

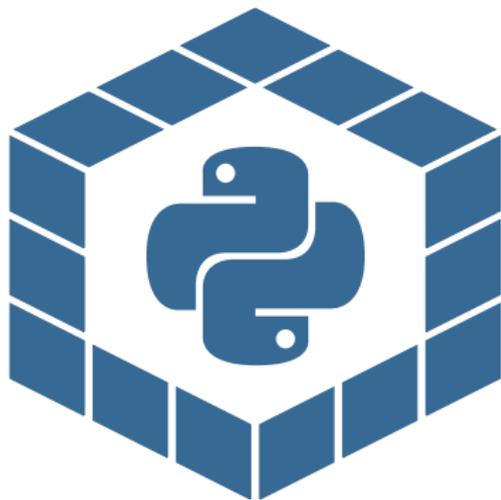
Grid Python Toolkit (GPT)

Christoph Lehner
(Uni Regensburg)

<https://github.com/lehner/gpt>

August 8, 2022 - Lattice 2022

Grid Python Toolkit (GPT)



<https://github.com/lehner/gpt>

- ▶ A toolkit for **lattice QCD** and related theories as well as **QIS** (a parallel digital quantum computing simulator) and **Machine Learning**
- ▶ Python frontend, C++ backend
- ▶ Built on Grid data parallelism (MPI, OpenMP, SIMD, and SIMT)

Code co-authors:

- ▶ M. Bruno
- ▶ D. Richtmann
- ▶ T. Blum
- ▶ S. Bürger
- ▶ P. Georg
- ▶ L. Jin
- ▶ D. Knüttel
- ▶ S. Meinel
- ▶ M. Schlemmer
- ▶ S. Solbrig
- ▶ T. Wettig
- ▶ T. Wurm

Guiding principles:

- ▶ **Performance Portability**

common Grid-based framework for current and future (exascale) architectures

- ▶ **Modularity / Composability**

build up from modular high-performance components, several layers of composability, “composition over parametrization”

Layout and dependencies

Python script / Jupyter notebook

gpt (Python)

- Defines data types and objects (group structures etc.)
- Expression engine (linear algebra)
- Algorithms (Solver, Eigensystem, ...)
- File formats
- Stencils / global data transfers
- QCD, QIS, ML subsystems

cgpt (Python library written in C++)

- Global data transfer system (gpt creates pattern, cgpt optimizes data movement plan)
- Virtual lattices (tensors built from multiple Grid tensors)
- Optimized blocking, linear algebra, and Dirac operators
- Vectorized ranlux-like pRNG (parallel seed through 3xSHA256)

Grid

Eigen

FFTW

The QCD module

Example: Load QCD gauge configuration and test unitarity

```
In [1]: import gpt as g
        U = g.load("ckpoint_lat.IEEE64BIG.1100")
        for mu in range(4):
            g.message("SU3 - Defect: ", g.norm2(U[mu] * g.adj(U[mu]) - g.identity(U[mu])))
GPT :      1.039211 s : SU3 - Defect:  3.345168726568745e-26
GPT :      1.094156 s : SU3 - Defect:  3.3476154954606903e-26
GPT :      1.146703 s : SU3 - Defect:  3.342180010368529e-26
GPT :      1.199097 s : SU3 - Defect:  3.3423193715873574e-26
```

Here: expression first parsed to a tree in Python (gpt), forwarded as abstract expression to C++ library (cgpt) for evaluation

