Doctoral Course in
Physics
Cycle XXXIV

PhD Coordinator: Prof.ssa Eleonora Luppi

Multilevel Algorithms in Lattice QCD for Exascale Machines

Scientific/Disciplinary (SDS) FIS/02

Candidate:
Gustavo Alonso Ramírez Hidalgo

Supervisor:
Prof. Dr. Andreas Frommer

Supervisor:
Prof. Dr. Constantia Alexandrou

Supervisor:
Prof. Dr. Raffaele Tripiccione

Co-Supervisor:
Prof. Dr. Sebastiano Fabio Schifano

Years 2018-2022

Acknowledgments

This PhD would not have been successfully completed without the contribution, in different ways, of many people.

I would like to thank first Fabio Schifano, Lele Tripiccione, Dina Alexandrou and Andreas Frommer, my (co-)supervisors. To Lele, who always had a great disposition whenever we had a question of any kind, and who is unfortunately not with us anymore, and to Fabio, for stepping in afterwards. To Dina, for her continuous help througout the PhD, and her advice, in particular physics-wise; thank you also for playing such a great role in making STIMULATE possible. And to Andreas, for welcoming me to his group, for creating such a nice working environment in it, for great advice throughout the different research projects, and for always being in good spirits, be it during coffee breaks or our (interesting and engaging) meetings. I'm also grateful to many of my colleagues, for many interesting conversations, nice coffee breaks, (recently) fun bouldering sessions, and the ocassional technical/conceptual help; in particular, thanks to Matthias Rottmann and Artur Strebel who were of great help especially at the beginning of my PhD. I would also like to thank many close friends: for providing good advice, great conversations, and a nice and relaxing time overall; this has been fundamental in distracting me from the (sometimes) absorbing life that a PhD can represent.

And last but not least, I would like to thank my closest family. A papi y mami: todavía no entiendo cómo lograron salir adelante ante semejantes adversidades, pero sin ese esfuerzo y el ejemplo de trabajo duro, nada de esto hubiera sido posible. Gracias, también, por enseñarme a pensar antes de actuar, así como apreciar la no-obviedad de las cosas. To my siblings, Arleth, Maricela, Kike and Fabricio: for teaching me so many lessons, sometimes without you even realizing. And of course, to Fleur: thank you for motivating me to be more careful, less cranky, more active, and just happier. The chances for us to meet were astronomically small, if we think about it, yet here we are. Can't wait to experience what's coming!

I thank the CLS and ETMC collaborations for providing me with configurations for the numerical tests, and in particular to Jacob Finkenrath for personally providing the twisted mass configurations and for multiple interesting conversations on various topics related to my research.

Foreword

The work presented in this thesis is in parts based on the following publications:

- Andreas Frommer, Mostafa Nasr Khalil, and Gustavo Ramirez-Hidalgo. A multilevel approach to variance reduction in the stochastic estimation of the trace of a matrix. *arXiv preprint arXiv:2108.11281*, 2021. Accepted in the SIAM Journal on Scientific Computing
- Jesus Espinoza-Valverde, Andreas Frommer, Gustavo Ramirez-Hidalgo, and Matthias Rottmann. Coarsest-level improvements in multigrid for lattice QCD on large-scale computers. *arXiv preprint arXiv:2205.09104*, 2022

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 765048.

All numerical results shown in this thesis were computed at the Jülich Supercomputing Centre (JSC) using the supercomputer JUWELS, and in two of the machines in our group: *aicomp03* and *aicomp04*.

Contents

Chapter 1

Introduction

Quantum Chromodynamics (QCD) [3, 4] is the theory describing the interaction of quarks and gluons (among themselves and among each other). As *confinement* does not allow isolated quarks in nature, analytic calculations in QCD, when seen the need to be matched with experiments, consist of the description of hadrons (i.e. objects somehow composed of quarks, and allowed to be in isolation due to no confinement restrictions). Analytic calculations in physics oftentimes rely on perturbative expansions of the theory under study. Such mathematical tools are not applicable when studying QCD in some particular energetic regimes. In such cases, alternative methods have to be employed, one of them being the use of numerical and computational methods.

Lattice QCD is the discretization of the continuous QCD theory on a four dimensional lattice, with the use of Wick rotations enabling the possibility of simulating the theory on a discretized Euclidean space-time [5]. Many challenges emerge, from the point of view of applied mathematics and high performance computing, when simulating QCD on the lattice, and it is one of the world's most demanding computational problems [6, 7]. The success of lattice QCD outweighs, by far, its difficulty, leveraging results in very good agreement with experiments, see e.g. [8].

As described in chapter 2, simulating the theory on the lattice implies the frequent solution of linear systems of equations. When trying to match the discretized QCD theory with the one in the continuum, the parameters of the theory on the lattice (e.g. quark masses and lattice volume) change, and these changes imply more ill-conditioned linear systems to be solved, which in the physics community is termed as "critical slowing down". Not only do the systems become more ill-conditioned, but also the associated matrices of coefficients increase in size. These two issues force an appropriate mixture of methods coming from numerical linear algebra and high performance computing.

To cope with those highly ill-conditioned and large systems of equations, the lattice QCD community has traditionally used methods like odd-even preconditioning [9, 10], deflation [11] or domain decomposition [12, 13], and although they all bring improvements over traditional Krylov methods, they still suffer from critical slowing down. Multigrid methods represent an attractive alternative for the solves in lattice QCD, due to their potential (e.g., for elliptic PDEs) of convergence independently of the conditioning of the linear system. Due to the random nature of the matrices appearing in lattice QCD simulations, the use of *geometric* multigrid methods (i.e. methods based on the underlying PDE only) was elusive for many years [14–17].

Multigrid (or multigrid-related) methods are currently being used in the lattice QCD community [11, 18–20], with multiple libraries implementing them [21–23]. The new developments and implementations that we explore in this thesis revolve around the DD-αAMG method, currently available for the clover-improved Wilson [24–26] and twisted mass [27–29] lattice discretizations.

When extremely large and ill-conditioned linear systems are being solved via multigrid methods, the scalability of the implementation is typically compromised as we move to a large number of compute nodes. When using (scalable) domain decomposition smoothers, as in DD-αAMG, this poor scalability is then caused by the (typically) bad scaling properties of the coarsest level in the multigrid hierarchy. Our first contribution here is on diminishing these scalability issues in DD-αAMG. We do this by integrating recycling methods [30] with a polynomial preconditioner [31–33]; we find that this combination has a great algorithmic performance for the problems at hand. Furthermore, we exploit locality by including a block-diagonal preconditioner as well, based on the idea of block Jacobi [34]. Finally, we explore communication hiding via pipelining [35]. All these methods, when combined, leverage a complex and powerful coarsest-level solver, which in the case of Wilson fermions gives us remarkable algorithmic and computational gains. When applied to twisted mass fermions, these Krylov-based methods (plus the block-diagonal preconditioner) allow us to get rid of an "artificially"-introduced coarsest-level parameter. We further broaden the algorithmic possibilities at the last level in DD-αAMG, by using an LU-based approximate solver as a preconditioner to restarted GMRES. Those approximate direct solves, done via the MUMPS library [36, 37], allow us to, in the twisted mass case, not only get rid of the same "artificial" parameter at the coarsest level, but they also give us improved algorithmic and computational performances, on a single node, with respect to all the other methods explored here.

Our second contribution deals with GPU programming. We extend the DD-αAMG library to become a hybrid GPU+CPU solver, by porting some parts of the code via CUDA C [38]. In doing so, we realize the importance of having a smoother based on domain decomposition, and furthermore we explore the

computational behaviour of the smoother when having different sizes for the domain decomposition blocks. We conclude that smaller blocks are better in terms of computational performance. We also notice the importance of using (more) aggressive coarsening in the multigrid hierarchy, when running with our hybrid solver. The resulting implementation currently performs with an execution time similar to the old CPU version, but with the great advantage that further GPU improvements at the finest level can render a hybrid solver largely outperforming the CPU version.

Our third and last contribution is on the development and testing of a new method for the computation of traces of functions of matrices, $\mathrm{tr}(f(A))$. This new method is based on multilevel Monte Carlo [39], in combination with a multigrid hierarchy. Although completely general in terms of the function f and the matrix A, the method is tested here on the inverse[1] (i.e. $f(A) = A^{-1}$), for three matrices: Schwinger, Wilson and twisted mass. We show that the method works in all three cases, with both algorithmic and computational gains in all, with remarkable results in Schwinger, and very good and promising results for Wilson and twisted mass. These results open new paths of research, where our multigrid multilevel Monte Carlo method can be used in combination with other methods such as deflation [40] and hierarchical probing [41].

The results on multilevel Monte Carlo applied to the Schwinger model, and presented in chapter 6 of this thesis, have been published in [1]. The results on Krylov-based coarsest-level improvements in chapter 4 are part of one of our papers currently under preparation. The LU-based and the application of agglomeration, both part of chapter 4, will be part of a paper soon to be prepared, within the context of Henning Leemhuis' PhD research. The code for the hybrid GPU+CPU solver discused in chapter 5 is already available here, and after porting the whole finest level to CUDA C we will prepare a paper to publish the corresponding results. Finally, the multilevel Monte Carlo results on its application to lattice QCD (i.e. for both Wilson and twisted mass fermions), in chapter 6, will be part of a paper soon to be prepared, and within the context of Jose Jiménez's PhD research.

The remaining of this thesis is structured as follows: we give a brief overview of lattice QCD, in chapter 2, with emphasis on the most important concepts in terms of where the computation of linear systems of equations and $\mathrm{tr}(f(A))$ appear. We then go and describe the whole DD-αAMG method in chapter 3, skipping some very specific and technical details along the way. In chapter 4, we describe the algorithmic nature of the new coarsest-level solver in DD-αAMG, both in the Krylov- and LU-based cases, with extensive numerical tests for both. Our new hybrid GPU+CPU DD-αAMG solver is presented in chapter 5, where we discuss

[1] We focus on the inverse here, due to its importance in lattice QCD.

technical details of its implementation, and extensive numerical tests showing the good performance and further potential of the solver. Finally, chapter 6 introduces our new method for the computation of $\mathrm{tr}(f(A))$ based on multilevel Monte Carlo, with its application to Schwinger, Wilson and twisted mass matrices.

Chapter 2

Quantum chromodynamics on the lattice

Oftentimes, there is a deep link between new algorithmic developments in applied mathematics, and the conceptual grounds of the physical systems they lay on. Understanding this connection is usually of utmost importance, for gaining a better intuition not only of the restrictions imposed on the algorithms, but also of the expected behaviour from them. This is particularly important in lattice quantum chromodynamics. Having a clear view of those conceptual grounds is useful, also, for a full understanding of the motivation behind the implementations and new developments. In this chapter we set up and define all the physics background behind the methods discussed in subsequent chapters.

We start this chapter by stating how quantum chromodynamics, the theory describing the strong force, fits in the standard model of theoretical physics. We then briefly describe, in sect. 2.2, the role that path integrals play when describing quantum chromodynamics on a lattice and we shortly outline the stochastic method employed to simulate lattice quantum chromodynamics i.e. hybrid Monte Carlo. The two lattice discretizations of relevance in this thesis, Wilson and twisted mass, are presented and briefly discussed in sect. 2.3. Finally, we shortly introduce two topics which will be of importance when we turn to the discussion of traces of functions of matrices in chapt. 6: disconnected diagrams (sect. 2.4) and the Schwinger model (sect. 2.5).

Sect. 2.1 is largely based on [42], sects. 2.2 and 2.4 on [43, 44], sect. 2.3 on [28, 45], and sect. 2.5 on [46].

2.1 The standard model and quantum chromodynamics

Three out of the four fundamental forces, electromagnetism, weak interation and the strong force, are currently described in a single theory known as the Standard Model of Particle Physics [42]. Via mostly group theory and quantum field theory, the standard model describes how different combinations of particles interact and behave under those interactions. To this end, after decades of theoretical developments and experimental advances and with the help of particle accelerators, a small set of particles is currently considered as "fundamental", and we present those particles, their charges and masses in a tabulated manner in tab. 2.1.

Generation	Leptons			Quarks		
	Particle	Q	mass/GeV	Particle	Q	mass/GeV
First	electron (e^-)	-1	0.0005	down (d)	-1/3	0.003
	neutrino (ν_e)	0	$< 10^{-9}$	up (u)	+1/3	0.005
Second	muon (μ^-)	-1	0.106	strage (s)	-1/3	0.1
	neutrino (ν_μ)	0	$< 10^{-9}$	charm (c)	+2/3	1.3
Third	tau (τ^-)	-1	1.78	bottom (b)	-1/3	4.5
	neutrino (ν_τ)	0	$< 10^{-9}$	top (t)	+2/3	174

Table 2.1: The twelve fundamental fermions divided into quarks and leptons, with their corresponding charge and mass. Table taken from [42].

But, not all fermions "feel" the three forces described by the standard model. To understand this better, we can look at the table in fig. 2.2: the quarks, for example, are the only fermions who feel the strong force.

					strong	electromagnetic	weak
Quarks	down,up	d,u	s,c	b,t	✓	✓	✓
Leptons	charged	e^-	μ^-	τ^-		✓	✓
	neutrinos	ν_e	ν_μ	ν_τ			✓

Table 2.2: The forces experienced by different fundamental fermions. Table taken from [42].

In its current form, the standard model describes the electromagnetic and weak forces in a single combined model known as electroweak theory. This allows understanding the electromagnetic and weak forces as a single force, which can be broken under certain conditions to split into the two separate constituent forces. On the other hand, the strong force, although not being currently understood as a single force in combination with the electroweak one, is part of the standard

model and as such shares many properties with electromagnetism and the weak force.

The common tools that describe these three forces are symmetries and quantum field theory: the theories that describe each of the forces are all quantum field theories, each of them with a different group associated to it. The electroweak force is described by the local $SU(2)\times U(1)$ group, which undergoes spontaneous symmetry breaking to give masses to certain particles in the standard model [47]. The strong force is described by the local $SU(3)$ group, there is no breaking in this case and the resulting theory is known as Quantum Chromodynamics (QCD).

With the number of generators of the group associated to each force, a corresponding number of *exchange bosons* (also known as *gauge bosons*) come into play for each of those forces [48]. For example, the group $U(1)$ which describes electromagnetism has a single degree of freedom, hence there is only a single exchange boson for that force i.e. the photon. The table in fig. 2.3 lists the gauge bosons corresponding to each of the four forces in nature, some of their properties, and an approximate value of the strength of the interaction in each of the forces.

Force	Strength	Boson		Spin	Mass/GeV
Strong	1	Gluon	g	1	0
Electromagnetism	10^{-3}	Photon	γ	1	0
Weak	10^{-8}	W boson	$W^{\pm}$	1	80.4
		Z boson	Z	1	91.2
Gravity	10^{-37}	Graviton?	G	2	0

Table 2.3: Exchange bosons for the four forces in nature. The relative strengths are approximate indicative values for two fundamental particles at a distance of 1 fm = 10^{-15} m (roughly the radius of a proton). Table taken from [42].

Usually, in physics, Taylor series (or other asymptotic expansions) can be used to e.g. simplify certain analytic calculations, that are otherwise very difficult to solve or simply not solvable at all. This is the case in the standard model, where series expansions are used throughout the whole theory. There are some situations, or more specifically some energetic regimes, in which series expansions are not useful in the standard model, for example when studying QCD in some low-energy interactions. An alternative to those tools is to keep the model as a whole, no analytic approximations, and treat it via computational methods.

The history of QCD combines a plethora of different areas involving physics and mathematics: representations in group theory, statistical mechanics, renormalization of non-Abelian group theories, experimental scattering, and many others, all of them converging in experimental settings taking place in the single largest and most complex machine built in human history: the Large Hadron Collider at

Conseil Européen pour la Recherche Nucléaire (CERN). We skip here a historical overview of QCD[2] and rather state some properties of QCD from a theoretical point of view. A full understanding of QCD implies a good understanding of at least modern quantum mechanics [50, 51], special relativity [52] and quantum field theory [53, 54]. We describe QCD here from an applied mathematics point of view, where we state the necessary concepts in order to describe how QCD can be simulated on computers.

