Twisted mass ensemble generation on GPU machines

Bartosz Kostrzewa[a,*]**, Simone Bacchio**[c]**, Jacob Finkenrath**[c]**, Marco Garofalo**[b]**,
Ferenc Pittler**[c]**, Simone Romiti**[b]**and Carsten Urbach**[b]**for the ETM Collaboration**

[a]*High-Performance Computing and Analytics Lab, Rheinische Friedrich-Wilhelms-Universität Bonn,
Friedrich-Hirzebruch-Allee 8, 53115 Bonn, Germany*

[b]*Helmholtz-Institut für Strahlen- und Kernphysik (Theorie) and Bethe Center for Theoretical Physics,
Rheinische Friedrich-Wilhelms-Universität Bonn, Nussallee 14-16, 53115 Bonn, Germany*

[c]*The Cyprus Institute, CASTORC, 20 Konstantinou Kavafi Street, 2121 Nicosia, Cyprus*
  E-mail: bartosz.kostrzewa@uni-bonn.de, s.bacchio@cyi.ac.cy,
  j.finkenrath@cyi.ac.cy, garofalo@hiskp.uni-bonn.de, sromiti@uni-bonn.de,
  urbach@hiskp.uni-bonn.de

We present how we ported the Hybrid Monte Carlo implementation in the tmLQCD software
suite to GPUs through offloading its most expensive parts to the QUDA library. We discuss
our motivations and some of the technical challenges that we encountered as we added the
required functionality to both tmLQCD and QUDA. We further present some performance details,
focussing in particular on the usage of QUDA's multigrid solver for poorly conditioned light quark
monomials as well as the multi-shift solver for the non-degenerate strange and charm sector in
$N_f = 2 + 1 + 1$ simulations using twisted mass clover fermions, comparing the efficiency of
state-of-the-art simulations on CPU and GPU machines. We also take a look at the performance-
portability question through preliminary tests of our HMC on a machine based on AMD's MI250
GPU, finding good performance after a very minor additional porting effort. Finally, we conclude
that we should be able to achieve GPU utilisation factors acceptable for the current generation
of (pre-)exascale supercomputers with subtantial efficiency improvements and real time speedups
compared to just running on CPUs. At the same time, we find that future challenges will require
different approaches and, most importantly, a very significant investment of personnel for software
development.

The 39th International Symposium on Lattice Field Theory (Lattice2022),
8-13 August, 2022
Bonn, Germany

---

*Speaker

1. Introduction

Simulations of lattice QCD by the Extended Twisted Mass Collaboration (ETMC) have evolved [1–3] over the past decade to a point where ensembles of gauge configurations directly at the physical point can reliably be simulated at multiple lattice spacings and in large volumes [4] using the implementation of the Hybrid Monte Carlo (HMC) algorithm in the tmLQCD software suite [5–8], which features hybrid OpenMP/MPI parallelisation, macro-based hardware optimisations for several architectures, nested Omelyan-Mryglod-Folk [9] and force-gradient [10] (FG) integrators, mass-preconditioning as well as a rational approximation for odd numbers of Wilson (clover) fermions or the non-degenerate strange and charm sector of the $N_f = 2 + 1 + 1$ Wilson twisted mass (clover) action.

Moving away from preprocessor macros since about 2015, tmLQCD has been extended with interfaces to the mixed-precision CG (MP-CG) and multishift CG (MS-CG) implementations in QPhiX [11–16] targeting SIMD architectures and DD-αAMG [17–19] for an efficient multigrid-preconditioned (MG-PC) solver for the most poorly conditioned monomials in our molecular dynamics (MD) Hamiltonian. In turn, we have extended these libraries to support our Dirac operators and have shown that MG very effectively reduces the cost of simulations close to or at the physical point [20]. In addition, tmLQCD has been used as glue code by various ETMC contraction codes, providing a simple interface and input file to use these libraries as well as QUDA [21, 22] and its highly efficient [23] MG-PC solver for the Wilson (clover) (twisted mass) operator.

