1 – WP1 - ACTIVITY ON GPUs WITHIN NPQCD - PI12

M. D'Elia (Pisa)

A few years ago, stimulated by a lack in computing power availability, we were attracted by the experimental (at that time) use of Graphics Cards by other lattice groups in the world. In the following I will try to summarize:

- Our code implementation and performances on single GPUs
- Physics projects and results
- Perspectives

2 – Lattice QCD and GPUs

The main numerical task is the sampling of gauge field configurations by dynamic Monte-Carlo:

- **Stochastic variables:** 3×3 **unitary complex matrices** $U_\mu(n)$ **(gauge link variables) associated to each elementary link of a (typically cubic) 4d space-time lattice of spacing** a**.** $4\, L_x\, L_y\, L_z\, L_t$ **matrixes on the whole How big our lattice?**

  $a \ll \text{shortest scale}\,; \quad L_s a \gg \text{largest scale} \implies a \leq 0.1\text{fm}\,; \;\; L_s \sim O(10^2)$

  $L_t \sim 1 - 2L_s$ **for** $T = 0$ **simulations;** $L_t \sim O(10)$ **for finite** T **simulations**

- **Equilibrium distribution:** $\mathcal{D}U e^{-S_G[U]} \det M[U]$
  - S_G **(pure gauge action): local term taking into account gluon-gluon interactions**
  - $\det M[U]$ **is the determinant of the fermion matrix: non-local term which takes into account dynamical fermion contribution.** M **is a** $N \times N$ **sparse matrix**

$$N = \text{Lattice\_sites} \cdot \text{N\_of\_Colors} \cdot \text{Dirac\_components}$$ **up to** $\sim 10^8 - 10^9$

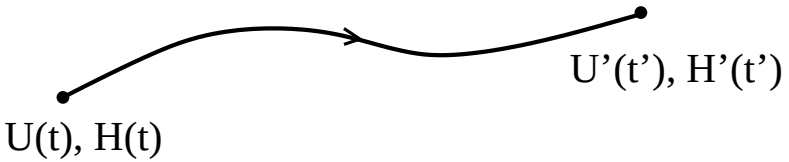
The typical algorithm: Hybrid Monte Carlo

- **Requires auxiliary variables:** $\mathcal{D}U e^{-S_G[U]} (\det M[U])^2 \rightarrow \mathcal{D}U\mathcal{D}H\mathcal{D}\Phi^\dagger\mathcal{D}\Phi\, e^{-\mathcal{H}}$
  $\mathcal{H} = S_G[U] - \Phi^\dagger (M[U]M[U]^\dagger)^{-1}\Phi + \frac{1}{2}\sum_{n,\mu} \mathrm{Tr} H_\mu^2(n)$

- **Pseudofermion fields Φ and conjugate momenta H_μ updated by global heatbath**

- **Most time taken by U_μ and H_μ evolution (Molecular Dynamics eqs, $d\mathcal{H}/dt = 0$)**
  Integration errors corrected by a Metropolis accept-reject step

$$U_\mu(n, t+\delta t) = e^{i\delta t H_\mu(n,t)} U_\mu(n,t)$$
$$H_\mu(n, t+\delta t) = H_\mu(n,t) + \delta t \dot{H}_\mu(n,t)$$



- **Heaviest task during trajectory: matrix inversion $(MM^\dagger)^{-1}\Phi$, needed for $\dot{H}_\mu$:**
  - **A conjugate gradient algorithm is used typically**
  - **The condition number of $MM^\dagger$ rapidly increases at low quark masses**
  - **$m_u, m_d \ll \Lambda_{\mathrm{QCD}}$ hence the inversion can take more then 90% of total time**
  - **Matrix inversion also needed to compute observables**

The first seminal papers on the implementation of lattice QCD on GPUs:

Egri et al. hep-lat/0611022 "Lattice as a video game"

OpenGL was used as a programming language. Sustained performance of ~ 30 GFLOPs for the Wilson kernel (fermion matrix multiplication) on an NVIDIA 8800 GTX.

The advent of the CUDA programming language brought many other groups into the GPUs play.

