tmLQCD on GPUs: minimum effort performance-portability

Bartosz Kostrzewa

Marco Garofalo, Simone Romiti, Carsten Urbach

Simone Bacchio, Jacob Finkenrath, Ferenc Pittler

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

Efficient simulations on GPU hardware, October 24th to 27th, ETH Zürich

Overview

- This talk: field report on experience in ETMC trying to make use of GPU machines for ensemble production given certain constraints
  - diverse physics projects within collaboration: so far impossible to establish collaborative effort on common software framework
  - culture of every PhD student reinventing the wheel and PIs not really coordinating on a technical level doesn't help
  - very limited number of people willing / able to work on production code (less than 1 FTE)
  - lattice action not used by anyone else (at least non-degenerate part)

Background & Motivation

- the twisted clover action
- physics and simulation goals

"Porting" our HMC

- exploring the solution landscape
- tmLQCD + QUDA

Future Challenges

- very dense GPU machines
- modular supercomputing architecure

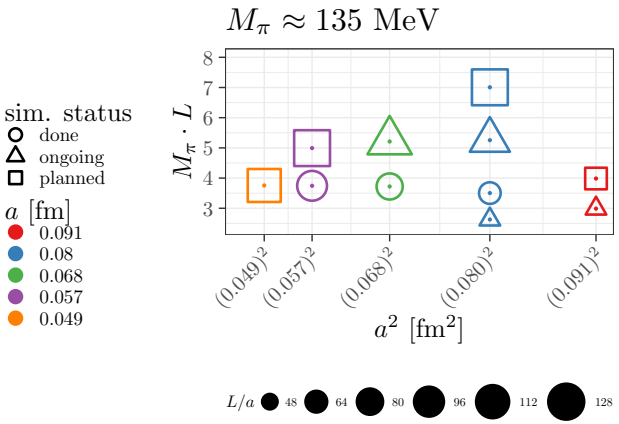
The $N_f = 2 + 1 + 1$ twisted clover action

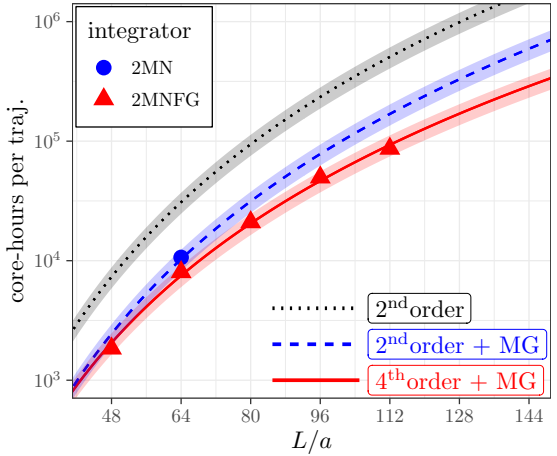
[10.1103/PhysRevD.95.094515, 10.1051/epjconf/201817502003, 10.1103/PhysRevD.98.054518]

$$
\begin{aligned}
S &= \beta \sum_{x;P} \left[b_0 \{ 1 - \tfrac{1}{3} \mathrm{ReTr} P^{1\times 1}(x) \} + b_1 \{ 1 - \tfrac{1}{3} \mathrm{ReTr} P^{1\times 2}(x) \} \right] \\
&+ \sum_x \bar{\chi}_\ell(x) \left[D_W(U) + m + i\mu_\ell \gamma_5 \tau^3 + \tfrac{i}{4} c_{\mathrm{SW}} \sigma^{\mu\nu} \mathcal{F}^{\mu\nu}(U) \right] \chi_\ell(x) \\
&+ \sum_x \bar{\chi}_h(x) \left[D_W(U) + m - \mu_\delta \tau^1 + i\gamma_5 \mu_\sigma \tau^3 + \tfrac{i}{4} c_{\mathrm{SW}} \sigma^{\mu\nu} \mathcal{F}^{\mu\nu}(U) \right] \chi_h(x)
\end{aligned}
$$

- $b_0 = 1 - 8b_1$, $b_1 = -0.331$ [Iwasaki; 1983]
- $c_{\mathrm{SW}} = 1 + 0.113(3) \frac{g_0^2}{\langle P \rangle}$ [Aoki,Frezzotti,Weisz; 1999; arXiv:hep-lat/9808007]
- Automatic $\mathcal{O}(a)$-improvement of all physical observables & simplified mixing patterns of composite operators (at $m = m_{\mathrm{cr}}$) [Frezzotti, Rossi; 2004; 10.1088/1126-6708/2004/08/007, 10.1088/1126-6708/2004/10/070]
- Vastly reduced pion mass splitting compared to action without clover term (essentially zero for fine lattice spacings)
- χ_ℓ two-flavour light quark field $\rightarrow$ determinant reduces to usual $MM^\dagger$ form (up to taking into account sign of twisted mass)
- χ_h strange and charm two-flavour quark field $\rightarrow$ no reduction: bona-fide two-flavour operators required

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





- State-of-the-art integrator & solvers → 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 $\geq$ 6 hours at this stage
- **Absolutely need GPU implementations of everything**

