Twisted mass ensemble generation on GPU machines

Bartosz Kostrzewa

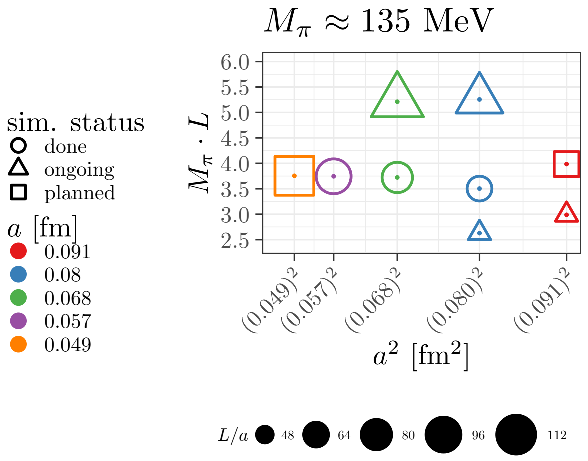
M. Garofalo, S. Romiti

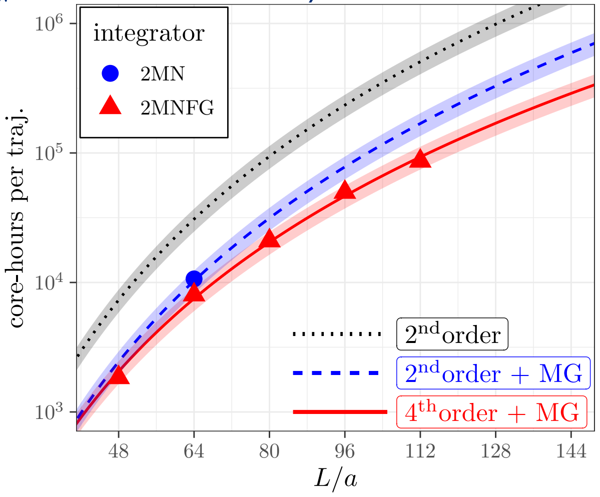
S. Bacchio, J. Finkenrath, F. Pittler

High Performance Computing & Analytics Lab,
Rheinische Friedrich-Wilhelms-Universität Bonn

August 8th 2022, 39th International Symposium on Lattice Field Theory, Bonn

The cost of ensemble generation (at phys. M_π on CPU machines)





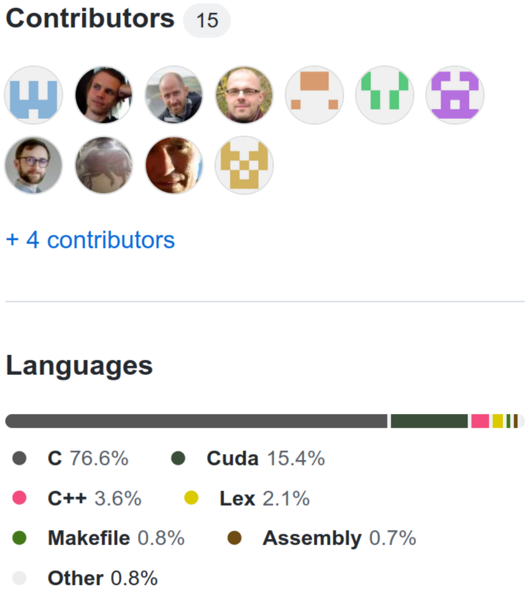
- State-of-the-art integrator & solvers $\rightarrow$ cost scales like $(L/a)^{9/2}$ at (roughly) constant acceptance
- need several further ensembles at larger $M_\pi \cdot L$
  - both at the finest and the coarsest lattice spacings
    - ★ more statistics needed due to autocorrelations (critical slowing down and pion mass splitting)
- cost $\mathcal{O}(10^9)$ core-hours & real time per trajectory ≥ 6 hours at this stage
- **Absolutely need GPU implementations of everything**

