LatticeGPU.jl

A julia code for Lattice QCD on GPUs

Fernando P. Panadero
and Guilherme Catumba, Carlos Pena, Alberto Ramos

IFT UAM-CSIC

41st Lattice Conference, University of Liverpool, July 30th

Motivation

- Need for GPU codes.
- Fast to develop while having high performance.
- Suitable for small and medium lattice studies (such as Finite size scaling).

ALPHA Collaboration

- Renormalization schemes using fermion gradient flow.

Outline

- **LatticeGPU.jl**: Open source package to perform Lattice simulations using GPUs developed in Julia.

CUDA.jl

Available @ **https://igit.ific.uv.es/alramos/latticegpu.jl**

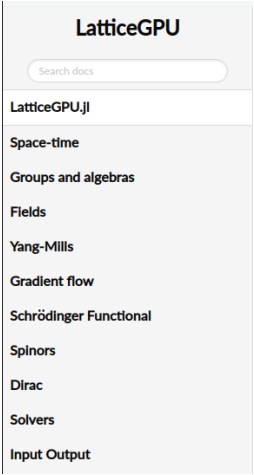
Outline

- **LatticeGPU.jl**: Open source package to perform Lattice simulations using GPUs developed in Julia.
- Documentation available @ **https://ific.uv.es/~alramos/docs/LatticeGPU/**
- Available features in the code.
- Some tests and performance.
- Example main codes.
- Conclusions.

Code available @ **https://igit.ific.uv.es/alramos/latticegpu.jl**

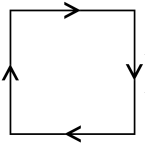
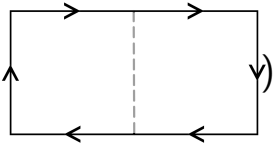
Boundary Conditions

- Periodic Boundary Conditions
- Schrödinger Functional BC
  - ➔ Aoki-Frezzotti-Weisz Choice B [hep-lat,9808007]
  - ➔ Orbifold construction (oqcd)
- Open BC
- Twisted BC (Gauge only)
- θ-BC for fermions



Monte-Carlo generation

- Yang-Mills action for 1×1 and 2×1 Wilson loops.

$$S_G^{\text{bulk}} = \frac{1}{g_0^2} \sum c_0 Tr\,(1-$$  $$) + c_1 Tr\,(1-$$  $$)$$

- HMC integrators with different precision.

$$\dot{P}_\mu = -\frac{\partial S_G}{\partial U_\mu} \qquad \dot{U}_\mu = P_\mu$$

- Available for SU(2) and SU(3) gauge groups. → Easy to extend

Monte-Carlo generation

```julia
function HMC!(U, int::IntrScheme, lp::SpaceParm, gp::GaugeParm,
              ymws::YMworkspace{T}; noacc=false) where T

    @timeit "HMC trayectory" begin
        ymws.U1 .= U
        randomize!(ymws.mom, lp, ymws)
        hini = hamiltonian(ymws.mom, U, lp, gp, ymws)

        MD!(ymws.mom, U, int, lp, gp, ymws)

        dh   = hamiltonian(ymws.mom, U, lp, gp, ymws) - hini
        pacc = exp(-dh)
        acc = true
        if (noacc)
            return dh, acc
        end
        if (pacc < 1.0)
            r = rand()
            if (pacc < r)
                U .= ymws.U1
                acc = false
            end
        end
        U .= unitarize.(U)
    end
    return dh, acc
end
```

```julia
struct SU3{T} <: Group
    u11::Complex{T}
    u12::Complex{T}
    u13::Complex{T}
    u21::Complex{T}
    u22::Complex{T}
    u23::Complex{T}
end


function inverse(a::SU3{T}) = ...
function dag(a::SU3{T})     = ...
function tr(a::SU3{T})      = ...

function Base.:*(a::SU3{T},b::SU3{T}) where T <: AbstractFloat

    bu31 = conj(b.u12*b.u23 - b.u13*b.u22)
    bu32 = conj(b.u13*b.u21 - b.u11*b.u23)
    bu33 = conj(b.u11*b.u22 - b.u12*b.u21)

    return SU3{T}(a.u11*b.u11 + a.u12*b.u21 + a.u13*bu31,
                  a.u11*b.u12 + a.u12*b.u22 + a.u13*bu32,
                  a.u11*b.u13 + a.u12*b.u23 + a.u13*bu33,
                  a.u21*b.u11 + a.u22*b.u21 + a.u23*bu31,
                  a.u21*b.u12 + a.u22*b.u22 + a.u23*bu32,
                  a.u21*b.u13 + a.u22*b.u23 + a.u23*bu33)
end
function Base.:/(a::SU3{T},b::SU3{T}) where T <: AbstractFloat
    ...
end
function Base.:\(a::SU3{T},b::SU3{T}) where T <: AbstractFloat
    ...
end
function unitarize(g::SU3{T}) where T <: AbstractFloat
    ...
end
```