2.2 Path integral and hybrid Monte Carlo

Statistical mechanics, which studies macroscopic phenomena in nature solely from the application of statistical methods and probability theory to large assemblies of microscopic objects [55], repeatedly makes use of probability distribution functions (p.d.f.s) and averages and variances in order to study and characterize a physical system. This is mainly done via the *partition function* which is, in probability terms, the normalization factor of the p.d.f. describing that system. The partition function is not the way to understand the full dynamics of the system, but rather it gives access to macroscopic features of it known as *observables*.

The concept of partition function permeates many areas of study in physics. In particular, it is fundamentally important in quantum mechanics and furthermore in quantum field theory, where it is realized as a *path integral* [56, 57]. The path integral in quantum field theory, just as in statistical mechanics, allows for the extraction of observables without having to work out many very small and complicated technical mathematical details of the theory.

We skip here all the mathematical and conceptual developments necessary to understand how the path integral emerges in QCD, and we simply present it as a tool that is necessary to obtain observables in the theory. Furthermore, as was mentioned before, for some energetic regimes QCD has to be simulated on computers in order to access values for those observables. Therefore, we restrict ourselves here to the path integral for QCD on the lattice.

Euclidean *correlators*, which are very important quantities in lattice QCD used e.g. in the extraction of masses of bound states (for example, the proton mass), can be expressed by means of path integrals in the following way [43]:

$$\langle O_2(t)O_1(0)\rangle = \frac{1}{Z}\int \mathcal{D}[\psi,\bar{\psi}]\mathcal{D}[U]e^{-S_F[\psi,\bar{\psi},U]-S_G[U]}O_2[\psi,\bar{\psi},U]O_1[\psi,\bar{\psi},U] \quad (2.1)$$

[2]Which can be found for example in [49].

where the partition function is given by

$$Z = \frac{1}{Z} \int \mathcal{D}[\psi, \bar{\psi}] \mathcal{D}[U] e^{-S_F[\psi,\bar{\psi},U] - S_G[U]}. \tag{2.2}$$

In eqs. 2.1 and 2.2, ψ embodies the fermionic information[3], and U is an indirect representation of the gauge bosons i.e. gluons, and the latter are known as *gauge links*. The gluonic *action* $S_G[U]$ contains information relevant to fully understand the dynamics of the interaction of gluons with each other. The fermionic action $S_F[\psi, \bar{\psi}, U]$, on the other hand, gives access to the dynamics of the interaction of fermions via (and with) gluons.

On the lattice, the measures in eqs. 2.1 and 2.2 are products of measures of all quark field components and products of measures of all gauge link variables:

$$\mathcal{D}[\psi, \bar{\psi}] = \prod_{n \in \Lambda} \prod_{f,\alpha,c} \mathtt{d}\psi^{(f)}(n)_{\alpha,c} \mathtt{d}\bar{\psi}^{(f)}(n)_{\alpha,c}, \quad \mathcal{D}[U] = \prod_{n \in \Lambda} \prod_{\mu=1}^{4} \mathtt{d}U_\mu(n). \tag{2.3}$$

The gauge link $U_\mu(n)$ in eq. 2.3 is an object connecting the lattice site n to the lattice site $n + \hat{\mu}$.

As can be seen from the measures in eq. 2.3, the integration in eq. 2.1 consists of a very-high-dimensional integral, an integration for each $\psi^{(f)}(n)_{\alpha,c}$, $\bar{\psi}^{(f)}(n)_{\alpha,c}$ and $U_\mu(n)$, where n runs over all the sites in the lattice. Using a deterministic integration method such as e.g. Simpson's rule [58] to compute eq. 2.1 is not a good option as the error usually grows nestedly with the dimensionality of the integral. Monte Carlo methods, in particular Markov Chain Monte Carlo (MCMC), are a much better alternative to solve such high-dimensional integrals.

The MCMC method of choice in current lattice QCD computations is hybrid Monte Carlo (HMC) [59]. This method is also known as Hamiltonian Monte Carlo. This algorithm takes, in the particular case of lattice QCD, the total action $S = S_F + S_G$ and defines it as proportional to a classical Hamiltonian. In order to fully describe the dynamics of a classical system characterized by that Hamiltonian and to use Hamiltonian mechanics [60], artificial auxiliary momenta $P_\mu(n)$ are introduced as conjugate (in the Hamiltonian-mechanics sense) to the gauge links $U_\mu(n)$[4]. HMC generates a set of configurations using *Markov chains*

[3] The fields ψ, following Fermi statistics and the Pauli exclusion principle, are Grassmannian fields. The study of Grassmannian variables is beyond the scope of this thesis (see for example [53]).

[4] Another step is necessary at this point: the introduction of *pseudofermions*. This allows us to move from a Grassmannian integral, which is not a good representation for numerical simulations, to integrals of complex-valued functions. As understanding HMC deeply is not necessary to follow the rest of this thesis, the use of pseudofermions is beyond the scope of this thesis and can be studied further e.g. in [43].

where each configuration is an "evolution" (in the Molecular Dynamics (MD) sense) of the previous one. We give an outline of the fundamental ideas of the HMC algorithm next [61–63]:

1. Choose an initial configuration U_0 and set $i = 1$.
2. Generate random momentum fields P conjugate to U_{i-1}.
3. Evolve the configuration U_{i-1} via MD to obtain a new candidate U'.
4. Accept $U_i = U'$ with some probability P_{acc}, otherwise set $U_i = U_{i-1}$, $i \leftarrow i+1$ and go to step 2.
5. If U_i is *thermalized*, save it.
6. Go to step 2 until enough configurations are generated.

For the initial configuration, two common approaches are a *cold start* where all gauge links are set to the identity, or a *hot start*, where all gauge links are random elements of the $SU(3)$ group. In step 4 the acceptance rate P_{acc} of the new configuration is determined using the *Metropolis-Hastings* algorithm [64, 65], which allows us to sample from the p.d.f. $e^{-S_F-S_G}$ after *thermalization*. The term *thermalized* in step 5 can be interpreted as a *converged* configuration i.e. given enough steps of the HMC algorithm, the configuration space reaches an *equilibrium state* in which the distribution of the gauge links follows the prescribed equilibrium distribution, such that new physical configurations can be generated by going back to step 2 and repeat the procedure. Thermalization is important to ensure that physically more likely configurations are also more likely to be produced by the HMC algorithm.

2.3 Lattice discretizations in quantum chromodynamics

A first step into computing the integral in eq. 2.1 is to choose an appropriate numerical integration scheme, for which the lattice QCD community has chosen HMC. A second step consists of fully describing the lattice on which the computations will take place and writing a discretized form of the total action $S = S_F + S_G$ on it.

In simulating QCD on the lattice via HMC, an important step that emerges is the need to solve a partial differential equation (PDE) on the lattice. This PDE corresponds to the Dirac equation [53, 66] for fermionic fields interacting via (and with) the gluonic fields. In the continuum, the operator $\mathcal{D}$ characterizing this PDE, also known as the covariant derivative of the theory, can be written as

$$\mathcal{D} = \sum_{\mu=0}^{3} \gamma_\mu \otimes (\partial_\mu + A_\mu) \tag{2.4}$$

where $\partial_\mu = \partial/\partial x_\mu$ and $A_\mu(x)$ is the field describing the gauge bosons (at the spacetime point x). The anti-Hermitian traceless matrices $A_\mu(x)$ are elements of $\mathfrak{su}(3)$, the Lie algebra spanning $SU(3)$. The Dirac matrices i.e. $\gamma_\mu \in \mathbb{C}^{4\times4}$ are Hermitian and unitary matrices which generate the Clifford algebra $C_{0,4}(\mathbb{R})$ [67].

Definition 2.1.
A set of Hermitian, unitary matrices $\{\gamma_\mu \in \mathbb{C}^{4\times4} : \mu = 0,1,2,3\}$ *is called a set of generators of the* Clifford algebra $C_{0,4}(\mathbb{R})$*, iff*

$$\gamma_\mu\gamma_\nu + \gamma_\nu\gamma_\mu = \begin{cases} 2 \cdot I_4, & \mu = \nu \\ 1, & \mu \neq \nu \end{cases} \qquad \mu,\nu = 0,1,2,3. \tag{2.5}$$

The matrices γ_μ *are called* γ-matrices *or* Dirac matrices.

Before writing an explicit discretization of the operator in eq. 2.4, let us give a more formal statement on the properties of the gluonic fields in the continuum.

Definition 2.2.
Let $\mathcal{C} := \{1,2,3\}$ *be the set of* color indices*,* $\mathcal{S} := \{0,1,2,3\}$ *the* spin indices *and*

$$\begin{aligned} \psi : \mathbb{R}^4 &\to \mathbb{C}^{12} \cong \mathbb{C}^{\mathcal{C}\times\mathcal{S}} \\ x &\mapsto (\psi_{10}(x), \psi_{20}(x), \psi_{30}(x), \psi_{11}(x), ..., \psi_{33}(x))^T \end{aligned}$$

a differentiable function. Then ψ *defines a* quark field *or* matter field*. Let* $\mathcal{M} = \{\psi : \psi\ matter\ field\}$*. The twelve component vector* $\psi(x)$ *is called* spinor*. Furthermore, for* $\mu = 0,1,2,3$

$$\begin{aligned} A_\mu : \mathbb{R}^4 &\to \mathfrak{su}(3) \\ x &\mapsto A_\mu(x) \end{aligned}$$

the set $\{A_\mu : \mu = 0,1,2,3\}$ *defines a* gauge field*, i.e., a gluonic counterpart of a quark field.*

The multiplication of a γ-matrix with ψ is defined by $(\gamma_\mu\psi)(x) := (\gamma_\mu \otimes I_3)\psi(x)$, with I_3 the identity in color space. In case operations act unambiguously on the color but differently on the spin degrees of freedom we use the notation ψ_σ to

denote those components of the quark field belonging to the fixed spin index σ. For a given point x, $\psi_\sigma(x)$ is thus represented by a three component column vector $\psi_\sigma(x) = (\psi_{1\sigma}(x), \psi_{2\sigma}(x), \psi_{3\sigma}(x))^T$. The value of the gauge field A_μ at point x acts non-trivially on the color and trivially on the spin degrees of freedom in the sense that $(A_\mu\psi)(x) := (I_4 \otimes A_\mu(x))\psi(x)$.

This allows us to define the effect of the covariant derivative in eq. 2.4 on matter fields in the following way.

Definition 2.3.
Let $\mathcal{M}$ be the space of matter fields. The continuum Dirac operator *is the map*

$$\mathcal{D} : \mathcal{M} \to \mathcal{M}$$

defined by

$$\mathcal{D} = \sum_{\mu=0}^{3} \gamma_\mu \otimes (\partial_\mu + A_\mu)$$

where $\partial_\mu = \partial/\partial x_\mu$ denotes the partial derivative in direction μ. Evaluating $\mathcal{D}\psi$ at $x \in \mathbb{R}^4$, we have

$$(\mathcal{D}\psi)(x) = \sum_{\mu=0}^{3} \gamma_\mu((\partial_\mu + A_\mu)\psi)(x). \tag{2.6}$$

The need for the gamma matrices in the covariant derivative come from imposing that the dynamics of the physical system is invariant under transformations by the Lorentz group i.e. under special relativity's boosts and rotations [53]. The fields A_μ represent the gauge bosons of QCD i.e. the gluons, which appear after enforcing invariance of the equations of motion of the system under local $SU(3)$ transformations.

Having defined the Dirac operator in the continuum $\mathcal{D}$ and its effect on matter fields ψ, the last step before discretizing the Dirac operator on the lattice is to define the lattice itself.

Definition 2.4.
Consider a four-dimensional torus $\mathcal{T}$. Then, a lattice $\mathcal{L}$ *with* lattice spacing a *is a subset of $\mathcal{T}$ such that for any $x, y \in \mathcal{L}$ there exists $p \in \mathbb{Z}^4$ fulfilling*

$$y = x + a \cdot p, \quad i.e., \; y_\mu = x_\mu + a \cdot p_\mu \quad for \; \mu = 0, 1, 2, 3$$

For shift operations on the lattice, let $\hat{\mu} \in \mathbb{R}^4$ *denote* shift vectors *defined by*

$$\hat{\mu}_\nu = \begin{cases} a, & \mu = \nu \\ 0, & else \end{cases} \qquad \mu, \nu = 0, 1, 2, 3$$

The full discretization of the Dirac equation, i.e. of the PDE associated to the operator in eq. 2.6, follows a clear path at this point:

1. Discretize $\psi(x)$ and $A_\mu(x)$ on the lattice[5].
2. Choose a discretization scheme for the differential part in eq. 2.6.

For the ψ field, it is sufficient to be defined at each lattice point only, as follows

$$\begin{aligned} \psi : \mathcal{L} &\to \mathbb{C}^{12} \\ x &\mapsto \psi(x) \end{aligned}$$

As in continuum QCD, the spinor $\psi(x)$ again has color and spin indices $\psi_{c\sigma}$, $c \in \mathcal{C}$ and $\sigma \in \mathcal{S}$ (see def. 2.2).

We have stated before that we are skipping here many specific details of continuum QCD, this in favour of more technical details of QCD on the lattice from an applied mathematics point of view. One such detail is that the continuum fields $A_\mu(x)$, associated to the gauge bosons, connect infinitesimally close spacetime points [53]. Those infinitesimally close points in spacetime become two points on the lattice next to each other. The discretization of continuum $A_\mu(x)$ onto the lattice is done via the introduction of $U_\mu(x)$ fields.

Definition 2.5.
Given a gauge field A_μ*, the corresponding discretized gauge field* U_μ *at point* x *is defined by the path ordered integral along the link* $(x, x + \hat{\mu})$

$$U_\mu(x) := e^{-\int_x^{x+\hat{\mu}} A_\mu(s) ds} \approx e^{-aA_\mu(x+\frac{1}{2}\hat{\mu})}$$

The discretized gauge field $U = \{U_\mu(x) : x \in \mathcal{L},\ \mu = 0, 1, 2, 3\}$ *is called* (gauge) configuration.

[5] In lattice QCD calculations, the initial point is always a discrete gauge configuration U (introduced in def. 2.5) and the translation from A_μ to U_μ is more of a theoretical interest and never performed in practice.

These gauge configurations are precisely the same ones introduced in eqs. 2.1 and 2.2, where we mentioned that gauge links are an indirect representation of the gauge bosons on the lattice. It is clear at this point why we refer to them as *gauge links*.

Since $U_\mu(x)$ connects x with $x + \hat{\mu}$, we regard $U_\mu(x)$ as being associated with the *link* between those two points. The link going in the opposite direction i.e. from $x + \hat{\mu}$ to x, is given by $U_\mu(x)^{-1}$. The matrices $U_\mu(x)$ satisfy

$$U_\mu(x) \in SU(3), \text{ in particular } U_\mu(x)^{-1} = U_\mu(x)^H$$

Fig. 2.1 illustrates the naming conventions for the representation of gauge links on the lattice.

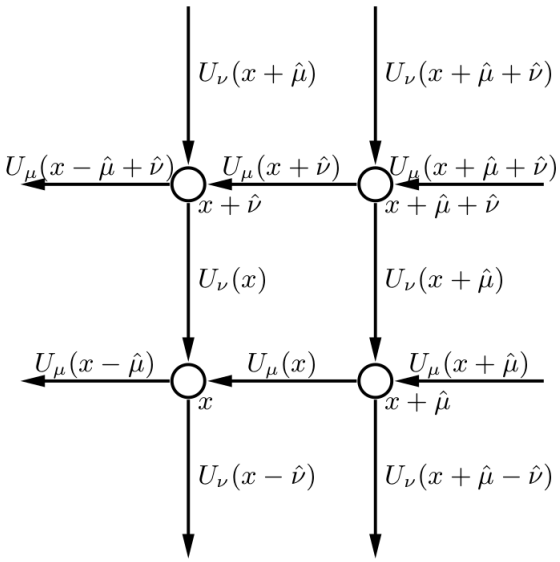

Figure 2.1: Our convention for gauge links on the lattice. Image taken from [45]. A more common convention is the one where the gauge links go in the opposite direcion [43].

There are multiple approaches to discretizing the continuum Dirac operator, resulting in different discretized operators D e.g. Wilson, Twisted Mass, Staggered, and others [5, 43, 68, 69]. We focus here on the two which are of interest to our computational implementations and tests: Wilson and Twisted Mass.

Wilson

We continue now by discretizing the differential part in the covariant derivative via centralized finite differences.