The tmLQCD software suite

[10.1016/j.cpc.2009.05.016, 10.22323/1.187.0416, 10.22323/1.187.0414, gh.com/etmc/tmLQCD]

https://github.com/etmc/tmLQCD

- current HMC production code of the Extended Twisted Mass Collaboration (ETMC)
- $\approx$ 150k lines (C), MPI + OpenMP, macro-based hardware specialization (intrinsics or inline assembly for SSE4, BlueGene[L,P,Q])
- mainly 2 to 3 people over $\sim$ 20 years
  - major contributions by another 3 to 4
  - small contributions by another 10 or so
- since around 2015, rely on (and extend) libraries
  - QPhiX for AVX2, AVX512 (Bálint Joó et al.) [10.1007/978-3-319-46079-6_30, gh.com/JeffersonLab/qphix]
  - DD-αAMG for MG solver on CPU [10.1137/130919507, 10.48550/arXiv.1307.6101, 10.1103/PhysRevD.94.114509, gh.com/sbacchio/DDalphaAMG]
  - QUDA for GPU operators and solvers (Kate Clark et al.) [10.1016/j.cpc.2010.05.002, 10.1145/2063384.2063478, 10.1109/SC.2016.67]
- long history of debates about future code for GPU machines without results (essentially lack of people power...)



Contributors 15

+ 4 contributors

Languages

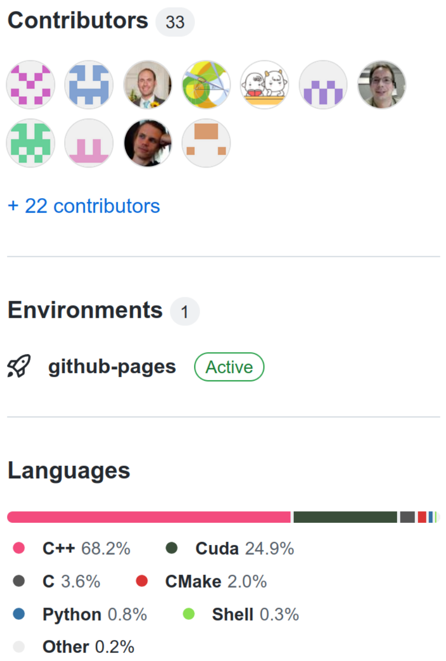
● **C** 76.6% ● **Cuda** 15.4%
● **C++** 3.6% ● **Lex** 2.1%
● **Makefile** 0.8% ● **Assembly** 0.7%
● **Other** 0.8%

Saved by the QUDA library

- First use with tmLQCD around 2015 (for observables)
- Work on interface for HMC started in 2018, first running version in 2021 (motivated by QUDA performance-portability efforts)

```
BeginExternalInverter QUDA          # equivalents of QUDA tests
  MGCoarseMuFactor = 1.0, 1.0, 60.0 # command line parameters
  MGNumberOfLevels = 3
  MGNumberOfVectors = 24, 32
  MGSetupSolver = cg
  [...]
EndExternalInverter
BeginMonomial CLOVERDETRATIO
  Timescale = 3
  kappa = 0.1394267
  2KappaMu = 0.000200774448
  rho = 0.0
  rho2 = 0.0018
  CSW = 1.69
  AcceptancePrecision =  1.e-21
  ForcePrecision = 1.e-18
  Name = cloverdetratio3light
  MaxSolverIterations = 500
  solver = mg
  useexternalinverter = quda         # enable QUDA pathway in solver
  usesloppyprecision = single        # driver for this monomial
EndMonomial
```

