Roman Gruber

Variance reduction in lattice determinations of the leading order hadronic vacuum polarisation contribution to the muon g-2

Diss. ETH No. TODO

ROMAN GRUBER

VARIANCE REDUCTION IN LATTICE DETERMINATIONS OF THE LEADING ORDER HADRONIC VACUUM POLARISATION CONTRIBUTION TO THE MUON G-2

DISS. ETH NO. TODO: TODO

VARIANCE REDUCTION IN LATTICE DETERMINATIONS OF THE LEADING ORDER HADRONIC VACUUM POLARISATION CONTRIBUTION TO THE MUON G-2

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

ROMAN GRUBER
Dipl., Eidgenössisches Polytechnikum

born on 13 February 1987
citizen of Switzerland

accepted on the recommendation of

Prof. Dr. Marina Krstić Marinković, supervisor
Prof. Dr. Thomas C. Schulthess, second topical supervisor
Dr. Isabel Campos Plasencia, external examiner

2025

Roman Gruber: *Variance reduction in lattice determinations of the leading order hadronic vacuum polarisation contribution to the muon g-2*, © 2025

DOI: 10.3929/ethz-a-**TODO**: todo

To Cheryl

ABSTRACT

The openQxD codebase was interfaced to QUDA, enabling utilization of modern GPU-based compute clusters in the observable evaluation phase of computational lattice field theory, where solves of the Dirac equation can be offloaded to QUDA. Features like spatial $C^\star$ boundary conditions and the QCD+QED Wilson-Clover Dirac operator were implemented directly in QUDA. Special emphasis was taken on versatility and convenience on the application level for interaction with the interface allowing hybrid and heterogeneous workloads. Further focus was drawn on efficient performance on a variety of modern compute infrastructure culminating in a range of scaling studies. Code robustness can be guaranteed by a CI/CD pipeline running directly on production machines at CSCS.

A novel variance reduction method called multigrid low-mode averaging was developed and investigated. The method is inspired by solver preconditioning methods such as multigrid and inexact deflation, where efficient low-mode subspaces are generated by exploiting local coherence of low modes. Since traditional methods show disadvantageous quadratic scaling in the lattice volume making them prohibitively expensive as volumes increase, the numerical study focuses on the volume scaling behavior of the proposed method. The main result of this thesis shows linear volume scaling of the new method, while achieving constant variance reduction with a constant number of low modes on all investigated lattices. This renders the method a promising candidate for observables suffering from the signal-to-noise ratio problem on large physical point lattices near the chiral limit.

ZUSAMMENFASSUNG

Das Simulationsprogramm openQxD wurde mit einem Interface zu QUDA ausgestattet, womit moderne GPU-basierte Rechencluster für die Observablenberechnung in der computergestützten Gitterfeldtheorie genutzt werden können, insbesondere durch das Auslagern der Lösung der Diracgleichung an QUDA. Funkionalitäten sowie raumliche $C^{\star}$ Randbedingungen sowie der QCD+QED Wilson-Clover Dirac Operator wurden direkt in QUDA implementiert. Besonderer Wert wurde auf Vielseitigkeit und Anwenderfreundlichekeit gelegt, um eine flexible Interaktion mit der Schnittstelle und hybride oder heterogene Arbeitlasten zu gewährleisten. Ein weiterer Schwerpunkt wurde auf die effiziente Nutzung von moderner Recheninfrastrukturen gelegt, was in einer Reihe von Skalierungsstudien resultierte. Die Robustheit des Codes wird durch einen CI/CD Pipeline gewährleistet, die direkt auf Produktionsmaschinen am CSCS ausgeführt wird.

Eine neue Varianzreduktionsmethode names Multigrid Low-Mode Averaging wurde entwickelt und untersucht. Die Methode ist inspiriert durch Vorkonditionierungstechniken wie Multigrid oder inexact Deflation, bei denen effizient Unterräume assoziiert zu niedrigen Moden mittels Ausnutzung lokaler Kohärenz konstruiert werden. Da herkömmliche Verfahren eine ungünstige quadratische Skalierung im Gittervolumen aufweisen und bei grossen Volumina schnell undurchführbar werden, konzentriert sich die numerische Untersuchung auf das Volumen-Skalierungsverhalten der neuen Methode. Das Hauptergebnis dieser Arbeit zeigt, dass die neue Methode eine lineare Volumenskalierung erreicht und gleichzeitig die Varianzreduktion bei einer konstanten Anzahl niedriger Moden auf allen untersuchten Gittern konstant halten kann. Dies macht sie zu einem vielversprechendem Kandidaten für Observablen, die unter dem Signal-zu-Rausch-Verhältnis leiden, speziell auf grossen physikalischen Gittern nahe des chiralen Limits.

DECLARATION OF ORIGINALITY

I, Roman Gruber, declare hereby that the contents of this thesis are based on my research at ETH Zürich. The work and results presented in this document have been produced by me in conjunction with my supervisor Prof. Marina Krstić Marinković, other members of the Computational Lattice Field Theory research group in Zürich, members associated to the PASC project [12] and members of the RC* collaboration. I confirm that I have not submitted any part of this work for another degree or qualification anywhere else.

The contents of part i are partly relying on ref. [P4] which is currently unpublished, but journal publication is in preparation. Prelimiary results were published in the conference proceeding ref. [P11].

The contents of part ii are based on a conference proceeding ref. [P6], whereas most of the results rely on ref. [P1]. Some of the plots are taken from ref. [P1], if so it is indicated specifically in the respective captions.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, Prof. Marina Marinković, for her continuous guidance, patience and for introducing me to this vast field of research with its many possibilities. I appreciate the freedom and trust (which I think I have not deserved) she gave me to explore my own ideas.

I thank Prof. Thomas Schulthess for introducing me to Marina during my master thesis and for serving as my second supervisor.

The people from the research group in Zürich, I would like to thank for their inputs and constructive feedback, specially Tim Harris for countless discussions about noise and for showing me repeatedly that variances of diagrams are diagrams themselves[1].

I thank the members of the RC* collaboration for accepting me as one of their own, giving me the opportunity to integrate my research into the broader perspective of the collaboration's scientific goals, for enduring my way-too-long presentations on various topics and for tolerating me being annoying about software development aspects.

My girlfriend Cheryl supported me all the way through, I am grateful for that. She made it through an early (even less readable) version of this document[2].

This work would not have been possible without my cats; my emotional support animals. I thank them for distracting me, chasing my mouse cursor, zoom-bombing me or sitting directly on my keyboarddddddddddddddd[3]. They were my constant companions throughout countless coffee-fueled days and nights.

I am grateful to Richard Brower, Kate Clark, Carlton De Tar, Aida El-Khadra, Leonardo Giusti, Steven Gottlieb, Bartosz Kostrzewa, Simon Kuberski, Christoph Lehner, Martin Lüscher, Mathias Wagner and Evan Weinberg for insightful discussions at conferences, workshops or visits.

The support for the PASC 2021-2024 project "Efficient QCD+QED Simulations with openQ*D software" [12] is gratefully acknowledged.

---

[1] Don't tell him, but I still don't entirely get it.

[2] Voluntarily!

[3] Undoubtedly, this led to some of my best results, but I attribute all errors in this document to them alone.

I acknowledge the access to Piz Daint and Daint Alps at the Swiss National Supercomputing Centre, Switzerland under the ETHZ's share with the project IDs c21, ch16, eth8, g021 and s1196.

I 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 [13] at Jülich Supercomputing Centre (JSC).

I acknowledge the EuroHPC Joint Undertaking for awarding this project access to the EuroHPC supercomputer LUMI, hosted by CSC (Finland) and the LUMI consortium through a EuroHPC Regular Access call.

I acknowledge the EuroHPC Joint Undertaking for awarding this project access to the EuroHPC supercomputer LEONARDO, hosted by CINECA (Italy) and the LEONARDO consortium through an EuroHPC Regular Access call.

Finally, I want to explicitly clarify how and to what extent I have used generative AI technologies in the writing of this thesis. Mainly it has been used for three tasks: (1) Asking clarification questions about technical details (e.g. the role of the SW-term in the Wilson Dirac operator); (2) Suggestions for reformulations of draft paragraphs. I never copied such rewritings but eventually used them as basis for refactoring pieces of text. Although there might be very few sentences in this document which are direct copies; (3) Help ensure completeness when reviewing known technologies or algorithms (e.g. a list of variance reduction methods). Responses were used as a basis.

TABLE OF CONTENTS

1 Introduction 1
  1.1 Fundamentals of lattice field theory 2
  1.2 Muon $g-2$ 13
  1.3 High-precision physics in particle phenomenology 22
  1.4 Motivations of this thesis 26
  1.5 Thesis objectives and scope 28
  1.6 Summary of contributions 30
  1.7 Structure of the thesis 32

I GPU Porting of Dirac Solvers

2 Introduction to Part I 35
3 openQxD 41
  3.1 Design principles 41
  3.2 Space-time indexing 43
  3.3 The spinor field 43
  3.4 The gauge field 45
  3.5 The clover field 46
  3.6 $C^\star$ boundary conditions 47
  3.7 Implementation of QCD+QED 47
  3.8 Summary 49
4 QUDA 51
  4.1 Kernel invocation 51
  4.2 Field representations 52
  4.3 Floating-point precision 54
  4.4 Compression formats 54
  4.5 Important interface functions 55
  4.6 Summary 56
5 Interfacing openQxD and QUDA 57
  5.1 Changes and additions to QUDA 57
  5.2 Changes and additions to openQxD 79
  5.3 Summary 86
6 A Developer's Guide for Working with the Interface 87
  6.1 General procedure 87
  6.2 Accessing the solver 88
  6.3 Summary 94

7 Performance Assessment of Interface Components 97
  7.1 Hardware specifications 97
  7.2 Tested ensembles 98
  7.3 Methodology 98
  7.4 The dual lattice 99
  7.5 The asynchronous solver 103
  7.6 The multiple right-hand sides solver 105
  7.7 Dirac operator 108
  7.8 Solver 110
  7.9 Summary 112
8 Automating Build and Deployment with CI/CD 115
  8.1 Continuous integration 115
  8.2 Continuous testing 115
  8.3 Continuous deployment 122
  8.4 Summary 123
9 Proposal for Field Memory Management in Heterogeneous Architectures 125
  9.1 Motivation 125
  9.2 Unified fields 127
  9.3 Implicit transfers 129
  9.4 Summary 133
10 Concluding remarks of part I 137

II Variance Reduction for Two-Point Functions

11 Introduction to Part II 141
12 Two-point correlator 149
  12.1 Lattice definition 149
  12.2 Two-point correlator 150
  12.3 Summary 154
13 Low-mode averaging 155
  13.1 Spectral decomposition 155
  13.2 The rest-rest term 157
  13.3 The eigen-eigen term 157
  13.4 The cross-term 158
  13.5 Summary 159
14 Subspace deflation 161
  14.1 Subspace definition 161
  14.2 Coarse operators 162
  14.3 Low-mode averaging 163

14.4 Summary 164
15 Local coherence and block decomposition 165
15.1 Definition and coarse lattice 165
15.2 Quality of approximation 168
15.3 Spin degrees of freedom 170
15.4 Summary 171
16 Multigrid 173
16.1 Multigrid preconditioning 173
16.2 Multigrid propagator 179
16.3 Multigrid low-mode averaging 183
16.4 Two-point connected correlator 184
16.5 Summary 190
17 Numerical results with $\mathcal{O}(a)$-improved Wilson fermions 191
17.1 Methodology 192
17.2 Total variance 193
17.3 Gauge variance 193
17.4 Relative variance 194
17.5 Absolute variance 196
17.6 Reaching gauge variance 198
17.7 Volume scaling 200
17.8 Cost 202
17.9 Summary 211
18 Spectral implications of chirality 213
18.1 Coarse chirality 215
18.2 Weak chirality preservation 218
18.3 Numerical range 221
18.4 Spectral hull 226
18.5 Numerical eigenvalue study 229
18.6 Numerical variance study 231
18.7 Methodology 231
18.8 Summary 233
19 Concluding remarks of part II 235

20 Summary of the thesis 239
20.1 Limitations 241
20.2 Outlook 242

A Appendix: Building the interface 245
A.1 Building QUDA using CMake 245

- A.2 Building QUDA using Spack 246
- A.3 Building openQxD against QUDA 247
- A.4 uenv 248

B Appendix: Running on the GPU 251
- B.1 The input file 251
- B.2 Parametrizing a simple solver 251
- B.3 Parametrizing a multigrid solver 252

C Appendix: Numerical range and spectral hull estimators 255
- C.1 Numerical range 255
- C.2 Spectral hull 257

D Appendix: Symmetry metrices 259

E Appendix: Notation 261
- E.1 Frequently used names 261
- E.2 Physical constants 261
- E.3 Frequently used symbols and conventions 261
- E.4 Index notation 262
- E.5 γ-matrix basis 262

F Appendix: License and Availability 263
- F.1 Code availablility 263
- F.2 Data availablility 263

Bibliography 265
List of Figures 289
List of Tables 299
Publications 303

1

INTRODUCTION

> *Abandon all hope, ye who enter here.*
> — Dante Alighieri, Inferno

This section/chapter was proofread 3 times.

The Standard Model of particle physics is one of the most successful frameworks for theoretical predictions of physical quantities. Its agreement with experimental data is unprecedented. Lattice field theory is currently the only regularization of quantum field theory that allows for the numerical computation of non-perturbative observables from first principles. Moreover, determinations from lattice field theory can be systematically improved. Gold-plated observables like the hadronic contributions to the anomalous magnetic momentum of the muon $g-2$, where low energy behavior of QCD plays an important role are accessible from basic principles only non-perturbatively with the application of lattice QCD. When adopting lattice regularization, the fields of the continuum theory are discretized in spacetime and represented as degrees of freedom associated to sites and links in a four dimensional spacetime hypercube. The discretization of the theory naturally regularizes ultraviolet and infrared divergences occurring in the continuum theory, due to its finite resolution and volume description. Feynman's path integral description of QFT is used as a representation, where the functional integral over field degrees of freedom turns into a high-dimensional regular integral. A typical lattice QCD ensemble, with a volume on the order of a few fermi and lattice spacings well below 1 fermi[1], contains at least $\mathcal{O}(10^8)$ degrees of freedom, including flavors, colors, and space-time indices. This introduces the need for modern high-performance computing (HPC) infrastructure for their evaluation[2]. Careful understanding of systematic and statistical errors allows studying physical quantities of interest and thorough comparison to experiment. Discretizing a theory comes with many decisions and choices; each with its advantages and disadvantages. Independent research groups in the field, working with different discretizations, regularly compare their results and discuss systematics. Once extrapolated to the continuum, lattice results should show independence of these choices and agree within their error estimates – this is often called universality.

---

[1] To simulate nucleons for instance, one requires lattice extents of a few fermi because the nucleon radius is about 1 fm, whereas the lattice spacing must be significantly smaller than that for a proper resolution.

[2] This is not the only reason. Such large lattices would fit into memory of a large node nowadays, e.g. HPE Superdome Flex [14] systems come with a maximal memory capacity of 48 TB. The bottleneck is usually the memory bandwidth, not capacity, and by this time to solution. Bandwidth can be multiplied by distribution over several nodes, reducing time to solution. Furthermore, slow decorrelation along the Molecular Dynamics trajectory leads to large autocorrelation times and a substantial increase in time to solution for sampling independent

1.1 FUNDAMENTALS OF LATTICE FIELD THEORY

Lattice field theory is a discretized version of continuum quantum field theory. Expectation values of observables in the context of continuum quantum field theory can be mathematically represented as functional integrals over the fields [15]

$$\langle \mathcal{O}[\psi,\bar{\psi},A]\rangle = \frac{1}{Z}\int \mathcal{D}\psi\,\mathcal{D}\bar{\psi}\,\mathcal{D}A\;\mathcal{O}[\psi,\bar{\psi},A]e^{iS[\psi,\bar{\psi},A]}\;. \tag{1.1}$$

In this equation, the fundamental degrees of freedom are the fields ψ, $\bar{\psi}$ and A. S denotes the action of the theory and $Z = \int \mathcal{D}\psi\,\mathcal{D}\bar{\psi}\,\mathcal{D}A e^{iS[\psi,\bar{\psi},A]}$ the partition function. The observable $\mathcal{O}$ is a functional of the fields and can be any time-ordered correlation function. Here, ψ and $\bar{\psi}$ may represent fermionic particles and A a gauge boson. The bosonic field is vectorial, the fermionic one spinorial and Grassmann-valued. It is a standard result in fermionic QFT that the fermion degrees of freedom in the partition function Z can be integrated out, such that

$$Z = \int \mathcal{D}A\,\det(D)e^{iS[A]}\;. \tag{1.2}$$

The non-locality of the determinant encodes all possible paths of the fermion represented by ψ. For the first time, we encounter the operator that has a central role in this document; the Dirac operator D, here in its continuum form $D = i\gamma^\mu D_\mu - m$ with the Dirac matrices obeying the Clifford algebra $\{\gamma^\mu,\gamma^\nu\} = 2\eta^{\mu\nu}$, where η is the Minkowski spacetime metric[3], m the fermion mass and D_μ are covariant derivatives

$$D_\mu = \partial_\mu + igA^a_\mu T^a\;, \tag{1.3}$$

---

gauge fields – a process inherently serial due to its Markov chain nature. This phenomenon is known as critical slowing down.

[3] In particle physics notation usually of signature $(+---)$.

where the T^a are the generators of the gauge group's Lie-algebra and g the coupling. These derivatives were introduced to retain Lorentz-invariance of the kinetic fermion part of the action S. Inevitably, this introduced another field A which is vector-valued, massless and mediates the interaction among fermions. In the context of quantum electrodynamics (QED) with gauge group $U(1)$, this results in the photon mediating the electromagnetic interaction, whereas for an $SU(3)$ gauge group this results in the 8 gluons mediating the strong interaction of quantum chromodynamics (QCD).

The fermionic quark fields belong to the N_c-dimensional fundamental representation of the color gauge group $SU(N_c)$, antiquark fields to the complex conjugate representation of $SU(N_c)$ which is N_c-dimensional too, and gluon fields to the adjoin representation which is $(N_c^2 - 1)$-dimensional.

In order to identify eq. (1.1) with a classical statistical model, we have to perform a Wick-rotation translating Minkowski into Euclidean spacetime, $t_m \to -it_e$, where t_m is the *real* Minkowski time and t_e is the (real valued) Euclidean time. The effect of this on the action is that e^{iS_m} turns into e^{-S_e} allowing the interpretation of the Euclidean action S_e which is real (for zero chemical potential) as a Boltzmann weight in a statistical sense as opposed to the oscillatory behavior of the Minkowski action exponential e^{iS_m}. The path integral becomes well-behaved, enabling numerical simulations via lattice QCD.

Under certain conditions, Euclidean correlation functions can be analytically continued to Wightman functions (Minkowski correlation functions) and vice versa via the Osterwalder-Schrader reconstruction theorem [16, 17], preserving the physical contents of the theory. Although this reconstruction is limited to time-reversible observables, its numerical inversion becomes non-trivial or ill-posed for real-time processes such as decay amplitudes, spectral functions or processes that involve a branch cut.

Due to the Wick-rotation, correlation functions on the lattice $C(t_e)$ have to be interpreted in Euclidean time. **TODO**: Tims comment Euclidean time evolution is thus exponentially decaying as opposed to oscillatory in Minkowski space,

$$C(t_e) = \sum_n A_n e^{-E_n t_e} \,, \qquad \text{(Euclidean)} \qquad (1.4)$$

$$C(t_m) = \sum_n A_n e^{-iE_n t_m} \,. \qquad \text{(Minkowski)} \qquad (1.5)$$

The Euclidean theory can be discretized by introducing a finite four dimensional spacetime lattice Λ of Cartesian coordinates. Lattice sites are usually evenly spaced by the dimensionful lattice spacing a, so that coordinates are given in units of a and are therefore dimensionless. We write fields defined on the lattice in resemblance of their continuum analogues as $\psi(x)$ with $x = an$, $n \in \Lambda$ a dimensionless, non-negative integer, subject to boundary conditions. The lattice spacing a provides the finite resolution of the discretization curing UV divergences with a momentum cutoff proportional to $1/a$ and the finite lattice volume $V = |\Lambda|$ cures IR divergences with a cutoff proportional to a. This provides a natural regularization of continuum theory.

Fermion fields are defined on the lattice sites with possible further *internal* degrees of freedom for spin and color, whereas gauge fields acting as parallel transports linking neighboring lattice sites. Depending on the gauge group, gauge links possess further internal degrees of freedom with values drawn from the gauge group.

The dynamics of the theory are governed by the continuum QCD action [18]

$$S = \int \mathrm{d}x \ \mathcal{L}(x) \ , \quad \mathcal{L}(x) = -\frac{1}{4} F^a_{\mu\nu}(x) F^{\mu\nu}_a(x) + \bar{\psi}(x)(D\psi)(x) \ , \tag{1.6}$$

with the continuum Dirac operator D and the Yang-Mills field strength tensor

$$F^a_{\mu\nu}(x) = \partial_\mu A^a_\nu(x) - \partial_\nu A^a_\mu(x) + g f^{abc} A^b_\mu(x) A^c_\nu(x) \ . \tag{1.7}$$

The coupling constant g is a free parameter of the theory and the structure constants f^{abc} parametrize the Lie-algebra of the gauge group.

1.1.1 *Dirac operator*

Discretizing the QCD action eq. (1.6) and replacing derivatives with symmetric differences results in the Wilson action [19],

$$S = S_G + S_F \ , \tag{1.8}$$

$$S_G = \frac{1}{g^2} \sum_{x \in a\Lambda} \sum_{\mu,\nu} \mathrm{Re}\,\mathrm{tr}\left[\delta_{\mu\nu}\mathbb{1} - U_{\mu\nu}(x)\right] \ , \tag{1.9}$$

$$S_F = a^4 \sum_{x \in a\Lambda} \bar{\psi}(x)(D_{\text{naive}}[U]\psi)(x) \ . \tag{1.10}$$

The gauge part of the action S_G turns into a sum over plaquettes (see fig. 1.1a for an illustration)

$$U_{\mu\nu}(x) = U_\mu(x) U_\nu(x + \hat{\mu}) U_\mu(x + \hat{\nu})^\dagger U_\nu(x)^\dagger \ , \tag{1.11}$$

where U_μ is the Lie-group-valued gauge field

$$U_\mu(x) = P\exp\left(ig\int_x^{x+a\hat{\mu}} \mathrm{d}z_\mu A_\mu(z)\right) \approx e^{igaA_\mu(x+\frac{a}{2}\hat{\mu})} , \tag{1.12}$$

as exponential of the Lie-algebra-valued field A and $\hat{\mu}$ is the unit 4-vector in direction μ. As constructed in eq. (1.12), gauge links are explicitly defined in between lattice sites.

The fermion part S_F introduces the discretized Dirac operator – the naive lattice Dirac operator

$$D_{\mathrm{naive}}[U] = m + \frac{1}{2}\sum_\mu \gamma_\mu \left(D_{+\mu} + D_{-\mu}\right) , \tag{1.13}$$

with the discretized covariant forward and backward derivatives

$$D_{\pm\mu}(x,y) = \pm\frac{1}{a}\left[U_{\pm\mu}(x)\delta_{x\pm\hat{\mu},y} - \delta_{x,y}\right] , \tag{1.14}$$

and the γ_μ are the Dirac matrices obeying the Euclidean Clifford algebra $\{\gamma_\mu, \gamma_\nu\} = 2\delta_{\mu\nu}$.

Since the gauge field lives on the links between lattice points, the parallel transporter in positive direction μ, linking point x with its neighbor $x+\hat{\mu}$, is related to the link in negative μ direction linking point $x+\hat{\mu}$ and x by Hermitian transpose. For $\mu = 1,2,3,4$ a direction, this results in

$$U_{\pm\mu}(x) = U_{\mp\mu}(x\pm\hat{\mu})^\dagger , \tag{1.15}$$

where ± denotes the link in positive or negative μ-direction, respectively.

When expanding in a, we find that the naive gauge action eq. (1.9) has $\mathcal{O}(a^2)$ discretization effects

$$\frac{1}{g^2}\,\mathrm{Re}\,\mathrm{tr}\left[\delta_{\mu\nu}\mathbb{1} - U_{\mu\nu}(x)\right] = \frac{1}{2}\,\mathrm{tr}\left[F^a_{\mu\nu}(x)F^b_{\mu\nu}(x)T^aT^b\right] + \delta_{\mu\nu}\mathcal{O}(a^2) . \tag{1.16}$$

The naive Dirac operator in eq. (1.13) introduces fermion doublers; unphysical poles in the fermion propagator $(D_{\mathrm{naive}})^{-1}$. These correspond to particles that do not exist in continuum theory. The Dirac operator can be extended by any term proportional to the lattice spacing a or positive powers thereof; such a term will vanish when taking the continuum limit $a \to 0$. Wilson proposed a solution to the fermion doubling problem [20] resulting in the Wilson-Dirac operator

$$D_W[U] = D_{\mathrm{naive}}[U] - \frac{a}{2}\sum_\mu D_{-\mu}D_{+\mu} . \tag{1.17}$$

The additional Wilson-term proportional to a removes the fermion doubler states by adding a term proportional to $1/a$ to their masses. Such states decouple in the continuum limit, but the term above breaks chiral symmetry and introduces $\mathcal{O}(a)$ discretization errors. Even more troubling, every discretization in four spacetime dimensions with the correct continuum limit and respecting locality, either comes with fermion doublers or breaks chiral symmetry. This is the famous Nielsen-Ninomiya no-go theorem [21, 22].

Another term proportional to a was introduced later [23] to cancel the $\mathcal{O}(a)$ discretization effects introduced by the Wilson-term systematically, while still eliminating the doublers,

$$D_{WC}[U] = D_W[U] + c_{\text{sw}}^{SU(3)} a \frac{i}{4} \sum_{\mu,\nu} \sigma_{\mu\nu} \hat{F}_{\mu\nu} \; . \tag{1.18}$$

This is called the Wilson-Clover (also SW) Dirac operator. Here, $\sigma_{\mu\nu} = \frac{i}{2}\left[\gamma_\mu, \gamma_\nu\right]$ and the discretized $SU(3)$ field strength tensor $\hat{F}_{\mu\nu}$ is defined as

$$\hat{F}_{\mu\nu}(x) = \frac{1}{8}\left\{Q_{\mu\nu}(x) - Q_{\nu\mu}(x)\right\} \; , \tag{1.19}$$

$$Q_{\mu\nu}(x) = U_\mu(x)U_\nu(x+\hat{\mu})U_\mu(x+\hat{\nu})^{-1}U_\nu(x)^{-1} \tag{1.20}$$

$$+ U_\nu(x)U_\mu(x-\hat{\mu}+\hat{\nu})^{-1}U_\nu(x-\hat{\mu})^{-1}U_\mu(x-\hat{\mu}) \tag{1.21}$$

$$+ U_\mu(x-\hat{\mu})^{-1}U_\nu(x-\hat{\mu}-\hat{\nu})^{-1}U_\mu(x-\hat{\mu}-\hat{\nu})U_\nu(x-\hat{\nu}) \tag{1.22}$$

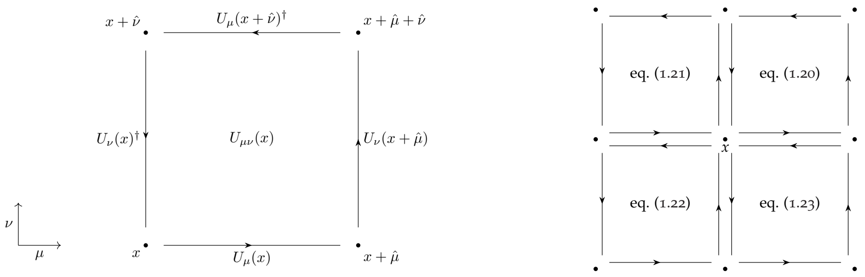
$$+ U_\nu(x-\hat{\nu})^{-1}U_\mu(x-\hat{\nu})U_\nu(x+\hat{\mu}-\hat{\nu})U_\mu(x)^{-1} \; . \tag{1.23}$$

The term proportional to $c_{\text{sw}}^{SU(3)}$ is called the clover- or SW-term and was proposed by Sheikholeslami and Wohlert [23] as addition to the action, see fig. 1.1b for an illustration. It is block-diagonal. Following an effective field theory analysis, it gives $\mathcal{O}(a)$ improvement à la Symanzik [24, 25] by tuning the algorithmic parameter $c_{\text{sw}}^{SU(3)}$ non-perturbatively such that $\mathcal{O}(a)$ discretization effects vanish [26].

There are many alternative discretizations to the above. All deal differently with chiral symmetry on the lattice and the doubling problem. Staggered fermions [27] reduce the 16 doublers to 4 *tastes* and retain a remnant U(1) chiral symmetry. Domain-wall fermions [28, 29] suppress doublers through the introduced fifth dimension of extent L_s and exact chiral symmetry is obtained in the limit[4] $L_s \to \infty$. Twisted-mass fermions [30]

---

[4] Typically L_s is chosen to be around 12 to 48.



(a) The plaquette $U_{\mu\nu}(x)$

(b) The clover term $Q_{\mu\nu}(x)$

FIGURE 1.1: Illustration of the plaquette term (left) which is a matrix-matrix product of gauge links along the lines as in eq. (1.11) and the clover term (right) which is a sum of rotated plaquettes that resemble the form of a clover leaf. The individual plaquettes correspond to the summands in eqs. (1.20) to (1.23).

remove doublers with the Wilson term and chiral symmetry breaking effects are improved by the mass twisting. Overlap fermions [31, 32] entirely remove doublers and retain exact chiral symmetry. This is in agreement with Nielsen-Ninomiya [21, 22], because their Dirac operator is non-local, but exponentially local [33, 34], making it computationally very expensive.

Summarizing, the QCD Wilson-Clover Dirac operator – which for simplicity we will call D in the remainder of this document – is a nearest-neighbor stencil operator acting on a linear space of quark (or spinor) fields defined on a regular 4D cubic lattice of V sites, typically in the range $10^6 - 10^8$. Applied to a spinor field $\psi(x)$ the Dirac operator can then be written as a stencil (the lattice spacing is set to $a = 1$)

$$
\begin{aligned}
(D\psi)(x) &= (4+m)\psi(x) \\
&-\frac{1}{2}\sum_{\mu=0}^{3}\left\{U_\mu(x)(1-\gamma_\mu)\psi(x+\hat{\mu}) + U_\mu(x-\hat{\mu})^{-1}(1+\gamma_\mu)\psi(x-\hat{\mu})\right\} \\
&+c_{\text{sw}}^{SU(3)}\frac{i}{4}\sum_{\mu,\nu=0}^{3}\sigma_{\mu\nu}F_{\mu\nu}(x)\psi(x),
\end{aligned}
\tag{1.24}
$$

where the gauge field $U_\mu(x)$ is the $SU(3)$-valued link between lattice points x and $x+\hat{\mu}$. The term $4+m = \frac{1}{2\kappa}$ where κ is the inverse mass.

Notable about this choice of discretization are some properties which we briefly review here, because they will become important later. The operator has the property of pseudo-Hermiticity with respect to $\gamma^5 = \gamma_0\gamma_1\gamma_2\gamma_3$,

$D^\dagger = \gamma^5 D \gamma^5$. Among many implications (as discussed later), its eigenvalues are either real or come in complex conjugate pairs and its determinant is real.

1.1.2 *Dirac equation*

Due to the nearest-neighbor coupling, a good parallelization scheme is achieved by domain decomposition, where a contiguous local volume of size $V_\mathrm{L} = L_0 L_1 L_2 L_3$, which divides the global problem size, is associated with a single compute unit. With increasing problem sizes, the Dirac operator is poorly conditioned and sophisticated preconditioning algorithms are required to obtain acceptable convergence rates of iterative solvers.

TL;DR: Definition of Dirac eq on lattice

The system of linear equations of interest is the Dirac equation

$$D\psi = \eta \,, \tag{1.25}$$

where different choices for the right-hand side η result in different propagator expressions. Solves of this equation appear repeatedly in lattice field theory and will be a central computational, algorithmic and theoretical motif throughout this thesis.

1.1.3 *Lattice expectation values*

Taking our previous considerations into account, we find for Euclidean expectation values

$$\langle \mathcal{O}[\psi, \bar{\psi}, U] \rangle = \frac{1}{Z} \int \mathcal{D}U \, \det(D) e^{-S_G[U]} \mathcal{O}[U] \,. \tag{1.26}$$

The integral measure $\mathcal{D}U$ is the Haar measure [35] of the group manifold. Fermion field dependencies of the observable in the integrand $\mathcal{O}[U]$ are Wick-contracted and replaced by traces over fermion propagators D^{-1} from lattice points x to y. The observable now depends on solutions to the Dirac equation eq. (1.25), rather than the numerically inaccessible Grassmann-valued fermion fields $\psi(x)$ and $\bar{\psi}(x)$[5].

---

[5] This is a pattern occurring repeatedly in the field. As soon as fermions are introduced, expressions become non-local, like the determinant when integrating out the fermion fields or the appearance of D^{-1} when Wick-contracting. This is the price to pay for a proper numerical treatment of Pauli's exclusion principle in Lagrangian field theory dynamics. A similar process happens in Hamiltonian dynamics where the boomerang might hit even harder since the Jordan-Wigner transformation is non-local.

The expectation value in its Euclidean form eq. (1.26) is now amenable for numerical evaluation using Monte Carlo simulations. We can define a probability density function for the gauge fields as

$$P(U) = \frac{1}{Z} \det(D) e^{-S_G[U]} \,, \tag{1.27}$$

importance sample a set of N gauge fields according to it $\{U_i\}_{i=1}^{N}$ and estimate the expectation value of the observable as a mean over the samples

$$\langle \mathcal{O}[\psi, \bar{\psi}, U] \rangle \approx \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}[U_i] \,, \tag{1.28}$$

with variance $\mathcal{O}^2/N$.

1.1.4 *Monte Carlo sampling*

For the remaining integral over the gauge field in eq. (1.26) Markov Chain Monte Carlo methods are usually applied to sample the integral[6]. According to the probability density eq. (1.27), sampling algorithms generate a Markov chain of so called *gauge field configurations*. It is challenging for the Markov process to satisfy ergodicity [36, 37] (all configurations can be reached) and detailed balance (probability flow into a configuration equals the flow out).

With dynamical fermions in lattice field theory, most common is the Hybrid Monte Carlo algorithm [38] (HMC). It introduces conjugate momentum variables π and defines a fictitious Hamiltonian that depends on the gauge field and its conjugate variable

$$\mathcal{H}[U, \pi] = \frac{1}{2}\pi^2 + S_G[U] + \phi^\dagger (D^\dagger D)^{-1} \phi \,. \tag{1.29}$$

The pseudofermion field ϕ was introduced to stochastically represent the determinant $\det\big(D^\dagger D\big)$. In this case we simulate two degenerate quark flavors[7]. This comes from the standard Gaussian integral formula

$$\det\Big(D^\dagger D\Big) \sim \int \mathcal{D}\phi^\dagger \, \mathcal{D}\phi e^{-\phi^\dagger (D^\dagger D)^{-1} \phi} \,. \tag{1.30}$$

---

[6] The field is very active in finding cheaper alternative solutions to this. Machine learning and AI are promising candidates.

[7] For a single flavor usually Rational HMC (RHMC) is used, where $\det(D) \sim \int \mathcal{D}\phi^\dagger \, \mathcal{D}\phi e^{-\phi^\dagger R(D) \phi}$ with $R(D) \approx D^{-1}$ a rational function approximation to D^{-1}.

We emphasize that this is not calculating the determinant, but sampling *according* to the determinant. Furthermore this requires repeatedly solving the Dirac equation, eq. (1.25).

Hamiltonian's equations are used to evolve the fields U, π in the fictitious simulation time τ,

$$\frac{\mathrm{d}U}{\mathrm{d}\tau} = \frac{\partial \mathcal{H}}{\partial \pi}, \quad \frac{\mathrm{d}\pi}{\mathrm{d}\tau} = -\frac{\partial \mathcal{H}}{\partial U}. \tag{1.31}$$

This is done using some integrator like leapfrog. The process of obtaining a new proposal for the fields U, π is called a molecular dynamics (MD) trajectory.

After an MD trajectory (one or multiple steps in τ), we accept or reject the new proposal U', π' according to the accept probability

$$P_{\text{accept}} = \min\left(1, e^{-(\mathcal{H}[U',\pi'] - \mathcal{H}[U,\pi])}\right). \tag{1.32}$$

Since the MD integration is not exact, the accept-reject step corrects for that. This is called the Metropolis-Hastings algorithm [39, 40] and the HMC is a special instance of it, where the new proposal is physics-inspired via Hamiltonian evolution.

We may define the autocorrelation time as the (fictitious) time in MD trajectories until the algorithm yields an independent sample. Monitoring this quantity is crucial for algorithmic tuning of gauge field generation.

The algorithm as described above satisfies detailed balance through the Metropolis accept-reject step. Ergodicity does not come for free. Only if the trajectory length is long enough (and randomized), the momentum variable is properly refreshed, the Dirac operator can be inverted [41] and the random number generator (RNG) produces uncorrelated numbers, then the Markov chain can explore all regions of configuration space and ergodicity is achieved. Over the years, many improvements emerged to optimize and speedup the process of gauge field sampling.

1.1.5 *Phenomenological aspects*

TL;DR: confinement

Due to the non-Abelian nature of the QCD gauge group $SU(3)$, the gauge bosons show self interaction vertices. This feature of QCD is responsible for confinement; quarks and gluons can never be observed in isolation. They must form bound color-singlet states, like mesons or baryons. A color singlet state has no net color charge, it transforms trivially under SU(3) gauge transformations. Confinement originates from the behavior of the strong coupling constant α_s at low energies. It can be seen from the linear behavior of the phenomenological model for the potential of a quark antiquark pair – the Cornell potential [42, 43]

$$V(r) = \text{constant} + \sigma r - \frac{4\alpha_s}{3r} , \tag{1.33}$$

where r is the distance between the quarks and σ is the QCD string tension. The linear term confines the quarks and energy to pull them apart grows indefinitely forming a "string" that holds the two quarks together. Thus, confinement appears at low energies (long distances). If enough energy is supplied, the string breaks and a new quark antiquark pair forms, such that quarks are never isolated. There is no analytic proof of the potential above in 4D spacetime, but lattice QCD determinations of it clearly show the confining behavior [44, 45].

TL;DR: asymptotic freedom

At high energies (short distances) the QCD coupling α_s becomes weak and the linear term in the potential effectively vanishes. The potential becomes Coulomb-like $1/r$. This is the regime where perturbation theory is applicable to QCD and quarks and gluon behave like free particles. The phenomenon of this free particle behavior is called asymptotic freedom [46, 47] and it is also a consequence of the gauge group being non-Abelian.

TL;DR: multiplet quark field, left/right handed spinors

The flavor multiplet consisting of N_f quark flavors

$$\psi(x) = (\psi_0(x), \dots, \psi_{N_f-1}(x))^T \tag{1.34}$$

can be decomposed into left- and right-handed components $\psi = \psi_L + \psi_R$ as

$$\psi_L = P_- \psi , \quad \psi_R = P_+ \psi , \quad P_\pm = \frac{1}{2}(1 \pm \gamma^5) . \tag{1.35}$$

The massless QCD Lagrangian is then invariant under independent flavor rotations of left- and right-handed spinors

$$\psi_L \to U_L \psi_L , \qquad U_L \in SU(N_f)_L , \tag{1.36}$$

$$\psi_R \to U_R \psi_R , \qquad U_R \in SU(N_f)_R . \tag{1.37}$$

The $U_{L,R}$ only act on the flavor indices of ψ mixing flavors, but not handedness. It implies that laws of physics are the same for left-handed and right-handed particles.

TL;DR: spontaneous chiral symmetry breaking

In QCD with multiple light quark flavors in the fundamental representation, spontaneous breaking of chiral symmetry [48] is a non-perturbative feature of the QCD vacuum. The chiral condensate $\langle\bar{\psi}\psi\rangle$ is a measure of it – in fact, it is the order parameter of chiral symmetry breaking. If the vacuum were to respect this symmetry, the chiral condensate would be exactly zero. The fact that this is not the case breaks chiral symmetry from

$$SU(N_f)_L \times SU(N_f)_R \quad \text{down to} \quad SU(N_f)_V \tag{1.38}$$

mixing left- and right-handed quarks. The broken down symmetry group $SU(N_f)_V$ is the subgroup which acts equally on left- and right-handed spinors, sometimes called the vector subgroup,

$$\psi_L \to U_V \psi_L \,, \tag{1.39}$$

$$\psi_R \to U_V \psi_R \,, \qquad\qquad U_V \in SU(N_f)_V \,. \tag{1.40}$$

The vacuum only satisfies the symmetry of the broken down vector subgroup. A non-zero chiral condensate $\langle \bar{\psi}\psi \rangle = \langle \bar{\psi}_L \psi_R + \bar{\psi}_R \psi_L \rangle$ then mixes left- and right-handed fields leading to hadronic mass generation [49] and the emergence of pseudo-Goldstone bosons [50–52] (the pions in QCD).

1.1.6 *Renormalization and the continuum limit*

TODO: Marina: Add a discussion on dimensional transmutation here.

TL;DR: regulators, renormalization, cont limit

To recover continuum quantum field theory from the lattice, we have to take the continuum limit. For the limit $a \to 0$, one needs to compute the observables of interest for several values of the lattice spacing a. The lattice spacing serves as a regulator of the theory; it makes divergent quantities finite by cutting off the divergent parts of integrals at the short (UV) distances, i. e. it cuts off UV divergences. As $a \to 0$ they turn into terms proportional to a^{-n} for some $n \geq 1$. In order to recover continuum QCD, the infinite-volume limit $L \to \infty$ has to be taken too, where L is the extent of the 4D lattice box of volume L^4. The finite volume is another regulator of the theory; it regularizes the long (IR) distance divergences, leading to finite-volume corrections scaling power-like L^{-m} for some $m \geq 1$ or exponentially $e^{-m_\pi L}$. Both regulator limits should be considered together to recover continuum QCD.

Naively taking the continuum limit without tuning the bare parameters to the lines of constant physics will yield divergent results. On the other hand, results should not depend on the regulator and renormalization is the process of removing the dependency of the regulator from physical quantities. Renormalization of the theory's parameters on the lattice is done by the choice of the lines of constant physics in the phase space of bare parameters. Operators like vector currents and bilinears require additional multiplicative renormalization constants. The tuning fixes certain physical quantities, like hadron masses or decay constants to external input. In general, after renormalization, the computed quantities still have lattice artifacts, i. e. terms proportional to powers of a^n for some $n \geq 1$. These are finally removed by the continuum limit $a \to 0$. As the lattice spacing becomes finer, simulation cost increases dramatically.

TL;DR: critical slowing down

For typical lattice QCD simulations with dynamical fermions, the computational simulation cost in the lattice spacing a and the lattice volume V can be modeled as [53]

$$\text{simulation cost} = \mathcal{O}(a^{-z} V^p) \,, \tag{1.41}$$

where z and p are exponents that depend on the action, physics parameters, and algorithms involved. However, the exponent associated to the lattice constant is in the regime $z \geq 5$ and the volume scaling $p = 1 - 2$. These numbers are deeply related to *critical slowing down*.

TL;DR: autocorrelation time dependency

As lattice spacings become finer, the autocorrelation time of the Markov chain diverges, configurations evolve slowly in the phase space and more molecular dynamics trajectories are required to generate statistically independent configurations. It is hard for the algorithm to advance into other topological sectors. In this thesis we will be concerned quite substantially with the volume scaling and critical slowing down will visit us again but seen from a different perspective.

1.2 MUON $g - 2$

In search of new fundamental physics, lattice field theory can predict non-perturbative quantities from first principles to directly probe the Standard Model (SM) of particle physics. **TODO**: This is to search for indirect proofs of its validity. A strong-enough tension with an experimental quantity would proof the Standard Model is not sufficient to describe fundamental interactions. A persistent tension between experimental measurements and SM predictions would indicate that the Standard Model is not sufficient to describe the interactions between particles. In recent years, the anomalous magnetic moment of the muon $g - 2$ has held a prominent role in the search of beyond Standard Model (BSM) physics, due to the longstanding discrepancy of $3 - 4\sigma$ [54] between its measurement and its theoretical prediction.

TL;DR: SM prediction and goals

The Standard Model (SM) prediction of the anomalous magnetic moment of the muon a_μ^{SM} has – as of now – a precision of 532 ppb (parts per billion) [55], roughly the precision of the former E821 experiment at BNL [56]. The SM prediction has not yet reached its precision goal of about 127 ppb [57] which is the final precision of the ongoing E989 experiment at Fermilab [58]. Another experiment measuring the quantity with a different method is in preparation at J-PARC [59], that can serve as a independent cross check. In this context, lattice QCD plays a crucial role, as it provides first-principles, non-perturbative calculations of the hadronic contributions to the $g-2$, which are currently the dominant sources if uncertainty in its theoretical prediction. Although current results seem to relax the tension [55, 58], tensions between data-driven and lattice determinations of the main hadronic contribution to the muon $g-2$ persists [55].

Light may be shed on those questions by decreasing error bars from both theory and experiments resulting in a resolution or tightening of the tension and ultimately a better understanding of nature. Dominant sources of error come from the leading order hadronic vacuum polarization (LO-HVP) term at order $\mathcal{O}(\alpha^2)$ and the subdominant hadronic light-by-light scattering (HLbL) term at order $\mathcal{O}(\alpha^3)$.

1.2.1 *The anomalous magnetic moment*

Due to their charge, leptons create a magnetic field around them leading to a magnetic dipole moment. A classical charged point particle moving on a circular orbit generates a dipole magnetic moment $\vec{\mu}$ that is proportional to its orbital angular momentum $\vec{L}$ by

$$\vec{\mu} = \frac{qe}{2m}\vec{L}. \tag{1.42}$$

where e is the elementary charge, q and m are the integer charge and mass of the particle. In quantum mechanics angular momenta $\vec{L}$ are quantized and for a spin-$\frac{1}{2}$ particle we may replace the angular momentum operator $\vec{L}$ by the spin angular momentum $\vec{S}$ which contributes to the angular momentum and is quantized in steps of size $\hbar/2$. Therefore, we find for the magnetic moment of a spin-$\frac{1}{2}$ particle

$$\vec{\mu} = g\frac{qe}{2m}\vec{S}. \tag{1.43}$$

We have sneaked in a dimensionless proportionality factor g here. Classically we would clearly expect $g = 1$, if spin behaved like angular momentum. However, when interacting with an electromagnetic field, the relativistic treatment leads to an additional term that doubles the expected magnetic moment. This magnetic moment can be characterized in terms of the dimensionless Landé g-factor also known as the gyro-magnetic ratio, parametrized by g in the formula above. In 1928, Dirac formulated the Dirac equation for relativistic spin-$\frac{1}{2}$ particles and derived their magnetic moment [60]. He obtained the g-factor to be $g = 2$ exactly. This result is independent of the mass and holds for every fundamental spin-$\frac{1}{2}$ particle including the electron, muon, tau and even the quarks[8]. The anomalous magnetic moment is defined as the deviation of the g-factor from its tree level result encoding the effects of relativistic quantum field theory,

$$a_\ell = \frac{g_\ell - 2}{2}. \tag{1.44}$$

The one-loop correction to the electron $a_e = \frac{\alpha}{2\pi}$ was already calculated analytically in 1948 by Schwinger [61]. Impressive advancements of higher order corrections up to $\mathcal{O}(\alpha^5)$ (i. e. to tenth order in e!) have been carried out since then for the electron [62–65].

1.2.2 *Which lepton?*

Since all virtual particles contribute to the magnetic moment, it can be argued [66] that corrections δa_ℓ to the anomalous magnetic moment of leptons scale with the mass as

$$\delta a_\ell \sim \frac{m_\ell^2}{\Lambda^2}, \tag{1.45}$$

where m_ℓ is the mass of the lepton and Λ is the energy scale where new physics appear, i. e. roughly the mass of a new unknown heavy virtual particle. This suggests that the anomalous magnetic moment of the heaviest known lepton, the tau with a mass of about 1.8 GeV (see table 1.1), should be the best candidate to lead to new physics. Admittedly the tau is very short lived with a lifetime of about 0.3 ps making it hard to access in experiments. Nevertheless, it is being investigated as an alternative path to new physics [67, 68]. The lepton with the longest lifetime, the electron with a lifetime of many years, is unfortunately very light and thus unlikely to produce heavy virtual particles. Furthermore the electrons $g-2$ is ex-

---

[8] It does not hold for composite spin-$\frac{1}{2}$ particles like the proton or neutron because they are not point-like; an assumption in Dirac's derivation.

Lepton ℓ	Mass	Lifetime [s]
Electron	0.51099895069(16) eV [71]	$2.139\,65 \times 10^{11}$ [72]
Muon	105.6583755(23) MeV [71]	$2.196\,981\,1 \times 10^{-6}$ [73, 74]
Tau	1.77686(12) GeV [71, 75]	2.903×10^{-13} [75]

Table 1.1: Lepton masses and their mean lifetimes.

perimentally measured to about 0.13 ppb [69], whereas the muon's to 127 ppb [58]. Taking this in account, the sensitivity to BSM physics is thus

$$\text{BSM sensitivity} \sim \frac{m_\ell^2}{\sigma_{\text{exp}}}\,, \tag{1.46}$$

where σ_{exp} is the experimental measurement precision. This leaves the muon with a lifetime accessible to nowadays experiments and a squared mass of about forty thousand times the one of the electron [70]. However its overall sensitivity is only about forty times higher than that of the electron, still making it the most promising candidate for indirect new physics searches.

1.2.3 *Which contribution?*

TODO: check flow

The largest error contribution stems from the LO-HVP contribution to the anomalous magnetic momentum of the muon. As the previous official SM prediction by the Muon $g-2$ Theory Initiative [57] includes the data-driven result for this contribution [54], the current iteration includes the determination on the lattice as official consensus result [55]. The reasons for this are manifold. The data-driven result – in tension with the experimental value [76, 77] – is itself obtained using experimental data, namely from the cross-section $\sigma(e^+e^- \to \text{hadrons})$ as a function of the center of mass energy $\sqrt{s}$ [78–83]. This is certainly not a first principles theory prediction, but was until recently the only one whose error was competitive with the experimental error estimations. Algorithmic, theoretical and computational advancements over the years made the quantity accessible in the framework of lattice QCD. In particular part ii of this work will focus on the leading order hadronic vacuum polarization contribution $a_\mu^{\text{LO-HVP}}$ to the muon $g-2$ which renders the most challenging contribution to reduce the variance of.

Contribution	Value $\times 10^{11}$
QED	$116\,584\,718.8(2)$
EW	$154.4(4)$
HlBl	$115.5(9.9)$
LO-HVP	$7\,132(61)$
N(N)LO-HVP	$-87.2(1.6)$
Total SM	$116\,592\,033(62)$

TABLE 1.2: Contributions to a_μ^{SM}. Data in this table was taken from ref. [55].

TL;DR: motivation of variance reduction

When analyzing where the error comes from in the SM prediction, it is common to decompose the full quantity into contributions

$$a_\mu^{\text{SM}} = a_\mu^{\text{QED}} + a_\mu^{\text{EW}} + a_\mu^{\text{LO-HVP}} + a_\mu^{\text{HlBl}} \,, \tag{1.47}$$

from QED, electroweak interactions, the leading order HVP and the hadronic light-by-light, see table 1.2 for their respective values. We find a variance (square of the error) contributions as in fig. 1.2. Clearly the variance of the leading order HVP contribution that is determined on the lattice is dominant with 97% contribution to the total variance [55]. From the lattice perspective, it is convenient to decompose $a_\mu^{\text{LO-HVP}}$ further into flavor contributions

$$a_\mu^{\text{LO-HVP}} = a_\mu^{\text{LO-HVP}}(ud) + a_\mu^{\text{LO-HVP}}(s) + a_\mu^{\text{LO-HVP}}(c) + a_\mu^{\text{LO-HVP}}(disc) \,. \tag{1.48}$$

Dominant is the light-quark connected contribution, $a_\mu^{\text{LO-HVP}}(ud)$ (see fig. 1.3), followed by a non-negligible contribution from disconnected diagrams. It is the light-quark connected contribution that suffers from a lacking translation average in the large distance regime, which is the leading source of the large error.

1.2.4 *The leading order HVP contribution*

The goal of the Muon $g-2$ Theory Initiative is to achieve a relative error contribution of the whole LO-HVP in the regime of a per mill [55, 57]. This poses an immense theoretical, algorithmic and computational challenge. Compared to hadronic scales the muon mass is small, implying the largest contribution to a_μ must come from large distances.

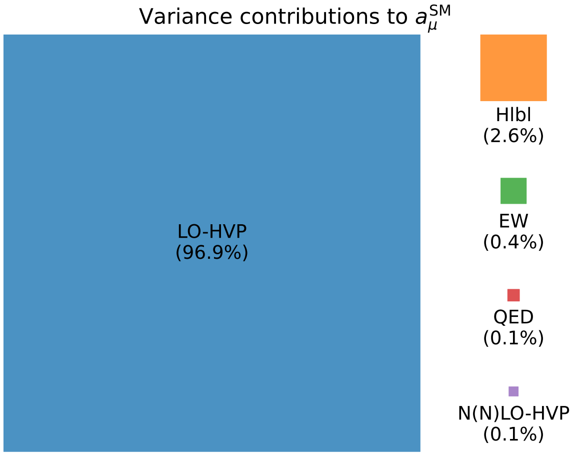


FIGURE 1.2: Variance contributions to a_μ^{SM}; the official Standard Model prediction to the anomalous magnetic moment of the muon $g-2$. The areas are proportional to the variances, whereas the side lengths are proportional to the errors. The contributions stand for: LO-HVP: lattice determination of the leading order HVP contribution (the one we deal with in this thesis), HlBl: mixed (phenomenology and lattice) result for the hadronic light-by-light contribution up to next-to-leading order (NLO), EW: electro-weak contribution, QED: quantum electrodynamics contribution up to tenth order, N(N)LO-HVP: phenomenological result of higher order terms of the HVP. Data for this plot was taken from ref. [55].

The observable of interest is the connected two-point isovector vector current correlator in the time-momentum representation [84]. The local electromagnetic vector current is defined as

$$j_\mu(x) = q\bar{\psi}(x)\gamma_\mu\psi(x)\ , \quad \mu = 0,1,2,3, \tag{1.49}$$

where q is the charge of the quark, ψ is the Grassmann-valued quark spinor field and γ_μ exposes the quantum numbers of a vector particle. From this current, we can define the two-point correlation function

$$C_{\mu\nu}(x;y) = \langle j_\mu(x) j_\nu(y)\rangle\ , \tag{1.50}$$

with a source y and a sink x. In order to relate this quantity to the energy states of the system, we perform the Fourier transform over the spatial components of the sink variable x, project to zero momentum and average over the spatial components of the current $\mu = \nu = 1,2,3$. Finally we exploit

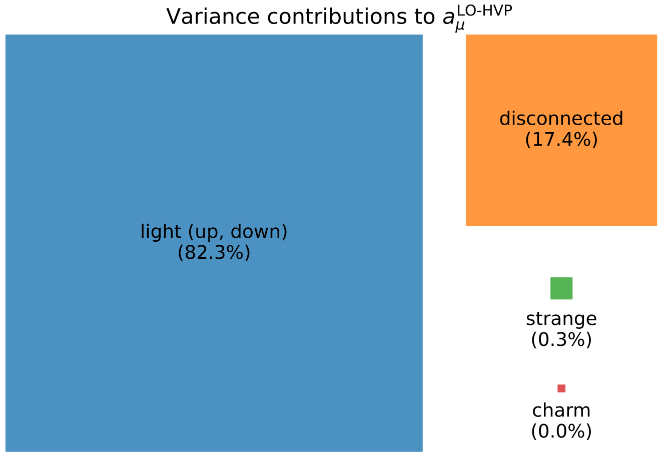


FIGURE 1.3: Variance contributions to $a_\mu^{\text{LO-HVP}}$ (blue square in fig. 1.2); the official Standard Model prediction to the leading order hadronic vacuum polarization contribution to the anomalous magnetic moment of the muon $g-2$. The areas are proportional to the variances, whereas the side lengths are proportional to the errors. The contributions stand for: light: light quark connected contribution (the one we deal with in this thesis), disconnected: sum of all disconnected flavor contributions, strange/charm: contribution of the strange/charm quark flavor. Data for this plot was taken from ref. [55].

translation invariance of the expression by averaging over some (at this point unspecified) set of distinct source points $y \in \Omega$. Furthermore, the correlation function only depends on the time difference between source and sink point $t = x_0 - y_0$ and we finally find the Euclidean time correlator

$$G(t) = -\frac{1}{3|\Omega|} \sum_{y\in\Omega} \sum_{k=1}^{3} \int \mathrm{d}^3\vec{x}\; C_{kk}(t+y_0, \vec{x}; y) \; . \tag{1.51}$$

This correlator contains all the non-perturbative hadronic information.

The set of source spacetime points Ω emphasizes the translation invariance of the expression. In the continuum, eq. (1.51) evaluates to the same result for every source point y. It is the lattice version of eq. (1.51), precisely defined in chapter 12, where the average over many source points $y \in \Omega$ becomes crucial[9] to achieve minimal variance of $G(t)$. Current high-precision determinations of the connected light-quark LO-HVP do not manage to realize the translation average in the long distance to reach minimal variance of eq. (1.51).

---

[9] The same applies to the average over the 3 spatial components $k = 1, 2, 3$.

The LO-HVP contribution can then be written as [84]

$$a_\mu^{\text{LO-HVP}} = \left(\frac{\alpha}{\pi}\right)^2 \int_0^\infty \mathrm{d}t\, G(t) K(t; m_\mu)\,, \tag{1.52}$$

where $K(t; m_\mu)$ is a known analytic kernel function for the leading order depending on the Euclidean time t and the muon mass m_μ only and is given by [85]

$$K(t; m_\mu) = 8\pi^2 \int_0^\infty \frac{\mathrm{d}\omega}{\omega} f(\omega^2) \left[\omega^2 t^2 - 4\sin^2\left(\frac{\omega t}{2}\right)\right]\,, \tag{1.53}$$

with

$$f(\omega^2) = \frac{m_\mu^2 \omega^2 Z^3 (1 - \omega^2 Z)}{1 + m_\mu^2 \omega^2 Z^2}\,, \quad Z = -\frac{\omega^2 - \sqrt{\omega^2 + 4 m_\mu^2 \omega^2}}{2 m_\mu^2 \omega^2}\,. \tag{1.54}$$

1.2.5 *Window quantities*

The community pushed forward the practice of comparing so called *windows* of the LO-HVP contribution, first introduced in ref. [86, 87]. The integral over Euclidean time in eq. (1.52) is split into three regions. Restricted to the windows, the quantity allows for better comparison among lattice groups, because each window has different sources of uncertainty. The boundaries of the integral are smoothed of by a convolution with a hyperbolic tangent,

$$a_\mu^{\text{LO-HVP}} = a_\mu^{\text{SD}} + a_\mu^{\text{W}} + a_\mu^{\text{LD}}\,, \tag{1.55}$$

$$a_\mu^{\text{SD}} = \left(\frac{\alpha}{\pi}\right)^2 \int_0^\infty \mathrm{d}t\, G(t) K(t; m_\mu) \left[1 - \Theta(t, t_0, \Delta)\right]\,, \tag{1.56}$$

$$a_\mu^{\text{W}} = \left(\frac{\alpha}{\pi}\right)^2 \int_0^\infty \mathrm{d}t\, G(t) K(t; m_\mu) \left[\Theta(t, t_0, \Delta) - \Theta(t, t_1, \Delta)\right]\,, \tag{1.57}$$

$$a_\mu^{\text{LD}} = \left(\frac{\alpha}{\pi}\right)^2 \int_0^\infty \mathrm{d}t\, G(t) K(t; m_\mu) \left[\Theta(t, t_1, \Delta)\right]\,, \tag{1.58}$$

with

$$\Theta(t_0, t_1, \Delta) = \frac{1}{2}\left(1 + \tanh\left(\frac{t_0 - t_1}{\Delta}\right)\right)\,. \tag{1.59}$$

The function is smooth and has a width parameterized by Δ, to which the community agreed for $\Delta = 0.15$ fm, $t_0 = 0.4$ fm and $t_1 = 1.0$ fm [87].

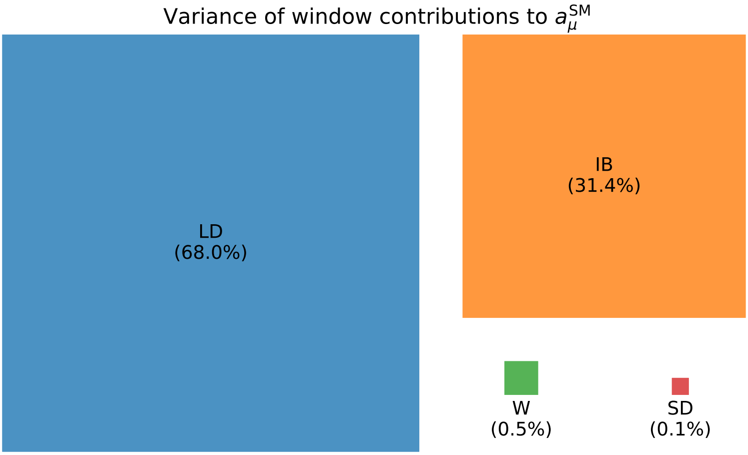


FIGURE 1.4: Variance contribution if the windows to $a_\mu^{\text{LO-HVP}}$ (blue square in fig. 1.2). The areas are proportional to the variances, whereas the side lengths are proportional to the errors. The contributions stand for: LD: long-distance window, IB: strong and QED isospin-breaking corrections, W: intermediate window, SD: short-distance window. Data for this plot was taken from ref. [55].

Through this choice, the intermediate window (W) is less sensitive to discretization effects and the signal-to-noise ratio problem which are dominant in the short-distance (SD) and long-distance (LD) window, respectively.

TL;DR: Motivation of QCD+QED

It is instructive to look at the decomposition into windows as in eq. (1.55) and isospin-breaking (IB) corrections

$$a_\mu^{\text{LO-HVP}} = a_\mu^{\text{SD}} + a_\mu^{\text{W}} + a_\mu^{\text{LD}} + a_\mu^{\text{IB}} \,, \tag{1.60}$$

where here the isospin-symmetric component was decomposed into windows and a remainder accounts for the first order strong and QED isospin-breaking (IB) corrections due to the mass difference between up and down quarks and QED effects. Concerning variance contributions (see fig. 1.4), the dominant overall contribution comes from the large distance regime, where a full translation average is prohibitively expensive, but also from the IB contribution. Contributions to the *value* of $a_\mu^{\text{LO-HVP}}$ from the IB effects are expected to be in the 1% regime, but we see dramatic contribution to the variance.

1.3 HIGH-PRECISION PHYSICS IN PARTICLE PHENOMENOLOGY

TL;DR: theory: high-precision lattice is +QED

The last section demands high precision determinations of physical quantities on the lattice such as the anomalous magnetic moment of the muon. Such determinations are required to target sub-percent precision on the resulting quantity [54, 55, 88–94]. This poses major challenges for a number of conceptual and computational reasons.

High precision determinations of gold-plated hadronic quantities like the one mentioned require us to consider quantum electrodynamic (QED) effects, because their contribution to this observable exceeds the sub-percent. This can be done perturbatively in a QCD-only simulation in the observable evaluation phase (often refereed to as RM123 method [95, 96]) or by dynamically simulating both theories QCD+QED already in the gauge field sampling phase [97–101].

TL;DR: our theory approach: we do QCD+QEDC + Cstar boundaries

The approach of the RC* collaboration for high precision lattice determinations is to non-perturbatively simulate QCD+QED directly in the Hybrid Monte Carlo algorithm when sampling gauge fields. This is accomplished through the use of C* spatial boundary conditions [102–105] which respect locality and Gauss' law, while at the same time reducing finite-size effects compared to periodic boundaries or other lattice QCD+QED formulations. This approach allows for a local and gauge-invariant formulation of QCD+QED in a finite volume. We will briefly introduce these boundaries and the dynamic incorporation of QED in the next two subsections.

1.3.1 *C* boundary conditions*

TL;DR: intro cstar

Gauss' law prohibits dynamical simulations of charged particles in a finite box with periodic boundary conditions. A local[10], compact and gauge-invariant formulation of QED theory on the lattice can be provided by using C* boundary conditions along spatial lattice extents [102–105]. This theory is usually referred to as QED$_\mathrm{C}$ [97]. It has some useful advantages like no zero-modes of the gauge field or small finite-volume effects due to its locality, compared to other (non-local) QED formulations on the lattice. Disadvantages are its partial breaking of flavor conservation and continuum and infinite volume limits do not commute, although mixing effects are exponentially suppressed.

TL;DR: other QED lattice formulations and their pros/cons

Numerous alternative formulations of QED on the lattice exist. QED$_\mathrm{L}$ [98,

---

[10] By locality we mean that the Dirac operator does not have interactions spanning the whole lattice, i. e. only nearest neighbor interactions.

106] is local and removes the spatial zero modes of the photon [97] but introduces potentially problematic non-localities in space, limits do not commute [107] and it has unphysical finite-volume effects. Furthermore, the fully local and consistent formulation $\mathrm{QED_m}$ [99] introduces a photon of non-zero mass γ_m, whose limit $\gamma_m \to 0$ does not commute with the other limits. $\mathrm{QED_{TL}}$ [108] is non-local in time and limits do not commute. $\mathrm{QED_{SF}}$ [109] restricts the zero mode of the photon field which allows charged states to propagate but is non-local in time. Infinite volume QED, QED_∞ [87, 100, 101, 110] subtracts $\mathrm{QED_L}$ long-range, power-law finite-volume effects proportional to $1/L$ but introduces non-localities.

The prescription used throughout this thesis is $\mathrm{QED_C}$ by employing spatial C* boundary conditions to fields. Fields are only periodic up to charge conjugation, that is

TL;DR: Def of Cstar boundaries

$$\begin{aligned} \psi(x + L_k\hat{k}) &= \psi^{\mathcal{C}}(x) = C^{-1}\bar{\psi}^{\mathrm{T}}(x)\,, \\ \bar{\psi}(x + L_k\hat{k}) &= \bar{\psi}^{\mathcal{C}}(x) = -\psi^{\mathrm{T}}(x)C\,, \\ A_\mu(x + L_k\hat{k}) &= A_\mu^{\mathcal{C}}(x) = -A_\mu(x)\,, \\ U_\mu(x + L_k\hat{k}) &= U_\mu^{\mathcal{C}}(x) = U_\mu^*(x)\,, \end{aligned} \tag{1.61}$$

where ψ is a fermion field. The vector $\hat{k}$ is the unit vector in spatial k-direction with $k = 1, 2, 3$, $\mu = 0, 1, 2, 3$ denotes the space-time direction and L_k is the spatial lattice extent in direction k. The star symbol U^* denotes element-wise complex conjugation (no transposition). $A_\mu(x) \in \mathfrak{u}(1)$ is the Lie-algebra-valued photon field, $U_\mu(x) \in SU(3)$ the Lie-group-valued gluon field and the subscript $\mathcal{C}$ stands for the charge conjugated field. Together they form the U(3)-valued compound field

$$W_\mu(x) = e^{iqA_\mu(x)}U_\mu(x) \in U(3)\,, \tag{1.62}$$

with q the charge. Finally C is the invertible charge conjugation matrix obeying

$$C^{-1}\gamma_\mu C = -\gamma_\mu^{\mathrm{T}} \quad \text{and} \quad \det(C) = 1 \tag{1.63}$$

with the Euclidean γ-matrices. Charge conjugating the fields twice should yield back the original field. Thus shifting by twice the spatial lattice extent, the fermion field transforms as

$$\psi(x + 2L_k\hat{k}) = C^{-1}\bar{\psi}^{\mathrm{T}}(x + L_k\hat{k}) = -C^{-1}C^{\mathrm{T}}\psi(x)\,, \tag{1.64}$$

$$\bar{\psi}(x + 2L_k\hat{k}) = -\psi^{\mathrm{T}}(x + L_k\hat{k})C = -\bar{\psi}(x)(C^{-1})^{\mathrm{T}}C\,. \tag{1.65}$$

If we chose the matrix C to be skew-symmetric $C^{\mathrm{T}} = -C$, the RHS equals $\psi(x)$ and $\bar{\psi}(x)$, respectively and the fermion field is periodic in space in twice the lattice extent L_k. Such a matrix C always exists in four dimensions.

TL;DR: physical, mirror and extended lattice

Since ψ and $\bar{\psi}$ are independent degrees of freedom, we see from eq. (1.61) that a fermion field can be seen as a single degree of freedom on a larger lattice extended in all four spacetime directions by a factor of two. The fermion field is then dynamic on that whole *extended lattice* Λ_{ext} of extents L_0 and $2L_k$, as compared to the gauge field which is only dynamic on the *physical lattice* Λ_{phys} of extents L_μ. Finally, the part of the extended lattice where fields are charge conjugated will be called the *mirror lattice* Λ_{mirr} such that $\Lambda_{\text{ext}} = \Lambda_{\text{phys}} \cup \Lambda_{\text{mirr}}$.

TL;DR: Extended lattice is requirement not implementation trick

Therefore, the photon and the gluon field on the mirror lattice are completely determined by charge conjugating their values on the physical lattice. Their integration measure in the path integral eq. (1.26) is simply the Haar measure and is given by

$$\mathcal{D}W\big|_{\Lambda_{\text{phys}}} = \prod_{\mu=0}^{3} \prod_{x \in \Lambda_{\text{phys}}} \mathrm{d}W_\mu(x) \,. \tag{1.66}$$

On the other hand, ψ and $\bar{\psi}$ are independent Grassmann variables on the physical lattice, whereas on the extended lattice $\bar{\psi}$ is completely determined by ψ. In the path integral the integration measure of the Grassmann variables therefore turns into

$$\mathcal{D}\psi\big|_{\Lambda_{\text{phys}}} \mathcal{D}\bar{\psi}\big|_{\Lambda_{\text{phys}}} = \prod_{x \in \Lambda_{\text{phys}}} \mathrm{d}\psi(x)\mathrm{d}\bar{\psi}(x) = \prod_{x \in \Lambda_{\text{ext}}} \mathrm{d}\psi(x) = \mathcal{D}\psi\big|_{\Lambda_{\text{ext}}} \,. \tag{1.67}$$

The fermion field is dynamic on the whole extended lattice, where the physical and mirror lattices can also be seen as a further internal degree of freedom or an additional index $i \in \{\text{phys}, \text{mirr}\}$ of the fermion field.

TL;DR: extended lattice is not 8x, but 2x larger

The extended lattice is 8 times larger than the physical one (2 in each spatial direction). However, the data points on the mirror lattice though are redundant, because no matter in which spatial direction we leave the physical lattice in eq. (1.61), we always end up in the same mirror lattice, we only enter from different directions. By defining shifted boundary conditions on the extended lattice, one can thus get away with an extended lattice that is just twice as large as the physical one, irrespective of the number of spatial C* directions. This is called the orbifold construction [111] which is useful for software implementations of C* boundary conditions. Saying that C* simulations are twice as expensive as periodic ones would still be ignorant to the many peculiarities of gauge field generation and observable evaluation. Such a comparison is highly non-trivial as has not been fully carried out [112] as of today[11].

1.3.2 *Lattice QCD+QED*

For QCD+QED simulations, the Dirac operator eq. (1.24) has to be modified to include electromagnetic interactions and it has to act on the pseudofermion doublet $(\psi, \bar{\psi})$ simultaneously. We will use the symbol ψ in the following to represent the pseudofermion doublet field on the extended lattice. In addition to the SU(3)-valued QCD gauge field $U_\mu(x)$, we have the compact $\mathfrak{u}(1)$-valued QED gauge field $A_\mu(x)$ which when combined, results in a U(3)-valued field $W_\mu(x) = e^{iqA_\mu(x)}U_\mu(x)$ with q the charge of a quark. These links are produced by multiplying the U(1) phase to the SU(3) matrices.

TL;DR: QED in Dop

For $\mathcal{O}(a)$-improvement, we add another SW-term with separate coefficient,

TL;DR: QED SW term

$$D \rightarrow D + qc_{\text{SW}}^{U(1)} \frac{i}{4} \sum_{\mu,\nu=0}^{3} \sigma_{\mu\nu} \hat{A}_{\mu\nu}\,, \tag{1.68}$$

where q is the charge and the $U(1)$ field strength tensor $\hat{A}_{\mu\nu}(x)$ is defined as

$$\begin{aligned} \hat{A}_{\mu\nu}(x) &= \frac{i}{4q_{el}} \text{Im}\left\{z_{\mu\nu}(x) + z_{\mu\nu}(x-\hat{\mu})\right. \\ &\qquad\qquad \left. + z_{\mu\nu}(x-\hat{\nu}) + z_{\mu\nu}(x-\hat{\mu}-\hat{\nu})\right\}, \\ z_{\mu\nu}(x) &= e^{i\left\{A_\mu(x)+A_\nu(x+\hat{\mu})-A_\mu(x+\hat{\nu})-A_\nu(x)\right\}}, \end{aligned}$$

similar to the SU(3) field strength with $q_{el} = \frac{1}{6}$ the elementary charge. We therefore have another algorithmic parameter to consider in the tuning; the $U(1)$ SW-coefficient $c_{\text{SW}}^{U(1)}$. This is usually set to its tree-level improvement value 1 [112]. Its deviation from 1 is small due to small quantum corrections in the Abelian case, $c_{\text{SW}}^{U(1)} = 1 + \mathcal{O}(\alpha_{em})$ with $\alpha_{em} \sim 1/137$ the electromagnetic coupling constant.

---

[11] One has to consider the missing experience in tuning C* simulations as opposed to periodic ones where the field has decades of experience but also partial lack of theoretical foundation, differences in observable evaluation just to name a few. Cost savings of incorporating QED dynamically, rather than perturbatively are not trivial to estimate either. Any attempt would go beyond the scope of this thesis.

TL;DR: Full QCD+QED Wilson Clover Dop

Therefore, the full lattice QCD+QED Wilson-Clover Dirac operator of a given flavor applied to a pseudofermion field ψ is

$$\begin{aligned} D_\mathrm{w}\psi(x) = (4+m)\psi(x) + \frac{1}{2}\sum_{\mu=0}^{3}\Big\{&U_\mu(x)e^{i\hat{q}A_\mu(x)}(1-\gamma_\mu)\psi(x+\hat{\mu}) \\ &+ U_\mu(x-\hat{\mu})^{-1}e^{-i\hat{q}A_\mu(x)}(1+\gamma_\mu)\psi(x-\hat{\mu})\Big\} \\ +c_\mathrm{sw}^{SU(3)}\frac{i}{4}\sum_{\mu,\nu=0}^{3}\sigma_{\mu\nu}\hat{F}_{\mu\nu}(x)\psi(x) &+ qc_\mathrm{sw}^{U(1)}\frac{i}{4}\sum_{\mu,\nu=0}^{3}\sigma_{\mu\nu}\hat{A}_{\mu\nu}(x)\psi(x), \end{aligned} \tag{1.69}$$

with the normalized integer charge $\hat{q} = \frac{q}{q_{el}}$ and the bare quark mass m. Here, $x \in \Lambda_\mathrm{ext}$, thus the Dirac operator is acting on a spinor field doublet defined on the extended lattice. Its dimension is double the periodic case and neighboring points $x \pm \hat{\mu}$ respect the shifted boundaries as discussed in section 1.3.1.

1.4 MOTIVATIONS OF THIS THESIS

TL;DR: high-precision and QED

In the search for BSM physics, lattice QCD has entered the era of sub-percent precision. At this level of accuracy, electromagnetic corrections can no longer be neglected and must be included alongside QCD in the lattice simulations. This is the case, for example, for the LO-HVP contribution to g-2, discussed in section 1.2, but also for other processes like leptonic [113, 114] and semi-leptonic [115] decays of pions and kaons. To address this, our collaboration has developed a code, openQxD, to simulate QCD+QED dynamically and non-perturbatively, making use of the advantages of C* boundary conditions.

TL;DR: need GPU code and variance reduction

The motivation for this thesis stems from two main sources. First, for observables sensitive to long-distance correlations, such as the light-quark connected LO-HVP, large-scale lattice simulations are necessary. Computing the full translation average with sufficient statistics to control the noise at large time separations quickly becomes computationally prohibitive. Variance reduction techniques that efficiently suppress noise while maintaining minimal variance[12] independent of lattice volume are essential. Second, since the openQxD code was originally designed for CPU-only clusters, which are being phased out in favor of modern GPU-based architectures, the software needs to be modernized to take full advantage of current GPU architectures.

---

[12] The term *minimal* or *gauge variance* will be rigorously defined in section 17.3. It is the reminiscent variance due to the gauge field fluctuations obtained by performing the full spacetime volume average of eq. (1.51) on the lattice.

1.4.1 *Long distance noise reduction for high-precision physics*

TL;DR: LD: currently no translation average

As discussed in section 1.2.3, the uncertainty in the Standard Model prediction for $g-2$ is dominated by the leading-order hadronic vacuum polarization (LO-HVP) contribution. When computed on the lattice, the main source of uncertainty within LO-HVP comes from the light-quark connected contribution ($a_\mu^{\text{LO-HVP}}(ud)$ in eq. (1.48)), which accounts for about 80 % to the error (fig. 1.3). Large, fine lattices cause standard evaluation methods to break down for such light quark masses. Furthermore, the long distance (LD) regime contributes about 70 % the total variance of the SM prediction (fig. 1.4). Current determinations do not achieve minimal variance in the LD regime, not even with the expensive low mode deflation methods that attempt a translation average. The variance reduction method developed in part ii of this thesis is aiming at the evaluation of exactly this piece realizing the full translation average. While tested on Wilson fermions, the method is more general and potentially useful to a larger part of the lattice community.

TL;DR: IB: need QCD+QED

Due to the large error contribution of the isospin-breaking (IB) corrections of over 30 % (fig. 1.4), good prescriptions for dynamic QCD+QED simulations are thus desperately needed, rather than perturbative expansions of QCD-only simulations. First results from our collaboration suggest that simulating QCD+QED directly, instead of adding IB effects perturbatively on the top of QCD-only results, can be beneficial for the uncertainty of the IB term [P2]. The details of this contribution, however, fall outside the scope of this thesis.

