Coulomb and Landau Gauge Fixing in GPUs using CUDA and MILC

Nuno Cardoso

NCSA, University of Illinois

Lattice 2014

- MILC and QUDA library does not have gauge fixing on GPUs
- When using MILC + QUDA: lot of computational time can be waisted on CPU gauge fixing
- Main goal was to create a library to be called from MILC code with support for single and multi-GPUs to perform gauge fixing

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

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

  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

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

  where

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

  is the lattice version of $\partial_\mu A_\mu = 0$.
- Two well known methods to fix the gauge:
  - **Relaxation algorithm: overrelaxation**
  - **Steepest descent method with FFTs**

Overrelaxation

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

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

for all x, where

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

- the local solution is then given by

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

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 replaces the gauge transformation $g(x)$ by $g^\omega(x)$ with $\omega \in [1, 2[$ in each step of the iteration.

Overrelaxation

calculate $F_g[U]$ and θ (optional)
while $\theta \geq \epsilon$ **do**
  for site parity = even, odd **do**
    for all x with same parity **do**
      for all SU(2) subgroups **do**
        local optimization, find $g(x) \in SU(2)$
        which is function of $U_\mu(x)$ and $U_\mu(x - \hat{\mu})$
        for all μ **do**
          apply $g(x)$ to $U_\mu(x)$ and $U_\mu(x - \hat{\mu})$
        end for
      end for
    end for
  end for
  calculate $F_g[U]$ and θ
end while

Overrelaxation

calculate $F_g[U]$ and θ (optional)
2: **while** $\theta >$ ϵ **do**
f
4:
6:
8:
10:
12:
e
14: c
end

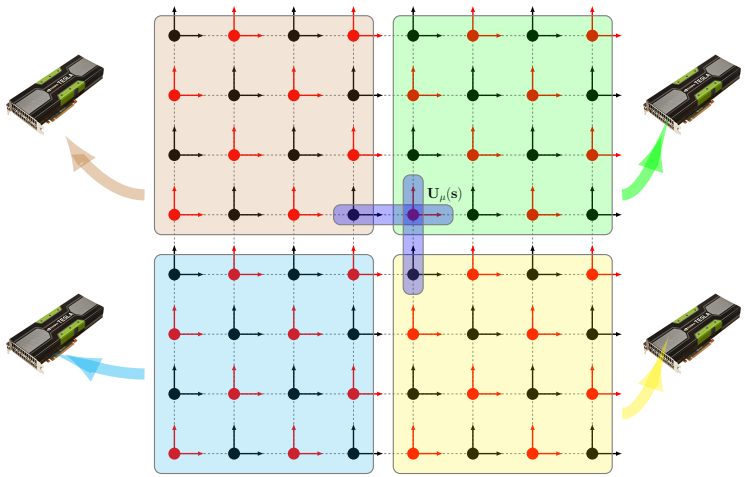
- to update a single lattice site: one needs to load 8 gauge links
  - $U_\mu(x)$
  - $U_\mu(x - \hat{\mu})$
- one thread per lattice site $\rightarrow$ high memory traffic and local memory usage!
- solution: 8 threads per lattice site, Mario Schröck and Hannes Vogt, Comput.Phys.Commun. 184, 1907-1919, 2013.
  - load gauge links to registers whenever possible
  - exchange data between threads through shared memory
  - $\sum_\mu \quad \rightarrow \quad$ can be done using:
    - CUDA Atomic functions
    - without CUDA Atomic functions (requires more shared memory)

Overrelaxation implementation using multi-GPUs

- CUDA + MILC + MPI

2D Lattice Example:

- Gauge array split by even and odd lattice sites
- Pre calculate interior lattice array index sites and lattice array index at the boundaries for each dimension partitioned
- Exchange face links
- Measure gauge fixing quality
- Main loop to perform gauge fixing,
  - Loop over all lattice sites by even and odd links separately
    - update top/bottom and right/left face links
    - pack top (same parity) and ghost (opposite parity) face links
    - exchange top face (same parity) and ghost (opposite parity) links
    - exchange ghost links
    - update interior links
    - unpack received links
  - reunitarize gauge at X steps
  - measure gauge fixing quality

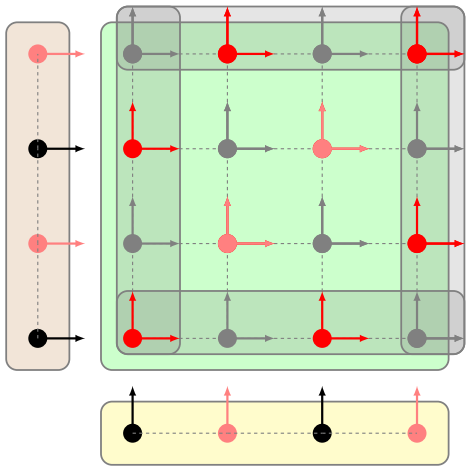


Overrelaxation implementation using multi-GPUs

- CUDA + MILC + MPI

2D Lattice Example:

- Gauge array split by even and odd lattice sites
- Pre calculate interior lattice array index sites and lattice array index at the boundaries for each dimension partitioned
- Exchange face links
- Measure gauge fixing quality
- Main loop to perform gauge fixing,
  - Loop over all lattice sites by even and odd links separately
    - update top/bottom and right/left face links
    - pack top (same parity) and ghost (opposite parity) face links
    - exchange top face (same parity) and ghost (opposite parity) links
    - exchange ghost links
    - update interior links
    - unpack received links
  - reunitarize gauge at X steps
  - measure gauge fixing quality

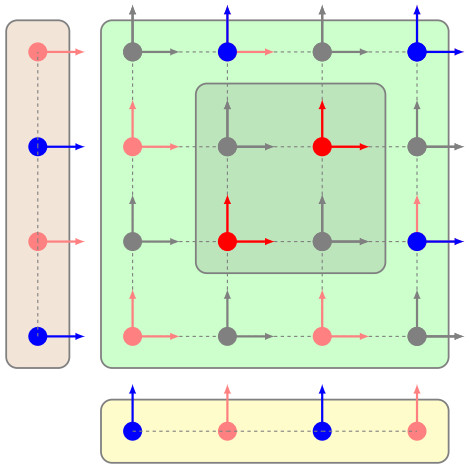


Overrelaxation implementation using multi-GPUs

- CUDA + MILC + MPI

2D Lattice Example:

