Gauge fixing using overrelaxation and simulated annealing on GPUs

Mario Schröck[*]

Institut für Physik, FB Theoretische Physik, Universität Graz, A–8010 Graz, Austria
E-mail: mario.schroeck@uni-graz.at

Hannes Vogt

Institut für Theoretische Physik, 72076 Tübingen, Germany
E-mail: hannes.vogt@uni-tuebingen.de

We adopt CUDA-capable Graphic Processing Units (GPUs) for Coulomb, Landau and maximally Abelian gauge fixing in 3+1 dimensional SU(3) lattice gauge field theories. The local overrelaxation algorithm is perfectly suited for highly parallel architectures. Simulated annealing preconditioning strongly increases the probability to reach the global maximum of the gauge functional. We give performance results for single and double precision. To obtain our maximum performance of 300 GFlops on NVIDIA's GTX 580 a very fine grained degree of parallelism is required due to the register limits of NVIDIA's Fermi GPUs: we use eight threads per lattice site, i.e., one thread per SU(3) matrix that is involved in the computation of a site update.

The 30th International Symposium on Lattice Field Theory
June 24–29, 2012
Cairns, Australia

---

[*]Speaker.

1. Motivation

Lattice QCD is the discrete version of the gauge theory of the strong interaction and by construction it is invariant under local gauge transformations of the form

$$g(x)U_\mu(x)g(x+\hat{\mu})^\dagger. \tag{1.1}$$

The lattice gauge fields or *link variables* $U_\mu(x)$ are connected to the continuum gauge fields via

$$U_\mu(x) = e^{iagA_\mu(x)} \tag{1.2}$$

and thus live in the Lie group SU(3) itself instead of in its algebra. Whereas physical observables are gauge independent, the study of gauge dependent quantities like the fundamental QCD Green's functions requires to fix the gauge, i.e., to choose a specific transformation $g(x) \in$ SU(3) for all x.

Gauge fixing on the lattice corresponds to an optimization problem with $\mathcal{O}(VN_c^2)$ degrees of freedom where $V = N_s^3 \times N_t$ is the lattice volume. Such being the case, the process of fixing the gauge on the lattice demands a major part of the whole simulation's computer time and the possible acceleration by highly parallel hardware architectures like graphics processing units (GPUs) is clearly beneficial. A first attempt of porting lattice gauge fixing with the overrelaxation algorithm to the GPU has been reported in [1]. The relaxation algorithm is particularly well suited to be accelerated by the use of GPUs due to its strict locality which also opens the door to an efficient future multi-GPU parallelization. An alternative approach based on the steepest descent method with Fourier acceleration has been presented in [2].

Here we present a code package for lattice gauge fixing based on the family of relaxation algorithms. The code is written in CUDA C++ and makes heavy use of template classes in order to facilitate the extension to other algorithms and applications. Besides the standard relaxation algorithm [3] our program supports overrelaxation [4] and stochastic relaxation [5] to overcome the problem of critical slowing down. Furthermore, we implemented the simulated annealing algorithm which can be applied as a "preconditioner" to the gauge fields in order to increases the probability to reach the global maximum of the gauge fixing functional [6].

The code can be used to fix gauge configurations to the covariant Landau gauge $\partial_\mu A_\mu = 0$, the Coulomb gauge $\partial_i A_i = 0$ and the maximally Abelian gauge.

In the remainder of this presentation we focus on the overrelaxation algorithm and the Landau gauge as a specific example.

2. Algorithm

On the lattice, gauge fixing is equivalent to maximizing the corresponding gauge functional, in the case of Landau gauge

$$F_g[U] = \mathfrak{Re} \sum_{\mu,x} \text{tr}\left[U_\mu^g(x)\right], \tag{2.1}$$

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

$$U_\mu^g(x) \equiv g(x)U_\mu(x)g(x+\hat{\mu})^\dagger. \tag{2.2}$$

The relaxation algorithm optimizes the value of $F_g[U]$ locally, i.e., for all x the maximum of $\mathfrak{Re}\,\mathrm{tr}\,[g(x)K(x)]$ with

$$K(x) := \sum_\mu \left(U_\mu(x)g(x+\hat{\mu})^\dagger + U_\mu(x-\hat{\mu})^\dagger g(x-\hat{\mu})^\dagger \right) \tag{2.3}$$

is sought. The local solution thereof is given by

$$g(x) = K(x)^\dagger / \sqrt{\det K(x)^\dagger} \tag{2.4}$$

in the case of the gauge group $SU(2)$ and for $SU(3)$ one iteratively operates in the three $SU(2)$ subgroups [7].