At the precision frontier, observables require future ensembles with even larger volumes at many lattice spacings and with high statistics to ensure adequate control over statistical and systematic uncertainties. Completed, ongoing and planned ETMC ensemble generation efforts in this direction directly at the physical point are shown in the left panel of fig. 1. The right panel of the same figure gives the approximate cost per trajectory (in core hours) of generating ensembles at the physical pion mass using $N_f = 2 + 1 + 1$ flavours of Wilson twised mass clover fermions as a function of the spatial lattice extent. The computational costs for these new simulations are such that our current
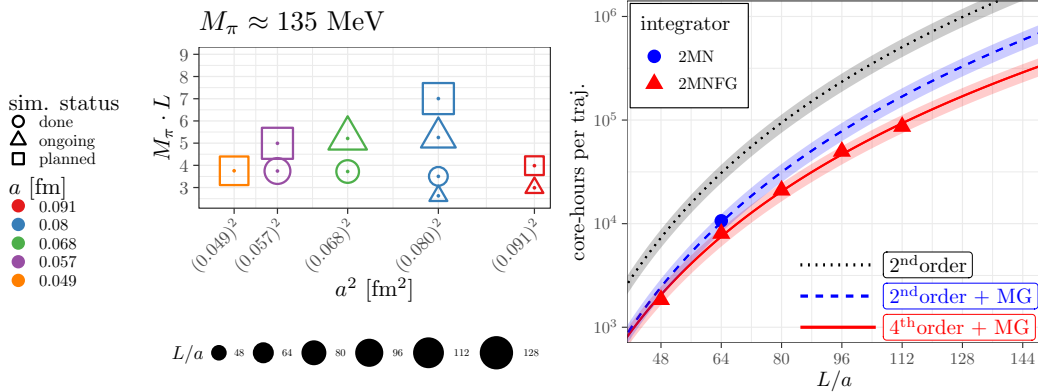

Figure 1: *Left: Overview of the completed (circles), ongoing (triangles) and planned (squares) ETMC ensemble generation at the physical point. The size of the symbol indicates the lattice size. Right: Cost per trajectory (at the physical pion mass) in units of core hours as a function of the lattice extent in lattice units. The points stem from production simulations on different machines at roughly constant acceptance rate while the lines represent fits of the cost models given in eqs.* (1) *to* (3)*, respectively. See text body for more details.*

strategy of generating ensembles on CPU machines while employing GPU machines for physics observables is no longer tenable. The figure relies on *actual* data stemming from different machines with different generations of CPUs and simulations at different lattice spacings with comparable acceptance rates. This data is hence subject to variations in CPU core efficiency, the number of MG iterations as well as the number integration steps required at coarser lattice spacings relative to simulations at finer ones. Mindful of these caveats, it is still possible to fit approximate models to the data with the black dotted, blue dashed and solid red curves given by functions of the forms:

$$C_{\mathrm{2MN,CG}} \approx c_2 \cdot (V/a^4)^{5/4} \tag{1}$$

$$C_{\mathrm{2MN,MG}} \approx c_2' \cdot (V/a^4)^{5/4} \tag{2}$$

$$C_{\mathrm{2MNFG,MG}} \approx c_4 \cdot (V/a^4)^{9/8}, \tag{3}$$

corresponding to three types of setups: MP-CG only with a nested second order integration scheme; the same scheme but employing DD-αAMG for the most poorly conditioned monomials; that same solver setup but employing a FG scheme instead (and appropriate mass-preconditioning). Even with the best setup, a $128^3 \cdot 256$ ensemble costs in excess of $2 \cdot 10^5$ core-hours per trajectory and, when run at a reasonable parallel efficiency, already a $96^3 \cdot 192$ ensemble entails a real time per trajectory of around six hours. Thus, not only would all currently planned runs require $O(10^9)$ core-hours, too much to apply for on available CPU machines, they would also take too much real time to perform without resorting to generating multiple independent Markov chains, complicating subsequent analysis. With the actual or imminent availability of (pre-)exascale machines based on accelerated architectures by various vendors in mind, porting our HMC implementation to these is thus not only necessary but also timely.

2. Even-odd and mass-preconditioned Dirac operators and framework choice

When used as a driver for inversions of the full Dirac operator from within contraction codes, tmLQCD's QUDA interface delegates even-odd preconditioning completely to QUDA by simply setting `solve_type` and `matpc_type` of `QudaInvertParam` correctly for the given solver[1]. In the HMC, however, the even-odd (and mass) preconditioning in the action must match that used in QUDA. To be specific, our degenerate determinant is written in terms of operators $Q^\pm$,

$$Q^\pm = \gamma_5(M_{\mathrm{clov}} \pm i\mu_\ell\gamma_5) \quad \text{s.t.} \quad (Q^+)^\dagger = Q^- \tag{4}$$

$$\det(Q^+Q^-) = \det(Q^2 + \mu_\ell^2), \tag{5}$$

where M_{clov} is the Wilson-clover Dirac operator and μ_ℓ is the light twisted quark mass. We employ asymmetric even-odd preconditioning[2] and thus require an implementation of