Example: create a pion propagator on a random gauge field

```
# double-precision 8^4 grid
grid = g.grid([8,8,8,8], g.double)

# pRNG
rng = g.random("seed text")

# random gauge field
U = g.qcd.gauge.random(grid, rng)

# Mobius domain-wall fermion
fermion = g.qcd.fermion.mobius(U, mass=0.1, M5=1.8, b=1.0, c=0.0, Ls=24,
                               boundary_phases=[1,1,1,-1])

# Short-cuts
inv = g.algorithms.inverter
pc = g.qcd.fermion.preconditioner

# even-odd-preconditioned CG solver
slv_5d = inv.preconditioned(pc.eo2_ne(), inv.cg(eps = 1e-4, maxiter = 1000))

# Abstract fermion propagator using this solver
fermion_propagator = fermion.propagator(slv_5d)

# Create point source
src = g.mspincolor(U[0].grid)
g.create.point(src, [0, 0, 0, 0])

# Solve propagator on 12 spin-color components
prop = g( fermion_propagator * src )

# Pion correlator
g.message(g.slice(g.trace(prop * g.adj(prop)), 3))
```

Example: solvers are modular and can be mixed

General design principle: use modularity of python code instead of large number of parameters to configure solvers/algorithms;

Python can also be used in configuration files

```
# Create an coarse-grid deflated, even-odd preconditioned CG inverter  
# (eig is a previously loaded multi-grid eigensystem)  
sloppy_light_inverter = g.algorithms.inverter.preconditioned(  
    g.qcd.fermion.preconditioner.eo1_ne(parity=g.odd),  
    g.algorithms.inverter.sequence(  
        g.algorithms.inverter.coarse_deflate(  
            eig[1],  
            eig[0],  
            eig[2],  
            block=200,  
        ),  
        g.algorithms.inverter.split(  
            g.algorithms.inverter.cg({"eps": 1e-8, "maxiter": 200}),  
            mpi_split=[1,1,1,1],  
        ),  
    ),  
)
```

Further example: Multi-Grid solver

```
def find_near_null_vectors(w, cgrid):
    slv = i.fgmres(eps=1e-3, maxiter=50, restartlen=25, checkres=False)(w)
    basis = g.orthonormalize(
        rng.cnormal([g.lattice(w.grid[0]), w.otype[0]] for i in range(30)))
    )
    null = g.lattice(basis[0])
    null[:] = 0
    for b in basis:
        slv(b, null)
    g.qcd.fermion.coarse.split_chiral(basis)
    bm = g.block.map(cgrid, basis)
    bm.orthonormalize()
    bm.check_orthogonality()
    return basis

mg_setup_3lvl = i.multi_grid_setup(
    block_size=[[2, 2, 2, 2], [2, 1, 1, 1]], projector=find_near_null_vectors
)

wrapper_solver = i.fgmres(
    {"eps": 1e-1, "maxiter": 10, "restartlen": 5, "checkres": False}
)
smooth_solver = i.fgmres(
    {"eps": 1e-14, "maxiter": 8, "restartlen": 4, "checkres": False}
)
coarsest_solver = i.fgmres(
    {"eps": 5e-2, "maxiter": 50, "restartlen": 25, "checkres": False}
)

mg_3lvl_kcycle = i.sequence(
    i.coarse_grid(
        wrapper_solver.modified(
            prec=i.sequence(
                i.coarse_grid(coarsest_solver, *mg_setup_3lvl[1]), smooth_solver
            )
        ),
        *mg_setup_3lvl[0],
    ),
    smooth_solver,
)
```

All algorithms implemented in Python – Example: Euler-Langevin stochastic DGL integrator

```
21
22 class langevin_euler:
23     @g.params_convention(epsilon=0.01)
24     def __init__(self, rng, params):
25         self.rng = rng
26         self.eps = params["epsilon"]
27
28     def __call__(self, fields, action):
29         gr = action.gradient(fields, fields)
30         for d, f in zip(gr, fields):
31             f @= g.group.compose(
32                 -d * self.eps
33                 + self.rng.normal_element(g.lattice(d)) * (self.eps * 2.0) ** 0.5,
34                 f,
35             )
36
```

Implemented algorithms:

- ▶ BiCGSTAB, CG, CAGCR, FGCR, FGMRES, MR solvers
- ▶ Multi-grid, split-grid, mixed-precision, and defect-correcting solver combinations
- ▶ Coarse and fine-grid deflation
- ▶ Implicitly restarted Arnoldi and Lanczos, power iteration
- ▶ Chebyshev polynomials
- ▶ All-to-all vector generation
- ▶ SAP and even-odd preconditioners
- ▶ MSPCG (additive Schwarz)
- ▶ Gradient descent and non-linear CG optimizers
- ▶ Runge-Kutta integrators, Wilson flow
- ▶ Fourier acceleration
- ▶ Coulomb and Landau gauge fixing
- ▶ Domain-wall-overlap transformation and MADWF
- ▶ Symplectic integrators (leapfrog, OMF2, and OMF4)
- ▶ Markov: Metropolis, heatbath, Langevin, (DD-)HMC

Implemented fermion actions:

- ▶ Domain-wall fermions: Mobius and zMobius
- ▶ Wilson-clover fermions both isotropic and anisotropic (RHQ/Fermilab actions); Open boundary conditions available

Example: stout-smearred heavy-quark Mobius DWF

```
# load configuration
U = g.load(config)
grid = U[0].grid

# smeared gauge link
U_stout = U
for n in range(3):
    U_stout = g.qcd.gauge.smear.stout(U_stout, rho=0.1)

fermion_exact = g.qcd.fermion.mobius(U_stout, {
    "mass": 0.6,
    "M5": 1.0,
    "b": 1.5,
    "c": 0.5,
    "Ls": 12,
    "boundary_phases": [1.0, 1.0, 1.0, -1.0],
})
```

Performance

Benchmark results committed to github

<https://github.com/lehner/gpt/tree/master/benchmarks/reference>

master [gpt / benchmarks / reference /](#)

lehner supermuc-ng timing		on Apr 1	History
..			
bnl_knl		8 months ago	
juron		8 months ago	
juwels_booster		2 months ago	
lrz_supermuc_ng		last month	
qpace3		8 months ago	
qpace4		4 months ago	
stampede2_knl		6 months ago	
summit		8 months ago	

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.20GHz, Totsu interconnect 0, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	Summit - IBM Power System AC922, IBM POWER9 22C 3.070GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Orca, Ridge National Laboratory United States	2,416,592	148,600.0	200,794.9	10,096
3	Sierra - IBM Power System AC922, IBM POWER9 22C 3.10GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LNL United States	1,572,480	94,640.0	125,712.0	7,638
4	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,640.0	79,215.0	2,646
6	Tianhe-2A - TH-1V8-FEP Cluster, Intel Xeon ES-2692V2 12C 2.20GHz, TH Express-2, Mellanox, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
7	JUWELS Booster Module - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR Infiniband/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich IPZ.J Germany	449,280	44,120.0	70,980.0	1,764

Results available for GPU and CPU architectures. In the following, focus on Juwels booster (NVIDIA A100) and QPace4 (A64FX, same as Fugaku).

Juwels Booster (node has $4 \times$ A100-40GB): Single-node domain-wall fermion \not{D} operator

```
51 =====
52           Initialized GPT
53           Copyright (C) 2020 Christoph Lehner
54 =====
55 GPT :      1.543473 s :
56           : DWF Dslash Benchmark with
57           :   fdimensions : [64, 32, 32, 32]
58           :   precision   : single
59           :   Ls           : 12
60           :
61 GPT :      7.958636 s : 1000 applications of Dhop
62           :   Time to complete           : 2.93 s
63           :   Total performance           : 11325.46 GFlops/s
64           :   Effective memory bandwidth : 7824.86 GB/s
65 GPT :      7.959499 s :
66           : DWF Dslash Benchmark with
67           :   fdimensions : [64, 32, 32, 32]
68           :   precision   : double
69           :   Ls           : 12
70           :
71 GPT :     17.420620 s : 1000 applications of Dhop
72           :   Time to complete           : 5.78 s
73           :   Total performance           : 5749.77 GFlops/s
74           :   Effective memory bandwidth : 7945.14 GB/s
75 =====
76           Finalized GPT
77 =====
```