Definition 2.6.
Let A_μ be a gauge field and U_μ the corresponding gauge configuration. Defining forward covariant finite differences[6]

$$(\Delta^\mu \psi_\sigma)(x) := \frac{U_\mu(x)\psi_\sigma(x+\hat{\mu}) - \psi_\sigma(x)}{a} \underset{a\to 0}{=} (\partial_\mu + A_\mu)\psi_\sigma(x)$$

and backward covariant finite differences[7]

$$(\Delta_\mu \psi_\sigma)(x) := \frac{\psi_\sigma(x) - U_\mu^H(x-\hat{\mu})\psi_\sigma(x-\hat{\mu})}{a}$$

the centralized covariant finite difference discretization of the Dirac operator $\mathcal{D}$ is given by

$$D_N := \sum_{\mu=0}^{3} \gamma_\mu \otimes (\Delta_\mu + \Delta^\mu)/2. \tag{2.7}$$

This is called the *naive* discretization of the Dirac operator. The naive discretization generates unphysical eigenvectors, a standard phenomenon when discretizing first order derivatives using central finite differences [70], also known as the "species doubling effect" [71, 72] or "red-black instability". The eigenspace for each eigenvalue of D_N has dimension 16, but only a one-dimensional subspace corresponds to an eigenfunction of the continuum operator. In order to avoid the doubling problem, Wilson introduced the stabilization term $a\Delta_\mu\Delta_\mu$, a centralized second order covariant finite difference.

Definition 2.7.
Given a gauge configuration U on a lattice $\mathcal{L}$ with $n_\mathcal{L}$ sites, lattice spacing a and mass parameter m_0, the Wilson discretization *of the Dirac operator (also known as* Wilson-Dirac operator*) is defined by*

[6]The use of upper and lower indices as in e.g. Δ^μ or Δ_μ is not meant to be as in General Relativity where they refer to either covariant or contravariant spaces. The notation is not only avoided here, but it would be useless anyways as we are dealing with a Euclidean space when studying computations on the lattice.

[7]Here we use μ as sub and superscript, to denote backward and forward covariant finite differences, respectively, instead of for example $\overleftarrow{\Delta}$ and $\overrightarrow{\Delta}$.

$$D_W := \frac{m_0}{a} I_{12n_\mathcal{L}} + \frac{1}{2} \sum_{\mu=0}^{3} (\gamma_\mu \otimes (\Delta_\mu + \Delta^\mu) - a I_4 \otimes \Delta_\mu \Delta^\mu) \tag{2.8}$$

where the mass parameter m_0 sets the quark mass[8]*.*

The anti-commutation relations of the γ-matrices (see def. 2.1) imply a non-trivial symmetry of D_W.

Lemma 2.8.
Defining $\gamma_5 := \gamma_0\gamma_1\gamma_2\gamma_3$ and with $\Gamma_5 := I_{n_\mathcal{L}} \otimes \gamma_5 \otimes I_3$, with sizes 12×12 and $12n_\mathcal{L}$×$12n_\mathcal{L}$, respectively, the Wilson-Dirac operator D_W is Γ_5-symmetric i.e.

$$(\Gamma_5 D_W)^H = \Gamma_5 D_W. \tag{2.9}$$

Proof. A proof of this lemma can be found in e.g. [45]. It relies on the fact that $\gamma_5\gamma_\mu = -\gamma_\mu\gamma_5$ for $\mu = 0, 1, 2, 3$. D_W itself is not Hermitian.

To reduce the order of the discretization error as a funtion of the lattice spacing a, the Sheikholeslami-Wohlert or *clover* term [74], depending on a parameter c_{sw}, is added to the Wilson-Dirac operator.

Definition 2.9.
Given a configuration of gauge links $\{U_\mu(x)\}$, the plaquette $Q_x^{\mu,\nu}$ *at lattice point x is defined as*

$$Q_x^{\mu,\nu} := U_\nu(x) U_\mu(x + \hat{\nu}) U_\nu^H(x + \hat{\mu}) U_\mu^H(x)$$

Furthermore, defining

$$Q_{\mu\nu}(x) := Q_x^{\mu,\nu} + Q_x^{\mu,-\nu} + Q_x^{-\mu,\nu} Q_x^{-\mu,-\nu}$$

the clover term C is defined as

$$C(x) := \frac{c_{sw}}{32} \sum_{\mu,\nu=0}^{3} (\gamma_\mu\gamma_\nu) \otimes (Q_{\mu\nu}(x) - Q_{\nu\mu}(x)) \tag{2.10}$$

with $c_{sw} \in \mathbb{R}$. For an arbitrary quark field ψ and a lattice site x, the clover improved *Wilson-Dirac operator D is defined as*

[8] For further details on the specific relation between quark masses and the mass parameter m_0 see [73].

$$D\psi(x) := D_W\psi(x) - C(x)\psi(x). \tag{2.11}$$

The clover term is diagonal on the lattice $\mathcal{L}$, which is computationally appealing. It removes $\mathcal{O}(a)$-discretization effects from the covariant finite difference discretization of the continuum Dirac operator. Different lattice QCD simulations require an appropriate tunning of c_{sw} [74]. The resulting discretized Dirac Dirac operator D has then local discretization errors of order $\mathcal{O}(a^2)$. On the other hand, in practice, m_0 is negative and for physically relevant mass parameters, the spectrum of D is contained in the right half plane (see fig. 2.2). Both operators, D and D_w, have some interesting and useful spectral properties, stated in lemma 2.10.

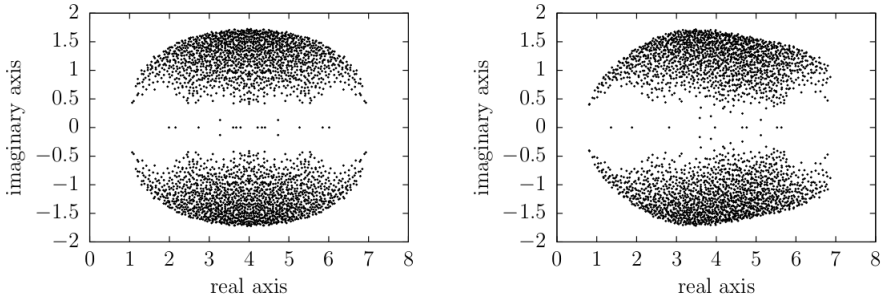

Figure 2.2: *Left panel*: spectrum of a 4^4 Wilson-Dirac operator with $m_0 = 0$ and $c_{sw} = 0$. *Right panel*: spectrum of a 4^4 clover improved Wilson-Dirac operator with $m_0 = 0$ and $c_{sw} = 1$. Image taken from [45].

Lemma 2.10. *(a) The clover improved Wilson-Dirac operator D is Γ_5-symmetric i.e.*

$$(\Gamma_5 D)^H = (\Gamma_5 D). \tag{2.12}$$

(b) Every right eigenvector ψ_λ to an eigenvalue λ of D corresponds to a left eigenvector

$$\bar{\psi}_{\bar{\lambda}} = \Gamma_5\psi_\lambda$$

to the eigenvalue $\bar{\lambda}$ of D and vice versa. In particular, the spectrum of D is symmetric with respect to the real axis.

(c) The spectrum of D_W is symmetric with respect to the vertical line $\mathrm{Re}(z) = \frac{4+m_0}{a}$ i.e.

$$\lambda \in \mathrm{spec}(D_W) \quad \Rightarrow \quad 2\frac{m_0+4}{a} - \lambda \in \mathrm{spec}(D_W).$$

Proof. A proof of this lemma can be found in e.g. [45].

Depending on the choice for the specific representation of the γ-matrices we will get slightly different expressions for the Dirac operator on the lattice. Despite these differences, the physical results will ultimately be the same for all those different representations. We use, througout this thesis, representations of the γ-matrices such that

$$\gamma_5 = \gamma_0\gamma_1\gamma_2\gamma_3 = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & -1 & \\ & & & -1 \end{pmatrix}$$

Twisted mass

The continuum QCD action (see eqs. 2.1 and 2.2) is invariant under *chiral symmetry* [42, 53] in the massless case. This means that, by applying these chiral transformations, the form of the action in the continuum changes but its actual value does not, implying that the underlying physics will remain the same. Having a different form of the action leads to a different form for the PDE to be solved when applying HMC to simulate the theory. This means a different expression for the Dirac operator in the continuum.

Following this procedure of: chiral symmetry transformation $\rightarrow$ new continuum Dirac operator $\rightarrow$ discretize on the lattice, one can end up with a new lattice discretization [68, 75, 76]

$$D_{TM}(\mu) = D + i\mu\Gamma_5 \tag{2.13}$$

where D is as defined in eq. 2.11 and a new parameter $\mu \in \mathbb{R}$ has been introduced, known as the *twisted mass* parameter. This is the clover improved Wilson-Dirac twisted mass discretization, we refer to it simply as the *twisted mass discretization* and to the operator in eq. 2.13 as the *twisted mass operator*.

The twisted mass operator in eq. 2.13 presents two very interesting properties. The first of them is very advantageous algorithmically, the second one not so much (more on the twisted mass operator in sect. 4.2.2):

1. The parameter μ "shields" the spectrum away from 0 in the sense that the smallest singular value of D_{TM} is $\sqrt{\lambda_{sm}^2 + \mu^2}$ with λ_{sm} the smallest eigenvalue in absolute value of the symmetrized clover-improved Wilson Dirac operator $Q = \Gamma_5 D$.

2. There is some flexibility in choosing the value of μ when simulating QCD with the twisted mass discretization. A particular choice of this parameter is $\mu = m_q$, where m_q is the quark mass, and this choice is called *maximal twist*. At maximal twist, the region of eigenvalues of $D_{TM}^H D_{TM}$ just above μ^2 becomes very dense, which represents a challenge for many algorithms used in solving the linear systems of equations, in particular when using Krylov-subspace-based methods.

2.4 Disconnected diagrams

When simulating QCD on the lattice, the extraction of some physical properties of a particular system can be done via, for example, eq. 2.1. The operators O_1 and O_2 are chosen such that they possess the same quantum numbers as the system under study. For details on how to choose appropriate forms for those operators, see e.g. [43].

A general local *meson*[9] interpolator has the form

$$O_M(n) = \bar{\psi}^{(f_1)}(n)\Gamma\psi^{(f_2)}(n) \tag{2.14}$$

where Γ is a monomial of γ-matrices, $n \in \mathcal{L}$ and f_i refers to a particular quark flavour (there are six quark flavours, as was shown in fig. 2.2). When $f_1 = f_2$, a condition known as *degenerate* flavours, combinations of the interpolator in eq. 2.14 are built to obtain the desired flavour symmetries. Fig. 2.4 lists the matrices Γ for the most commonly used interpolators together with the corresponding quantum numbers.

An example of an interpolator in the degenerate case ($f_1 = f_2$) is $O_S = (\bar{u}\Gamma u + \bar{d}\Gamma d)/\sqrt{2}$ [10]. After some technical steps, which again fall outside of the scope of this thesis, the fermionic part (i.e. the fermionic integration) in eq. 2.1 can be written as ($*$ denotes complex conjugation)

[9] A composite subatomic particle with two or more quarks is known as *hadron*. Those with two quarks are called *mesons* and those with three quarks are *baryons* [42].

[10] The fermionic fields u and d correspond to the two lightest quarks i.e. up and down (see the table in fig. 2.2). The factor of $1/\sqrt{2}$ is known as a *Clebsch–Gordan coefficient*. How to use those coefficients, the use of isospin in the construction of composite subatomic particles and the construction of interpolators such as O_S are all beyond the scope of this presentation (see [42] for more on all of these topics).

State	J^{PC}	Γ	Particles
Scalar	0^{++}	$\mathbb{1}$, γ_4	$f_0, a_0, K_0^*, \ldots$
Pseudoscalar	0^{-+}	γ_5 , $\gamma_4\gamma_5$	$\pi^{\pm}, \pi^0, \eta, K^{\pm}, K^0, \ldots$
Vector	1^{--}	γ_i , $\gamma_4\gamma_i$	$\rho^{\pm}, \rho^0, \omega, K^*, \phi, \ldots$
Axial vector	1^{++}	$\gamma_i\gamma_5$	$a_1, f_1, \ldots$
Tensor	1^{+-}	$\gamma_i\gamma_j$	$h_1, b_1, \ldots$

Table 2.4: Quantum numbers of the most commonly used meson interpolators. Table taken from [43].

$$\begin{aligned}\langle O_S(n)O_S^*(m)\rangle_F &= -\frac{1}{2}\mathrm{tr}[\Gamma D_u^{-1}(n|m)\Gamma D_u^{-1}(m|n)] \\ &= +\frac{1}{2}\mathrm{tr}[\Gamma D_u^{-1}(n|n)]\mathrm{tr}[\Gamma D_u^{-1}(m|m)] \\ &= +\frac{1}{2}\mathrm{tr}[\Gamma D_u^{-1}(n|n)]\mathrm{tr}[\Gamma D_d^{-1}(m|m)] + u \leftrightarrow d.\end{aligned} \tag{2.15}$$

The first term in the right hand side in eq. 2.15 can be associated to fermionic lines propagating first from n to m and then from m back to n, in a connected way, as can be seen in the left-side panel in fig. 2.3. The traces in the other terms in that equation e.g. $\mathrm{tr}[\Gamma D_u^{-1}(n|n)]$, on the other hand, are associated to *disconnected* terms i.e. loops of fermionic lines that go from a point n in spacetime back to the same point n in a single continuous manner (see the right pannel in fig. 2.15).

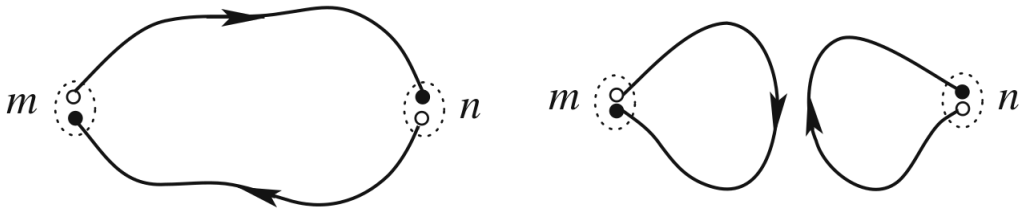

Figure 2.3: *Left panel*: connected pieces of a meson correlator. *Rigt panel*: disconnected pieces of a meson correlator. Image taken from [43].

The traces appearing in eq. 2.15 are often computed via stochastic methods. Numerically, the disconnected contributions need much more computational effort and higher statistics than the connected parts, and many studies avoid considering those mesons or drop the disconnected pieces.

The computation of correlators of the form presented in eq. 2.15, therefore, leads

to a new type of quantity vastly studied in applied mathematics: the trace of the function of a matrix A i.e. $\mathrm{tr}(f(A))$. In the particular case of lattice QCD, the function of interest is $f(A) = \Gamma A^{-1}$, with A being replaced by the Dirac matrix. See [44], in particular chapter 3 therein, for a thorough description on how disconnected diagrams are related to $\mathrm{tr}(\Gamma D^{-1})$ and the importance of Monte Carlo methods in their computation.

2.5 Other theories: the Schwinger model

In our road towards the development of a new method for computing $\mathrm{tr}(f(A))$ in general and the application of it to the particular case of the Dirac matrix in lattice QCD, we have used another model which shares some properties with QCD, namely the Schwinger model: this is a description of Quantum Electrodynamics (QED) in (1+1)-dimensional i.e. two spacetime dimensions [77, 78]. In the development of algorithms for lattice QCD, the operators arising in lattice QED are oftentimes used as a first test-bed, because of the similar spectral behavior and symmetries.

The Dirac operator of lattice QED acts on a 2-dimensional Euclidean space with 2 spin components[11], with the generators of the Clifford algebra now given by the Pauli-matrices

$$\gamma_1 = \sigma_1 = \begin{pmatrix} & 1 \\ 1 & \end{pmatrix} \quad \text{and} \quad \gamma_2 = \sigma_2 = \begin{pmatrix} & i \\ -i & \end{pmatrix} \tag{2.16}$$

The gauge field A_μ of continuum QED in the Schwinger formulation is given as a continuous real-valued function, with the gauge configurations $U \in U(1)$ on the lattice a subset of the complex numbers with modulus one.

