Monte Carlo Simulations of Spin Glass Systems on Multi-core Engines

Sebastiano Fabio Schifano

[1] University of Ferrara and INFN-Ferrara

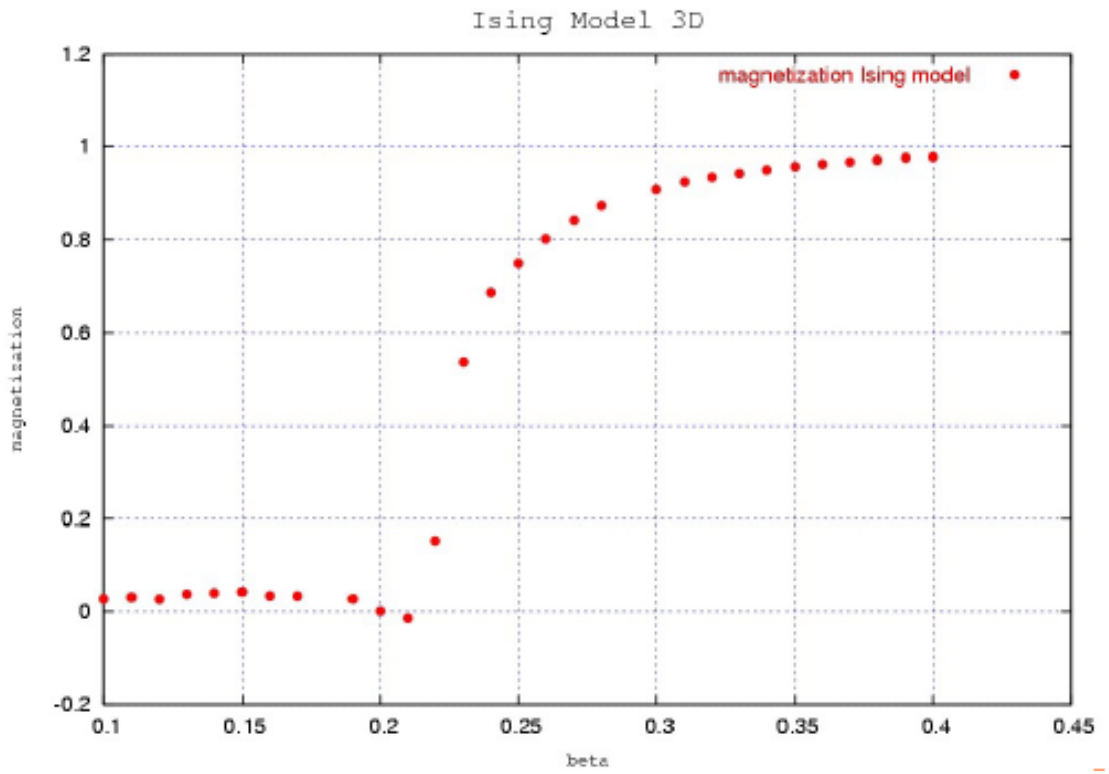
CCR 2010

January, 25-27 2010, Napoli - Italy

Talk Outline

Spin-Glass

The Spin-glass is a statistic model to study some behaviours of complex macroscopic systems like **disordered magnetic materials**.



An apparently trivial generalization of ferromagnet.

Spin-Glass Models

Ising Model

$$E(\{S\}) = -J \sum_{\langle ij \rangle} s_i \cdot s_j, \qquad J > 0, \;\; s_i, s_j \in \{-1, +1\}$$

Edwards Anderson Model (Binary)

$$E(\{S\}) = \sum_{\langle ij \rangle} J_{ij} \cdot s_i \cdot s_j, \qquad J_{ij}, s_i, s_j \in \{-1, +1\}$$

Edwards Anderson Model (Gaussian)

$$E(\{S\}) = \sum_{\langle ij \rangle} J_{ij} \cdot s_i \cdot s_j, \qquad J_{ij} \in \mathbb{R}, s_i, s_j \in \{-1, +1\}$$

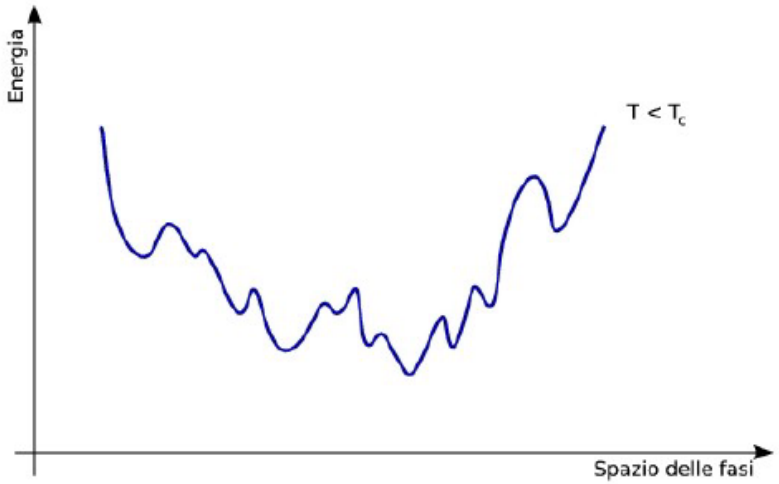
Heisenberg Model

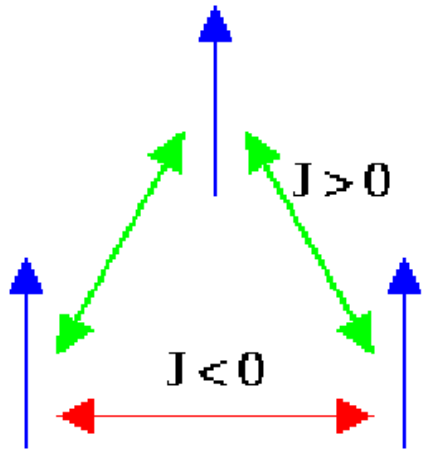
$$E(\{S\}) = \sum_{\langle ij \rangle} J_{ij} \cdot \vec{s_i} \cdot \vec{s_j} \qquad J_{ij} \in \mathbb{R}, s_i, s_j \in \mathbb{R}^3$$

EA Simulation is Computer Challenging

Frustration effects make the energy function landscape corrugated:

$$E(\{S\}) = -\sum_{\langle ij \rangle} J_{ij} s_i s_j, \quad s_i, s_j \in \{+1, -1\}, \quad J_{ij} \in \{+1, -1\}$$





Frustation effects make the approach to the thermal equilibrium a slowly computing process.

