo} &= \text{hi} \gg 15; \\
\text{if(lo > 0x7fffffff) lo} &= 0x7fffffff; \\
\text{return lo;}
\end{align*}
\]
Linear Congruential PRNG

- Lehmer-Park-Miller MINSTD
  \[ R_{n+1} = (16807 \cdot R_n) \mod (2^{31} - 1), \]

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

- Easy implementation: one word state

- Coalescence is obtained by definition

  \[
  \begin{array}{ccccccc}
  \text{R[i]} & 0 & 1 & 2 & 3 & 4 & 5 & \cdots \\
  \hspace{1em} & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \\
  \end{array}
  \]

MINSTD overflow handling

No 64 bits variables and no 'mod' involved

\[
\begin{align*}
\text{RNGT lo} & = 16807*(\text{seed}\&0xffff); \\
\text{RNGT hi} & = 16807*(\text{seed}>>16); \\
\text{lo} & += (\text{hi}\&0x7fff)<<16; \\
\text{lo} & += \text{hi}>>15; \\
\text{if(lo} & > \quad 0x7fffffff) \text{lo} \quad -= \quad 0x7fffffff; \\
\text{return lo};
\end{align*}
\]
Linear Congruential PRNG

- Lehmer-Park-Miller MINSTD
  \[ R_{n+1} = (16807 \cdot R_n) \mod (2^{31} - 1) \]

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

- Easy implementation: one word state

- Coalescence is obtained by definition

\[ R[i] \]

\[ \begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & \ldots \\
\uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \\
\end{array} \]

- 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

```c
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

