Lattice QCD simulations on GPUs

C. Bonati (Pisa), G. Cossu (KEK), M. D'Elia (Genova)

A. Di Giacomo (Pisa), P. Incardona (Genova)

SMFT 2011 - BARI **September 22, 2011**

1 – Lattice QCD: Computational Complexity

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

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

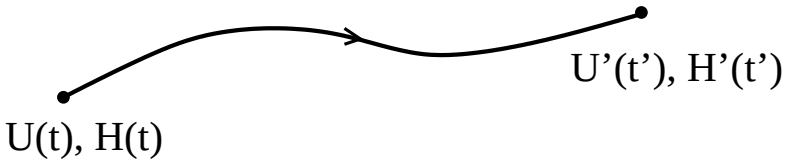
The typical algorithm: Hybrid Monte Carlo

- **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, inversion can take most of the total time**
  - **Matrix inversion also needed to compute observables on the sampled gauge configurations (e.g. quark propagators)**

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: the renowned series of APE machines, QCDOC, QPACE, ...

HOW MUCH DEMANDING?

Latest estimate, based on improved algorithms and discretizations, of the numerical cost for 2 Wilson fermions on a $L_s^3 \times 2L_s$ lattice:

L. Del Debbio, L. Giusti, M. Lüscher, R. Petronzio, N. Tantalo, JHEP 0702, 056 (2007)

$$0.05 \left(\frac{\#\ \text{Indep. Confs}}{100} \right) \left(\frac{L_s}{3\ \text{fm}} \right)^5 \left(\frac{20\text{MeV}}{\bar{m}_q} \right) \left(\frac{0.1\ \text{fm}}{a} \right)^6 \text{TFlop} \cdot \text{year}$$

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

- Numbers grow easily for specific requirements (heavy quark physics, exact chiral symmetry via Ginsparg-Wilson fermions) going to the Petaflop scale.

- **Such large amount of computational power is, or will soon become available in the next future, to a few big groups in the world, thanks to large scale parallel supercomputers, like BlueGene, built in the same spirit as for APE machines: network of dedicated (multi)-processors with fast intercommunications.**
  Most tasks in a lattice simulation are ideally suited for a SIMD parallelization with local communications.

- **In this context the advent of Graphics Processing Units (GPUs), with their many-core architectures, represents an ongoing breakthrough, like for other fields of High Performance Computing:**
  - **Present GPUs make medium size computational power (1-10 Teraflops) available at low cost to a large number of groups: that is essential for the development of new ideas and for faster progress in the field.**
  - **They have the potential for being efficient computing nodes in large scale supercomputing facilities.**

- **In the following I will illustrate the potential of GPUs for lattice QCD simulations, based on our recent experience.**