The Edwards-Anderson (EA) Model

The system variables are spins (± 1), arranged in D-dimensional (usually D=3) lattice of size L .

- Spins s_i interacts only with its nearest neighbours
- Pair of spins (s_i, s_j) share a coupling term J_{ij}
- The energy of a configuration $\{S\}$ is computed as:

$$E(\{S\}) = \sum_{\langle ij \rangle} J_{ij} s_i s_j$$

- Each configuration $\{S\}$ has a probability given by the **Boltzmann** factor:

$$P(\{S\}) \propto e^{\frac{-E(\{S\})}{kT}}$$

- Average of macroscopic observable (**magnetization**) are defined as:

$$\langle M \rangle = \sum_{\{S\}} M(\{S\}) P(\{S\}) \quad \text{where} \quad M(\{S\}) = \sum_i s_i$$

The EA model and Monte Carlo Algorithms

- A lattice size $L = 80$ has $80^3 \Longrightarrow 2^{80^3}$ different configurations
- pratically impossible to manage to generate all configurations
- not all configurations have the same probability and are equally important

Monte Carlo algorithms, like the Metropolis and Heatbath, are adopted:

- configurations are generated according to their probability
- observables average are computed as **unweighted** sums of Monte Carlo generated configurations:

$$\langle M \rangle \sim \sum_i M(\{S_i^{\mathrm{MC}}\})$$

Metropolis Algorithm

Require: set of $\{S\}$ and $\{J\}$

1: **loop** // loop on Monte Carlo steps
2: **for all** $s_i \in \{S\}$ **do**
3: $s'_i = (s_i == 1)\ ?\ -1\ :\ 1$ // flip tentatively value of s_i
4: $\Delta E = \sum_{\langle ij \rangle}(J_{ij} \cdot s'_i \cdot s_j) - (J_{ij} \cdot s_i \cdot s_j)$ // compute energy change
5: **if** $\Delta E \leq 0$ **then**
6: $s_i = s'_i$ // accept new value of s_i
7: **else**
8: $\rho =$ rnd() // compute a random number $0 \leq \rho \leq 1,\ \rho \in \mathbb{Q}$
9: **if** $\rho < e^{-\beta \Delta E}$ **then** // $\beta = 1/T,\ T =$ Temperature
10: $s_i = s_i$‘ // accept new value of s_i
11: **end if**
12: **end if**
13: **end for**
14: **end loop**

Spin-glass is Computer Challenging

State-of-the-art simulation-campaign bringing a lattice $L = 48 \ldots 80$ to the thermal equilibrium are organized as following:

- *Hundreds* (*Thousands*) systems, **samples**, with different initial values of spins and couplings are simulated.
- for each sample the simulation is repeated *2-4* times with different initial spin-values (coupling values kept fixed), **replicas**.
- Each simulation may requires $10^{12} \ldots 10^{13}$ Monte Carlo update steps.

$$80^3 \times 10\,\text{ns} \times 10^{11}\ \text{MC-steps} \approx 16\ \text{years}$$

Exploiting parallelism:

- **Samples** and **replicas** can be simulated in parallel since no information is exchanged between them.
- Right trade-off between **internal-** and **external-**parallelism is necessary.

Parallel Simulation of Spin Glass

Several levels of parallelism can be exploited in Monte Carlo Spin Glass simulations.

- The lattice can be divided in a *checkerboard* scheme: alghorithm is first applied to all **white** spins, and then to all **blacks** (order is irrelevant).
- SIMD instructions can be used to update up to V (white or black) spins in parallel (**internal parallelism**).
- The lattice can be divided in several sub-lattices and allocated to different *cores*. Boundaries need to be updated after updating the bulk (**internal parallelism**).
- Several lattices (**samples** or **replicas**) can be simulated in parallel using **multispin**-coding approach (**external parallelism**)

Multispin Encoding (1)

Multispin encoding allows to simulate several systems in parallel.

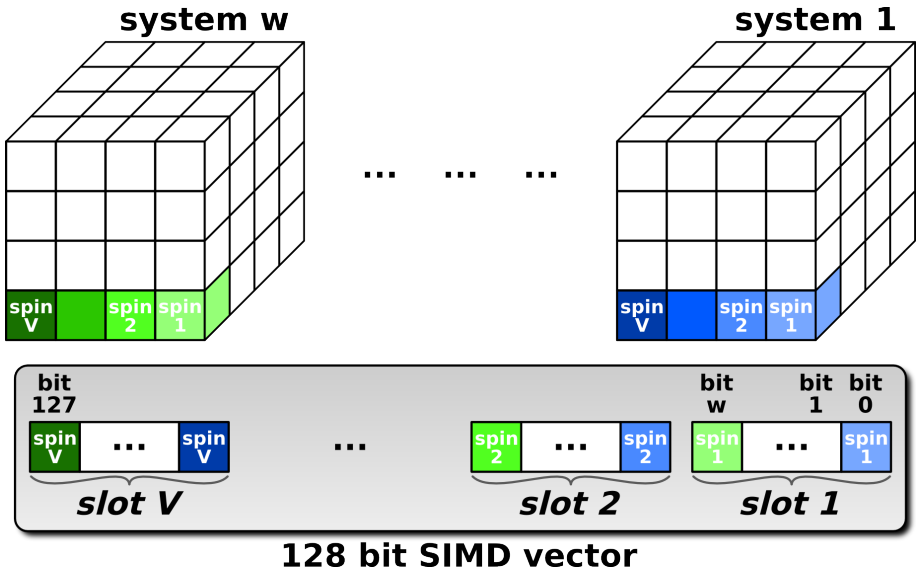
Assuming to run simulation on a k-bit architecture ($k = 32, 64, 128$):