https://github.com/lattice/quda



Contributors 33

+ 22 contributors

Environments 1

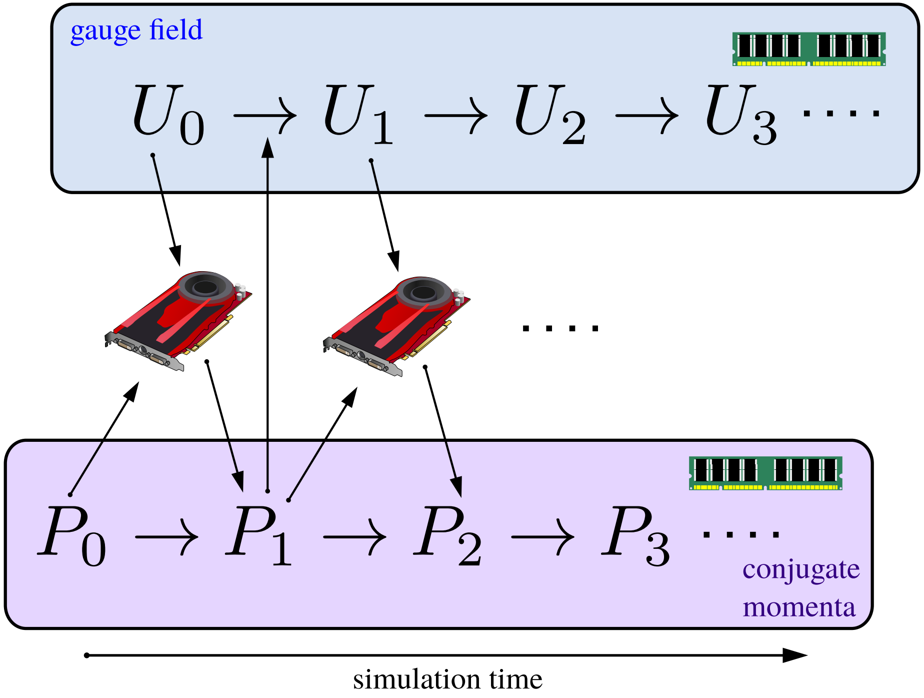
github-pages Active

Languages

● **C++** 68.2%	● **Cuda** 24.9%
● **C** 3.6%	● **CMake** 2.0%
● **Python** 0.8%	● **Shell** 0.3%
● **Other** 0.2%	

Hybrid CPU/GPU HMC

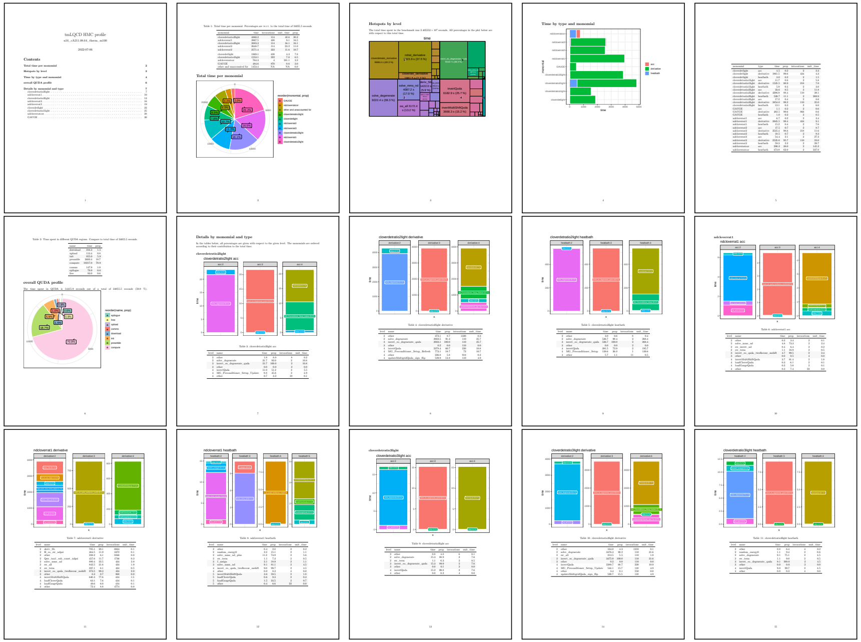
- gauge field and conjugate momenta in host memory
- solvers and gauge term derivative on device
- need to keep track of gauge field state
  - solution: tag host and device objects
  - using checksum too restrictive
  - $\rightarrow$ simply use trajectory time (real number)
  - when host and device tags disagree, update device copy (optional: use thresholds)
  - nice side-effect: natural mechanism to track MG setup
- incremental port: need good mechanisms to identify hotspots *and* their causes