- **C.Rebbi et al. (LATTICE08) "Blasting through Lattice Calculations using CUDA" Wilson kernel 100 GFLOPs**
- **Kenji Ogawa (TWQCD) (Workshop GPU supercomputing 2009, Taipei) Wilson kernel 120 GFlops**
- **K. Ibrahim et al. "Fine-grained parallelization of LQCD kernel routine on GPU" Speedup 8.3x on 8800GTX (Wilson kernel)**
- **M. A. Clark et al., arXiv:0911.3191 "Solving Lattice QCD systems of equations using mixed precision solvers on GPUs" up to 150-200 Gflops for Wilson kernel on a GeForce GTX 280**

  M. A. Clark et al., arXiv:1011.0024 "Parallelizing the QUDA Library for Multi-GPU Calculations in Lattice QCD" up to 4 Tflops for Wilson kernel on a cluster of 32 NVIDIA GTX 285
- **plus many others, unlisted in the last couple of years, not all of them for production, mostly for measurements (just Dirac operator inversion)**

3 – OUR IMPLEMENTATION

A few years ago we have decided to port our code for the simulation of QCD with standard staggered fermions to GPU

New code for QCD with staggered quarks written from scratch.

Philosophy: GPU not just an accelerator, almost whole code runs actually on it.

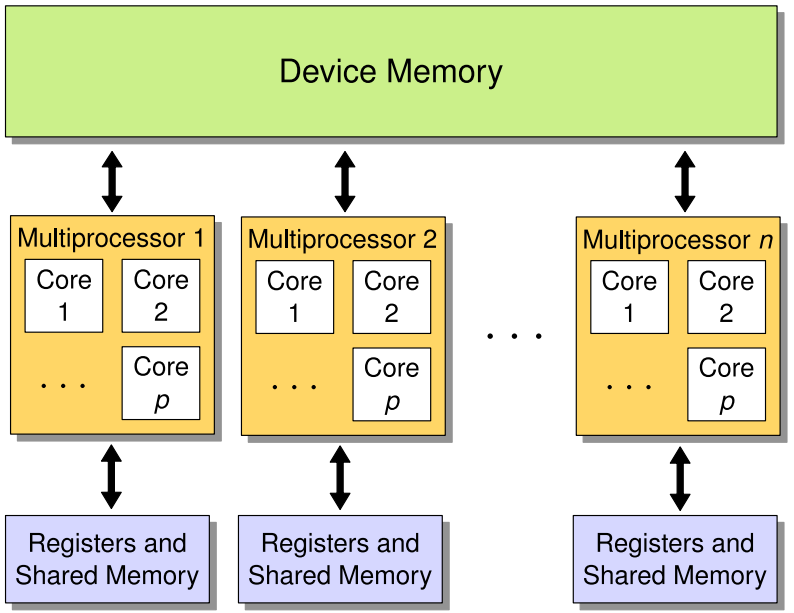
Efficiency around 10%

C. Bonati, G. Cossu, M. D'E. and A. Di Giacomo, "Staggered fermions simulations on GPUs," arXiv:1010.5433

C. Bonati, G. Cossu, M. D'Elia, P. Incardona, "QCD simulations with staggered fermions on GPUs," Comput. Phys. Commun. 183, 853 (2012) [arXiv:1106.5673 [hep-lat]].

Remember that a GPU is made of many cores having access to a global device memory with a bandwidth O(100) GB/s. This is one first bottleneck for problems with a low computations/data_loading ratio, such as lattice QCD (typical performances are around 10%).

A more serious bottleneck is the connection of the device memory to the host RAM, which goes through a PCI express bus at 5 GB/s



OUR IMPLEMENTATION - general features

- **Most lattice QCD applications use GPUs as accelerators for specific demanding parts of the code, e.g. the matrix inversion or some expensive measurements.**

- **Our philosophy has been that of reducing as much as possible the CPU/GPU data exchange by putting most of the Monte-Carlo chain on the GPU.**