Our typical MD Hamiltonian

gauge and light sector

gauge monomial

$$\frac{\beta}{3}\sum_x \left(c_0 \sum_{\substack{\mu,\nu=1 \\ 1\le\mu<\nu}}^{4} \{1-\mathrm{Re}(U^{1\times1}_{x,\mu,\nu})\} + c_1 \sum_{\substack{\mu,\nu=1 \\ \mu\neq\nu}}^{4} \{1-\mathrm{Re}(U^{1\times2}_{x,\mu,\nu})\} \right)$$

- force evaluated several hundred times per trajectory
  - must be offloaded to GPU

asymmetric even-odd preconditioning

$$Q^{\pm} = \gamma_5(M_{\text{clov}} \pm i\mu_\ell\gamma_5) \;\rightarrow\; (Q^{+})^{\dagger} = Q^{-}$$

$$\det(Q^{\pm}) = \det(M_{ee} \pm i\mu_\ell\gamma_5)\cdot\det(\hat{Q}^{\pm})$$

$$\hat{Q}^{\pm} = \gamma_5\left[(M_{oo} \pm i\mu_\ell\gamma_5) - M_{oe}(M_{ee} \pm i\mu_\ell\gamma_5)^{-1}M_{eo}\right]$$

- support for asym. precon not a given in most frameworks
- issues with MG

degenerate determinant (ratios)

$$\int \mathcal{D}\phi_j^{\dagger}\,\mathcal{D}\phi_j\;\exp\left\{-\phi_j^{\dagger}\,\frac{1}{\hat{W}^{+}(\rho_t)\hat{W}^{-}(\rho_t)}\,\phi_j\right\}$$

$$\int \mathcal{D}\phi_i^{\dagger}\,\mathcal{D}\phi_i\;\exp\left\{-\phi_i^{\dagger}\,\hat{W}^{-}(\rho_t)\frac{1}{\hat{W}^{+}(\rho_b)\hat{W}^{-}(\rho_b)}\hat{W}^{+}(\rho_t)\,\phi_i\right\}$$

- $\hat{W}^{\pm}(\rho) = \hat{Q}^{\pm} \pm i\rho$ s.t. $\hat{W}^{+}(\rho)\hat{W}^{-}(\rho) = \hat{Q}^{+}\hat{Q}^{-} + \rho^2$ and clover inverse ρ-independent
- 3-4 preconditioning masses, 2-3 timescales, MG solves for smallest ρ

Our typical MD Hamiltonian

"heavy" sector

even-odd preconditioning in the heavy sector: τ^1 $\rightarrow$ need genuine two-flavour operators

$$\hat{Q}_h = \gamma_5 \left[(M_{oo} + i\mu_\sigma \gamma_5 \tau^3 - \mu_\delta \tau^1) - M_{oe}(M_{ee} + i\mu_\sigma \gamma_5 \tau^3 - \mu_\delta \tau^1)^{-1} M_{eo} \right]$$

rational approximation partial fractions

$$\mathcal{R}(\hat{Q}_h^2) = \prod_{i=1}^{N} \frac{\hat{Q}_h^2 + a_{2i-1}}{\hat{Q}_h^2 + a_{2i}} \approx \frac{1}{\sqrt{\hat{Q}_h^2}}$$

- $N \approx 10$, with $\mathcal{R}$ split across 2-3 monomials on 2-3 timescales (usually 3)
- on CPU machines, accelerate solution of smallest shifts using DD-αAMG [Bacchio, Finkenrath, 2019, Comput.Phys.Commun. 236 (2019) 51-64]

rational approximation correction factor

$\det(|\hat{Q}_h|\mathcal{R})$

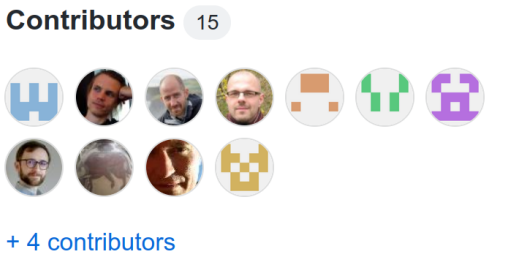
- $B \cdot B^\dagger = |\hat{Q}_h|\mathcal{R}$
- $B = (1+Z)^{1/4} = \sum_{i=0}^{m} c_i Z^i = 1 + \frac{1}{4}Z - \frac{3}{32}Z^2 + \frac{7}{128}Z^3 + \ldots$ with $Z = \hat{Q}_h^2 \mathcal{R}^2 - 1$
- contributes about 10 to 20% of total runtime on CPU machines (5 to 7 solves depending on volume) $\rightarrow$ benefits from DD-αAMG acceleration

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]

- 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] with SPI comms)
- C. Urbach and 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]
  - tmLQCD used as a library driver to allow physics programs to make use of QPhiX, DD-αAMG and QUDA

https://github.com/etmc/tmLQCD



State of the tmLQCD QUDA interface in 2018

- Basic structure due to Mario Schröck (around 2015) with extensions by BK
- Full interface to QUDA MG since about 2017/2018 with regular maintenance to follow QUDA development
- Wiring up of components required for HMC non-trivial
  - really worth the effort knowing that it's a suboptimal (and ephemeral) solution?

