PROCEEDINGS OF SCIENCE

Autotuning multigrid parameters in the HMC on different architectures

Marco Garofalo,[a] **Bartosz Kostrzewa,**[a,*] **Simone Romiti**[b] **and Aniket Sen**[a]

[a] *Helmholtz-Institut für Strahlen und Kernphysik (Theory), Rheinische Friedrich-Wilhelms-Universität Bonn, Nussallee 14-16, 53115 Bonn, Germany*

[b] *Institute for Theoretical Physics, Albert Einstein Center for Fundamental Physics, University of Bern, CH-3012 Bern, Switzerland*

E-mail: garofalo@hiskp.uni-bonn.de, kostrzewa@hiskp.uni-bonn.de, simone.romiti@unibe.ch, sen@hiskp.uni-bonn.de

Multigrid-preconditioned solvers have proven crucial for the efficient generation of ensembles of gauge configurations at physical quark mass parameters. A high performance implementation of such a solver for GPUs by different vendors and for different types of Wilson fermions is provided in the QUDA library. It features an autotuner which chooses an optimal communication policy and an optimal set of kernel launch parameters for each kernel, problem size and domain decomposition on each architecture.

The performance of the multigrid solver additionally depends on a large number of algorithmic parameters such as block sizes, numbers of vectors, maximum iterations as well as convergence thresholds. Many of these parameters have to be tuned on a per-level basis, making the search space large and an exhaustive approach essentially computationally intractable. In addition, once a good parameter set is found, in general it will fail to be optimal on a different machine or for a different domain decomposition.

We present a simple autotuner for these parameters implemented in the tmLQCD software suite which requires only some intuition on the order in which parameters are to be tuned and the step sizes to be used in the tuning procedure. The simple approach converges quickly, producing stable iteration counts and times-to-solution across gauge configurations of a given ensemble. We demonstrate its applicability to the adjustment of multigrid setups between different machines or different sets of physical parameters on the basis of results from machines with NVIDIA and AMD GPUs.

The 41st International Symposium on Lattice Field Theory (LATTICE2024)
28 July - 3 August 2024
Liverpool, UK

*Speaker

PoS(LATTICE2024)276

1. Introduction

The generation of ensembles of gauge configurations in lattice QCD relies on the Hybrid Monte Carlo (HMC) [1] algorithm. The simulations of the Extended Twisted Mass Collaboration (ETMC) are performed using the tmLQCD software suite [2–5] which features hybrid OpenMP/MPI parallelisation and nested Omelyan-Mryglod-Folk [6] and force-gradient [7] integrators. In an $N_f = 2 + 1 + 1$ simulation using Wilson twisted mass (clover) fermions the light determinant is simulated using several determinant ratios employing Hasenbusch mass splitting [8] and the strange and charm sector relies on a rational approximation [9] split over multiple partial fractions. Spread over multiple time scales [10], simulations close to or at the physical point with twisted mass fermions are made efficient using multigrid (MG) preconditioning [11] which significantly reduces the cost of the most poorly conditioned monomials in the molecular dynamics Hamiltonian.

Support for GPUs in tmLQCD is provided through an interface [12] to the QUDA [13, 14] library which also features a highly efficient MG-preconditioned solver [15] for the Wilson (clover) (twisted mass) operator. The evolution of the performance of this *tmLQCD + QUDA* setup is shown in the left panel of fig. 1 which compares time per trajectory in the simulation of a $64^3 \cdot 128$ ensemble at the physical point on 512 SuperMUC nodes to the timing on eight JUWELS [16] Booster [17] nodes at three stages of development: (1) with just the fermionic inversions and the gauge force offloaded, (2) with the light quark force terms offloaded and (3) with all force calculations offloaded to QUDA. The right panel of the same figure shows a similar comparison for a $112^3 \cdot 224$ ensemble at the physical point originally run on 343 Frontera [18] nodes and the same simulation running on 28 LUMI-G nodes. The speedup in real time per trajectory is around 2.8 and 2.6, respectively, and the comparatively small number of GPUs required implies an overall improvement in energy efficiency of around one order of magnitude.

