東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

MLPhYs 学術変革領域研究(A) 学習物理学の創成
Foundation of "Machine Learning Physics"

Automatic Differentiation for Lattice Gauge Theory

Information Technology Center /
Department of Advanced Materials Science (AMS),
Graduate School of Frontier Sciences
The University of Tokyo

UTokyo 東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

GRADUATE SCHOOL OF FRONTIER SCIENCES
THE UNIVERSITY OF TOKYO

Yuki Nagai

AD in QCD

"Lattice Gauge Theory via LLVM-Level Automatic Differentiation"

Y. Nagai, A. Tomiya and H. Ohno, arXiv:2602.20516

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

About me

Yuki Nagai



Born in Hokkaido, the northern island in Japan

- 2005/03 B. Eng. at Department of Applied Physics, Hokkaido University
- 2010/03 Ph.D at Department of Physics, the University of Tokyo
- 2010/04-2024/01 Scientist -> Senior Scientist, Japan Atomic Energy Agency
- 2024/02 Associate Professor in The University of Tokyo

Research interests

Condensed matter theory
Superconductivity,
Material science
Machine-learning and Physics

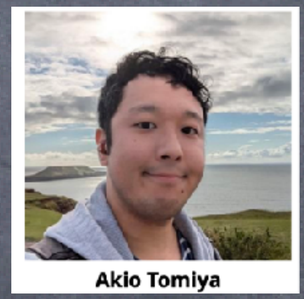
I am a condensed matter theorist



Collaborators

JuliaQCD project

LatticeQCD codes written in Julia



Akio Tomiya (Tokyo Woman's Christian Univ.)



Hiroshi Ohno (Univ. of Tsukuba)

High-energy theorists!

Outline

- Introduction: JuliaQCD project
- Automatic Differentiation
- How do we perform automatic differentiation on a Lie-group-valued gauge field?
- How can we efficiently differentiate in-place code?
- Benchmarks
- Summary

Introduction: JuliaQCD project

Public codes for LQCD

Name (Historical order, old->new)	Language	URL	Paper
MILC code	C/C++	https://github.com/milc-qcd/milc_qcd	https://inspirehep.net/literature/321665
Lattice Tool kit	Fortran	https://github.com/tsuchim/Lattice-Tool-Kit/	NPB Proc.Suppl. 106 (2002) 1037-1039
CPS (Columbia physics system)	C/C++/Assemblar	https://github.com/RBC-UKQCD/CPS	https://arxiv.org/abs/hep-lat/0306023
Chroma	C++	https://github.com/JeffersonLab/chroma	arXiv:hep-lat/0409003
QUDA(backend)	C++/CUDA	https://github.com/lattice/quda	arXiv:1011.0024
Bridge++	C++/GPU	https://bridge.kek.jp/Lattice-code/index_e.html	J.Phys.Conf.Ser. 523 (2014) 012046
Grid	C++/GPU	https://github.com/paboyle/Grid	arXiv:1512.03487
JuliaQCD	Julia	https://github.com/juliaqcd/	https://arxiv.org/abs/2409.03030
SimuLATeQCD	C++/CUDA	https://github.com/LatticeQCD/SIMULATeQCD	https://arxiv.org/abs/2306.01098

and more

We made LQCD code with Julia

1. Benchmark test for *Julia itself* since LQCD is a hardest problem
2. Easy to use. **It can work on PC and supercomputers**
3. Minimize time/effort for “code development” + “execution”

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

JuliaQCD Project

JuliaQCD

Current status

LatticeMatrices.jl v1.2.2: backends for gaugefields and pseudo-fermion fields
for CPU threads, multi-CPUs, GPU, multi-GPUs

Multi-GPU calculation works
on Wisteria (JP), Miyabi (JP) and Polaris (USA) with NVIDIAGPUs
Pegasus (JP) with AMDGPUs, and Aurora (USA) with IntelGPUs
with use of JACC.jl, one same code works on different GPUs