1.4.2 *Efficient usage of HPC resources*

TL;DR: world energy politics

In the current world, reducing energy consumption is more important than ever. This obligates us as researchers to treat the resources we are allowed to use with extra care, specially in the field of computational lattice field theory – one of the biggest consumers of HPC. Since modern HPC infrastructure trends towards massively parallel GPU systems or heterogeneous architectures, research in computational sciences has the responsibility to utilize these new compute resources efficiently and sensibly. As available pure-CPU compute resources are decreasing yearly, it is expected for them to vanish entirely in the not-so-far future. Heterogeneous systems can be significantly more energy efficient than the legacy CPU-only clusters when the computational power of GPUs is properly utilized. If that does not happen, a stagnation of the field is inevitable. Only large collaborations will thrive, and the field may lose in terms of diversity of independent techniques that help cross-check and confirm results.

TL;DR: keep up with the HPC-dashians

Enabling openQxD to run on GPUs as realized in part i, thus opens important doors for exploring isospin-breaking corrections to hadronic observables with $C^\star$ boundaries. It is evident that HPC hardware continues to change[13]. As soon as the GPU is harvested in terms of efficient usage, the HPC community may continue with practicing heterogeneous computing[14] – it already does. Therefore, we explored heterogeneous computing as well in section 6.2.4.

TL;DR: algorithmic improvements are needed

As important as software developments and efficient utilization of available resources are, the field will not advance without algorithmic developments. Specially methods to systematically and iteratively improve estimations of observables by obtaining minimal variance are pivotal for numerical high precision studies of physical quantities. As traditional estimators either fail the translation average or become prohibitively expensive as we approach large, physical lattices, we need efficient variance reduction methods utilizing the available resources more efficiently.

1.5 THESIS OBJECTIVES AND SCOPE

The list below provides a comprehensive summary of the objectives of this thesis, all aimed at addressing the challenges outlined in section 1.4.

1. Design and efficient implementation to offload of expensive solves of the Dirac equation to the GPU. Special attention must be given to ensure that features – such as $C^\star$ boundary conditions and QCD+QED Dirac operators as of eq. (1.69) – do not hinder performance. Implementations should be done in a modular and versatile fashion for convenient usage for application-level developers and lay foundation for future developments. *Corresponding chapters: 5 and 6.*

---

[13] An example for a new hardware trend is the intelligence processing unit [116] (IPU) specially designed for machine learning workloads.

[14] Heterogeneous computing refers to the *simultaneous* utilization of different types of compute units such as CPUs or GPUS within a single system.

2. Performance assessment and benchmark of all implemented features on target HPC infrastructure including GPUs. *Corresponding chapters: 7.*
3. Establishment of a CI/CD automated testing pipeline running on target hardware and production problems. *Corresponding chapters: 8.*
4. Algorithmic development of variance reduction methods based on the low-mode deflation technique targeted at observables sensitive to low-mode noise where full translation averaging is infeasible. *Corresponding chapters: 14 to 17.*
5. Theoretical understanding of spectral properties of Dirac operators and their coarse-grid variants. *Corresponding chapters: 18.*

The above objectives all target efficient evaluation of observables from a machine-centric but also from an algorithmic perspective.

The scope of these developments concerns offloading inversions to the GPU and does not extend generally to the generation of gauge configurations, although design decisions were made with that in mind. A proposal for a direct continuation of the software developments in the direction of gauge field generation is stated in chapter 9, it is not the only possibility to continue. Benchmarks are restricted to systems with NVIDIA GPUs, because of availability of compute resources, although data for AMD GPUs has been investigated too, collected on the Finnish machine *LUMI*. Some of this can be found in ref. [P11] and currently unpublished in ref. [P4], where also data from the decommissioned *Daint hybrid* machine at CSCS can be found. This data is from an old version of the code and was therefore left out of this document.

The first part of this document is dedicated to the advancements in software development by interfacing the simulation program openQxD with QUDA. The foundation of this work was my master's thesis [P3, P5] and it is partially based on refs. [P4, P11].

The second part investigates variance reduction using low-mode deflation techniques and is based on refs. [P1, P6]. Some of the plots and illustrations presented in this thesis were taken from ref. [P1]. If so, it is indicated specifically in the captions.

A path not further pursued during my PhD was the evaluation of the leading order HVP of the muon $g-2$ using C* boundary conditions and QCD+QED gauge ensembles. Publications related to this can be found in refs. [P2, P7–P10].

1.6 SUMMARY OF CONTRIBUTIONS

In this section, I declare my own achievements and contributions to this thesis according to article 13 of "Detailed stipulations regarding the doctorate" [117]. I wrote all chapters myself. I received valuable feedback from colleagues and collaborators, which helped improve the presentation and clarity of this thesis. All figures, plots and images were produced by me. Some were taken from the publication [P1].

CHAPTER 1 : This chapter is introductory, based on established literature and does not contain original research.

Part I: GPU Porting of Dirac Solvers

- CHAPTER 2 : This chapter is introductory, based on established literature and does not contain original research.
- CHAPTER 3 : I compiled this chapter based on access to the source code of openQxD and its documentation. No original results are presented here.
- CHAPTER 4 : I compiled this chapter based on access to the source code of QUDA and its documentation. No original results are presented here.
- CHAPTERS 5 AND 6 : These chapters describe the software components I developed as part of this project. I was responsible for implementing the core functionality of interfacing openQxD with QUDA. The design of the code was discussed and refined in collaboration with members of the RC* collaboration, CSCS engineers and group member related to the PASC project [118]. The latter also contributed to specific components and architectural decisions. All documentation and the write-up of this chapter were done by me.
- CHAPTER 7 : This chapter shows performance data of interface components. I conducted all related runs independently and the analysis of data was done by me. Feedback from members of the group and the RC* collaboration was incorporated.
- CHAPTER 8 : The automated testing pipeline was designed and introduced to the development repository [119] by me. CSCS engineers were involved in consultation for best practices.

CHAPTER 9 : The memory management proposal was developed by me and a small proof-of-concept program was written by me too.

CHAPTER 10 : This chapter was written by me and reflects my own views and conclusions.

Part II: Variance Reduction for Two-Point Functions

CHAPTER 11 : This chapter is introductory, based on established literature and does not contain original research.

CHAPTER 12 : This chapter introduces the relevant formulae required later. Apart from notation, no original results are presented here.

CHAPTER 13 : This chapter introduces the important variance reduction method low-mode averaging and some of its many variants. No original results are presented here.

CHAPTER 14 : The novelty in this chapter is the deflation of arbitrary subspaces required later.

CHAPTER 15 : Local coherence is a concept known in literature. This chapter briefly introduces the concept. No original results are presented here.

CHAPTER 16 : Multigrid preconditioning is an established concept and widely used in the field. The original research consists of the decomposition of the propagator, its usage as variance reduction method and developments that follow. This was developed by me based on advise and feedback of colleagues and collaborators.

CHAPTER 17 : I have developed the observable evaluation code, based on functionality of openQxD, the recursive multigrid code and all results were generated by me. The gauge fields used for the numerical analysis of the method, apart from the lattice F7 [120], were generated by Tim Harris. Feedback from my supervisor Marina Marinković and Tim Harris helped improve the simulation strategy.

CHAPTER 18 : Theoretical insight from this chapter are my own. Textbook results from numerical range theory were used, but also indicated if so. Ideas about preservation of chirality are partially based on existing literature. Many of the formal proofs are original results.

CHAPTER 19 : This chapter was written by me and reflects my own views and conclusions.

CHAPTER 20 : This chapter was written by me and reflects my own views and conclusions.

1.7 STRUCTURE OF THE THESIS

This document is split into two parts which are to a large extent independent of each other but pulling on the same rope. Part i deals with the advancements in software development, GPU porting and the interface to QUDA. It deals with objectives 1 to 3 as stated in section 1.5 associated to chapters 5 to 9. Its heart piece provides a documentation of the design choices, all the functionality that have entered the openQxD and QUDA codebases, and finishes with an assessment of performance. Part ii deals with the remaining objectives 4 to 5 as stated in section 1.5 associated to chapters 14 to 18, introduces the new variance reduction method which we call *multigrid low-mode averaging* and investigates its application to an observable where the full translation average is infeasible. Results are compared to established estimators. It finishes with theoretical considerations about chirality on coarse subspaces. Both parts have separate outlines and introductions.

Part I

GPU PORTING OF DIRAC SOLVERS

2

INTRODUCTION TO PART I

> ***TODO***: *When allocating resources on a cluster, it is the responsibility of the programmer to utilize all the available resources as efficient as possible.*
>
> — Thomas C. Schulthess

This section/chapter was proofread 1 times.

TL;DR: ML dictates HPC trends -> GPU

In the last few years, modern hardware in supercomputing centers experienced a paradigm shift away from classical monolithic general-purpose CPU architectures towards high throughput and massively parallel GPU accelerators [121–124]. These are substantially different machines targeted at specific problem classes. Namely the computational challenges that come with large machine learning and artificial intelligence (ML/AI) workloads required for instance for large language models – one of the biggest consumers of high performance computing (HPC) and the main influencers of new hardware developments and trends in the area of scientific computing. The ML/AI community has pushed hardware manufacturers to build specialized integrated circuits directly aiming at their problems. An example of such a new hardware development that entered the GPU is the tensor core [125, 126] – a hardware unit to perform small dense matrix multiply-accumulate operations.

TL;DR: ML is dense GEMM, multi-GPU, mem-bound, training is markov chain

A typical machine learning pipeline consists of about 90-95 % linear algebra in the form of memory bandwidth limited dense large matrix-matrix or matrix-vector multiplications (GEMM, GEMV) and a small remainder of about 5-10 % non-linear activation functions which are usually compute bound, because they often involve transcendental functions[1][2]. These GEMMs may have multiple thousands of rows and columns and are tiled into many sub-matrices to fit the tensor cores [128]. Memory requirements are proportional to the number of parameters which may go into the trillions for very large models [129]. Since the training steps are comparable to a Markov chain, ML pipelines need to scale out and utilize hundreds of GPUs concurrently to be able to process in a reasonable time.

TL;DR: what is a GPU

Originally designed for efficient parallel real-time graphics rendering,

---

[1] The larger the large language models the more the linear algebra part dominates.

[2] GEMMs may become compute bound on high-bandwidth memory (HBM) and datatypes where no tensor cores are available [127].

GPUs are well-suited for this class of problems, since they have higher memory throughput than CPUs. They were designed with a wider memory bus at the cost of latency, heavily optimized for high throughput and sequential memory access as opposed to CPUs which are optimized for random memory access and low latency. The GPU is a latency hiding machine using caches with a less stringent cache coherency protocol than the CPU. Explicit synchronization through memory barriers might be triggered manually by the developer to enforce all threads to see the same memory. While CPUs have powerful individual cores capable of branch prediction and high single-thread performance, GPUs have thousands of lightweight threads with limited control flow handling, not efficient for frequent branching and thread divergence. The loose cache coherency protocol, the wider memory bus, the reduced memory hierarchy depth and the highly parallel execution model – featuring both SIMD (Single Instruction, Multiple Data) and SIMT (Single Instruction, Multiple Threads) – make GPUs well-suited for many computational scientific tasks, such as machine learning, data analytics, and large-scale simulations.

TL;DR: lattice is Dirac equation solves, sparse, SpMv, Krylov

Lattice QCD simulations are dominated by solves of the Dirac equation – a large sparse linear system of equations governed by the Dirac operator. This is a derivative operator connecting nearest neighbor lattice points on a 4D Euclidean spacetime lattice with covariant derivatives that depend on an additional field connecting the links of the lattice called the gauge field. Solves of this equation are performed using Krylov subspace methods [130, 131], that iteratively refine their solution up to a desired convergence criterion. This class of algorithms implicitly construct a polynomial in the Dirac operator applied to the right-hand side. The main computational kernel is thus the application of this stencil operator to one or multiple vectors (SpMv).

TL;DR: computationally lattice is comparable to ML

Even though Lattice QCD simulations are a somewhat different kind of computational problem compared to machine learning, they can profit from the hardware advancements on GPU accelerators utilizing modern clusters. Most lattice simulations consist of two parts with slightly different computational motifs; sampling of gauge fields and evaluation of observables. However, both are dominated by repeated solves of the Dirac equation. For gauge field sampling, the gauge field defining the Dirac operator might change from solve to solve. On the other hand in case of observable evaluation, the gauge field does not change too frequently, meaning that the same linear system of equations has to be solved for possibly thousands of right-hand sides. The goal of this work is to offload solves in the context of observable evaluation to the accelerator.

TODO: check if this still has a roter faden, because I removed theory: high-precision lattice is +QED and our theory approach: we do QCD+QEDC + Cstar boundaries

TL;DR: our CPU approach (up to now): we did this on CPU, MPI

Our codebase, openQxD [111, 132], extends the widely used openQCD-1.6 package [133] which supports the $O(a)$-improved Wilson fermion discretization (from now on, we will refer to these codebases simply as openQxD and openQCD, respectively). OpenQxD allows the dynamic simulation of QCD together with QED by employing $C^\star$ boundary conditions. These features make openQxD a versatile tool for achieving higher levels of precision. The codebase is designed for efficient parallel execution on pure-MPI CPU-based supercomputing architectures. In order to make openQxD run on modern GPU-based clusters, we have many options.

TL;DR: alternative: openmp offloading

Recent new introductions to the openMP standard [134, 135] consist of powerful GPU offloading directives, that aim at rapid porting of openMP applications to run on GPUs with minimal effort. With a pure-MPI code as basis, investigating this possibility resulted in disappointing results, since a large fractions of the core memory layouts had to be refactored to optimize for such an implementation.

TL;DR: alternative: existing solver suite

Plenty of Krylov solver suites with efficient implementations on various GPU architectures exist [136–143]. Nevertheless an efficient implementation of the Dirac stencil is crucial for such libraries to perform well. Often these general solver suites are not on eye level with state-of-the-art domain-specific solvers that are highly optimized for lattice QCD and its unusual stencil operator. These suites often take the operator in terms of a sparse matrix format, a matrix-free form or a stencil formulation that does not allow fine grained control. The way the gauge field is represented and applied is crucial for the performance of a matrix-vector operation. Precondition methods like multigrid lack comparable implementations for general problems, in contrast to the ones optimized for the various discretizations of the lattice Dirac operator.

TL;DR: alternative: own CUDA/HIP implementation

Another alternative would be to change openQxD's memory layouts to ones more favorable for GPUs and incorporate GPU code in terms of CUDA and/or HIP runtime calls directly into the core components of the application. Certainly this would be the cleanest solution without any external dependencies, but the intrusions into core components will be non-trivial and breaking the CPU code unless one maintains two versions. The effort of such an undertaking will be enormous and results can be expected only late in the development process, optimized code even only at the end.

TL;DR: computationally: We do this now with solves to QUDA

Among the various possibilities for GPU-accelerating lattice QCD calculations, we choose to interface openQxD with the QUDA library [144–146] – a lattice QCD library which provides optimized algorithms for NVIDIA and AMD GPUs, including an efficient iterative solver for the Dirac equation as well as a state-of-the-art multigrid preconditioning tailored for lattice QCD. We implement $C^\star$ boundary conditions and QED effects, as these are features that are particular to openQxD and do not yet exist in QUDA. In this way we combine openQxD's comprehensive physics toolset with QUDA's GPU-accelerated operations, thus enhancing our simulations with an efficient GPU backend.

TL;DR: review: quda

It makes sense to briefly review strategies of other simulation software packages common in the field. As mentioned in the last paragraph, QUDA was designed as pure-GPU code. It is capable of running on various hardware, such as NVIDIA and AMD, but support for SYCL on Intel platforms or a thread backend to run on CPUs are in development. QUDA developers interfaced to Chroma [147, 148], CPS [149] and QDP/C [150] already in 2009 [144].

TL;DR: review: openqcd

Recent releases of openQCD (the package which openQxD is based on) include openMP thread support [133]. Further developments go into direction of a CUDA/HIP porting of large parts of the software package. Development is still in progress in a private GitHub repository and a release date is not communicated yet, a status report can be found in ref. [151].

TL;DR: review: tmlqcd

Developers of the software package tmLQCD [152] already went the path we did by coupling to QUDA. They offload solves of the Dirac equation but in the meantime large parts of the Hybrid Monte Carlo (HMC) algorithm are offloaded too [153–155].

TL;DR: review: grid

Grid [156, 157] is mainly targeted at domain wall and Möbius fermions, but simulations of Wilson-Clover fermions are supported too [158]. It has its own GPU backend and runs efficiently on NVIDIA, AMD and Intel GPUs as well as on CPUs with MPI and openMP thread support [159]. Newest features include blocked solvers for domain wall fermions [160].

TL;DR: review: chroma

Chroma [147] was developed and optimized for CPU based machines [161] and is fully interfaced with QUDA in terms of Dirac equation solves and the HMC.

TL;DR: review: milc

MILC [162, 163] is used for dynamical simulations of staggered and HISQ fermions. It is interfaced to both QUDA and Grid.

TL;DR: Definition of problem to offload

We are interested in solving the lattice Dirac equation eq. (1.25) on the

GPU. It is the most time intensitve kernel given many right-hand sides η. Offloading this kernel allows us to retain the functionality of existing applications while interfacing only a minimal set of parameters. The interface needs to transmit field data from one application to the other. Different conventions, memory layouts, data types, precisions, storage formats, freedom in γ-basis choice, boundary conditions and many more have to be taken into account when designing an interface. The work hereafter concentrates on the measurement part of the workflow, i. e. offloading of solves of the Dirac equation. Although the gauge fields are periodically updated along the Markov chain, in the measurement part of the workflow they are kept fixed.

TL;DR: remainder of the part

The rest of this part is arranged as follows. Chapters 3 and 4 introduce the openQxD and QUDA codebases, their design choices, memory layouts and other features relevant for this work. Chapter 5 details the process of interfacing openQxD with the QUDA library: it discusses the modifications to the memory layout of lattice fields, the implementation of $C^\star$ boundary conditions in QUDA, the extension of QUDA to handle QCD+QED simulations and the various ways to access the interface. Chapter 6 describes the steps needed to call QUDA solvers from within openQxD. Chapter 7 presents benchmark tests of various components of the interface. The strategy for continuous integration and continuous delivery (CI/CD) is discussed in chapter 8. Finally, chapter 9 proposes a way to continue the developments considering gauge field configuration generation on GPUs.

3

OPENQXD

This section/chapter was proofread 1 times.

In this chapter, we outline some of the main features, design choices and peculiarities of openQxD, the simulation program worked on in this thesis (section 3.1). Following a description of the spacetime indexing (section 3.2), we discuss internal memory structures for spinor (section 3.3), gauge (section 3.4) and clover fields (section 3.5). We continue explaining how $C^\star$ boundary conditions (section 3.6) and dynamical QCD+QED (section 3.7) were implemented, before concluding the chapter with a brief summary (section 3.8).

TL;DR: openqxd and openqcd intro

The openQxD code [111] is an extension of the state-of-the-art lattice QCD Markov Chain Monte Carlo code openQCD [133, 164] which complies to the C89 standard with MPI parallelization and has proven strong-scalability to hundreds of CPU-based nodes including specific optimizations for x86 architectures. Due to the extensive suite of legacy applications based on openQCD as well as its renowned stability and reproducibility, our goal is to accelerate measurement routines by offloading the solution to the Dirac equation eq. (1.25) to the GPU accelerator using the QUDA library.

3.1 DESIGN PRINCIPLES

TL;DR: lattice geometry as compile time constants

The lattice topology in terms of process grid, global and local lattice sizes is defined globally at compile time in a header file called `include/global.h` using C preprocessor directives, see listing 3.1. This implies that openQxD must be recompiled every time the lattice or the process grid changes.

```
#define NPROC0 1
#define NPROC1 2
#define NPROC2 2
#define NPROC3 2

#define L0 8
#define L1 8
#define L2 8
#define L3 8
```

```
#define NPROC0_BLK 1
#define NPROC1_BLK 1
#define NPROC2_BLK 1
#define NPROC3_BLK 1
```

Listing 3.1: Excerpt of the file `include/global.h` describing the openQxD process grid.

TL;DR: lack of convenience in openqxd

The program was probably never intended to be used as framework to do arbitrary lattice calculations apart from gauge field sampling. There is no convenient single function to initialize and finalize a program written in openQxD. The codebase is written in a very static way, highly optimized for its exact purpose, making additions and extension to the functionality hard to implement. This exposes itself in a lack of convenience functions for bootstrapping, input file parsing and validation, logging, profiling, field management, log file handling, field index abstraction and many more and culminates usually in lots of code duplication and boilerplate code.

TL;DR: translation units with lots of global state

A common pattern are translation units carrying static global state variables allocated by some initialization function and later accessed from functions within the translation unit. This implies that many functions in openQxD have non-trivial side effects.

TL;DR: Static field allocation scheme

Spinor field allocation and management is handled by a set of functions that allocate, reserve and release fields. The number of fields required for the whole run – including fields used by solvers – has to be known when calling the field allocation function. This has to happen early in the program. Although this practice has a very clean and predictable memory footprint as its advantage, it does not allow a more modern flexible programming style with on demand field allocation.

TL;DR: Ancient C standard

The usage of a quite old standard of the C programming language has implications in the codebase as well. The absence of modern programming paradigms has the advantage that everything is implemented explicitly in a procedural form with little abstraction. On the other hand, modern language features that have impact on performance such as function inlining, the `restrict` keyword for better loop unrolling and automatic vectorization, variable length arrays on the stack instead of the heap, native complex arithmetic support and many more do not exist in C89.

TL;DR: Field layouts differ

In order to offload the aforementioned system of linear equations, all involved fields defined on the lattice in openQxD have to be mapped to fields on the lattice in QUDA. In the following, we discus the layouts of these fields and the way they are represented in openQxD.

3.2 SPACE-TIME INDEXING

TL;DR: spacetime convention and ordering

Denoting the rank-local lattice extent in direction $\mu = 0, 1, 2, 3$ by $L_\mu \in \mathbb{N}$, we can write the lattice coordinate as a 4-vector, $x = (x_0, x_1, x_2, x_3)$, where $x_\mu \in \{0, \dots, L_\mu - 1\}$. OpenQxD puts time coordinate first $x = (t, \vec{x})$ which we refer to as (txyz)-convention. From that we can create a lexicographical index

$$\Lambda(x, L) := L_3 L_2 L_1 x_0 + L_3 L_2 x_1 + L_3 x_2 + x_3. \tag{3.1}$$

OpenQxD orders the indices in so called cache-blocks; a decomposition of the rank-local lattice into equal blocks of extent $B_\mu \in \mathbb{N}$ in direction μ. Within a block, points are indexed lexicographically $\Lambda(b, B)$ as in eq. (3.1), but the L_μ replaced by B_μ and x replaced by the block local Euclidean index b, such that $b_\mu = x_\mu \mod B_\mu \in \{0, \dots, B_\mu - 1\}$. Furthermore, the blocks themselves are indexed lexicographically within the rank-local lattice decomposition into blocks, i. e. $\Lambda(n, N_B)$, where we denote the number of blocks in direction μ as $N_{B,\mu} = L_\mu / B_\mu$, and the Euclidean index of the block as n, such that $n_\mu = \lfloor x_\mu / B_\mu \rfloor \in \{0, \dots, N_{B,i} - 1\}$.

TL;DR: even odd ordering

In addition, openQxD employs even-odd ordering, that is all even-parity lattice points (those where the sum $\sum_{\mu=0}^{3} x_\mu$ of the rank-local coordinate x is even) come first followed by all odd-parity points.

TL;DR: total spacetime index + illustration

Therefore, the total rank-local unique lattice index in openQxD is

$$\hat{x} = \left\lfloor \frac{1}{2}\Big(V_B \Lambda(n, N_B) + \Lambda(b, B) \Big) \right\rfloor + P(x)\frac{V}{2}, \tag{3.2}$$

where $V_B = B_0 B_1 B_2 B_3$ is the volume of a block,

$$P(x) = \tfrac{1}{2}\left(1 - (-1)^{\sum_\mu x_\mu}\right) \tag{3.3}$$

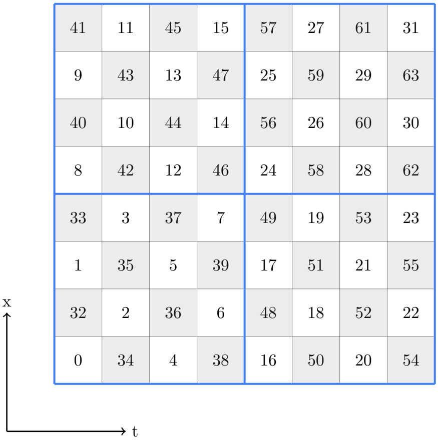
gives the parity of index x, and b, n are related to x as described in the text above (see fig. 3.1 for an example). This is implemented in openQxD by means of the mapping array `ipt`, $\hat{x} := \mathrm{ipt}\left[\Lambda(x, L)\right]$.

3.3 THE SPINOR FIELD

TL;DR: spinor struct

The double precision spinor field is stored as an array of V_L `spinor_dble` structs (listing 3.2) where $V_\mathrm{L} = L_0 L_1 L_2 L_3$ is the rank-local lattice volume.

```
typedef struct
{
   complex_dble c1,c2,c3;
```



41	11	45	15	57	27	61	31
9	43	13	47	25	59	29	63
40	10	44	14	56	26	60	30
8	42	12	46	24	58	28	62
33	3	37	7	49	19	53	23
1	35	5	39	17	51	21	55
32	2	36	6	48	18	52	22
0	34	4	38	16	50	20	54

FIGURE 3.1: 2D example (8 × 8 local lattice) of the rank-local unique lattice index in openQxD (in time-first convention (txyz)). The blue rectangles denote cache blocks of size 4 × 4. Gray sites are odd, white sites are even lattice points.

```
} su3_vector_dble;

typedef struct
{
   su3_vector_dble c1,c2,c3,c4;
} spinor_dble;
```

LISTING 3.2: The spinor field struct

An individual field does not carry any further information apart from its actual data. Meta information like the local size of the field are globally available in terms of compile time constants and the precision of the underlying floating-point number is encoded in the naming of the struct. Furthermore, there is a single precision variant of this struct called `spinor` holding structs of type `su3_vector` again holding `complex`.

TL;DR: complex numbers

As mentioned above openQCD complies to the C89 standard. A native implementation of complex numbers was introduced in C99. This is the reason why openQCD, and by this openQxD, has two structs representing complex floats as defined in listing 3.3.

```
typedef struct
{
   float re,im;
```

```c
} complex;

typedef struct
{
   double re,im;
} complex_dble;
```

Listing 3.3: The `complex` and `complex_dble` struct

These are alignment-compatible to the C99 native data types `float complex` and `double complex`, respectively, thus they can be cast safely[1]. Nevertheless, arithmetic using these structs is cumbersome and error-prone.

The spinor field has indices (x, α, a), where $x = (x_0, x_1, x_2, x_3)$ is the lattice index (Euclidean 4-vector), $\alpha \in \{0, 1, 2, 3\}$ is the spinor index and $a \in \{0, 1, 2\}$ is the color index. From listing 3.2 we can read off the index order of a spinor field: color runs fastest, followed by spin, followed by space-time running slowest. Functions operating on such spinor fields take base pointers of type `spinor_dble*` as input and output arguments.

TL;DR:
spinor field
indices

3.4 THE GAUGE FIELD

The gauge field is stored as an array of $4V_L$ `su3_dble` structs (listing 3.4).

TL;DR:
gauge struct

```c
typedef struct
{
   complex_dble c11,c12,c13,c21,c22,c23,c31,c32,c33;
} su3_dble;
```

Listing 3.4: The gauge field struct

As can be seen, there is no enforcement of the matrices being $SU(3)$ nor compression of the gauge field exploiting the gauge group symmetry. All 9 complex numbers are explicitly written out as row-major matrix. An advantage of such a storage model is that it can easily be extended to arbitrary gauge groups. This made implementing QCD+QED which has a gauge group of $U(3)$ straightforward, see section 3.7. A clear downside is performance. By exploiting the Lie-algebra structure of $\mathfrak{su}(3)$, the gauge field could be stored with 8 real degrees of freedom instead. This would reduce the size of the struct to 8 floating-point numbers instead of 18. In memory bound kernels this would result in a speedup of about two.

---

[1] When including `<complex.h>` for complex number support in C99 and newer, one has to be careful because the single precision complex struct in openQxD has the same name as the complex datatype. Without renaming, there will be naming conflicts.

TL;DR: gauge field index

The gauge field has indices $(x_{odd}, \pm\mu, a, b)$, where x_{odd} is the lattice index (only odd points, $V_{\mathrm{L}}/2$ elements), $\mu \in \{\pm 0, \pm 1, \pm 2, \pm 3\}$ is the direction of the gauge link (in positive as well as negative direction, i. e. 8 possibilities) and a, b parameterize the row and column of the $SU(3)$-valued gauge link (row-major). Thus, openQxD stores the gauge links in all 8 directions at odd lattice points in contrast to most other common conventions which store 4 gauge links in positive directions for every lattice point.

3.5 THE CLOVER FIELD

TL;DR: clover struct + storage form

Finally, it is convenient to precompute the site-local coupling called the clover field which is a function of the gauge-field variables, eq. (1.19). The clover field is an array of $2V_{\mathrm{L}}$ `pauli_dble` structs (listing 3.5).

```c
typedef struct
{
   double u[36];
} pauli_dble;
```

LISTING 3.5: The clover field struct

It is stored in a compressed form exploiting its symmetry and block diagonal form,

$$c_{\mathrm{sw}}^{SU(3)} \frac{i}{4} \sum_{\mu,\nu=0}^{3} \sigma_{\mu\nu} \hat{F}_{\mu\nu}(x) = c_{\mathrm{sw}}^{SU(3)} \begin{pmatrix} A_+(x) & 0 \\ 0 & A_-(x) \end{pmatrix}, \tag{3.4}$$

where $A_\pm(x)$ are 6×6 Hermitian matrices that can be stored internally as 36 real numbers u_i, $i \in \{0, \dots, 35\}$ – which is exactly how it is done in openQxD. The first 6 entries $u_0, \dots, u_5$ are the diagonal real numbers, whereas the remaining 15 pairs $\{u_j, u_{j+1}\}$ for $j = 6, 8, \dots, 34$ denote the real and imaginary parts of the strictly upper triangular part in row-major order:

$$\begin{pmatrix} u_0 & u_6 + iu_7 & u_8 + iu_9 & u_{10} + iu_{11} & u_{12} + iu_{13} & u_{14} + iu_{15} \\ \cdot & u_1 & u_{16} + iu_{17} & u_{18} + iu_{19} & u_{20} + iu_{21} & u_{22} + iu_{23} \\ \cdot & \cdot & u_2 & u_{24} + iu_{25} & u_{26} + iu_{27} & u_{28} + iu_{29} \\ \cdot & \cdot & \cdot & u_3 & u_{30} + iu_{31} & u_{32} + iu_{33} \\ \cdot & \cdot & \cdot & \cdot & u_4 & u_{34} + iu_{35} \\ \cdot & \cdot & \cdot & \cdot & \cdot & u_5 \end{pmatrix} . \qquad (3.5)$$

The clover field has indices $(x, \pm, i)$, where x is the lattice index, $\pm \in \{+, -\}$ denotes the chirality (i. e. the upper $(+)$ or lower $(-)$ 6×6-block of the 12×12 clover matrix).

TL;DR: clover field indices

In addition to the data structs discussed in sections 3.3 to 3.5 taken over from openQCD, the openQxD package – which provides an extension to incorporate electromagnetic gauge fields – implements new functionality such as C* boundary conditions and additional degrees of freedom. These are described in more detail in the following two sections.

TL;DR: new stuff in openqxd compared to openqcd

3.6 C* BOUNDARY CONDITIONS

C* boundary conditions on the lattice can be implemented by means of doubling the lattice in a certain spatial dimension (taken arbitrarily as x-direction in the following). The doubled lattice will be called the extended lattice Λ_{ext} and the two copies physical lattice Λ_{phys} and mirror lattice Λ_{mirror}. This is called the orbifold construction, see fig. 3.2.

TL;DR: Cstar BCs = doubling lattice

In openQxD the fermion field is defined on the extended lattice where the space-time lattice index determines whether we are on the physical or on the mirror lattice. The gauge fields are copied to the mirror lattice sites and charge conjugated. This implementation has advantages and disadvantages: it is simple to extend code that already operates on periodic fields but the gauge fields are redundant in memory making an application of the Dirac operator roughly twice as expensive as it could be.

TL;DR: spacetime index define physical/mirror

3.7 IMPLEMENTATION OF QCD+QED

The code stores the $SU(3)$ gauge fields internally as an array of structs listing 3.4 which are 9 complex numbers. The $U(1)$ field is stored in

TL;DR: gauge groups and how they're stored

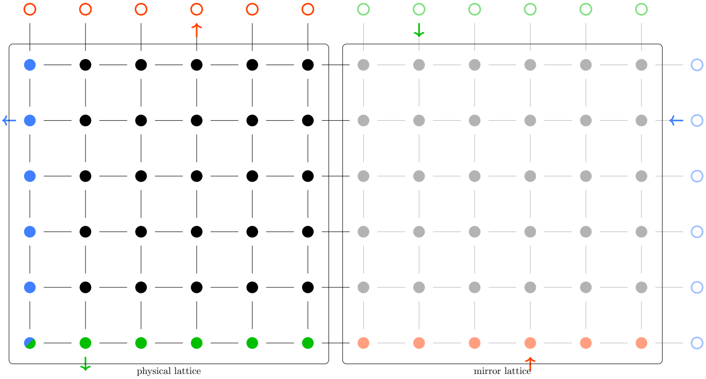


FIGURE 3.2: 2D example of a 6×6 lattice with C* boundary conditions on both directions. We have the (doubled) x-direction (horizontal) and a direction with C* boundaries (vertical). Left is the physical, right the mirror lattice. Unfilled lattice points are exterior boundary points, whereas filled points are interior (boundary) points. The points of the same color are identified with each other. Notice that in x-direction we have regular periodic boundary conditions, since C* boundaries are periodic over twice the lattice extent. The red, green and blue arrows indicate the path taken when leaving the physical lattice and entering the mirror lattice.

terms of its Lie-algebra $\mathfrak{u}(1)$, which are real phases $A_\mu(x)$, as simple array of double precision floats. This is done such that, when the charge $\hat{q}$ changes in eq. (1.69) the $U(1)$ value can be regenerated without taking logarithms and re-exponentiating. Additionally, the compound $U(3)$-valued field $e^{i\hat{q}A_\mu(x)}U_\mu(x)$ is held in memory as an array of structs of type listing 3.4. This is not a problem, since the `su3_dble` struct parametrizes all 9 complex numbers of an arbitrary 3×3 complex matrix. All functions using the gauge field, such as the Dirac operator, now operate on the compound field instead of the $SU(3)$-valued one. All fields are defined on the extended lattice and when copying the field values to the mirror lattice, the boundaries eq. (1.61) are imposed on the gauge and clover fields.

TL;DR: Why are they separate in memory?

The $SU(3)$ and $U(1)$ fields are kept separately in memory, because their clover terms differ both in coefficients and field strength tensors, see eq. (1.69). Fortunately, the QED clover term does neither change block structure nor Hermiticity of the blocks. Therefore, openQxD simply adds the QED term on top of the QCD term, eq. (1.68), and obtains the sum of the two clover terms, still stored in memory as discussed in section 3.5. Without further changes, functions depending on the clover field now naturally operate on the sum of the two clover fields. This will become important later, because QUDA is not capable of generating the $U(1)$ clover term.

3.8 SUMMARY

The simulation code used by the RC* collaboration is openQxD. It is capable of C* boundary conditions implemented through the orbifold construction, required for dynamical QCD+QED simulations. The code runs efficiently on CPU cluster parallelized purely via MPI. This is reflected via data structures as arrays of structs and linear algebra kernels being optimized for CPUs, little abstraction was implemented. We wish to enable simulations on modern GPU-based clusters by offloading solves to the Dirac equation to GPUs.

4

QUDA

This section/chapter was proofread 1 times.

In this chapter, we outline some of the main features, design choices and peculiarities of QUDA, the library we offload Dirac equation solves to. We go through an example of a simple kernel and how it is being called internally (section 4.1), followed by how lattice fields are abstracted (section 4.2). QUDA comes with mixed precision solvers allowing fields to change their floating-point precision (section 4.3) and it supports compressed gauge field formats exploiting the gauge group symmetry (section 4.4). We finish with a list of important functions exposed via QUDA's header file (section 4.5) and a brief summary (section 4.6).

TL;DR: what is quda

QUDA [144] is a library written in CUDA C++ that contains a suite of efficient kernels and solvers to implement lattice QCD simulations. It can be used as a plug and play substitute for certain compute intensive tasks. QUDA is highly optimized to run on multiple GPUs and has backends for CUDA, HIP, SYCL and multi-threading, where it performs autotuning to determine empirically the best kernel launch parameters for every kernel separately. It thus runs efficiently on a variety of different hardware vendors.

TL;DR: qudas feature set

It has a very powerful and customizable solver suite, including a state-of-the-art multigrid implementation and comes with support for many different lattice discretizations, such as Wilson, Wilson-Clover, Twisted mass, Twisted-Clover, staggered, improved staggered, Domain-Wall and Möbius. In the meantime there is a large community of lattice practitioners actively working on and maintaining the software project.

TL;DR: current release

The current release version 1.1.0 complies with the C++14 and 17 standards and is released under an NVIDIA license similar to the MIT license.

4.1 KERNEL INVOCATION

TL;DR: overloading, templates, functors, backend abstraction

The codebase heavily relies on C++ features such as operator overloading and templates. Device kernels, launched on backends such as CUDA, HIP, SYCL, or threading models, are typically represented as functors – functions that carry internal state, implemented in C++ as classes overloading the function-call operator, `operator()`. This approach enables seamless abstraction of implementation details from the underlying backends.

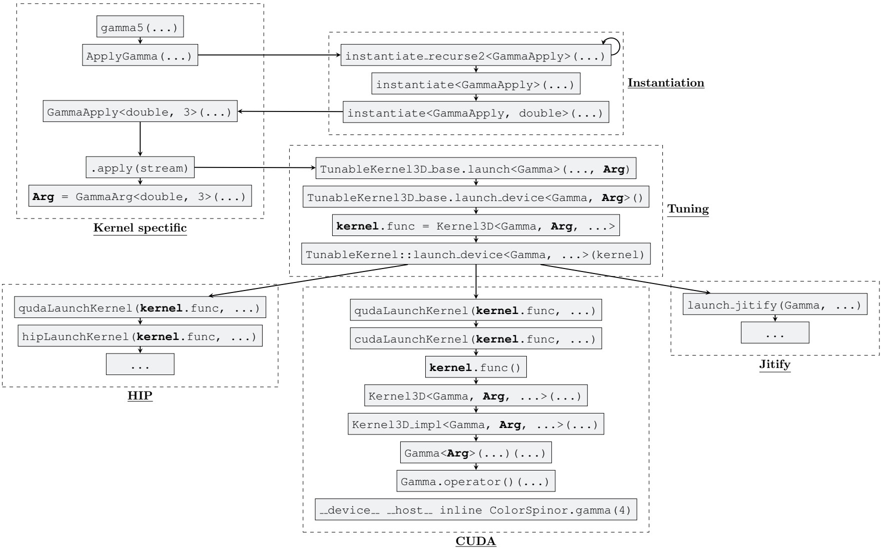

gamma5(...)
ApplyGamma(...)
GammaApply<double, 3>(...)
.apply(stream)
Arg = GammaArg<double, 3>(...)
Kernel specific
instantiate_recurse2<GammaApply>(...)
instantiate<GammaApply>(...)
instantiate<GammaApply, double>(...)
Instantiation
TunableKernel3D_base.launch<Gamma>(..., Arg)
TunableKernel3D_base.launch_device<Gamma, Arg>()
kernel.func = Kernel3D<Gamma, Arg, ...>
TunableKernel::launch_device<Gamma, ...>(kernel)
Tuning
qudaLaunchKernel(kernel.func, ...)
hipLaunchKernel(kernel.func, ...)
...
HIP
qudaLaunchKernel(kernel.func, ...)
cudaLaunchKernel(kernel.func, ...)
kernel.func()
Kernel3D<Gamma, Arg, ...>(...)
Kernel3D_impl<Gamma, Arg, ...>(...)
Gamma<Arg>(...)(...)
Gamma.operator()(...)
__device__ __host__ inline ColorSpinor.gamma(4)
CUDA
launch_jitify(Gamma, ...)
...
Jitify

FIGURE 4.1: Call graph of the `gamma5` function in QUDA as an example of how kernels are abstracted from the backends. This function acts on fields already transferred to QUDA, thus there is no gamma basis transformation.

TL;DR: example of gamma5

As an introductory example, we provide the call stack of the function `gamma5` in fig. 4.1. The function applies the γ^5 matrix to an input spinor and writes the result to an output spinor. The function itself is just a wrapper around `ApplyGamma` which itself calls the instantiation of the functor `GammaApply`. `GammaApply` calls its `apply` method in its constructor triggering tuning for the `Gamma` functor. Finally in the tuning kernels, the `launch_device` call dispatches to the correct backend and instructs the runtime API to launch the kernel, for instance via `cudaLaunchKernel` in case of CUDA. This executes the previously packed device function which ultimately instantiates the `Gamma` functor and then calls it via its function-call operator.

4.2 FIELD REPRESENTATIONS

TL;DR: field are object instances, inheritance

Naturally, fields on the lattice are represented as objects, and pointers to the fields' values as well as various meta information about the field are accessible via their properties and accessors. A spinor field is represented by an instance of `ColorSpinorField`, a gauge field by `GaugeField` and a

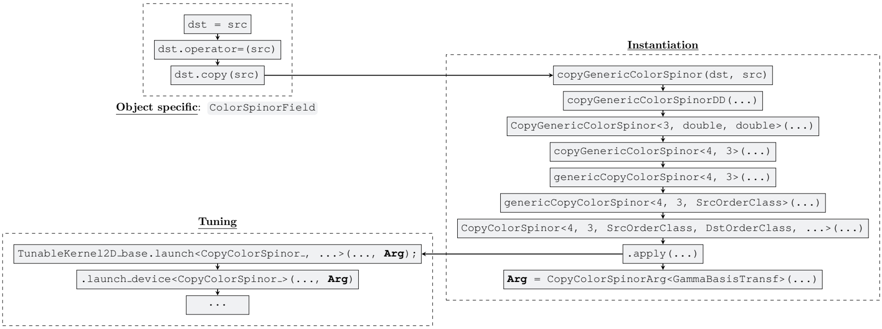

dst = src
dst.operator=(src)
dst.copy(src)
Object specific: ColorSpinorField
Instantiation
copyGenericColorSpinor(dst, src)
copyGenericColorSpinorDD(...)
CopyGenericColorSpinor<3, double, double>(...)
copyGenericColorSpinor<4, 3>(...)
genericCopyColorSpinor<4, 3>(...)
genericCopyColorSpinor<4, 3, SrcOrderClass>(...)
CopyColorSpinor<4, 3, SrcOrderClass, DstOrderClass, ...>(...)
.apply(...)
Arg = CopyColorSpinorArg<GammaBasisTransf>(...)
Tuning
TunableKernel2D_base.launch<CopyColorSpinor_, ...>(..., Arg);
.launch_device<CopyColorSpinor_>(..., Arg)
...

FIGURE 4.2: Copy process triggered by equating two fields of type `ColorSpinorField`.

clover field as `CloverField`, all of which inherit from the `LatticeField` class. Different internal degrees of freedom are realized in the inheriting classes. The actual numbers are accessible via pointers whose accessors are called `data()`.

4.2.1 *Field transfer*

TL;DR: copy/move overloading

Copying and moving fields is efficiently handled by overloading the copy-assignment and move-assignment operators, `operator=`, of classes representing fields. Fields carry meta information about whether their data lies on the CPU or GPU. Field transfers between CPU and GPU, as well as reordering of its data, is fully abstracted away and pushed into the function bodies of the copy and move overload implementations.

TL;DR: spinor equating example

We illustrate this in fig. 4.2. For example, when equating two fields of type `ColorSpinorField`, the copy-assignment operator is triggered which initiates the copy process. After some instantiation of template parameters such as the floating-point precision of the source `src` and destination `dst` fields (both `double` in this example), spin and color degrees of freedom (4 and 3, respectively) and the order classes for the source and destination (more on reordering in section 5.1.2), the copy functor `CopyColorSpinor_` is launched through the tuning procedure with the correct gamma basis transformation as argument. The remainder of the diagram after the last point (indicated by `...`) is very similar to the call graph of `gamma5`, fig. 4.1.

TL;DR: quda translates gamma bases

Thus, input fields might have different γ-basis conventions than output fields. QUDA transforms every field coming from an external application on the CPU into its preferred internal γ-basis representation and transforms back when storing the field to the applications memory region. This is part of the data reordering procedure.

4.3 FLOATING-POINT PRECISION

TL;DR: template make it easy for precisions

The usage of C++ templates enables simple replacements for the precision of fields on the lattice in terms of floating-point numbers as illustrated in figs. 4.1 and 4.2. QUDA allows kernels and fields to operate with the usual IEEE 754 floats like double, single, half and quarter. Assigning fields of different precision to each other triggers conversion. This makes implementations of mixed precision solvers straightforward and readable.

4.4 COMPRESSION FORMATS

TL;DR: internal dof with symmetries

For fields whose internal degrees of freedom carry additional symmetry, such as the SU(3)-valued gauge field, QUDA provides compression formats for efficient representation in memory. This makes sense in memory-bound scenarios, because in kernels using such a field the produced memory traffic is decreased and the arithmetic intensity increased when the field is represented in a compressed format. For certain gauge groups QUDA allows the gauge field to be stored in the following formats

- `QUDA_RECONSTRUCT_NO`: This reconstruction type allows to use any arbitrary gauge group by storing all 9 complex (18 real) numbers per gauge link explicitly.

- `QUDA_RECONSTRUCT_8`: Reconstruct each gauge link from 8 real numbers. This is inspired by the Lie-algebra of SU(3) which has 8 real degrees of freedom. It is the smallest representation in terms of bytes.

- `QUDA_RECONSTRUCT_9`: If the SU(3) matrix additionally carries a U(1) phase, the Lie-algebra will have 9 real degrees of freedom.

- `QUDA_RECONSTRUCT_12`: A reconstruction type using 12 real numbers that is meant to be used for SU(3)-valued gauge links and is more stable in reconstructing the 3×3 complex matrix.

- `QUDA_RECONSTRUCT_13`: Similar to the previous entry but with an additional U(1) phase.

- `QUDA_RECONSTRUCT_10`: A 10-number parameterization used for storing the momentum field.

Comparing `QUDA_RECONSTRUCT_NO` with `QUDA_RECONSTRUCT_8`, we see that the former stores 18 real numbers, whereas the latter only needs 8. Memory boundedness of a kernel involving such a field implies roughly a speedup of $18/8 > 2$ when using the compressed format over the uncompressed one.

4.5 IMPORTANT INTERFACE FUNCTIONS

In the following, we will list interface functions exposed by QUDA's interface via `quda.h` that are relevant in the remainder of this document, especially for the discussion of the interface, chapter 5. More detailed descriptions can be found in the doxygen string of each function. These functions may trigger field transfers or reordering:

TL;DR: some interface functions in quda.h

- `initQuda()`: initialize QUDA.
- `endQuda()`: finalize QUDA.
- `loadGaugeQuda()`: load the gauge field from the CPU.
- `saveGaugeQuda()`: save the gauge field to the CPU.
- `freeGaugeQuda()`: free the resident gauge field in QUDA.
- `loadCloverQuda()`: load the clover field from the CPU.
- `freeCloverQuda()`: free the resident clover field in QUDA.
- `newMultigridQuda()`: create a new multigrid instance.
- `updateMultigridQuda()`: update an existing multigrid instance.
- `destroyMultigridQuda()`: destroy an existing multigrid instance.
- `invertQuda()`: call the inverter on CPU fields.
- `eigensolveQuda()`: call the eigensolver on CPU fields.
- `MatQuda()`: call the Dirac operator on CPU fields.
- `plaqQuda()`: calculate the plaquette over the resident $SU(3)$ gauge field.

4.6 SUMMARY

QUDA is a lattice QCD library running efficiently on GPUs of various vendors via CUDA, HIP and SYCL backend abstraction. It comes with a state-of-the-art mixed-precision multigrid solver optimized for many lattice QCD Dirac operators such as the Wilson-Clover. Abstraction is practiced in terms of operator overloading, class inheritance and heavy templating. $C^\star$ boundary conditions and the QCD+QED Wilson-Clover Dirac operator are missing features which we implemented.

5

INTERFACING OPENQXD AND QUDA

> *When I wrote this, only God and I understood what I was doing. Now, God only knows.*
> — Karl Weierstrass

This chapter documents the code required for interfacing openQxD with QUDA and describes design choices made. A working interface requires substantial code additions and changes in both applications. Therefore, this chapter is divided into two major sections: developments made in QUDA (section 5.1) and developments made in openQxD (section 5.2). The first section describes how to access interface functions, translation of lattice coordinates, field transfer, how C* boundaries and QCD+QED are introduced to QUDA and the various ways to interface the solver. The second section documents the code that entered openQxD, how to setup solvers and how the dual process grid was implemented. The chapter ends with a summary (section 5.3).

This section/chapter was proofread 1 times.

TL;DR: interface intro

5.1 CHANGES AND ADDITIONS TO QUDA

We will start with the QUDA side, since these developments are crucial preliminaries to understand the changes and additions on the openQxD side. QUDA exposes its functionality in terms of a header file `quda.h` declaring API functions which we can directly call on the side of openQxD by including the header. In order to use these functions, one has to build QUDA as a library, include the main header file and link against it.

TL;DR: start with quda

As a teasing example for such an API function, we consider the solver wrapper whose call signature looks as follows:

TL;DR: a teaser

```
void invertQuda(void *h_x, void *h_b, QudaInvertParam *param);
```

Here, `h_x` and `h_b` are void pointers pointing to the host spinor fields in openQxD order, i. e. base pointers to arrays of `spinor_dble` structs, as described in **??**. The `QudaInvertParam` struct parametrizes the solver (see `quda.h` and ref. [144] for details).

The interface for openQxD provides a second header file

TL;DR: 2nd header file for the interface `quda_openqcd_interface.h` which contains convenient openQxD-specific interface functions. All functions provided in this header file have a naming pattern as

```
openQCD_qudaFunctionName(...);
```

where `FunctionName` is a describing name written in CamelCase.

5.1.1 *Initialization*

TL;DR: Definition of init

In order to call the API functions, QUDA has to be initialized with the function

```
void initQuda(int device);
```

where `device` is the CUDA device number to use or -1 if QUDA should decide itself how to allocate devices to processes. This function is called by the initialization function provided by the interface

```
void openQCD_qudaInit(
    openQCD_QudaInitArgs_t init,
    openQCD_QudaLayout_t layout,
    char *infile
);
```

where

- `init` is a struct holding information coming from openQxD such as the general verbosity setting, the log file descriptor, the local lattice volume and the number of boundary points.
- `layout` is a struct holding layout information such as the local and global lattice extents, the process topology, the CUDA device number to use, the number of spatial $C^\star$ directions and a rank number to process grid coordinate mapping. Additionally, this struct holds function pointers to functions in openQxD that need to be called in the interface code such as querying the boundary conditions, the gauge group or Dirac operator parameters.
- `infile` is the file path to the input file used by the utility interfacing QUDA.

TL;DR: What does the function do

The function initializes the communication grid topology in QUDA by calling `initCommsGridQuda()` with the information given in `layout`. This is the process grid and a function pointer of type

```
typedef int (*QudaCommsMap)(const int *coords, void *fdata);
```

which takes 4D Euclidean coordinates of the process in QUDA and translates them into MPI rank numbers. The argument `fdata` may hold additional information for the function to determine the ranks. This function is the QUDA-equivalent of the function `ipr_global()` in openQxD. Moreover, the initialization function sets the prefix prepended to all log messages coming from QUDA and reads in the general verbosity from the input file and sets it. Finally, `initQuda` is called with the device given in `layout`. From the perspective of openQxD, one usually does not call this function explicitly but rather `quda_init()`, see section 5.2.2.

5.1.2 *Field reordering*

TL;DR: Re-ordering intro

A crucial part of the interface is passing fields in openQxD living on the lattice from and to QUDA. The memory layout of lattice fields is a fundamental difference between the two applications. We implement an interface that reorders the fields to agree with the different conventions.

TL;DR: How it's meant to be done

Since openQxD is not the first lattice application interfacing QUDA, there is already an intended way to do so, namely to implement specific field reordering classes [144]:

- `OpenQCDOrder` in `include/gauge_field_order.h` for the gauge field.
- `OpenQCDDiracOrder` in `include/color_spinor_field_order.h` for the spinor field.
- `OpenQCDOrder` in `include/clover_field_order.h` for the clover field.

Figure 5.1 illustrates the interplay between parameter struct values and the instantiation of reordering classes in QUDA. The aforementioned members of the two parameter structs `ColorSpinorParam` and `QudaInvertParam` decide on the field reordering classes QUDA instantiates. The enum constants were added to allow redirecting to the openQxD reorder classes or γ-matrix transformations specified in the last column of fig. 5.1. All reorder classes have to implement a `load` and a `save` method.

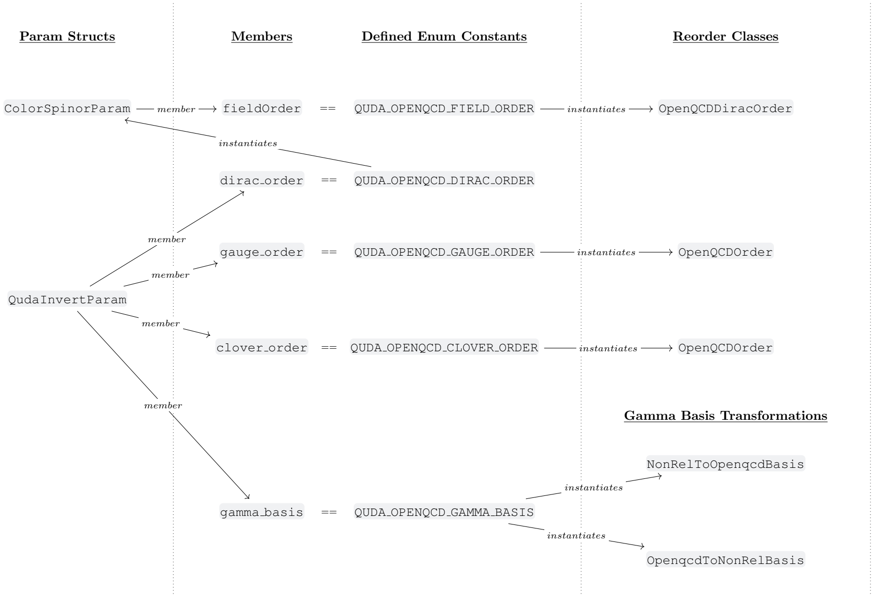

Param Structs
Members
Defined Enum Constants
Reorder Classes
ColorSpinorParam
member
fieldOrder
==
QUDA_OPENQCD_FIELD_ORDER
instantiates
OpenQCDDiracOrder
instantiates
dirac_order
==
QUDA_OPENQCD_DIRAC_ORDER
member
gauge_order
==
QUDA_OPENQCD_GAUGE_ORDER
instantiates
OpenQCDOrder
member
QudaInvertParam
member
clover_order
==
QUDA_OPENQCD_CLOVER_ORDER
instantiates
OpenQCDOrder
member
Gamma Basis Transformations
NonRelToOpenqcdBasis
instantiates
gamma_basis
==
QUDA_OPENQCD_GAMMA_BASIS
instantiates
OpenqcdToNonRelBasis

Figure 5.1: Names of parameter structs, their members, enum constants defined for interfacing openQxD together with basis transformation and reorder classes in QUDA that were implemented to make the different memory layouts compatible.

The `load` *method*

TL;DR: load method in reorder classes

As the name suggests, it is called in parallel with a (local) checkerboard index `x_cb`, a `parity` and – in case of the gauge field –, a direction `dir`.

```
// include/color_spinor_field_order.h:OpenQCDDiracOrder
__device__ __host__ inline void load(
    complex v[length / 2],
    int x_cb,
    int parity = 0
) const;

// include/gauge_field_order.h:OpenQCDOrder
__device__ __host__ inline void load(
    complex v[length / 2],
    int x_cb,
    int dir,
    int parity,
    Float = 1.0
) const;
```

```
// include/clover_field_order.h:OpenQCDOrder
__device__ __host__ inline void load(
  RegType v[length],
  int x_cb,
  int parity
) const;
```

Listing 5.1: The load methods from the reorder classes.

The array `v` is the output where in the body of the method we need to copy the field values from openQxD. The instantiated object holds a property called `field`, `gauge` or `clover` for the base pointer to the openQxD array for the spinor, gauge or clover field, respectively. Thus, we have to calculate the correct offset from the field base pointer. QUDA provides an (inlined) function `getCoords()` to translate the checkerboard and parity index into a (local) Euclidean 4-vector $x = (\vec{x}, t)$ in the (xyzt)-convention with time coordinate last.

The `save` method

The `save` method works similarly to the `load` method, only now the values come from the `v` array and are copied to the correct offset from the field base pointer.

TL;DR: save method in reorder classes

```
// include/color_spinor_field_order.h:OpenQCDDiracOrder
__device__ __host__ inline void save(
    const complex v[length / 2],
    int x_cb,
    int parity = 0
) const;

// include/gauge_field_order.h:OpenQCDOrder
__device__ __host__ inline void save(
    const complex v[length / 2],
    int x_cb,
    int dir,
    int parity
) const;

// include/clover_field_order.h:OpenQCDOrder
__device__ __host__ inline void save(
    const RegType v[length],
    int x_cb,
    int parity
) const;
```

Listing 5.2: The save methods from the reorder classes.

TL;DR: All fields share spacetime index

All of the fields have the Euclidean space-time index in common, so we begin by discussing the order of the lattice sites in the following section.

Space-time coordinates

TL;DR: Arrays no good on GPUs

As discussed in section 3.2, openQxD implements the space-time index by means of the mapping array `ipt` that returns the actual base pointer offset when fed with the lexicographical local lattice index. Such mapping arrays are not recommended on GPUs due to shared memory contentions and memory usage. Instead, it is advisable to write a pure function $f\colon x \mapsto \hat{x}$ that implements the indexing array of openQxD, eq. (3.2), by calculating the index on the fly such that the compiler can properly inline the calculation. This has been implemented as `openqcd::ipt()` which is called in the load and save functions of all reorder classes.

TL;DR: xyzt conventions + more indices

After translating QUDA's checkerboard/parity index into a (xyzt) 4-vector using `getCoords`, we can permute the coordinates into the (txyz)-convention, query the mapping function f to obtain the offset from the base pointer in openQxD for the desired lattice point and finally copy the data to or from the location pointed to by the variable `v`. The data can have additional internal indices that we describe for each field type separately in the following.

Spinor field

TL;DR: Spinor indices and their order

As mentioned previously, spinor fields have indices (x, α, a), where we described how to transform the space-time index x in the previous section. Both QUDA and openQxD use the same order for spinor index α and color index a. Thus, at each space-time index we can copy $12 = 4 \cdot 3$ consecutive complex numbers (i. e. `24*sizeof(Float)` bytes), where `Float` is a template parameter for real numbers (`double`, `float`, `half`) to or from the output array `v`. For illustration purposes, the actual implementations are shown in the following.

```
/**
 * @brief      Gets the offset in Floats from the openQCD base pointer to
 *             the spinor field.
 *
 * @param[in]  x_cb    Checkerboard index coming from quda
 * @param[in]  parity  The parity coming from quda
 *
 * @return     The offset.
 */
__device__ __host__ inline int getSpinorOffset(int x_cb, int parity) const
{
  int x_quda[4], x_openqcd[4];
  getCoords(x_quda, x_cb, dim, parity);       // x_quda contains xyzt
  openqcd::rotate_coords(x_quda, x_openqcd);  // x_openqcd contains txyz
  return openqcd::ipt(x_openqcd, L) * length;
}

__device__ __host__ inline void load(complex v[length / 2], int x_cb,
                                     int parity = 0) const
{
  auto in = &field[getSpinorOffset(x_cb, parity)];
  block_load<complex, length / 2>(v, reinterpret_cast<const complex *>(in));
}

__device__ __host__ inline void save(const complex v[length / 2], int x_cb,
                                     int parity = 0) const
{
  auto out = &field[getSpinorOffset(x_cb, parity)];
  block_store<complex, length / 2>(reinterpret_cast<complex *>(out), v);
}
```

Clover field

Similarly to the spinor field, for each space-time index, we can copy $72 = 2 * 36$ real numbers (i. e. `72*sizeof(Float)` bytes) to the output array `v`. We did not implement a save function for clover fields, since we never need to transfer them from device to host.

TL;DR: Clover indices + no save function

OpenQxD stores the clover field as two arrays u of length 36 that represent two Hermitian 6×6 matrices (one for each chirality $\pm$), compare eq. (3.5), whereas QUDA stores these 36 numbers in a slightly different format (see `include/clover_field_order.h` [144]):

TL;DR: Clover index order

$$\begin{pmatrix} u_0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ u_6 + iu_7 & u_1 & \cdot & \cdot & \cdot & \cdot \\ u_8 + iu_9 & u_{16} + iu_{17} & u_2 & \cdot & \cdot & \cdot \\ u_{10} + iu_{11} & u_{18} + iu_{19} & u_{24} + iu_{25} & u_3 & \cdot & \cdot \\ u_{12} + iu_{13} & u_{20} + iu_{21} & u_{26} + iu_{27} & u_{30} + iu_{31} & u_4 & \cdot \\ u_{14} + iu_{15} & u_{22} + iu_{23} & u_{28} + iu_{29} & u_{32} + iu_{33} & u_{34} + iu_{35} & u_5 \end{pmatrix} . \quad (5.1)$$

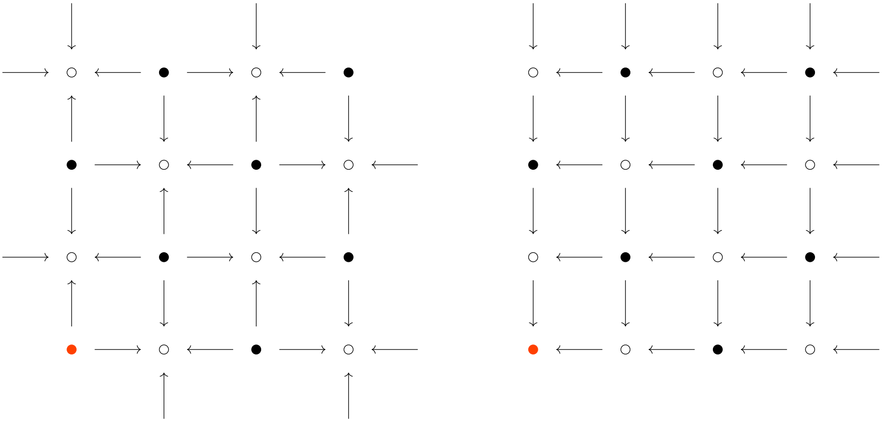


FIGURE 5.2: 2D example (4 × 4 local lattice) of how and which gauge fields are stored in memory in openQxD (left) and QUDA (right). Filled lattice points have even, unfilled odd parity. The red filled lattice point denotes the origin. Arrows represent gauge fields and the arrow head points to the lattice point where we store the field.

TL;DR: row-major vs col-major

We see that the diagonal elements are the same in both applications, but QUDA stores the strictly lower triangular part in column-major order. So we can transfer the clover field from openQxD to QUDA by specifying how these 36 numbers transform. In particular, the QED clover field does not affect the block structure of the clover term and we can transfer the clover term in QCD+QED (see section 3.7 more for details). On the other hand, a pure QCD clover field can be calculated natively within QUDA and no additional transfer of fields is required in that case.

Gauge field

TL;DR: gauge indices + 8x odd vs. 4x all

QUDA associates 4 gauge fields for each space-time point (one for each positive direction $\mu = 0, 1, 2, 3$), whereas openQxD stores 8 (forward and backward) directions of gauge fields for only the odd-parity points (see `main/README.global` and ref. [111] for more information). When looking at local lattices in a multi-rank scenario, this implies that openQxD locally stores gauge fields on the boundaries only for odd-parity points and not for even-parity points (see fig. 5.2). These even-parity boundary fields are stored in a buffer space, but they have to be communicated from neighboring lattices first.

This requires us to transfer the missing gauge fields from one rank to the other before entering any QUDA interface function. The gauge field is loaded to and saved from QUDA by the following two API calls

TL;DR: Boundary point need to be populated

```
void loadGaugeQuda(void *h_gauge, QudaGaugeParam *param);
void saveGaugeQuda(void *h_gauge, QudaGaugeParam *param);
```

where the void pointer `h_gauge` points to the gauge fields in the CPU memory (the missing fields have to be available already) and `param` is a struct holding information about the gauge field and how it should be interpreted by the various Dirac operators supported by QUDA.

We implement an ordering class called `OpenQCDOrder` with corresponding load and save methods. The only difference to the spinor and clover field order classes is that the methods are called with an additional direction variable `dir`, see listings 5.1 and 5.2. For a fixed space-time x and direction μ, the remaining two color indices (a, b) of the gauge field are row-major in both applications, thus we can copy 3×3 consecutive complex numbers to or from `v`. The variable `dir` runs from 0 to 3 and denotes the positive link direction in QUDA convention (see fig. 5.2 right).

TL;DR: direction in load/save

5.1.3 $C^\star$ boundary conditions

QUDA does not support $C^\star$ boundary conditions, so we implement them in the QUDA library. Unlike periodic boundary conditions, we identify the field shifted by one lattice length not with itself but with its charge conjugation. Thus, $C^\star$ boundary conditions are implemented by translation from physical to mirror lattice and choosing appropriate boundary conditions (see fig. 3.2).

TL;DR: Cstar intro

We choose to implement $C^\star$ boundary conditions in the same way as they are implemented in openQxD – by doubling of the lattice in x-direction and by imposing shifted boundaries as illustrated in fig. 3.2. Note that openQxD requires at least two ranks in each direction of the process grid, where the lattice has $C^\star$ boundary conditions. This ensures that a rank only stores points in either the physical or mirror lattice.

TL;DR: Also orbi construction

When QUDA initializes its communication grid topology, we specify the neighbors of each rank in all directions. This also respects the specific process placement when using $C^\star$ boundaries, see section 5.1.1. For QUDA to setup communication among neighboring ranks, the function `comm_rank_displaced()` calculates the neighboring rank number given one of the (positive or negative) 8 directions (see its call graph fig. 5.3.). The

TL;DR: Trick qudas process grid

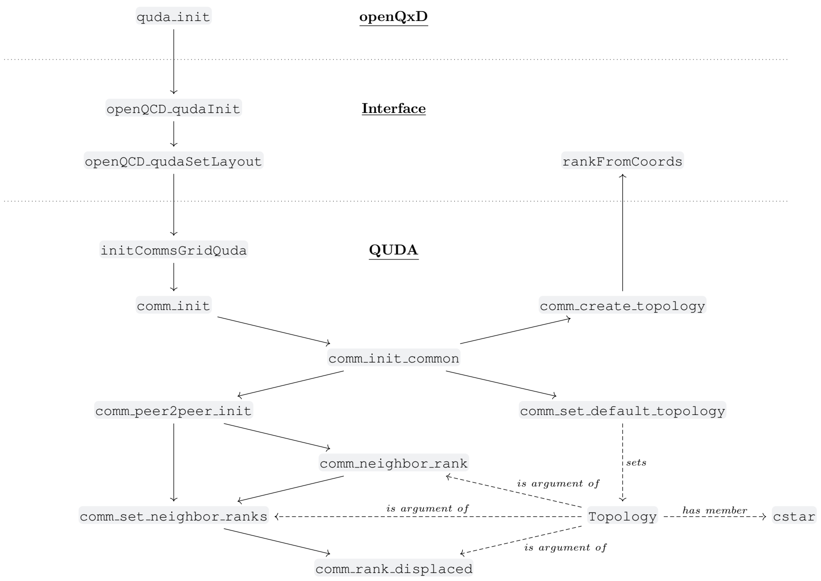

quda_init
openQxD
openQCD_qudaInit
Interface
openQCD_qudaSetLayout
rankFromCoords
initCommsGridQuda
QUDA
comm_init
comm_create_topology
comm_init_common
comm_peer2peer_init
comm_set_default_topology
comm_neighbor_rank
sets
is argument of
comm_set_neighbor_ranks
is argument of
Topology
has member
cstar
is argument of
comm_rank_displaced

FIGURE 5.3: Call graph for the function `comm_rank_displaced`. Starting from the initialization function `quda_init` in openQxD, see section 5.2.2 which calls the interface initialization functions, see section 5.1.1, calling the communication grid setup which itself calls the constructor of the `Communicator` class which finally sets the grid topology.

function reads the `cstar` property from the `Topology` struct that can take values 0, 1, 2 or 3 denoting the number of spatial directions with $C^\star$ boundaries, where `cstar=0` corresponds to its default value indicating periodic boundaries. To respect the rank placement of openQxD, we change this function to achieve shifted boundary conditions as in fig. 3.2: consider the case of two ranks, one of which contains all physical points, the other the mirror points. The neighbor of the physical lattice in "upward" direction is the mirror rank (and vice versa) which is different from periodic boundary conditions.

5.1.4 *Lattice QCD+QED*

TL;DR: Gauge field always U3 in 8/9 compressed fmt

The lattice QCD+QED Dirac operator acts on the U(3)-valued gauge fields which are the combined SU(3) links with a U(1) phase. QUDA allows arbitrary gauge links and has compressed formats for U(3) valued links. Depending on the gauge group, the code decides which compressed format should be instantiated by QUDA. This is `QUDA_RECONSTRUCT_8` in case of SU(3) and `QUDA_RECONSTRUCT_9` for U(3), see section 4.4.

QUDA does not support simulating Wilson-Clover QCD+QED natively, especially the QED Clover term is missing. This feature involves modifying the clover term and by this the Wilson-Dirac operator. The QCD+QED Clover term consists of both the QCD SU(3) and the QED U(1) term. Both can be calculated in openQxD. The resulting term is still diagonal in spacetime and chirality and is Hermitian in color and spin. Therefore, it has the same representation in memory as the SU(3) clover term alone and we can upload this new clover field to QUDA using the clover field reordering class as discussed in section 5.1.2. Thus, we decide on the gauge group whether the clover term is generated natively in QUDA or transferred from openQxD.

TL;DR: Clover term transfer or generate

Transferring the QCD+QED clover term and the U(3) links as well as the changes in the process grid topology enables QUDA to successfully handle the QCD+QED Wilson-Clover Dirac operator eq. (1.69). This concludes our implementation of QCD+QED in QUDA.

TL;DR: QCD+QED WC DOP achieved

5.1.5 *Tracking of changing parameters and fields*

Dynamic properties which might change in a run over time such as (Dirac operator parameters and pointers to the gauge and clover fields[1]) are treated by handing over function pointers to functions, which return structs holding currently active values for these parameters at the time of calling. The handover of these function pointers is done by `quda_init()`, see section 5.2.2. We distinguish between tracked parameters, tracked fields and tracked multigrid instances.

TL;DR: Parameter tracking intro

Tracked parameters

An example of a tracked parameter is the quark mass m_0 (or equivalently, its inverse mass κ) that might change in a run when processing multiple flavors. OpenQxD manages such parameters as global static variables in a translation unit which offers non-pure functions to set, get and print the values. For the Dirac operator parameters these functions are

TL;DR: How are such params dealt with in openqxd

---

[1] The pointers themselves might not change but the data they point to.

```c
extern dirac_parms_t set_dirac_parms1(dirac_parms_t *par);
extern dirac_parms_t dirac_parms(void);
extern void print_dirac_parms(void);
```

where the returned Dirac parameter struct

```c
typedef struct
{
   int qhat;
   double m0,su3csw,u1csw,cF[2],theta[3];
} dirac_parms_t;
```

is static in the translation unit and holds the current values.

TL;DR: How to we deal with them in interface

The `quda_init()` function will hand over a function pointer to the get function `dirac_params()` to the interface. The interface code will synchronize these parameters into its own global state. Offloaded functionality such as the Dirac operator or the inverter that depend on such parameters will compare the current values with the returned ones and notice a change. Some parameters, such as the flavor charge `qhat` for instance, might even trigger a re-transfer of the gauge field. The tracked parameters are (compare eq. (1.69))

- m_0: the quark mass (or equivalently the inverse quark mass κ),
- $c_{\text{SW}}^{SU(3)}$: the $SU(3)$ SW-coefficient for the clover field,
- $c_{\text{SW}}^{U(1)}$: the $U(1)$ SW-coefficient for the clover field,
- $\hat{q}$: the integer valued quark charge.

Tracked fields

TL;DR: How are changing field dealt with in openqxd + interface, gauge field

Fields that have to be kept in sync are the gauge and the clover field. In openQxD, tracking a changing gauge field is implemented by means of a revision counter incrementing every time the residing gauge field has changed. A query function for the revision number of the gauge field is handed over in `quda_init()`. The interface keeps the current revision in its state, compares revisions using the provided query function and updates them every time the gauge field is transferred.

TL;DR: clover field

The clover field depends on the revision of the gauge field because it is generated from it as well as on all the tracked parameters above. The interface holds the revision of the clover field separately and requires an update if at least one of the following statements is true:

- The gauge field requires an update.
- At least one of the tracked parameters has changed.
- The internal revision counter of the gauge field that was used to generate or transfer the clover field is not equal to its current revision.

Tracked multigrid instances

TL;DR: How and why MG has to track

Multigrid preconditioning generates a subspace from approximate low modes of the Dirac operator[2]. This requires tracking of all the parameters, gauge and clover fields, because the multigrid preconditioning operator depends on them. Therefore for every multigrid instance separately, we have to track all the parameters from above as well as the gauge field revisions. This is to allow different interface usage scenarios:

- Scenario 1): As an example, we assume the user wants to perform multiple solves with two different flavors, i. e. two different values of κ. We further assume the main loop in the program is written in a way that alternates between the two κ values and invokes the same multigrid solver instance for every solve. This requires the interface to update the multigrid instance every time the user switches flavor. Therefore, we store the values of the tracked parameters that were used when generating the subspace.
- Scenario 2): As in the previous scenario, we assume alternating between two flavors. However this time, the user specified two multigrid instances – one for each flavor. Again, in a main loop of the program, the user might loop over the flavors, sets the new flavor parameters and calls the corresponding solver instance. By this, the multigrid subspaces require no update on every solver call because from the perspective of the multigrid instances the parameters never change from call to call.

Clearly scenario 1) requires less memory than scenario 2) at the cost of updating the multigrid instance every time the flavor changes. However, a separate tracking of parameters and fields allows the user to flexibly work the way they prefer, without the interface imposing a programming paradigm. This was achieved by introducing an additional member in the `QudaInvertParam` struct that holds all these status information for every solver instance separately. The subspace of a multigrid instance is updated if at least one of the following statements is true:

---

[2] Large parts of part ii depend on multigrid and a thorough introduction to it will be given there.

- The gauge field requires an update.
- The clover field requires an update.
- At least one of the tracked parameters has changed.
- The internal revision counter of the gauge field that was used to generate this multigrid instance is not equal to its current revision.

We distinguish between two types of updates: thin and fat updates. A thin update happens if only tracked parameters have changed, a fat update if the gauge field has changed. A fat update destroys the current multigrid instance entirely and generates it from anew. As the name suggests, the fat update is computationally more expensive than the thin update.

Dependency graph

TL;DR: tracking dependency graph as overview

The dependency graph for the tracking of fields and parameters is plotted in fig. 5.4. The fields and parameters on the left and right of the dotted line live in openQxD and QUDA, respectively. The fields and parameters get transferred from left to right. On the right, the gauge and clover field as well as every multigrid instance holds their own set of parameters and keep track of synchronizing them as explained above. The dashed lines pointing to the clover field in QUDA are optional in the sense that if the gauge group is $SU(3)$ it is generated from the $SU(3)$ gauge field residing in QUDA, else it is transferred from openQxD.

5.1.6 *The solver setup function*

TL;DR: Solver setup + its call graph

One of the most important functions provided by the interface is the solver setup, see fig. 5.5. It is not meant to be called directly.

```c
void *openQCD_qudaSolverGetHandle(int id);
```

The solver identifier refers to the solver section of the input file that was passed when initializing. It makes sure that all necessary preliminaries are fulfilled for kernels that act on gauge or clover fields. The whole tracking mechanism discussed in section 5.1.5 is implemented by this function and is invoked by kernels, such as the Dirac operator or the inverter, to make

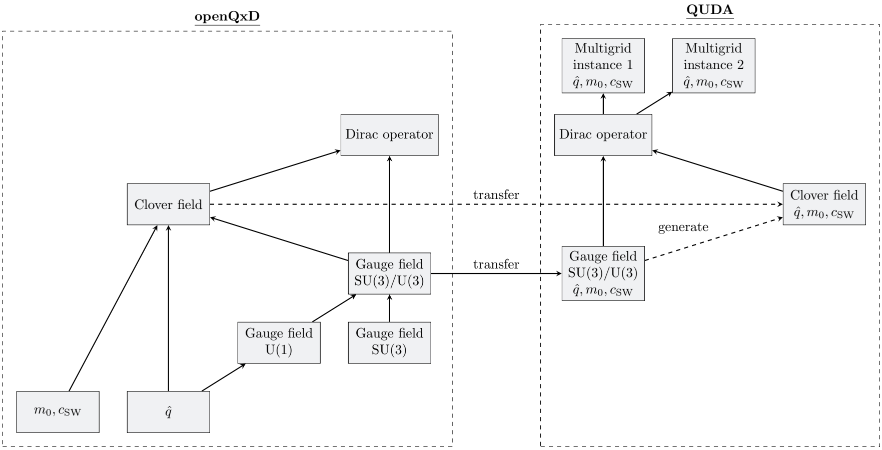

openQxD
QUDA
Multigrid instance 1 $\hat{q}, m_0, c_{\rm SW}$
Multigrid instance 2 $\hat{q}, m_0, c_{\rm SW}$
Dirac operator
Dirac operator
Clover field
transfer
Clover field $\hat{q}, m_0, c_{\rm SW}$
generate
Gauge field SU(3)/U(3)
transfer
Gauge field SU(3)/U(3) $\hat{q}, m_0, c_{\rm SW}$
Gauge field U(1)
Gauge field SU(3)
$m_0, c_{\rm SW}$
$\hat{q}$

FIGURE 5.4: Dependency graph of the various parameters and fields kept track by the interface.