- Gauge array split by even and odd lattice sites
- Pre calculate interior lattice array index sites and lattice array index at the boundaries for each dimension partitioned
- Exchange face links
- Measure gauge fixing quality
- Main loop to perform gauge fixing,
  - Loop over all lattice sites by even and odd links separately
    - update top/bottom and right/left face links
    - pack top (same parity) and ghost (opposite parity) face links
    - exchange top face (same parity) and ghost (opposite parity) links
    - exchange ghost links
    - update interior links
    - unpack received links
  - reunitarize gauge at X steps
  - measure gauge fixing quality



Steepest descent method with FFTs

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

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

- 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**, Davies et. al, doi:10.1103/PhysRevD.37.1581, 1987.
- At each iteration one chooses

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

with

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

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**.

Steepest descent method with FFTs

calculate $\Delta(x)$, $F_g[U]$ and θ
while $\theta \geq \epsilon$ **do**
  for all elements of $\Delta(x)$ matrix **do**
    apply FFT forward
    apply p^2_{max}/p^2
    apply FFT backward
    normalize
  end for
  for all x **do**
    obtain $g(x)$ from $\Delta(x)$ and reunitarize
  end for
  for all x **do**
    for all μ **do**
      $U_\mu(x) \rightarrow g(x)U_\mu(x)g^\dagger(x+\hat{\mu})$
    end for
  end for
  calculate $\Delta(x)$, $F_g[U]$ and θ
end while

Steepest descent method with FFTs

calculate $\Delta(x)$, $F_g[U]$ and θ
while $\theta \geq \epsilon$ **do**
  for all elements of $\Delta(x)$ matrix **do**
    apply FFT forward
    apply p^2_{max}/p^2
    apply FFT backward
    normalize
  end for
  for all x **do**
    obtain $g(x)$ from $\Delta(x)$ and reunit
  end for
  for all x **do**
    for all μ **do**
      $U_\mu(x) \rightarrow g(x)U_\mu(x)g^\dagger(x+\hat{\mu})$
    end for
  end for
  calculate $\Delta(x)$, $F_g[U]$ and θ
end while

FFT on GPUs: CUDA CUFFT library

- support for 1D, 2D and 3D FFTs;
- there is no direct support for 4D FFTs
- solution:
  - 4×1D FFTs: necessary reorder data 7 times per $\Delta(x)$ element
  - 3D+1D FFTs: necessary reorder data 2 times per $\Delta(x)$ element
  - 2×2D FFTs: necessary reorder data 2 times per $\Delta(x)$ element
- cufftPlanMany() allows the execution of multiple independent FFTs in parallel.
- best choice for our 4D problem: 2D + 2D FFTs.
  N. Cardoso et al., Comput.Phys.Commun. 184, 124-129, 2013. In a Tesla C2075 and 32^4 lattice volume
  - 2×2D FFTs is 39-43% faster than 4×1D FFTs
  - 2×2D FFTs is 7-8% faster than 3D+1D FFTs.

Steepest descent method with FFTs

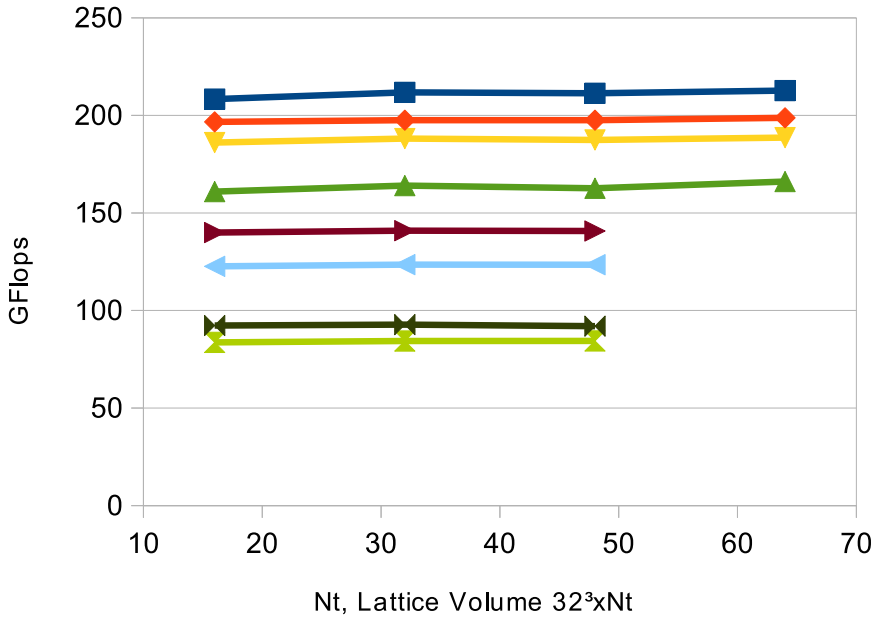
calculate $\Delta(x)$, $F_g[U]$ and θ
while $\theta \geq \epsilon$ **do**
  for all elements of $\Delta(x)$ matrix **do**
    apply FFT forward
    apply p^2_{max}/p^2
    apply FFT backward
    normalize
    e
    f
    e
    f
    e
  calculate $\Delta(x)$, $F_g[U]$ and θ
end while

- Only single GPU implementation
- Too heavy for multi GPU due to the number of FFTs per gauge fixing step
- One solution to avoid using FFTs is to use a multigrid implementation of the Fourier acceleration method Cucchieri et. al, Phys.Rev.D57, arXiv:hep-lat/9711047, 1998.
  not implemented here!

