LATTICE QCD computations on GPUs

Massimo D'Elia

Università di Genova & INFN

Stato e Prospettive del Calcolo Scientifico - Laboratori Nazionali di Legnaro
16-18 febbraio 2011

1 – Lattice QCD and GPUs

- **Lattice QCD is among the most computationally demanding research topics in theoretical physics.**

- **By its continuous need for powerful computational resources, it has often stimulated the development of new hardware facilities for High Performance Computing (partial list: APE machines series, QCDOC, QPACE, ...)**

- **The aim is the numerical computation of of Quantum ChromoDynamics, the theory of strong interactions, by evaluating its Feynman path integral discretized on a space-time lattice.**

- **The final goal is uncovering the non-perturbative properties of QCD, including confinement, chiral symmetry breaking, the computation of Standard Model parameters and the structure of the QCD Phase Diagram.**

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

- **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** $\sim 10^8 - 10^9$

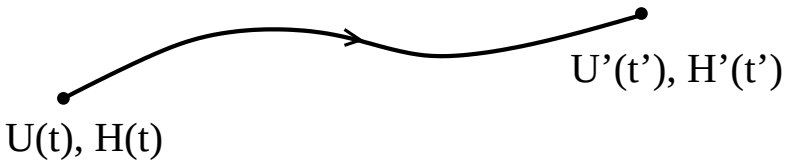
The typical algorithm: Hybrid Monte Carlo

- **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 on the sampled gauge configurations (e.g. quark propagators)**

HOW COSTLY?

Latest estimate, based on improved algorithms and discretizations, of the numerical cost for 100 statistically independent gauge configurations for 2 Wilson fermions:

L. Del Debbio, L. Giusti, M. Lüscher, R. Petronzio, N. Tantalo 2006

$$0.03 \left(\frac{L_s}{3\ \mathrm{fm}}\right)^5 \left(\frac{L_s}{2L_t}\right) \left(\frac{0.2}{\hat{m}/m_s}\right) \left(\frac{0.1\ \mathrm{fm}}{a}\right)^6 \mathrm{TFlop} \cdot \mathrm{year}$$

- Putting in reasonable numbers one easily reaches order of $10 - 100$ Tflops · year.

- Numbers grow easily for specific requirements (heavy quark physics, exact chiral symmetry via Ginsparg-Wilson fermions) going up to 1-10 Petaflops · year.

- Such power is or will soon become available in the next future, but only for a few large groups. Problem is not only hardware but also power consumption.

In this context, similarly to many other computational fields, the advent of GPUs represents an ongoing breakthrough. It makes possible the availability of enough computational power at low cost to a large number of groups and is therefore essential for the development of new ideas and of progress in the field.

The first seminal paper 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 has brought many other groups into the GPUs play. This is only a partial list of contributions

- **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 other ~ 10 unlisted contributions to the last LATTICE 2010 Conference in Sardinia.**

2 – OUR IMPLEMENTATION

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

C. Bonati (Pisa), G. Cossu (KEK, Japan), A. Di Giacomo (Pisa), M. D'E. (Genova), P. Incardona (Genova)

People in red have done or are doing most of the heavy work.

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

Our present installation

- **6 S1070 units (= 24 C1060) up and running in Pisa**
- **2 S1070 units up and running in Genoa**
- **4 C2050 (Fermi) cards to be installed in Genoa**

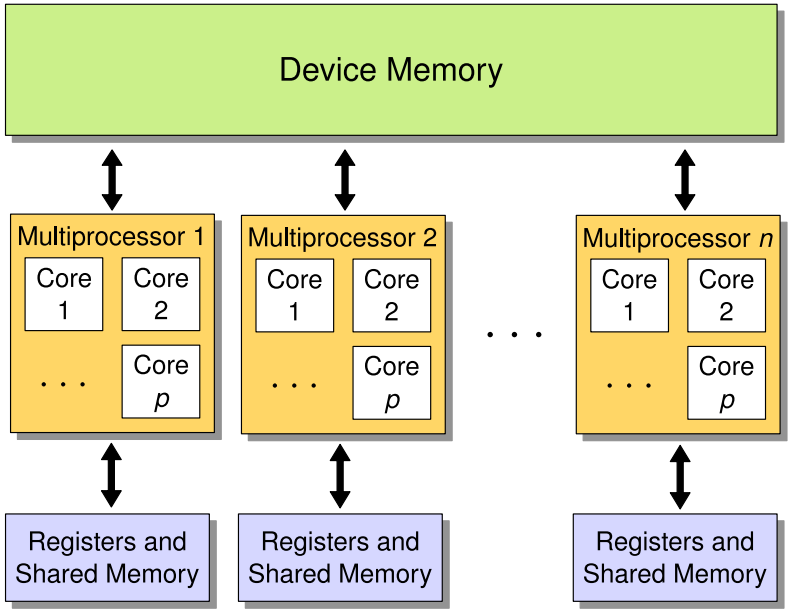
Specification of the NVIDIA cards used for our benchmarks