One type of input file to control all aspects of calling solvers.

```
BeginExternalInverter QUDA
  MGCoarseMuFactor = 1.0, 1.0, 60.0
  MGNumberOfLevels = 3
  MGNumberOfVectors = 24, 32
  MGSetupSolver = cg
  [...]
EndExternalInverter

BeginOperator CLOVER
  kappa = 0.1394267
  2KappaMu = 0.000200774448
  CSW = 1.69
  SolverPrecision =  1.e-21
  MaxSolverIterations = 80
  solver = mg
  useexternalinverter = quda
  usesloppyprecision = single
EndOperator
```

Finding the right framework for the job

Debates since around 2015 on how to approach the performance-portability problem with limited person-power.
The following are impressions / opinions that we collected in this process. (and may well be wrong)

Grid

+ Excellent vector data structures, easy to extend
+ Very good performance on SIMD architectures
- Unclear multi-node performance on GPU machines (back in 2018)
- Unclear (to us) how DD-αAMG-implementation performs on GPU machines (to this day)
  - would probably need to wire up QUDA interface for QUDA-MG
- Significant effort required to support degenerate twisted-clover & especially non-degenerate doublet

Chroma + QDP-JIT + QUDA

+ Excellent whole-program performance
+ QUDA interface $\rightarrow$ acccess to QUDA-MG
- Significant effort required to support non-degenerate twisted-clover doublet
- Hard to compile, lots of effort to run on new machines as they come online (at least for us)
- XML input: steep learning curve for students
- Performance-portability to future non-NVIDIA GPUs unclear (back in 2019)

QUDA

+ Excellent performance on NVIDIA hardware
+ QUDA-MG
+ Quite a bit of QUDA experience in the ETMC
+ Effort to add non-degenerate twisted-clover reasonable
+ Gauge force easy to wire up
- Need driver code which itself needs to be future-proof
- Unclear performance-portability (back in 2019)
- Implementation of fermionic forces incomplete, neeeds extension to support non-degenerate twisted-clover

tmLQCD + QUDA in the HMC

- Work on interface for HMC started in 2018, first running version in 2021 (motivated by QUDA performance-portability efforts)

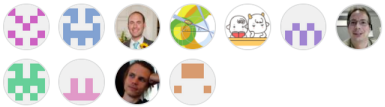
→ gh.com/etmc/tmLQCD/tree/quda_work

```
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
```

QUDA

gh.com/lattice/quda

Contributors 33



+ 22 contributors

Environments 1

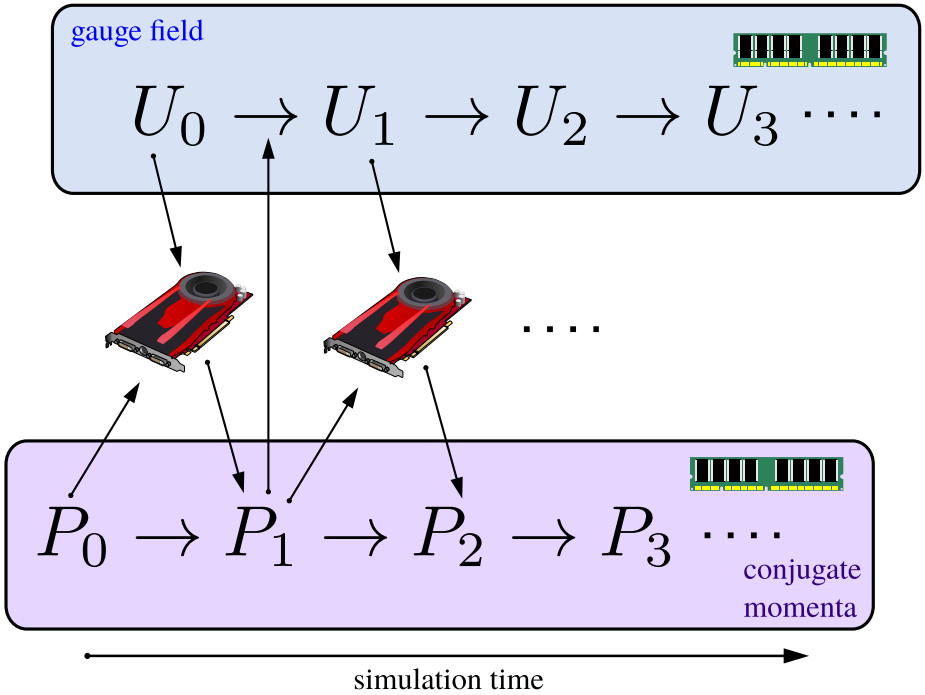
github-pages Active

Languages

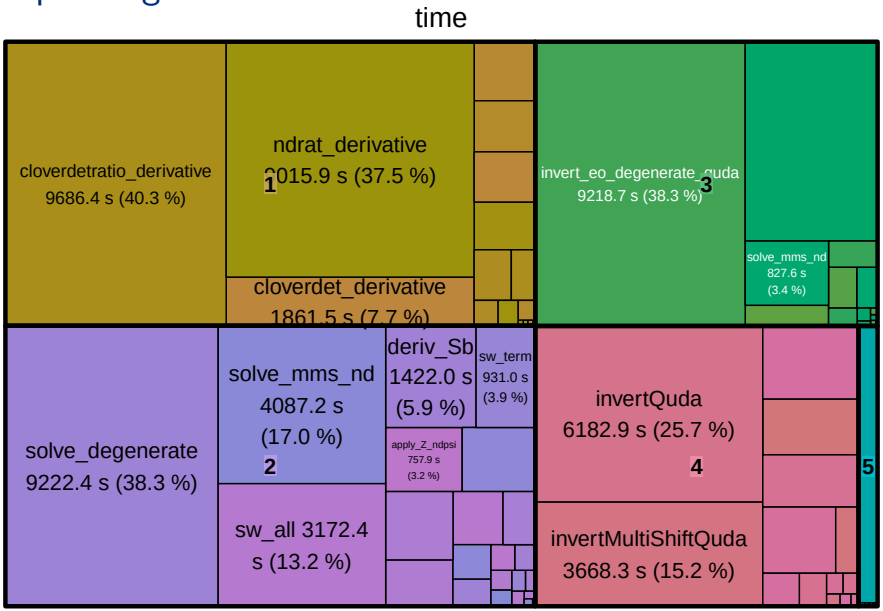
C++ 68.2% Cuda 24.9%
C 3.6% CMake 2.0%
Python 0.8% Shell 0.3%
Other 0.2%