Measurements

- **Yang-Mills Gradient Flow**
  - ➔ Wilson and Zeuthen flow

$$a^2\partial_t V_\mu(t,x) = \left(-g_0^2\partial_{x,\mu}S[V]\right)V_\mu(t,x) \qquad V_\mu(0,x) = U_\mu(x)$$

$$a^2\partial_t V_\mu(t,x) = \left[-g_0^2\left(1+\frac{a^2}{12}\nabla_\mu^*\nabla_\mu\right)\partial_{x,\mu}S_{LW}[V]\right]V_\mu(t,x)$$

  - ➔ Fixed and adaptive step size. A. Ramos' talk, Fri @ 14:15

$$V_\mu(t,x) \quad \to \quad V_\mu(t+\epsilon,x) \quad \to \quad ... \quad \to V_\mu(t+9\epsilon,x) \quad \to \quad V_\mu(t+10\epsilon,x) \quad \to \quad ...$$

  - ➔ Plaquette and Clover energy density, Topological charge

[M. Lüscher, ArXiv: 1006.4518][A. Ramos and S. Sint, ArXiv: 1508.05552]

Measurements

- **Fermion gradient flow**

  ➔ Extension of the YMGF to quark fields

$$\partial_t \chi(t,x) = \sum_\mu \nabla^*_\mu \nabla_\mu \chi(t,x) \qquad \chi(0,x) = \psi(x)$$

  ➔ Kernel of the heat equation

$$\partial_t K(t,x;s,y) = \sum_\mu \nabla_\mu \nabla_\mu K(t,x;s,y) \quad \text{if} \quad t \geq s \quad \lim_{t\to s}, K(t,x;s,y) = \delta(x-y)$$

  - Adjoint flow equation $\rightarrow$ "backflow" of the fermion fields

[M. Lüscher, ArXiv: 1302.5246]

Measurements

- **Fermion propagators**

  - ➔ O(a) Improved Wilson Fermions with twisted mass.

$$D_w = \frac{1}{2}\gamma_\mu\left(\nabla_\mu + \nabla_\mu^*\right) - \frac{1}{2}\nabla_\mu\nabla_\mu^* + c_{sw}\frac{i}{4}F_{\mu\nu}\sigma_{\mu\nu} + m + i\mu\gamma_5 + bndry$$

  - ➔ Conjugate Gradient Solver. Solver efficiency $\Longleftrightarrow$ GPU Memory

$$\langle\overline{\psi}(x)\Gamma_1\psi(x)\overline{\psi}(y)\Gamma_2\psi(y)\rangle \rightarrow D_w(y|x)\psi(x) = \eta(y) \rightarrow \psi(x) = D_w^{-1}(x|y)\eta(y)$$

Point source and stochastic source. Available 16 options for Γ in spin.