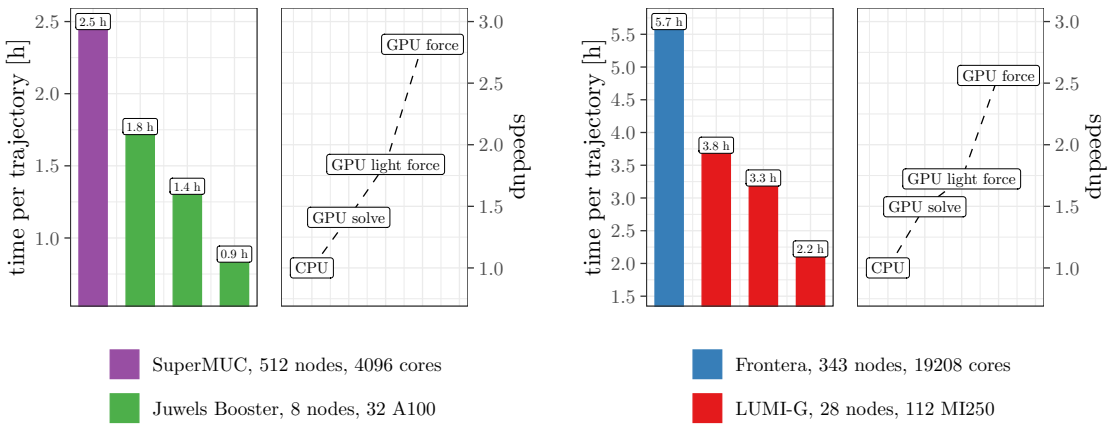

Figure 1: ***Left:*** *Time per unit length trajectory of tmLQCD + QUDA (green) for a* $64^3 \cdot 128$ *ensemble at the physical point compared to the machine it was originally generated on (purple) using tmLQCD + DD-αAMG [19]. The different speedups correspond to increasing levels of QUDA offloading.* ***Right:*** *The same kind of comparison between tmLQCD + QUDA and tmLQCD + DD-αAMG + QPhiX [20–22] running a* $112^3 \cdot 224$ *ensemble at the physical point on LUMI-G and Frontera, respectively.*

parameter	symbol	sensible choices
`mg-mu-factor`	μ_ℓ	$5 \cdot 5 \cdot 15 = 375$
`mg-coarse-solver-tol`	r_ℓ	$4^2 = 16$
`mg-coarse-solver-maxiter`	n_ℓ	$4^2 = 16$
`mg-nu-post`	ν_ℓ^{post}	$4^3 = 64$
`mg-nu-pre`	ν_ℓ^{pre}	$4^3 = 64$
`mg-smoother-tol`	r_ℓ^s	$3^3 = 9$
`mg-omega`	ω_ℓ	$3^3 = 9$
total		$\approx 10^{10}$ combs

parameter	sensible choices
`mg-block-size`	$\approx (2^4)^2 = 256$
`mg-nvec`	$\approx 2^2 = 4$
total	≈ 1024 combs

Table 1: *Subset of tunable algorithmic parameters for the QUDA MG solver and estimates of the number of "sensible" choices which would need to be tested in an exhaustive search.* ***Left:*** *Parameters for which the MG setup does not have to be regenerated.* ***Right:*** *Parameters for which the MG setup has to be regenerated.*

2. Tuning the Multigrid Solver

A key ingredient enabling this efficiency at the physical pion mass is a tuned three-level MG setup. In the inversion for the force calculation of the most poorly conditioned determinant ratio, the MG is around 100 times faster [12] than the fastest mixed-precision conjugate gradient solver in QUDA. This comes at the cost of having to tune a large number of algorithmic parameters on multiple levels of the MG aggregation hierarchy. The search space for these algorithmic parameters consists of two types: those that need the MG setup to be regenerated, such as the block sizes or the number of null vectors, and those which do not. Of the latter we concern ourselves with the set shown on the left-hand side of table 1 using the names employed for the corresponding command-line arguments of the built-in test programs of the QUDA library.