2 – A few words on GPUs

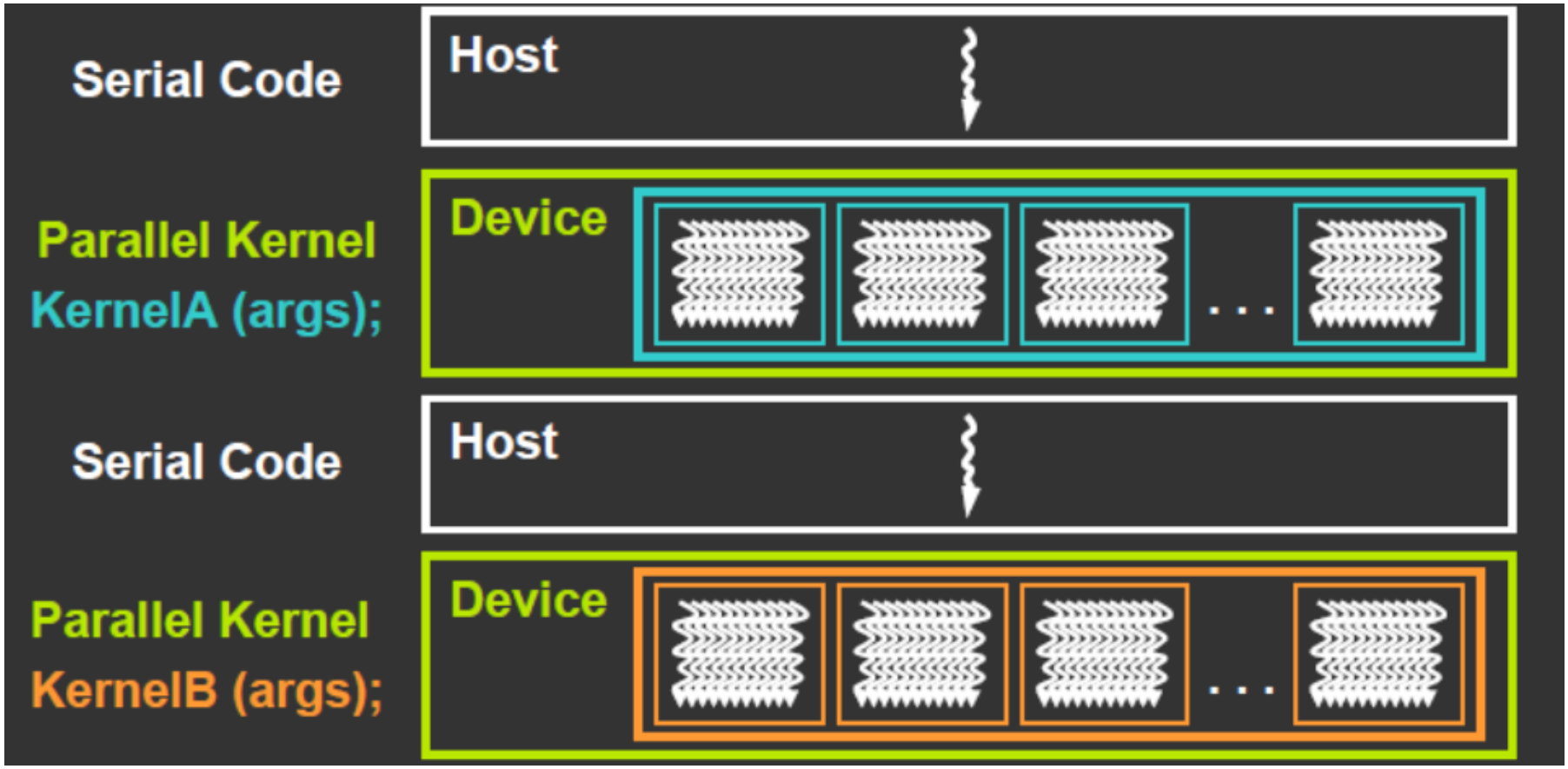
- GPUs are originally meant as graphic coprocessors

- They have recently developed to general purpose computational facilities: due to their many-core architecture they are perfectly suited for numerical tasks allowing SIMD parallelization.
  - Lots of math units
  - Fast access to on-board memory
  - Run the same program (thread) on many cores

- The advent of CUDA (Compute Unified Device Architecture) on NVIDIA GPUs, allowing for an easy programming interface, has been a major boost in the last 2-3 years.

As a matter of fact, a GPU is still a coprocessor (accelerator)

The main (serial) program runs on the CPU (host), which sometimes launches parallel kernels to be executed on the GPU (device), in order to accelerate specific portions of the code.



Lattice QCD and GPUs

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.**

3 – OUR IMPLEMENTATION (see arXiv:1106.5673)

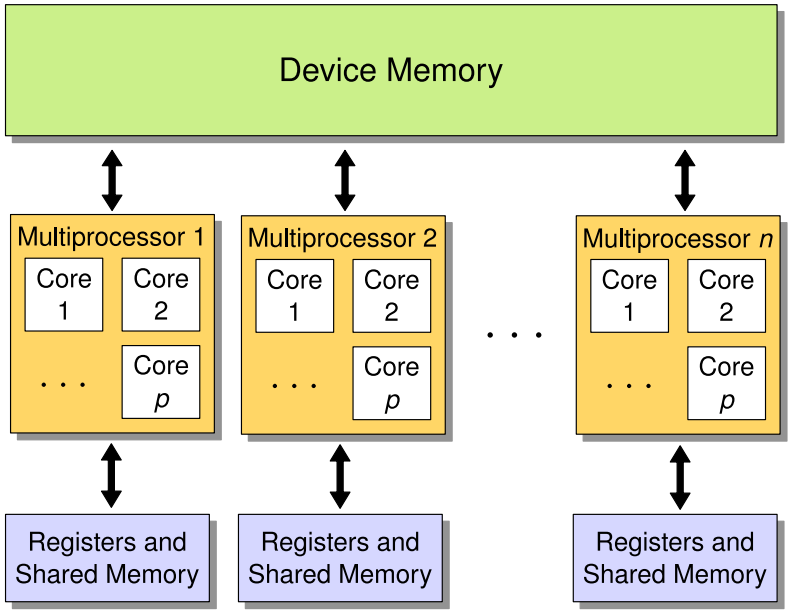
We have ported to GPU a code simulating QCD with standard staggered fermions

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

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.