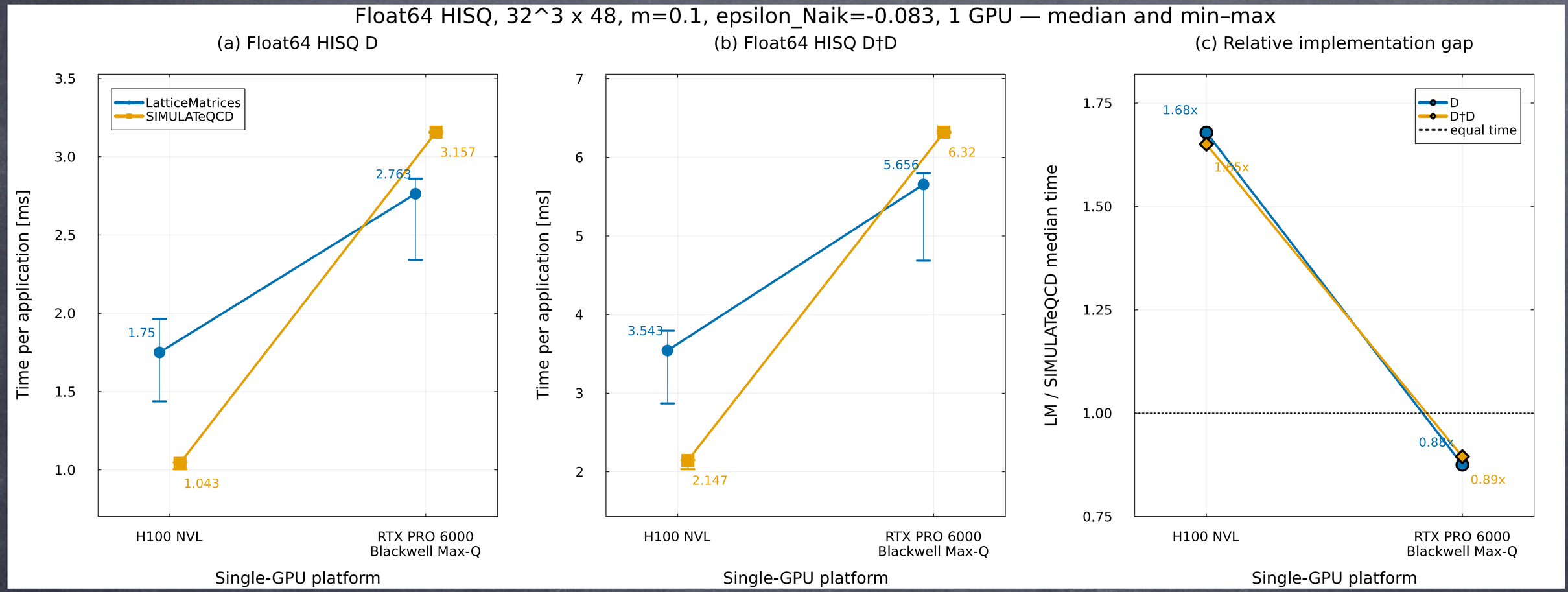
Gaugefields.jl v1.1.4: SU(N) gaugefields, HMC, heatbath, gaugefixing etc.

LatticeDiracOperators.jl v1.1.1: Wilson, Wilson-clover, staggered, HISQ, Shamir domain-wall, Möbius domain-wall, and generalized domain-wall operators

QCDMeasurements.jl v1.0.0: several measurements like Topological charge, Pion, Meson, etc

LatticeQCD.jl v2.0.1 *Wrapper* for LatticeDiracOperators.jl, QCDMeasurements.jl & Gaugefields.jl

JuliaQCD vs. SIMULATeQCD: HISQ benchmark



C++ GPU code

L. Mazur et al., "SIMULATeQCD: A simple multi-GPU lattice code for QCD calculations"
Computer Physics Communications 300 (2024) 109164

Comparable speed on GPUs!

Automatic Differentiation

Automatic Differentiation (AD)

In LatticeQCD

Actions

$$S[U,\phi] = S_g[U] + S_f[U,\phi]$$

Probabilities

$$P[U,\phi] = e^{-S[U,\phi]}/Z$$

In HMC, we need forces

$$H = \frac{1}{2}\langle P, P\rangle + S$$

$$\dot{U} = PU$$

$$\dot{P} = -\nabla_{\mathrm{Lie}} S$$

$$U \in \mathrm{SU}(N)$$

$$P \in \mathfrak{su}(N)$$

In ML, we need parameter derivatives

$$\frac{\partial S_{\vec{\theta}}[U,\phi]}{\partial \vec{\theta}} \quad \text{or} \quad \frac{\partial P_{\vec{\theta}}[U,\phi]}{\partial \vec{\theta}}$$

Automatic differentiation techniques

Python: PyTorch, JAX, TensorFlow etc

Julia: Flux.jl, Zygote.jl, Enzyme.jl etc

We can get gradient of functions

Can we use these in LatticeQCD?

Problems of automatic differentiation in LatticeQCD

Two obstacles in lattice QCD

1 Complex valued variables on 4D space-time lattice

$$S[U, \phi] = S_g[U] + S_f[U, \phi]$$

The gauge field takes values in a Lie group $U \in \mathrm{SU}(N_c)$

How do we perform automatic differentiation on a Lie-group-valued gauge field?

2 Many in-place operations in LatticeQCD codes

U is repeatedly updated in place

conventional AD techniques

We need a "graph" of calculations



In-place operations break a "graph"

AD can not be done without copies

How can we efficiently differentiate in-place code?

How do we perform automatic differentiation on a Lie-group-valued gauge field?

Problems of automatic differentiation in LatticeQCD

Two obstacles in lattice QCD

1 Complex valued variables on 4D space-time lattice

$$S[U,\phi] = S_g[U] + S_f[U,\phi]$$

The gauge field takes values in a Lie group $U \in \mathrm{SU}(N_c)$

How do we perform automatic differentiation on a Lie-group-valued gauge field?

2 Many in-place operations in LatticeQCD codes

U is repeatedly updated in place

conventional AD techniques

We need a "graph" of calculations



In-place operations break a "graph"

AD can not be done without copies

How can we efficiently differentiate in-place code?