gauge field
$U_0 \rightarrow U_1 \rightarrow U_2 \rightarrow U_3 \cdots$
$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \cdots$
conjugate momenta
simulation time

tmLQCD's profiler

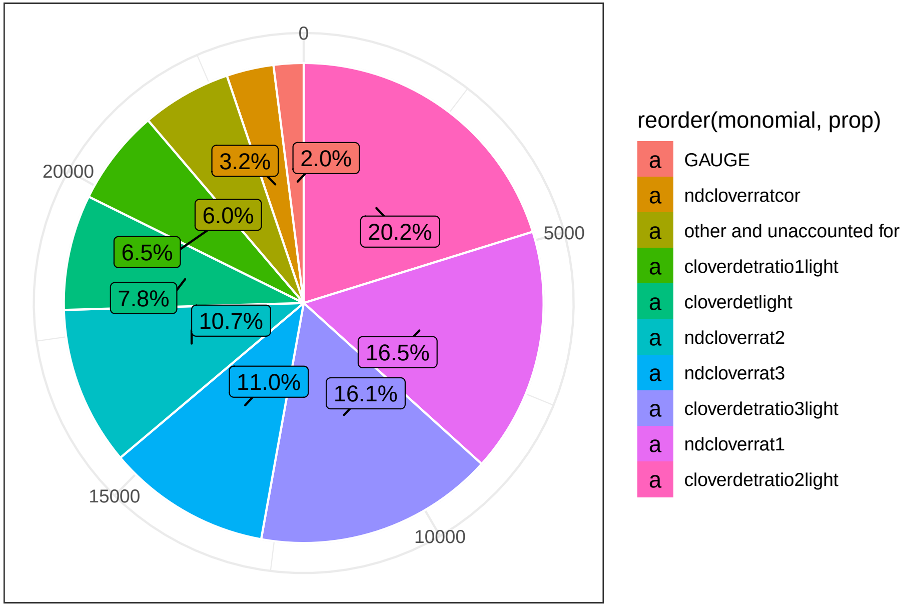
```
tm_stopwatch_push(&g_timers, __func__, "");
[...]
tm_stopwatch_pop(&g_timers, 0, 0, "TM_QUDA");
```

- introduced stack-based profiler into tmLQCD (and accompanying analysis scripts)
  - output simply to stdout with `level0/level1/level3/...` tags
  - analysis parses log file (176 lines of R) and renders Rmarkdown report
  - Tables and plots with context and identification of call tree depth
  - Visualize also QUDA's finalisation profile (see backup slides)



