素粒子物理におけるJuliaの活用 ～格子QCDの大規模計算に向けて～

Akio Tomiya (IPUT Osaka, Assistant Prof.)
akio_at_yukawa.kyoto-u.ac.jp

MLPhYs Foundation of "Machine Learning Physics"
Grant-in-Aid for Transformative Research Areas (A)

Outline

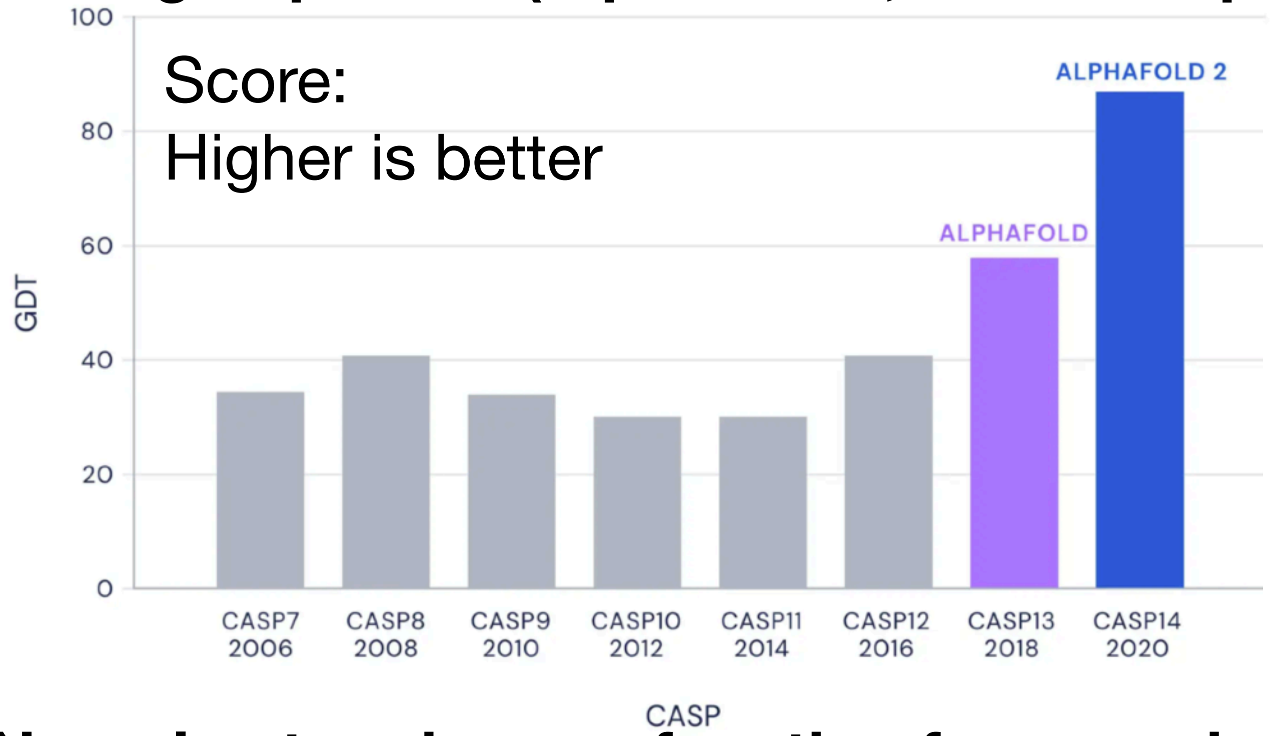
- 素粒子物理学? 格子QCD?
- Julia言語での取り組み
- 何ができれば良いのか

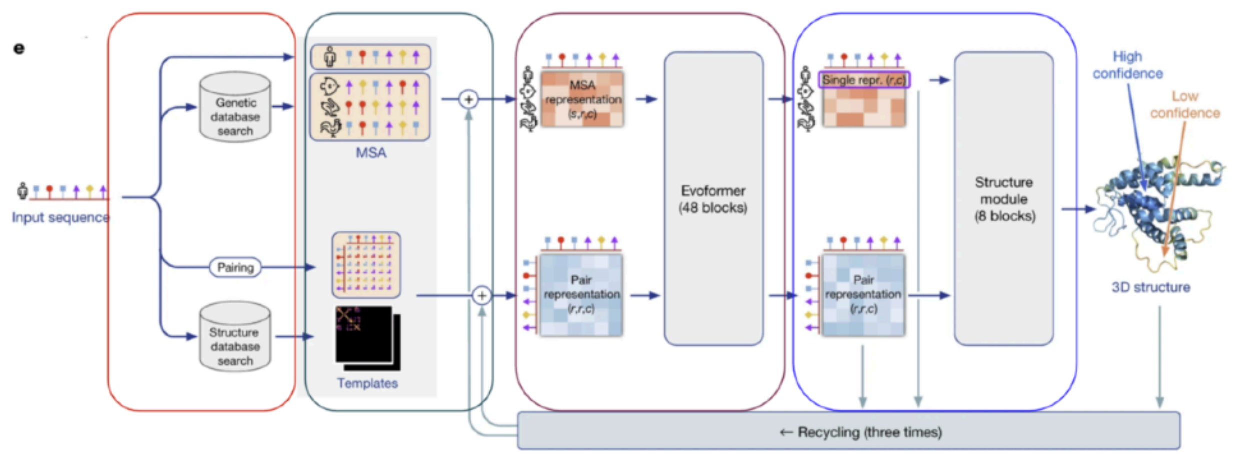
How to treat gauge fields with neural networks?

Applications of neural network in natural science

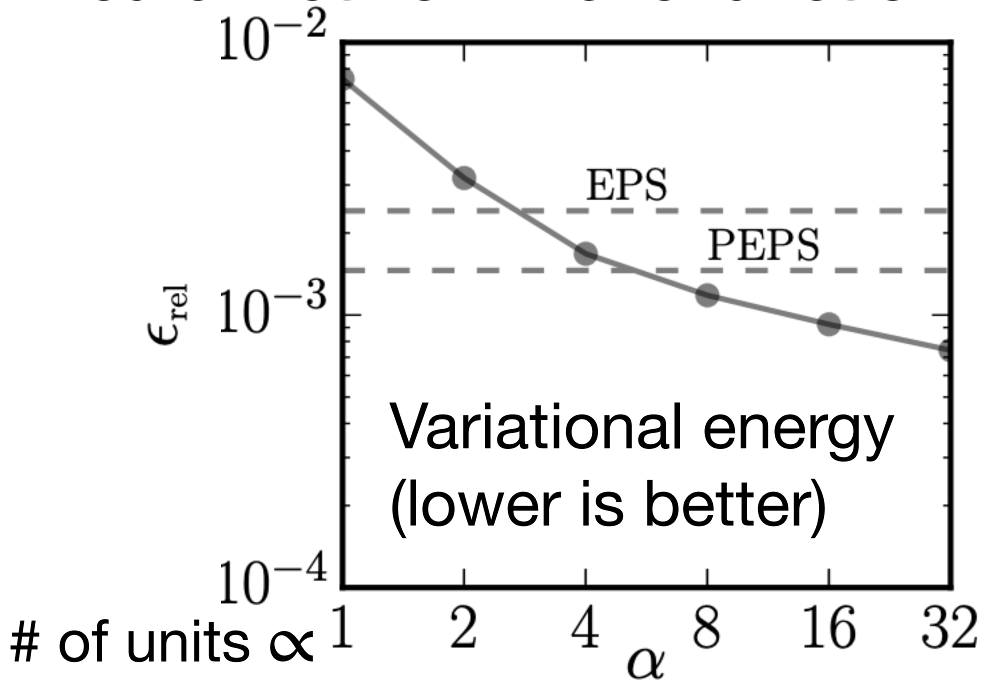
Neural network have been good job

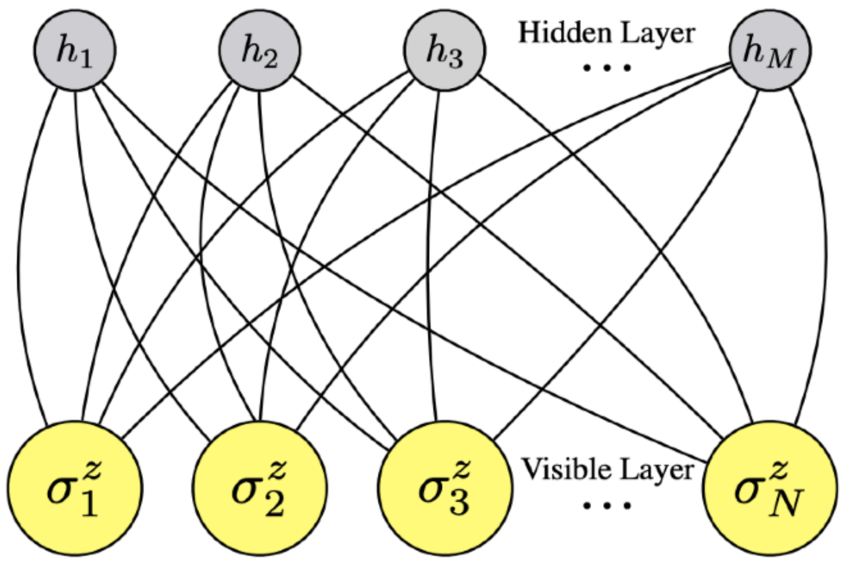
Folding of protein (AlphaFold2, John Jumper+, Nature, 2020+)




e
Input sequence
Genetic database search
Pairing
Structure database search
MSA
Templates
MSA representation (s,r,c)
Pair representation (r,r,c)
Evoformer (48 blocks)
Single repr. (r,c)
Pair representation (r,r,c)
Structure module (8 blocks)
High confidence
Low confidence
3D structure
← Recycling (three times)

Neural network wave function for many body (Carleo Troyer, Science 355, 602 (2017))





Neural net + Expert knowledge → Best performance

What is neural networks?

Affine transformation + element-wise transformation

Layers of neural nets $l = 2,3,\cdots,L,\ \vec{u}^{(1)} = \vec{x}$

$$\begin{cases} \vec{z}^{(l)} = W^{(l)}\vec{u}^{(l-1)} + \vec{b}^{(l)} \\ u_i^{(l)} = \sigma^{(l)}(z_i^{(l)}) \end{cases}$$

Affine transf.

(b=0 called linear transf.)

element-wise (local)

A fully connected neural net

$$f_\theta(\vec{x}) = \sigma^{(3)}(W^{(3)}\sigma^{(2)}(W^{(2)}\vec{x} + \vec{b}^{(2)}) + \vec{b}^{(3)})$$

θ **is a set of parameters:** $w_{ij}^{(l)}, b_i^{(l)}, \cdots$

Neural network = map between vectors and vectors

Physicists terminology: Variational transformations

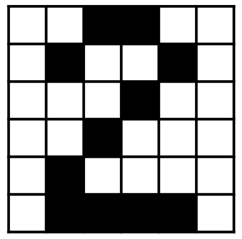
What is the neural networks?

Akio Tomiya

Neural network is a universal approximator

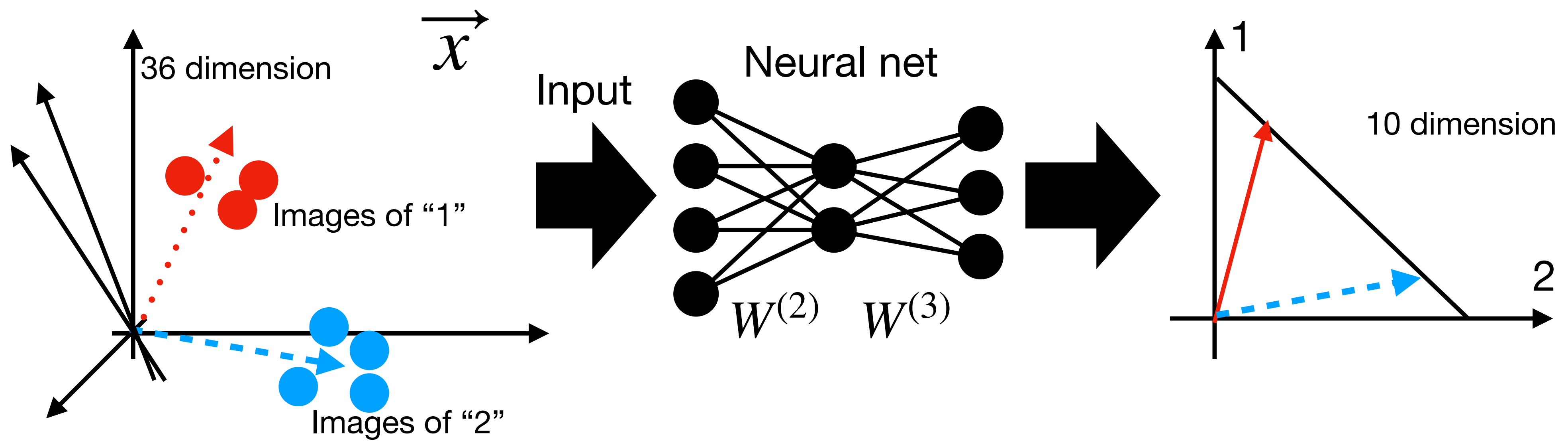
Example: Recognition of hand-written numbers

6x6



$$= \begin{pmatrix} 0.000 \\ 0.000 \\ 0.8434 \\ 0.756 \\ 0.3456 \\ 0.64 \\ 0.251 \\ \vdots \end{pmatrix} = \vec{x}$$

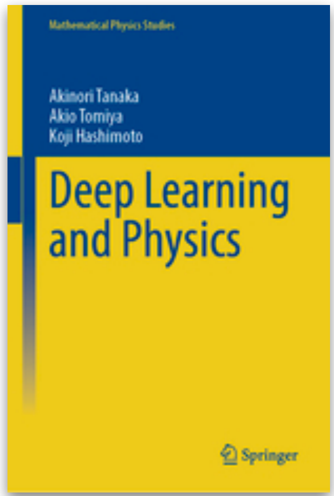
Image is a vector (6x6=36 dim)



“0” = (1,0,0,…)
“1” = (0,1,0,…)
“2” = (0,0,1,…)
…
“9” = (0,0,…,1)

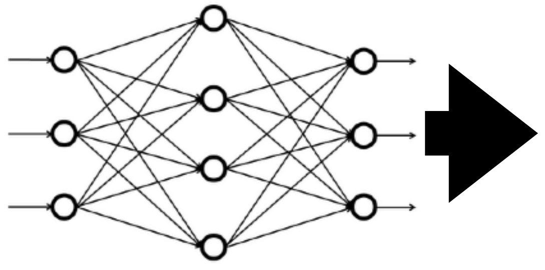
Fact: Neural network can mimic any function
= A systematic variational function.