Hybrid Monte Carlo

In lattice QCD simulation, we have to do HMC (Hamiltonian/Hybrid Monte Carlo)

Molecular dynamics for gauge fields

$$U(\tau + \Delta\tau) = e^{\Delta\tau P(\tau)} U(\tau)$$
$$P(\tau + \Delta\tau) = P(\tau) - \Delta\tau F(\tau)$$

"Hamiltonian"

$$H = \frac{1}{2}\langle P, P\rangle + S$$
$$\dot{U} = PU$$
$$\dot{P} = -\nabla_{\mathrm{Lie}} S$$
$$S(U) = S_g(U) + S_f(U)$$

S is usually complicated

Variables

HMC(p,q)	SU(N) HMC(U,P)
q	$U \in \mathrm{SU}(N)$
p	$P \in \mathfrak{su}(N)$
$\dot{q} = p$	$\dot{U} = PU$
$V(q)$	$S(U)$
∇V	$\nabla_{\mathrm{Lie}} S \in \mathfrak{su}(N)$
$\dot{p} = -\nabla V$	$\dot{P} = -\nabla_{\mathrm{Lie}} S$

How to calculate $\nabla_{\mathrm{Lie}} S$?

Automatic Differentiation

There are many packages like PyTorch, JAX in Python
or Zygote.jl, Enzyme.jl in Julia

What they can do:

$$f : \mathbb{R}^n \to \mathbb{R}$$

With AD, we can obtain $\dfrac{\partial f}{\partial x_i}$

$$x = (x_1, x_2, \cdots, x_n)$$

They treat **real** variables

definition of gradient ∇f

$$df(dx) = \langle \nabla f, dx \rangle$$

In SU(N) HMC

$U \in \mathrm{SU}(N)$ Lie group　　$P \in \mathfrak{su}(N)$ Lie algebra

U: complex NxN matrix
with $U^+U = 1$, $\det U = 1$

P: complex NxN matrix
with $P^+ = -P$, $\mathrm{tr}(P)=0$

definition of gradient $\nabla_{\mathrm{Lie}} S \in \mathfrak{su}(N)$

$$dS_U(PU) = \langle \nabla_{\mathrm{Lie}} S, P \rangle$$

How to calculate $\nabla_{\mathrm{Lie}} S$?

Automatic differentiation for Lie group

What they can do:

$$f : \mathbb{R}^n \to \mathbb{R}$$

With AD, we can obtain $\dfrac{\partial f}{\partial x_i}$

$$x = (x_1, x_2, \cdots, x_n)$$

They treat **real** variables

We can only know $\dfrac{\partial S}{\partial a_{ij}}, \dfrac{\partial S}{\partial b_{ij}}$

N x N Complex variables are defined as

$$U_{ij} = a_{ij} + ib_{ij} \qquad U \to \{a_{ij}, b_{ij}\}$$

$$x = (a_{11}, b_{11}, a_{12}, b_{12}, \cdots, a_{NN}, \cdots b_{NN})$$

$2N^2$ variables

By the AD software,
we can obtain $\dfrac{\partial S}{\partial x_i} \to \dfrac{\partial S}{\partial a_{ij}}, \dfrac{\partial S}{\partial b_{ij}}$

How to obtain $dS_U(PU) = \langle \nabla_{\text{Lie}} S, P \rangle$ and $\nabla_{\text{Lie}} S$?

$\langle A, B \rangle \equiv \text{ReTr}\left(A^\dagger B\right)$ **real Frobenius inner product**

Automatic differentiation for Lie group

N x N Complex variables are defined as

$$U_{ij} = a_{ij} + ib_{ij} \quad U \to \{a_{ij}, b_{ij}\}$$

$$\delta U_{ij} = \delta a_{ij} + i\delta b_{ij}$$

We define "derivative" G: $$G_{ij} = \frac{\partial S}{\partial a_{ij}} + i\frac{\partial S}{\partial b_{ij}}$$

If U has no constraint, δS is defined by

$$\delta S = \sum_{ij}\left(\frac{\partial S}{\partial a_{ij}}\delta a_{ij} + \frac{\partial S}{\partial b_{ij}}\delta b_{ij}\right) = \mathrm{Re}\left(\sum_{ij} G^*_{ij}\delta U_{ij}\right) = \mathrm{ReTr}\left(G^\dagger \delta U\right)$$

Now set $U \in \mathrm{SU}(N) \to \delta U = PU, P \in \mathfrak{su}(N)$

$$\delta S(P) = \mathrm{ReTr}\left(G^\dagger PU\right) = \mathrm{ReTr}\left(UG^\dagger P\right) = \langle GU^\dagger, P\rangle$$

$$\langle A, B\rangle \equiv \mathrm{ReTr}\left(A^\dagger B\right)$$

Automatic differentiation for Lie group

N x N Complex variables are defined as

$$U_{ij} = a_{ij} + ib_{ij} \quad U \to \{a_{ij}, b_{ij}\}$$

$$\delta U_{ij} = \delta a_{ij} + i\delta b_{ij}$$

We define "derivative" G: $$G_{ij} = \frac{\partial S}{\partial a_{ij}} + i\frac{\partial S}{\partial b_{ij}}$$