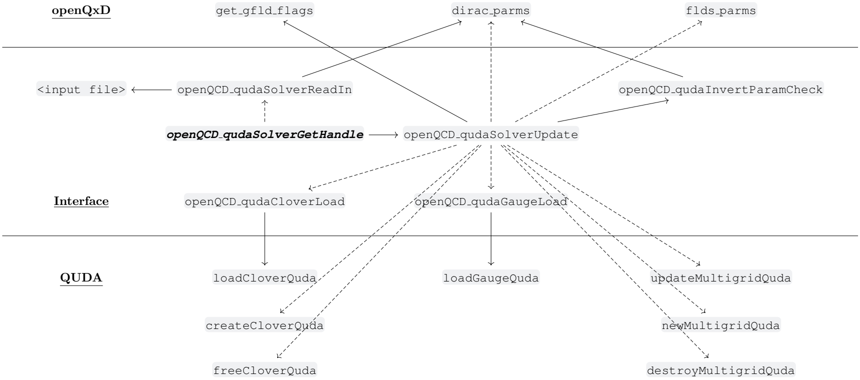

openQxD
get_gfld_flags
dirac_parms
flds_parms
<input file>
openQCD_qudaSolverReadIn
openQCD_qudaInvertParamCheck
openQCD_qudaSolverGetHandle
openQCD_qudaSolverUpdate
Interface
openQCD_qudaCloverLoad
openQCD_qudaGaugeLoad
QUDA
loadCloverQuda
loadGaugeQuda
updateMultigridQuda
createCloverQuda
newMultigridQuda
freeCloverQuda
destroyMultigridQuda

FIGURE 5.5: Call graph of the `openQCD_qudaSolverGetHandle` function. Solid lines imply the function will always call the function the arrow points to, dashed lines will call only if necessary. The graph divides into namespaces in openQxD (top) to obtain latest parameters and revision counter, the interface (center) and QUDA (bottom).

sure they act on synchronized fields and parameters. The call graph is divided into three namespaces based on where the functions are defined; in openQxD (top), in the interface code (center) and in QUDA (bottom). Functions required for state tracking come from openQxD; `get_gfld_flags` returns the current gauge field revision, `dirac_parms` the currently active Dirac operator parameters and `flds_parms` the gauge group. Solid lined arrows indicate that the pointed function is called every time by the pointing function, dashed arrows indicate that the pointed function is called only when necessary.

5.1.7 *The solver kernel*

TL;DR: Main solver function + arg discussion

Next, we enable offloading Dirac equation solves to the GPU. This is achieved by interfacing QUDA's solvers. From the side of openQxD, this kernel exposes itself by the interface function

```
double openQCD_qudaInvert(
    int id,
    double mu,
    spinor_dble *source,
    spinor_dble *solution,
    int *status
);
```

which closely resembles native solver function calls in openQxD in its signature. It takes 5 arguments:

- `id`: The solver identifier from the input file, i. e. `Solver #`. The input file is the one which was given to `quda_init()`.
- `mu`: In order to mimic the behavior of the openQCD solvers (e.g. `tmcg`, `sap_gcr`, `dfl_sap_gcr`), the invert-function accepts a twisted mass parameter `mu` explicitly.
- `source`: The source spinor field η (the right-hand side) given as a regular openQxD array of `spinor_dble` that might have been allocated and reserved using `alloc_wsd()` and `reserve_wsd()`, respectively. This field has to be allocated on the CPU and might be initialized as a point or random source when calling the function.
- `solution`: The solution spinor field ψ as an allocated `spinor_dble`, see `source`.

- `status`: If the solver is able to solve the Dirac equation up to the desired accuracy (see `invert_param->tol`), then `status` reports the total number of iteration steps (see `invert_param->iter`). A value of -1 indicates that the solver failed. This is the common behavior of solvers in openQxD.

TL;DR: Details about solver function

All fields passed and returned are host (CPU) fields in openQxD order. The function may be called at any time after `quda_init()`, see section 5.2.2. It will read in the input file provided to the interface by `quda_init()`, parse the parameter and populate the solver parameters internally. The whole tracking mechanism discussed in section 5.1.5 is applied before this function calls the actual solver in QUDA. That means Dirac operator parameters like $\kappa, \hat{q}$ or the SW-coefficients will be synchronized, gauge and clover field may be transferred or generated and multigrid subspaces will be generated or updated if necessary. This makes the function versatile and suitable for different use cases, some of which are discussed in section 6.2.

5.1.8 *The multiple right-hand sides solver kernel*

TL;DR: mrhs kernel + arg discussion

For repeated solves of the Dirac equation with unchanged gauge field configuration and Dirac operator parameters, QUDA offers the possibility to batch multiple solves together and by this reduce the overall memory footprint while increasing the arithmetic intensity, data locality and cache re-usage. This class of solvers is known as multiple right-hand sides (RHS) or block Krylov solvers [160, 165–170]. The interface function for a batched multiple right-hand side solve is

```
void openQCD_qudaInvertMultiSrc(
    int id,
    double mu,
    spinor_dble** sources,
    spinor_dble** solutions,
    int *status,
    double *residual
);
```

where the arguments are very similar to the standard solver kernel in section 5.1.7: the arguments `id` and `mu` are exactly the same, `sources` and `solutions` are now arrays of `num_src >= 1` fields, `status` and `residual` are already allocated arrays to hold the individual statuses and residuals of the solutions. The number of right-hand sides can be specified by the parameter `num_src`[3] in the input file under the solver section specified by `id`. The standard solver kernel and this function behave equally; the standard solver kernel even is a wrapper around this function.

5.1.9 *The asynchronous solver kernel*

TL;DR: async solver + implemenation concept

The solver kernels discussed in the two previous sections are executed by all ranks entering the function and remain blocking until the solver function returns. We have implemented an asynchronous way of calling the solver which may become interesting in certain observable evaluation scenarios, see section 6.2.4. This is enabled by spawning a worker thread for each MPI rank that enters QUDA and executes kernels, fig. 5.6. The main thread stays on the CPU during the time of the asynchronous operation. It should communicate using communicator 1, setup such that all the main threads are part of it but not the worker threads. The worker threads on the other hand will communicate using communicator 2 that includes all worker threads but not the main threads. Therefore, MPI calls issued by the main threads do not interfere with possibly concurrent MPI calls issued by the worker threads allowing seamless concurrent operation on the CPU and GPU. When the work is done, the worker threads are joined by the main threads and results are collected to ensure proper cleanup. The worker thread only exist during the asynchronous period which is between the start and wait functions discussed in section 5.1.9.

TL;DR: Function reference

The asynchronous solver interface consists of four functions listing 5.3, instead of one for the synchronous case.

```
void openQCD_qudaInvertAsyncSetup(int id, double mu);
void openQCD_qudaInvertAsyncDispatch(void *source, void *solution, int *status);
MPI_Comm openQCD_qudaInvertAsyncStart(void);
void openQCD_qudaInvertAsyncWait(double *residual);
```

Listing 5.3: The asynchronous solver interface functions.

They are split into a setup, dispatch, start and wait procedure and have to be called in this exact order.

Setup

TL;DR: setup function triggers whole setup

The setup function `openQCD_qudaInvertAsyncSetup` takes as argu-

---

[3] The standard solver kernel – if called with `num_src>1` – will fail.

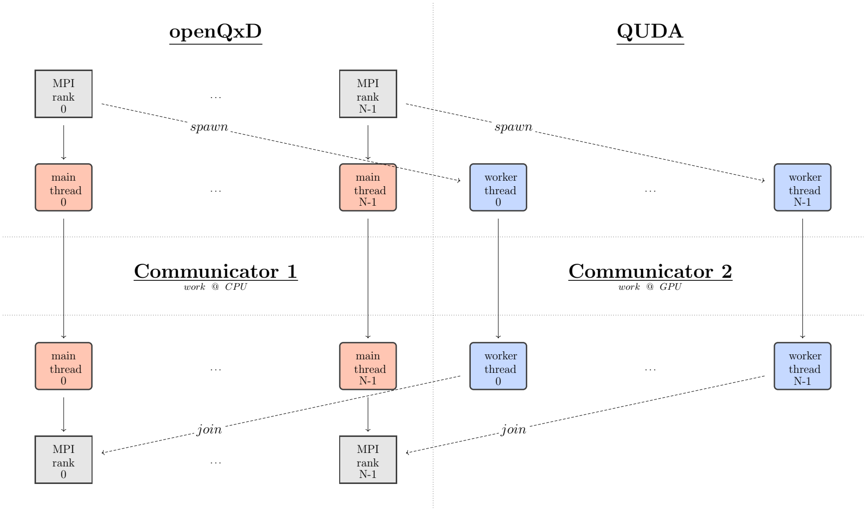

openQxD
QUDA
MPI rank 0
MPI rank N-1
spawn
spawn
main thread 0
main thread N-1
worker thread 0
worker thread N-1
Communicator 1
work @ CPU
Communicator 2
work @ GPU
main thread 0
main thread N-1
worker thread 0
worker thread N-1
join
join
MPI rank 0
MPI rank N-1
...

Figure 5.6: Thread spawn and join concept for the asynchronous solver interface. All MPI ranks spawn their individual worker thread which interacts with QUDA and communicates with a communicator setup to contain all worker threads. The main threads return immediately back to the CPU code and communicate using their dedicated communicator.

ments everything that might trigger transfers or updates of parameters; a solver identifier and the twisted mass parameter as discussed in section 6.2.1. This function will trigger necessary transfers or generation of fields and multigrid setups such that a subsequent solver call acts on synchronized data. Therefore, if Dirac operator parameters in openQxD are changed *after* calling this function and before starting the solves, they will not be re-transferred or synchronized.

Dispatch

The dispatch function `openQCD_qudaInvertAsyncDispatch` takes the remaining subset of arguments of the synchronous solver call, namely the source and solution spinors and a status variable. The function exit immediately, since all it does is adding a solve to the end of a worker queue. It can be called many times successively with different source and solution spinors to register multiple solves without invoking the actual solver yet. This is the job of the start function discussed next.

TL;DR: dispatch function is instantaneous

Start

TL;DR: start function is instant for main thread

The start function `openQCD_qudaInvertAsyncStart` takes no arguments. It will fire up the solves registered by `openQCD_qudaInvertAsyncDispatc` asynchronously. The solves will be worked through in order of dispatching. Every rank entering this function will spawn its own worker thread using `pthread_create` that enters the QUDA solver in place of the main thread, fig. 5.6. The main thread returns from the function immediately after spawning the worker thread ready to continue executing code on the CPU.

TL;DR: start function returns mpi comm, work on CPU, MPI INIT

If the CPU code involves MPI communication, the programmer must use the MPI communicator returned by this function instead of every communicator that was active before entering. The returned communicator will properly include all the main threads but not the worker threads. This is only possible if MPI was initialized with the proper level of thread support[4]. Such an MPI initialization routine is provided by the interface `quda_mpi_init` which can be used instead of `MPI_Init`, see section 5.2.1.

Wait

TL;DR: wait blocks until joined, collect results

Finally, the wait function `openQCD_qudaInvertAsyncWait` will instruct the main threads to wait for the worker threads to finish and reap them using `pthread_join`. If the worker threads have already finished when the main threads enter they will simply collect the results of the worker threads. That is, the residuals, status variables and solution vectors. Thus, when exiting the wait function, the solves have finished and output variables are populated as if the solver was called synchronously. This gives the programmer the opportunity to perform work on the CPU between the start and the wait function.

Multiple right-hand sides

TL;DR: mrhs is natural in async, batching of Nd solves

The asynchronous interface in this section naturally works in conjunction with the multiple right-hand sides solver kernel introduced in section 5.1.8 and was designed with a joint usage in mind, meaning that one can register a number of dispatched solves N_d with the dispatch function and trigger a series of batched solves with the start function. The start function will iterate through the N_d dispatched solves, collect batches of N_{rhs} of them (where N_{rhs} is equal to the number of right-hand sides) and call the multiple right-hand sides solver kernel as many times as required to solve all the N_d equations.

---

[4] This is usually the highest level of thread support, `MPI_THREAD_MULTIPLE` introduced in MPI-2.0 [171]. The MPI implementation must ensure that the MPI functions are thread safe which will degrade their performance slightly. Not all MPI implementations support such a high level of thread support. The interface code will gracefully error if the required thread support is not available by the MPI implementation.

TL;DR: batching example

As an example, assume we have dispatched $N_d = 12$ solves for a point source and the number of right-hand sides is $N_{\text{rhs}} = 5$. This will call the multiple right-hand sides solver routine 3 times in a loop. Its first iteration will solve the first 5 dispatched sources in parallel, then the second 5 solves in parallel, and finally with the remaining last 2 solves in parallel. Irrespective of the number of right-hand sides, the wait function will wait until all $N_d = 12$ solves are complete, collect the results and return.

5.1.10 *Other offloaded kernels*

TL;DR: Why other kernels?

The main kernel which is offloaded to QUDA is the solver, requiring all the previous and following sections to work correctly. The many ways how it is interfaced are discussed in the previous sections. However, for testing and illustration purposes, some other kernels that only require a subset of the interface functionality have been interfaced too. They are mostly called from within the check routines that enter the CI/CD pipeline, see chapter 8, such that parts of the interface can be tested in isolation.

TL;DR: incomplete list of other kernels

We do not discuss all kernels here but merely provide an incomplete list with brief explanations. More detailed descriptions can be found in the doxygen string of each function.

- `openQCD_qudaPlaquette()`: calculate the plaquette of the currently resident gauge field in QUDA.
- `openQCD_qudaGaugeLoad()`: transfer the gauge field explicitly independent of whether is it outdated or not.
- `openQCD_qudaCloverLoad()`: transfer the clover field explicitly independent of the gauge group and whether is it outdated or not.
- `openQCD_qudaGaugeFree()`: free the currently resident gauge field in QUDA.
- `openQCD_qudaCloverFree()()`: free the currently resident clover field in QUDA.

- `openQCD_qudaNorm()`: take the global norm of a CPU spinor field (involves spinor field transfer).
- `openQCD_qudaGamma()`: apply a γ-matrix in openQxD convention (involves spinor field transfer).
- `openQCD_qudaDw()`: apply the Wilson-Clover Dirac operator to a spinor[5] (involves spinor field transfer).
- `openQCD_qudaEigensolve()`: call the eigensolver[5] (involves spinor field transfer).

TL;DR: list of noloads kernels

Additionally, we have implemented a few interface functions that operate on device fields rather than host fields to illustrate how the development of the interface could continue.

- `openQCD_qudaH2D()`: allocate a device field, explicitly transfer a host spinor to the device and return its device pointer.
- `openQCD_qudaD2H()`: explicitly transfer a device spinor to the host.
- `openQCD_qudaSpinorFree()`: free a device spinor field allocated by `openQCD_qudaH2D()`.
- `openQCD_qudaDw_NoLoads()`: apply the Wilson-Clover Dirac operator to a device spinor[5].
- `openQCD_qudaNorm_NoLoads()`: take the global norm of a device spinor field.

5.1.11 *Finalization*

TL;DR: finalize

Finally, to destroy the QUDA context, we call the finalize function

```
void openQCD_qudaFinalize(void);
```

which deallocates all the solvers, eigensolvers and finalizes QUDA by calling `endQuda()`. Usually, this function does not have to be called explicitly, but one should rather call `quda_finalize()` which itself calls this function, see section 5.2.6.

---

[5] This kernel is fully worked out, functioning and behaves as the solver triggering the full parameter tracking mechanism.

5.2 CHANGES AND ADDITIONS TO OPENQXD

TL;DR: changes in openqxd + naming pattern

This section describes all the changes in openQxD core components, new modules and functions. All the interface functions which are related to or interact with QUDA have a naming pattern with prefix `quda_`.

5.2.1 *MPI Initialization*

TL;DR: How to init MPI

If the asynchronous solver functions from section 5.1.9 are used one should initialize MPI with the function

```
void quda_mpi_init(int *argc, char ***argv);
```

rather than the usual `MPI_Init` to ensure that MPI is initialized with the correct thread support and throws an error if initialization fails.

5.2.2 *QUDA Initialization*

TL;DR: Init of quda + arg discussion

To initialize QUDA the initialization function

```
openQCD_QudaInitArgs_t quda_init(char *infile, FILE *logfile);
```

takes two arguments. The `infile` argument takes the path to an input file where the interface expects some parameters and solver configurations to be specified. The solver parameters from the input file are parsed in the interface code, not in openQxD[6]. The reader is advised to refer to appendix B.1 for more details on the expected syntax of the input file. The second argument – `logfile` – points to an open file descriptor where QUDA should log its output to. It usually points to the same log file as is used by the main program itself (can be `stdout`). In QUDA, only rank 0 prints log messages. The return struct can be safely ignored (it is indeed required for some of the pipeline tests, see chapter 8).

TL;DR: Init under the hood

The function initializes a struct that describes the current lattice setup such as local and global lattice dimensions, boundary conditions (including the number of spatial C*-directions), the gauge group, the process grid and its mapping to MPI ranks and other static non-changing parameters. For properties with changing values such as Dirac operator parameters we hand over the relevant function pointers. For a detailed description of how the tracking of changing parameters works, see section 5.1.5. Finally, an MPI communicator is set up for usage in QUDA. This information is passed to QUDA by calling `openQCD_qudaInit()`, see section 5.1.1, which itself initializes the communication grid (and arranges the ranks as needed by the boundary conditions). QUDA is then initialized by calling `initQuda()`.

---

[6] We deliberately relinquished from passing the many solver parameters explicitly to QUDA for multiple reasons. In the fast development cycle of QUDA's codebase, new solver and parameters enter very frequently, requiring an update of the code every time. This should be done in QUDA where it belongs to. Furthermore, parsing of input files in openQxD is done in a very static, bulky and code-duplicating manner. We wanted to solve the problem of parsing the solver once and for all in the interface, forcing the programmer to not even try it to do it themselves.

TL;DR: Where to call it?

It is important where exactly `quda_init` is called in the bootstrapping phase of a main program. The boundary conditions have to be set by `set_bc_parms` *before* calling it because the process grid depends on them. The solvers have to be parsed by `read_solver_parms` *after* calling it because this might trigger field transfers already. We have done our best to throw meaningful error messages if `quda_init` is not called in the proper place.

5.2.3 *Solver parsing, validation and printing*

TL;DR: integration into solver parsing

The QUDA solver interface is integrated into core components of openQxD that parse, validate and print solver parameters. This means that the legacy functions

```c
void read_solver_parms(int isp);
void print_solver_parms(int *isap,int *idfl);
void write_solver_parms(FILE *fdat);
void check_solver_parms(FILE *fdat);
```

work with QUDA solvers out of the box. The last two functions did not even require any modification. For this to work, we wrote the interface functions

```c
void *openQCD_qudaSolverGetHandle(int id);
int openQCD_qudaSolverGetHash(int id);
void openQCD_qudaSolverPrintSetup(int id);
```

TL;DR: describe GetHandle

The first function returns a pointer to the solver context, i. e. pointing to the corresponding `QudaInvertParam` struct which configures the solver (see section 5.1.6). From the struct we may obtain settings as the requested relative residual for instance.

TL;DR: describe GetHaash

The second function returns a hash from the relevant subset of the many members of this struct. This is to be able to write QUDA solver parameters to disk using `write_solver_parms()` when a run has finished and to detect if settings have changed in `check_solver_parms()` when a run is restarted or continued. This is common practice in openQxD and fully supported by the interface. The hash has a length of 4 bytes and is generated using `std::hash`. We misuse the integer valued member `nmx` from the `solver_parms_t` struct to store this hash and to detect changes in the solver configuration.

The third function prints the whole solver setup, that is, all the parameters in the `QudaInvertParam` struct as well as the additional properties added by the interface.

TL;DR: describe PrintSetup

OpenQxD has 4 different solvers listed in the `solver_t` enum. To be able to distinguish the new solver, we added a fifth solver type named `QUDA` which collectively represents all available solvers in QUDA.

TL;DR: addition to solver struct

5.2.4 *The U(3) field*

The fact that openQxD and QUDA have substantially different memory layouts, see fig. 5.2, implies that some lattice points in the interior of QUDA's local lattice are boundary points in openQxD's local lattice and vice versa. Therefore, QUDA needs to access boundary points of the field on the CPU when performing the transfer and reordering of the data.

TL;DR: need boundary point transfer

In case of QCD+QED, openQxD holds the SU(3), the U(1) and the U(3) fields separate in memory. To be compatible with both gauge groups we always hand over the compound U(3) field to QUDA which is the SU(3) field with a U(1) phase multiplied to it. In case of QCD only, the U(3) gauge field is still SU(3) valued. Natively, this field does not have boundary points in openQxD. We thus added them as well as a function called `copy_bnd_hd` to populate them from the neighboring lattices. When transferring the field to QUDA, we hand over the U(3) field base pointer with populated boundary points.

TL;DR: su3 u1 and u3 field separate, we added copy-bdnud function

5.2.5 *Dual process grid*

Up to now, the interface code assumes that we have exactly one instance of QUDA per rank, i. e. the number of ranks and the number of QUDA instances (and by this the number of GPUs) are in one-to-one correspondence. This imposes major limitations for applications that consist of a fixed CPU part utilizing inversions via GPU. An example of such a program would be an observable contraction code utilizing MPI, performing inversions

TL;DR: 1to1 rank-gpu, cpu-gpu code pattern O1 GPUs but O100 cores

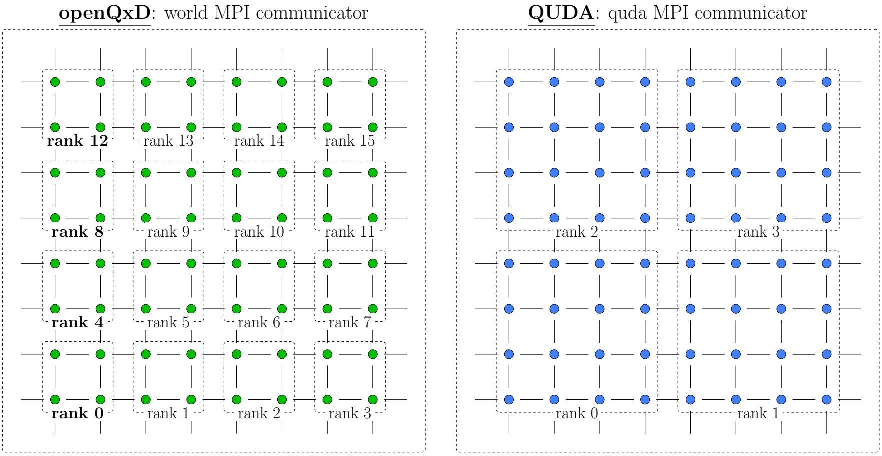

openQxD: world MPI communicator
rank 12
rank 13
rank 14
rank 15
rank 8
rank 9
rank 10
rank 11
rank 4
rank 5
rank 6
rank 7
rank 0
rank 1
rank 2
rank 3
QUDA: quda MPI communicator
rank 2
rank 3
rank 0
rank 1

FIGURE 5.7: Example of a global 8×8 lattice in 2D hosted by two different process grids. Left a 4×4 process grid with $N_w = 16$ ranks having local lattices of size 2×2 (a process block grid of 1×1) and right a 2×2 process grid with $N_q = 4$ ranks with local lattices of size 4×4.

on the GPU and contracting its results on the CPU (see figs. 6.1 and 6.2 and listings 6.4 and 6.5). Such programs have an alternating CPU-GPU pattern. The CPU code usually strong scales well with the number of ranks, so one should fill up all available CPU cores with processes. However, on modern clusters we usually have 1-8 GPUs per node, but the number of CPU cores per node is significantly higher in the regime $\mathcal{O}(100)$. In this section, we will address this asymmetry of parallelizability between CPU and GPU and enable above use-cases to nevertheless perform well.

TL;DR: dual process grid as concept + intergrid op

To enable such use cases, we have implemented the concept of what we call the *dual process grid* in openQxD, see fig. 5.7, i. e. an introduction of a second process grid topology. Both lattices must have the same global extents but may have different process grids and local lattices. The ranks in the process grid of openQxD, which we will call the *native grid* (left in fig. 5.7), are part of the world MPI communicator `MPI_COMM_WORLD` or a copy thereof, whereas a subset of the 16 ranks will host the local lattices of QUDA, called *dual grid* (right in fig. 5.7), and are part of an independent MPI communicator. All ranks will enter interface functions, but only a subset of them – the *dual ranks* – will interact with QUDA. For this we need to bring the data of spinor, gauge and clover fields to the dual ranks. This is

```
#define QUDA_NPROC0 NPROC0
#define QUDA_NPROC1 NPROC1
#define QUDA_NPROC2 NPROC2
#define QUDA_NPROC3 NPROC3

#define QUDA_L0 L0
#define QUDA_L1 L1
#define QUDA_L2 L2
#define QUDA_L3 L3
```

Listing 5.4: Excerpt of the file `include/quda_global.h` describing the (second) dual process grid. Its default values are equal to the native process grid and local lattice sizes from `include/global.h`, compare listing 3.1.

realized via an inter-grid operator that moves field data from native ranks to dual ranks and vice versa.

TL;DR: How to specify the dual grid

The dual process grid can be specified in a second header file called `include/quda_global.h` which looks like a reduced version of the original `include/global.h` file, see listing 5.4.

TL;DR: Constraints, compile time decision

Constraints for the dual lattice are the same as for the native one with some additional ones:

1. Both global lattices have to be equal.

2. The number of total processes in the native process grid N_w must be an integer multiple of the number of processes in the dual lattice N_q. This is guaranteed by a valid choice of the native process block grid (more on that later).

3. The z-direction, `L3` and `QUDA_L3` have to be equal in both grids.

At compile time the two process grids are checked for the constraints and compatibility. By default both grids are equal. This can be determined at compile time and is represented by the compile time constant `GRIDS_EQUAL`, making sure that code unnecessary for hosting the dual process grid will only be compiled if the two grids differ, allowing legacy performance if the dual grid feature is not used.

TL;DR: Which are the dual ranks, mappings native to dual

The subset of ranks which will interact with QUDA is simply determined by the condition that $c(r_w) = 0$ with

$$c(r_w) = \begin{cases} 0 & \text{if } r_w \equiv 0 \mod (N_w/N_q), \\ \texttt{MPI\_UNDEFINED} & \text{otherwise,} \end{cases} \tag{5.2}$$

where r_w is the rank number in the world communicator and N_w, N_q are the total number of processes in the native and dual process grid, respectively. Therefore, every (N_w/N_q)-th world rank will be part of the QUDA communicator. Their enumeration will be in order of the world rank enumeration. To determine dual ranks from native (world) ranks and vice versa, we have the mappings

$$r_w(r_q) = (N_w/N_q)\, r_q, \tag{5.3}$$

$$r_q(r_w) = \begin{cases} (N_q/N_w)\, r_w & \text{if } c(r_w) = 0, \\ \texttt{MPI\_INVALID\_RANK} & \text{otherwise}, \end{cases} \tag{5.4}$$

where r_w and r_q are the ranks numbers in the world and QUDA MPI communicators, respectively. These equations hold for periodic boundary conditions, C* boundaries will be discussed separately.

TL;DR: How to choose dual grid properly

We thus see that if we chose the two process grids properly, fig. 5.8 as opposed to fig. 5.7, we can make the operator transferring data between the two lattices to only require communication *inside* the node, not across nodes. We can achieve this by ensuring that:

1. Every native process grid extent is an integer multiple of the corresponding dual process grid extent.
2. The native process block grid is the component-wise ratio of the native process grid divided by the dual process grid.

Else the numbering of ranks would be lexicographic as in fig. 5.7 and the inter-grid operators would involve inter-node communication.

The code only compiles for valid process block grid choices. By this we make sure that one node only hosts entire blocks (block do not span multiple nodes) and one block actually corresponds to one GPU. This finally allows to implement the inter-grid data movement operator in terms of shared memory which reduces the overhead of this feature to negligible amounts. The implementation uses MPI shared memory windows [172] and data transfer is zero copying by directly accessing the dual ranks memory[7].

This feature is entirely opaque to the programmer and every application interfacing QUDA solvers is capable of the dual process grid out-of-the-box without any code changes. It also works seamlessly together with the asynchronous solver, multiple right-hand sides or both.

---

[7] If ranks are placed properly, data movement does not cross NUMA domains either. More on that in chapter 7.

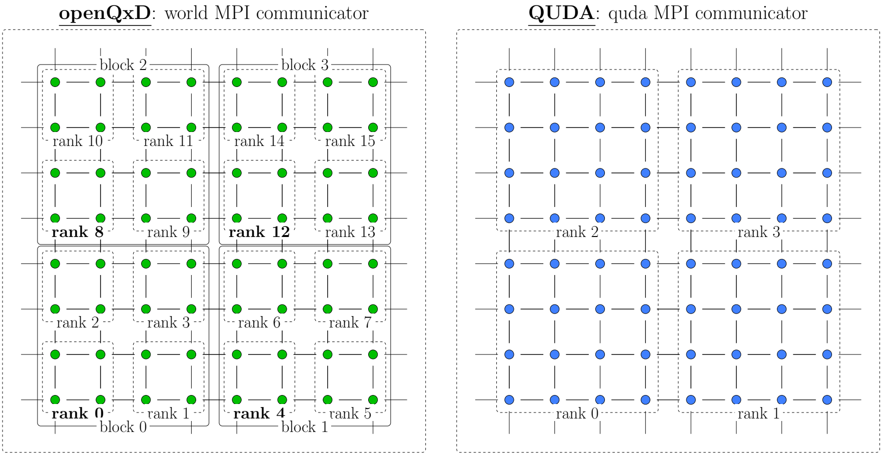

openQxD: world MPI communicator
block 2
block 3
rank 10
rank 11
rank 14
rank 15
rank 8
rank 9
rank 12
rank 13
rank 2
rank 3
rank 6
rank 7
rank 0
rank 1
rank 4
rank 5
block 0
block 1
QUDA: quda MPI communicator
rank 2
rank 3
rank 0
rank 1

FIGURE 5.8: The same two process grids as in fig. 5.7, but this time the openQxD (native) process grid has a non-trivial process block grid of extents 2×2. As opposed to fig. 5.7 this is the proper choice for this process block grid to optimize the communication pattern of the inter-grid operator. In both figures the bold written world rank numbers will be the subset of QUDA ranks, i. e. ranks with world rank numbers 0, 4, 8 and 12 (multiples of $N_w/N_q = 4$) on the left. They will become ranks with QUDA rank numbers 0, 1, 2 and 3, respectively, on the right.

C⋆ boundary conditions

In order to achieve the same behavior for C⋆ boundaries as for periodic ones, the formulas eqs. (5.2) to (5.4) have to be adjusted as

$$c(r_w) = \begin{cases} 0 & \text{if } r_w \equiv 0, 1 \mod (2N_w/N_q), \\ \texttt{MPI\_UNDEFINED} & \text{otherwise}, \end{cases} \tag{5.5}$$

and

$$r_w(r_q) = \begin{cases} (N_w/N_q)\, r_q & r_q \text{ even}, \\ (N_w/N_q)\,(r_q - 1) + 1 & r_q \text{ odd}, \end{cases} \tag{5.6}$$

$$r_q(r_w) = \begin{cases} (N_q/N_w)\, r_w & \text{if } c(r_w) = 0 \text{ and } r_w \text{ even}, \\ (N_q/N_w)\,(r_w - 1) + 1 & \text{if } c(r_w) = 0 \text{ and } r_w \text{ odd}, \\ \texttt{MPI\_INVALID\_RANK} & \text{otherwise}. \end{cases} \tag{5.7}$$

With these functions, we can eliminate inter-node communication in the exact same way as for periodic boundary conditions.

5.2.6 *Finalization*

TL;DR: finalize and where to call it

The finalize function

```
void quda_finalize(void);
```

requires no arguments and returns nothing. It is meant to be called right before the host application closes its logfile handle, since this function will write a summary to the logfile (if verbosity is at least `QUDA_SUMMARIZE`). The function is a mere wrapper around `openQCD_qudaFinalize`, see section 5.1.11.

5.3 SUMMARY

TODO: Add a bit more to this summary

This chapter introduced a comprehensive interface to access all solvers in QUDA from within openQxD with great versatility. Solvers can be accessed in numerous ways; either directly, with multiple RHS, or in the case of partial CPU workloads; in a heterogeneous way, or by defining a dual lattice. Even combining the heterogeneous solver call with multiple RHSs and the dual grid all at once is possible. This provides a lot of flexibility to the programmer accessing the functionality and for the person running the code efficiently in various production environments.

6

A DEVELOPER'S GUIDE FOR WORKING WITH THE INTERFACE

> *Programming is the art of telling another human being what one wants the computer to do.*
> — Donald Knuth

This section/chapter was proofread 1 times.

TL;DR: develop intro

This chapter describes how to easily port an existing program within the framework of openQxD to use the solvers available through the interface to QUDA from the viewpoint of an openQxD application level developer. It starts with the general procedure of porting (section 6.1), followed by a description of the many ways how the solver can be accessed (section 6.2). The chapter concludes with a brief summary including limitations of the code (section 6.3).

6.1 GENERAL PROCEDURE

TL;DR: basic header inclusion

In order to use QUDA in openQxD, we require to include the necessary header files that provide the interface to QUDA

```
#ifdef QUDA_INTERFACE
#include "quda_utils.h"
#endif /* QUDA_INTERFACE */
```

and link to `libquda.so`. Details on how to write a `Makefile` to link and compile against QUDA are discussed in appendix A. We add the compile time flag `QUDA_INTERFACE` for convenience such that one can enable or disable QUDA interfacing at wish. For a description of the provided functions in `quda_utils.h` please refer to section 5.2.

TL;DR: main program example

A usual application ported to use the QUDA interface may look like listing 6.1. Note that the only additional required parts are highlighted, the rest is standard. This makes all solvers in QUDA accessible in openQxD.

```
[...]

#ifdef QUDA_INTERFACE
#include "quda_utils.h"
```

```c
#endif /* QUDA_INTERFACE */

[...]

int main(int argc, char *argv[])
{
  [...]

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

  if (my_rank == 0)
    flog = freopen(log_file, "w", stdout);

  [...]

#ifdef QUDA_INTERFACE
  quda_init(infile, flog);
#endif /* QUDA_INTERFACE */

  /* application logic */

#ifdef QUDA_INTERFACE
  quda_finalize();
#endif /* QUDA_INTERFACE */

  if (my_rank == 0)
    fclose(flog);

  MPI_Finalize();
  exit(0);
}
```

LISTING 6.1: Example GPU-ported host application

6.2 ACCESSING THE SOLVER

TL;DR: solve function snippet

A paradigm that occurs quite often when writing applications in openQxD is a code snippet that dispatches to the correct solver. Such a function could easily be ported to dispatch to QUDAs solvers as well. That could look like the function `solve()` in listing 6.2 – again the relevant parts are highlighted.

TL;DR: same signature

We reiterate that the solver interface function(s) have the same signature as solver calls in openQxD. It was intentionally designed this way to allow seamless integration in such use cases.

If we build without the flag `QUDA_INTERFACE`, the highlighted parts are not compiled and the code falls back to the standard function that dispatches the available solvers present in openQxD[1]. With the flag set, we enable a new solver type `QUDA`. If the input file provides a solver of that type, the function in the snippet will offload the solve by calling the interface function `openQCD_qudaInvert()`. The variable `id` in the code snippet listing 6.2 is the solver identifier from the input file. It has to be passed to the interface function because there could be multiple solver definitions; possibly with some offloaded, some not. The snippet assumes that the input file has been parsed already by `read_solver_parms()`.

TL;DR: compile time constant

```c
/**
 * @brief      Solve the Dirac equation. This function assumes that the solvers
 *             are already parsed by read_solver_parms().
 *
 * @param[in]  id        Solver identifier
 * @param[in]  source    Source spinor
 * @param[out] solution  Solution spinor
 * @param[out] status    Solver status
 */
void solve(int id, spinor_dble *source, spinor_dble *solution, int *status)
{
  [...]
  solver_parms_t sp = solver_parms(id);

  if (sp.solver == CGNE) {
    /* call CGNE */
  } else if (sp.solver == SAP_GCR)    {
    /* call GCR with SAP preconditioning */
  } else if (sp.solver == DFL_SAP_GCR) {
    /* call GCR with deflation and SAP preconditioning */
#ifdef QUDA_INTERFACE
  } else if (sp.solver == QUDA) {
    /* call QUDA solver */
    openQCD_qudaInvert(id, 0.0, source, solution, status);
  }
#endif /* QUDA_INTERFACE */
  } else {
    error_root(1, 1, "solve ["__FILE__"]", "Unknown or unsupported solver");
  }

```

---

[1] If then called with a QUDA solver in the input file, the program will gracefully exit with a message `Unknown solver QUDA`.

```
  [...]
}
```

Listing 6.2: Example function to dispatch to the right solver. The call to the solver in QUDA is incorporated.

TL;DR: many modes to access solver

There are multiple modes on how the solver interface can be operating for maximal flexibility for the programmer. They target different motives and should be chosen according to the needs:

Section 6.2.1:	Ideal for new programs or programs using *only* QUDA solvers.
Section 6.2.2:	Ideal to porting existing programs.
Section 6.2.3:	Ideal for workloads with multiple batched solves.
Section 6.2.4:	Ideal for programs with alternating CPU-GPU workload.

6.2.1 *The simple way*

TL;DR: simplest mode, dont care mode

The solver function `openQCD_qudaInvert` can be safely called right after QUDA was initialized by `quda_init()`. It will parse the input parameters from the input file itself, transfer the gauge fields and generate or transfer the clover field if necessary. The interface code holds information about the relevant parameters for the Dirac operator and whether the fields are up-to-date or not. Namely these are:

- whether the gauge field is in sync or not,
- whether the clover field is in sync or not,
- `kappa/m0`: the (inverse) mass of the current flavor,
- `su3csw`: the SW-coefficient for the $SU(3)$ clover term,
- `u1csw`: the SW-coefficient for the $U(1)$ clover term,
- `qhat`: the charge of the current flavor.

At the time of calling the solver function, if the interface detects a change in one or multiple of these parameters on the side of openQxD, a synchronization of parameters and regeneration or re-transfer of the gauge and/or clover fields is triggered. This mode might be useful for new programs that do not plan to use solvers in openQxD anymore.

6.2.2 *The openQCD way*

Application-level developers and most existing programs in openQxD are very used to a standard workflow as

TL;DR: mode for legacy openqxd integration

1. parsing and validating the input file for parameters (including solvers),
2. printing the parsed parameters to the log file,
3. comparing solver parameters to stored ones from previous runs,
4. storing parsed solver parameters to disk and
5. performing some domain logic (solving the Dirac equation, for instance).

To support this programming paradigm, we made the interface flexible. It is fully integrated into the relevant core components of the openQxD, see section 5.2.3. This means that the programmer can use the functions `read_solver_parms` to read solver sections from the input file, `print_solver_par` to print information about the parsed solvers, `write_solver_parms` to write solver settings to disk and `check_solver_parms` to compare solver setups. All these functions work with QUDA solvers as well. For more details about these functions, see section 5.2.3. We refer to appendix B for an explanation on how the QUDA interface code expects the input file to look like.

6.2.3 *The multiple right-hand sides way*

If many repeated solves of the same Dirac equation are needed, they can be batched in stacks of `num_src` source and solution fields. As described in section 5.1.8, the batched fields can be solved all at once by improving memory access patterns at the cost of additional memory requirements. Every batched source spinor generates its own Krylov subspace leading to a multiplication of the required memory amount by a factor `num_src`.

TL;DR: simple mrhs mode

For the multiple right-hand sides solver kernel the same concepts as for the standard kernel hold as it can be called as described in sections 6.2.1 and 6.2.2.

TL;DR: mrhs also simple + legacy

A simple program using this solver kernel might look as listing 6.3.

TL;DR: mrhs example

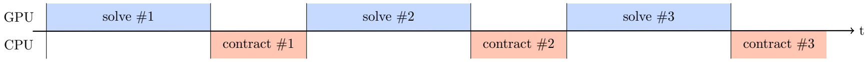

GPU
CPU
solve #1
contract #1
solve #2
contract #2
solve #3
contract #3
t

FIGURE 6.1: Serial timeline profile of a program where solves are offloaded to QUDA on the GPU, whereas contractions are computed within openQxD on the CPU.

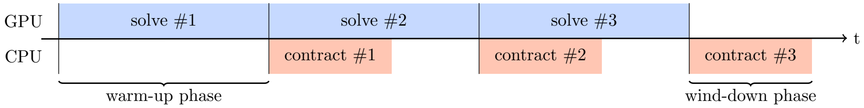

GPU
CPU
solve #1
solve #2
solve #3
contract #1
contract #2
contract #3
t
warm-up phase
wind-down phase

FIGURE 6.2: A timeline profile of a program issuing asynchronous calls to the solver, where after a warm-up phase an overlapping pattern of GPU and CPU work arises.

```c
for (int i = 0; i < Nsrc; ++i) {
  for (int j = 0; j < 12; ++j) {
    setup_source(j, sources[j]);
  }
  // num_src = 12
  openQCD_qudaInvertMultiSrc(id, mu, sources, solutions, states, residuals);
  [...]
}
```

LISTING 6.3: Example code employing a solver with 12 right-hand sides as compared to a single RHS in listing 6.4.

6.2.4 *The asynchronous way*

TL;DR: Async motivation, concept

Assume a scenario where the programmer wants to evaluate observables and contractions are implemented as MPI and/or openMP code on the CPU, whereas inversions are offloaded to QUDA. The timeline profile of such a program would look like fig. 6.1. To optimize such a scenario, we have implemented asynchronous interface functions for the solver, described in section 5.1.9. That is, to enable overlapping of contraction code (or any other arbitrary code) on the CPU with solver calls on the GPU, see fig. 6.2.

A synchronous example

TL;DR: serial example

A serial program might look as listing 6.4. A simple loop computing some inversions (`Nsolv` in the code might be set to 12 for a point source, for instance) and a subsequent contraction of the obtained solutions. We assume the `contract` function to compute an observable using the propagators as input and to store the result to disk. This is repeated `Nsrc` times for different source spinors. This coding pattern corresponds to fig. 6.1 where we see an alternating idling pattern for the CPU and the GPU. It does not fully utilize the available resources.

```c
for (int i = 0; i < Nsrc; ++i) {
  for (int j = 0; j < Nsolv; ++j) {
    residual = openQCD_qudaInvert(id, mu, source[j], solution[j], status);
  }
  contract(i, solution, ...);
}
```

Listing 6.4: Example code employing a serial timeline corresponding to fig. 6.1.

An asynchronous example

TL;DR: async example

A program as listing 6.4 can be easily rewritten into one that uses the asynchronous solver interface. This may look as in listing 6.5. For function descriptions, please refer to section 5.1.9. We can clearly identify the warm-up phase as the first iteration of the outer loop, and the wind-down phase as the finishing contraction in the last line. In between, we see the filled pipeline which solves the current set of sources and contracts the last set of solutions concurrently. Note that the contraction between the start and wait function must use the returned MPI communicator `comm`[2]. Whatever communicator contained all ranks before the asynchronous calls (including `MPI_COMM_WORLD`) may safely be used again after the threads have joined, i. e. after the wait function returned.

```c
openQCD_qudaInvertAsyncSetup(id, mu);
for (int i = 0; i < Nsrc; ++i) {
  for (int j = 0; j < Nsolv; ++j)
    openQCD_qudaInvertAsyncDispatch(source[j], solution[j], status[j]);

  comm = openQCD_qudaInvertAsyncStart();

```

---

[2] We note that the global communicator `MPI_COMM_WORLD` provided by MPI to contain all processes will not work as expected in the above scenario between the start and wait functions and should be avoided because all threads belong to this communicator by definition. Communication using `MPI_COMM_WORLD` will most certainly error or misbehave on strict MPI implementations.

```
  if (i != 0) /* i==0 -> warm-up */
    contract(i, last_solution, comm, ...);

  openQCD_qudaInvertAsyncWait(residuals);

  for (int j = 0; j < Nsolv; ++j)
    flip(last_solution[j], solution[j]); /* flips pointers */
}
contract(i, last_solution, MPI_COMM_WORLD, ...); /* wind-down */
```

Listing 6.5: Example code employing a pipelining timeline corresponding to fig. 6.2.

6.3 SUMMARY

We have seen how to use the interface to port existing applications written in openQxD, write new ones and how to easily implement heterogeneous computing by overlapping solves on the GPU with arbitrary workload on the CPU. Through the parameter and revision tracking system, the interface allows flexible usage over many use cases and stays consistent with openQxD's internal state.

6.3.1 *Limitations*

TL;DR: Forced field transfers for solver kernels

Currently, there are some limits regarding the interface. Irrespective of how the solver is called, source fields will always be transferred to the GPU and solution fields be transferred back. A field transfer is expensive, since it consists of data reordering and memory copy between host and device. Reordering can be configured to happen either on the host or on the device. In any case, further memory copying is introduced. These transfers – as of now – cannot be disabled. In a future scenario where we may want to further process the fields on GPU after the solver call, it will be inefficient to perform unnecessary transfers between all kernels. A conceptual solution to this problem is outlined in chapter 9.

TL;DR: Forced field transfers for all kernels

The same holds for some of the other kernels, like the Dirac operator or the γ-matrix application. Current versions of these kernels are just for illustration and testing purposes and should not be used in a production code. Since they transfer fields back and fourth and are themselves very lightweight as opposed to the solver kernel, these kernels are highly inefficient; the largest fraction of their runtime will be spent in the field transfers.

Although some of them have versions which do not trigger transfers and operate directly on device fields. They serve as proof of concept examples to pave the ground for a possible continuation of developments in terms of chapter 9.

7

PERFORMANCE ASSESSMENT OF INTERFACE COMPONENTS

> *The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.*
>
> — Tom Cargil

This section/chapter was proof-read 1 times.

TL;DR: intro

This chapter will present performance assessments of the various implementations discussed in chapter 5. The goal is to determine production readiness of the code to run on current accelerator-based super-computing centers, focusing on good performance on the very recent GH200 (Grace-Hopper) node at CSCS [127]. All machines involved in data taking are specified in section 7.1. The physics program of the $\mathrm{RC}^\star$ collaboration is centered around lattices with $\mathrm{C}^\star$ boundary conditions and occasionally periodic ones. Thus, we ran performance tests on lattices specified in section 7.2, whereas the methodology of measurements is discussed in section 7.3. Among the investigated features were runtime contributions of the dual process grid (section 7.4), the efficiency of heterogeneous workloads using the asynchronous solver with concurrent workload on the CPU (section 7.5), the blocked solver acting on multiple RHS at the same time (section 7.6), asymptotic strong and weak scaling behavior of the Dirac operator (section 7.7) and finally scaling of multigrid solver used in production (section 7.8). The chapter ends with concluding remarks and limitation (section 7.9).

7.1 HARDWARE SPECIFICATIONS

TL;DR: machine we ran on

The performance measures that follow were obtained on the machines

- *Daint-Alps* at CSCS, Switzerland, 4× NVIDIA® Grace @ 3GHz (72 Arm Neoverse V2 cores), 4× 128GB LPDDR memory, 4× NVIDIA® Tesla® H100 (96GB HBM2 memory)

- *Leonardo Booster* at CINECA, Italy, 1× Intel® Xeon® Platinum 8358 @ 2.6 GHz (32 cores, Icelake), 8× 64 GB DDR4-3200 memory, 4× NVIDIA® Ampere A100 (64GB HBM2 memory) [173]

The Quad GH200 node is an interesting machine, since it comes with many new concepts and a sophisticated memory hierarchy [127]. All compute units (CPUs and GPUs) within the node are able to access each others memory directly, enabling zero-copy access. This hierarchy of the unified address space is fully coherent and is abstracted with the help of individual NUMA domains. The chips themselves are connected via NVLink-C2C, a high bandwidth low latency chip-to-chip interconnect. This will be one of the target machines of the code and careful placement of processes to match NUMA domains was taken special emphasis on.

7.2 TESTED ENSEMBLES

TL;DR: lattices we ran on

The performance measures that follow were obtained using the ensembles described in table 7.1.

Name	Global size V_G	α	Pion mass	Boundaries
G8	$64^3 \times 128$	0	180 MeV	periodic
F7	$48^3 \times 96$	0	270 MeV	periodic
A360	$32^3 \times 64$	0.05	360 MeV	C*
A400	$32^3 \times 64$	0	400 MeV	C*
C380	$48^3 \times 96$	0.05	380 MeV	C*
D300	$64^3 \times 128$	0	300 MeV	C*

Table 7.1: Ensembles used for the performance tests in this chapter. G8 and F7 has been generated by the CLS initiative [120], while A360, A400, C380 and D300 were generated by the RC* collaboration [112]. The last column refers to spatial boundary conditions and α is the electromagnetic coupling. Ensembles with $\alpha \neq 0$ are dynamic QCD+QED simulations.

7.3 METHODOLOGY

TL;DR: library versions

Most runs were conducted on the GH200 nodes at CSCS with a few exceptions. The MPI implementation was Cray MPICH 8.1.30 CUDA-aware with GTL on top of Libfabric 1.15.2.0. We used CUDA 12.4 with Driver version 550.54.15 and GCC 13.2.

Timing measurements were obtained by taking an `MPI_Barrier` to wait for all participating processes followed by a call to `MPI_Wtime`. The absolute time returned by `MPI_Wtime` had a granularity of 32 ns. This methodology was used to determine start and stop times, and differences were taken to extract timings of certain routines. All measurements were averaged over at least 5 independent runs if not stated otherwise.

TL;DR: how timings were taken

For energy measurements, we read from the PM counters provided by HPE that collect point-in-time telemetry of power data on node or blade level. Their refresh rate was 10 Hz and the power management counter support version was 3. The numbers were taken from the sysfs filesystem under `/sys/cray/pm_counters` on each node. No information about the granularity of these metrics was available, thus, they are only indicative and their precision has to be taken with a grain of salt.

TL;DR: how energy meas. were taken

All runs conducted were setup with correct NUMA placement and pinning of processes. The Quad GH200 node has 36 NUMA domains [127]. Each Grace CPU is affine to 128 GB DDR memory and NUMA domains 0,1,2 and 3. Each H100 GPU is affine to 96 GB HBM memory and NUMA domains 4, 12, 20 and 28. The remaining NUMA domains are used only when multi-instance GPU (MIG) is activated. However, GPU n is also affine to NUMA domain n for $n = 0, 1, 2, 3$. Therefore, maximal performance can be achieved by placing MPI processes evenly divided into NUMA domains. If not all 288 cores are used, which is usually the case, we evenly fill NUMA domains. Examples of explicit bindings are discussed in the main text.

TL;DR: numa setup of the quad GH200

The word *kernel* is used for both; functions implemented on the CPU using C/C++ or functions implemented on the GPU using CUDA. We measure timings and other metrics of kernels.

TL;DR: what is a kernel

7.4 THE DUAL LATTICE

In this section, we investigate and quantify the performance slowdown imposed by the usage of the dual process grid (section 5.2.5) and show that it is basically negligible. We argued that one has to use the correct process block grid to optimize the performance of the inter-grid operator. Implementation-wise this is forced and the code does not compile if the process block grid was not chosen properly.

TL;DR: dual: correct blk grid is forced

We start investigating the effect of varying the number of ranks per node in fig. 7.1. The plot shows the number of ranks per node versus the cost

TL;DR: dual: ranks per node

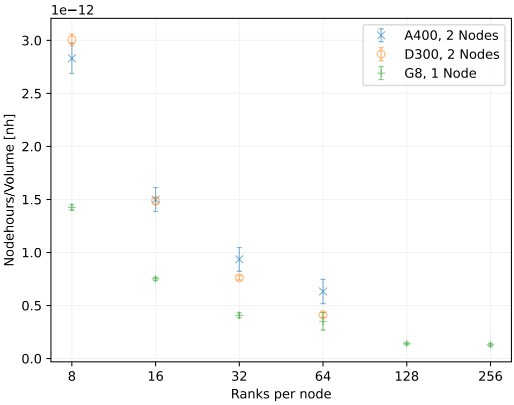


Figure 7.1: Number of native ranks per node versus application time of the inter-grid operator moving data between the native and dual process grid, keeping the number of dual ranks and nodes fixed. The data was taken on *Daint-Alps*, see section 7.1.

in nodehours per physical lattice point for an application of the inter-grid operator while keeping the number of nodes fixed. 4 ranks per node would be the case where no dual lattice is used, i. e. both lattices are equal and the cost of the inter-grid operator would be 0 identically. The plot shows the usual strong scaling behavior until the node memory bandwidth is saturated.

TL;DR: dual: amount of data movement

The processes were pinned such that data exchange only happens inside a NUMA domain. Nevertheless, the whole field has to be moved in memory in all cases. As expected, for the periodic lattice G8 compared to the the $C^\star$ lattice D300 (both have the same physical lattice size), we observe roughly a ratio of 2.

TL;DR: dual: fill node is good

The dual lattice is primarily used when there is some non-negligible pure-MPI workload left on the CPU. We see that the inter-grid operator performs best when filling the node with ranks. This is fortunate because this usually holds true for the CPU workload as well. To summarize, we can say that when working with the dual lattice, it is fine to use as many ranks per node as desired.

TL;DR: dual: strong scaling nodes

We claimed in section 5.2.5 that inter-node communication is eliminated. This is shown in fig. 7.2. We observe embarrassingly parallel behavior with respect to the number of nodes, even when going up to 128 nodes (512 GPUs) on the smallest lattice making the inter-grid operator strong scaling

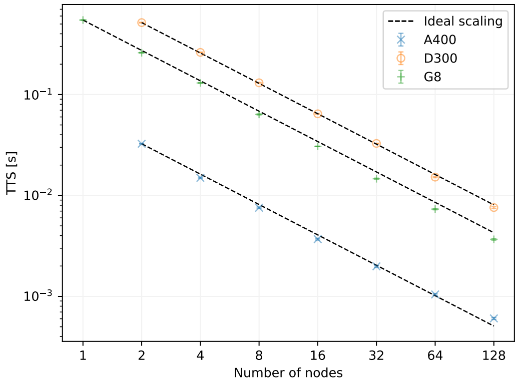


FIGURE 7.2: Strong scaling of inter-grid operator. Number of nodes versus application time of the inter-grid operator, keeping the number of native and dual ranks per node fixed. The data was taken on *Daint-Alps*, see section 7.1.

ideal. The base-case for the C⋆ lattices is 2 nodes (8 GPUs) because of an implementation restriction[1].

Finally, we discuss the contribution of the inter-grid operator to a solve of the Dirac equation using a multigrid solver – a production example of the dual grid in action is given in fig. 7.3. The plot shows five contributions; moving the source vector from the native to the dual lattice (blue) and subsequently to the GPU (green), solving the linear system of equations on the GPU (purple), moving the solution vector from the GPU back to the dual lattice (red) and subsequently to the native lattice (yellow). We see that the inter-grid contribution decreases as one increases the node count, same holds true for CPU-GPU transfers. This is expected, since both kernels involve no communication. The only contribution that does not strong scale perfectly – the solver itself as we will see later – increases its contribution when increasing the number of nodes. However, we observe that the inter-grid operator contributes negligibly to the overall solve.

TL;DR: dual: contribution negligible

Regarding the dual lattice, we ensured that native and dual processes corresponding to the same local lattices were in the same NUMA domain to prevent communication from one NUMA domain to another further optimizing its performance. Due to the difference in processes numbering

TL;DR: dual: numa placement: 10-20 percent additional speedup

---

[1] The current implementation demands a process grid length of at least 2 in every direction that is a spatial C⋆-direction. With 3 C⋆ directions this gives a minimum of 8 processes, i. e. 8 GPUs unless one uses MPS or MIG. MPS gives a severe performance degradation of the GPU part, and MIG is not available on request on *Daint-Alps*.

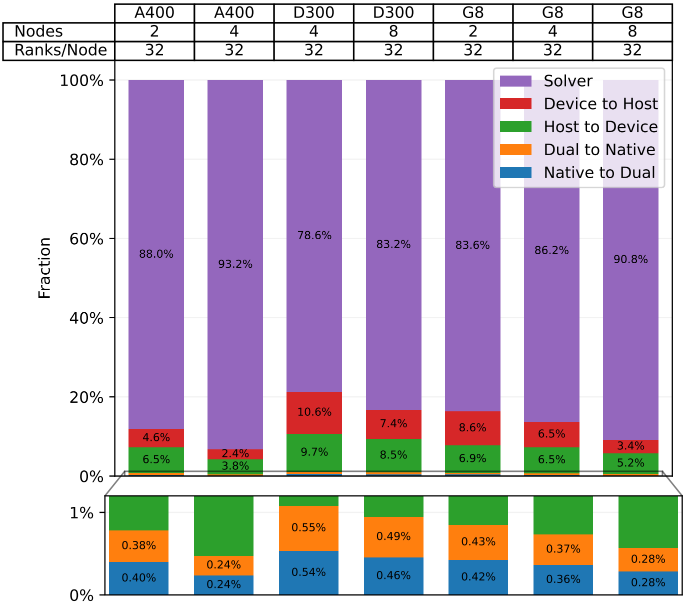


FIGURE 7.3: Contribution of data movement kernels to the solver. The plot shows different lattices and setups specified on the x-axis. Blue and yellow indicate the inter-grid data movement operator between the native and dual process grids, whereas green and red indicate the CPU-GPU data transfer and reordering procedure. Finally purple shows the contribution of the actual Krylov solver on the GPU. The contribution of the inter-grid operator is negligible in all cases as can be seen in the inset. The data was taken on *Daint-Alps*, see section 7.1.

in periodic and C* cases, see eqs. (5.2) and (5.5), the CPU-binding has to be different as well. To illustrate this, we show an explicit example of the pinning and core distribution for a run with a single GH200 node and 32 rank per node in tables 7.2 and 7.3 for periodic and C* boundary conditions, respectively. The additional effect on performance of correct NUMA placement was in the regime of 10 % to 20 %.

TL;DR: dual: summary

We investigated the behavior and contributions of the dual process grid. Its contribution was insignificant in all cases and can even be pushed down further by increasing the number of nodes due to its perfect strong scaling. However, the solver algorithm does not strong scale perfectly as we will see later. This trade-off has to be taken into account when considering the setup. As usual in HPC, the best performing setup is a compromise between

Native				Dual			
MPI rank	Core ID	NUMA domain	CPU ID	MPI rank	Core ID	NUMA domain	GPU ID
0-7	0-7	0	0	0	0	0	0
8-15	72-79	1	1	8	72	1	1
16-23	144-151	2	2	16	144	2	2
24-31	216-223	3	3	24	216	3	3

TABLE 7.2: Example of NUMA process placement to core IDs with periodic boundaries. With 32 tasks per node, we divide them into 8 tasks per NUMA domain associated to one CPU, filling only 8 from the available 72 cores. The dual MPI rank is calculated using eq. (5.2) and should be in the same NUMA domain. The GPUs are in NUMA domains 4,12,20 and 28 but also affine to 0, 1, 2 and 3, respectively.

multiple competing motives. The largest contribution should be considered primarily.

7.5 THE ASYNCHRONOUS SOLVER

Heterogeneous computing can be achieved by overlapping expensive solves on the GPU with some independent work on the CPU by using the asynchronous solver interface introduced in sections 5.1.9 and 6.2.4. The performance of different kernels on the CPU overlapping with solves on the GPU were assessed in fig. 7.4. Kernels on the CPU are described as follows:

TL;DR: async: CPU kernel description

a) *sleep()*: This kernel is trivial and serves as a baseline. While the GPU performs the solve, the CPU repeatedly calls `nanosleep()` for about the same amount of time.

b) *I/O*: A pure I/O kernel performing a read-in of a large amount of data located at scratch with very few MPI calls of little data amounts.

c) *I/O+MPI*: A kernel performing the same read-in as before, but additionally the data is scattered among all ranks via MPI.

d) *Dirac operator*: Repeated applications of the Dirac stencil, a memory bandwidth bound operation including halo exchange and barriers.

Native				Dual			
MPI rank	Core ID	NUMA domain	CPU ID	MPI rank	Core ID	NUMA domain	GPU ID
$0, 2, \ldots, 14$	0-7	0	0	0	0	0	0
$1, 3, \ldots, 15$	72-79	1	1	1	72	1	1
$16, 18, \ldots, 30$	144-151	2	2	16	144	2	2
$17, 19, \ldots, 31$	216-223	3	3	17	216	3	3

Table 7.3: Example of NUMA process placement to core IDs with 32 tasks per node and $C^\star$ boundaries. For detail see table 7.2.

e) *MPI_Allreduce()*: Repeated applications of `MPI_Allreduce()` to comparably small amounts of data. This is a pure MPI kernel doing only communication.

f) *Memory*: A pure local memory traffic producing kernel without any communication, accessing rank-local data repeatedly.

TL;DR: async: leonardo, contributions

In fig. 7.4, we observe clear speedups of the heterogeneous case compared to the serial execution model. Slight performance degradation only occurs in kernels heavily utilizing MPI. Furthermore, the performance of the solve on the GPU is only negligibly affected by the concurrent work done on the CPU.

TL;DR: async: leonardo, speedups

Figure 7.5 shows the speedup of the overlapping case over the serial workload. The GPU solver as well as the CPU kernels are barely affected by overlapping, resulting in a total speedup close to the analytic limit of 2 for all kernels. This means we can successfully hide the entire CPU or GPU work by instantiating a pipelined execution model and fully utilize all available resources in a node.

TL;DR: async: daint, contributions

The data for the heterogeneous solver was taken on *Leonardo Booster*. It is worth looking at the same plot for a GH200 node at *Daint-Alps*, see fig. 7.6. The data shows a completely different picture: it is not worth doing entirely. On the contrary; performance is even degraded compared to the serial case. This originates from the fact that the power module of the GH200 node at CSCS is capped at around 650 W and the CPU will always draw as much power from it as it can, letting the GPU starve. The two devices compete for energy. On such a machine a serial execution model performs better, because when the CPU draws power, the GPU is idle and vice versa. As

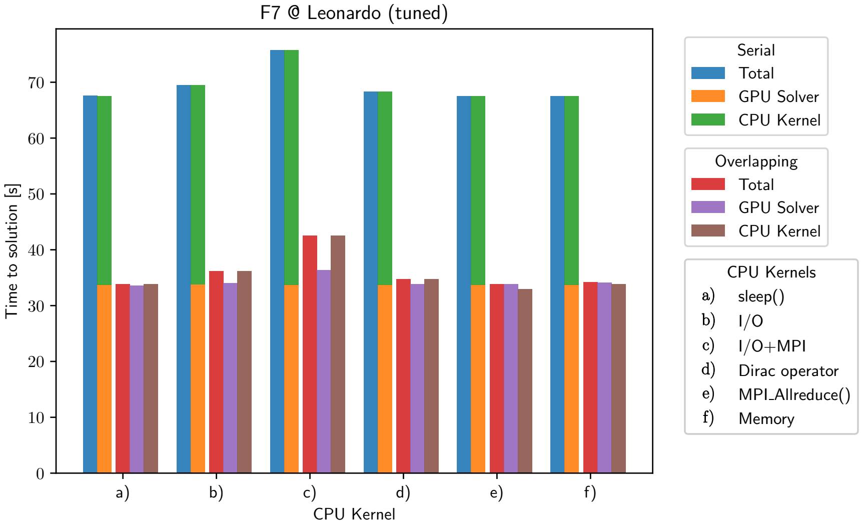


FIGURE 7.4: Contributions from different kernels a) to f) on the CPU to a Krylov solve on the GPU. Blue, yellow and green show the serial pipeline (see fig. 6.1 and listing 6.4), whereas red, purple and brown show the overlapping pipeline (see fig. 6.2 and listing 6.5) averaged over 10 solves. The data was taken on *Leonardo Booster*, see section 7.1.

backed by the data in the plot, it is currently[2] not a good idea to overlap CPU and GPU calculations on the GH200 node.

7.6 THE MULTIPLE RIGHT-HAND SIDES SOLVER

The multiple right-hand sides solver interface is discussed in section 5.1.8 and can be accessed as described in section 6.2.3. Figure 7.7 shows its speedup versus the number of right-hand sides.The base case is a single right-hand side. The theoretical maximal speedup[3] is around 1.375-4 . On the small lattice A400 we reach a higher speedup as on the large lattice

TL;DR:
mrhs: daint
speedup
data

---

[2] The problem is known by CSCS, NVIDIA and HPE.

[3] This number is obtained by considering a performance model as

$$\text{cost}(1) = \text{mem}(D) + 2x \cdot \text{mem}(\psi) \qquad \text{single RHS}, \qquad (7.1)$$

$$\text{cost}(\infty) = 2x \cdot \text{mem}(\psi) \qquad \text{for } N_{\text{rhs}} \to \infty, \qquad (7.2)$$

for the cost of an application of the Wilson-Clover Dirac operator, where mem(A) denotes the memory traffic of the object A. Naively $x = 8$ for fetching 8 neighboring points and $x = 1$

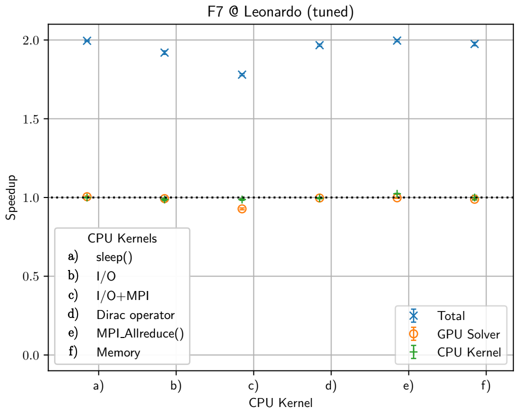


FIGURE 7.5: Absolute speedup of the heterogeneous versus the serial execution model averaged over 10 solves. A speedup of 2 is the analytical maximum. The effect of overlapping for the GPU solve as well as the individual kernels are indicated by yellow and green, whereas the blue shows the net speedup when overlapping. The data was taken on *Leonardo Booster*, see section 7.1.

D300. This is due to the fact that the increase in data locality and parallelism has a higher impact on small problems which are already in the bad strong scaling region as opposed to large ones. However, the speedup directly translates to cost efficiency.

TL;DR: mrhs: daint energy consumption

It is worth to study the impact on energy consumption in fig. 7.8. The better temporal data locality and larger local problem size allows us to come closer to saturating the GPU. As we increase the number of RHSs, the energy required per solve decreases, whereas the power drawn from the power module increases. Saturation can be observed at about 12 to 16 RHSs. This is a clear indication that the lattice considered is too small for 2 Quad GH200 nodes, not exposing enough parallelizability. On the other hand, solving 16 or more RHSs at the same time exposes substantially more parallelizability to the GPU and is able to saturate the 2 nodes. Even more importantly the energy cost per solve decreases by a factor of about 1.6.

---

for perfect caching. The reality is in between. Thus, using $\mathrm{mem}(D) = 6 \cdot \mathrm{mem}(\psi)$, the ratio of these two $\frac{\mathrm{cost}(1)}{\mathrm{cost}(\infty)} = \frac{x+3}{x} \in [1.375, 4]$.

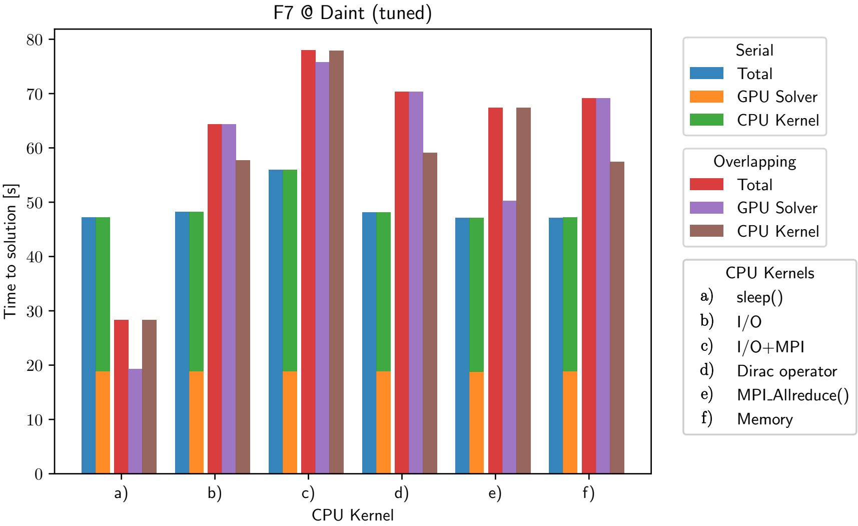


Figure 7.6: Contributions from different kernels a) to f) on the CPU to a Krylov solve on the GPU. Blue, yellow and green show the serial pipeline (see fig. 6.1 and listing 6.4), whereas red, purple and brown show the overlapping pipeline (see fig. 6.2 and listing 6.5) averaged over 10 solves. The data was taken on *Daint-Alps*, see section 7.1.

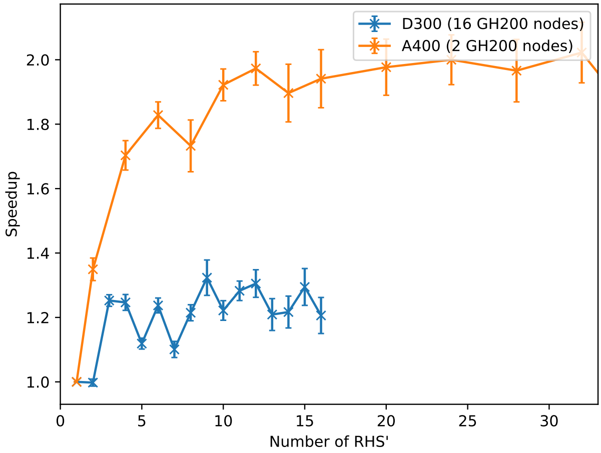


Figure 7.7: Number of right-hand sides versus speedup compared to a single right-hand side. For small problems, a twofold speedup is reached. The data was taken on *Daint-Alps*, see section 7.1.

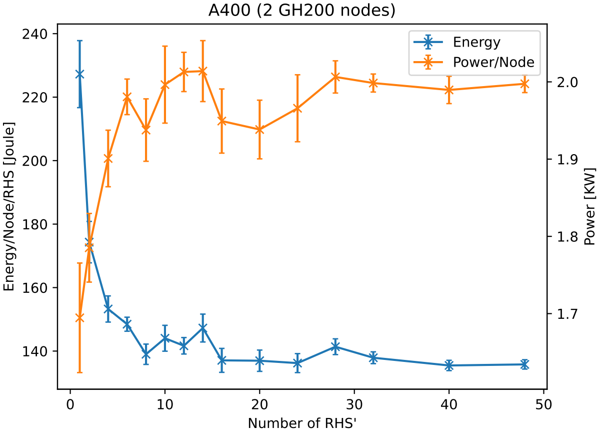


Figure 7.8: Number of right-hand sides versus energy consumption per RHS per node (blue) and versus power drawn from the power module per node (yellow) of the A400 lattice running on 2 nodes. Increasing the number of RHSs saturates the nodes. The data was taken on *Daint-Alps*, see section 7.1.

7.7 DIRAC OPERATOR

TL;DR: Dop important kernel in solvers

One of the most important kernels of every lattice application is the Dirac operator. It is crucial to have an efficient implementation of the Dirac stencil, since iterative Krylov solvers themselves call the operator once or twice in every iteration. We are interested in the pure nearest-neighbor stencil as it appears in solver algorithms. Thus, we assume the field to already reside in HBM memory on the GPU and the following plots will neither include H2D nor D2H transfers.

TL;DR: Dop: strong scaling daint

Figure 7.9 shows strong scaling of the GPU implementation of the Dirac operator. We focused on a periodic and a C* lattice with the same global physical lattice extents of $T \times L^3 = 128 \times 64^3$. The time taken was averaged over 10 000 invocations to the operator and at least 5 independent runs were conducted for every data point. We observe decent strong scaling. For this lattice size on a GPU, it is expected that for 64 to 256 nodes (256 to 1024 GPUs) the scaling is far from ideal. One reason is a too small local problem size to saturate the high parallelizability of the H100. For comparison, the local lattice size for the 256 node case is 16^4 resulting in a local vector size of about 12.58 MB. One H100 has 132 streaming multiprocessors (SMs), 64 warps per SM with a warp size of 32. Therefore, the 16^4 local lattice, with

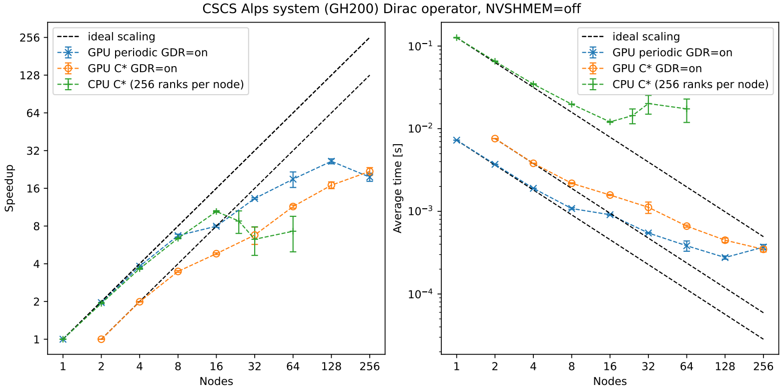


FIGURE 7.9: Strong scaling of Dirac stencil (left panel) and average time for one application (right panel). The operator was applied 10 000 times and the average was taken. The base case for the C⋆ lattice is two nodes for implementation restrictions. The physical global lattice size was $T \times L^3 = 128 \times 64^3$ for all runs. Every data point consists of at least 5 independent runs. The data was taken on *Daint-Alps*, see section 7.1.

its 65 536 lattice points, is distributed to 270 336 available CUDA threads on a single H100. Even the smaller 64 node case with 262 144 local lattice points is barely able to saturate the machine.

Since the performance of the operator only depends on the problem size and boundary conditions but not on physics parameters, we can easily run it on synthetic data to obtain a weak scaling plot as in fig. 7.10. In the weak scaling study, the physical local lattice size is kept constant. This is $64 \times 32 \times 64 \times 64$ for the periodic lattice and $128 \times 32 \times 64 \times 32$ for the C⋆ lattice. These numbers were chosen such that both lattices show the exact same global lattice extents for the 2 node case and onward. Additionally, the number of local lattice points (about 8 m for periodic and double that for C⋆) is significantly larger than the number of CUDA threads in the H100. As expected, since the local problem size stays large we observe solid weak scaling of the Dirac stencil, with still 80 % efficiency at 1024 GPUs. The good weak scaling makes the software stack ready for larger problem sizes.

TL;DR: Dop: weak scaling daint

For the strong and weak scaling runs, we did not utilize NVIDIA shared memory (NVSHMEM) because it was not available on the cluster. NVSHMEM is NVIDIAs implementation of the OpenSHMEM standard [174], a parallel programming model and library designed to facilitate efficient

TL;DR: Dop: no nvshmem

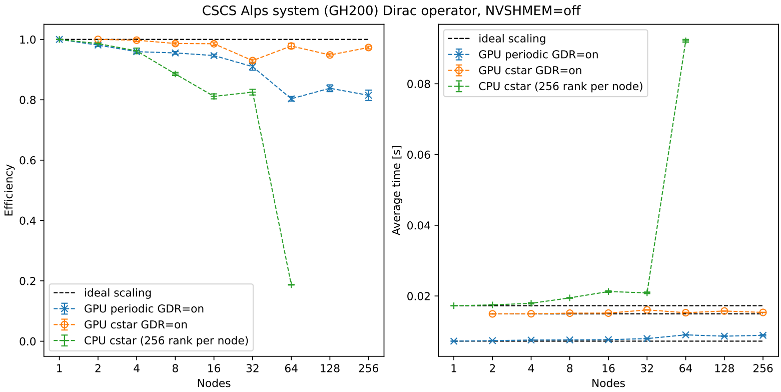


FIGURE 7.10: Weak scaling of Dirac stencil (left panel) and average time for one application (right panel). The operator was applied 10 000 times and the average was taken. The base case for the $C^\star$ lattice is two nodes for implementation restrictions. The physical local lattice size was $64 \times 32 \times 64 \times 64$ and $128 \times 32 \times 64 \times 32$ for the periodic and $C^\star$ lattice, respectively. Every data point consists of at least 5 independent runs. The data was taken on *Daint-Alps*, see section 7.1.

data sharing and communication between GPUs, both within a single node and across multiple nodes. NVIDIA promises less overhead resulting in more efficient strong scaling when using NVSHMEM [175]. However, the Cray MPI implementation was CUDA-aware, Peer to Peer access (P2P) from GPUs within the same node was utilized, and GPU-direct RDMA (GDR) was enabled to further remedy bad strong scaling behavior. Nevertheless, the error bars might be slightly underestimated because of temporal correlations among runs, especially for runs with high node counts.

7.8 SOLVER

TL;DR: solver intro

As opposed to the Dirac operator, the convergence rate of a Krylov solve *does* depend on the physics parameters and the background gauge field non-trivially. Therefore, we cannot present a thorough weak scaling study for the solver. For the discussion of strong scaling of the solver that follows, we took representative gauge field configurations and fixed physics parameters.

TL;DR: solver: strong scaling on daint

Figure 7.11 shows strong scaling of a production multigrid solver on a periodic and a $C^\star$ lattice. The relative residual is chosen to be 10^{-12} for all solves. We see very poor strong scaling above 16 to 32 nodes. This

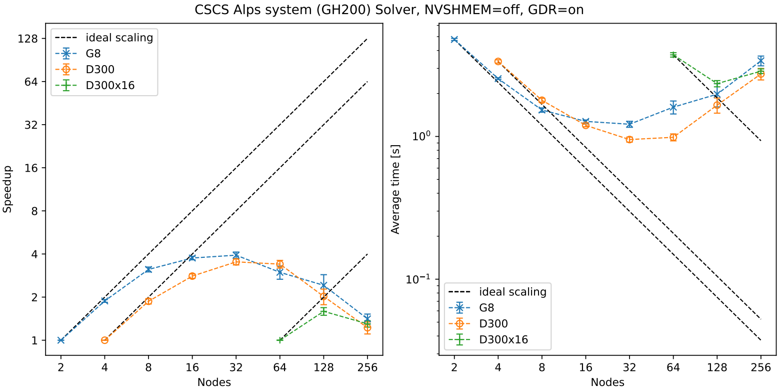


FIGURE 7.11: Strong scaling of a multigrid solver (left panel) and average time to solution for one solve (right panel). The base cases were chosen to be the smallest node count for the computation to fit in GPU memory. The data was taken on *Daint-Alps*, see section 7.1.

is expected because unfortunately multigrid preconditioning inherently possesses bad strong scaling behavior. This is due to the fact that multigrid coarsens the lattice into a smaller lattice with significantly fewer degrees of freedom and solves the Dirac equation on the coarse lattice in every iteration step. The repeated coarse lattice solves, which dominate the runtime, are by far not able to saturate the GPU especially for high node counts. Clearly, an unpreconditioned solver would show better strong scaling that is comparable to the Dirac operator itself (see fig. 7.9), but on ill-conditioned systems such a solver would take thousands of iteration steps or never converge at all. Multigrid is the state-of-the-art production solver for ill-conditioned systems in the field. Its time to solution is paramount with the trade-off of bad strong scaling.