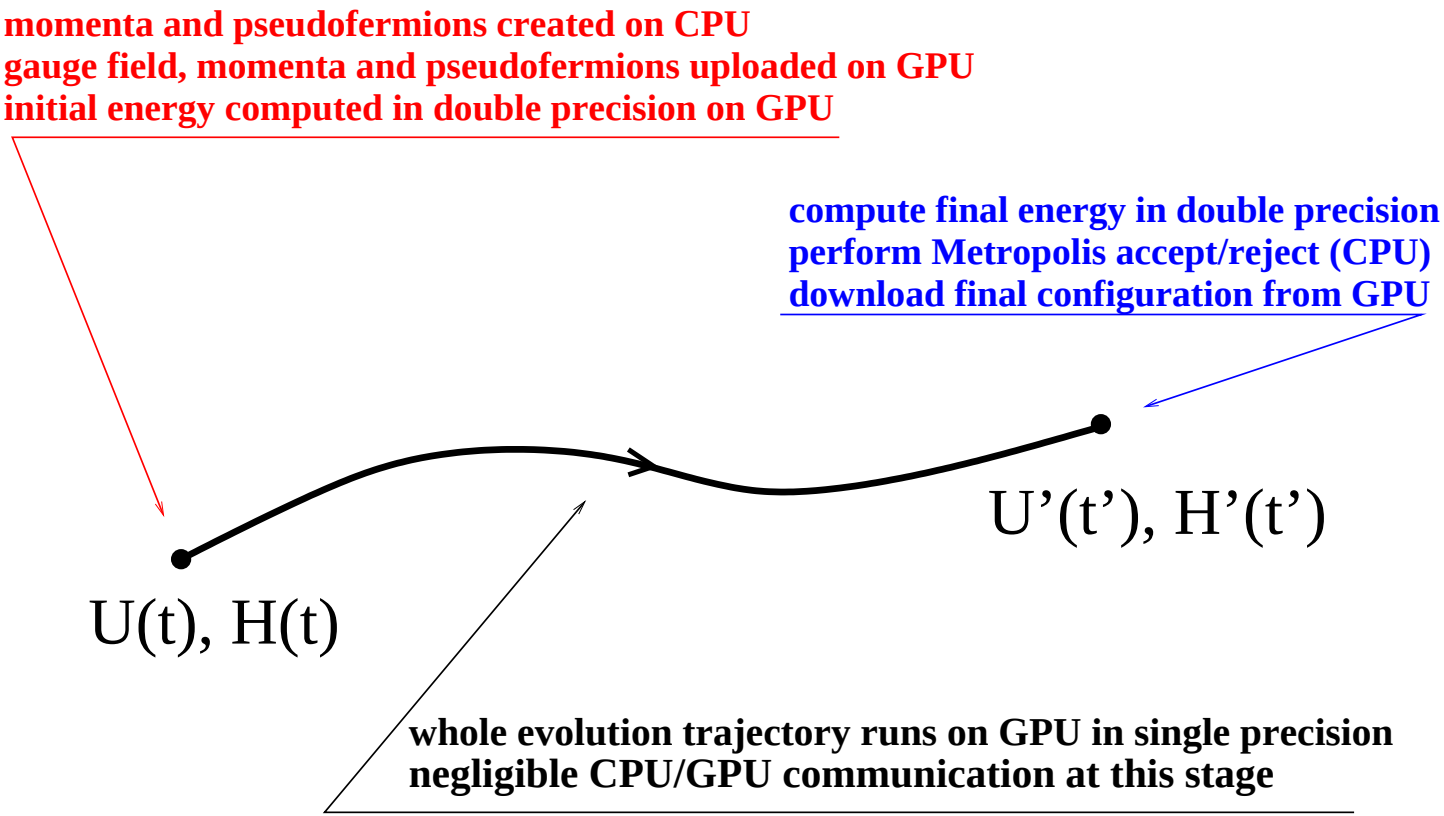
- **That has been done gradually (first we have put the inverter on the GPU, then gradually every other piece). Asymptotically the CPU becomes not more than a mere controller of the GPU flow**

  GPU is the computer ...

- **Single precision floating point arithmetic always outperforms the double one**

  Therefore we make use of double precision only when strictly necessary.

