Application and Performance of PyQUDA

第四届中国格点量子色动力学研讨会

蒋翔宇
中国科学院理论物理研究所
2024年10月12日

PyQUDA

What can we do?

- A Python wrapper for QUDA
  - https://github.com/CLQCD/PyQUDA
- Wilson/clover/HISQ fermion propagator
- Clover/HISQ RHMC (without stout smearing in the action)
- Distillation (Laplacian eigensystem, perambulator)
- APE/stout/HYP gauge smearing, Gaussian quark smearing
- Wilson/Symanzik flow
- Gauge fixing (WIP)

Hardware and software

- 1x AMD EPYC 7532 32-Core Processor
- 8x 16GB DDR4 2133MHz memory (~8x 17 GB/s)
- 4x NVIDIA Tesla P100 PCIe 16GB (~4x 700 GB/s)
- Debian GNU/Linux 12 (bookworm) with Linux kernel 6.1
- GCC 11.3.0, Clang/LLVM 14.0.6, CUDA toolkit 11.8, CUDA runtime 12.2
- Latest QMP, QDP-JIT, QUDA, Chroma and MILC
- All GPUs are running PCIe 3.0 x16 (~4x 16 GB/s)

Wilson flow scaling

Compare with Chroma+QDP-JIT

- $\frac{\mathrm{d}}{\mathrm{d}t} V_t(x,\mu) = -g_0^2 \frac{\partial S_g(V_t)}{\partial V_t(x,\mu)} V_t(x,\mu),\ V_t(x,\mu)\,|_{t=0} = U_\mu(x)$
- Take field energy $E = \mathscr{E}^2 + \mathscr{B}^2$, we have some scaling parameters
  - $t^2\langle E\rangle\,|_{t=t_0} = 0.3$
  - $t\frac{\mathrm{d}}{\mathrm{d}t}\left(t^2\langle E\rangle\right)|_{t=w_0^2} = 0.3$

```
from pyquda import init, getLogger
from pyquda.utils import io

init([1, 1, 1, 4], resource_path=".cache")
gauge = io.readChromaQIOGauge("/public/ensemble/C24P29/beta6.20
t0, w0 = gauge.wilsonFlowScale(1200, 0.01)
```

Wilson flow scaling

- 24^3x72, 1/4 GPUs
- 0.01 flow time per step
- PyQUDA gives 6~11x performance
- Has been used on scale setting for CLQCD ensembles

```
X_ttFF = 2.03 a = 0.10099098164878 ttFF = 0.300008444328942
X_ttFF_itp = 2.02992329311151 a_itp = 0.100992889758021 ttFF_itp = 0.3
X_tDFF = 2.65 a = 0.105965907653857 tDFF =  0.299934886272252
X_tDFF_itp = 2.64550253357446 a_itp = 0.106055942671235 tDFF_itp = 0.3
END_ANALYZE_wflow
 more step needed to get  X_ttFF2
 more step needed to get  X_tDFF2
END_ANALYZE_wflow
WILSON_FLOW: total time = 137.152009 secs
WILSON_FLOW: ran successfully

performWFlowQuda: flow t, plaquette, E_tot, E_spatial, E_temporal, Q char
performWFlowQuda: 2.660000e+00 9.9839930842969338e-01 +5.2345300187157939
PyQUDA INFO: t2E(2.66)=0.37037440600425475, tdt2E(2.6550000000000002)=0.3
PyQUDA INFO: t0=2.029924122564845, w0=1.626499929550949
PyQUDA INFO: wilsonFlowScale time: 12.545963258016855 secs
```

```
X_ttFF = 2.03 a = 0.10099098164878 ttFF = 0.300008444328942
X_ttFF_itp = 2.02992329311151 a_itp = 0.100992889758021 ttFF_itp = 0.3
X_tDFF = 2.65 a = 0.105965907653857 tDFF =  0.299934886272237
X_tDFF_itp = 2.64550253357457 a_itp = 0.106055942671232 tDFF_itp = 0.3
END_ANALYZE_wflow
 more step needed to get  X_ttFF2
 more step needed to get  X_tDFF2
END_ANALYZE_wflow
WILSON_FLOW: total time = 44.888519 secs
WILSON_FLOW: ran successfully

performWFlowQuda: flow t, plaquette, E_tot, E_spatial, E_temporal, Q char
performWFlowQuda: 2.660000e+00 9.9839930842969338e-01 +5.2345300187157939
PyQUDA INFO: t2E(2.66)=0.37037440600425475, tdt2E(2.6550000000000002)=0.3
PyQUDA INFO: t0=2.029924122564845, w0=1.626499929550949
PyQUDA INFO: wilsonFlowScale time: 6.5474567748606205 secs
```