For this reason, the plot shows a third (somewhat artificial) lattice D300x16. This lattice consists of periodically extended copies of D300 in all 4 spacetime directions by a factor of 2 respecting global boundary conditions, i. e., 16 times larger than the original D300. Such a lattice belongs to the largest ones appearing in the field (sometimes called monster lattice). D300x16 has physical lattice extents $T \times L^3 = 256 \times 128^3$. The fact that this lattice scales well up to 512 GPUs indicates that the code is ready for substantially larger problem sizes.

The discussion in the previous paragraph suggests that if we saturate the

TL;DR: solver: scaling with mrhs (weak) on daint

GPU by exposing more parallelizability on the lattices which are too small, we should observe better scaling behavior. This can be achieved by solving for multiple right-hand sides at the same time. Figure 7.12 shows a variant of a weak scaling study where not the local problem size was held constant but the computational work per node in terms of RHSs. As the number of nodes increases the number of RHSs increases proportionally, thus, the local workload stays constant as opposed to the local lattice size. The plot shows some interesting features worth mentioning. The super-linear efficiency for the small node counts comes from the better data locality and increased arithmetic intensity when processing more than one RHS at the same time. The improvement diminishes again quickly, since even though the local size of the RHSs stays constant, the local *problem* size – and by this the local operator size in memory – decreases inversely proportional to the node count. Processing multiple RHS turns matrix-vector into matrix-matrix multiplications, increasing the arithmetic intensity. In the plot, the matrix representing the RHSs stays constant in total size but its dimensions change, whereas the operator matrix size decreases. There are numerous competing concepts affecting the overall performance in this scenario. As the number of RHS increases, we expect better temporal data locality and cache reuse until batch sizes exceed cache or shared memory capacity clearly visible in the plot. Since the amount of local variables scales linear in the number of RHSs, we expect variables to the moved to slower memory resulting in register spilling [176] as the number RHSs increases. Finally, as the RHS matrix changes shape stride access pattern can cause data in cache to repeatedly being evicted resulting in cache thrashing [177].

7.9 SUMMARY

TL;DR: performance data conclusion

Wrapping up the performance chapter, we conclude that on a cluster with GH200 nodes with lattices currently accessible to us, we can utilize the resources of the node efficiently. However, performing runs with more than 32 GH200 nodes is not advisable because we are clearly in the bad strong scaling region then unless we study larger lattices as the ones considered or increase the number of RHSs substantially. In a field where the total problem size naturally grows with the available compute power, good weak scaling makes our software stack future-proof. The fact that the largest lattice considered strong scales up to 512 GPUs indicates that the code is ready for the exascale.

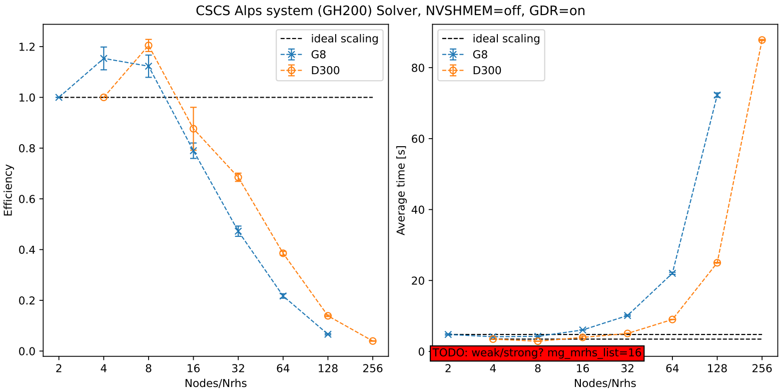


FIGURE 7.12: Variant of weak scaling of a multigrid solver w.r.t. number of RHSs (left panel) and average time to solution for all RHSs (right panel). The base cases were chosen to be the smallest node count for the computation to fit in GPU memory. The data was taken on *Daint-Alps*, see section 7.1. **TODO**: remove red from plot

Computations on the CPU can efficiently be overlapped with concurrent solves on the GPU if power supply of the machine allows it as proven on *Leonardo Booster*. Finally, for maximizing node utilization of CPU workloads, it can conveniently be scaled out within the node by using the dual process grid adding negligible performance overhead.

The inherent challenging strong scaling of multigrid is not a big problem for multiple reasons. First, we now have many tools available to mitigate it, such as overlapping multigrid solves with computations on the CPU or solving an array of RHSs in one batch. Second, in reality the only relevant metric usually boils down to time or energy to solution.

The data still shows some non-negligible contributions of H2D and D2H transfers (red and green in fig. 7.3). In future developments, the H2D transfers could be eliminated to a large extent by setting up the source vector directly on the GPU, especially when dealing with random or point sources. Instead of handing over the field data, we hand over a description of the field instead. Field transfers in general can be reduced by employing a memory management system keeping track of field status. A system minimizing CPU-GPU traffic will be introduced in chapter 9.

7.9.1 *Limitations*

Evaluating observables is only one part of computational lattice field theory workflow. Gauge field generation is the other major part. This is an inherently serial process insofar that a Markov chain of independent gauge field configurations has to be generated. This requires few solves of the Dirac equation with a frequently changing background gauge field. With the current interface gauge field generation is not efficient for multiple reasons:

- As long as the gauge field is generated on the CPU, we have to transfer it every time it changes.
- Few solves imply few RHSs, reducing our possibilities to performance tune. Thus, the effect on performance of blocked solvers is degraded.
- Few RHSs cause H2D/D2H transfers of gauge and clover fields to contribute more to the overall solve.
- To achieve reasonable throughput of generated gauge fields, it is common in the field to scale out. This requires good strong scaling behavior.

8

AUTOMATING BUILD AND DEPLOYMENT WITH CI/CD

> *If debugging is the process of removing software bugs, then programming must be the process of putting them in.*
>
> — Edsger W. Dijkstra

This section/chapter was proofread 1 times.

TL;DR: CICD intro, git private

This chapter describes the continuous integration testing and deployment (CI/CD) pipeline as introduced in openQxD's development repository [119]. At the time of writing this thesis the repository is private. As part of the goals of the PASC project [178] and thus as part of this thesis, we instantiated CI/CD paradigms to the development workflow of openQxD. This chapter details our strategy for continuous integration (section 8.1), continuous testing directly on target hardware at CSCS (section 8.2), software deployment to CSCS infrastructure (section 8.3) and finishes with a summary (section 8.4).

8.1 CONTINUOUS INTEGRATION

TL;DR: CI strat, everything via merge-reqs, review, test

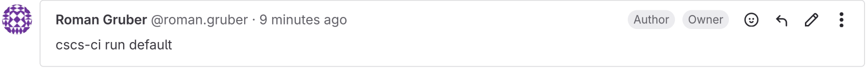
New features, bug-fixes or code changes enter the codebase by means of a separate branch and a corresponding merge-request, which is reviewed, tested and finally merged to the master branch. The master branch stays clean and functional, such that at every point in time one can release a new version of the software by just taking the newest commit to the master branch as version tag.

8.2 CONTINUOUS TESTING

TL;DR: testing gitlab+CSCS cronjob + on every merge-req

We decided on a testing strategy as follows:

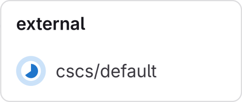
- For continuous testing, the master branch is tested on a daily basis overnight at 1 am (CET) on local resources at ETH. This runs the GitLab pipeline described in section 8.2.1.



Roman Gruber @roman.gruber · 9 minutes ago

cscs-ci run default

FIGURE 8.1: The comment to add to a pull-request that starts the CSCS default pipeline.



external

cscs/default

FIGURE 8.2: The job in the external stage that represents the CSCS default pipeline. One can click on the job to be redirected to CSCS servers where the whole pipeline including status of the jobs is visible.

- For continuous testing, the master branch is tested every 3 months on GH200 machines at CSCS. This runs the CSCS pipeline described in section 8.2.2.
- For continuous integration, every commit into every open merge-request is automatically tested on local resources at ETH. This runs the GitLab pipeline described in section 8.2.1.
- For continuous integration, open merge-requests can be tested by triggering the CSCS pipeline manually. This runs the CSCS pipeline described in section 8.2.2. The pipeline can be trigger by commenting on any merge-request with `cscs-ci run default`, see fig. 8.1. This will invoke a webhook that triggers CSCS services to pull the repository and run the pipeline. A short time after commenting, a new job appears in the pipeline view with stage external, see fig. 8.2.

TL;DR: 2 pipelines: gitlab + CSCS

Thus there are two separate pipelines. One that runs on via GitLab on local resources at ETH, and one that runs via CSCS on GH200 nodes at CSCS. Both pipelines are independent and completely detached from each other, they may run in parallel or not.

8.2.1 *GitLab pipeline*

TL;DR: ETH runner, GPU required but not for all jobs

The entrypoint for the local GitLab pipeline is in the openQxD-devel repository [119] under `.gitlab-ci.yml` directly in the root. Parts of this pipeline may run on official GitLab runners (if time is available). The test jobs require runners to expose one or more local NVIDIA GPUs to the docker image to be able to run CUDA code. Clearly the official GitLab runners do not provide GPUs. We registered a local machine at ETH which specifications

- 2x AMD EPYC 9124 16-Core Processor
- 8x NVIDIA GeForce A2000 (12 GB, CUDA-architecture `sm_86`)
- 1.5 TB main memory

as an additional runner for this pipeline. The machine has 8 GPUs which makes it ideal for various testing scenarios, like lattices with $C^\star$-boundaries in all 3 spatial directions which require a minimum of 8 ranks.

The pipeline consists of three stages; prepare, build and test. In general the stages depends on each other, although there are jobs in the build stage which depend on nothing. The individual stages will be discussed separately in the following.

TL;DR: Split into stages

The prepare stage

All jobs in the prepare stage may run on official GitLab runners. As the name suggests, this stage has some preparatory jobs, whose artifacts will be exposed to subsequent jobs. The job called `download-quda` downloads a copy of QUDA from its official GitHub repository [179] via `git clone` and exposes it as artifact. We download the latest developments committed to the `develop` branch, i. e. their CI branch. All subsequent tests that depend on this job, will use the newest available developments of QUDA. Therefore, we will notice breaking changes immediately.

TL;DR: prep stage, can run everywhere, artifacts, qudas develop branch

The build stage

All jobs in the build stage may run on official GitLab runners. However, this stage consists of a matrix of jobs that compile openQxD with various combinations of compilers and MPI implementations. This can be easily extended to cover missing combinations. At the time of writing, we cover compilers gcc-10, gcc-11, gcc-12, gcc-13, gcc-14, clang-18 and two MPI libraries openMPI [180] and MPICH [181]. These compile-jobs search through the whole repository. In every directory they find a `Makefile`, they will call `make`. This makes sure that all programs and modules in the project are compiled.

TL;DR: build, runs everywhere, matrix of MPI+CC, job name pattern

ensemble	$L_0 \times L^3$	m_π [MeV]	boundary conditions	# configs
D5d	48×24^3	439	periodic	2
A360	64×32^3	360	$C^\star$ in xyz directions	2

TABLE 8.1: Ensembles used in the CI/CD pipeline. The periodic lattice D5d is taken from ref. [120], whereas the $C^\star$ lattice is taken from ref. [112].

These jobs do not depend on any other job and will fail if one of the `make` commands fails. Their job names have the pattern

```
build-openqxd/<mpi>:<version>-<compiler>:<version>
```

where `<mpi>` is replaced by either `openmpi` or `mpich` and `<compiler>` with `gcc` or `clang` and their respective versions.

TL;DR: depend on prep/build, build quda jobs

The remaining build jobs may depend on the prepare stage or on other build jobs. The jobs with pattern

```
build-quda/cuda:<version>-<cuda_arch>
```

build QUDA for the CUDA architecture given by `<cuda_arch>` using certain CUDA versions given by `<version>`. They can also easily be extended to cover more versions and architectures.

TL;DR: build openqxd vs quda, their deps, BCs, lattices

Finally the jobs to build openQxD against QUDA have the pattern

```
build-openqxd/quda-checks/<cuda_arch>-<bcs>
```

They depend on the previous build jobs that build QUDA with the corresponding `<cuda_arch>`. The placeholder `<bcs>` is substituted with the boundary conditions of the lattice we want to test against. Currently we run tests on two lattices, see table 8.1. Thus we compile openQxD on a process grid of 1x2x2x2 with 8 ranks for these two lattices. Again this can be extended to other lattices.

The test stage

TL;DR: deps, name pattern of test jobs, test binaries discussion

All jobs in the test stage may ***not*** *run on official GitLab runners, they require a GPU and thus may only run on the local resource specified above or on a registered runner exposing a GPU.* Jobs in this stage depend on the build jobs that build openQxD against QUDA discussed at the end of the previous section. They may not run if previous jobs in the dependency chain failed. They have names as

```
test-openqxd/quda-check<N>/<cuda_arch>-<bcs>
```

where `<N>` is the number of the check we run, i. e. the check corresponds to `devel/quda/check<N>.c` which was built in the previous stage. We briefly discuss the binaries:

- `devel/quda/check1`: Checks the indexing functions of fields, i. e. the arrays `ipt[VOLUME]` and `iup[VOLUME][4]` (see `main/README.global`) against the pure function implementations in QUDA, `openQCD_qudaIndexIpt` and `openQCD_qudaIndexIup` respectively. Afterwards the transfers of the gauge field to QUDA and back are performed and compared two each other. Finally the plaquette on QUDA and on openQxD are computed and compared. This last test is done only with the $SU(3)$-valued gauge field, even if a QCD+QED lattice is given as input, since QUDA is not capable of calculating the QCD+QED plaquette.

- `devel/quda/check2`: This check performs a host-to-device and device-to-host transfer of a spinor field and compares the two on the host. Further the norm function as well as the gamma matrix applications are compared to make sure the gamma basis is the same. Finally we compare the Wilson-Clover Dirac operator by acting on a random spinor field on both openQxD and QUDA and compare them.

- `devel/quda/check3`: We have implemented acceptance tests of solvers in QUDA in this check. This means we call the solver in QUDA according to its configuration from the input file provided to the executable. The relative residual of the yielded solution is checked explicitely against the Dirac operator in openQxD and compared to the desired relative residual from the solver configuration. This tests the synchronous solver interface from section 6.2.1 with multiple solver instances, one and multiple RHSs, with and without the dual process grid, all on a periodic and a $C^\star$ lattice.

- `devel/quda/check4`: Consists of a check of the eigensolver of QUDA. This is currently not tested.

- `devel/quda/check5`: This check consists of acceptance tests for the asynchronous solver interface from section 6.2.4. It performs the same tests as `check3` but with the asynchronous solver.

- `extras/main/lowrnk/pbp`: This utility serves as an example on how to access the QUDA solver via the traditional solver interface described in section 6.2.2 and accepts solver configurations from openQxD as well. This test verifies if both types of solvers can coexit properly.

TL;DR: solver checks, multiple gauges, flavors, periodic, cstar, tracking mech

For solver checks, `check3` and `check5`, we decided to verify certain interface functionalities apart from a correct solution. We run on 2 gauge configurations, 2 different flavors, 3 different solvers and 2 random solves each. The input files can be found under `ci/input/*.in` in [119]. That implies 24 actual calls to a solver in QUDA under varying internal states. We test multiple gauge configs and flavors to verify if the interface properly updates gauge and clover fields and is aware of changing parameters due to changing flavors. For instance, if on a QCD+QED $C^\star$-lattice we change the value of the inverse charge, `qhat`, we have to retransfer the gauge field to QUDA, because $\hat{q}$ appears in the U(1)-phase of the U(3)-valued gauge field, see eq. (1.69). This has to be triggered automatically by the interface code when changing parameters (see section 5.1.5).

TL;DR: Why 3 solvers

Additionally, we test 3 different solvers. One is a multigrid solver solving a single RHS similar to one that would be used in a production run. The second solver is also a multigrid solver solving two RHSs to check if multiple different instances of multigrid do not interfere with each other when switching from one to the other. Finally, a BiCGSTAB solver without any preconditioning is tested to check if other types of solvers can be interfaced successfully.

TL;DR: where are the configs?

In order to test on multiple gauge configurations, we need them available in the test jobs. This is achieved by enabling the package registry on GitLab and uploading the gauge configs as `tar.gz` archives. These can be found in the GitLab project under `Deploy -> Package Registry`. The test jobs download the archive corresponding to the lattice they run on, verify the MD5 sums, unpack and run the binaries.

8.2.2 *CSCS pipeline*

TL;DR: runs on GH200 at CSCS, costs, list of stages

This pipeline may only run on the official GH200 nodes at CSCS and is associated with an account having compute quota on the Daint Alps vCluster. *Running this pipeline consumes nodehours of the associated account.* The entrypoint for this pipeline is in the openQxD-devel repository [119] under `ci/cscs_default_pipeline.yml`. It consists of two stages; build and test.

The build stage

In this stage, we build and install the newest develop version of QUDA using Spack [182] directly into a so called uenv [183] image. For this to work, we need QUDA as a Spack package, see appendix A.2. Using this machinery, the descriptive build job for QUDA is very simple:

TL;DR: build uenv, spack package, descriptive

```yaml
build-quda/uenv/daint-gh200:
  stage: build
  extends: .uenv-builder-daint-gh200
  variables:
    UENV_RECIPE: ci/uenv-recipes/quda/daint-gh200
```

We see that it simply extends the runner `.uenv-builder-daint-gh200` provided by CSCS and included a few lines above. We also include a local file for global definitions, `ci/include/cscs/00-variables.yml`. Together with the global definitions we have some important variables that change the behavior of the aforementioned job:

- `UENV_NAME`: specifies the name of the uenv image we want to build.
- `UENV_VERSION`: specifies the version of the uenv image we want to build.
- `UENV_RECIPE`: points to a uenv recipe.

The uenv recipe is a descriptive way on how to specify an environment on CSCS compute and login nodes in terms of its software dependencies. It consists of a directory with several YAML-files:

TL;DR: uenv recipe description

- `config.yaml`: Description of the package, its name, brief description in a sentence and which Spack version to use.
- `compilers.yaml`: Description of the compilers for bootstrapping and the compiler to use when compiling all software and dependencies. This works as follows: the system compiler (usually a very old gcc) is used to compile the bootstrap compiler (usually a newer gcc). Then the bootstrap compiler is used to compile the compiler we want to use to compile our software stack (usually a recent gcc).
- `environments.yaml`: Description of the environment, i. e. `gcc-env.specs` list all Spack specs that should be compiled and installed, `gcc-env.mpi` is dictated by CSCS to be `cray-mpich` with CUDA enabled (CUDA-aware MPI), `gcc-env.variants` adds Spack variant modifiers to

all packages that support them in the spec list. `gcc-env.views` configures the available views in the image. The view `spack` is always added to the image. The view `modules` is added if in `config.yaml` we set `modules: true`. The `openqxd` view is added in the `post-install` script. The specs contain a spec for QUDA as

```
quda@experimental +openqcd +multigrid +wilson +clover
```

which builds and installs QUDA with the interface code. This is currently installed using the additional package description under `repo/` describing the QUDA build and install procedures.

- `modules.yaml`: Description of the `modules` view if it was enabled in `config.yaml`. This is for legacy behavior if we want to generate Tcl [184] or Lmod [185] modules from the software given in the `gcc-env.specs` in `environments.yaml`. These can be loaded once the uenv is started.
- `post-install`: A post installation script that currently adds the `openqxd` view which specifies the environment variables required to build openQxD.
- `repo/`: An additional repo path to add to Spack. Under this path currently lies the QUDA package description until it is available in the official Spack repositories.

TL;DR: summary + transition to test stage

However, this builds a uenv image containing QUDA built for the GH200 nodes with CUDA architecture `sm_90` using the correct MPI implementation provided by CSCS. If this stage was successful, the test stage is being executed.

The test stage

TL;DR: test stage, same checks as gitlab pipeline, GH200 specs

Analogue to the GitLab pipeline (section 8.2.1) we run the same checks using the same lattices (table 8.1) and input files. All that differs is that we run all checks sequentially in one job. We do not discuss the individual checks again and refer the reader to section 8.2.1 for more details about them. All tests run on 2 GH200 nodes.

8.3 CONTINUOUS DEPLOYMENT

TL;DR: CD CSCS pipeline, comment triggers

For continuous deployment, open merge-requests can be deployed to CSCS infrastructure by triggering the CSCS deploy pipeline manually. This runs the CSCS deploy pipeline described in section 8.3.1. The pipeline can be triggered by commenting on a merge-request with

```
cscs-ci run deploy
```

in the same way as described in section 8.2 for the default pipeline.

8.3.1 *CSCS deploy pipeline*

The entrypoint for this pipeline is in the openQxD-devel repository [119] under `ci/cscs_deploy_pipeline.yml`. *One should only run this pipeline when the default pipeline finished successfully.* The pipeline consists of only one stage; deploy.

TL;DR: entrypoint, deploy stage

The deploy stage

This stage currently consists of only one job that deploys the previously built uenv image to the service namespace of the CSCS uenv registry. When deployed, a user on Daint Alps can just pull and start the image. One will find an already compiled version of QUDA with all required environment variables set to build openQxD against QUDA, see appendix A.3 for more details on building openQxD and listing A.1 for the required environment variables and appendix A.4 for more information on how to run the uenv.

TL;DR: deploy stage, put uenv into cscs registry

8.4 SUMMARY

All parts of the interface are continuously tested on a daily basis. Testing on GH200 nodes at CSCS can manually be triggered, conveniently as comments in merge-requests. Occurring software bugs or compiler issues can be identified directly when they appear and future developments will not enter the codebase unless all tests are successful. Testing directly on target infrastructure increases quality, robustness, stability and trust in the code substantially while guaranteeing functionality. The direct deployment to CSCS user environment registries simplifies the process of setup, compile and run in a production environment, since the steps of setting up the correct environment and properly compiling the code essentially vanish from the workflow. Deploying the whole software environment rather than just the code considerably increases reproducibility of simulations.

9

PROPOSAL FOR FIELD MEMORY MANAGEMENT IN HETEROGENEOUS ARCHITECTURES

This section/chapter was proofread 1 times.

TL;DR: mm intro

In future developments, we plan to unify spinor fields among openQxD and QUDA. This chpater describes a proposal for a memory management system for fields that does not come with tremendous implementation efforts, providing backward compatibility for already written programs and a framework to write programs utilizing the GPU with little porting effort, maximal flexibility for the developer and minimizing CPU-GPU traffic. It starts by motivating the need for such a system (section 9.1) with explicit example codes and continues with a field unification scheme (section 9.2). For minimal code refactoring on the side of openQxD, we propose an (optional) procedure to automatically transfer fields from the GPU to CPU (section 9.3) using posix mechanisms, before concluding the chapter (section 9.4).

TL;DR: keep track of state on field level

The key idea is to not deal with fields on the CPU separately from fields on the GPU. The system keeps track of where each field currently resides (CPU, GPU or both), where it was last updated (CPU or GPU) and implicitly transfer the field if a function reads or writes to it. Such a system should be backward compatible with the memory layout of openQxD. Using C macros, we may guide the compiler to the GPU-variants of certain functions that overwrite their CPU equivalents.

TL;DR: legacy coding possible

This will enable us to work in the framework of openQxD in the same way as before and at the same time use GPU features. Existing software with minimal adjustments will benefit from such a memory management system too.

9.1 MOTIVATION

TL;DR: intermediate phase, minimize unneeded transfers, porting of functions

OpenQxD is a pure MPI code that runs efficiently on CPUs. The transition to utilize accelerators will be a long running intermediate phase, where parts of the code will run on the CPU and others on the GPU. We are mostly concerned with this intermediate phase in this chapter. More and more linear algebra kernels will be ported to the GPU throughout this phase. This requires a flexible system that deals with transferring and reordering the fields appropriately. We want to avoid transferring in every single function. Some transfers might be unnecessary, while others are crucial for code consistency.

TL;DR: reordering expensive, gamma bases, hard-coded gamma basis

One challenge is that the two applications have vastly different memory layouts for fields and reordering has to be performed from one to the other, see chapter 5. A second problem is the different γ basis conventions that require data transformation. When transferring and reordering a field to QUDA the γ-matrix basis is changed to QUDAs internal one via basis transformation. This even mixes up individual values of the fields and therefore excludes useful technologies like unified memory, unless one drastically changes the memory layout of all field types and the γ-matrix basis in openQxD. This is not realistic, since γ-matrices are hard-coded in most of the performance-critical kernels, like the Dirac operator.

TL;DR: introduce example program

We will start describing the memory management system using a simple example program (that program has no special meaning apart from serving as a instructive example). Consider listing 9.1, where we want to emphasize the highlighted lines, which we assume in the following as functions ported to GPU, whereas the remaining non-highlighted functions still run on the CPU.

```c
int main(int argc, char *argv[]) {
   spinor_dble **wsd;
   int N = 10;

   [...]

   alloc_wsd(N+1);        // Allocate N+1 spinor fields
   sd = reserve_wsd(N+1); // Reserve them; sd[i] holds pointer to i-th field

   for (i=0; i<N; i++) {
      random_sd(VOLUME, sd[i], 1.0);          // Randomly initialize field
      scale_dble(VOLUME, 3.0, sd[i]);         // Scale field by factor 3
      Dw_dble(0.0, sd[i], sd[N]);             // Apply Dirac operator to first
      Dw_dble(0.0, sd[N], sd[i]);             // field and store to second
      r = norm_square_dble(VOLUME, 1, sd[i]); // Take the norm squared
   }

   release_wsd();
}
```

LISTING 9.1: An example program written in openQxD.

TL;DR: Dop overload to GPU just as AVX/SSE versions

Therefore, the Dirac operator, `Dw_dble()`, is a function that takes fields in openQxD and transfers the input field to the GPU, calls the Dirac operator in QUDA and transfers the output field back. This type of function overloading already finds practice in openQxD for functions that employ AVX or SSE instructions. It is achieved with compile time macros and C preprocessor directives, see listing 9.2. The snippet shows three implementations of the same function; a default implementation which is selected if no compile time flag is set, an implementation using AVX instructions and an implementation wrapping the GPU function call via QUDA's interface API. All of them act on regular spinor fields in openQxD. At this stage, the GPU implementation has to transfer field(s) back and forth.

```c
#if (defined AVX)
// implementation using AVX intrinsics
functionA(spinor_dble *s) { ... }
#elif (defined GPU_OFFLOADING)
// GPU overloading of the function
functionA(spinor_dble *s) { ... }
#else
// default implementation without intrinsics
functionA(spinor_dble *s) { ... }
#endif
```

LISTING 9.2: Different implementations of the same example function `functionA()`.

We now go back to the example program, listing 9.1, and imagine the highlighted functions to be offloaded as in listing 9.2. This poses a problem, because the field `sd[N]` will be downloaded from the GPU in line 13 and uploaded again in line 14, even though the field is never processed on the CPU. In the next section, we propose a simple solution for this problem.

TL;DR: problems in example program with unneeded transfers

Finally, we want to state requirements of the memory management system in descending priority:

TL;DR: Statement of requirements

(R1) Fully *backwards compatible* with the memory layout of openQxD.

(R2) No changes in *existing programs* written in openQxD. No rewrite / adjustment of existing programs, if a new function is ported to GPU.

(R3) No changes in *existing functions* that operate on spinor fields on the CPU.

9.2 UNIFIED FIELDS

The first step will be a unification of fields. We want to establish a one-to-

TL;DR: unify field, state attached to every field

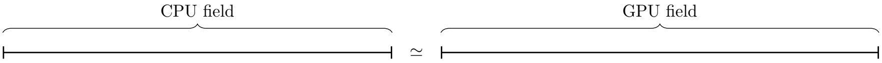

CPU field
GPU field
≃

Figure 9.1: Each field in openQxD corresponds to a field in QUDA.

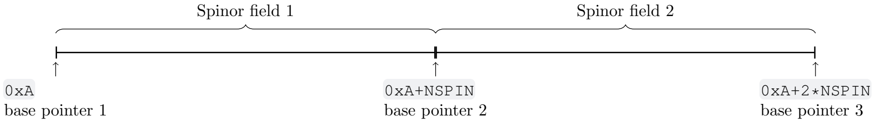

Spinor field 1
Spinor field 2
0xA
base pointer 1
0xA+NSPIN
base pointer 2
0xA+2*NSPIN
base pointer 3

Figure 9.2: Current field allocation scheme.

one correspondence between CPU and GPU fields as depicted in fig. 9.1. Every time a field is (de-)allocated on the CPU, we (de-)construct the GPU field. This can be done lazily, i. e. the actual object and data allocation for the GPU field may happen as soon as the field hits the GPU. By this, a field living on the CPU only occupies no memory on the GPU. Our main concern now is to maintain consistency among the data, if the field is being written by the CPU or the GPU. Thus, we equip every field on the CPU with some meta information about its state.

TL;DR: spinors field allocation scheme in openqxd

In openQxD, spinor fields are large arrays of structs as illustrated in fig. 9.2. Allocating multiple fields with `alloc_wsd()` places the fields consecutive in memory corresponding to a single call to `malloc()`.

TL;DR: How to alter

We propose to alter the current allocation scheme for fields as in fig. 9.3. We stick a struct holding information called `spinor_info` at the end of every spinor field. We spaced them out a bit, but nevertheless every field pointer still points to the first struct in the array of structs which can be passed into legacy functions; no violation of requirement (R1).

TL;DR: what information? status, pointer to GPU and basic mechanism

The `spinor_info` struct holds information about the field such as

- A void-pointer to the field on the GPU that points to the object instance in QUDA representing the field.

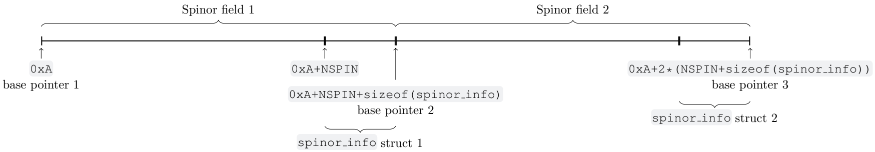

Spinor field 1
Spinor field 2
0xA
base pointer 1
0xA+NSPIN
0xA+NSPIN+sizeof(spinor_info)
base pointer 2
spinor_info struct 1
0xA+2*(NSPIN+sizeof(spinor_info))
base pointer 3
spinor_info struct 2

Figure 9.3: Proposed field allocation scheme.

- A status flag indicating whether the field is in sync between CPU and GPU (both hold the same data), the field on the CPU is newer (CPU code has manipulated the field, such that the GPU field is outdated), or the other way around. Here we have to be careful in a potential implementation. A field that is not in one of the three mentioned states should never occur. This would lead to inconsistency.

- Other useful meta information, such as field size in bytes, precision, number of times the field was transferred for optimization purposes, etc.

For an implementation of the above field unification proposal only field allocation, deallocation, reservation and release functions need to be altered[1]. Furthermore, all legacy openQxD function still work and operate on the base pointers in the same way as before. Functions that are offloaded to the GPU will take the same CPU base-pointers as the legacy function. They will navigate to the `spinor_info` struct by pointer arithmetic, check if the field needs to be transferred and do so if required. To operate on the field, they grab the GPU-field pointer. Initially this will be a NULL-pointer indicating the field on the GPU is not instantiated yet, triggering instantiation and memory allocation. Finally they will update the `spinor_info` struct according to how they process the field (read-only, write-only or read-write).

TL;DR: currently we need explicit transfers

In the current stage of the memory management system, the GPU needs to explicitly transfer fields back to the CPU if it has changed them, such that the next CPU function operates on the updated values honoring requirements (R2) and (R3). In the next section, we will introduce a mechanism to loosen this requirement of explicit transfers while still respecting all requirements.

9.3 IMPLICIT TRANSFERS

TL;DR: dilemma of implicit transfer + transition to concepts

When loosening the requirement mentioned in the previous section, we are posed with two problems: 1) every function within openQxD that operates on spinor fields has to transfer fields if needed by the `spinor_info` struct and 2) every function within openQxD that operates on spinor fields has to maintain information held by the `spinor_info` struct. According to requirement (R3) we are not supposed to touch any function implemented in openQxD. This might sound challenging at first sight, but a little creative thinking brings us to a solution using powerful operating system mechanisms. For this we need to introduce two concepts.

---

[1] Explicitly, these are `alloc_wsd()`, `reserve_wsd()`, `release_wsd()` and their single precision variants.

9.3.1 *Memory protection*

TL;DR: mprotect, posix, OS mechanism, page-alignment

Posix-compliant operating systems offer system calls for protecting a region of previously allocated memory. The region must be page-aligned, meaning that the pointer pointing to the begin of the region has to be the beginning of a new page and the region has to end at the end of a page, i. e. the base pointer address and the size of the region in bytes have to be multiples of the page size[2]. Allocating page-aligned memory cannot be done using the usual portable standard C library wrapper for dynamic memory allocation, `malloc()` (which allocates 8-byte aligned memory) but rather `posix_memalign()` or `mmap()`. However, when accessing a protected region of memory (by dereferencing a pointer) the operating system behaves as if we have never allocated the memory, i. e. it triggers a segmentation fault.

TL;DR: Why mprotect exists

One might ask what is such a mechanism good for? When it comes to data security, operating systems have to offer mechanisms to prevent untrusted third party libraries to access sensitive data such as passwords that might lie in memory. Such regions can be protected and if a third party library attempts to access them a segmentation fault will be thrown.

9.3.2 *Signals*

TL;DR: IPC, signals, catch signal, segfault catchable, register handler

The second ingredient is the posix-standardized inter process communication (IPC) layer called signals. It enables processes to send and receive signals from and to other processes in the same node. The behavior upon receiving a signal can be changed to be either the default behavior of the signal, ignore the signal or catch the signal by defining a signal handler. This is a user defined function with signature

```c
void handler(int signo, siginfo_t *info, void *ucontext);
```

that processes the signal. Not all signals are catchable. The segmentation fault (`SIGSEGV`) with signal number 11 is a catchable signal, meaning we can register a signal handler and process an occurring segmentation fault

---

[2] On most modern operating system the page size is 4096 bytes, if unknown it can be queried using the standard C library function `getpagesize()`.

0xA
0xA+NSPIN

FIGURE 9.4: A spinor field in memory.

bringing the program back into a consistent state. A signal handler can be registered with the code snippet listing 9.3.

```c
#include <signal.h>

struct sigaction action;
action.sa_sigaction = handler; // function name of the singal handler
action.sa_flags = SA_SIGINFO | SA_RESTART;
sigaction(SIGSEGV, &action, NULL);
```

LISTING 9.3: Registering a signal handler.

This will register a signal handler that receives additional information triggered by the `SA_SIGINFO` flag from the `siginfo_t` struct, namely the offending pointer address in `info->si_addr` among many others. We will need this information later. The flag `SA_RESTART` will continue the code by restarting the system call which caused the segmentation fault after the signal handler has returned.

9.3.3 *Implicit transfers through memory protection and signals*

We now have everything to assemble a scheme to trigger implicit field transfers for us. We may again assume to have a spinor field in memory as depicted in fig. 9.4. At this stage we assume the base pointer to be page-aligned and the size of the allocation to be a multiple of the page size as discussed in section 9.3.1. Furthermore, we assume the GPU has altered its copy of the field and the `spinor_info` struct indicates the CPU field to be outdated. Therefore, we put a memory protection over the field, fig. 9.5. Now consider entering a legacy function on the CPU with this field. The function will access the data at some memory address in the protected region, fig. 9.6. Ultimately this will trigger a segmentation fault and we enter the previously registered signal handler.

TL;DR: Engineering the scheme to force entering signal handler

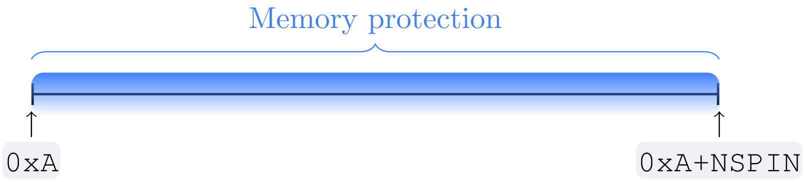


Figure 9.5: A protected spinor field in memory visualized with a blue shield.

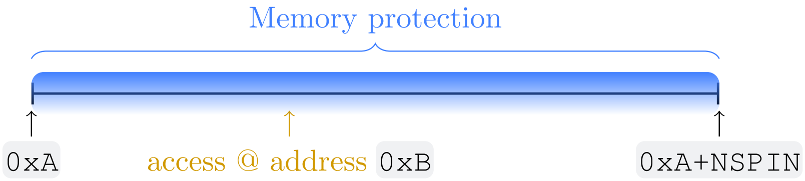


Figure 9.6: Accessing data at memory address `0xB`.

TL;DR: clean recovery, inspect offending address, transfer, ... updated field

What remains to do is properly process the signal and recover the application state before the signal happened but with updated field values. The signal handler will check if the segmentation fault was triggered by an actual memory access violation or by a memory protection violation. If it is an actual segmentation fault, we continue by raising it. This can be determined by looking at the offending pointer address which is given as payload to the signal, see section 9.3.2. If the address is in between the begin and end of a protected region, we know it is a memory protection violation, else it is a segmentation fault. However in the former case, we proceed by first removing the protection and to navigate to the `spinor_info` struct, check if the field is up-to-date, transfer if necessary, update the `spinor_info` struct accordingly such that the next GPU function that operates on the field will trigger a transfer and finally exit the signal handler gracefully. This will restart the instruction that caused the segmentation fault which will be either a load or store from the openQxD function, but we removed the protection and updated the field values. Thus the legacy function will continue processing, now with the updated field values. Note that the mode of how the field values are accessed within a legacy function does not matter, be it AVX or SSE intrinsics or dereferencing some pointer in the middle of the region, it will always trigger the segmentation fault and the procedure described above.

9.3.4 *State diagram*

For the sake of completeness, the lifetime of such a field can be modeled as a finite-state machine whose state diagram is depicted in fig. 9.7. We note that with this scheme, we cannot distinguish between a potential read or write in a segmentation fault on the CPU implying some unnecessary transfers only happening for very rare code patterns which we consider negligible. With this we have four different actions caused by the triggers:

TL;DR: state diag of field, read/write indistinguishable

1. *CPU r/w/rw*: As stated before, reads and writes cannot be distinguished on the CPU. The action they trigger is the following. Memory protection is removed if in place, the field is transferred to the CPU if and only if the old state was `GPU_NEWER` and the state is changed to `CPU_NEWER` no matter what it was before.
2. *GPU r*: If the field on the GPU is not allocated yet (pointer is `nullptr`), allocate it, transfer the field if and only if the old state was `CPU_NEWER` (if enabled, disable memory protection for this), enable memory protection if not enabled yet and change the state to `IN_SYNC`.
3. *GPU w*: If the field on the GPU is not allocated yet (pointer is `nullptr`), allocate it, enable memory protection if not enabled yet and change the state to `GPU_NEWER`.
4. *GPU rw*: If the field on the GPU is not allocated yet (pointer is `nullptr`), allocate it, transfer the field if and only if the old state was `CPU_NEWER` (if enabled, disable memory protection for this), enable memory protection if not enabled yet and change the state to `GPU_NEWER`.

9.4 SUMMARY

We proposed a memory management system for spinor fields that automatically keeps track of field states, unifies CPU and GPU fields and transfers field data from CPU to GPU and vice versa only if necessary. We require no modification of legacy openQxD functions. Only the field allocation and deallocation functions need to be touched. This is a minimal-effort-maximal-outcome approach that allows openQxD developers to continue working the way they are used to, while having the benefit of GPU offloaded functionality. We also emphasize that this solution is meant for an intermediate

TL;DR: summary, approach, also allocation changes

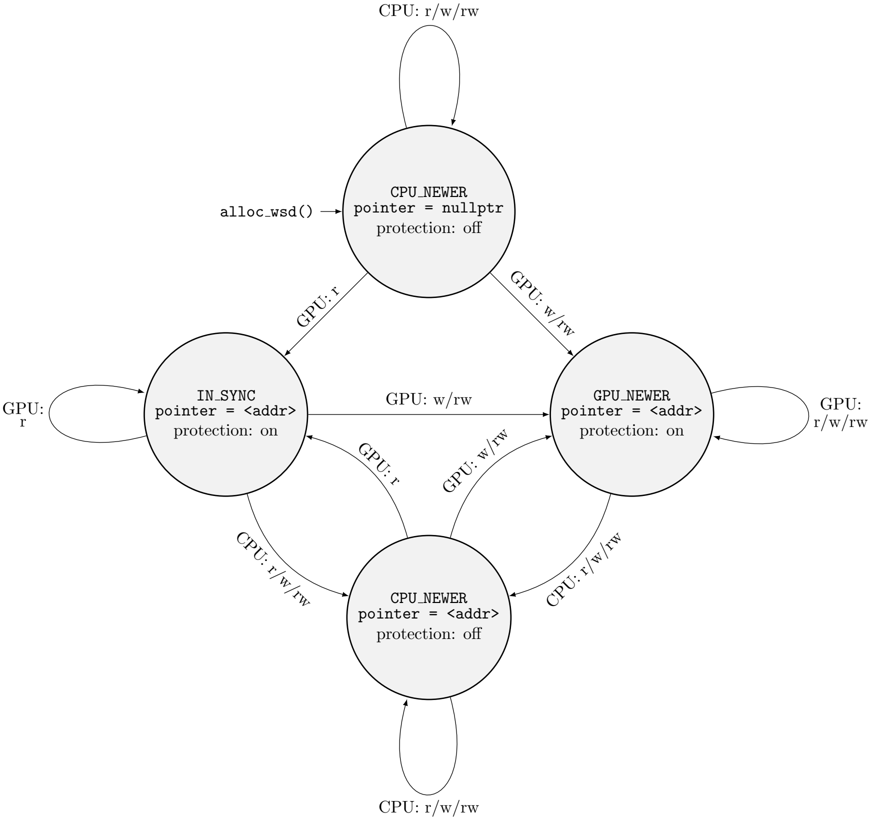

CPU: r/w/rw
CPU_NEWER
pointer = nullptr
protection: off
alloc_wsd()
GPU: r
GPU: w/rw
GPU: r
IN_SYNC
pointer = <addr>
protection: on
GPU: w/rw
GPU_NEWER
pointer = <addr>
protection: on
GPU: r/w/rw
GPU: r
GPU: w/rw
CPU: r/w/rw
CPU: r/w/rw
CPU_NEWER
pointer = <addr>
protection: off
CPU: r/w/rw

Figure 9.7: The state diagram of a spinor field as explained in the main text. There are three main states; `IN_SYNC` indicating that the field values are equal on CPU and GPU, `CPU_NEWER` and `GPU_NEWER` indicating that the CPU/GPU has changed the field and the other one is outdated respectively. The key indicated as *pointer* denotes the address of the field on the GPU initialized to `nullptr`. The protection states whether the memory on the CPU is protected according to section 9.3.1. The arrows denote triggers that cause actions to change the state of the field. There are three different triggers: read-only (*r*), write-only (*w*) and read-and-write (*rw*) on two devices: CPU and GPU. For instance an arrow with text *GPU: w/rw* indicates that the state change is triggered if the *GPU* either *writes* to the field or *reads and writes* to the field.

phase, where certain functionality will stay on the CPU (probably for a long time) and we incrementally port kernel by kernel to the GPU without much overhead. This scheme was explained for spinor fields but can trivially be extended to gauge and clover fields.

10

CONCLUDING REMARKS OF PART I

> *If this code works, it was written by Paul DiLascia. If not, I don't know who wrote it.*
>
> — Famous code comment by Paul DiLascia

This section/chapter was proofread 1 times.

TL;DR: lay groundwork, offloading heavy lifting

The main goal of this part of the thesis was to enable the RC* collaboration to run the observable evaluation part of non-perturbative QCD+QED simulations on GPU-accelerated computing centers. Offloading repeated solves of the Dirac equation was the focus of the developments in particular, since it represents the most expensive kernel of lattice simulations. This work marks by no means the end of the story; the work merely lays crucial foundation for ongoing software developments.

TL;DR: enabled more funding, compute allocations

As a direct outcome, this work enabled the RC* collaboration to acquire substantial compute allocations at leading supercomputing facilities, advancing the collaboration's physics programme and securing a second round of funding for further code development [118]. **TODO**: Marina: I would rather focus on the physics/implementation and algorithmic part, "for exploring isospin breaking corrections to hadronic observables with Cstar BCs" Future work, including support for configuration generation and unified field treatment (chapter 9), builds directly upon the developments established here.

TL;DR: cicd, easy integration

The openQxD package now interfaces seamlessly with solver kernels provided by QUDA, offering flexible and abstracted integration that reduces the complexity for application-level developers. Existing tools can benefit from the improvements with low porting effort. Code robustness and stability was improved by introducing an automated CI/CD pipeline.

TL;DR: scientific software is hard to do right

Writing scientific software is challenging and can be as frustrating as it is rewarding. Practices common in industry – such as modular design, code readability, testing, deployment, and scalability – require significantly more effort to achieve in the context of scientific computing. However, that does not temper the satisfaction one feels when a piece of code finally works or a tenfold speedup was achieved.

TL;DR: testing is pain, on real-world problems

Testing scientific software is not straightforward. One lesson learned was that testing should reflect production use cases ideally running on the target system using target parameters and target problem sizes, as opposed to a single local workstation on toy problems. Due to its parallel nature and utilization of multi-threading, multi-processing, CPUs, GPUs and being cutting edge in hardware and software developments, systematic testing of scientific software is more essential than ever. This is particularly true when we compute observables for which no independent validation exists yet. In such cases, trust in the software must be earned through rigorous and reproducible validation workflows.

TL;DR: summary of interface, challenges overcome

The interface allows accessing solvers in various ways; directly via openQxD mechanisms suitable for application-level developers, directly after initialization simplifying new programs that only use the GPU solver, asynchronously ideal for heterogeneous workloads where CPU work and GPU solves overlap. Programs alternating or overlapping CPU workloads with GPU solves may lead to imbalanced scaling due to their differing performance characteristics. The dual grid approach resolves this by enabling strong scaling of the CPU part separate to the GPU part. The above mentioned features can be combined and all of them are capable of solving for multiple right-hand sides improving the energy footprint and time to solution.

TL;DR: summary performance data

The performance data shows that the GH200 node can efficiently be utilized. The excellent weak scaling indicates readiness of the code for larger problem sizes. As discussed in the last paragraph, many mechanisms exist to help mitigate strong scaling challenges, especially those inherent to multigrid solvers. While the dual grid performance impact is negligible, CPU-GPU data transfer overheads remain a concern. We proposed a field memory management system aiming at these overheads and reducing node internal PCI express or NVLink chip-to-chip interconnect pressure.

TL;DR: transition to part 2

In the next part, we will approach the problem of efficient utilization of computational resources from an orthogonal perspective: through algorithmic development informed by physical insight. The goal is to improve the estimation of physical observables not merely by accelerating code but by enhancing the efficiency of the simulations themselves via theoretical and algorithmic method development.

Part II

VARIANCE REDUCTION FOR TWO-POINT FUNCTIONS

11

INTRODUCTION TO PART II

> *I used to say that discussions about noise are simply noise ...*
>
> — Nazario Tantalo

TODO: a) Precise observable evaluations in the context of numerical lattice field theory require significant computational resources. Requirements in error budget pose challenges, both for systematic and statistic error estimations. This opens the vast research field of variance reduction methods. As outlined in the introduction, the dominant uncertainty of the current SM prediciton of the muon's anomalous magnetic moment arises from the leading-order HVP contribution [55]. Since the minimal (gauge) variance is usually not reached, powerful variance reduction methods are needed to tackle the light-quark connected part in particular, which suffers from poor signal-to-noise ratios at large distances.

This section/chapter was proofread 1 times.

TL;DR: HVP needs variance reduction methods

TODO: b) Since the QCD action and the path integral measure are invariant under spacetime translations, n-point functions such as eq. (1.50) are translation invariant too; for all spacetime points y it holds

TL;DR: translation invariance

$$C_{\mu\nu}(x_1; x_2) = C_{\mu\nu}(x_1 + y; x_2 + y) \ . \tag{11.1}$$

For quark-line connected correlation functions on the lattice, the variance is expected to decrease inversely proportional to the averaged volume due to the mass gap in QCD [186]. This means that the Euclidean time correlator eq. (1.51) can be averaged over all source lattice points y resulting in minimal variance – variance related only to the fluctuations of the background gauge field. This minimal variance which we will call the gauge variance will be formalized precisely and determined in section 17.3. It gives us a lower bound of what is possible to achieve, but also a variance reduction goal for given gauge field configurations.

Naively evaluating eq. (1.51) for a single source point y has an associated cost of 12 Dirac equation solves as outlined in chapter 12. Thus, a full volume average scaling as $\mathcal{O}(V^2)$ is infeasible. Although, we can calculate an estimation of the volume averaged quantity using improved estimators, for instance the standard method introduced in the following paragraph.

TL;DR: naive evaluation is V2

TL;DR: simple Hutchinson

TODO: c) A standard estimator for traces **TODO**: Tim: Traces"is not defined really carefully. The problem here with the "exact" method (point sources) is the volume averaging over propagators is the Hutchinson method [187], where a stochastic identity is introduced for an estimation of single inverse matrix traces. As a negative byproduct, this introduces additional stochastic noise on top of the gauge noise which has to be suppressed separately. However, this enables stochastic all-to-all propagators [188] when using random wall-sources; sources with support only on a certain time slice. A necessity stemming from correlation functions often being investigated as functions of Euclidean time. This introduces an implicit spatial lattice volume average as discussed in the previous paragraph and is expected to reduce the variance proportional to the spatial volume. The stochastic noise can be any set with zero mean and unit variance, like Gaussian or subsets thereof; a popular choice is $\mathbb{Z}_2 = \{\pm 1\}$ which minimizes stochastic variance over traces of real matrices [189].

TL;DR: index dilution

Further developments of stochastic trace estimation led to dilution schemes (also called partitioning in earlier works [190]). Random sources with support only on certain spin [191, 192], color, time [193–195], space slices [188], even-odd indices or combinations thereof [196–199]. Apart from positive impact on the variance, dilution also has algorithmic consequences. Spin dilution for instance makes it possible with just 4 inversions to reconstruct every γ-matrix monomial possibly appearing in an interpolation operator. This makes it possible to evaluate multiple diagrams with the same 4 spin-diluted noise sources. Dilution on an internal degree of freedom like color and/or spin tends to reduce the overall stochastic noise introduced by the stochastic identity, now being exactly zero on off-diagonal dilution indices [200, 201]. Variance caused by even-odd interactions can be cancelled by even-odd dilution [188, 201, 202]. The structure of the observable should always be a guidance for an effective dilution scheme. Clearly the extreme case would be a dilution in all indices; color, spin, time and space, resulting in an exact estimation using point sources but at infeasible cost.

TL;DR: stochastic estimators reduce variance globally and are V2 too

TODO: d) Standard stochastic estimators as the ones introduced in the last paragraphs reduce variance globally across all Euclidean times of a correlator $G(t)$, i. e. uniformly over the lattice. They are agnostic to the physical observable or the region in Euclidean time where variance is most problematic. On increasing volumes, such estimators require thousands of sources to resolve the large distance properly and a $\mathcal{O}(V^2)$ scaling emerges again when attempting the true volume average.

Many hadronic Euclidean time correlators such as the connected LO-HVP suffer a systemic phenomenon called the signal-to-noise ratio problem [203, 204]. Both the signal as well as the gauge noise decay exponentially in Euclidean time but at different rates. **TODO**: Tim: Here again, you are considering two-point functions (in the time-momentum representation, for instance) This means that the correlator signal gets lost in the gauge noise at large distances. Where the signal decays as $e^{-E_0 t}$ with $E_0 > 0$ the lowest energy state, the gauge noise, which is proportional to the magnitude squared of the correlator, decays as $e^{-E_{\text{noise}} t/2}$ where $E_{\text{noise}} > 0$ is the energy of the lightest state of the squared correlator. The ratio of the two $e^{-(E_0 - E_{\text{noise}}/2)t}$ decays exponentially. If $2E_0 > E_{\text{noise}}$ the signal deteriorates especially for large Euclidean times t.

TL;DR: signal/noise ratio problem and parisi/lepage

The pion correlator for instance has $E_0 = m_\pi$ the pion mass and $E_{\text{noise}} = 2m_\pi$ leading to a constant signal-to-noise ratio. The signal of that correlator clearly does not deteriorate as time increases, since the pion has the lightest mass among the hadrons. Following the analysis of Parisi and Lepage, $E_{\text{noise}} = 3/2 m_\pi$ for baryon correlators where $E_0 = m_b$ the baryon mass. Since baryon masses are in the regime of one or multiple GeV and the pion masses are about 135 MeV, we observe a signal loss already early in the decay. For the isovector light-quark connected LO-HVP correlator, we find $E_0 = m_\rho$ the mass of the ρ-meson which is about 770 MeV and $E_{\text{noise}} = 2m_\pi$ as for baryons – again a loss in signal at large distances.

TL;DR: pion corr is good, baryons, mesons bad

According to the last paragraphs, variance reduction should therefore systematically target the large distance part of the connected LO-HVP. A method called low-mode averaging (LMA) [205–207], which will be discussed in detail in chapter 13, deflates low eigenmodes of the Dirac operator and enables full spacetime lattice volume averaging over the correlator restricted to this deflated low-mode subspace. This is justified by the fact that low-lying eigenmodes govern the infrared (long distance) behavior of the theory [206]. If enough low modes have been deflated, this results in a significant reduction of variance in the long distance. Nearly all current high-precision determinations of the connected LO-HVP employ LMA for this reason [55, 90–93, 208–216].

TL;DR: LMA treats LD specifically

To resolve the large distance of the connected LO-HVP contribution via translation averaging introduced by LMA, the number of low modes has been driven to the limit of affordability; 1000 to 6000 [88, 90, 91, 215] still without reaching gauge variance. The number of low modes required for constant variance reduction is proportional to the lattice volume, because the size of the low-mode subspace is so too [217]. Therefore, the compu-

TL;DR: LMA is V2 too

tational work to determine this subspace scales at least quadratic in the lattice volume and again the $\mathcal{O}(V^2)$ scaling hits us, limiting the achievable precision. This fact renders LMA on large-scale lattices prohibitively expensive, making it the limiting factor in statistical precision of the connected LO-HVP.

TL;DR: we call it V2 problem

This ever emerging quadratic scaling in the lattice volume, closely related to critical slowing down, we will refer to as the V^2-problem [218].

TL;DR: intro disconnected

TODO: disconnected In the following, we will briefly review some related variance reduction methods targeting disconnected diagrams, since some of the motives were carried over to developments in this thesis.

TL;DR: hierarchical probing

For disconnected diagrams which are closely related to single inverse matrix traces, methods based on specific choice of probing vectors have been developed. They go under the name of hierarchical probing [219, 220]. Based on the analysis in ref. [187], the variance of the trace is the Frobenius norm of the matrix with zeroed diagonal entries. The probing method builds structured vectors that systematically cancel noise coming from short distance on nearby lattice sites resulting in significantly reduced variance as compared to standard stochastic estimators for disconnected diagrams [220].

TL;DR: frequency splitting

In a very similar tone, the method called frequency splitting [221, 222] separates high and low frequencies associated to short and long distances in disconnected diagrams by introducing a telescopic sum in the quark masses. The propagator itself is split, turning the disconnected diagram into possibly multiple pieces; the short distance (high frequency) part is split into a bulk which is calculated exactly (i. e. at its gauge noise level) and a remainder, estimated stochastically, that contributes little to the variance. The remaining long distance (low frequency) part (differences of quark propagators with different masses) can then very efficiently be estimated with stochastic split-even estimators making use of the identity $D_1^{-1} - D_2^{-1} = (m_2 - m_1) D_2^{-1} D_1^{-1}$ where D_i is the Dirac operator with mass m_i [223]. This is sometimes referred to as the one-end trick in context of twisted mass fermions [224–226]. Crucial for this method is that the Dirac operator is better conditioned for large quark masses.

TL;DR: Multilevel Monte Carlo

We want to briefly discuss multilevel Monte Carlo (MLMC) methods [227, 228]. Multilevel Monte Carlo is a general framework to split an unbiased stochastic estimator into a sum of estimators. The individual terms contribute differently to the overall variance and one hopes to find decompositions where terms dominating the variance are cheap to estimate. MLMC does not propose actual decompositions but formalizes the general procedure, describing the heart of nearly every variance reduction method.

TL;DR: MG-MLMC

In this spirit, deflation of low modes was used to estimate traces over inverse matrices in [229, 230], but speedups where only obtained together with hierarchical probing. A method called multigrid multilevel Monte Carlo (MG-MLMC) decomposes the propagator into multigrid levels using an adaptive algebraic multigrid approach and then estimates the trace separately on all levels. This was investigated on the 2D (gauge-covariant) Laplacian and the Schwinger model [231]. A similar method was previously been used for trace estimation of single inverse matrices using inexact deflation à la Lüscher [232]. It was recently improved to use the Hutch++ method on coarse levels [233]. The Hutch++ method [234] decomposes the trace into a projected low-rank term computed exactly, whereas the remainder is estimated using standard Hutchinson [187]. First tests on disconnected QCD diagrams where conducted [235] resulting in a moderate variance reduction of about 12 % compared to standard Hutchinson.

This is to be expected, since the stochastic variance in disconnected diagrams stems from the high modes, not from the low modes. The variance of an random inverse matrix trace comes from the off-diagonal parts. In case of the quark propagator the off-diagonal parts decrease exponentially. Therefore it is the closest off-diagonal part that contribute dominantly. These are short distanced, i. e. originate from the high modes. Hierarchical probing reduces short-distance off-diagonal noise by introducing special probing vectors. Frequency splitting splits off the high modes using the propagator difference $D_1^{-1} - D_2^{-1}$. Both methods attempt to estimate this high frequency part more precisely resulting in a net-variance reduction. Low-mode deflation as is done with MG-MLMC is thus not expected to be beneficial.

TL;DR: intro unrelated methods

TODO: e) What will follow are other methods only loosely related to this thesis, since they target reduction of variance on different quantities and problems, but are nevertheless worth mentioning for additional context.

TL;DR: multi-level integration

The fundamental signal-to-noise restriction on correlation functions discussed above led to the development of algorithms employing integration techniques with multiple levels. Such multi-level algorithms[1] use domain decomposition into blocks **TODO**: Tim: overlapping block -> Not true for pure gauge theory. to split the lattice into domains, write the action in terms of contributions from domains which finally leads to a factorization of the observable itself. Each factor depends on a different set of gauge fields with support only on the domain of the decomposed lattice. If applied appropriately, they lead to exponential signal-to-noise suppression in the prefactor of $e^{-(E_0 - E_{\text{noise}}/2)t}$, circumventing the Parisi-Lepage argument. If applied to pure gauge theory this either completely removes the signal-to-noise problem or suppresses it substantially [236–240]. Once fermions enter the game, non-locality of the determinant and the fermion propagator are challenging to deal with in the factorization [241–244]. In the meantime it has been successfully applied to connected and disconnected diagrams of meson two-point functions [245] and the light-connected leading-order HVP of the muon $g-2$ [246, 247].

---

[1] We note that multi-level Monte Carlo schemes for propagator computations is not related to the multi-level integration schemes in the context of generation gauge field samples.

TL;DR: multi-level is at gauge field sampling

A multi-level integration scheme is a variance reduction method entering the gauge field generation phase. Gauge fields in different domains are drawn according to the domain decomposed action. This reduces the variance of the full translation averaged correlator, i. e. the minimal variance. Reaching the gauge noise level with state-of-the-art estimators on large physical pion mass ensembles becomes prohibitively expensive, even more so if applied multi-level schemes reduce the gauge noise further. In this work we are concerned with methods to reach the gauge noise with as little as possible computational resources, not with a reduction of the gauge noise itself. We thus expect the gauge fields to be generated already from now on. Variance reduction methods discussed so far can thus be applied *together* with a multi-level integration scheme to be even more efficient.

TL;DR: smearing methods

TODO: Tim: next 3 paragraphs: either remove completely or shrink and move below the disconnected diagrams which are much more relevant to the thesis In order to increase overlap with the true ground state of the system and reducing excited state contamination, investigation of different operators, with equal quantum numbers was put forward. This led to smearing techniques for gauge and fermion fields. Common smearing kernels are Jacobi [193, 248–250], Wuppertal [251] or momentum smearing [252]. These techniques can be supplemented by smearing of gauge fields using APE [253], HYP [254] or Stout smearing [255].

TL;DR: distillation

A noteworthy smearing method in the context of this thesis is distillation [256, 257], where low modes of the 3D spatial gauge-covariant Laplacian are generated. Source fields are then projected to that low-rank linear subspace; the projector acts as a smearing kernel. Similar to LMA, the cost is proportional the number of modes determined and correlation functions restricted to that subspace can be volume averaged easily, since the restricted propagators are all-to-all. The method was also investigated as a substitute for low-mode averaging [258] with no success.

Smearing of gauge and fermion fields has no effect on the asymptotic exponential behavior of the correlator but modifies its absolute value. This makes smearing very suitable for spectroscopy studies, where one is interested in the exponential decay rate for mass extraction but unsuitable if contributions from excited states are required or the correlator amplitude is important too. This is the case for the LO-HVP of the muon $g-2$.

TL;DR: smearing changes amplitude

The remainder of this part is structured as follows. Chapter 12 introduces the main observable of interest – the connected leading-order HVP of the muon $g-2$. This is followed by the state-of-the-art variance reduction method for this observable called low-mode averaging in chapter 13 which this thesis is based upon. Continuing with chapter 14, we generalize the concept of low-mode averaging to arbitrary subspaces, whereas in chapter 15 a method to approximately determine the low mode subspace used in solver preconditioning is shown. These pieces are put together in chapter 16 to construct a variance reduction method called *multigrid low-mode averaging*, which we will show, solves the V^2-problem. In chapter 17, we show it in action applied to the LO-HVP and provide numerical evidence for its flat volume dependence. This part is closed in chapter 18 with theoretical considerations about chirality.

TL;DR: remainder of the part

12

TWO-POINT CORRELATOR

This section/chapter was proofread 3 times.

TODO: * 2pt corr * estimators * basically different prop decomps * point prop * stochastic prop * spin diagonal * time dilution * LMA prop? in the traditional formulation, eq (12) in lma-final.tex

TL;DR: intro

We will introduce the light-quark connected meson correlator – the observable of interest – in this chapter. The spacetime lattice with all degrees of freedom is formally introduced (section 12.1) before the two-point correlator is brought into a form suitable for computational evaluation (section 12.2) followed by a discussion of common evaluation strategies. The chapter ends with a brief summary (section 12.3).

The observable of interest in this part is the the connected leading order hadronic vacuum polarization contribution (LO-HVP) to the anomalous magnetic moment of the muon as introduced in section 1.2. The connected LO-HVP piece which dominates the error budget is itself a comparably simple observable, but suffers from the signal-to-noise ratio problem. The piece we calculate on the lattice is the Euclidean time correlator eq. (1.51), which in a post-processing phase enters the main integral for the connected LO-HVP contribution eq. (1.52). Before we move on, we need to define the degrees of freedom on a spacetime lattice.

12.1 LATTICE DEFINITION

TL;DR: lattice definition

We define a *lattice* $\mathbb{L}$ as the set of its indices. The fine-grid lattice is defined as the triple $\mathbb{L} = (\Lambda, \Xi, \mathfrak{C})$ and its index set is the cross-product $\mathcal{I} = \Lambda \times \Xi \times \mathfrak{C}$ with

$$\begin{aligned} \text{the spacetime lattice } \Lambda &= \mathcal{T} \times \Sigma \,, \\ \text{the temporal lattice } \mathcal{T} &= \{0, \dots L_0 - 1\} \,, \\ \text{the spatial lattice } \Sigma &= \left\{ \vec{x} = (x_1, x_2, x_3) \in \mathbb{N}^3 \;\middle|\; 0 \leq x_i < L_i \right\} \,, \\ \text{the spins } \Xi &= \{0, \dots N_s - 1\} \,, \\ \text{the colors } \mathfrak{C} &= \{0, \dots N_c - 1\} \,, \end{aligned} \tag{12.1}$$

where $L_0, L_i, N_s, N_c \in \mathbb{N}$. The product $V = L_0 L_1 L_2 L_3$ is called the *lattice volume*, whereas the product $N_s N_c$ parametrizes the internal degrees of freedom of a spinor field on each lattice point. The product of the two $d = VN_sN_c$ gives us the dimension over the complex numbers $\mathbb{C}$ of spinor fields and operators acting on spinor fields on the lattice. The native lattice where fermion fields live with $(N_s, N_c) = (4, 3)$, we will call the *fine-grid lattice* $\mathbb{L}$. Furthermore, the Hilbert space of spinor fields $\mathcal{V}$ is the tensor product of Hilbert spaces,

$$\mathcal{V} = \mathcal{H}_{L_0} \otimes \mathcal{H}_{L_1} \otimes \mathcal{H}_{L_2} \otimes \mathcal{H}_{L_3} \otimes \mathcal{H}_{N_s} \otimes \mathcal{H}_{N_c} \cong \mathbb{C}^d \ , \tag{12.2}$$

where each individual space corresponds to an index set in eq. (12.1) and is isomorphic to $\mathcal{H}_N \cong \mathbb{C}^N$. We note, the Dirac operator which is an endomorphism on this Hilbert space, $D\colon \mathcal{V} \to \mathcal{V}$, is not separable into a tensor product, because its action to a spinor field couples and mixes all individual degrees of freedom non-trivially, compare eq. (1.69).

12.2 TWO-POINT CORRELATOR

To obtain a formula which we can evaluate on a computer, we perform a Wick contraction of eq. (1.50) on the lattice. We will use Wick's theorem ($S = D^{-1}$ is the quark propagator)

$$\langle \psi[x]\,\underset{[a]}{[\alpha]}\ \bar{\psi}[y]\,\underset{[b]}{[\beta]} \rangle = a^{-4} S\,{}_{[x;y]}\,\underset{[ab]}{[\alpha\beta]} \ , \tag{12.3}$$

to get rid of the Grassmann fields and obtain an expression in terms of quark propagators[1]. We obtain connected and disconnected contributions,

$$C_{\mu\nu}(x;y) = C^{\text{conn}}_{\mu\nu}(x;y) + C^{\text{disc}}_{\mu\nu}(x;y) \ , \tag{12.4}$$

$$C^{\text{conn}}_{\mu\nu}(x;y) = \operatorname{tr}\left\{ S_{[x;y]}\gamma_\mu S_{[y;x]}\gamma_\nu \right\} \ , \tag{12.5}$$

$$C^{\text{disc}}_{\mu\nu}(x;y) = \operatorname{tr}\left\{ S_{[x;x]}\gamma_\mu \right\} \cdot \operatorname{tr}\left\{ S_{[y;y]}\gamma_\nu \right\} \ , \tag{12.6}$$

where the trace is over color and spin indices. As motivated in section 1.4, we will concentrate on the connected (light-quark) piece, that contributes about 98% of the value of $a_\mu^{\text{LO-HVP}}$ and 97% to the variance (see figs. 1.2 and 1.3 and ref. [55]). On the lattice the Fourier integral of the connected bare isovector vector correlator, eq. (1.51), turns into a sum over spatial lattice points and can be evaluated using the result in eq. (12.5) as

---

[1] We note that continuum objects like the spinor field $\psi(x)\,\underset{a}{\alpha}$ will have a discretized lattice analogue $\psi[x]\,\underset{[a]}{[\alpha]}$, emphasizing that spinors are just vectors with a multi-index or multi-dimensional arrays put on a computer.

$$G^{\mathrm{conn}}(t) = -\frac{a^3}{3|\Omega|} \sum_{y\in\Omega} \sum_{\vec{x}} \sum_{k=1}^{3} \mathrm{tr}\left\{ S_{[t+y_0,\vec{x};y]} \gamma_k S_{[y;t+y_0,\vec{x}]} \gamma_k \right\} . \tag{12.7}$$

This is the form suitable for computational evaluation. There are many strategies to evaluate this observable. In the following we will discuss some of them.

12.2.1 *Point propagator*

The most straightforward way to evaluate eq. (12.7) is by using point sources. They are defined to be fields which have support only on one single lattice point y, spin β and color b, i. e. they are standard basis vectors. With these sources as right-hand sides, we solve the Dirac equation 12 times, once for each color-spin combination

$$D\psi^{y,\beta,b} = \eta^{y,\beta,b} , \quad \beta = 0,1,2,3 , \quad b = 0,1,2 . \tag{12.8}$$

In this equation $\eta^{y,\beta,b}$ is a point source,

$$\eta^{y,\beta,b}[x]_{\substack{[\alpha]\\ [a]}} = \delta_{\alpha\beta}\delta_{ab}\delta_{xy} , \tag{12.9}$$

and the solution $\psi^{y,\beta,b} = D^{-1}\eta^{y,\beta,b}$ is the column of the matrix D^{-1} corresponding to source index (y, β, b). This allows us to define the propagator – the Green's function to the lattice Dirac operator – in terms of solutions to eq. (12.8), explicitly

$$S_{[x;y]\,\substack{[\alpha\beta]\\ [ab]}} = \psi^{y,\beta,b}[x]_{\substack{[\alpha]\\ [a]}} . \tag{12.10}$$

Since this equality holds for fixed source point y but open spin and color indices α, β, a, b and open lattice sink point x, the above is called the *point-to-all propagator*. This not an approximation but an exact equation. It makes it possible to evaluate eq. (12.7) for some restricted set of source points $y \in \Omega$.

We note that one may increase the precision by adding more distinct points to the set Ω, but every source point amounts 12 solves of the Dirac equation, which might quickly become expensive. However the cost of this estimator – and by this its precision – is under control of the user and can be improved arbitrarily even post hoc. Ideal would be to include all lattice points in Ω, resulting in a volume average and minimal variance but the cost for this is impracticable.

12.2.2 *Stochastic propagator*

Instead of an exact evaluation as in the previous section, we can introduce a stochastic estimator for eq. (12.7). We apply the Hutchinson method for estimating the trace of an inverse matrix [187, 259]. Our noise vectors have to be diluted in time-direction, because of the fact that in eq. (12.7) the sink time $x_0 = t + y_0$ is shifted relative to the source time y_0 and thus depends on it. We cannot add a stochastic identity in time. This means that we use $N_{\rm st}$ stochastic noise sources $\eta_n^{\langle t_n \rangle}$ distributed uniformly in time with support only on a certain time slice given by $t_n \in \{0, \dots L_0 - 1\}$,

$$\eta_n^{\langle z_0 \rangle}[x] = 0 \quad \text{unless } z_0 = x_0 \,. \tag{12.11}$$

The sources have zero mean

$$\lim_{N_{\rm st}\to\infty} \frac{1}{N_{\rm st}} \sum_{n=0}^{N_{\rm st}-1} \eta_n^{\langle z_0 \rangle}[x]_{\substack{[\alpha]\\[a]}} = 0 \,, \tag{12.12}$$

and unit variance

$$\lim_{N_{\rm st}\to\infty} \frac{1}{N_{\rm st}} \sum_{n=0}^{N_{\rm st}-1} \eta_n^{\langle z_0 \rangle}[x]_{\substack{[\alpha]\\[a]}} \left(\eta_n^{\langle z_0 \rangle}[y]_{\substack{[\beta]\\[b]}} \right)^\dagger = \delta_{\alpha\beta}\, \delta_{ab}\, \delta_{\vec{x}\vec{y}}\, \delta_{x_0 z_0}\, \delta_{y_0 z_0} \,. \tag{12.13}$$

The noise can be any (sub-)Gaussian set of noise for instance U(1) or $\mathbb{Z}_2 = \{-1, 1\}$. This allows us to write a stochastic identity in spin, color and space at (for now) fixed time slice z_0 as

$$\lim_{N_{\rm st}\to\infty} \frac{1}{N_{\rm st}} \sum_{n=0}^{N_{\rm st}-1} \eta_n^{\langle z_0 \rangle} (\eta_n^{\langle z_0 \rangle})^\dagger = \mathbb{1}_\Xi \otimes \mathbb{1}_{\mathfrak{C}} \otimes \mathbb{1}_\Lambda^{\langle z_0 \rangle} \equiv \mathbb{1}_{\rm st}^{\langle z_0 \rangle} \,, \tag{12.14}$$

with the identities on the tensor spaces

$$(\mathbb{1}_\Xi)_{[\alpha\beta]} = \delta_{\alpha\beta} \,, \qquad \text{for } \alpha, \beta \in \Xi \,, \tag{12.15}$$

$$(\mathbb{1}_{\mathfrak{C}})_{[ab]} = \delta_{ab} \,, \qquad \text{for } a, b \in \mathfrak{C} \,, \tag{12.16}$$

$$\mathbb{1}_\Lambda^{\langle z_0 \rangle}{}_{[x;y]} = \delta_{\vec{x}\vec{y}}\, \delta_{x_0 z_0}\, \delta_{y_0 z_0} \,, \qquad \text{for } x, y \in \Lambda, z_0 \in \mathcal{T} \,. \tag{12.17}$$

We notice that if $N_{\rm st} < \infty$, eq. (12.14) is not an equality. Evaluating observables with it will introduce non-negligible stochastic noise on top of the gauge noise from the estimator itself which has to be taken care of when estimating errors. However, the identity eq. (12.14) can be plugged in eq. (12.7) at any place. All choices are equivalent in terms of computational cost. The stochastic estimator inherently introduces a spatial lattice volume average on time slice z_0, i. e. $|\Omega| = |\Sigma| = L_1 L_2 L_3$. It can be seen as a stochastic all-to-all propagator on a fixed time slice z_0. Upon including all time slices, it is truly a stochastic all-to-all propagator. Thus by plugging in the stochastic identity, we find

$$C(x_0, y_0) = \frac{1}{|\Sigma|} \sum_{\vec{x},\vec{y}} \operatorname{tr}\left\{ S_{[x;y]} \gamma_k S_{[y;x]} \gamma_k \right\} \tag{12.18}$$

$$= \frac{1}{|\Sigma|} \sum_{\vec{x},w,z} \operatorname{tr}\left\{ S_{[x;w]} \gamma_k \mathbb{1}_{\mathrm{st}}^{\langle y_0 \rangle}{}_{[w;z]} S_{[z;x]} \gamma_k \right\} \tag{12.19}$$

$$\approx \frac{1}{|\Sigma| N_{\mathrm{st}}} \sum_{n=0}^{N_{\mathrm{st}}-1} \sum_{\vec{x},w,z} \operatorname{tr}\left\{ S_{[x;w]} \gamma_k \eta_n^{\langle y_0 \rangle}[w] (\eta_n^{\langle y_0 \rangle}[z])^\dagger S_{[z;x]} \gamma_k \right\} . \tag{12.20}$$

The trace turns into a spinor product by using γ^5-Hermiticity of the lattice Dirac operator D and its propagator S,

$$C(x_0, y_0) \approx \frac{1}{|\Sigma| N_{\mathrm{st}}} \sum_{n=0}^{N_{\mathrm{st}}-1} \sum_{\vec{x},w,z} \left(S_{[x;z]} \gamma^5 \eta_n^{\langle y_0 \rangle}[z], \gamma^5 \gamma_k S_{[x;w]} \gamma_k \eta_n^{\langle y_0 \rangle}[w] \right) . \tag{12.21}$$

Finally, the sums on z and w are factorized and can be pulled inside

$$C(x_0, y_0) \approx \frac{1}{|\Sigma| N_{\mathrm{st}}} \sum_{n=0}^{N_{\mathrm{st}}-1} \sum_{\vec{x}} \left((S \gamma^5 \eta_n^{\langle y_0 \rangle})[x], \gamma^5 \gamma_k (S \gamma_k \eta_n^{\langle y_0 \rangle})[x] \right) . \tag{12.22}$$

In this step we define the solutions of the noise sources

$$D \psi_n^{\Gamma \langle y_0 \rangle} = \Gamma \eta_n^{\langle y_0 \rangle} , \tag{12.23}$$

which finally give us the stochastic estimator in terms of a fermion bilinear

$$C(x_0, y_0) \approx \frac{1}{N_{\mathrm{st}}} \sum_{n=0}^{N_{\mathrm{st}}-1} C_n(x_0, y_0) , \tag{12.24}$$

$$C_n(x_0, y_0) = \frac{1}{|\Sigma|} \sum_{\vec{x}} \left(\psi_n^{\gamma^5 \langle y_0 \rangle}[x], \gamma^5 \gamma_k \psi_n^{\gamma_k \langle y_0 \rangle}[x] \right) . \tag{12.25}$$

This expression holds for a fixed time slice y_0. We may uniformly sample different times $y_0 \in \mathcal{T}$ and take the sample average or just average over all times

$$G^{\text{conn}}(t) = \frac{1}{L_0} \sum_{y_0 \in \mathcal{T}} C(t + y_0, y_0) \; . \tag{12.26}$$

This analysis shows that a stochastic evaluation of eq. (12.7) amounts of $4N_{\text{st}}$ solves of the Dirac equation. 4 because for every individual source, we need a solve for every γ-matrix that appears in the correlator, see eq. (12.23), meaning for $\Gamma \in \{\gamma^5, \gamma_1, \gamma_2, \gamma_3\}$.

12.3 SUMMARY

We have introduced the main observable of interest in this thesis – the connected leading order hadronic vacuum polarization contribution (LO-HVP) to the anomalous magnetic moment of the muon, a_μ. Two common simple strategies for its evaluation have been discussed

First, a direct evaluation using point sources that goes along the choice of the set of sources points (called Ω in the text). Its cost is linear in the cardinality of that set $|\Omega|$ and to obtain a full translation average (i. e. minimal variance) the set has to be equal to be full lattice $\Omega = \Lambda$, resulting in at least a quadratic cost scaling in the volume $\mathcal{O}(V^2)$.

Second, a stochastic estimator using random wall-sources; the Hutchinson method. With its stochastic identity fixed on a time-slice, it introduces an implicit translation average over the spatial lattice $|\Omega| = L_1 L_2 L_3$. Although reducing the variance proportional to the spatial volume, it introduces additional non-negligible variance due to its stochastic nature that has to be suppressed separately by increasing the number of samples. We will see later that on large lattices, this number becomes proportional to the volume and a $\mathcal{O}(V^2)$ cost scaling arises again.