Just as in def. 2.7, stabilization of the naive discretization of lattice QED is necessary in order to suppress the doubling problem.

Definition 2.11.
Given a gauge configuration $U_\mu(x)$ on a lattice $\mathcal{L}$ with $n_\mathcal{L}$ sites, lattice spacing a and mass parameter m_0, the Wilson-Schwinger operator S_W is defined by

$$S_W := \frac{m_0}{a} I_{2n_\mathcal{L}} + \frac{1}{2} \sum_{\mu=0}^{1} (\sigma_\mu \otimes (\Delta_\mu + \Delta^\mu) - a I_2 \otimes \Delta_\mu \Delta^\mu) \tag{2.17}$$

where the mass parameter m_0 sets the fermion mass.

[11] Two spin components and no color unlike in QCD, implies that the Dirac operator of lattice QED has 2 degrees of freedom per lattice site (in turn, the lattice QCD Dirac operator has 12).

The analogous to the lattice QCD matrix γ_5 is $\sigma_3 = i\sigma_1\sigma_2$ in lattice QED. Furthermore, the analogous to Γ_5 is

$$\Sigma_3 = \sigma_3 \otimes I_{n_\mathcal{L}}. \tag{2.18}$$

Similarly to lattice QCD, the Wilson-Schwinger operator S_W is Σ_3-symmetric i.e.

$$(\Sigma_3 S_W)^H = \Sigma_3 S_W. \tag{2.19}$$

Some properties of the lattice QCD Dirac operator are shared by the Schwinger operator S_W, as stated next.

Lemma 2.12.
The eigenvalues λ of S_W are either real or appear in complex conjugate pairs.

Proof. The proof of this lemma follows from the proof of the second point in lemma 2.10.

Lemma 2.13.
For any right eigenvector v with eigenvalue λ, i.e., fulfilling $S_W v = \lambda v$, the vector $\Sigma_3 v$ is the left eigenvector to the eigenvalue $\bar{\lambda}$ satisfying

$$(\Sigma_3 v)^H S_W = \bar{\lambda}(\Sigma_3 v)^H$$

Proof. The proof of this lemma follows from the proof of the second point in lemma 2.10.

Chapter 3

Domain decomposition aggregation-based αdaptive algebraic multigrid method

The Domain Decomposition aggregation-based αdaptive algebraic multigrid method (DD-αAMG) is a solver for linear systems of equations arising in simulations of lattice QCD involving Wilson or twisted mass fermions. In this chapter, we introduce all the conceptual background necessary for a full understanding of this method. All the methods implemented in later chapters are done so in the context of DD-αAMG.

Sect. 3.1 contains mostly some of the basic tools from numerical linear algebra necessary for understanding DD-αAMG. Sect. 3.2 gives a general introduction to multigrid, from a rather conceptual point of view. We then present, in sect. 3.3, how algebraic multigrid solvers are realized, focusing on the particular problem of lattice QCD, with sect. 3.3.4 outlining how the ingredients described in 3.3.1, 3.3.2 and 3.3.3 are combined to give rise to DD-αAMG. Finally, sect. 3.3.5 briefly deals with an extension of DD-αAMG from Wilson to twisted mass fermions.

The contents of this chapter are largely based on [34, 45, 79–81].

3.1 Numerical linear algebra fundamentals

Before introducing multigrid methods, in particular algebraic multigrid (AMG), it is useful to introduce first some basics of numerical linear algebra (NLA). All of the concepts presented in this section can be found in e.g. ref.s [34, 79, 82].

3.1.1 Eigenvalues, singular values and conditioning

Certain types of matrices are important in general for the understanding of numerical linear algebra, but in particular for the development of appropriate algorithms to solve certain types of problems (such as solving and/or eigensolving).

Definition 3.1.
We call a matrix $A \in \mathbb{C}^{n\times n}$

- *symmetric, if* $A = A^T$,
- *Hermitian, if* $A = A^H$,
- *unitary, if* $A^H A = I$,
- *normal, if* $A^H A = AA^H$,
- *a projection, if* $A^2 = A$,
- *sparse, if the number of non-zero (nnz) entries per row is significantly smaller than* n *and independent of* n.

Remark 3.2.

1. *We can also define non-square* semi-unitary *matrices:* $A \in \mathbb{C}^{n\times m}$ *with* $n \geq m$ *is semi-unitary, if every column vector* a_i *has unit length and* $\langle a_i, a_j \rangle = 0$ *holds for all* $i \neq j$.
2. *If* A *is a projection then* $(I - A)$ *also defines a projection, as* $(I - A)^2 = I - 2A + A^2 = I - 2A + A = I - A$.

As in many other areas of physics and mathematics, projectors and unitary matrices permeate the whole NLA to aid in solving problems by transforming them from their original form into a simpler or easier one, without being too invasive on the underlying properties of the problem itself.

Later in this thesis, in particular when we discuss deflation in Krylov methods via e.g. GCRO-DR and spectral mappings by means of a polynomial preconditioner, having concepts such as eigenvalue decomposition and singular value decomposition are of upmost importance. We procede then by introducing concepts related to eigenvalues and eigenvectors.

Definition 3.3.
Given a square matrix $A \in \mathbb{C}^{n\times n}$ *we call* $\lambda \in \mathbb{C}$ *an* eigenvalue *of* A *if and only if there exists a nonzero vector* $x \in \mathbb{C}^n$ *such that*

$$Ax = \lambda x. \tag{3.1}$$

Additional characteristics and terms related to eigenvalues:

- *x is called an* eigenvector (belonging to λ).
- *A pair (λ, x) of eigenvalue λ and its eigenvector x is called an* eigenpair.
- *The set of all eigenvalues of A is called* spectrum *of A and is denoted by* $\text{spec}(A)$.
- *The* spectral radius of A *is defined as* $\rho(A) := \max_{\lambda \in \Lambda(A)} (|\lambda|)$.
- *Eigenvalues λ_i are the roots of the characteristic polynomial of A, i.e.,* $p_A(\lambda) := \det(A - \lambda I) = 0$.
- *The multiplicity m_i of an eigenvalue in $p_A(\lambda)$ is called* algebraic multiplicity *of λ.*
- *The* geometric multiplicity *of λ_i is denoted by g_i and is the dimension of the eigenspace of λ_i, i.e. the nullspace of $A - \lambda_i I$*

Definition 3.4.
A square matrix $A \in \mathbb{C}^{n \times n}$ is called diagonalizable *if and only if $g_i = m_i$ for all $\lambda_i \in \Lambda$. We define in this case the* eigenvalue decomposition

$$A = XDX^{-1},$$

where each column x_i of X contains an eigenvector of A belonging to the eigenvalue $D_{i,i} = \lambda_i$ of the diagonal matrix D.

More general (and sometimes more important) than the eigenvalue decomposition is the concept of *singular value decomposition.*

Definition 3.5.
Given a matrix $A \in \mathbb{C}^{m \times n}$ we can define the singular value decomposition (SVD) *as the matrix decomposition*

$$A = U\Sigma V^H,$$

where $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ are unitary matrices and $\Sigma \in \mathbb{C}^{m \times n}$ is a diagonal matrix with non-negative entries $\sigma_{i,i}$. We call the column vectors u_i and v_i left *and* right singular vectors (of $\sigma_{i,i}$)*, respectively, and $\sigma_{i,i}$ the* singular values.

The SVD introduced in def. 3.5 exists for any matrix $A \in \mathbb{C}^{n \times m}$, and it is unique up to complex scalar factors of absolute value 1 (a proof of this can be found in e.g. [79, 83]).

As stated in chapter 2, a very frequent operation in lattice QCD simulations is the solution of a linear system of equations. The matrices involved in those solves

(in the lattice QCD context) are large, sparse and the systems hard to solve[12]. Those properties imply that the solution of linear systems of equations usually represent most of the overall execution time in simulations.

Conditioning is a notion that allows us to quantify how "difficult" it is to solve a system of equations, from the properties of its associated matrix of coefficients. Different simulations in lattice QCD will lead to changing Dirac matrices, which in turn will lead to changing conditioning. The conditioning of the Dirac matrix can be drastically different from one simulation to another, depending on the parameters of the simulation (mass, lattice spacing, etc.).

More specifically, and in general, conditioning describes how the output y of a function f is affected by perturbations in the input x, and it is independent of the algorithm which might implement this function on a computer, i.e. conditioning is a property of the problem itself. Inaccuracies in numerical solutions of problems on computers come from two sources: one of them is conditioning of the problem, and the other one is *stability*[13] of the algorithm used to solve the problem.

We interpret a problem as *well-conditioned* if small changes in the input only cause small changes in the output, and vice versa we interpret a problem as *ill-conditioned* if small changes in the input lead to big changes in the output.

Definition 3.6.
Let $f : X \to Y$ be a problem and let $x \in X$. Let δx be some (infinitesimal) perturbation of x and $\delta f := f(x + \delta x) - f(x)$. We define the absolute condition number $\hat{\kappa} = \hat{\kappa}(x)$ *of f at x as*

$$\hat{\kappa} = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \frac{\|\delta f\|}{\|\delta x\|}. \tag{3.2}$$

The relative condition number $\kappa = \kappa(x)$ *is defined as*

$$\kappa = \sup_{\delta x} \left(\frac{\|\delta f\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right). \tag{3.3}$$

If f is differentiable, then

$$\hat{\kappa} = \|J(x)\| \text{ and } \kappa = \frac{\|J(x)\|}{\|f(x)\|/\|x\|}, \tag{3.4}$$

where $J(x)$ is the Jacobian of f at x.

[12] A metric on how hard it is to solve a linear system of equations will be introduced soon, namely *conditioning*.

[13] Stability comes from the discretization of the continuous problem on the computer. We will not further discuss stability here. For more on stability, see e.g. [79].

Well-conditioned problems have a small condition number, whereas ill-conditioned problems have a large one.

The main interest in this thesis is on functions that map an input vector b to an output vector x such that $x = A^{-1}b$ is the solution of a linear system of equations. For this specific mapping, it is known that the condition number can be bounded as $\kappa \leq \|A\| \cdot \|A^{-1}\|$ [79]). It is common practice to talk about the condition number of the matrix A and define it as $\kappa(A) := \|A\| \cdot \|A^{-1}\|$. When the norm is chosen to be the Euclidean 2-norm, then $\|A\|_2 = \sigma_1$, i.e. the largest singular value and $\|A^{-1}\|_2 = \frac{1}{\sigma_n}$, i.e. the reciprocal of the smallest singular value. Then

$$\kappa(A) = \frac{\sigma_1}{\sigma_n}. \tag{3.5}$$

Being σ_1 the largest singular value and σ_n the smallest one, the quotient of both resulting in $\kappa(A)$ gives an indirect measure of the extent of the region in the real axis that we need to have access to in order to solve the problem, and it indicates as well the closeness of that region to the origin of the axis. When continuous problems are solved on the computer, finite precision is necessary, which can be implemented on computers via e.g. the IEEE-754 standard [84]. Precisions in IEEE-754 being non-uniform throughout the real axis, the condition number is telling us how, in solving a problem, we might jump between regions with different separations of a number to its neighbor point on the discretized axis, which might in turn lead to catastrophic rounding errors.

3.1.2 Iterative methods for sparse linear systems of equations

For large and sparse matrices, such as those appearing in lattice QCD simulations, we cannot invert A directly i.e. we cannot compute the full form of A^{-1}, due to both storage and computation time restrictions[14]. Therefore, in finding approximations to x for $Ax = b$ we need to make use of iterative methods.

For large sparse matrices, iterative solvers have been developed whose computational costs are typically dominated by matrix-vector products, which have computational complexity of $\mathcal{O}(n)$. These methods have the additional advantage that the matrix does not need to be stored in memory, but rather only requires a routine for the action Ax of the matrix A on a vector x. Also, iterative methods can be terminated early to give an approximate solution, whereas direct methods typically only yield a feasible solution at the last step of the algorithm.

[14] To compute the full A^{-1}, direct methods have to be employed [85]. Direct methods have the obvious drawbacks of taking up huge amounts of memory and having computational complexities of $\mathcal{O}(n^2)$ or worse.

At the k-th step in an iterative method, with a corresponding approximate solution $x^{(k)}$, the *residual* can be computed to indirectly access the error of the approximation $x^{(k)}$ with respect to the exact solution x.

Definition 3.7.
Defining the residual $r^{(k)} := b - Ax^{(k)}$ *and the* error $e^{(k)} := x - x^{(k)}$ *for a given* $x^{(k)} \in \mathbb{C}^n$*, we formulate the* residual equation:

$$Ae^{(k)} = r^{(k)}. \tag{3.6}$$

The problem of finding the error $e^{(k)}$ in eq. 3.6 is clearly equivalent to the problem of finding x in $Ax = b$, since $x = x^{(k)} + e^{(k)}$. Since the error will typically not be available, the quality of an approximation $x^{(k)}$ can only be measured via the residual $r^{(k)}$. Looking at eq. 3.6, we see that $\|e^{(k)}\| \leq \|A^{-1}\| \cdot \|r^{(k)}\|$. This shows that if $\|A^{-1}\|$ is large, the error can still be large, despite the residual being small. Many iterative methods follow the idea of updating the iteration vector $x^{(k)}$ in every step starting from an initial vector $x^{(0)}$ by approximating the error via

$$x^{(k+1)} := x^{(k)} + \tilde{e}^{(k)}. \tag{3.7}$$

where $\tilde{e}^{(k)}$ is an approximation of $e^{(k)}$ in eq. 3.6.

The following three sections present first two classes of iterative methods (relaxation schemes in sect. 3.1.2.1 and Krylov subspace methods in sect. 3.1.2.2), followed by the important concept of preconditioning (sect. 3.1.2.3). These are all very relevant and necessary in solving ill-conditioned systems of equations with large and sparse matrices.

3.1.2.1 Relaxation schemes

Starting with an initial approximation $x^{(0)}$, relaxation methods modify the components of the approximation, one or a few at a time and in a certain order, until convergence is reached. Each of these modifications, called relaxation steps, is aimed at annihilating one or a few components of the residual vector.

A typical way of obtaining different relaxation methods is via e.g. a splitting [34].

Definition 3.8.
Let A, M and N be three given matrices satisfying $A = M - N$. The pair of matrices M, N is a splitting of A, if M is nonsingular.

By using the previous splitting on the system of equations $Ax = b$, we can write $x = M^{-1}Nx + M^{-1}b$. The latter relation can be then taken to the relaxation method, via the splitting, like this

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b. \tag{3.8}$$

Different forms for the splitting M, N lead to different relaxation methods. One example is $M := D$ and $N := L + U$, which corresponds to the diagonal of A and the off-diagonal part of A, respectively. This first example gives us the *Jacobi method*. A second example of a possible splitting is $M := D - L$ and $N := U$, building up with this the *Gauss-Seidel method*. Considerations on the convergence of Jacobi and Gauss-Seidel can be found, again, in e.g. [34]. Gauss-Seidel typically converges faster than Jacobi (for regular splittings, but not necessarily always), but in turn Jacobi is highly parallelizable and Gauss-Seidel scales badly in a parallel setting.

Coloring comes to the rescue, enabling Gauss-Seidel to be able to perform well in a parallel setting. If the problem matrix A comes from e.g. a 2-dimensional rectangular equispaced lattice, and furthermore the matrix is such that the interaction is only between each site and its four nearest neighbors, then we can color the lattice in a red-black manner. With this we can decouple neighboring lattice sites and render Gauss-Seidel parallelizable. This is the *red-black Gauss-Seidel method*.

Opting for algorithms that operate on single-elements of a matrix is a waste of resources from the point of view of modern computer hardware. Thanks to features such as paging, cache and others [86], algorithms based on *blocks* can be used instead: for a matrix A, we can group sets of variables into *block variables*. By defining a block decomposition of A and compatible block vectors x and b:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,p} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p,1} & A_{p,2} & \cdots & A_{p,p} \end{bmatrix}, \; x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}, \; b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix} \tag{3.9}$$

where $A_{i,j} \in \mathbb{C}^{\ell_i \times \ell_j}$, $x_i \in \mathbb{C}^{\ell_i}$ and $b_i \in \mathbb{C}^{\ell_i}$ with $\sum_{i=1}^{p} \ell_i = n$, we can formulate block variants of Jacobi and Gauss-Seidel. In implementations of these algorithms, we can then make use of BLAS3 [87] fundamental operations to improve the performance from a computational point of view.