Gauge fixing (WIP)

Compare with Chroma+QDP-JIT

- A rotation $g(x)$ implies a gauge transform $U'_\mu(x) = g(x)U_\mu(x)g^\dagger(x+\hat{\mu})$
- Coulomb/Landau gauge $\sum_\mu \partial_\mu A_\mu(x) = 0$
- Archived by minimizing the functional $F(U) = -\sum_{x,\mu} \mathrm{ReTr}\left[g(x)U_\mu(x)g^\dagger(x+\hat{\mu})\right]$
- Implemented by over-relaxation algorithm

```
from pyquda import init
from pyquda.utils import io

init([1, 1, 1, 4], resource_path=".cache")
gauge = io.readQIOGauge("/public/ensemble/C24P29/be
rotation = gauge.fixingOVR2(1e-8, 10000, 4, 1.5)
```

Gauge fixing (WIP)

- 24^3x72, 1/4 GPUs
- Landau gauge, tolerance is 1e-8
- Use difference of functional (delta) as the criterion
- PyQUDA gives 6~9x performance

```
COULGAUGE: end: iter= 3038  tgfold= 0.830329390335408
COULOMB_GAUGEFIX: total time = 250.34483 secs
COULOMB_GAUGEFIX: ran successfully

3038 iter: functional=0.830329390335408, functional dif
PyQUDA INFO: fixingOVR2: 27.02609696611762 secs
```

```
COULGAUGE: end: iter= 3038  tgfold= 0.830329390335408
COULOMB_GAUGEFIX: total time = 85.354983 secs
COULOMB_GAUGEFIX: ran successfully

3038 iter: functional=2.989185805207468, functional dif
PyQUDA INFO: fixingOVR2: 13.994787024101242 secs
```