For each parameter p_i^ℓ we can conceive of a number of "sensible" choices as shown in table 1, resulting in around 10^{13} combinations in a naïve outer product, rendering an exhaustive search intractable. Luckily we may be able to search the space more efficiently relying on some intuition:

1. the metric is time-to-solution: t_{new} at each tuning step and the best time t_{best},
2. we always start at the coarsest grid and work our way up,
3. we start with the most relevant parameter at each level,
4. we stop moving into a particular direction when t_{new} does not improve more than some factor $\{\delta \,|\, \delta < 1.0\}$, that is, $t_{\text{new}} > \delta \cdot t_{\text{best}}$ (or if we reach the maximum number of steps for that direction),
5. if the solver does not yet converge after a step into a particular direction we continue tuning into that direction in order to not "give up" too early[1],
6. to avoid being misled by fluctuations, we accept the result of a particular tuning direction only if the improvement is better than a factor $\{\rho \,|\, \delta < \rho < 1.0\}$, that is, $t_{\text{new}} < \rho \cdot t_{\text{best}}$,
7. if the improvement is not good enough, we reset the parameter in question to its value in the previously best setup and move on to the next parameter and/or level.

The full procedure as well as the parameter names used in the tmLQCD input file are spelled out in the LaTeX-documentation shipped with tmLQCD[2] and a pseudocode is given in algorithm 1.

[1] this allows the auto-tuner to find a setup which at least converges if the initial parameters were especially poor

[2] The implementation itself can be found in the `quda_mg_tune_params` function and its children in

Algorithm 1 Approach for the tuning of the parameter set $\{p_i^\ell\}$ on all levels $\{\ell\}$

Number of gauge configurations: $N_{\text{conf}} \approx 5$
Number of tuning iterations per gauge configuration: $N_{\text{tune}} \approx 100$ to 300
Number of levels: $N_\ell = 3$
Parameters on level ℓ: $p_i^\ell \in \{\mu_\ell, r_\ell, n_\ell, \nu_\ell^{\text{post}}, \nu_\ell^{\text{pre}}, r_\ell^s, \omega_\ell\} = \vec{p}^\ell$
Change in parameter p_i^ℓ per tuning step: $\pm\Delta_i^\ell$
Number of tuning steps in direction p_i^ℓ: n_i^ℓ
Tuning tolerance (δ) and ignore threshold (ρ): $\{\delta \,|\, 0.99 < \delta < 1.0\}$; $\{\rho \,|\, \delta < \rho < 1.0\}$
Maximum number of iterations for the (outer) solver: $n_{\text{iter}}^{\text{max}} \approx 400$
for $i_{\text{conf}} \in 1, 2, \ldots, N_{\text{conf}}$ **do**
    if $i_{\text{conf}} = 1$ **then**
        invert $Dx = b$ with initial MG parameter set $\{p_i^\ell\} \rightarrow t_{\text{best}}$
        $\{p_i^\ell\}_{\text{best}} = \{p_i^\ell\}$
    end if
    for $i_{\text{tune}} \in 1, 2, \ldots, N_{\text{tune}}$ **do**
        for $\ell \in (N_\ell - 1), (N_\ell - 2), \ldots, 0$ **do**
            for $p_i^\ell \in p_1^\ell, p_2^\ell, \ldots, p_N^\ell$ **do**
                for $n \in 1, 2, \ldots, n_i^\ell$ **do**
                    $p_i^\ell = p_i^\ell \pm \Delta_i^\ell$
                    invert $Dx = b$ using MG parameter set $\{p_i^\ell\} \rightarrow t_{\text{new}}$
                    if $(n_{\text{iter}} < n_{\text{iter}}^{\text{max}})$ && $(t_{\text{new}} > \delta \cdot t_{\text{best}})$ **then**
                        break
                    end if
                end for
                if $t_{\text{new}} < \rho \cdot t_{\text{best}}$ **then**
                    $t_{\text{best}} = t_{\text{new}}$
                    $\{p_i^\ell\}_{\text{best}} = \{p_i^\ell\}$
                else
                    $\{p_i^\ell\} = \{p_i^\ell\}_{\text{best}}$
                end if
            end for
        end for
    end for
end for

For twisted mass fermions in the MG without coarse-grid-deflation, the most relevant parameter is the *coarse μ scaling factor*, denoted as μ_ℓ in table 1. It was found [19] that the coarse-grid operator develops a large number of small eigenvalues close to the target twisted quark mass, severely slowing down convergence on the coarse grid. Because only poor accuracy is required on the coarse level, the reduction of the density of small eigenvalues obtained by scaling the twisted quark mass by μ_ℓ in the coarse-grid operator significantly improves time-to-solution. Close to the physical point the solver might not converge at all without such a scaling factor and for QUDA it appears that factors between 30 and 150 perform best depending on the target quark mass and lattice spacing.

