A display of GPU implementations in Condensed Matter Physics four distinctive cases

Ezequiel Ferrero

Université Grenoble Alpes and CNRS, LIPHY, F-38000 Grenoble, France

Contact: ezequiel.ferrero@ujf-grenoble.fr
www.ezequielferrero.com

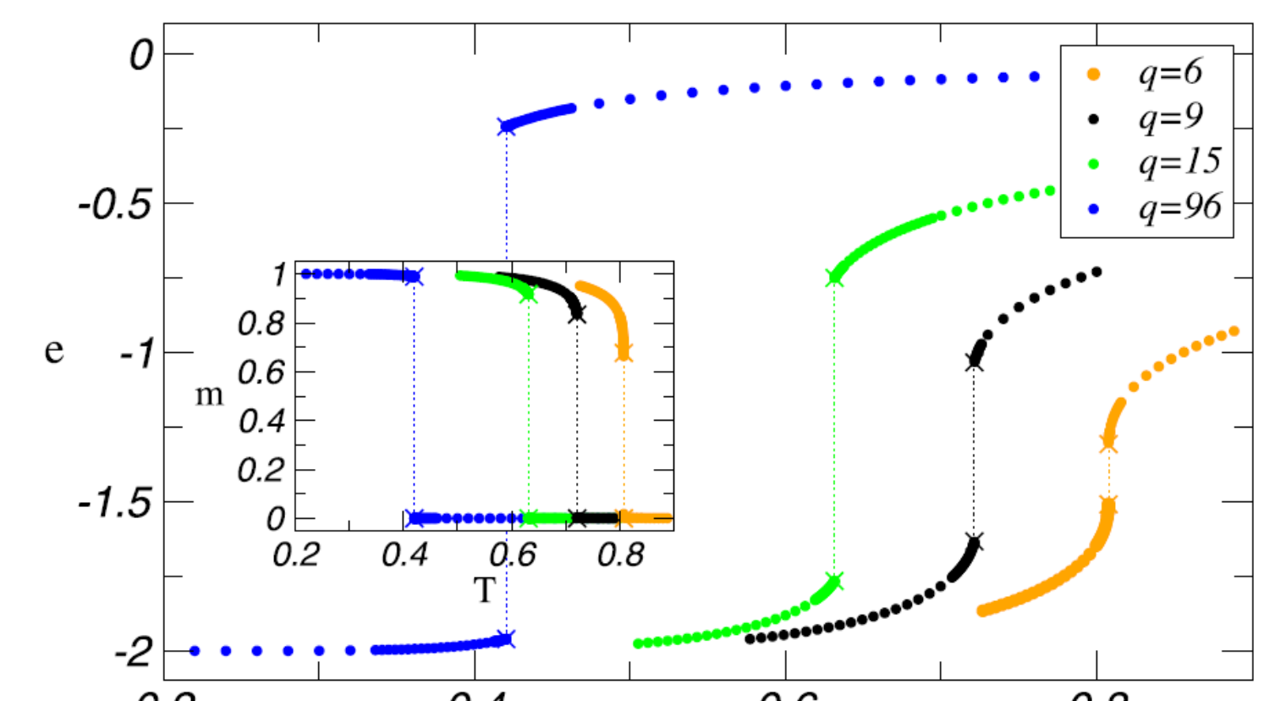
General Purpose Graphics Processing Units have shaken up the computational scientific community. In many cases, people adopted parallelism with CUDA, before even knowing what was exactly MPI or OMP about. As statistical physicists working in condensed matter, with the aim of accelerating our simulations, we have implemented in the last few years some massively parallel codes. We present here four examples of *ad-hoc* models that effectively describe the phenomenology of a physical system at a given scale. We can classify them in two families: (i) lattice based Monte Carlo simulations (for classical spin models and electron glasses); (ii) overdamped dynamics of scalar systems (for elastic lines in disordered media and sheared amorphous solids). Dimensionality, interaction range, particular dynamics, and other details of the model, determine the parallelization strategy in each case.

q-state Potts model Monte Carlo (classical spins system)

$$H = -J\sum_{(i,j)} \delta_K(s_i,s_j)$$

$$s_i = 1,2,\dots,q \qquad J>0 \qquad \delta_K(x,x') = \begin{cases}1 & \text{if } x=x' \\ 0 & \text{otherwise}\end{cases}$$

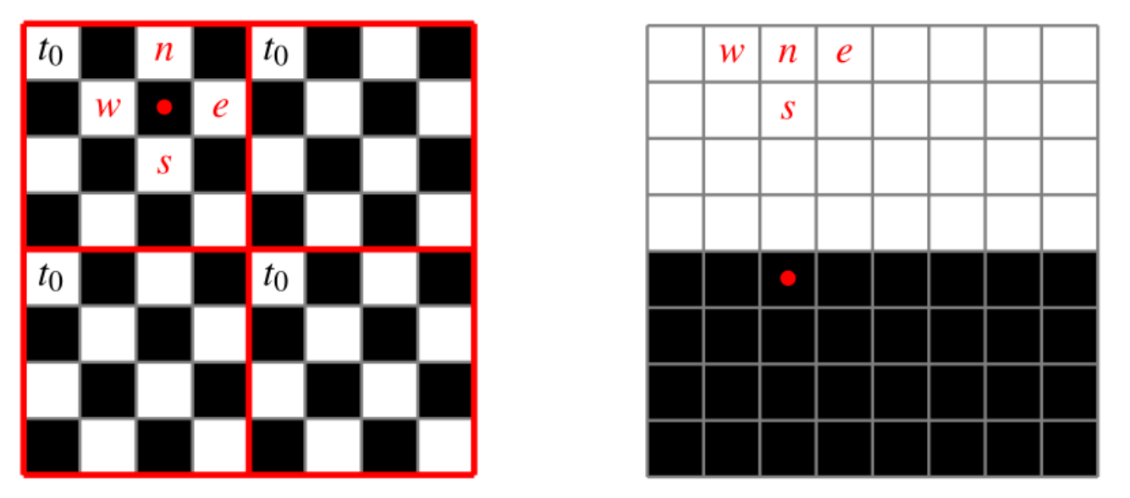
Spin flips are accepted with probability $p = \tau^{-1} e^{-\Delta H/k_B T}$



Equilibrium energy per spin *e* and magnetization *m* (inset) versus temperature. Exact values at the transition marked as crosses.