Distance preconditioning

And restart solver

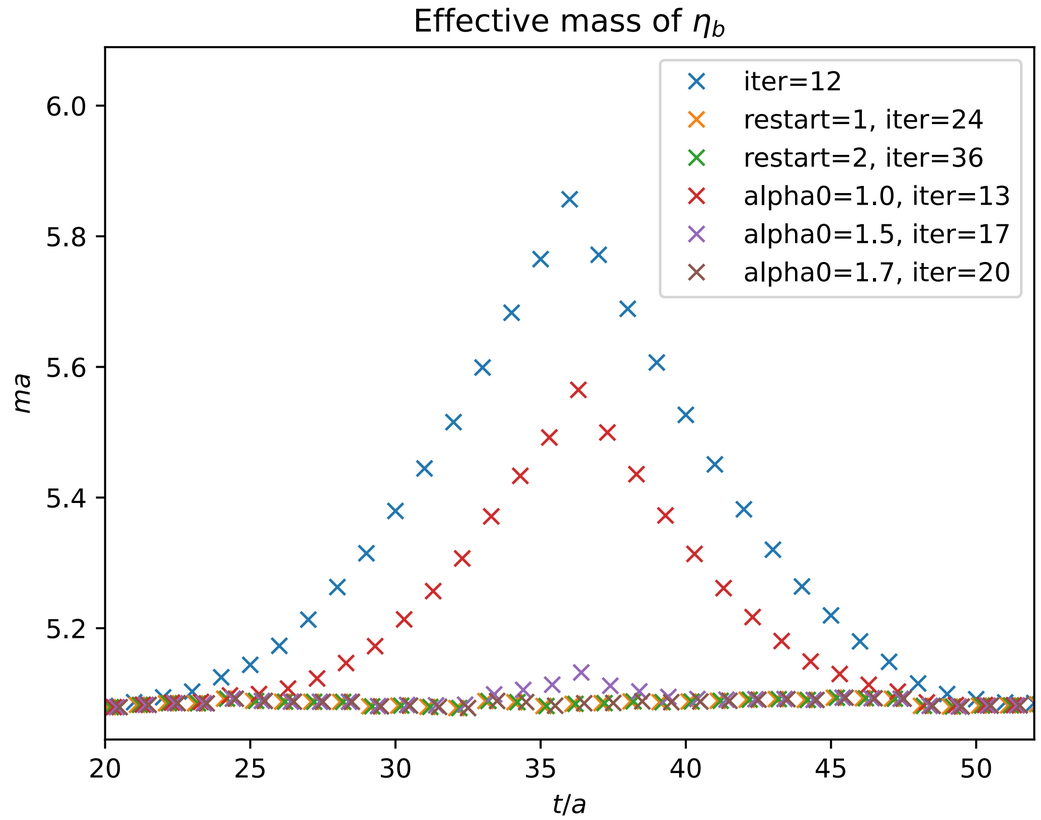
- For heavy quark, propagator at $t = \frac{T}{2}$ have amplitude $e^{-m\frac{T}{2}} < 2 \times 10^{-16}$
- $\mathscr{M}x = b \Rightarrow \alpha^{-1}(t)\mathscr{M}\alpha(t)x' = b'$, causes an extra coefficient in the hopping term, take $\alpha(t) = \cosh\left[m_0\left(t - \frac{T}{2}\right)\right]$
  - $\mathscr{M}'_{xy} = \sum_{\mu}(1-\gamma_\mu)\frac{\alpha(x_0)}{\alpha(y_0)}U_\mu(x)\delta_{x+\hat{\mu},y} + (1+\gamma_\mu)\frac{\alpha(x_0)}{\alpha(y_0)}U_\mu(x-\hat{\mu})(x-\hat{\mu})\delta_{x-\hat{\mu},y}$
- $\mathscr{M}x = b = \mathscr{M}x' + r \Rightarrow x = x' + \mathscr{M}^{-1}r$

Distance preconditioning

And restart solver

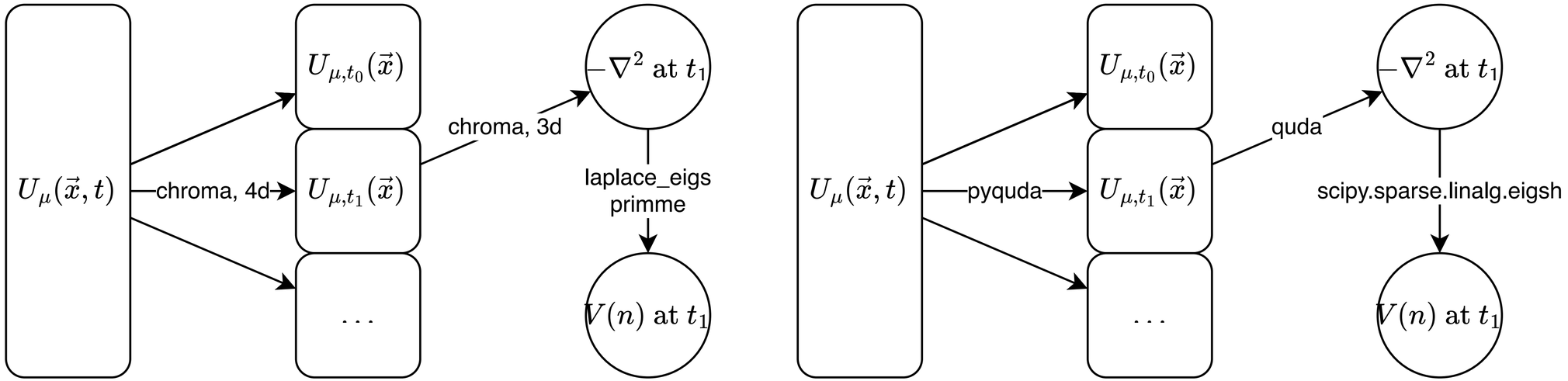
- They all uses more iterations to solve the precise propagator
- Have been used to generate charm and bottom quark propagator for CLQCD ensembles

```
setPrecision(sloppy=8)
u0, nu = 0.951479, 3.610
init(resource_path=".cache")
latt_info = core.LatticeInfo([24, 24, 24, 72], -1, 1 / nu)
gauge = io.readChromaQIOGauge("/public/ensemble/C24P29/beta6.20_mu-0.2770_ms-0.
dirac = core.getClover(latt_info, 5.5575, 1e-15, 1000, 1, nu / u0**3, (1 + nu)
dirac.loadGauge(gauge)
propag = core.invert(dirac, "wall", 0)
propag = core.invert(dirac, "wall", 0, restart=1)
dirac.invert_param.distance_pc_t0 = 0
dirac.invert_param.distance_pc_alpha0 = 1.0
propag = core.invert(dirac, "wall", 0)
```