Compare to HBM bandwidth of 1,555 GB/s per GPU

QPACE4 (node has one A64FX): Single-node domain-wall fermion \mathcal{D} operator

```
108 =====
109           Initialized GPT
110           Copyright (C) 2020 Christoph Lehner
111 =====
112 GPT :      0.265714 s :
113           : DWF Dslash Benchmark with
114           :   fdimensions : [24, 24, 24, 24]
115           :   precision  : single
116           :   Ls         : 8
117           :
118 GPT :      20.218240 s : 1000 applications of Dhop
119           :   Time to complete           : 3.67 s
120           :   Total performance          : 954.90 GFlops/s
121           :   Effective memory bandwidth : 677.11 GB/s
122 GPT :      20.218842 s :
123           : DWF Dslash Benchmark with
124           :   fdimensions : [24, 24, 24, 24]
125           :   precision  : double
126           :   Ls         : 8
127           :
128 GPT :      45.245379 s : 1000 applications of Dhop
129           :   Time to complete           : 7.36 s
130           :   Total performance          : 475.80 GFlops/s
131           :   Effective memory bandwidth : 674.77 GB/s
132 =====
133           Finalized GPT
134 =====
```

Compare to HBM bandwidth of 1,000 GB/s per A64FX

Juwels Booster (node has 4× A100-40GB): Single-node site-local matrix products

```
=====
      Initialized GPT
      Copyright (C) 2020 Christoph Lehner
      =====
GPT :   1.589357 s :
      : Matrix Multiply Benchmark with
      :   fdimensions : [48, 48, 48, 128]
      :   precision   : single
      :
GPT :  10.985099 s : 10 matrix_multiply
      :   Object type           : ot_matrix_color(3)
      :   Time to complete      : 0.0058 s
      :   Effective memory bandwidth : 5271.36 GB/s
      :
GPT :  16.689329 s : 10 matrix_multiply
      :   Object type           : ot_matrix_spin(4)
      :   Time to complete      : 0.01 s
      :   Effective memory bandwidth : 5333.21 GB/s
      :
GPT :  62.892583 s : 10 matrix_multiply
      :   Object type           : ot_matrix_spin_color(4,3)
      :   Time to complete      : 0.097 s
      :   Effective memory bandwidth : 5057.37 GB/s
      :
GPT :  62.262581 s :
      : Matrix Multiply Benchmark with
      :   fdimensions : [48, 48, 48, 128]
      :   precision   : double
      :
GPT :  72.003471 s : 10 matrix_multiply
      :   Object type           : ot_matrix_color(3)
      :   Time to complete      : 0.012 s
      :   Effective memory bandwidth : 5264.01 GB/s
      :
GPT :  78.174681 s : 10 matrix_multiply
      :   Object type           : ot_matrix_spin(4)
      :   Time to complete      : 0.02 s
      :   Effective memory bandwidth : 5439.91 GB/s
      :
GPT : 128.232979 s : 10 matrix_multiply
      :   Object type           : ot_matrix_spin_color(4,3)
      :   Time to complete      : 0.22 s
      :   Effective memory bandwidth : 4416.45 GB/s
      :
      =====
                        Finalized GPT
      =====
```

Compare to HBM bandwidth of 1,555 GB/s per GPU

Juwels Booster (node has $4 \times$ A100-40GB): Inner product (reduction)

```
GPT :      28.406798 s : 100 rank_inner_product
      :      Object type           : ot_vector_singlet(12)
      :      Block                  : 4 x 4
      :      Data resides in       : accelerator
      :      Performed on         : accelerator
      :      Time to complete      : 0.13 s
      :      Effective memory bandwidth : 4827.16 GB/s
      :
      :      rip: timing: unprofiled      = 0.000000e+00 s (= 0.00 %)
      : rip: timing: rip: view          = 9.706020e-04 s (= 0.70 %)
      : rip: timing: rip: loop          = 1.369879e-01 s (= 99.30 %)
      : rip: timing: total              = 1.379585e-01 s (= 100.00 %)
      :
```