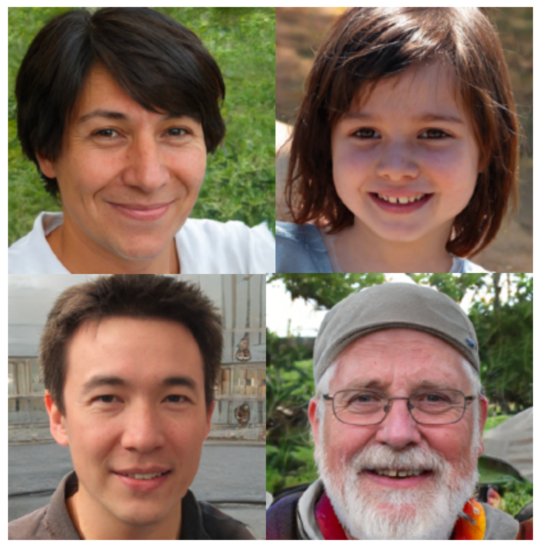
In this example, NN mimics image (36-dim vector) and label (10-dim vector)



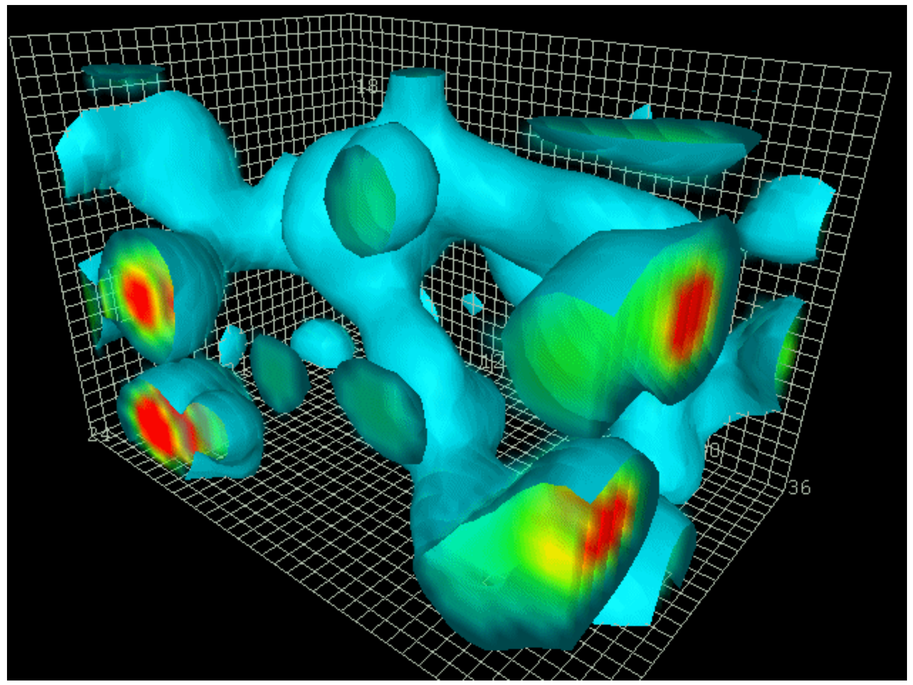
ML for LQCD is needed

- Neural networks
  - Data processing techniques mainly for 2d image (a picture = pixels = a set of real #)
  - Neural network helps data processing e.g. AlphaFold2
- Lattice QCD requires numerical effort but is more complicated than pictures
  - 4 dimension
  - **Non-abelian gauge d.o.f. and symmetry**
  - Fermions (Fermi-Dirac statistics)
  - Exactness of algorithm is necessary
- Q. How can we deal with neural nets?





thispersondoesnotexist.com



http://www.physics.adelaide.edu.au/theory/staff/leinweber/VisualQCD/QCDvacuum/

Gauge covariant neural network
= trainable smearing

Akio Tomiya

AT Y. Nagai arXiv: 2103.11965

<u>Smearing</u> = Gauge covariant way of transform gauge configurations

Covariant sum

$$U_\mu(n) \to U_\mu^{\rm smr}(n) = \mathcal{N}\left[(1-\alpha)U_\mu(n) + \frac{\alpha}{6}V_\mu^\dagger[U](n)\right]$$

Staple

$$V_\mu^\dagger[U](n) = \sum_{\mu\neq\nu} U_\nu(n)U_\mu(n+\hat\nu)U_\nu^\dagger(n+\hat\mu) + \cdots$$

$$\mathcal{N}[M] = \frac{M}{\sqrt{M^\dagger M}}$$

Normalization or projection

<u>Gauge covariant neural network</u> = General smearing with <u>tunable parameters</u> w

$$\begin{cases} z_\mu^{(l)}(n) = w_1^{(l)}U_\mu^{(l-1)}(n) + w_2^{(l)}\mathcal{G}_{\bar\theta}^{(l)}[U] \\ \mathcal{N}(z_\mu^{(l)}(n)) \end{cases}$$

Train (tune, fitting)

point-wise (local)

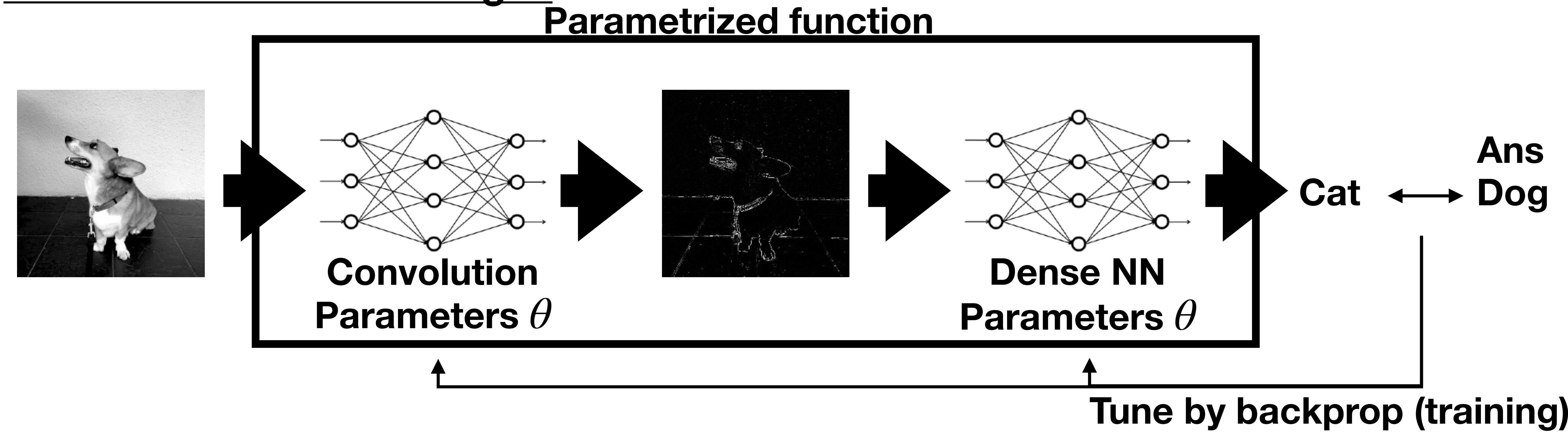
Gauge covariant NN: $U_\mu^{\rm NN}(n)[U] = U_\mu^{(4)}(n)\left[U_\mu^{(3)}(n)\left[U_\mu^{(2)}(n)\left[U_\mu(n)\right]\right]\right]$

Gauge covariant variational map: $U_\mu(n) \mapsto U_\mu^{\rm NN}(n) = U_\mu^{\rm NN}(n)[U]$

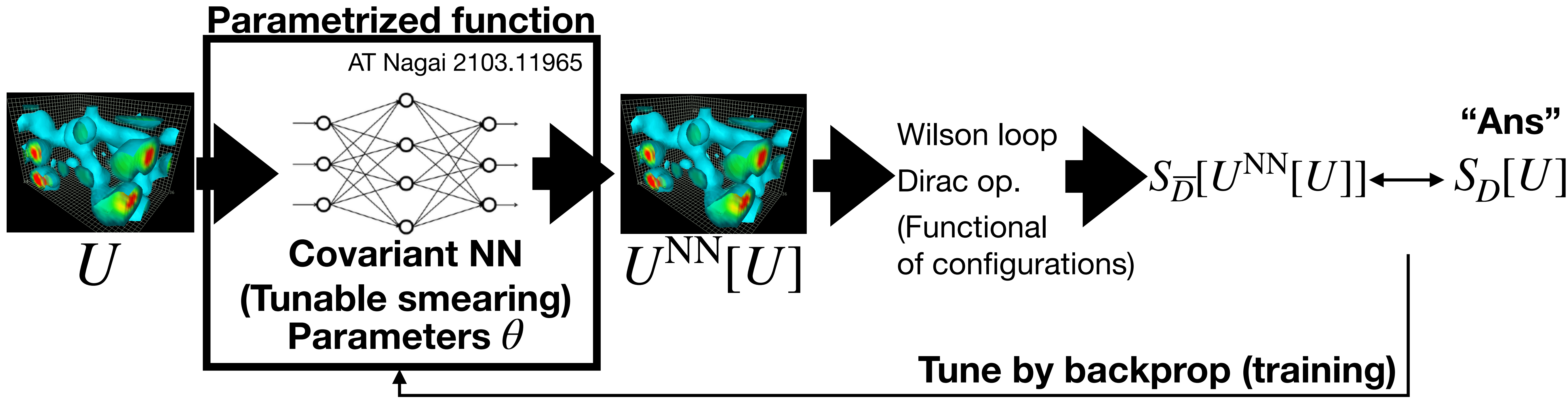
Gauge covariant neural network

Schematic illustrations for neural networks (NN)

Neural networks for images



Neural networks for gauge configurations



http://www.physics.adelaide.edu.au/theory/staff/leinweber/VisualQCD/QCDvacuum/

Alternative realization of gauge symmetric neural net: gauge equivariant neural net -> Tilo Wettig's talk

Gauge covariant neural network
= trainable smearing

AT Y. Nagai arXiv: 2103.11965

Dictionary

	(convolutional) Neural network	Gauge Covariant Neural network
Input	Image (2d data, structured)	gauge config (4d data, structured)
Output	Image (2d data, structured)	gauge config (4d data, structured)
Symmetry	Translation	Translation, rotation(90°), Gauge sym.
with Fixed param	Image filter	(APE/stout …) Smearing
Local operation	Summing up nearest neighbor with weights	Summing up staples with weights
Activation function	Tanh, ReLU, sigmoid, …	projection/normalization in Stout/HYP/HISQ
Formula for chain rule	Backprop	"Smeared force calculations" (Stout)
Training?	Backprop + Delta rule	AT Nagai 2103.11965

Well-known

(Index i in the neural net corresponds to n & μ in smearing. Information processing with NN is evolution of scalar field)

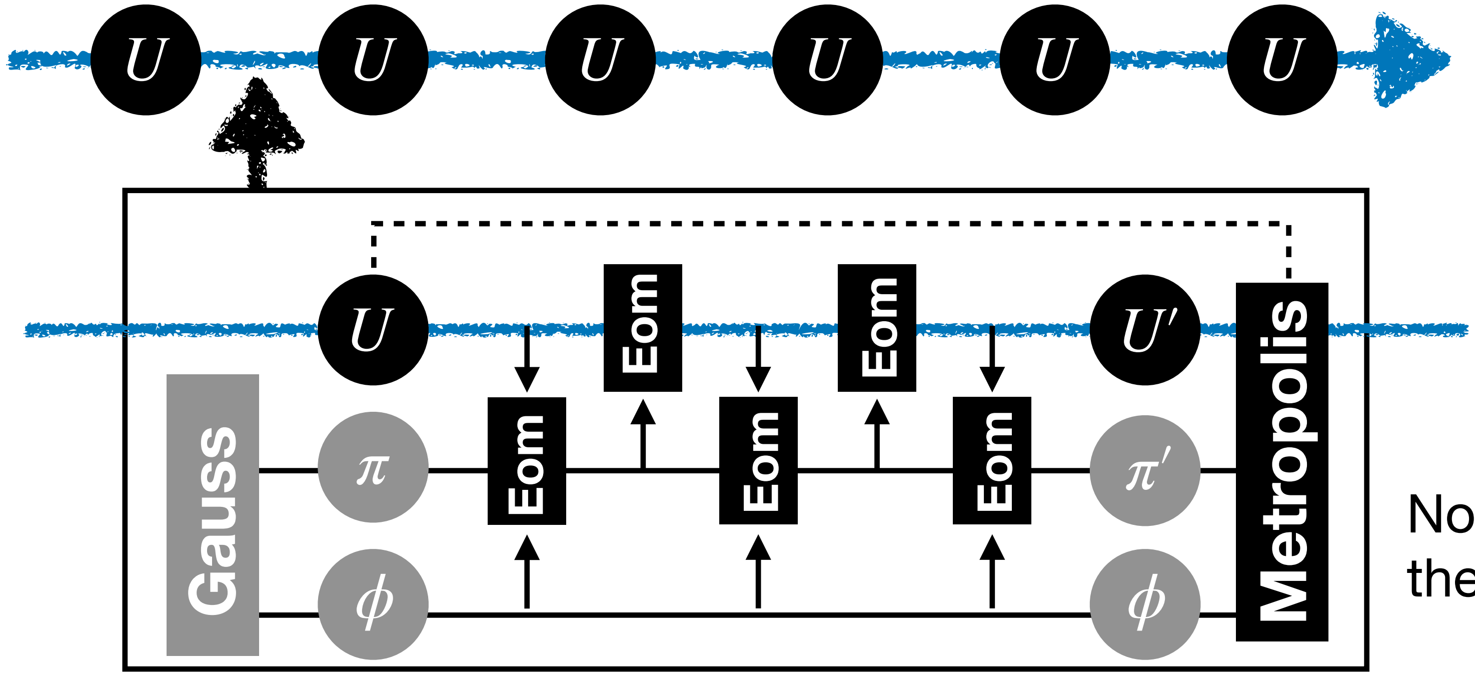
SLHMC = Exact algorithm with ML

Akio Tomiya

SLHMC for gauge system with dynamical fermions

Gauge covariant neural network can mimics gauge invariant functions
-> It can be used in simulation? -> Self learning HMC!