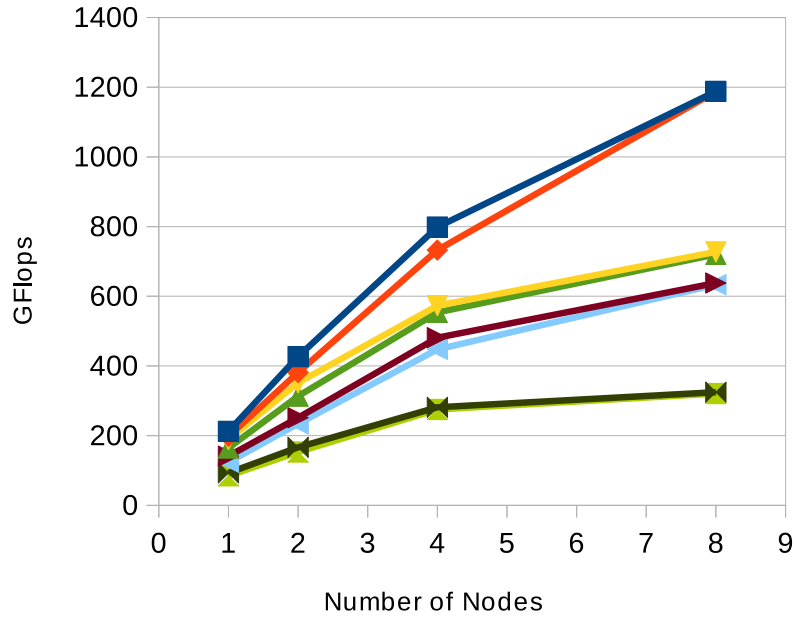
- Performance tests done in Blue Waters
  - GPUs Tesla K20X with ECC code enabled by default
  - one GPU per node
- Performance results only for SU(3)
- Overrelaxation:
  - single and multi-GPU
  - Coulomb and Landau
  - with/without using AtomicAdd()
  - with/without using texture memory
  - full SU(3) matrix and 12 real parameters memory storage
- Steepest descent method with FFTs:
  - only single GPU
  - Coulomb and Landau
  - with/without using texture memory
  - full SU(3) matrix and 12 real parameters memory storage

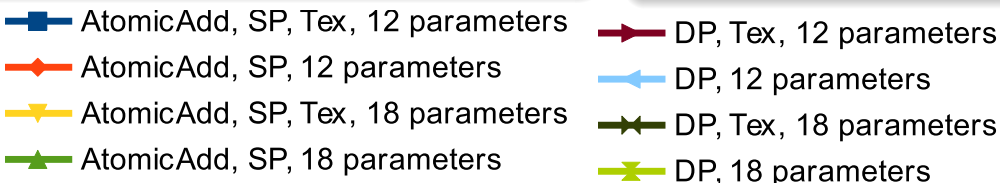
Results: Landau gauge fixing with Overrelaxation

Single GPU performance



Multi-GPU performance, 32^4 lattice volume

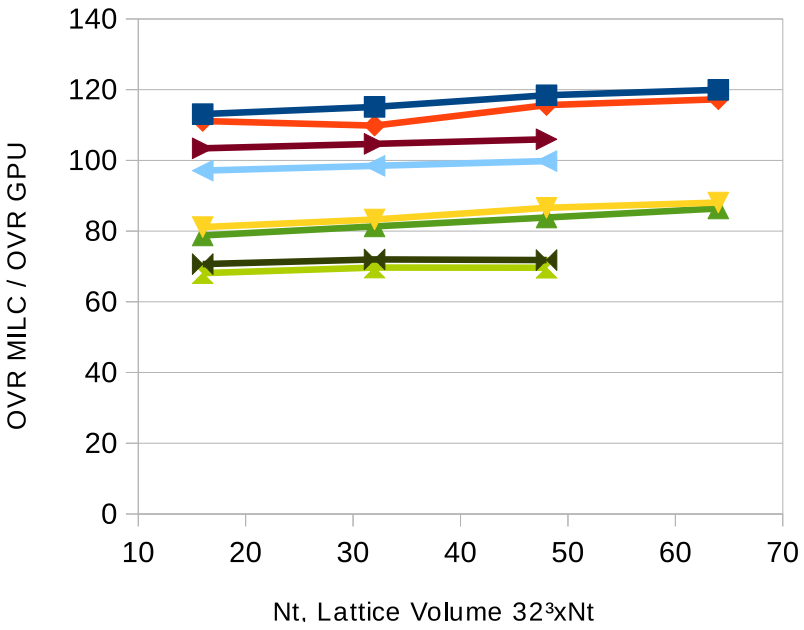


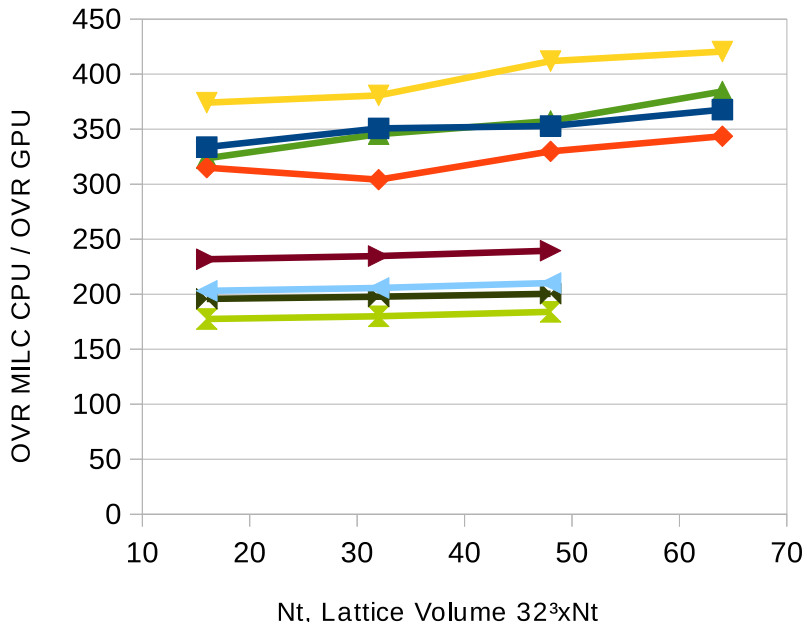


AtomicAdd means using CUDA atomicAdd function
SP/DP means single/double precision
Tex means using Texture memory
12/18 parameters are the number of SU(3) real parameters used to store gauge array in memory

Also supports 8 real parameters, no performance results done yet.