$$\hat{Q}^\pm = \gamma_5 \left[(M_{oo} \pm i\mu_\ell\gamma_5) - M_{oe}(M_{ee} \pm i\mu_\ell\gamma_5)^{-1}M_{eo} \right] \tag{6}$$

and for mass-preconditioning, we further require

$$\hat{W}^\pm(\rho) = \hat{Q}^\pm \pm i\rho \quad \text{s.t.} \quad \hat{W}^+(\rho)\hat{W}^-(\rho) = \hat{Q}^+\hat{Q}^- + 2\mu_\ell\rho + \rho^2 \tag{7}$$

---

[1]It should be noted that QUDA's parameter set is very large and that (in most cases) a gamma basis change is required.

[2]`ODD_ODD_ASYMMETRIC` in QUDA-parlance

where ρ is the preconditioning mass, leaving the inverse of the clover term ρ-independent. In the heavy sector we need a two-flavour operator in which the diagonal depends on both $\bar{\mu}$ and $\bar{\epsilon}$:

$$\hat{Q}_h = \gamma_5 \left[(M_{oo} + i\bar{\mu}\gamma_5\tau^3 - \bar{\epsilon}\tau^1) - M_{oe}(M_{ee} + i\bar{\mu}\gamma_5\tau^3 - \bar{\epsilon}\tau^1)^{-1} M_{eo} \right] . \tag{8}$$

When we evaluated different frameworks for implementing our HMC for GPU machines, we considered essentially three factors: availability of a proven-efficient GPU implementation of a MG-PC solver; ease of implementation of eqs. (6) to (8) in the given framework and the availability of a flexible input format for setting up the HMC and RHMC components required for our simulations. Although doing so would have contributed towards solving the wider performance-portability challenge of various ETMC codebases, time limitations ruled out extending Grid [24–26] or the Chroma+QDP-JIT+QUDA tripos [27–30].

The efforts of the QUDA development team towards a device target abstraction and the resulting development or availability of backends for CUDA, HIP, SYCL and OpenMP, clearly favoured a compromise of proceeding with tmLQCD + QUDA for the time being without solving the more general issue. As first steps, we only had to implement: more fine-grained tracking of the state of the host and device gauge fields (and related objects such as the clover term and MG setup); eqs. (7) and (8) as well as an extension of tmLQCD's QUDA interface to support the kinds of solves required within the HMC. We further enabled the gauge derivative to be offloaded and helped to modify QUDA's gradient flow interface such that the values of pertinent flowed observables can be returned to the host application (if desired) in the addition to the flowed gauge field.

3. Laying the groundwork

Before the main implementation phase, we extended the mechanism by which tmLQCD tracks the state of its gauge field (and the various copies thereof, as well as their halos). To this end, we introduced the concept of a `tm_GaugeState_t` struct at program scope which contains a `gauge_id` member, corresponding simply to the position along a trajectory in the HMC, expressed as a real number[3]. Whenever the gauge field is updated, the function `update_tm_gauge_id` is called and functionality exists which cascades to the state of the clover field and its inverse. A similar set of states and functions were introduced to the QUDA interface to track the state of the device gauge and clover fields as well as the MG setup, such that these can be refreshed as necessary. At the cost of efficiency, this allowed us to keep the representative gauge and conjugate momentum fields on the host and to add functionality step by step, allowing us to test our developments while being able to mix QUDA functions with our own CPU functions in different places without having to worry about breaking any synchronisation assumptions.

As a further preparatory step, we implemented a simple profiling mechanism to quantify relative costs associated with different parts of our MD Hamiltonian combined with hierarchical information about the function call tree. While it can be argued that the latter can be obtained using readily available profiling tools, our wish to profile full scale simulations with $O(10)$ monomials employing MG-PC, MP-CG and MS-CG solvers on GPU machines makes this ineffective without explicit (profiler-specific) annotations. Since functions are often called from multiple monomials and sometimes even at different levels of the call tree (depending on context), it may be difficult