Compare to HBM bandwidth of 1,555 GB/s per GPU

Performance summary

Machine	Operation	Performance	Bandwidth
Booster	\mathcal{D}	12 TF/s	7.8 TB/s
Booster	ColorMatrix \times		5.2 TB/s
Booster	SpinColorMatrix \times		5.1 TB/s
Booster	SpinColorVector $\langle \cdot, \cdot \rangle$		4.8 TB/s
QPace4	\mathcal{D}	0.95 TF/s	0.68 TB/s
SuperMUC-NG	\mathcal{D}	0.72 TF/s	0.51 TB/s

Single-node SP performance of Wilson \mathcal{D} and linear algebra on Jewels Booster (4xA100, HBM BW 1.6 TB/s per A100), Qpace4 (A64FX, HBM BW of 1 TB/s per node), and the SuperMUC-NG (Skylake 8174). The \mathcal{D} performance is inherited from Grid, the linear algebra performance is based on cgpt.

Example applications

RBC ensemble generation

(the generating GPT scripts are linked below; around 200 lines of Python script each)

ID	a^{-1}/GeV	N_f	$L^3 \times T \times L_s$	$b + c$	$m_{\text{res}} \times 10^4$	m_π/MeV	m_K/MeV	m_{D_s}/GeV	$m_\pi L$	Code
48l	1.73	2+1	$48^3 \times 96 \times 24$	2	6.1	139	499	–	3.87	CPS
64l	2.35	2+1	$64^3 \times 128 \times 12$	2	3.1	139	507	–	3.77	CPS
96l	2.69	2+1	$96^3 \times 192 \times 12$	2	2.3	132	486	–	4.70	CPS
1	1.73	2+1	$32^3 \times 64 \times 24$	2	6.3	208	513	–	3.85	GPT script
2	1.73	2+1	$24^3 \times 48 \times 32$	2	4.6	284	534	–	3.96	GPT script
3	1.73	2+1	$32^3 \times 64 \times 24$	2	6.5	210	597	–	3.88	GPT script
4	1.74	2+1	$24^3 \times 48 \times 24$	2	6.2	279	534	–	3.84	GPT script
5	1.75	2+1+1	$24^3 \times 48 \times 24$	2	6.7	280	536	1.99	3.84	GPT script
6	1.75	2+1+1	$24^3 \times 48 \times 24$	2	6.7	280	536	1.5	3.84	GPT script
7	1.76	2+1+1	$24^3 \times 48 \times 24$	2	7.9	284	540	1.39	3.88	GPT script
8	2.37	2+1+1	$32^3 \times 64 \times 12$	2	3.0	280	536	1.99	3.88	GPT script
9	2.37	2+1	$32^3 \times 64 \times 12$	2	3.0	281	535	–	3.80	GPT script
A	1.76	2+1	$24^3 \times 48 \times 8$	2	41.5	303	548	–	4.15	GPT script
B	1.73	2+1	$32^3 \times 64 \times 24$	2	6.1	139	499	–	2.58	GPT script
C	1.73	2+1	$64^3 \times 128 \times 24$	2	6.1	139	499	–	5.16	GPT script
D	1.74	2+1	$32^3 \times 64 \times 24$	2	6.2	279	534	–	5.12	GPT script
E	3.50	2+1	$48^3 \times 192 \times 12$, openBC	2	1.4	280	535	–	3.87	GPT script

Further examples

- ▶ QED corrections to $g-2$ HVP and tau decays (Tuesday talk Mattia Bruno)
- ▶ Ensemble parameters and $g-2$ HVP (Friday talk C.L.)
- ▶ $g-2$ HLbL project of RBC/UKQCD (combined with QLattice)

Also applied by BNL and Bielefeld groups for ongoing projects.