GPU Speedup over CPU MILC code in Single node





- GPU code: includes MILC ↔ GPU gauge field copy and reorder
- GPU code: only gauge fixing in GPU

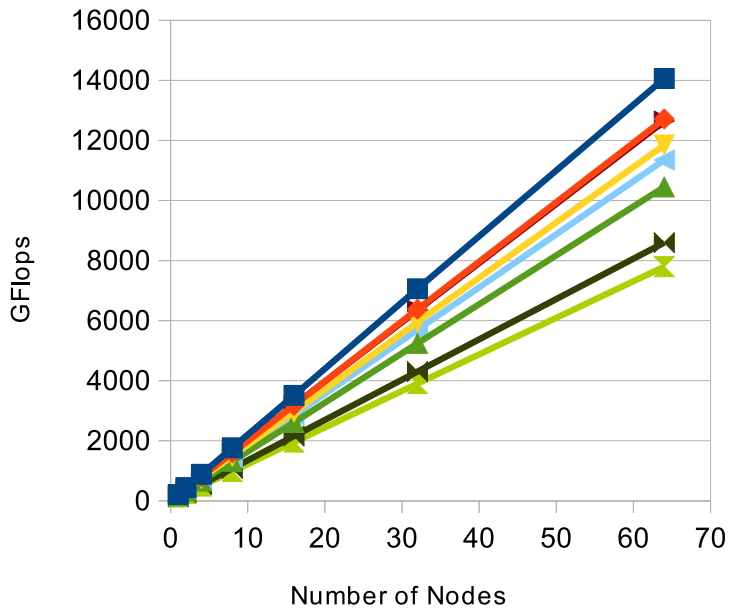
AtomicAdd, SP, Tex, 12 parameters
AtomicAdd, SP, 12 parameters
AtomicAdd, SP, Tex, 18 parameters
AtomicAdd, SP, 18 parameters
DP, Tex, 12 parameters
DP, 12 parameters
DP, Tex, 18 parameters
DP, 18 parameters

The performance impact in converting MILC format to/from GPU format as well as the copies depends on the number of the gauge fixing steps.
In this example the number of gage fixing iterations was around 170.

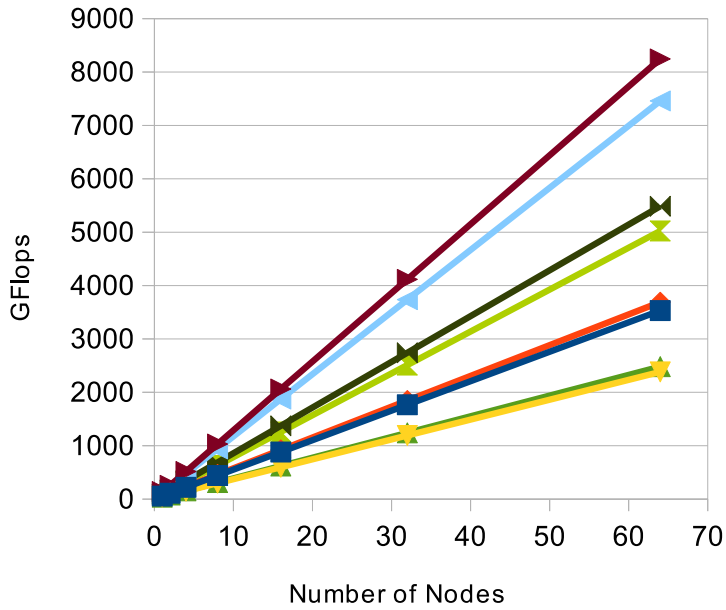
Weak Scaling

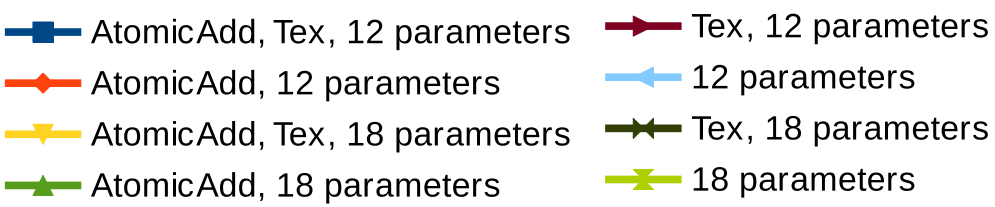
- Fixed node local lattice volume to 32^4
- Lattice volume: $32^3 \times Nt$ with $Nt = 32 \times$ (Number of nodes)

Single precision



Double precision



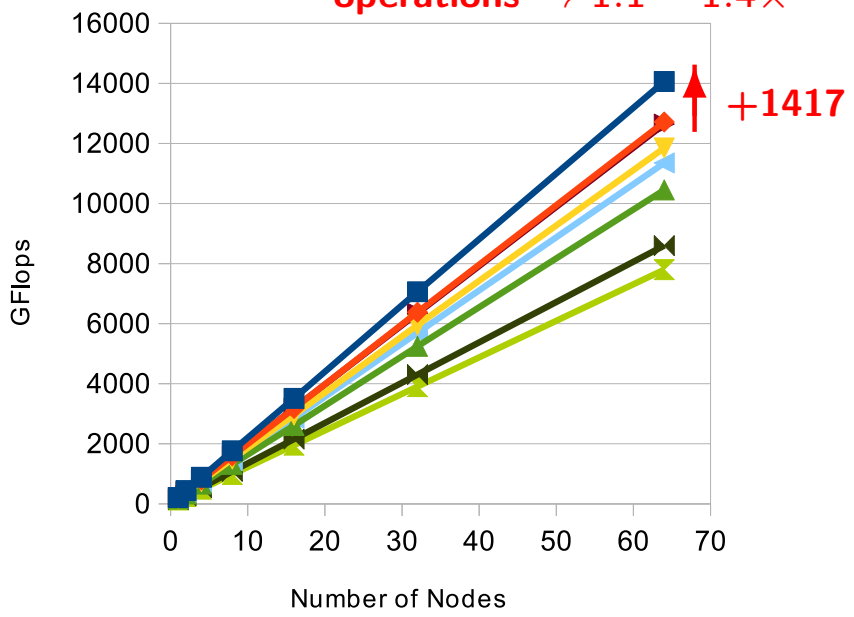


Weak Scaling

- Fixed node local lattice volume to 32^4
- Lattice volume: $32^3 \times Nt$ with $Nt = 32 \times$ (Number of nodes)