---

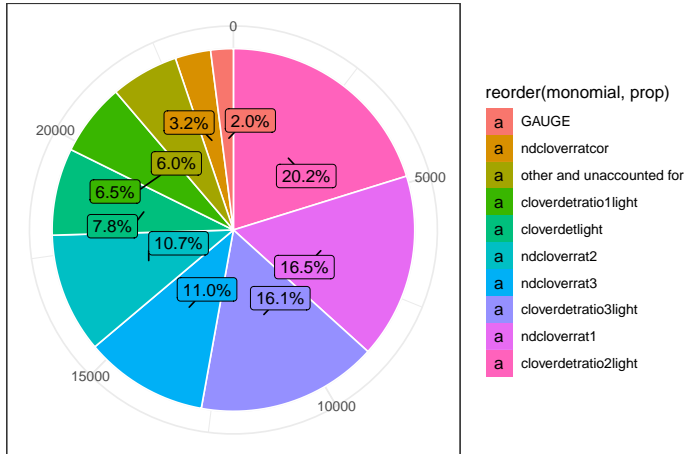[3]or the gauge configuration number when tmLQCD is used as a solver driver from within contraction codes

Figure 2: *Profile of tmLQCD's HMC running multiple trajectories of a $N_f = 2+1+1$, $V/a^4 = 64^3 \cdot 128$ ensemble at the physical point on 16 Marconi-100 nodes (4 NVIDIA V100 GPUs per node, $2 \times$ 16-core IBM POWER9 AC922 CPUs per node). The absolute time in seconds spent in each monomial is shown as well as what fraction of the total this corresponds to.*

to differentiate between physically and algorithmically reasonable hot spots and those caused by incorrect parameter / algorithm choices or implementation / interfacing mistakes. The new stack-based profiler is used through just two functions: `tm_stopwatch_push` and `tm_stopwatch_pop`. The former starts a timer at a given level of the call tree and annotates it with (caller-defined) context information (similar to UNIX paths), while the latter stops the timer of the current level and prints the measured time to `stdout` with the aforementioned context information.

As a specific example, the function `cloverdet_derivative`, which calculates the force due to a Wilson clover (twisted mass) fermion determinant, is bracketed by such calls:

```
1  tm_stopwatch_push(&g_timers, __func__, mnl->name);
2  [...]
3  tm_stopwatch_pop(&g_timers, 0, 1, "");
```

and the same applies to almost all functions further down the call tree, giving output like:

```
1  # TM_QUDA: Time for invert_eo_degenerate_quda 2.937360e-01 s level: 3 proc_id: 0 /HMC/cloverdetlight:clover[...]
2  # : Time for gamma5 5.561520e-04 s level: 3 proc_id: 0 /HMC/cloverdetlight:cloverdet_derivative/solve_degen[...]
3  [...]
4  # : Time for sw_all 4.972867e-01 s level: 2 proc_id: 0 /HMC/cloverdetlight:cloverdet_derivative
5  # : Time for cloverdet_derivative 1.252163e+00 s level: 1 proc_id: 0 /HMC/cloverdetlight:cloverdet_derivative
```

An R script is provided in `profiling/hmc_mk2` which generates a PDF report from such a log file (even if it is incomplete) visualising and tabulating the distribution of the time spent in various parts, as shown exemplarily in fig. 2, where the different labels correspond to different monomials in our MD Hamiltonian (going into the details is beyond the scope of the present proceedings).

4. Implementation details and performance

4.1 Multigrid solver in the light sector

The importance of using a MG-PC solver in simulations close to or at the physical point can hardly be overstated. While one could worry about the (minor) setup overhead at the start of a run and due to setup refreshes or updates of the coarse-grid operators, the algorithmic superiority more than makes up for these. As shown in fig. 3, in computing the fermionic derivative, QUDA's MG is faster than its MP-CG by about a factor of 100 at the physical light quark mass. In practice, we optimize costs by employing a mixture of solvers for different monomials in the light sector, usually MG for the two most poorly conditioned determinant ratios and MP-CG everywhere else, leading to an overall whole-application speedup of about a factor of four compared to using MP-CG alone.
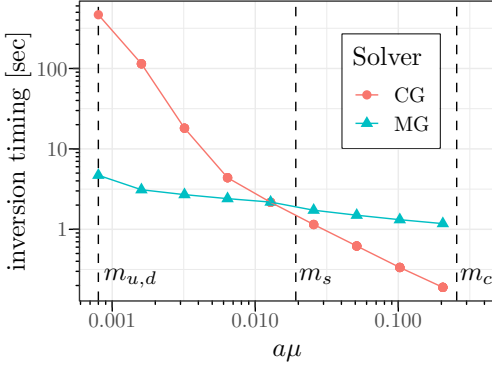
Figure 3: *QUDA-MG and mixed-precision CG compared as a function of the twisted quark mass. The timings refer to a single inversion in the case of CG and two inversions in the case of MG (including some unavoidable overheads), as required in the calculation of the fermion derivative. The dashed vertical lines indicate the physical light and approximate strange and charm quark masses.*

