Coulomb and Landau Gauge Fixing in GPUs using CUDA and MILC

Nuno Cardoso[*]
NCSA, University of Illinois
E-mail: ncardoso@illinois.edu

In this work, we present the GPU implementation of the overrelaxation and steepest descent method with Fourier acceleration methods for Laudau and Coulomb gauge fixing using CUDA for SU(N) with N>2. A multi-GPU implementation of the overrelaxation method is also presented using MPI and CUDA. The GPU performance was measured on BlueWaters and compared against the gauge fixing of the CPU MILC code.

The 32nd International Symposium on Lattice Field Theory
23-28 June, 2014
Columbia University New York, NY

[*]Speaker.

1. Introduction

On the lattice, the Coulomb/Landau gauge is defined by maximising the functional

$$F_U[g] = \frac{1}{4N_cV}\sum_x\sum_\mu \mathrm{Re}\left[\mathrm{Tr}\left(g(x)U_\mu(x)g^\dagger(x+\hat{\mu})\right)\right] \tag{1.1}$$

with N_c the dimension of the gauge group and V the lattice volume. On the gauge fixing process, the quality of the gauge fixing is measured by

$$\theta = \frac{1}{N_cV}\sum_x \mathrm{Tr}\left[\Delta(x)\Delta^\dagger(x)\right] \tag{1.2}$$

where

$$\Delta(x) = \sum_\nu \left[U_\nu(x-a\hat{\nu}) - U_\nu(x) - \text{h.c.} - \text{trace}\right] \tag{1.3}$$

is the lattice version of $\partial_\mu A_\mu = 0$, where $\mu = 0,1,2,3$ for landau gauge and $\mu = 0,1,2$ for the Coulomb gauge (where the temporal direction is along $\mu = 3$). Two well known methods to fix the gauge are the relaxation algorithm via overrelaxation and the steepest descent method with FFTs.

The relaxation algorithm aims to optimize the value of $F_U[g]$ locally, i.e., searching the maximum of

$$f^g(x) = \mathrm{Re}\,\mathrm{Tr}\left[g(x)K(x)\right] \tag{1.4}$$

for all x, where

$$K(x) = \sum_\mu \left(U_\mu(x)g^\dagger(x+\hat{\mu}) + U_\mu^\dagger(x-\hat{\mu})g^\dagger(x-\hat{\mu})\right) \tag{1.5}$$

The local solution is then given by

$$g(x) = K^\dagger(x)\left(\det K^\dagger(x)\right)^{-1/2} \tag{1.6}$$

in the case of the gauge group SU(2). For $N > 2$ one iteratively operates in the $(N(N-1)/2)$ SU(2) subgroups. The overrelaxation algorithm, [1], replaces the gauge transformation $g(x)$ by $g^\omega(x)$ with $\omega \in [1,2[$ in each step of the iteration. The gauge fixing with overrelaxation algorithm is described in algorithm 1.

The naive steepest descent method chooses at each step of the iterative procedure

$$g(x) = \exp\left[\frac{\alpha}{2}\left(\sum_\nu \Delta_{-\nu}\left[U_\nu(x) - U_\nu^\dagger(x)\right] - \text{trace}\right)\right] \tag{1.7}$$

However, when it is applied to larger lattices, this method faces the problem of critical slowing down. This problem can be attenuated by Fourier acceleration[2]. At each iteration one chooses

$$g(x) = \exp\left[\hat{F}^{-1}\frac{\alpha}{2}\frac{p_{\max}^2a^2}{p^2a^2}\hat{F}\left(\sum_\nu \Delta_{-\nu}\left[U_\nu(x) - U_\nu^\dagger(x)\right] - \text{trace}\right)\right] \tag{1.8}$$

with

$$\Delta_{-\nu}\left(U_\mu(x)\right) = U_\mu(x-a\hat{\nu}) - U_\mu(x) \tag{1.9}$$

p^2 are the eigenvalues of $\left(-\partial^2\right)$, a is the lattice spacing and $\hat{F}$ represents a fast Fourier transform (FFT). For the parameter α, the optimal value is 0.08. For numerical purposes, it is enough to expand to first order the exponential, followed by a reunitarization. The gauge fixing algorithm using the steepest descent method with FFTs is described in algorithm 2.