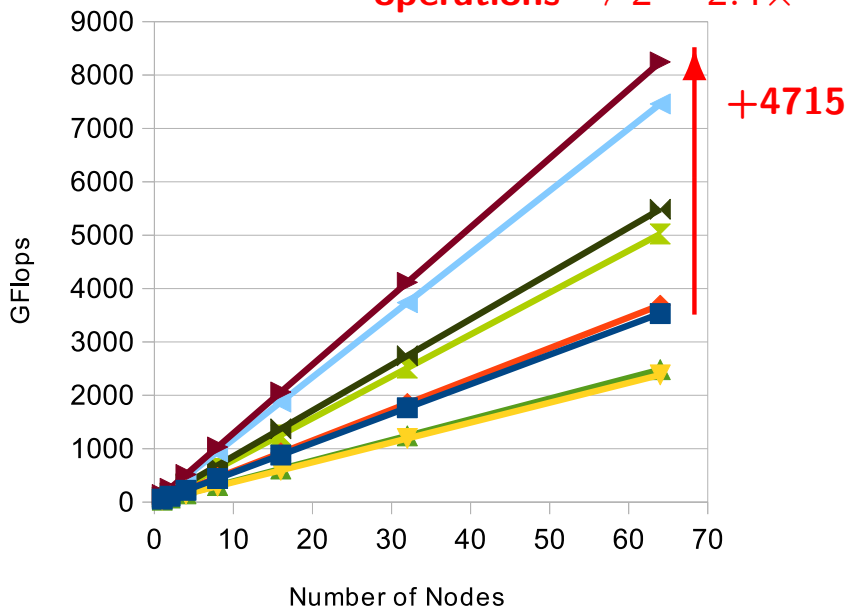
Single precision

USING CUDA Atomic operations $\rightarrow 1.1 - 1.4\times$



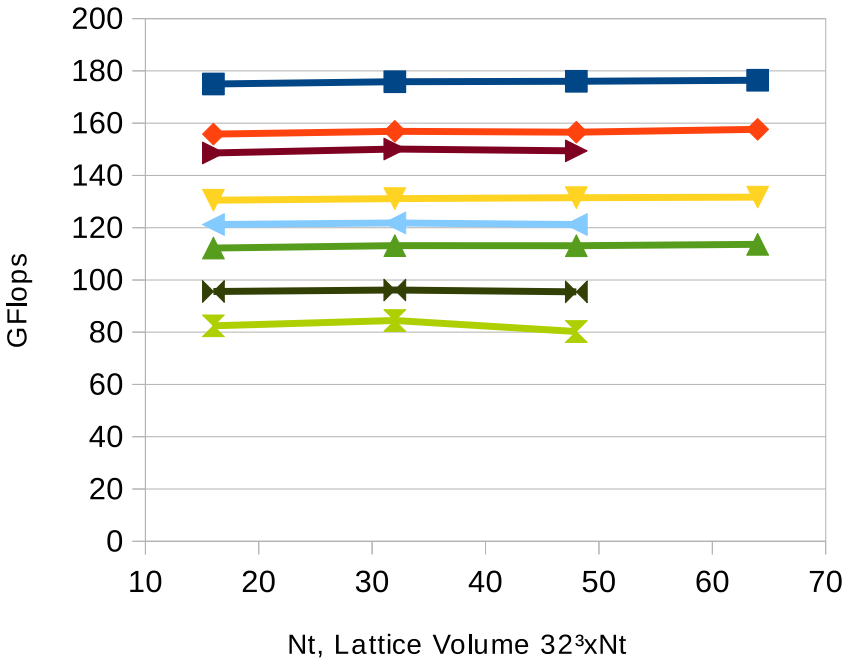
Double precision

NOT using CUDA Atomic operations $\rightarrow 2 - 2.4\times$

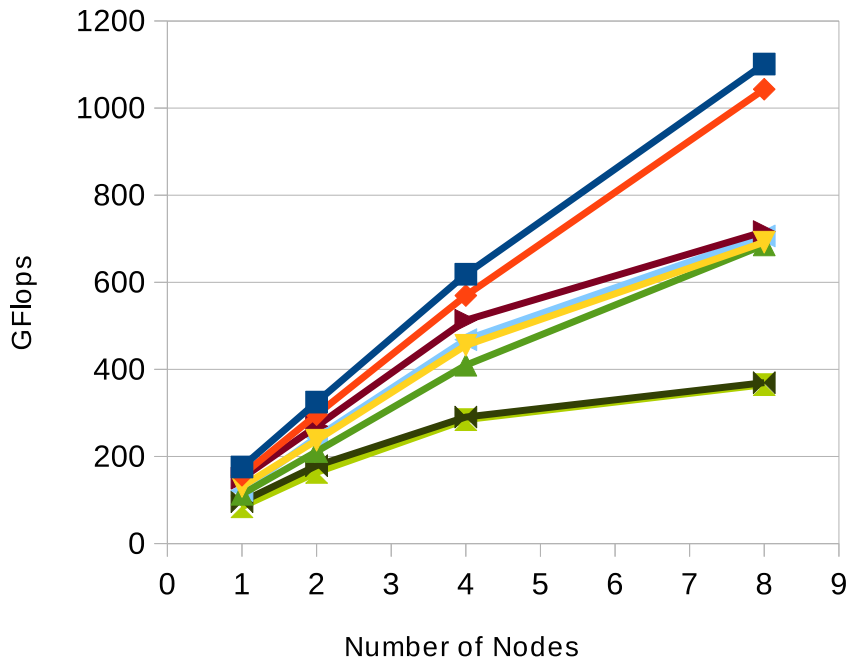


AtomicAdd, Tex, 12 parameters
AtomicAdd, 12 parameters
AtomicAdd, Tex, 18 parameters
AtomicAdd, 18 parameters
Tex, 12 parameters
12 parameters
Tex, 18 parameters
18 parameters