Other theories

- **Scalar field observables**

  - ➔ Two Higgs Doublet with SU(2) *Guilherme Catumba's talk, Wed. @ 11:35AM*

    Guilherme Catumba et al. [2210.09855][2312.04178][2407.15422]

  - ➔ Perturbation of action parameters (NSPT) with $\lambda\phi^4$

    Guilherme Catumba et al. [2307.15406][2401.06456]

    https://igit.ific.uv.es/alramos/lambdaphi4.jl

Tests on the code

- **Consistency checks inside the code** (@ ./latticegpu.jl/test/)
  - ➔ Adaptive step size consistency check for Gauge flow

    Fixed step size $\Longleftrightarrow$ Adaptive step size

  - ➔ Free propagators for different BC

$$\sum_m D_W^{-1}(n|m)e^{iapm} = \left(m\mathbb{1} + ia^{-1}\sum_\mu \gamma_\mu sin(p_\mu a) + a^{-1}\sum_\mu (1 - cos(p_\mu a))\right)^{-1} e^{iapn}$$

    + analytic $f_P(x_0)$, $f_A(x_0)$ for SF in [Lüscher and Weisz, hep-lat/9606106]

  - ➔ Free theory analytic solution for Fermion flow

Tests on the code

- Consistency checks inside the code
- **Reproduced numerical results:**
  - ➔ Plaquette values against openQCD https://luscher.web.cern.ch/luscher/openQCD
  - ➔ Quenched Pion mass O. Haan et al. , Phys. Lett. B 190 (1987), 147-150
  - ➔ Quenched M_π and F_π with twisted mass.

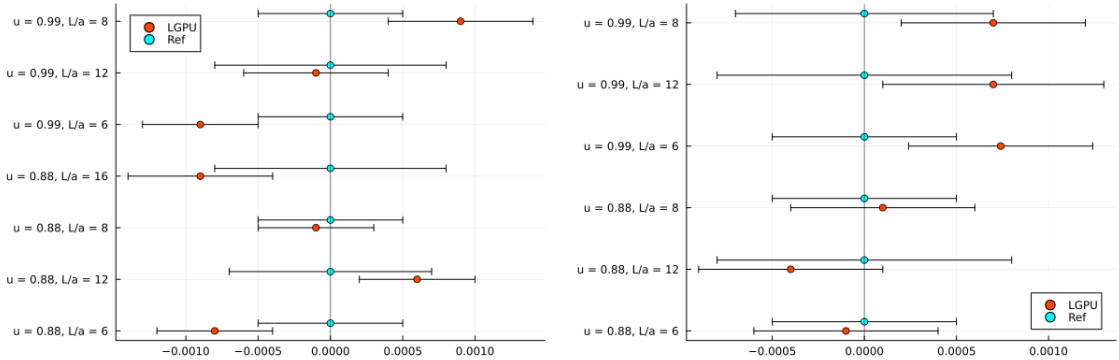
    K. Jansen et al. [χLF], Phys. Lett. B 586 (2004), 432-438
  - ➔ Quenched mass renormalisation Z_P

    M. Guagnelli et al. [ALPHA], JHEP 05 (2004), 001
  - ➔ Chiral condensate and vacuum-to-pion matrix element with fermion flow in CLS configurations*.

    M. Lüscher, JHEP 04 (2013), 123

* Preliminary

Z_P results comparison



Comparison of the results for Z_P in the unimproved (left) and improved (right) case from M. Guagnelli et al. [ALPHA], JHEP 05 (2004), 001, [hep-lat/0402022].

Performance and limitations

- **YMGF timing** : Integration of both Wilson and Zeuthen flow (V = 24^4)

  $CPU \sim 5.95$ h (1 core,Intel Xeon IvyBridge) $GPU \sim 4.35$ mins (Nvidia A100)

  $\Longrightarrow 1 \quad GPU \sim 80 \quad$ CPU cores

- **sfcf timing** : Time it takes to apply Dirac operator ($V = 32^4$)

  $CPU \sim 1.4$ s (1 core, Intel Xeon E5540) $GPU \sim 0.018$ s (Nvidia A100)