HMC

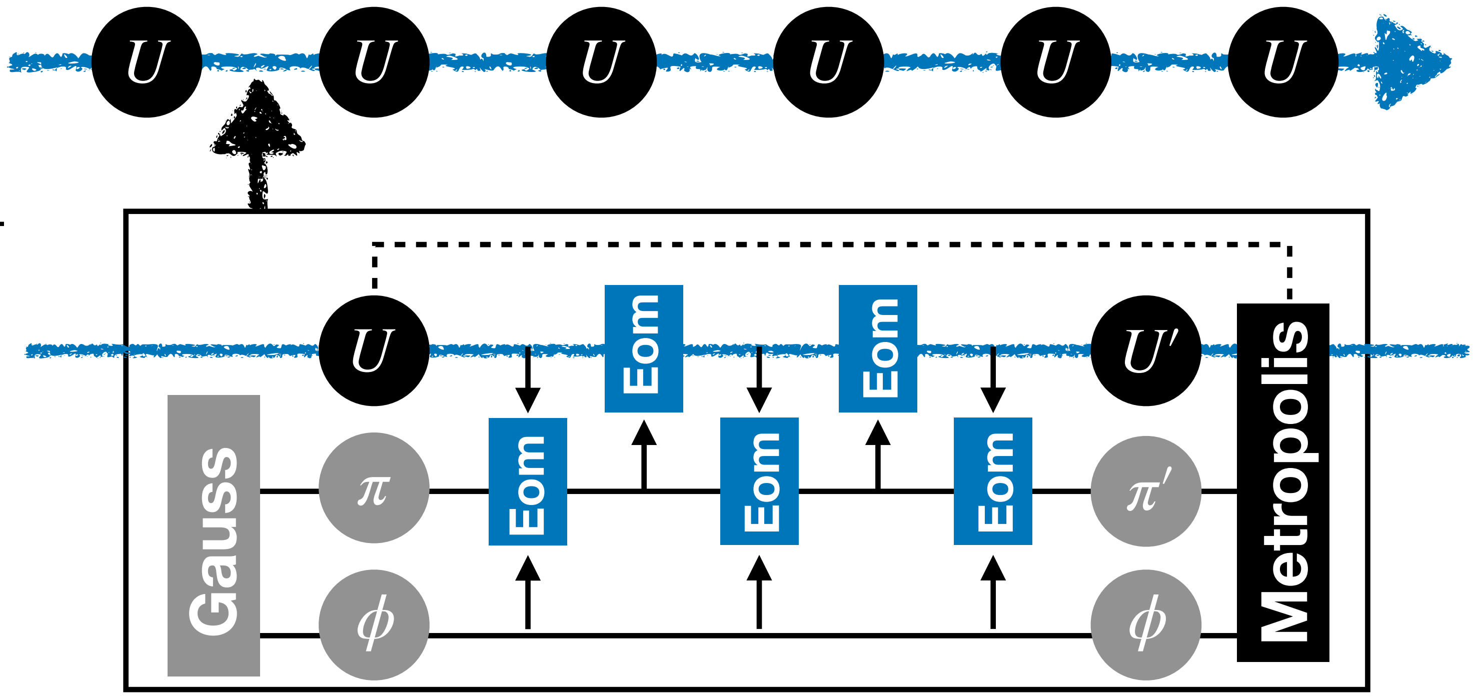


Eom **Metropolis**
Both use

$$H_{\text{HMC}} = \frac{1}{2}\sum \pi^2 + S_{\text{g}} + S_{\text{f}}$$

Non-conservation of H cancels since the molecular dynamics is reversible

Self Learning HMC



Metropolis

$$H = \frac{1}{2}\sum \pi^2 + S_{\text{g}} + S_{\text{f}}[U]$$

Eom

$$H = \frac{1}{2}\sum \pi^2 + S_{\text{g}} + S_{\text{f}}[U^{\text{NN}}[U]]$$

Neural net approximated fermion action but exact

arXiv: 2103.11965
and reference therein

SLHMC works as an adaptive reweighting!

Application for the staggered in 4d

Akio Tomiya

Problems to solve

arXiv: 2103.11965

Mimic different action (Dirac operator):

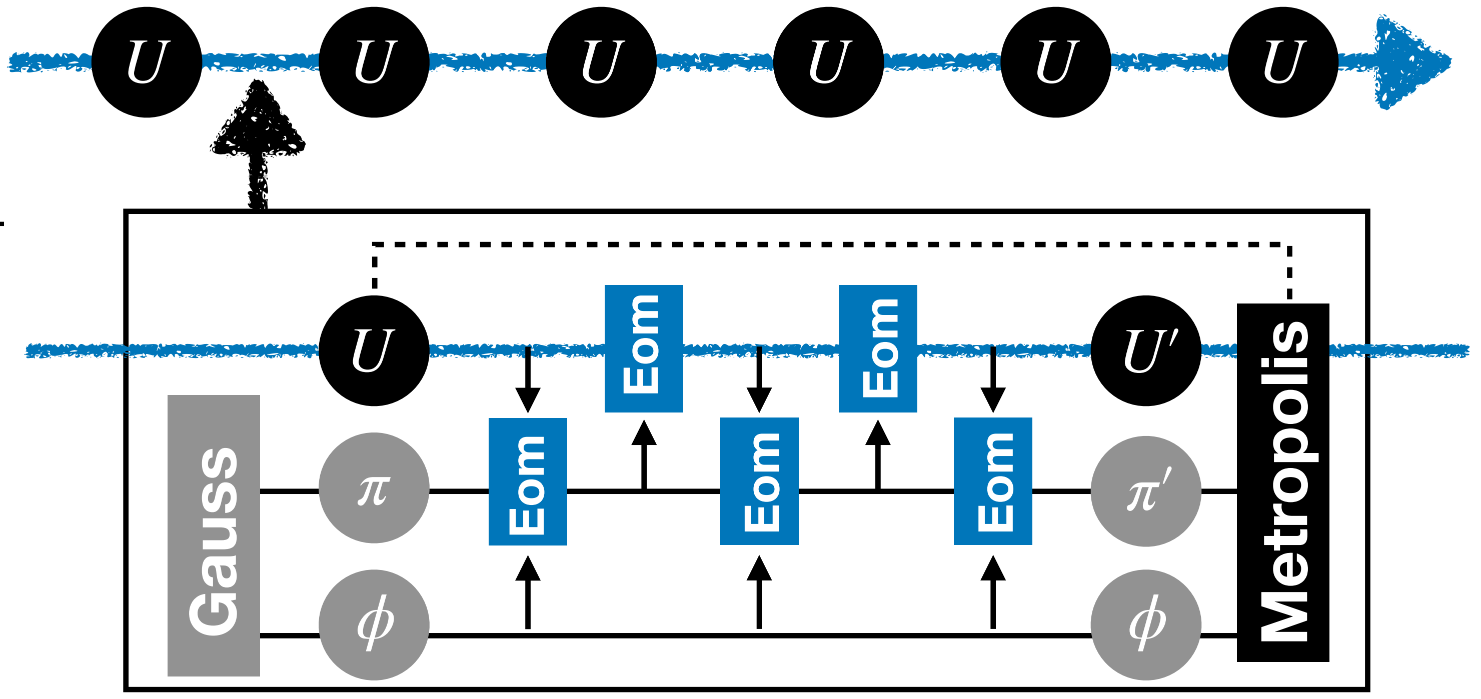
(Final target: Domain-wall vs overlap)
A toy problem: Staggered (heavy) vs Staggered (light)

Target action (Metropolis) $S[U] = S_{\rm g}[U] + S_{\rm f}[\phi, U; m = 0.3],$

Action in MD $S_\theta[U] = S_{\rm g}[U] + S_{\rm f}[\phi, \underline{U^{\rm NN}_\theta[U]}; m_{\rm h} = 0.4],$

mimic

Self Learning HMC



Metropolis

$$H = \frac{1}{2}\sum \pi^2 + S_{\rm g} + S_{\rm f}[U]$$

Eom

$$H = \frac{1}{2}\sum \pi^2 + S_{\rm g} + S_{\rm f}[U^{\rm NN}[U]]$$

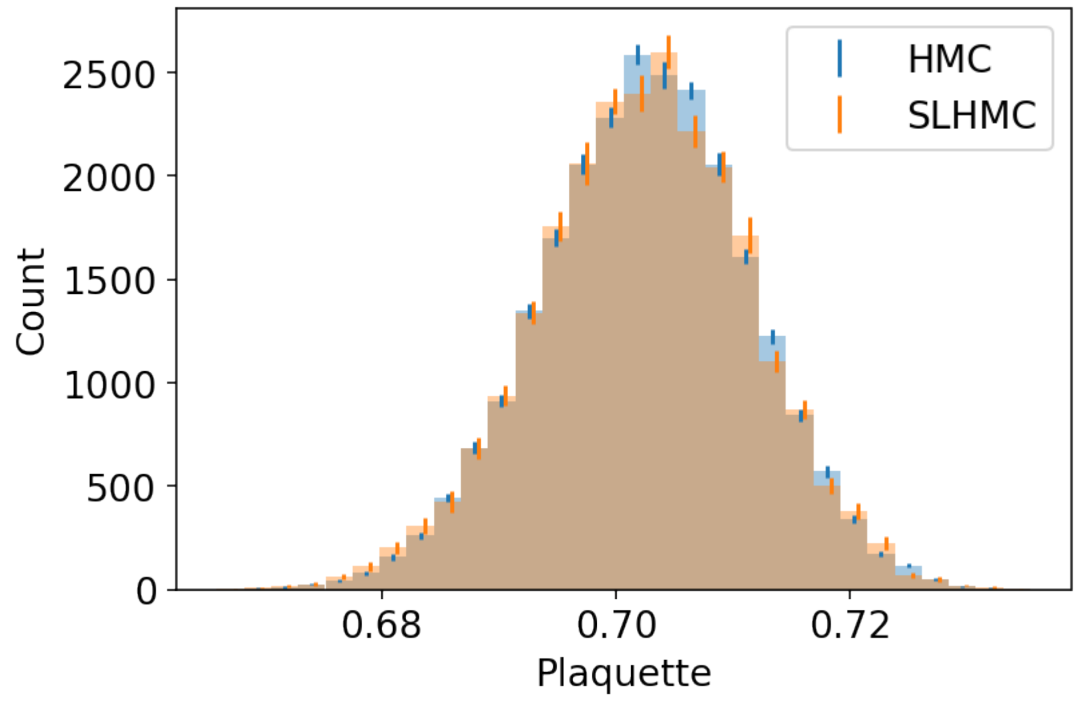
Neural net approximated fermion action but exact

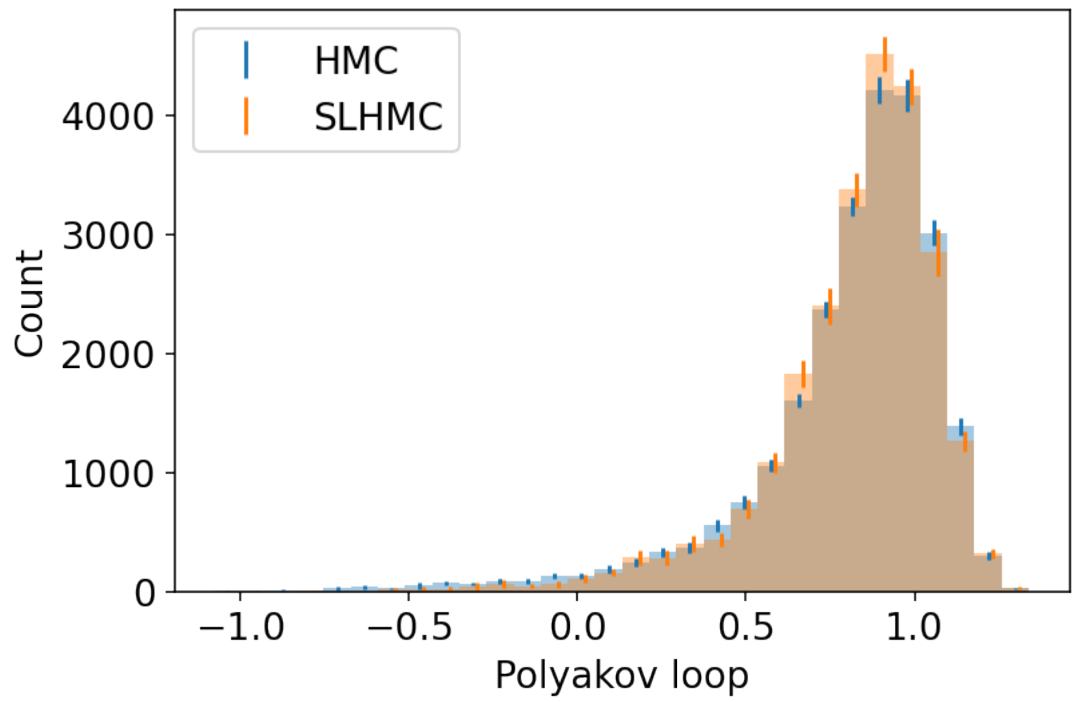
SLHMC works as an adaptive reweighting!