Single GPU performance

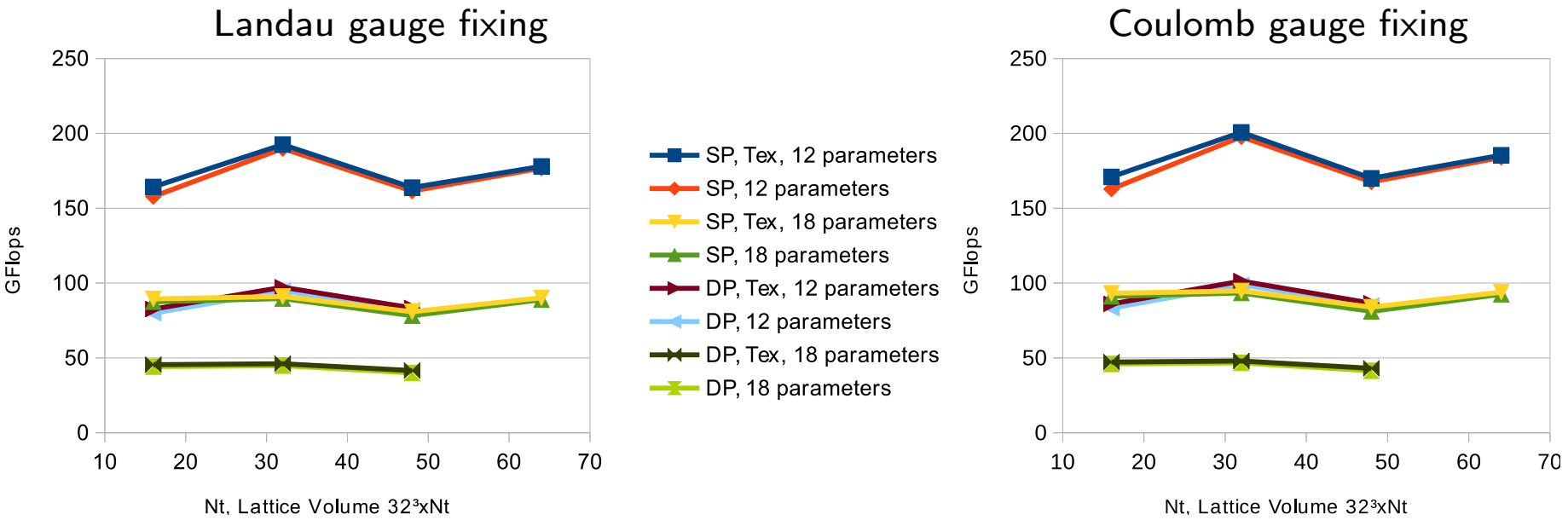


Multi-GPU performance, 32^4 lattice volume



Changed lattice partition MILC order to $T \rightarrow Z \rightarrow Y \rightarrow X$.

Results: GF with Steepest Descent algorithm with Fourier acceleration



Adapted previous code to support SU(N) with N>2, added Coulomb gauge fixing and even/odd lattice site order

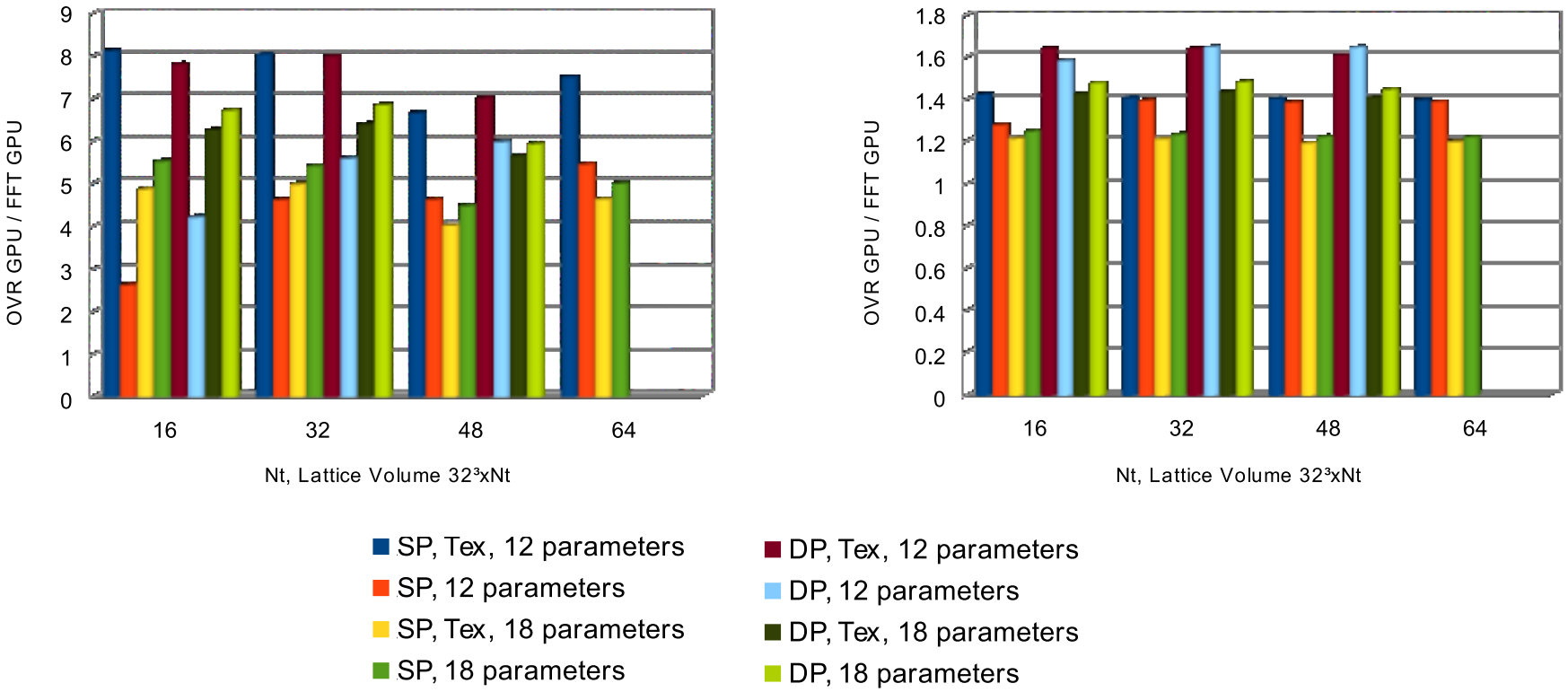
N. Cardoso, P. J. Silva, P. Bicudo, O. Oliveira, Landau Gauge Fixing on GPUs, Comput.Phys.Commun. 184 (2013) 124-129. arXiv:1206.0675, doi:10.1016/j.cpc.2012.09.007