Defining the matrix $I_i \in \mathbb{C}^{n \times \ell_i}$ with the identity on the i-th block row and zero everywhere else as the canonical injection from the i-th block column $A_{\cdot,i}$ into A and I_i^T as the trivial injection from the i-th block row $A_{i,\cdot}$ into A, respectively,

we obtain the block inverse $A_{i,i}^{-1} := (I_i^T A I_i)^{-1}$. Then we can apply the same block-analog splittings to Jacobi and Gauss-Seidel, where D, L and U are block diagonal, lower block triangular and upper block triangular matrices, respectively. These block Jacobi and Gauss-Seidel methods are also termed the *additive* and *multiplicative* Schwarz Alternating Procedure (SAP) [88, 89], which is a *domain decomposition* method for discretized partial differential equations. SAP is a crucial building block for the two-level method that we introduce in sect. 3.3.1, so we give a description for these methods in alg. 3.1 and alg. 3.2.

Algorithm 3.1: Additive SAP (block Jacobi)

input: Matrix A with blocks $A_{i,j}$, right hand side b, initial guess $x^{(0)}$
output: Solution x

for $k = 0, 1, 2, \ldots$
  $r^{(k)} \leftarrow b - Ax^{(k)}$
  foreach diagonal block $A_{i,i}$ **do**
    $x^{(k)} \leftarrow x^{(k)} + I_i A_{i,i}^{-1} I_i^T r^{(k)}$
  $x^{(k+1)} \leftarrow x^{(k)}$

Algorithm 3.2: Multiplicative SAP (block Gauss-Seidel)

input: Matrix A with blocks $A_{i,j}$, right hand side b, initial guess $x^{(0)}$
output: Solution x

for $k = 0, 1, 2, \ldots$
  foreach diagonal block $A_{i,}$ **do**
    $r^{(k)} \leftarrow b - Ax^{(k)}$
    $x^{(k)} \leftarrow x^{(k)} + I_i A_{i,i}^{-1} I_i^T r^{(k)}$
  $x^{(k+1)} \leftarrow x^{(k)}$

From alg. 3.2 it is apparent that multiplicative SAP has to be performed sequentially, which is not the case for additive SAP (alg. 3.1). This makes additive SAP a natural choice in a parallel computing environment. However, coloring comes to the rescue again: we can decouple the sequential block solves of multiplicative SAP by using an appropriate coloring scheme leading to e.g. *red-black* SAP [24], which will be the method of choice for the multigrid method in sect. 3.3.1.

The segmentation in blocks via submatrices of A as in eq. 3.9 has a direct correspondence in terms of a domain decomposition of the underlying lattice. We now state these corresponding lattice blocks in a precise manner for the lattice of interest to us here.

Definition 3.9.

Asumme that $\{\mathcal{T}_1^0, ..., \mathcal{T}_{\ell_0}^0\}$ *is a partitioning of* $\{1, ..., N_t\}$ *into* ℓ_0 *blocks of consecutive time points,*

$$\mathcal{T}_j^0 = \{t_{j-1} + 1, ..., t_j\},\ j = 1, ..., \ell_0,\ 0 = t_0 < t_1 < ... < t_{\ell_0} = N_t$$

and similarly for the spatial dimensions with partitionings $\{\mathcal{T}_1^\mu, ..., \mathcal{T}_{\ell_\mu}^\mu\}$, $\mu = 1, 2, 3$.

A block decomposition *of* $\mathcal{L}$ *is a partitioning of* $\mathcal{L}$ *into* $\ell = \ell_0\ell_1\ell_2\ell_3$ *lattice blocks* $\mathcal{L}_i$*, where each lattice block is of the form*

$$\mathcal{L}_i = \mathcal{T}_{j_0(i)}^0 \times \mathcal{T}_{j_1(i)}^1 \times \mathcal{T}_{j_2(i)}^2 \times \mathcal{T}_{j_3(i)}^3$$

Accordingly we define a block decomposition of all $12n_{\mathcal{L}}$ *variables in* $\mathcal{V} = \mathcal{L} \times \mathcal{C} \times \mathcal{S}$ *into* ℓ *blocks* $\mathcal{V}_i$ *by grouping all spin and color components corresponding to the lattice block* $\mathcal{L}_i$*, i.e.,*

$$\mathcal{V}_i = \mathcal{L}_i \times \mathcal{C} \times \mathcal{S}$$

Another block decomposition $\{\mathcal{L}_i' : i = 1, ..., t'\}$ *is called* refinement *of* $\{\mathcal{L}_i : i = 1, ..., t\}$ *if for each* $\mathcal{L}_i'$ *there exists a* $\mathcal{L}_j$ *such that*

$$\mathcal{L}_i' \subseteq \mathcal{L}_j$$

3.1.2.2 Krylov subspace methods

Let us start off by defining a *Krylov subspace.*

Definition 3.10.
Let $A \in \mathbb{C}^{n\times n}$ *and* $r \in \mathbb{C}^n$*. Then the* m-th Krylov subspace *is defined as*

$$\mathcal{K}_m(A, r) := \text{span}\{r, Ar, A^2r, \ldots, A^{m-1}r\}.$$

If unambiguous we use the shorthand $\mathcal{K}_m$*.*

Krylov subspace methods [34] only require matrix-vector multiplications and have modest storage requirements and are thus favorable in cases when what is available is a function receiving as input a vector x and returning as output the application of a matrix A on that vector i.e. Ax.

There are many Krylov-based methods. We focus for now on the generalized minimal residual method (GMRES). GMRES is an iterative method aimed at solving large non-symmetric linear systems of equations represented as $Ax = b$

with some nonsingular matrix $A \in \mathbb{C}^{n\times n}$ and $x, b \in \mathbb{C}^n$. It is a Krylov subspace method i.e. it searches for an approximate solution $x \in x^{(0)} + \mathcal{K}_m(A, r^{(0)})$ with $x^{(0)}$ and $r^{(0)}$ being, correspondingly, the initial guess and the initial residual [90]. Another interesting way to reformulate this statement is that the approximate solution can be written as a polynomial expression:

$$x = x^{(0)} + \phi(A)r^{(0)}, \tag{3.10}$$

where $\phi(\alpha)$ is a polynomial of degree at most $k-1$ in α. The main feature that characterizes GMRES and distinguishes it from other algorithms such as the Full Orthogonalization Method (FOM) [91] is that it draws the approximate solution x from $x^{(0)} + \mathcal{K}_k(A, r^{(0)})$ by minimizing the 2-norm of the residual. Taking this into account, it is then possible to rephrase GMRES as a polynomial optimization method. Denoting $\mathbb{P}_k$ as the set of all polynomials of degree k:

$$\begin{aligned}\min_{x\in x^{(0)}+\mathcal{K}_k(A,r^{(0)})} \|b - Ax\|_2 &= \min_{\phi\in\mathbb{P}_{k-1}} \|b - A(x^{(0)} + \phi(A)r^{(0)})\|_2 \\ &= \min_{\phi\in\mathbb{P}_{k-1}} \|(I - A\cdot\phi(A))r^{(0)}\|_2 \\ &= \min_{\pi\in\mathbb{P}_k,\pi(0)=1} \|\pi(A)r^{(0)}\|_2.\end{aligned} \tag{3.11}$$

where the polynomial π, usually called GMRES polynomial, is defined as $\pi(\alpha) = 1 - \alpha\phi(\alpha)$ and it is such that it minimizes the residual 2-norm of $\pi(A)r^{(0)}$ within the polynomial space $\mathbb{P}_k$ [83]. The internal working of GMRES go briefly as follows: first an orthonormal basis for $\mathcal{K}_k(A, r^{(0)})$ is constructed by means of the Arnoldi process [92]:

Algorithm 3.3: Arnoldi process

Data: v_1, such that $\|v_1\|_2 = 1$
Result: Set of vectors V_k, and Hessenberg matrix $\bar{H}_k$.

$r^{(0)} = b - Ax^{(0)}$
$\beta = \|r^{(0)}\|_2$
$v_1 = r^{(0)}/\beta$
for $j = 1, \ldots, k$
  for $i = 1, \ldots, j$
    $h_{i,j} = (Av_j, v_i)$
    $w_j = w_j - h_{i,j}v_i$
  $h_{j+1,j} = \|w_j\|_2$
  If $h_{j+1,j} = 0$ then STOP
  $v_{j+1} = w_j/h_{j+1,j}$

which results in the set of orthonormal vectors $V_k = [v_1, v_2, \cdots, v_k]$ i.e. $V_k \in \mathbb{C}^{n\times k}$ is orthonormal. The orthogonalization generates scalars $h_{ij} \in \mathbb{C}$ which when arranged as a Hessenberg matrix $H_k = (h_{ij}) \in \mathbb{C}^{k\times k}$ satisfy the following recurrent relation

$$AV_k = V_{k+1}\bar{H}_k, \tag{3.12}$$

where $\bar{H}_k$ corresponds to H_k with an extra row $(\ 0\ 0\ \cdots\ h_{k+1,k}\ )$ at the bottom. The next step consists of writing the approximate solution in terms of V_k:

$$x = x^{(0)} + V_k y,$$

where y is a k-vector resulting from the minimization of the function $J(y)$ defined as

$$J(y) = \|\beta e_1 - \bar{H}_k y\|_2,$$

with e_1 an m-vector of the form $e_1 = (\ 1\ 0\ 0\ \cdots\ 0)^H$ and β the norm of $r^{(0)}$ [34].

A major drawback of GMRES is that because it is based on the Arnoldi process, the computational work and memory required increase with each iteration (i.e. they grow, respectively, as $\mathcal{O}(n \cdot k^2)$ and $\mathcal{O}(nk)$ at the k-th iteration). Therefore, for very large systems, accessing a satisfactory number of iterations with GMRES may quickly become prohibitive. To circumvent this difficulty, restarted GMRES (also denoted GMRES(m)) was proposed in [90]. This approach consists of restarting the orthonormal base V_k for the Krylov subspace every time it reaches a maximum number m of vectors, where m is small compared to n and is chosen in such a way that memory and computational costs become manageable [93]. The idea is that each new cycle uses as the initial guess the approximate solution obtained on the previous restart (the first cycle starts with the original proposal $x^{(0)}$). This means that the residuals from consecutive cycles will be related through $r^{(c)} = \pi_c(A)r^{(c-1)}$, where $r^{(c)}$ and $r^{(c-1)}$ are, respectively, the residuals of the cycles c and $c-1$, and π_c is the c-clycle GMRES polynomial. In terms of $r^{(0)}$, $r^{(c)}$ can be expressed as $r^{(c)} = \Pi_c(A)r^{(0)}$, where $\Pi_c(A) = \pi_c \cdots \pi_1$, a polynomial of degree $c \cdot m$. From here it can be seen that the Krylov subspace corresponding to the $(c+1)$-st cycle is

$$\mathcal{K}_m(A, \Pi_c(A)r^{(0)}) = \text{span}\{\Pi_c(A)r^{(0)}, A\Pi_c(A)r^{(0)}, \ldots, A^{m-1}\Pi_c(A)r^{(0)}\}, \tag{3.13}$$

and the approximate solution that is drawn from it has the form:

$$x = x^{(0)} + \phi_{c+1}(A)\Pi_c(A)r^{(0)}. \tag{3.14}$$

The convenience of restarted GMRES comes with a subtle cost: the robustness of the method gets compromised in the sense that there is no preserved orthogonality between the subspaces constructed in consecutive cycles. This comes with the negative side effect that restarted GMRES generally converges more slowly than GMRES and in fact, may even stagnate [94]. In such scenarios preconditioning might become useful or even necessary.

3.1.2.3 Preconditioning

As described earlier, the convergence of iterative methods oftentimes depends on the condition number $\kappa(A)$ of the system matrix A. In the particular case of GMRES, the distribution of eigenvalues is also of importance for convergence (among other factors e.g. contidioning of the matrix of eigenvectors $\kappa(X)$ in the decomposition $A = X\Lambda X^{-1}$, provided such a decomposition exists [34]). More specifically, if some subsets of eigenvalues are clustered too close to each other, the GMRES polynomial might have a hard time interpolating over them. Furthermore, unpredictable and even paradoxical behavior can be seen when using restarted GMRES [95].

The idea of *preconditioning* is to reduce the condition number by transforming the problem to an equivalent one with a smaller condition number and possibly a more scattered spectrum. In general we are interested in a matrix M which is in some way close to A^{-1}, such that

$$1 = \kappa(I) \approx \kappa(MA) \ll \kappa(A).$$

We define *left preconditioning* via

$$Ax = b \Leftrightarrow MAx = Mb,$$

and *right preconditioning* via

$$Ax = b \Leftrightarrow AMy = b,$$

where $x = My$. As a consequence, in preconditioned methods every matrix vector multiplication also requires the application of the preconditioner. Thus from a practical point of view the application of M needs to be significantly cheaper compared to the solution of linear systems with A, since they are applied in every iteration, but should still be "close enough" to A^{-1} to have a notable impact on the condition number.

When a non-stationary[15] preconditioner is to be employed in conjunction with GMRES, the relation in eq. 3.12 does not hold in general anymore, and switching to a flexible method such as FGMRES is necessary. The FGMRES algorithm is presented in alg. 3.4 [34].

Algorithm 3.4: Flexible GMRES (FGMRES)

Data: Initial guess $x^{(0)}$.
Result: Sets of vectors Z_m and V_{m+1}, and Hessenberg matrix $\bar{H}_m$.

1 $r^{(0)} = b - Ax^{(0)}$
2 $\beta = \|r^{(0)}\|_2$
3 $v_1 = r^{(0)}/\beta$
4 **for** $j = 1, \cdots, m$
5 $\quad z_j = M_j^{-1} v_j$
6 $\quad w = Az_j$
7 $\quad$ **for** $i = 1, \cdots, j$
8 $\quad\quad h_{i,j} = (w, v_i)$
9 $\quad\quad w = w - h_{i,j} v_i$
10 $\quad h_{j+1,j} = \|w\|_2$
11 $\quad v_{j+1} = w/h_{j+1,j}$
12 $\quad$ Define : $Z_m := [z_1, ..., z_m]$
13 $\quad$ Define : $\bar{H}_m := \{h_{i,j}\}_{1 \le i \le j+1; 1 \le j \le m}$
14 $y_m = \texttt{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$
15 $x_m = x^{(0)} + Z_m y$
16 If satisfied, stop. Else, set $x^{(0)} \leftarrow x_m$ and GOTO 1.

In the case of FGMRES, the relation in eq. 3.12 takes the following form:

$$AZ_k = V_{k+1} \bar{H}_k, \tag{3.15}$$

which still allows us to minimize the norm of the residual (see line 14 in alg. 3.4) in a cheap fashion. An obvious drawback of FGMRES versus plain GMRES is that not only the V_{m+1} vectors need to be saved but also the Z_m vectors.

3.2 Multigrid methods

The iterative methods introduced in sect. 3.1.2 i.e. relaxation and Krylov-based solvers, all suffer from a common complication: the larger the condition number

[15] See e.g. [34] for the difference between stationary and non-stationary methods. Stationary methods come from splittings as defined in def. 3.8. Jacobi and Gauss-Seidel classify as stationary, whereas GMRES is considered a non-stationary method.

$\kappa(A)$ of the matrix of coefficients A of the system of equations, the larger is typically the *iteration count* for the solver to reach a certain relative tolerance. A solver is said to suffer from *critical slowing down* if, as $\kappa(A)$ grows, that iteration count increases as well[16], with the variation of the iteration count as a function of $\kappa(A)$ depending on the solver of choice. In many scientific computing applications, the larger A the larger $\kappa(A)$, which increases the difficulty of the problem two-fold: the larger the matrix the more computationally expensive the matrix-vector multiplications with it will be. Furthermore, the larger $\kappa(A)$ is, the more iterations it takes to solve the linear system of equations with those methods from sect. 3.1.2. If A grows in size, it is of course necessary to invest more computational work to apply that matrix on a vector, this is unavoidable. The question then is: can we have a solver for which, if $\kappa(A)$ increases, its iteration count does not? Here is where multigrid methods become remarkably useful.

Extensive presentations of multigrid methods can be found in e.g. [97, 98]. Our presentation here is, in turn, rather brief, and it follows mostly [80].