The next most relevant parameters are the coarse-grid solver tolerance and the maximum number of coarse-grid iterations, r_ℓ and n_ℓ, respectively. While one may think that tuning just the tolerance would be sufficient, we find in practice that tuning both is beneficial. On certain gauge configurations the tuned coarse-grid tolerance might only be reached after many coarse-grid

`quda_interface.c` of the tmLQCD master branch [5]. The autotuner is driven by the `deriv_mg_tune` executable.

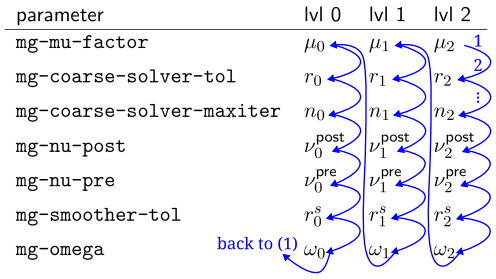

Table 2: *Order of the tuning of the various algorithmic parameters of the QUDA MG solver on the finest (0), intermediate (1) and coarsest (2) levels of the MG aggregation hierarchy.*

iterations, slowing down the solver as a whole (depending on the architecture). The n_ℓ parameter then helps to limit the number of coarse-grid iterations while a sufficiently stringent r_ℓ helps to maintain an acceptable number of intermediate and fine grid iterations. The remaining parameters enable further fine-tuning and we show the order in which they are swept through in table 2.

3. Dependence on the Target Architecture

The original motivation for the development of this autotuner came from the early-access phase on LUMI-G where we observed relatively poor performance of the coarse-grid operator in QUDA on AMD GPUs compared to NVIDIA GPUs. As shown in the left panel of fig. 2, for the smallest per-node coarse-grid volumes, we observed up to a factor of 10 performance degradation on LUMI-G compared to JUWELS Booster. This resulted in the time-to-solution of the MG solver increasing by about a factor of seven when a production MG setup was moved from JUWELS Booster to LUMI-G, and the time per trajectory by a factor of two or more.

For larger per-node coarse-grid volumes the performance gap decreases and the autotuner was able to find a different balance between coarse, intermediate and fine grid iterations, reducing the overall degradation to less than a factor of two and the impact on the time per trajectory to a small factor close to one. With recent updates of the ROCm version on LUMI-G the performance gap has reduced substantially as shown in the right panel of fig. 2. For a per-node volume corresponding to an intermediate-grid operator in a representative MG setup, the operator actually appears to be faster LUMI-G (rightmost point).

4. Tuning Strategies and Example Runs

A tuning run requires a sensible initial parameter set, $\{p_i^\ell\}_{\text{init}}$ and an appropriate increment (or decrement) as well as a maximum number of tuning iterations for each parameter on each level. Two strategies seem to produce reasonable results: (1) starting from a parameter set for which the solver converges well but which is expensive (stringent tolerances, many coarse-grid iterations, many smoothing iterations) or (2) starting from a parameter set for which the solver barely converges[3] (lax tolerances, few coarse-grid iterations and a small number of smoothing iterations).

The first strategy, which we term "tuning from above", is aimed at reducing the cost of the solver per fine-grid iteration in order to improve time-to-solution. Consequently, maximum iteration counts are tuned into the negative direction while tolerances are tuned into the positive direction. An