By the AD, we can obtain G

Now set $U \in \mathrm{SU}(N) \to \delta U = PU, P \in \mathfrak{su}(N)$

$$\delta S(P) = \langle GU^\dagger, P \rangle$$

We need

$$dS_U(PU) = \langle \nabla_{\mathrm{Lie}} S, P \rangle$$

$$\nabla_{\mathrm{Lie}} S \in \mathfrak{su}(N)$$

With the use of the relation if $P \in \mathfrak{su}(N)$

$$\langle A, P \rangle = \langle P_{\mathfrak{su}(N)}(A), P \rangle$$

$$A = P_{\mathfrak{su}(N)}(A) + A_\perp \qquad P_{\mathfrak{su}(N)}(A) = \tfrac{1}{2}(A - A^\dagger) - \tfrac{1}{2N}\mathrm{Tr}(A - A^\dagger) I.$$

$$\nabla_{\mathrm{Lie}} S = P_{\mathfrak{su}(N)}(GU^\dagger)$$

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

Automatic differentiation for Lie group

N x N Complex variables are defined as

$$U_{ij} = a_{ij} + ib_{ij} \quad U \to \{a_{ij}, b_{ij}\}$$

$$\delta U_{ij} = \delta a_{ij} + i\delta b_{ij}$$

We define "derivative" G: $G_{ij} = \frac{\partial S}{\partial a_{ij}} + i\frac{\partial S}{\partial b_{ij}}$

By the AD, we can obtain G

$$\nabla_{\text{Lie}} S = P_{\mathfrak{su}(N)}(GU^\dagger)$$

$$P_{\mathfrak{su}(N)}(A) = \frac{1}{2}(A - A^\dagger) - \frac{1}{2N}\,\text{Tr}(A - A^\dagger)\, I$$

$$P_{\mathfrak{su}(N)}(A^\dagger) = -P_{\mathfrak{su}(N)}(A)$$

$$\to \nabla_{\text{Lie}} S = -P_{\mathfrak{su}(N)}(UG^\dagger)$$

$$\dot{U} = PU$$

$$\dot{P} = P_{\mathfrak{su}(N)}(UG^\dagger)$$

We can do the HMC!

How can we efficiently differentiate in-place code?

Problems of automatic differentiation in LatticeQCD

Two obstacles in lattice QCD

1 Complex valued variables on 4D space-time lattice **Solved!**

$$S[U, \phi] = S_g[U] + S_f[U, \phi]$$

The gauge field takes values in a Lie group $U \in \mathrm{SU}(N_c)$

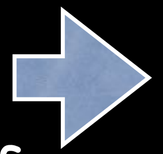
How do we perform automatic differentiation on a Lie-group-valued gauge field?

2 Many in-place operations in LatticeQCD codes

U is repeatedly updated in place

conventional AD techniques

We need a "graph" of calculations



In-place operations break a "graph"

AD can not be done without copies

How can we efficiently differentiate in-place code?

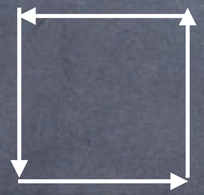
In-place operations

We need temporal arrays for calculation in LatticeQCD

Examples of in-place operations

V[1,2,1,1,1,1] = 2

U[1,2,1,1,1,1] = U[1,2,2,1,1,1,1]+U[1,2,1,2,1,1]

Plaquette: V 

Derivatives are also arrays

$\frac{\partial S}{\partial U}$:N x N matrices on 4D lattice

We do not want to allocate huge memories

Automatic differentiation

Computational graph

x -> y=f(x) -> z=g(y) -> S=h(z)