Depending on the type of mass preconditioning employed, using QUDA-MG in the HMC requires some care. To be specific, when ρ in eq. (7) is 20 to 30 times larger than the target quark mass, the setup fails to precondition the fine grid operator well, resulting in hundreds of outer GCR iterations. In determinant ratios with large ρ in the denominator, we counter this by employing MP-CG in the heatbath step while still using MG in the derivative and acceptance step. With more effort, as was done for DD-αAMG [18], QUDA-MG could be modified to more effectively deal with eq. (7). Another inefficiency which can perhaps be resolved in the future, is that coarsening currently only supports the symmetrically even-odd-preconditioned operator. We simply ignore this and set:

- `matpc_type = QUDA_MATPC_ODD_ODD` in `QudaMultigridParam.invert_param`
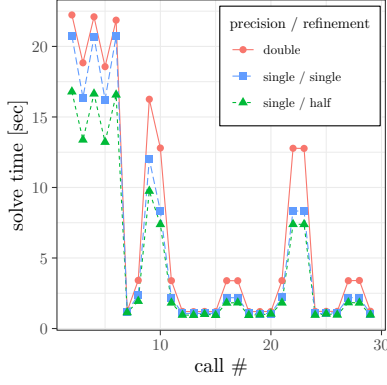- `matpc_type = QUDA_MATPC_ODD_ODD_ASYMMETRIC` in the outer solver,

likely resulting in slightly higher iteration counts than with a more consistent setup.

4.2 Mixed-precision multi-shift solver in the heavy sector

As can be seen in fig. 2, a significant fraction of total runtime is spent in monomials simulating partial fractions of the rational approximation of the non-degenerate $1 + 1$ sector of our action (`ndcloverratN` and `ndcloverratcor` in the figure). In simulations on CPU machines, the calculation of the correction term for the rational approximation [31, 32] contributes significantly to the total cost. This is shown in the table of fig. 4: using only QPhiX MS-CG takes about 1360 seconds and dropts to around 990 seconds employing DD-αAMG refinement [19]. On 32 NVIDIA A100 GPUs instead, this takes only 220 seconds using double precision MS-CG and improves down to around 170 seconds when single precision MS-CG with double-half shift-by-shift refinement is used. Similar improvements are seen in all calls of MS-CG, as shown in the left panel of fig. 4.

4.3 Efficiency

In our profiles, we can identify monomials which are very much GPU-dominated as shown in the left panel of fig. 5, where almost all of the time is spent in the solver driver (`solve_degenerate`), the corresponding QUDA function (`invertQuda`) or unavoidable overheads related to the MG solver. On the other hand, as shown in the right panel of the same figure, we still have monomials with very poor offloading fractions. In the force calculation for the rational approximation, outer products must be computed on multiple two-flavour spinors and for the largest shifts, the solver requires only very few iterations. Since we have not yet extended QUDA to support exactly what we require, these outer products on the CPU thus dominate total cost in this case.

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

CPU: 3072 Intel Xeon Platinum 8168 cores (64 Juwels nodes)

GPU: 32 A100 + 384 EPYC Rome 7402 cores (8 Juwels Booster nodes)

Figure 4: *Left: Timings of MS-CG calls along a trajectory ($64^3 \cdot 128$ lattice) running on 32 A100 GPUs (8 Juwels Booster nodes) using full double precision (red circles), single precision MS-CG with double-single shift-by-shift refinement (blue squares) or single precision MS-CG with double-half shift-by-shift refinement (green triangles). Right: Comparison of timings of the rational approximation correction term in the same simulation between running on 3072 CPU cores (64 Juwels nodes) and 32 A100 GPUs.*
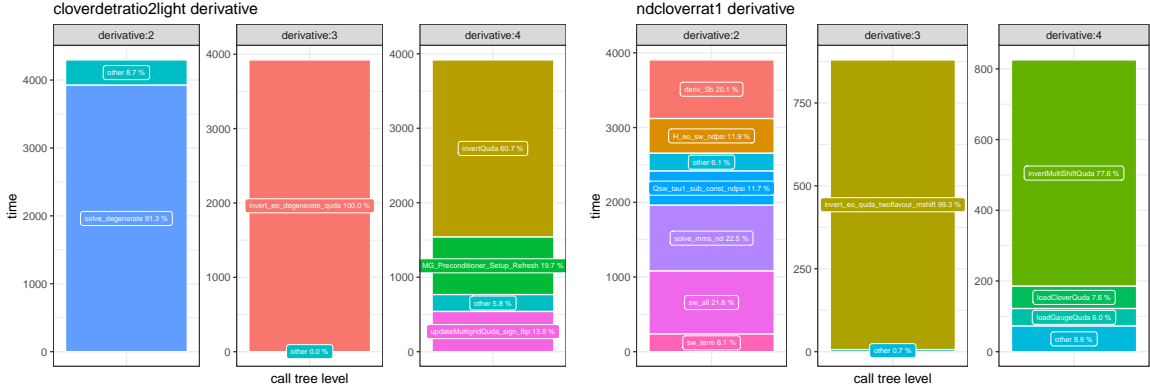
Figure 5: *Left: Hierarchical profile of the force calculation of a monomial (see also fig. 2) dominated by the* `solve_degenerate` *function running mostly on the GPU (specifically* `invertQuda` *and unavoidable MG overheads). Right: Profile of a monomial derivative where only about 20% is offloaded to the GPU (*`solve_mms_nd`*).*