  $\Longrightarrow 1 \quad GPU \sim 75 \quad$ CPU cores

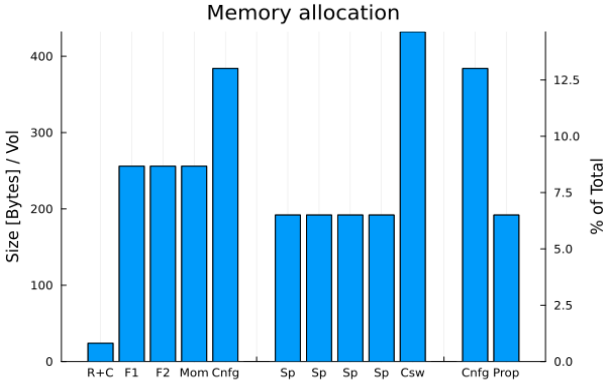
I. Campos *et al.* [ALPHA], Eur. Phys. J. C 78 (2018) no.5, 387, [1802.05243] .

1404 cnfgs. $u_{GF} \sim 5.9$ 12 noise sources. 1GPU $\sim$ 80CPUs.

➔ 512 CPU cores $\rightarrow$ $\sim$ 3 days (oqcd v1.6)

➔ 20 GPUs $\rightarrow$ $\sim$ 23 hours (LGPU)

Performance and limitations



Size	Memory	A100 (40GB)
32^4	1.86 GB	✓
64^4	29.8 GB	✓
68^4	37.97 GB	✓
72^4	47.73 GB	✗
$32^3 \times 64$	3.72 GB	✓
$48^3 \times 96$	18.86 GB	✓
$56^3 \times 112$	34.93 GB	✓
$64^3 \times 128$	59.59 GB	✗
$48^3 \times 144$	28.28 GB	✓
$56^3 \times 168$	52.40 GB	✗
$64^3 \times 192$	89.39 GB	✗

Combined -> 2.95 KB / Vol. | Gauge -> 1.56 KB / Vol. | *Dirac* -> 1.78 KB / Vol.

Production code examples

```julia
# HMC and Flow Measurements
using LatticeGPU

lp = SpaceParm{4}((16,16,16,16),(4,4,4,4),BC_PERIODIC,(0,0,0,0,0,0))
gp = GaugeParm{Float64}(SU3{Float64},6.0,5/3,(1.0,1.0),(0.0,0.0),lp.iL)
intsch = omf4(Float64,0.01,50)
wflw = wfl_rk3(Float64, 0.01, 1.0E-7)
ymws = YMworkspace(SU3,Float64,lp);
U = vector_field(SU3{Float64},lp);

Nmc = 2
Nth = 500
Tflow = 2.0
fill!(U,one(SU3{Float64}));
U0_CPU = Array(U);

for _ in 1:Nth
    HMC!(U,intsch,lp,gp,ymws)
end
for i in 1:Nmc
    dh,acc = HMC!(U,intsch,lp,gp,ymws)
    U0_CPU .= Array(U)

    println("\n### MC step n",i," ###")
    println("Delta H : ",dh)
    println("Accepted/rejected : ", acc)
    println("Plaquette :  ",                plaquette(U, lp, gp, ymws))
    println("Energy density plaquette: ",Eoft_plaq(U, gp, lp, ymws))
    println("Energy density clover: ",   Eoft_clover(U, gp, lp, ymws))
    println("Qtop : ",                     Qtop(U, gp, lp, ymws))

    ns, eps = flw_adapt(U, wflw, Tflow, gp, lp, ymws)
    println("### Flow ###")
    println("Plaquette : ",                plaquette(U, lp, gp, ymws))
    println("Energy density plaquette: ",Eoft_plaq(U, gp, lp, ymws))
    println("Energy density clover: ",   Eoft_clover(U, gp, lp, ymws))
    println("Qtop : ",                     Qtop(U, gp, lp, ymws))

    copyto!(U,U0_CPU)
    save_cnfg("./runname_cnfg_n"*string(i), U, lp, gp)
end
```