```c
lo -= ((-((lo&0x80000000)>>31))&0x7fffffff);
```
Linear Congruential PRNG

- Lehmer-Park-Miller MINSTD

\[ R_{n+1} = (16807 \cdot R_n) \mod (2^{31} - 1), \]

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

- Easy implementation: one word state

- Coalescence is obtained by definition

  \[
  \begin{array}{cccccc}
  R[i] & 0 & 1 & 2 & 3 & 4 & 5 & \ldots \\
  \hline
  \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\
  \end{array}
  \]

- 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

\[
\begin{align*}
\text{RNGT lo} & = 16807 \cdot (\text{seed} \& 0xffff); \\
\text{RNGT hi} & = 16807 \cdot (\text{seed} \gg 16); \\
\text{lo} & += (\text{hi} \& 0x7fff) \ll 16; \\
\text{lo} & += \text{hi} \gg 15; \\
\text{if(lo} & > 0x7ffffffff) \text{lo} -= 0x7ffffffff; \\
\text{return lo;}
\end{align*}
\]

Avoiding 'if' statement

We need to avoid branching

\[
\text{lo} -= (((\text{lo} \& 0x80000000) \gg 31)) \& 0x7ffffffff;
\]

Matteo Lulli - arXiv: 1114.0127 - lullimat.org
Lagged-Fibonacci-like PRNGs: Parisi-Rapuano

**Definition**

- Modified lagged-Fibonacci PRNG
  \[ \text{ira}[i] = \text{ira}[i - 24] + \text{ira}[i - 55] ; \]
  \[ \text{R} = \text{ira}[i]^\text{ira}[i - 61] ; \]
- At least 62 entries state

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

\[ \text{seed} = \text{ira}[(i - 24)\times\text{threads} + \text{threadId}] + \text{ira}[(i - 55)\times\text{threads} + \text{threadId}] ; \]
\[ \text{ira}[i\times\text{threads} + \text{threadId}] = \text{seed} ; \]
\[ \text{seed} \oplus \text{ira}[(i - 61)\times\text{threads} + \text{threadId}] ; \]

The method is general and it can be applied to the well-known Mersenne-Twister (MT19937)

Huge memory consumption \(\propto N \times N\) threads \(\times N\) state

Parisi-Rapuano: \(N_{PR} = 62\)

Mersenne Twister MT19937: \(N_{MT} = 624\)
Lagged-Fibonacci-like PRNGs: Parisi-Rapuano

Definition
- Modified lagged-Fibonacci PRNG
  \[ \text{irma}[i] = \text{irma}[i - 24] + \text{irma}[i - 55]; \]
  \[ R = \text{irma}[i] \cdot \text{irma}[i - 61]; \]
- 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
<table>
<thead>
<tr>
<th>Lagged-Fibonacci-like PRNGs: Parisi-Rapuano</th>
</tr>
</thead>
</table>

**Definition**
- Modified lagged-Fibonacci PRNG
  
  $\text{ira}[i] = \text{ira}[i - 24] + \text{ira}[i - 55];$
  
  $R = \text{ira}[i] \oplus \text{ira}[i - 61];$

- 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

$\text{seed} = \text{ira}[(i - 24) \times \text{threads} + \text{threadId}]$

$+ \text{ira}[(i - 55) \times \text{threads} + \text{threadId}]$

$\text{ira}[i \times \text{threads} + \text{threadId}] = \text{seed};$

$\text{seed} \oplus = \text{ira}[(i - 61) \times \text{threads} + \text{threadId}];$

- The method is general and it can be applied to the well-known Mersenne-Twister (MT19937)
- Huge memory consumption $\propto N_{\text{threads}} \times N_{\text{state}}$
- 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
- $t_{urand}^{INIT}$: 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
- $t_{urand}^{INIT}$: time needed to initialize the PRNG using /dev/urandom
- $t_{GEN}$: is the generation time

![cuRand Host API](image1)

![My Host API](image2)
Benchmarks: host API

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

![Graph showing benchmark results for cuRand Host API and My Host API]
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**

![Graph showing MT19937 test results.](image)
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
Outline for section 3

1. Cubic stencils
2. PRNGs
3. Multi-GPU and MPI
4. Results
5. Conclusions & Perspectives
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

Stream 0

- Boundary
- D2H
- MPI
- H2D

Stream 1

- Bulk

