EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Science
Institute for Theoretical Physics

cuLGT: Lattice Gauge Fixing on GPUs

Landau, Coulomb and Maximally Abelian gauge

10.09.2014, Hannes Vogt, Mario Schröck (INFN Roma Tre)

INFN
Istituto Nazionale
di Fisica Nucleare

Motivation

- QCD action is gauge invariant
- Fixing a gauge is necessary in the continuum
- Gauge-variant two point functions play an important role in confinement scenarios

- On the lattice:
  - high dimensional optimization problem
  - many local maxima: Gribov copies
  - algorithms are nicely parallelizable

- cuLGT
  - CUDA-based code for gauge fixing and more
  - code is publicly available: www.culgt.com

Outline

- Gauge fixing on the lattice
- Standard optimizations
- More optimization: 8 threads per site
- Performance of cuLGT1
- Improvements: cuLGT2

Gauge fixing on the lattice: Landau gauge

- Action is invariant under gauge transformation of the links

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

- Landau gauge ($\partial_\mu A_\mu(x) = 0$) is achieved by maximizing

$$F^U[g] = \frac{1}{N_d N_c V} \sum_x \sum_\mu \mathrm{Re}\,\mathrm{tr}\left[g(x)U_\mu(x)g^\dagger(x+\hat{\mu})\right]$$

Gauge fixing on the lattice: Landau gauge

- Action is invariant under gauge transformation of the links

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

- Landau gauge ($\partial_\mu A_\mu(x) = 0$) is achieved by maximizing

$$F^U[g] = \frac{1}{2N_d N_c V} \sum_x \mathrm{Re}\,\mathrm{tr}\,[g(x)K(x)]$$

Gauge fixing on the lattice: Landau gauge

- Action is invariant under gauge transformation of the links

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

- Landau gauge ($\partial_\mu A_\mu(x) = 0$) is achieved by maximizing

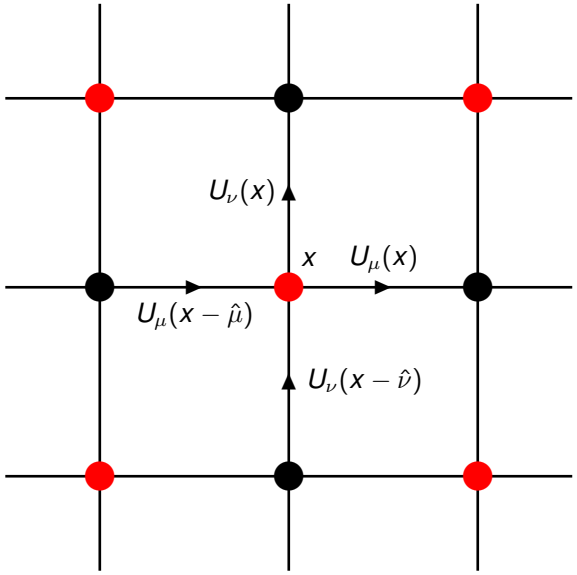
$$F^U[g] = \frac{1}{2 N_d N_c V} \sum_x \mathrm{Re\,tr}\,[g(x) K(x)]$$

- optimize locally:

$$g(x)K(x) = g(x) \sum_\mu \left[U_\mu(x) \overbrace{g(x+\hat{\mu})^\dagger}^{\mathbb{1}} + U_\mu(x-\hat{\mu})^\dagger \overbrace{g(x-\hat{\mu})^\dagger}^{\mathbb{1}} \right]$$

- SU(2): optimum for $g(x) = K^\dagger(x) / \det(K^\dagger)$
  for $N_c > 2$ iterate over SU(2) subgroups

Gauge fixing on the lattice: Landau gauge



update:

- $U_\mu(x) \to g(x) U_\mu(x)$
- $U_\mu(x-\hat{\mu}) \to U_\mu(x-\hat{\mu}) g(x)^\dagger$
- ...

iterate until:

$$\theta \approx \max_x |\partial_\mu A_\mu(x)| < \epsilon$$

- overrelaxation: $g(x) \to g(x)^\omega, \omega \in [1,2)$
- simulated annealing: $\mathrm{d}P(g) = \mathrm{e}^{-\frac{1}{T}\mathrm{Re}\,\mathrm{tr} g(x)K(x)}\mathrm{d}g$

Pseudocode

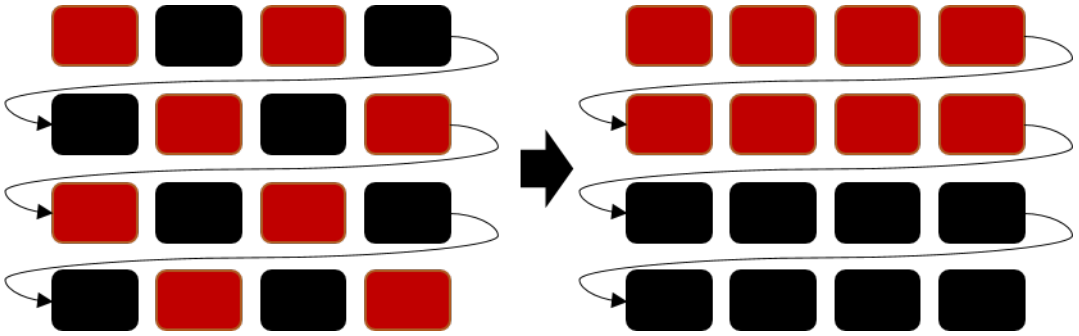
Algorithm 1

while precision θ not reached **do**
  for sublattice = even, odd **do**
    for all x of sublattice **do**
      for all SU(2) subgroups **do**
        local optimization: find $g(x) \in$ SU(2)
        which is a function of $U_\mu(x),\ U_\mu(x-\hat{\mu})$
        for all μ **do**
          apply $g(x)$ to $U_\mu(x),\ U_\mu(x-\hat{\mu})$
        end for
      end for
    end for
  end for
end while