In order to reduce the *critical slowing down* of the relaxation algorithm on large lattices, the authors of [4] suggested to apply an overrelaxation algorithm which replaces the gauge transformation $g(x)$ by $g^\omega(x)$, $\omega \in [1,2)$ in each step of the iteration. In practice the exponentiation of the gauge transformation is done to first or second order.

Due to the strict locality of the overrelaxation algorithm (only nearest neighbor interactions) we can perform a checkerboard decomposition of the lattice and operate on all sites of one of the two sublattices ("RED" and "BLACK") at the same time.

A measure of the quality of the gauge fixing is the average L_2-norm of the gauge fixing violation $\Delta^g \neq 0$

$$\theta \equiv \frac{1}{VN_c} \sum_x \mathrm{tr}\left[\Delta^g(x)\Delta^g(x)^\dagger\right], \tag{2.5}$$

where the sum runs over all sites x and V is the number of lattice sites.

Algorithm 1 Overrelaxation

  while precision θ not reached **do**
    for sublattice = RED, BLACK **do**
      for all x of sublattice **do**
        for all SU(2) subgroups **do**
          $g(x) \to \sum_\mu \left\{ U_\mu^\dagger(x) + U_\mu(x-\hat{\mu}) \right\}$ $\to$ 60 Flop
          $g(x) \to g^\omega(x)$, project to SU(2) $\to$ 19 Flop
          for all μ **do**
            $U_\mu(x) \to g^\omega(x)U_\mu(x)$ $\to$ 84 Flop
            $U_\mu(x-\hat{\mu}) \to U_\mu(x-\hat{\mu})g^\omega(x)^\dagger$ $\to$ 84 Flop
          end for
        end for
      end for
    end for
  end while

The algorithm is summarized in Alg. 1. In total the overrelaxation algorithm requires 751 flop per site and SU(2) subgroup iteration and thus 2253 flop/site for SU(3).

architecture	Fermi
compute capability	2.0
# SMs (streaming multiprocessors)	16
# total CUDA cores	512
device memory	1.5 GB
memory bandwidth	192.4 GB/s
ECC available	no
L2 cache	768 KB
L1 cache / SM	16 KB or 48 KB
shared memory / SM	16 KB or 48 KB
32-bit registers / SM	32768
max. registers / thread	63

Table 1: Hardware details of the NVIDIA GeForce GTX 580.

3. Hardware

We use NVIDIA's GeForce GTX 580 for our study. The GTX 580 is the high end graphical processing unit of the Fermi architecture which is NVIDIA's third generation of GPUs devoted to high performance computing using NVIDIA's CUDA (Compute Unified Device Architecture) programming environment. Recently the successor of the Fermi architecture has been released (Kepler). All hardware details are summarized in Tab. 1.

4. Overrelaxation on the GPU

4.1 First attempt

CUDA supports natively only lattices up to three dimensions, for that reason 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 32 threads per multiprocessor.

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 $F_g[U]$ and the gauge precision θ 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 the RED and BLACK sublattices consecutively.

The GPU can read data from global device memory in a fast way only if the data is accurately coalesced: the largest memory throughput is achieved when consecutive threads read from consecutive memory addresses. In order to do so we rearrange the gauge field into two blocks for the RED and BLACK sublattices. Moreover, for the same sake of memory coalescing, we choose the site index running fastest which results in a storage layout in which the gauge matrices do not lie anymore in consecutive memory blocks.

The overrelaxation algorithm on the GPU is bandwidth bound. Thus, in order to reduce memory traffic, we use the unitarity of SU(3) matrices to reconstruct the third line of each matrix on the fly instead of reading it from global memory. A minimal 8 parameter reconstruction [8] turned out

to be numerically not stable enough for our purpose since we not only have to read the gauge fields but also store them at the end of each iteration step.

