GAUGE FIXING IN LATTICE QCD WITH MULTI-GPUs*

Mario Schröck

Institut für Physik, FB Theoretische Physik, Universität Graz
8010 Graz, Austria

Hannes Vogt

Institut für Theoretische Physik
Auf der Morgenstelle 14, 72076 Tübingen, Germany

(Received June 24, 2013)

Here, we present the cuLGT[1] code for gauge fixing in lattice gauge field theories with graphic processing units (GPUs). Implementations for SU(3) Coulomb, Landau and maximally Abelian gauge fixing are available and the overrelaxation, stochastic relaxation and simulated annealing algorithms are supported. Performance results for single and multi-GPUs are given.

DOI:10.5506/APhysPolBSupp.6.763
PACS numbers: 11.15.Ha, 12.38.Gc

1. Introduction

Gauge fixing in the lattice QCD is necessary in order to, *e.g.*, compare lattice results to continuum physics in a given renormalization scheme at a given scale. The popular Landau gauge requires the four dimensional gradient of the gauge field to vanish at each space-time point of the lattice. The latter continuum condition translates to a large scale optimization problem in lattice gauge field theories. Finding its maxima is very expensive in terms of computational costs and a possible acceleration by high performance fine grained parallel architectures, like graphic processing units (GPUs), is highly desirable. Here, we present a code written in CUDA which has been developed for the purpose of lattice gauge fixing on GPUs. The code makes strong use of template classes and algorithm abstraction to increase its flexibility and to extend its applicability to related problems in lattice gauge field theory.

* Presented by M. Schröck at the Workshop "Excited QCD 2013", Bjelašnica Mountain, Sarajevo, Bosnia–Herzegovina, February 3–9, 2013.

[1] `www.cuLGT.com`

The lattice QCD gauge fixing on GPUs was first presented in [1] and a detailed discussion of our code can be found in Ref. [2]. Authors of [3] use the Fourier accelerated deepest descent method for gauge fixing in the lattice QCD.

In the following discussion, we restrict ourselves to the example of the Landau gauge fixing and we refer to [2] for the details of the other gauges and algorithms which are supported by cuLGT.

2. Lattice Landau gauge

The continuum Landau gauge condition,

$$\partial_\mu A_\mu(x) = 0\,, \tag{1}$$

is fulfilled if and only if the lattice gauge functional

$$F^g[U] = \frac{1}{N_c N_d V} \mathfrak{Re} \sum_{\mu,x} \mathrm{tr}\left[U^g_\mu(x)\right] \tag{2}$$

resides in a stationary point with respect to gauge transformations $g(x) \in \mathrm{SU}(N_c)$. Here, we denoted a gauge transformation of the link variables as

$$U^g_\mu(x) \equiv g(x) U_\mu(x) g(x+\hat{\mu})^\dagger\,. \tag{3}$$

N_c is the number of colors, $N_c = 3$ for QCD, N_d is the number of space-time dimensions, (here $N_d = 4$) and V is the total number of lattice points.

A measure θ of the Landau gauge precision is the average L_2-norm of the gauge fixing violation $\Delta(x)$, *i.e.*, the discrete derivative of the continuum gauge fields

$$\Delta(x) \;\equiv\; \sum_\mu \left(A_\mu(x) - A_\mu(x-\hat{\mu})\right) = 0\,, \tag{4}$$

$$\theta \;\equiv\; \frac{1}{N_c V} \sum_x \mathrm{tr}\left[\Delta(x)\Delta(x)^\dagger\right]\,. \tag{5}$$

3. The relaxation algorithms

The idea of the relaxation algorithms is to sweep over the lattice site by site while optimizing the gauge functional locally. All sites of one of the two *parity subsets* (checker board decomposition) can be optimized at the same time because the newly generated local optimum depends on the nearest neighbors only.

Instead of taking the complete global gauge functional into account,

$$F^g[U] = \frac{1}{2N_c N_d V} \mathfrak{Re} \sum_x f^g(x)\,, \tag{6}$$

the relaxation algorithm aims at optimizing the value of $F^g[U]$ locally, *i.e.*, we search for the maximum of

$$f^g(x) = \mathfrak{Re}\ \mathrm{tr}\,[g(x)K(x)] \tag{7}$$

for all x. Here, we defined

$$K(x) := \sum_\mu \left(U_\mu(x) g(x+\hat{\mu})^\dagger + U_\mu(x-\hat{\mu})^\dagger g(x-\hat{\mu})^\dagger \right). \tag{8}$$

For SU(2), the maximum thereof is given by

$$g(x) = K(x)^\dagger / \sqrt{\det K(x)^\dagger} \tag{9}$$

and for SU(3), one iteratively operates in the three SU(2) subgroups [4], and thereby optimizes the local SU(3) gauge functional.