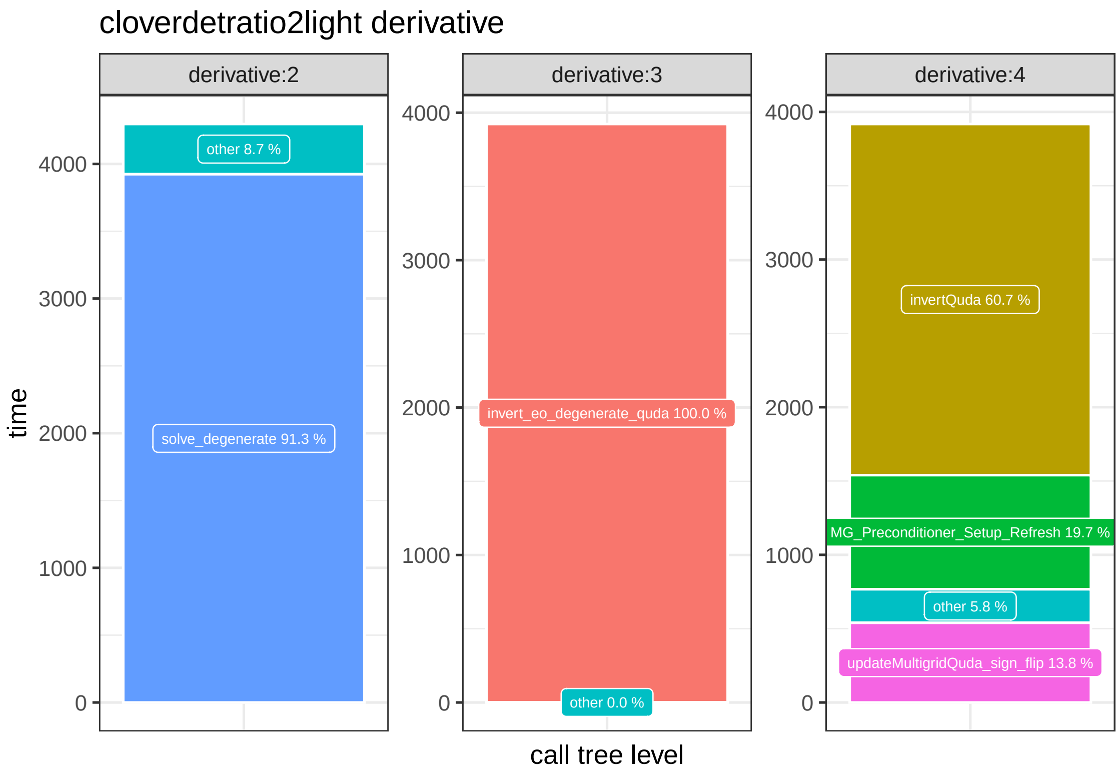
tmLQCD's profiler

- combine view on physical and computational hotspots
- focus on splitting of the MD Hamiltonian at this global level $\Rightarrow$

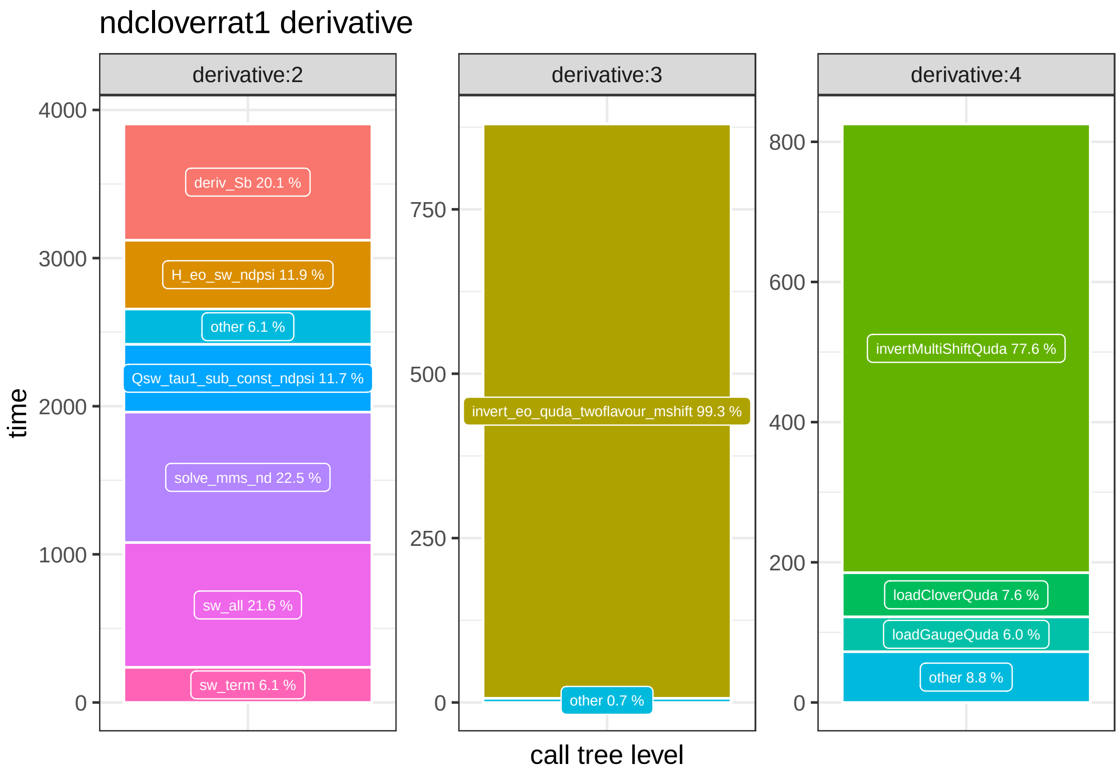


(profile from $64^3 \cdot 128$ physical point simulation on 16 Marconi 100 nodes)