back propagation

dS/dh -> dS/dz -> dS/dy -> dS/dx

if there is an in-place operation like y=g(y),

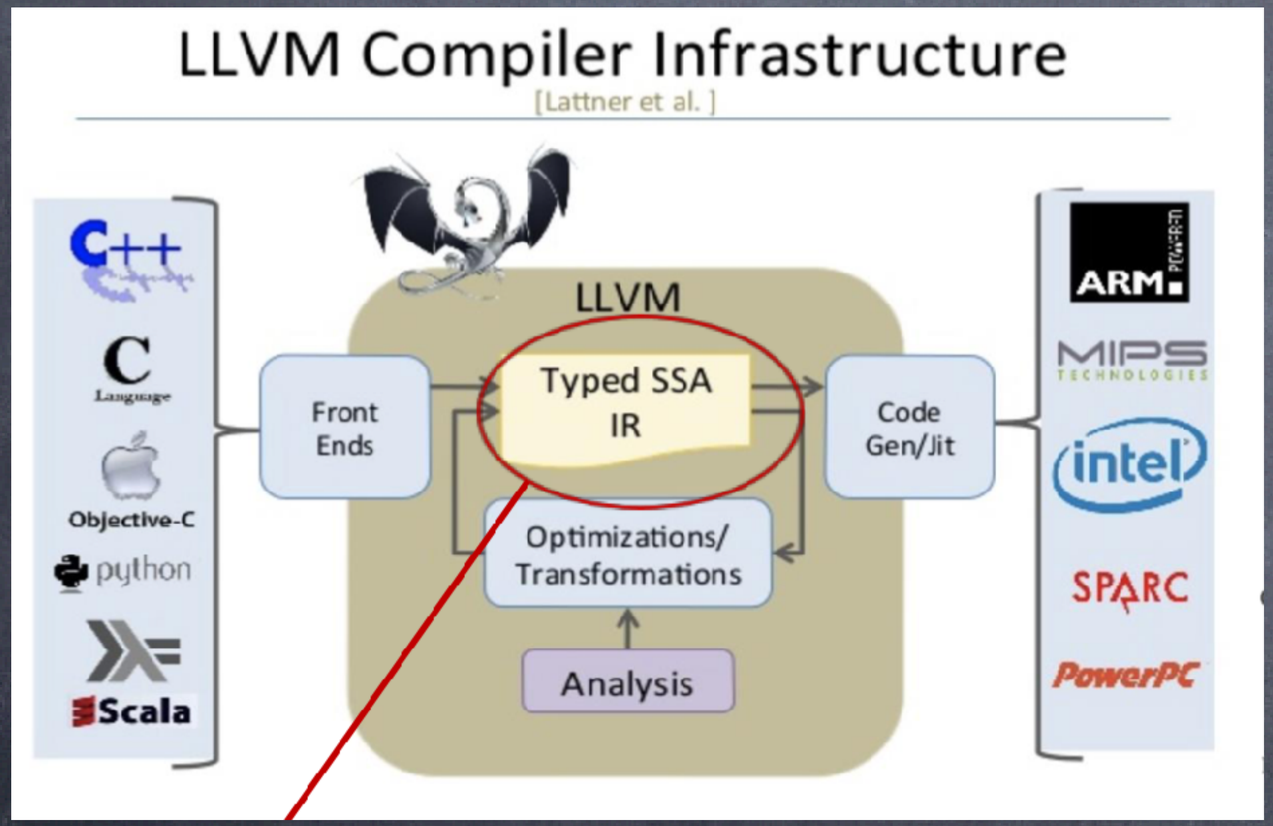
x -> y=f(x) -> y=g(y) -> S=h(y)

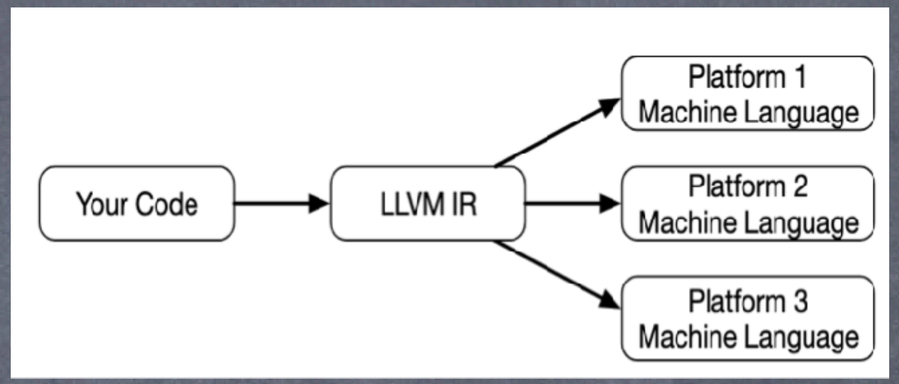
Overwritten values must be recovered or stored for the reverse pass.

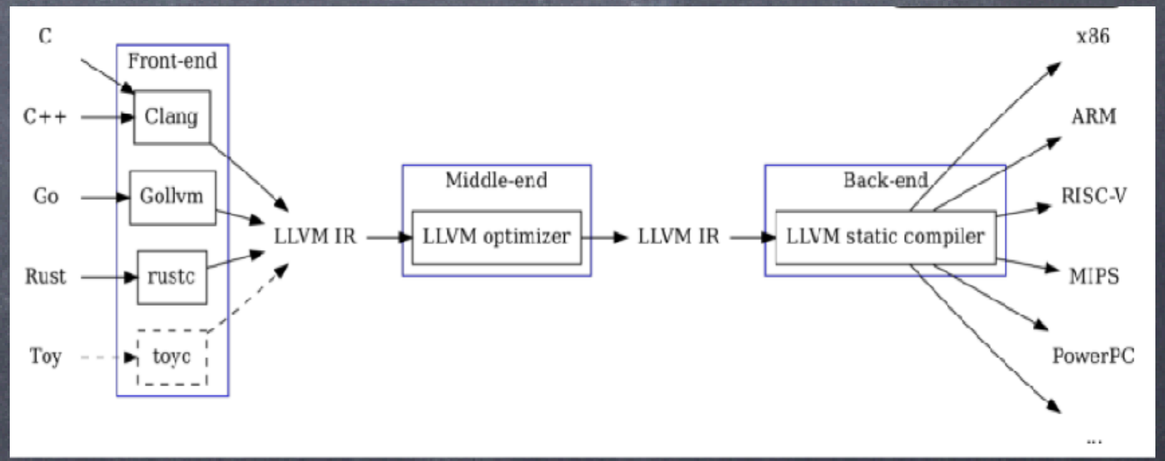
Julia uses LLVM

We focus on that Julia uses the LLVM

LLVM is a modular and reusable compiler infrastructure that provides a modern intermediate representation (IR), powerful optimizations, and code generation for many programming languages.