GPU	Cores	Bandwidth GB/s	Gflops (peak) single	Gflops (peak) double	Device Memory GB
Tesla C1060	240	102	933	78	4
Tesla C2050*	448	144	1030	515	3

* thanks to Edmondo Orlotti and Massimo Bernaschi for making a C2050 card available for our tests

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.

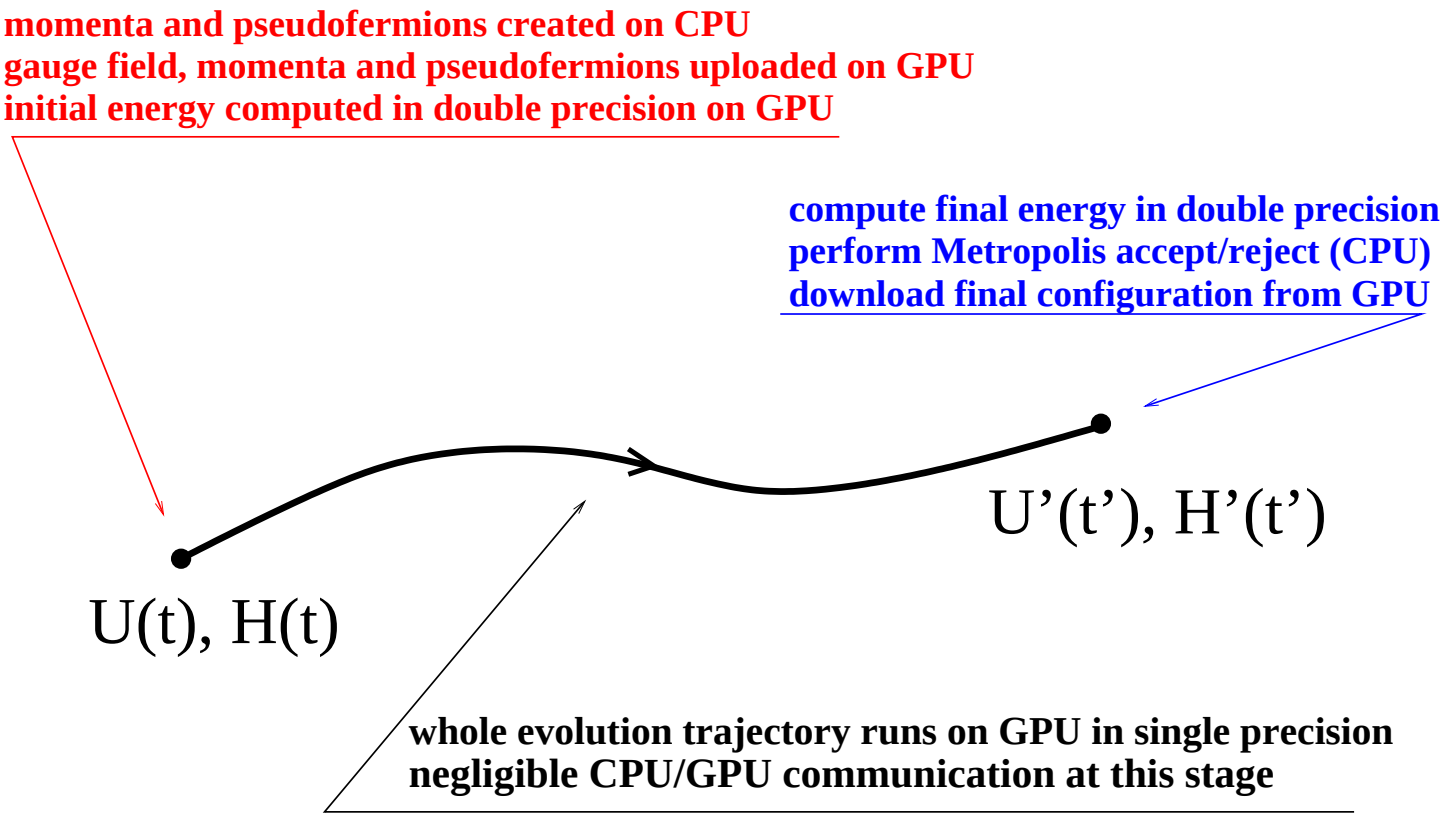
- That has been done gradually (first we have put the inverter on the GPU, then gradually every other piece). Asymptotically the CPU becomes not more than a mere controller of the GPU flow