Algorithm 1 Overrelaxation algorithm.

```
    calculate $F_g[U]$ and $\theta$ (optional)
 2: while $\theta \geq \varepsilon$ do
      for site parity = even, odd do
 4:     for all $x$ with same parity do
          for all SU(2) subgroups do
 6:         local optimization, find $g(x) \in SU(2)$
            which is function of $U_\mu(x)$ and $U_\mu(x-\hat{\mu})$
 8:         for all $\mu$ do
              apply $g(x)$ to $U_\mu(x)$ and $U_\mu(x-\hat{\mu})$
10:         end for
          end for
12:     end for
      end for
14:   calculate $F_g[U]$ and $\theta$
    end while
```

Algorithm 2 Steepest descent method with FFTs.

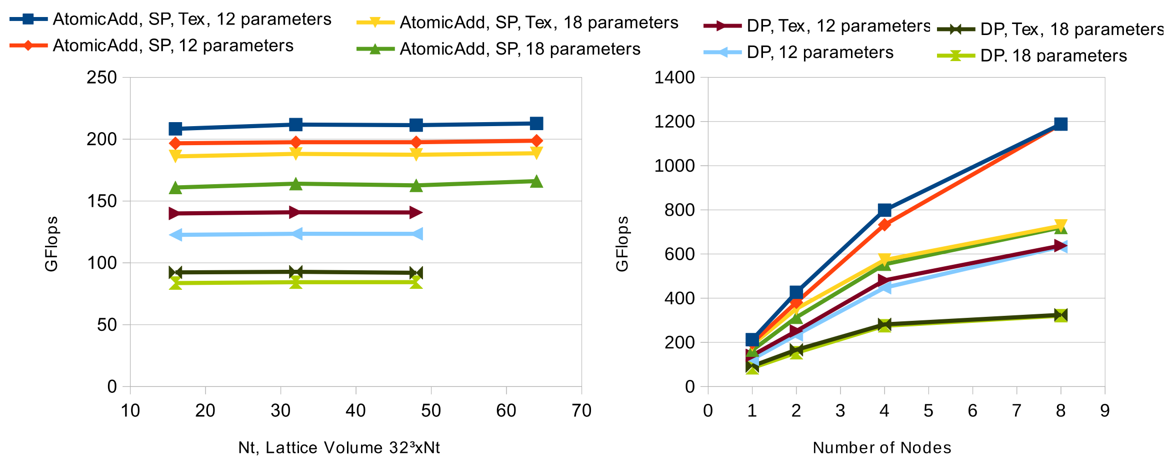
```
    calculate $\Delta(x)$, $F_g[U]$ and $\theta$
 2: while $\theta \geq \varepsilon$ do
      for all elements of $\Delta(x)$ matrix do
 4:     apply FFT forward
        apply $p^2_{\max}/p^2$
 6:     apply FFT backward
        normalize
 8:   end for
      for all $x$ do
10:     obtain $g(x)$ from $\Delta(x)$ and reunitarize
      end for
12:   for all $x$ do
        for all $\mu$ do
14:       $U_\mu(x) \to g(x)U_\mu(x)g^\dagger(x+\hat{\mu})$
        end for
16:   end for
      calculate $\Delta(x)$, $F_g[U]$ and $\theta$
18: end while
```

2. Implementation

The gauge links are stored in a 1-dimensional array with size $volume \times 4 \times Elems$ in the GPU global memory, with $Elems = 4,6,9$ complex elements in the case of SU(3) and $volume = nx*ny*nz*nt$. The memory access is done as
$id = (i+j\times nx+k\times nx\times ny+t\times nx\times ny\times nz)/2+site\_parity\times volume/2+\mu\times volume$
where the even and odd sites are kept separate and each SU(N) complex element is stored with stride $4\times volume$.

In overrelaxation method, each gauge link, $U_\mu(x)$, to be updated are dependent on the neighbors at $U_\mu(x)$ and $U_\mu(x-\hat{\mu})$, i.e., in total eight gauge links must be loaded from memory. Therefore, assigning one thread per lattice site leads to a high memory traffic and local memory usage. A solution to avoid this is using eight threads per single lattice site, [3]. Since the GPU warp size is 32, the minimum block size should be 32×8 threads in order to maintain the memory reads coalesced from global memory and the grid size must be $volume/32$. The gauge links are loaded into registers whenever possible and the data exchange between threads is done through shared memory. The $\sum_\mu$ can be done using only shared memory (more shared memory consumption) or using CUDA Atomic functions in shared memory, although double precision atomic additions are not yet supported natively). The multi-GPU implementation was done using MPI. The code was implemented in order to support any type of lattice partition however all local lattice dimensions must be an even number. At each gauge fixing step, the updates are first done in even lattice sites and then in the odd sites. Each node needs to exchange the top links with the same direction as the partitioned dimension and also since the exchanged links are updated we need to exchange them back. To overlap the gauge link updates with node communication, we update first the bottom and top links in the partitioned dimensions and then overlap the exchange top links (current parity) and the ghost links (opposite parity) with the update of the remaining gauge links. In order to support any kind of lattice partition, we pre-calculate the border and the interior lattice sites separately.