Typical structure of a molecular dynamics trajectory



OUR IMPLEMENTATION - fine structure

$M\Phi$ (Dirac Operator) Kernel

- **Parallelization: each thread reconstructs $M\Phi$ on one site i.e. $\sum_\mu U_\mu(n) \times \Phi(n+\hat{\mu})$**
- **Gauge and pseudofermion fields from CPU to threads:**
  - **Only first two rows of each SU(3) gauge matrix are passed from host $\rightarrow$ device global (texture) memory and from there to threads, to reduce memory exchange. Last row reconstructed during computation.**
  - **Reordering of gauge variables stored on global memory necessary to guarantee coalesced memory access (contiguous threads read contiguous memory locations). This is strictly necessary to avoid access latencies which disrupt performance**
  - **pseudofermions to global memory with reordering as well**

$u_{11}(1)$	$u_{11}(2)$	$u_{11}(3)$	$\cdots$	$\cdots$	$u_{12}(1)$	$u_{12}(2)$	$u_{12}(3)$	$\cdots$	$\cdots$
$\cdots$	$u_{22}(1)$	$u_{22}(2)$	$u_{22}(3)$	$\cdots$	$\cdots$	$u_{23}(1)$	$u_{23}(2)$	$u_{23}(3)$	$\cdots$

Figure 1: Gauge field storage model adopted to achieve coalesced memor access. All u_{ij} elements of gauge matrixes are stored contiguously.

OUR IMPLEMENTATION - Inverter performance

Staggered Dirac operator kernel performance figures on a C1060 card (single prec.).

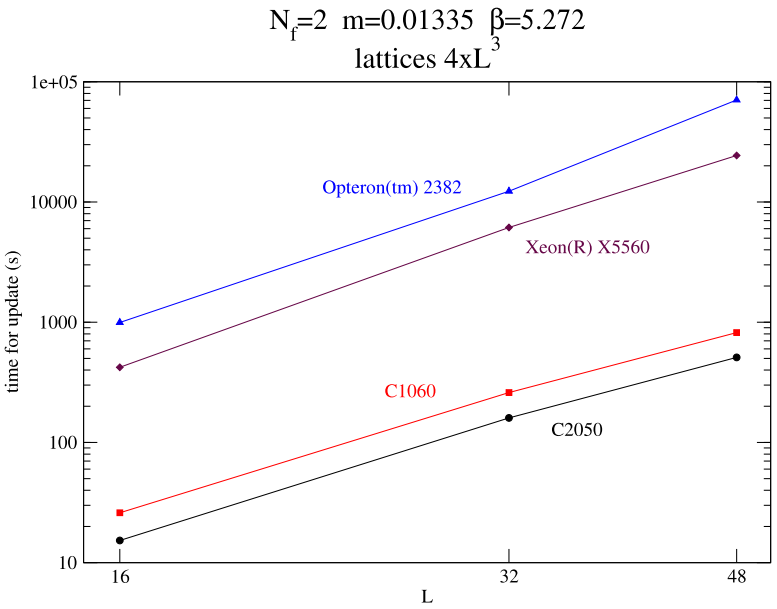
Lattice	Bandwidth GB/s	Gflops
4×16^3	56.84 ± 0.03	49.31 ± 0.02
32×32^3	64.091 ± 0.002	55.597 ± 0.002
4×48^3	69.94 ± 0.02	60.67 ± 0.02