3.2.1 Motivation

Before tackling complicated systems such as the Schwinger matrix in lattice QED and the Dirac matrix in lattice QCD, let us first introduce a simpler model in order to be able to motivate multigrid in a simpler manner.

The Poisson equation $\nabla^2\phi = f$ is commonly found in many different areas of physics. For example, in electrostatics [99] it is used to describe the scalar potential created by a distribution of charge over space and in Newtonian gravity [100], the gravitational potential can be computed from a matter source. When discretized on a lattice, the Poisson equation serves as a good setting in which to test different algorithms and their implementations.

Let us consider a two-dimensional setting:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x,y), \quad 0 < x < 1, \quad 0 < y < 1 \tag{3.16}$$

with $u = 0$ on the boundary of the unit square. After applying second-order finite differences, we end up with a system of equations $Ax = b$ with[17]

[16] Critical slowing down can also appear e.g. in the context of Markov Chain Monte Carlo, where it is seen as problematic in the sense of a rise in the autocorrelation time [96].

[17] See e.g. [80] for a description on how this discretization is done. The motivation described in this section was partially based on that same reference.

$$A = \begin{pmatrix} B & -I & & \\ -I & B & -I & \\ . & . & . & \\ & . & . & -I \\ & & -I & B \end{pmatrix} \tag{3.17}$$

where

$$B = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & . & . & . & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix} \tag{3.18}$$

Due to the discretization scheme chosen, each variable is coupled to nearest neighbors only. We call this the *Laplace 2D* problem.

Let us choose one of the solvers from sect. 3.1.2 and represent the application of ν iterations of it by $M^{(\nu)}$, i.e. $M^{(\nu)}$ approximates A^{-1}. Starting from an initial guess $x^{(0)}$, the initial residual is $r^{(0)} = b - Ax^{(0)}$. Then, the residual equation (see eq. 3.6) which takes the form of $Ae = r$ can be roughly solved by applying $M^{(\nu)}$ which results in an approximant of the form

$$x \approx x^{(0)} + M^{(\nu)} r^{(0)}. \tag{3.19}$$

We can re-arrange eq. 3.19 in the following way

$$x \approx x^{(0)} + M^{(\nu)} r^{(0)} = x^{(0)} + M^{(\nu)}(b - Ax^{(0)}) = M^{(\nu)} b + (I - M^{(\nu)} A)x^{(0)}. \tag{3.20}$$

If, after a certain number of iterations ν the operator $M^{(\nu)}$ is equal to A^{-1} up to some accuracy, then the system of equations has been solved. The term in parenthesis in the far-right of eq. 3.20 is termed the *error propagator* of the solver used

$$E_s := (I - M^{(\nu)} A). \tag{3.21}$$

The error propagator gives us an indication of the quality of a solver in terms of number of iterations to solution, and its analysis with regard to the spectrum of A is of upmost importance in the development of fastly convergent algorithms. If we write successive iterations of $M^{(\nu)}$ as $x^{(k+1)} = Mb + E_s x^{(k)}$ (with M corresponding to $M^{(\nu)}$ with $\nu = 1$), and if the method is convergent to the actual solution, after many iterations we will get $x = Mb + E_s x$, and by combining both relations we

can write $e^{(k+1)} = E_s e^{(k)}$ with the error $e^{(k)} = x - x^{(k)}$. The error propagator, then, gives us a quantification on the reduction of the error.

An interesting observation can be made at this point: let us choose for $x^{(0)}$ a random initial guess. With this initial guess, and for the actual solution x, the error $e^{(0)} = x - x^{(0)}$ can be written in terms of a spectral decomposition (provided this decomposition exists, which is the case in the Laplace 2D example that we are discussing here).

$$e^{(0)} = \sum_{i=1}^{N} c_i v_i \tag{3.22}$$

where v_i are the eigenvectors of A and c_i are just coefficients of the decomposition. The eigenvectors v_i to associated large eigenvalues λ_i in eq. 3.22 are also known as *high frequency modes* and the ones corresponding to small eigenvalues are known as *low frequency modes*. Therefore, we can re-write eq. 3.22 as

$$e^{(0)} = e^{(0)}_{low} + e^{(0)}_{high}. \tag{3.23}$$

We can at this point run a few iterations of the methods introduced in sect. 3.1.2, in particular relaxation schemes, and come to some important realizations in terms of the components of the error that are tackled better by them. Let us choose, in particular, the Gauss-Seidel method, which can be applied to the Laplace 2D problem described before, results of which are displayed in fig. 3.1.

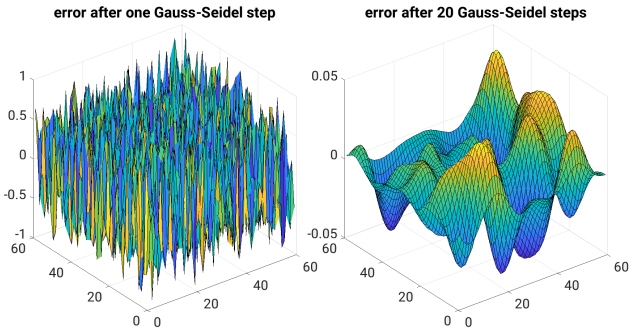

Figure 3.1: Error $e^{(k)}$ of the Gauss-Seidel method when applied to the Laplace 2D problem with random initial guess $x^{(0)}$ and $k = 1$ iterations for the left plot and $k = 20$ iterations for the right plot.

As can be seen from fig. 3.1, it is visually clear that high-frequency components of the error have been removed, i.e. parts of the error that oscillate rapidly have been reduced. This is typical of relaxation methods: a few iterations of them lead to a quick removal of high-frequency components of the error. But, numerical tests

indicate that after a few iterations the reduction of the error starts to stagnate and more iterations of the solver lead to almost no reduction in the error. In terms of eq. 3.23, these methods are good are removing e_{high} but quite bad when it comes to dealing with e_{low}. Basically the same observations can be concluded when using a Krylov-based method e.g. GMRES instead of Gauss-Seidel as we did before. These methods are then good *smoothers* i.e. they are good at smoothing the error by quickly reducing e_{high} with just a few iterations.

Although they are good at smoothing the error quickly, they continue to be bad solvers due to their limited action on e_{low}. We would like, though, to keep using those first few iterations of those methods which are good at smoothing, and complement them with something that deals with e_{low} efficiently. Here is where having more than one grid becomes beneficial [80].

3.2.2 Two levels and multilevel multigrid

The error in fig. 3.1 has been smoothed down. At that point, we can create a coarser lattice i.e. one with a larger lattice spacing and thus with less grid points and represent the whole system there, which implies representing the error in the right plot in fig. 3.1 on that coarser lattice as well. This takes us to a two-grid correction scheme [80] (the quantities with a bar, e.g. $\bar{x}$, correspond to variables on the coarse grid):

1. Apply ν_1 iterations of the *smoother* (e.g. Gauss-Seidel, Jacobi, GMRES) on $Ax = b$ at the fine grid, with initial guess $x^{(0)}$, obtaining with this a first approximant $x^{(a)}$.
2. Compute the residual, still at the fine grid: $r^{(a)} = b - Ax^{(a)}$.
3. Transport this residual to the coarse grid: $r^{(a)} \to \bar{r}^{(a)}$.
4. Solve $\bar{A}\bar{e} = \bar{r}^{(a)}$ at the coarse grid, to obtain $\bar{e}^{(a)}$.
5. Transport the error back to the fine grid: $\bar{e}^{(a)} \to e^{(a)}$.
6. Perform a correction step at the fine grid: $x^{(b)} = x^{(a)} + e^{(a)}$.
7. Apply ν_2 iterations of the *smoother* on $Ax = b$ at the fine grid, with initial guess $x^{(b)}$, obtaining with this a new approximant $x^{(c)}$.

Points 1 and 7 in the two-grid scheme above remove e_{high}, while points 2 to 6 are meant to efficiently deal with e_{low}. Clearly, to have a good reduction of e_{low} the operators for transporting from the fine to the coarse grid and viceversa need to be carefully chosen, as well as the construction of $\bar{A}$.

Let us be a bit more precise now and make the following associations: the operator in charge of transporting from the fine to the coarse grid is named the *restrictor* R, the one transporting from the coarse to the fine grid is the *prolongator* (or *interpolator*) P, and for the construction of the coarse-grid matrix $\bar{A}$ we use here the Petrov-Galerkin approach

$$\bar{A} = RAP. \tag{3.24}$$

Following the points above from the two-grid scheme and a re-arrangement similar to the one in eq. 3.20 we can write the error propagator for the coarse-grid correction as follows

$$E_c := I - P\bar{A}^{-1}RA. \tag{3.25}$$

Although the error in fig. 3.1 looks "smooth" in the fine grid employed, it will become more oscillatory when transported to the coarse grid i.e. some portions of e_{low} in the finer grid will be seen as high-frequency components from the point of view of the coarser one i.e. $e_{low} \rightarrow \bar{e}_{high} + \bar{e}_{low}$. This clearly opens up the possibility of a multigrid hierarchy with more than two levels, where $\bar{e}_{high}$ can be treated via a smoother at the coarse grid, and then a new third level has to be created to correct for that part of the error not efficiently removed by a few iterations of the smoother i.e. $\bar{e}_{low}$. Illustration and a precise description of schemes with more than two levels are held until later sections and discusssed in the context of algebraic multigrid.

The success of the coarse-grid correction at every level depends then on an appropriate choice for both P and R. Very efficient operators P and R have been succesfully constructed in many applications appearing in scientific computing [101–103], such that the smoother and the coarse-grid correction complement each other very well to the point that critical slowing down does not seem to affect the solver. We will describe one such construction in upcoming sections in the context of lattice QCD.

3.3 Algebraic multigrid

As described in sect. 3.2.2, an appropriate construction of P and R, in order to efficiently deal with e_{low}, is of upmost importance in multigrid methods to have an algorithm as independent of $\kappa(A)$ as possible. An obvious way to construct P is the one illustrated in fig. 3.2, corresponding to a linear interpolation from the coarse to the fine grid when dealing with a one-dimensional lattice. This is clearly a *geometric multigrid* approach i.e. it is solely based on the geometry of the

lattice, regardless of the entries of the matrix A of the linear system of equations. Evidently, other choices for P are possible when using geometric multigrid, e.g. *constant*, *cubic*, etc.

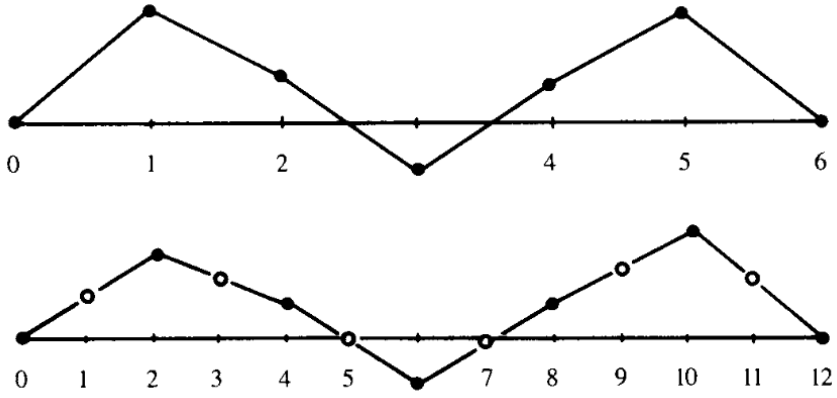

Figure 3.2: Linear interpolation of a vector on the coarse grid to the fine grid in a one-dimensional lattice. Image taken from [80].

In matrices arising from the discretization of PDEs with smooth coefficients, the lowest modes of A can be closely connected to geometric considerations i.e. to the associated lattice. The lowest modes, as we have suggested before, are important in multigrid to properly deal with e_{low} i.e. it is important that the fine-grid matrix A and the coarse-grid matrix A_c are similar to each other in the sense of their lowest modes. This all indicates that, in a geometric multigrid construction we can build P from geometric considerations and have the reductions of e_{low} that we want.

When A contains random entries, the associations from geometry to low modes is generally not possible, and the construction of P is then based on the information of A itself rather than on the geometry of the underlying lattice. This leads to general and powerful methods anyway, as not all matrices are generated from the discretization of a partial differential equation on a grid. A multigrid construction where P is constructed based on properties of A itself regardless of the lattice is known as *algebraic multigrid*, which we describe now in the context of lattice QCD.

All the developments discussed in the next three chapters revolve around the multigrid method DD-αAMG (Domain decomposition aggregation-based αdaptive algebraic multigrid method [24, 45]), which can be considered as both a code package and a conceptual framework for solving linear systems of equations involving the Dirac matrix.

The sects. 3.3.1, 3.3.2 and 3.3.3 establish, in increasing level of complexity, the main ingredients of DD-αAMG, and we introduce there some formal statements as well as some more heuristic motivations on the inner workings of this multigrid method. Building on this, sect. 3.3.4 presents DD-αAMG in a more compact way, and introduces the important cycling strategy known as K-cycles.

In chap. 6 we will use the DD-αAMG method to construct our own algebraic multigrid solver in the context of lattice QED i.e. an AMG solver built for the Schwinger operator described in sect. 2.5.

3.3.1 Algebraic multigrid in lattice QCD

As described in chapter 2, the Dirac matrix appearing in lattice QCD simulations is a function of the gauge links (which are an indirect representation, on the lattice, of the gauge bosons from QCD in the continuum). The evaluation of the path integral, as also described in that chapter, is performed via a Monte Carlo procedure, which leads to the gauge links being random, rendering with this the Dirac operator D to be a matrix with random entries. This randomness suggests already that algebraic rather than geometric multigrid is the method of choice to solve the linear equations appearing in lattice QCD.

As presented in chapter 2, there are different approaches to discretizing the Dirac operator. These different approaches lead to various lattice discretizations and therefore to different forms of the operator D e.g. Wilson, Twisted Mass, Staggered, and others [5, 43, 68, 69]. In lattice QCD, adaptive (algebraic) multigrid methods have been established as the most efficient methods for solving linear systems involving the Wilson Dirac operator [18, 24, 104–106], and they have also found their way into other lattice QCD discretizations (e.g. twisted mass [27], staggered [19] and Domain-Wall [20] fermions). They demonstrate significant speedups compared to conventional Krylov subspace methods, achieving orders of magnitude faster convergence and insensitivity to conditioning.

Let us adapt now our notation in order to fit better the one typically used in lattice QCD discussions of linear solvers (see e.g. [45]).

Definition 3.11.
Let the linear system of equations $D\psi = \eta$, $n = 12n_{\mathcal{L}}$ and $n_c < n$ be given. Assuming we have full rank linear restriction *and* prolongation/interpolation *operators*

$$R : \mathbb{C}^n \to \mathbb{C}^{n_c},$$
$$P : \mathbb{C}^{n_c} \to \mathbb{C}^n$$

we define a Petrov-Galerkin projection of D, i.e., the coarse grid operator

$$D_c := RDP, \tag{3.26}$$

and the corresponding coarse grid correction

$$\psi \leftarrow \psi + PD_c^{-1}Rr$$

with $r = \eta - D\psi$.

The relations in def. 3.11 are, of course, in agreement with our previous introduction of a two-grid scheme, see eqs. 3.24 and 3.25. The coarse grid correction for a current iterate ψ restricts the current residual r via R to the subspace, where we solve

$$D_c e_c = Rr \tag{3.27}$$

and transporting the coarse error e_c via P back to the original space as a correction for ψ. In eq. 3.20, one step of coarse grid correction can be summarized as

$$\psi \leftarrow (I - PD_c^{-1}RD)\psi + PD_c^{-1}R\eta \tag{3.28}$$

with the associated coarse grid error propagation operator (see eqs. 3.21 and 3.25) being

$$E = I - PD_c^{-1}RD \tag{3.29}$$

The operator in eq. 3.29 is a projector. The goal is for this projector to easily remove low modes, with the high modes being treated by the smoother. We define now the subspaces relevant to the construction of a good coarse grid correction in eq. 3.29.

Definition 3.12.
Let us define the near kernel *as the space spanned by the right eigenvectors belonging to small (in modulus) eigenvalues of D. By* near right kernel *we mean*

the same as near kernel, and we define near left kernel *as the subspace spanned by small left eigenvectors.*

As discussed in [24], by choosing range(P) to approximately contain the near right kernel and range(R) the near left kernel, we have then a good complement between the two-grid correction in eq. 3.28 and the smoother. A rather geometric interpretation of the effect of E on different components of the error, and how this can impose conditions on P and R, can be found in chapter 5 of [80].