For more details about these standard tricks of GPU programming in lattice QCD we refer to [8] and references therein.
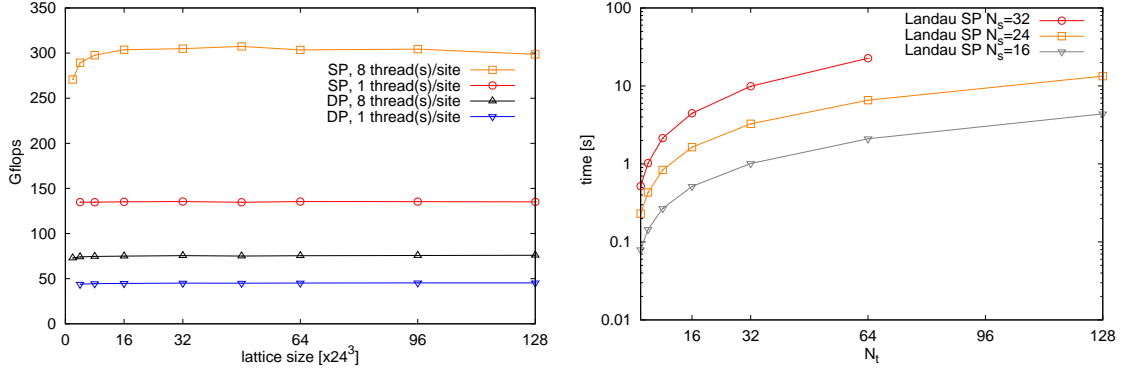

Figure 1: Left: performance for 1000 iterations. Right: time needed for 1000 iterations on $N_s^3 \times N_t$ lattices in SP.

The performance of this first attempt can be read off from the left plot of Fig. 1. There we show the number of flops achieved for 1000 iterations of the overrelaxation kernel in single precision (red line, circles) and double precision (blue line, down pointing triangles) for lattices of spatial volume 24^3 and varying temporal extend.

4.2 Improvement

In the beginning of each iteration of the overrelaxation algorithm each thread has to read its eight neighbor links from global memory and at the end of each iteration they have to be written back into global memory. These eight SU(3) matrices per site equal 8×18 reals = 144 reals and therewith exceed the register limit of 63 per thread (see Tab. 1) what results in register spills to global memory and as a consequence negatively effect the bandwidth bound performance of the kernel.

In order to reduce register spills we switch to a finer parallelization granularity: instead of assigning one thread to one lattice site we now tie eight threads to a single lattice site, i.e., one thread for each of the eight neighbors of a site. Then each thread needs only 18 registers to store the gauge link.

In order to avoid warp divergences the kernel is invoked with a thread block size of $8 \times 32 = 256$. By doing so, each of the eight warps (warp size is 32 on the Fermi) takes care of one neighbor type of the 32 sites and thus all threads within one warp follow the same instruction path.

The gauge transformation is then accumulated in shared memory. Since one operates on the SU(2) subgroups of SU(3) and an SU(2) matrix can conveniently be represented by four reals, this requires $4 \times 32 = 128$ reals or 512 bytes per thread block.

5. Performance

On the left hand side of Fig. 1 we show that with the fine parallelization granularity of eight

5

threads per lattice site we achieve a maximum performance of 300 Gflops for single precision (SP) and thus an improvement by a factor more than two compared to the conventional one thread per site strategy. On the right hand side of Fig. 1 the time required to run 1000 iterations of the overrelaxation algorithm on different lattice sizes is presented.
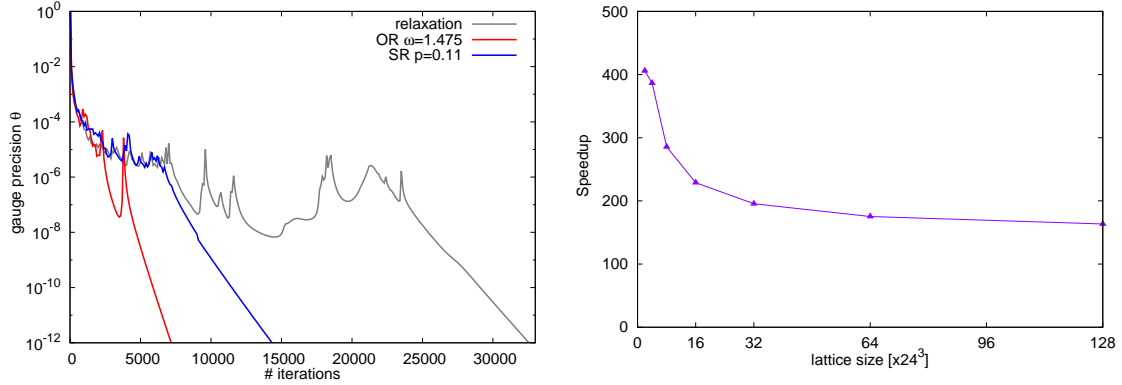

Figure 2: Left: number of iterations on a randomly chosen $\beta = 6.1$, 32^4 lattice. Right: speedup over CPU.