[3] or even does not converge at all

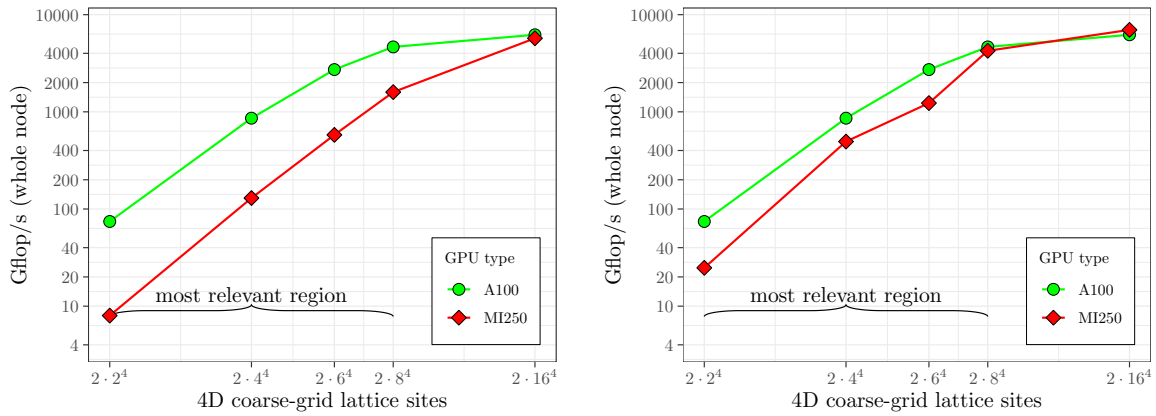

Figure 2: ***Left:*** *Single-precision QUDA coarse Dslash benchmark (24 colours) from October 2022 on a single JUWELS Booster node (4 × A100) using an unknown QUDA commit and CUDA version (**green**) and a single LUMI-G node (4 × MI250) using QUDA commit* `dd6207e6e` *and HIP version 50221153 (**red**) as a function of the coarse-grid lattice volume. The indicated regions corresponds to the per-node coarsest-grid lattice volume in realistic scenarios.* ***Right:*** *The same comparison between one node (4 × A100) of the Marvin HPC cluster at the University of Bonn (**green**) using CUDA 12.1 and a single LUMI-G node (**red**) using HIP 50631062 in July 2024 using commit* `6198d60a7` *of the QUDA* `develop` *branch on both machines.*

parameter	symbol	lvl 0	lvl 1	lvl 2
`mg-mu-factor`	μ_ℓ	1.0 , 0.125 , 10	1.0 , 0.25 , 10	30.0 , 5.0 , 20
`mg-coarse-solver-tol`	r_ℓ	– , – , 0	0.05 , 0.10 , 10	0.05 , 0.10 , 10
`mg-coarse-solver-maxiter`	n_ℓ	– , – , 0	50 , -5 , 10	50 , -5 , 10
`mg-nu-post`	ν_ℓ^{post}	6 , -2 , 3	6 , -2 , 3	6 , -2 , 3
`mg-nu-pre`	ν_ℓ^{pre}	6 , -2 , 3	6 , -2 , 3	6 , -2 , 3
`mg-smoother-tol`	r_ℓ^s	0.05 , 0.10 , 5	0.05 , 0.10 , 5	0.05 , 0.10 , 5
`mg-omega`	ω_ℓ	0.85 , 0.05 , 4	0.85 , 0.05 , 4	0.85 , 0.05 , 4

Table 3: *Initial parameters, increments and maximum number of steps in a "tuning from above" strategy for a* $112^3 \cdot 224$ *ensemble at* M_π^{phys} *on 28 LUMI-G nodes on the finest (0), intermediate (1) and coarsest (2) levels of the MG aggregation hierarchy.*

example of such as set of initial parameters, increments and maximum number of tuning iterations for each p_i^ℓ is shown in table 3 for a $112^3 \cdot 224$ ensemble tuned on 28 LUMI-G nodes. The evolution of the corresponding run is shown in fig. 3 in which the top two panels show the history of the number of outer solver iterations and the time-to-solution, respectively, and the other panels show the evolution of the $\{p_i^\ell\}$. Dashed vertical lines indicate when the gauge configuration is switched and the time-to-solution evolves from around 36 seconds to between seven and ten seconds.

The other strategy, termed "tuning from below", aims instead to increase the number of coarse-grid iterations as little as possible and to lower the tolerances as little as necessary starting from low iteration limits and lax tolerances. The increments are thus positive for the former and negative for the latter as given in table 4 with the corresponding evolution shown in fig. 4 from non-convergence (46 seconds at 600 iterations) down to around seven seconds. In both strategies the search direction for μ_ℓ is chosen to be positive because it was found that this is more reliable.