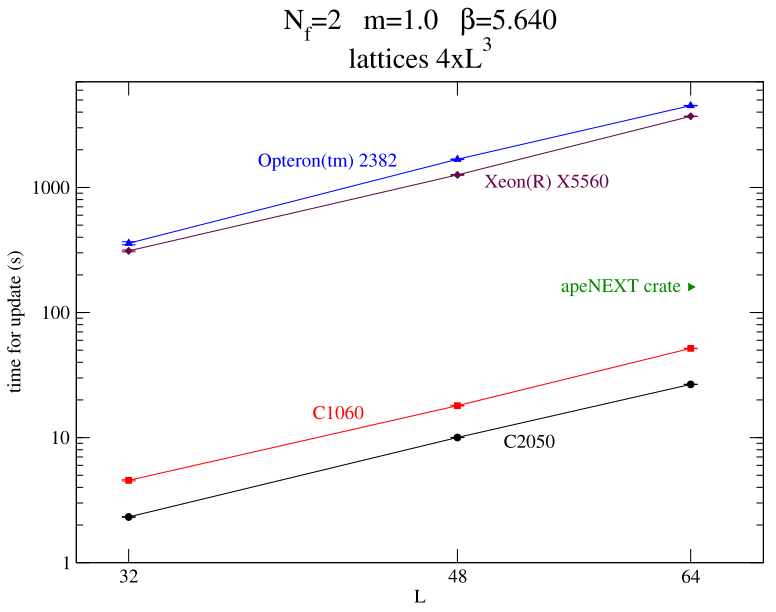
Note that we reach sustained 60 GFLOPs (7% performance) and 70 GBytes/s (70 % bandwidth peak)): no much room for further improvement. Similar numbers are achieved by other groups.

Main reason: the staggered fermion kernel needs more than 1 transferred byte for each floating point operation.

The situation is better by about a factor 2 for Wilson fermions.

Global performance





Run times on different architectures. For the Opteron and Xeon runs a single core was used.

On Kepler, a factor $\sim$ 1.5 is gained with respect to Fermi.

PHYSICS PROJECTS:

Phase diagram of strong interactions, color confinement, deconfinement, non-zero baryon density, non-zero chemical potential and background fields.

Phenomenological interest for Early Universe evolution and for heavy ion collisions.

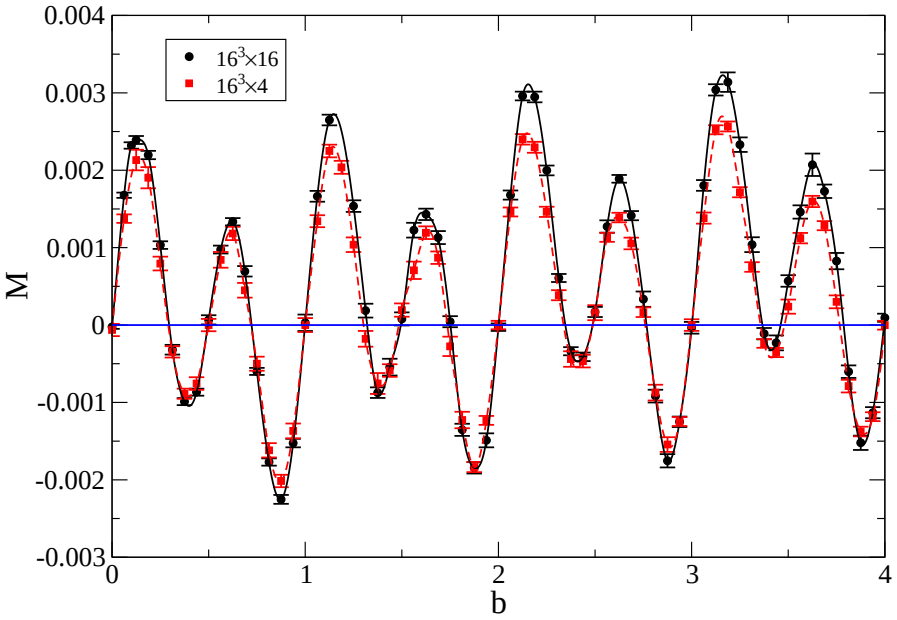
Features: lattices limited in temporal size (finite temperature); many different simulations needed to explore the phase diagram