In Fig. 2, on the left, we give an example of how tuning of the overrelaxation (OR) and stochastic relaxation (SR) parameters can reduce the required number of iterations to achieve the aimed gauge quality (here $\theta < 10^{-12}$). This information combined with the information from Fig. 1 tells that the required time to fix the randomly chosen gauge configuration of lattice size 32^4 to the Landau gauge with the overrelaxation algorithm Fig. 2 to the precision of $\theta < 10^{-12}$ is of the order of one minute.

Lastly, we compare our performance to the overrelaxation kernel of the FermiQCD library [9] run in parallel with MPI on all four cores of the Intel Core i7-950 ("Bloomfield") processor @ 3.07GHz. The ratio of the time needed by FermiQCD to the time needed by our CUDA kernel on the GTX 580 for varying lattice sizes is plotted in Fig. 2 (r.h.s.). We find a speedup of evidently more than 150 for all lattice sizes, or in other words, assuming linear weak scaling, the performance of our code on one GTX 580 GPU is equivalent to the performance of the FermiQCD library on 150 Intel Core i7-950 CPUs (i.e. 600 cores) for the same algorithm.

6. Summary

We presented a CUDA implementation for gauge fixing on the lattice based on the relaxation algorithms. In particular, our code can be used to fix gauge field configurations to Landau, Coulomb or the maximally Abelian gauges using simulated annealing, overrelaxation or stochastic relaxation. Using a fine parallelization granularity of eight CUDA threads per lattice site we achieve a maximum performance of 300 Gflops in single precision on NVIDIA's GTX 580. Comparing this to the performance of the overrelaxation algorithm as implemented in the FermiQCD library run on the Intel Core i7-950 ("Bloomfield") quadcore processor @ 3.07GHz in parallel using MPI, we find a speedup of more than 150. Our code will be available for download shortly.

Acknowledgments

References

[1] M. Schröck, *The chirally improved quark propagator and restoration of chiral symmetry*, *Phys.Lett.* **B711** (2012) 217–224, [arXiv:1112.5107].

[2] N. Cardoso, P. J. Silva, P. Bicudo, and O. Oliveira, *Landau Gauge Fixing on GPUs*, arXiv:1206.0675.

[3] J. Mandula and M. Ogilvie, *The Gluon Is Massive: A Lattice Calculation of the Gluon Propagator in the Landau Gauge*, *Phys.Lett.* **B185** (1987) 127–132.

[4] J. E. Mandula and M. Ogilvie, *Efficient gauge fixing via overrelaxation*, *Phys.Lett.* **B248** (1990) 156–158.

[5] P. de Forcrand, *Multigrid techniques for quark propagator*, *Nucl.Phys.Proc.Suppl.* **9** (1989) 516–520.

[6] G. Bali, V. Bornyakov, M. Müller-Preussker, and K. Schilling, *Dual superconductor scenario of confinement: A Systematic study of Gribov copy effects*, *Phys.Rev.* **D54** (1996) 2863–2875, [hep-lat/9603012].

[7] N. Cabibbo and E. Marinari, *A New Method for Updating SU(N) Matrices in Computer Simulations of Gauge Theories*, *Phys. Lett. B* **119** (1982) 387.

[8] M. Clark, R. Babich, K. Barros, R. Brower, and C. Rebbi, *Solving Lattice QCD systems of equations using mixed precision solvers on GPUs*, *Comput.Phys.Commun.* **181** (2010) 1517–1528, [arXiv:0911.3191].

[9] **FermiQCD** Collaboration, M. Di Pierro et al., *www.fermiqcd.net*, *Nucl. Phys. Proc. Suppl.* **129** (2004) 832–834, [hep-lat/0311027].