  GPU is the computer ...

- Single precision floating point arithmetic always outperforms the double one (although in the Fermi architecture such problem is strongly reduced).

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

Sketch of our implementation



Everything is implemented by a homemade C code supplemented with CUDA kernels

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 precision).

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

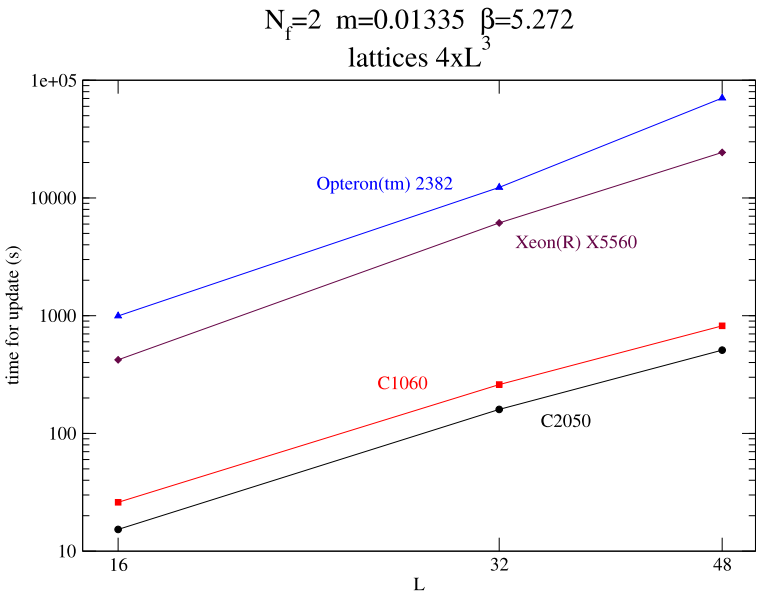
We have tested our code in two different regimes: 2 light flavors ($am_q \simeq 0.01$) and 2 heavy flavors ($am_q = 1$), corresponding to a different incidence of the Dirac kernel performance.

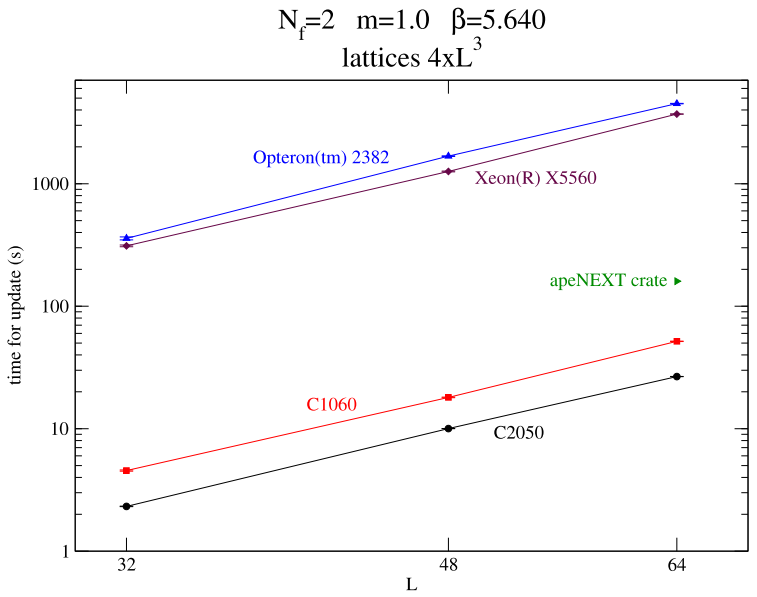
NVIDIA C1060 time gains over CPU and apeNEXT.