HMC and Flow

- Parameters and fields

```julia
using LatticeGPU
lp = SpaceParm{4}((16,16,16,16),(4,4,4,4),BC_PERIODIC,(0,0,0,0,0,0))
gp =
GaugeParm{Float64}(SU3{Float64},6.0,5/3,(1.0,1.0),(0.0,0.0),lp.iL)
intsch = omf4(Float64,0.01,50)
ymws = YMworkspace(SU3,Float64,lp);
U = vector_field(SU3Float64,lp);
```

- HMC step

```julia
dh,acc = HMC!(U,intsch,lp,gp,ymws)
...
println("Plaquette :  ", plaquette(U, lp, gp, ymws))
```

- YM Flow

```julia
ns, eps = flw_adapt(U, wflw, Tflow, gp, lp, ymws)
...
println("Energy density clover:  ", Eoft_clover(U, gp, lp, ymws))
```

- Save config

Production code examples

```julia
# Propagator
using LatticeGPU

Tsrc = 1
cnfg_names = ["./runname_cnfg_n1","./runname_cnfg_n2"]
lp = SpaceParm{4}((16,16,16,16),(4,4,4,4),BC_PERIODIC,(0,0,0,0,0,0))
gp = GaugeParm{Float64}(SU3{Float64},6.0,5/3,(1.0,1.0),(0.0,0.0),lp.iL)
dpar = DiracParam{Float64}(SU3fund,0.2,1.0,(1.0,1.0,1.0,1.0),0.0,1.0)
dws = DiracWorkspace(SU3fund,Float64,lp);
psi = scalar_field(Spinor{4,SU3fund{Float64}},lp)

fill!(psi,zero(eltype(scalar_field(Spinor{4,SU3fund{Float64}},lp))))
for i in 1:length(cnfg_names)

    U = read_cnfg(cnfg_names[i]);
    Csw!(dws,U,gp,lp)
    niter = propagator!(psi, U, dpar, dws, lp, 10000, 1.0e-13, Tsrc)

    println("\n### Cnfg name : ",cnfg_names[i]," ###")
    println("Inversion converged in ",niter," iterations with random source in t=", Tsrc)

    pp_corr = zeros(lp.iL[4])
    for t in 1:lp.iL[4] for a in 1:lp.iL[1] for b in 1:lp.iL[2] for c in 1:lp.iL[3]
        pp_corr[t] += norm2(psi[(point_index(CartesianIndex{lp.ndim}((a,b,c,t)),lp))...])
    end end end end
    println("\nEstimation for the pp correlator:")
    for t in 1:lp.iL[4]
        println(pp_corr[t]./prod(lp.iL[1:3]))
    end

    ap_corr = zeros(ComplexF64,lp.iL[4])
    for t in 1:lp.iL[4] for a in 1:lp.iL[1] for b in 1:lp.iL[2] for c in 1:lp.iL[3]
        ap_corr[t] += dot(psi[(point_index(CartesianIndex{lp.ndim}((a,b,c,t)),lp))...],
            dmul(Gamma{4},psi[(point_index(CartesianIndex{lp.ndim}((a,b,c,t)),lp))...]))
    end end end end
    println("\nEstimation for the ap correlator:")
    for t in 1:lp.iL[4]
        println(ap_corr[t]./prod(lp.iL[1:3]))
    end
end
```

Propagator

- Parameters and fields

```julia
using LatticeGPU
...
dpar = DiracParam{Float64}(SU3fund,
          0.2,1.0,(1.0,1.0,1.0,1.0),0.0,1.0)
dws = DiracWorkspace(SU3fund,Float64,lp);
psi = scalar_field(Spinor{4,SU3fund{Float64}},lp)
```

- Propagator computation

```julia
U = read_cnfg(cnfg_names[i]);
Csw!(dws,U,gp,lp)
niter = propagator!(psi,U,dpar,dws,lp,1000,1.0e-13,Tsrc)
```

- Contractions

```julia
 pp_corr[t]+=
norm2(psi[(point_index(CartesianIndex{lp.ndim}((a,b,c,t)),lp))...])
```

Production code examples

- **sfcf.jl** : Schrödinger functional correlation functions.

```
$ julia sfcf.jl -i sfcf.in -c cnfg
```
$\rightarrow \quad f_P(x_0), f_A(x_0), f_1, \ldots$