3.1. Overrelaxation

Replacing the local gauge transformation $g(x)$ by $g^\omega(x)$, $\omega \in [1,2)$ reduces the *critical slowing down* of the relaxation algorithm on large lattices [5]. In practice, the exponentiation of the gauge transformation is done to first order.

3.2. Stochastic relaxation

The stochastic relaxation algorithm replaces the local gauge update $g(x)$ by $g^2(x)$ with probability p and can speed up the convergence on large lattices.

4. Single-GPU implementation

We assign eight CUDA threads to each lattice site of a given parity in order to calculate and apply the local gauge update (9). The two parity sublattices are treated consecutively and the relaxation algorithm is iterated until the requested gauge precision θ has been reached. A variable data storage pattern for the gauge fields is adopted in order to meet the memory coalescing constraints of the hardware. In Fig. 1, we compare the performance of the code on different NVIDIA devies.

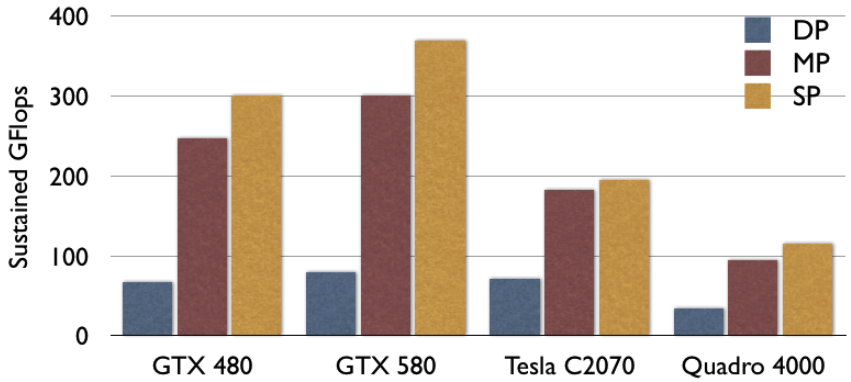

Fig. 1. Performance of the overrelaxation kernel on different NVIDIA devices in single (SP), mixed (MP) and double precision (DP) on a 32^4 lattice.

5. Multi-GPU implementation

For the multi-GPU implementation, we decided for a decomposition of the lattice along the temporal axis, see Fig. 2. In each step of the iteration, the gauge links of the neighbor device in the temporal direction have to be exchanged via MPI in order to calculate the gauge update (9). After the gauge update has been calculated, it has to be applied to all connected gauge links, therefore, it has to be copied to the neighbor device. In detail, the following set of instructions has to be carried out on each device in order to transfer the links $U_0(t_{\max})$ of device i to device $i+1$:

1. `cudaMemcpyDeviceToHost` of $U_0(t_{\max})$ (inactive parity);
2. `MPI_Send` of $U_0(t_{\max})$ to device $i+1$ and `MPI_Recv` of $U_0(t_{\min}-1)$ from device $i-1$;
3. `cudaMemcpyHostToDevice` of $U_0(t_{\min}-1)$;
4. update $U_\mu(t_{\min})$ (active parity) which affects $U_0(t_{\min}-1)$ (inactive);
5. `cudaMemcpyDeviceToHost` of $U_0(t_{\min}-1)$ (inactive parity);
6. `MPI_Send` of $U_0(t_{\min}-1)$ to device i and `MPI_Recv` of $U_0(t_{\max})$ from device $i+1$;
7. `cudaMemcpyHostToDevice` of $U_0(t_{\max})$.

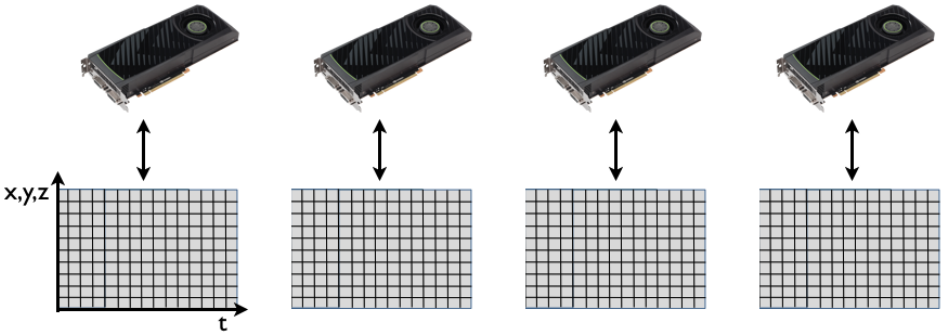

Fig. 2. The four dimensional lattice is split along the temporal axis and distributed to the devices. The bottle neck is the communication at the boundaries via the PCI-bus.