Parallelization strategy

Checkerboard scheme (interactions limited to nearest neighbors)



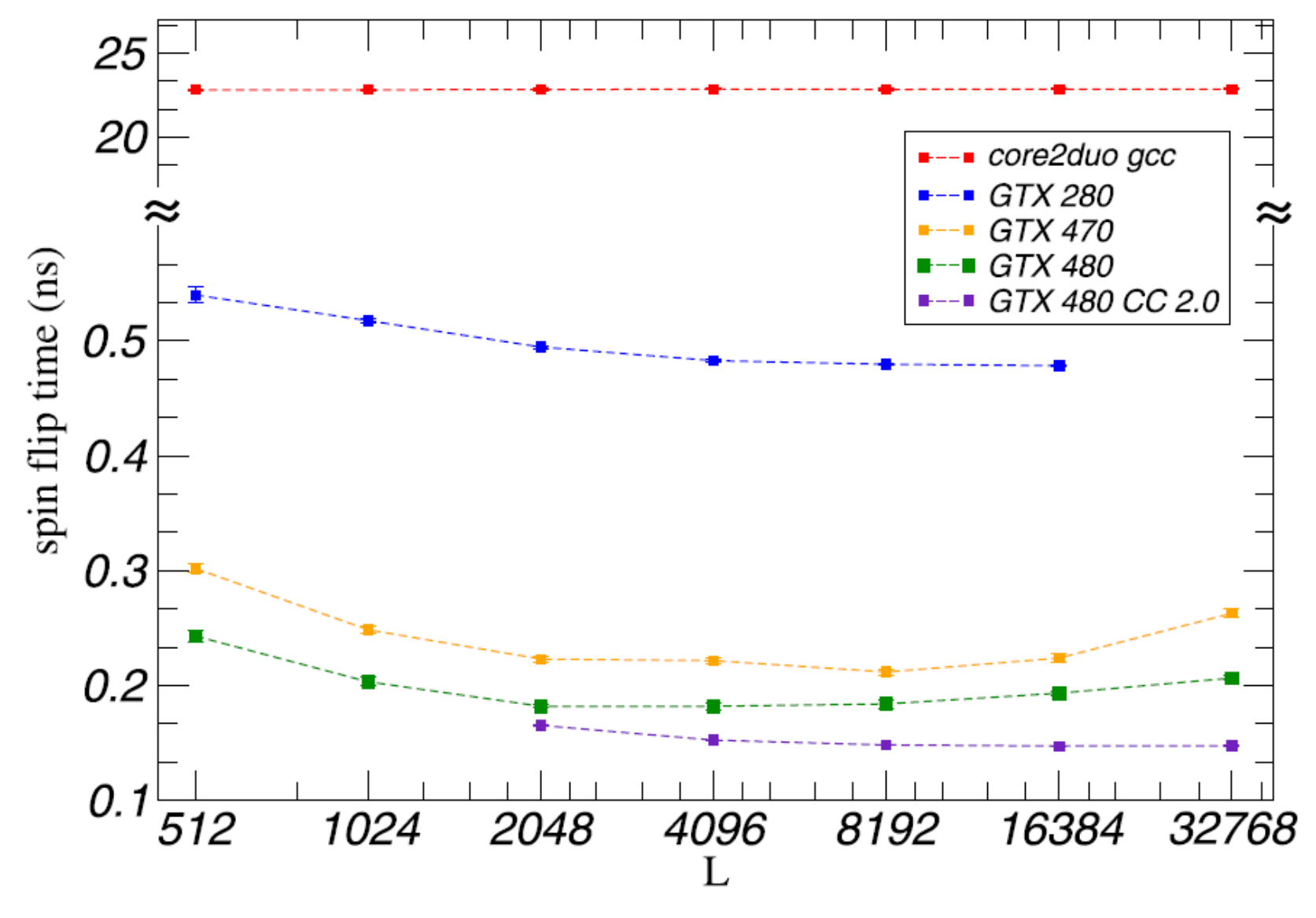
Stencil compaction-- **Left:** an 8×8 checkerboard framed in 4×4 (red marking), the cells updated by thread t_0 are singled out, we also marked the neighbors of cell •. **Right:** packed checkerboard showing first half of whites, where the neighboring cells *n*, *e*, *s*, *w* are marked, also in the second half of black cells • is singled out.

Code features

- Pure **CUDA C**
- Implements **Multiply With Carry RNG** with FRAMESxFRAMES/2 independent generators.
- Computes **multiple outputs per *thread***. Two consecutive kernels (black/white) of typically 512×512/2 threads are launched.
- Comprises the remapping of a two-dimensional stencil of four points in order to **save memory transfers**.
  → **Encodes each spin in a byte**, allowing simulations with *q*<256 and L^2 < *available RAM.*
  → Uses **registers for RNG states**.
  → Implements **parallel sums** (butterfly-like algorithm) for averages calculations using **shared memory**.

Performance

- Largely dominated by the update routine (involving 2 RNG calls per site)
- **GTX 280** provides **42x to 47x** speedup
- **GTX 470** provides **76x to 108x** speedup
- **GTX 480** provides **95x to 155x** speedup
- There are **two competing** factors in the loop of the update kernel:
  - One **decreasing with L**. We have one RNG for each thread, the global memory for the RNG state is retrieved one time at the beginning and stored at the end. The larger the L, the single load/store **latency is distributed** into more cells.
  - The second factor is **increasing in L** and is given by the **inherent overhead** incurred by a loop (comparison and branching), that for L = 32 768 amounts to 4096 repetitions.



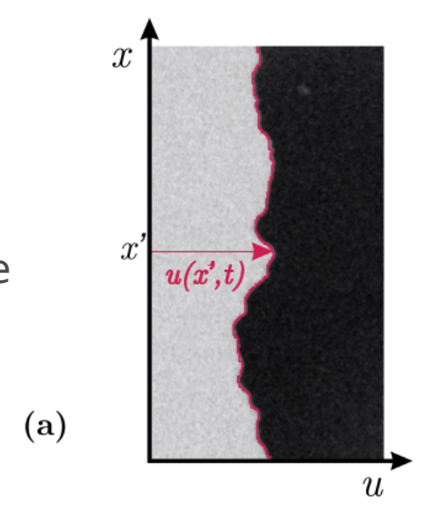
Spin flip time in nanoseconds vs. lattice size running on an Intel Core 2 Duo E8400@3.0 GHz CPU, and running on GTX 280, GTX 470 and GTX 480 NVIDIA GPUs.

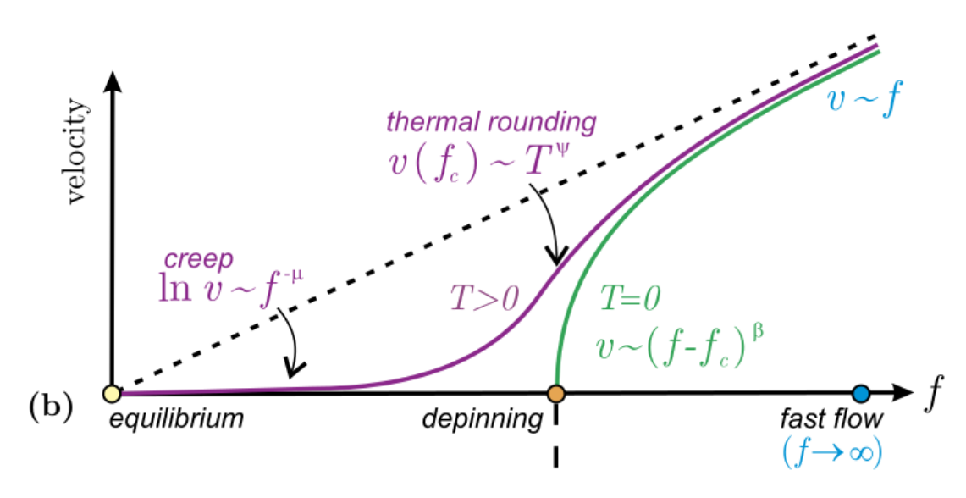
[1] E.E. Ferrero, J.P. De Francesco, N. Wolovick, S. A. Cannas, *Comp. Phys. Commun.* **183** 1578 (2012)
[2] E.E. Ferrero, F. Romá, S. Bustingorry, P.M. Gleiser, *Phys. Rev. E* **86** 031121 (2012)

https://bitbucket.org/ezeferrero/potts
https://bitbucket.org/ezeferrero/potts-glass