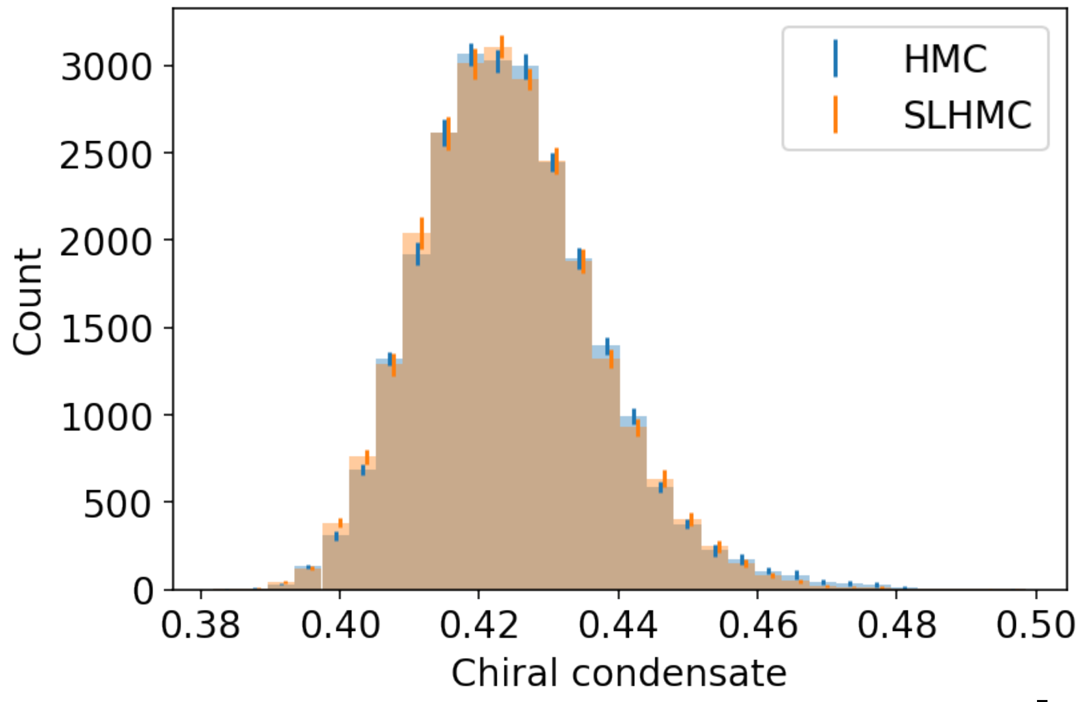
Application for the staggered in 4d

Results are consistent with each other

arXiv: 2103.11965







Expectation value

Algorithm	Observable	Value
HMC	Plaquette	0.7025(1)
SLHMC	Plaquette	0.7023(2)
HMC	|Polyakov loop|	0.82(1)
SLHMC	|Polyakov loop|	0.83(1)
HMC	Chiral condensate	0.4245(5)
SLHMC	Chiral condensate	0.4241(5)

Implemented by LatticeQCD.jl julia

Julia language on Fugaku, Lattice code in Julia

(LatticeQCD.jl and GomalizingFlow.jl)

Benchmark

Speed of Julia ~ Clang

1. Open source scientific language (Just in time compiler/LLVM backend)
2. Fast as C/Fortran (faster sometimes), Practical as Python
3. Machine learning friendly

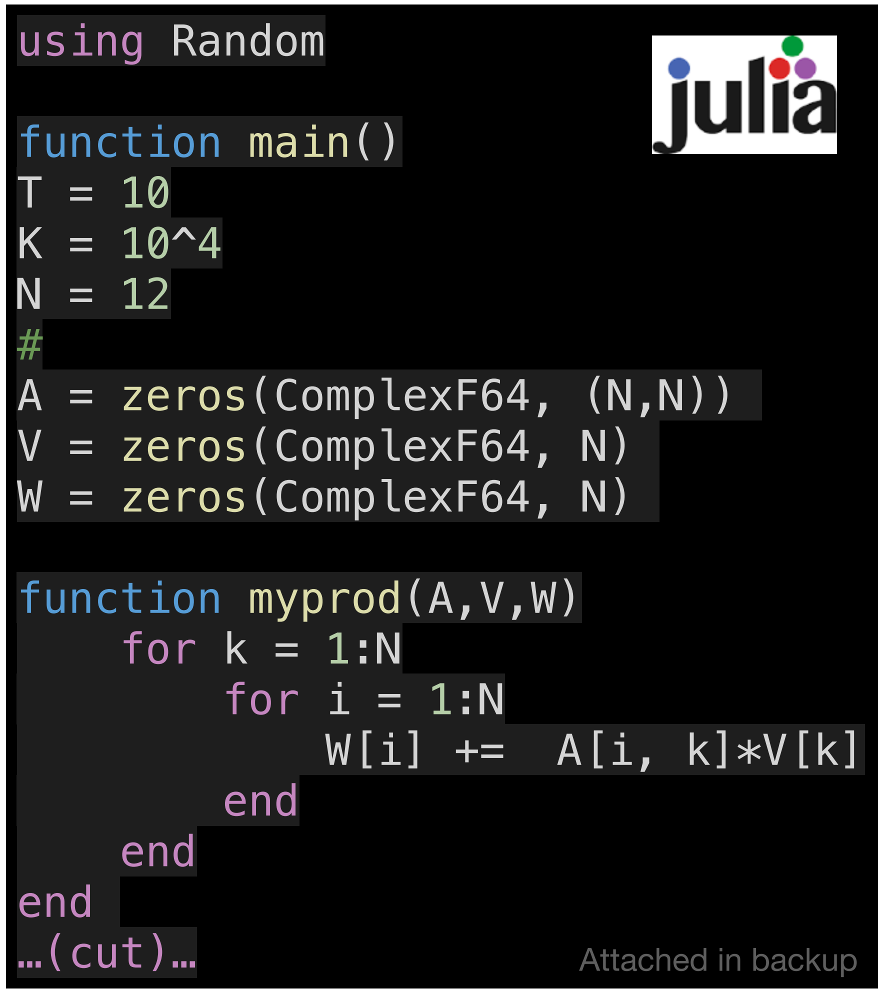
	Compiler	Benchmark (sec) single core	Type	Parallelism	GPU	Pros👍	Cons👎	Column, row	Note
Julia (1.8)	JIT, LLVM	**0.0014**	Dynamic & Static	MPI, others	CUDA	Fast Practical ML feiendly	not major	column-major	
C	Clang (LLVM)	**0.0033**	Static	MPI, others	CUDA	Fast	Long codes	row-major	
Python +Numba	(CPython) JIT, LLVM	**0.0131**	Dynamic	Available	Numba-CUDA	Practical ML feiendly	Not fully supported	row-major (Numpy)	(Rosetta2 is used in benchmark)

C and Julia have similar speed

Benchmarks are performed on m1 mac mini (similar tendency on Xeon)
Benchmark: Multiplications for 12dim vector and 12x12 complex matrix for 10^4 times (repeated 10 times)

Benchmark

Code comparison

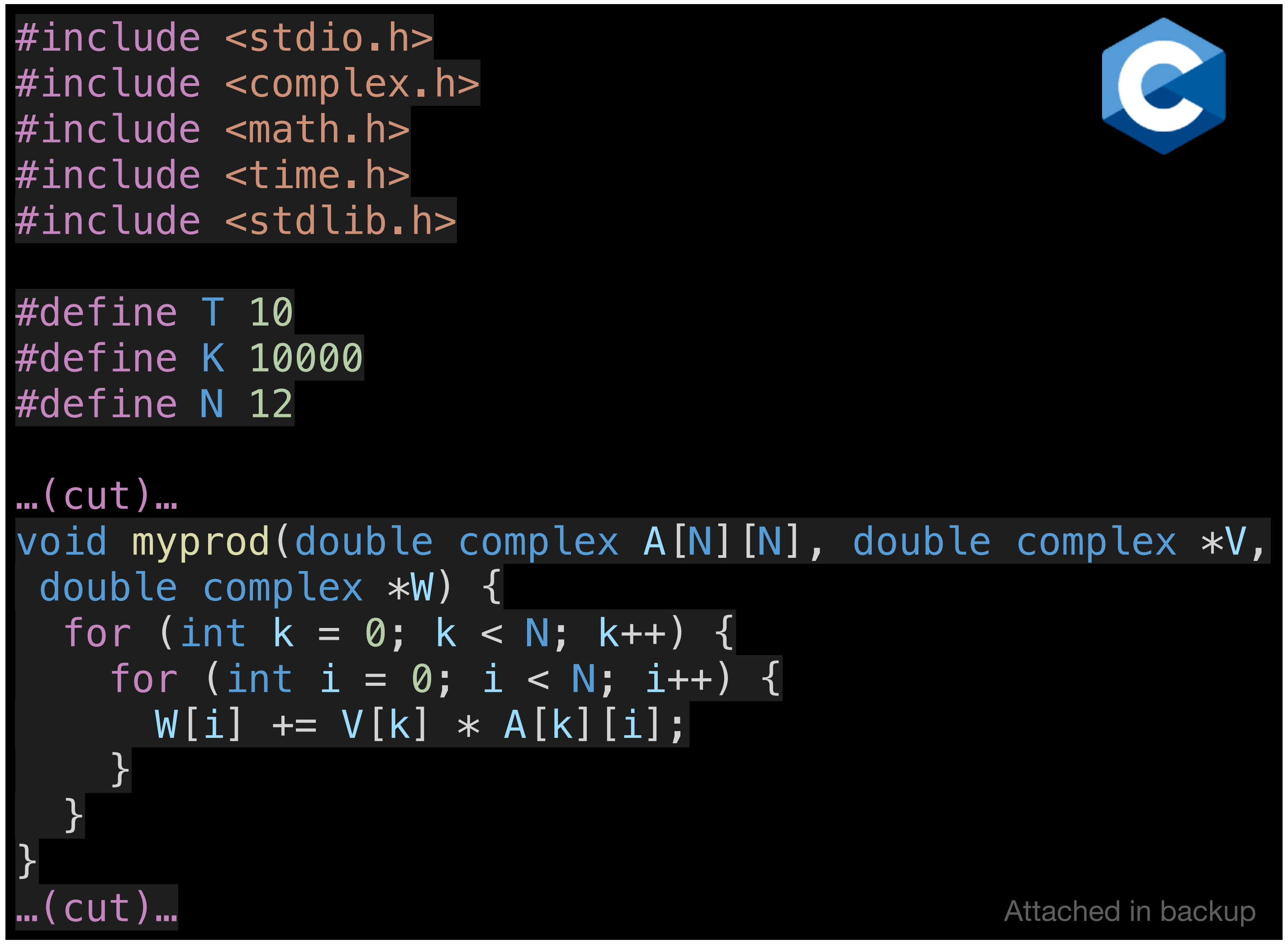


```julia
using Random

function main()
T = 10
K = 10^4
N = 12
#
A = zeros(ComplexF64, (N,N))
V = zeros(ComplexF64, N)
W = zeros(ComplexF64, N)

function myprod(A,V,W)
    for k = 1:N
        for i = 1:N
            W[i] +=  A[i, k]*V[k]
        end
    end
end
…(cut)…
```

Attached in backup



```c
#include <stdio.h>
#include <complex.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

#define T 10
#define K 10000
#define N 12

…(cut)…
void myprod(double complex A[N][N], double complex *V,
 double complex *W) {
  for (int k = 0; k < N; k++) {
    for (int i = 0; i < N; i++) {
      W[i] += V[k] * A[k][i];
    }
  }
}
…(cut)…
```

Attached in backup

- Complex matrix (12x12) times complex vector (d=12)
    - One set= 10^4 times, and repeated 10 times and averaged
- Code of Julia looks like Python (short, simple) but fast as C
  Julia: 0.0014 (sec), C: 0.0033 (sec). Single core performance is similar

Benchmark

Why Julia? (My personal opinion)