Even with these inefficiencies we reach overall GPU utilisation fractions between 30 and 70% depending on the machine, the size of the machine partition and the simulation parameters (larger partitions, lighter quark mass → higher utilisation). In order to *attempt* an apples-to-apples comparison, fig. 6 shows the cost per trajectory in node-hours on 3072 Intel Xeon Platinum 8168 cores (64 Juwels nodes) compared to the cost in GPU-hours on 32 NVIDIA A100 (8 Juwels Booster nodes) for a $64^3 \cdot 128$ ensemble at the physical point. Since a Juwels node uses about a factor of four less power than a Juwels Booster node, we can estimate that in addition to reducing real time by a factor of around 1.7, energy efficiency has improved by about a factor of three, with further improvement expected when the remaining parts of the fermionic force are offloaded.
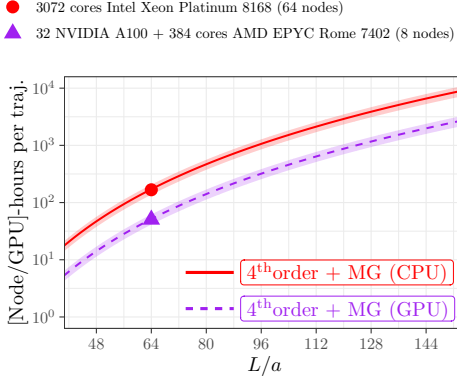
machine	real time	CPU node-hrs / GPU-hrs	kWh
64 nodes (Juwels)	2.61 h	167	~ 84
32 GPUs (Juwels Booster)	1.58 h	50.6	~ 24

Figure 6: *Left:* *Cost per trajectory for a $64^3 \cdot 128$ simulation at the physical point (in node-hours for the CPU machine, GPU-hours for the GPU machine).* *Right:* *Real time and cost per trajectory on 64 Juwels nodes and 8 Juwels Booster nodes, respectively. Estimate of the respective power usage per trajectory.*
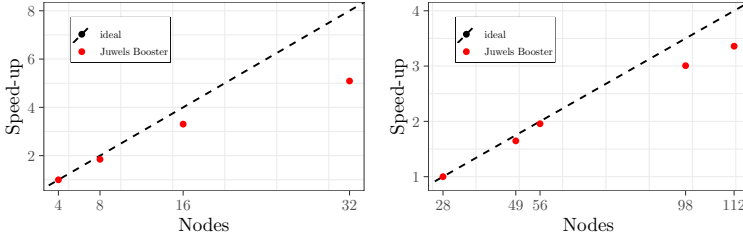

Figure 8: *Left:* *Strong-scaling of a $64^3 \cdot 128$ simulation at the physical point from 4 to 32 Juwels Booster nodes.* *Right:* *Strong-scaling of a $112^3 \cdot 224$ simulation at the physical point from 28 to 112 Juwels Booster nodes.*

4.4 Strong-scaling and performance-portability

In fig. 8 we show full-application behaviour for a $64^3 \cdot 128$ simulation scaled from 4 to 32 Juwels Booster nodes and a $112^3 \cdot 224$ simulation scaled from 28 to 112 nodes. While strong scaling is quite good, it will worsen once the remaining parts of the fermionic force are offloaded as it will become more dominated by the scalability of the MG algorithm.