Laplacian eigensystem

- $$\left(-\nabla^2\right)_{\vec{x},\vec{y}} = \left(-\sum_{i=1}^{3}\nabla_i^2\right)_{\vec{x},\vec{y}} = 6\delta_{\vec{x},\vec{y}} - \sum_{i=1}^{3}\left[U_i(\vec{x},t)\delta_{\vec{x}+\hat{i},\vec{y}} + U_i^\dagger(\vec{x}-\hat{i},t)\delta_{\vec{x}-\hat{i},\vec{y}}\right]$$



Laplacian eigensystem

- Use `cupyx.scipy.sparse.linalg.eigsh` to solve the Laplacian eigensystem
- Has been used to generate eigenvalues for distillation method by IHEP

```
import cupy as cp
from cupyx.scipy.sparse import linalg

from pyquda import enum_quda, init, core, quda
from pyquda.field import LatticeGauge, LaplaceLatticeInfo, LatticeSt
from pyquda.utils import io

t = 3
n_ev = 120
tol = 1e-9

init(resource_path=".cache")
gauge = io.readChromaQIOGauge("/public/ensemble/C24P29/beta6.20_mu-0
gauge.smearSTOUT(10, 0.12, 3)
Lx, Ly, Lz = gauge.latt_info.global_size[:3]
laplace_latt_info = LaplaceLatticeInfo([Lx, Ly, Lz, 1])
gauge_tmp = LatticeGauge(laplace_latt_info, core.cb2(gauge.lexico()[
gauge_tmp.pure_gauge.loadGauge(gauge_tmp)


def Laplacian(x):
    x = x.reshape(Lz * Ly * Lx * Nc, -1)
    ret = cp.empty_like(x, "<c16")
    for i in range(x.shape[1]):
        tmp = gauge_tmp.pure_gauge.laplace(LatticeStaggeredFermion(l
        ret[:, i] = tmp.data.reshape(Lz * Ly * Lx * Nc)
    return ret


A = linalg.LinearOperator((Lz * Ly * Lx * Nc, Lz * Ly * Lx * Nc), ma
evals, evecs = linalg.eigsh(A, n_ev, which="SA", tol=tol)
```