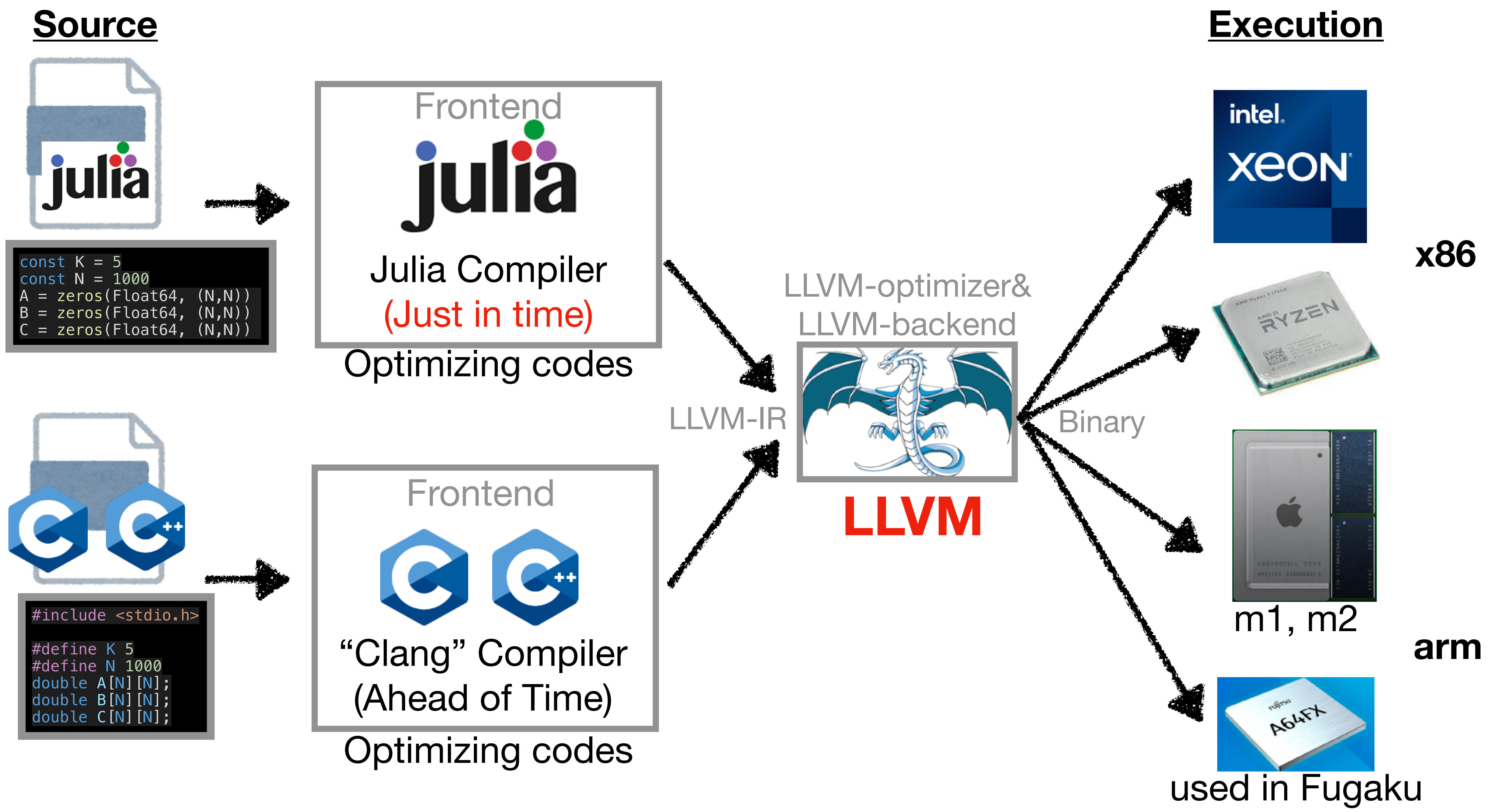
[1] https://akio-tomiya.github.io/julia_in_physics/

[2] https://qr.ae/prgSG5

- Modern scientific programming language
- Easy to make codes. Fast as C/C++ (Julia& C use LLVM)
- Fewer compiling/dependency issues.
- Many people are potentially interested in. (More than 400 people registered to "Julia in physics 2022 online workshop" [1]). 4,923 public repo on Github
- No two Language problem. "The fact that while the users are programming in a high-level language such as R and Python, the performance-critical parts have to be rewritten in C/C++ for performance". [2]
  - Neural network friendly (Flux.jl). Tensor networks also (iTensor.jl).
- Works on/with
  - Xeon, Radeon/Apple silicon/A64FX
  - MPI, GPU

LLVM?

LLVM = common backend for making binaries on multiple architectures



```
const K = 5
const N = 1000
A = zeros(Float64, (N,N))
B = zeros(Float64, (N,N))
C = zeros(Float64, (N,N))
```

```
#include <stdio.h>

#define K 5
#define N 1000
double A[N][N];
double B[N][N];
double C[N][N];
```

https://www.fujitsu.com/jp/about/businesspolicy/tech/fugaku/
https://ja.wikipedia.org/wiki/Apple_M1
https://ja.wikipedia.org/wiki/Ryzen
https://ja.wikipedia.org/wiki/Xeon
https://gigazine.net/news/20200623-japan-fugaku-fastest-supercomputer/
See: https://en.wikipedia.org/wiki/LLVM and related pages

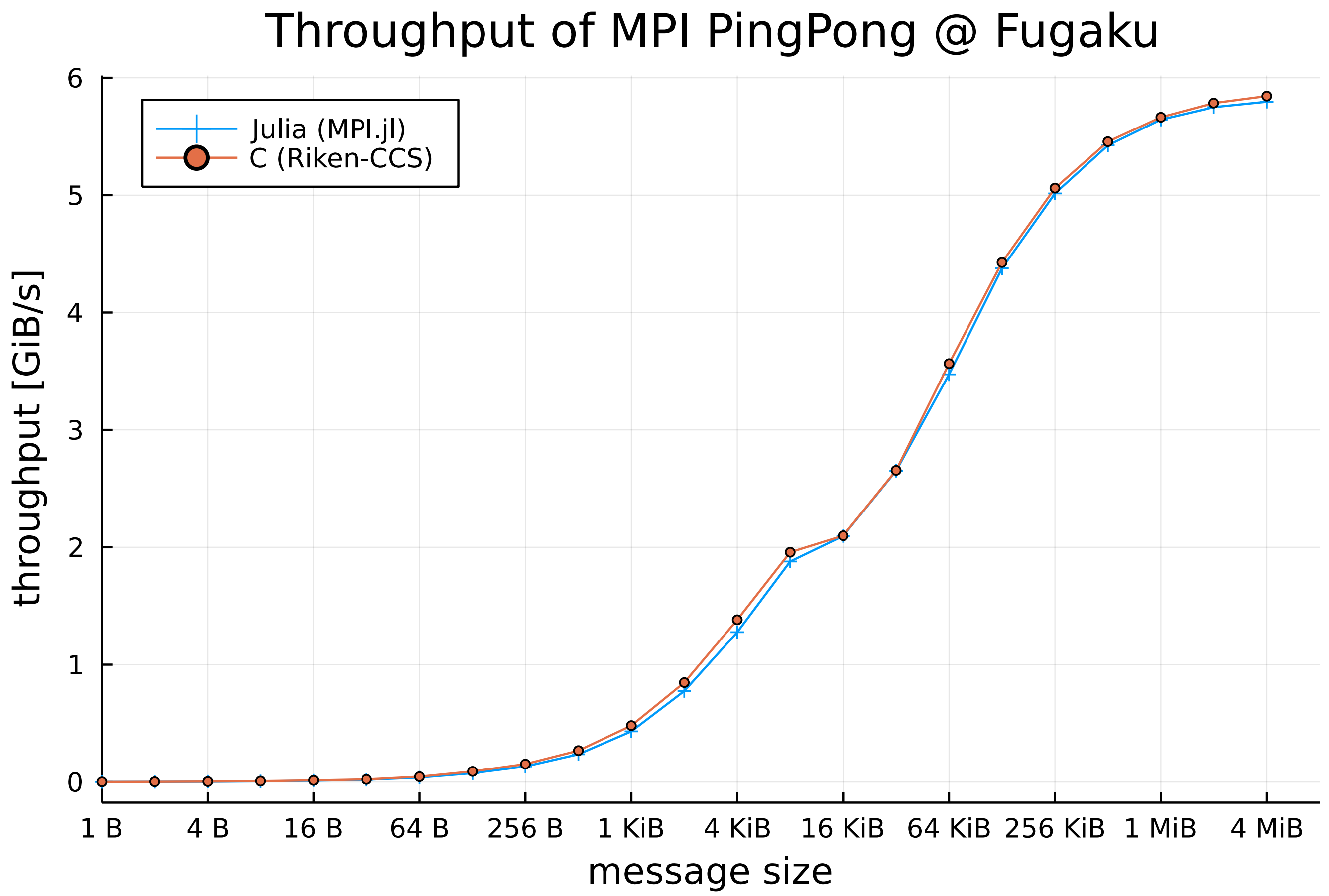
Julia is ready on Fugaku(?)

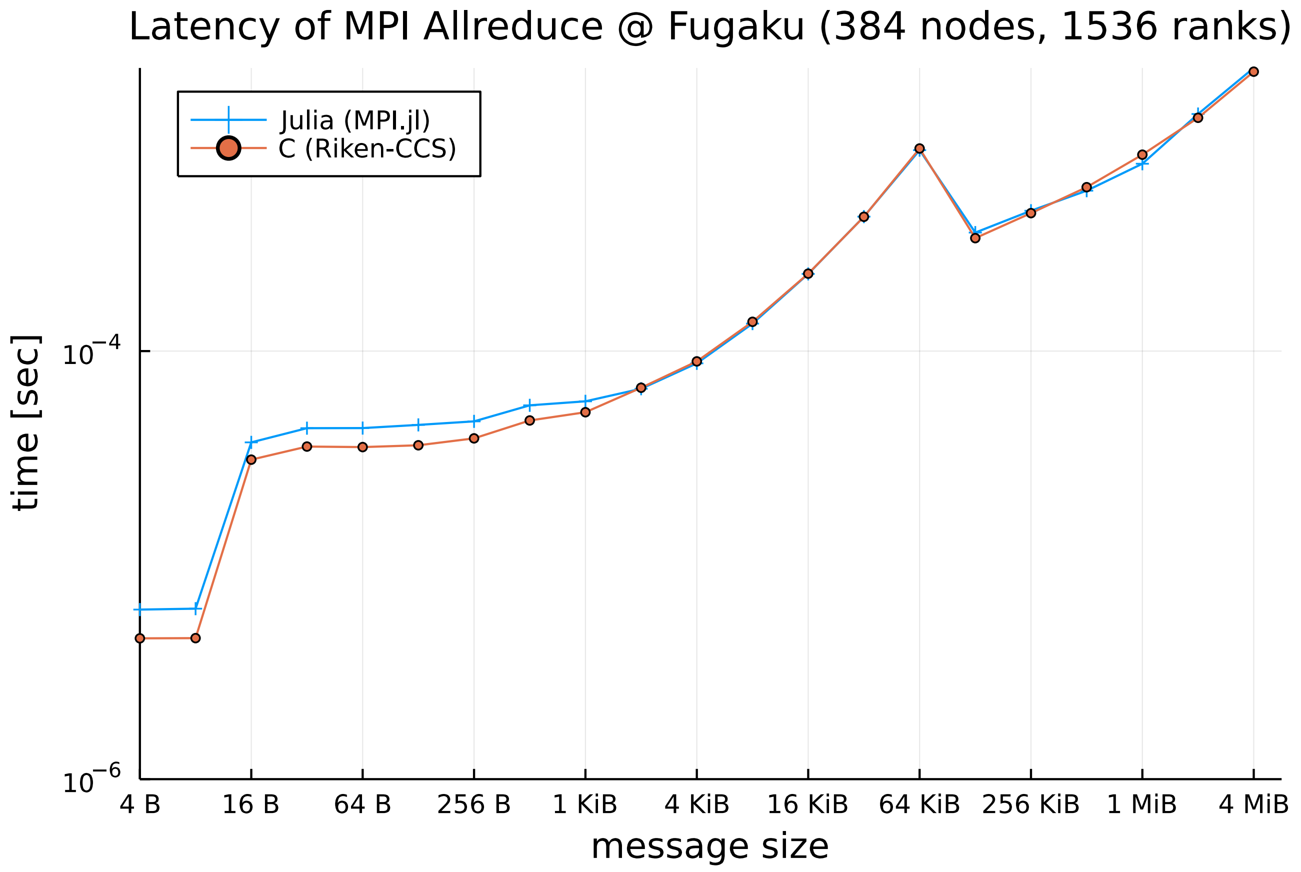
Parallelization with A64FX/Fugaku

M. Giordano, arXiv:2207.12762v1 [cs.DC] 26 Jul 2022

Tests of MPI + julia on Fugaku

Send-Recv performance





julia has similar scaling of MPI with C
(no obvious overhead)

Lattice QCD code

Open source LQCD code in Julia Language

Akio Tomiya

AT & Y. Nagai in prep

LatticeQCD.jl Open source (Julia Official package, Now updated to v1.0)

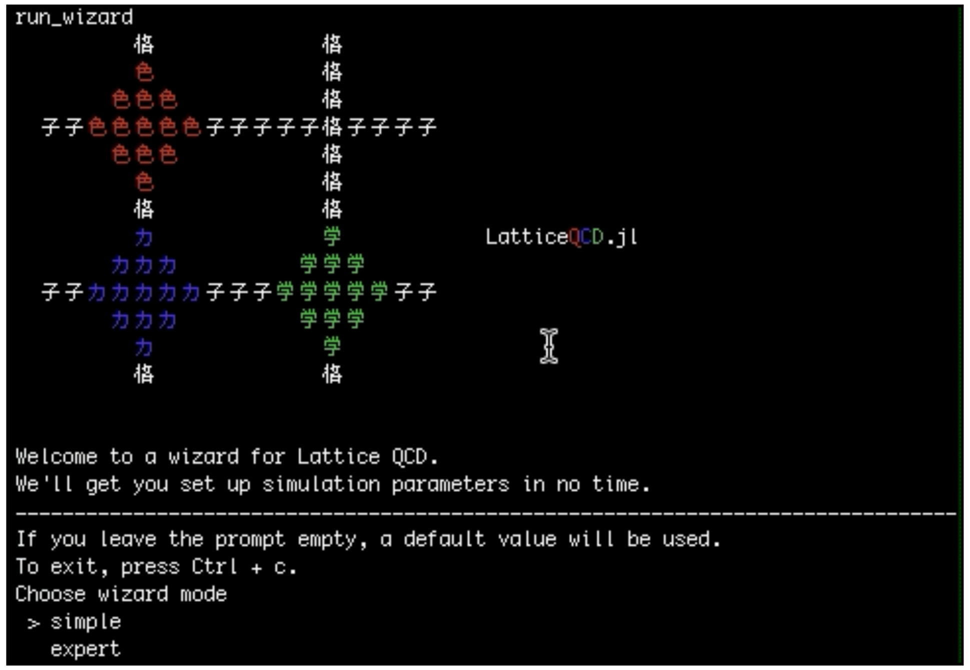
Machines: Laptop/desktop/Jupyter/Supercomputers