$t$
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

1. Cubic stencils
2. PRNGs
3. Multi-GPU and MPI
4. Results
5. Conclusions & Perspectives
Three-dimensional Edwards-Anderson model

- Three-dimensional Spin Glass model

\[ H = - \sum_{\langle ik \rangle} J_{ik} \sigma_i \sigma_k, \quad \sigma_i \in \{-1, +1\}, \quad J_{ik} \in \{-1, +1\} \]

- Bimodal disorder

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

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

\[ J_{ik} = -1 \rightarrow 1 \]
\[ J_{ik} = +1 \rightarrow 0 \]
\[ \sigma_i = +1 \rightarrow 1 \]
\[ \sigma_i = -1 \rightarrow 0 \]
\[ e_{ik} = \sigma_i \hat{\sigma}_k \hat{J}_{ik} \]
The smaller the lattice the more disorder realizations are needed to attain saturation.

Let $k$ be the number of coded disorder realizations and $\text{psFlip}_{n,\text{mstd}}$

\[
\text{psFlip}_{n,\text{mstd}}(L, k) = t_{sw} \cdot n \cdot (32 \cdot k \cdot 4 \cdot L^3)^{-1} ,
\]
Single- & Multi-GPU Results

- Values for psFlip<sub>1,mstd</sub> for different values of \( L = 4k \)
- Sliced (red), Standard (green), Standard-Bitwise (blue)
Single- & Multi-GPU Results

- Values for $\text{psFlip}_{1,m\text{std}}$ for different values of $L = 4k$
- Sliced (red), Standard (green), Standard-Bitwise (blue)
- Bandwidth for different values of $L = 4k$
- Sliced (red), Standard (green), Standard-Bitwise (blue)
Single- & Multi-GPU Results

- Values for \( psFlip_{1,x} \) 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

\[ \eta_{SC} = \frac{psFlip_{1,x}}{psFlip_{N,x}} \]

[Graph showing strong-scaling efficiency with various GPU counts (NGPU = 2, 4, 8).]
Single- & Multi-GPU Results (CSCS Piz Daint)

The diagram shows the performance of a cubic lattice algorithm as a function of lattice size $L$ for different numbers of GPUs ($N_{GPU}$).

- $N_{GPU} = 1$
- $N_{GPU} = 2$
- $N_{GPU} = 4$
- $N_{GPU} = 8$
- $N_{GPU} = 16$
- $N_{GPU} = 32$

The $y$-axis represents the number of operations per flip $N/mstd/N$ on a logarithmic scale, while the $x$-axis shows the lattice size $L$ in linear scale.
Single- & Multi-GPU Results (CSCS Piz Daint)

- Data scaling

\[
\frac{L}{N} ps\text{Flip}_{N,x}(L, k) \propto t_{sw} \left( \frac{L}{N} \right)^{-2}, \quad x = \frac{L}{N}.
\]
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)

<table>
<thead>
<tr>
<th></th>
<th>$T_c$</th>
<th>$\nu$</th>
<th>$\eta$</th>
<th>$\omega$</th>
<th>$z$</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janus 2013</td>
<td>1.1019(29)</td>
<td>2.562(42)</td>
<td>−0.3900(36)</td>
<td>1.12(10)</td>
<td></td>
<td>75 (FPGA)</td>
</tr>
<tr>
<td>Our Work¹</td>
<td>1.099(5)</td>
<td>2.47(10)</td>
<td>−0.39(1)</td>
<td>1.3(2)</td>
<td>6.80(15)</td>
<td>3.1 (GPU)</td>
</tr>
<tr>
<td>Haus. et al. 2008</td>
<td>1.109(10)</td>
<td>2.45(15)</td>
<td>−0.375(10)</td>
<td>1.0(1)</td>
<td></td>
<td>40 (CPU)</td>
</tr>
</tbody>
</table>

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

¹M. Lulli, G. Parisi & A. Pelissetto in preparation
Outline for section 5

1. Cubic stencils
2. PRNGs
3. Multi-GPU and MPI
4. Results
5. Conclusions & Perspectives
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 $psFlip_{1,mstd}$ for different values of $L = 2n$
- Sliced (red), Standard (green), Standard-Bitwise (blue)
Even and Odd subsequences for $L = 2n$

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

What is happening?

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

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

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

![Graph of branch/\(k\) vs \(L\)](image_url)
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)

\[ J_{ik} = -1 \Rightarrow 1 \]
\[ J_{ik} = +1 \Rightarrow 0 \]