Standard optimizations for GPUs

M. A. Clark et al.,
Comput.Phys.Commun. 181 (2010)

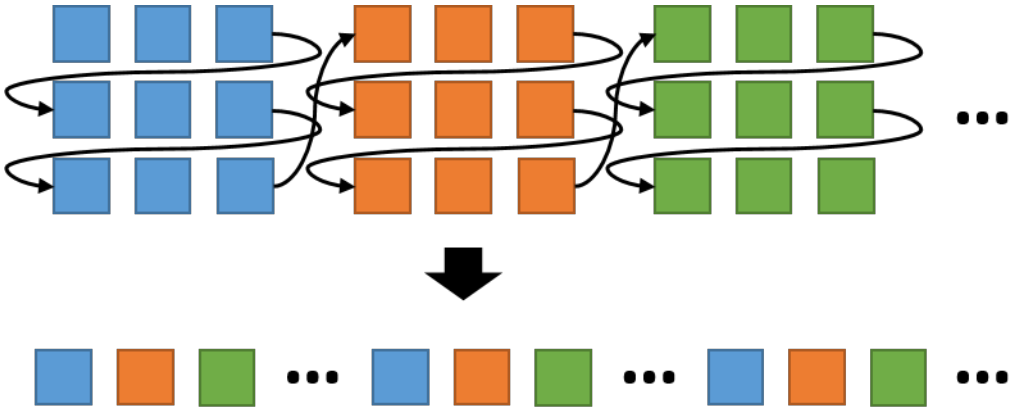
- reorder memory layout for coalescing:
  - memory split into even/odd sublattice



Standard optimizations for GPUs

M. A. Clark et al.,
Comput.Phys.Commun. 181 (2010)

- reorder memory layout for coalescing:
  - memory split into even/odd sublattice
  - site index running faster than color/Dirac indices



Standard optimizations for GPUs

M. A. Clark et al.,
Comput.Phys.Commun. 181 (2010)

- reorder memory layout for coalescing:
  - memory split into even/odd sublattice
  - site index running faster than color/Dirac indices
- code is memory bound:
  - store only parts of the $N_c \times N_c$ matrix in global memory
  - reconstruct in registers using SU(N_c) properties
  - SU(3): 12 parameters instead of 18

Standard optimizations for GPUs

M. A. Clark et al.,
Comput.Phys.Commun. 181 (2010)

- reorder memory layout for coalescing:
  - memory split into even/odd sublattice
  - site index running faster than color/Dirac indices
- code is memory bound:
  - store only parts of the $N_c \times N_c$ matrix in global memory
  - reconstruct in registers using SU(N_c) properties
  - SU(3): 12 parameters instead of 18
- use texture cache to load links