Functions: SU(Nc)-heatbath, (R)HMC, Self-learning HMC, SU(Nc) Stout
Dynamical Staggered, Dynamical Wilson, Dynamical Domain-wall
Measurements

Start LQCD in 5 min

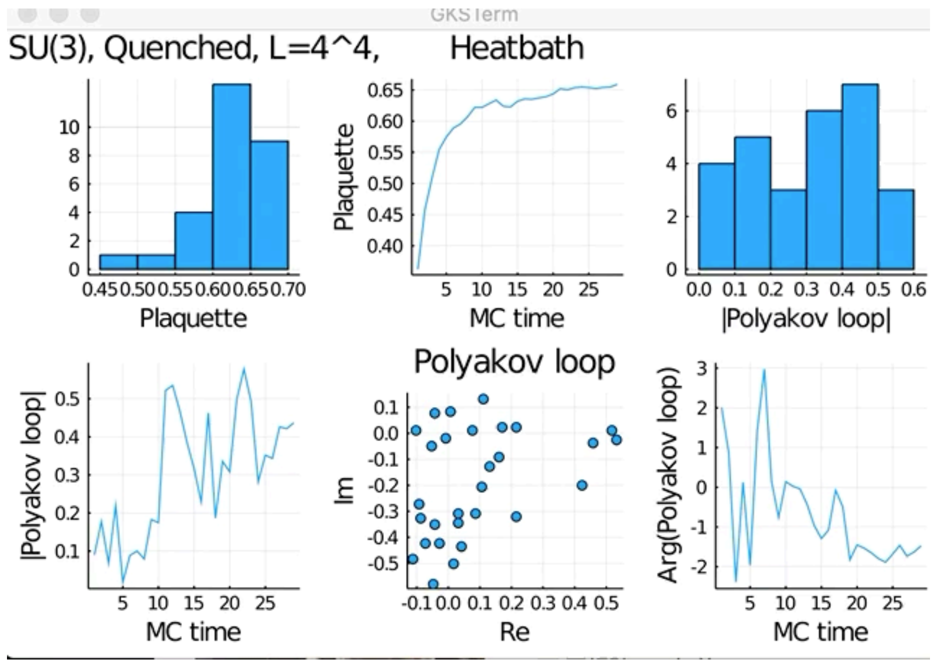
1. Download Julia binary
2. Add the package through Julia package manager
3. Execute!

https://github.com/akio-tomiya/LatticeQCD.jl



```
run_wizard
　　　　　格　　　　　　　格
　　　　　色　　　　　　　格
　　　　色色色　　　　　　格
　子子色色色色色子子子子子格子子子子
　　　　色色色　　　　　　格
　　　　　色　　　　　　　格
　　　　　格　　　　　　　格
　　　　　力　　　　　　　学　　　　　　　　LatticeQCD.jl
　　　　力力力　　　　　学学学
　子子力力力力力子子子学学学学学子子
　　　　力力力　　　　　学学学
　　　　　力　　　　　　　学
　　　　　格　　　　　　　格


Welcome to a wizard for Lattice QCD.
We'll get you set up simulation parameters in no time.
--------------------------------------------------------------------------------
If you leave the prompt empty, a default value will be used.
To exit, press Ctrl + c.
Choose wizard mode
 > simple
   expert
```

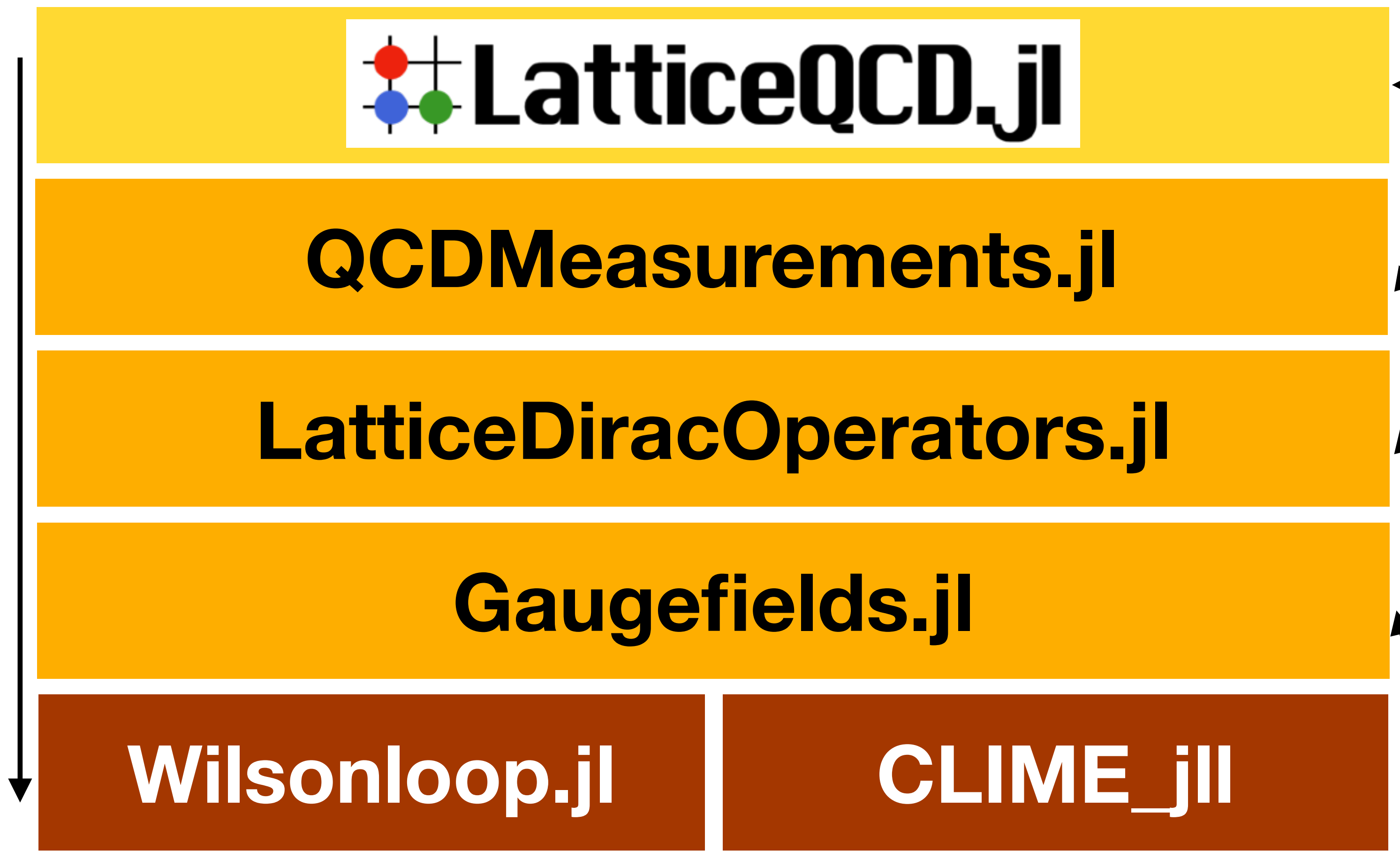


Package structure

Akio Tomiya

Our lattice QCD codes are constructed by following repositories

Dependency (Automatically solved)



Wrapper for LatticeDiracOperators.jl & Gaugefields.jl, QCDMeasurements.jl
- Wizard for parameter files
- HMC/RHMC for SU(Nc)
  - Stout + Wilson/Staggered/DW
- Heatbath for SU(Nc)
- Measurements
- Jupyter, Colab/**PC/Supercomputers**

etc

Measurements in LQCD (Correlator, Flow, Qtop, etc)

Fermions (+HMC), Wilson, KS, DW, MPI
PC/Supercomputers

Gauge fields (+HMC/Heatbath), MPI
PC/Supercomputers

ILDG I/O

Symbolic operations of Wilson/Polyakov loops

See https://github.com/akio-tomiya/LatticeQCD.jl in detail

Benchmark of Julia + QCD

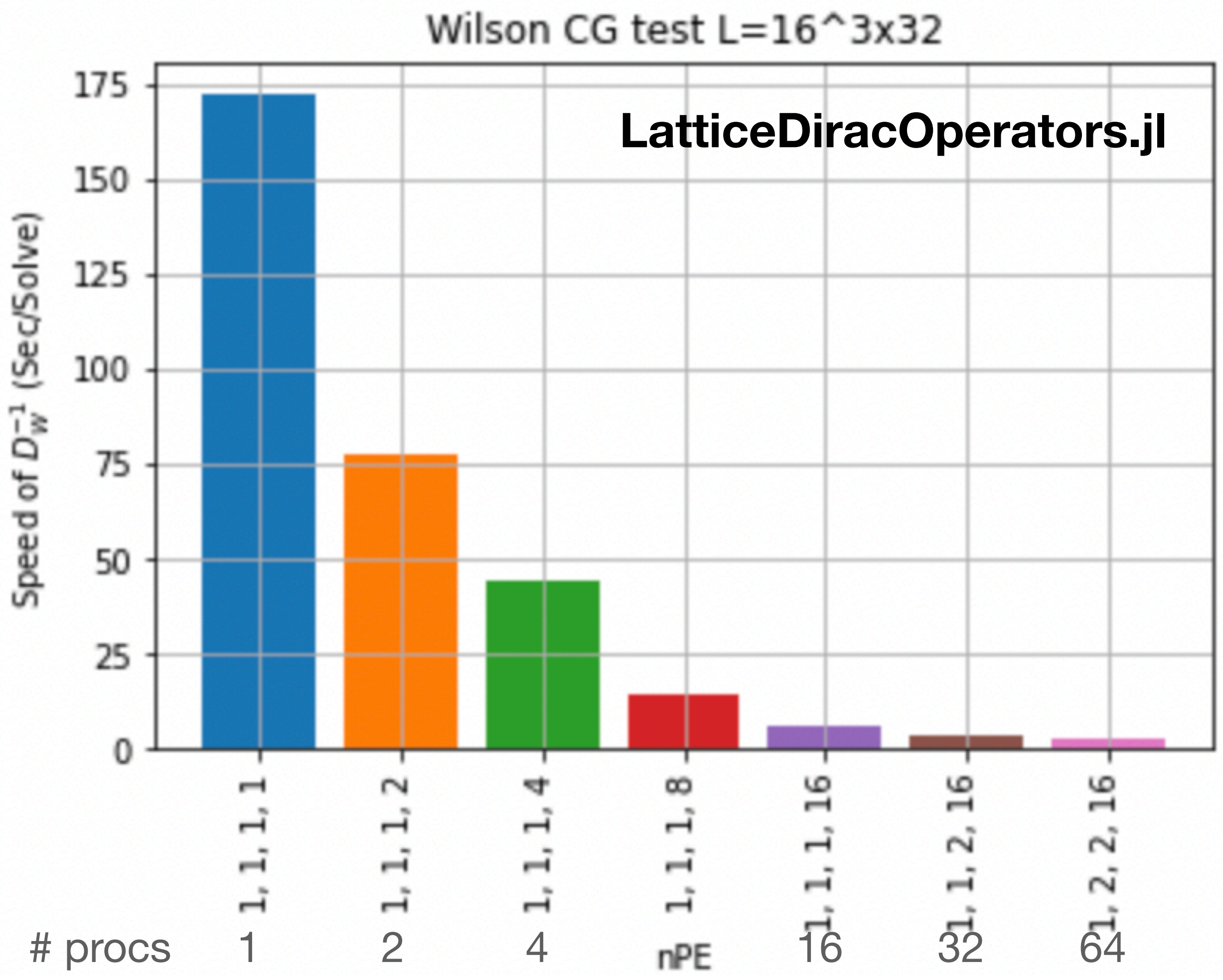
Wilson inversion / MPI parallel, Strong Scaling