Finally, to demonstrate the performance-portability of our approach, in fig. 7 we show the time per trajectory of thermalised $32^3 \cdot 64$ simulations running at different pion masses on a single node of Juwels Booster (4 NVIDIA A100 GPUs) and a single node of a *preliminary* test system at JSC (4 AMD MI250 GPUs). It should be noted that the environment (ROCm version, drivers, software stack) on the AMD-based test system was not at all final and that none of the algorithmic parameters, in particular the MG setup, were retuned for the MI250-based machine. Still, with just a little bit of work on the tmLQCD *build system*, we were able to run tmLQCD + QUDA unmodified with good performance on an architecture wich we had never used before.
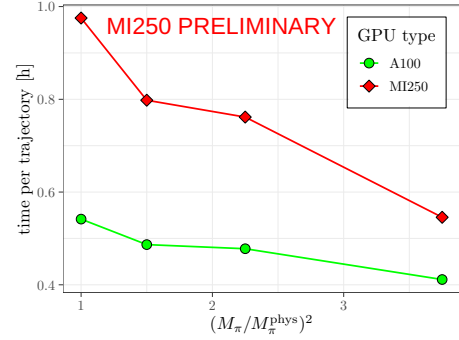

Figure 7: *Time per trajectory for well-thermalised $N_f = 2 + 1 + 1, V/a^4 = 32^3 \cdot 64$ simulations from about $M_\pi \sim 250\,MeV$ down to the physical point (using the same number of integration steps for all pion masses) on a single Juwels Booster node with 4 A100 GPUs (green circles) and a single node of a test system at JSC with 4 MI250 GPUs.*

5. Conclusions

We have shown in this contribution that employing QUDA through its C interface to offload the most expensive parts of the HMC is suitable for the current generation of GPU-accelerated super-computers. Although we still need to offload a number of parts of the fermion force, we expect to eventually be able to reach GPU utilisations between 60 and 70% on (pre-)exascale machines for state-of-the-art simulations. Because we are dependent on a backend being available for a particular architecutre, QUDA is likely not a general solution for our performance-portability problem, for instance to target future CPU generations. We will also run into issues if very dense GPU configurations are coupled with weak driver CPUs to keep total power consumption under control, as we would need to offload a much larger fraction of tmLQCD's functionality, necessitating support for different memory layouts and execution spaces and thus essentially a complete rewrite. Further improvements to scalability could be obtained through task parallelism (e.g. multiple monomials on the same time scale, the n-th root trick or through modular supercomputing architectures [33]) at the cost of substantial refactoring. These improvements could be achieved by moving to Grid or perhaps by using Kokkos [34] as a performance-portability layer. In addition, the interfacing layer provided by Lyncs [35] could allow combining different libraries (Grid and QUDA, for example) and enable both task parallelism and higher programmer productivity. All of these challenges, how-ever, require a significant personnel investment into software development for lattice field theory, ideally as a community effort.

References

[1] **ETM** Collaboration, A. Abdel-Rehim *et al.*, Phys. Rev. D **95**, 094515 (2017), arXiv:1507.05068 [hep-lat].

[2] C. Alexandrou *et al.*, Phys. Rev. D **98**, 054518 (2018), arXiv:1807.00495 [hep-lat].

[3] **Extended Twisted Mass** Collaboration, C. Alexandrou *et al.*, Phys. Rev. D **104**, 074520 (2021), arXiv:2104.06747 [hep-lat].

[4] J. Finkenrath *et al.*, PoS **LATTICE2021**, 284 (2022), arXiv:2201.02551 [hep-lat].

[5] K. Jansen and C. Urbach, Comput. Phys. Commun. **180**, 2717 (2009), arXiv:0905.3331 [hep-lat].

[6] A. Abdel-Rehim *et al.*, PoS **LATTICE2013**, 414 (2014), arXiv:1311.5495 [hep-lat].

[7] A. Deuzeman, K. Jansen, B. Kostrzewa and C. Urbach, PoS **LATTICE2013**, 416 (2014), arXiv:1311.4521 [hep-lat].

PoS(LATTICE2022)340

[8] C. Urbach *et al.*, https://github.com/etmc/tmLQCD .

[9] I. Omelyan, I. Mryglod and R. Folk, Computer Physics Communications **151**, 272 (2003).

[10] M. A. Clark, B. Joo, A. D. Kennedy and P. J. Silva, Phys. Rev. D **84**, 071502 (2011), arXiv:1108.1828 [hep-lat].

[11] B. Joó *et al.*, Lattice qcd on intel xeon phi coprocessors, in *Supercomputing*, edited by J. M. Kunkel, T. Ludwig and H. W. Meuer, pp. 40–54, Berlin, Heidelberg, 2013, Springer Berlin Heidelberg.