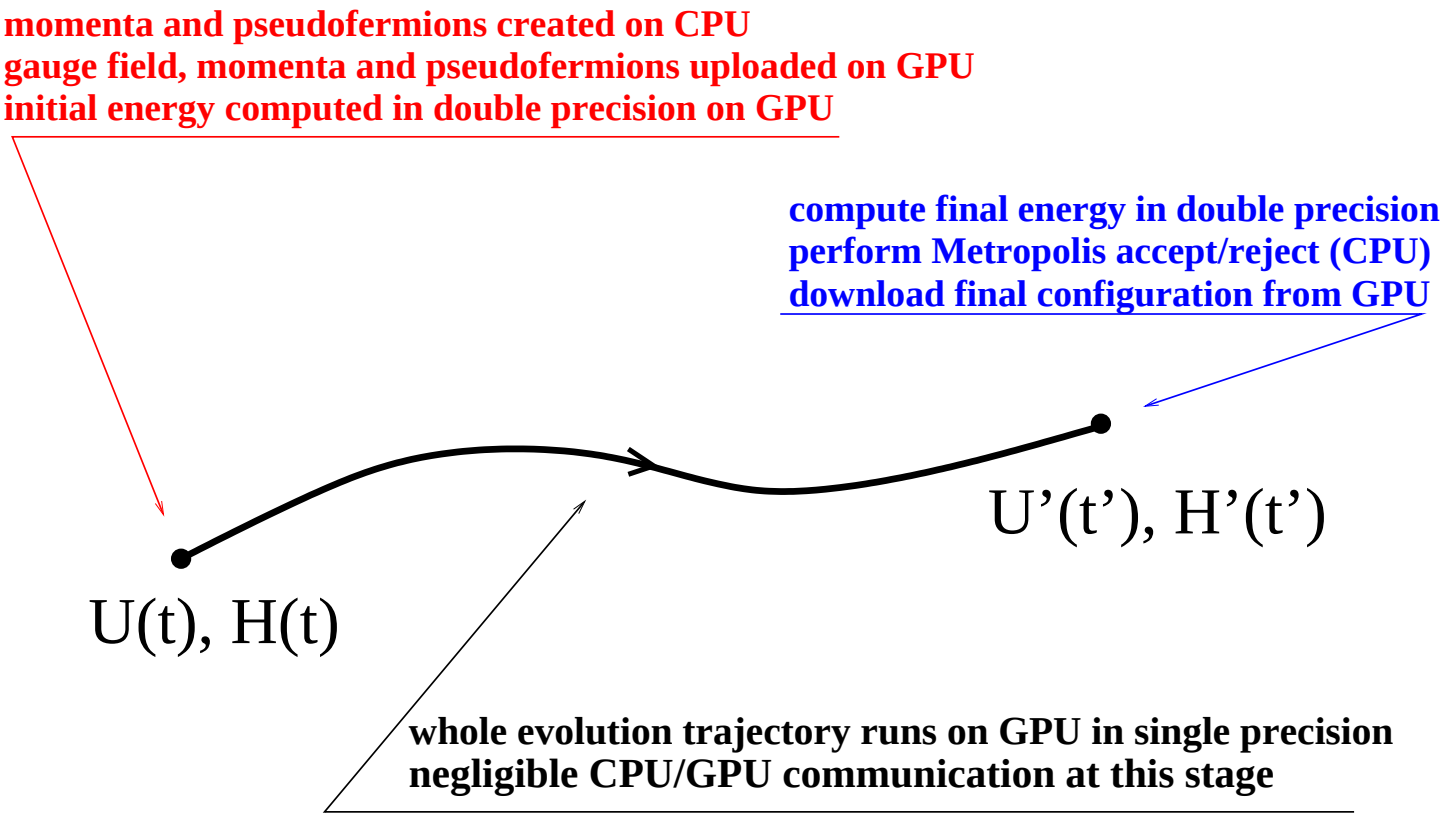
- That has been done gradually (first we have put the inverter on the GPU, then gradually every other piece). Asymptotically the CPU becomes not more than a mere controller of the GPU flow