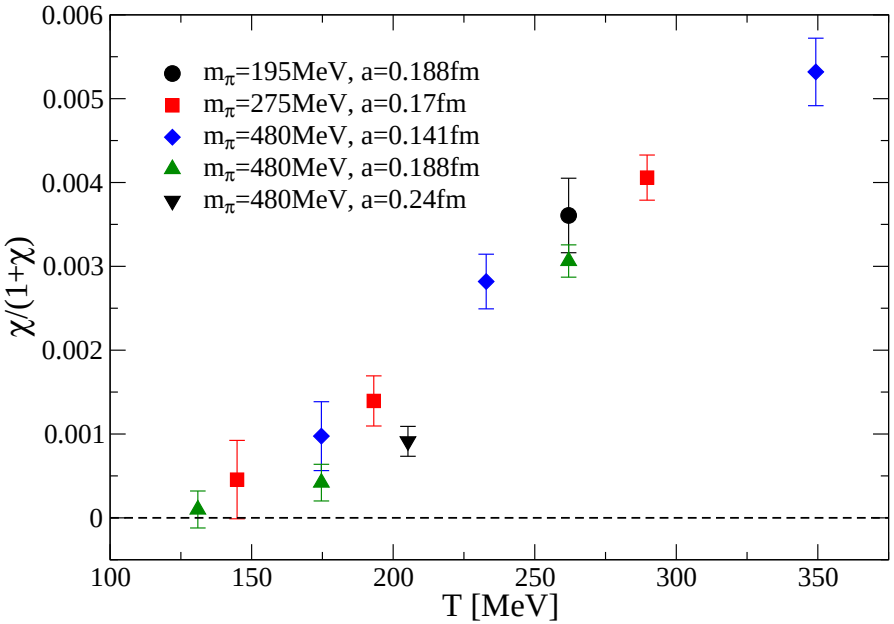
Papers based on our present GPU implementation

- C. Bonati, G. Cossu, M. D. and F. Sanfilippo, "The Roberge-Weiss endpoint in $N_f = 2$ QCD," Phys. Rev. D 83, 054505 (2011).
- C. Bonati, P. de Forcrand, M. D., O. Philipsen and F. Sanfilippo, "Constraints on the two-flavor QCD phase diagram from imaginary chemical potential," arXiv:1201.2769 [hep-lat]; update at Lattice 2013 and forthcoming publication.
- M. D., M. Mariti and F. Negro, "Susceptibility of the QCD vacuum to CP-odd electromagnetic background fields," Phys. Rev. Lett. 110:082002 (2013). + talk at Lattice 2013
- C. Bonati, M. D., M. Mariti, F. Negro, F. Sanfilippo, "Magnetic susceptibility of strongly interacting matter across deconfinement", talk at Lattice 2013 and forthcoming publication

GPU hardware: GPU farms at INFN Pisa (30 cards), Genova (14), Bari (2); QUONG cluster in Rome

An example: determination of the magnetic susceptibility of the Quark-Gluon Plasma





- **TASK: perform several simulations at interpolating values of an external magnetic field $\vec{B}$, integrate, obtain free energy differences between different magnetic field quanta.**
- **Finally, the magnetic susceptibiility is obtained from the quadratic term in B in the free energy.**
- **Results (anteprima): strongly interacting matter becomes a strong paramagnet right after crossing the deconfinement transition**

LINES OF FUTURE DEVELOPMENT:

We aim at reaching accurate determinations on $32^3 \times 8 \ \rightarrow 64^3 \times 16$ lattices.

RHMC algorithm with improved fermionic actions and physical parameters.

Scaling to larger scale structures (not single GPU) is unavoidable: memory requirements increase with lattice size and with action and algorithmic improvement.

Large scale finite T simulations may require up to $O(100)$ GB RAM on state of the art lattices ($a \leq 0.1$ fm if $L_t \geq 12$, hence a $64^3 \times 16$ is state-of-the-art)

Currently, for larger lattices with improved actions, we are exploiting the CSNIV cluster and BG/Q resources. However, for finite T physics, a machine with 32-64 GPUs would work equally well.

OUR FIRST STEPS WITH MULTI-GPUS

Preliminary results reported in Comput. Phys. Commun. 183, 853 (2012).

lattice size	1 GPU	2 GPUs	4GPUs
4×64^3	239	134	95
4×96^3	800^*	421^*	249

Table 1: NVIDIA C1060 update time (in seconds) by using 1, 2 or 4 GPUs (CUDA implementation). The numbers denoted by * are extrapolated from simulations performed on smaller lattice sizes because of the impossibility to allocate the corresponding large lattices in the device memory.

We are also looking at the developments by the MILC collaboration: they have an α-version of their production code running on multiGPUs.

Leonardo Cosmai has recently started to test it.

We hope developments on both sides in the next few months.