The next chapter will introduce the state-of-the-art method how this observable is evaluated for high-precision studies.

13

LOW-MODE AVERAGING

This section/chapter was proofread 1 times.

TL;DR: LMA original and usages

This chapter introduces the variance reduction method of low-mode averaging [205–207] (LMA) and some of its many variants which had a large influence in this thesis. We start with the general propagator decomposition (section 13.1), followed by an individual discussion of the emerging terms related to the high modes (section 13.2), low modes (section 13.3) and the crossing term (section 13.4). The chapter ends with an analysis of its problems in terms of unfortunate quadratic volume scaling and challenges that emerge when evaluating the pieces introduced through the propagator decomposition (section 13.5).

LMA consists of decomposing the quark propagator $S = D^{-1}$ into a truncated spectral decomposition and a remainder, defined precisely in the following. The method has been widely used in the field for observables with poor signal in the long distance sensitive to low modes. Examples are the LO-HVP [88, 91, 215, 260, 261] and HLbL [262] contributions to the muon $g-2$ or derived quantities [263], baryon three-point functions [264, 265] and masses [266], nucleon charges [267], reweighting of the RHMC [268], decay constants [195], low energy couplings [269], form factors [270] and many more. Clearly, the method is only beneficial if the volume average over the deflated subspace captures the large distance behavior of the observable and one can afford a numerical determination of the low-mode subspace.

13.1 SPECTRAL DECOMPOSITION

In its simplest form, LMA introduces an improved estimator for eq. (12.7) using the spectral decomposition of the Hermitian Dirac operator,

$$Q = \sum_{n=0}^{12V-1} \lambda_n \xi_n \xi_n^\dagger \,, \qquad Q = \gamma^5 D \,, \qquad Q^\dagger = Q \,, \tag{13.1}$$

into an orthonormal eigenbasis $\mathcal{B} = \{\xi_n\}_{n=0}^{12V-1}$ with

$$Q\xi_n = \lambda_n \xi_n \,, \qquad \xi_n^\dagger \xi_m = \delta_{nm} \,, \tag{13.2}$$

with real eigenvalues $\lambda_n \in \mathbb{R}$. The same decomposition holds for the propagator

$$S_{[x;y]} = \sum_{n=0}^{12V-1} \frac{1}{\lambda_n} \xi_n[x] \xi_n^\dagger[y] \gamma^5 \,, \tag{13.3}$$

where the γ^5 at the end is because $S = Q^{-1}\gamma^5$. This is actually a singular value decomposition[1]. Analogously, we emphasize just as eq. (12.10) is an exact point-to-all propagator (no approximation involved), the above is an exact all-to-all propagator.

In its current state, this propagator is not appropriate for numerical computations, because the full eigen-decomposition into all $12V$ modes is numerically inaccessible. Although we can truncate the sum in eq. (13.3) and only take the $N_\mathrm{c} \ll 12V$ modes with lowest magnitude eigenvalues, i. e. the largest $\frac{1}{\lambda_n}$. These are the modes that contribute most to the propagator and to the variance [206]. To do this, we define an orthogonal projector to the space of the N_c lowest modes as

$$P = \sum_{n=0}^{N_\mathrm{c}-1} \xi_n \xi_n^\dagger \,, \qquad P^2 = P = P^\dagger \,. \tag{13.4}$$

The full all-to-all propagator is then decomposed into two terms

$$S = \underbrace{\sum_{n=0}^{N_\mathrm{c}-1} \frac{1}{\lambda_n} \xi_n \xi_n^\dagger \gamma^5}_{=S_e} + \underbrace{(1-P)Q^{-1}\gamma^5}_{=S_r} \,, \tag{13.5}$$

one contribution along the low eigenmode space S_e and a remainder S_r. We note that this decomposition is an exact equality.

The two-point connected correlator eq. (12.7) has two slots for propagators. We can either plug eq. (13.5) in only one slot or in both of them. Both options result in different valid estimators. We will continue with the latter and since the propagator is now a sum of two terms, we obtain four individual terms,

$$G^\mathrm{conn}(t) = G^\mathrm{conn}_{ee}(t) + G^\mathrm{conn}_{er}(t) + G^\mathrm{conn}_{re}(t) + G^\mathrm{conn}_{rr}(t) \,. \tag{13.6}$$

The terms can be worked out using properties of the γ-matrices

$$\{\gamma^5, \gamma_\mu\} = 0 \,, \qquad (\gamma^5)^2 = \mathbb{1} \,, \tag{13.7}$$

$$\gamma_\mu^\dagger = \gamma_\mu \,, \qquad (\gamma^5)^\dagger = \gamma^5 \,. \tag{13.8}$$

---

[1] To be more precise, $S = \sum_n \sigma_n \psi_n \phi_n^\dagger$, where $\sigma_n = \frac{1}{|\lambda_n|}$ are the singular values with $\psi_n = \mathrm{sign}(\lambda_n)\xi_n$ and $\phi_n = \gamma^5 \xi_n$ the left and right singular vectors, respectively.

Since their evaluation schemes differ quite substantially, we will individually discuss them in the following.

13.2 THE REST-REST TERM

The rest-rest term is the simplest one in the decomposition. It is just the standard form eq. (12.7), but with S being replaced by the projected $\tilde{S} = (1-P)S$,

$$G^{\mathrm{conn}}(t) = -\frac{a^3}{3|\Omega|} \sum_{y\in\Omega} \sum_{\vec{x}} \sum_{k=1}^{3} \mathrm{tr}\left\{ \tilde{S}_{[t+y_0,\vec{x};y]} \gamma_k \tilde{S}_{[y;t+y_0,\vec{x}]} \gamma_k \right\} . \tag{13.9}$$

If enough low modes were deflated, this term does not significantly contribute to the variance in the large distance. It can then be estimated using standard estimators, for instance point- or stochastic time-diluted sources as introduced in sections 12.2.1 and 12.2.2. Evaluation of this term differs only slightly from the original correlator. Still, linear systems of equation of the type $D\psi = \eta$ have to be solved, but followed by projecting the solution with $1-P$.

13.3 THE EIGEN-EIGEN TERM

The eigen-eigen term can be worked out as

$$\begin{aligned} G^{\mathrm{conn}}_{ee}(t) = -\frac{a^3}{3V} \sum_{k=1}^{3} \sum_{n,m=0}^{N_c-1} \frac{1}{\lambda_n \lambda_m} \sum_{\vec{x},y} & \left(\xi_n[y], \gamma^5 \gamma_k \xi_m[y] \right) \\ & \cdot \left(\xi_m[t+y_0,\vec{x}], \gamma^5 \gamma_k \xi_n[t+y_0,\vec{x}] \right) , \end{aligned} \tag{13.10}$$

where the $(\cdot,\cdot)$ denotes the standard spinor product on the lattice Hilbert space. Since the eigenmodes are defined on every lattice point, we can easily perform a full volume average of this contribution $|\Omega| = V$ as seen in the prefactor. This is characteristic to this type of estimator but also crucial, because now the eigen-eigen term is at the minimal variance (its gauge noise level) and its variance cannot be lowered further for a fixed gauge config[2]. Due to the exact averaging over the subspace of low modes, this class of estimators is known under the name low-mode *averaging*. All deflated low modes have to be in memory at the same time, because the above is a all-to-all contraction in the eigenvector indices n, m. We will later see that it can be written as a trace of objects living in the eigenspace.

---

[2] Without employing multi-level sampling schemes.

13.4 THE CROSS-TERM

The two crossing terms, rest-eigen and eigen-rest, show some symmetry,

$$G_{er}^{\text{conn}}(t) = \frac{a^3}{3L^3|\mathcal{T}|} \sum_{y_0 \in \mathcal{T}} \sum_{n=0}^{N_c-1} \frac{1}{\lambda_n} \sum_{k=1}^{3} \left(\tilde{S}\gamma_k \xi_n^{\langle y_0 \rangle}, \gamma^5 \gamma_k \xi_n^{\langle t+y_0 \rangle} \right) , \tag{13.11}$$

$$G_{re}^{\text{conn}}(t) = \frac{a^3}{3L^3|\mathcal{T}|} \sum_{y_0 \in \mathcal{T}} \sum_{n=0}^{N_c-1} \frac{1}{\lambda_n} \sum_{k=1}^{3} \left(\gamma^5 \gamma_k \xi_n^{\langle t+y_0 \rangle}, \tilde{S}\gamma_k \xi_n^{\langle y_0 \rangle} \right) , \tag{13.12}$$

where $\tilde{S} = (1 - P)S$ and we have introduced the time-diluted modes

$$\xi_n^{\langle t \rangle}[x] = \begin{cases} \xi_n[x] & \text{if } x_0 = t \,, \\ 0 & \text{otherwise} \,. \end{cases} \tag{13.13}$$

They were defined such that the original mode can be recovered by summing over all times and they are mutually orthogonal with respect to the time dilution but not orthonormal anymore

$$\xi_n = \sum_{t=0}^{L_0-1} \xi_n^{\langle t \rangle} \,, \qquad \left(\xi_n^{\langle t \rangle}, \xi_m^{\langle s \rangle} \right) \sim \delta_{ts} \,. \tag{13.14}$$

Both cross-terms are in general complex valued, but they are complex conjugates of each other and thus we only need to estimate one of them

$$\begin{aligned} G_{\times}^{\text{conn}}(t) &= G_{er}^{\text{conn}}(t) + G_{re}^{\text{conn}}(t) && (13.15) \\ &= 2\,\text{Re}\{G_{er}^{\text{conn}}(t)\} \,. && (13.16) \end{aligned}$$

Clearly the cross-term is expensive to calculate exactly, since we need to solve the Dirac equation on every time-diluted mode for every γ-matrix we want to evaluate the correlator for, i. e. solve the systems

$$D\psi = (1 - P)\gamma_k \xi_n^{\langle t \rangle} \,, \tag{13.17}$$

for all $k = 1, 2, 3$, $t \in \{0, \ldots L_0 - 1\}$ and $n = 0, \ldots, N_c - 1$. In total $3N_cL_0$ solves to calculate the piece exactly! Many improved estimators will not include the whole time extend, all γ-matrices or all modes but a subset of them. Additionally, many LMA-scenarios employ heavily truncated solves on subsets for the above and correct the bias regularly with differences of high precision and truncated solves to obtain an unbiased estimator [89, 260]. Alternatively one can introduce a stochastic estimator for the cross-term [271, 272]. In all variants of LMA, this term comes at a large associated cost overhead due to its non-negligible variance contribution. We will call this the *cross-term problem*.

Another noteworthy variant of LMA goes under the name of all-mode averaging (AMA) [87, 273–275]. It comes with a slightly different propagator decomposition

$$S = \underbrace{S - S_{\mathrm{AMA}}}_{S_{\mathrm{r}}} + \underbrace{\sum_{n=0}^{N_{\mathrm{c}}-1} \frac{1}{\lambda_n} \xi_n \xi_n^\dagger \gamma^5 + P_n(Q) P \gamma^5}_{S_{\mathrm{AMA}}} \ . \qquad (13.18)$$

Here $P_n(Q)$ is a polynomial in Q usually obtained from an implicitly generated polynomial of a truncated solve (TSM) [202] preconditioned by the eigenmodes and P is defined as in eq. (13.4). Here, the TSM does not need bias correction, since the above is a exact decomposition, irrespective of the truncation scheme. For this estimator, it is also possible to set $N_c = 0$ [273]. The terms involving S_r are treated with very few sources, since they do not contribute much to the overall variance. Nevertheless, AMA merely moves the problem of the cross-term into the term including S_{AMA} without solving it at its root. The polynomial's purpose is to extend the averaging from low modes to all modes, hence the name *all*-mode averaging. However, in all scenarios the cost of this term poses one of the major limiters of LMA, a problem which we aim to solve in the remainder of this part.

13.5 SUMMARY

Clearly the cost and the effectiveness of LMA and variants critically depend on the number of low modes N_{c} where the spectral sum is truncated – an algorithmic parameter not easy to chose properly. Due to Banks and Casher [217], N_{c} need to be proportional to the lattice volume, a condition hard to satisfy on large lattices. We called this the *V^2-problem*. Furthermore, we encounter two competing motives. First, the number of low modes should be large enough to capture all the LD behavior and that the cross-term contributes negligibly to the overall variance. Second, the number of low modes should be small enough to have an affordable cross-term. This is a non-trivial interplay of computational cost and variance reduction.

It is not feasible anymore on large lattices to drive the number of eigenmodes so high to suppress the variance on the cross-term such that its evaluation is cheap. An often unmentioned issue is the immense memory footprint of $\mathcal{O}(1000)$ low modes that have to be in memory all at the same time, sometimes even driving the required node count of the job into the bad strong scaling region. Many variants of LMA try to circumvent the cross-term problem by reducing the cost of its estimator. Still, these variants do not solve the problem at its root, the V^2-problem. Revisiting current high-precision determinations of the connected LO-HVP of the muon $g-2$, the numbers of 1000 to 6000 low modes used in these studies [88, 90, 91, 215] are not high enough; they are mere thresholds of what is affordable.

We have introduces a new class of estimators for the connected LO-HVP contribution based on low-mode deflation. The propagator is decomposed into a truncated spectral sum and a remainder. The two-point correlator involving the spectral sum can be easily volume averaged pushing its variance to its gauge variance, whereas the variance on the remaining terms is considered small. Unfortunately, this is not the case anymore, because a number of low modes proportional to the lattice volume is not affordable on modern large scale simulations. We mentioned limits of LMA in terms of the two related problems: the V^2-problem (number of low modes required is proportional to the lattice volume) and the cross-term problem (remaining cross piece still contributes non-negligibly to the overall variance).

14

SUBSPACE DEFLATION

This section/chapter was proofread 3 times.

TL;DR: subspace deflation intro

The method of low-mode averaging from the last chapter will be generalized to cover not only subspaces spanned by low modes but arbitrary subspaces spanned by an arbitrary set of (currently unspecified) orthonormal basis fields (section 14.1). We will discuss general operators acting on the lattice and how to restrict them to the subspaces – a process which we will call coarsening (section 14.2). Revisiting LMA shows that it is a special case of subspace deflation (section 14.3), before we close the chapter with a summary (section 14.4).

14.1 SUBSPACE DEFINITION

TL;DR: arbitrary ONB

We want to formalize and generalize the idea of low-mode averaging from the previous chapter to an arbitrary orthonormal basis of N spinor fields $\phi_i \in \mathcal{B}$ spanning an arbitrary subspace $\operatorname{span}\{\mathcal{B}\} \subseteq \mathcal{V}$ of the fine-grid lattice Hilbert space $\mathcal{V} = \mathbb{C}^d$. The space $\operatorname{span}\{\mathcal{B}\}$ is isomorphic to a Hilbert space $\hat{\mathcal{V}}$ of dimension N. In accordance to section 12.1, for now we define the dimension over the complex numbers of the subspace as $\hat{d} = \dim_{\mathbb{C}}(\hat{\mathcal{V}})$, whereas the dimension of the fine space $\mathcal{V}$ is $d = \dim_{\mathbb{C}}(\mathcal{V})$. The choice of the basis fields and its implications will be discussed later.

TL;DR: restrictor definition

We define two non-square matrices that act as inter-grid operators between the vector spaces $\mathcal{V}$ and $\hat{\mathcal{V}}$. One which we call the *restrictor* R restricts a spinor ψ to the subspace and acts as

$$\begin{aligned} R\colon \mathcal{V} &\longrightarrow \hat{\mathcal{V}} \,, \\ \psi &\longmapsto \hat{\psi} \,, \\ (R\psi)_{[i]} &= (\phi_i, \psi) \,, \quad i \in \{0, \dots, N-1\} \,, \end{aligned} \tag{14.1}$$

where the subscript notation $\psi_{[i]}$ indicates the i-th component of the vector ψ (see appendix E.4) and the scalar product goes over all indices of spinors; color, spin and spacetime.

TL;DR: prolongator definition

The second one which we call the *prolongator* T prolongates information back from the subspace to the vector space of spinors. It acts as

$$\begin{aligned} T\colon \hat{\mathcal{V}} &\longrightarrow \mathcal{V}\,, \\ \hat{\psi} &\longmapsto \psi\,, \\ T\hat{\psi} &= \sum_{i=0}^{N-1} \hat{\psi}_{[i]}\phi_i\,. \end{aligned} \tag{14.2}$$

TL;DR: projector definition

The prolongator is related to the restrictor by Hermitian transposition $T = R^\dagger$. Therefore, the restrictor is a $\hat{d} \times d$ matrix, whereas the prolongator is a $d \times \hat{d}$ matrix. We can build the concatenation of the two and obtain the $d \times d$ Hermitian projector P, projecting spinor fields to the space along the ϕ_i,

$$P = TR\colon \mathcal{V} \longrightarrow \mathcal{V}\,, \qquad P^2 = P = P^\dagger\,. \tag{14.3}$$

14.2 COARSE OPERATORS

TL;DR: coarse operator definition

Next, we define operators on the subspace from operators acting on the lattice $\mathbb{L}$. We consider such operators acting on the subspace as *coarsenings* on fine-grid operators. For an operator on the lattice $K\colon \mathcal{V} \longrightarrow \mathcal{V}$, we define the coarse operator as

$$\hat{K} = RKT\colon \hat{\mathcal{V}} \longrightarrow \hat{\mathcal{V}}\,. \tag{14.4}$$

In index notation, this is equivalent to

$$\hat{K}_{[ij]} = (\phi_i, K\phi_j) = \mathrm{tr}_{\mathcal{I}}\left(K\phi_j\phi_i^\dagger\right)\,, \tag{14.5}$$

where the trace goes over all spinor indices $\mathcal{I} = \Lambda \times \Xi \times \mathfrak{C}$. The coarse operator may inherit certain properties of the fine-grid one, if the coarsening – and by this the choice of the deflation basis $\mathcal{B}$ – is done properly. We will discuss these choices later. A subset of indices may be left open, such as $\hat{K}[x]$ or $\hat{K}[x_0]$ indicating a partial trace in eq. (14.5).

TL;DR: partial trace definition and open indices

We define the *partial trace* as follows. Assuming two finite-dimensional Hilbert spaces, $\mathcal{H}_A$ and $\mathcal{H}_B$, we define the partial trace as the completely positive, trace preserving map (CPTPM) acting on the endomorphisms of the tensor product, yielding an endomorphism on the remaining Hilbert space,

$$\begin{aligned} \mathrm{tr}_B\colon \mathrm{End}(\mathcal{H}_A \otimes \mathcal{H}_B) &\longrightarrow \mathrm{End}(\mathcal{H}_A)\,, \\ O &\longmapsto \mathrm{tr}(O) = \sum_i (\mathbb{1}_A \otimes \psi_i)^\dagger O (\mathbb{1}_A \otimes \psi_i)\,, \end{aligned} \tag{14.6}$$

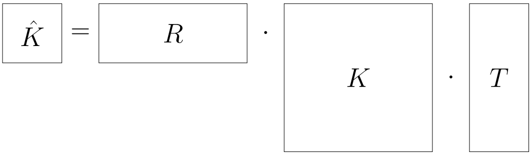


FIGURE 14.1: Depiction of the fact that the prolongator and restrictor are non-square matrices and coarse operator are restrictions of their fine-grid versions by dimensional reduction.

where $\{\psi_i\}_i$ is an orthonormal basis of $\mathcal{H}_B$ and $\mathbb{1}_A$ is the identity operator on $\mathcal{H}_A$. When the subscript on the trace is absent, we implicitly trace out all degrees of freedom resulting in a number; $\mathrm{tr}\colon \mathrm{End}(\mathcal{H}) \longrightarrow \mathbb{C}$. Therefore, if in eq. (14.5) the partial trace is taken, the index is absent in the trace on the right hand side.

$\hat{K}$ is the restriction of K to the subspace spanned by the $\phi_i \in \mathcal{B}$. As long as $N < 12V$, the coarse operator is smaller in dimension as depicted in fig. 14.1. Trivially, when taking $K = \mathbb{1}$ we find the $\hat{d} \times \hat{d}$ identity on the subspace,

TL;DR: restriction and identity

$$\hat{\mathbb{1}} = RT\colon \hat{\mathcal{V}} \longrightarrow \hat{\mathcal{V}} \,. \tag{14.7}$$

14.3 LOW-MODE AVERAGING

We can immediately see that low-mode averaging as introduced in the last chapter is a variant of subspace deflation with a particular choice for the fields spanning the subspace $\hat{\mathcal{V}}$ and the operator K as

TL;DR: LMA as specific choice of subspace deflation

$$K = \gamma^5 D \,. \qquad \text{the Hermitian Dirac operator,} \tag{14.8}$$

$$\phi_i = \xi_i \,, \qquad \text{the } N = N_c \text{ exact lowest modes of } Q. \tag{14.9}$$

We assume the eigenvalues of K are enumerated in increasing order with respect to their magnitudes $|\lambda_0| \leq |\lambda_1| \leq \ldots \leq |\lambda_{N-1}|$. The coarse Hermitian Dirac operator $\hat{K}$ (as well as its inverse $\hat{K}^{-1}$) are then diagonal matrices with the eigenvalues along their diagonals,

$$\hat{K} = \mathrm{diag}\left(\lambda_0, \lambda_1, \ldots, \lambda_{N-1}\right) \,, \qquad \hat{K}^{-1} = \mathrm{diag}\left(\frac{1}{\lambda_0}, \frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_{N-1}}\right) \,. \tag{14.10}$$

Furthermore, with the ϕ_i eigenmodes the projector commutes with the operator, $[P, Q] = 0$, and thus the propagator S decomposes exactly as in the LMA case

$$S_r = (1 - P)S \,, \tag{14.11}$$

$$S_e = PS = T\hat{Q}^{-1}R\gamma^5 \,, \tag{14.12}$$

compare eq. (13.5). This shows us, that S_e is a low-rank approximation to S in terms of its spectral decomposition with

$$\|S_e - S\|_2 = \left|\frac{1}{\lambda_N}\right| \,, \qquad \mathrm{rank}(S_e) = N \,, \tag{14.13}$$

where $\|\cdot\|_2$ is the spectral norm. This family of approximations conserves the low-mode behavior of the propagator which is responsible for the large distance behavior of the vector two-point correlator. The rank of the approximation is the number of deflated low-modes N. Although this parameter is under direct user control the memory requirements of LMA scale as $\mathcal{O}(N)$.

14.4 SUMMARY

TL;DR: subspace deflation summary

We have generalized the idea of low-mode averaging to a deflation of an arbitrary subspace. This led to a definition of linear operators restricted to that subspace which we refer to as coarsenings or coarse-grid operators. This chapter introduced the deflation of arbitrary subspaces specified by an orthonormal basis set of spinor fields. To move data from the fine to the coarse subspace and vice versa, we specified inter-grid operators referred to as restrictor and prolongator. This led to the observation that low-mode averaging is variant of subspace deflation with a specific basis choice making the coarse operator diagonal. In the next chapter we introduce the decomposition of the lattice into blocks. This will make it clear, why the Hilbert space $\hat{\mathcal{V}}$ was called "coarse".

15

LOCAL COHERENCE AND BLOCK DECOMPOSITION

Teach your algorithm physics and it will perform better.
— Unknown

This section/chapter was proofread 2 times.

TODO: * local coherence * Local coherence * block decomp -> subspace deflation with non-trivial lattice index

TL;DR: lc intro

In chapter 14, we have introduced a subspace with spinors and operators acting on it. We did not yet attempt to interpret its structure. In this chapter – with the concept of local coherence of low quark modes [218], also known as weak approximation property [276] – we will introduce the coarse lattice (section 15.1). Referring to operators restricted to these subspaces as *coarse* operators will then be manifest. The degree to which the Wilson-Clover Dirac operator exhibits local coherence of low modes is quantified numerically section 15.2 and preliminary consideration about the coarse spin structure are made (section 15.3) before summarizing the chapter (section 15.4).

15.1 DEFINITION AND COARSE LATTICE

TL;DR: free theory low modes and staircase functions

In momentum space, the low modes of the Dirac operator in the free quark theory are sharp delta functions at small momenta $|p|$ making them smooth, slowly oscillating plane waves in position space x, whose amplitude only depends on p and is spread over the whole lattice. A good approximation for them in position space are constant spinor fields. In order to include the slow oscillations, we may project these fields to a block decomposition of the lattice whose block size catches well the resolution of the wave, see fig. 15.1. The actual modes can then be approximated by staircase-shaped functions constant on each block. The smaller the block size, the smoother the approximation, and in the limit of the block size approaching the lattice spacing a on all 4 extents, we retain the exact form of the modes on the lattice.

TL;DR: lc definition

These considerations hold in the free theory, but in the interacting theory with dynamic gauge fields the situation looks different. Inspired by the free theory, the property of local coherence of some linear independent set of

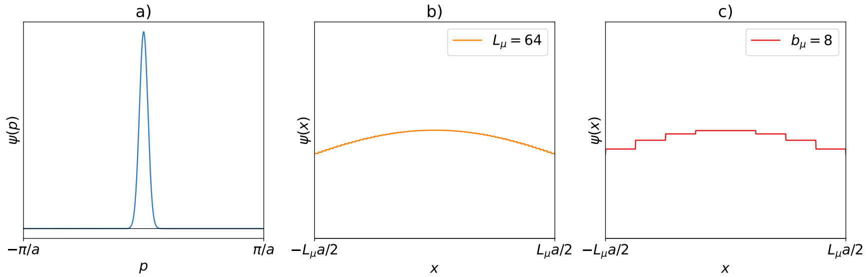


FIGURE 15.1: Quantitative picture of low modes in the free theory. Panel a) shows the sharp peak in momentum space centered around zero, whereas panel b) shows the same mode in position space on a finite lattice. Panel c) shows the approximation when using constant modes projected to blocks. The fact that the low modes are flat in position space makes the approximation suitable.

fields $\mathcal{M}$, states that these fields are to a good approximation contained in the span of another (possibly larger) set of fields $\Pi(\mathcal{B})$ generated by block projecting a small bootstrap set $\mathcal{B} \subset \mathcal{M}$, symbolically

$$|\mathcal{B}| \ll |\Pi(\mathcal{B})| \,, \qquad \mathcal{B} \xrightarrow[\text{projection}]{\text{block}} \Pi(\mathcal{B}) \,, \qquad \mathcal{V}_{\mathcal{B}} \subset \mathcal{V}_{\Pi(\mathcal{B})} \,, \tag{15.1}$$

where $\mathcal{V}_{\mathcal{A}}$ is the vector space generated by the span of the vectors $\mathcal{A}$, $\mathcal{V}_{\mathcal{A}} = \operatorname{span}\{\mathcal{A}\}$. The key equation describing the property is that

$$\mathcal{V}_{\mathcal{M}} \underset{\sim}{\subsetneq} \mathcal{V}_{\Pi(\mathcal{B})} \tag{15.2}$$

holds up to small deficits. Local coherence of low modes means that the $N \gg \hat{N}_c$ lowest modes of the Dirac operator can be well represented using a block-projected basis out of the $\hat{N}_c$ lowest modes.

TL;DR: block projection and coarse lattice

A few things still need to be formalized for the above procedure, for instance the block projection. We start by considering a block decomposition of the full spacetime lattice Λ into $\hat{V} \in \mathbb{N}$ equally-sized disjoint blocks

$$\Lambda = \{(x_0, x_1, x_2, x_3) \mid 0 \leq x_\mu < L_\mu\} = \bigcup_{\hat{y} \in \hat{\Lambda}} B_{\hat{y}} \,. \tag{15.3}$$

The block extents are multiples of the lattice extents (no summation over μ)

$$L_\mu = b_\mu \hat{L}_\mu \,, \qquad b_\mu, \hat{L}_\mu \in \mathbb{N} \,. \tag{15.4}$$

They are defined to contain the subset of lattice points they cover

$$B_{\hat{y}} = \{x \in \Lambda \mid x_\mu - \hat{y}_\mu b_\mu < b_\mu\} \,, \tag{15.5}$$

parametrized by their position $\hat{y}$ in the block grid. The $\hat{y}$ are new coarse coordinates in a coarse lattice $\hat{\Lambda}$ with coarse extents $\hat{L}_\mu$ defined analogous to the fine-grid lattice,

$$\hat{\Lambda} = \{\hat{y} = (\hat{y}_0, \hat{y}_1, \hat{y}_2, \hat{y}_3) \mid 0 \leq \hat{y}_\mu < \hat{L}_\mu\} \; . \tag{15.6}$$

The block projection is defined via projectors to the blocks $B_{\hat{y}}$ acting on spinor fields ψ as

TL;DR: block projection def and gram schmidt reorthonormalization

$$P_{\hat{y}}\psi[x] = \begin{cases} \psi[x] & \text{if } x \in B_{\hat{y}} \; , \\ 0 & \text{otherwise.} \end{cases} \tag{15.7}$$

The projected set in eq. (15.1) can now be specified as the reorthonormalized projected spinors

$$\Pi(\mathcal{B}) = \mathcal{GS}\left(\{P_{\hat{y}}\psi \mid \psi \in \mathcal{B} \text{ and } \hat{y} \in \hat{\Lambda}\}\right) \; , \tag{15.8}$$

where $\mathcal{GS}(\mathcal{A})$ is the Gram-Schmidt orthonormalized set provided an arbitrary linear independent set $\mathcal{A}$.

This procedure specifies a second coarser lattice $\hat{\mathbb{L}}$ analogue to section 12.1 but with different degrees of freedom for spin, color and spacetime. As of now, we will interpret the field index $\hat{a}$ in $\psi_{\hat{a}} \in \mathcal{B}$ for some (arbitrary) enumeration of the fields as coarse color. The block index $\hat{y} \in \hat{\Lambda}$ is straightforwardly interpreted as coarse spacetime index and finally the coarse spin is trivial for now. This gives us $(\hat{N}_s, \hat{N}_c) = (1, |\mathcal{B}|)$ as well as a coarse volume as product of the coarse lattice extents and coarse spinors defined on the coarse lattice with coarse internal degrees of freedom given by the coarse spin and colors.

TL;DR: coarse color, coarse spacetime indices

This leads to a redefinition of the restrictor and prolongator, since for every field in $\Pi(\mathcal{B})$ one can associate a coarse lattice index $\hat{x} \in \hat{\Lambda}$ and a coarse color $\hat{a} \in \hat{\mathfrak{C}} = \{0, \dots, \hat{N}_c - 1\}$. The restrictor then acts on a fine-grid spinor ψ as

$$(R\psi)[\hat{x}]_{\hat{a}} = (\phi_{\hat{x},\hat{a}}, \psi) \; , \quad \phi_{\hat{x},\hat{a}} \in \Pi(\mathcal{B}) \; , \tag{15.9}$$

whereas the prolongator acts on a coarse-grid spinor $\hat{\psi}$ as

$$T\hat{\psi} = \sum_{\hat{x}\in\hat{\Lambda}} \sum_{\hat{a}\in\hat{\mathfrak{C}}} \hat{\psi}[\hat{x}]_{\hat{a}} \phi_{\hat{x},\hat{a}} \; . \tag{15.10}$$

The field index i from eqs. (14.1) and (14.2) just became a multi-index $i = (\hat{x}, \hat{a})$. We note that restrictor and prolongator are block diagonal and applying them involves no communication.

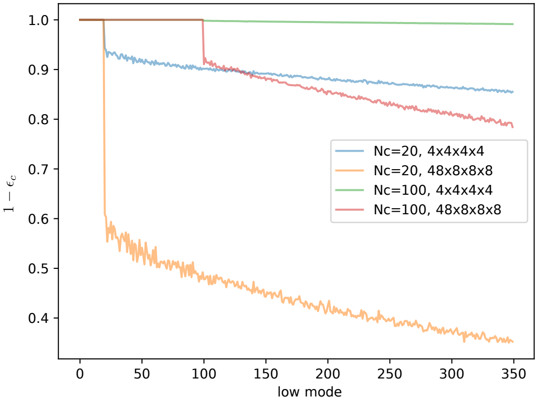


FIGURE 15.2: Local coherence of the low modes of Q tested on ensemble F7 of global lattice size $(L_0, L_1, L_2, L_3) = (96, 48, 48, 48)$, see table 17.1. The figure shows the value of $1 - \epsilon_i = \|P\xi_i\|$ on the y-axis versus the eigenmode number i on the x-axis, where ξ_i is the i-th normalized lowest mode for two different values of N_c and block sizes. This plot is taken from publication [P1].

15.2 QUALITY OF APPROXIMATION

TL;DR: deficit

The quality of the approximation can be quantified with the deficit

$$\epsilon_\psi = \frac{\|(1-P)\psi\|}{\|\psi\|} \in [0,1] \,, \qquad \psi \in \mathcal{M} \,, \tag{15.11}$$

where P is the Hermitian projector to the block projected subspace,

$$P = \sum_{\phi \in \Pi(\mathcal{B})} \phi\phi^\dagger \,. \tag{15.12}$$

TL;DR: deficit plotted, we observe lc

We showcase local coherence of low modes of the Hermitian Dirac operator $Q = \gamma^5 D$ in fig. 15.2. The plot shows the fraction of the 350 lowest modes that are contained in the blocked subspace constructed from only the $\hat{N}_c = 20, 100$ lowest ones, $\mathcal{B} = \{\xi_i \mid 0 \leq i < \hat{N}_c\}$. The modes were determined up to relative precision of $\|Q\xi - \lambda\xi\| / \|\xi\| \leq 10^{-12}$ for an eigenmode ξ with eigenvalue λ. Different number of low modes $\hat{N}_c$ and block extents where used for the bootstrap set to generate subspaces of differing sizes. Block sizes of $b_\mu = 4, 8$ correspond to physical extents of $b_\mu a \approx 0.25\,\text{fm}$ and $b_\mu a \approx 0.5\,\text{fm}$ respectively. As analytically expected, the spinors in the bootstrap set are represented exactly with no deficit, i. e. $1 - \epsilon_i = 1$ for $\psi \in \mathcal{B}$. Spinors in $\mathcal{M} \setminus \mathcal{B}$ on the other hand have deficits larger than zero, but nevertheless they are well represented. The effect of a finer block grid or an increase in low modes results in better overlaps as expected. Even with a very modest number of $\hat{N}_c = 20$ low modes and a block size of 4^4, we already contain $85 - 90\%$ of the 350 lowest modes. Notice that for $\hat{N}_c = 100$ and block size of 4^4, the dimension $\hat{d}$ of the coarse-grid operator $\hat{Q}$ is already 3.2% of the dimension d of the fine-grid Dirac operator Q.

We note that this property has been crucial to create subspaces with inexact rough low modes to obtain powerful preconditioners for Krylov subspace solvers, called inexact deflation [218] or multigrid [276]. Furthermore, the property has been used to accelerate the Lanczos algorithm to generate low modes [277]. The extension of what will follow to using modes determined with a low accuracy is straightforward as we will not assume any special properties of the fields in the bootstrap set.

TL;DR: used in inexact deflation and MG algos

Finally we briefly revisit low-mode averaging one more time to show that it is a pathological case of the above. The projector P used in low-mode averaging, see eq. (13.4), can be obtained by a block extent equal to the fine-grid lattice extent of $b_\mu = L_\mu$, i. e. one big block covering the whole lattice, making the coarse lattice trivial with a volume $\hat{V} = 1$. It leaves only non-trivial coarse color degrees of freedom. LMA has $\Pi(\mathcal{B}) = \mathcal{B}$ and the deficits of modes not in the bootstrap set will be maximal,

TL;DR: LMA as trivial blocking

$$\epsilon_\psi = 1 \, , \qquad \psi \in \mathcal{M} \setminus \mathcal{B} \, . \tag{15.13}$$

Therefore, fig. 15.2 for that choice will be a shifted Heaviside function as $\theta(\hat{N}_c - 1 - i)$ where

$$\theta(i) = \begin{cases} 1 & i \geq 0 \, , \\ 0 & i < 0 \, . \end{cases} \tag{15.14}$$

It is instructive to shortly touch upon the other extreme, where the block extents are as *small* as possible $b_\mu = 1$. This choice results in a coarse lattice geometry equal to the fine one, $\hat{\Lambda} = \Lambda$. A valid choice as long as $\hat{N}_c \leq 12$, because else the Gram-Schmidt reorthonormalization will certainly fail. The coarsening then only affects the internal degrees of freedom of a spinor field. If $\hat{N}_c = 12$ we obtain exactly the same lattice, $\hat{\mathbb{L}} = \mathbb{L}$, possibly rotated mixing spins and colors but with the same physics contents.

TL;DR: extreme blocking as identity

15.3 SPIN DEGREES OF FREEDOM

TL;DR: spins intro

For the sake of completion, we will discuss briefly the possibility to preserve spin degrees of freedom on the coarse grid, while its theoretical analysis will be deferred to its own chapter, see chapter 18.

TL;DR: spin indices and preserving them

Wilson Dirac spinors have 4 spin degrees of freedom. Two of them are chiral the other two non-chiral and the Dirac spin is the tensor product of those. For the spin index set Ξ, we can write

$$\Xi = \mathfrak{X} \otimes \{0,1\}\,, \qquad \mathfrak{X} = \{+,-\}\,, \tag{15.15}$$

where $\mathfrak{X}$ contains the positive and negative chiralities. Based on that, we can either preserve the two chiralities, the two other spin indices, all four spin indices explicitly or non of them on the coarse subspace, $\widehat{N_s} = 1, 2$ or 4. The $\widehat{N_s} = 1$ case is already discussed above. The other options are readily carried into execution by defining projectors for them.

15.3.1 *Chirality preservation*

TL;DR: preserving the chiral ones with the chiral projectors

By defining the chiral projectors

$$P_\pm = \frac{1}{2}\left(\mathbb{1} \pm \gamma^5\right)\,, \qquad P_\pm^2 = P_\pm = P_\pm^\dagger\,, \qquad P_\pm P_\mp = 0\,, \tag{15.16}$$

we can easily extent the construction in chapter 15 to include these projectors. Consider the set extension

$$\mathcal{B} \longmapsto P_+\mathcal{B} \cup P_-\mathcal{B}\,. \tag{15.17}$$

This will preserve the chiral structure of a spinor projected to the subspace by the projector in eq. (15.12) leading to an important but simple equation

$$[P, \gamma^5] = 0\,. \tag{15.18}$$

The coarse lattice still exposes these chiral indices $\widehat{N_s} = 2$ with $\hat{\Xi} = \mathfrak{X}$. We will see in chapter 18 that this choice has huge impact in the coarse operator condition number and the variance contribution of the two-point correlator and that eq. (15.18) alone is not sufficient.

TL;DR: preserving the chiral ones with LR, singular vectors

Equation (15.18) can be achieved using other projectors too. Instead of the chiral projectors, another valid choice would be

$$P_0 = \mathbb{1}\,, \qquad P_1 = \gamma^5\,, \tag{15.19}$$

although P_1 is not a projector, since $(\gamma^5)^2 = \mathbb{1}$. However, this particular choice results in an equivalent subspace as produced by eq. (15.16). It has a nice interpretation though, meaning that if we use eigenvectors of $\gamma^5 D$ and apply P_0 and P_1 we obtain left and right singular vectors of D.

15.3.2 *Spin preservation*

Alternatively one can preserve all 4 spin degrees of freedom explicitly by defining the (basis dependent) spin projectors

TL;DR: preserve all 4 spins

$$(P_\sigma)_{[\alpha\beta]} = \delta_{\alpha\beta}\delta_{\alpha\sigma}\,, \qquad P_\sigma^2 = P_\sigma = P_\sigma^\dagger\,, \tag{15.20}$$

which project to spin $\sigma \in \Xi$. Analogous to before we can extend the bootstrap set by projecting all vectors

$$\mathcal{B} \longmapsto \bigcup_{\sigma\in\Xi} P_\sigma \mathcal{B}\,. \tag{15.21}$$

In this case, the coarse lattice has all the spin indices preserved $\widehat{N_s} = 4$. The fine and the coarse spins are equal $\hat{\Xi} = \Xi$.

15.4 SUMMARY

We have introduced the property of local coherence of low modes stating that low modes of the Dirac operator are well approximated by locally constant modes. A property that is exploited in efficient preconditioning methods for Krylov subspace solvers. Such modes can be obtained by block projection which leads to a new coarser lattice by interpreting the block spacetime index as coarse lattice index. Field indices from the bootstrap set can be interpreted as coarse color degrees of freedom. In this framework, low-mode averaging imposes a trivial spacetime lattice with a single coarse spacetime index.

TL;DR: lc summary

16

MULTIGRID

> *It is a tale told by an idiot, full of sound and fury signifying nothing.*
>
> — William Shakespeare, Macbeth

This section/chapter was proofread 3 times.

TODO: * Multigrid * Local coherence * block decomp -> subspace deflation with non-trivial lattice index * MG decomposition -> as subspace defl., LMA as subsp. defl. with trivial lattice index * recursion * solver precond -> multiplicative vs. additive * MG LMA * prop decomp * Estimator for 2pt vector corr.

TL;DR: mg intro

In this chapter we will discuss multigrid as a concept and its prominent role as a solver preconditioner (section 16.1). After that, we will construct a decomposed propagator in terms of contributions from multigrid levels (section 16.2). By inserting this decomposition into the connected two-point correlator introduced in chapter 12, it enables us to define a new hierarchical stochastic sampling scheme, where every multigrid level is evaluated with separate statistics. This defines a new class of variance reduction methods which we call *multigrid low-mode averaging* (section 16.3). The method will be applied to the two-point correlator (section 16.4) and the section finished with a summary (section 16.5).

16.1 MULTIGRID PRECONDITIONING

TL;DR: solve ill Dirac eq, Krylov solvers suppress low mode components

As in part i, we are interested to solve the Dirac equation numerically,

$$D\psi = \eta \ , \tag{16.1}$$

possibly for many different right-hand sides η. A suitable class of iterative solvers for this kind of problem are Krylov subspace solvers [130, 131]. We will additionally assume the Dirac operator D to be non-normal, non-Hermitian[1] and close to singular, i. e. having a large condition number. This is a scenario that occurs often in simulations in the chiral regime, where chiral symmetry is spontaneously broken. The presence of eigenvalues near the complex origin (in the former called *low modes* or *near-null vectors*) pose problems when it comes to numerical inversion. The error of the iterative solution is then dominated by the low-mode contributions of the operator D which converge the slowest or stall entirely. This can be easily seen, from the iterative solution ψ from a Krylov solver, that is an implicitly generated polynomial of finite degree n in the operator whose coefficients depend on the right-hand side η.

---

[1] More important properties of the Wilson and Wilson-Clover Dirac operator will be delayed to chapter 18.

TL;DR: show explicitly the low mode contributions

Let us assume for simplicity that the operator is the Hermitian Dirac operator $Q = \gamma^5 D$, then the polynomial applied to the right-hand side is

$$\psi = Q^{-1}\eta \approx \mathcal{P}_n^{(\eta)}(Q)\eta \,, \qquad \mathcal{P}_n^{(\eta)}(x) = \sum_{i=0}^{n} \alpha_i^{(\eta)} x^i \,. \tag{16.2}$$

If we only keep the highest order term of the polynomial and write η as linear combination of eigenvectors of Q, we find

$$\psi \approx \sum_{i=0} \alpha_i Q^n \xi_i + \mathcal{O}\left(Q^{n-1}\right) = \sum_{i=0} \alpha_i \lambda_i^n \xi_i + \mathcal{O}\left(Q^{n-1}\right) \,. \tag{16.3}$$

The lowest mode components of the solution vector are suppressed, whereas the high mode components are amplified. This is a peculiarity of Krylov solvers, as they generate a solution in the Krylov subspace [278, 279]

$$\mathcal{K}_n(Q,\eta) = \operatorname{span}\left\{\eta, Q\eta, Q^2\eta, \ldots, Q^n\eta\right\} \,, \tag{16.4}$$

whereas the vectors $Q^m\eta$ have suppressed low mode components for increasing m.

TL;DR: instead: amplify low-mode components

In order to complement a Krylov solver, the above analysis shows that a preconditioner for ill-conditioned systems should preferably amplify and not suppress the low-mode components of the solution vector. The stalling of Krylov solvers due to near-singularity of the Dirac operator is sometimes referred to as critical slowing down near the chiral limit.

16.1.1 V^2-*problem*

TL;DR: V2 problem of solvers, determining null vectors

Related to the problem in the near chiral limit is the V^2 problem [218] which we have already touched in chapter 11 when reviewing the variance reduction method low-mode averaging. It states that the dimension of the low mode subspace scales linear in the volume of the lattice [217]. This is the main motivation behind preconditioning methods such as multigrid [276]. The idea is to generate an approximate subspace having large overlaps with the true low mode subspace that is comparably cheap to obtain. This is not possible with traditional low mode deflation methods, as their cost, memory and storage requirements scale as $\mathcal{O}(V^2)$. This is a variant of critical slowing down. The property of local coherence (chapter 15) is crucial for a computationally cheap generation of such a subspace. For this type of preconditioning one generates approximate low mode vectors by either solving the homogeneous system[2] with $\eta = 0$

$$D\xi = 0 \,, \tag{16.5}$$

or inverse iterate a random right-hand side η

$$\xi = \left(D^{-1}\right)^{m} \eta \,. \tag{16.6}$$

Both are done to very low precision, where m is 5 to 10. The numerical precision of such modes is in the regime 0.1.

The setup phase consist of a definition of a bootstrap set $\mathcal{B}$ that consists of $N_c = 10 - 30$ approximate low modes and a description of a coarse lattice and spin degrees of freedom as in eq. (15.3) and section 15.3 respectively. Using local coherence, we can thus generate powerful large subspaces at low cost having large overlap with the true low mode subspace as seen in fig. 15.2 using only a few approximate low modes. Therefore, we have a prolongator T, a restrictor R and a coarse Dirac operator $\hat{D} = RDT$. Instead of solving eq. (16.1), we solve the left-preconditioned system

TL;DR: bootstrap set, MG left preconditioning, smoothing

$$LD\psi = L\eta \,, \qquad L = T\hat{D}^{-1}R \,. \tag{16.7}$$

In every solver iteration the coarse system

$$\hat{D}\hat{\psi} = \hat{\eta} \,, \tag{16.8}$$

for some $\hat{\eta}$ has to be solved at low precision with a very lax sopping criterion or with a fixed number of iteration steps, optionally with some pre- and post-smoothing.

The coarse operator $\hat{D}$ exhibits a blocked structure just as the fine-grid Dirac operator D, where nearest neighbor interactions appear with respect to the coarse lattice $\hat{\Lambda}$, see fig. 16.1. The solver algorithm for the coarse system can be the same as for the fine system. Meaning that one can

TL;DR: coarse and fine ops have same sparsity structure, multiplicative preconditioning

---

[2] The homogeneous system has no solution if D is invertible. One runs a fixed number of iterations of some solver, for instance BiCGSTAB. The produced solution is then rich in low modes, i. e. the slow-to-converge directions.

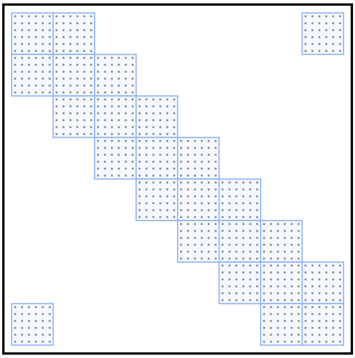


FIGURE 16.1: 1D example ($\hat{V} = 8$) of the structure of the Dirac operator D and its coarse-grid representation $\hat{D}$ using periodic boundary conditions. Each block is of size $N_c N_s \times N_c N_s$. Nearest neighbor interactions of D make neighboring blocks of $\hat{D}$ occupied. Notice that in 4D every block has 8 neighbors. The matrix has $(2d+1)\hat{V}N_c^2 N_s^2$ non-zero entries, where d is the dimensionality of spacetime. This plot is taken from publication [P1].

left-precondition the coarse system as well with an even coarser lattice. Multigrid is meant to be applied recursively by introducing multiple coarse lattices called levels. This is obtained by just applying multigrid preconditioning to the coarse system eq. (16.8) itself. We will refer to eq. (16.7) as *multiplicative* preconditioning.

TL;DR: solve coarsest system exactly

The coarsest system may be so small that it can be solved exactly to machine precision. Many multigrid implementations have a fixed coarsest grid with a trivial spacetime index set $|\hat{\Lambda}| = 1$. The corresponding coarse operator has a small dimension only given by the coarse spin and color degrees of freedom $N_s N_c = \mathcal{O}(10)$ and its inverse can be held in memory up to machine precision.

TL;DR: lüscher deflation, additive preconditioning

It is worth discussing a similar alternative to multigrid called inexact deflation developed independently in ref. [218]. It can be seen as a special case of multigrid, although there are some differences. The subspace is built similarly, except that the spin degrees of freedom are coarsened away, $N_s = 1$. Apart from this the preconditioning *subtracts* the low modes by solving

$$LD\chi = L\eta \,, \qquad L = 1 - DT\hat{D}^{-1}R \tag{16.9}$$

for χ followed by reconstructing the full solution as

$$\psi = \chi + T\hat{D}^{-1}R\eta \,. \tag{16.10}$$

The above will be referred to as *additive* preconditioning, compare multiplicative preconditioning eq. (16.7). We note that LD is non-invertible. This is not an issue as long as the full solution is consistent, i. e. there exist a ψ that solves $D\psi = \eta$ [280]. There are infinitely many solutions χ. Nevertheless the full solution vector ψ is unique after projecting in eq. (16.10).

Both inexact deflation and multigrid, have a very flat scaling behavior in the pion mass and are scaling close to linear in the lattice volume $\mathcal{O}(V)$. This makes them ideal preconditioners for ill-conditioned systems. Although they have an expensive setup phase for generating the coarse subspaces, when solving for many right-hand sides this cost is amortized quickly.

TL;DR: flat in pion mass and linear in V

The coarse system is restricted to the low mode subspace of the operator. Evidently, if the subspaces are generated from low modes, we clearly reduce the error components of the solution vector ψ associated to the low modes and the bias of Krylov solvers towards suppressing low-mode components as described above is solved.

TL;DR: V2 problem solved

16.1.2 *Smoothing*

Multigrid preconditioning usually comes with the option of pre- and post-smoothing. In the preconditioner eq. (16.7), one can add two operators

TL;DR: definition of pre and post smoothers

$$F_{\text{pre}} \colon \mathcal{V} \longrightarrow \mathcal{V}\,, \tag{16.11}$$

$$F_{\text{post}} \colon \mathcal{V} \longrightarrow \mathcal{V}\,, \tag{16.12}$$

to perform some sort of smoothing before and after the inversion on the coarse grid. The left preconditioner then turns into

$$L = F_{\text{post}} T \hat{D}^{-1} R F_{\text{pre}} \tag{16.13}$$

in every iteration step before adding the correction to the solution vector.

Smoothing usually consists of a few steps of a local solver, for instance a GCR on local blocks of the lattice. The effect is a reduction of the high mode components of the solution vector, since multigrid only affects the low mode components. It suffices for the solver to be block local, since the high modes have high frequencies – they do not require global information to reduce in every step. Local solvers strong scale ideally, because of the suppressed communication and locality and they can run on all blocks in parallel. Such solvers are sometimes referred to as *communication-avoiding* solvers [281], for example CA-GCR.

TL;DR: communication avoiding solver for smoothing

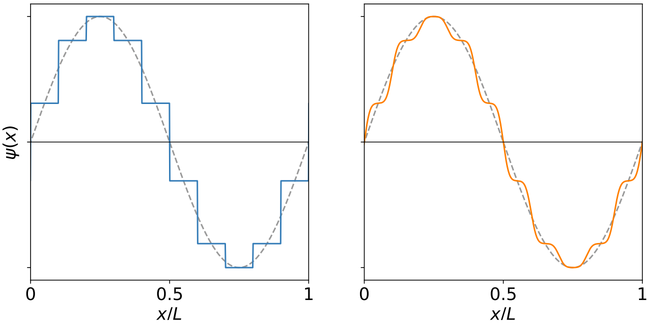


Figure 16.2: The effect of post-smoothing. The prolongation of the solution vector after a coarse-grid solve might look similar to the blue line in the right panel. Post-smoothing has the effect of filing off the edges of the staircase-shaped prolonged solution as depicted by the yellow line in the left panel. The prolonged error correction is then closer to the actual solution on the finer level (dashed gray line in both panels) and thus of higher quality.

TL;DR: Illustration of post smoothing

Pre-smoothing aims to reduce the high frequency components of the current error before restricting, whereas post-smoothing aims to dampen those high frequency modes introduced during interpolation. Both types of smoothing are introduced to balance the reduction in the error of high and low mode components. Without smoothing the high mode components might not be reduced properly. See fig. 16.2 for an illustration of the effect of post-smoothing.

16.1.3 *Summary*

TL;DR: intuitive argument

Multigrid preconditioning of the Dirac equation as $LD\psi = L\eta$ can be seen through the following intuitive argument. The Dirac operator is a nearest neighbor stencil. In every iteration it is applied to the current solution vector. Information in the current solution vector on a certain lattice site slowly propagates through the lattice; one site per iteration in each direction. Even though the coarse Dirac operator is still nearest neighbor, information on the coarse grid travels faster through the lattice, because sites are further apart (i. e. one coarse grid hop corresponds to multiple fine-grid hops depending on the choice of block size). This enhances non-local information exchange on the lattice with fewer iterations on the coarse grid. Additionally the coarse grid is constructed from approximate low modes which are

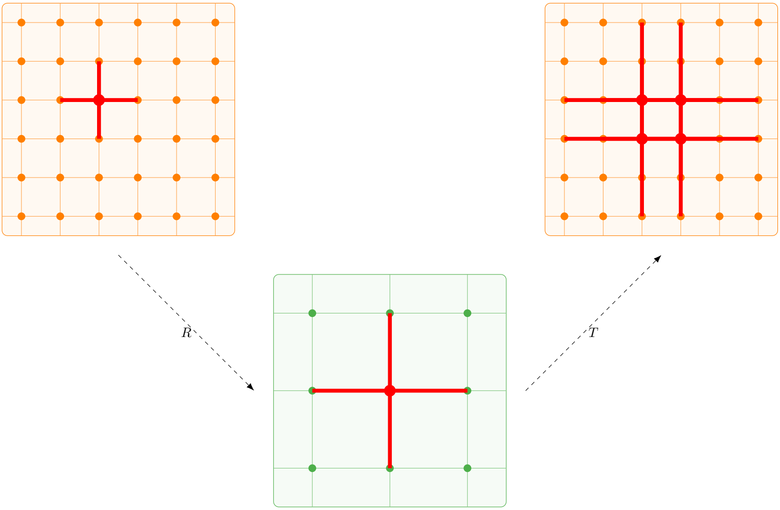


Figure 16.3: Illustration that the coarse grid Dirac stencil transports information further through the lattice. Upper left: Dirac stencil on a 6×6 lattice connecting nearest neighbors. Instead we could restrict the vector to the coarse 3×3 grid, apply the coarse Dirac stencil (bottom center) and prolong back. The result is a longer range interaction (upper right). The information transfer is restricted to the coarse degrees of freedom, i. e. the low modes of the fine grid which in turn are associated to long-range interactions. Therefore, not *all* information has propagated the longer distance but the most relevant information has.

associated to long-range interactions allowing long-distance propagation of information. This concept is illustrated in fig. 16.3.

16.2 MULTIGRID PROPAGATOR

With the definition of a coarse lattice, we can define a propagator decomposition as

TL;DR: 2-lvl MG prop decomposition

$$S = D^{-1} = \underbrace{D^{-1} - T\hat{D}^{-1}R}_{S_0} + \underbrace{T\hat{D}^{-1}R}_{S_1} \tag{16.14}$$

by adding and subtracting the piece along the coarse subspace. We will generalize this equation in a recursive way by applying it repeatedly to $\hat{D}^{-1}$ introducing N_{lvl} levels of multigrid.

16.2.1 *Recursive notation*

TL;DR: all can be formulated recursively

Before we continue, we will formulate multigrid recursively and shift in notation by defining the ℓ-th recursive coarse Dirac operator as coarsening of the $(\ell-1)$-th coarse Dirac operator[3]

$$D^{(\ell)} = R^{(\ell-1)} D^{(\ell-1)} T^{(\ell-1)} \,, \qquad \ell \in \{1, \dots, N_{\text{lvl}} - 1\} \tag{16.15}$$

where we define level 0 (also denoted as L0) to be the fine grid

$$D^{(0)} = D \,, \qquad R^{(0)} = R \,, \qquad T^{(0)} = T \,. \tag{16.16}$$

For clarity, for $\ell = 1$, the notation changes to $D^{(1)} \equiv \hat{D}$. The inter-grid operators move data from and to vector spaces defined on levels ℓ and $\ell + 1$

$$R^{(\ell)} \colon \mathcal{V}^{(\ell)} \longrightarrow \mathcal{V}^{(\ell+1)} \,, \tag{16.17}$$

$$T^{(\ell)} \colon \mathcal{V}^{(\ell+1)} \longrightarrow \mathcal{V}^{(\ell)} \,. \tag{16.18}$$

We therefore have analogous to eq. (16.14) the full propagator on level $\ell = 0, \dots, N_{\text{lvl}} - 2$,

$$\left(D^{(\ell)}\right)^{-1} = \left\{ \left(D^{(\ell)}\right)^{-1} - T^{(\ell)} \left(D^{(\ell+1)}\right)^{-1} R^{(\ell)} \right\} + T^{(\ell)} \left(D^{(\ell+1)}\right)^{-1} R^{(\ell)} \,, \tag{16.19}$$

with $\ell = 0$ being exactly equal to eq. (16.14). For convenience we define the compound inter-grid operators moving data directly between levels 0 and $\ell > 0$ as

$$\mathcal{R}^{(\ell)} = R^{(\ell-1)} \cdots R^{(0)} \;:\; \mathcal{V}^{(0)} \longrightarrow \mathcal{V}^{(\ell)} \,, \tag{16.20}$$

$$\mathcal{T}^{(\ell)} = T^{(0)} \cdots T^{(\ell-1)} \;:\; \mathcal{V}^{(\ell)} \longrightarrow \mathcal{V}^{(0)} \,, \tag{16.21}$$

with base cases $\mathcal{R}^{(0)} = \mathcal{T}^{(0)} = \mathbb{1}$ for $\ell = 0$. The recursive notation is depicted in fig. 16.4.

TL;DR: smoothing in recursive notation more powerful

In this notation, pre- and post-smoother can be added to every level separately by extending the restrictors and prolongators,

$$R^{(\ell)} \longrightarrow R^{(\ell)} F^{(\ell)}_{\text{pre}} \,, \tag{16.22}$$

$$T^{(\ell)} \longrightarrow F^{(\ell)}_{\text{post}} T^{(\ell)} \,, \tag{16.23}$$

---

[3] The notation with the hat is clean and concise but as in real life, wearing multiple hats is awkward – $\hat{\hat{\hat{D}}}$.

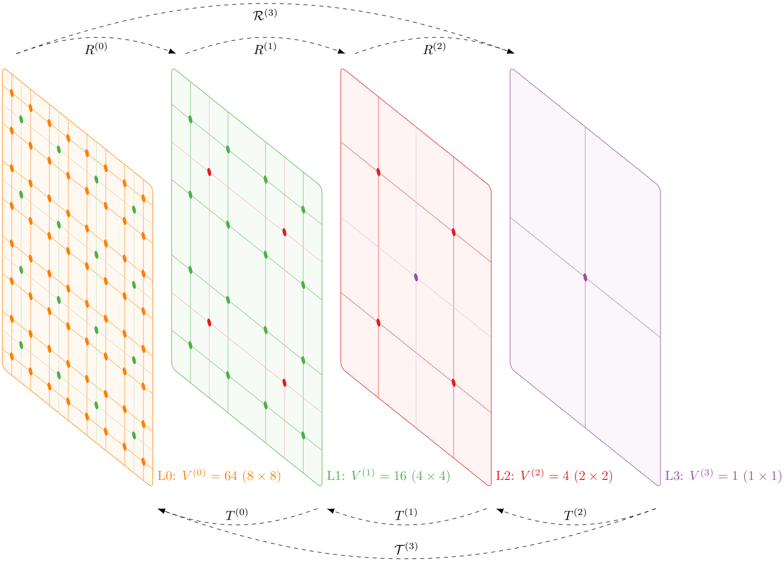

$\mathcal{R}^{(3)}$
$R^{(0)}$
$R^{(1)}$
$R^{(2)}$
L0: $V^{(0)} = 64$ (8×8)
L1: $V^{(1)} = 16$ (4×4)
L2: $V^{(2)} = 4$ (2×2)
L3: $V^{(3)} = 1$ (1×1)
$T^{(0)}$
$T^{(1)}$
$T^{(2)}$
$\mathcal{T}^{(3)}$

FIGURE 16.4: 2D illustration of a recursive lattice coarsening. The finest lattice on level 0 in orange has a size of $V^{(0)} = 64$. The block size for level 1 in green is $b_0 \times b_1 = 2 \times 2$ resulting in a coarse lattice size of $V^{(1)} = 16$. The coarsest lattice in purple has a trivial lattice size of $V^{(3)} = 1$ and thus corresponds to LMA, where no blocking is done. The inter-grid operators $R^{(\ell)}$ and $T^{(\ell)}$ move data on adjacent levels, whereas the compound inter-grid operators $\mathcal{R}^{(\ell)}$ and $\mathcal{T}^{(\ell)}$ move data between levels 0 and ℓ. This plot is taken from publication [P1].

with smoother acting on level ℓ,

$$F^{(\ell)}_{\text{post}} \colon \mathcal{V}^{(\ell)} \longrightarrow \mathcal{V}^{(\ell)} \,, \tag{16.24}$$

$$F^{(\ell)}_{\text{pre}} \colon \mathcal{V}^{(\ell)} \longrightarrow \mathcal{V}^{(\ell)} \,. \tag{16.25}$$

16.2.2 *Propagator decomposition*

Now, we are ready to discuss a propagator decomposition using N_{lvl} levels of multigrid. A general decomposition into N_{lvl} levels as in eq. (16.14) can be obtained by repeatedly plugging eq. (16.19) into the full propaga-

TL;DR: N-lvl MG prop decomposition

tor $S = D^{-1}$. With the notation from the previous section, we obtain a decomposition as

$$S = \sum_{\ell=0}^{N_{\text{lvl}}-1} S_\ell \,, \tag{16.26}$$

into differences on the inner levels $\ell < N_{\text{lvl}} - 1$,

$$S_\ell = \mathcal{T}^{(\ell)} \left(D^{(\ell)}\right)^{-1} \mathcal{R}^{(\ell)} - \mathcal{T}^{(\ell+1)} \left(D^{(\ell+1)}\right)^{-1} \mathcal{R}^{(\ell+1)} \,, \tag{16.27}$$

and a single contribution on the coarsest level

$$S_{N_{\text{lvl}}-1} = \mathcal{T}^{(N_{\text{lvl}}-1)} \left(D^{(N_{\text{lvl}}-1)}\right)^{-1} \mathcal{R}^{(N_{\text{lvl}}-1)} \,. \tag{16.28}$$

The equations above describe a telescoping sum, but we may consider every difference separately.

16.2.3 *A note on recursion*

TL;DR: loop formulation more flexible

As formalized, every subspace defined via its lattice $\mathbb{L}^{(\ell)}$ is a valid coarsening of all its previous levels, including the fine-grid lattice $\mathbb{L}^{(0)} = \mathbb{L}$ by defining a certain block size relative to the next finer level. A relaxed formulation of the decomposition above may allow coarse lattices to be only valid coarsenings of the fine-grid lattice. This gives slightly higher flexibility in the choice of coarse lattices.

TL;DR: example of recursive vs. loop

As an example consider a fine-grid lattice with extents that are not powers of two, for instance $L_\mu = 48$. The first level of multigrid might be chosen by a block of size $b_\mu^{(1)} = 4$ resulting in a coarse lattice of extents $L_\mu^{(1)} = 48/4 = 12$. At this point, with the recursive formulation, we do not have much freedom to choose the next levels. In table 16.1, we collected possible levels in such a scenario for the recursive and the relaxed formulation. We see the relaxed formulation allows coarse levels that are not valid coarsenings of each other, for instance L2 and L3 in table 16.1, but all are valid coarsenings of the fine grid.

TL;DR: weird for solvers, but good for MG LMA

The relaxed formulation might not be meaningful in a solver preconditioning scenario, where inter-grid data movement among subsequent levels is crucial for overall performance. In terms of a propagator decomposition it indeed makes sense, not only because it provides more flexibility to fine-tune the evaluation of an observable as seen later, but also because we usually move data from and to level 0 and other levels. The two variants

	Recursive notation		Relaxed notation	
Level	Volume	Block size	Volume	Block size
L0	$V = 48^4$	-	$V = 48^4$	-
L1	$V^{(1)} = 12^4$	$b^{(1)} = 4^4$	$V^{(1)} = 12^4$	$b^{(1)} = 4^4$
L2	$V^{(2)} = 6^4$	$b^{(2)} = 8^4$	$V^{(2)} = 8^4$	$b^{(2)} = 6^4$
L3	$V^{(3)} = 1^4$	$b^{(3)} = 48^4$	$V^{(3)} = 6^4$	$b^{(3)} = 8^4$
L4	-	-	$V^{(4)} = 4^4$	$b^{(4)} = 12^4$
L1	-	-	$V^{(5)} = 1^4$	$b^{(5)} = 48^4$

TABLE 16.1: Example of the recursive and relaxed formulation of multigrid. The block size is denoted with respect to the fine grid. The relaxed formulation allows more choice in coarse spacetime volumes.

also differ when it comes to smoothing. The recursive variant allows control of pre- and post-smoother between all levels separately, whereas the relaxed variant only allows pre- and post-smoothing on level 0.

Implementation effort as well as performance of the two variants clearly differ and we will not discuss them further. The implementation used for this project is the relaxed variant allowing more freedom in the coarse grid creation, i. e. the one where level 0 is coarsened for all levels. The formulation of the method and its subsequent analysis that will follow are independent of the choice of formulation made at this point.

TL;DR: we use loop variant in implementation

16.3 MULTIGRID LOW-MODE AVERAGING

We are now equipped to formulate a novel variance reduction method to evaluate n-point functions.

TODO: We note that the method will only lead to a fair variance reduction on observables sensitive to low mode noise. The method is expected to be beneficial whenever low-mode averaging is beneficial.

We will use the decomposition of the quark propagator established in the last section, resulting in a hierarchical N_{lvl}-level scheme which naturally arises from the hierarchical decomposition of the fine-grid lattice. For generality we will start with a connected n-point function

TL;DR: n-pt function decomposition, grouping and illustration

$$C(x_1; x_2; \ldots; x_n) = \operatorname{tr}\left\{ S_{[x_1;x_2]}\Gamma_1 S_{[x_2;x_3]}\Gamma_2 \cdots S_{[x_n;x_1]}\Gamma_n \right\} , \tag{16.29}$$

where the Γ_i are monomials of γ-matrices and the S are propagators from lattice points x_i to x_j. Plugging in the multigrid propagator decomposition eq. (16.26), we find $(N_{\mathrm{lvl}})^n$ contributions

$$C_{\ell_1,\ell_2,\dots,\ell_n}(x_1;x_2;\dots;x_n) = \mathrm{tr}\left\{ S_{\ell_1\,[x_1;x_2]}\Gamma_1 S_{\ell_2\,[x_2;x_3]}\Gamma_2 \cdots S_{\ell_n\,[x_n;x_1]}\Gamma_n \right\} , \tag{16.30}$$

indexed by the multigrid levels $\ell_i \in \{0,\dots,N_{\mathrm{lvl}}-1\}$, and $i \in \{1,\dots,n\}$. Clearly we recover the original n-point function eq. (16.29) by simply summing all levels

$$C(x_1;x_2;\dots;x_n) = \sum_{\ell_1=0}^{N_{\mathrm{lvl}}-1} \cdots \sum_{\ell_n=0}^{N_{\mathrm{lvl}}-1} C_{\ell_1,\ell_2,\dots,\ell_n}(x_1;x_2;\dots;x_n) \,. \tag{16.31}$$

Furthermore, we may define the level-k-contribution (also denoted as Lk) to the full correlator by collecting all components of the correlator tensor eq. (16.30), including propagators on level k or coarser but not finer levels, i. e.

$$C_{\mathrm{L}k}(x_1;x_2;\dots;x_n) = \sum_{i=1}^{n} \sum_{\ell_1=k}^{N_{\mathrm{lvl}}-1} \dots \sum_{\ell_i=k}^{k} \dots \sum_{\ell_n=k+1}^{N_{\mathrm{lvl}}-1} C_{\ell_1,\dots,\ell_n}(x_1;x_2;\dots;x_n) \,, \tag{16.32}$$

for $k \in \{0,\dots,N_{\mathrm{lvl}}-1\}$. The sum in ℓ_i is trivial only participating one summand with $\ell_i = k$. Using this grouping of levels, the full correlator eq. (16.29) can be recovered by summing over all levels,

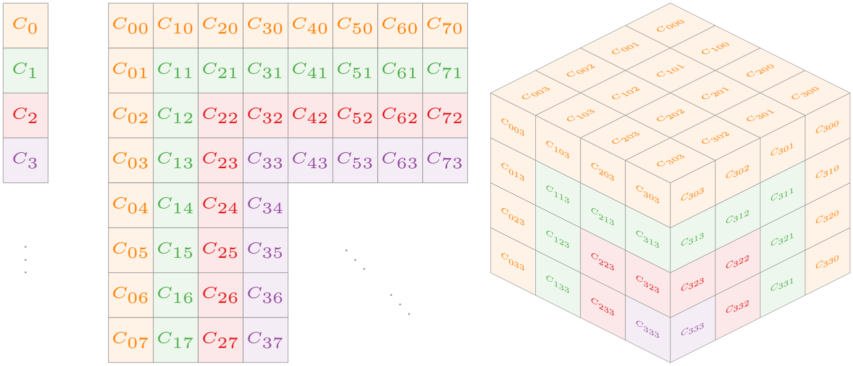
$$C(x_1;x_2;\dots;x_n) = \sum_{k=0}^{N_{\mathrm{lvl}}-1} C_{\mathrm{L}k}(x_1;x_2;\dots;x_n) \,. \tag{16.33}$$

This decomposition into level-contributions comes natural as illustrated in fig. 16.5 for correlator tensors with ranks $n = 1,2,3$. Crucial about this decomposition is that every level can be evaluated separately with a different strategy in terms of number and type of sources. An evaluation of $C_{\mathrm{L}k}$ requires only inversions of the Dirac operator $D^{(k)}$ on level k and coarser levels but not on finer levels $0,\dots,k-1$. $C_{\mathrm{L}0}$ requires fine-grid solves. The fact that the coarse Dirac operators have exactly the same sparsity structure as the fine one (see fig. 16.1) allows to use the same solvers algorithms for inversions.

16.4 TWO-POINT CONNECTED CORRELATOR

TL;DR: 2-pt example simplified heavily

We will continue with the connected bare isovector vector current correlator


C_0
C_1
C_2
C_3
C_{00} C_{10} C_{20} C_{30} C_{40} C_{50} C_{60} C_{70}
C_{01} C_{11} C_{21} C_{31} C_{41} C_{51} C_{61} C_{71}
C_{02} C_{12} C_{22} C_{32} C_{42} C_{52} C_{62} C_{72}
C_{03} C_{13} C_{23} C_{33} C_{43} C_{53} C_{63} C_{73}
C_{04} C_{14} C_{24} C_{34}
C_{05} C_{15} C_{25} C_{35}
C_{06} C_{16} C_{26} C_{36}
C_{07} C_{17} C_{27} C_{37}
C_{000} C_{001} C_{002} C_{003} C_{100} C_{101} C_{102} C_{103} C_{200} C_{201} C_{202} C_{203} C_{300} C_{301} C_{302} C_{303}
C_{013} C_{023} C_{033} C_{113} C_{123} C_{133} C_{213} C_{223} C_{233} C_{313} C_{323} C_{333}
C_{310} C_{311} C_{312} C_{320} C_{321} C_{322} C_{330} C_{331} C_{332}

(a) rank-1 tensor (b) rank-2 tensor (c) rank-3 tensor

FIGURE 16.5: (a) Grouping into contributions of multigrid levels of a 1-point correlator. (b) Grouping into contribution on multigrid levels of a 2-point connected correlator into 8 multigrid levels. Elements of the correlator tensor C_{ij}, eq. (16.30) are assigned to grid levels Lk. The sum of the orange shaded elements correspond to the L0-contribution of the correlator, green to L1, red to L2 etc. Summing all tensor elements is equal to summing all levels and results in the full correlator. (c) Example of a 3-point connected correlator tensor grouping into 4 multigrid levels. This plot is an extended version of a plot taken from publication [P1].

in the time-momentum representation already touched upon in section 12.2 and eq. (12.7) – our target observable. Although the decomposition can be applied to any n-point function as seen above. For a two-point function the formulas simplify heavily. We have the full correlator

$$G(t) = \frac{1}{|\Omega|} \sum_{y\in\Omega} \sum_{\vec{x}\in\Sigma} C(y_0 + t, \vec{x}; y) \ , \tag{16.34}$$

$$C(x;y) = \operatorname{tr}\left\{ S_{[x;y]} \Gamma_1 S_{[y;x]} \Gamma_2 \right\} \ , \tag{16.35}$$

and the rank-2 correlator tensor

$$C_{ij}(x;y) = \operatorname{tr}\left\{ S_{i[x;y]} \Gamma_1 S_{j_{[y;x]}} \Gamma_2 \right\} \ , \qquad i,j \in \{0,\dots,N_{\mathrm{lvl}}-1\} \ . \tag{16.36}$$

The contribution into levels for a two-point function is depicted in fig. 16.5 (b). Therefore, the Euclidean time correlator eq. (12.7) can be written as sum of level contributions

$$G(t) = \sum_{k=0}^{N_{\mathrm{lvl}}-1} G_{\mathrm{L}k}(t) \,. \tag{16.37}$$

16.4.1 *Two-level multigrid LMA*

TL;DR: 2-lvl MG

For only two multigrid levels, $N_{\mathrm{lvl}} = 2$, we have a simple propagator decomposition eq. (16.14) and two level-contributions for the correlator

$$\begin{aligned} C_{\mathrm{L}0}(x;y) &= C(x;y) - C_{\mathrm{L}1}(x;y) \\ &= \operatorname{tr}\left\{ S_{[x;y]}\Gamma_1 S_{[y;x]}\Gamma_2 \right\} - \operatorname{tr}\left\{ S_{1\,[x;y]}\Gamma_1 S_{1\,[y;x]}\Gamma_2 \right\} \,, \end{aligned} \tag{16.38}$$

$$C_{\mathrm{L}1}(x;y) = \operatorname{tr}\left\{ S_{1\,[x;y]}\Gamma_1 S_{1\,[y;x]}\Gamma_2 \right\} \,. \tag{16.39}$$

The Euclidean time correlator enjoys the same decomposition

$$G(t) = G_{\mathrm{L}0}(t) + G_{\mathrm{L}1}(t) \,, \tag{16.40}$$

where we can choose different evaluation strategies for the two terms. Clearly for every sample of the fine-grid L0-term we have to solve the full Dirac equation, $D\psi = \eta$ for some source η, whereas for the coarse-grid L1-term we have to solve the coarse Dirac equation $\hat{D}\hat{\psi} = \hat{\eta}$ for some coarse source $\hat{\eta}$ in the smaller subspace $\hat{\mathcal{V}}$. Solving the coarse system is significantly cheaper than solving the fine-grid system if the subspace was chosen properly. This is, if the coarse dimension is much smaller than the fine one, $\dim_{\mathbb{C}}(\hat{\mathcal{V}}) \ll \dim_{\mathbb{C}}(\mathcal{V})$. Our strategy will be as follows: the expensive L0-term will be evaluated with as few as possible sources, for instance 1 to 10, whereas the cheap L1-term will be sampled with many sources, say 100 to 1000. We will present different estimators for the level contributions, but every estimator that is valid on the fine-grid correlator can be safely used on coarse levels too.

16.4.2 *Point-source estimator on level k*

TODO: todo

16.4.3 *Stochastic estimator on level k*

As introduced in section 12.2.2 we may use a stochastic estimator for level k. By using eq. (12.22) as basis we plug in the propagator decomposition. The final equation only changes slightly by replacing the propagator S with the propagators on the involved levels

TL;DR: stochastic estimator on lvl similar to plain stochastic estimator

$$C_{ij,n}(x_0,y_0) = \frac{1}{L^3}\sum_{\vec{x}} \Big((S_j\gamma^5\eta_n^{\langle y_0\rangle})[x], \gamma^5\Gamma_2(S_i\Gamma_1\eta_n^{\langle y_0\rangle})[x]\Big) , \tag{16.41}$$

compare eq. (12.22). Evaluating the above, requires us to solve the linear systems of equation

$$D^{(\ell)}\psi_n^{\Gamma\langle t\rangle(\ell)} = \mathcal{R}^{(\ell)}\Gamma\eta_n^{\langle t\rangle} , \tag{16.42}$$

on levels $\ell = i, j, i+1, j+1$ and time-slice t analogous to eq. (12.23). According to eq. (16.27), the fine-grid stochastic propagator is evaluated as prolonged differences of these solves,

$$\phi^{\Gamma\langle t\rangle}_{n(\ell)} = S_\ell\Gamma\eta_n^{\langle t\rangle} = \mathcal{T}^{(\ell)}\psi_n^{\Gamma\langle t\rangle(\ell)} - \mathcal{T}^{(\ell+1)}\psi_n^{\Gamma\langle t\rangle(\ell+1)} , \tag{16.43}$$

$$\phi^{\Gamma\langle t\rangle}_{n(N_{\text{lvl}}-1)} = S_{N_{\text{lvl}}-1}\Gamma\eta_n^{\langle t\rangle} = \mathcal{T}^{(N_{\text{lvl}}-1)}\psi_n^{\Gamma\langle t\rangle(N_{\text{lvl}}-1)} , \tag{16.44}$$

for $\ell < N_{\text{lvl}} - 1$, where the coarsest level contribution has no subtracted term. The final term can be written in terms of the prolonged differences eq. (16.43) – a scalar product in the fine-grid as

$$C_{ij,n}(x_0,y_0) = \frac{1}{L^3}\sum_{\vec{x}} \Big(\phi^{\gamma^5\langle y_0\rangle}_{n(j)}[x], \gamma^5\Gamma_2\phi^{\Gamma_1\langle y_0\rangle}_{n(i)}[x]\Big) . \tag{16.45}$$

compare eq. (12.24) or directly in terms of the coarse spinor solutions

$$\begin{aligned} C_{ij,n}(x_0,y_0) = \frac{1}{L^3} &\sum_{\ell_1\in\mathcal{A}_i}\sum_{\ell_2\in\mathcal{A}_j} (-1)^{i+j+\ell_1+\ell_2} \\ &\cdot\sum_{\vec{x}} \Big((\mathcal{T}^{(\ell_2)}\psi_n^{\gamma^5\langle y_0\rangle(\ell_2)})[x], \gamma^5\Gamma_2(\mathcal{T}^{(\ell_1)}\psi_n^{\Gamma_1\langle y_0\rangle(\ell_1)})[x]\Big) , \end{aligned} \tag{16.46}$$

where the index set $\mathcal{A}_\ell$ is defined as

$$\mathcal{A}_\ell = \{\ell, \ell+1\} \cap \{0, \dots, N_{\text{lvl}} - 1\} . \tag{16.47}$$

The final estimator using N_{st} stochastic time-diluted fine-grid wall-sources $\eta_n^{\langle t_n \rangle}$ with support on time-slice t_n chosen uniformly from $\{0, \ldots, L_0 - 1\}$ is

$$G_{\mathrm{L}k}(t) = G_{kk}(t) + \sum_{i=k+1}^{N_{\mathrm{lvl}}-1} \Big(G_{ik}(t) + G_{ki}(t) \Big) \;, \tag{16.48}$$

$$G_{ij}(t) = \frac{1}{N_{\mathrm{st}}} \sum_{n=0}^{N_{\mathrm{st}}-1} C_{ij,n}(t + t_n; t_n) \;. \tag{16.49}$$

TL;DR: spin sources only 4 inversion, no matter the spin structure

We will use spin-diagonal random wall-sources [224] allowing an unbiased estimator for each spin matrix element separately. The full spin structure of the correlator given by Γ_1 and Γ_2 can be reconstructed using only 4 spin-diagonal stochastic sources. In total, this amounts $4N_{\mathrm{st}}$ inversions per noise-source, no matter the γ-structure. For $S_i = S_j = S$, this is just the usual stochastic estimator for the two-point correlator. It is straightforward to generalize to other source types.