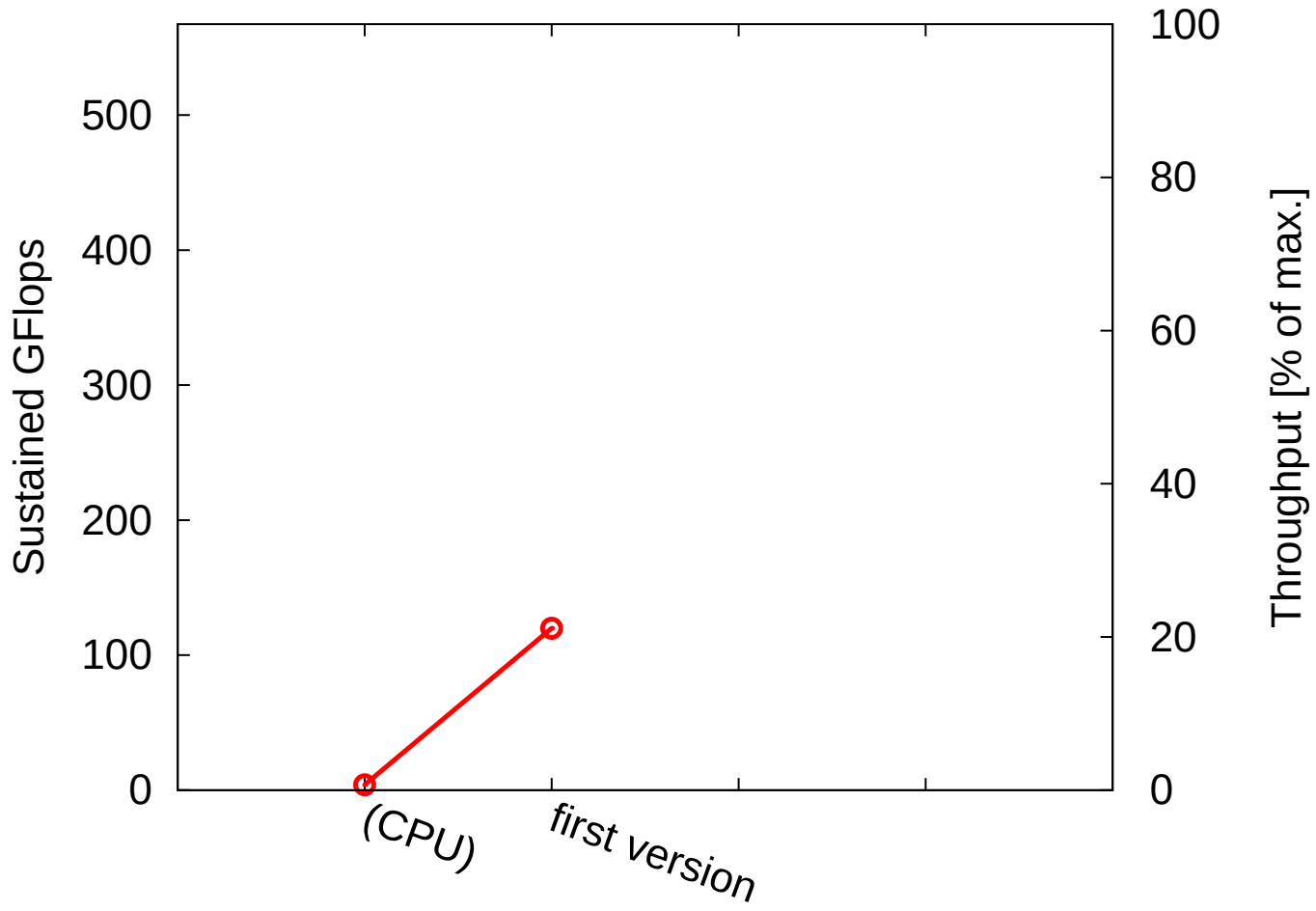
Standard optimizations for GPUs

M. A. Clark et al.,
Comput.Phys.Commun. 181 (2010)

- reorder memory layout for coalescing:
  - memory split into even/odd sublattice
  - site index running faster than color/Dirac indices
- code is memory bound:
  - store only parts of the $N_c \times N_c$ matrix in global memory
  - reconstruct in registers using SU(N_c) properties
  - SU(3): 12 parameters instead of 18
- use texture cache to load links

- our standard setup:
  - GTX 580 (Fermi)
  - 32^4 SU(3) lattice in SP
  - Landau gauge, Overrelaxation

Performance: Overrelaxation



Performance limit: register spilling

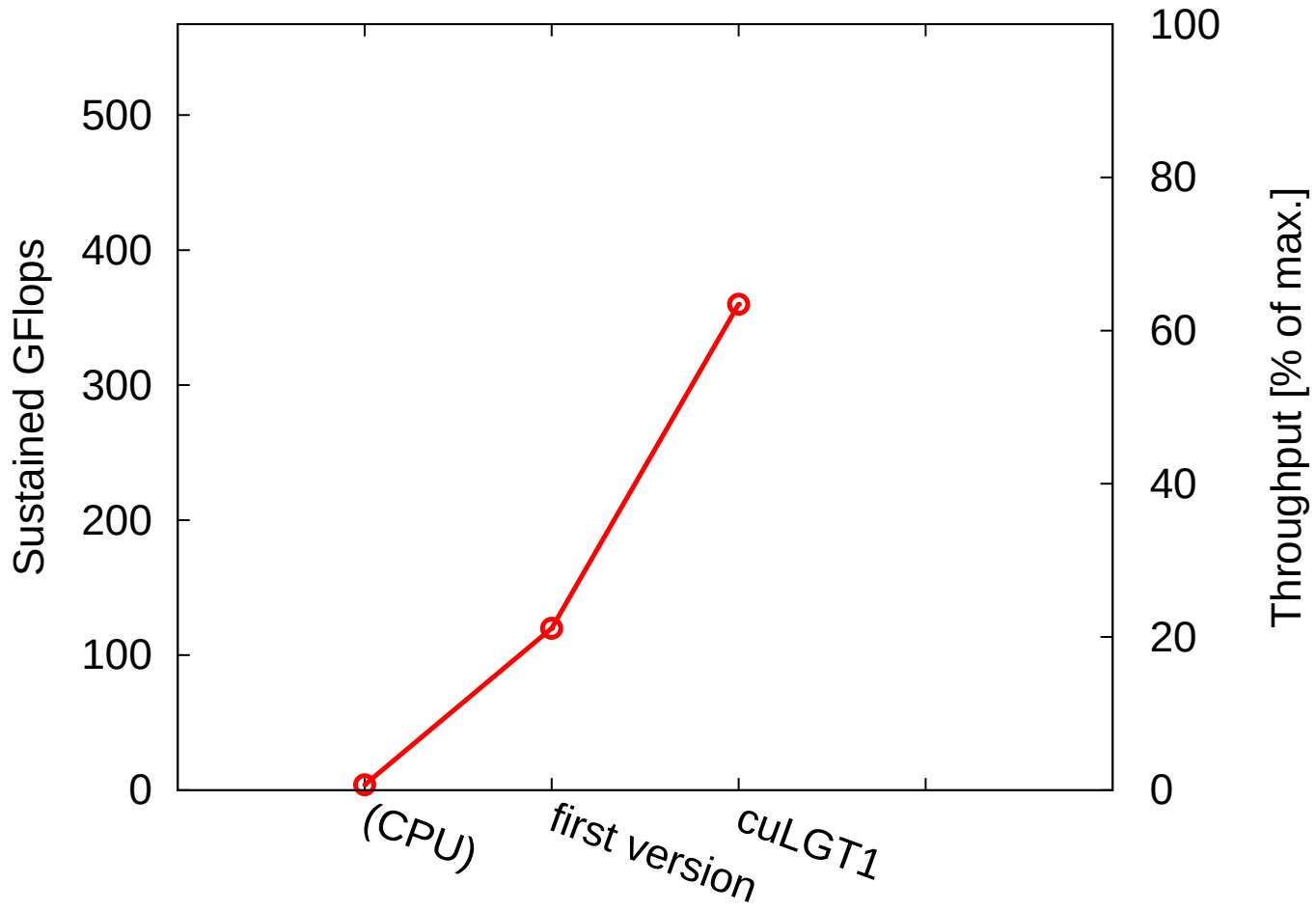
- at each site: 8 links * 18 parameters = 144 floats
- Fermi: only 63 32-bit registers/kernel
- kernel signature: 63 registers, 512 Byte stack frame, 964 Byte spill stores, 1232 Byte spill loads

Solution:

- 8 threads per site (1 thread per link)
  - only 18 floats (registers) for the link
  - communication via shared memory

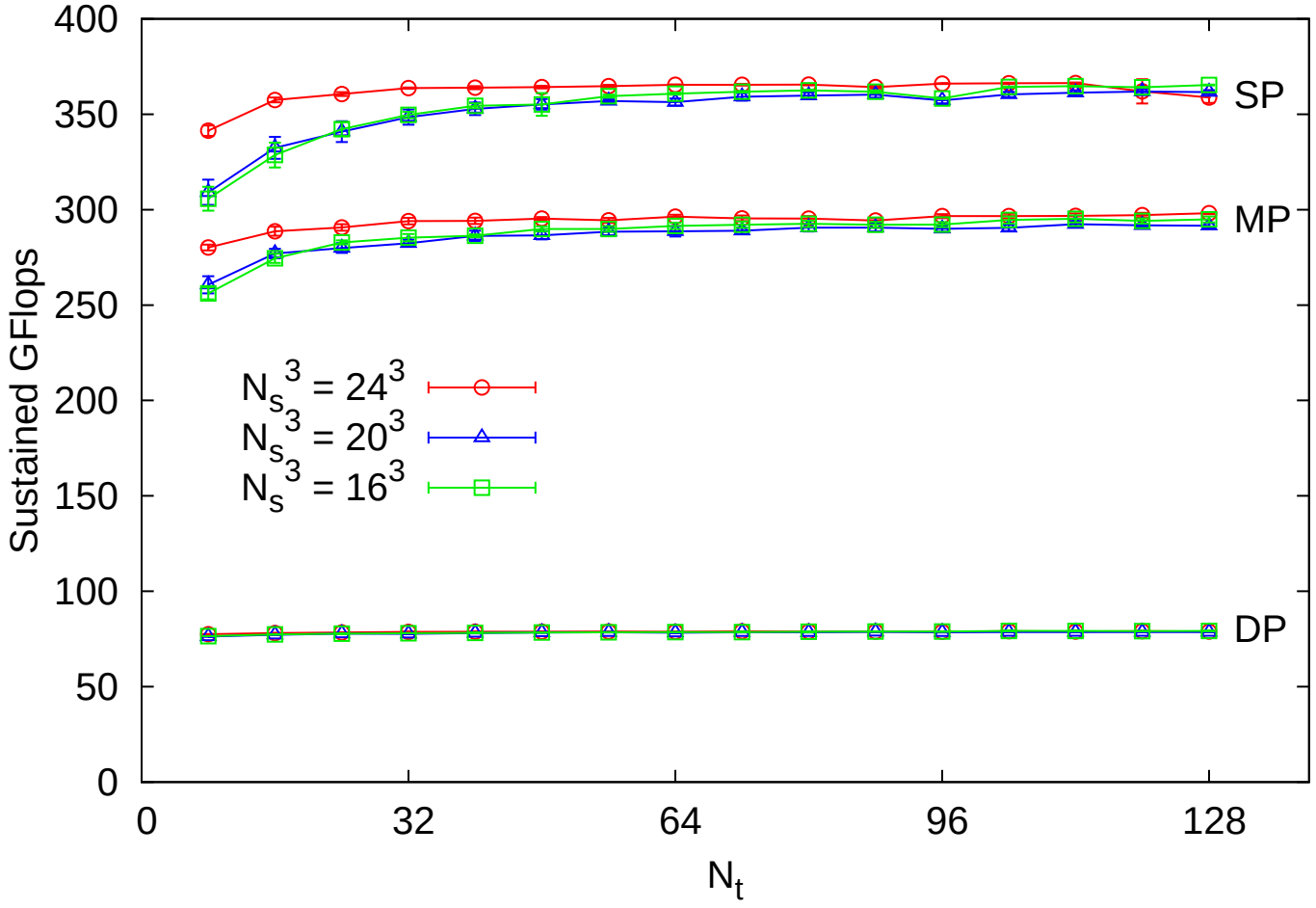
  - in practice: 34 registers (no register spilling)
  - optimized for higher occupancy: 32 register

