Karl-Franzens-Universität Graz
Institut für Physik, Fachbereich Theoretische Physik

Gauge Fixing in Lattice QCD on GPUs

Mario Schröck

Introduction

Gauge dependent correlators like the fundamental two-point functions may only be studied after fixing the $\mathrm{SU}(3)$ gauge freedom. Popular gauges are the Landau and the Coulomb gauge which read in Euclidean space,

$$\sum_{\mu=1}^{m} \partial_\mu A_\mu(x) = 0,$$

with m equal to 3 or 4 for Coulomb or Landau gauge, respectively.
On the lattice, removing the gauge freedom is equivalent to maximizing the corresponding gauge functional

$$F_g[U] = \mathfrak{Re} \sum_{\mu,x} \mathrm{tr}\left[U^g_\mu(x) + U^g_\mu(x-\hat{\mu})^\dagger\right]$$

with respect to gauge transformations $g(x) \in \mathrm{SU}(3)$ where

$$U^g_\mu(x) \equiv g(x) U_\mu(x) g(x+\hat{\mu})^\dagger.$$

The lattice gauge fields $U_\mu(x)$ are connected to the continuum ones via $U_\mu(x) = \mathrm{e}^{iaA\mu(x)}$.
The cost of fixing the gauge on the lattice increases exponentially with the lattice volume and is one of the most expensive tasks in extracting physical content out of given gauge configurations.
We show how the process of lattice gauge fixing with the overrelaxation algorithm can be accelerated using NVIDIA's CUDA (Compute Unified Device Architecture) programming environment for GPUs (Graphical Processing Units). We compare performance of the algorithm to conventional calculations on the CPU and present techniques to relax the bandwidth restrictions of the GPU.
For a general discussion of lattice gauge fixing and its problems see, e.g., [1].

Gauge Fixing via Overrelaxation

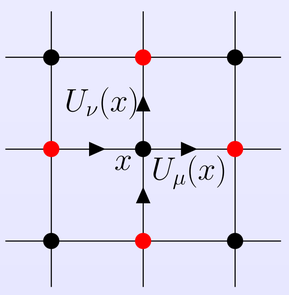
The gauge functional has degrees of freedom of $\mathcal{O}(mN_cV)$ where N_c is the number of colors and $V = L^3 \times T$ is the lattice volume. The idea of the relaxation algorithm is to optimize the value of $F_g[U]$ locally [2], i.e., for all x maximize $\mathfrak{Re}\ \mathrm{tr}\,[g(x)K(x)]$ where we defined

$$K(x) := \sum_\mu \Big(U_\mu(x) g(x+\hat{\mu})^\dagger + U_\mu(x-\hat{\mu})^\dagger g(x-\hat{\mu})^\dagger \Big).$$

The local solution thereof is given, in the case of the gauge group $\mathrm{SU}(2)$, by

$$g(x) = \frac{K(x)^\dagger}{\sqrt{\det K(x)^\dagger}}$$

and for $\mathrm{SU}(3)$ we can iteratively operate in the $\mathrm{SU}(2)$-subgroups [3].



In order to reduce the *critical slowing down* of the relaxation algorithm on large lattices, the authors of [4] suggested to apply an *over*-relaxation algorithm which replaces the gauge transformation $g(x)$ by $g^\omega(x)$ in each step of the iteration. This method has widely been studied and the value of ω was found to well adapted at around 1.7, see references in [1].
The iteration is stopped when $\theta < \varepsilon^2$ where θ is the finite difference approximation of the first derivative of $A_\mu(x)$ averaged over V.

The "Fermi Architecture"

After NVIDIA introduced CUDA in 2006 together with their first generation of hardware supporting this new GPU computing model, the G80 family, the Fermi architecture is now already the third generation and was released in 2009. Its major improvements are: higher double precision performance, ECC (error correction code) support, L1 and L2 caches and more shared memory on the multi-processor level.

For our study we used the NVIDIA GeForce GTX 480 which includes in total 480 cores. Threads are started in warps of size 32 and a multiprocessor handles many warps concurrently.

NVIDIA GeForce GTX 480	
Multiprocessors	15
Cores/MP	32
Cores	480
Global memory	1.5 GB
Shared memory/block	48 KB
Warp size	32
Clock rate	1.40 GHz
Peak performance	1345 Gflops (SP)
Memory bandwidth	177 GB/s

Mapping Lattice QCD to the GPU

Since CUDA supports only lattices of up to 3D natively, we linearize the 4D lattice index using divisions and modulo conversions of V by the spatial and temporal extent of the lattice. We assign each lattice site to a separate thread and start 256 threads per multiprocessor simultaneously.
A function which is called from the host system and which performs calculations on the GPU is called a kernel. We implemented two kernels, one which checks the current value of the gauge fixing functional and the gauge precision after every 100th iteration step and a second which does the actual work, i.e., which performs an overrelaxation step. The latter is invoked for lattice sites of even and odd parity consecutively.