The steepest descent method with FFTs implemented here is a generalization of the implementation described in [4]. Here, we generalized the previous implementation to support even/odd lattice array and SU(N) with N>2. Due to the high number of FFTs per gauge fixing step, $N \times N \times (\text{4D FFT} + \text{4D IFFT}))$ (in SU(3), using the 12 real number parametrization, instead of 9, we can reduce this to 6) and due to a lack of support for 4D FFTs in GPUs, we haven't done a multi-GPU implementation. However, one solution to avoid using FFTs is to use a multigrid implementation of the Fourier acceleration method [5].



(a) Single GPU performance. (b) Multi-GPU performance, 32^4 lattice volume.

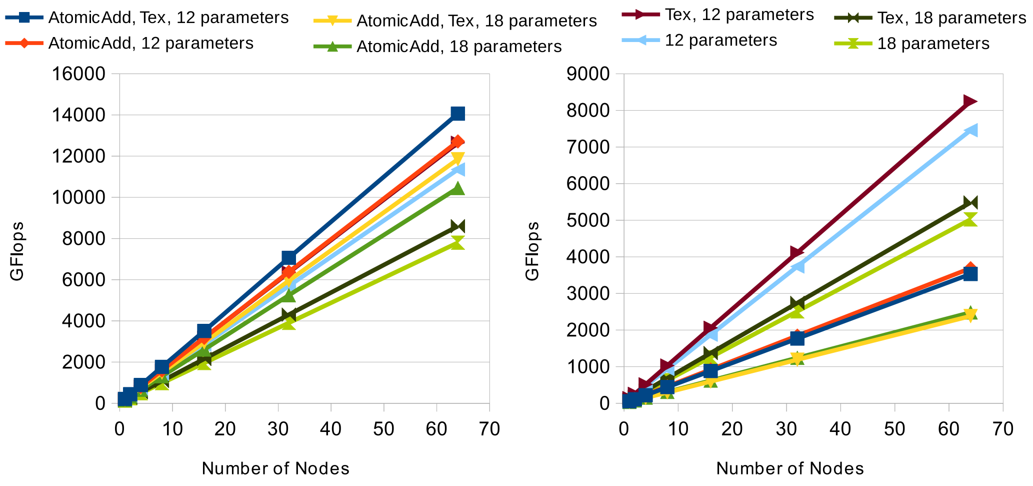
Figure 1: SU(3) performance of the Overrelaxation algorithm. AtomicAdd means using CUDA atomicAdd function, SP/DP means single/double precision, Tex means using Texture memory, 12/18 parameters are the number of SU(3) real parameters used to store the gauge array in memory.

3. Results

The performance tests were done in BlueWaters system with MILC, [6], and CUDA 5.5. Each node in BlueWaters system has one Tesla K20X GPU, with ECC code enabled by default. In this paper, we only present the performance results for the SU(3) case. Despite the code for SU(3) supports global memory storage with 18 (full SU(3) matrix), 12 and 8 real parameters, here we only present the performance results for the full matrix and 12 real number parameterization. We measured the performance for 1000 iterations with link reunitarization at each 20 steps. The overrelaxation boost parameter was set at 1.5 and in stepeest descent method with FFTs $\alpha = 0.08$.

The MILC code by default splits the lattice between nodes by the following order $X \to Y \to Z \to T$, where X is the lattice fastest index. Therefore, we changed the MILC code to allow the layout lattice partition by the following priority order $T \to Z \to Y \to X$ which is preferable for GPUs.