  The 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

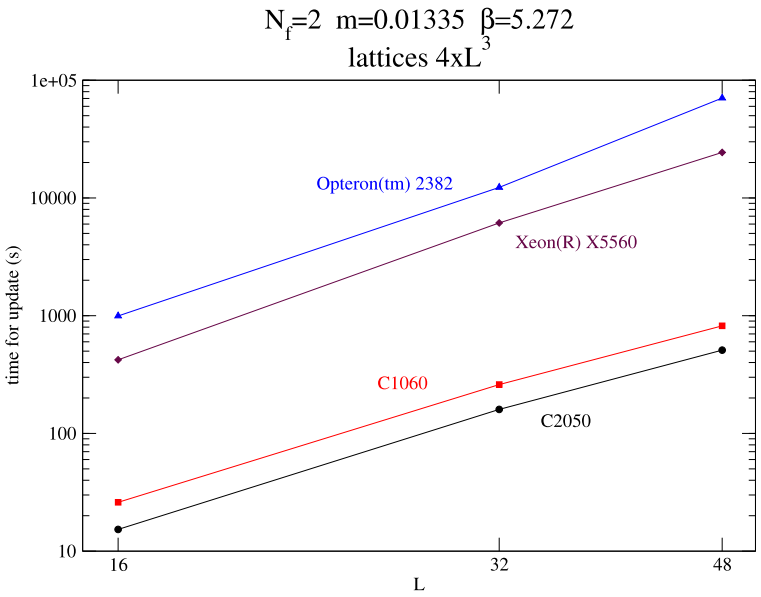
We have tested our code in two different regimes: 2 light flavors ($am_q \simeq 0.01$) and 2 heavy flavors ($am_q = 1$), corresponding to a different incidence of the Dirac kernel performance.

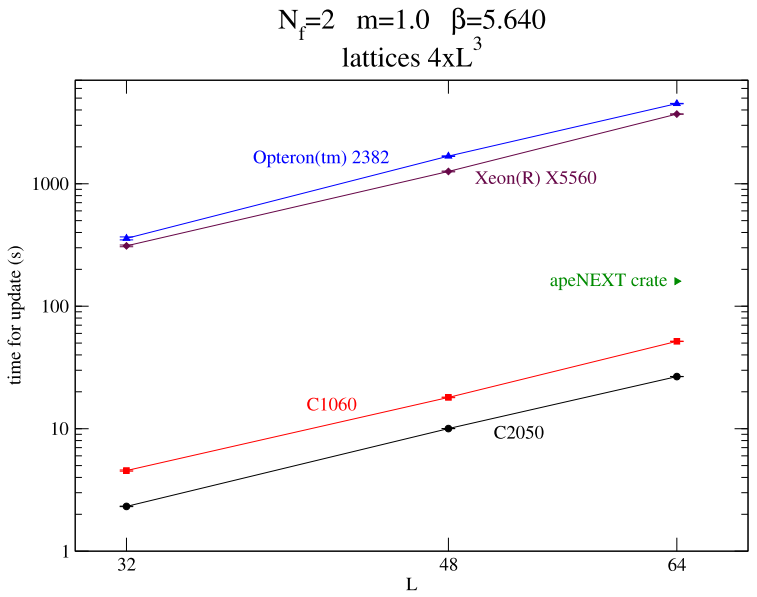
C1060 time gains over Xeon X5560 and Opteron 2382 (both single core) and apeNEXT crate (256 nodes)

	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 Opteron and Xeon we refer to single cores.

Why high mass more efficient? Pure gauge cost predominant with respect to dynamical fermions cost: computations/data_loading ratio much more favourable.

- **pure gauge: mostly matrix-matrix multiplications. For SU(3) pass 2+2 rows (12 complex numbers) to make 27 complex-complex multiplications.**
- **dynamical part: mostly matrix-vector multiplications. For SU(3) pass 2 rows + 1 vector (9 complex numbers) to make 9 complex-complex multiplications.**

	C1060			C2050		
spatial size	32	48	64	32	48	64
gain factor over Xeon (single core)	80	90	120	135	145	210

Table 1: NVIDIA C1060 and C2050 time gains over CPU for the pure gauge part sections of the code (link evolution and pure gauge contribution to momenta evolution in molecular dynamics).