Quenched Edwards-Wilkinson model (elastic manifolds in disordered media)

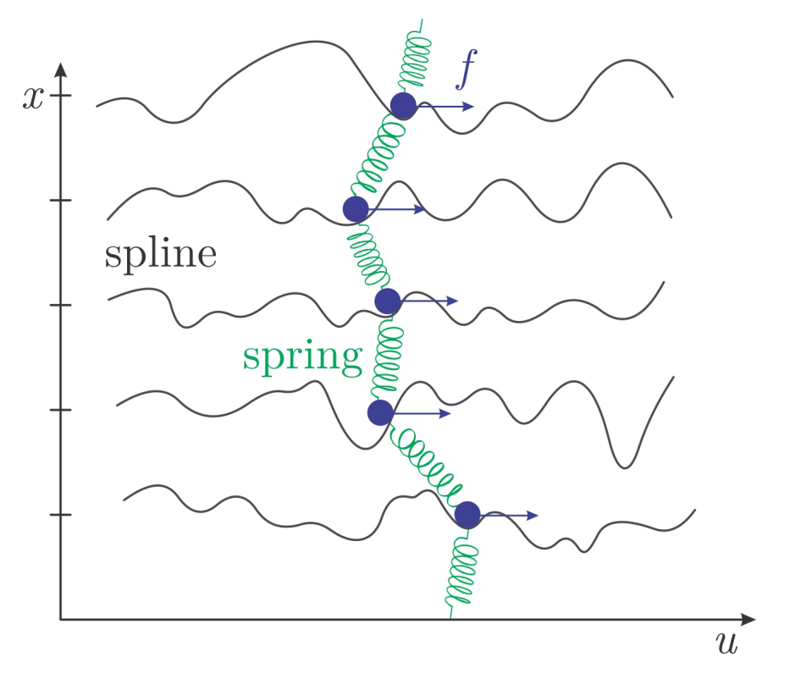
$$\underset{\text{interface velocity}}{\eta\partial_t u(x,t)} = \underset{\text{elastic interactions}}{c\partial_x^2 u(x,t)} + \underset{\text{pinning potential}}{F_p(u,x)} + \underset{\text{external force}}{f}$$





Implementation

Euler integration in discretized time



- Line of size ***L***
  - discretize *x*=0...(*L*-1)
  - keep *u(x,t)* as a *real* variable
  - periodic boundary conditions (*u*[0] coupled with *u*[*L*-1])
- Two different disorder schemes
  - Spline from a presorted array *LxM*
  - Dynamically generated disorder

Code features

- **C++** and **CUDA C**
- Massively **parallel** dynamics integration (alternating even and odd sites), L/2 threads.
- **Averages** of displacement velocity and quadratic width made by **Thrust** transforms and reductions.
- **Coexisting** CUDA **kernels and Thrust functions**, wrapping pointers.
- Uses alternatively two different RNGs:
  → **MWC**, as in Potts model.
  → **Philox**, a **counter based RNG** of the Random123 library.
- Uses **cuFFT** to compute structure factor S(q,t)

Benchmarking

Mean update time execution and their related CPU vs. CPU+GPU practical speed-ups.
System size L=65536 taken as an example.

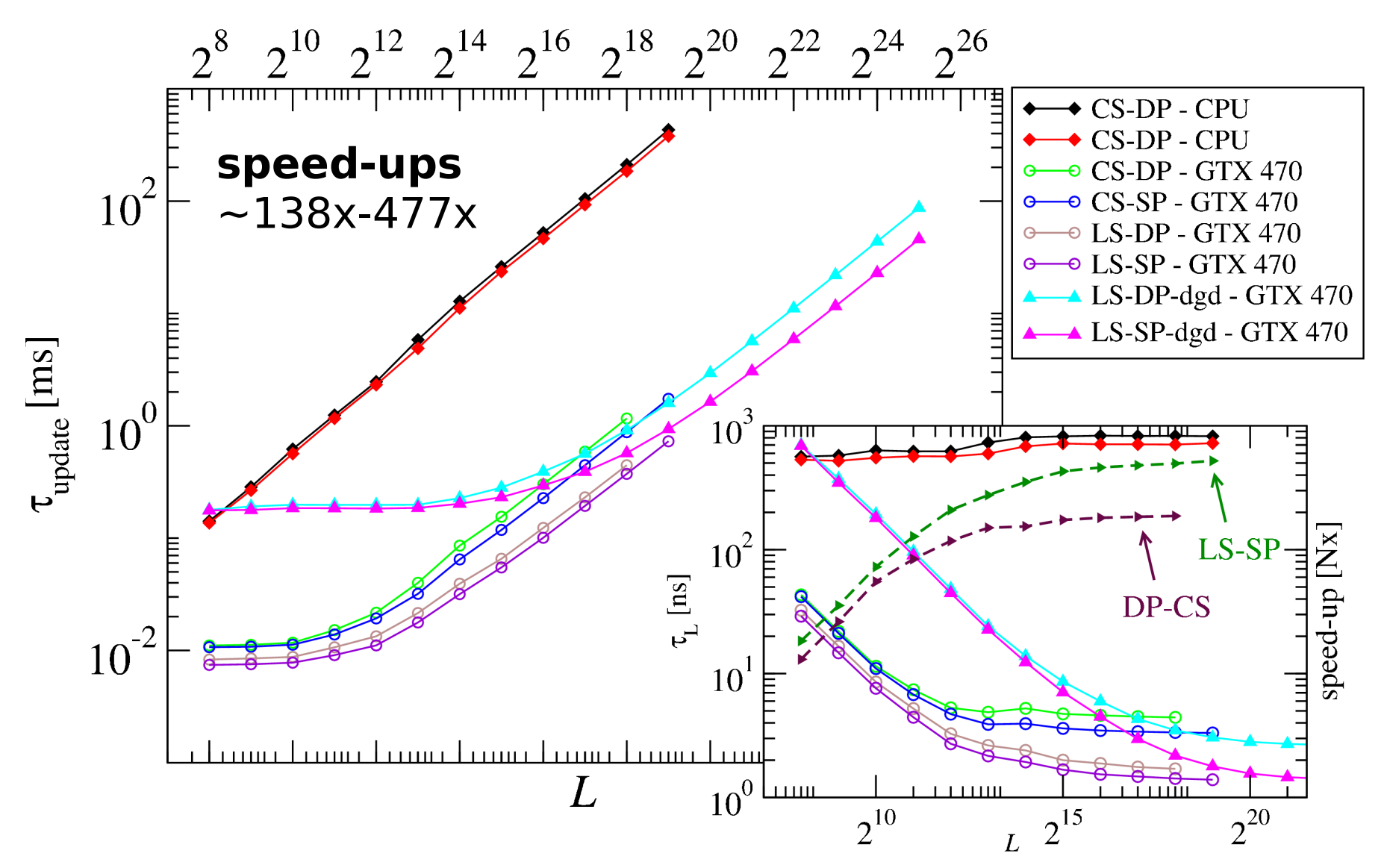
Case	single-core CPU	CPU + GTX 470		CPU + Tesla C2075	
	time[ms]	*time[ms]*	*psu*	*time[ms]*	*psu*
CS DP	54.49	0.301	181	0.396	138
CS SP	∼ 54	0.227	238	0.298	181
LS DP	46.46	0.123	377	0.152	305
LS SP	∼ 46	0.101	455	0.128	359