Available @ **https://igit.ific.uv.es/fernando.p.csic.es/sfcf.jl**

- **ferflow.jl** : Fermionic (and YM) gradient flow measurements.

```
$ julia main.jl -i ./ferflow.in -c 'config_list' -G 1
```

Available @ **https://gitlab.ift.uam-csic.es/Fernando.P/ferflow.jl**

→ Detailed .log + .bdio (*http://bdio.org/*) files

Conclusions

- Good balance between being **efficient** and **fast-to-develop**.
- It is a very **flexible** code, providing a very good testing enviroment for new ideas.
- **Easy** to use without GPU technical knowledge.
- Expected future developments: TWBC for fermions, GPU-parallelization, dynamical fermions...
- Ready for fermion flow studies.

Thank you for your attention!

Backup

Backup: Fermion Flow Numerical Tests

$$P_t^{rs}(x) = \overline{\chi}_r(t,x)\gamma_5\chi_s(t,x) \quad \rightarrow \quad \int d^3x \langle P_R^{ud}(x) P_{R,t}^{du}(0)\rangle = -\frac{G_\pi G_{\pi,t}}{M_\pi} e^{-M_\pi x_0}\left(1 + e^{-\Delta E x_0}\right) =$$

$$= -\sum_{v,w} \langle tr\{[K(t,y;o,v)S(v,x)_{dd}]^\dagger K(t,y;o,w)S(w,x)_{uu}\}\rangle$$

$$\Sigma_t^{rr} = -\langle \overline{\chi}_r(t,x)\chi_r(t,x)\rangle = \sum_{v,w} \langle tr\left\{K(t,x;0,v)(S(v,w)_{rr} - c_{fl}\delta_{vw})K(t,x;0,w)^\dagger\right\}\rangle$$

➔ Preliminary Results:

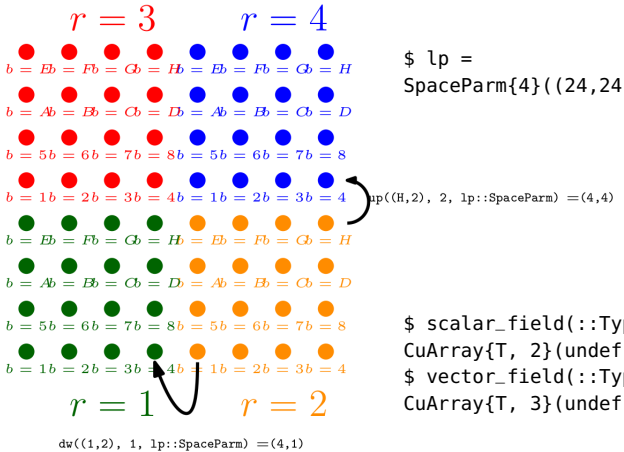
	M_π	F_π	$G_\pi^{\overline{MS}}(2GeV)$	$\sqrt{8t_1}$	$\Sigma(t_1)/G^{ud}(t_1)$	$\sqrt{8t_2}$	$\Sigma(t_2)/G^{ud}(t_2)$
LGPU	292(9) MeV	79(8) MeV	$(489(11)MeV)^2$	0.43 fm	98(4) MeV	0.53 fm	90(3) MeV
[1]	203 MeV	93 MeV *	$(513(6)MeV)^2$	0.4 fm	93(1) MeV	0.5 fm	91(2) MeV

[1] M. Lüscher, *Chiral symmetry and the Yang–Mills gradient flow*, JHEP 04 (2013), 123, [ArXiv: 1302.5246].