write once, run anywhere

We can do the AD with the use of the LLVM

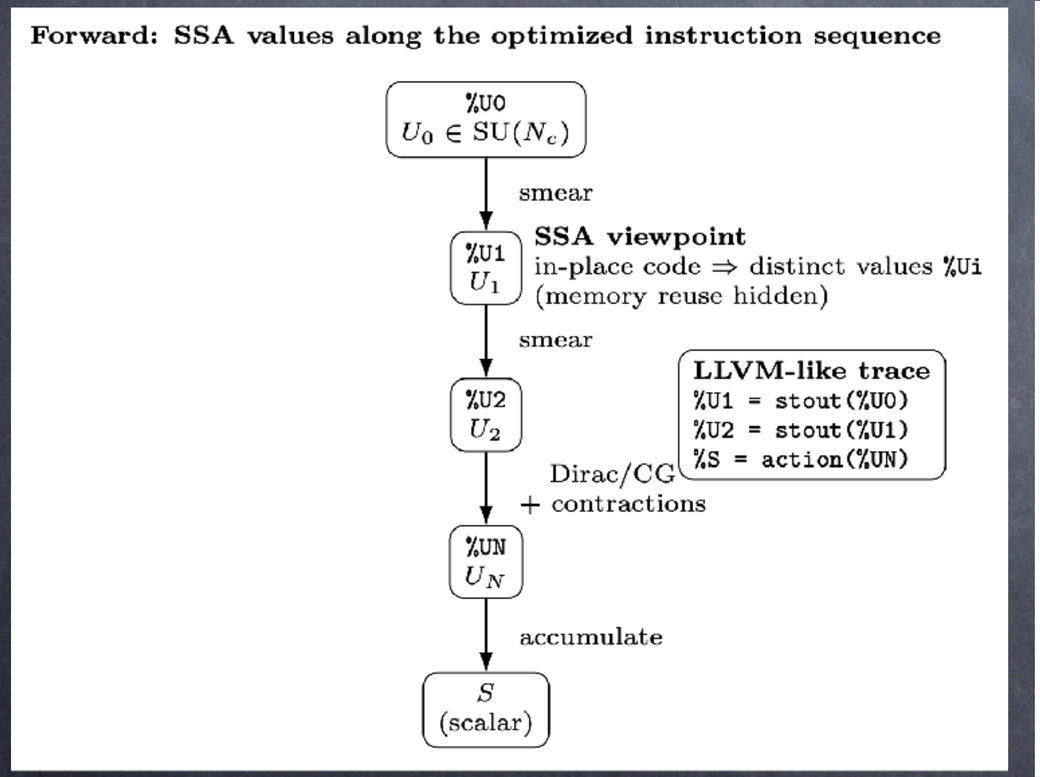
Y. Nagai, A. Tomiya and H. Ohno, arXiv:2602.20516

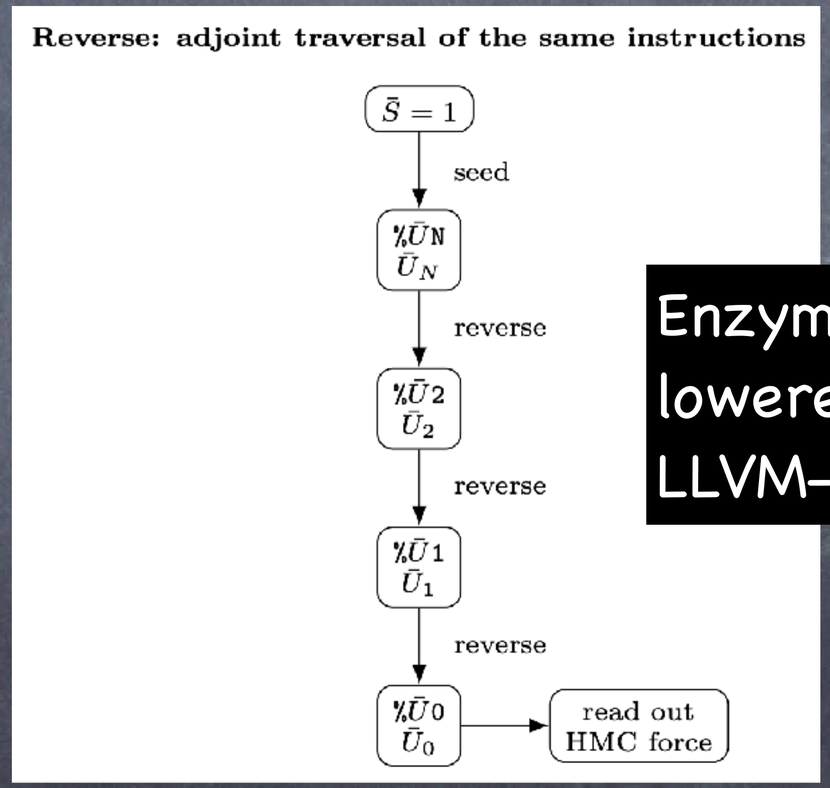
LLVM-level automatic differentiation

LLVM-level automatic differentiation

LLVM IR is expressed in static single assignment (SSA) form: each intermediate value is defined exactly once.