Conclusion

- Coulomb and Landau gauge fixing implemented on GPUs
- Big speedup using GPUs to perform gauge fixing
- Overrelaxation implemented in single and multi-GPUs
- Steepest descent algorithm with Fourier acceleration converges faster than overrelaxation algorithm, in general
  However, steepest descent algorithm with Fourier acceleration requires 1.5× more memory.
- Steepest descent algorithm with Fourier acceleration not implemented in multi-GPU
  - needs multi-GPU 4D FFTs
  - needs much more data exchange between GPUs
  - needs to perform $2k$ FFTs per step, with k the number of $\Delta(x)$ elements
  - one solution to avoid using FFTs is to use a multigrid implementation of the Fourier acceleration method
    Cucchieri et. al, Phys.Rev.D57, arXiv:hep-lat/9711047, 1998.
    not implemented here!
- this library supports:
  - SU(N) with N>2, maximum N supported is limited by hardware resources
  - SU(3) parameterization with 8 and 12 real numbers

Thanks

Landau gauge fixing: GPU performance, single node



Using MILC stop criterium 10^{-7}.
In this example, the steepest descent method with FFTs only needed around 17 iterations to converged and the overrelaxation around 140-180 iterations.

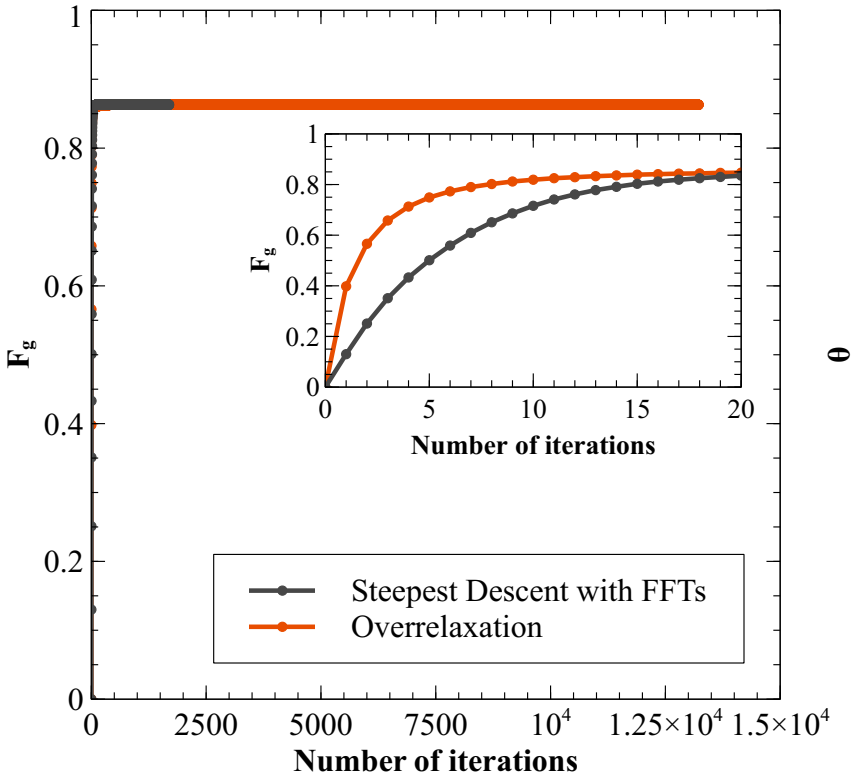
- Speed up of the GPU Landau gauge fixing with the steepest descent method with FFTs over GPU Landau gauge fixing with overrelaxation. Right figure includes the time to copy and reorder from MILC format to GPU format, the left figure does not include this time.