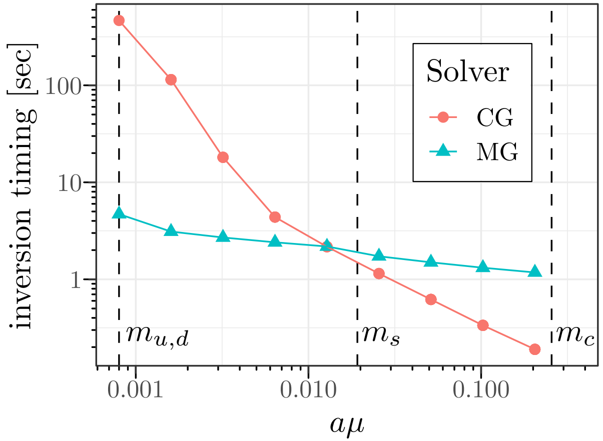
GPU-dominated parts



CPU-dominated parts



MG solver in the light sector



Comparison between MG-preconditioned-GCR and mixed-precision CG (GPU)
MG timing: two inversions + unavoidable overheads from coarse operator updates between D and $D^\dagger$ inversions

In practice we employ

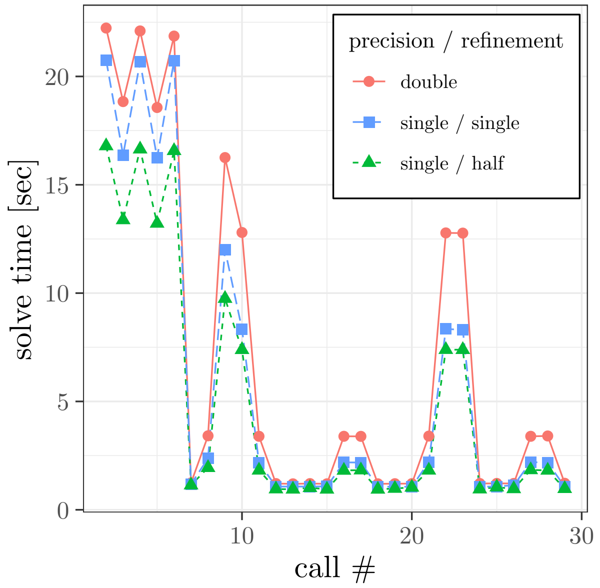
- 2 to 3 ρ-shifts (shifting the EO-operator)
- 3-4 time scales

$\rightarrow$ per trajectory need to solve systems with:

- $\rho = 0$ about $\mathcal{O}(100)$ times
- $\rho \approx 0.001$ about $\mathcal{O}(100)$ times
- $\rho \approx 0.01$ about $\mathcal{O}(200)$ times
- $\rho \approx 0.1$ about $\mathcal{O}(400)$ times

MG requires two solves in derivative and an update of the coarse operator (due to twisted mass sign change), but easily wins up to $\rho \approx am_s$.
We employ both MG and CG to minimize total cost.

Multi-shift solver for the $1 + 1$ sector



Rational Approximation Correction Term

- $64^3 \cdot 128$ lattice
- CPU: 3072 cores Intel Platinum 8168 (64 Juwels nodes)
- GPU: 32 A100 (8 Juwels Booster nodes)

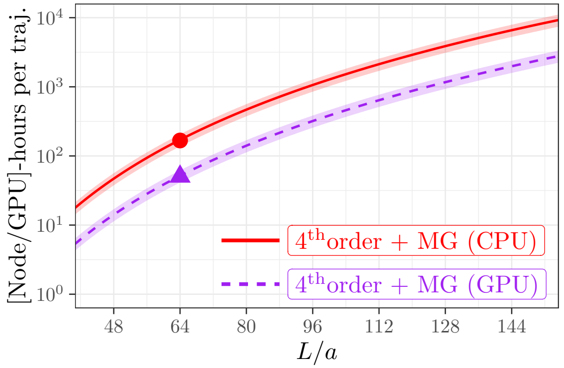
Machine / Algorithm	HB	ACC
(CPU) QPhiX multi-shift CG	810 s	550 s
(CPU) DD-αAMG accelerated multi-shift CG	590 s	400 s
(GPU) QUDA mshift CG (double)	145 s	93 s
(GPU) QUDA mshift CG (single / single)	127 s	79 s
(GPU) QUDA mshift CG (single / half)	103 s	66 s