We state now a two-level multigrid method, with the same structure as in sect. 3.2.2 but with the newly introduced notation of this section. Based on numerical experiments [45], a good choice for the smoother is multiplicative SAP (see alg. 3.2) with red-black block coloring. We use here $M_{SAP}^{(\nu)}$ to represent this SAP smoother, and in particular this two-grid scheme uses ν steps of post-smoothing only.

Algorithm 3.5: Two-level V-cycle with post-smoothing

input: ψ, η, ν
output: ψ
1 $r \leftarrow \eta - D\psi$
2 $\psi \leftarrow \psi + PD_c^{-1}Rr$
3 $r \leftarrow \eta - D\psi$
4 $\psi \leftarrow \psi + M_{SAP}^{(\nu)} r$

Alg. 3.5 can be recursively extended to a true multigrid method by recursively calling it every time the coarse-grid solve needs to be performed. As discussed in sect. 3.2.2, going from fine to coarse grid is motivated by the decomposition $e_{low} \rightarrow \bar{e}_{high} + \bar{e}_{low}$, which is at the basis of this recursion. In LQCD, the multigrid hierarchy consists of two or three levels, rendering the coarsest level still quite large and difficult to solve in some cases. A more thorough discussion of these aspects and the need for a good coarsest-level solver are presented in chapter 4.

3.3.2 Aggregation-based prolongation and restriction

An important point concluded in sect. 3.3.1 motivates the construction of R in terms of P, namely, that range(R) needs to be spanned by the near left kernel of D in order to have a good complement between smoother and coarse grid correction. Furthermore, the construction of P itself is motivated by the phenomenon of *local coherence*.

Local coherence comes from the observation in [11] that eigenvectors belonging to small eigenvalues of D tend to (approximately) coincide on a large number of lattice blocks $\mathcal{L}_i$ (see def. 3.9). More specifically, local coherence means that we can

approximately represent many eigenvectors belonging to small eigenvalues from just a few by decomposing them into the parts belonging to each of the lattice blocks[18]. Local coherence is the conceptual base for aggregation-based interpolation as constructed in [107, 108] for general cases. It is of upmost importance in lattice QCD computations [18, 104, 105]. The idea of taking a few eigenvectors and being able to generate many small eigenvectors from them resonates with the properties of P discussed before. Hence, we state it again: range(P) should approximately contain the near kernel of D due to $e_{low} \in \text{range}(P)$. We know then that, provided that local coherence holds, we can then compute a small set of small eigenvectors and then apply the decomposition as suggested by local coherence such that we obtain as many vectors as necessary in order to construct P. Of course, it is better to have a sparse form for P, as done in the following construction.

Definition 3.13.
An aggregation $\{\mathcal{A}_1, ..., \mathcal{A}_s\}$ *is a partitioning of the set* $\mathcal{V} = \mathcal{L} \times \mathcal{C} \times \mathcal{S}$ *of all variables (see def. 3.9). It is termed a* lattice-block-based *aggregation if each* $\mathcal{A}_i$ *is of the form*

$$\mathcal{A}_i := \mathcal{L}_{j(i)} \times \mathcal{W}_i$$

where $\mathcal{L}_j$ *are the lattice blocks of a block decomposition of the lattice* $\mathcal{L}$ *and* $\mathcal{W}_i \subseteq \mathcal{C} \times \mathcal{S}$.

The key difference between the decomposition based on lattice blocks as introduced in def. 3.9 and the one from def. 3.13 is that an aggregate does not have to contain all spin and color variables, which implies that a lattice block $\mathcal{L}_i$ can be associated with more than one aggregate.

By combining the notion of local coherence with this newly introduced concept of aggregates, we can specify the construction of the interpolation operator as follows:

Definition 3.14.
Consider a set $\{v_1, \ldots, v_N\} \subseteq \mathbb{C}^n$ *of so-called* test vectors *representing the near kernel and a set of aggregates* $\{\mathcal{A}_1, \ldots, \mathcal{A}_s\}$. *The interpolation operator* P *is then defined by decomposing the test vectors over the aggregates as in fig. 3.3.*

Formally, each aggregate $\mathcal{A}_i$ *induces* N *variables* $(i-1)N+1, ..., iN$ *on the coarse system, and we define*

$$Pe_{(i-1)N+j} := \mathcal{I}_{\mathcal{A}_i} \mathcal{I}_{\mathcal{A}_i}^H v_j, \quad \text{for } i = 1, \ldots, s,\ j = 1, \ldots, N \tag{3.30}$$

[18] See [11] for a full qualitative analysis of this phenomenon.

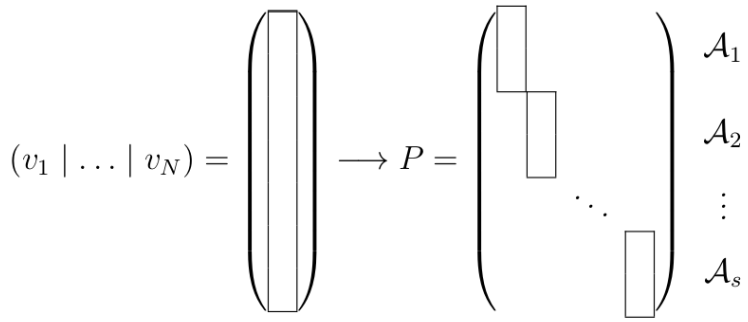

Figure 3.3: Construction of P from the decomposition, based on local coherence, of a few vectors from the near kernel of the Dirac operator. Image taken from [45].

where $\mathcal{I}^H_{\mathcal{A}_i}$ represents the trivial restriction operator for the aggregate $\mathcal{A}_i$, i.e., $\mathcal{I}_{\mathcal{A}_i}\mathcal{I}^H_{\mathcal{A}_i} v_j$ leaves the components of v_j from $\mathcal{A}_i$ unchanged while zeroing all others.

To avoid stability issues, after constructing P as in def. 3.14 the columns of P are then orthonormalized locally[19]. This construction of a sparse interpolation operator is such that each aggregate can be associated to a single site on the coarse lattice, and the sparsity and nearest-neighbor structure of $D_c = RDP$ resembles the one of the Dirac operator at the fine grid.

In a two-level hierarchy, the degrees of freedom per lattice site at the finest level is 12, due to a direct discretization of continuum QCD on the lattice. The number of degrees of freedom per lattice site at the coarse grid will depend on the choice for the aggregates, e.g. if we choose the aggregates such that each lattice block is associated to one aggregate only, then N is the number of degrees of freedom per lattice site in the coarse lattice.

3.3.3 Petrov-Galerkin approach

We describe now a more precise construction of R as well as a more precise definition of the aggregates to be considered from here onwards. In sect. 3.3.2, we have concluded that range(R) needs to be spanned by the smallest left eigenvectors of D. This, in combination with the relation between right and left eigenvectors of D (see lemma 2.10), leads to the natural choice

$$R = (\Gamma_5 P)^H. \tag{3.31}$$

[19]For more details on this orthonormalization procedure and more details on the efficient application of P on vectors, see [45].

However, as pointed out in [104], it seems desirable to have $R = P^H$ by taking the special spin-structure of the Dirac operator into account when defining the aggregates.

Definition 3.15.
The aggregation $\{\mathcal{A}_i : i = 1, ..., s\}$ *is termed* Γ_5-compatible *if any given aggregate* $\mathcal{A}_i$ *is composed exclusively of fine variables with spin 0 and 1 or of fine variables with spin 2 and 3.*

From def. 3.14 and assuming we choose to have a Γ_5-compatible aggregation, we can show that

$$\Gamma_5 P = P\Gamma_5^c.$$

where Γ_5^c acts as the identity on aggregates with spins 0 or 1, and as negative identity for aggregates with spins 2 or 3.

The following lemma is the last pillar in deciding the kind of aggregates to be used, as well as the relation between R and P.

Lemma 3.16.
Let the aggregation be Γ_5*-compatible and* P *a corresponding aggregation-based prolongation (see def. 3.14) and* $R = (\Gamma_5 P)^H$*. Consider the two coarse grid operators*

$$D_c^{PG} = RDP \qquad \textit{and} \qquad D_c = P^H DP$$

Then

(i) $D_c = \Gamma_5^c D_c^{PG}$.

(ii) $I - PD_c^{-1}P^H D = I - P(D_c^{PG})^{-1}RD$.

(iii) D_c^{PG} *is Hermitian,* D_c *is* Γ_5^c*-symmetric.*

(iv) For the field of values $\mathcal{F}(D) := \{\psi^H D\psi : \psi^H\psi = 1\}$*, we have* $\mathcal{F}(D_c) \subseteq \mathcal{F}(D)$.

Proof. A proof of this lemma can be found in e.g. [45].

Therefore, choosing D_c over D_c^{PG} seems to be a "better" choice in the sense that it gives us coarser representations of the Dirac operator that are closer in their properties to the finest-level one, rendering a cleaner recursive construction of the

multigrid hierarchy, and with some extra advantages that have been found from numerical experiments[20].

Now that we have completely specified the details on the construction of P and how R relates to P, as well as the construction of the coarse matrix D_c, the last missing piece is a more detailed specification on how the aggregates are realized (this is, of course, Γ_5-compatible).

Definition 3.17.
Let $\{\mathcal{L}_j : j = 1, ..., n_{\mathcal{L}_c}\}$ *be a block decomposition of the lattice* $\mathcal{L}$*. Then the* standard aggregation $\{\mathcal{A}_{j,\tau} : j = 1, ..., n_{\mathcal{L}_c}, \tau = 0, 1\}$ *with respect to this block decomposition is given by*

$$\mathcal{A}_{j,0} := \mathcal{L}_j \times \{0, 1\} \times \mathcal{C} \quad \textit{and} \quad \mathcal{A}_{j,0} := \mathcal{L}_j \times \{2, 3\} \times \mathcal{C}$$

The standard aggregates induce a coarse lattice $\mathcal{L}_c$ with $n_{\mathcal{L}_c}$ sites where each coarse lattice site corresponds to one lattice block $\mathcal{L}_j$ and holds $2N$ variables with N the number of test vectors. Hence, the overall system size of the coarse system is $n_c = 2Nn_{\mathcal{L}_c}$. With standard aggregation, $D_c = P^H DP$ is such that the coarse lattice points can be arranged as a 4D periodic lattice and the system represents a nearest neighbor coupling on the torus.

3.3.4 Domain decomposition aggregation-based αdaptive algebraic multigrid method

With all the ingredients covered in sects. 3.3.1, 3.3.2 and 3.3.3, we can go ahead now and describe the DD-αAMG method. The description here will be at a rather superficial level i.e. we will state the algorithms involved in the method and omit further deeper conceptual considerations as well as comparisons against other implementations of algebraic multigrid available in the lattice QCD community. For more on all of these aspects, see [45]. A crucial missing piece from previous sections is how the test vectors are constructed, which is mentioned in sect. 3.3.4.3.

3.3.4.1 Two-levels DD-αAMG

For the two-level scheme of DD-αAMG, $M_{SAP}^{(\nu)}$ is taken as the smoother, which as mentioned in sect. 3.3.1 consists of a red-black multiplicative SAP (see alg. 3.2). The coarse system D_c is obtained as $D_c = P^H DP$, see lemma 3.16. From defs.

[20]For a complete discussion of this choice, see e.g. [45].

3.14 and 3.17, P is an aggregation-based interpolation operator, i.e. the aggregates are from a Γ_5-compatible standard aggregation..

When using two levels, in DD-αAMG the smoother and the coarse grid correction are combined into a standard V-cycle[21] with no pre- and ν steps of post-smoothing so that the iteration matrix of one V-cycle is given by [45]

$$C^{(\nu)} = M_{SAP}^{(\nu)} + PD_c^{-1}P^H - M_{SAP}^{(\nu)}DPD_c^{-1}P^H. \tag{3.32}$$

Instead of using this V-cycle as a stand-alone solver, DD-αAMG uses this two-level method as a right preconditioner of flexible GMRES[22] [34], with the preconditioner matrix given by $C^{(\nu)}$ in eq. 3.32

3.3.4.2 Multilevel DD-αAMG

The multilevel approach of DD-αAMG, based on the two-level one from sect. 3.3.4.1, combines again the red-black multiplicative SAP smoother and the interpolation operator based on the standard aggregation (see, again, def. 3.17). The operations $M_{SAP}^{(\nu)}$ and P are taken to be of the same type on all levels in the multigrid hierarchy. We define now, in more precise terms, this multigrid hierarchy.

Definition 3.18.
Let L be the number of levels employed and denote $D_1 := D$. Furthermore, let n_ℓ, $\ell = 1, ..., L$ be the dimension of the underlying vector space on each level ℓ. Just as in def. 3.11 we define interpolation operators

$$P_\ell : \mathbb{C}^{n_{\ell+1}} \to \mathbb{C}^{n_\ell}, \quad \ell = 1, ..., L-1$$

which transfer information from level $\ell + 1$ to ℓ. Accordingly, the operators P_ℓ^H transfer information from level ℓ to $\ell + 1$. Using these operators we recursively define coarse-level operators

$$D_\ell : \mathbb{C}^{n_\ell} \to \mathbb{C}^{n_\ell}, \quad D_\ell = P_{\ell-1}^H D_{\ell-1} P_{\ell-1}$$

for $\ell = 2, ..., L$. The complementary smoothers on each level are denoted by

$$M_\ell, \ \ell = 1, ..., L-1$$

Having all these ingredients, we call

[21] See [80] for a description of the different cycling strategies.
[22] A flexible method is necessary due to the coarse grid being solved via GMRES to very low relative tolerance, which renders the two-level method non-stationary..

$$\{(P_\ell, D_{\ell+1}, M_\ell) : \ell = 1, ..., L-1\}$$

a multigrid hierarchy.

When more than two levels are under use, we can choose among several different cycling strategies. Different cycling strategies can be found in e.g. [34, 45], and the one under common use in DD-αAMG is known as the K-cycle, as suggested in [109]. A K-cycle optimally recombines several coarse-level solves, again in a recursive manner. More precisely, on every level an approximate solution of the coarse-level system is obtained by a few iterations of a flexible Krylov subspace method (in the case of DD-αAMG this is flexible GMRES), preconditioned by the K-cycle from level $\ell+1$ to L. A fundamental difference from the original approach in [109] is that DD-αAMG uses a stopping criterion based on the reduction of the residual rather than a fixed number of iterations.

The K-cycle used in DD-αAMG is stated in alg. 3.6.

Algorithm 3.6: K-cycle

input: ℓ, η_ℓ
output: ψ_ℓ
if $\ell = L$ **then**
  $\psi_\ell \leftarrow D_\ell^{-1}\eta_\ell$
else
  $\psi_\ell = 0$ **for** $i = 1$ to μ
    $\psi_\ell \leftarrow \psi_\ell + M_\ell(\eta_\ell - D_\ell\psi_\ell)$
  $\eta_{\ell+1} \leftarrow P^H(\eta_\ell - D_\ell\psi_\ell)$
  $\psi_{\ell+1} \leftarrow \text{FGMRES}(D_{\ell+1}, \eta_{\ell+1})$ with preconditioner = K-cycle
  $\psi_\ell \leftarrow \psi_\ell + P_\ell\psi_{\ell+1}$
  for $i = 1$ to ν
    $\psi_\ell \leftarrow \psi_\ell + M_\ell(\eta_\ell - D_\ell\psi_\ell)$

3.3.4.3 Adaptive setup phase in multilevel DD-αAMG

We have now provided, in the previous sections of this chapter, not only a relatively detailed discussion on all of the necessary elements composing the DD-αAMG method, but also the specific algorithmic arrangements in the method itself. We have not explained, though, how we obtain the test vectors, which should approximate low eigenmodes and are used in the construction of P and R. This is the task of the setup procedure that we explain now. The setup described in this section corresponds to the one currently implemented in DD-αAMG. This

is an adaptive procedure (in some ways, based on [108, 110]; see [45] for these associations, where the connection between adaptivity and bootstrap AMG as introduced in [108] is discussed in the context of DD-αAMG), where the method is used to aproximate the near kernel with updates on the multigrid solver itself as the test vectors improve. Alg. 3.7 contains an algorithmic description of this setup phase.