- spins and couplings are represented by binary values $\{0, 1\}$
- a k-bit architectural word hosts k-spins of k different systems
- Metropolis update procedure can be bit-wise coded (no conditional statements, only bit-wise operations)

Require: ρ pseudo-random number
Require: $\psi = \text{int}\,(-(1/4\beta)\log\rho)$, encoded on two bits
Require: $\eta = (\text{ not } X_i)$, encoded on two bits
$c_1 = (\psi[0] \text{ and } \eta[0])$
$c_2 = (\psi[1] \text{ and } \eta[1]) \text{ or } ((\psi[1] \text{ or } \eta[1]) \text{ and } c_1)$
$s_i' = s_i \text{ xor } (c_2 \text{ or not } X_i[2])$ // update value of spin s_i

Multispin Encoding (2)

We enhanced **multispin encoding** approach combining it with SIMD-instructions to exploit both **internal-** and **external-**parallelism.



- the 128-bit SIMD-word is divided in $V = 2 \ldots 64$ slots
- each slot hosts one spin-values of a system
- each slot hosts w spin-values of w different lattices.

V = internal-parallelism degree, w = external-parallelism degree.

Spin Glass: Random Number Generation

At each MC-step V (pseudo-)random numbers are needed.
Same random value can be shared among the w lattice-replicas.

The Parisi-Rapuano generator is a popular choise for Spin Glass simulations:

WHEEL[k] = WHEEL[k-24] + WHEEL[k-55]

ρ = WHEEL[k] $\oplus$ WHEEL[k-61]

- WHEEL is an array of unsigned integer
- SIMD instructions can be used to generate several random numbers in parallel.

NVIDIA GT200 Specs

Processor:

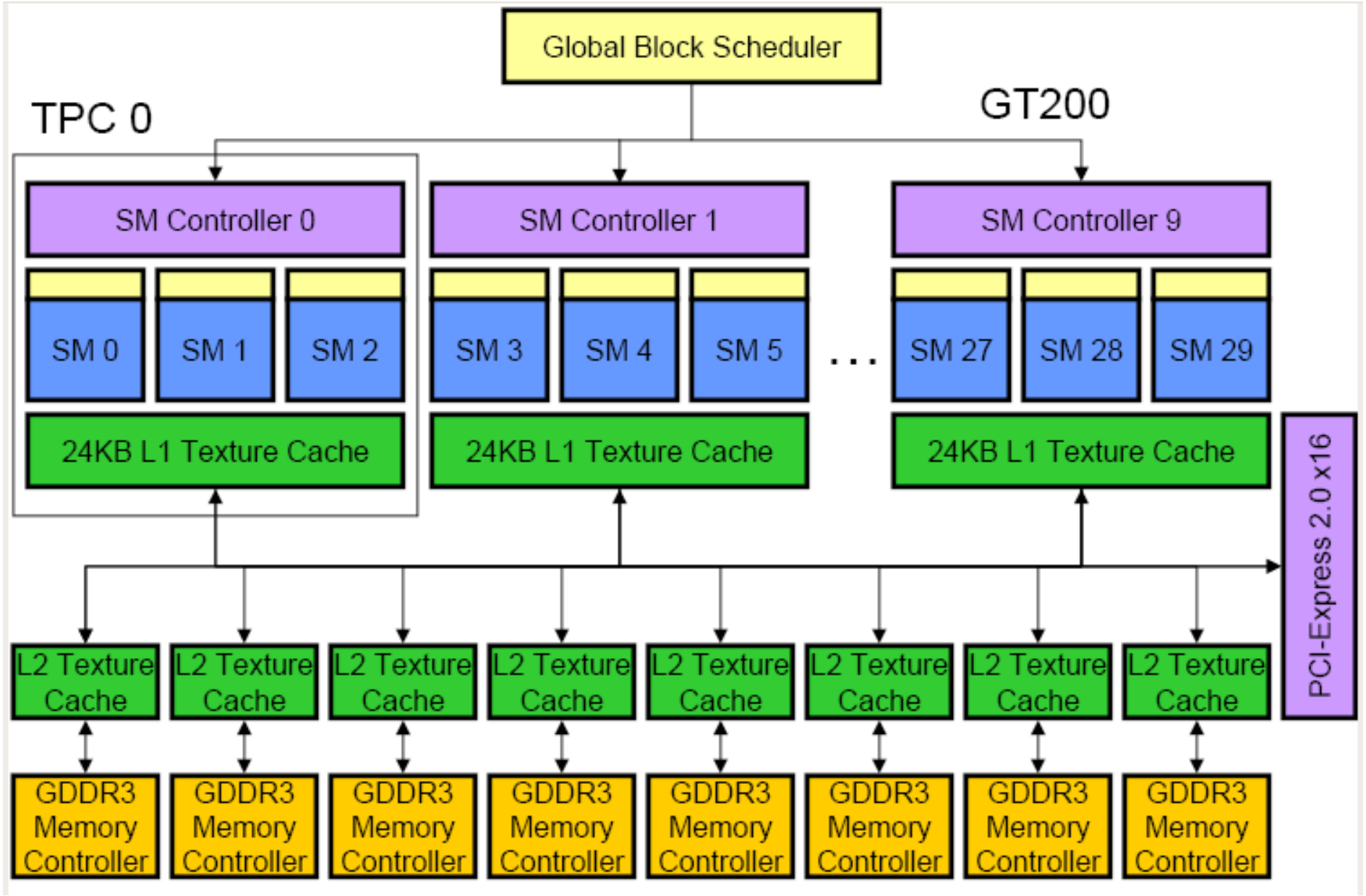
- 1.3 GHz, **936 Gflops SP**, 78 Gflops DP
- 10 **Texture Processor Cluster** (TPC)
- 1 TPC includes 3 **Streaming Multiprocessor** (SM) + 1 Texture Memory
- 1 SM include 8 **Streaming Processor** (SP)
  loosely corresponding to a modern CPU-core with 8-way SIMD computing-unit
- a total of $10 \times 3 \times 8 = 240$ **threads**