Mutation is explicit at the IR level





Enzyme differentiates the lowered program at the LLVM-IR level

Enzyme.jl: Julia version of Enzyme

Y. Nagai, A. Tomiya and H. Ohno, arXiv:2602.20516

AD and custom adjoints

LatticeMatrices.jl can use multi-core or GPUs via JACC.jl

We want to use multi-core or GPUs to obtain forces

In Enzyme.jl, we can design custom Enzyme rules for reverse AD

For example

primal $A_{ij} = \sum_k B_{ik} C_{kj}$ -> parallel for loop via JACC.jl

adjoint of B $\bar{B}_{ik} \equiv \frac{\partial L}{\partial B_{ik}} = \sum_j \bar{A}_{ij} C_{kj}$ -> parallel for loop via JACC.jl

adjoint of C $\bar{C}_{kj} \equiv \frac{\partial L}{\partial C_{kj}} = \sum_i B_{ik} \bar{A}_{ij}$ -> parallel for loop via JACC.jl

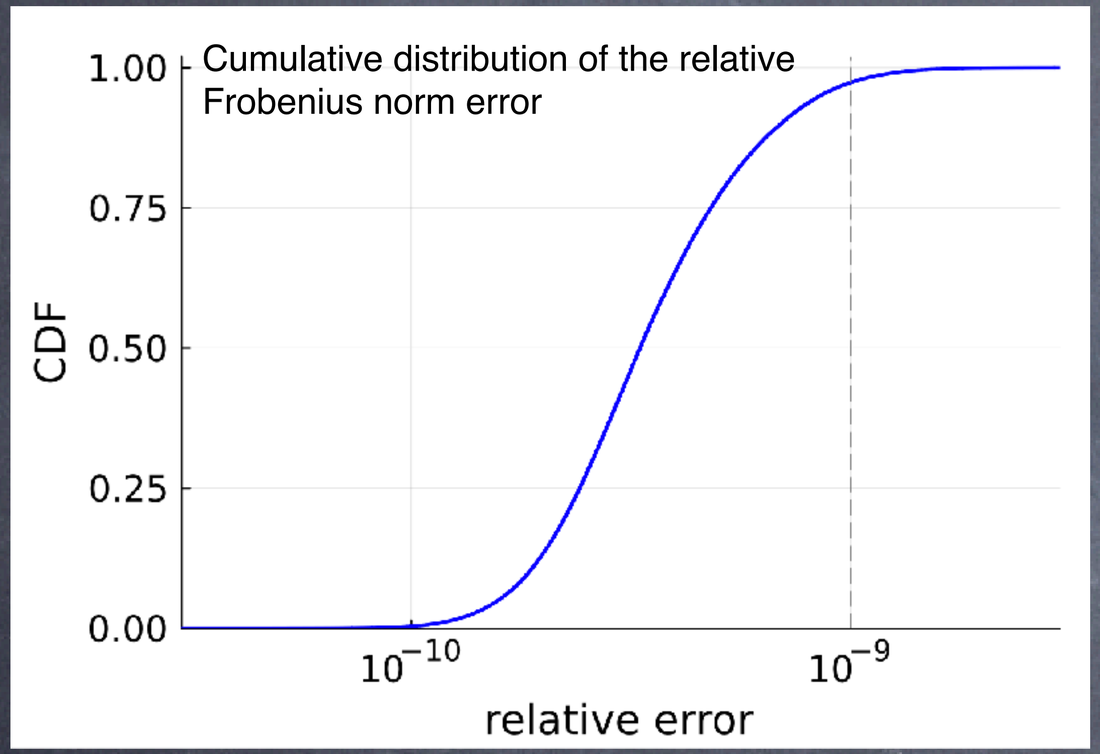
We can use JACC.jl in AD!