Platform: AMD Phenom II X4 955 Processor @3.2GHz, NVIDIA Tesla C2075, NVIDIA GTX 470.

Case	single-core CPU	CPU + GTX 480	
	time[ms]	*time[ms]*	*psu*
CS DP	42.22	0.238	186
CS SP	∼ 42	0.179	235
LS DP	36.55	0.098	373
LS SP	∼ 36	0.076	477

Platform: Intel Core 2 Quad CPU Q9550 @2.83GHz, NVIDIA GTX 480.

Acronyms: **CS**: Cubic Spline, **LS**: Linear Spline, **DP**: Double Precision, **SP**: Single Precision, **psu**: practical speed-up.



Update step mean-computing time in ms. Platform: AMD Phenom II X4 955 Processor @3.2GHz, NVIDIA GTX 470

[1] E.E. Ferrero, S. Bustingorry, A.B. Kolton, *Phys. Rev. E* **87** 032122 (2013)
[2] A.B. Kolton, S. Bustingorry, E.E. Ferrero, A. Rosso, JSTAT **P12**004 (2013)

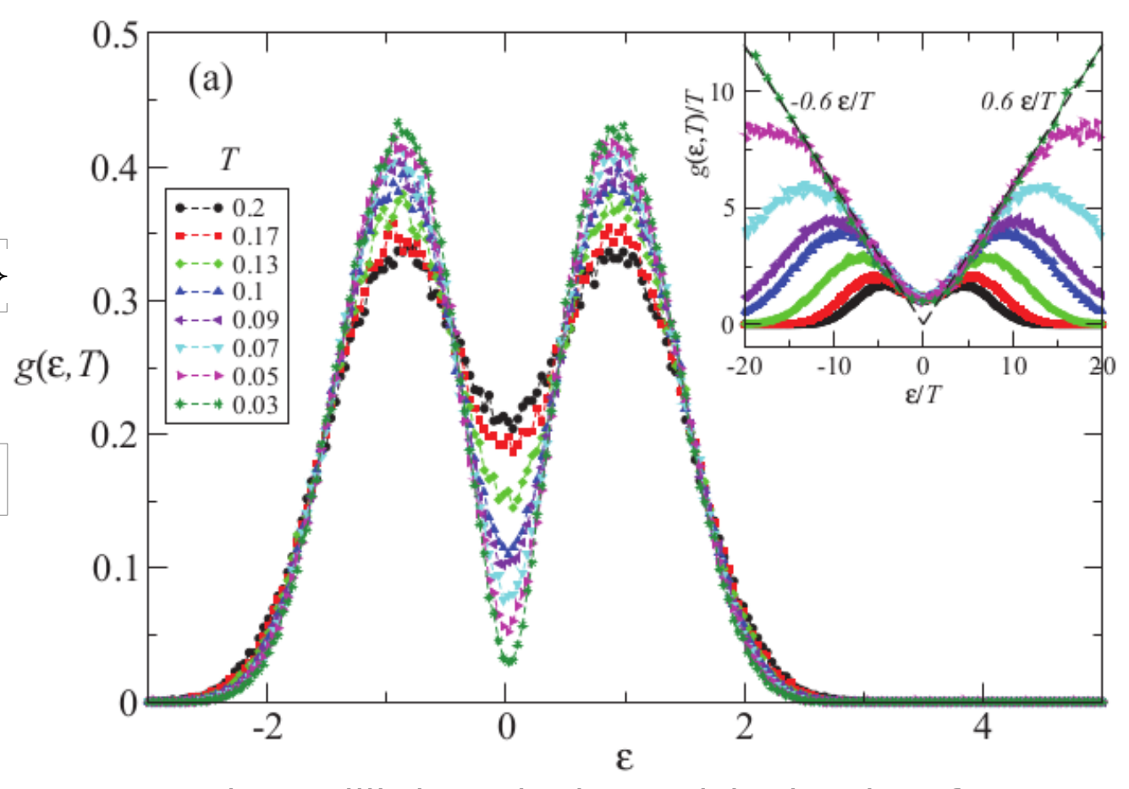
https://bitbucket.org/ezeferrero/qew

Coulomb glass Kinetic Monte Carlo (electrons glass system)

$$H = \sum_{i=0}^{N-1} \underset{\text{random potential}}{\phi_i} n_i + \frac{1}{2}\sum_{i\neq j}\frac{(n_i-\nu)(n_j-\nu)}{r_{ij}} - E\sum_{i=0}^{N-1} n_i x_i \qquad \boxed{n_i \in \{0,1\}}$$

Kinetics $\Gamma_{ij} = \tau_0^{-1}\theta_{ij}e^{-2r_{ij}/\xi}\min[1,e^{-\epsilon_{ij}/k_BT}] \qquad \boxed{\theta_{ij} = \delta_{n_i,1}\delta_{n_j,0}}$

$$\epsilon_{ij} = \epsilon_i - \epsilon_j - \frac{1}{r_{ij}} - E_{ij} \qquad \epsilon_i = \phi_i + \sum_j \frac{1}{r_{ij}}(n_j-\nu)$$