Memory:

- 4GB Global Memory, 512-bit, 102.4 GB/s, 400 . . . 600 cycles of latency
- Constant memory 64 KB (RO)
- Texture memory 256 KB (RO, two dimensional locality)

160 Watt (typical, w/Memory), $<$ 6 Gflops/Watt (SP, w/MUL)

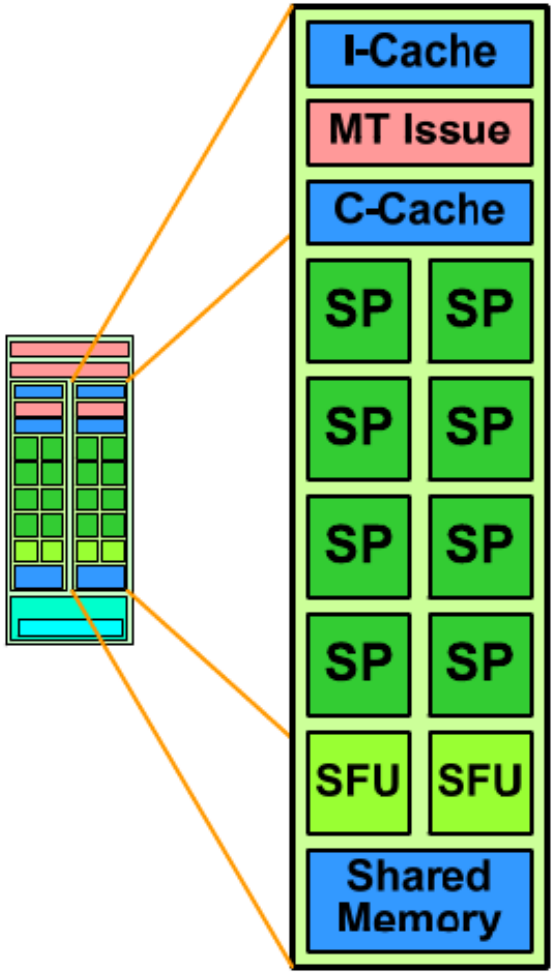
Architecture of GP-GPU: NVIDIA GT200



NVIDIA GT200 SM Specs

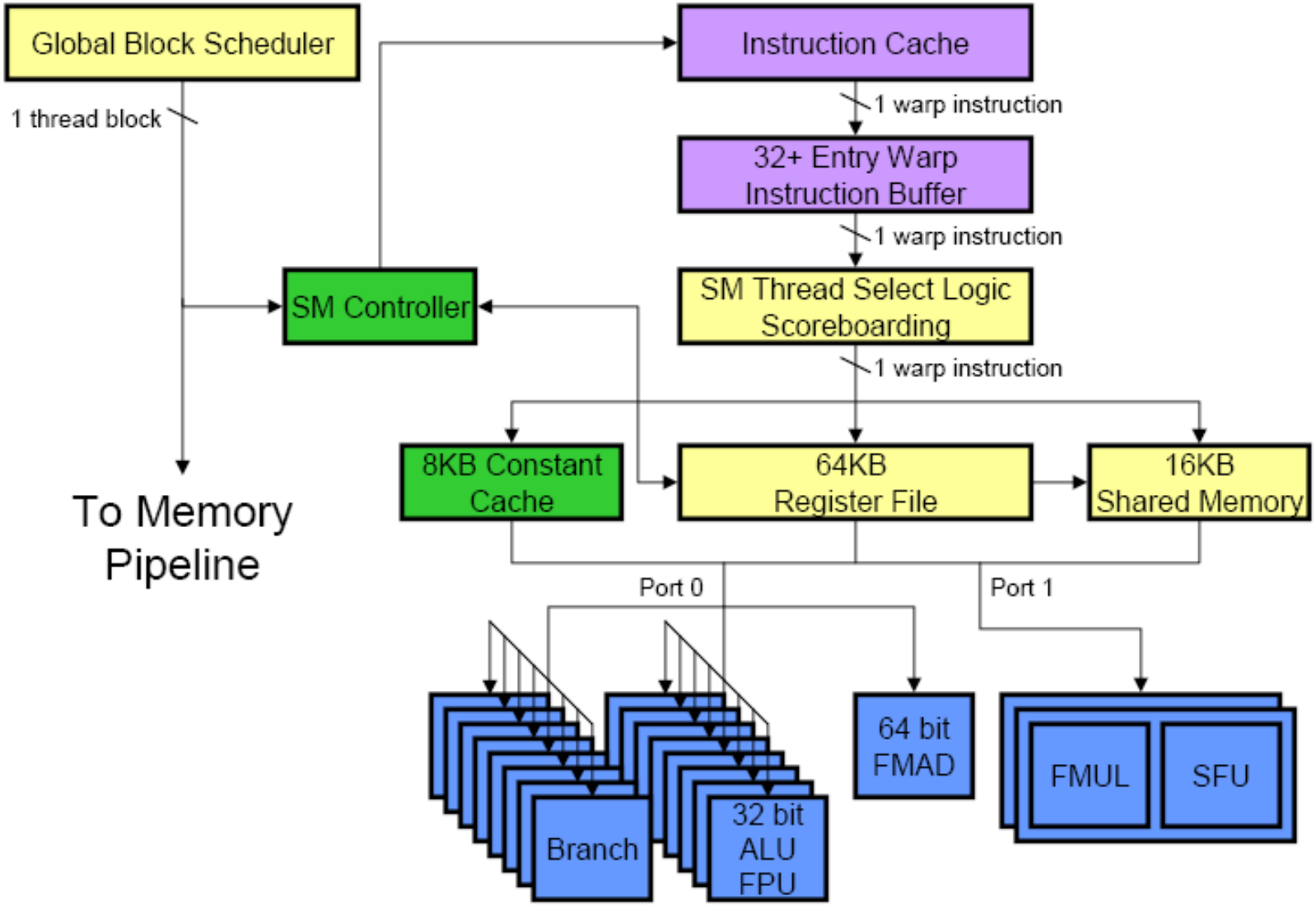
Each SM includes:

- 64KB register file (2KB for each SP)
- 16KB shared memory
- 8KB constant cache
- 8 32-bit ALU/FPU
- 1 64-bit FMAD
- 8 branch units

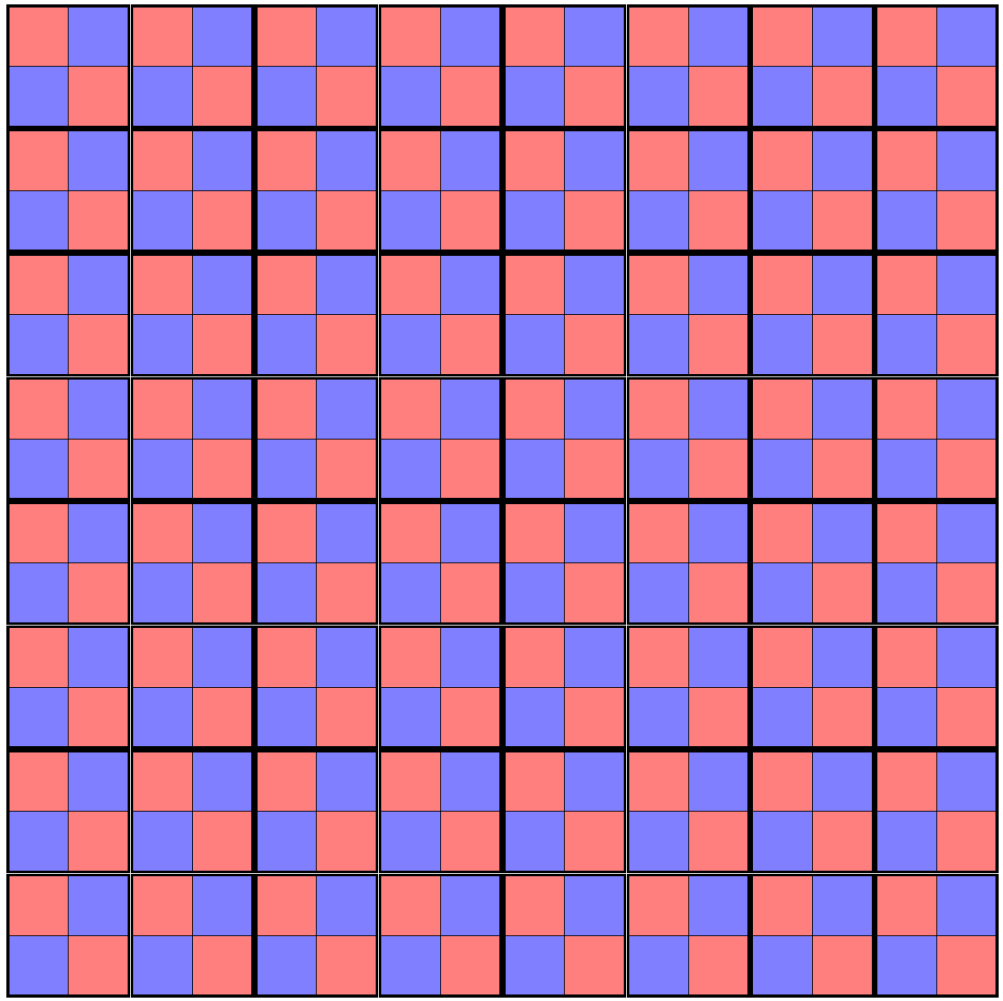


1 SM executes in parallel up to 8 thread and manages up to 1K threads

The SM Architecture

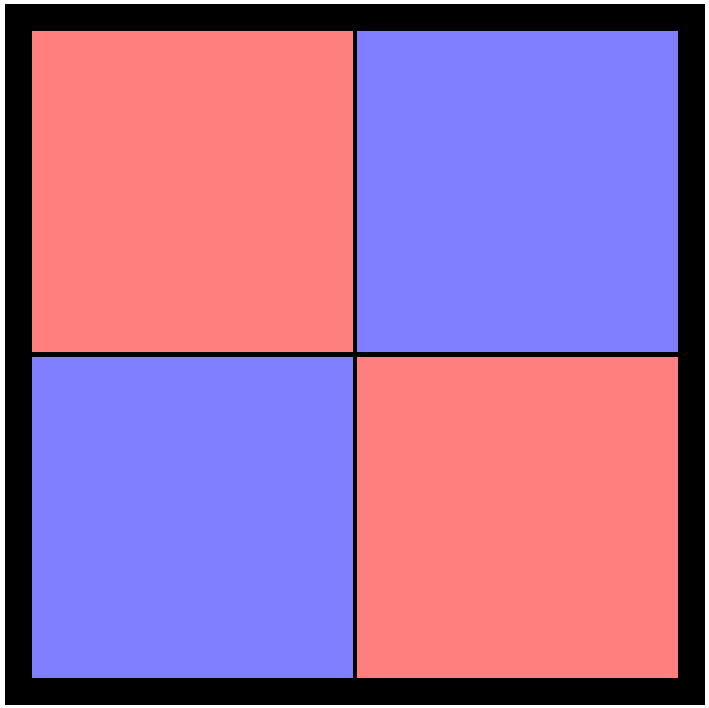


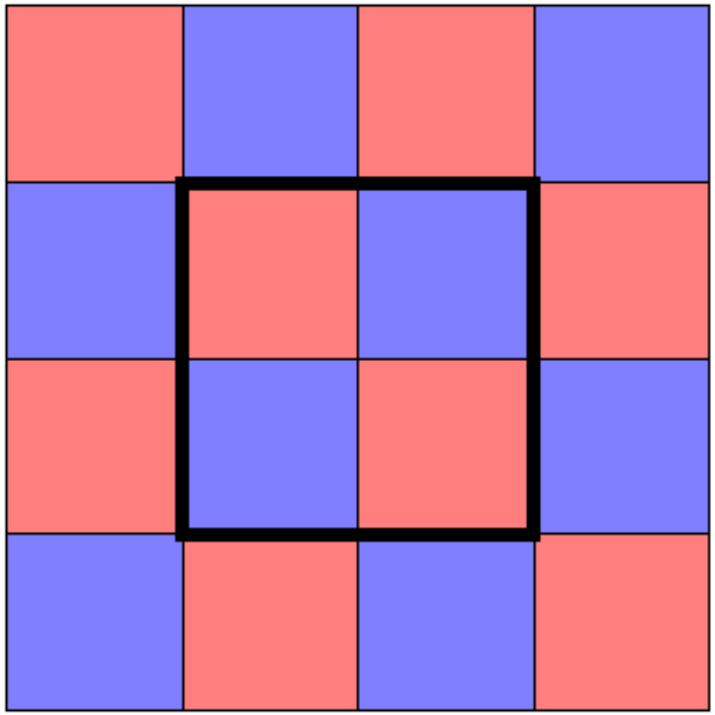
Spin Glass Simulation on GP-GPU



A 3D-lattice is divided in a 2D-grid of $L/2 \times L/2 \times L$ sub-lattices

Spin Glass Simulation on GP-GPU





- Each sub-lattice is a 3D-grid of $2 \times 2 \times L$ points.
- Each block loads a sub-lattice of $4 \times 4 \times L$ points.

Spin Glass Simulation on GP-GPU

- lattice is allocated to GPU memory replicating border surfaces