TL;DR: sources cascade down, but not up

We want to emphasize that a stochastic sample of level ℓ also requires solves on all coarser levels. However this gives us a stochastic sample on all coarser levels too. We may use the same stochastic source as sample on multiple levels, since these are independent traces.

16.4.4 *Exact estimator on coarsest level*

TL;DR: formulas for exact coarsest lvl

When estimating the coarsest grid contribution, its dimension might be small enough for an exact estimation. This is the case if its dimension no larger than say 10^4. The coarsest term can then be brought into a simpler form

$$\begin{aligned} G_{\mathrm{L}(N_{\mathrm{lvl}}-1)}(t) &= \frac{1}{|\Lambda|} \sum_{y_0=0}^{L_0-1} \\ &\cdot \operatorname{tr} \left\{ (D^{(N_{\mathrm{lvl}}-1)})^{-1} {\Gamma_1}^{(N_{\mathrm{lvl}}-1)}[y_0] (D^{(N_{\mathrm{lvl}}-1)})^{-1} {\Gamma_2}^{(N_{\mathrm{lvl}}-1)}[t+y_0] \right\} \;. \end{aligned} \tag{16.50}$$

The Γ-matrices are defined as in eq. (14.5) as coarsened operators with the time extent as open index, i. e. defined with a partial trace over space Σ, color $\mathfrak{C}$ and spin Ξ but not time, see eq. (14.6). Explicitly

$$\left({\Gamma_k}^{(N_{\mathrm{lvl}}-1)} \right)_{ij} = \operatorname{tr}_{\Sigma \times \Xi \times \mathfrak{C}} \left(\Gamma_k \phi_j \phi_i^\dagger \right) \;. \tag{16.51}$$

TL;DR: compare to full-vol averaged corr

The trace in eq. (16.50) is over all coarse degrees of freedom present on the coarse grid. The fine-grid spacetime dependency in the trace is shifted into the coarse Γ-matrices making a full lattice volume average straightforward. Let us compare this form with the full lattice volume averaged correlator $|\Omega| = |\Lambda|$ on the fine grid,

$$\mathcal{G}(t) = \frac{1}{|\Lambda|}\sum_{y\in\Lambda}\sum_{\vec{x}} \operatorname{tr}\left\{S_{[t+y_0,\vec{x};y]}\Gamma_1 S_{[y;t+y_0,\vec{x}]}\Gamma_2\right\} . \tag{16.52}$$

We see that it has exactly the same form, just with some replacements. The fine-grid Γ-matrices are replaced with the coarse ones, the fine-grid inverse Dirac operator $S = D^{-1}$ is replaced with the coarse one $(D^{(N_{\text{lvl}}-1)})^{-1}$ and finally the trace over color, spin and space is replaced by a trace over coarse color, spin and space. Note that the original fine-grid trace over color, spin and space is moved to the Γ-matrices eq. (16.51). However, the term is calculated exactly and averaged over the full spacetime volume, its variance will be minimal, i. e. the gauge variance of that level.

Revisiting low-mode averaging one more time, we notice that the above form is exactly equal to the eigen-eigen term of low-mode averaging worked out in eq. (13.10), if the coarsest lattice is trivial. To see this, we remember the conclusions drawn in section 14.3 and write the trace in index notation. Then eq. (16.50) turns quickly into

TL;DR: LMA does exact coarsest lvl, but without blocking

$$G_{\text{L1}}(t) = \frac{1}{|\Lambda|}\sum_{y_0=0}^{L_0-1}\sum_{k,l,m,n} (\hat{Q}^{-1})_{[kl]} \left(\widehat{\gamma^5\Gamma_1}[y_0]\right)_{[lm]} (\hat{Q}^{-1})_{[mn]} \left(\widehat{\gamma^5\Gamma_2}[t+y_0]\right)_{[nk]} . \tag{16.53}$$

Using the fact that in case of low-mode averaging the coarse Hermitian Dirac operator $\hat{Q}$ is diagonal with eigenvalues along its diagonal, $(\hat{Q}^{-1})_{[kl]} = \delta_{kl}\lambda_k^{-1}$, and reversing the definition of the coarse Γ-matrices as in eq. (14.5);

$$\begin{aligned} G_{\text{L1}}(t) = \frac{1}{|\Lambda|}\sum_{y_0=0}^{L_0-1}\sum_{k,m}\frac{1}{\lambda_k\lambda_m}\sum_{\vec{y}}\Big(\phi_l[y],\gamma^5\Gamma_1\phi_m[y]\Big) \\ \cdot\sum_{\vec{x}}\Big(\phi_n[t+y_0,\vec{x}],\gamma^5\Gamma_2\phi_k[t+y_0,\vec{x}]\Big) , \end{aligned} \tag{16.54}$$

we readily find eq. (13.10) identically, with the bootstrap vectors chosen as low modes $\phi_i = \xi_i$ as is done in LMA.

In fact even in a multigrid scenario with more than 2 levels, one has the freedom to choose the coarsest lattice to be trivial resulting in a final layer of traditional low-mode averaging. Even though such a layer does add only negligible cost, it is only useful for small enough lattices as we will see in the numerical study in chapter 17.

TL;DR: terminating lvl is LMA

16.4.5 V^2 *problem of LMA*

TODO: where to put this?

16.4.6 *Cross-term problem of LMA*

TODO: where to put this?

TODO: Motivated by the previous sections and the results in refs. [230, 277], we want to use the subspace $\hat{\mathcal{V}}$ generated from the lattice $\hat{\mathbb{L}} = (\hat{\Lambda}, \hat{\Xi}, \hat{\mathfrak{C}})$ as averaging- or sample-subspace in our LMA-scenario.

16.5 SUMMARY

TL;DR: summary MG as preconditioner

As a preliminary foundation, we looked at multigrid as a preconditioner which is specially powerful on ill-conditioned systems, because of the low-mode suppression inherent to Krylov solvers. Multigrid scales close to linear in the lattice volume, even though the dimension of the low-mode subspace does so too. Low-mode error components in the iterative solution are treated by solving the coarse Dirac equation to low precision in every outer Krylov iteration.

TL;DR: summary MG as prop decomposition

The quark propagator (the inverse of the Dirac operator) can be decomposed into a contribution along the multigrid subspace and a remainder. This can be repeated recursively, leading to a decomposition of an arbitrary n-point function into N_{lvl} multigrid level contributions in a natural way. Every level can be evaluated individually with a different evaluation strategy. Low-mode averaging is a special case of this decomposition with a trivial coarse lattice.

17

NUMERICAL RESULTS WITH $\mathcal{O}(a)$-IMPROVED WILSON FERMIONS

This section/chapter was proofread 3 times.

> *Information: the negative reciprocal value of probability.*
>
> — Claude Shannon

TL;DR: numerical study intro

In this chapter, we study the application of multigrid low-mode averaging to the connected two-point light-quark meson correlator introduced in chapter 12, that contributes the dominant error to the leading order HVP of the muon $g-2$. We start with defining the measurement methodology (section 17.1), followed by a formal definition of the total and gauge variances (sections 17.2 and 17.3). We plot and investigate relative and absolute variance contributions (sections 17.4 and 17.5) and determine work required to reach the minimal variance (section 17.6) on all estimators. Numerical data shows the ability of the method to reach minimal variance with significantly smaller cost as compared to the naive stochastic estimator, but also compared to low-mode averaging – the state-of-the-art method for this class of observables. We will show numerical evidence of its linear volume scaling, as opposed to the quadratic scaling of low-mode averaging (section 17.7) and compare the cost of the different estimators (section 17.8) before summarizing the chapter (section 17.9).

The numerical study was carried out using $N_f = 2$ flavours of non-perturbative $\mathcal{O}(a)$-improved Wilson-Clover fermions all with the same bare quark masses and lattice parameters. They were generated starting from the CLS ensemble labelled F7 [120] and their specifications are collected in table 17.1. They all share the same pion mass of $m_\pi \approx 270\,\text{MeV}$ as well as the same lattice spacing $a = 0.0658$ but differ in volume. The extents span $L \approx 2.16.3$ fm. In this section, we demonstrate that the number of required low modes N_c for the full translation average and by this for minimal variance is independent of the volume for multigrid LMA, where the gain in variance reduction remains constant.

For every ensemble in table 17.1 we investigated three estimators. The first one is a plain stochastic estimator as described in section 12.2.2 using spin-diagonal time-diluted random wall-sources with $\mathbb{Z}_2$ noise [224] sampled randomly uniform over different time-slices. This estimator is

ensemble	$L_0 \times L^3$	L (fm)	$m_\pi L$	# configs
E7	64×32^3	2.1	2.9	100
F7	96×48^3	3.2	4.3	100
G7	128×64^3	4.2	5.8	100
H7	192×96^3	6.3	8.6	5

TABLE 17.1: Ensembles used in the variance reduction study. All lattices have a pion mass $m_\pi = 270\,\text{MeV}$ and a lattice spacing of $a = 0.0658(10)$ fm with $N_f = 2$ $\mathcal{O}(a)$-improved Wilson fermions with lattice coupling $\beta = 5.3$, hopping parameter $\kappa = 0.13638$ and $c_{\text{SW}} = 1.90952$ [120, 282]. The largest ensemble H7 is only used to compute some variances which can be accurately determined from a few measurements. F7 has been generated by the CLS initiative [120], while the others were generated by Tim Harris.

labelled "stochastic" in the plots. Furthermore, we ran a low-mode averaging estimator using a constant number of $N_c = 50$ low modes on all lattices labelled "LMA". The eigen-eigen piece – corresponding to the L1 term in multigrid parlance – is evaluated exactly according to eq. (13.10), whereas the remaining three terms are collected together, labelled L0 and evaluated stochastically [271] just as the plain stochastic estimator. Finally we employ a multigrid LMA scheme with at least one non-trivial coarse lattice. This class of estimators is labelled "MG LMA". All the levels are usually evaluated stochastically just as the plain stochastic estimator, only exception is the coarsest level which is contracted exactly if its lattice was trivial eq. (16.50). Details about the LMA estimators can be found in table 17.2. For all ensembles and estimators that include low modes, we used a fixed number of $N_c = 50$ low modes of the Hermitian Dirac operator.

17.1 METHODOLOGY

Low modes of the Hermitian Dirac operator were determined to a precision of $\|Q\xi - \lambda\xi\| / \|\xi\| \leq 10^{-12}$ using the library PRIMME [283].

TL;DR: how data was taken

All time measurements were taken analogous to section 7.3. Estimators for the variances will be discussed in the main text as soon as they appear. As estimator for the variance of the variance the method of jackknife resampling [284, 285] was used throughout this document. The dominant measure for cost is in terms of fine-grid inversions. Reason and motivation for this choice are discussed in section 17.8.

Runs were performed on the machines

TL;DR: machine we ran on

- *Piz Daint multicore* at CSCS, Switzerland, 2× Intel® Xeon® E5-2695 v4 @ 2.10GHz (2× 18 cores, 64/128 GB memory)
- *LUMI-C* at CSC, Finland, 2× AMD EPYC™ 7763 @ 2.45GHz (2× 64 cores, 256-1024 GB memory)
- *JUWELS* at Forschungszentrum Jülich, Germany, 2× Intel® Xeon® Platinum 8168 @ 2.7 GHz (2× 24 cores, 96-192 GB DDR4 memory @ 2666 MHz)

17.2 TOTAL VARIANCE

The form of the total absolute variance depends on the number of stochastic sources N_{st} and includes a contribution from the stochastic noise. It is defined as

$$\sigma_G^2(t) = \langle G(t)^2 \rangle_U - \langle G(t) \rangle_U^2 \,, \tag{17.1}$$

where $G(t)$ is the stochastic estimator eq. (16.48) with a single stochastic source $N_{\mathrm{st}} = 1$ and $\langle \cdot \rangle_U$ is the average over gauge field configurations.

17.3 GAUGE VARIANCE

The gauge variance is defined as the variance of the correlator $\mathcal{G}(t)$ averaged over the full spacetime lattice eq. (16.52). It is the minimal variance given by

$$\sigma_{\mathcal{G}}(t) = \langle \mathcal{G}(t)^2 \rangle_U - \langle \mathcal{G}(t) \rangle_U^2 \,. \tag{17.2}$$

The gauge variance is defined such that the variance of the sample mean over N uncorrelated configurations would be $\sigma_{\mathcal{G}}/\sqrt{N}$. We may use the samples from the pure stochastic estimator to estimate the gauge variance

$$\sigma_{\mathcal{G}}^2(t) \approx \frac{1}{N_{\mathrm{st}}(N_{\mathrm{st}}-1)} \sum_{m \neq n = 0}^{N_{\mathrm{st}}-1} \Big[\langle C_m(t_m + t, t_m) C_n(t_n + t, t_n) \rangle_U - \langle C_m(t_m + t, t_m) \rangle_U \langle C_n(t_n + t, t_n) \rangle_U \Big] \tag{17.3}$$

			Block size $b_0/a \times (b_j/a)^3$		
Ensemble	Estimator	N_{lvl}	$\ell = 1$	2	3
E7	LMA	2	64×32^3	-	-
	2-level MG LMA	2	8×8^3	-	-
	3-level MG LMA	3	8×8^3	64×32^3	-
F7	LMA	2	96×48^3	-	-
	2-level MG LMA	2	8×8^3	-	-
	3-level MG LMA	3	8×8^3	96×48^3	-
G7	LMA	2	128×64^3	-	-
	2-level MG LMA	2	8×8^3	-	-
	4-level MG LMA	4	4×4^3	8×8^3	128×64^3
H7	LMA	2	192×96^3	-	-
	2-level MG LMA	2	8×8^3	-	-
	4-level MG LMA	4	4×4^3	8×8^3	192×96^3

TABLE 17.2: Details about all the estimators including low modes from figs. 17.7a to 17.7c. For the different ensembles the table renders the number of multigrid levels, the block sizes used to generate the coarse lattices on each level. When the block size matches the lattice volume, as is the case in the LMA schemes, no blocking is performed on that level. All estimators use the same $N_c = 50$ low modes.

using eq. (12.24). The total variance of the stochastic estimator as defined in eq. (17.1) in the limit of infinite stochastic samples equals the gauge variance $\sigma_{\mathcal{G}}$,

$$\lim_{N_{\mathrm{st}}\to\infty} \sigma_G(t) = \sigma_{\mathcal{G}}(t) \ . \tag{17.4}$$

This limit is verified in fig. 17.1 for the E7 ensemble at time-slice $t \approx 1.3\,\mathrm{fm}$ for the isovector vector correlator up to 4096 stochastic sources.

17.4 RELATIVE VARIANCE

We start with presenting relative variance plots for all ensembles in fig. 17.2. The plots compare the LMA estimator with a 2-level MG LMA estimator.

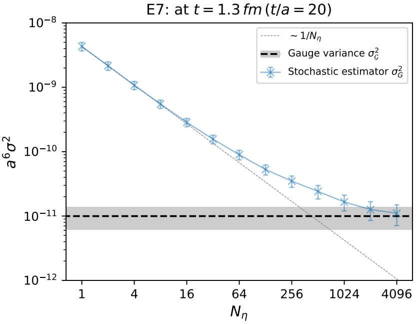


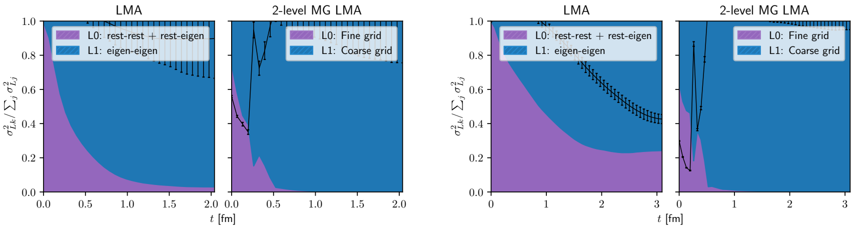
FIGURE 17.1: Verification of the limit eq. (17.4). The x-axis denotes the number of stochastic sources and the y-axis its corresponding variance. The black dashed line with grey error bands represents $\sigma_G(t)$ calculated using eq. (17.3) and the blue solid line is the limit of $\sigma_G(t)$ as defined in eq. (17.1). This plot is taken from publication [P1].

Shown on the y-axis is the relative variance of the corresponding level; the variance of one stochastic source normalized by the sum of variances of all levels, i. e.

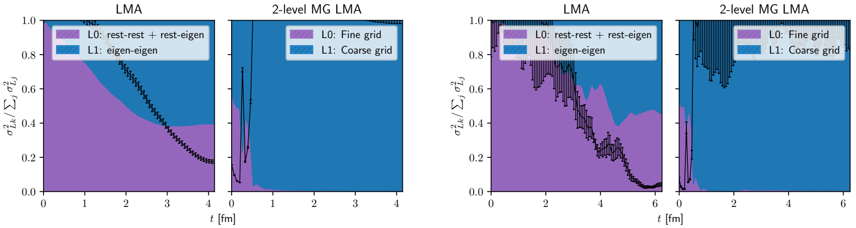
$$\sigma_{rel}^2 = \frac{\sigma^2}{\sum_\ell \sigma_{\mathrm{L}\ell}^2} \in [0,1] \,. \tag{17.5}$$

This gives us an indication on how much variance contribution comes from which level, although without considering covariance among levels. The variance of the sum of all levels – the total variance, when summing – is plotted as black solid line normalized by the sum of variances, highlighting that there is non-negligible covariance among the terms for the LMA data. The only difference between the LMA and the MG LMA data in this plot is that for LMA the L1 coarse lattice is trivial, whereas for MG LMA it is not. The effect of a non-trivial coarse grid – as opposed to a trivial one – reduces the variance contribution from the remaining piece on the fine grid by orders of magnitude on all four ensembles.

Figure 17.2 shows the V^2-problem of low-mode averaging, see section 16.1.1. When looking at the LMA contributions from the smallest lattice E7 to the largest one H7, we observe a substantial increase in contribution from the remainder term (sum of rest-rest, rest-eigen and eigen-rest). On the other hand in the MG LMA scenario, the variance contribution of the L0-term stays negligible, even when holding the number of low modes constant. This is thanks to keeping the block size fixed, resulting in a coarse



(a) E7 (b) F7



(c) G7 (d) H7

FIGURE 17.2: Relative variance eq. (17.5) of one stochastic source of the two levels of LMA (left) and a corresponding 2-level MG LMA (right). The four groups of panels correspond to the four ensembles from table 17.1. In all panels the purple and blue shaded regions denote fine-grid and coarse-grid contributions respectively. The solid black line shows the variance of the sum.

lattice size that scales together with the volume. This shows that the cost of sampling the data scales linear in the volume.

17.5 ABSOLUTE VARIANCE

We now have a qualitative picture of the variance contributions. Next, we will investigate the absolute variances of one single stochastic source in figs. 17.3 to 17.6. The plain stochastic estimator is plotted on all panels as blue solid line along with its gauge variance as black dashed line with grey error band. Where the plain stochastic estimator gives us an upper bound for the variances, the gauge variance is the minimal variance indicating a lower bound.

Crucial for the analysis is the position of the L0-variance, the yellow dashed line in all panels. The closer it is to the gauge variance the better, since our ultimate goal is to push down all variances to the gauge variance.

When the gauge variance is reached, we have optimally used the available gauge field configurations. In this endeavor clearly the fine-grid variance is the most expensive one to push down, as it involves expensive fine-grid inversions.

Pushing down the variance of the L0-term can be achieved in multiple ways: 1) simply increasing the number of stochastic sources on the fine-grid, but that is associated with substantial cost, 2) increasing the number of low modes for the LMA estimator, but that increases the computational and memory cost even more, or 3) employing an MG LMA scenario.

The plots show that we are able to push down the L0 variance to the gauge noise with only one single stochastic source on the fine-grid on all four ensembles irrespective or their volume. Even though introducing stochastic noise, the L0 term is so subdominant in the relevant large time regime, that its total variance (stochastic and gauge) is below the gauge noise of the full correlator and therefore does only contribute negligibly when evaluated with a single stochastic source. We note that for F7, fig. 17.4, which seems to be a small exception, we did not use an ideal setup. A block size of 8×8^3 for the first coarse level L1 is probably already too large, a better choice would be 6×6^3, whereas 4×4^3 is presumably too aggressive. The fact that on larger lattices G7 and H7, we are able to reach gauge variance with a single stochastic source on L0 implies that with proper choice of blocking it is clearly possible on the intermediate F7 too.

Of special interest is the smallest lattice E7 fig. 17.3. Low-mode averaging with 50 low modes is able to realize the long distance translation average. When comparing, the non-trivial lattice of MG LMA still beats the trivial one of LMA in terms of variance contribution. We note that the subspace generated for the trivial lattice (LMA) is contained in the subspace of any non-trivial lattice (MG LMA) by construction. This implies that a non-trivial coarse lattice will always have the better variance contribution than a trivial one, assuming the same low modes were used in their construction.

On the larger lattices, G7 and H7 figs. 17.5 and 17.6, we also show 4-level scenarios. We found that on large lattices is it beneficial to dampen the cost of the possibly-very-large L1 term by deflating it again. The coarsest level on these two lattices is trivial, i. e. corresponds to a layer of LMA. This level does not contribute to the overall variance reduction, because deflation of 50 low modes without blocking is simply not worth doing on these large lattices. This can be seen in the corresponding LMA plots (left panels of figs. 17.5 and 17.6), where the plain stochastic estimator coincides with L0. The fact they coincide clearly shows that the effect of the pure low mode

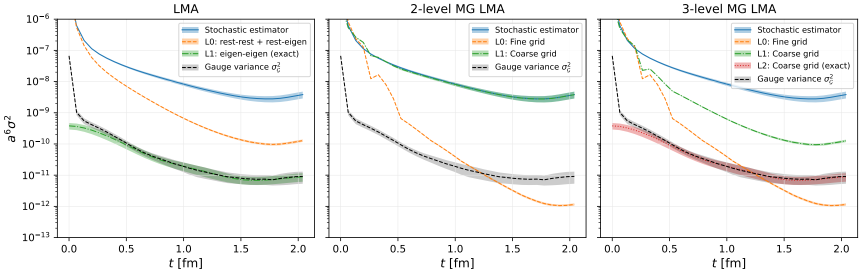


FIGURE 17.3: E7: Absolute total variance eq. (17.1) of a single stochastic source. The left panel shows data for the LMA estimator, center and right show 2-level multigrid and a 3-level multigrid LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison. This plot is an extended version of a plot taken from publication [P1].

deflation quickly becomes negligible as one increases the lattice volume keeping the number of low modes constant.

In order to enjoy the same variance contribution for LMA as on the smallest lattice E7, one has to increase the number of low modes proportional to the volume. Such an increase would correspond to roughly $(\mathrm{E7}, \mathrm{F7}, \mathrm{G7}, \mathrm{H7}) \approx (50, 250, 800, 4000)$ low modes, only to obtain the variance contribution of fig. 17.3 leftmost panel, which is still not enough to push the L0 noise down to gauge noise[1]. The gauge noise level can be reached by either further increasing the number of low modes or by using an MG LMA scenario as in the right panels.

17.6 REACHING GAUGE VARIANCE

In order to determine how many sources are required on every level to reach the gauge variance, we consult the plot series in figs. 17.7a to 17.7c. The required numbers are assembled in table 17.3. The plots show the variance against the number of stochastic sources N_{st}. We expect the variance to decrease roughly as $1/N_{\mathrm{st}}$ (dotted lines in the plots). The data in the plots are extracted on time-slice $t \approx 1.3\,\mathrm{fm}$. This corresponding to the time extent where the signal-to-noise ratio problem starts to be severe [260] and where the long-distance window (LD) is located (see section 1.2.5).

---

[1] The memory footprint of 4000 low modes of the ensemble H7 would be of order 130 TB.

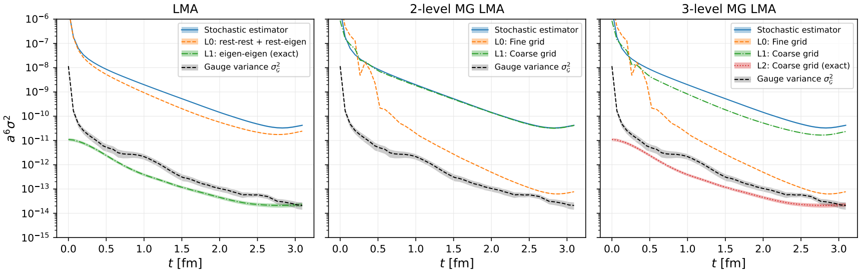


FIGURE 17.4: F7: Absolute total variance eq. (17.1) of a single stochastic source. The left panel shows data for the LMA estimator, center and right show 2-level multigrid and a 3-level multigrid LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison. This plot is an extended version of a plot taken from publication [P1].

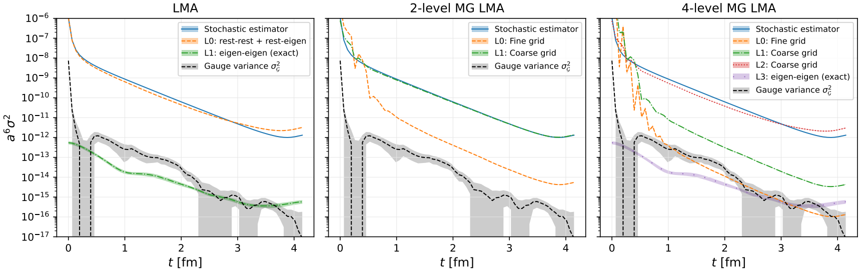


FIGURE 17.5: G7: Absolute total variance eq. (17.1) of a single stochastic source. The left panel shows data for the LMA estimator, center and right show 2-level multigrid and a 4-level multigrid LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison. This plot is an extended version of a plot taken from publication [P1].

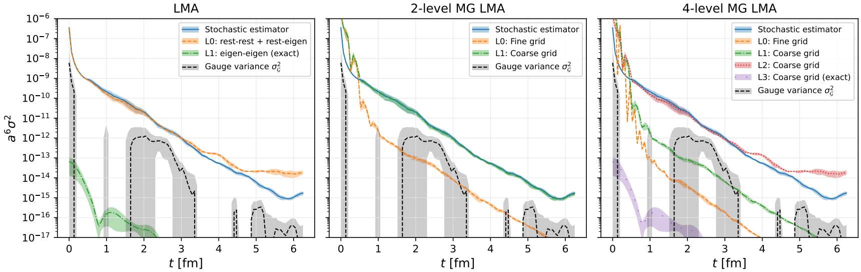
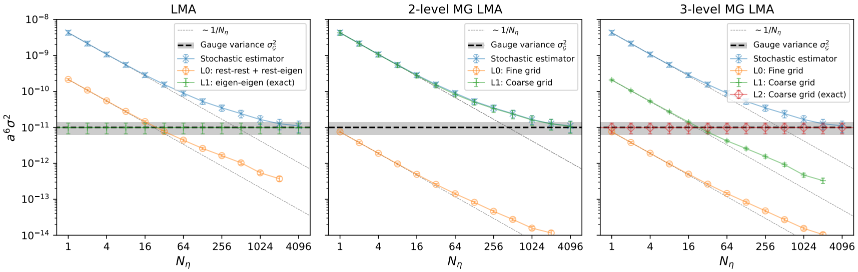


FIGURE 17.6: H7: Absolute total variance eq. (17.1) of a single stochastic source. The left panel shows data for the LMA estimator, center and right show 2-level multigrid and a 4-level multigrid LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison.

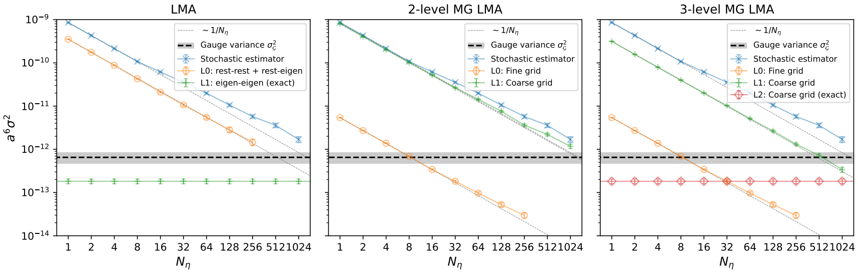
We considered reaching the gauge variance as soon as the line and the gauge variance were consistent with one another. On most ensembles one single source on the fine-grid is enough – as explained earlier on F7 the setup was not ideal. The important point is that the bulk of the variance is captured on the coarse lattices. Having $\mathcal{O}(1000)$ sources on the fine lattice is expensive and infeasible but on coarse lattices absolutely fine. For instance the L2 operator on G7 in the 4-level scheme fig. 17.7c is a factor 512 smaller in its dimension than the L0 operator. Inversions on the L2 lattice are cheaper by about that factor, assuming the same solver algorithm. It is thus simple to sample that level excessively using 1024 sources to push its variance down to the gauge variance. The cost of these 1024 L2-inversion is roughly the same as the cost of about 2 L0-inversions. On the same lattice, we observe that the LMA estimator basically broke down, because the L0-noise is right up with the stochastic estimator, i. e. at the upper bound value giving no variance reduction.

17.7 VOLUME SCALING

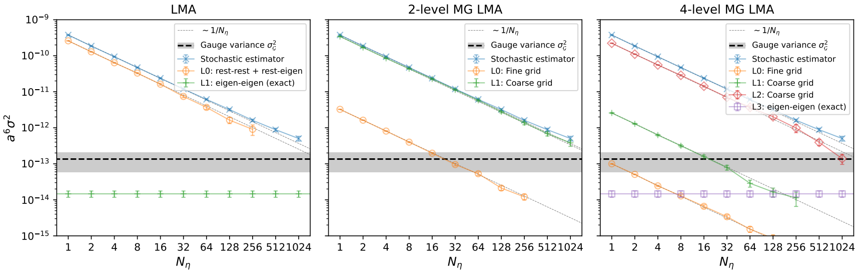
Finally, we discuss volume scaling. Figure 17.8 shows the variance on time-slice $t \approx 1.3\,\text{fm}$ for one single stochastic source on the LMA, 2-level MG LMA and a pure stochastic estimator as a function of the lattice extent. The behavior of the stochastic estimator is straightforward to predict. Its variance decreases with inversely proportional to the *spatial* lattice volume.



(a) E7



(b) F7



(c) G7

FIGURE 17.7: Absolute total variance eq. (17.1) vs. the number of stochastic sources for lattices E7 (top), F7 (center) and G7 (bottom). The left panels show data for the LMA estimator, center the simple 2-level MG LMA estimator and right a 3-level (E7, F7) or 4-level (G7) MG LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison. This plot is an extended version of a plot taken from publication [P1].

This is expected since we use time-diluted random wall-sources which have support on the spatial volume of their respective time-slice, giving an implicit spatial lattice volume averaging, see eq. (12.24). Thus, the larger lattices result in a reduced variance of the stochastic estimator, because of the larger spatial lattice volume and implicit averaging thereof.

The gauge variance on the other hand is defined in eqs. (17.2) and (17.3) as the variance of the full lattice volume averaged correlator eq. (16.52) and is thus expected to decrease inversely proportional to the full *spacetime* lattice volume. The usefulness of the pure stochastic estimator is therefore limited, because it chases the gauge variance which decreases faster by a factor of the temporal lattice extent. Although being a simple method, this is one of the reasons why it is rarely been used standalone on large lattices close to the physical point.

As already touched upon, we observe that the effect of pure low mode deflation (LMA), as the lattice volume increases, becomes negligible unless one also increases the number of low modes proportional to the volume. This is apparent when looking at the L0 line in the left panel. It even increases in variance and eventually saturates the expected upper bound; the pure stochastic estimator. The exactly evaluated L1 term is not of concern, since its variance is bound from above by the gauge variance.

2-level MG LMA on the right panel shows that the L0 term on the fine grid behaves mostly as $1/L^3$. Deviations of that are due to finite volume effects on the smallest lattice E7. The time-slice $t \approx 1.3\,\text{fm}$ is already quite close to its maximal extent of $2.1\,\text{fm}$. However, the remaining three data points behave as $1/L^3$ as expected. Variances of the L1 term and the pure stochastic estimator basically coincide. This is exactly what we aim for with the method. It shows that independent of the lattice volume, we can capture the bulk of the variance in the cheap-to-evaluate L1 term on the coarse lattice. All that remains is to push down the variance of the L1 term by sampling it with the number of sources as stated in table 17.3. The cost of inversion is basically shifted from the expensive fine grid to the cheap coarse grid, where the number of inversions to reach gauge noise roughly stays the same.

17.8 COST

Next, we compare cost of the different estimators. A breakdown to reach gauge noise can be seen in table 17.3. We list the three ensembles E7, F7 and G7 with their estimators. Table 17.3 displays two columns for cost; *measured*

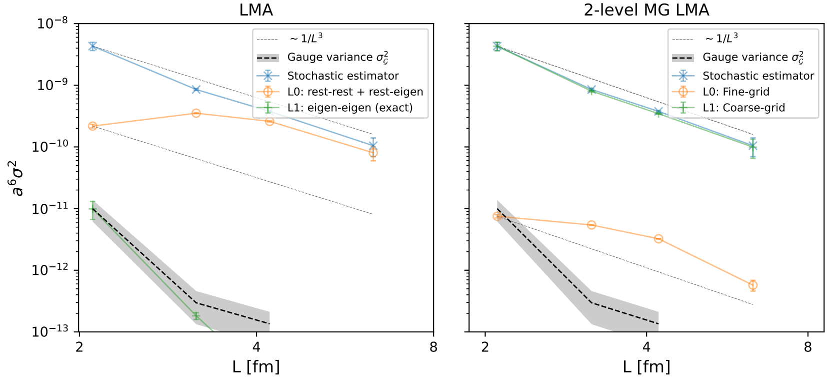


FIGURE 17.8: Absolute total variance eq. (17.1) vs. lattice extent L. The left panel shows data for the LMA estimator, whereas the right panel shows a simple 2-level multigrid LMA with equal block size on all lattices, see table 17.2. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in both panels for comparison. This plot is taken from publication [P1].

and *modeled* cost, both in terms of fine-grid inversions. We decided for this cost metric, because it is machine-independent yet expressive. A conversion to absolute time to solution (TTS) can readily be made by plugging in the time of a fine-grid inversion on some system of interest. Speedups are ratios of these numbers and absolute TTS will cancel in such ratios.

The measured cost is obtained by direct time-measurements of our own implementation. At the time of measuring, no multigrid solver was available; therefore the coarsening and the subspaces had to be implemented from scratch. Clearly an existing performant multigrid implementation is advantageous for a performant implementation of the method.

Our own implementation suffered specially on large intermediate lattices. The reason for this is that we had a very performant solver for the fine-grid inversions; a GCR solver with inexact deflation and additional preconditioning in terms of Schwarz-alternating procedure. On the other hand, the coarse grid solver was a simple even-odd preconditioned GCR. These two solvers are not on eye-level and thus the performance of our coarse grid inversion was substantially degraded by that fact.

The measured timings are assembled in table 17.4, where we note that this table has to be taken with a grain of salt. For instance on the G7 lattice with a large intermediate L1 lattice, we see that the L1 solver needed about

30 times more iteration steps than the fine grid L0 for the same relative residual. Main reason for that is the difference in preconditioning. However, it directly results in about three times slower inversions in terms of TTS. Nevertheless we see clear improvement even with our inefficient coarse

Ens.	Estimator	N_c	N_{st}				cost	
			L0	L1	L2	L3	meas.	mod.
E7	Stochastic		1024				4096	4096
	LMA	50	16	⋆			64	64
	2-lvl MG LMA	50	1	1024			100.4	12.3
	3-lvl MG LMA	50	1	16	⋆		5.5	4.1
F7	Stochastic		2048				8192	8192
	LMA	50	1024	⋆			4096	4096
	2-lvl MG LMA	50	16	2048			462.3	80.7
	3-lvl MG LMA	50	16	1024	⋆		263.2	72.3
G7	Stochastic		4096				16384	16384
	LMA	50	2048	⋆			8192	8192
	2-lvl MG LMA	50	16	2048			557.8	80.7
	4-lvl MG LMA	50	1	16	1024	⋆	466.7	14.4

⋆ The level was evaluated exactly as described in section 16.4.4.

Table 17.3: Cost comparison to reach gauge noise of the different estimators used in this document. The unit of cost is number of fine-grid inversions. For trivial lattices on the coarsest grid the contribution was calculated exactly and its associated cost is zero. The actual cost is estimated by direct measurement table 17.4 and modeled using a performance model discussed in section 17.8.2. We did not add the cost of generating $N_c = 50$ low modes for multiple reasons. If the number is as small as 50, one can safely store them to disk and reuse them for different observables. Their cost is amortized quickly. As stated multiple times in the text, the LMA estimator requires significantly more modes on the larger lattices to be able to compete with the MG LMA estimator, resulting in a cost increase. The goal of this document is to compare the two estimators using *equal* resources in terms of low modes and show the volume dependence of both of them when they are held constant. However, if one insists to add the cost, one can add 100 to 200 to the numbers above. This amounts to a cost of 2 to 4 inversions per low mode.

grid solver. In section 17.8.1 we will show numerically that the coarse Dirac operators are indeed easier to invert in all cases, assuming the same solver algorithm.

The stochastic estimator in table 17.3 is our baseline. It is straightforward to determine its cost in terms of fine-grid inversions; it is read off from figs. 17.7a to 17.7c by inspecting where blue line enters the gauge variance error band and multiplying by a factor 4. We observe an increase in stochastic sources for larger lattices. As discussed in section 17.7, we expect the stochastic estimator to degrade in efficiency, because the gauge variance decreases inversely proportional to the lattice volume, whereas for the stochastic estimator it is the spatial lattice volume.

We have evaluated an LMA estimator with $N_c = 50$ low modes on all lattices. Whereas on the smallest lattice E7 it is very beneficial, decreasing the cost by a factor of 64 with respect to the baseline, it becomes less and less significant on the other lattices giving only a speedup of about 2. Clearly we could make the speedup of this estimator roughly constant by increasing the number of low modes proportional to the lattice volume. Its cost is determined by counting the number of stochastic sources required on the fine grid to reach gauge variance.

Finally, we inspect the MG LMA estimators. On the three lattices we used difference multigrid schemes. The measured and the modeled cost for the coarse grid inversion was calculated using the cost formula

$$\text{cost} = 4 \sum_{\ell=0}^{N_{\text{lvl}}-1} \frac{N_{\text{st}}(\text{L}\ell)}{\Phi(\text{L}\ell)} , \tag{17.6}$$

where N_{lvl} is the number of multigrid levels. The number of stochastic sources used on level ℓ, denoted as $N_{\text{st}}(\text{L}\ell)$ is read off from the corresponding columns in table 17.3. For the measured cost we read of the ratio $\Phi(\text{L}\ell)$ from table 17.4 which is the dimensionless ratio of the average measured TTS of a fine grid divided by a coarse grid inversion. The ratio of the modeled cost will be discussed separately in section 17.8.2. The factor 4 comes from the fact that we use spin-diagonal random wall-sources on all levels which amounts one inversion per spin degree of freedom.

The smallest lattice E7 shows two estimators; a two-level scheme using the $N_c = 50$ low modes to create a subspace with block size 8^4 (see table 17.2) and a three-level scheme that is the combination of the LMA estimator with the two-level scheme. This lattice is of special interest, because we see that the two-level MG LMA estimator is worse than the traditional LMA estimator using our implementation. Even though the remaining 16 fine-grid sources decreased to a single one, we had to introduce 1024 sources on the coarse grid as opposed to none. Using our inefficient coarse inverter, this trade-off resulted in a net cost increase. However when instantiating a third level these 1024 coarse grid sources decreased to only 16. We emphasize that this is the same factor 64 as the LMA estimator had over the stochastic one, which is not a coincidence, when inspecting fig. 17.7a. Deflation of pure low modes had the effect of pushing down the L0 variance of the LMA estimator (fig. 17.7a right). In the exact same way we pushed down the L1 variance (fig. 17.7a center) by deflating it with the pure low modes (fig. 17.7a left). However, the intermediate non-trivial lattice between the fine grid and the trivial lattice has the effect of both pushing down the L0 noise to the gauge variance and decreasing the required sources on L1 to 16.

We go on discussing the intermediate lattice F7, where the MG LMA scheme was not chosen ideally. The LMA estimator is already not beneficial anymore, making the three-level scheme only marginally better than the two-level one. With ideally chosen coarse grids, we would expect its cost in terms of fine-grid inversions to be in between the costs of E7 and G7. This could have been reached with another intermediate coarse lattice of block size of either 6^4 or 4^4 with respect to L0, similar as to the four-level scheme of G7.

Finally the largest lattice fully analyzed is G7. We have employed the same two-level scheme as for the other lattices but additionally ran a four-level scheme with another non-trivial coarse lattice and traditional LMA as final level. The final level on this lattice did not contribute and could have been removed safely without cost impact. However the final lattices – if trivial – were not considered in the cost estimation, because these are just $N_\mathrm{s} N_\mathrm{c} = 100$ dimensional all to all contractions, see eq. (16.50).

The cost of MG LMA compared to LMA decreased by introducing non-trivial coarse lattices, over which we sample. Even with our modest implementation we observe huge speedups over the plain stochastic but also over the LMA estimator with a constant number of low modes.

17.8.1 *Condition of coarse Dirac operators*

We will continue investigating condition numbers of coarse Dirac operators. The success of the method crucially depends on the fact that the coarse Dirac operators are better conditioned than the fine grid one. This is because the convergence behavior of Krylov subspace solvers depends on the eigenvalue distribution and the condition number of the operator to solve. For instance, for conjugate gradient (CG) the number of iteration steps scales as [286]

$$N_{\text{it}} \sim \mathcal{O}\left(\sqrt{\kappa(D)}\right) \log\left(\frac{1}{\epsilon}\right) , \tag{17.7}$$

where ϵ is the precision (relative residual) of the solution ψ after N_{it} iterations $\epsilon = \|D\psi - \eta\| / \|\eta\|$ and $\kappa(D)$ is the condition number of the operator D,

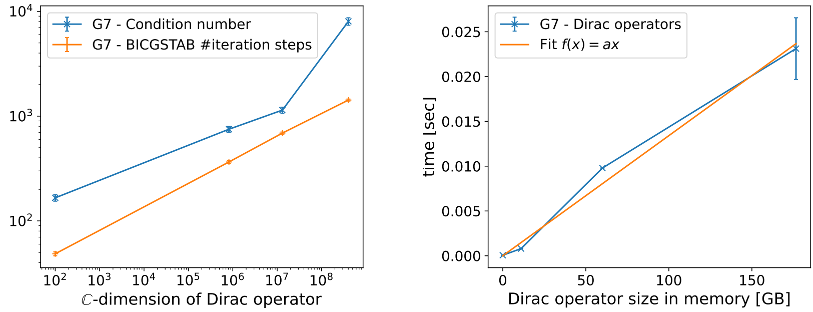
$$\kappa(D) = \frac{\sigma_{max}(D)}{\sigma_{min}(D)} , \tag{17.8}$$

where $\sigma_{max}(D)$ and $\sigma_{min}(D)$ are the largest and smallest singular values of D. We set $\kappa(D) = \infty$ if $\sigma_{min}(D) = 0$. Even though for other Krylov subspace solvers there is no closed formula for the iteration count as for CG, the rule of thumb "smaller condition number implies smaller iteration count" is generally applicable.

To empirically show this, we used the G7 ensemble, where we studied a 4-level scheme, thus 4 Dirac operators, all with different dimensions. Their

Ens.	Type	Lattice	Blk.	TTS [core-h]	Ratio Φ	# iteration
E7	fine	64×32^3		0.378(2)	1.0	35.7(2)
	coarse	8×4^3	8^4	0.009	42.6(2)	140.5(3)
F7	fine	96×48^3		3.03(1)	1.0	43.77(15)
	coarse	12×6^3	8^4	0.15	20.6(1)	337.6(1.3)
G7	fine	128×64^3		12.6(5)	1.0	46.53(23)
	coarse	32×16^3	4^4	42(2)	0.29(2)	1417(22)
	coarse	16×8^3	8^4	0.76(5)	16.6(1.2)	502(6)

TABLE 17.4: Average cost in terms of core-hours for an inversion on the fine and coarse grids. The block size is indicated in the fourth column, empty corresponds to the fine lattice. The time to solution (TTS) is reported in core-hours and Φ is the ratio of the TTS of a fine grid solve divided by a coarse grid solve. Φ is used as input for the measured cost calculation using eq. (17.6) in table 17.3. The last column reports the average number of iterations required for the solver to converge. It emphasizes the unequal solvers used on the different grids. We note that one should take care when comparing timings from one lattice to another, because they were taken on different machines with different node setups.



(a) Coarse operator condition

(b) Memory bandwidth model

FIGURE 17.9: (a) Estimated condition number and average number of iteration steps for a Biconjugate gradient stabilized solve (BiCGSTAB) vs. the dimension of the operator on G7. This plot is taken from publication [P1]. (b) Time for one application of a fine- or coarse-grid Dirac operator vs. their memory footprints on G7.

approximate condition number as well as their iteration counts to solve a random right-hand side to a certain precision was plotted in fig. 17.9a as a function of the dimension over the complex numbers. The leftmost operator is the diagonal Dirac operator emerging in an LMA scenario, see eq. (14.10), and the rightmost one is the operator on the fine-grid with dimension $12V$. The two intermediate data points correspond to the L1 and L2 operators for the 4-level MG LMA estimator of G7 in table 17.2.

Our assumptions from above are clearly met by fig. 17.9a; the smaller the operator dimension the smaller the condition number and by this the smaller the iteration count. It is thus save to assume – with the same solver algorithm on all levels – that coarser levels are cheaper to perform solves on than finer levels.

17.8.2 *Performance model*

Based on the insights of the previous section, we will quantify the improvement factor of a coarse grid solve compared to a fine grid one. This will give us an estimate for the ratio Φ to use in the cost formula eq. (17.6) to obtain a cost estimate as if we have used the same solver algorithm on all levels, i. e. as if we had access to a performant true multigrid solver.

The high coarse grid iteration count of our disadvantageous implementation is refuted considering the numerical results from the previous section. A second problem emerges as one increases the dimension of the coarse operators, because they – even though having the same sparsity pattern as the fine one – are less sparse in general, since they usually have more color degrees of freedom.

Sparse matrix vector multiplication is the dominant kernel in a Krylov solve of a lattice Dirac operator. It is a memory bandwidth bound operation. The time of a Dirac operator application is proportional to its size in memory, see fig. 17.9b. The rightmost data point in the plot is not the fine grid operator as one might assume but the L1 operator! Because of the deceased sparsity, the L1 operator from the 4-level scheme of G7 is about 3 times larger in memory than the fine grid Dirac operator it is constructed from, even though its dimension is about a factor 30 smaller. Besides the different solver algorithms on fine and coarse grids, this fact is responsible for the degradation of performance of our implementation. On first sight this poses a clear problem. How can this be beneficial, if the operator memory footprint increases in some cases? In the following we will argue that this problem is in fact neither an issue and nor a performance limiter.

We will start with formalizing the cost of a single inversion involving operator K. Motivated by fig. 17.9b, our choice for the unit of cost is memory traffic,

$$\mathrm{cost}(K) = \mathrm{iter}(K)\left[\mathrm{mem}(K) + (N_K + 1)\cdot \mathrm{mem}(\psi)\right] \tag{17.9}$$

where $\mathrm{iter}(K)$ denotes the average number of iterations required to solve a linear system of equation with the operator K for some right-hand side and some fixed relative residual. The memory traffic in bytes produced by applying an operator K or reading a field ψ is denoted by $\mathrm{mem}(K)$ and $\mathrm{mem}(\psi)$, respectively. The number N_K depends on the sparsity pattern of the operator, its implementation and the machine[2]. We assume that every solver iteration consists of one application of K which involves one read of K, N_K reads and one write of the right-hand side ψ. We have neglected further operations on the vectors, like scalar products, norms or axpys[3], because we assume the operator memory traffic to dominate.

---

[2] If K is a nearest neighbor stencil, the right-hand side may be read multiple times, depending on the sparsity pattern of the stencil, implementation thereof, cache hierarchy, size, latency and many more. If $K = D$ the Wilson-Dirac operator in 4D spacetime, $N_K = N_D = 1 - 9$, because for every lattice point, we need to read all direct neighbors in 8 directions including the point itself, compare eq. (1.69). In the worst case this amounts 9 reads of the same point, but if cache reusage patterns are favorable this number will be closer to 1.

With the performance model defined in eq. (17.9), we are clearly confronted with the issue mentioned before. Therefore, we continue with the definition of the cost of N_{rhs} simultaneous solves as

$$\text{cost}(K, N_{\text{rhs}}) = \text{iter}(K)\left[\text{mem}(K) + (N_K + 1)N_{\text{rhs}} \cdot \text{mem}(\psi)\right] . \tag{17.10}$$

This equation is motivated by the fact that when applying the operator K to N_{rhs} right-hand sides simultaneously, the operator has to be read only once instead of N_{rhs} times, if implemented in a multiple right-hand sides fashion [160]. Meaning the operator can operate on many right-hand sides at the same time. This turns matrix-vector multiplications into matrix-matrix multiplications improving temporal cache locality and increasing arithmetic intensity.

Finally we can define the speedup of a coarse grid solve of $\hat{K}$ with respect to a fine grid solve of K as the ratio of their cost

$$\text{Sp}(N_{\text{rhs}}) = \frac{\text{iter}(K)\left[\text{mem}(K) + (N_K + 1)N_{\text{rhs}} \cdot \text{mem}(\psi)\right]}{\text{iter}(\hat{K})\left[\text{mem}(\hat{K}) + (N_{\hat{K}} + 1)N_{\text{rhs}} \cdot \text{mem}(\hat{\psi})\right]} . \tag{17.11}$$

With one right-hand side and assuming the operator to dominate, $\text{mem}(K) \gg \text{mem}(\psi)$, we find

$$\text{Sp}(1) \approx \frac{\text{iter}(K)}{\text{iter}(\hat{K})} \frac{\text{mem}(K)}{\text{mem}(\hat{K})} . \tag{17.12}$$

This is exactly what we observe with our modest implementation: on some coarse grids both the iteration count and the memory traffic blow up making the inversion slow compared to the fine grid.

On the other hand using a multiple right-hand side (mRHS) solver in the asymptotic limit of many right-hand sides, we find instead

$$\lim_{N_{\text{rhs}} \to \infty} \text{Sp}(N_{\text{rhs}}) = \frac{\text{iter}(K)}{\text{iter}(\hat{K})} \frac{(N_K + 1)}{(N_{\hat{K}} + 1)} \frac{\text{mem}(\psi)}{\text{mem}(\hat{\psi})} . \tag{17.13}$$

We emphasize that the dependence of the memory traffic of the operator $\text{mem}(K)$ dropped, c.f. eq. (17.12). Conservatively we will assume the iteration count to not change from fine to coarse grid, i.e. $\text{iter}(K) = \text{iter}(\hat{K})$, even though section 17.8.1 suggested $\text{iter}(K) > \text{iter}(\hat{K})$, but quantifying that factor is not trivial.

---

[3] This stands for $a\vec{x} + \vec{y}$, scalar times vector plus vector, "a x plus y". It resembles the BLAS level-1 routine call family of the same name.

The coarse operator $\hat{K}$ has the same sparsity pattern as the fine one. This suggests $N_K \approx N_{\hat{K}}$ although spin and color degrees of freedom differ making $\hat{K}$ usually less sparse than K. This might result in a less efficient implementation of the coarse operator. For simplicity, we will assume equally-efficient implementations, $N_K = N_{\hat{K}}$.

The memory traffic produced by the coarse and fine-grid fields are easily worked out. They are proportional to the dimension,

$$\text{mem}(\psi) = 12V \cdot \text{sizeof}(Float) \,, \tag{17.14}$$

$$\text{mem}(\hat{\psi}) = \hat{N}_s \hat{N}_c \hat{V} \cdot \text{sizeof}(Float) \,. \tag{17.15}$$

The irrelevant size of the primitive datatype in bytes, sizeof(Float), cancels in the ratio and since the produced memory traffic of an inversion cost(K) is proportional to the TTS of the same, we can interpret the speedup as ratio

$$\Phi(\text{L}\ell) = \lim_{N_{\text{rhs}} \to \infty} \text{Sp}(N_{\text{rhs}}) \approx \frac{\text{mem}(\psi)}{\text{mem}(\hat{\psi})} = \frac{12V}{\hat{N}_s \hat{N}_c \hat{V}} \,. \tag{17.16}$$

Thus the cost ratio in our performance model is just the ratio of the dimensions of the respective Hilbert spaces.

For the determination of the modeled cost in table 17.3 we used eq. (17.6) with eq. (17.16) as ratio Φ. The values are assembled in table 17.5. A mRHS solver is expected to be computationally even more beneficial on the coarse grids than on the fine grid, because the coarse grid strong scales badly as we have seen in chapter 7 and stacking multiple solves mitigates that.

17.9 SUMMARY

We have shown the consistency and performance of the new method, we refer to as multigrid low-mode averaging as compared to the plain stochastic and the traditional low-mode averaging estimators. The most important finding in this chapter is that the achieved variance reduction is flat in the lattice volume as opposed to traditional sampling schemes. Due to the good overlap with the true low-mode subspace, we observe speedups of more than an order of magnitude compared to the plain stochastic and LMA estimators.

The majority of the variance is contained in the cheap parts, whereas the expensive remainder shows suppressed variance contribution. This hierarchical sampling scheme leads to a flat scaling behavior in the required number of low modes, leading to a linear volume-scaling in computational

Ens.	Type	Level	Lattice	N_s	N_c	Ratio Φ
E7	fine	L0	64×32^3	4	3	1.0
	coarse	L1	8×4^3	2	50	491.52
F7	fine	L0	96×48^3	4	3	1.0
	coarse	L1	12×6^3	2	50	491.52
G7	fine	L0	128×64^3	4	3	1.0
	coarse	L1	32×16^3	2	50	30.72
	coarse	L2	16×8^3	2	50	491.52

TABLE 17.5: Ratio Φ for the modeled cost of the different MG LMA estimators used in this document.

cost, effectively eliminating the V^2-problem inherent to this class of problems. Since coarse operators are better conditioned than fine ones, sampling coarse subspaces is always less expensive than sampling fine ones, keeping the cost factor under control.

18

SPECTRAL IMPLICATIONS OF CHIRALITY

This section/chapter was proofread 2 times.

The main result we want to establish numerically in this chapter is

$$\kappa(\hat{D}) \leq \kappa(D) \;. \tag{18.1}$$

The condition number of the coarse Dirac operator should be bounded from above by the fine-grid condition number. We will not mathematically proof this inequality, but provide strong numerical evidence for its truth under certain conditions. This section continues specifying spin and chirality, followed by the concept of coarse chirality (section 18.1) and continues with implications of its weak preservation (section 18.2). Numerical ranges, spectral hulls and their symmetries of coarse and fine operators are investigated (sections 18.3 and 18.4), followed by a study of the chiral implications to low-lying spectra of Hermitian Dirac operators (section 18.5) as well as their variance contribution (section 18.6). The section ends with a description of the numerical methods (section 18.7) and a summary (section 18.8).

To achieve eq. (18.1), we need to discuss the spin degrees of freedom carried over to the coarse grid as they have significant algorithmic and theoretical implications. Recapping section 15.3, a Dirac spinor has 4 spin degrees of freedom; a tensor product of two chiral ones and two non-chiral ones. When generating the coarse grid, we may define projectors to individual spins or chiralities, projecting the basis fields, which finally leads to additional spin degree of freedom on the coarse grid inherited from the fine grid. Formally, assuming we have two spin projectors P_0 and P_1 that only act on the spins, we redefine the restrictor and prolongator analogue to eqs. (15.9) and (15.10) one more time by considering the spin degrees of freedom. The coarse spinors now posses coarse spacetime $\hat{x} \in \hat{\Lambda}$, coarse color $\hat{a} \in \hat{\mathfrak{C}}$ and additionally coarse spin $\hat{\alpha} \in \hat{\Xi} = \{0, 1\}$ indices. The spin indices are inherited from the spin projectors P_0 and P_1 – in case of two projectors $\hat{N}_s = 2$. The restrictor then acts on a fine-grid spinor ψ as

$$(R\psi)[\hat{x}]_{\substack{[\hat{a}] \\ {[\hat{\alpha}]}}} = (\phi_{\hat{x},\hat{a},\hat{\alpha}}, \psi) \;, \quad \phi_{\hat{x},\hat{a},\hat{\alpha}} \in \Pi(\mathcal{B}) \;, \tag{18.2}$$

whereas the prolongator acts on a coarse-grid spinor $\hat{\psi}$ as

$$T\hat{\psi} = \sum_{\hat{x}\in\hat{\Lambda}} \sum_{\hat{a}\in\hat{\mathfrak{C}}} \sum_{\hat{\alpha}\in\hat{\Xi}} \hat{\psi}[\hat{x}]_{\substack{[\hat{a}]\\[\hat{\alpha}]}} \phi_{\hat{x},\hat{a},\hat{\alpha}} \ . \tag{18.3}$$

The field index i from eqs. (14.1) and (14.2) just became a multi-index $i = (\hat{x}, \hat{a}, \hat{\alpha})$. However, this is the final form, all indices on the fine grid have a coarse analogue now.

We want to formalize this and come up with a sufficient condition for coarse Dirac operators have condition numbers and lead to a fair variance reduction. To simplify things, we choose the Weyl basis for the γ-matrices (see eq. (E.2)), because it makes chirality manifest. This choice makes γ^5 diagonal as

$$\gamma^5 = \gamma_0\gamma_1\gamma_2\gamma_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} . \tag{18.4}$$

It acts on positive chirality as identity and on negative chirality as minus identity. By this the 4 spin degrees of freedom of a Dirac spinor are divided into chiral and non-chiral ones. **TODO**: Tim: doesnt like chiral and non-chiral ... This makes the chiral degree of freedom manifest in γ^5 and we can write it as tensor product

$$\gamma^5 = \chi^5 \otimes \mathbb{1}_S \, , \qquad \chi^5 = \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} , \qquad \mathbb{1}_S = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} , \tag{18.5}$$

where χ^5 only acts on the chiral degrees of freedom and the identity $\mathbb{1}_S$ on the remaining spins, which we will refer to as *non-chiral* spin indices. Such a decomposition is also possible in the Dirac- or Majorana basis. The chiral operator χ^5 will then be a different Pauli matrix.

On the fine grid, γ^5 only acts non-trivially on the spins. Regarding notation, we usually suppress the identities in the tensor product when acting on spinors and morally identify

$$\gamma^5 \equiv \mathbb{1}_\Lambda \otimes \gamma^5 \otimes \mathbb{1}_{\mathfrak{C}} \, , \tag{18.6}$$

where $\mathbb{1}_\Lambda$ and $\mathbb{1}_{\mathfrak{C}}$ are identities acting on the spacetime and color degrees of freedom, respectively. In this chapter, we may leave the identities explicit for clarity and define the (strictly separable[1]) chiral operator acting on a spinor as

$$\Gamma^5 = \mathbb{1}_\Lambda \otimes \chi^5 \otimes \mathbb{1}_S \otimes \mathbb{1}_{\mathfrak{C}} \,, \tag{18.8}$$

using the decomposition into lattice, chiral, non-chiral and color spaces.

We will assume throughout this chapter that we have determined N_c lowest modes of $\Gamma^5 K$

$$\Gamma^5 K \xi_i = \lambda_i \xi_i \,, \tag{18.9}$$

where K is a Γ^5-Hermitian operator; $K^\dagger = \Gamma^5 K \Gamma^5$. Furthermore we have constructed coarse multigrid subspaces as described in chapter 15 with restrictor R, prolongator $T = R^\dagger$ and Hermitian projector $P = TR$ as in eqs. (14.1) to (14.3).

18.1 COARSE CHIRALITY

Since Γ^5 is an operator acting on Dirac spinors

$$\Gamma^5 \colon \mathcal{V} \longrightarrow \mathcal{V} \,, \tag{18.10}$$

we can coarsen it using the usual formula, eq. (14.4),

$$\widehat{\Gamma^5} = R\Gamma^5 T \colon \hat{\mathcal{V}} \longrightarrow \hat{\mathcal{V}} \,, \tag{18.11}$$

where R and T are restrictor and prolongator defined in eqs. (14.1) and (14.2). Similarly as the fine chirality operator, the coarse one is diagonal in spacetime. The remaining indices though are non-trivial

$$\widehat{\Gamma^5}_{[\hat{x};\hat{y}]} = \delta_{\hat{x}\hat{y}} \cdot \rho^5(\hat{x}) \,, \tag{18.12}$$

where $\rho^5(\hat{x})$ acts only on coarse spins and coarse colors, potentially different for every coarse lattice point $\hat{x}$. As opposed to the fine grid, we lose strict separability of ρ^5 into spin (chiral and non-chiral indices) and color, as described in eq. (18.5), in general.

The spin degrees of freedom can be coarsened in many ways. The are certain coarsening choices that reinstantiate strict separability and make $\rho^5(\hat{x})$ independent of $\hat{x}$. Table 18.1 summarizes many choices, some of

---

[1] Let $\mathcal{H}_A$ and $\mathcal{H}_B$ be finite-dimensional Hilbert spaces. A linear operator $S \colon \mathcal{H} \to \mathcal{H}$ acting on their tensor product $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$ is called *strictly separable* if it can be written as,

$$S = A \otimes B \,, \tag{18.7}$$

for some linear operators $A \colon \mathcal{H}_A \to \mathcal{H}_A$ and $B \colon \mathcal{H}_B \to \mathcal{H}_B$.

which are listed only to investigate coarse chirality implications and to help identify a sufficient condition for eq. (18.1). Some preserve the trace on the fine-grid $\mathrm{tr}\left(\Gamma^5\right) = 0$, some preserve chirality $[P, \Gamma^5] = 0$ and some preserve fine-grid low modes $P\xi_i = \xi_i$, eq. (18.9).

Even though case (2wc) produces a Hermitian coarse operator $\hat{K}$, it falls out immediately, since the coarse subspace must contain the low modes (last column). This is a restriction of LMA and multigrid LMA that by design require to capture the low mode subspace. Projecting parts away would be counter productive. However, the remaining choices are all valid. Cases (2sc) and (2sv) are physically equivalent, they only differ in basis. When appearing in traces, they evaluate to the same result. Case (1) preserves no spin degrees of freedom, case (2nc) preserves the non-chiral ones and cases (2sc), (2sv), (2wc) and (4) preserve chiral degrees of freedom, (4) even both.

Preservation of the trace $\mathrm{tr}\left(\widehat{\Gamma^5}\right) = 0$ implies that both chiral degrees of freedom are treated symmetrically, i. e. both are explicit on coarse subspaces, none is preferred over the other. We therefore define the term *weak chirality preservation* of coarse subspaces as subspaces whose projector obeys

$$[P, \Gamma^5] = 0 \ . \tag{18.13}$$

If additionally $\mathrm{tr}\left(\widehat{\Gamma^5}\right) = 0$, we call the subspace *strong chirality preserving*. Only cases (2sc), (2sv) and (4) strongly preserve chirality in that way. We will find that only these cases lead to well condition coarse operators $\hat{K}$.

We want to further investigate the properties of the coarse chirality operator $\widehat{\Gamma^5}$. For this we look at its eigenvalues and eigenvectors,

$$\widehat{\Gamma^5}\hat{\psi} = \lambda\hat{\psi} \ . \tag{18.14}$$

Plugging in the definitions of the coarse objects, we obtain

$$R\Gamma^5\psi = \lambda R\psi \ , \tag{18.15}$$

where we used $P = TR$ and the fact that for every coarse spinor $\hat{\psi}$ we have $P\psi = \psi$ for its prolonged spinor $\psi = T\hat{\psi}$. This is just an equation restricted to the subspace and we can prolong it to the fine grid by applying T to both sides,

$$P\Gamma^5\psi = \lambda\psi \ . \tag{18.16}$$

If chirality is weakly preserved, $[P, \Gamma^5] = 0$, we find the eigenvalue equation for the fine-grid chirality operator $\Gamma^5\psi = \lambda\psi$. Therefore, if $[P, \Gamma^5] = 0$ the

ID	Spins	$\rho^5(\hat{x})$	$\mathrm{tr}\Big(\widehat{\Gamma^5}\Big)$	$[P,\Gamma^5]$	$P\xi_i = \xi_i$
1	-	Hermitian	$\in \mathbb{R}$	$\neq 0$	☑
2sc	$P_\pm = \frac{1}{2}\left(\mathbb{1} \pm \Gamma^5\right)$	$\chi^5 \otimes \mathbb{1}_{\hat{\mathfrak{C}}}$	0	0	☑
2sv	$P_i = \mathbb{1}, P_g = \Gamma^5$	$B_{\hat{x}}\left(\chi^5 \otimes \mathbb{1}_{\hat{\mathfrak{C}}}\right) B_{\hat{x}}^{-1}$	0	0	☑
2nc	$P_{0,2}, P_{1,3}$	Hermitian	$\in \mathbb{R}$	$\neq 0$	☑
2wc	P_0, P_1 or P_2, P_3	$\pm\mathbb{1}_{\hat{S}} \otimes \mathbb{1}_{\hat{\mathfrak{C}}}$	$\pm 2N_c$	0	☐
4	P_0, P_1, P_2, P_3	$\chi^5 \otimes \mathbb{1}_{\hat{S}} \otimes \mathbb{1}_{\hat{\mathfrak{C}}}$	0	0	☑

TABLE 18.1: Projectors and the form of coarse chiral and non-chiral operators. Coarse spin indices are inherited from the spin projectors (second column). The table uses the definition of the spin projectors $(P_\sigma)_{[\alpha\beta]} = \delta_{\alpha\beta}\delta_{\alpha\sigma}$ and $P_{\sigma,\rho} = \frac{1}{2}(P_\sigma + P_\rho)$ for $\sigma, \rho = 0, 1, 2, 3$. The matrix B is some irrelevant block-diagonal change-of-basis matrix. The IDs stand for Nxx, where N denotes the number of coarse spin degrees of freedom $\hat{N}_s$ and the letters xx stand for sc: strongly chiral, nc: non-chiral, wc: weakly chiral and sv: singular vectors.

spectrum of the coarse chirality operator is a subset of the fine one; explicitly either

$$\sigma(\widehat{\Gamma^5}) = \{+1\}\,, \qquad \sigma(\widehat{\Gamma^5}) = \{-1\}\,, \qquad \text{or} \qquad \sigma(\widehat{\Gamma^5}) = \{+1, -1\}\,. \tag{18.17}$$

Clearly for the first two spectra the coarse chirality operator will be proportional to the identity and will not be traceless corresponding to both cases (2wc) in table 18.1. A trace-preserving operator appears if both eigenspaces of the fine chirality operator are coarsened symmetrically.

Lemma 18.1.1 (Weak chirality preservation). *Assuming that P and Γ^5 are Hermitian, the following statements are all equivalent:*

$$[P, \Gamma^5] = 0\,, \qquad \textit{(weak chirality preservation)} \tag{18.18}$$

$$\sigma(\widehat{\Gamma^5}) \subseteq \sigma(\Gamma^5)\,, \tag{18.19}$$

$$(\widehat{\Gamma^5})^2 = \mathbb{1}\,, \qquad \textit{(involution)} \tag{18.20}$$

$$\widehat{\Gamma^5}(\widehat{\Gamma^5})^\dagger = (\widehat{\Gamma^5})^\dagger\widehat{\Gamma^5} = \mathbb{1}\,, \qquad \textit{(unitary)} \tag{18.21}$$

$$\mathrm{tr}\Big([P, \Gamma^5]^2\Big) = 0\,. \tag{18.22}$$

Proof. Assuming the first equality to be true, we have already shown that the spectrum has to be one of eq. (18.17). Thus, eq. (18.18) implies eq. (18.19).

Looking at an operator with one of the three possible spectra, all of them square to an operator with spectrum $\{1\}$, which is the identity operator. Thus, eq. (18.19) implies eq. (18.20).

Coarse operators inherit Hermiticity from fine ones. Thus, a Hermitian involution is unitary; eq. (18.20) implies eq. (18.21).

Assuming we have a unitary, we calculate explicitely,

$$\begin{aligned}
\operatorname{tr}\left([P,\Gamma^5]^2\right) &= 2\left[\operatorname{tr}\left(P\Gamma^5 P\Gamma^5\right) - \operatorname{tr}(P)\right] && (18.23)\\
&= 2\left[\operatorname{tr}\left(R\Gamma^5 T R\Gamma^5 T\right) - \operatorname{rank}(P)\right] && (18.24)\\
&= 2\left[\operatorname{tr}\left(\widehat{\Gamma^5}\widehat{\Gamma^5}\right) - \dim(\hat{\mathcal{V}})\right] && (18.25)\\
&= 2\left[\operatorname{tr}\left((\widehat{\Gamma^5})^\dagger\widehat{\Gamma^5}\right) - \dim(\hat{\mathcal{V}})\right] && (18.26)\\
&= 2\left[\operatorname{tr}(\mathbb{1}) - \dim(\hat{\mathcal{V}})\right] && (18.27)\\
&= 0\,, && (18.28)
\end{aligned}$$

where in the last step we used the unitary property of $\widehat{\Gamma^5}$. Thus, eq. (18.21) implies eq. (18.22).

Finally, assuming eq. (18.22) and Hermiticity of P and Γ^5, we find that

$$[P,\Gamma^5] = -[P,\Gamma^5]^\dagger \tag{18.29}$$

is skew-Hermitian. Its eigenvalues are pure imaginary, $\sigma([P,\Gamma^5]) = \{i\lambda_j\}_j$ for $\lambda_j \in \mathbb{R}$. The trace over the operator squared is zero by assumption, thus

$$0 = \sum_j (i\lambda_j)^2 = -\sum_j \lambda_j^2\,. \tag{18.30}$$

This is only possible if $\lambda_j = 0$ for all j. Thus, eq. (18.22) implies eq. (18.18) concluding the proof. □

18.2 WEAK CHIRALITY PRESERVATION

The property of weak chirality preservation $[P,\Gamma^5] = 0$ is worth further inspection.

Lemma 18.2.1 (Implications of weak chirality preservation). *Assuming weak chirality preservation and K is a Γ^5-Hermitian operator on the fine-grid lattice, then it holds*

$$\widehat{\Gamma^5}\hat{K} = \widehat{\Gamma^5 K}\,, \tag{18.31}$$

$$\hat{K}\widehat{\Gamma^5} = \widehat{K\Gamma^5}\,, \tag{18.32}$$

$$\hat{K} \textit{ is } \widehat{\Gamma^5}\textit{-Hermitian.} \tag{18.33}$$

Proof. The first equation is readily shown as

$$\widehat{\Gamma^5}\hat{K} = R\Gamma^5 TRKT = R\Gamma^5 PKT = RP\Gamma^5 KT = R\Gamma^5 KT = \widehat{\Gamma^5 K}\,. \tag{18.34}$$

The second equation is just the same again, and for the third equation we note that Hermiticity is preserved when coarsening and since $\Gamma^5 K$ is Hermitian by assumption, so is $\widehat{\Gamma^5 K}$ and $\widehat{\Gamma^5}\hat{K}$, thus, $\hat{K}^\dagger = \Gamma^5\hat{K}\Gamma^5$. □

These are properties we may wish to have on the coarse grid, but we note that for case (2wc) in table 18.1 this is satisfied, although $\hat{K}$ can become ill-conditioned as seen later.

18.2.1 *Singular value decomposition*

On the fine-grid, the singular value decomposition (SVD) of a Γ^5-Hermitian operator K is

$$K = \sum_i |\lambda_i| (\Gamma^5 \psi_i) \tilde{\psi}_i^\dagger\,, \tag{18.35}$$

where the ψ_i and λ_i are eigenmodes and eigenvalues of the Hermitian operator $\Gamma^5 K$ and $\tilde{\psi}_i = \text{sign}(\lambda_i)\psi_i$. The singular values $\sigma_i = |\lambda_i|$ are the magnitudes of eigenvalues λ_i of $\Gamma^5 K$. We note that the case (2sv) resembles the singular value decomposition of D.

Lemma 18.2.2 (Coarse singular value decomposition). *If chirality is weakly preserved, $[P,\Gamma^5] = 0$, the singular value decomposition of $\hat{K}$ is equivalent to eq.* (18.35) *but with every symbol hatted.*

Proof. Weak chirality preservation implies $\widehat{\Gamma^5}$-Hermiticity of $\hat{K}$ and $\widehat{\Gamma^5}$ is an involution as of lemma 18.2.1. As in the fine case, we look at the eigenvalue decomposition of the coarse Hermitian operator,

$$\widehat{\Gamma^5 K} = \widehat{\Gamma^5}\hat{K} = \sum_i \hat{\lambda}_i \hat{\psi}_i \hat{\psi}_i^\dagger\,, \tag{18.36}$$

with real eigenvalues $\hat{\lambda}_i$ and eigenvectors $\hat{\psi}_i$. Using that $\widehat{\Gamma^5}$ is an involution, we obtain

$$\hat{K} = \widehat{\Gamma^5}\widehat{\Gamma^5}\hat{K} = \sum_i \hat{\lambda}_i \widehat{\Gamma^5}\hat{\psi}_i\hat{\psi}_i^\dagger \,, \tag{18.37}$$

and with definitions $\hat{\sigma}_i = |\hat{\lambda}_i|$ and $\tilde{\hat{\psi}}_i = \text{sign}(\lambda_i)\hat{\psi}_i$ we get eq. (18.35) with hatted quantities. □

An implication of the singular value decomposition eq. (18.35) is that the condition numbers of the non-Hermitian and Hermitian operators are all equal

$$\kappa(K) = \kappa(\Gamma^5 K) = \kappa(K\Gamma^5)\,, \tag{18.38}$$

and according to lemma 18.2.2 the same holds for their weak chirality preserving coarse variants. Regarding coarse operator condition, it therefore does not matter if we coarsen the Dirac operator itself or its Hermitian version, $Q = \Gamma^5 D$.

18.2.2 *Eigenvalues and determinant*

Γ^5-Hermiticity of K also impacts the determinant and eigenvalues. A careful analysis reveals that the determinant of K is real and eigenvalues are either real or come in complex conjugate pairs.

Lemma 18.2.3 (Coarse eigenvalues and determinant)**.** *If chirality is weakly preserved, eigenvalues of $\hat{K}$ are either real or come in complex conjugate pairs and its determinant is real.*

Proof. On the coarse grid, we can directly infer the determinant as $\det\left(\widehat{\Gamma^5}\right) = \pm 1$, because of lemma 18.1.1. Then for the determinant of $\hat{K}$, we find

$$\det\left(\hat{K}\right)^\star = \det\left(\hat{K}^\dagger\right) = \det\left(\widehat{\Gamma^5}\right)^2 \det\left(\hat{K}\right) = \det\left(\hat{K}\right)\,, \tag{18.39}$$

i. e. the determinant is real and eigenvalues are either real or come in complex conjugate pairs as on the fine grid. □

Summarizing lemmas 18.2.1 to 18.2.3, weak chirality preservation transfers many properties from the fine to the coarse grid. To represent the Dirac operator on the coarse grids properly, it is clearly not a bad idea to preserve chirality. From table 18.1, we see that all cases except (1) lead to $[P, \Gamma^5] = 0$. On the other hand, case (2wc) inherits all the properties, but projects to only one of the two chiralities which will turn out to be not enough.

18.3 NUMERICAL RANGE

Before we analyze coarsenings and operators, we need some definitions.

1. The *numerical range* of an operator $A\colon \mathcal{V} \to \mathcal{V}$ is defined as
$$W(A) := \left\{ \psi^\dagger A \psi \,\middle|\, \psi \in \mathcal{V} \text{ and } \|\psi\| = 1 \right\}. \tag{18.40}$$
2. The *open right-half plane* is defined as $\mathbb{H} = \{z \in \mathbb{C}\colon \operatorname{Re}(z) > 0\}$.
3. An operator A is called *accretive* if its numerical range is a subset of the open right-half plane; $W(A) \subset \mathbb{H}$.
4. Finally, the *convex hull* of a set of complex numbers $\mathcal{A} \subset \mathbb{C}$ is defined as the smallest convex set in $\mathbb{C}$ that contains $\mathcal{A}$, mathematically
$$\operatorname{conv}\{\mathcal{A}\} = \left\{ \sum_{j=1}^{n} \lambda_j z_j \,\middle|\, z_j \in \mathcal{A},\ \lambda_j \geq 0,\ \sum_{j=1}^{n} \lambda_j = 1,\ n \in \mathbb{N} \right\}. \tag{18.41}$$
This is the union of all convex combinations of points in $\mathcal{A}$.