Optimizations

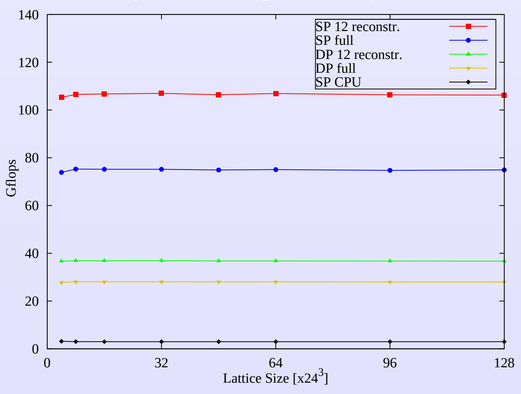
The GPU can read data from global device memory in a fast way only if the data is accurately coalesced; in order to do so we rearrange the gauge field into two blocks, one for even and one for odd lattice sites. Moreover, for the same sake of memory coalescing, we choose the site index running fastest.
Applying the overrelaxation algorithm to one lattice site needs 2253 floating point operations and we have to read and write eight $\mathrm{SU}(3)$ matrices for every site. Each matrix consists out of 9 complex numbers, or 18 reals, and in single precision (4 bytes/real) this sums up to a total data transfer of 1152 bytes per site. Comparing the ratio data transfer per floating point operation, $1152/2253 \approx 0.5$, with the theoretical peak performance of the GTX 480, $177/1345 \approx 0.1$, we clearly see that we are solely constrained by memory bandwidth and not by the maximum number of arithmetic instructions.
In order to reduce memory traffic we use the unitarity of $\mathrm{SU}(3)$ matrices to reconstruct the third line of each matrix on the fly when needed instead of storing it:

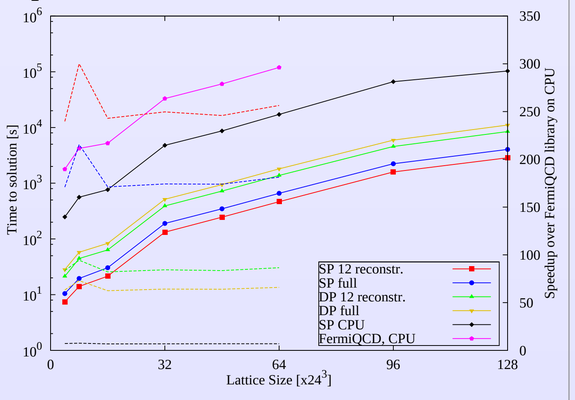
$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} \in \mathrm{SU}(3), \quad \mathbf{c} = (\mathbf{a} \times \mathbf{b})^*.$$

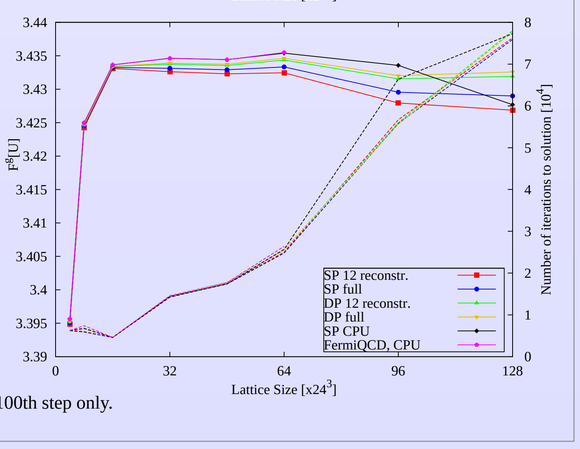
A minimal 8 parameter reconstruction [5] turned out to be numerically not stable enough for our purpose since we not only have to read the gauge fields but also write them in each step of the iteration.
For more details we refer to [5], [6] and references therein.

Results

We compare the performance of our GPU kernels on the GTX 480 in single and double precision to the performance of the same code compiled for the Intel Core i7 processor with the Intel Compiler (Version 12.0) using the compiler flag xSSE4.2.







Furthermore we compare to the FermiQCD library [7] which is an open source library for Lattice QCD applications written in C++.[a] We run both CPU programs on a single core of the Core i7 Nehalem Bloomfield processor. In each case we stop the algorithm when the Landau gauge is reached within a precision of $\varepsilon^2 = 5.0 \cdot 10^{-7}$.

[a] We modified FermiQCD to also check the gauge precision every 100th step only.

Conclusion

The expensive task of fixing the gauge in lattice QCD can greatly be accelerated by the use of GPUs. In single precision we gained a factor of around 250 versus the FermiQCD library and a factor of approximately 35 in comparison to our own optimized code for the CPU. When switching to double precision performance decreases by a factor of 2.9.
A future multi GPU implementation will allow for processing lattices that do not fit into the device memory of a single GPU.

Acknowledgments

This research was supported by the Research Executive Agency (REA) of the European Union under Grant Agreement number PITN-GA-2009-238353 (ITN STRONGnet). Furthermore I want to thank Gundolf Haase and Gernot Lassnig for helpful discussions.

References

[1] M.L. Paciello *et al.*, Int.J.Mod.Phys. **A16**, 3487 (2001)
[2] J.E. Mandula and M. Ogilvie, Phys. Lett. **B185**, 127 (1987)
[3] N. Cabibbo and E. Marinari, Phys. Lett. **B119**, 387 (1982)
[4] J.E. Mandula and M. Ogilvie, Physics Lett. **B248**, 156 (1990)
[5] R. Babich, K. Barros, R.C. Brower, C. Rebbi, Comput.Phys.Commun.**181**, 1517 (2010)
[6] R. Babich, M.A. Clark, Balint Joo, . JLAB-IT-10-01, (2010)
[7] M. Di Pierro *et al.*, Nucl. Phys. Proc. Suppl. **129**, 832 (2004)