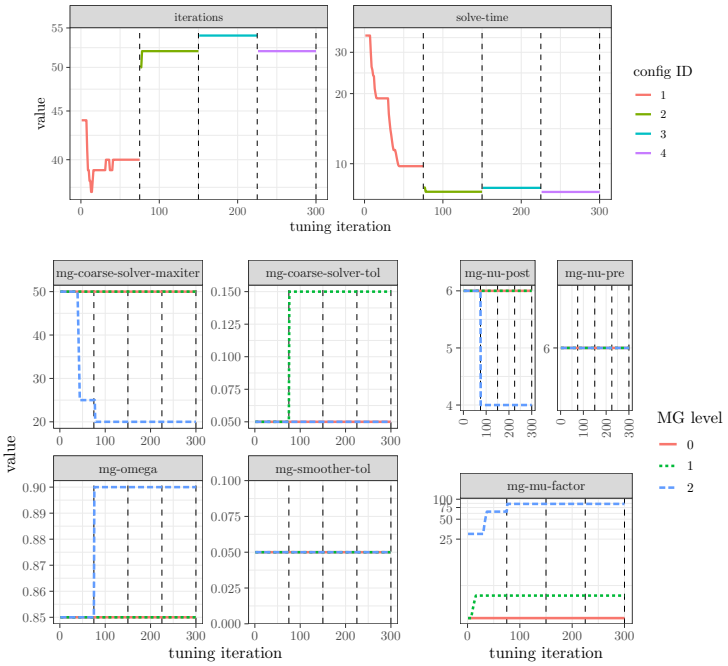

Figure 3: *Tuning history using the "tuning from above" strategy for a* $112^3 \cdot 224$ *ensemble at* M_π^{phys} *on 28 LUMI-G nodes starting from the parameters in table 3. The top panels show the number of outer solver iterations and the time-to-solution while the lower panels show the different parameter values with colours and line types indicating the MG level. The dashed vertical lines indicate the switching of gauge configurations.*

parameter	symbol	lvl 0	lvl 1	lvl 2
`mg-mu-factor`	μ_ℓ	1.0 , 0.125 , 10	1.0 , 0.25 , 10	30.0 , 5.0 , 20
`mg-coarse-solver-tol`	r_ℓ	– , – , 0	0.55 , -0.10 , 10	0.55 , -0.10 , 10
`mg-coarse-solver-maxiter`	n_ℓ	– , – , 0	5 , 5 , 10	5 , 5 , 10
`mg-nu-post`	ν_ℓ^{post}	1 , 2 , 4	1 , 2 , 4	1 , 2 , 4
`mg-nu-pre`	ν_ℓ^{pre}	1 , 2 , 4	1 , 2 , 4	1 , 2 , 4
`mg-smoother-tol`	r_ℓ^s	0.55 , -0.10 , 5	0.55, -0.10 , 5	0.55 , -0.10 , 5
`mg-omega`	ω_ℓ	0.85 , 0.05 , 4	0.85 , 0.05 , 4	0.85 , 0.05 , 4

Table 4: *Initial parameters, increments and maximum number of steps in a "tuning from below" strategy for a* $112^3 \cdot 224$ *ensemble at* M_π^{phys} *on 28 LUMI-G nodes on the finest (0), intermediate (1) and coarsest (2) levels of the MG aggregation hierarchy.*

5. Conclusion and Outlook

We have presented a simple approach for semi-automatically tuning a subset of the algorithmic parameters of the QUDA MG solver from within tmLQCD. With sensible starting parameters the tuning procedure is seen to converge in a few hundred inversions to an efficient MG setup even starting from non-convergence. We have so far found the autotuner to be quite useful when moving from one machine to another or when the domain decomposition is changed. This applies to both the HMC and measurement campaigns as it can also fine-tune MG setups with coarse-grid deflation.