Teaching

LGT lecture based on interactive GPT notebooks

(link to lecture)

Chapters:

- Chapter 1: path integral formulation of quantum mechanics ([Jupyter Notebook](#), [PDF](#))
- Chapter 2: Markov Chain Monte Carlo (MCMC) methods ([Jupyter Notebook](#), [PDF](#), last update 09.11. 10:14)
- Chapter 3: statistics of continuous variables ([Jupyter Notebook](#), [PDF](#), last update 07.11. 21:00)
- Chapter 4: scalar field theory on a lattice ([Jupyter Notebook](#), [PDF](#), last update 02.12. 10:45)
- Chapter 5: symmetries of fundamental field theories ([Jupyter Notebook](#), [PDF](#), last update 21.11. 18:30)
- Chapter 6: gauge theories on a lattice ([Jupyter Notebook](#), [PDF](#), last update 28.11. 16:30)
- Chapter 7: static quark potential and spectrum of pure QCD ([Jupyter Notebook](#), [PDF](#), last update 03.12. 10:11)
- Chapter 8: strong coupling expansion ([Jupyter Notebook](#), [PDF](#), last update 05.12. 21:11)
- Chapter 9: continuum limit and phase transitions ([Jupyter Notebook](#), [PDF](#), last update 13.12. 10:14)
- Chapter 10: fermions on a lattice ([Jupyter Notebook](#), [PDF](#), last update 19.12. 18:04)
- Chapter 11: symmetries of the Wilson action ([Jupyter Notebook](#), [PDF](#), last update 10.01. 09:35)
- Chapter 12: chiral symmetry on a lattice ([Jupyter Notebook](#), [PDF](#), last update 24.01. 10:00)
- Chapter 13: Hybrid Monte Carlo (HMC) ([Jupyter Notebook](#), [PDF](#), last update 28.01. 09:46)
- Chapter 14: Hadron spectroscopy ([Jupyter Notebook](#), [PDF](#), last update 04.02. 09:56)

The machine learning module

Example: train simple feed-forward network

```
In [ ]: import gpt as g
        grid = g.grid([4, 4, 4], g.double)
        rng = g.random("test")

        # network and training data
        n = g.ml.network.feed_forward([g.ml.layer.nearest_neighbor(grid)] * 2)
        training_input = [rng.uniform_real(g.complex(grid)) for i in range(2)]
        training_output = [rng.uniform_real(g.complex(grid)) for i in range(2)]

        # cost functional
        c = n.cost(training_input, training_output)

        # train network
        W = n.random_weights(rng)
        gd = g.algorithms.optimize.gradient_descent
        gd(maxiter=4000, eps=1e-4, step=0.2)(c)(W, W)
```

The quantum computing module

Example: create and measure a 5-qubit bell state

```
import gpt as g
from gpt.qis.gate import *

rng = g.random("qis_test")

# initial state with 5 qubits, stored in double-precision
st = g.qis.backends.dynamic.state(rng, 5, precision=g.double)

g.message("Initial state:\n",st)

# prepare Bell-type state
st = (H(0) | CNOT(0,1) | CNOT(0,2) | CNOT(0,3) | CNOT(0,4)) * st

g.message("Bell-type state:\n",st)

# measure
st = M() * st

g.message("After single measurement:\n",st)
g.message("Classically measured bits:\n",st.classical_bit)
```

```
GPT :      197.943668 s : Initial state:
      :      + (1+0j) |00000>
GPT :      197.949198 s : Bell-type state:
      :      + (0.7071067811865475+0j) |00000>
      :      + (0.7071067811865475+0j) |11111>
GPT :      197.951478 s : After single measurement:
      :      + (1+0j) |11111>
GPT :      197.952545 s : Classically measured bits:
      :      [1, 1, 1, 1, 1]
```

How to use GPT?

<https://github.com/lehner/gpt>

Quick Start

The fastest way to try GPT is to install [Docker](#), start a [Jupyter](#) notebook server with the latest GPT version by running

```
docker run --rm -p 8888:8888 gptdev/notebook
```

and then open the shown link `http://127.0.0.1:8888/?token=<token>` in a browser. You should see the tutorials folder pre-installed.

The docker images are automatically generated for each version that passes the CI interface.

CI currently has test coverage of 96%, running on each pushed commit.

Thank you