Performance: Overrelaxation



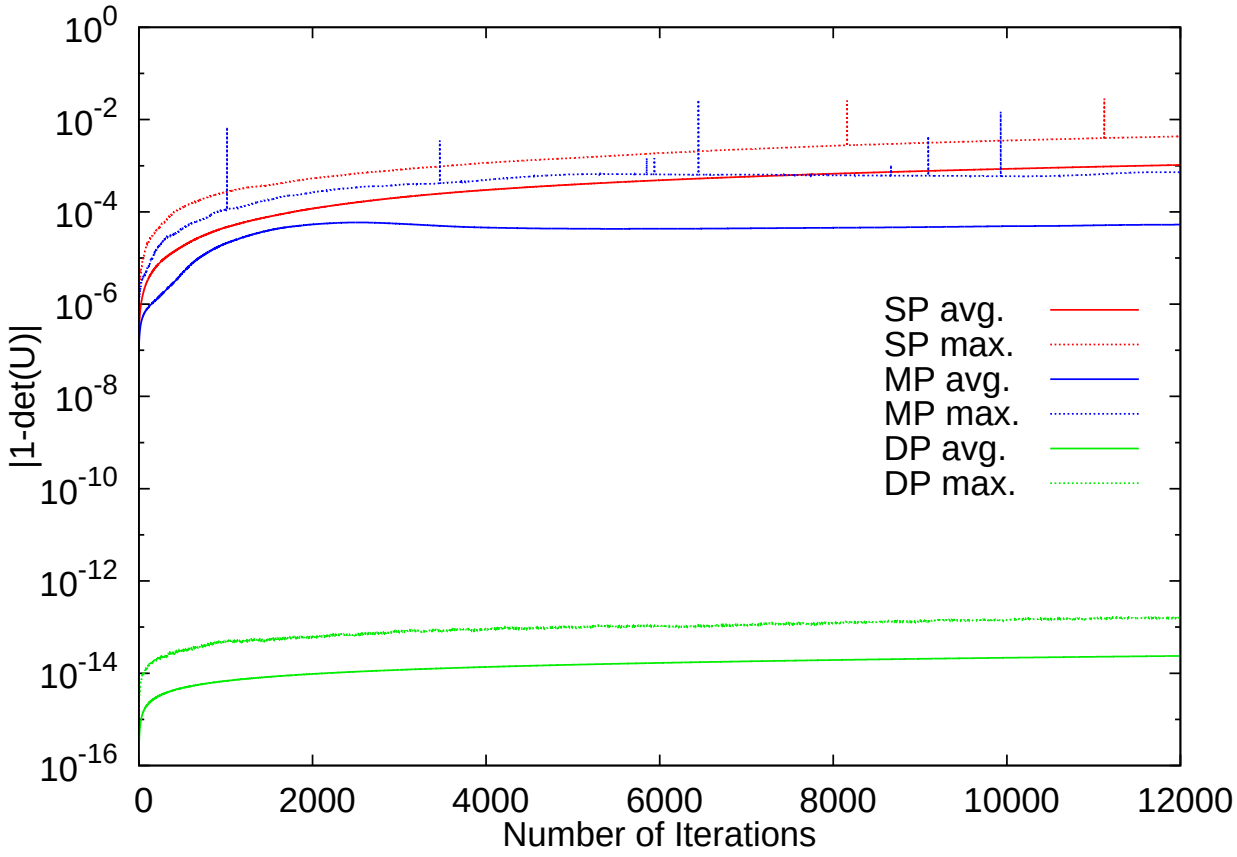
Performance: Overrelaxation

MS, HV,
Comput.Phys.Commun. 184 (2013)



Conservation of unitarity

MS, HV,
Comput.Phys.Commun. 184 (2013)



cuLGT2

Problems of cuLGT1

- structural problems
  - code evolved over time
  - readability/maintainability vs. performance
  - not well modularized
  - compile time constants
  - two codes for SU(2) and SU(3)
- performance obstacles
  - parameterization of links hardcoded
  - not easy to switch to 4 threads/site
  - no autotune possibility

$\Rightarrow$ time for a complete makeover

cuLGT2: Link parameterization

- in cuLGT1:
  - large array of float/double variables
  - `GlobalLink<Pattern> link(x,mu)` $\rightsquigarrow$ array
  - `LocalLink link` $\rightsquigarrow$ 3×3 complex matrix
- links $U_\mu(x)$ are optimally coalesced, neighbours $U_\mu(x - \hat{\mu})$ not!
  - float4/double2 would be better for non-coalesced loads

- solution: new layer of abstraction
  - parameterization: choose datatype & number of elements
  - examples for SU(3):
    - $12 \times$ `float`
    - $3 \times$ `float4`
    - $9 \times$ `Complex<float>`
    - $18 \times$ `float`

cuLGT2: an example

```
typedef LocalLink<SUNRealFull<3,float> > LOCALLINK;

typedef SU3Vector4<float> PARAMTYPE;
typedef GPUPattern<SITETYPE,PARAMTYPE> PATTERNTYPE;
typedef GlobalLink<PATTERNTYPE,USETEXTURE> GLOBALLINK;

__global__ polyakovLoop( float4* U, float* polyakov, LatDim<4> dim )
{
  SITETYPE site( blockIdx.x * blockDim.x + threadIdx.x );
  LOCALLINK linkProduct; linkProduct.identity();
  for( int t = 0; t < dim.getDimension( TDIR ); t++ )
  {
    GLOBALLINK glob( U, site, TDIR );
    LOCALLINK link = glob;
    linkProduct *= link;
    site.setNeighbour( TDIR );
  }
  polyakov[site.getIndex()] = linkProduct.retrace();
}
```

cuLGT2: an example

```
typedef LocalLink<SUNRealFull<3,float> > LOCALLINK;

typedef SU3Vector4<float> PARAMTYPE;
typedef GPUPattern<SITETYPE,PARAMTYPE> PATTERNTYPE;
typedef GlobalLink<PATTERNTYPE,USETEXTURE> GLOBALLINK;

__global__ polyakovLoop(...) {...}

int main()
{
  LatDim<4> dim( Nt, Nx, Ny, Nz );
  GaugeConfiguration<PATTERN> config( dim );
  config.allocateMemory();
  // allocate dPlaquette, load the configuration to host memory
  config.copyToDevice();

  GLOBALLINK.bindTexture( config.getDevicePointer(), config.getSize() );
  polyakovLoop<<<GRIDSIZE,BLOCKSIZE>>>( config.getDevicePointer(),
     dPolyakov, dim );

  reduction( dPolyakov );
}
```