It is currently not integrated into the HMC and thus relies on first generating a small number of reasonably thermalised gauge configurations less efficiently and then fine-tuning the MG setup as the simulation evolves. The results currently also have to be manually transcribed into input files for production runs such that tighter integration with the tmLQCD solver driver would be a welcome quality-of-life improvement, as would support for automatically tuning parameters such as block sizes, which require (part of) the setup to be regenerated. Finally, an integration into the QUDA

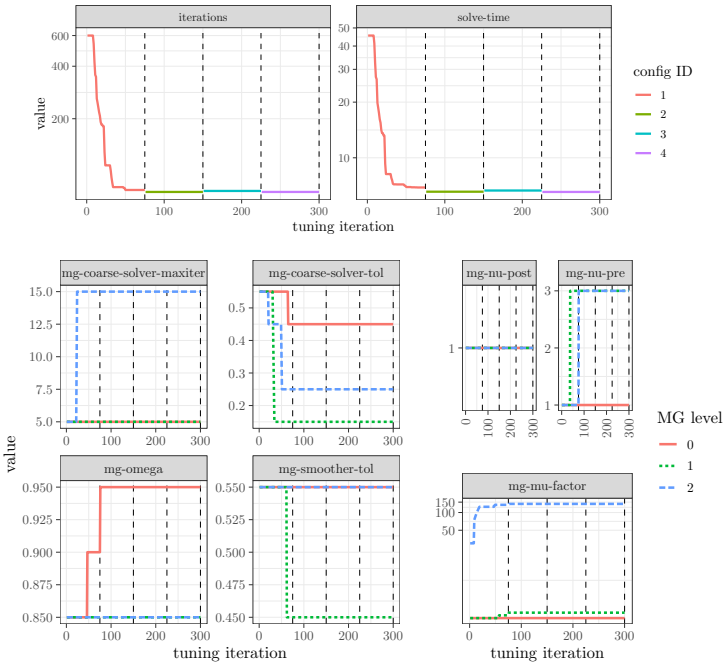

Figure 4: *Tuning history using the "tuning from below" strategy for a* $112^3 \cdot 224$ *ensemble at* M_π^{phys} *on 28 LUMI-G nodes starting from the parameters in table 4. The top panels show the number of outer solver iterations and the time-to-solution while the lower panels show the different parameter values with colours and line types indicating the MG level. The dashed vertical lines indicate the switching of gauge configurations.*

library could be useful for the wider community but the parameter input and output formats would need to be well-designed and the performance of the autotuner for other fermion discretisations would need to be studied.

Acknowledgments

We would like to thank the QUDA developers for their tremendous work as well as the many pleasant and productive interactions during this and previous efforts. We thank the ETMC for the most enjoyable collaboration. This project was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the CRC 1639 NuMeriQS – project no. 511713970. This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) and the NSFC through the funds provided to the Sino-German Collaborative Research Center CRC 110 "Symmetries and the Emergence of Structure in QCD" (DFG Project-ID 196253076 - TRR 110, NSFC Grant No. 12070131001). The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer JUWELS [16] at Juelich Supercomputing Centre. The authors gratefully acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources (Project ID PHY21001). The authors gratefully acknowledge the Swiss National Supercomputing Centre (CSCS) and the EuroHPC Joint Undertaking for awarding this project access to the LUMI supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CSC (Finland) and the LUMI consortium through the Chronos programme under project IDs CH17-CSCS-CYP and CH21-CSCS-UNIBE as well as the EuroHPC Regular Access Mode under project ID EHPC-REG-2021R0095. The authors gratefully acknowledge access to the Bender and Marvin clusters of the University of Bonn as well as support by the HRZ-HPC team.

References

[1] S. Duane, A.D. Kennedy, B.J. Pendleton and D. Roweth, *Hybrid Monte Carlo*, *Phys. Lett. B* **195** (1987) 216.

[2] K. Jansen and C. Urbach, *tmLQCD: A Program suite to simulate Wilson Twisted mass Lattice QCD*, *Comput. Phys. Commun.* **180** (2009) 2717 [0905.3331].

[3] A. Abdel-Rehim, F. Burger, A. Deuzeman, K. Jansen, B. Kostrzewa, L. Scorzato et al., *Recent developments in the tmLQCD software suite*, *PoS* **LATTICE2013** (2014) 414 [1311.5495].

[4] A. Deuzeman, K. Jansen, B. Kostrzewa and C. Urbach, *Experiences with OpenMP in tmLQCD*, *PoS* **LATTICE2013** (2014) 416 [1311.4521].