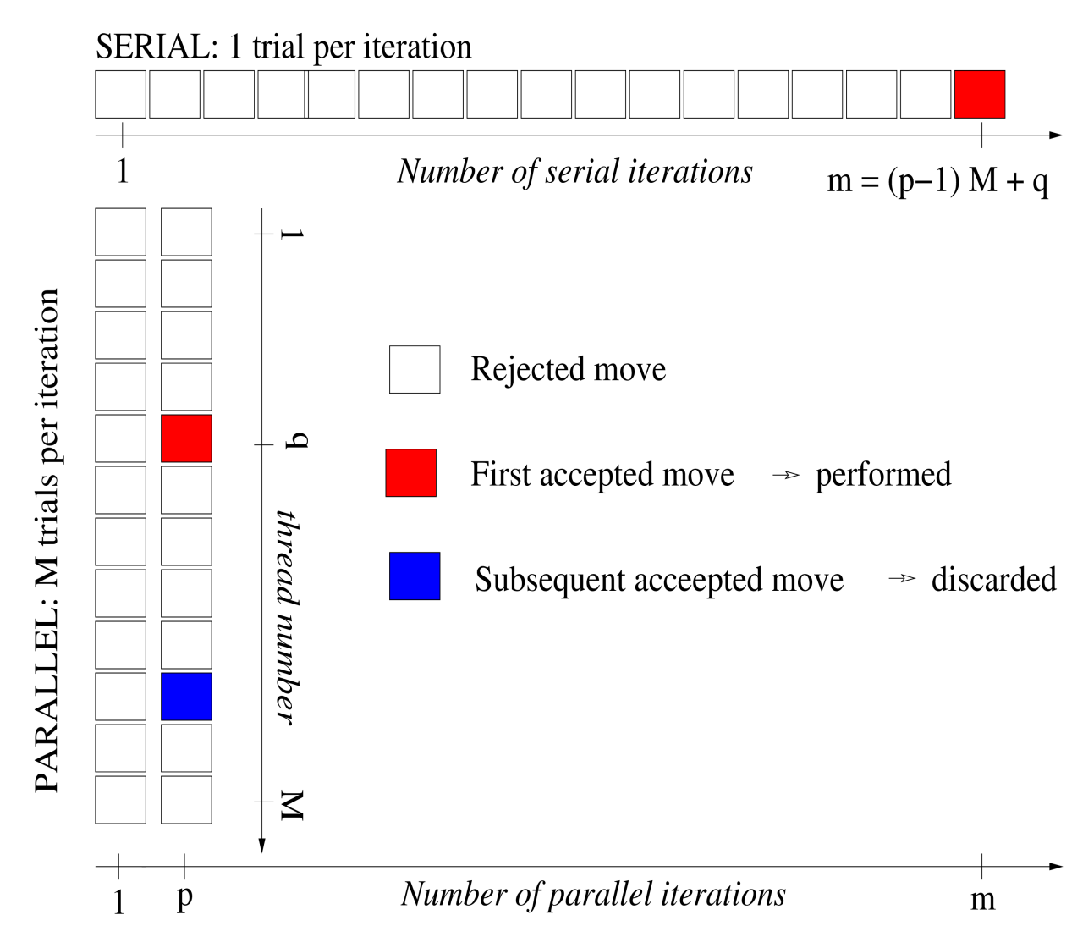


Pseudo-equilibrium single-particle density of states

Parallel Kinetic Monte Carlo

Markov-chain MC with proposal matrix $Q_{\alpha\beta}$ and acceptance $P_{\alpha\beta}$ between conf. α and β.

1. Propose with independent threads (k=1,...,M), moves $\alpha \to \beta_k$.
2. Accept the move $\alpha \to \beta_k$ independently with prob $P_{\alpha\beta_k}$ (without updating)
3. If at least one thread has accepted a move, update the conf. to β_q, where *q* is the lowest label among the threads accepting.

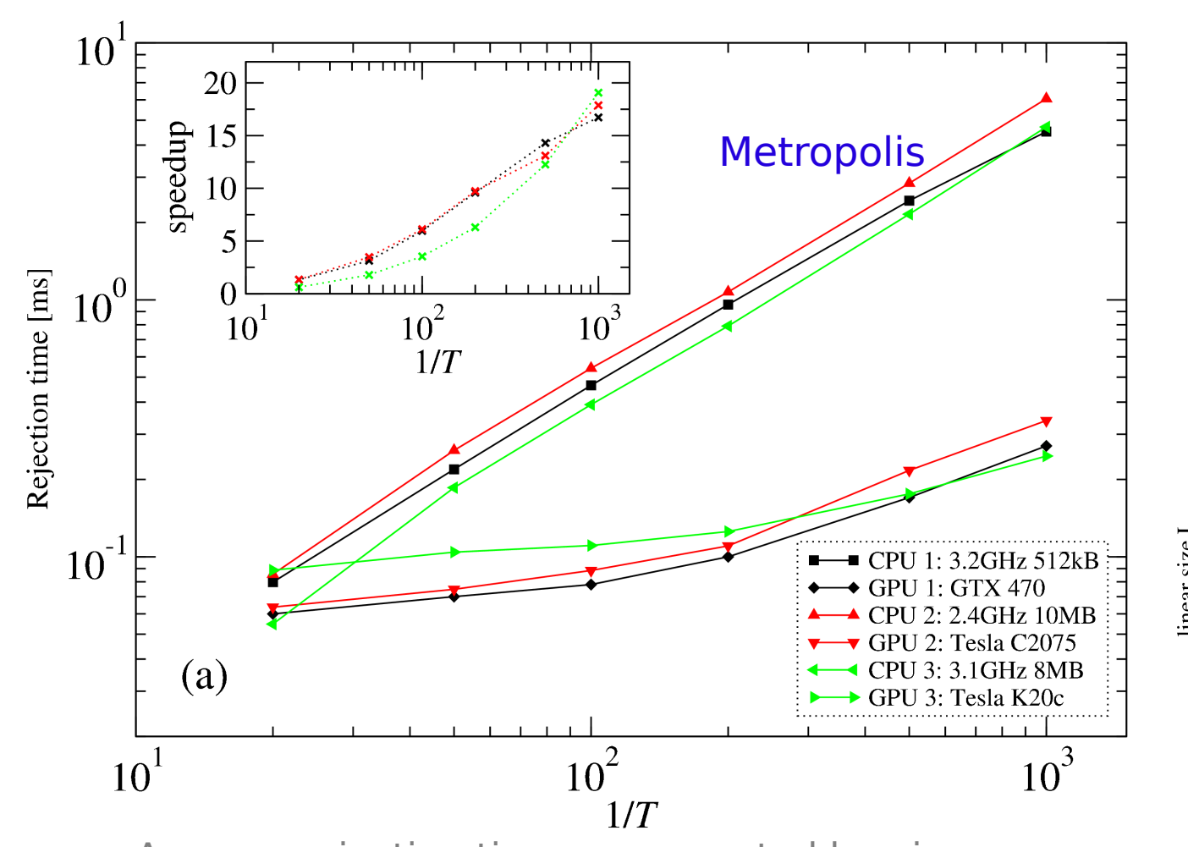


SERIAL: 1 trial per iteration
1
Number of serial iterations
m = (p−1) M + q
PARALLEL: M trials per iteration
thread number
1
q
M
Rejected move
First accepted move → performed
Subsequent acceepted move → discarded
1
p
Number of parallel iterations
m

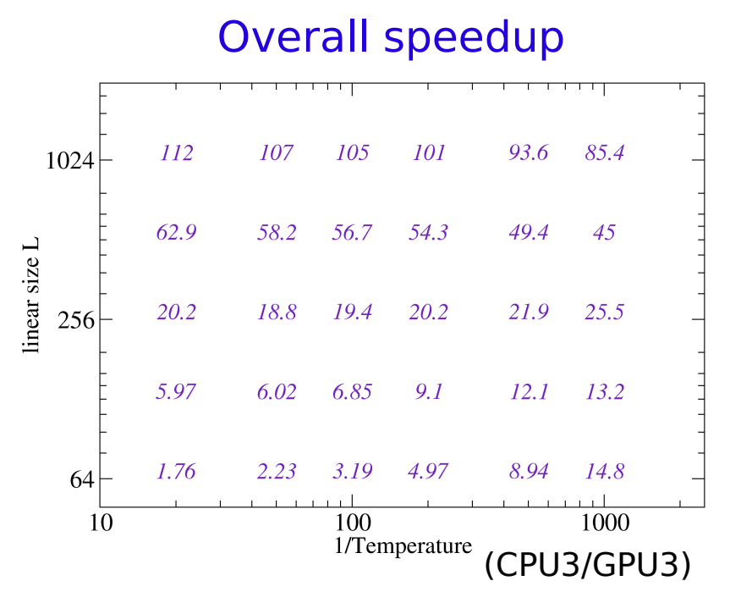
Code features

- Pure **CUDA C**
- Trial hop (*i* → *j*): **tower sampling** with **vectorized** binary search.
- Metropolis: **Parallel rejection**
- Update Local Energies: **embarrassingly parallel** kernel.
- Global observables: **parallel reductions**.