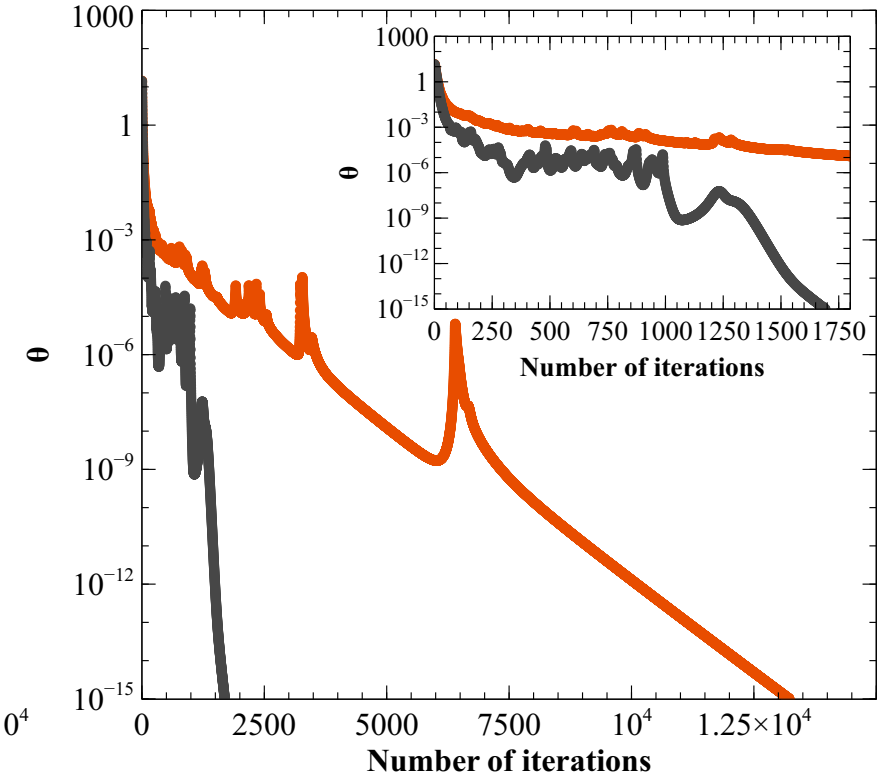
Gauge fixing quality results example

MILC input file for su3_ora:

```
prompt 0
nx 32
ny 32
nz 32
nt 32
(...)
trajecs 5
traj_between_meas 1
beta 6.0
steps_per_trajectory 4
qhb_steps 4
(...)
```

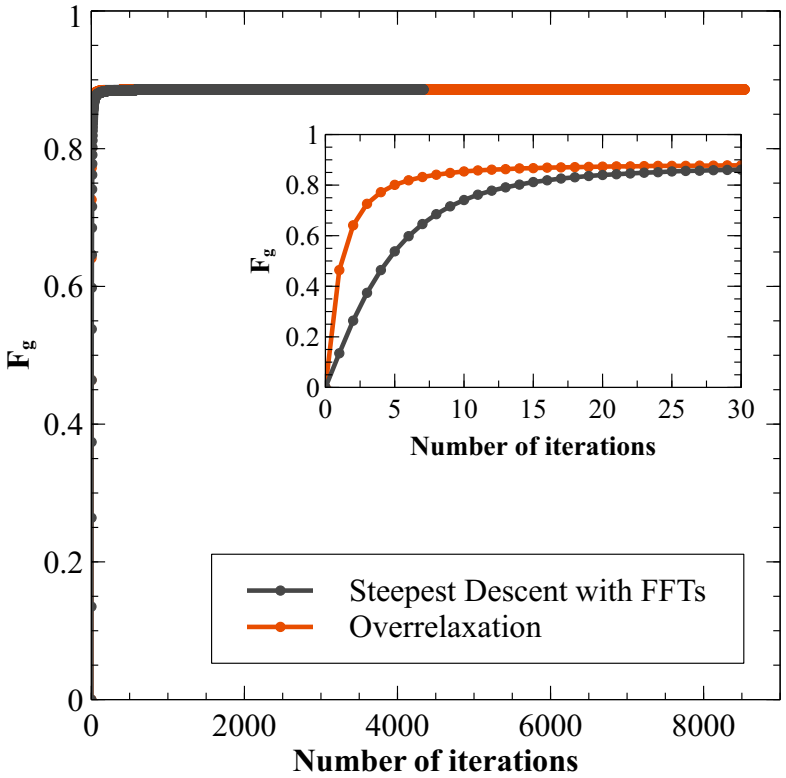
Landau gauge fixing quality results

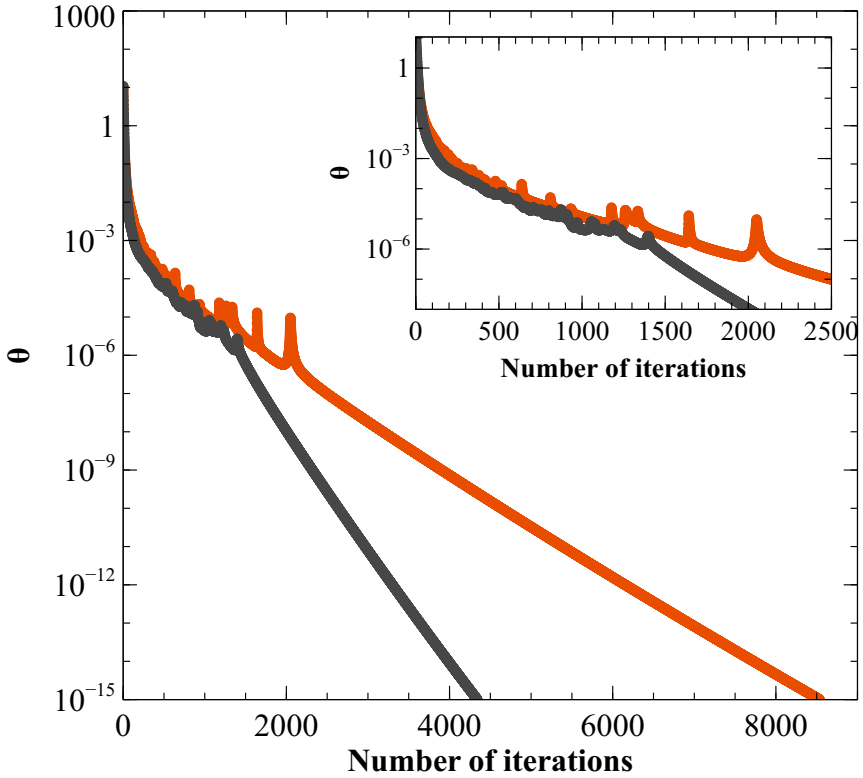




Number of iterations to achieve a gauge fixing quality of $\theta < 10^{-15}$. In these results, θ parameter was used as stop criterium instead of the default MILC criterium.

Coulomb gauge fixing quality results





Number of iterations to achieve a gauge fixing quality of $\theta < 10^{-15}$. In these results, θ parameter was used as stop criterium instead of the default MILC criterium.