The performance results for the Landau gauge fixing using the overrelaxation algorithm are presented in figures 1 and 2 for single and multi-GPU. The use of CUDA atomic operations, in single precision allow a speedup of $1.1-1.4\times$, although for double precision not using CUDA atomics we obtain a speedup of $2-2.4\times$, figure 2. The double precision atomic additions is not yet supported natively by the hardware. In figure 3, we compare the single GPU performance with the CPU MILC code, with and without taking in account the copies from and to the GPU and the gauge field reorder. Without taking into account the copies to and from GPU, the GPU performance is around 350/200 times faster than CPU MILC code in single/double precision.



(a) Single precision. (b) Double precision.

Figure 2: SU(3) weak Scaling for overrelaxation algorithm. Fixed node local lattice volume to 32^4. Lattice volume: $32^3 \times Nt$ with $Nt = 32 \times$ (Number of nodes). SP/DP means single/double precision, Tex means using Texture memory.

In figure 2 we present the performance results for the weak scaling of the overrelaxation algorithm for a fixed local lattice volume of 32^4 The results show a good performance scaling.

In figure 4 the performance results for the Coulomb gauge fixing with overrelaxation in single GPU are presented. Although we are expecting similar or better performance results for the Coulomb gauge fixing in comparison with the Landau gauge fixing, we obtained better performance for Landau gauge fixing in single precision. A detailed analysis of this behavior was not done yet.

The multi-GPU performance of gauge fixing with overrelaxation, figures 1b and 4b, shows a good performance scaling up to four nodes for a lattice volume of 32^4. For eight nodes, the performance shows a visible decrease due to the weak overlap in MPI communications and interior gauge links update. With eight nodes, MILC splits the lattice by half in Y, Z and T dimensions.

The performance results for the steepest descent algorithm for Landau and Coulomb gauge fixing are presented in figure 5. Further tests have to be done in order to better understand the

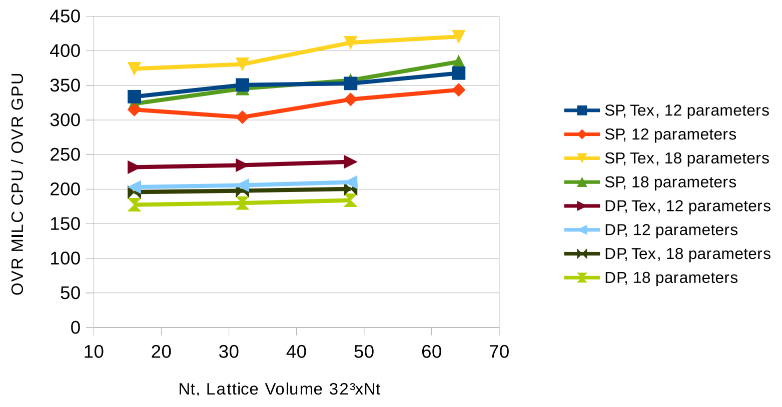
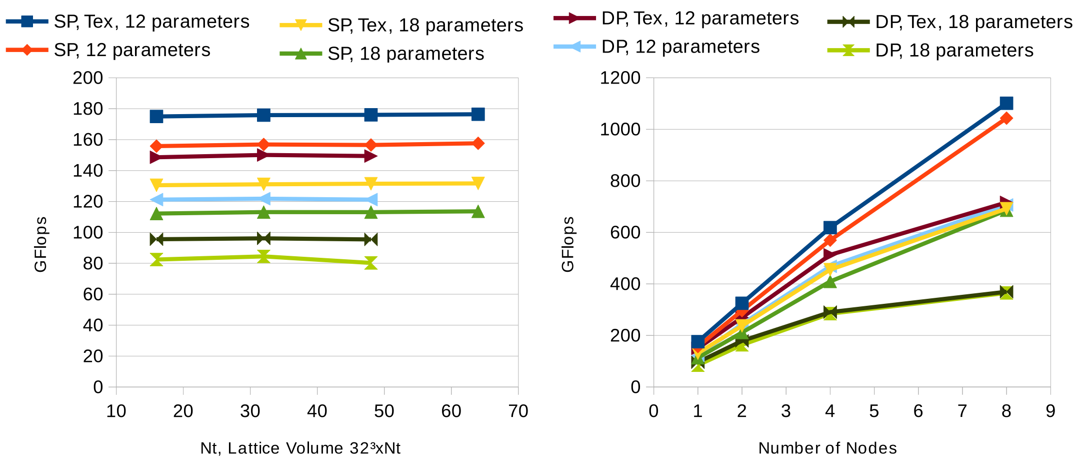


Figure 3: Single GPU speed up over CPU MILC code in single node for the Landau gauge fixing with overrelaxation. Using only CUDA atomic operations in single precision. SP/DP means single/double precision, Tex means using Texture memory.