- lattice is divided in a 2D-grid $(L/2 \times L/2 \times L)$ sub-lattices $\Longrightarrow$ many blocks to hide memory access latency
- each block updates a sub-lattice of $(2 \times 2 \times L)$ points $\Longrightarrow$ many *warps* to allow memory-coalescing
- each block is configured as a grid of $(2 \times 2 \times L)/2$ threads
- each block loads a sub-lattice of $(4 \times 4 \times L)$ points, the bulk plus neighbors
- each thread applies the MC-step to a single point

Spin Glass Simulation on GP-GPU

The code executed by the host:

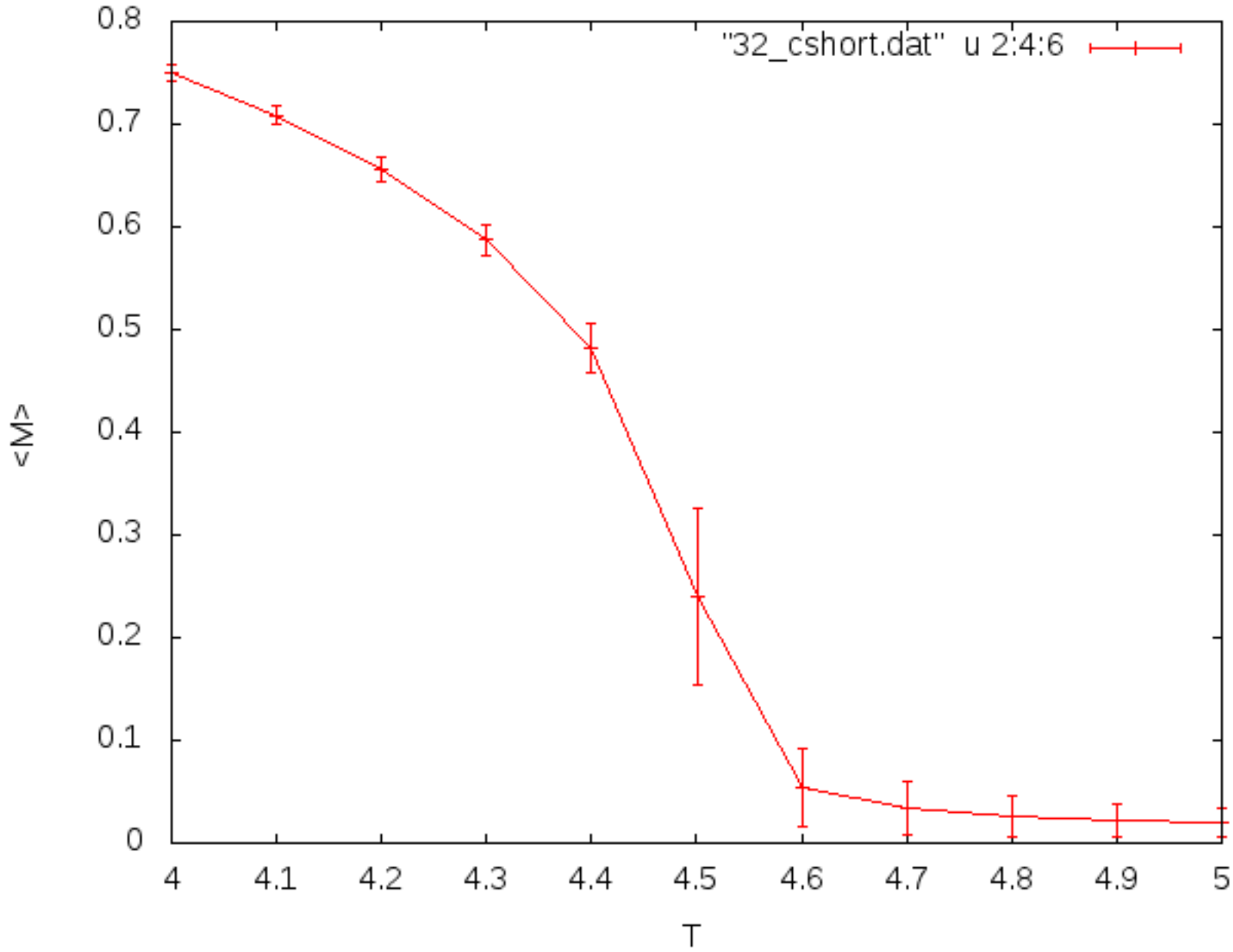
```
1: loop   // loop on Monte Carlo steps
2:     MCupdate(red)
3:     updateBorder()
4:     MCupdate(blu)
5:     updateBorder()
6: end loop
```