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



The problem with profiling tools

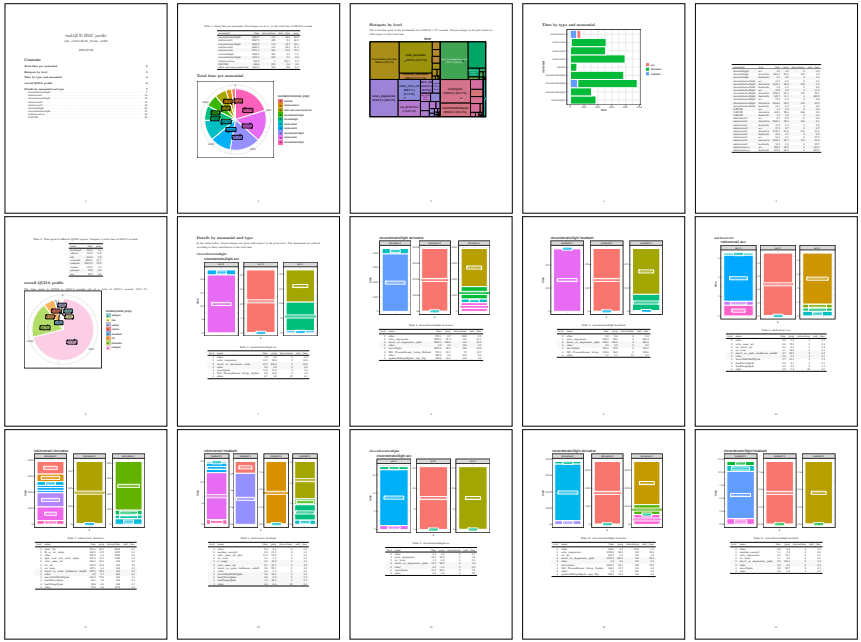


- HMC with many monomials and MG is kinda complicated: would like to profile real-world examples to get a feel for real-world balance of hot-spots
  - same functions called in multiple places, sometimes even at different depths of the call tree
  - profiling tools (without tagging or markup) do not give sufficient context
  - unclear if profile is physically sensible or result of specific problems with certain parameter combinations or algorithms

tmLQCD's profiler

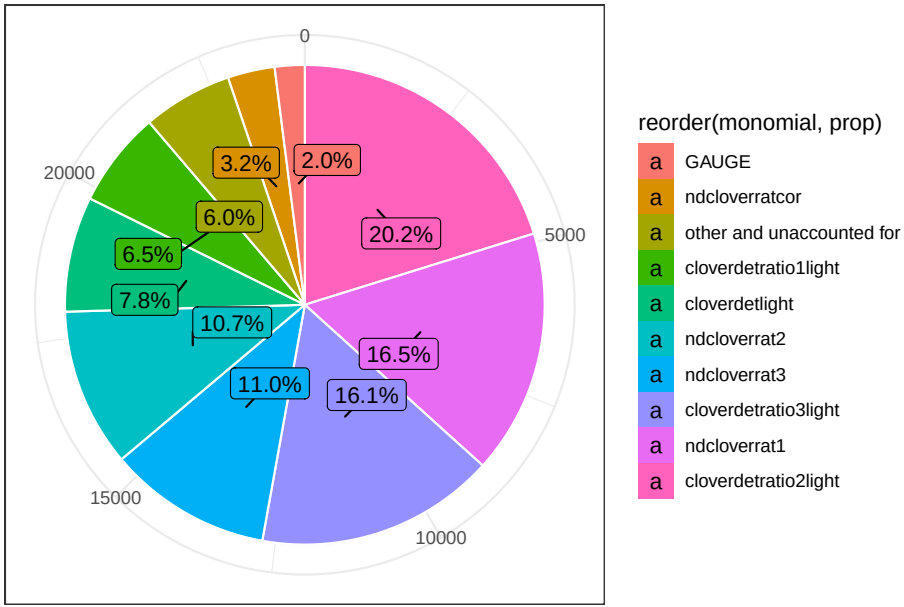
```
tm_stopwatch_push(&g_timers, __func__, "");
[...]
tm_stopwatch_pop(&g_timers, 0, 0, "TM_QUDA");
```

- introduced stack-based profiler into tmLQCD (and accompanying analysis scripts)
  - output simply to stdout with `level0/level1/level2/...` 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