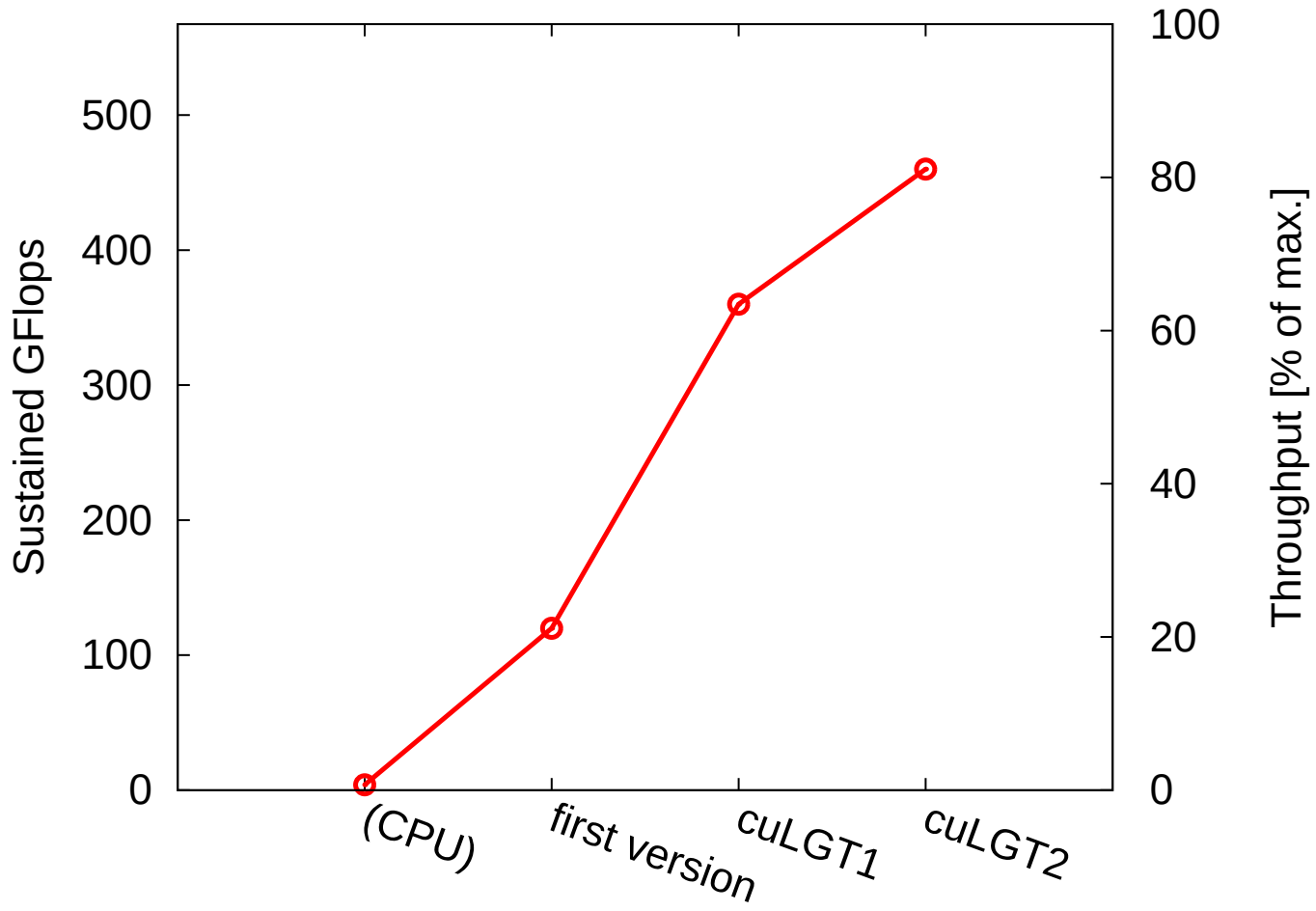
cuLGT2: what do we have now?

- ✔ Flexible datastructure
  - easily switch from SU(3) to SU(2)
  - easily try different optimized parameterizations
- ✔ Modularized:
  - MILC interface
- ✔ Autotune utility selects optimal setup for different architectures
  - 4 or 8 threads per site
  - different blocksizes
- ✔ Unit tests