- Similar real time improvements in the derivative terms
- mixed-precision refinement really helps with the expensive solves (factor ≈ 1.5)

Current state of the port

● 3072 cores Intel Xeon Platinum 8168 (64 nodes)

▲ 32 NVIDIA A100 + 384 cores AMD EPYC Rome 7402 (8 nodes)



(real trajectories at $M_\pi \sim 135$ MeV on $64^3 \cdot 128$ lattice)

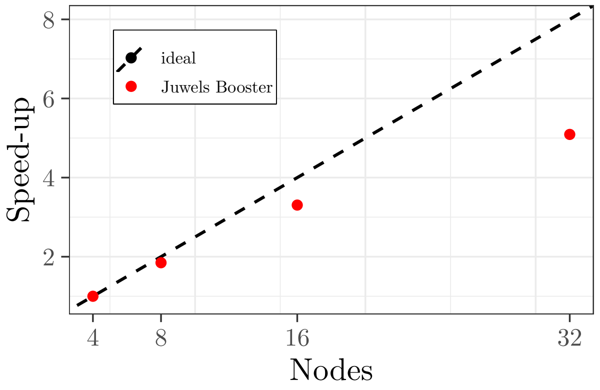
machine	real time	node-hours (CPU) / GPU-hours	kWh
64 nodes (Juwels)	2.61 h	167	~ 84
32 GPUs (Juwels Booster)	1.58 h	50.6	~ 24

- CPU strong scaling to 64 nodes okay, not great beyond that $\rightarrow$ real throughput limitation
- gets (much) worse for larger volumes where many more nodes are required (depends on machine though)
- Improvement factor CPU/GPU in energy usage already ~ 3.5
- Expect another factor of 2 to 2.5
- Finally we will be able to run a trajectory in less than one hour again!

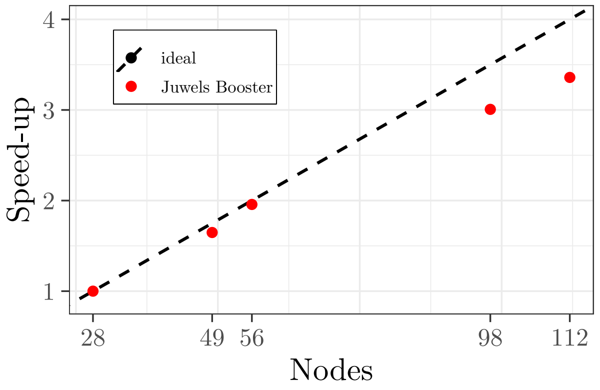
Current state of the port

HMC Strong scaling

$64^3 \cdot 128$ @ $M_\pi \sim 135$ MeV



$112^3 \cdot 224$ @ $M_\pi \sim 135$ MeV



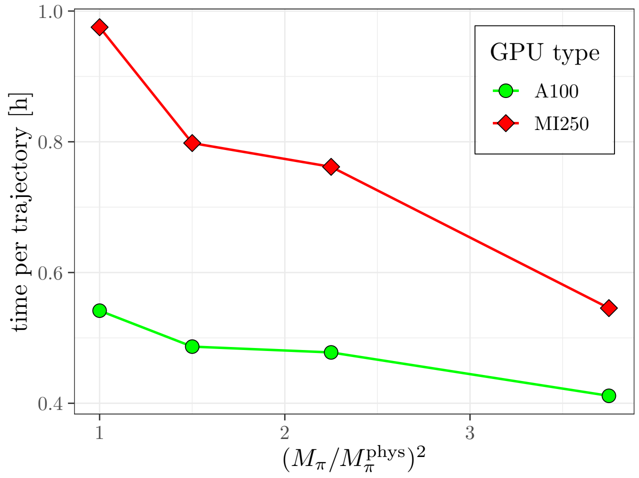
- see excellent whole-program scalability on Juwels Booster and very good absolute per trajectory times
- Scalability will get worse as we move the CPU-dominated parts fully to GPU
  - more of the scaling behaviour will depend on the MG, which does not scale well by definition

What about performance-portability?