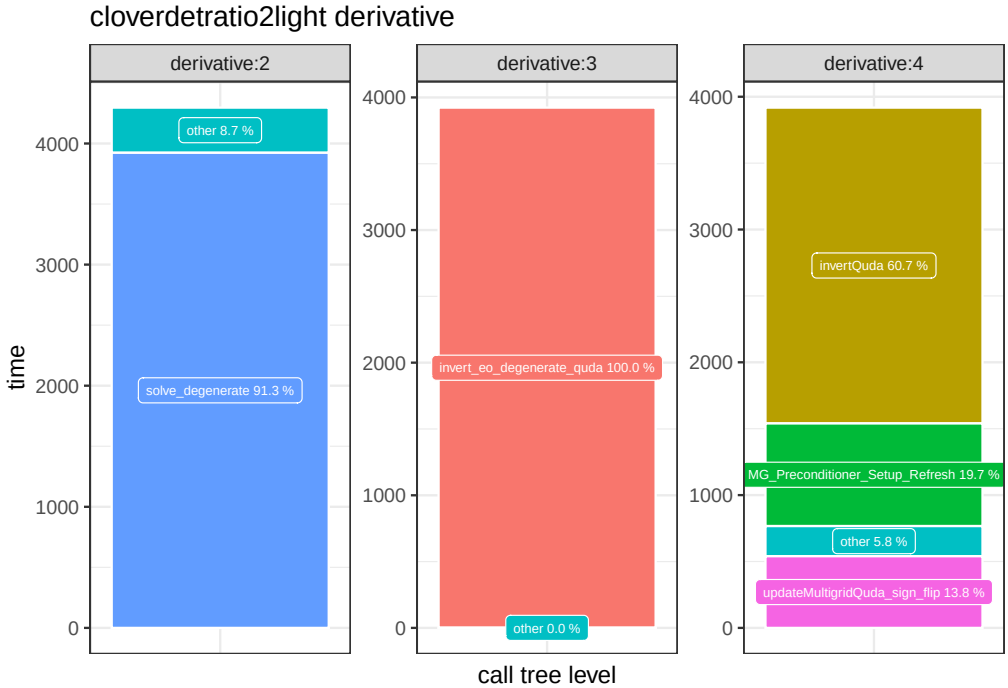
tmLQCD's profiler

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



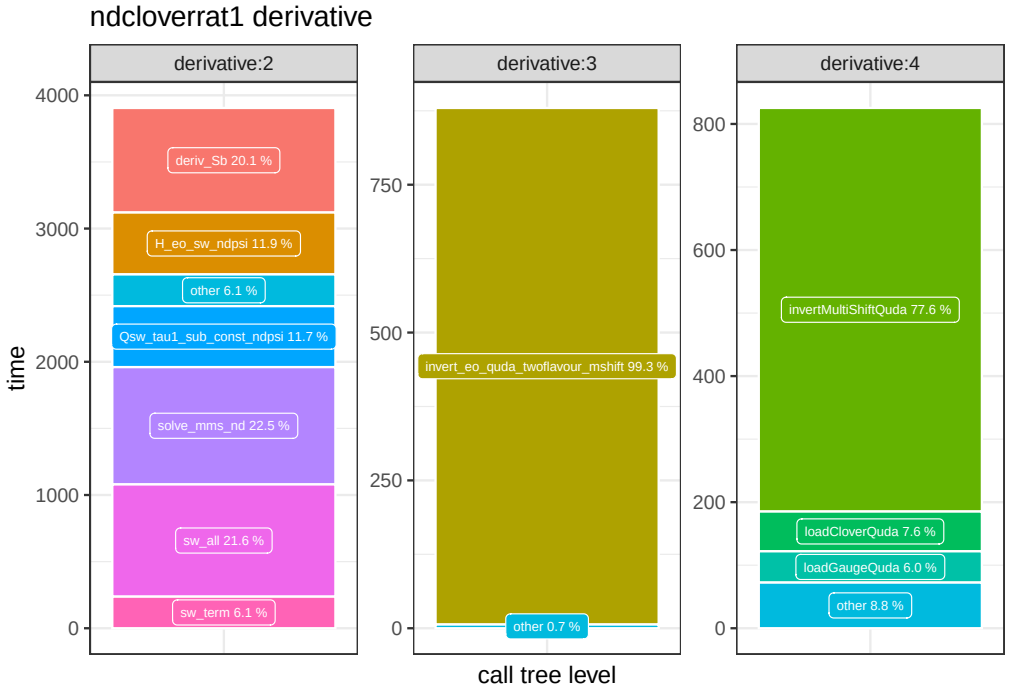
(profile from $64^3 \cdot 128$ physical point simulation on 16 Marconi 100 nodes)