[12] S. Heybrock *et al.*, Lattice qcd with domain decomposition on intel® xeon phi co-processors, in *SC '14*, pp. 69–80, 2014.

[13] B. Joó, D. D. Kalamkar, T. Kurth, K. Vaidyanathan and A. Walden, Optimizing wilson-dirac operator and linear solvers for intel® knl, in *International Conference on High Performance Computing*, pp. 415–427, Springer, 2016.

[14] M. Schröck, S. Simula and A. Strelchenko, PoS **LATTICE2015**, 030 (2016), arXiv:1510.08879 [hep-lat].

[15] B. Joó, D. D. Kalamkar, T. Kurth, K. Vaidyanathan and A. Walden, Optimizing wilson-dirac operator and linear solvers for intel® knl, in *High Performance Computing*, edited by M. Taufer, B. Mohr and J. M. Kunkel, pp. 415–427, Cham, 2016, Springer International Publishing.

[16] B. Joó *et al.*, https://github.com/JeffersonLab/qphix .

[17] A. Frommer, K. Kahl, S. Krieg, B. Leder and M. Rottmann, SIAM J. Sci. Comput. **36**, A1581 (2014), arXiv:1303.1377 [hep-lat].

[18] C. Alexandrou *et al.*, Phys. Rev. D **94**, 114509 (2016), arXiv:1610.02370 [hep-lat].

[19] C. Alexandrou, S. Bacchio and J. Finkenrath, Comput. Phys. Commun. **236**, 51 (2019), arXiv:1805.09584 [hep-lat].

[20] S. Bacchio, C. Alexandrou and J. Finkenrath, EPJ Web Conf. **175**, 02002 (2018), arXiv:1710.06198 [hep-lat].

[21] M. A. Clark, R. Babich, K. Barros, R. C. Brower and C. Rebbi, Comput. Phys. Commun. **181**, 1517 (2010), arXiv:0911.3191 [hep-lat].

[22] R. Babich *et al.*, Scaling Lattice QCD beyond 100 GPUs, in *SC '11*, 2011, arXiv:1109.2935 [hep-lat].

[23] M. A. Clark *et al.*, Accelerating Lattice QCD Multigrid on GPUs Using Fine-Grained Parallelization, in *SC '16*, pp. 795–806, 2016, arXiv:1612.07873 [hep-lat].

[24] P. A. Boyle, G. Cossu, A. Yamaguchi and A. Portelli, PoS **LATTICE2015**, 023 (2016).

[25] D. Richtmann, P. A. Boyle and T. Wettig, PoS **LATTICE2018**, 032 (2019), arXiv:1904.08678 [hep-lat].

[26] D. Richtmann, N. Meyer and T. Wettig, MRHS multigrid solver for Wilson-clover fermions, in *39th International Symposium on Lattice Field Theory*, 2022, arXiv:2211.13719 [hep-lat].

[27] **SciDAC, LHPC, UKQCD** Collaboration, R. G. Edwards and B. Joo, Nucl. Phys. B Proc. Suppl. **140**, 832 (2005), arXiv:hep-lat/0409003.

[28] F. Winter, arXiv:1111.5596 [hep-lat].

[29] F. T. Winter, PoS **LATTICE2013**, 042 (2014).

[30] F. Winter, M. Clark, R. Edwards and B. Joó, A framework for lattice qcd calculations on gpus, in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pp. 1073–1082, 2014.

[31] M. A. Clark and A. D. Kennedy, Phys. Rev. Lett. **98**, 051601 (2007), https://link.aps.org/doi/10.1103/PhysRevLett.98.051601.

[32] M. Lüscher, Computational strategies in lattice qcd, 2010.

[33] E. Suarez, N. Eicker and T. Lippert, Modular supercomputing architecture: from idea to production, in *Contemporary high performance computing*, pp. 223–255, CRC Press, 2019.

[34] H. Carter Edwards, C. R. Trott and D. Sunderland, Journal of Parallel and Distributed Computing **74**, 3202 (2014), https://www.sciencedirect.com/science/article/pii/S0743731514001257, Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[35] S. Bacchio, J. Finkenrath and C. Stylianou, PoS **LATTICE2021**, 542 (2022), arXiv:2201.03873 [hep-lat].

[36] Jülich Supercomputing Centre, Journal of large-scale research facilities **5** (2019), http://dx.doi.org/10.17815/jlsrf-5-171.