(a) Single GPU performance.

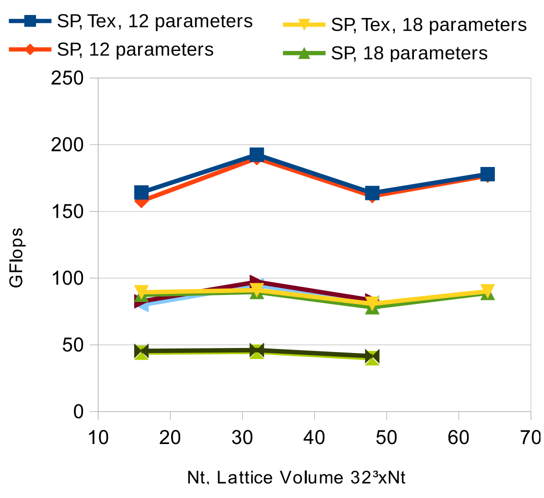
(b) Multi-GPU performance, 32^4 lattice volume.

Figure 4: SU(3) performance results for Coulomb gauge fixing with overrelaxation. Using only CUDA atomic operations in single precision. SP/DP means single/double precision, Tex means using Texture memory.

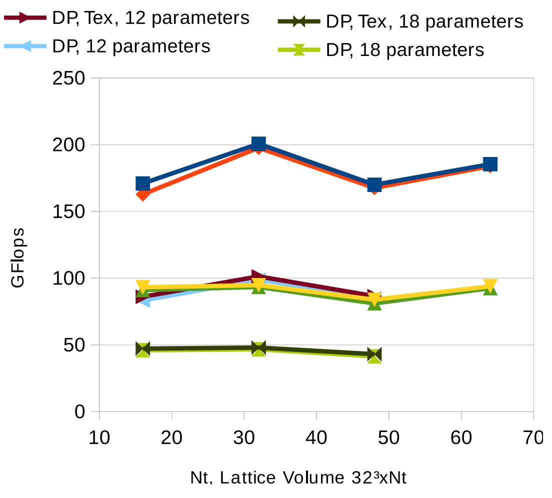
performance fluctuations in changing the volume. These fluctuations are not visible when using the gauge fixing with overrelaxation, see figures 1a and 4a.

4. Conclusion

The SU(N) code, with N>2, for Coulomb and Landau gauge fixing was implemented on CUDA. The SU(3) code also supports 8 and 12 real number parametrization to store the gauge array in GPU memory. Only the gauge fixing with overrelaxation method supports multi-GPUs using MPI. In general, the steepest descent algorithm with Fourier acceleration converges faster than overrelaxation algorithm. However, this method requires 1.5× more memory than the overrelaxation. This library is currently being ported to QUDA library, [7].



(a) Landau gauge fixing.



(b) Coulomb gauge fixing.

Figure 5: SU(3) performance results for gauge fixing with steepest descent algorithm with Fourier acceleration in single GPU. SP/DP means single/double precision, Tex means using Texture memory.

Acknowledgments

This work is supported by the NSF award PHY-1212270. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This work is also part of the Lattice QCD on Blue Waters PRAC allocation supported by the National Science Foundation award ACI 0832315.

References

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

[2] C. Davies, G. Batrouni, G. Katz, A. S. Kronfeld, G. Lepage, et al., Fourier acceleration in lattice gauge theories. I. Landau gauge fixing, Phys.Rev. D37 (1988) 1581.

[3] M. Schröck, H. Vogt, Coulomb, Landau and Maximally Abelian Gauge Fixing in Lattice QCD with Multi-GPUs, Comput.Phys.Commun. 184 (2013) 1907–1919.

[4] N. Cardoso, P. J. Silva, P. Bicudo, O. Oliveira, Landau Gauge Fixing on GPUs, Comput.Phys.Commun. 184 (2013) 124–129.

[5] A. Cucchieri, T. Mendes, A Multigrid implementation of the Fourier acceleration method for Landau gauge fixing, Phys.Rev. D57 (1998) 3822–3826.

[6] MIMD Lattice Computation (MILC) `http://www.physics.indiana.edu/~sg/milc.html`.

[7] Quda: A library for qcd on gpus, `http://lattice.github.io/quda/`.