Theorem 18.3.1 (The numerical range). *This theorem collects some standard properties and results about the numerical range of an linear operator $A\colon \mathcal{V} \to \mathcal{V}$, that will become important in this chapter. For proofs, see [287].*

$$W(A) \text{ is convex, closed and compact}\,, \tag{18.42}$$
$$W(\alpha \mathbb{1} + \beta A) = \alpha + \beta W(A)\,, \tag{18.43}$$
$$W(A+B) \subseteq W(A) + W(A)\,, \tag{18.44}$$
$$W(A^\dagger) = W(A)^\star\,, \tag{18.45}$$
$$W(UAU^\dagger) = W(A) \text{ for any unitary } U\,, \tag{18.46}$$
$$\sigma(A) \subseteq \operatorname{conv}\{\sigma(A)\} \subseteq W(A)\,, \tag{18.47}$$
$$W(A) \text{ accretive} \iff A + A^\dagger \text{ HPD}\,, \tag{18.48}$$
$$A \text{ normal} \iff \operatorname{conv}\{\sigma(A)\} = W(A)\,. \tag{18.49}$$

The abbreviation HPD means Hermitian positive definite and $\sigma_{max}(A)$ is the largest singular value of A or $\sigma_{max}(A) = \|A\|_2$. The sum of two sets in eq. (18.44) *is the Minkowski sum; $X + Y = \{x + y \mid x \in X,\ y \in Y\}$. Equation* (18.42) *is the famous theorem by Toeplitz and Hausdorff [288, 289].*

The numerical range of an operator on the fine grid is related to the numerical range of the coarse operator.

Theorem 18.3.2 (Numerical range of coarse operators)**.** *Assume K to be an operator acting on the fine-grid lattice, $K\colon \mathcal{V} \to \mathcal{V}$ and $\hat{K}$ a coarsening of K acting on a coarse grid, $\hat{K}\colon \hat{\mathcal{V}} \to \hat{\mathcal{V}}$. Then*

$$W(\hat{K}) \subseteq W(K)\ . \tag{18.50}$$

Proof. We look at the numerical range of the coarse operator

$$\begin{aligned} W(\hat{K}) &= \left\{ \hat{\psi}^\dagger \hat{K} \hat{\psi} \;\middle|\; \hat{\psi} \in \hat{\mathcal{V}} \text{ and } \|\hat{\psi}\| = 1 \right\} \\ &= \left\{ (R\psi)^\dagger \hat{K} (R\psi) \;\middle|\; \psi \in \mathcal{V} \text{ and } \|R\psi\| = 1 \text{ and } P\psi = \psi \right\} \\ &= \left\{ \psi^\dagger TRKTR\psi \;\middle|\; \psi \in \mathcal{V} \text{ and } \|R\psi\| = 1 \text{ and } P\psi = \psi \right\}\ , \end{aligned}$$

where we've used that $R^\dagger = T$. We use that $P = P^\dagger = TR$ and $\|R\psi\| = (R\psi)^\dagger R\psi = \psi^\dagger P\psi = \|\psi\|$ if $P\psi = \psi$ and find

$$\begin{aligned} W(\hat{K}) &= \left\{ \psi^\dagger K\psi \;\middle|\; \psi \in \mathcal{V} \text{ and } \|\psi\| = 1 \text{ and } P\psi = \psi \right\} \\ &\subseteq \left\{ \psi^\dagger K\psi \;\middle|\; \psi \in \mathcal{V} \text{ and } \|\psi\| = 1 \right\} \\ &= W(K)\ . \end{aligned}$$

□

Importantly, this result holds for every imaginable coarsening of K. Accretivity eq. (18.48), is a very strong property, since according to the theorem above, every coarse operator is analytically invertible irrespective of the coarsening strategy. As we will see, the Dirac operator is *not* accretive.

18.3.1 *Hermitian positive definite systems*

First, we will look at coarsenings of Hermitian positive definite (HPD) systems. They form the easiest class of operators and we will analytically proof that for such an operator the condition numbers obey $\kappa(\hat{A}) \leq \kappa(A)$, irrespective of how it is coarsened.

Theorem 18.3.3 (Condition of coarse HPD operators)**.** *Assume A to be a Hermitian positive definite operator acting on the fine-grid lattice, $A\colon \mathcal{V} \to \mathcal{V}$.*

Then the condition number of the coarse-grid operator $\hat{A}$ is bound from above by the condition number of the fine-grid operator,

$$\kappa(\hat{A}) \leq \kappa(A)\ . \tag{18.51}$$

Proof. A is Hermitian and positive definite, thus invertible: $\forall \lambda \in \sigma(A)$ we have $\lambda > 0$. A is normal, and according to eq. (18.49) its numerical range is equal to its spectral hull. Its spectral hull is the interval in $\mathbb{R}_+$ delimited by the smallest and largest eigenvalue of A, i. e.

$$W(A) = \text{conv}\{\sigma(A)\} = [\lambda_{min}, \lambda_{max}] \subset \mathbb{R}_+\ . \tag{18.52}$$

Defining the smallest and largest eigenvalue of $\hat{A}$ as μ_{min} and μ_{max} respectively, we use theorem 18.3.2 and find $[\mu_{min}, \mu_{max}] \subseteq [\lambda_{min}, \lambda_{max}]$, thus $\mu_{min} \geq \lambda_{min} > 0$ and $0 < \mu_{max} \leq \lambda_{max}$, which directly implies $\kappa(\hat{A}) \leq \kappa(A)$, since all eigenvalues are positive. As a consequence, $\hat{A}$ is HPD as well. □

Theorem 18.3.3 shows that with a HPD operator we do not need to care about how to coarsen. Coarse operators are better conditioned in *every* case. Meaning that one can always coarsen $D^\dagger D$ instead of D, safely. However, this operator is now next-to-nearest neighbor and its coarse variant $\widehat{D^\dagger D}$ is significantly less sparse and harder to implement efficiently. Nevertheless this has been done in the field [170, 290].

18.3.2 *Symmetries*

We continue with properties that lead to symmetries in numerical ranges.

Lemma 18.3.1 (Symmetry about the real axis)**.** *Assume K be a Γ^5-Hermitian operator, i. e. $K^\dagger = \Gamma^5 K \Gamma^5$. Then the numerical range of K is symmetric about the real axis,*

$$W(K) = W(K)^\star \iff \forall z \in W(K)\colon \bar{z} \in W(K)\ . \tag{18.53}$$

Proof. We use eq. (18.45) as $W(K^\dagger) = W(K)^\star$ and eq. (18.46) subsequently $W(K^\dagger) = W(\Gamma^5 K (\Gamma^5)^\dagger) = W(K)$, since Γ^5 is unitary. □

By substituting $K = D$, the numerical ranges of the pure-Wilson eq. (1.17) and Wilson-Clover Dirac operator eq. (1.18) are symmetric about the real axis, just as their spectra are.

Lemma 18.3.2 (Symmetry about the complex origin)**.** *Assume K be an operator such that there exists a unitary U such that $UKU^\dagger = -K$. Then the numerical range of K is symmetric about the complex origin,*

$$W(K) = -W(K)\ . \tag{18.54}$$

Proof. Straightforwardly $W(K) = W(UKU^\dagger) = W(-K) = -W(K)$, using first eq. (18.46), then eq. (18.43). $\square$

Corollary 18.3.1 (Symmetry about the imaginary axis)**.** *Assume K be an operator as in lemmas 18.3.1 and 18.3.2. Then the numerical range of K is symmetric about the imaginary axis,*

$$W(K) = -W(K)^\star \ . \tag{18.55}$$

Proof. Putting together the results of lemmas 18.3.1 and 18.3.2, we arrive at the result. $\square$

18.3.3 *Dirac operators*

Up to now, the discussion has been general. We will now look at symmetries exposed by Dirac operators on the lattice. The hopping term of the Wilson (Clover) Dirac operator, defined as the off-diagonal part,

$$H(x,y) = \frac{1}{2}\sum_{\mu=}^{3}\left\{U_\mu(x)(1-\gamma_\mu)\delta_{x+\hat{\mu},y} + U_\mu(x-\hat{\mu})^\dagger(1+\gamma_\mu)\delta_{x-\hat{\mu},y}\right\} \ , \tag{18.56}$$

obeys lemmas 18.3.1 and 18.3.2: Consider the parity operator that flips signs of odd lattice points only, $V(x,y) = \delta_{x,y}(-1)^{\sum_\mu x_\mu}$ satisfying $V^2 = \mathbb{1}$ and $V^\dagger = V$. With this the hopping term satisfies

$$VHV^\dagger = -H \quad \text{and} \quad \Gamma^5 H \Gamma^5 = H^\dagger \ . \tag{18.57}$$

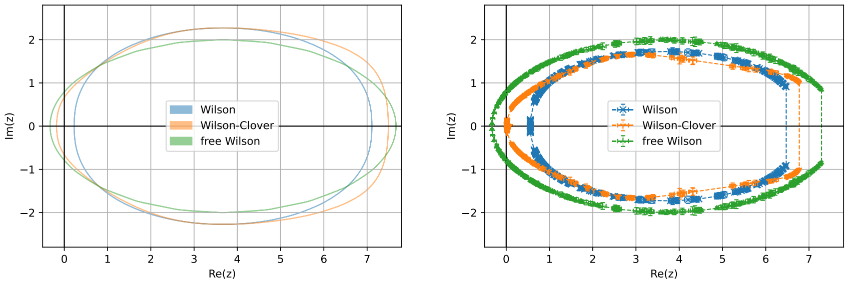
It follows that the numerical range of the hopping term $W(H)$ is symmetric about the real and imaginary axis as well as about the complex origin. It is a superellipse[2] with the complex origin as center point. Furthermore, the unimproved Wilson operator eq. (1.17) is just $D_W = (4+m) - H$. It is thus the same superellipse, but shifted into the right-half plane with center point $\frac{1}{2\kappa} = 4 + m$ (κ is the hopping parameter). It is worth mentioning that D_W is accretive if and only if the horizontal semi-axis s of the superellipse satisfies $s < \frac{1}{2\kappa}$.

Regarding the Wilson-Clover Dirac operator $D_{WC} = D_W + C$, eq. (1.18), with the Hermitian, indefinite Clover-term C, we know from theorem 18.3.1, that

$$W(D_{WC}) \subseteq \frac{1}{2\kappa} + W(H) + W(C) \ , \tag{18.58}$$

---

[2] A superellipse is a mixture between an ellipse and a rectangle. It is specified by the set of points (x,y) that satisfy the equation $|\frac{x}{s}|^n + |\frac{y}{t}|^n = 1$, where s,t are horizontal and vertical semi-axes and $n \geq 2$. In our case, the point (x,y) is identified with the complex number $z = x + iy$.



(a) Numerical range

(b) Spectral hull

FIGURE 18.1: Boundary estimates of the numerical ranges (a) and spectral hulls (b) of the Wilson and Wilson-Clover Dirac operators.

and that $W(C)$ is the real line segment limited by its largest and smallest algebraic eigenvalues due to eq. (18.49). For the numerical range $W(D_{WC})$, the superellipse of $W(D_W)$ is thus squeezed or stretched along the real axis by the values of $W(C)$, but the imaginary extent is not altered, since $W(C)$ is real[3].

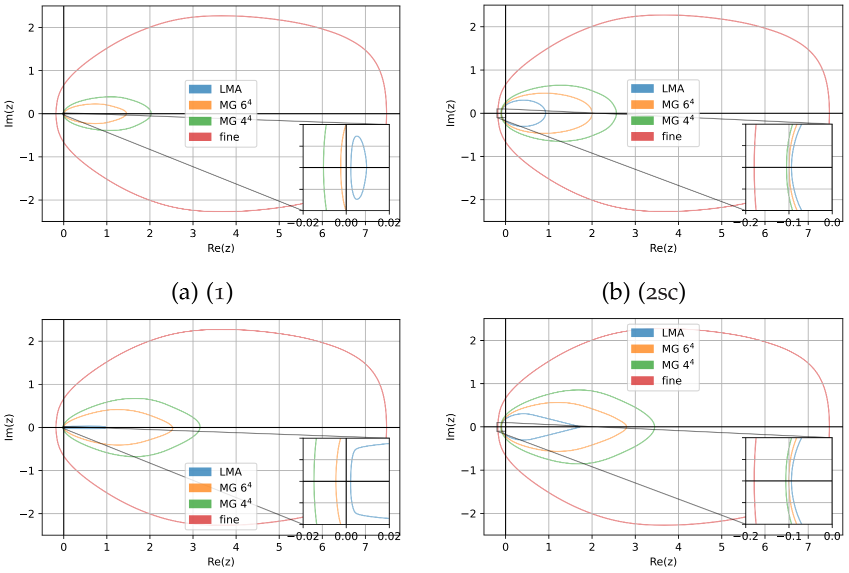
We have determined the numerical ranges of the Wilson and Wilson-Clover Dirac operators of a representative configuration of a dynamical periodic lattice in fig. 18.1a (see appendix C for an algorithm). We can clearly see the symmetries discussed above. On the same gauge config, the Clover-term stretches the numerical range into the left-half plane leading to $0 \in W(D_{WC})$ – a loss of accretivity. Then, coarse operators have the potential for zero or arbitrarily small eigenvalues.

Numerical ranges of different coarse Wilson-Clover Dirac operators are plotted in fig. 18.2. Lemma 18.3.1 suggests that the (1) and (2nc) cases, should not show symmetry about the real axis, because the coarse operator is not Γ^5-Hermitian. It is barely visible in the plots, but those numerical ranges are indeed *not* symmetric, whereas the ones in figs. 18.2b and 18.2d are. For a numerical verification of the numerical range and spectral hull symmetries, please refer to appendix D.

Furthermore the numerical range around the complex origin is related to the low mode subspace. Thus, we want this regime to be well covered. Comparing figs. 18.2b and 18.2d which correspond to the strong chirality

---

[3] For stabilized Wilson fermions [291] the exponential of the Clover term is taken instead. The Clover term – now HPD – is *guaranteed* to move the numerical range further into the right-half plane, resulting in a higher potential (but no guarantee) for accretivity.



(a) (1)

(b) (2sc)

(c) (2nc)

(d) (4)

FIGURE 18.2: Numerical range boundary estimates of coarse and fine Wilson-Clover Dirac operators. All subspaces were generated with $N_c = 20$ low modes of $\Gamma^5 D$. The panels correspond to spin projectors in table 18.1. Every panel shows different coarsening block sizes, where “LMA” corresponds to low-mode averaging, “fine” to the fine-grid operator and the MG aggregate block sizes are indicted in the labels.

preserving cases, (2sc) and (4) in table 18.1, the coarse numerical range around the origin is identical (see insets), suggesting that keeping all 4 spins does not give any more benefit than just keeping the 2 chiral ones.

18.4 SPECTRAL HULL

We continue with studying the spectral hull, $\mathrm{conv}\{\sigma(A)\}$, for fine and coarse Dirac operators. According to theorem 18.3.1, the spectral hull is always contained in the numerical range of an operator. For a non-normal operator they do not coincide. Comparing the areas of the numerical range and the spectral hull can be a measure for non-normality.

We have determined the spectral hulls of the Wilson and Wilson-Clover Dirac operators fig. 18.1b (see appendix C for an algorithm). Both spectral hulls lie in the open right half-plane, a property known as positive stability. We see that the spectrum of the Wilson-Clover operator underwent the same elongation caused by adding the Clover-term, similar to the numerical range.

Eigenvectors associated to the extremal eigenvalues lying on the boundary of the numerical range are orthogonal to all other eigenvectors [287], i. e. the operator restricted to these eigenvectors is normal. Notably, the numerical range and spectral hull of the free Wilson operator (with unit gauge fields) are identical, except in the region corresponding to the largest real parts. The operator is acting normal on an eigenvector ξ associated to an extremal eigenvalue as well as on the corresponding eigenvector $\Gamma^5\xi$ of $D^\dagger$,

$$[D, D^\dagger]\xi = [D, D^\dagger]\Gamma^5\xi = 0\,. \tag{18.59}$$

According to figs. 18.1a and 18.1b, this is not true for the Wilson or Wilson-Clover operator with dynamical gauge fields.

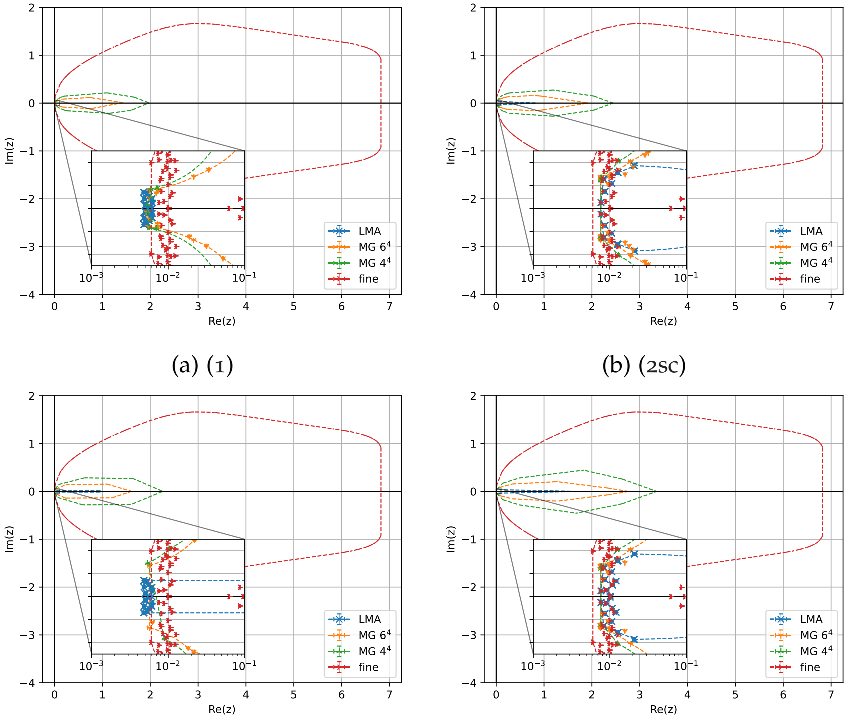
Spectral hulls of different coarse Wilson-Clover Dirac operators are plotted in fig. 18.3. If chirality is strongly preserved, we observe spectral hull inclusion from coarser to finer grids,

$$\operatorname{conv}\{\sigma(\hat{D}_{\text{LMA}})\} \subseteq \operatorname{conv}\{\sigma(\hat{D}_{6^4})\} \subseteq \operatorname{conv}\{\sigma(\hat{D}_{4^4})\} \subseteq \operatorname{conv}\{\sigma(D_{\text{fine}})\}\,. \tag{18.60}$$

This is a strong result, but unfortunately it does not directly translate to $\kappa(\hat{D}) \leq \kappa(D)$ yet, because D is non-normal. The coarse spectra seem under control, but what about the coarse singular values responsible for the numerical condition? If chirality is weakly preserved, lemma 18.2.2 showed equality of condition numbers of Hermitian and non-Hermitian operators. Since, we cannot infer condition numbers from the above, we have to look at the spectral properties of the Hermitian operators and their coarsenings.

To summarize, we collected condition numbers of relevant fine and coarse Hermitian and non-Hermitian operators in table 18.2. Clearly the problem of spurious eigenvalues is not as severe for coarsenings of the non-Hermitian Dirac operator as compared to the Hermitian version, where the condition number explodes and thus quickly becomes numerically inaccessible. It is the smallest magnitude singular value (fourth column) responsible for the operators to become ill-conditioned as the highest one is capped by the maximal singular value on the fine grid. Boldface quantities coincide with the smallest singular value of the fine grid up to determination precision, whereas gray quantities fall below it.

This shows that weak chirality preservation alone is not enough to guarantee well conditioned systems. For the strong chirality preserving cases



(a) (1)

(b) (2sc)

(c) (2nc)

(d) (4)

Figure 18.3: Spectral hull boundary estimates of coarse and fine Wilson-Clover Dirac operators. This plots series corresponds to fig. 18.2. Note that the inset x-axis is log-scale, meaning that the convex hull might not look convex everywhere, because the logarithm is a non-linear transformation.

on the other hand, the smallest singular value is capped by the smallest one on the fine grid. Empirically, we find that strong chirality preservation leads to preservation of extremal eigen- and singular values. According to eq. (18.38), the singular values of coarse Hermitian and non-Hermitian operators are equal, as are their condition numbers. Spectra of Hermitian operators are more easily accessible numerically, due their real spectra and orthogonality of eigenvectors which leads to greater stability and efficiency of eigensolvers. For that reason, we further investigated in the next section a larger chunk of the low lying spectrum of Hermitian Dirac operators and how they behave under coarsening.

Operator A	Grid	Chirality preservation	$\sigma_{min}(A)$	$\sigma_{max}(A)$	$\kappa(A)$
D_{WC}, Q_{WC}	fine	-	**5.5e−3**	7.6	1380
$\hat{D}_{WC}$	LMA	no	4.2e−3	0.016	3.8(1)
(1)	MG 6^4	no	4.4e−3	1.5	328(8)
	MG 4^4	no	4.5e−3	2.0	450(10)
$\hat{Q}_{WC}$	LMA	no	**5.5e−3**	0.023	4.16(7)
(1)	MG 6^4	no	6.7e−4	1.1	1615(2)
	MG 4^4	no	4.1e−6	1.6	383 881(1)
$\hat{D}_{WC}, \hat{Q}_{WC}$	LMA	strong	**5.5e−3**	1.0	185(3)
(2sc)	MG 6^4	strong	**5.5e−3**	2.0	366(6)
	MG 4^4	strong	**5.5e−3**	2.6	466(8)
$\hat{D}_{WC}, \hat{Q}_{WC}$	LMA	weak	6.2e−2	1.7	26.9
(2wc)	MG 6^4	weak	2.4e−4	2.8	11 659(1)
	MG 4^4	weak	1.6e−5	3.4	213 179(1)

Table 18.2: Extremal singular values $\sigma_{min,max}(A)$ and condition numbers $\kappa(A)$ for some coarse and fine, Hermitian and non-Hermitian Dirac operators. D_{WC} indicates the Wilson-Clover Dirac operator and Q is the Hermitian one $Q = \Gamma^5 D$. For all coarsenings the $N_c = 20$ lowest modes of Q_{WC} were taken. Gray quantities indicate smallest singular values smaller than the fine grid one. Associated operators are numerically problematic.

18.5 NUMERICAL EIGENVALUE STUDY

In this section, we investigated the Hermitian Dirac operator $Q = \Gamma^5 D$ and its coarsenings and looked into a large chunk of its low lying spectrum to better understand the findings until now. Condition numbers of Q and D are equal and with weak chirality preservation the same holds for the coarse operators. We generated multigrid subspaces in two different ways. From what we have learned so far, it is enough to only investigate the two cases:

- case (1): naive coarsening with $\hat{N}_s = 1$ remaining coarse spins.

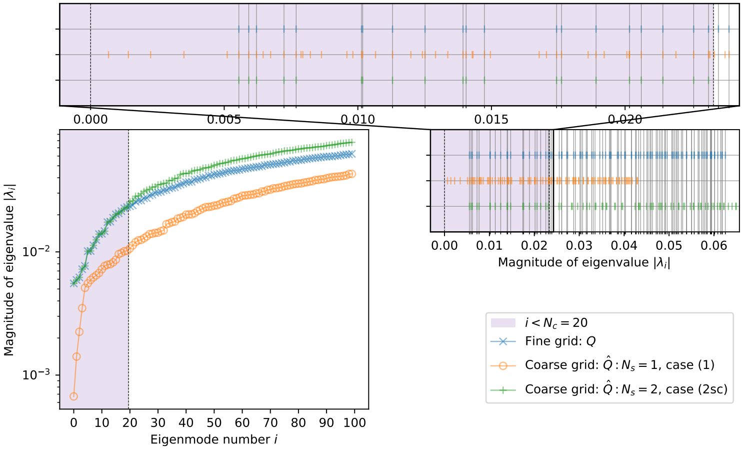


FIGURE 18.4: Study of low lying spectra of coarse and fine Hermitian Dirac operators. Lower left: eigenvalue magnitude versus mode number of the 100 lowest modes, lower right: eigenvalue magnitudes between fine and different coarse operators for comparison, and top panel: zoom of lower right of the interesting region. The three operators considered are fine in blue, coarse case (1) in yellow and coarse case (2sc) in green, c.f. table 18.1. Both coarse operators were constructed using a 6^4 block size and $N_c = 20$ low modes of Q. This plot is taken from publication [P1].

- case (2sc): strong chirality preservation with $\hat{N}_s = 2$ coarse spins.

Figure 18.4 presents the results. In blue the first 100 actual smallest-magnitude eigenvalues of the fine-grid Hermitian operator $Q = \Gamma^5 D$ are plotted. The $N_c = 20$ lowest ones where taken to generate the coarse subspace for both scenarios. For both coarse operators we also plotted their first 100 smallest-magnitude eigenvalues in yellow and green. In the lower left panel their magnitudes are plotted, the top panel is a zoom to the relevant region of the lower right panel, which shows a tick for every eigenvalue at its magnitude for comparison. Every fine-grid eigenvalue has a grey vertical line drawn for eye guidance. The shaded region denotes the first 20 fine-grid modes for which we expect perfect overlap by construction for both coarse subspaces as can be confirmed in the zoomed panel.

When coarsening all spins (yellow), we observe coarse eigenvalues lower than the lowest fine one as anticipated in the last sections. However, we also see other spurious ones in the purple region. These low eigenvalues are responsible for ill-conditioned coarse systems and the slowing down of multigrid as a preconditioner as well as multigrid as variance reduction method as introduced. On the other hand, when chiral properties are strongly preserved (green), we cannot find a single eigenvalue in the critical purple region that is not an inherited one from the fine-grid. Coarse eigenvalues above the purple region are distorted in both cases as expected, but they do not have impact in the condition.

18.6 NUMERICAL VARIANCE STUDY

The preserved chiral properties do not only have impact on the spectrum but also the variance contribution of coarse subspaces. We generated the same two subspaces as in the previous section for the lattice F7 with $N_c = 50$ low modes of Q for the "LMA" and "2-level MG LMA" estimators from table 17.2 and looked at their L0 variance contribution. Figure 18.5 shows the impact of coarse chirality on the variances. The yellow data corresponds to the (1) case, whereas the green data to the (2sc) case. We can clearly see a tremendous benefit from of strong chirality preservation as it removes the bulk of the variance from the expensive L0 term. Variance contributing to the large distance regime therefore originates from both chiral sectors equally. Even though the (2sc)-subspace is twice as large as the one from (1), it is a tradeoff absolutely worth doing, not only for the coarse condition but also for its variance contribution.

18.7 METHODOLOGY

For an explanation of the algorithm for numerical determinations of spectral hulls, numerical ranges and the symbols used in this chapter, please refer to appendix C. For numerical range determinations [292], we ignored all symmetries and used $N_\theta = 128$ uniformly distributed angles for all operators

$$\Theta = \left\{ \frac{2\pi n}{N_\theta} \mid n = 0, \dots, N_\theta - 1 \right\} . \tag{18.61}$$

We have plotted the shaded region $W_{\text{out}}(A) \setminus W_{\text{in}}(A)$ as approximation to $\partial W(A)$. Note that in the plots, this region is so small that it appears visually as a line and is in most cases even smaller than the width of the line.

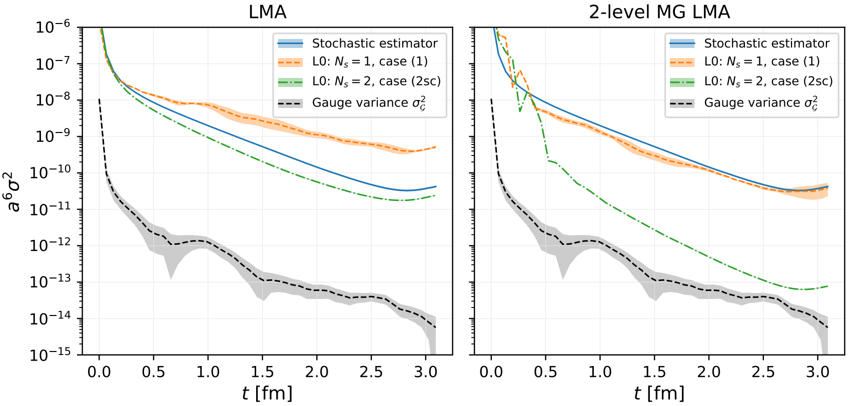


FIGURE 18.5: Variance contribution of the L0-term if all spin degrees of freedom are coarsened case (1) in yellow as compared to strong chirality preservation case (2sc) in green. Variance contributions of both LMA (left) and MG LMA (right) profit from these considerations, although for LMA, the benefit is more subtle. This plot is taken from publication [P1].

The eigenvalues and eigenvectors of the Hermitian systems

$$H_\theta = \frac{1}{2}\left(e^{i\theta}A + e^{-i\theta}A^\dagger\right) \tag{18.62}$$

were determined using the Generalized Davidson method with locally optimal restarting from the library PRIMME [283]. The eigenvalues close to some arbitrary shift $z \in \mathbb{C}$ for the non-normal fine grid Dirac operators were determined using the implicitly restarted Arnoldi algorithm [293] implemented in QUDA[4], whereas for the non-normal coarse grid operators we used the Krylov-Schur method implemented in SLEPc [294].

The condition numbers were determined by estimating the largest and smallest magnitude eigenvalues of the HPD operator $A^\dagger A$, whose square root equal the largest and smallest singular values of A. For this again the Generalized Davidson method with locally optimal restarting from the library PRIMME [283] was used.

---

[4] The shifts had to be implemented manually.

18.8 SUMMARY

The numerical range studies revealed important information about Wilson-type Dirac operators. Critically, the Wilson-Clover operator is not accretive, even more so for physical masses, i. e. the operator itself, but also its coarse variants have the potential for zero or arbitrarily small modes in their spectra. Weak chirality preservation $[P, \Gamma^5] = 0$ is responsible for the inheritance of many properties, amongst others equal singular values for non-Hermitian $\hat{K}$ and Hermitian $\widehat{\Gamma^5 K}$ coarsenings. Empirically we observed spectral hull inclusion and overlap of the N_c lowest modes for coarsening schemes that strongly preserve chiral properties, no so for weak chiral, non-chiral or no spin preservation at all. Therefore, it is the chiral degree of freedom alone and only if coarsened symmetrically, that is responsible for inheriting well behaved spectra.

For Hermitian positive definite systems, we are provably guaranteed to have better conditioned coarse systems, no matter how we coarsen. We believe that careful study of numerical range theory on other lattice discretizations reveals further algorithmic and theoretical insights.

Concluding, the main finding of this chapter is:

For an invertible Γ^5-Hermitian fine-grid Dirac operator D (Γ^5 being Hermitian), we find strong numerical evidence that if chirality is strongly preserved, i. e. if

$$[P, \Gamma^5] = 0 \quad \textbf{and} \quad \operatorname{tr}\left(\widehat{\Gamma^5}\right) = 0 \,, \tag{18.63}$$

then coarse Dirac operators $\hat{D}$ satisfy

$$\kappa(\hat{D}) \leq \kappa(D) \,. \tag{18.64}$$

19

CONCLUDING REMARKS OF PART II

TODO: * Conclusions * Outlook * reaching gauge noise is hard. multi-level schemes (that reduce gauge noise) make it even harder

This section/chapter was proofread 1 times.

We have introduced a novel variance reduction method called *multigrid low-mode averaging* (MG LMA), combining the efficient multigrid preconditioner with low-mode deflation. The method can be applied to any n-point function. Beneficial are expected specially where full translation averaging is challenging and the long distance (LD) regime is of interest. This is the case for a large class of observables, including meson two-point functions important in the high-precision determination of the leading-order HVP of the muon $g-2$, but also baryon correlators important for hadron spectroscopy. For disconnected diagrams leading to one-point functions, we do not expect a huge gain in variance reduction, since the success of methods like frequency splitting suggest that noise originates from the high modes, rather than the low modes. However, as a rule of thumb, whenever LMA is beneficial, MG LMA should be too.

The proposed method shows linear cost scaling in the volume, overcoming the V^2-problem. The method shifts expensive Dirac equation solves to cheap coarse-grid Dirac equation solves, reducing the problem size by factors 30 to 500 – a factor that can be reinvested directly into additional statistics for the translation average in the long distance. Two concepts were crucial for this. First, local coherence of low modes allows building efficient large approximate low-mode subspaces with good overlap to the true subspace using only a small number of precisely determined lowest modes. Most importantly, numerical results show that this number is independent of the lattice volume. Second, strongly preserving chirality on the coarse subspaces empirically led to well-conditioned coarse operators resulting in efficient solves of the coarse Dirac equation. We showed numerically that the method can reach the minimal (gauge) variance on all investigated lattices with cost linear in the volume, as opposed to the standard stochastic estimator and traditional LMA that requires substantially more low modes.

TL;DR: recap of method and results, challenges overcome, idea of the method, local coherence, volume scaling, reach gauge variance

The insights when studying chirality on the subspaces and the fact that most lattice discretizations differ in their treatment of chiral symmetry, lead to the conclusion that MG LMA has to be studied and possibly adjusted

TL;DR: chirality on other discretizations on each discretization separately. As multigrid algorithms continue to evolve rapidly, the introduced method has the potential to gain from these improvements.

TL;DR: twisted mass and MG

For twisted mass fermions multigrid had to be adjusted by introducing a pure algorithmic parameter – a factor for the coarse twisted mass [295] – to improve the condition of the coarse grid twisted mass Dirac operator. They observed that, if not adjusted, the solver performance might degrade or stall entirely. If that factor is unequal to one, the coarse operator is not the restriction of the fine one anymore. This trade off might be fine for a preconditioner but might impact the variance contribution and degrade the performance of MG LMA compared to Wilson-Clover fermions.

TL;DR: staggered and MG

The staggered fermion Dirac operator shows spurious eigenvalues on coarse grids [296] making it especially challenging to obtain a performant multigrid implementation. The solution seems to be a transformation of the operator by the Kähler-Dirac spin structure [297, 298] before coarsening. This results in a spectrum similar to the Wilson operator and multigrid can be successfully formulated on the Kähler-Dirac transformed operator which was verified on the Schwinger model [296] and later on full lattice QCD [299]. A straightforward application of MG LMA without taking these developments into consideration may not perform well.

TL;DR: domain wall and MG

For Domain Wall fermions with its Dirac operator spectrum enclosing the origin one usually coarsens the normal HPD system $D_{\mathrm{DW}}^{\dagger} D_{\mathrm{DW}}$ [170, 290] or a related operator (called the Pauli-Villars operator [300]). The latter has only been tested on the 2D Schwinger model. As proven in theorem 18.3.3 for the former which is a HPD system, the coarse operators will always be at least as well conditioned as the fine one, no matter how it is coarsened. Admittedly this operator is more expensive to apply and complex to coarsen as it is a distance-two stencil. On the other hand, when naively coarsening D_{DW}, coarse operators show spurious eigenvalues [300] similar to the staggered case. Again, we do not expect a naive application of MG LMA to the Domain Wall discretization to be successful without considering the developments made in recent multigrid literature.

TL;DR: concluding discretizations

Multigrid low-mode averaging – despite being very successful on the Wilson discretization as shown – has to be studied for all of the above discretizations individually and carefully. There is still a lot of theoretical and algorithmic work to do.

TL;DR: outlook: baryons

A promising imminent application without fundamental changes would be baryon correlators which suffer from expensive translation averaging.

As low-mode averaging is applied to these problems with success, potential exists for MG LMA to perform well.

20

SUMMARY OF THE THESIS

This section/chapter was proofread 0 times.

TL;DR: intro

Both parts of this thesis address different aspects of improved efficiency in lattice QCD simulations, namely offloading observable evaluation to GPUs and scalable variance reduction in long-distance correlation functions. Nevertheless, they both contribute to the same shared objective of high-precision lattice studies. Furthermore, the software developments in part i are directly applicable to part ii.

TL;DR: summary of objectives

The objectives reached in this thesis are:

- **Interface to QUDA**: The interface to the solver suite in QUDA was implemented and is ready to be used in production lattice QCD computations. Several utilities are already utilizing the interface for inversions and are used in production, such as two-point correlation functions, RM123 sea-quark diagrams or valence RM123 contributions. The developments led to a second round of funding in terms of a PASC project [118] aimed at a continuation of this work. For hybrid workloads, we have implemented the dual process grid enabling to scale out the remaining CPU-workload independent of the GPU work. The asynchronous solver allows heterogeneous computing and multiple right-hand side solvers provide better strong scaling.

- **Performance assessment and benchmark**: All parts of the interface were benchmarked in isolation and together with other relevant components. The benchmarks show clearly readiness of the code for larger problem sizes, crucial to go to the continuum and infinite volume limits. On intermediate problems, many ways to improve scaling behavior are present, like multiple right-hand sides or heterogeneous solvers.

- **CI/CD automated testing**: An automated testing pipeline covering all functionality provided by the interface was implemented in the development repository, running on local resources and on GH200 (Grace-Hopper) nodes at CSCS.

- **Algorithmic development of variance reduction methods**: A new variance reduction method called multigrid low-mode averaging was introduced. It is inspired by multigrid, the powerful solver preconditioner, and low-mode averaging, the state-of-the-art method for high-precision determinations of observables suffering from the high translation average cost in the long distance. The most important result: the number of required low modes of the Dirac operator to reach minimal variance is independent of the lattice volume as opposed to traditional LMA schemes. Thus its overall computational cost scales linear in the volume, not quadratic solving the V^2-problem. This makes the translation average in the long distance feasible and minimal variance can be reached as demonstrated systematically in chapter 17. The method is especially powerful on large physical lattices, where traditional estimators cannot reach minimal variance at feasible cost anymore. An explicit methodical demonstration using $\mathcal{O}(a)$-improved Wilson-Clover fermion was given.

- **Theoretical understanding of spectral properties of Dirac operators**: We formalized chirality on coarse subspaces, collected relevant results from the field of numerical ranges, studied spectral properties and variance contributions of coarse systems. We proposed that strong chirality preservation eq. (18.63), leads to well conditioned coarse systems for γ^5-Hermitian Dirac operators. This proposal is backed by strong numerical evidence.

TL;DR: broader impact, scientific context, big picture, GPU

The advancements of this thesis enable the RC* collaboration to harvest the power of modern GPU-based clusters for observable evaluation with the openQxD software package. Based on openQCD, the interface developments in openQxD are directly applicable to openQCD too. Since openQCD is widely used in the field, this enables a large part of the lattice community to profit from the interface code developed in this thesis with minimal additional investment.

TL;DR: broader impact, scientific context, big picture, noise reduction

The variance reduction method MG LMA solves the challenging translation average in the long distance regime with cost linear in the volume. With our successful demonstration using Wilson-Clover fermions in chapter 17, the method is ready for use by that part of the community and a potential for broader adoption across other discretizations is conceivable with suitable modification.

TL;DR: broader impact, scientific context, big picture, observables for MGLMA

Many correlators are affected by the high cost of translation averaging in the long distance. MG LMA, designed to address this issue, can be applied to such observables with the potential to improve efficiency.

20.1 LIMITATIONS

Ever though the GH200 node offers zero-copy memory access among CPUs and GPUs, we are still faced with recurring CPU-GPU interconnect pressure when accessing spinor fields residing on distant memory. On systems without unified memory architecture, this even includes **recurring explicit host–device data transfers**. The need for a field management system arises for an improvement of this inefficiency.

Gauge and clover fields, although not critical in the observable evaluation phase, since they change only slowly or not at all, become problematic when considering the Hybrid Monte Carlo algorithm for gauge field generation. Repeated transfers will slow down the whole process.

The current implementation transfers the entire clover field, when the gauge group is U(3), leading to an addition unnecessary transfer. Equipping QUDA with the ability to generate the U(1) clover term natively would make this transfer obsolete.

When studying the strong scaling behavior of the multigrid solver in section 7.8, we observed quick **strong scaling degradation** on currently available lattice sizes. Although larger lattices showed better scaling, this is a clear restriction for current production runs in terms of node counts. We provide many mechanisms to alleviate this (the blocked or heterogeneous solver), although the amount of algorithmic parameters for performance tuning has increased substantially.

Furthermore, the poor strong scaling behavior of multigrid is expected to impact the variance reduction method too, due to its heavy reliance on coarse lattices constructed via multigrid, limiting its ability to efficiently scale out.

The variance reduction method was tested on **unphysical pion masses** of $m_\pi \approx 270\,\text{MeV}$ instead of the physical value $m_\pi \approx 135\,\text{MeV}$. Since we have not rigorously studied the dependence of the pion mass to the variance contributions, this remains potential gap in the current analysis as presented in chapter 17.

The emergence **large intermediate coarse Dirac operators** with memory footprints exceeding that of the fine-grid as discussed in section 17.8.2 necessitate to introduce block- or multiple-RHS solvers. This introduces significant implementation overhead and requires careful tuning of additional algorithmic parameters.

Even though many of the developments in chapter 18 about coarse chirality are formal and mathematically proven, its main result about coarse operators being better conditioned than fine ones, is solely backed by strong **numerical evidence**. The formal proofs are mere motivations, provide context or attempt to explain the heuristic main result.

Finally, we want to clarify that the method does not *solve* the V^2-problem (quadratic cost in the lattice volume) once and for all. This is an inherent scaling factor of lattice QCD simulations and we never changed its *asymptotic* behavior. All we did is defer its quadratic effect to larger volumes, i. e. we reduced the value of the coefficient in front of V^2. The problem is solved on lattice volumes accessible to the lattice community **currently and in the near future**.

20.2 OUTLOOK

Software developments will continue with a **field unification scheme** considering modern memory hierarchies as proposed in chapter 9, followed by offloading parts of the Hybrid Monte Carlo (HMC) algorithm to GPUs. This will be approached in terms of a continuing PASC project "openQxD: Efficient QCD+QED Simulations with Various Boundary Conditions on GPUs" [118] in the next three years. Even regarding observable evaluation, further functionality such as the U(1) clover field generation or access to QUDA's contraction routines could be considered for offloading.

Regarding code developments, we believe that a sustainable compute strategy includes **heterogeneous workflows**, organizing work in a way to process previous solver results on the CPU while performing the next solves on the GPU simultaneously. Such an approach introduces small changes to existing programs, but at the same time hides many of the current inefficiencies, like the recurring CPU-GPU field transfers, behind the simultaneous processing. Some of the current inefficiencies then become less imminent or vanish entirely and it does not hinder further developments.

Our current implementation of the variance reduction lacks an efficient coarse grid solver. With access to a **true recursive multigrid solver on the GPU** that is capable of multiple RHSs, it is natural to build on this existing functionality when improving the implementation. Future developments should go in that direction as has the highest impact with lowest effort. With access to a powerful multigrid solver in QUDA, all ingredients are given to substantially improve our modest coarse-grid solver degrading the performance of the variance reduction (section 17.8).

Since multigrid preconditioning has to be studied on every **Dirac operator discretization** separately to achieve a performant formulation, we expect our variance reduction method to require the same separate treatment. Future research could be invested in one of the other commonly used discretizations.

Plenty of observables suffer from **infeasible translation average costs in the long distance**. These provide promising applications of the variance reduction method with potential for reduced variance compared to current estimators. Interesting such candidates are baryon correlators, baryon and meson spectroscopy studies, nucleon form factors, spectral densities, decay constants or matrix elements.

A

APPENDIX: BUILDING THE INTERFACE

> *The cecal appendix may contribute to the increase in longevity through a reduction of extrinsic mortality.*
>
> — [301]

This section/chapter was proofread 1 times.

TL;DR: building intro

This chapter discusses the various ways on how to build QUDA and openQxD with and without QUDA support. A recommended compilation from scratch is presented below, first by compiling the QUDA library according to our simulation's needs, then by compiling openQxD with the QUDA library linked to it. QUDA can be compiled in many ways; directly using CMake (appendix A.1) or via spack (appendix A.2). Appendix A.3 describes how openQxD is compiled against QUDA directly, whereas appendix A.4 describes the same on a cluster using user environments.

A.1 BUILDING QUDA USING CMAKE

TL;DR: quda as a library

QUDA needs to be compiled as a library. We refer to the QUDA documentation [144] and its official GitHub page [179] for generic compilation instructions and focus on enabling support for openQxD.

TL;DR: build flags for interface, wilson-clover, mg, mpi, ...

Building QUDA with enabled openQxD interface requires setting a few CMake build-flags. To enable the Wilson-Clover Dirac operator we set the following flags in `CMakeLists.txt` [144]:

```
QUDA_DIRAC_DEFAULT_OFF=ON # disables ALL Dirac operators
QUDA_DIRAC_WILSON=ON      # enables Wilson-Dirac operators
QUDA_DIRAC_CLOVER=ON      # enables Wilson-clover operators
```

We enable the openQCD interface by setting (we may disable all the other interfaces)

```
QUDA_INTERFACE_OPENQCD=ON # enable openQCD interface
```

which builds all necessary code for the interfacing with openQxD. Furthermore, we enable multi-GPU support using MPI and the multigrid solver by

```
QUDA_MPI=ON          # enable MPI
QUDA_MULTIGRID=ON    # enable multigrid preconditioning
```

In a production build one would add desired null space vector sizes to the list

```
QUDA_MULTIGRID_NVEC_LIST=6,24,32
```

This means that, in a multigrid setup, the number of coarse color degrees of freedom can be either 6, 24 or 32. The list can be extended at wish at the cost of compile time. Furthermore, we want to enable double, single and half precision support by

```
QUDA_PRECISION=14   # 4-bit number to specify which precisions we will enable
                    # (8: double, 4: single, 2: half, 1: quarter).
```

For QCD-only and QCD+QED simulations, we require QUDA to be able to store the gauge fields in the compressed formats with 8, 12 or 18 real degrees of freedom for $SU(3)$ fields and with 9, 13 or 18 real degrees of freedom for $U(3)$ fields. For this we enable all reconstruction types by setting

```
QUDA_RECONSTRUCT=7  # 3-bit number to specify which reconstructs we will enable
                    # (4: reconstruct-no, 2: reconstruct-12/13,
                    #  1: reconstruct-8/9).
```

The above mentioned CMake flags are just an excerpt of the many available flags. One might call

```
cmake -LAH
```

in the build directory of QUDA for a list of all available CMake flags together with a brief description. We only covered the ones which are crucial to build openQxD against QUDA. For a comprehensive guide on the remaining compile flags and how to build QUDA, we refer the reader to the official documentation [179].

A.2 BUILDING QUDA USING SPACK

TL;DR: how to build with spack

We have created a Spack [182] package for QUDA, that can be found in the official Spack repositories [302]. The advantage of this is that it makes it very easy to build and install QUDA. This package will be useful in chapter 8 to build and deploy automatically. However, the build instructions above are full of intricate details and pitfalls and one might suffer a lot until one obtains a working build[1]. If Spack is available on a machine, one can just invoke

```
spack install quda cuda_arch=80
```

in a terminal and QUDA is properly built and installed on the current system (the architecture `cuda_arch` of the GPU may have to be adjusted). It is also straightforward to build against a HIP target:

```
spack install quda +rocm amdgpu_target=gfx90a # AMD Instinct MI250
```

This even works on a machine without a GPU installed which might be useful for testing purposes. To build and install QUDA as described in appendix A.1 one would call

```
spack install quda@develop +mpi +multigrid +openqcd \
                                  +wilson +clover cuda_arch=80
```

with the appropriate CUDA architecture.

TODO: Command above only works if feature/openqxd is merged into develop

A.3 BUILDING OPENQXD AGAINST QUDA

After QUDA has been built according to the previous section, we compile openQxD[2]. Note that we require a compiler compliant with C99 and the MPI 3.0 standard [172]. We refer the user to the official openQxD documentation [111] and its GitLab repository [132] on how to compile and focus here on the changes that have to be made to compile against QUDA. Note that we need to dynamically link every openQxD binary that interfaces QUDA.

TL;DR: C99, MPI1.2 and dynamic linking to quda lib

Hence, when compiling an openQxD program, we add the path to QUDA's `include` directory (e.g., `-I<path>` for gcc), the path to QUDA's `lib` directory (e.g., `-L<path>` for gcc), and the named library (e.g., `-lquda` for gcc) in the linking phase. An example `Makefile` can be found under `devel/quda/Makefile` in the openQxD development repository [119].

TL;DR: gcc flags for this

To compile openQxD against QUDA, we require some environment

TL;DR: env vars of openqxd + their description variables to be set properly, listing A.1. The environment variable `GCC` should point to a recent gcc compiler[3], whereas `CC` usually points to the MPI C compiler wrapper of the used MPI implementation[4]. `MPI_HOME` and `MPI_INCLUDE` have to point to directories of the MPI implementation and its include directory where the file `mpi.h` lies[5]. These two directories are passed to the compile commands as `-I${MPI_INCLUDE}` (directories to be searched for header files) and `-L${MPI_HOME}/lib` (directories to be searched for `-l`, i. e. for named libraries.) respectively. The variable `QUDA_BUILD` is very similar. It points to the path where QUDA was built. In the Makefiles it has 3 roles. It is being added to the generated compiler commands as `-I${QUDA_BUILD}/include -L${QUDA_BUILD}/lib` and the flag `-rpath ${QUDA_BUILD}/lib` as an absolute path is being added to the underlying linker in the linking stage. This makes the binary find `libquda.so` even if `LD_LIBRARY_PATH` is not set properly.

---

[1] I did.

[2] In this section we expect QUDA to be built and the shared library `libquda.so` as well as header files, like `quda.h`, to be available under a path known to the reader.

```
export GCC=<path>            # compiler to build dependencies
export CC=<path>             # MPI C compiler wrapper to build source programs
                             # and link object files
export MPI_HOME=<path>       # adds -L${MPI_HOME}/lib to compiler command
export MPI_INCLUDE=<path>    # adds -I${MPI_INCLUDE} to compiler command
export QUDA_BUILD=<path>     # adds -I${QUDA_BUILD}/include -L${QUDA_BUILD}/lib
                             # and passes -rpath ${QUDA_BUILD}/lib to the linker
```

Listing A.1: Environment variables to build openQxD

A.4 UENV

CSCS has changed the way users interact with software on their clusters from the legacy Tcl [184] or Lmod [185] modules to more versatile user environments called uenv. This allows recent library versions and at the same time a provides robust declarative way to describe dependencies using spack. We provide a user environment, deployed via the automated testing pipeline, see chapter 8, that includes a compiled version of QUDA for *Daint Alps* with all flags set properly for usage together with openQxD. As of now, the uenv is in the service namespace and can be downloaded and started as in listing A.2.

---

[3] At the time of writing this thesis, the authors recommend gcc version 12 or newer.

[4] Usually just `mpicc` in the users `$PATH`.

[5] Usually we have `MPI_INCLUDE=${MPI_HOME}/include`.

```
uenv image pull service::quda/experimental   # pull image from store
uenv start quda/experimental --view=openqxd  # start image
```

Listing A.2: Pull and start the QUDA user environment on *Daint Alps*.

The view `openqxd` sets all environment variables necessary to compile openQxD from the git repository and link it to QUDA. This makes it very easy to build openQxD properly on *Daint Alps* against QUDA and run it.

B

APPENDIX: RUNNING ON THE GPU

This chapter describes how to run the interface checks manually[1] and how the interface exposes itself to the user of openQxD once the software has been succesfully compiled. Appendix B.1 discusses the general input file format, whereas appendices B.2 and B.3 discuss the appearances of general and multigrid solver sections in input files.

This section/chapter was proofread 1 times.

TL;DR: running intro

B.1 THE INPUT FILE

The contents of the input file may be written in the traditional openQCD INI-inspired style. If the input file contains a section called `QUDA`, settings from that section will be read in in the bootstrapping phase, i. e. when calling `quda_init()`. An example may look as follows:

TL;DR: infile format for general section

```
[QUDA]
verbosity QUDA_VERBOSE # general verbosity
```

This defaults to `QUDA_VERBOSE` and may be overwritten by verbosities set in solver sections. Possible values include (in order or increasing verbosity) `QUDA_SILENT`, `QUDA_SUMMARIZE`, `QUDA_VERBOSE` and `QUDA_DEBUG_VERBOSE`.

B.2 PARAMETRIZING A SIMPLE SOLVER

It remains to explain how the interface code expects the input file to look like. In our design choices, we made this as flexible as possible and compatible to legacy behavior of openQxD. Therefore, a QUDA solver can be described in the same way as an openQCD solver. A minimal solver section may look as

TL;DR: infile format for solver section + simple example

```
[Solver 3]
solver          QUDA
maxiter         2048
gcrNkrylov      10
tol             1e-4
```

---

[1] For automated testing, please refer to chapter 8.

```
reliable_delta  1e-5
inv_type        QUDA_BICGSTAB_INVERTER
verbosity       QUDA_VERBOSE
solution_type   QUDA_MAT_SOLUTION
solve_type      QUDA_DIRECT_SOLVE
```

The section name in the above example is `Solver 3`, thus the solver identifier is 3. This is the argument `id` that has to be provided when calling the solver with one of the various solver kernels, see sections 5.1.7 to 5.1.9 and 6.2. For the section to be a valid QUDA solver section we need to specify the key-value pair `solver = QUDA`, else the setup function will throw an error message if fed with `id=3`. All the remaining keys parameterize the various members of the `QudaInvertParam` struct and we refer the reader to its documentation [144, 179]. We note that every enum in `<quda>/include/quda_enum.h` can be used as value for the key-value pairs in the input file for readability, so that the user doesn't have to write `2` for `QUDA_GCR_INVERTER` for instance. The example section above is minimal, i. e. it only sets required parameters; it solves the Wilson-Clover Dirac equation using a BiCGSTAB solver without any preconditioning to a relative residual of 10^{-4}. We may have mutiple sections as the one above each one enumerated by a unique identifier `id`.

B.3 PARAMETRIZING A MULTIGRID SOLVER

TL;DR: MG example solver section

This section describes a more involved example of a section that parametrizes a multigrid solver. It may look like

```
[Solver 4]
solver                  QUDA
maxiter                 256
gcrNkrylov              10
tol                     1e-12
reliable_delta          1e-5
inv_type                QUDA_GCR_INVERTER
verbosity               QUDA_VERBOSE
inv_type_precondition   QUDA_MG_INVERTER # use multigrid preconditioning
solution_type           QUDA_MAT_SOLUTION
solve_type              QUDA_DIRECT_SOLVE
mass_normalization      QUDA_MASS_NORMALIZATION
cuda_prec_sloppy        QUDA_SINGLE_PRECISION
cuda_prec_precondition  QUDA_HALF_PRECISION
```

Different from the simple example in the previous section is that we specify a preconditioner via `inv_type_precondition`. Multigrid is special in the sense that it requires many more parameters to be set. Namely the number of levels and an additional section of parameters for each level separately. This is specifed by a second section in the input file that may look as

```
[Solver 4 Multigrid]
n_level                 2
generate_all_levels     QUDA_BOOLEAN_TRUE
run_verify              QUDA_BOOLEAN_FALSE
compute_null_vector     QUDA_COMPUTE_NULL_VECTOR_YES
```

This addtional section describes the multigrid preconditioner to consist of 2 levels. The name of the section is fixed to be `[Solver <N> Multigrid]` where `<N>` has to be substituted with the identifier of the solver section that specifies `inv_type_precondition=QUDA_MG_INVERTER`. The two levels are specified by two separate sections as

```
[Solver 4 Multigrid Level 0]
geo_block_size              4 4 4 4
n_vec                       24
n_vec_batch                 2
verbosity                   QUDA_VERBOSE
spin_block_size             2
precision_null              QUDA_HALF_PRECISION
smoother                    QUDA_CA_GCR_INVERTER
smoother_tol                0.25
smoother_solve_type         QUDA_DIRECT_PC_SOLVE
nu_pre                      0
nu_post                     8
omega                       0.8
cycle_type                  QUDA_MG_CYCLE_RECURSIVE
coarse_solver               QUDA_GCR_INVERTER
coarse_solver_tol           0.25
coarse_solver_maxiter       50
coarse_grid_solution_type   QUDA_MATPC_SOLUTION
location                    QUDA_CUDA_FIELD_LOCATION

[Solver 4 Multigrid Level 1]
precision_null              QUDA_HALF_PRECISION
coarse_solver               QUDA_CA_GCR_INVERTER
smoother                    QUDA_CA_GCR_INVERTER
smoother_tol                0.25
smoother_solve_type         QUDA_DIRECT_PC_SOLVE
spin_block_size             1
coarse_solver_tol           0.25
coarse_solver_maxiter       50
coarse_grid_solution_type   QUDA_MATPC_SOLUTION
```

```
cycle_type                  QUDA_MG_CYCLE_RECURSIVE
nu_pre                      0
nu_post                     8
omega                       0.8
location                    QUDA_CUDA_FIELD_LOCATION
```

Again, the section names are fixed to be `[Solver <N> Multigrid Level <M>]`, where `<N>` is the solver section identifier as above and `<M>` indicates the level, starting from zero being the finest coarse level. Assignable parameters for each level are those in the `QudaMultigridParam` struct that hold arrays of length `QUDA_MAX_MG_LEVEL`, whereas all the other properties can be specified in the general multigrid section named `[Solver <N> Multigrid]`. By this, all possible parameters for a multigrid solver can be specified exhaustively. Some properties have no effect on the finest coarse level, some have no effect on the coarsest level. We refer the reader to the QUDA documentation for further details.

C

APPENDIX: NUMERICAL RANGE AND SPECTRAL HULL ESTIMATORS

This section/chapter was proofread 1 times.

This appendix explains the numerical algorithms used for the boundary estimates of the numerical range (appendix C.1) and spectral hull (appendix C.2) of operators discussed in chapter 18.

C.1 NUMERICAL RANGE

We have estimated the numerical ranges in two ways [292]: by determining a boundary from the inside and a boundary from the outside. What we show as numerical range is the shaded area between these two boundaries. We will use property eq. (18.43) from theorem 18.3.1 in the form

$$W(A) = e^{-i\theta} W(e^{i\theta} A) \tag{C.1}$$

for any $\theta \in [0, 2\pi]$ and the fact that the numerical range of a Hermitian operator H is a part of the real line segment $W(H) = [\lambda_{\min}(H), \lambda_{\max}(H)]$, where $\lambda_{\min}(H)$ and $\lambda_{\max}(H)$ are the smallest and largest algebraic eigenvalues of H, respectively.

One crucial observation is that for any linear operator A, its Hermitian part $H_A = \frac{1}{2}(A + A^\dagger)$ obeys

$$\max_{z \in W(A)} \mathrm{Re}(z) = \max_{r \in W(H_A)} r = \lambda_{\max}(H_A)\,. \tag{C.2}$$

The second equality is clear since H_A is Hermitian. To show the first equality, we note that

$$W(H_A) = \mathrm{Re}[W(A)]\,. \tag{C.3}$$

The set inclusion, $W(H_A) \subseteq \mathrm{Re}[W(A)]$ follows directly from eq. (18.43). For the set inclusion in the other direction, let us assume we have an $r \in \mathrm{Re}[W(A)]$, thus there exist a $z \in W(A)$ such that $r = \mathrm{Re}(z)$ and a ψ such that $z = \psi^\dagger A \psi$. Then

$$r = \mathrm{Re}[\psi^\dagger A \psi] = \frac{1}{2}\left[\psi^\dagger A \psi + \psi^\dagger A^\dagger \psi\right] = \frac{1}{2}\psi^\dagger (A + A^\dagger)\psi \in W(H_A)\,. \tag{C.4}$$

This concludes the other side. Through the set equality eq. (C.3), we also have the maxima equality eq. (C.2).

Therefore, $\lambda_{\max}(H_A)$ gives us a tangent in the complex plane $L(y) = \lambda_{\max}(H_A) + iy$ which is a boundary for the numerical range of A and the point $p = \psi^\dagger A \psi$ is the extremal point in $W(A)$ with largest real-part. The vector ψ is an eigenvector of the system $H_A\psi = \lambda_{\max}(H_A)\psi$.

By rotating A with different angles θ, we may find other tangents to $W(A)$ and boundary points p_θ in other directions. The algorithm is as follows:

1. Define a set of probing angles Θ of $N_\theta \geq 3$ angles, $|\Theta| = N_\theta$. The angles may be uniformly distributed[1] from $[0, 2\pi)$.
2. For every $\theta \in \Theta$:
    - 2.1. Solve for the largest eigenvalue λ_θ and one corresponding eigenvector ψ_θ of the Hermitian operator $H_\theta = \frac{1}{2}\left(e^{i\theta}A + e^{-i\theta}A^\dagger\right)$.
    - 2.2. Compute boundary points $p_\theta = \psi_\theta^\dagger A \psi_\theta \in P_\Theta$.
    - 2.3. Compute tangents $L_\theta(y) = e^{-i\theta}(\lambda_\theta + iy)$ rotated back by θ.
3. Define the inner approximation set as convex hull of all probed boundary points, $W_{\text{in}}(A) = \operatorname{conv}\{P_\Theta\}$.
4. Compute intersection points $q_\theta \in Q_\Theta$ of all tangents L_θ.
5. Define the outer approximation set as convex hull of the tangent intersection points, $W_{\text{out}}(A) = \operatorname{conv}\{Q_\Theta\}$.
6. The region $W_{\text{out}}(A) \setminus W_{\text{in}}(A)$ always contains the boundary $\partial W(A)$ and is an approximation thereof.

For the algorithm above, we have by construction

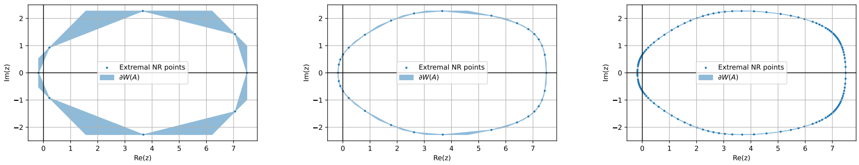
$$W_{\text{in}}(A) \subseteq W(A) \subseteq W_{\text{out}}(A)\,, \tag{C.5}$$

since the numerical range is convex and compact. The probing angles give a resolution for the precision of the two bounding sets. With enough probing angles, the difference between $W_{\text{in}}(A)$ and $W_{\text{out}}(A)$ becomes smaller. We may even define a residual on the precision of the numerical range estimate as the weighted area difference

$$\Delta(A, \Theta) = \frac{\operatorname{Area}(W_{\text{out}}(A)) - \operatorname{Area}(W_{\text{in}}(A))}{\operatorname{Area}(W_{\text{out}}(A))}\,. \tag{C.6}$$

---

[1] If one has information about the shape or symmetries of the numerical range, one can choose the probing angles more suitable for the operator of interest.



(a) $N_\theta = 8$, $\Delta = 0.19$ (b) $N_\theta = 32$, $\Delta = 0.016$ (c) $N_\theta = 128$, $\Delta = 0.001$

FIGURE C.1: Numerical range boundary using different number of probing angles N_θ. The plots show a clear resolution refinement from left to right.

The closer to zero, the better the approximation. In fig. C.1, the same numerical range was determined using different values of N_θ. In the leftmost panel the tangents are still clearly visible.

C.2 SPECTRAL HULL

Once the numerical range is determined and we have access to the boundary points P_Θ, the spectral hull is straightforward to determine: The spectral hull is always a subset of the numerical range, eq. (18.47). Therefore, the eigenvalues closest to the boundary points of the numerical range $p_\theta \in P_\Theta$ either lie on the boundary of the spectral hull or outside of it. It is thus enough to determine the eigenvalues closest to the shifts $p_\theta \in P_\Theta$, or equivalently the smallest magnitude eigenvalue λ_θ of the shifted systems

$$A_\theta \psi = \lambda_\theta \psi \,, \quad \text{with} \quad A_\theta = A - p_\theta \mathbb{1} \,. \tag{C.7}$$

If we do that for p_θ fine enough, we find extremal eigenvalues λ_θ of A in all directions probed by the p_θ. The convex hull of the set $\Lambda_\Theta = \{\lambda_\theta\}_\theta$ gives us a inner approximation of the true spectral hull,

$$\Sigma_{\text{in}} = \text{conv}\{\Lambda_\Theta\} \subseteq \text{conv}\{\sigma(A)\} \,. \tag{C.8}$$

D

APPENDIX: SYMMETRY METRICES

This section/chapter was proofread 1 times.

In order to quantify the symmetry of the numerical range of an operator we have implemented a metric; the intersection-over-union (IoU). Assume we have a continuous subset set of complex numbers $S \subseteq \mathbb{C}$ and a reflection operator $\mathcal{R}\colon z \mapsto \mathcal{R}(z)$ and $\mathcal{R}(S) = \{\mathcal{R}(z) \mid z \in S\}$. We define the asymmetry score with respect to $\mathcal{R}$ for continuous sets as

$$M_c(\mathcal{R}; S) = 1 - \frac{\mathrm{Area}(S \cap \mathcal{R}(S))}{\mathrm{Area}(S \cup \mathcal{R}(S))} \in [0, 1] \ . \tag{D.1}$$

An asymmetry score of 0 implies perfect symmetry, whereas 1 indicates no symmetry.

For the spectrum which is a point set, we have a different metric. Assume we have a point set $S = \{s_1, \ldots, s_n\} \subset \mathbb{C}$ and a set of non-negative real errors $E = \{e_1, \ldots, e_n\}$ such that s_i has error e_i on its real and imaginary parts. We define the asymmetry score for point-sets as

$$M_p(\mathcal{R}; S) = \frac{1}{n} \sum_{i=1}^{n} \delta(i) \in [0, 1] \ , \tag{D.2}$$

$$\delta(i) = \begin{cases} 0 & \text{if } \exists j\colon \left\| s_i - \mathcal{R}(s_j) \right\|_\infty \leq e_i + e_j \\ 1 & \text{otherwise} \end{cases} \tag{D.3}$$

where the norm is defined as

$$\|z - w\|_\infty := \max(\mathrm{Re}(z - w), \mathrm{Im}(z - w)) \ . \tag{D.4}$$

Thus for every point in S, if its error rectangle does not overlap with the error rectangle of any point in $R(S)$, we count 1 to the score.

Table D.1 shows asymmetry scores for various numerical ranges and spectral hulls determined numerically. We see high degrees of symmetry about the real axis for all operators that preserve chirality. Gray colored scores correspond to analytical perfect symmetries. Broken symmetries shows themselves as symmetry scores larger than eigenvalue precision.

Operator	Symmetry $\mathcal{R}$	$M_c(\mathcal{R}; W(D))$	$M_p(\mathcal{R}; \sigma(D))$
Wilson (fine)	real axis	1.2×10^{-5}	0.027
	center point	1.0×10^{-5}	0.29
Wilson-Clover (fine)	real axis	1.1×10^{-5}	0.037
	center point	0.070	0.53
(1), 6^4 (coarse)	real axis	0.013	0.33
	center point	0.031	0.65
(2sc), 6^4 (coarse)	real axis	1.1×10^{-8}	0.063
	center point	0.031	1.0

TABLE D.1: Asymmetry scores eqs. (D.1) and (D.2) for numerical ranges (third column) and spectra (fourth column). For gray quantities, we expect 0 from the analytical studies above. The numerical range eigenvalues were determined up to 10^{-6} and the spectral hull eigenvalues only up to 10^{-1}. We see good agreement with theory up to numerical precision.

E

APPENDIX: NOTATION

E.1 FREQUENTLY USED NAMES

openQCD	A framework for lattice QCD simulations on CPUs.
openQxD	A framework for lattice QCD+QED simulations on CPUs based on openQCD v1.6.
QUDA	A library for performing calculations in lattice QCD on GPUs.

E.2 PHYSICAL CONSTANTS

$\hbar$	reduced Planck constant, $\hbar = 1.054\,571\,817 \times 10^{-34}\,\mathrm{J\,s}$
e	elementary electric charge, $e = 1.602\,176\,634 \times 10^{-19}\,\mathrm{C}$

[71]

E.3 FREQUENTLY USED SYMBOLS AND CONVENTIONS

UNITS. We shall work with natural units $\hbar = c = 1$.

SPINOR PRODUCT. $(\cdot, \cdot)$ denotes the standard spinor product where by convention the first component is conjugated.

SPINOR NORM. The spinor norm is defined to be the one induced by the spinor product, $\|\psi\| = \sqrt{(\psi, \psi)}$.

HERMITIAN TRANSPOSE. The conjugate transpose of a linear operator A or a spinor ψ is denoted with the dagger symbol $A^\dagger$, $\psi^\dagger$ and usually applies to all indices unless stated otherwise.

DIRAC ADJOINT. The Dirac adjoint of a spinor ψ is denoted by $\bar{\psi}$ and defined as $\bar{\psi} = \psi^\dagger i\gamma^0$.

E.4 INDEX NOTATION

A vector v may have multiple indices associated to it. With only one index unless stated otherwise, the i-th component of the vector v will be denoted with a special subscript resembling array-notation as $v_{[i]}$. On the other hand, a regular super- or subscript as v_i or v^i will denote the i-th element of a list of vectors. By this we can easily write $v_{i[j]}$ denoting the j-th *component* of the i-th *vector*.

SPACETIME INDICES. Spacetime indices are denoted with Latin letters starting from $x, y, z, w, \ldots$

SPINOR INDICES. Spinor indices are denoted with Greek letters starting from $\alpha, \beta, \ldots$

LORENTZ INDICES. Lorentz indices are denoted with Greek letters starting from $\mu, \nu, \rho, \sigma, \ldots$

COLOR INDICES. Color indices are denoted with Latin letters starting from $a, b, c, \ldots$

An example carrying multiple indices is the propagator S, for instance

$$S_{[x;y]\,\underset{[ab]}{[\alpha\beta]}}\ . \tag{E.1}$$

E.5 γ-MATRIX BASIS

We work in the Weyl basis,

$$\gamma_0 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \qquad \gamma_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix}, \tag{E.2}$$

$$\gamma_2 = \begin{pmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \end{pmatrix}, \qquad \gamma_3 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \tag{E.3}$$

but all conclusions drawn in this document hold for other choices too.

F

APPENDIX: LICENSE AND AVAILABILITY

> *Free software is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech", not as in "free beer".*
>
> — Richard Stallman

This section/chapter was proofread 1 times.

This work is licensed under a Creative Commons "Attribution-NonCommercial-ShareAlike 4.0 International" license.

F.1 CODE AVAILABLILITY

All code produced for part i of this thesis is licenced under the GNU General Public License (GPL), either version 2 of the License, or any later version. The interface code implemented in openQxD can be found in the Gitlab repository [119], under the branch name `main-thesis-release`. The related interface code in QUDA can be found in the GitHub repository [303] under the branch name `tree/feature/openqxd`. Both merge-requests are currently under review.

All code produced for part ii of this thesis is licenced under the GNU General Public License (GPL), either version 2 of the License, or any later version. The code implemeting the mulrgrid low-mode averaging method in openQxD can be found in the Gitlab repository [119], under the branch name `34-lma-scheme`.

F.2 DATA AVAILABLILITY

All data produced can be found under ref. [304]. A description of the datasets and structure of the data can be found on the Zenodo page.

12. Marinkovic, M. K., Campos, I., Della Morte, M. & Korcyl, P. *Efficient QCD+QED Simulations with openQ*D software* `https://pasc-ch.org/projects/2021-2024/eniac-enabling-the-icon-model-on-heterogeneous-architectures/index.html`. 2021/2024.
13. Jülich Supercomputing Centre. JUWELS Cluster and Booster: Exascale Pathfinder with Modular Supercomputing Architecture at Juelich Supercomputing Centre. *Journal of large-scale research facilities* **7** (2021).
14. HPE. *HPE Superdome Flex and Scale-up Servers* `https://www.hpe.com/emea_europe/en/servers/superdome.html`. (Online; accessed 2025-07-17). 2025.
15. Feynman, R. P. Space-Time Approach to Non-Relativistic Quantum Mechanics. *Rev. Mod. Phys.* **20**, 367 (2 1948).
16. Osterwalder, K. & Schrader, R. AXIOMS FOR EUCLIDEAN GREEN'S FUNCTIONS. *Commun. Math. Phys.* **31**, 83 (1973).
17. Osterwalder, K. & Schrader, R. Axioms for Euclidean Green's Functions. 2. *Commun. Math. Phys.* **42**, 281 (1975).
18. Yang, C. N. & Mills, R. L. Conservation of Isotopic Spin and Isotopic Gauge Invariance. *Phys. Rev.* **96**, 191 (1 1954).
19. Gattringer, C. & Lang, C. B. *Quantum chromodynamics on the lattice* (Springer, Berlin, 2010).
20. Ginsparg, P. H. & Wilson, K. G. A remnant of chiral symmetry on the lattice. *Phys. Rev. D* **25**, 2649 (10 1982).
21. Nielsen, H. B. & Ninomiya, M. Absence of Neutrinos on a Lattice. 1. Proof by Homotopy Theory. *Nucl. Phys. B* **185** (eds Julve, J. & Ramón-Medrano, M.) [Erratum: Nucl.Phys.B 195, 541 (1982)], 20 (1981).
22. Nielsen, H. B. & Ninomiya, M. Absence of Neutrinos on a Lattice. 2. Intuitive Topological Proof. *Nucl. Phys. B* **193**, 173 (1981).
23. Sheikholeslami, B. & Wohlert, R. Improved continuum limit lattice action for QCD with wilson fermions. *Nuclear Physics B* **259**, 572 (1985).

24. Symanzik, K. Concerning the continuum limit in some lattice theories. *Le Journal de Physique Colloques* **43**, C3 (1982).
25. Symanzik, K. Continuum limit and improved action in lattice theories: (II). O(N) non-linear sigma model in perturbation theory. *Nuclear Physics B* **226**, 205 (1983).
26. Lüscher, M., Sint, S., Sommer, R. & Weisz, P. Chiral symmetry and O(a) improvement in lattice QCD. *Nuclear Physics B* **478**, 365 (1996).
27. Kogut, J. & Susskind, L. Hamiltonian formulation of Wilson's lattice gauge theories. *Phys. Rev. D* **11**, 395 (2 1975).
28. Kaplan, D. B. A Method for simulating chiral fermions on the lattice. *Phys. Lett. B* **288**, 342 (1992).
29. Shamir, Y. Chiral fermions from lattice boundaries. *Nucl. Phys. B* **406**, 90 (1993).
30. Frezzotti, R., Grassi, P. A., Sint, S. & Weisz, P. Lattice QCD with a chirally twisted mass term. *JHEP* **08**, 058 (2001).
31. Neuberger, H. Exactly massless quarks on the lattice. *Phys. Lett. B* **417**, 141 (1998).
32. Neuberger, H. More about exactly massless quarks on the lattice. *Phys. Lett. B* **427**, 353 (1998).
33. Hernandez, P., Jansen, K. & Luscher, M. Locality properties of Neuberger's lattice Dirac operator. *Nucl. Phys. B* **552**, 363 (1999).
34. Boyle, P., Juttner, A., Marinkovic, M. K., Sanfilippo, F., Spraggs, M. & Tsang, J. T. An exploratory study of heavy domain wall fermions on the lattice. *JHEP* **04**, 037 (2016).
35. Haar, A. Der Massbegriff in der Theorie der kontinuierlichen Gruppen. *Annals of mathematics* **34**, 147 (1933).
36. Finkler, J. A. & Goedecker, S. Funnel hopping Monte Carlo: An efficient method to overcome broken ergodicity. *The Journal of Chemical Physics* **152**, 164106 (2020).
37. Adler, S. L. Stochastic algorithm corresponding to a general linear iterative process. *Phys. Rev. Lett.* **60**, 1243 (13 1988).
38. Duane, S., Kennedy, A., Pendleton, B. J. & Roweth, D. Hybrid Monte Carlo. *Physics Letters B* **195**, 216 (1987).
39. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics* **21**, 1087 (1953).

40. Hastings, W. K. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **57**, 97 (1970).

41. Gupta, S., Irback, A., Karsch, F. & Petersson, B. The Acceptance Probability in the Hybrid Monte Carlo Method. *Phys. Lett. B* **242**, 437 (1990).

42. Eichten, E., Gottfried, K., Kinoshita, T., Kogut, J., Lane, K. D. & Yan, T. .-. Spectrum of Charmed Quark-Antiquark Bound States. *Phys. Rev. Lett.* **34**, 369 (6 1975).

43. Eichten, E., Gottfried, K., Kinoshita, T., Lane, K. D. & Yan, T. .-. Charmonium: The model. *Phys. Rev. D* **17**, 3090 (11 1978).

44. Born, K., Laermann, E., Sommer, R., Walsh, T. & Zerwas, P. The interquark potential: a QCD lattice analysis. *Physics Letters B* **329**, 325 (1994).

45. Kawanai, T. & Sasaki, S. Interquark Potential with Finite Quark Mass from Lattice QCD. *Phys. Rev. Lett.* **107**, 091601 (9 2011).

46. Gross, D. J. & Wilczek, F. Ultraviolet Behavior of Non-Abelian Gauge Theories. *Phys. Rev. Lett.* **30**, 1343 (26 1973).

47. Politzer, H. D. Reliable Perturbative Results for Strong Interactions? *Phys. Rev. Lett.* **30**, 1346 (26 1973).

48. Nambu, Y. Quasi-Particles and Gauge Invariance in the Theory of Superconductivity. *Phys. Rev.* **117**, 648 (3 1960).

49. Nambu, Y. & Jona-Lasinio, G. Dynamical Model of Elementary Particles Based on an Analogy with Superconductivity. I. *Phys. Rev.* **122**, 345 (1 1961).

50. Nambu, Y. Axial vector current conservation in weak interactions. *Phys. Rev. Lett.* **4** (ed Eguchi, T.) 380 (1960).

51. Goldstone, J. Field Theories with Superconductor Solutions. *Nuovo Cim.* **19**, 154 (1961).

52. Goldstone, J., Salam, A. & Weinberg, S. Broken Symmetries. *Phys. Rev.* **127**, 965 (3 1962).

53. Luscher, M. *Advanced lattice QCD* in *Les Houches Summer School in Theoretical Physics, Session 68: Probing the Standard Model of Particle Interactions* (1998), 229.

54. Aoyama, T. *et al.* The anomalous magnetic moment of the muon in the Standard Model. *Phys. Rept.* **887**, 1 (2020).