4 – Production Runs

- **We have started our GPU adventure mainly because we were lacking adequate computer resources for our physics program**
  - **Nature of color confinement and of the deconfinement transition**
  - **Phase diagram of QCD (high T, finite density, strong background fields)**
  - **Properties of deconfined matter**
- **Our main pre-GPU resources consisted of 4 apeNEXT crates: about 1 Teraflop, i.e. two order of magnitudes less than competing groups in the world.**
  Now we have gained one order of magnitude and we have already started intensive production runs since last summer.
- **First production activities and results:**
  - **Order of the $N_f = 2$ transition in the chiral limit**
  - **QCD phase diagram at finite chemical potentials and in background magnetic fields (see e.g. results in C. Bonati, G. Cossu, M. D'E., F. Sanfilippo "The Roberge-Weiss endpoint in $N_f = 2$ QCD," Phys. Rev. D 83, 054505 (2011) [arXiv:1011.4515]**
  - **Mid-long term project: implementation of exact chiral symmetry via overlap fermions**

5 – Ongoing Developments

OPENCL implementation

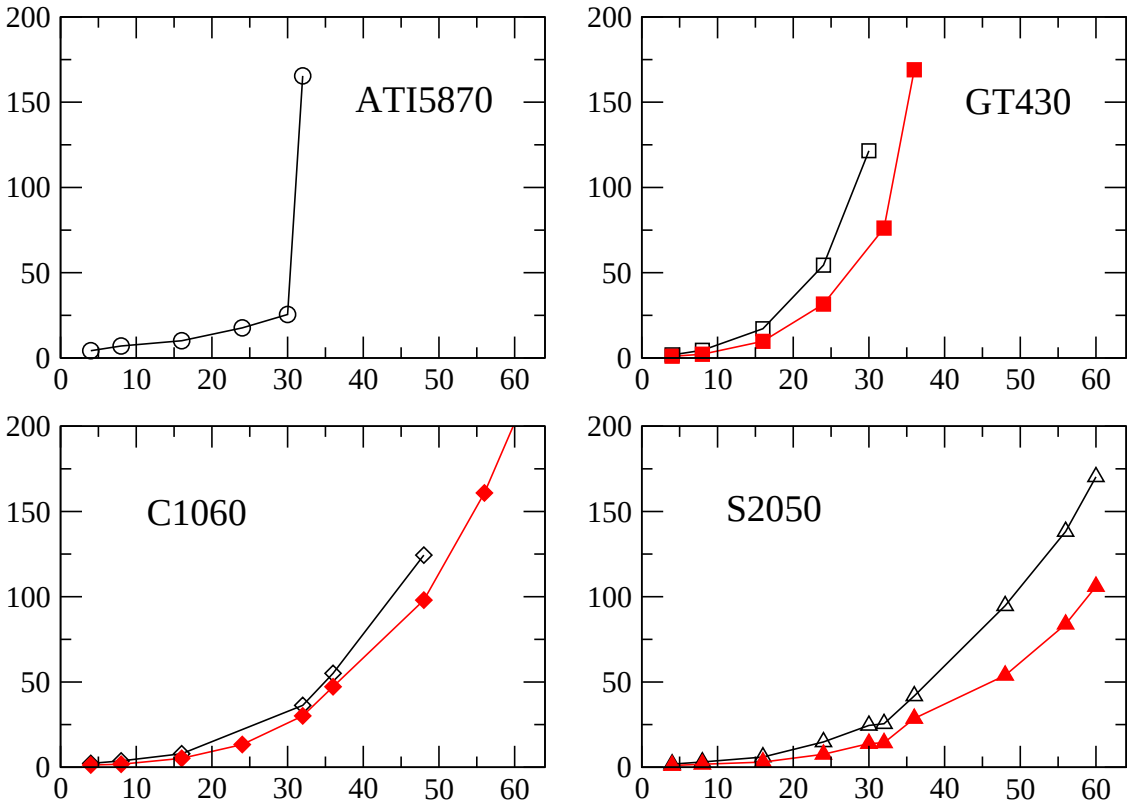


Figure 2: **Update time (in seconds) for a lattice $L_s^3 \times 4$ (the spatial dimension L_s is on the abscissa) for a theory with two fermions of mass $m = 0.1$ at coupling $\beta = 5.59$ and for various GPUs. Results obtained with the OpenCL implementation are shown by empty symbols, while full symbols represents the CUDA results.**

Multi-GPU implementation

- **Present simulations run on a single GPU. Optimal for our running projects on lattices as large as $48^3 \times 4$**
- **MultiGPU extension unavoidable for simulations at $T = 0$ and also at $T \neq 0$ when approaching the continuum limit (finer lattices)**
- **We have tested a preliminary multiGPU extension of our code. The lattice is partitioned along a single direction. Multidirectional partitioning and multiCPU (MPI) extension in progress**

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

Table 2: 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 hope to have more encouraging results in the next future!

THANK YOU!