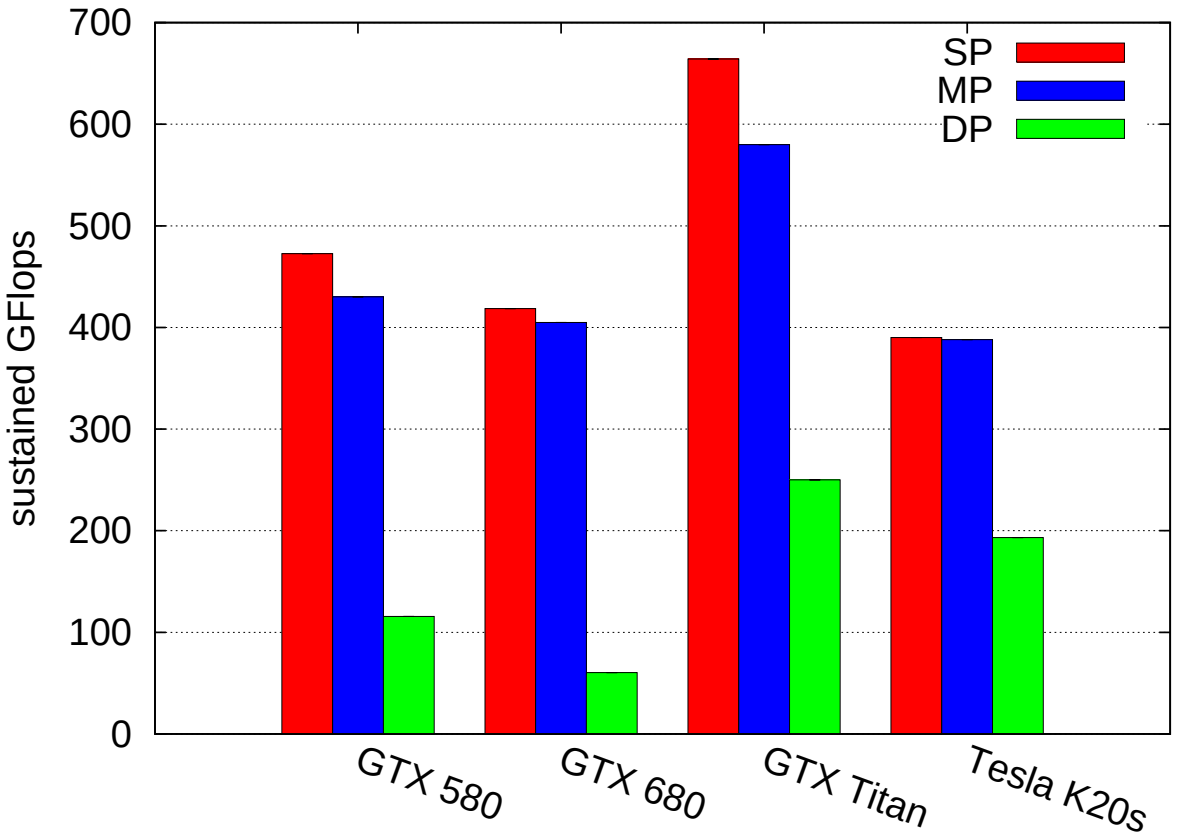
- ✗ not yet: MultiGPU

- ? Performance

Performance: Overrelaxation



cuLGT2: Performance on different GPUs



Summary: What does cuLGT offer?

- gauge types:
  - Landau gauge
  - Coulomb gauge
  - Maximally Abelian gauge
- algorithms:
  - Overrelaxation
  - Simulated Annealing
- groups:
  - SU(2)
  - SU(3)
  - SU(N) for Landau/Coulomb gauge easy to implement
- MultiGPU: only Landau gauge
- integration in other frameworks: MILC

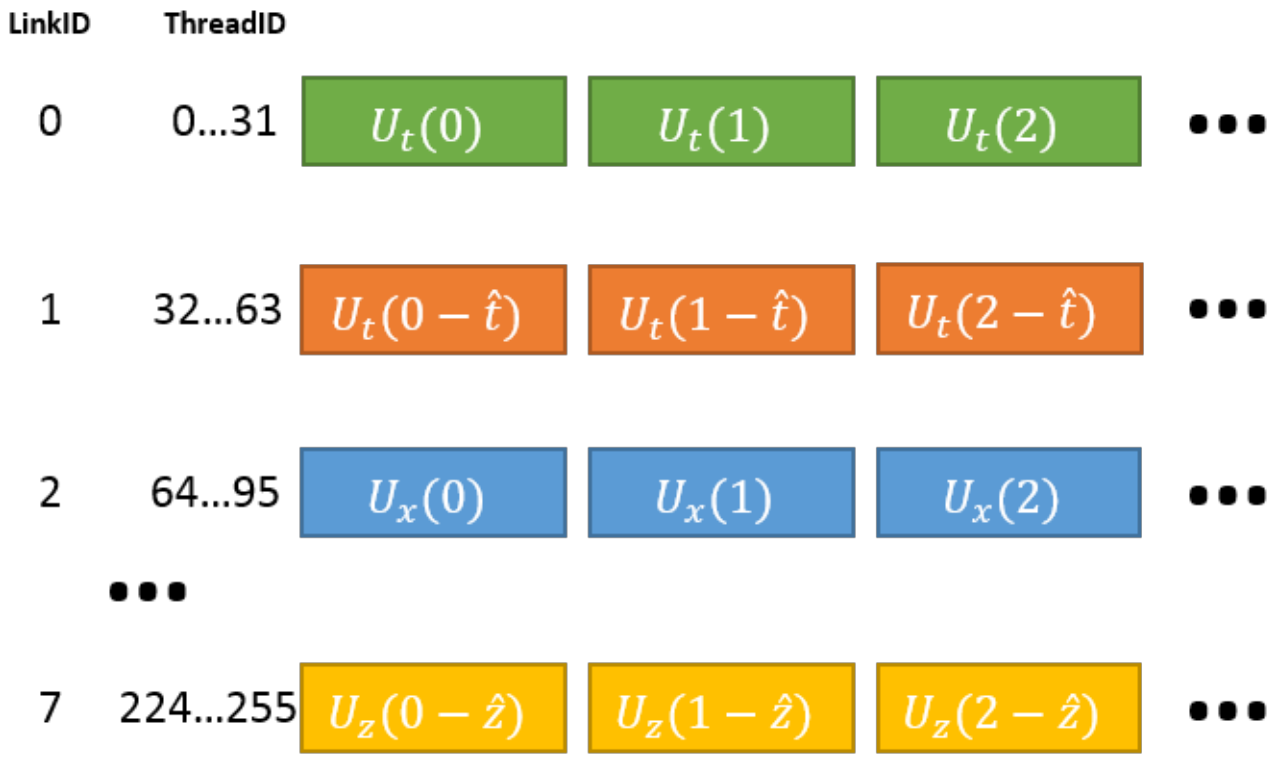
Thank you.

Contact:

Faculty of Science
Institute for Theoretical Physics
Auf der Morgenstelle 14
72076 Tübingen