GPU-dominated parts



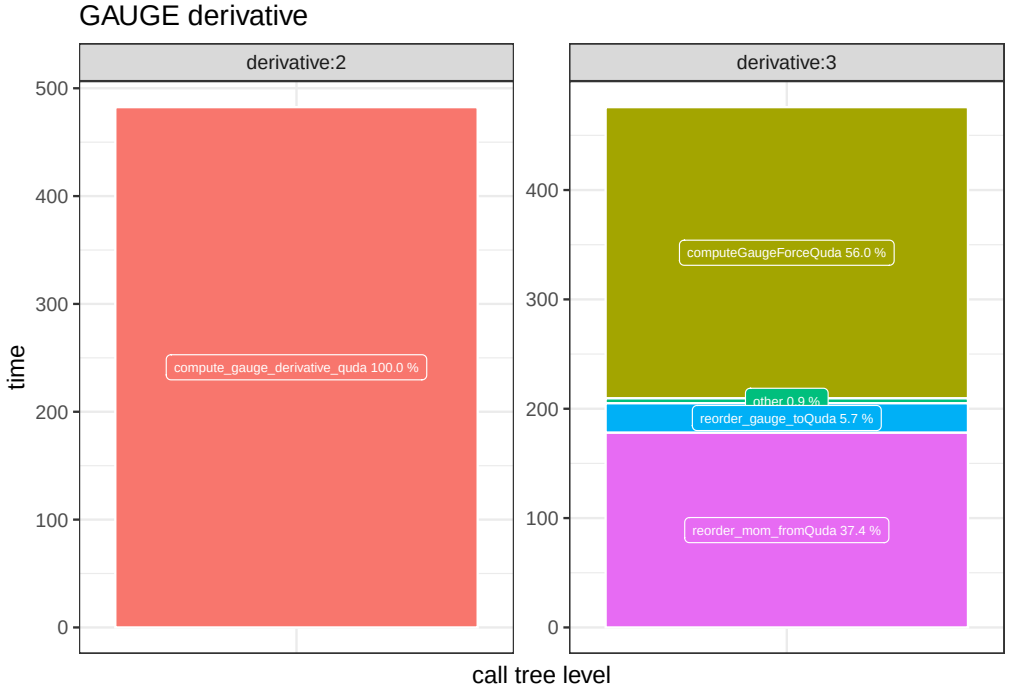
- Good: more than 90% of time spent in solver or unavoidable MG "overheads"

CPU-dominated parts



- Bad: only slightly more than 20% of time spent in solver → currently working on implementing remainder in QUDA

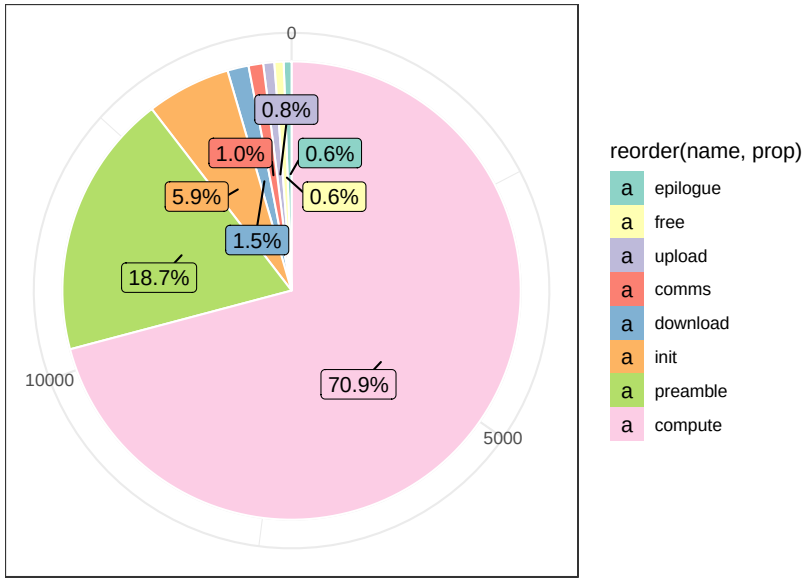
What about host-device transfers?



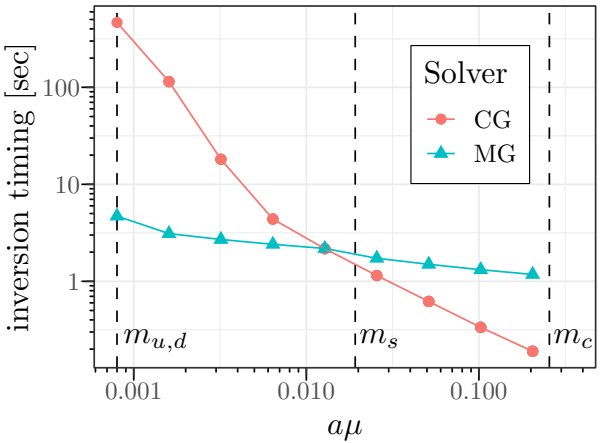
- host-device transfers not a big deal even for gauge derivative, but we should move our field reordering completely to QUDA

QUDA's finalisation profile

- Same analysis script also visualises QUDA's finalisation profile
- On Marconi 100, spend about 50 to 70% of time in QUDA
- of that, spend about 70 to 80% of 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



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

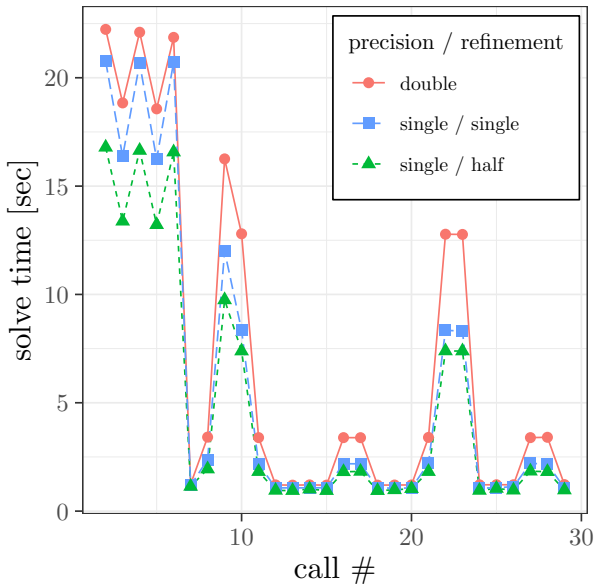
- 2 to 3 ρ-shifts (shifting the EO-operator)
- 3-4 time scales