RHMC with HISQ

Compare with MILC

- $\mathscr{M}_i = \begin{pmatrix} 2m_i & D_{eo} \\ D_{oe} & 2m_i \end{pmatrix}$, and $M_i = 4m_i^2 - D_{eo}D_{oe}$ after applying the even-odd preconditioning
- $S_f = \sum_i \bar{\psi}\left(4m_i^2 - D_{eo}D_{oe}\right)^{-\frac{N_f}{4}}\psi = \sum_i \bar{\psi}\left(4m_i^2 - D_{eo}D_{eo}^{\dagger}\right)^{-\frac{N_f}{4}}\psi = \sum_i \bar{\psi}M_i^{-\frac{N_f}{4}}\psi$
- We can apply mass precondition
  $\det M_l^{\frac{2}{4}} = \det M_l^{\frac{2}{4}} \det M_s^{-\frac{1}{4}} \det M_s^{\frac{1}{4}} = \det\left(M_l^{\frac{2}{4}} M_s^{-\frac{1}{4}}\right) \det M_s^{\frac{1}{4}}$
- All rational approximation parameters are calculated based on $-D_{eo}D_{oe}$

RHMC with HISQ

Compare with MILC

- Following https://github.com/lattice/quda/wiki/MILC-with-QUDA to compile MILC with QUDA
- Symanzik tree-level improved gauge action
- HISQ fermion action
  - $N_f = 2 + 1 + 1$, $m_{u/d} = 0.0012$, $m_s = 0.0323$, $m_c = 0.432$, $\epsilon_c = -0.116203$
  - 5 pseudo fermions
- MILC 3G1F integrator, 24 steps with 0.02 evolution time per step

```python
from pyquda import init, getLogger, core
from pyquda.hmc import HMC, INT_3G1F
from pyquda.action import SymanzikTreeGauge, HISQFermion
from pyquda.utils.io import readMILCGauge, writeNPYGauge

beta, u_0 = 7.3, 0.880
tol, maxiter = 1e-6, 2500
start, stop, warm, save = 0, 2000, 500, 5
t = 0.48

init([1, 1, 1, 1], resource_path=".cache", enable_force_monitor=True)
latt_info = core.LatticeInfo([16, 16, 16, 32], t_boundary=-1, anisotropy=1.0)

monomials = [
    SymanzikTreeGauge(latt_info, beta, u_0),
    HISQFermion(latt_info, [0.0012, 0.0323, 0.2], [2, 1, -3], 1e-4, maxiter),
    HISQFermion(latt_info, 0.2, 1, 1e-6, maxiter),
    HISQFermion(latt_info, 0.2, 1, 1e-6, maxiter),
    HISQFermion(latt_info, 0.2, 1, 1e-6, maxiter),
    HISQFermion(latt_info, 0.432, 1, 1e-6, maxiter, naik_epsilon=-0.116203),
]

gauge = readMILCGauge("./s16t32_beta7.3_ml0.0012ms0.0323mc0.432.600")
gauge.toDevice()
gauge.projectSU3(2e-15)

hmc = HMC(latt_info, monomials, INT_3G1F(24))
hmc.initialize(10086, gauge)
```

```python
for i in range(start, stop):
    s = perf_counter()

    hmc.gaussMom()
    hmc.samplePhi()

    kinetic_old, potential_old = hmc.actionMom(), hmc.actionGauge()
    energy_old = kinetic_old + potential_old

    hmc.integrate(t, 2e-15)

    kinetic, potential = hmc.actionMom(), hmc.actionGauge()
    energy = kinetic + potential

    accept = hmc.accept(energy - energy_old)
    if accept or i < warm:
        hmc.saveGauge(gauge)
    else:
        hmc.loadGauge(gauge)
```

RHMC with HISQ

- MILC uses 44~45 secs to perform one trajectory

```
ACTION: g,h,f = 1.82801648619565e+06  -2.79039134329127e+02  9.83517170797286e+05  2.81125
DG = -4.480336e+02, DH = 4.842874e+02, DF = -3.608213e+01, D = 1.717018e-01
ACTIONTIME: time = 1.246947e+00
CHECK: delta S = 1.717018e-01
Aggregate time to do one trajectory 4.491103e+01
PLAQ:  1.7990744656305382   1.7998944201691263

ACTION: g,h,f = 1.82751687326134e+06  -1.49731148998004e+03  9.82513420529116e+05  2.80853
DG = -4.996129e+02, DH = 5.255448e+02, DF = -2.593101e+01, D = 8.236622e-04
ACTIONTIME: time = 1.374373e+00
CHECK: delta S = 8.236622e-04
Aggregate time to do one trajectory 4.467800e+01
PLAQ:  1.7992539348297540   1.8004709553252449

ACTION: g,h,f = 1.83070351193743e+06  -2.65918083988630e+03  9.83150381103455e+05  2.81119
DG = 3.186639e+03, DH = -2.961505e+03, DF = -2.251469e+02, D = -1.335321e-02
ACTIONTIME: time = 1.459933e+00
CHECK: delta S = -1.335321e-02
Aggregate time to do one trajectory 4.635962e+01
PLAQ:  1.7981020856842045   1.7974236640743180
```