\[ \sigma_i = +1 \Rightarrow 1 \]
\[ \sigma_i = -1 \Rightarrow 0 \]
\[ e_{ik} = \sigma_i \sim \sigma_k \sim J_{ik} \]

<table>
<thead>
<tr>
<th>e_{ik0}</th>
<th>\sigma_{i0}</th>
<th>\sigma_{k0}</th>
<th>J_{ik0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_{ik1}</td>
<td>\sigma_{i1}</td>
<td>\sigma_{k1}</td>
<td>J_{ik1}</td>
</tr>
<tr>
<td>e_{ik2}</td>
<td>\sigma_{i2}</td>
<td>\sigma_{k2}</td>
<td>J_{ik2}</td>
</tr>
</tbody>
</table>

\[ \sum_{\langle ik \rangle} = s_{2i0} s_{1i0} s_{0i0} \]

<table>
<thead>
<tr>
<th>e_{ik0}</th>
<th>s_{2i0}</th>
<th>s_{1i0}</th>
<th>s_{0i0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_{ik1}</td>
<td>s_{2i1}</td>
<td>s_{1i1}</td>
<td>s_{0i1}</td>
</tr>
<tr>
<td>e_{ik2}</td>
<td>s_{2i2}</td>
<td>s_{1i2}</td>
<td>s_{0i2}</td>
</tr>
</tbody>
</table>
More on MultiSpin-Coding

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

\[
J_{ik} = -1 \rightarrow 1 \\
J_{ik} = +1 \rightarrow 0 \\
\sigma_i = +1 \rightarrow 1 \\
\sigma_i = -1 \rightarrow 0 \\
e_{ik} = \sigma_i \sigma_k J_{ik}
\]

\[
\sum_{\langle ik \rangle} = s_{i0} s_{i1} s_{i2} \\
s_{1i0} s_{1i1} s_{1i2} \\
s_{2i0} s_{2i1} s_{2i2}
\]

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)

\[
J_{ik} = -1 \rightarrow 1 \\
J_{ik} = +1 \rightarrow 0 \\
\sigma_i = +1 \rightarrow 1 \\
\sigma_i = -1 \rightarrow 0 \\
e_{ik} = \sigma_i \wedge \sigma_k \wedge J_{ik}
\]

Metropolis Dynamics - Avoiding 'if' statements

<table>
<thead>
<tr>
<th>State</th>
<th>( \Delta E )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0, 0)</td>
<td>-12</td>
</tr>
<tr>
<td>(0, 0, 1)</td>
<td>-8</td>
</tr>
<tr>
<td>(0, 1, 0)</td>
<td>-4</td>
</tr>
<tr>
<td>(0, 1, 1)</td>
<td>0</td>
</tr>
<tr>
<td>(1, 0, 0)</td>
<td>4</td>
</tr>
<tr>
<td>(1, 0, 1)</td>
<td>8</td>
</tr>
<tr>
<td>(1, 1, 0)</td>
<td>12</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
</tr>
</tbody>
</table>
More on MultiSpin-Coding

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

\[ J_{ik} = -1 \rightarrow 1 \]
\[ J_{ik} = +1 \rightarrow 0 \]
\[ \sigma_i = +1 \rightarrow 1 \]
\[ \sigma_i = -1 \rightarrow 0 \]
\[ e_{ik} = \sigma_i \hat{o} \sigma_k \hat{o} J_{ik} \]

Metropolis Dynamics - Avoiding 'if' statements

Normalized transition probabilities & random number comparison

\[ R_{max} \exp(-\beta \Delta E) = \text{EXP12, EXP8, EXP4} \]
\[ \text{cond12} = -(R < \text{EXP12}); \quad \text{cond8} = -(R < \text{EXP8}); \quad \text{cond4} = -(R < \text{EXP4}); \]

Spin flip: \[ \text{spin} = \text{mask} \hat{o} \text{spin}; \]
\[ \text{mask} = \text{cond12} \mid (\neg \text{sum2}) \]
\[ \mid ((\text{sum2} \& (\text{sum2} \neg \text{sum1})) \& (\text{cond8} \mid (\text{cond4} \& (\neg \text{sum0}))))]; \]
### Sliced Addressing

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

$$\begin{align*}
\text{smz} &= i + (\text{SM}(z - 1, d_{hL}) - z) \times d_A; \\
\text{spy} &= \text{smz} + (\text{SP}(y + 1, d_L) - y) \times d_L; \\
\text{smy} &= i + (\text{SM}(y - 1, d_L) - y) \times d_L; \\
\text{smx} &= \text{spy} - x + \text{SM}(x - 1, d_L); \\
\text{spx} &= \text{smy} - x + \text{SP}(x + 1, d_L);
\end{align*}$$

### Standard Addressing

$$x = i \% d_{hL}; \ y = (i/d_{hL})\%d_L; \ z = i/d_{hA};$$

$$\begin{align*}
\text{par} &= (y \wedge z) \& 1; \\
\text{spx} &= i - x + \text{SP}(x + 1 - (\text{par} \wedge 1), d_{hL}); \\
\text{smx} &= i - x + \text{SM}(x - 1 + \text{par}, d_{hL}); \\
\text{spy} &= i + (\text{SP}(y + 1, d_L) - y) \times d_{hL}; \\
\text{smy} &= i + (\text{SM}(y - 1, d_L) - y) \times d_{hL}; \\
\text{spz} &= i + (\text{SP}(z + 1, d_L) - z) \times d_{hA}; \\
\text{smz} &= i + (\text{SM}(z - 1, d_L) - z) \times d_{hA};
\end{align*}$$

For the sliced scheme, $$\text{spz} = i$$, and parity is not involved.