$\rightarrow$ 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



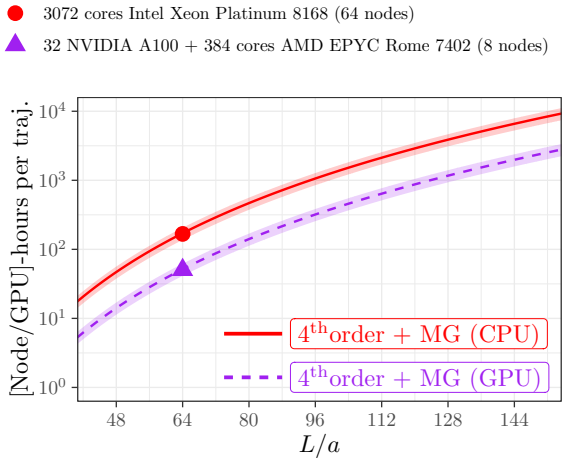
Rational Approximation Correction Term

- $64^3 \cdot 128$ 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)
- Further improvement expected through developments presented by Kate at LAT'22

Current state of the port



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

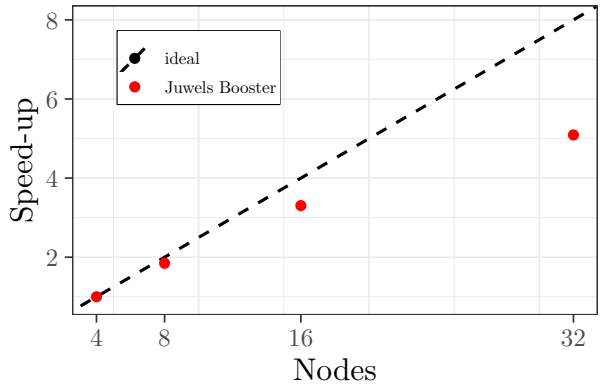
machine	real time	node-hours (CPU) / GPU-hours	kWh
64 nodes (Juwels)	2.61 h	167	~ 84
32 GPUs (Juwels Booster)	1.58 h	50.6	~ 24

- 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 (remaining parts of fermion derivative)
- Finally we will be able to run a trajectory in less than one hour again!

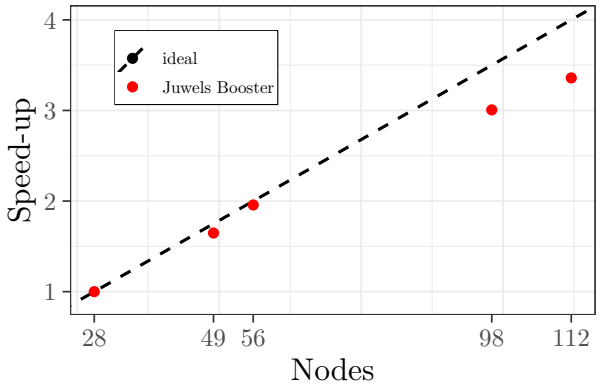
Current state of the port

HMC Strong scaling

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



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



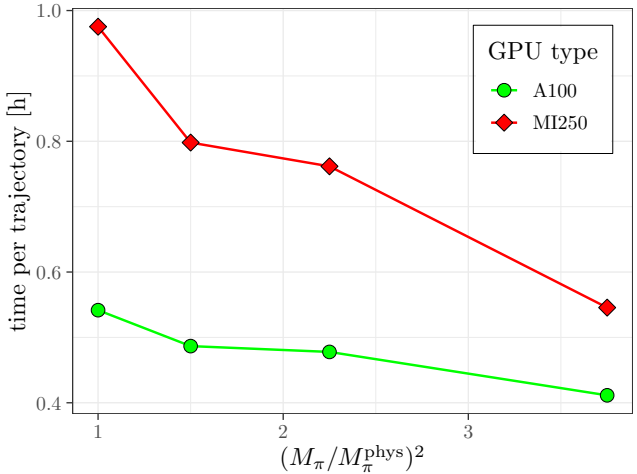
- see excellent whole-program scalability on Juwels Booster and very good absolute per trajectory times
- Scalability will get worse as we move the CPU-dominated parts fully to GPU
  - more of the scaling behaviour will depend on the MG, which does not scale well by definition

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!

Lessons learned and outlook?

- Saved by QUDA's performance-portability push
  - enabled us to implement reasonably efficient HMC on current generation of GPU machines ($\sim 50\%$ GPU usage now, somewhat more soon when fermionic forces fully available)
- lower device memory footprint than a full GPU port
  - we can run a $64^3 \cdot 128$ HMC (including MG) on just 16 A100 (40GB)
    - ⋆ can run MG at close to optimal number of GPUs
    - ⋆ advantage will evaporate on machines with larger GPU memories and/or fewer CPU cores / GPU

Biggest regrets?