In order to hide the slow data exchange over the low-bandwidth PCI-bus, we perform asynchronous memory transfers: we overlap the data exchange on the boundaries with calculations in the inner part of the domain. In Table I we compare the time needed to update one time-slice in the inner part of the domain with the time needed to copy the data at the boundaries to the host memory and from the host memory to the neighboring device.

TABLE I

Time in microseconds needed to copy the data at the boundaries from device to host (D2H) and host to device (H2D) compared to the time needed to update one time-slice with the overrelaxation kernel. The two most right columns give the ratios.

N_s^3	D2H [μs]	H2D [μs]	Kernel [μs]	D2H/kernel	H2D/kernel
16	0.0398	0.0368	0.0209	1.90	1.76
32	0.2543	0.2276	0.1443	1.76	1.58
64	1.2510	1.1830	1.0489	1.19	1.13
128	8.9597	8.7169	8.3041	1.08	1.05

Figure 3 confirms the predictions of Table I that linear weak scaling is achieved with asynchronous memory transfers.

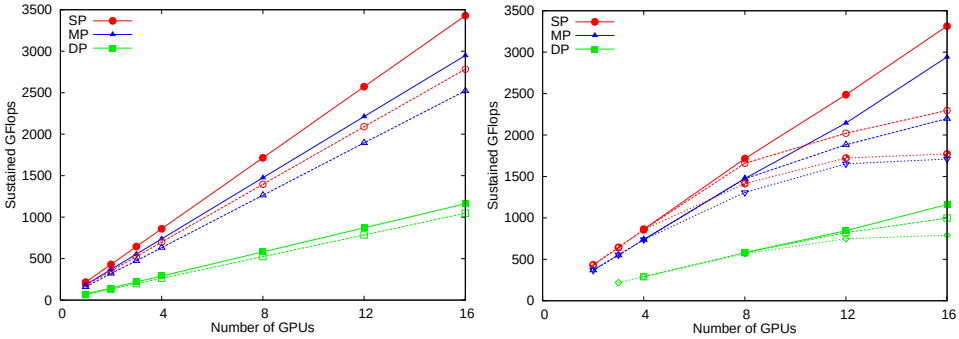

Fig. 3. Left: weak scaling on the NVIDIA Tesla C2070. The full symbols correspond to a lattice size of $64^3 \times 32$ per GPU and the open symbols to 48^4 per GPU. Right: strong scaling on the Tesla C2070. The spatial lattice volume is kept fixed at 64^3 and the total temporal extent varies for the three lines (per precision) from the top downwards $N_\mathrm{t} = 256, 128, 96$.

6. Summary

The local relaxation algorithms for lattice gauge fixing are well suited to be accelerated with highly parallel architectures like GPUs. With the aim of retaining maximum performance in a multi-GPU implementation it is crucial to overlap the data exchange between the devices by calculations in the inner part of the domain. This allows for linear weak scaling and a maximum performance of ~ 3.5 Teraflops adopting 16 Tesla C2070 GPUs.

Support by the Research Executive Agency (REA) of the European Union under Grant Agreement PITN-GA-2009-238353 (ITN STRONGnet) and by the Austrian Science Fund (FWF) through grant DK W1203-N16 is gratefully acknowledged.

REFERENCES

[1] M. Schröck, *Phys. Lett.* **B711**, 217 (2012) [arXiv:1112.5107 [hep-lat]].

[2] M. Schröck, H. Vogt, *Comput. Phys. Commun.* **184**, 1907 (2013) [arXiv:1212.5221 [hep-lat]].

[3] N. Cardoso, P.J. Silva, P. Bicudo, O. Oliveira, *Comput. Phys. Commun.* **184**, 124 (2013) [arXiv:1206.0675 [hep-lat]].

[4] N. Cabibbo, E. Marinari, *Phys. Lett.* **B119**, 387 (1982).

[5] J.E. Mandula, M. Ogilvie, *Phys. Lett.* **B248**, 156 (1990).