Akio Tomiya

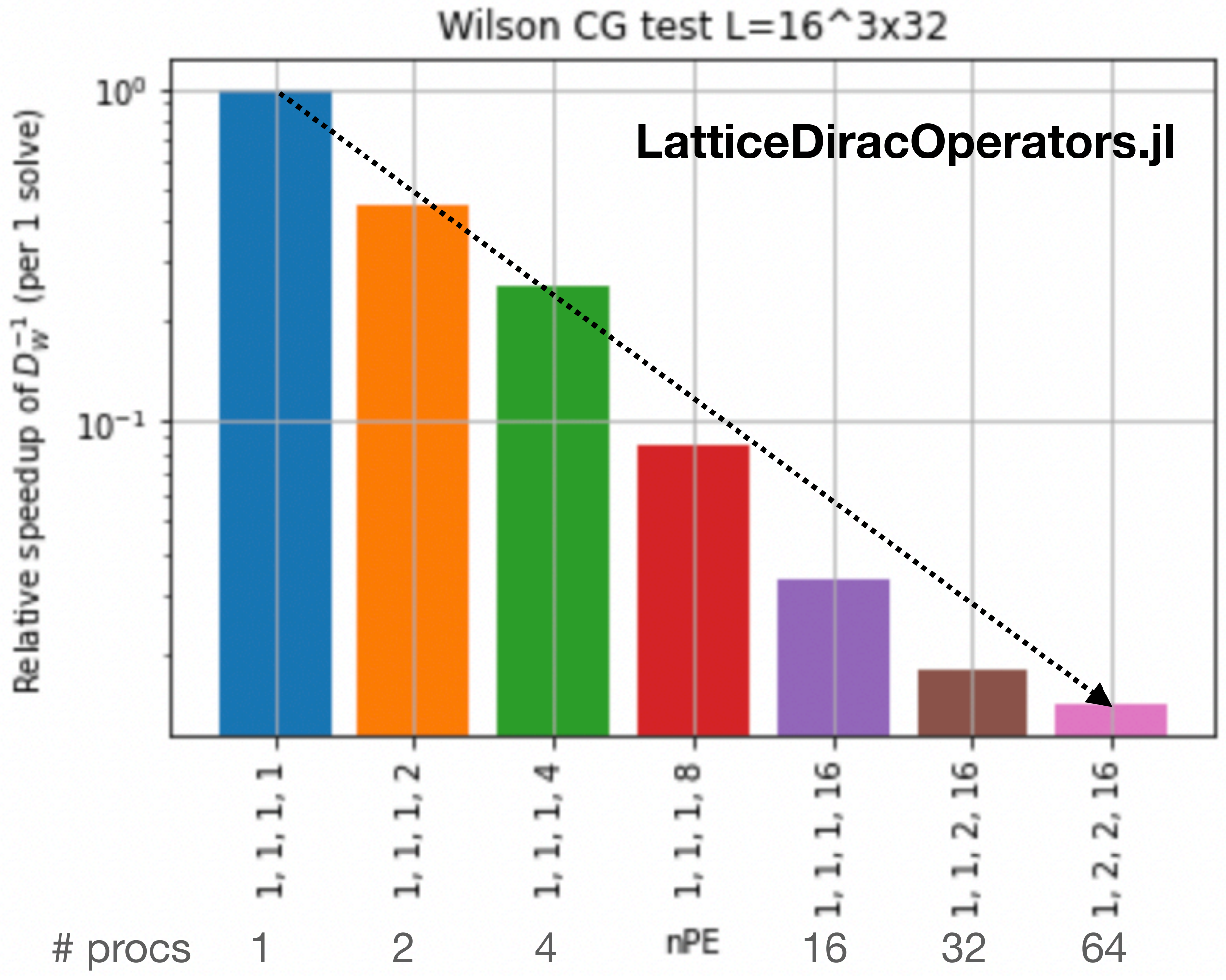
AT & Y. Nagai in prep

Tested on Yukawa-21@YITP

Absolute execution time



Relative speed up



It looks scaling well

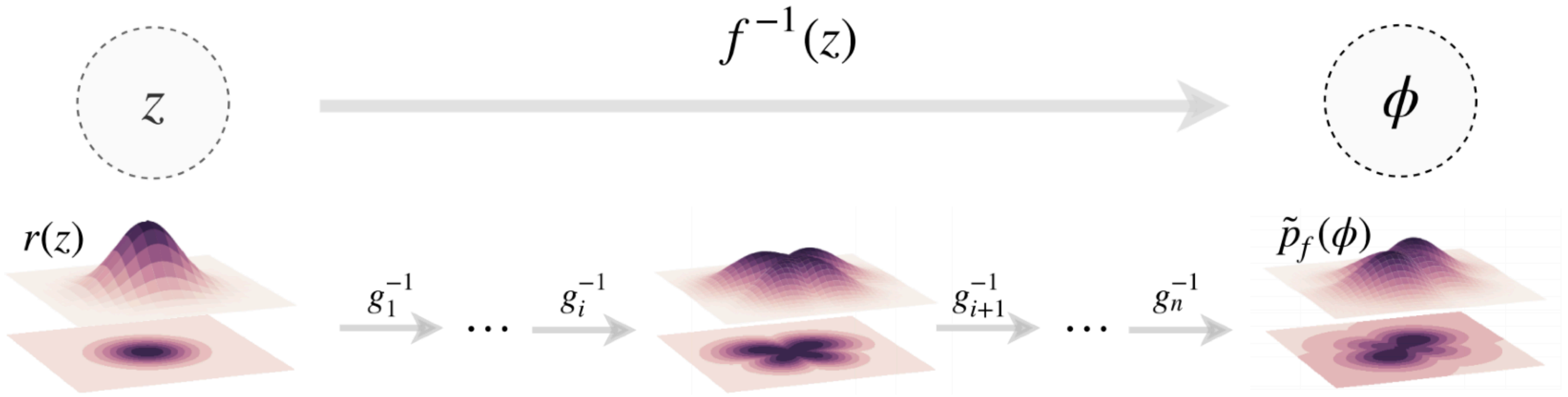
We need more contributors!
Please help us

We thank to H. Ohno & Issaku Kanamori

Flow based sampling algorithm

Trivializing map realized using neural network

Normalizing flow? = Trivializing map, exact MCMC with ML



(a) Normalizing flow between prior and output distributions

Change of variable by a neural network (Normalizing flow)

$$\phi = F^{(\mathrm{NN})}[\varphi]$$

Sampling from Gaussian
→ Inverse trivializing map (neural net)
→ QFT configurations → Tractable Jacobian (by even-odd strategy)
→ After sampling, Metropolis-Hastings test → exact!

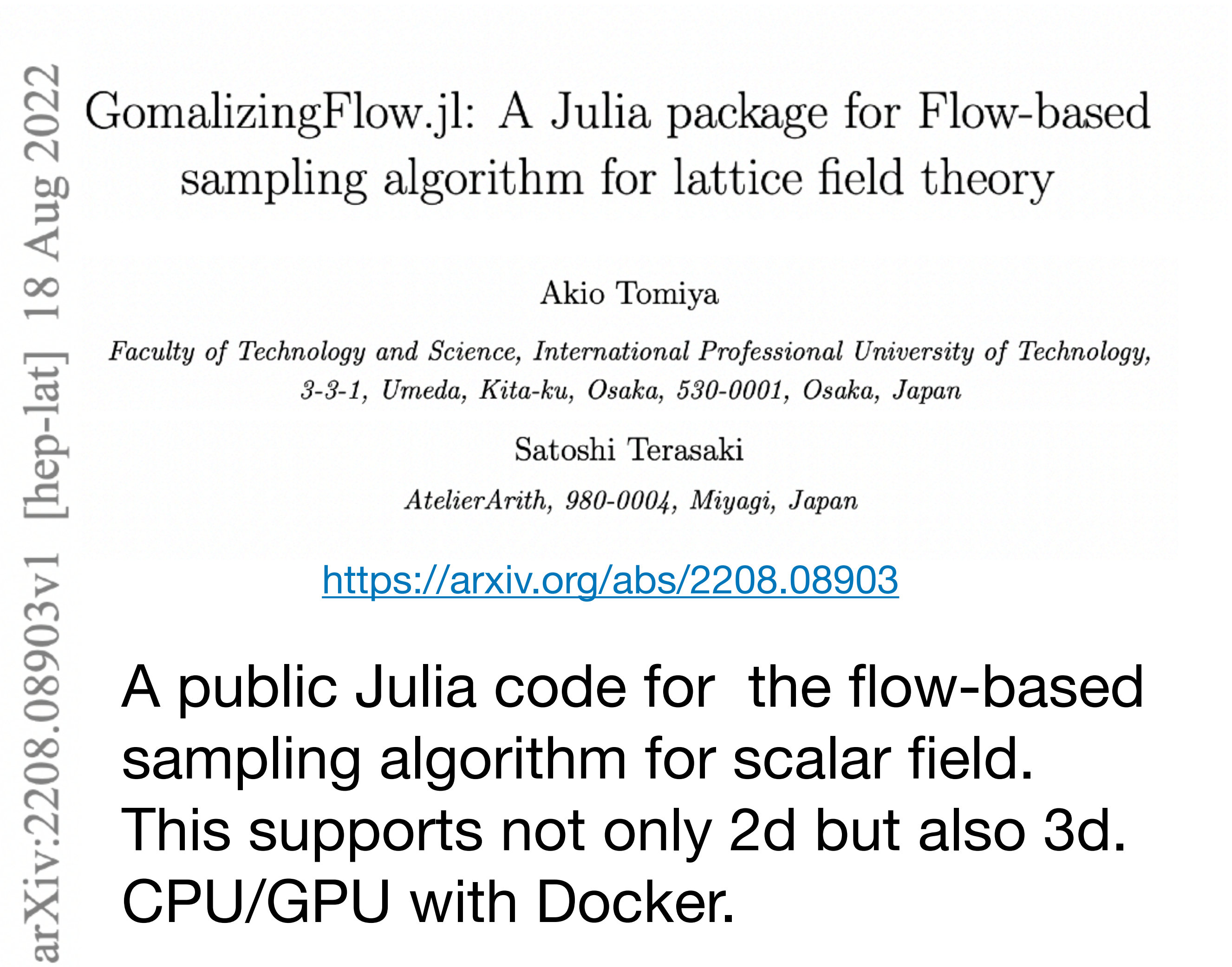
arxiv 1904.12072, 2003.06413, 2008.05456

Related talk: Nobuyuki Matsumoto (Feb 16)

Normalizing flow in Julia

A public code in Julia Language

Akio Tomiya



arXiv:2208.08903v1 [hep-lat] 18 Aug 2022

GomalizingFlow.jl: A Julia package for Flow-based sampling algorithm for lattice field theory

Akio Tomiya

Faculty of Technology and Science, International Professional University of Technology, 3-3-1, Umeda, Kita-ku, Osaka, 530-0001, Osaka, Japan

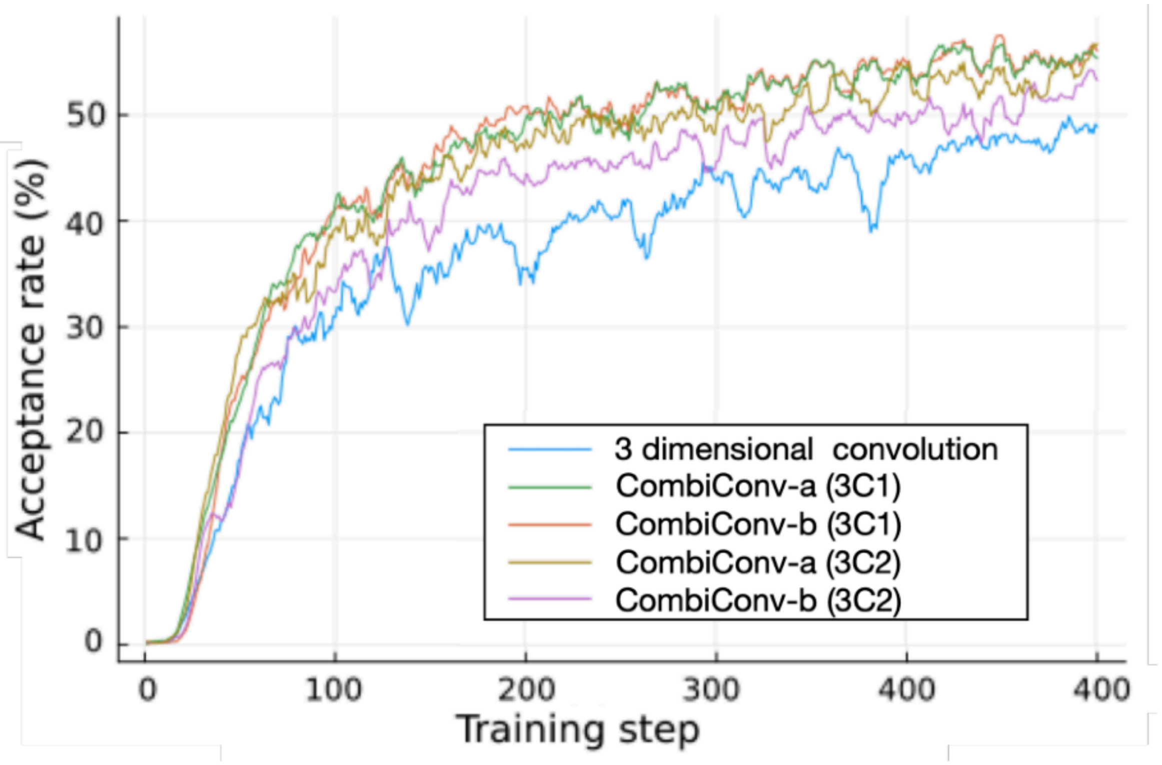
Satoshi Terasaki

AtelierArith, 980-0004, Miyagi, Japan

https://arxiv.org/abs/2208.08903

A public Julia code for the flow-based sampling algorithm for scalar field.
This supports not only 2d but also 3d.
CPU/GPU with Docker.

https://github.com/AtelierArith/GomalizingFlow.jl



A new type of convolution improves acceptance rate (~ shorten the autocorrelation)

I reported in NeurIPS 2022 workshop
https://ml4physicalsciences.github.io/2022/

Summary

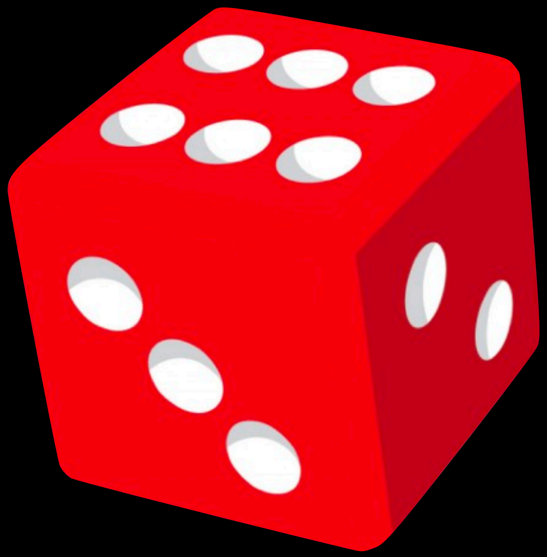
LQCD + Machine learning by Julia language

arXiv: 2103.11965
arXiv: 2208.08903

- Machine learning for LQCD
  - Neural net (NN) + expert knowledge -> Best performance
    e.g. AlphaFold2, NN wave functions
  - NN can deal with 4d non-abelian gauge symmetric data now
  - Self-learning HMC with NN works for dynamical fermions
- Julia language for LQCD/HPC/ML
  - Julia has similar speed with C (w/ & w/o MPI), and machine learning friendly
  - Two Julia codes for lattice field theory
    - LatticeQCD.jl: A suite lattice QCD code, machine learning
    - GomalizingFlow.jl: Trivializing map via a neural network

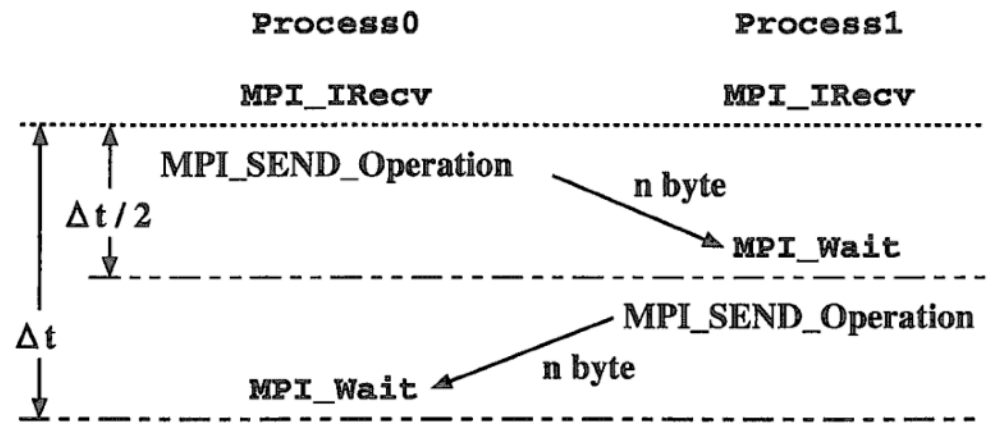
MLPhys Foundation of "Machine Learning Physics"
Grant-in-Aid for Transformative Research Areas (A)
KAKENHI: 20K14479, 22H05112, 22H05111, 22K03539

Thanks!