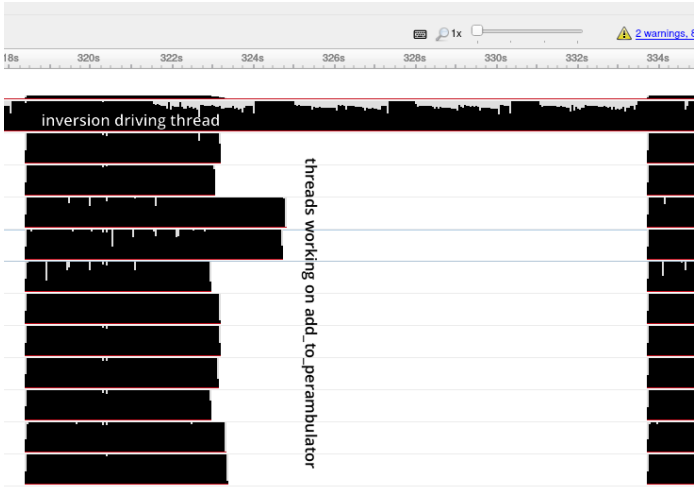
- QUDA interface should have been implemented directly in QUDA rather than in tmLQCD
  - major source of pain due to inability of using QUDA objects directly where it makes sense

- HMC with 70 % efficiency sufficient for current generation (Juwels Booster, Marconi 100, Leonardo, LUMI-G)
  - problem on future machines with low-power CPUs driving dense high-power GPU configurations (Jupiter?)
- Some workloads have massive memory requirements $\rightarrow$ need to scale to a number of GPUs where the MG is not that efficient any more
  - Maybe a targeted hybrid approach is required? Needs co-design effort!

Hybrid GPU-CPU perambulators

In the context of the HISKP / ETMC effort on spectroscopy and scattering using stochastic distillation:
[10.1103/PhysRevD.96.054516, 10.1103/PhysRevD.99.034511, 10.1103/PhysRevD.96.034510, 10.1103/PhysRevD.98.114511, epja/s10050-020-00057-4, 10.1016/j.physletb.2021.136449]

- Perambulator: $\tau_{\alpha\beta}(t',t) = V^\dagger(t') M^{-1}_{\alpha\beta}(t',t) V(t)$
  [Peardon et al., Phys. Rev. D 80, 054506]
  - invert Dirac operator on sources $\delta_{\alpha\beta} V(t)$, multiply by $V^\dagger(t') \, \forall \, t'$
  - $V(t)$: N first eigenvectors of gauge-covariant 3D Laplacian for each time-slice t (several hundred GB on $48^3 \cdot 96$ lattice)
    - ⋆ impractical, difficult or impossible to fit into GPU memory when MG is most efficient (2-3 nodes of Juwels Booster)
    - ⋆ Instead: have one thread driving GPU inversions while all other threads do $V^\dagger$ multiplication in the background
    - ⋆ Result: Essentially 100% GPU utilisation (cost: doubling of host buffers for propagators and pointer swapping)

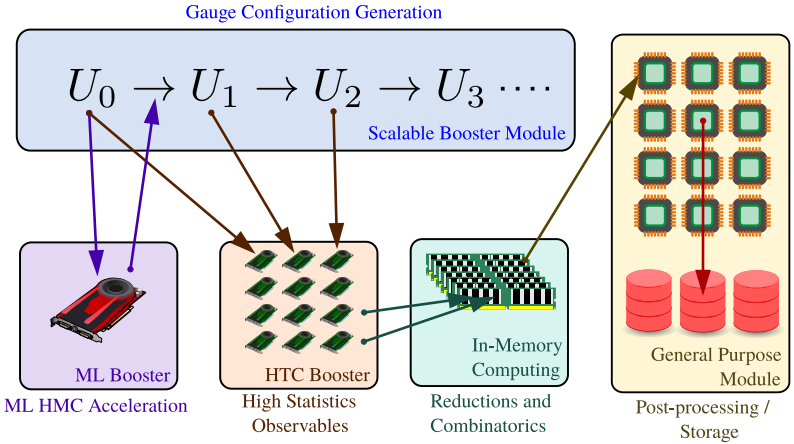


Future workloads and machines?

- Indications that lattice gauge theory will continue to play an important role in the coming decade
  - R ratio, $(g-2)_\mu$, heavy flavour physics, β-decay, nuclear structure, non-perturbative BSM, ...

[Snowmass 2021 LGT report]

- Massive statistics, many lattice spacings, large volumes $\rightarrow$ will eventually reach a situation where we can't even store all gauge configurations



Hypothetical future LGT workflow running on MSA with many different modules.

- In Europe, Juelich Supercomputer Center is strongly pushing the idea of a Modular Supercomputing Architecture (MSA)
  Suarez, Eicker, Lippert, Kreuzer et al., [Cont. High Perf. Comp, 2019], [FZ Juelich, 2021]
- Exploiting the MSA will require thinking about task parallelism and workflow management in addition to performance engineering.

Conclusions and Outlook

- thanks to QUDA devs, we were able to improve our HMC's 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
  - people problem: need to be able to offer attractive positions
  - perhaps work with labs and/or HPC centers to provide these positions

Thanks for your attention!