hannes.vogt@uni-tuebingen.de

8 threads/site: sketch



8 threads/site: Pseudocode

Algorithm 2 eight-threads-per-site kernel

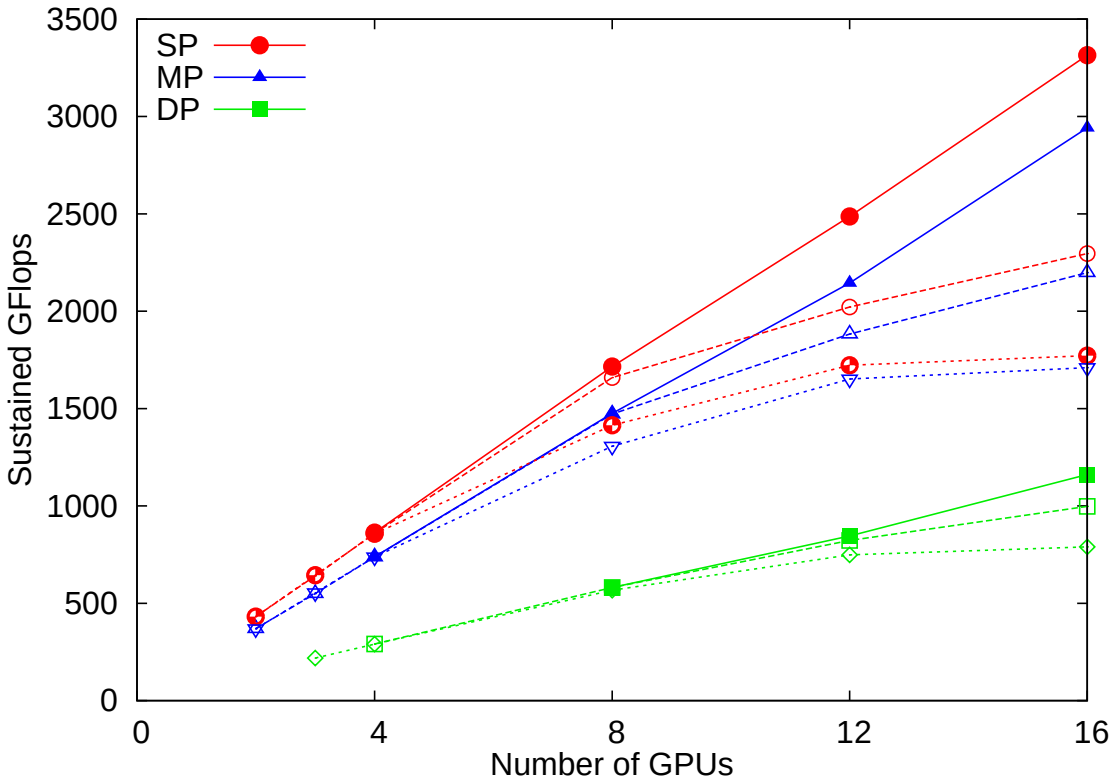
```
__shared__ g
loadAndReconstructLink( x, μ ) → U
for all SU(2) subgroups do
  getSubgroup( U, i ) → u
  atomicAdd( K, u )
  __syncthreads()
  if LINK_ID == 0 then
    g† / det(g†) → g
  end if
  __syncthreads()
  update(U,g)
end for
saveLink( U )
```

MultiGPU

- split the lattice along the time direction:
  - Nt/nprocs timeslices per GPUs
  - temporal links of the neighbouring timeslice at boundary
- data layout: timeslices split in parity
- asynchronous memory transfer (MPI)
  - update inner timeslices
  - asynchronously copy boundary timeslice, update when finished
  - (only inactive parity needs to be copied)
- $\sim$ 8 timeslices per GPU to hide data exchange

MultiGPU: strong scaling

MS, HV,
Comput.Phys.Commun. 184 (2013)

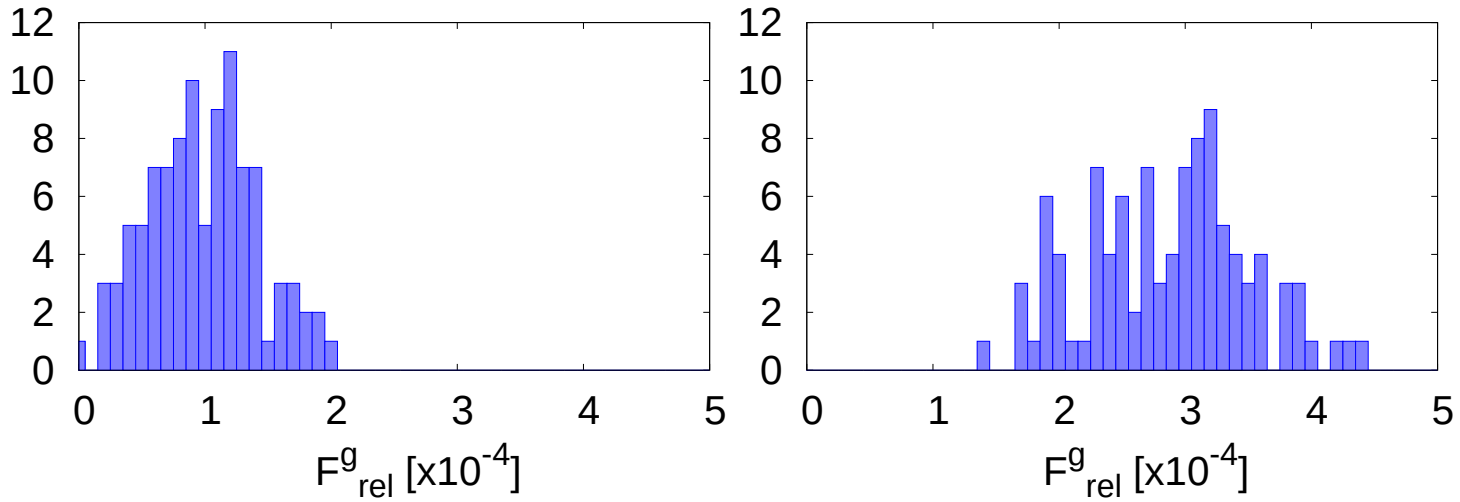


on Tesla C2070: $64^3 \times N_t$ with $N_t = 256, 128, 96$ (top to bottom)

Simulated Annealing

MS, HV,
Comput.Phys.Commun. 184 (2013)

- used to find the global maximum
- propose a gauge transformation in the heatbath of surrounding links $\mathrm{d}P(g) = \mathrm{e}^{-\frac{1}{T}\mathrm{Re}\,\mathrm{tr} g(x)K(x)}\mathrm{d}g$



The relative deviation from the maximal gauge functional.
l.h.s.: 10000, r.h.s. no simulated annealing steps.

Conservation of the plaquette