MI250 PRELIMINARY

Single-node comparison on a $32^3 \times 64$ lattice on

- Juwels Booster ($4\times$ A100)
- Jureca DC-MI200 ($4\times$ AMD MI-250, `ROCm 5.2.0`, still being fine-tuned!).



(full HMC run, thermalised configuration, comparable acceptance rate)

$(M_\pi/M_\pi^{\text{phys}})^2$	time A100 [h]	time MI250 [h]	ratio
3.75	0.411	0.546	1.33
2.25	0.478	0.762	1.59
1.50	0.487	0.798	1.64
1.00	0.542	0.975	1.80

- Time investment (for us)[a]:
  - 2-3 hours to adjust tmLQCD build system & compile code
  - few hours with JSC admins and AMD experts to resolve a few ROCm issues

  ! get an HMC which runs on MI-250 and is *at most* a factor of 2 slower even at the physical point (at least on a single node) $\rightarrow$ excellent!

[a] major thanks to Bálint Joó and QUDA devs for many hundreds of hours of effort which make this possible!

Conclusions and Outlook

- thanks to QUDA devs, we were able to improve our energy efficiency by factor of ≈ 3 already, another factor of ≈ 2 remaining
- will allow us to complete ensemble set on current & upcoming machines
- probably the end of the line for tmLQCD
  - C is too limiting, data layouts too inflexible
- time to join forces with others and / or redesign our toolset completely
  - excellent performance of QUDA-MG means that it will play a role no matter what
- prepare for modular exascale machines

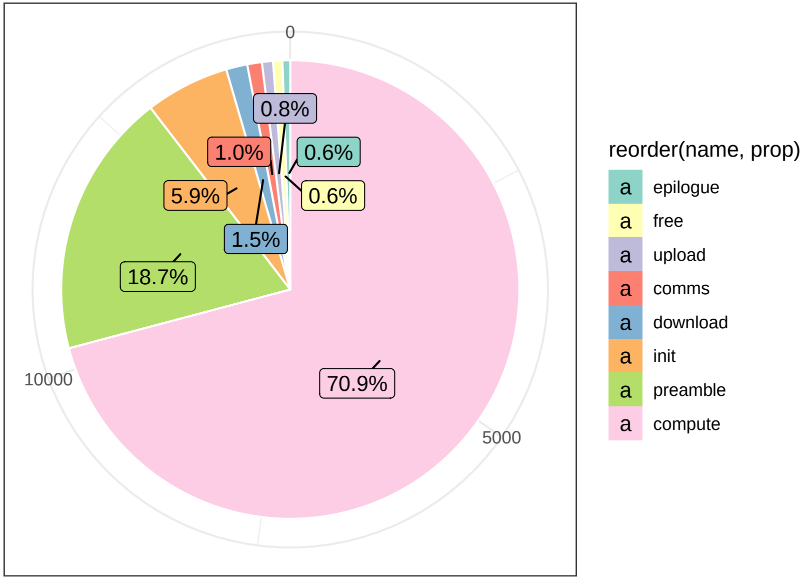
Thanks for your attention!

Backup

Backup Slides

QUDA's finalisation profile (backup)

- Same analysis script also visualises QUDA's finalisation profile
- in general spend 70 to 80 % of QUDA time in compute
- host-device memory traffic is a tiny overhead (for now)
- our poor decisions: too much time spent in memory allocations and frequent reinitialisations (*init* and *preamble*)
- $\rightarrow$ some potential for future improvement here


reorder(name, prop)
a epilogue
a free
a upload
a comms
a download
a init
a preamble
a compute