55. Aliberti, R. *et al.* The anomalous magnetic moment of the muon in the Standard Model: an update (2025).
56. Bennett, G. W., Bousquet, B., Brown, H. N., Bunce, G., Carey, R. M., Cushman, P., Danby, G. T., Debevec, P. T., Deile, M., Deng, H., Deninger, W., Dhawan, S. K., Druzhinin, V. P., Duong, L., Efstathiadis, E., Farley, F. J. M., Fedotovich, G. V., Giron, S., Gray, F. E., Grigoriev, D., Grosse-Perdekamp, M., Grossmann, A., Hare, M. F., Hertzog, D. W., Huang, X., Hughes, V. W., Iwasaki, M., Jungmann, K., Kawall, D., Kawamura, M., Khazin, B. I., Kindem, J., Krienen, F., Kronkvist, I., Lam, A., Larsen, R., Lee, Y. Y., Logashenko, I., McNabb, R., Meng, W., Mi, J., Miller, J. P., Mizumachi, Y., Morse, W. M., Nikas, D., Onderwater, C. J. G., Orlov, Y., Özben, C. S., Paley, J. M., Peng, Q., Polly, C. C., Pretz, J., Prigl, R., zu Putlitz, G., Qian, T., Redin, S. I., Rind, O., Roberts, B. L., Ryskulov, N., Sedykh, S., Semertzidis, Y. K., Shagin, P., Shatunov, Y. M., Sichtermann, E. P., Solodov, E., Sossong, M., Steinmetz, A., Sulak, L. R., Timmermans, C., Trofimov, A., Urner, D., von Walter, P., Warburton, D., Winn, D., Yamamoto, A. & Zimmerman, D. Final report of the E821 muon anomalous magnetic moment measurement at BNL. *Phys. Rev. D* **73**, 072003 (7 2006).
57. Muon $g-2$ Initiative. *The Muon $g-2$ Theory Initiative* `https://muon-gm2-theory.illinois.edu/`. (Online; accessed 2025-07-16). 2020.
58. Aguillard, D. P. *et al.* Measurement of the Positive Muon Anomalous Magnetic Moment to 127 ppb (2025).
59. Abe, M., Bae, S., Beer, G., Bunce, G., Choi, H., Choi, S., Chung, M., da Silva, W., Eidelman, S., Finger, M., Fukao, Y., Fukuyama, T., Haciomeroglu, S., Hasegawa, K., Hayasaka, K., Hayashizaki, N., Hisamatsu, H., Iijima, T., Iinuma, H., Ikeda, H., Ikeno, M., Inami, K., Ishida, K., Itahashi, T., Iwasaki, M., Iwashita, Y., Iwata, Y., Kadono, R., Kamal, S., Kamitani, T., Kanda, S., Kapusta, F., Kawagoe, K., Kawamura, N., Kim, B., Kim, Y., Kishishita, T., Kitamura, R., Ko, H., Kohriki, T., Kondo, Y., Kume, T., Lee, M. J., Lee, S., Lee, W., Marshall, G. M., Matsuda, Y., Mibe, T., Miyake, Y., Murakami, T., Nagamine, K., Nakayama, H., Nishimura, S., Nomura, D., Ogitsu, T., Ohsawa, S., Oide, K., Oishi, Y., Okada, S., Olin, A., Omarov, Z., Otani, M., Razuvaev, G., Rehman, A., Saito, N., Saito, N. F., Sasaki, K., Sasaki, O., Sato, N., Sato, Y., Semertzidis, Y. K., Sendai, H., Shatunov, Y., Shimomura, K., Shoji, M., Shwartz, B., Strasser, P., Sue, Y., Suehara, T., Sung, C., Suzuki, K., Takatomi, T., Tanaka,

M., Tojo, J., Tsutsumi, Y., Uchida, T., Ueno, K., Wada, S., Won, E., Yamaguchi, H., Yamanaka, T., Yamamoto, A., Yamazaki, T., Yasuda, H., Yoshida, M. & Yoshioka, T. A new approach for measuring the muon anomalous magnetic moment and electric dipole moment. *Progress of Theoretical and Experimental Physics* **2019**, 053C02 (2019).

60. Dirac, P. A. M. The quantum theory of the electron. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* **117**, 610 (1928).
61. Schwinger, J. On Quantum-Electrodynamics and the Magnetic Moment of the Electron. *Phys. Rev.* **73**, 416 (4 1948).
62. Laporta, S. & Remiddi, E. The analytical value of the electron (g- 2) at order $\alpha 3$ in QED. *Physics Letters B* **379**, 283 (1996).
63. Aoyama, T., Hayakawa, M., Kinoshita, T. & Nio, M. Tenth-Order QED Contribution to the Electron $g-2$ and an Improved Value of the Fine Structure Constant. *Phys. Rev. Lett.* **109**, 111807 (11 2012).
64. Aoyama, T., Hayakawa, M., Kinoshita, T. & Nio, M. Tenth-order electron anomalous magnetic moment: Contribution of diagrams without closed lepton loops. *Phys. Rev. D* **91**, 033006 (3 2015).
65. Nio, M. *QED tenth-order contribution to the electron anomalous magnetic moment and a new value of the fine-structure constant* in *PDF), Fundamental Constants Meeting* (2015).
66. Berestetskii, V. B., Krokhin, O. N. & Khlebnikov, A. K. CONCERNING THE RADIATIVE CORRECTION TO THE mu-MESON MAGNETIC MOMENT. *Soviet Phys. JETP* **Vol: 3** (1956).
67. Fael, M., Mercolli, L. & Passera, M. Towards a determination of the tau lepton dipole moments. *Nucl. Phys. B Proc. Suppl.* **253-255** (eds Hayasaka, K. & Iijima, T.) 103 (2014).
68. Lees, J. P. *et al.* Measurement of the branching fractions of the radiative leptonic τ decays $\tau \to e\gamma\nu\bar{\nu}$ and $\tau \to \mu\gamma\nu\bar{\nu}$ at BaBar. *Phys. Rev. D* **91**, 051103 (2015).
69. Fan, X., Myers, T. G., Sukra, B. A. D. & Gabrielse, G. Measurement of the Electron Magnetic Moment. *Phys. Rev. Lett.* **130**, 071801 (2023).
70. Jegerlehner, F. & Nyffeler, A. The muon g- 2. *Physics Reports* **477**, 1 (2009).
71. Mohr, P. J., Tiesinga, E., Newell, D. B. & Taylor, B. N. *Codata Internationally Recommended 2022 Values of the Fundamental Physical Constants* 2024.

72. Agostini, M. *et al.* Test of Electric Charge Conservation with Borexino. *Phys. Rev. Lett.* **115**, 231802 (23 2015).
73. Beringer, J. *et al. PDG Live Particle Summary Quarks (u, d, s, c, b, t, b′, t′, Free). Particle Data Group, PR D86, 010001* 2012.
74. Patrignani, C. Review of Particle Physics. *Chinese Physics C* **40**, 100001 (2016).
75. Tanabashi, M. *et al.* Review of Particle Physics. *Phys. Rev. D* **98**, 030001 (3 2018).
76. Bennett, G. W. *et al.* Final Report of the Muon E821 Anomalous Magnetic Moment Measurement at BNL. *Phys. Rev.* **D73**, 072003 (2006).
77. Abi, B. *et al.* Measurement of the Positive Muon Anomalous Magnetic Moment to 0.46 ppm. *Phys. Rev. Lett.* **126**, 141801 (2021).
78. Davier, M., Hoecker, A., Malaescu, B. & Zhang, Z. Reevaluation of the hadronic vacuum polarisation contributions to the Standard Model predictions of the muon $g-2$ and $\alpha(m_Z^2)$ using newest hadronic cross-section data. *Eur. Phys. J.* **C77**, 827 (2017).
79. Keshavarzi, A., Nomura, D. & Teubner, T. Muon $g-2$ and $\alpha(M_Z^2)$: a new data-based analysis. *Phys. Rev.* **D97**, 114025 (2018).
80. Colangelo, G., Hoferichter, M. & Stoffer, P. Two-pion contribution to hadronic vacuum polarization. *JHEP* **02**, 006 (2019).
81. Hoferichter, M., Hoid, B.-L. & Kubis, B. Three-pion contribution to hadronic vacuum polarization. *JHEP* **08**, 137 (2019).
82. Davier, M., Hoecker, A., Malaescu, B. & Zhang, Z. A new evaluation of the hadronic vacuum polarisation contributions to the muon anomalous magnetic moment and to $\boldsymbol{\alpha}(\mathbf{m}_\mathbf{Z}^\mathbf{2})$. *Eur. Phys. J.* **C80**. [Erratum: Eur. Phys. J. C80, 410 (2020)], 241 (2020).
83. Keshavarzi, A., Nomura, D. & Teubner, T. The $g-2$ of charged leptons, $\alpha(M_Z^2)$ and the hyperfine splitting of muonium. *Phys. Rev.* **D101**, 014029 (2020).
84. Bernecker, D. & Meyer, H. B. Vector Correlators in Lattice QCD: Methods and applications. *Eur. Phys. J. A* **47**, 148 (2011).

85. Blum, T. Lattice calculation of the lowest-order hadronic contribution to the muon anomalous magnetic moment. *Nucl. Phys. B Proc. Suppl.* **140** (eds Bodwin, G. T., Sinclair, D. K., Eichten, E., Holmgren, D., Kronfeld, A. S., Mackenzie, P., Okamoto, M., Simone, J. & El-Khadra, A. X.) 311 (2005).
86. Lehner, C. A precise determination of the HVP contribution to the muon anomalous magnetic moment from lattice QCD. *EPJ Web Conf.* **175** (eds Della Morte, M., Fritzsch, P., Gámiz Sánchez, E. & Pena Ruano, C.) 01024 (2018).
87. Blum, T., Boyle, P. A., Gülpers, V., Izubuchi, T., Jin, L., Jung, C., Jüttner, A., Lehner, C., Portelli, A. & Tsang, J. T. Calculation of the hadronic vacuum polarization contribution to the muon anomalous magnetic moment. *Phys. Rev. Lett.* **121**, 022003 (2018).
88. Blum, T. *et al.* The long-distance window of the hadronic vacuum polarization for the muon g-2 (2024).
89. Borsanyi, S. *et al.* Hadronic vacuum polarization contribution to the anomalous magnetic moments of leptons from first principles. *Phys. Rev. Lett.* **121**, 022002 (2018).
90. Djukanovic, D., von Hippel, G., Kuberski, S., Meyer, H. B., Miller, N., Ottnad, K., Parrino, J., Risch, A. & Wittig, H. The hadronic vacuum polarization contribution to the muon $g-2$ at long distances (2024).
91. Boccaletti, A. *et al.* High precision calculation of the hadronic vacuum polarisation contribution to the muon anomaly (2024).
92. Bazavov, A. *et al.* Hadronic vacuum polarization for the muon $g-2$ from lattice QCD: Complete short and intermediate windows (2024).
93. Alexandrou, C. *et al.* Lattice calculation of the short and intermediate time-distance hadronic vacuum polarization contributions to the muon magnetic moment using twisted-mass fermions. *Phys. Rev. D* **107**, 074506 (2023).
94. Bazavov, A. *et al.* Hadronic vacuum polarization for the muon $g-2$ from lattice QCD: Complete short and intermediate windows (2024).
95. De Divitiis, G. M. *et al.* Isospin breaking effects due to the up-down mass difference in Lattice QCD. *JHEP* **04**, 124 (2012).
96. De Divitiis, G. M., Frezzotti, R., Lubicz, V., Martinelli, G., Petronzio, R., Rossi, G. C., Sanfilippo, F., Simula, S. & Tantalo, N. Leading isospin breaking effects on the lattice. *Phys. Rev. D* **87**, 114505 (2013).

97. Lucini, B., Patella, A., Ramos, A. & Tantalo, N. Charged hadrons in local finite-volume QED+QCD with C* boundary conditions. *JHEP* **02**, 076 (2016).
98. Uno, S. & Hayakawa, M. QED in Finite Volume and Finite Size Scaling Effect on Electromagnetic Properties of Hadrons. *Progress of Theoretical Physics* **120**, 413 (2008).
99. Endres, M. G., Shindler, A., Tiburzi, B. C. & Walker-Loud, A. Massive Photons: An Infrared Regularization Scheme for Lattice QCD + QED. *Phys. Rev. Lett.* **117**, 072002 (7 2016).
100. Blum, T., Christ, N., Hayakawa, M., Izubuchi, T., Jin, L., Jung, C. & Lehner, C. Using infinite volume, continuum QED and lattice QCD for the hadronic light-by-light contribution to the muon anomalous magnetic moment. *Phys. Rev. D* **96**, 034515 (2017).
101. Feng, X. & Jin, L. QED self energies from lattice QCD without power-law finite-volume errors. *Phys. Rev. D* **100**, 094509 (2019).
102. Wiese, U.-J. C- and G-periodic QCD at finite temperature. *Nuclear Physics B* **375**, 45 (1992).
103. Polley, L. Boundaries for SU (3) cx U (1) el lattice gauge theory with a chemical potential. *Zeitschrift für Physik C Particles and Fields* **59**, 105 (1993).
104. Kronfeld, A. & Wiese, U.-J. SU(N) gauge theories with C-periodic boundary conditions (I). Topological structure. *Nuclear Physics B* **357**, 521 (1991).
105. Kronfeld, A. & Wiese, U.-J. SU suN) gauge theories with C-periodic boundary conditions (II). Small-volume dynamics. *Nuclear Physics B* **401**, 190 (1993).
106. Borsanyi, S. *et al.* Ab initio calculation of the neutron-proton mass difference. *Science* **347**, 1452 (2015).
107. Patella, A. QED Corrections to Hadronic Observables. *PoS* **LATTICE2016**, 020 (2017).
108. Duncan, A., Eichten, E. & Thacker, H. Electromagnetic splittings and light quark masses in lattice QCD. *Phys. Rev. Lett.* **76**, 3894 (1996).
109. Gockeler, M., Horsley, R., Laermann, E., Rakow, P. E. L., Schierholz, G., Sommer, R. & Wiese, U. J. QED: A Lattice Investigation of the Chiral Phase Transition and the Nature of the Continuum Limit. *Nucl. Phys. B* **334**, 527 (1990).

110. Asmussen, N., Green, J., Meyer, H. B. & Nyffeler, A. Position-space approach to hadronic light-by-light scattering in the muon $g - 2$ on the lattice. *PoS* **LATTICE2016**, 164 (2016).
111. Campos, I., Fritzsch, P., Hansen, M., Marinkovic, M. K., Patella, A., Ramos, A. & Tantalo, N. openQ*D code: a versatile tool for QCD+QED simulations. *The European Physical Journal C* **80**, 1 (2020).
112. Bushnaq, L., Campos, I., Catillo, M., Cotellucci, A., Dale, M., Fritzsch, P., Lücke, J., Krstić Marinković, M., Patella, A. & Tantalo, N. First results on QCD+QED with C* boundary conditions. *JHEP* **03**, 012 (2023).
113. Boyle, P. *et al.* Isospin-breaking corrections to light-meson leptonic decays from lattice simulations at physical quark masses. *JHEP* **02**, 242 (2023).
114. Carrasco, N., Lubicz, V., Martinelli, G., Sachrajda, C. T., Tantalo, N., Tarantino, C. & Testa, M. QED corrections to hadronic processes in lattice QCD. *Phys. Rev. D* **91**, 074506 (7 2015).
115. Tantalo, N. *Lattice calculation of isospin corrections to Kl2 and Kl3 decays* in *7th International Workshop on the CKM Unitarity Triangle* (2013).
116. Jia, Z., Tillman, B., Maggioni, M. & Scarpazza, D. P. Dissecting the graphcore ipu architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413* (2019).
117. ETH Zürich. *Detailed stipulations regarding the doctorate* 2022.
118. Marinkovic, M. K. *openQxD: Efficient QCD+QED Simulations with Various Boundary Conditions on GPUs* `https://pasc-ch.org/projects/2025-2028/openqxd-efficient-qcdqed-simulations-with-various-boundary-conditions-on-gpus/index.html`. 2025/2027.
119. RC* Collaboration. *openQxD-devel* `https://gitlab.com/rcstar/openQxD-devel`. (Online; accessed 2025-02-19). 2019–2025.
120. CLS. *Coordinated Lattice Simulations* `https://wiki-zeuthen.desy.de/CLS/`. (Online; accessed 2025-02-19). 2012–2023.
121. TOP500.org. *TOP500* `https://www.top500.org/lists/top500/2024/11/`. (Online; accessed 2025-03-26). 2025.
122. Khan, A., Sim, H., Vazhkudai, S. S., Butt, A. R. & Kim, Y. *An analysis of system balance and architectural trends based on top500 supercomputers* in *The International Conference on High Performance Computing in Asia-Pacific Region* (2021), 11.

123. Matsuoka, S., Aoki, T., Endo, T., Nukada, A., Kato, T. & Hasegawa, A. GPU accelerated computing–from hype to mainstream, the rebirth of vector computing. *Journal of Physics: Conference Series* **180**, 012043 (2009).
124. Navarro, C. A., Hitschfeld-Kahler, N. & Mateu, L. A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures. *Communications in Computational Physics* **15**, 285 (2014).
125. Markidis, S., Der Chien, S. W., Laure, E., Peng, I. B. & Vetter, J. S. *Nvidia tensor core programmability, performance & precision* in *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (2018), 522.
126. Martineau, M., Atkinson, P. & McIntosh-Smith, S. *Benchmarking the NVIDIA V100 GPU and Tensor Cores* in *Euro-Par 2018: Parallel Processing Workshops* (eds Mencagli, G., B. Heras, D., Cardellini, V., Casalicchio, E., Jeannot, E., Wolf, F., Salis, A., Schifanella, C., Manumachu, R. R., Ricci, L., Beccuti, M., Antonelli, L., Garcia Sanchez, J. D. & Scott, S. L.) (Springer International Publishing, Cham, 2019), 444.
127. Fusco, L., Khalilov, M., Chrapek, M., Chukkapalli, G., Schulthess, T. & Hoefler, T. Understanding data movement in tightly coupled heterogeneous systems: A case study with the Grace Hopper superchip. *arXiv preprint arXiv:2408.11556* (2024).
128. Novikov, A., Podoprikhin, D., Osokin, A. & Vetrov, D. P. Tensorizing neural networks. *Advances in neural information processing systems* **28** (2015).
129. Isaev, M., McDonald, N. & Vuduc, R. *Scaling infrastructure to support multi-trillion parameter LLM training* in *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)* (2023).
130. Krylov, A. On the numerical solution of equation by which are determined in technical problems the frequencies of small vibrations of material systems. *News Acad. Sci. USSR* **7**, 491 (1931).
131. Saad, Y. *Iterative methods for sparse linear systems* (SIAM, 2003).
132. RC* Collaboration. *openQxD* `https://gitlab.com/rcstar/openQxD`. (Online; accessed 2025-02-19). 2019–2025.

133. Lüscher, M. *et al. openQCD, Simulation programs for lattice QCD* `https://luscher.web.cern.ch/luscher/openQCD/`. (Online; accessed 2025-02-20). 2012–2023.

134. Dagum, L. & Menon, R. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering* **5**, 46 (1998).

135. OpenMP Architecture Review Board. *OpenMP Application Program Interface - Version 4.0* `https://www.openmp.org/`. (Online; accessed 2025-03-26). 2013.

136. Anzt, H., Kreutzer, M., Ponce, E., Peterson, G. D., Wellein, G. & Dongarra, J. Optimization and performance evaluation of the IDR iterative Krylov solver on GPUs. *The International Journal of High Performance Computing Applications* **32**, 220 (2018).

137. Anzt, H., Dongarra, J., Kreutzer, M., Wellein, G. & Köhler, M. *Efficiency of General Krylov Methods on GPUs – An Experimental Study* in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2016), 683.

138. Anzt, H., Sawyer, W., Tomov, S., Luszczek, P. & Dongarra, J. Acceleration of GPU-based Krylov solvers via Data Transfer Reduction. *International Journal of High Performance Computing Applications* (2015).

139. Yamazaki, I., Rajamanickam, S., Boman, E. G., Hoemmen, M., Heroux, M. A. & Tomov, S. *Domain Decomposition Preconditioners for Communication-Avoiding Krylov Methods on a Hybrid CPU/GPU Cluster* in *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC 14)* (IEEE, New Orleans, LA, 2014).

140. Tomov, S., Luszczek, P., Yamazaki, I., Dongarra, J., Anzt, H. & Sawyer, W. *Optimizing Krylov Subspace Solvers on Graphics Processing Units* in *Fourth International Workshop on Accelerators and Hybrid Exascale Systems (AsHES), IPDPS 2014* (IEEE, Phoenix, AZ, 2014).

141. Knibbe, H., Oosterlee, C. & Vuik, C. GPU implementation of a Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method. *Journal of Computational and Applied Mathematics* **236**. Aspects of Numerical Algorithms, Parallelization and Applications, 281 (2011).

142. Ganesan, S. & Shah, M. *SParSH-AMG: A library for hybrid CPU-GPU algebraic multigrid and preconditioned iterative methods* 2020.
143. Anzt, H., Gates, M., Dongarra, J., Kreutzer, M., Wellein, G. & Köhler, M. Preconditioned krylov solvers on gpus. *Parallel Computing* **68**, 32 (2017).
144. Clark, M. A., Babich, R., Barros, K., Brower, R. C. & Rebbi, C. Solving Lattice QCD systems of equations using mixed precision solvers on GPUs. *Computer Physics Communications* **181**, 1517 (2010).
145. Babich, R., Clark, M. A., Joo, B., Shi, G., Brower, R. C. & Gottlieb, S. *Scaling lattice QCD beyond 100 GPUs* in *International Conference for High Performance Computing, Networking, Storage and Analysis* (2011).
146. Clark, M. A., Joó, B., Strelchenko, A., Cheng, M., Gambhir, A. & Brower, R. C. *Accelerating lattice QCD multigrid on GPUs using fine-grained parallelization* in *International Conference for High Performance Computing, Networking, Storage and Analysis* (2016).
147. Edwards, R. G. & Joo, B. The Chroma software system for lattice QCD. *Nucl. Phys. B Proc. Suppl.* **140** (eds Bodwin, G. T., Sinclair, D. K., Eichten, E., Holmgren, D., Kronfeld, A. S., Mackenzie, P., Okamoto, M., Simone, J. & El-Khadra, A. X.) 832 (2005).
148. USQCD. *Chroma* `https://github.com/JeffersonLab/chroma`. (Online; accessed 2025-05-23). 2005–2025.
149. Jung, Chulwoo. *CPS* `https://usqcd-software.github.io/CPS.html`. (Online; accessed 2025-05-23). 2007.
150. Osborn, James. *SciDAC QFT Data Parallel library* `https://usqcd-software.github.io/qdp`. (Online; accessed 2025-05-23). 2007.
151. Rago, Antonio. *OpenQCD on GPU* `https://pos.sissa.it/466/283/`. (Online; accessed 2025-05-23). 2024.
152. Jansen, K. & Urbach, C. tmLQCD: A Program suite to simulate Wilson Twisted mass Lattice QCD. *Computer Physics Communications* **180**, 2717 (2009).
153. Kostrzewa, B., Bacchio, S., Finkenrath, J., Garofalo, M., Pittler, F., Romiti, S. & Urbach, C. Twisted mass ensemble generation on GPU machines. *PoS* **LATTICE2022**, 340 (2023).
154. Finkenrath, J. Review on Algorithms for dynamical fermions. *PoS* **LATTICE2022**, 227 (2023).

155. Garofalo, M., Kostrzewa, B., Romiti, S. & Sen, A. Autotuning multigrid parameters in the HMC on different architectures. *PoS* **LATTICE2024**, 276 (2025).
156. Boyle, P., Yamaguchi, A., Cossu, G. & Portelli, A. Grid: A next generation data parallel C++ QCD library (2015).
157. Boyle, Peter. *Grid* `https://github.com/paboyle/Grid`. (Online; accessed 2025-05-23). 2015–2025.
158. Richtmann, D., Boyle, P. A. & Wettig, T. Multigrid for Wilson Clover Fermions in Grid. *PoS* **LATTICE2018**, 032 (2019).
159. Yamaguchi, A., Boyle, P., Cossu, G., Filaci, G., Lehner, C. & Portelli, A. Grid: OneCode and FourAPIs. *PoS* **LATTICE2021**, 035 (2022).
160. Boyle, P. A. Multiple right hand side multigrid for domain wall fermions with a multigrid preconditioned block conjugate gradient algorithm (2024).
161. McClendon, C. *Optimized lattice qcd kernels for a pentium 4 cluster* tech. rep. (Thomas Jefferson National Accelerator Facility (TJNAF), Newport News, VA . . ., 2001).
162. NVIDIA Corporation. *QUDA* `https://github.com/milc-qcd/milc_qcd`. (Online; accessed 2025-05-23). 2005–2025.
163. Bazavov, Alexei and Bernard, Claude and Burch, Tom and DeGrand, Tom and DeTar, Carleton and Foley, Justin and Gottlieb, Steve and Heller, Urs and Hetrick, James and Levkova, Ludmila and McNeile, Craig and Orginos, Kostas and Osborn, James and Rummukainen, Kari and Sugar, Bob and Toussaint, Doug. *The MILC Code* `https://web.physics.utah.edu/~detar/milc/milcv7.pdf`. (Online; accessed 2025-05-23). 2016.
164. Lüscher, M. & Schaefer, S. Lattice QCD with open boundary conditions and twisted-mass reweighting. *Comput. Phys. Commun.* **184**, 519 (2013).
165. Sakurai, T., Tadano, H. & Kuramashi, Y. Application of block Krylov subspace algorithms to the Wilson-Dirac equation with multiple right-hand sides in lattice QCD. *Comput. Phys. Commun.* **181**, 113 (2010).
166. Nakamura, Y., Ishikawa, K. .-., Kuramashi, Y., Sakurai, T. & Tadano, H. Modified Block BiCGSTAB for Lattice QCD. *Comput. Phys. Commun.* **183**, 34 (2012).

167. Birk, S. & Frommer, A. A CG Method for Multiple Right Hand Sides and Multiple Shifts in Lattice QCD Calculations. *PoS* **LATTICE2011** (ed Vranas, P.) 027 (2011).
168. Clark, M. A., Strelchenko, A., Vaquero, A., Wagner, M. & Weinberg, E. Pushing Memory Bandwidth Limitations Through Efficient Implementations of Block-Krylov Space Solvers on GPUs. *Comput. Phys. Commun.* **233**, 29 (2018).
169. Richtmann, D., Heybrock, S. & Wettig, T. Multiple right-hand-side setup for the DD-αAMG. *PoS* **LATTICE2015**, 035 (2016).
170. Boyle, P. A. Hierarchically deflated conjugate gradient (2014).
171. Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 2.0* 1997.
172. Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 3.0* 2012.
173. CINECA Supercomputing Centre, SuperComputing Applications and Innovation Department. LEONARDO: A Pan-European Pre-Exascale Supercomputer for HPC and AI applications. *Journal of large-scale research facilities* **8**. A186 (2024).
174. The OpenSHMEM Team. *OpenSHMEM* `http://www.openshmem.org/`. (Online; accessed 2025-05-02). 2025.
175. NVIDIA. *NVSHMEM* `https://developer.nvidia.com/nvshmem`. (Online; accessed 2025-05-02). 2025.
176. Chaitin, G. J., Auslander, M. A., Chandra, A. K., Cocke, J., Hopkins, M. E. & Markstein, P. W. Register allocation via coloring. *Computer Languages* **6**, 47 (1981).
177. Denning, P. J. *Thrashing: its causes and prevention* in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I* (Association for Computing Machinery, San Francisco, California, 1968), 915.
178. Campos, I., Cardoso, N., Marinkovic, M. K., Della Morte, M. & Korcyl, P. *Efficient QCD+QED Simulations with openQ*D software* `https://pasc-ch.org/projects/2021-2024/eniac-enabling-the-icon-model-on-heterogeneous-architectures/index.html`. (Online; accessed 2025-02-19). 2021–2024.
179. NVIDIA Corporation. *QUDA* `https://github.com/lattice/quda`. (Online; accessed 2025-02-19). 2017–2025.

180. The Open MPI Project. *Open MPI: Open Source High Performance Computing* `https://www.open-mpi.org`. (Online; accessed 2025-02-20). 1998–2024.
181. The MPICH Team. *MPICH: A High-Performance and Widely Portable Implementation of the MPI Standard* `https://www.mpich.org`. (Online; accessed 2025-02-20). 1992–2025.
182. Gamblin, T., LeGendre, M., Collette, M. R., Lee, G. L., Moody, A., de Supinski, B. R. & Futral, S. *The Spack Package Manager: Bringing Order to HPC Software Chaos* in (Austin, Texas, US, 2015).
183. CSCS. *uenv: Command line tool for working with user environments at CSCS.* `https://eth-cscs.github.io/uenv/`. (Online; accessed 2025-02-20). 2025.
184. The Tcl Team. *TcL* `https://www.tcl-lang.org/`. (Online; accessed 2025-02-20). 2025.
185. The Lmod Team. *Lmod: A New Environment Module System* `https://github.com/TACC/Lmod`. (Online; accessed 2025-02-20). 2021.
186. Lüscher, M. Stochastic locality and master-field simulations of very large lattices. *EPJ Web Conf.* **175** (eds Della Morte, M., Fritzsch, P., Gámiz Sánchez, E. & Pena Ruano, C.) 01002 (2018).
187. Hutchinson, M. A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines. *Communications in Statistics - Simulation and Computation* **18**, 1059 (1989).
188. Foley, J., Jimmy Juge, K., O'Cais, A., Peardon, M., Ryan, S. M. & Skullerud, J.-I. Practical all-to-all propagators for lattice QCD. *Comput. Phys. Commun.* **172**, 145 (2005).
189. Bernardson, S., McCarty, P. & Thron, C. Monte Carlo methods for estimating linear combinations of inverse matrix entries in lattice QCD. *Comput. Phys. Commun.* **78**, 256 (1993).
190. Wilcox, W. *Noise methods for flavor singlet quantities* in *Interdisciplinary Workshop on Numerical Challenges in Lattice QCD* (1999), 127.
191. Wilcox, W. & Lindsay, B. Disconnected loop noise methods in lattice QCD. *Nucl. Phys. B Proc. Suppl.* **63** (eds Davies, C. T. H., Barbour, I. M., Bowler, K. C., Kenway, R. D., Pendleton, B. J. & Richards, D. G.) 973 (1998).
192. Alexandrou, C., Christaras, D., O Cais, A. & Strelchenko, A. Exact calculation of disconnected loops. *PoS* **LATTICE2010** (ed Rossi, G.) 035 (2010).

193. Güsken, S., Löw, U., Mütter, K.-H., Sommer, R., Patel, A. & Schilling, K. Non-singlet axial vector couplings of the baryon octet in lattice QCD. *Physics Letters B* **227**, 266 (1989).

194. Ó Cais, A., Juge, K. J., Peardon, M. J., Ryan, S. M. & Skullerud, J.-I. Improving algorithms to compute all elements of the lattice quark propagator. *Nucl. Phys. B Proc. Suppl.* **140** (eds Bodwin, G. T., Sinclair, D. K., Eichten, E., Holmgren, D., Kronfeld, A. S., Mackenzie, P., Okamoto, M., Simone, J. & El-Khadra, A. X.) 844 (2005).

195. Bali, G. S., Collins, S., Dürr, S. & Kanamori, I. $D_s \to \eta, \eta'$ semileptonic decay form factors with disconnected quark loop contributions. *Phys. Rev. D* **91**, 014503 (2015).

196. Morningstar, C. Exploring Excited Hadrons. *PoS* **LATTICE2008** (eds Aubin, C., Cohen, S., Dawson, C., Dudek, J., Edwards, R., Joo, B., Lin, H.-W., Orginos, K., Richards, D. & Thacker, H.) 009 (2008).

197. Bulava, J., Edwards, R., Juge, K. J., Morningstar, C. J. & Peardon, M. J. Multi-hadron operators with all-to-all quark propagators. *PoS* **LATTICE2008** (eds Aubin, C., Cohen, S., Dawson, C., Dudek, J., Edwards, R., Joo, B., Lin, H.-W., Orginos, K., Richards, D. & Thacker, H.) 100 (2008).

198. Foley, J., Wong, C. H., Bulava, J., Juge, K. J., Lenkner, D., Morningstar, C. & Peardon, M. A novel method for evaluating correlation functions in lattice hadron spectroscopy. *PoS* **Lattice2010** (ed Rossi, G.) 098 (2014).

199. Morningstar, C. *Baryon Spectroscopy from Lattice QCD* in *34th International Conference on High Energy Physics* (2008).

200. Babich, R., Brower, R. C., Clark, M. A., Fleming, G. T., Osborn, J. C., Rebbi, C. & Schaich, D. Exploring strange nucleon form factors on the lattice. *Phys. Rev. D* **85**, 054510 (2012).

201. Morningstar, C., Bulava, J., Foley, J., Juge, K. J., Lenkner, D., Peardon, M. & Wong, C. H. Improved stochastic estimation of quark propagation with Laplacian Heaviside smearing in lattice QCD. *Phys. Rev. D* **83**, 114505 (2011).

202. Bali, G. S., Collins, S. & Schafer, A. Effective noise reduction techniques for disconnected loops in Lattice QCD. *Comput. Phys. Commun.* **181**, 1570 (2010).

203. Parisi, G. Strategy for computing the hadronic mass spectrum. *Phys. Rep.;(Netherlands)* **103** (1984).

204. Lepage, G. Invited lectures given at TASI'89 Summer School. *Boulder, CO, Jun*, 4 (1989).

205. Neff, H., Eicker, N., Lippert, T., Negele, J. W. & Schilling, K. On the low fermionic eigenmode dominance in QCD on the lattice. *Phys. Rev. D* **64**, 114509 (2001).

206. DeGrand, T. A. & Schaefer, S. Improving meson two point functions in lattice QCD. *Comput. Phys. Commun.* **159**, 185 (2004).

207. Giusti, L., Hernandez, P., Laine, M., Weisz, P. & Wittig, H. Low-energy couplings of QCD from current correlators near the chiral limit. *JHEP* **04**, 013 (2004).

208. Spiegel, S. & Lehner, C. High-precision continuum limit study of the HVP short-distance window. *Phys. Rev. D* **111**, 114517 (2025).

209. Blum, T., Boyle, P. A., Bruno, M., Chakraborty, B., Erben, F., Gülpers, V., Hackl, A., Hermansson-Truedsson, N., Hill, R. C., Izubuchi, T., Jin, L., Jung, C., Lehner, C., McKeon, J., Meyer, A. S., Tomii, M., Tsang, J. T. & Tuo, X.-Y. Long-Distance Window of the Hadronic Vacuum Polarization for the Muon $g-2$. *Phys. Rev. Lett.* **134**, 201901 (20 2025).

210. Bazavov, A. *et al.* Hadronic Vacuum Polarization for the Muon g-2 from Lattice QCD: Long-Distance and Full Light-Quark Connected Contribution. *Phys. Rev. Lett.* **135**, 011901 (2025).

211. Blum, T., Boyle, P. A., Gülpers, V., Izubuchi, T., Jin, L., Jung, C., Jüttner, A., Lehner, C., Portelli, A. & Tsang, J. T. Calculation of the Hadronic Vacuum Polarization Contribution to the Muon Anomalous Magnetic Moment. *Phys. Rev. Lett.* **121**, 022003 (2 2018).

212. Borsanyi, S. *et al.* Leading hadronic contribution to the muon magnetic moment from lattice QCD. *Nature* **593**, 51 (2021).

213. Lehner, C. & Meyer, A. S. Consistency of hadronic vacuum polarization between lattice QCD and the R ratio. *Phys. Rev. D* **101**, 074515 (7 2020).

214. Wang, G., Draper, T., Liu, K.-F. & Yang, Y.-B. Muon g-2 with overlap valence fermions. *Phys. Rev. D* **107**, 034513 (3 2023).

215. Aubin, C., Blum, T., Golterman, M. & Peris, S. Muon anomalous magnetic moment with staggered fermions: Is the lattice spacing small enough? *Phys. Rev. D* **106**, 054503 (2022).

216. Blum, T., Boyle, P. A., Bruno, M., Giusti, D., Gülpers, V., Hill, R. C., Izubuchi, T., Jang, Y.-C., Jin, L., Jung, C., Jüttner, A., Kelly, C., Lehner, C., Matsumoto, N., Mawhinney, R. D., Meyer, A. S. & Tsang, J. T. Update of Euclidean windows of the hadronic vacuum polarization. *Phys. Rev. D* **108**, 054507 (5 2023).
217. Banks, T. & Casher, A. Chiral symmetry breaking in confining theories. *Nuclear Physics B* **169**, 103 (1980).
218. Lüscher, M. Local coherence and deflation of the low quark modes in lattice QCD. *JHEP* **07**, 081 (2007).
219. Tang, J. M. & Saad, Y. A probing method for computing the diagonal of a matrix inverse. *Numerical Linear Algebra with Applications* **19**, 485 (2012).
220. Stathopoulos, A., Laeuchli, J. & Orginos, K. Hierarchical Probing for Estimating the Trace of the Matrix Inverse on Toroidal Lattices. *SIAM J. Sci. Comput.* **35**, S299 (2013).
221. Giusti, L., Harris, T., Nada, A. & Schaefer, S. Frequency-splitting estimators of single-propagator traces. *Eur. Phys. J. C* **79**, 586 (2019).
222. Giusti, L., Harris, T., Nada, A. & Schaefer, S. Frequency-splitting estimators of single-propagator traces. *PoS* **LATTICE2019**, 157 (2020).
223. Whyte, T., Stathopoulos, A., Romero, E. & Orginos, K. Optimizing shift selection in multilevel Monte Carlo for disconnected diagrams in lattice QCD. *Comput. Phys. Commun.* **294**, 108928 (2024).
224. Boucaud, P. *et al.* Dynamical Twisted Mass Fermions with Light Quarks: Simulation and Analysis Details. *Comput. Phys. Commun.* **179**, 695 (2008).
225. McNeile, C. & Michael, C. Decay width of light quark hybrid meson from the lattice. *Phys. Rev. D* **73**, 074506 (7 2006).
226. Foster, M. & Michael, C. Quark mass dependence of hadron masses from lattice QCD. *Phys. Rev. D* **59**, 074503 (7 1999).
227. Giles, M. B. Multilevel monte carlo path simulation. *Operations research* **56**, 607 (2008).
228. Giles, M. B. Multilevel Monte Carlo methods. *Acta Numerica* **24**, 259 (2015).
229. Gambhir, A. S., Stathopoulos, A. & Orginos, K. Deflation as a Method of Variance Reduction for Estimating the Trace of a Matrix Inverse. *SIAM J. Sci. Comput.* **39**, A532 (2017).

230. Stathopoulos, A., Alcalde, E. R., Gambhir, A. S. & Orginos, K. *Deflation for Monte-Carlo estimation of the trace of a matrix inverse* The Tenth International Workshop on Lattice QFT and Numerical Analysis (QCDNA X). 2017.

231. Frommer, A., Khalil, M. N. & Ramirez-Hidalgo, G. A Multilevel Approach to Variance Reduction in the Stochastic Estimation of the Trace of a Matrix. *SIAM Journal on Scientific Computing* **44**, A2536 (2022).

232. Romero, E., Stathopoulos, A. & Orginos, K. Multigrid deflation for Lattice QCD. *J. Comput. Phys.* **409**, 109356 (2020).

233. Frommer, A. & Khalil, M. N. MGMLMC++ as a Variance Reduction Method for Estimating the Trace of a Matrix Inverse. *PoS* **LATTICE2022**, 017 (2023).

234. Meyer, R. A., Musco, C., Musco, C. & Woodruff, D. P. in *2021 Symposium on Simplicity in Algorithms (SOSA)* 142 ().

235. Frommer, A., Jimenez-Merchan, J., Ramirez-Hidalgo, G. & Urrea-Niño, J. A. Variance Reduction in Trace Estimation for Lattice QCD Using Multigrid Multilevel Monte Carlo. *PoS* **LATTICE2024**, 279 (2025).

236. Luscher, M. & Weisz, P. Locality and exponential error reduction in numerical lattice gauge theory. *JHEP* **09**, 010 (2001).

237. Meyer, H. B. Locality and statistical error reduction on correlation functions. *JHEP* **01**, 048 (2003).

238. Della Morte, M. & Giusti, L. Exploiting symmetries for exponential error reduction in path integral Monte Carlo. *Comput. Phys. Commun.* **180**, 813 (2009).

239. Della Morte, M. & Giusti, L. Symmetries and exponential error reduction in Yang-Mills theories on the lattice. *Comput. Phys. Commun.* **180**, 819 (2009).

240. Della Morte, M. & Giusti, L. A novel approach for computing glueball masses and matrix elements in Yang-Mills theories on the lattice. *Journal of High Energy Physics* **2011**, 1 (2011).

241. Cè, M., Giusti, L. & Schaefer, S. Domain decomposition, multi-level integration and exponential noise reduction in lattice QCD. *Phys. Rev. D* **93**, 094507 (2016).

242. Cè, M., Giusti, L. & Schaefer, S. A local factorization of the fermion determinant in lattice QCD. *Phys. Rev. D* **95**, 034503 (2017).

243. Giusti, L., Cè, M. & Schaefer, S. Multi-boson block factorization of fermions. *EPJ Web Conf.* **175** (eds Della Morte, M., Fritzsch, P., Gámiz Sánchez, E. & Pena Ruano, C.) 01003 (2018).

244. Cè, M., Giusti, L. & Schaefer, S. Local multiboson factorization of the quark determinant. *EPJ Web Conf.* **175** (eds Della Morte, M., Fritzsch, P., Gámiz Sánchez, E. & Pena Ruano, C.) 11005 (2018).

245. Giusti, L., Harris, T., Nada, A. & Schaefer, S. Multi-level integration for meson propagators. *PoS* **LATTICE2018**, 028 (2018).

246. Dalla Brida, M., Giusti, L., Harris, T. & Pepe, M. Multi-level Monte Carlo computation of the hadronic vacuum polarization contribution to $(g_\mu - 2)$. *Phys. Lett. B* **816**, 136191 (2021).

247. Giusti, L., Brida, M. D., Harris, T. & Pepe, M. Multi-level computation of the hadronic vacuum polarization contribution to $(g_\mu - 2)$. *PoS* **LATTICE2021**, 356 (2022).

248. Best, C., Göckeler, M., Horsley, R., Ilgenfritz, E.-M., Perlt, H., Rakow, P., Schäfer, A., Schierholz, G., Schiller, A. & Schramm, S. π and ρ structure functions from lattice QCD. *Phys. Rev. D* **56**, 2743 (5 1997).

249. Collins, S. *Gauge invariant smearing and the extraction of excited state masses using Wilson fermions at Beta = 6.2* in *The International Symposium on Lattice Field Theory: Lattice 92* (1992).

250. Allton, C. R. *et al.* Gauge invariant smearing and matrix correlators using Wilson fermions at Beta = 6.2. *Phys. Rev. D* **47**, 5128 (1993).

251. Güsken, S. A study of smearing techniques for hadron correlation functions. *Nuclear Physics B - Proceedings Supplements* **17**, 361 (1990).

252. Bali, G. S., Lang, B., Musch, B. U. & Schäfer, A. Novel quark smearing for hadrons with high momenta in lattice QCD. *Phys. Rev. D* **93**, 094515 (2016).

253. Albanese, M., Costantini, F., Fiorentini, G., Flore, F., Lombardo, M., Tripiccione, R., Bacilieri, P., Fonti, L., Giacomelli, P., Remiddi, E., Bernaschi, M., Cabibbo, N., Marinari, E., Parisi, G., Salina, G., Cabasino, S., Marzano, F., Paolucci, P., Petrarca, S., Rapuano, F., Marchesini, P. & Rusack, R. Glueball masses and string tension in lattice QCD. *Physics Letters B* **192**, 163 (1987).

254. Hasenfratz, A. & Knechtli, F. Flavor symmetry and the static potential with hypercubic blocking. *Phys. Rev. D* **64**, 034504 (2001).

255. Morningstar, C. & Peardon, M. Analytic smearing of $\mathrm{SU}(3)$ link variables in lattice QCD. *Phys. Rev. D* **69**, 054501 (5 2004).

256. Peardon, M., Bulava, J., Foley, J., Morningstar, C., Dudek, J., Edwards, R. G., Joo, B., Lin, H.-W., Richards, D. G. & Juge, K. J. A Novel quark-field creation operator construction for hadronic physics in lattice QCD. *Phys. Rev. D* **80**, 054506 (2009).

257. Knechtli, F., Korzec, T., Peardon, M. & Urrea-Niño, J. A. Optimizing creation operators for charmonium spectroscopy on the lattice. *Phys. Rev. D* **106**, 034501 (2022).

258. Bushnaq, L. N. *Exploring efficient methods for precision QCD calculations on the lattice* PhD thesis (Trinity Coll., Dublin, 2023).

259. Michael, C. & Peisa, J. Maximal variance reduction for stochastic propagators with applications to the static quark spectrum. *Phys. Rev. D* **58**, 034506 (1998).

260. Kuberski, S. Muon $g-2$: Lattice calculations of the hadronic vacuum polarization. *PoS* **LATTICE2023**, 125 (2024).

261. Bazavov, A. *et al.* Hadronic vacuum polarization for the muon $g-2$ from lattice QCD: Long-distance and full light-quark connected contribution (2024).

262. Lin, T., Bruno, M., Feng, X., Jin, L.-C., Lehner, C., Liu, C. & Luo, Q.-Y. Lattice QCD calculation of the π^0-pole contribution to the hadronic light-by-light scattering in the anomalous magnetic moment of the muon (2024).

263. Margari, F. *et al.* Smeared R-ratio in isospin symmetric QCD with Low Mode Averaging. *PoS* **LATTICE2024**, 446 (2025).

264. Yang, Y.-B., Alexandru, A., Draper, T., Gong, M. & Liu, K.-F. Stochastic method with low mode substitution for nucleon isovector matrix elements. *Phys. Rev. D* **93**, 034503 (2016).

265. Ohki, H., Takeda, K., Aoki, S., Hashimoto, S., Kaneko, T., Matsufuru, H., Noaki, J. & Onogi, T. Nucleon strange quark content from $N_f = 2+1$ lattice QCD with exact chiral symmetry. *Phys. Rev. D* **87**, 034509 (2013).

266. Bali, G., Castagnini, L. & Collins, S. Meson and baryon masses with low mode averaging. *PoS* **LATTICE2010** (ed Rossi, G.) 096 (2010).

267. Yamanaka, N., Hashimoto, S., Kaneko, T. & Ohki, H. Nucleon charges with dynamical overlap fermions. *Phys. Rev. D* **98**, 054516 (2018).

268. Kuberski, S. Low-mode deflation for twisted-mass and RHMC reweighting in lattice QCD. *Comput. Phys. Commun.* **300**, 109173 (2024).

269. Bernardoni, F., Garron, N., Hernandez, P., Necco, S. & Pena, C. Light quark correlators in a mixed-action setup. *PoS* **LATTICE2011** (ed Vranas, P.) 109 (2011).

270. Aoki, S. *et al.* Pion form factors from two-flavor lattice QCD with exact chiral symmetry. *Phys. Rev. D* **80**, 034508 (2009).

271. Lynch, M. & DeTar, C. *Contrasting low-mode noise reduction techniques for light HISQ meson propagators* in *The 39th International Symposium on Lattice Field Theory,* (2023), 252.

272. Bazavov, A. *et al.* Light-quark connected intermediate-window contributions to the muon g-2 hadronic vacuum polarization from lattice QCD. *Phys. Rev. D* **107**, 114514 (2023).

273. Blum, T., Izubuchi, T. & Shintani, E. New class of variance-reduction techniques using lattice symmetries. *Phys. Rev. D* **88**, 094503 (2013).

274. Shintani, E., Arthur, R., Blum, T., Izubuchi, T., Jung, C. & Lehner, C. Covariant approximation averaging. *Phys. Rev. D* **91**, 114511 (2015).

275. Blum, T., Boyle, P. A., Izubuchi, T., Jin, L., Jüttner, A., Lehner, C., Maltman, K., Marinkovic, M., Portelli, A. & Spraggs, M. Calculation of the hadronic vacuum polarization disconnected contribution to the muon anomalous magnetic moment. *Phys. Rev. Lett.* **116**, 232002 (2016).

276. Babich, R., Brannick, J., Brower, R. C., Clark, M. A., Manteuffel, T. A., McCormick, S. F., Osborn, J. C. & Rebbi, C. Adaptive multigrid algorithm for the lattice Wilson-Dirac operator. *Phys. Rev. Lett.* **105**, 201602 (2010).

277. Clark, M. A., Jung, C. & Lehner, C. Multi-Grid Lanczos. *EPJ Web Conf.* **175** (eds Della Morte, M., Fritzsch, P., Gámiz Sánchez, E. & Pena Ruano, C.) 14023 (2018).

278. Wright, S. J. *Numerical optimization* 2006.

279. Simoncini, V. & Szyld, D. B. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM Journal on Scientific Computing* **25**, 454 (2003).

280. Kaasschieter, E. F. Preconditioned conjugate gradients for solving singular systems. *Journal of Computational and Applied mathematics* **24**, 265 (1988).
281. Naumov, M. *S-step and communication-avoiding iterative methods* tech. rep. (Technical Report NVR-2016-003, NVIDIA, 2016).
282. Jansen, K. & Sommer, R. O(a) improvement of lattice QCD with two flavors of Wilson quarks. *Nucl. Phys. B* **530**. [Erratum: Nucl.Phys.B 643, 517–518 (2002)], 185 (1998).
283. Stathopoulos, A. & McCombs, J. R. PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description. *ACM Transactions on Mathematical Software* **37**, 21:1 (2010).
284. Efron, B. *The jackknife, the bootstrap and other resampling plans* (SIAM, 1982).
285. Shao, J. & Tu, D. *The jackknife and bootstrap* (Springer Science & Business Media, 2012).
286. Shewchuk, J. R. *et al.* An introduction to the conjugate gradient method without the agonizing pain (1994).
287. Gustafson, K. E., Rao, D. K., Gustafson, K. E. & Rao, D. K. *Numerical range* (Springer, 1997).
288. Toeplitz, O. Das algebraische Analogon zu einem Satze von Fejér. *Mathematische Zeitschrift* **2**, 187 (1918).
289. Hausdorff, F. Der wertvorrat einer bilinearform. *Mathematische Zeitschrift* **3**, 314 (1919).
290. Cohen, S. D., Brower, R. C., Clark, M. A. & Osborn, J. C. Multigrid Algorithms for Domain-Wall Fermions. *PoS* **LATTICE2011** (ed Vranas, P.) 030 (2011).
291. Francis, A., Fritzsch, P., Lüscher, M. & Rago, A. Master-field simulations of O(a)-improved lattice QCD: Algorithms, stability and exactness. *Comput. Phys. Commun.* **255**, 107355 (2020).
292. Johnson, C. R. Numerical determination of the field of values of a general complex matrix. *SIAM Journal on Numerical Analysis* **15**, 595 (1978).
293. Lehoucq, R. B. Implicitly Restarted Arnoldi Methods and Subspace Iteration. *SIAM Journal on Matrix Analysis and Applications* **23**, 551 (2001).

294. Hernandez, V., Roman, J. E. & Vidal, V. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software* **31**, 351 (2005).

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

296. Brower, R. C., Clark, M. A., Strelchenko, A. & Weinberg, E. Multigrid algorithm for staggered lattice fermions. *Phys. Rev. D* **97**, 114513 (2018).

297. Becher, P. & Joos, H. The Dirac-Kahler Equation and Fermions on the Lattice. *Z. Phys. C* **15**, 343 (1982).

298. Bodwin, G. T. & Kovacs, E. V. THE EQUIVALENCE OF DIRAC-KAHLER AND STAGGERED LATTICE FERMIONS IN TWO-DIMENSIONS. *Phys. Rev. D* **38**, 1206 (1988).

299. Ayyar, V., Brower, R. C., Clark, M. A., Wagner, M. & Weinberg, E. Optimizing Staggered Multigrid for Exascale performance. *PoS* **LATTICE2022**, 335 (2023).

300. Brower, R. C., Clark, M. A., Howarth, D. & Weinberg, E. S. Multigrid for chiral lattice fermions: Domain wall. *Phys. Rev. D* **102**, 094517 (2020).

301. Collard, M. K., Bardin, J., Laurin, M. & Ogier-Denis, E. The cecal appendix is correlated with greater maximal longevity in mammals. *Journal of Anatomy* **239**, 1157 (2021).

302. Gruber, Roman. *quda* `https://packages.spack.io/package.html?name=quda`. (Online; accessed 2025-02-19). 2025.

303. Gruber, Roman. *QUDA fork* `https://github.com/chaoos/quda`. (Online; accessed 2025-05-23). 2017–2025.

304. Gruber, R. *Dataset for PhD Thesis* `https://doi.org/10.5281/zenodo.15533835`. 2025.

LIST OF FIGURES

Figure 1.1 Illustration of the plaquette term (left) which is a matrix-matrix product of gauge links along the lines as in eq. (1.11) and the clover term (right) which is a sum of rotated plaquettes that resemble the form of a clover leaf. The individual plaquettes correspond to the summands in eqs. (1.20) to (1.23). 7

Figure 1.2 Variance contributions to a_μ^{SM}; the official Standard Model prediction to the anomalous magnetic moment of the muon $g-2$. The areas are proportional to the variances, whereas the side lengths are proportional to the errors. The contributions stand for: LO-HVP: lattice determination of the leading order HVP contribution (the one we deal with in this thesis), HlBl: mixed (phenomenology and lattice) result for the hadronic light-by-light contribution up to next-to-leading order (NLO), EW: electro-weak contribution, QED: quantum electrodynamics contribution up to tenth order, N(N)LO-HVP: phenomenological result of higher order terms of the HVP. Data for this plot was taken from ref. [55]. 18

Figure 1.3 Variance contributions to $a_\mu^{\text{LO-HVP}}$ (blue square in fig. 1.2); the official Standard Model prediction to the leading order hadronic vacuum polarization contribution to the anomalous magnetic moment of the muon $g-2$. The areas are proportional to the variances, whereas the side lengths are proportional to the errors. The contributions stand for: light: light quark connected contribution (the one we deal with in this thesis), disconnected: sum of all disconnected flavor contributions, strange/charm: contribution of the strange/charm quark flavor. Data for this plot was taken from ref. [55]. 19

Figure 1.4 Variance contribution if the windows to $a_\mu^{\text{LO-HVP}}$ (blue square in fig. 1.2). The areas are proportional to the variances, whereas the side lengths are proportional to the errors. The contributions stand for: LD: long-distance window, IB: strong and QED isospin-breaking corrections, W: intermediate window, SD: short-distance window. Data for this plot was taken from ref. [55]. 21

Figure 3.1 2D example (8×8 local lattice) of the rank-local unique lattice index in openQxD (in time-first convention (txyz)). The blue rectangles denote cache blocks of size 4×4. Gray sites are odd, white sites are even lattice points. 44

Figure 3.2 2D example of a 6×6 lattice with $C^\star$ boundary conditions on both directions. We have the (doubled) x-direction (horizontal) and a direction with $C^\star$ boundaries (vertical). Left is the physical, right the mirror lattice. Unfilled lattice points are exterior boundary points, whereas filled points are interior (boundary) points. The points of the same color are identified with each other. Notice that in x-direction we have regular periodic boundary conditions, since $C^\star$ boundaries are periodic over twice the lattice extent. The red, green and blue arrows indicate the path taken when leaving the physical lattice and entering the mirror lattice. 48

Figure 4.1 Call graph of the `gamma5` function in QUDA as an example of how kernels are abstracted from the backends. This function acts on fields already transferred to QUDA, thus there is no gamma basis transformation. 52

Figure 4.2 Copy process triggered by equating two fields of type `ColorSpinorField`. 53

Figure 5.1 Names of parameter structs, their members, enum constants defined for interfacing openQxD together with basis transformation and reorder classes in QUDA that were implemented to make the different memory layouts compatible. 60

Figure 5.2 2D example (4×4 local lattice) of how and which gauge fields are stored in memory in openQxD (left) and QUDA (right). Filled lattice points have even, unfilled odd parity. The red filled lattice point denotes the origin. Arrows represent gauge fields and the arrow head points to the lattice point where we store the field. 64

Figure 5.3 Call graph for the function `comm_rank_displaced`. Starting from the initialization function `quda_init` in openQxD, see section 5.2.2 which calls the interface initialization functions, see section 5.1.1, calling the communication grid setup which itself calls the constructor of the `Communicator` class which finally sets the grid topology. 66

Figure 5.4 Dependency graph of the various parameters and fields kept track by the interface. 71

Figure 5.5 Call graph of the `openQCD_qudaSolverGetHandle` function. Solid lines imply the function will always call the function the arrow points to, dashed lines will call only if necessary. The graph divides into namespaces in openQxD (top) to obtain latest parameters and revision counter, the interface (center) and QUDA (bottom). 71
Figure 5.6 Thread spawn and join concept for the asynchronous solver interface. All MPI ranks spawn their individual worker thread which interacts with QUDA and communicates with a communicator setup to contain all worker threads. The main threads return immediately back to the CPU code and communicate using their dedicated communicator. 75
Figure 5.7 Example of a global 8×8 lattice in 2D hosted by two different process grids. Left a 4×4 process grid with $N_w = 16$ ranks having local lattices of size 2×2 (a process block grid of 1×1) and right a 2×2 process grid with $N_q = 4$ ranks with local lattices of size 4×4. 82
Figure 5.8 The same two process grids as in fig. 5.7, but this time the openQxD (native) process grid has a non-trivial process block grid of extents 2×2. As opposed to fig. 5.7 this is the proper choice for this process block grid to optimize the communication pattern of the inter-grid operator. In both figures the bold written world rank numbers will be the subset of QUDA ranks, i. e. ranks with world rank numbers 0, 4, 8 and 12 (multiples of $N_w/N_q = 4$) on the left. They will become ranks with QUDA rank numbers 0, 1, 2 and 3, respectively, on the right. 85
Figure 6.1 Serial timeline profile of a program where solves are offloaded to QUDA on the GPU, whereas contractions are computed within openQxD on the CPU. 92
Figure 6.2 A timeline profile of a program issuing asynchronous calls to the solver, where after a warm-up phase an overlapping pattern of GPU and CPU work arises. 92
Figure 7.1 Number of native ranks per node versus application time of the inter-grid operator moving data between the native and dual process grid, keeping the number of dual ranks and nodes fixed. The data was taken on *Daint-Alps*, see section 7.1. 100
Figure 7.2 Strong scaling of inter-grid operator. Number of nodes versus application time of the inter-grid operator, keeping the number of native and dual ranks per node fixed. The data was taken on *Daint-Alps*, see section 7.1. 101

Figure 7.3 Contribution of data movement kernels to the solver. The plot shows different lattices and setups specified on the x-axis. Blue and yellow indicate the inter-grid data movement operator between the native and dual process grids, whereas green and red indicate the CPU-GPU data transfer and reordering procedure. Finally purple shows the contribution of the actual Krylov solver on the GPU. The contribution of the inter-grid operator is negligible in all cases as can be seen in the inset. The data was taken on *Daint-Alps*, see section 7.1. 102
Figure 7.4 Contributions from different kernels a) to f) on the CPU to a Krylov solve on the GPU. Blue, yellow and green show the serial pipeline (see fig. 6.1 and listing 6.4), whereas red, purple and brown show the overlapping pipeline (see fig. 6.2 and listing 6.5) averaged over 10 solves. The data was taken on *Leonardo Booster*, see section 7.1. 105
Figure 7.5 Absolute speedup of the heterogeneous versus the serial execution model averaged over 10 solves. A speedup of 2 is the analytical maximum. The effect of overlapping for the GPU solve as well as the individual kernels are indicated by yellow and green, whereas the blue shows the net speedup when overlapping. The data was taken on *Leonardo Booster*, see section 7.1. 106
Figure 7.6 Contributions from different kernels a) to f) on the CPU to a Krylov solve on the GPU. Blue, yellow and green show the serial pipeline (see fig. 6.1 and listing 6.4), whereas red, purple and brown show the overlapping pipeline (see fig. 6.2 and listing 6.5) averaged over 10 solves. The data was taken on *Daint-Alps*, see section 7.1. 107
Figure 7.7 Number of right-hand sides versus speedup compared to a single right-hand side. For small problems, a twofold speedup is reached. The data was taken on *Daint-Alps*, see section 7.1. 107
Figure 7.8 Number of right-hand sides versus energy consumption per RHS per node (blue) and versus power drawn from the power module per node (yellow) of the A400 lattice running on 2 nodes. Increasing the number of RHSs saturates the nodes. The data was taken on *Daint-Alps*, see section 7.1. 108

Figure 7.9 Strong scaling of Dirac stencil (left panel) and average time for one application (right panel). The operator was applied 10 000 times and the average was taken. The base case for the $C^\star$ lattice is two nodes for implementation restrictions. The physical global lattice size was $T \times L^3 = 128 \times 64^3$ for all runs. Every data point consists of at least 5 independent runs. The data was taken on *Daint-Alps*, see section 7.1. 109
Figure 7.10 Weak scaling of Dirac stencil (left panel) and average time for one application (right panel). The operator was applied 10 000 times and the average was taken. The base case for the $C^\star$ lattice is two nodes for implementation restrictions. The physical local lattice size was $64 \times 32 \times 64 \times 64$ and $128 \times 32 \times 64 \times 32$ for the periodic and $C^\star$ lattice, respectively. Every data point consists of at least 5 independent runs. The data was taken on *Daint-Alps*, see section 7.1. 110
Figure 7.11 Strong scaling of a multigrid solver (left panel) and average time to solution for one solve (right panel). The base cases were chosen to be the smallest node count for the computation to fit in GPU memory. The data was taken on *Daint-Alps*, see section 7.1. 111
Figure 7.12 Variant of weak scaling of a multigrid solver w.r.t. number of RHSs (left panel) and average time to solution for all RHSs (right panel). The base cases were chosen to be the smallest node count for the computation to fit in GPU memory. The data was taken on *Daint-Alps*, see section 7.1. **TODO**: remove red from plot 113
Figure 8.1 The comment to add to a pull-request that starts the CSCS default pipeline. 116
Figure 8.2 The job in the external stage that represents the CSCS default pipeline. One can click on the job to be redirected to CSCS servers where the whole pipeline including status of the jobs is visible. 116
Figure 9.1 Each field in openQxD corresponds to a field in QUDA. 128
Figure 9.2 Current field allocation scheme. 128
Figure 9.3 Proposed field allocation scheme. 128
Figure 9.4 A spinor field in memory. 131
Figure 9.5 A protected spinor field in memory visualized with a blue shield. 132
Figure 9.6 Accessing data at memory address `0xB`. 132

Figure 9.7	The state diagram of a spinor field as explained in the main text. There are three main states; `IN_SYNC` indicating that the field values are equal on CPU and GPU, `CPU_NEWER` and `GPU_NEWER` indicating that the CPU/GPU has changed the field and the other one is outdated respectively. The key indicated as *pointer* denotes the address of the field on the GPU initialized to `nullptr`. The protection states whether the memory on the CPU is protected according to section 9.3.1. The arrows denote triggers that cause actions to change the state of the field. There are three different triggers: read-only (*r*), write-only (*w*) and read-and-write (*rw*) on two devices: CPU and GPU. For instance an arrow with text *GPU: w/rw* indicates that the state change is triggered if the *GPU* either *writes* to the field or *reads and writes* to the field. 134
Figure 14.1	Depiction of the fact that the prolongator and restrictor are non-square matrices and coarse operator are restrictions of their fine-grid versions by dimensional reduction. 163
Figure 15.1	Quantitative picture of low modes in the free theory. Panel a) shows the sharp peak in momentum space centered around zero, whereas panel b) shows the same mode in position space on a finite lattice. Panel c) shows the approximation when using constant modes projected to blocks. The fact that the low modes are flat in position space makes the approximation suitable. 166
Figure 15.2	Local coherence of the low modes of Q tested on ensemble F7 of global lattice size $(L_0, L_1, L_2, L_3) = (96, 48, 48, 48)$, see table 17.1. The figure shows the value of $1 - \epsilon_i = \|P\xi_i\|$ on the y-axis versus the eigenmode number i on the x-axis, where ξ_i is the i-th normalized lowest mode for two different values of N_c and block sizes. This plot is taken from publication [P1]. 168
Figure 16.1	1D example ($\hat{V} = 8$) of the structure of the Dirac operator D and its coarse-grid representation $\hat{D}$ using periodic boundary conditions. Each block is of size $N_c N_s \times N_c N_s$. Nearest neighbor interactions of D make neighboring blocks of $\hat{D}$ occupied. Notice that in 4D every block has 8 neighbors. The matrix has $(2d+1)\hat{V} N_c^2 N_s^2$ non-zero entries, where d is the dimensionality of spacetime. This plot is taken from publication [P1]. 176

Figure 16.2 The effect of post-smoothing. The prolongation of the solution vector after a coarse-grid solve might look similar to the blue line in the right panel. Post-smoothing has the effect of filing off the edges of the staircase-shaped prolonged solution as depicted by the yellow line in the left panel. The prolonged error correction is then closer to the actual solution on the finer level (dashed gray line in both panels) and thus of higher quality. 178

Figure 16.3 Illustration that the coarse grid Dirac stencil transports information further through the lattice. Upper left: Dirac stencil on a 6×6 lattice connecting nearest neighbors. Instead we could restrict the vector to the coarse 3×3 grid, apply the coarse Dirac stencil (bottom center) and prolong back. The result is a longer range interaction (upper right). The information transfer is restricted to the coarse degrees of freedom, i. e. the low modes of the fine grid which in turn are associated to long-range interactions. Therefore, not *all* information has propagated the longer distance but the most relevant information has. 179

Figure 16.4 2D illustration of a recursive lattice coarsening. The finest lattice on level 0 in orange has a size of $V^{(0)} = 64$. The block size for level 1 in green is $b_0 \times b_1 = 2 \times 2$ resulting in a coarse lattice size of $V^{(1)} = 16$. The coarsest lattice in purple has a trivial lattice size of $V^{(3)} = 1$ and thus corresponds to LMA, where no blocking is done. The inter-grid operators $R^{(\ell)}$ and $T^{(\ell)}$ move data on adjacent levels, whereas the compound inter-grid operators $\mathcal{R}^{(\ell)}$ and $\mathcal{T}^{(\ell)}$ move data between levels 0 and ℓ. This plot is taken from publication [P1]. 181

Figure 16.5 (a) Grouping into contributions of multigrid levels of a 1-point correlator. (b) Grouping into contribution on multigrid levels of a 2-point connected correlator into 8 multigrid levels. Elements of the correlator tensor C_{ij}, eq. (16.30) are assigned to grid levels Lk. The sum of the orange shaded elements correspond to the L0-contribution of the correlator, green to L1, red to L2 etc. Summing all tensor elements is equal to summing all levels and results in the full correlator. (c) Example of a 3-point connected correlator tensor grouping into 4 multigrid levels. This plot is an extended version of a plot taken from publication [P1]. 185

Figure 17.1 Verification of the limit eq. (17.4). The x-axis denotes the number of stochastic sources and the y-axis its corresponding variance. The black dashed line with grey error bands represents $\sigma_{\mathcal{G}}(t)$ calculated using eq. (17.3) and the blue solid line is the limit of $\sigma_G(t)$ as defined in eq. (17.1). This plot is taken from publication [P1]. 195

Figure 17.2 Relative variance eq. (17.5) of one stochastic source of the two levels of LMA (left) and a corresponding 2-level MG LMA (right). The four groups of panels correspond to the four ensembles from table 17.1. In all panels the purple and blue shaded regions denote fine-grid and coarse-grid contributions respectively. The solid black line shows the variance of the sum. 196

Figure 17.3 E7: Absolute total variance eq. (17.1) of a single stochastic source. The left panel shows data for the LMA estimator, center and right show 2-level multigrid and a 3-level multigrid LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison. This plot is an extended version of a plot taken from publication [P1]. 198

Figure 17.4 F7: Absolute total variance eq. (17.1) of a single stochastic source. The left panel shows data for the LMA estimator, center and right show 2-level multigrid and a 3-level multigrid LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison. This plot is an extended version of a plot taken from publication [P1]. 199

Figure 17.5 G7: Absolute total variance eq. (17.1) of a single stochastic source. The left panel shows data for the LMA estimator, center and right show 2-level multigrid and a 4-level multigrid LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison. This plot is an extended version of a plot taken from publication [P1]. 199

Figure 17.6 H7: Absolute total variance eq. (17.1) of a single stochastic source. The left panel shows data for the LMA estimator, center and right show 2-level multigrid and a 4-level multigrid LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison. 200
Figure 17.7 Absolute total variance eq. (17.1) vs. the number of stochastic sources for lattices E7 (top), F7 (center) and G7 (bottom). The left panels show data for the LMA estimator, center the simple 2-level MG LMA estimator and right a 3-level (E7, F7) or 4-level (G7) MG LMA estimator. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in all panels for comparison. This plot is an extended version of a plot taken from publication [P1]. 201
Figure 17.8 Absolute total variance eq. (17.1) vs. lattice extent L. The left panel shows data for the LMA estimator, whereas the right panel shows a simple 2-level multigrid LMA with equal block size on all lattices, see table 17.2. The blue solid line represents the pure stochastic estimator without any deflation and the black dashed line indicates the gauge variance of the full correlator. Both appear in both panels for comparison. This plot is taken from publication [P1]. 203
Figure 17.9 (a) Estimated condition number and average number of iteration steps for a Biconjugate gradient stabilized solve (BiCGSTAB) vs. the dimension of the operator on G7. This plot is taken from publication [P1]. (b) Time for one application of a fine- or coarse-grid Dirac operator vs. their memory footprints on G7. 208
Figure 18.1 Boundary estimates of the numerical ranges (a) and spectral hulls (b) of the Wilson and Wilson-Clover Dirac operators. 225

Figure 18.2 Numerical range boundary estimates of coarse and fine Wilson-Clover Dirac operators. All subspaces were generated with $N_c = 20$ low modes of $\Gamma^5 D$. The panels correspond to spin projectors in table 18.1. Every panel shows different coarsening block sizes, where "LMA" corresponds to low-mode averaging, "fine" to the fine-grid operator and the MG aggregate block sizes are indicted in the labels. 226
Figure 18.3 Spectral hull boundary estimates of coarse and fine Wilson-Clover Dirac operators. This plots series corresponds to fig. 18.2. Note that the inset x-axis is log-scale, meaning that the convex hull might not look convex everywhere, because the logarithm is a non-linear transformation. 228
Figure 18.4 Study of low lying spectra of coarse and fine Hermitian Dirac operators. Lower left: eigenvalue magnitude versus mode number of the 100 lowest modes, lower right: eigenvalue magnitudes between fine and different coarse operators for comparison, and top panel: zoom of lower right of the interesting region. The three operators considered are fine in blue, coarse case (1) in yellow and coarse case (2sc) in green, c.f. table 18.1. Both coarse operators were constructed using a 6^4 block size and $N_c = 20$ low modes of Q. This plot is taken from publication [P1]. 230
Figure 18.5 Variance contribution of the L0-term if all spin degrees of freedom are coarsened case (1) in yellow as compared to strong chirality preservation case (2sc) in green. Variance contributions of both LMA (left) and MG LMA (right) profit from these considerations, although for LMA, the benefit is more subtle. This plot is taken from publication [P1]. 232
Figure C.1 Numerical range boundary using different number of probing angles N_θ. The plots show a clear resolution refinement from left to right. 257

LIST OF TABLES

Table 1.1 Lepton masses and their mean lifetimes. 16

Table 1.2 Contributions to a_μ^{SM}. Data in this table was taken from ref. [55]. 17

Table 7.1 Ensembles used for the performance tests in this chapter. G8 and F7 has been generated by the CLS initiative [120], while A360, A400, C380 and D300 were generated by the RC* collaboration [112]. The last column refers to spatial boundary conditions and α is the electromagnetic coupling. Ensembles with $\alpha \neq 0$ are dynamic QCD+QED simulations. 98

Table 7.2 Example of NUMA process placement to core IDs with periodic boundaries. With 32 tasks per node, we divide them into 8 tasks per NUMA domain associated to one CPU, filling only 8 from the available 72 cores. The dual MPI rank is calculated using eq. (5.2) and should be in the same NUMA domain. The GPUs are in NUMA domains 4,12,20 and 28 but also affine to 0, 1, 2 and 3, respectively. 103

Table 7.3 Example of NUMA process placement to core IDs with 32 tasks per node and C* boundaries. For detail see table 7.2. 104

Table 8.1 Ensembles used in the CI/CD pipeline. The periodic lattice D5d is taken from ref. [120], whereas the C* lattice is taken from ref. [112]. 118

Table 16.1 Example of the recursive and relaxed formulation of multigrid. The block size is denoted with respect to the fine grid. The relaxed formulation allows more choice in coarse spacetime volumes. 183

Table 17.1 Ensembles used in the variance reduction study. All lattices have a pion mass $m_\pi = 270\,\mathrm{MeV}$ and a lattice spacing of $a = 0.0658(10)$ fm with $N_f = 2$ $\mathcal{O}(a)$-improved Wilson fermions with lattice coupling $\beta = 5.3$, hopping parameter $\kappa = 0.13638$ and $c_{\mathrm{sw}} = 1.90952$ [120, 282]. The largest ensemble H7 is only used to compute some variances which can be accurately determined from a few measurements. F7 has been generated by the CLS initiative [120], while the others were generated by Tim Harris. 192

Table 17.2 Details about all the estimators including low modes from figs. 17.7a to 17.7c. For the different ensembles the table renders the number of multigrid levels, the block sizes used to generate the coarse lattices on each level. When the block size matches the lattice volume, as is the case in the LMA schemes, no blocking is performed on that level. All estimators use the same $N_c = 50$ low modes. 194

Table 17.3 Cost comparison to reach gauge noise of the different estimators used in this document. The unit of cost is number of fine-grid inversions. For trivial lattices on the coarsest grid the contribution was calculated exactly and its associated cost is zero. The actual cost is estimated by direct measurement table 17.4 and modeled using a performance model discussed in section 17.8.2. We did not add the cost of generating $N_c = 50$ low modes for multiple reasons. If the number is as small as 50, one can safely store them to disk and reuse them for different observables. Their cost is amortized quickly. As stated multiple times in the text, the LMA estimator requires significantly more modes on the larger lattices to be able to compete with the MG LMA estimator, resulting in a cost increase. The goal of this document is to compare the two estimators using *equal* resources in terms of low modes and show the volume dependence of both of them when they are held constant. However, if one insists to add the cost, one can add 100 to 200 to the numbers above. This amounts to a cost of 2 to 4 inversions per low mode. 204

Table 17.4 Average cost in terms of core-hours for an inversion on the fine and coarse grids. The block size is indicated in the fourth column, empty corresponds to the fine lattice. The time to solution (TTS) is reported in core-hours and Φ is the ratio of the TTS of a fine grid solve divided by a coarse grid solve. Φ is used as input for the measured cost calculation using eq. (17.6) in table 17.3. The last column reports the average number of iterations required for the solver to converge. It emphasizes the unequal solvers used on the different grids. We note that one should take care when comparing timings from one lattice to another, because they were taken on different machines with different node setups. 207

Table 17.5 Ratio Φ for the modeled cost of the different MG LMA estimators used in this document. 212

Table 18.1 Projectors and the form of coarse chiral and non-chiral operators. Coarse spin indices are inherited from the spin projectors (second column). The table uses the definition of the spin projectors $(P_\sigma)_{[\alpha\beta]} = \delta_{\alpha\beta}\delta_{\alpha\sigma}$ and $P_{\sigma,\rho} = \frac{1}{2}(P_\sigma + P_\rho)$ for $\sigma, \rho = 0, 1, 2, 3$. The matrix B is some irrelevant block-diagonal change-of-basis matrix. The IDs stand for Nxx, where N denotes the number of coarse spin degrees of freedom $\hat{N}_s$ and the letters xx stand for sc: strongly chiral, nc: non-chiral, wc: weakly chiral and sv: singular vectors. 217

Table 18.2 Extremal singular values $\sigma_{min,max}(A)$ and condition numbers $\kappa(A)$ for some coarse and fine, Hermitian and non-Hermitian Dirac operators. D_{WC} indicates the Wilson-Clover Dirac operator and Q is the Hermitian one $Q = \Gamma^5 D$. For all coarsenings the $N_c = 20$ lowest modes of Q_{WC} were taken. Gray quantities indicate smallest singular values smaller than the fine grid one. Associated operators are numerically problematic. 229

Table D.1 Asymmetry scores eqs. (D.1) and (D.2) for numerical ranges (third column) and spectra (fourth column). For gray quantities, we expect 0 from the analytical studies above. The numerical range eigenvalues were determined up to 10^{-6} and the spectral hull eigenvalues only up to 10^{-1}. We see good agreement with theory up to numerical precision. 260

PUBLICATIONS

Articles in peer-reviewed journals:

P1. Gruber, R., Harris, T. & Krstic Marinkovic, M. Multigrid low-mode averaging. *Phys. Rev. D* **111**, 074508 (2025).

P2. Altherr, A. *et al.* Comparing QCD+QED via full simulation versus the RM123 method: U-spin window contribution to a_μ^{HVP} (2025).

P3. Gruber, R., Kozhevnikov, A., Marinkovic, M., Schulthess, T. & Solca, R. Towards Lattice QCD+QED Simulations on GPUs. *PASC '23* (2023).

P4. Altherr, A., Coles, J., Antonio Fernández De la Garza, J., Gruber, R., Harris, T., Maier, S., Krstić Marinković, M., Parato, L. & Vogt, H. O(a)-improved QCD+QED Wilson Dirac operator on GPUs. unpublished (N.D.).

Conference contributions:

P5. Gruber, R., Kozhevnikov, A., Marinković, M. K., Schulthess, T. C. & Solcà, R. *Performance optimizations for porting the openQ*D package to GPUs* in. **LATTICE2021** (2022), 491.

P6. Gruber, R., Harris, T. & Krstić Marinković, M. *Variance reduction via deflation with local coherence* in (2024).

P7. Altherr, A. *et al. Strange and charm contributions to the HVP from C* boundary conditions* in. **LATTICE2022** (2023), 281.

P8. Altherr, A. *et al. Hadronic vacuum polarization with C* boundary conditions* in. **LATTICE2022** (2023), 312.

P9. Altherr, A. *et al. Tuning of QCD+QED simulations with C⋆ boundary conditions* in. **LATTICE2022** (2023), 259.

P10. Lücke, J. *et al.* $N_f = 1 + 2 + 1$ *QCD+QED simulations with C⋆ boundary conditions* in. **LATTICE2022** (2023), 064.

P11. Altherr, A. *et al.* $O(a)$*-improved QCD+QED Wilson Dirac operator on GPUs* in *41st International Symposium on Lattice Field Theory* (2025).

NUMBER OF TODOs: 34