* physical value

Backup: GPU Spacetime structure

$(b, r) \equiv$ Lattice point in D dimensions




```
$ lp =
SpaceParm{4}((24,24,24,24),(4,4,4,4),BC_PERIODIC,TWB_Nmunu);
```

```
$ scalar_field(::Type{T}, lp::SpaceParm) where {T} =
CuArray{T, 2}(undef, lp.bsz, lp.rsz)
$ vector_field(::Type{T}, lp::SpaceParm) where {T} =
CuArray{T, 3}(undef, lp.bsz, lp.ndim, lp.rsz)
```

Backup: Free Fermion flow solution

$$\partial_t \chi(t,n) = D_\mu D_\mu \chi(t,n) \qquad \chi(0,n) = \psi(n) \qquad U_\mu = \mathbb{1}$$

$$\langle \chi(t,m)\overline{\psi}(n)\rangle = \frac{1}{V}\sum_{p_\mu} e^{-4t/a^2 \sum_\mu sin^2(ap_\mu/2)} D^{-1}(p) e^{iap(n-m)}$$

$$\sum_{n,n'} K(t,m;0,n) D_w^{-1}(n|n') e^{iapn'} = \underbrace{D_w^{-1}(p) e^{-4t/a^2 \sum_\mu sin^2(ap_\mu/2)} e^{iapm}}_{\text{Solution of the heat equation [ArXiv: 1207.2096]}}$$

M. Lüscher, JHEP 04 (2013), 123, [ArXiv: 1302.5246].

Backup: Some absolute timings

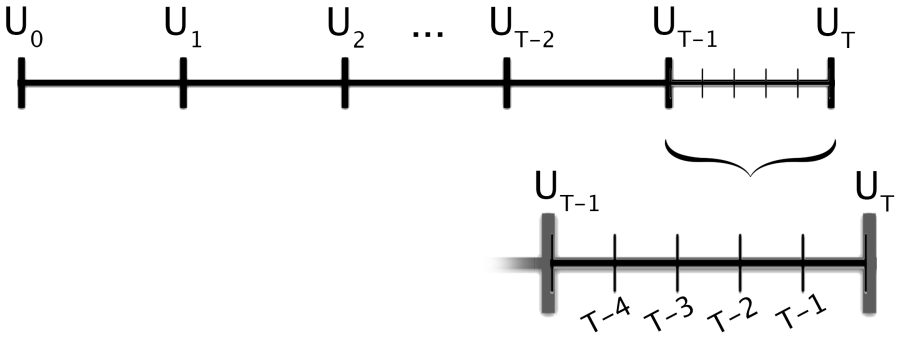
Times for a 16^4 Lattice with Periodic BC in a Nvidia A100.

	HMC		YM Flow		Dirac
Leapfrog	4.08	Euler	1.93	Unimproved	7.71
Omelyan o2	7.78	RK2	3.77	Improved	10.90
Omelyan o4	19.10	RK3	5.77		

All times are in seconds.

- HMC : Time to perform 100 MC step (accepted or not) divided in 20 integration steps.
- YM flow : Time to perform 1000 integration steps of Wilson flow.
- Dirac : Time to apply the Dirac operator 1000 times.

Backup: Hierarchical scheme



Backup: CG snippet

```julia
function CG!(si, U, A, dpar::DiracParam, lp::SpaceParm, dws::DiracWorkspace{T}, maxiter::Int64 = 10, tol=1.0) where {

    dws.sr .= si
    dws.sp .= si
    norm = CUDA.mapreduce(x -> norm2(x), +, si)
    fill!(si,zero(eltype(si)))
    err = 0.0

    tol = tol * norm

    iterations = 0
    sumf = scalar_field(Complex{T}, lp)

    niter = 0
    for i in 1:maxiter
        A(dws.sAp, U, dws.sp, dpar, dws, lp)

        prod = field_dot(dws.sp,dws.sAp,sumf,lp)

        alpha = norm/prod

        si     .= si     .+ alpha .* dws.sp
        dws.sr .= dws.sr .- alpha .* dws.sAp

        err = CUDA.mapreduce(x -> norm2(x), +, dws.sr)

        if err < tol
        niter = i
            break
        end

        beta = err/norm
        dws.sp .= dws.sr .+ beta .* dws.sp

        norm = err;
    end

    if err > tol
    error("CG! not converged after $maxiter iterations (Residuals: $err)")
    end

    return niter
end
```