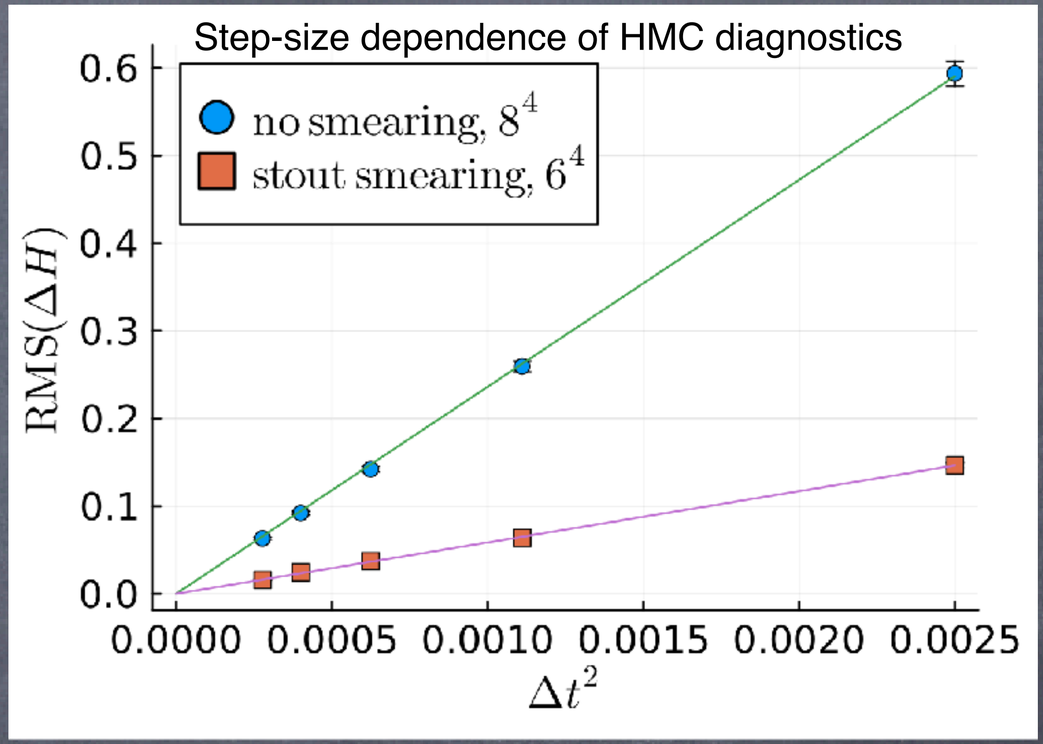
Reverse kernels are parallelized through the same backend abstraction

Benchmarks

Y. Nagai, A. Tomiya and H. Ohno, arXiv:2602.20516

Accuracy





LLVM-level AD is consistent with hand-written implementation

The compiler-generated adjoint faithfully reproduces the required force contributions in this nontrivial setting.

Y. Nagai, A. Tomiya and H. Ohno, arXiv:2602.20516

wall-clock time

TABLE I. Representative wall-clock time (seconds) for a single force evaluation of the Wilson fermion action on a 24^4 lattice. CPU results correspond to single-thread execution. GPU results are obtained on an NVIDIA H100 using JACC.jl.

Implementation	CPU	GPU
Hand-written	195.53	4.61
LLVM-AD	162.50	3.39

Thanks to JACC.jl, we can easily use NVIDIA H100 without changing codes

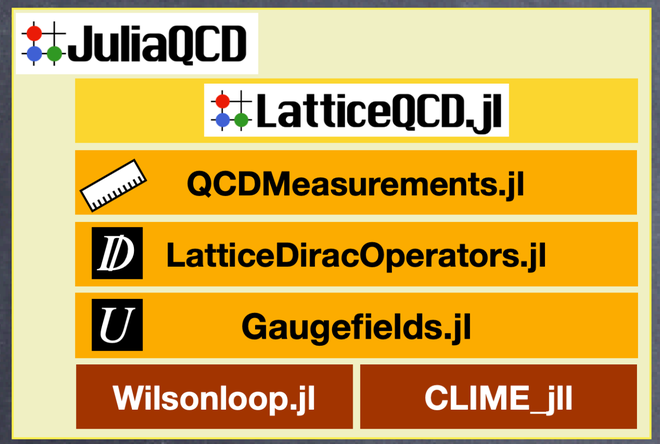
We confirmed that it works with multicore-CPU and AMD GPUs

Summary

Summary

We provide LLVM-level automatic differentiation in QCD

Forces can be obtained from actions with many in-place operations



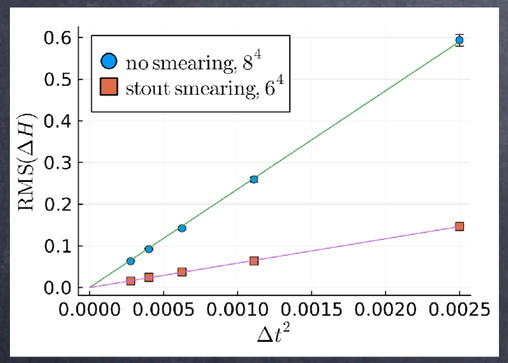
We can do HMC without defining the forces

The forces are automatically derived

Forces for both gauge fields and fermion fields

The parameter derivatives are also supported

All calculations can be done on multi-GPU machines



"Lattice Gauge Theory via LLVM-Level Automatic Differentiation"

Y. Nagai, A. Tomiya and H. Ohno, arXiv:2602.20516

Portable and differentiable lattice QCD is now possible through compiler-level design in Julia