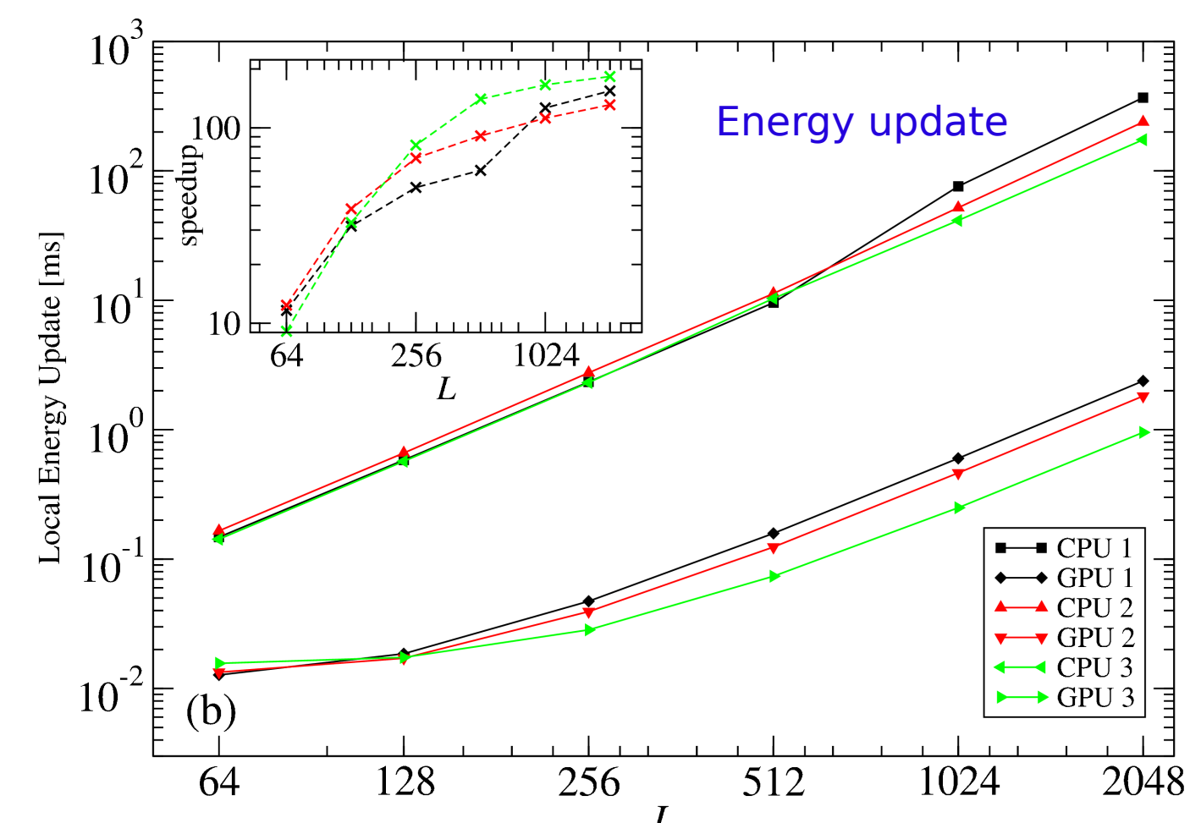
Performance



Average rejection time per executed hop in ms



- For large N=L^2 the computation **scales as O(N)** both in CPU and GPU, but with an important ratio, controlled by the optimal number of concurrent threads, yielding **speedups over 100x**.
- This speedup **dominates** the *total* computation time at large N.



Average update time of energies per executed hop in ms

- **Speedup** *per accepted hop* **monotonically increases** with decreasing temperature (acceptance), roughly independent on L.
- **No signs of saturation** for the speedup up to values as low as T=0.001.

[1] E. E. Ferrero, A. B. Kolton, M. Palassini, *AIP Conf. Proc.* **1610** 71 (2014)

https://bitbucket.org/ezeferrero/coulomb_glass

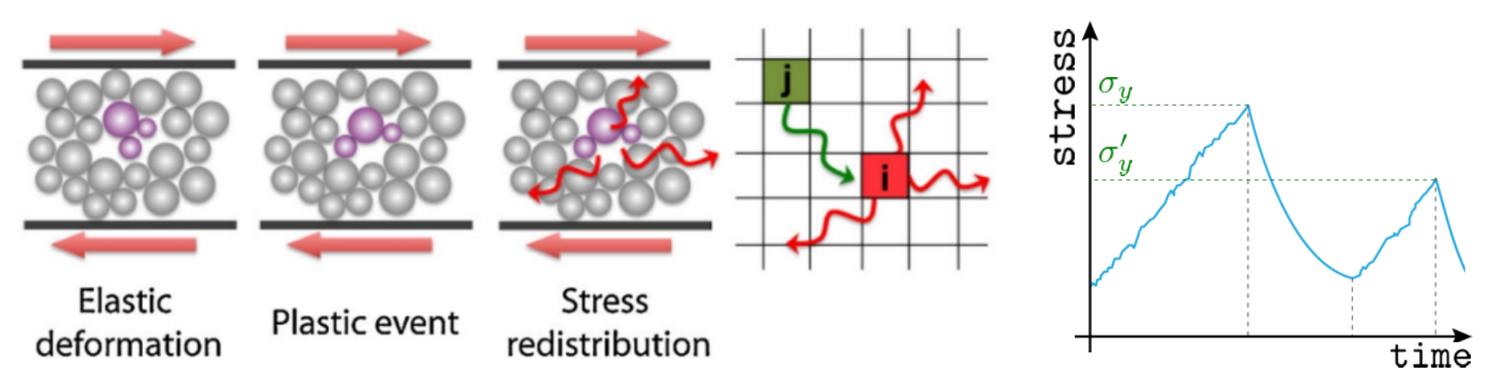
Athermal elasto-plastic model (sheared amorphous solids)