RHMC with HISQ

- PyQUDA uses 32~33 secs to perform one trajectory

```
PyQUDA INFO: Trajectory 1:
Plaquette = [0.5996775235815205, 0.5998103142086956, 0.5995447329543454]
P_old = -2584172.375699158, K_old = 873.1068479729922
P = -2583794.685546202, K = 495.4612721727095
Delta_P = 377.69015295570716, Delta_K = -377.6455758002827
Delta_E = 0.04457715526223183
acceptance rate = 95.64%
accept? True
warmup? True
HMC time = 32.179 secs

PyQUDA INFO: Trajectory 2:
Plaquette = [0.6001975440988594, 0.6002479335874091, 0.6001471546103097]
P_old = -2584016.045943022, K_old = -1122.301297210296
P = -2586511.95777814, K = 1373.6153233072105
Delta_P = -2495.911835118197, Delta_K = 2495.9166205175065
Delta_E = 0.004785398952662945
acceptance rate = 99.52%
accept? True
warmup? True
HMC time = 33.096 secs

PyQUDA INFO: Trajectory 3:
Plaquette = [0.5995482764517526, 0.5997016962003643, 0.5993948567031409]
P_old = -2586327.3927370217, K_old = -459.1197368338825
P = -2583505.2341964496, K = -3281.247696418602
Delta_P = 2822.15854057204, Delta_K = -2822.1279595847195
Delta_E = 0.030580987222492695
acceptance rate = 96.99%
accept? True
warmup? True
HMC time = 32.594 secs
```

RHMC with HISQ

- PyQUDA allocate more fields on the GPU memory
- `computeKSLinkQuda` and `updateGaugeFieldQuda` benefits most from less/faster data transfer between host and device
- Will be used to generate some HISQ ensembles in the future

```
computeKSLinkQuda Total time =    11.020 secs
           download     =     2.345 secs ( 21.281%),       with      378 calls at 6.204e+03 us per call
             upload     =     8.400 secs ( 76.228%),       with      756 calls at 1.111e+04 us per call
               init     =     0.003 secs (  0.028%),       with     3780 calls at 8.135e-01 us per call
            compute     =     0.116 secs (  1.049%),       with     1512 calls at 7.643e+01 us per call
              comms     =     0.114 secs (  1.033%),       with      378 calls at 3.012e+02 us per call
               free     =     0.001 secs (  0.008%),       with     2268 calls at 3.695e-01 us per call
  total accounted       =    10.979 secs ( 99.626%)
  total missing         =     0.041 secs (  0.374%)

updateGaugeFieldQuda Total time =    38.938 secs
             download     =     1.312 secs (  3.369%),       with      180 calls at 7.289e+03 us per call
               upload     =    37.603 secs ( 96.570%),       with      540 calls at 6.963e+04 us per call
                 init     =     0.001 secs (  0.003%),       with     2700 calls at 4.278e-01 us per call
              compute     =     0.004 secs (  0.010%),       with      540 calls at 6.865e+00 us per call
                 free     =     0.000 secs (  0.001%),       with     1080 calls at 3.843e-01 us per call
    total accounted       =    38.920 secs ( 99.953%)
    total missing         =     0.018 secs (  0.047%)
```