[5] C. Urbach, B. Kostrzewa, S. Bacchio, R. Baron, B. Blossier, F. Burger et al., *tmLQCD github repository*, https://github.com/etmc/tmLQCD .

[6] I. Omelyan, I. Mryglod and R. Folk, *Symplectic analytically integrable decomposition algorithms: classification, derivation, and application to molecular dynamics, quantum and celestial mechanics simulations*, *Comput. Phys. Commun.* **151** (2003) 272.

[7] M.A. Clark, B. Joo, A.D. Kennedy and P.J. Silva, *Improving dynamical lattice QCD simulations through integrator tuning using Poisson brackets and a force-gradient integrator*, *Phys. Rev. D* **84** (2011) 071502 [1108.1828].

[8] M. Hasenbusch, *Speeding up the hybrid Monte Carlo algorithm for dynamical fermions*, *Phys. Lett. B* **519** (2001) 177 [hep-lat/0107019].

[9] M.A. Clark and A.D. Kennedy, *The RHMC algorithm for two flavors of dynamical staggered fermions*, *Nucl. Phys. B Proc. Suppl.* **129** (2004) 850 [hep-lat/0309084].

[10] C. Urbach, K. Jansen, A. Shindler and U. Wenger, *HMC algorithm with multiple time scale integration and mass preconditioning*, *Comput. Phys. Commun.* **174** (2006) 87 [hep-lat/0506011].

[11] S. Bacchio, C. Alexandrou and J. Finkerath, *Multigrid accelerated simulations for Twisted Mass fermions*, *EPJ Web Conf.* **175** (2018) 02002 [1710.06198].

[12] ETM collaboration, *Twisted mass ensemble generation on GPU machines*, *PoS* **LATTICE2022** (2023) 340 [2212.06635].

[13] M.A. Clark, R. Babich, K. Barros, R.C. Brower and C. Rebbi, *Solving Lattice QCD systems of equations using mixed precision solvers on GPUs*, *Comput. Phys. Commun.* **181** (2010) 1517 [0911.3191].

[14] R. Babich, M.A. Clark, B. Joó, G. Shi, R.C. Brower and S. Gottlieb, *Scaling lattice QCD beyond 100 GPUs*, *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (2011) 70 [1109.2935].

[15] M.A. Clark, B. Joó, A. Strelchenko, M. Cheng, A. Gambhir and R.C. Brower, *Accelerating Lattice QCD Multigrid on GPUs Using Fine-Grained Parallelization*, *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2016) 795 [1612.07873].

[16] Jülich Supercomputing Centre, *JUWELS: Modular Tier-0/1 Supercomputer at the Jülich Supercomputing Centre*, *JLSRF* **5** (2019) A135.

[17] D. Alvarez, *JUWELS Cluster and Booster: Exascale Pathfinder with Modular Supercomputing Architecture at Juelich Supercomputing Centre*, *JLSRF* **7** (2021) A183.

[18] D. Stanzione, J. West, R.T. Evans, T. Minyard, O. Ghattas and D.K. Panda, *Frontera: The Evolution of Leadership Computing at the National Science Foundation*, *Practice and Experience in Advanced Research Computing 2020: Catch the Wave* (2020) 106.

[19] C. Alexandrou, S. Bacchio, J. Finkenrath, A. Frommer, K. Kahl and M. Rottmann, *Adaptive Aggregation-based Domain Decomposition Multigrid for Twisted Mass Fermions*, *Phys. Rev. D* **94** (2016) 114509 [1610.02370].

[20] B. Joó, D.D. Kalamkar, T. Kurth, K. Vaidyanathan and A. Walden, *Optimizing Wilson-Dirac Operator and Linear Solvers for Intel KNL*, *High Performance Computing* (2016) 415.

[21] M. Schröck, S. Simula and A. Strelchenko, *Accelerating Twisted Mass LQCD with QPhiX*, *PoS* **LATTICE2015** (2016) 030 [1510.08879].

[22] B. Joó, D. Kalamakar, K. Vaidyanathan, M. Smelyanskiy, T. Kurth, A. Walden et al., *QPhiX github repository*, https://github.com/JeffersonLab/qphix .