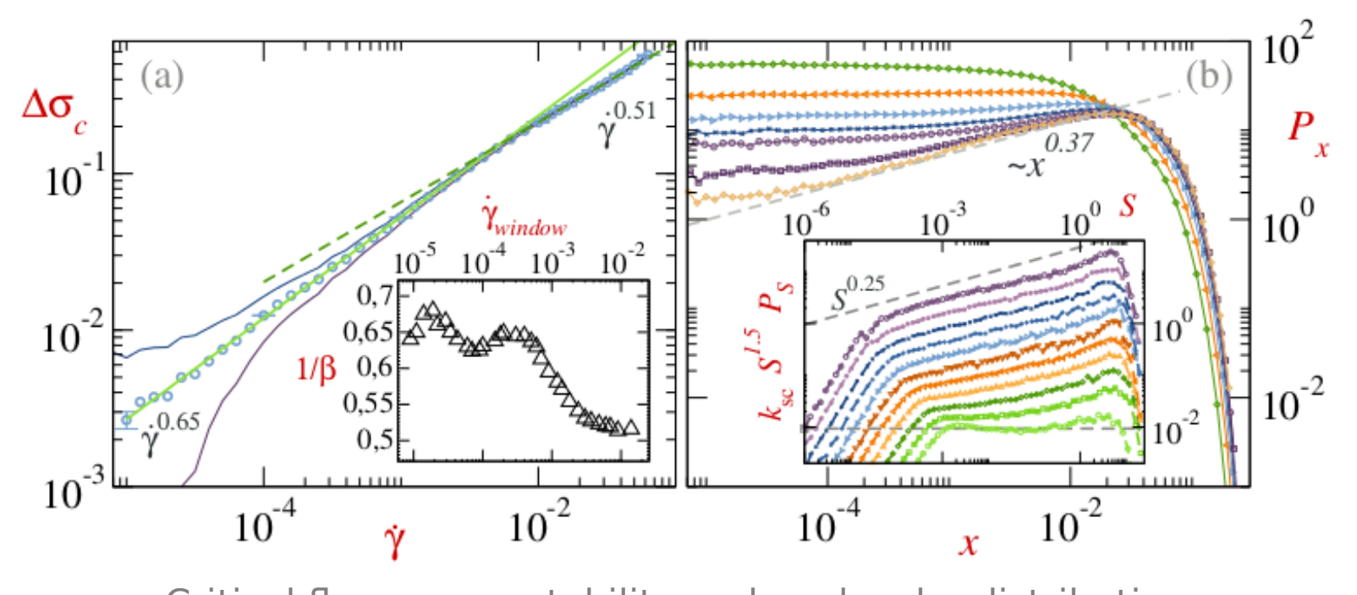
$$\underset{\text{Stress change}}{\partial_t\sigma(\mathbf{r},t)} = \mu\underset{\text{shear-rate}}{\dot\gamma^{\text{ext}}} + \mu\int d\mathbf{r}' G(\mathbf{r},\mathbf{r}')\dot\gamma^{\text{plast}}(\mathbf{r}',t)$$

Plastic strain rate $\dot\gamma^{\text{plast}}(\mathbf{r},t) = n(\mathbf{r},t)\frac{\sigma(\mathbf{r},t)}{\mu\tau}$

Elastic propagator **Long-range!!** $G(r,\theta) = cos(4\theta)/\pi r^d \qquad r = |\mathbf{r}'-\mathbf{r}|$

$$n(\mathbf{r},t): \begin{cases} 0\to 1 & \text{if} \quad \sigma > \sigma_y \longleftarrow \text{randomly taken from } P(\sigma_y) \\ 0 \leftarrow 1 & \text{when} \quad \int dt'|\partial_t\sigma(t')/\mu + \dot\gamma^{pl}(t')| \geq \gamma_c \end{cases}$$





Critical flow-curve, stability and avalanche distributions

Parallelization strategy

"Pseudo-spectral" method: transform Fourier, compute and anti-transform at each step

$$\int d\mathbf{r}' G(|\mathbf{r}-\mathbf{r}'|)n(\mathbf{r}')\sigma(\mathbf{r}') \longrightarrow G_{\mathbf{q}}\tilde n(\mathbf{q})\tilde\sigma(\mathbf{q})$$

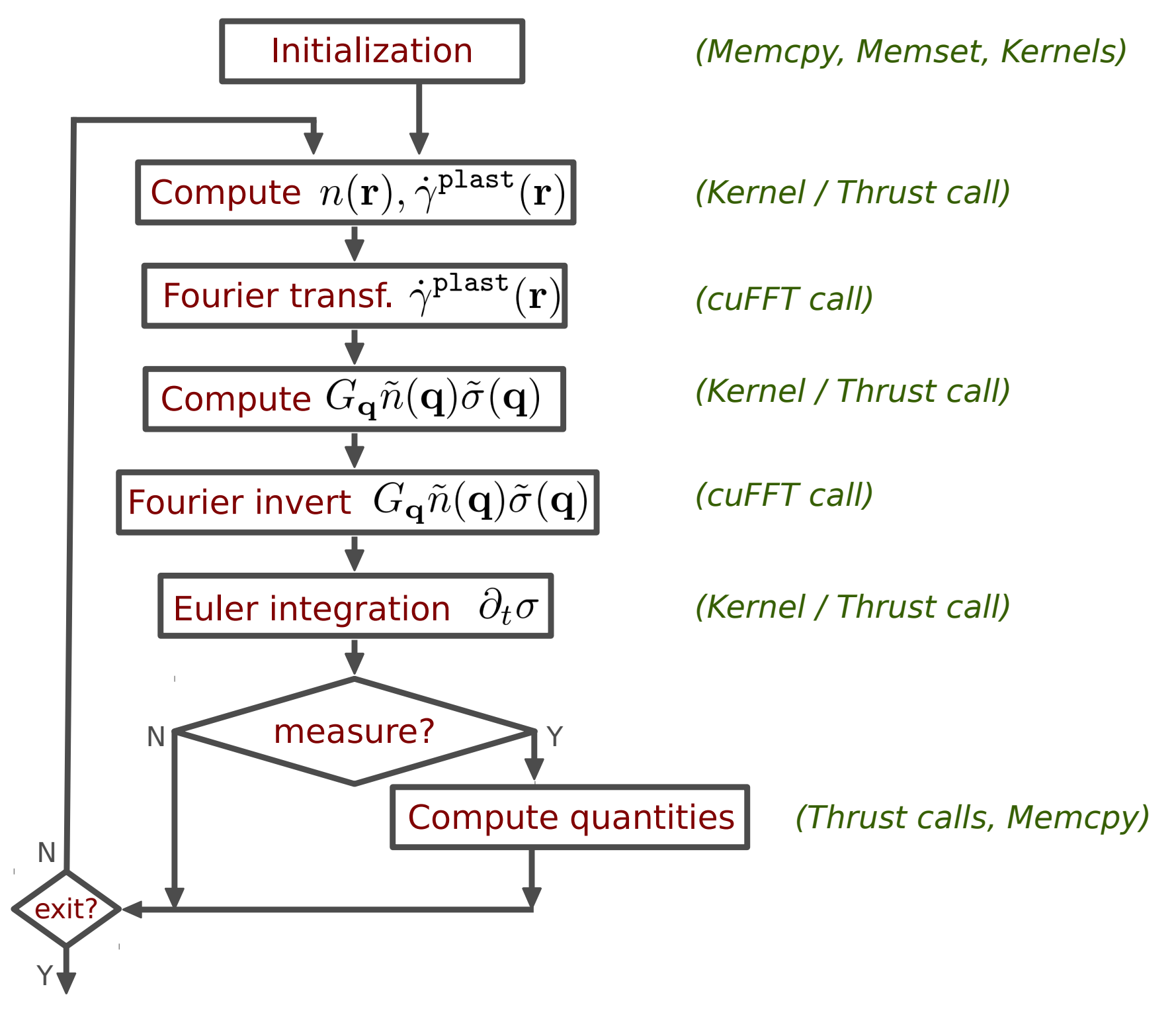
$$G(r,\theta) = cos(4\theta)/\pi r^d \longrightarrow \hat G = -4\frac{q_x q_y}{q^2}$$