MILC

```
Device memory used = 1319.9 MiB
```

```
computeKSLinkQuda Total time =     4.956 secs
           download     =     0.531 secs ( 10.704%),       with      405 calls at 1.310e+03 us per call
             upload     =     3.983 secs ( 80.364%),       with      795 calls at 5.010e+03 us per call
               init     =     0.003 secs (  0.055%),       with     7230 calls at 3.743e-01 us per call
            compute     =     0.299 secs (  6.034%),       with     1620 calls at 1.846e+02 us per call
              comms     =     0.120 secs (  2.418%),       with      405 calls at 2.959e+02 us per call
               free     =     0.001 secs (  0.014%),       with     2430 calls at 2.905e-01 us per call
  total accounted       =     4.936 secs ( 99.589%)
  total missing         =     0.020 secs (  0.411%)

  updateGaugeFieldQuda Total time =     0.516 secs
                 init     =     0.000 secs (  0.092%),       with     2160 calls at 2.185e-01 us per call
              compute     =     0.003 secs (  0.497%),       with      540 calls at 4.748e+00 us per call
                 free     =     0.001 secs (  0.130%),       with     2160 calls at 3.111e-01 us per call
      total accounted       =     0.004 secs (  0.719%)
      total missing         =     0.512 secs ( 99.281%)
```

PyQUDA

```
Device memory used = 1561.4 MiB
```

16^3x32, 1 GPU, from free gauge

```
PyQUDA INFO: Trajectory 1:
Plaquette = [0.7204612523812091, 0.7207429023061924, 0.7201796024562256]
P_old = -4413033.078023404, K_old = 873.1068479729922
P = -3159670.7098042895, K = -1252501.346048676
Delta_P = 1253362.3682191144, Delta_K = -1253374.452896649
Delta_E = -12.0846775341779
acceptance rate = 100.00%
accept? True
warmup? True
HMC time = 20.466 secs
ACTION: g,h,f = 1.25008479351193e+06  -1.25273398691584e+06  9.85427043212037e+05  9.82777849808133e+05
DG = 1.250085e+06, DH = -1.251971e+06, DF = 1.873790e+03, D = -1.207656e+01
ACTIONTIME: time = 5.672421e-01
CHECK: delta S = -1.207656e+01
Aggregate time to do one trajectory 3.154932e+01
PLAQ:  2.1624554169347561   2.1620717287765316
```

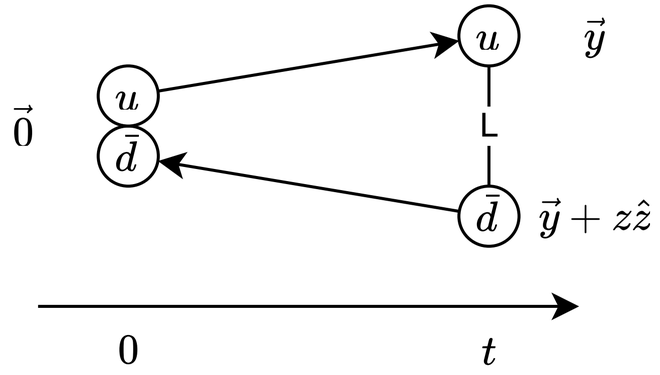
32^3x64, 4 GPUs, from free gauge

```
PyQUDA INFO: Trajectory 2:
Plaquette = [0.6566182728405672, 0.6565451388498317, 0.6566914068313026]
P_old = -50596747.02112688, K_old = 6859.600398017135
P = -45604715.844899595, K = -4985184.076069808
Delta_P = 4992031.176227286, Delta_K = -4992043.676467825
Delta_E = -12.500240541994572
acceptance rate = 100.00%
accept? True
warmup? True
HMC time = 96.053 secs
ACTION: g,h,f = 2.49802598408089e+07  -4.99166446458193e+06  1.57337938838013e+07  3.57223892600283e+07
DG = 4.977869e+06, DH = -4.988442e+06, DF = 1.056109e+04, D = -1.228314e+01
ACTIONTIME: time = 3.209237e+00
CHECK: delta S = -1.228314e+01
Aggregate time to do one trajectory 1.541774e+02
PLAQ:   1.9697942248617086      1.9700082878304221
```