- *MCupdate()* is invoked on a grid of $L/2 \times L/2 \times L$ blocks each one configured as $(L/2 \times L/2 \times L)/2$ threads
- *updateBorder()* is invoked on a grid of L blocks of L threads

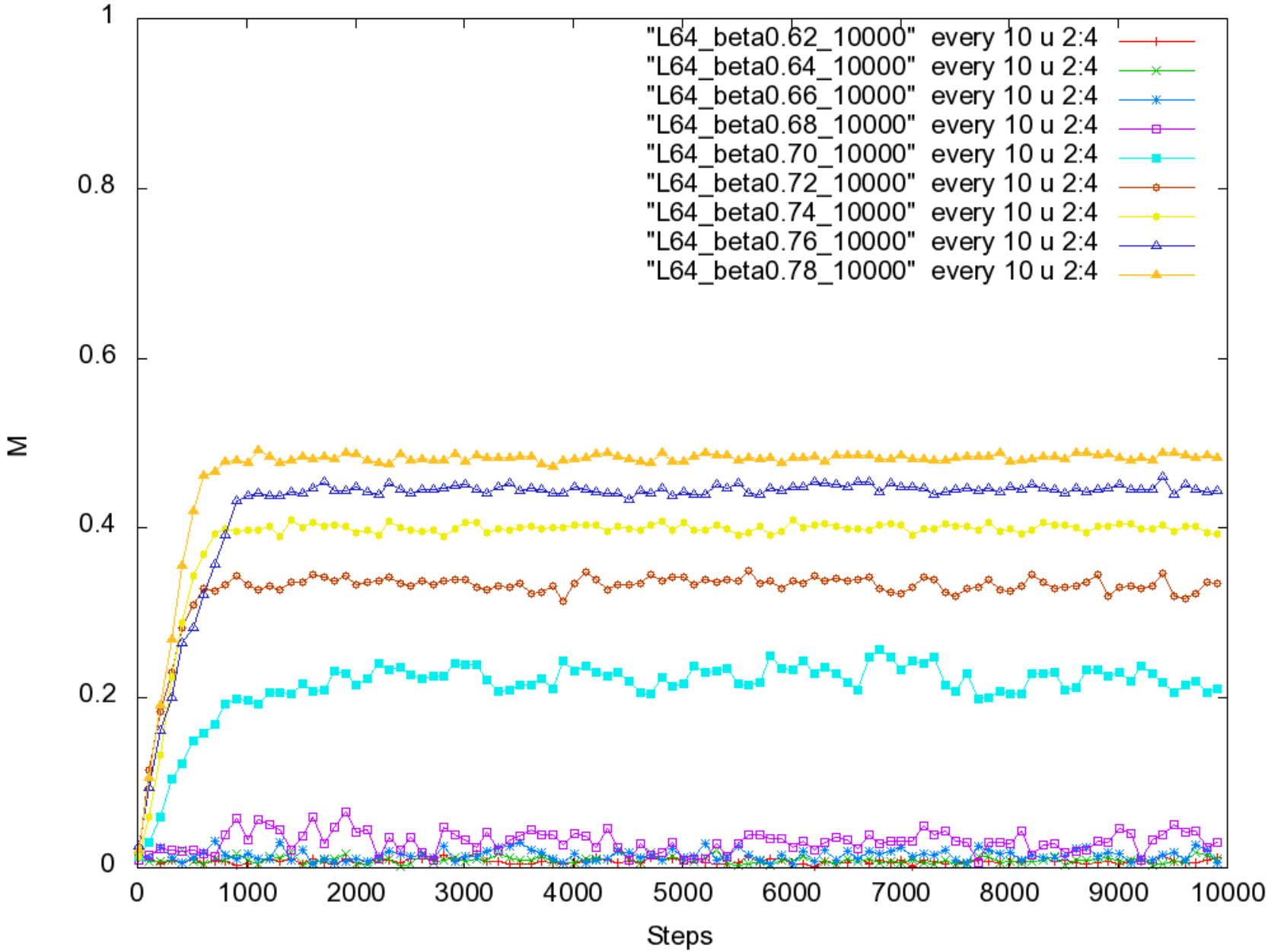
The code executed by the GPU:

```
Require: color = {red, blu}
1: load a set of (4 × 4 × L) points
2: apply MC-step to bulk's points
3: save bulk to memory   // required to sync blocks
```

Ising Magnetizzazion (with J_{ij} constant)