Embarrassingly **parallel in Fourier** space, local in *(q_x,q_y)*

Code features

- **C++** and **CUDA C**
- Intensive use of **cuFFT**
- Uses **Thrust** for averages and extrema finding
- Uses **Philox RNG**

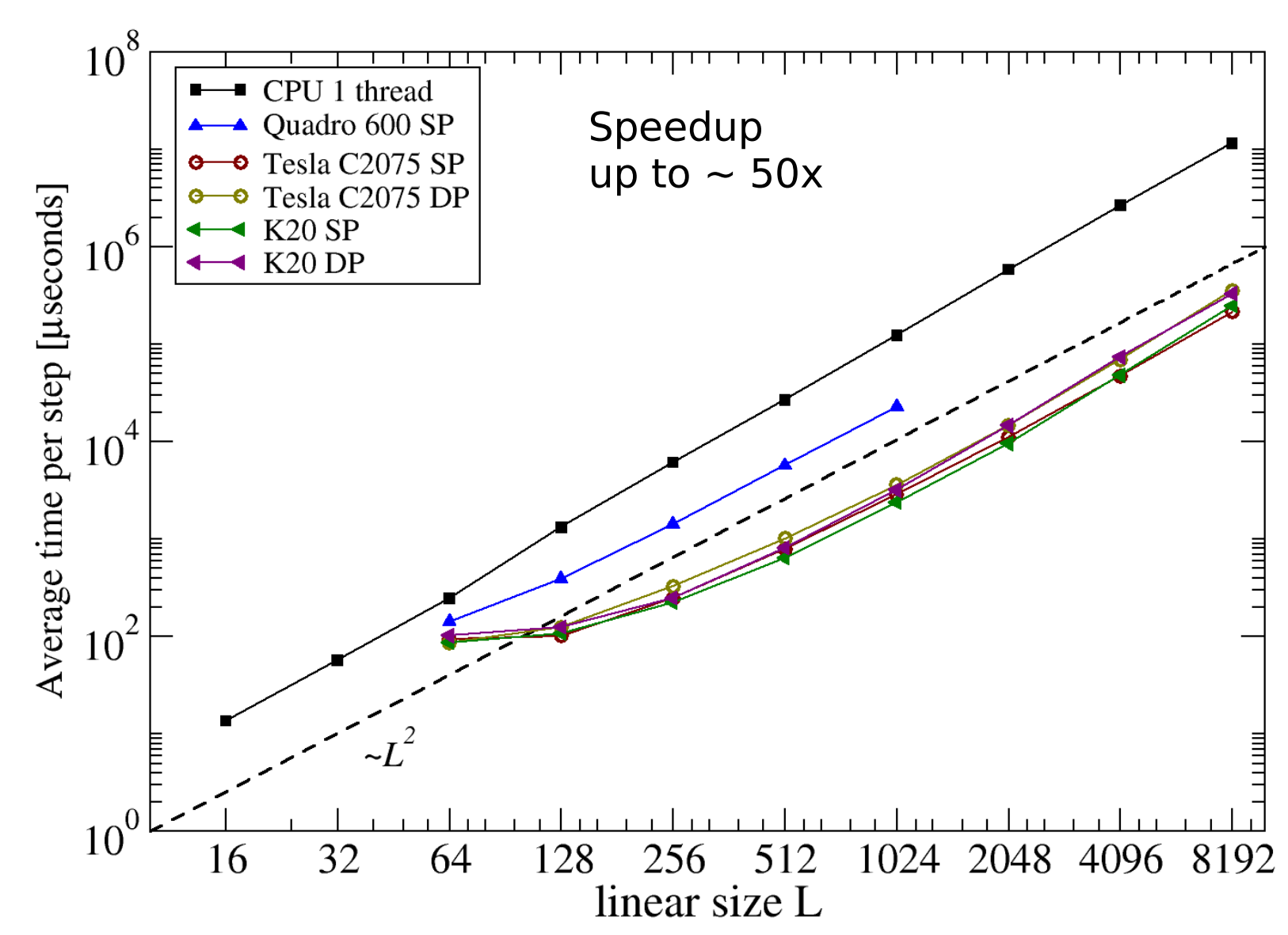
General work-flow



Initialization
(Memcpy, Memset, Kernels)
Compute $n(\mathbf{r}), \dot\gamma^{\text{plast}}(\mathbf{r})$
(Kernel / Thrust call)
Fourier transf. $\dot\gamma^{\text{plast}}(\mathbf{r})$
(cuFFT call)
Compute $G_{\mathbf{q}}\tilde n(\mathbf{q})\tilde\sigma(\mathbf{q})$
(Kernel / Thrust call)
Fourier invert $G_{\mathbf{q}}\tilde n(\mathbf{q})\tilde\sigma(\mathbf{q})$
(cuFFT call)
Euler integration $\partial_t\sigma$
(Kernel / Thrust call)
measure?
N
Y
Compute quantities
(Thrust calls, Memcpy)
exit?
N
Y

Performance

Just a fast benchmark so far...



Platforms:
- Intel Xeon E5-2609 @2.4GHz 10M, Quadro 600 & Tesla c2075
- Intel Sandy Bridge EP E5-2670 @2.6GHz 20M, NVIDIA K20

[1] E. E. Ferrero, K. Martens, J.-L. Barrat, *Phys. Rev. Lett.* **113**, 248301 (2014)
[2] C.Liu, E.E. Ferrero, F. Puosi, J.-L. Barrat, K. Martens, arXiv 1506.08161 (2015)

https://bitbucket.org/ezeferrero/ep

General considerations:

We have **adopted** the the **GPGPU approach** to address different problems in Condensed Matter physics. The **goal** in each case was to **implement and verify a code** to **produce valid physical results**. Very good **practical speedups** have been attained respect to what people have been using before for the same kind of problems, even when optimization is not seriously considered.
Present **challenges** include: *i)* exploit better the hardware, starting with CPU-GPU **concurrency** and multi-GPU approaches; *ii)* deal better with **I/O operations**, sometimes they drain all the obtained performance, dilemma "output raw vs compute on-the-fly"; *iii)* migrate from ad-hoc CUDA kernels to library calls for better **portability**; *iv)* make the jump to Open CL.