Pion distribution amplitude

An example

- $C_2(z,\vec{p},t) = \sum_{\vec{y}} e^{-i\vec{p}\cdot\vec{y}} \mathrm{Tr}\left[\gamma_5 S^\dagger(\vec{y},t;0,0)\gamma_5\gamma_4\gamma_5 L(\vec{y},t;\vec{y}+z\hat{z},t) S(\vec{y}+z\hat{z},t;0,0)\gamma_5\right]$
- The Wilson link $L(\vec{y},t;\vec{y}+z\hat{z},t) = U_z(\vec{y})U_z(\vec{y}+\hat{z})U_z(\vec{y}+2\hat{z})\cdots U_z(\vec{y}+(z-1)\hat{z})$
- Implemented by multiple $\psi'(x) = U_\mu(x)\psi(x+\hat{\mu})$
- $C_2(z,\vec{p},t) = \sum_{\vec{y}} e^{-i\vec{p}\cdot\vec{y}} \mathrm{Tr}\left[\gamma_5 S^\dagger(\vec{y},t;0,0)\gamma_5\gamma_4\gamma_5 S^{(z)}(\vec{y},t;0,0)\gamma_5\right]$



Pion distribution amplitude

- Get fermion matrix for inversion
- Get gauge from file, and apply APE and stout smearing to it
- Get momentum phase
- Prepare gauge for momentum smearing

```
init([1, 1, 1, 4], resource_path=".cache")
latt_info = LatticeInfo([24, 24, 24, 72], -1, 1.0)
dirac = core.getDirac(latt_info, -0.2400, 1e-8, 1000, 1.0, 1.160
gauge_ori = io.readChromaQIOGauge("/public/ensemble/C24P29/beta6
gauge_ori.toDevice()
gauge_prop = gauge_ori.copy()
gauge_prop.stoutSmear(1, 0.125, 4)
gauge_smear = gauge_ori.copy()
gauge_smear.smearAPE(ape_times, 2.5, 3)
dirac.loadGauge(gauge_prop)
pion = cp.zeros((len(z_list), len(t_src_list), latt_info.Lt), "<

mom_smear = [0, 0, mom_smear_z]
mom_phase = phase.MomentumPhase(latt_info).getPhase([0, 0, -mom_

print("初始化与参数设置成功")
for i in range(3):
    k = 2 * cp.pi * mom_smear[i] / latt_info.global_size[i]
    gauge_smear.data[i] *= cp.exp(-1j * k)
```

Pion distribution amplitude

- Get point source and then apply gaussian smearing to get the shell source
- Get the quark propagator
- Contract and shift the propagator for some times

```
for t_idx, t_src in enumerate(t_src_list):
    point_source = source.source12(latt_info, "point", [6, 20
    shell_source = source.gaussian12(point_source, gauge_smea
    dirac.loadGauge(gauge_prop, True)
    propag = core.invertPropagator(dirac, shell_source)

    propag_shift = propag.copy()
    gauge_ori.pure_gauge.loadGauge(gauge_ori)
    print("开始收缩")
    for z_iz in z_list:

        pion[z_iz, t_idx] += contract(
            "wtzyx,wtzyxjiba,jk,wtzyxklba,li->t",
            mom_phase,
            propag.data.conj(),
            G5 @ G4G5,
            propag_shift.data,
            G5 @ G5,
        )

        for spin in range(4):
            for color in range(3):
                fermion = propag_shift.getFermion(spin, color
                fermion_shift = gauge_ori.pure_gauge.covDev(f
                propag_shift.setFermion(fermion_shift, spin,

    print(f"{t_src}位置点源计算成功")
```

Summary and outlook

- PyQUDA and some derived projects have been used in recent works
  - Thanks to all developers and users
- Better gauge/quark field shift and matrix product operation
- Fermion force for stout smeared fermion action
  - Should speedup HMC/RHMC with clover action
- Framework for C/C++/CUDA plugins
  - Should speedup special contractions like baryon

Thanks for your attention!