Heisenberg Magnetizzazion (with J_{ij} constant)



Performance Analysis

Perfomance assessments:

- comparison of GPU performances with Intel-NH and IBM-CBE
- optimized code for NH and CBE (SSE, cache-allignement)
- SUT metric: average time to process one spin of one system
- GUT metric: average time to process one spin of all systems (using multi-spin encoding)

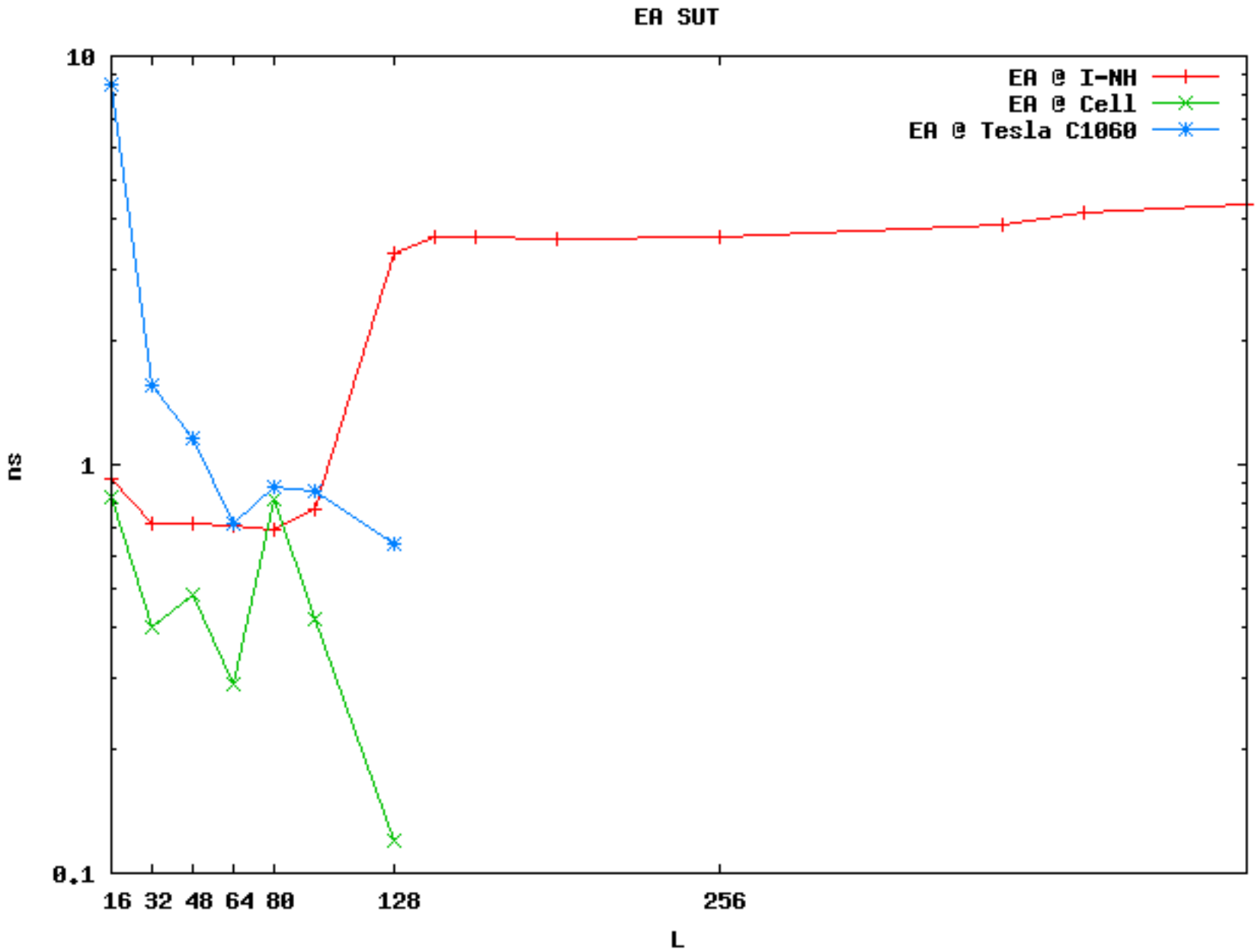
Results

3D Edwards Anderson Model SUT (ns/spin)				
L	Janus	I-NH (4 cores)	Cell-BE (8 SPE)	Tesla C1060
16	0.0016	0.92	0.83	8.48
32	0.0016	0.72	0.40	1.56
48	0.0016	0.72	0.48	1.16
64	0.0016	0.71	0.29	0.72
80	0.0016	0.69	0.82	0.88
96	–	0.78	0.42	0.86
128	–	3.29	0.12	0.64

3D Heisenberg Model SUT (ns/spin)				
L		I7 1.6 GHz[1]		Tesla C1060
16		3570 DP		–
32		3570 DP		34.4 SP / 139.0 DP
48		3570 DP		29.6 SP / 134.8 DP
64		3570 DP		31.0 SP / 131.5 DP
80		3570 DP		28.3 SP / 130.7 DP
96		3570 DP		29.2 SP / 129.6 DP
128		3570 DP		28.9 SP / 129.1 DP

[1] not yet fully optimized

SUT


EA SUT
EA @ I-NH
EA @ Cell
EA @ Tesla C1060
10
1
0.1
ns
16 32 48 64 80
128
256
L

Conclusion

- No conclusions yet . . . work in progress . . .
- Results on Intel-NH are OK while the lattice size fits the cache
- Performance of Tesla C1060 for Heisenberg code is around 20 Gflops
- Results on Tesla C1060 are penalized by:
  - lacking of synchronization among TPC to handle boundary conditions
  - volatility of shared-memory: flashed on global memory at each iteration

Credits:

1. F. Belletti for the Cell code
2. M. Cinti for the Intel code
3. N. Mantovani for the GPU code