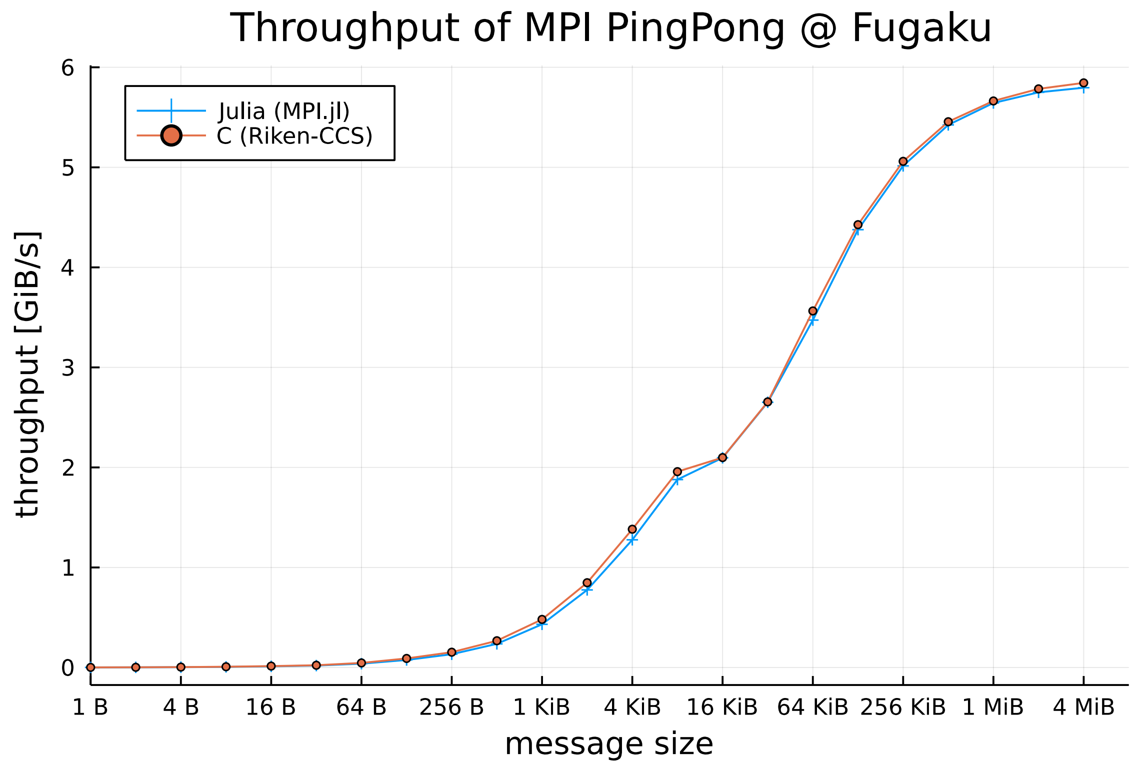
MPI benchmark

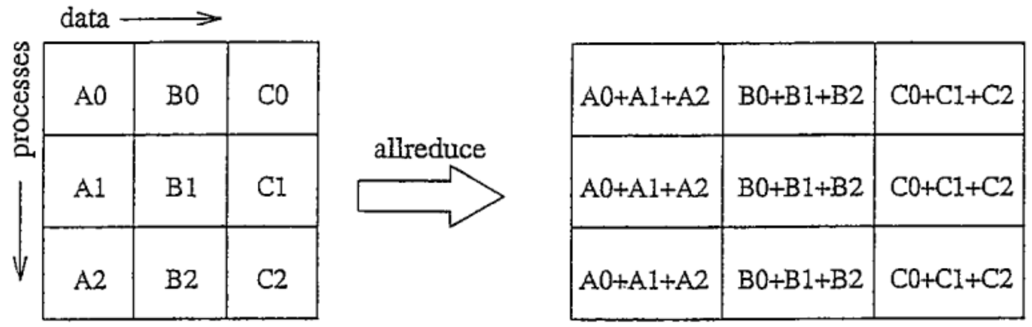
PingPong



This section describes the performance of PingPong as a one-to-one communication. PingPong sends data between two ranks by Send communication from one rank, receives the data at the other rank, and then sends the data back to the original rank by Recv communication.

M. Giordano, arXiv:2207.12762v1 [cs.DC] 26 Jul 2022

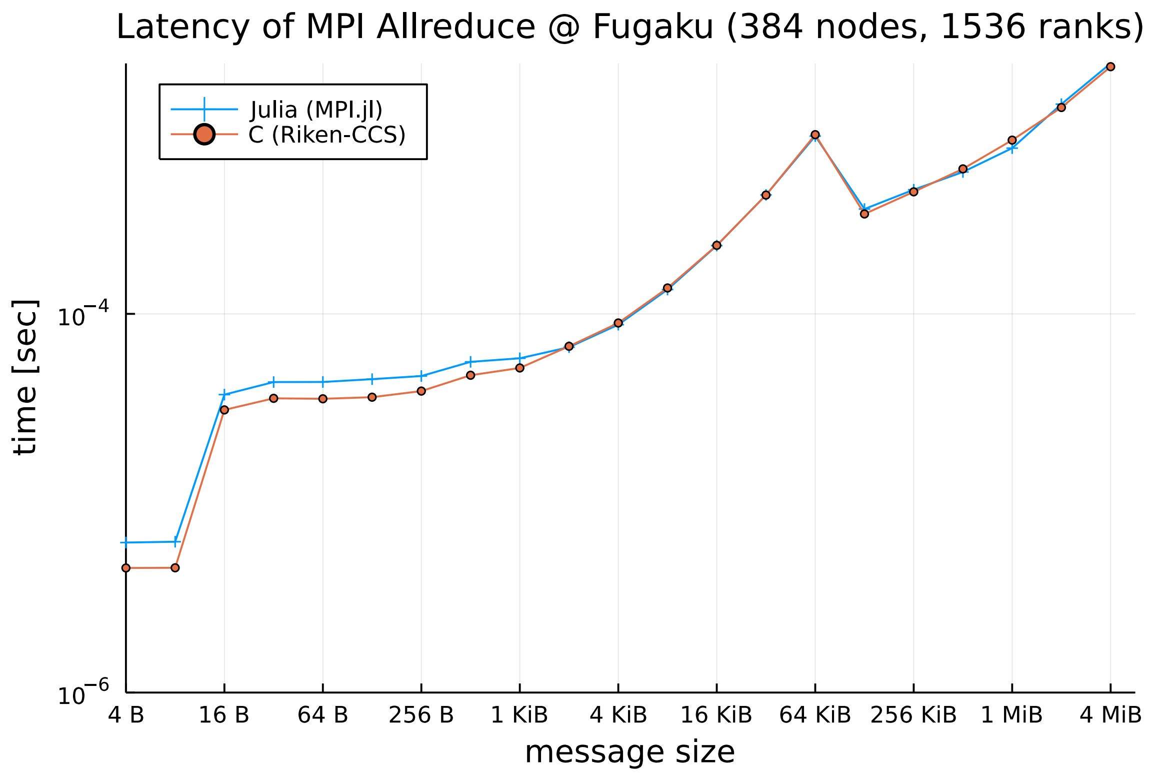




A0	B0	C0
A1	B1	C1
A2	B2	C2

A0+A1+A2	B0+B1+B2	C0+C1+C2
A0+A1+A2	B0+B1+B2	C0+C1+C2
A0+A1+A2	B0+B1+B2	C0+C1+C2

Allreduce collects data from all ranks, performs a set operation, and transfers the result to and transfers the result to all ranks.



https://jopss.jaea.go.jp/pdfdata/JAERI-Data-Code-2001-010.pdf

Introduction

Configuration generation with machine learning is developing

Year	Group	ML	Dim.	Theory	Gauge sym	Exact?	Fermion?	Reference
2017	**AT, Akinori Tanaka**	RBM + HMC	2d	Scalar	-	No	No	arXiv: 1712.03893
2018	K. Zhou+	GAN	2d	Scalar	-	No	No	arXiv: 1810.12879
2018	J. Pawlowski +	GAN +HMC	2d	Scalar	-	Yes?	No	arXiv: 1811.03533
2019	MIT+	Flow	2d	Scalar	-	Yes	No	arXiv: 1904.12072
2020	MIT+	Flow	2d	U(1)	Equivariant	Yes	No	arXiv: 2003.06413
2020	MIT+	Flow	2d	SU(N)	Equivariant	Yes	No	arXiv: 2008.05456
2020	**AT, Akinori Tanaka +**	SLMC	**4d**	SU(N)	Invariant	Yes	Partially	arXiv: 2010.11900
2021	M. Medvidovic´+	A-NICE	2d	Scalar	-	No	No	arXiv: 2012.01442
2021	S. Foreman	L2HMC	2d	U(1)	Yes	Yes	No	
2021	**AT+**	SLHMC	**4d**	QCD	Covariant	Yes	YES!	This talk
2021	L. Del Debbio+	Flow	2d	Scalar, O(N)	-	Yes	No	
2021	MIT+	Flow	2d	Yukawa	-	Yes	Yes	
2021	**S. Foreman, AT+**	Flowed HMC	2d	U(1)	Equivariant	Yes	No but compatible	arXiv: 2112.01586
2021	XY Jing	Neural net	2d	U(1)	Equivariant	Yes	No	
2022	J. Finkenrath	Flow	2d	U(1)	Equivariant	Yes	Yes (diagonalization)	arxiv: 2201.02216
2022	MIT+	Flow	2d, 4d	U(1), QCD	Equivariant	Yes	Yes	arXiv:2202.11712 +
2022	AT+	Flow	2d, 3d	Scalar		Yrs		

+…

Benchmark

Code comparison

```julia
using Random

function main()
T = 10
K = 10^4
N = 12
#
A = zeros(ComplexF64, (N,N))
V = zeros(ComplexF64, N)
W = zeros(ComplexF64, N)

function myprod(A,V,W)
    for k = 1:N
        for i = 1:N
            W[i] +=  A[i, k]*V[k]
        end
    end
end

function test(A,V,W)
    for jj=1:T
        runtimes=[]
        for r=1:K
            A .= rand(N,N) + im*rand(N,N)
            V .= rand(N)   + im*rand(N)
            W .= 0
            tmp = @elapsed myprod(A,V,W)
            push!(runtimes,tmp)
        end
        println("$(jj-1) $(sum(runtimes)) #W[1] = $(W[1])")
    end
end
test(A,V,W)
end

if abspath(PROGRAM_FILE) == @__FILE__
    main()
end
```

```c
#include <stdio.h>
#include <complex.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

#define T 10
#define K 10000
#define N 12

double urand(){
    double m, a;
    m = RAND_MAX + 1;
    a = (rand() + 0.5)/m;
    a = (rand() + a)/m;
    return (rand() + a)/m;
}

void myprod(double complex A[N][N], double complex *V, double complex *W) {
  for (int k = 0; k < N; k++) {
    for (int i = 0; i < N; i++) {
      W[i] += V[k] * A[k][i];
    }
  }
}

void test(double complex A[N][N], double complex *V, double complex *W) {
  for (int jj = 0; jj < T; jj++) {
    double runtimes = 0;
    for (int r = 0; r < K; r++) {
      for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
          A[i][j] = urand() + urand() * I;
        }
        V[i] = urand() + urand() * I;
        W[i] = 0.0 + 0.0 * I;
      }
      clock_t start = clock();
      myprod(A, V, W);
      clock_t end = clock();
      runtimes += (double)(end - start) / CLOCKS_PER_SEC;
    }
    printf("%d %f # W[0] = %f %f\n", jj, runtimes,  creal(W[0]),  cimag(W[0]) );
  }
}

int main() {
  double complex A[N][N];
  double complex V[N];
  double complex W[N];

  test(A, V, W);

  return 0;
}
```