	high mass			low mass		
spatial size	32	48	64	16	32	48
Opteron (single core)	65	75	75	40	50	85
Xeon (single core)	50	50	50	15	25	30
apeNEXT crate	∼3			∼1		

Same for NVIDIA C2050 (same code as for C1060, no specific C2050 improvement).

	high mass			low mass		
spatial size	32	48	64	16	32	48
Opteron (single core)	115	130	140	65	75	140
Xeon (single core)	85	85	100	30	40	50
apeNEXT crate	∼6			∼2		





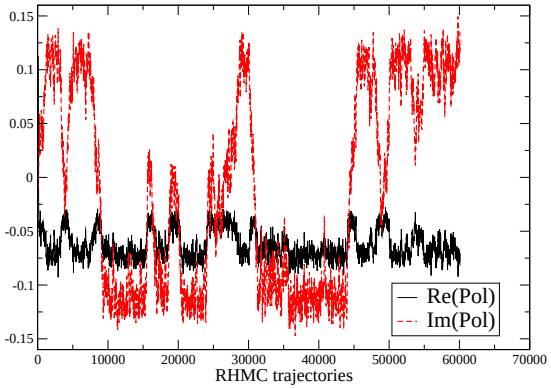
Run times on different architectures. For the Opteron and Xeon runs a single core was used.

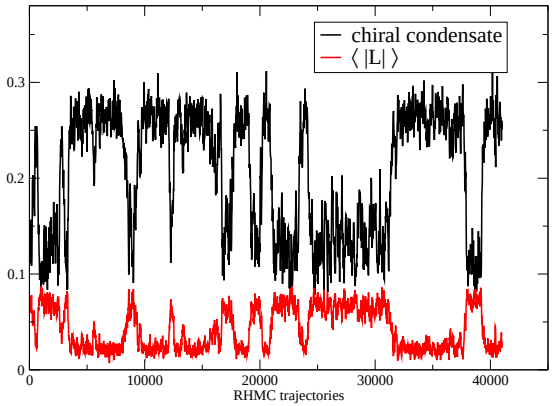
STABILITY:

We have been running a Monte-Carlo chain continuously on a C1060 for more than 100 days without a crash

3 – Production Runs

- **Our code is running for production on our installation since last summer.**
- **Main topics: QCD phase diagram and confinement, nature deconfinement transition at zero and non-zero baryon density, QCD in background magnetic fields.**
- **First production results:** C. Bonati, G. Cossu, M. D'Elia, F. Sanfilippo "The Roberge-Weiss end-point in $N_f = 2$ QCD," arXiv:1011.4515 **(Exploration of the QCD phase diagram)**





4 months of run. Would have taken 2-3 years on our apeNEXT resources (and likely published before by other groups)

- **More results coming this year.**

4 – Planned Future Developments

- **Extension to multiGPU:**
  - Present code runs on a single GPU. Optimal for lattices as large as $48^3 \times 4$
  - multiGPU extension unavoidable for large scale simulations.
  - Main difficulty: frequent communications in and out of the GPU unavoidable, but they go through the PCI express bottleneck (5GB/s). Possible solutions:
    - Software: overlapping communications and computations (see e.g. M. A. Clark et al., arXiv:1011.0024, they reach a $10\times$ acceleration on a 32 GPU cluster)
    - Algorithm: use algorithms which require minimal communication among different lattice portions, like the Schwarz domain decomposition algorithm by Lüscher (see e.g. Osaki - Ishikawa, arXiv:1011.3318, authors gain a factor 2)
    - Improve Communications among GPUs: future CUDA features and dedicated communication links (see talk by Davide Rossetti on APEnet)
  - Starting collaboration on this topic with the APE group in Rome
- **Porting our code also to OpenCL, in order to test different hardware possibilities**
- **Extension to dynamical overlap fermions:**

  needed to implement exact chiral symmetry on the lattice, they are one of the most computationally demanding challenges for future simulations