Algorithm 3.7: Bootstrap AMG setup

input: smoothing method M, number of iterative phases k
output: Intergrid operators $P = R^H$ and coarse grid operator D_c
// Initial phase
Define set of random test vectors $W = [w_1, \ldots, w_{n_{tv}}]$
for $j = 1, \ldots, n_{tv}$
    $w_j \leftarrow M w_j$
construct P and D_c from W
perform initial phase for D_c
// Iterative phase
for $i = 0, \ldots, k$
    for $j = 0, \ldots, n_{tv}$
        $w_j \leftarrow AMG(w_j)$
    update P and D_c
    perform iterative phase for D_c

The larger the value of k at a certain level ℓ in alg. 3.7, the better the test vectors at ℓ approximate low modes of D_ℓ. This improves the quality of the solver at level ℓ, in the sense of the coarse grid correction at ℓ being a better complement to the smoother at that same level, which implies in turn a reduction in the iteration count in the outermost FGMRES at that same level. Based on this and the fact that the setup phase is in general quite expensive, the number of setup iterations k to be chosen at different levels in the setup phase depends also on whether we want to solve for a single right hand side or many. So, when we need to solve for many right hand sides, more time can be spent on the setup phase to improve the quality of the solver, but when only one right hand side is provided then the setup phase time needs to be minimized in combination with the solve time.

We close this whole description of DD-αAMG by displaying its effectiveness over conventional Krylov subspace methods. Fig. 3.4 shows a comparison of two, three and four levels DD-αAMG versus a very optimized implementation of BiCGStab. The left plot shows the time for a single solve of a linear system with the Dirac operator, which is dependent on a mass shift $m_0 \in \mathbb{R}$. This shift is an indicator for the ill-conditioning of the system i.e. the smaller m_0 the more ill-conditioned the matrix. The multigrid method outperforms the Krylov subspace method by

more than two orders of magnitude for physically relevant mass shifts. We also see that depending on the conditioning, it is (sometimes) beneficial to use additional multigrid levels. The right plot shows the same situation, but includes the time spent for the multigrid setup phase. Due to the bootstrap approach for the setup, the overall cost of the multigrid method is dominated by the setup cost, but is still clearly favorable over BiCGStab by one order of magnitude.

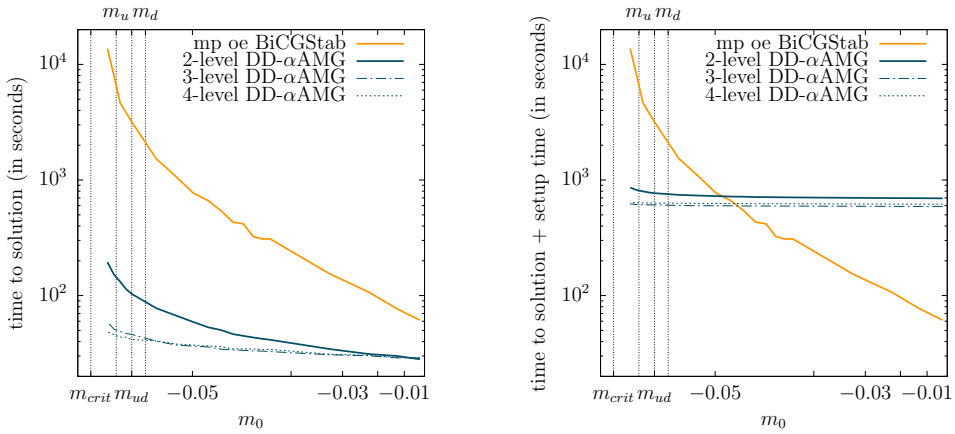

Figure 3.4: Comparing computational cost for solving linear systems with a configuration from a BMW collaboration configuration[23] using DD-αAMG and a Krylov subspace method. The left plot reports on timings for the solve only, whereas the right plot includes the multigrid setup time. Both plots were generated on the JUROPA high performance computer from the Jülich Supercomputing Centre.

Some more specific details such as odd-even preconditioning, mixed precision and the solver employed in the blocks solves in the smoother, are introduced later in this thesis when needed.

3.3.5 DD-αAMG for twisted mass fermions

The DD-αAMG library for clover-improved Wilson-Dirac fermions has been extended to deal with the clover-improved Wilson-Dirac twisted mass operator [28] (see eq. 2.13). The conceptual construction of the method in the twisted mass case is the same as the one outlined in sects. 3.3.4.1 and 3.3.4.2. A critical difference with respect to the Wilson case is the construction of the coarse-grid operator.

In the same way as explained before, the relation between the prolongator and the restrictor is $R = P^H$. This leads to a coarsening of the operator in eq. 2.13 of the form

[23] Lattice of size 64^4, $m_0 = -0.0529$ – see [81] for more details on the configuration employed in this numerical experiment.

$$D_{TM,c} = R(D + i\mu\Gamma_5)P = P^H DP + i\mu P^H \Gamma_5 P$$

Now, since the coarsening is done via Γ_5-compatible aggregates (see the standard aggregation described in def. 3.17), the prolongator satisfies $\Gamma_5 P = P\Gamma_{5,c}$, and then

$$D_{TM,c} = P^H DP + i\mu\Gamma_{5,c}. \tag{3.33}$$

The coarse-grid operator shares some properties with the fine-grid one, in particular the high density of eigenvalues (in absolute value) around the minimum for the $D^H_{TM,c}D_{TM,c}$ operator. This is challenging for iterative solvers such as Krylov-subspace-based methods. As described in [28], the operator in eq. 3.33 is generalized in the sense that μ is allowed to vary, denoted as

$$D_{TM,c} = P^H DP + i\delta\mu\Gamma_{5,c}, \qquad \delta \geq 1 \tag{3.34}$$

which allows one to artificially decrease the condition number by increasing δ. As numerical experiments show, this does not substantially increase the iteration count of the outermost FGMRES but largely decreases the iteration count in the coarsest-level GMRES. Although, from the same numerical experiments it is seen that, as δ increases, the density of eigenvalues (in absolute value) of $D^H_{TM,c}D_{TM,c}$ around the smallest ones increases even further. This makes it harder for GMRES to construct an appropriate polynomial to approximate the inverse of $D_{TM,c}$.

Chapter 4

Coarsest level improvements

In adaptive algebraic multigrid implementations for lattice QCD, $L = 2$ or $L = 3$ levels are typically used [22, 111] and the coarsest level is usually solved via a Krylov-subspace-based method, e.g. GMRES, possibly enhanced with a simple preconditioner and explicit deflation. In a parallel environment, the coarsest level solves suffer from an unfavorable ratio of communication vs. computation: a processor will have relatively few components to update, but a matrix vector multiplication will require a relatively high amount of data to be communicated between neighboring processors. Even more importantly, the communication cost for global reductions (mainly arising in the form of dot products) becomes quite large in comparison to computation [45]. As a result, coarsest level solves usually dominate the cost and can reach up to $\sim 95\%$ of the overall compute time in some cases. Hence, improving scalability of the coarsest-level solver is mandatory to improve the scalability of the whole multigrid solver.

In sect. 4.1, we consider a combination of three major techniques to improve the coarsest level solves:

1. Reducing the number of iterations by preconditioning with operators which do not require global communication.
2. Reducing the number of iterations by approximate deflation using Krylov recycling techniques.
3. Hiding global communication by rearranging computations.

As it will turn out, these techniques can yield a solver much less sensitive to conditioning when approaching m_{crit} (see sect. 4.2.1.3) and improve scalability (see sect. 4.2.1.4). Furthermore, in the particular case of the twisted mass discretization, they prove to be useful in eliminating the artificially introduced parameter at the coarsest level (see eq. 3.34).

The three methods mentioned above are somehow related to Krylov subspace methods. Due to the large density of eigenvalues around the smallest ones in the case of the twisted mass discretization, see sect. 3.3.5, solving becomes difficult for Krylov based methods particularly at the coarsest level. We propose using a different approach in sect. 4.3, based on an incomplete LU factorization.

Sects. 4.2 and 4.4 present results from Krylov- and LU-based methods, respectively. Sects. 4.1 and 4.2 are largely based on [2][24]. The techniques used in sect. 4.3 come from collaborative work with Henning Leemhuis, which resulted in his M.Sc. thesis [112], hence sects. 4.3 and 4.4 are partially based on that thesis.

4.1 Krylov based improvements

This chapter deals exclusively with improvements of the coarsest level solves in adaptive algebraic multigrid methods. On the coarsest level, if we order odd lattice sites before even lattice sites, the matrix D_L (i.e. the matrix at level $\ell = L$, see def. 3.18) has the structure

$$D_L = \begin{pmatrix} D_{ee} & D_{eo} \\ D_{oe} & D_{oo} \end{pmatrix}. \tag{4.1}$$

Herein, D_{oe} and D_{eo} represent the coupling with the nearest neighbors on the lattice. For the standard Wilson discretization, the diagonal matrices D_{ee} and D_{oo} are just multiples of the identity. When we include the clover term, D_{ee} and D_{oo} describe a self-coupling between the variables at each lattice point, i.e. they are block diagonal with the size of each block equal to n_{tv}, the number of variables per grid points[25]. If we take the spin-preserving approach, see def. 3.17, the self-coupling is between variables of the same spin only, i.e. we actually have two diagonal blocks, each again with size n_{tv} which now is half the number of variables per lattice site.

In order to solve coarsest level systems

$$D_L\psi = \eta \Longleftrightarrow \begin{pmatrix} D_{oo} & D_{oe} \\ D_{eo} & D_{ee} \end{pmatrix} \begin{pmatrix} \psi_o \\ \psi_e \end{pmatrix} = \begin{pmatrix} \eta_o \\ \eta_e \end{pmatrix},$$

the standard approach is to solve the odd-even reduced system

$$(D_{ee} - D_{eo}D_{oo}^{-1}D_{oe})\psi_e = \eta_e - D_{eo}D_{oo}^{-1}\eta_o$$

[24] A paper on our work to improve the coarsest-level of DD-αAMG via Krylov-based methods will soon be uploaded to arXiv.

[25] At the end of sect. 3.3.3 we labeled this quantity as N, being this the number of test vectors used in connecting level $L-1$ to L.

for ψ_e with a Krylov subspace method like GMRES, possibly enhanced with a deflation procedure, and then retrieve $\psi_o = D_{oo}^{-1}(\eta_o - D_{oe}\psi_e)$ [45].

For future reference we denote

$$D_c = D_{ee} - D_{eo}D_{oo}^{-1}D_{oe} \tag{4.2}$$

the odd-even reduced matrix of the system at the coarsest level. For each lattice site, it describes a coupling with the variables from the 48 even lattices at distance 2. This is why D_c is not formed explicitly – a matrix-vector multiplication with D_c is rather done by using the factored form (4.2), where each multiplication with D_{eo} and D_{oe} involves the 8 nearest neighbor sites only. Since D_{oo} and D_{ee} are diagonal across the the lattice sites, multiplying with them does not involve other lattice sites, and the explicit computation of their inverse boils down to compute inverses of the local, small $n_{tv} \times n_{tv}$ diagonal blocks they are made up from.

The GMRES, introduced in sect. 3.1.2.2, is the best possible method to solve

$$D_c x = b,$$

The GMRES method relies on the Arnoldi process. The Arnoldi process introduced in alg. 3.3 relies on modified Gram Schmidt [34]. We presented now in alg. 4.1 based on classical Gram Schmidt

Algorithm 4.1: Arnoldi process

Data: (residual) vector r_0, matrix A, dimension k
Result: orthonormal matrix $V_k = [v_1|\cdots|v_k] \in \mathbb{C}^{n\times k}$, and Hessenberg matrix $\bar{H}_k = (h_{ij} \in \mathbb{C}^{(k+1)\times k}$.

1 $\beta = \|r_0\|_2$;
2 $v_1 = r_0/\beta$;
3 **for** $j = 1, \ldots, k$
4 **for** $i = 1, \ldots, j$
5 $h_{i,j} = (Av_j, v_i)$;
6 $w_j = Av_j - \sum_{i=1}^{j} h_{i,j}v_i$;
7 $h_{j+1,j} = \|w_j\|_2$;
8 **if** $h_{j+1,j} = 0$ **then**
9 STOP
10 $v_{j+1} = w_j/h_{j+1,j}$;

Lines 4-6 in the Arnoldi process orthogonalize Av_j against $v_1, \ldots, v_j$. This is done with the classical Gram-Schmidt procedure. It is known that the following mathematically equivalent modified Gram-Schmidt procedure (as used in alg. 3.3)

which uses the partially orthogonalized vector w_j for the computation of the $h_{i,j}$ is numerically more stable.

```
1 for j = 1, ..., k
2 |   w_j = A v_j;
3 |   for i = 1, ..., j
4 |   |   h_{i,j} = (w_j, v_i); w_j = w_j − h_{i,j} v_i;
```

When using classical Gram-Schmidt as in alg. 4.1 on a parallel computer, the computation of the k inner products can be fused into one single global reduction operation. This represents a substantial saving when the computational work load per process is small, as it is typically the case when solving systems with the coarsest grid marix D_c. Stability is a minor concern, since very inaccurate solves on the coarsest level typically are sufficient, reducing the initial residual by just one order of magnitude. However, the number of iterations required is nonetheless high (some hundreds or even thousands). This makes GMRES increasingly inefficient, since the orthogonalizations in the Arnoldi process require k vector-vector operations in step k, eventually becoming far more costly than the matrix-vector multiplication. This is why, usually, GMRES must be restarted, as explained in sect. 3.1.2.2.

In the context of restarted GMRES, *preconditioning* becomes useful (see sect. 3.1.2.3), and in this chapter we use it to improve scalability by trading global reductions against increased local computations. In sect. 4.1.0.1 we describe how we use a left block diagonal preconditioner to accomplish this, and then in sect. 4.1.0.2 we use a right preconditioner.

Furthermore, we explore the interplay of those preconditioners with a deflation+recycling method, specifically GCRO-DR [30], in sect. 4.1.0.3. Both preconditioners allow for a reduction of the iteration count in coarsest-level solves with the immediate advantage of having less dot products at that level but at the expense of an increase in local work (i.e. matrix-vector multiplications), particularly for the polynomial preconditioner. Moreover, preconditioning tends to cluster the spectrum of the preconditioned matrix such that the number of small eigenmodes becomes small. It is particularly in such situations that deflation achieved using GCRO-DR can yield substantial accelerations of convergence, since the deflation will damp the influence of these small eigenmodes. A minor computational drawback of using deflation+recycling is that recycling vectors have to be constructed, which requires additional work, and they are then deflated via projection in each Arnoldi iteration. But those deflations can be merged with the Arnoldi dot products when using classical Gram Schmidt. This increases the risk of instability, but, as explained before, the coarsest level can be solved with low accuracy.

Finally, in sect. 4.1.0.4 we integrate pipelining with the algorithms obtained so far as a means to hide global communications and thus further improve scalability of the coarsest level solves.

4.1.0.1 Block diagonal preconditioner

Left preconditioning uses a non-singular matrix B to transform the original system $D_c x = b$ into

$$B^{-1} D_c x = B^{-1} b. \tag{4.3}$$

GMRES is now applied to this system. This means that in each iteration we now have a multiplication with $B^{-1} D_c$, which is done as two consecutive matrix-vector multiplications. This is why multiplication with B^{-1} should be easy and cheap in computational cost. We took B to be an approximate block Jacobi preconditioner. More precisely B is the bock diagonal of D_{ee}, where each $n_{tv} \times n_{tv}$ diagonal block B_i corresponds to all variables associated with lattice site i. We compute B_i^{-1} once for each i, and then perform the matrix vector products with B_i^{-1} as direct vector products. This is computationally more efficient than computing an LU-factorization of B_i and then perform two triangular solves each time we need to multily with B_i^{-1}. Note that all multiplications with B_i^{-1} can be done in parallel and they do not require any communication if we — as we always do — keep all variables for a given lattice site on one processor.

The diagonal blocks of D_{ee} are not identical to those of D_c, and one might expect to obtain a more efficient preconditioner when using those of D_c. Computing the part of $D_{eo} D_{oo}^{-1} D_{oe}$ contributing to each diagonal block of D_c can, in principle, be done in parallel, but it requires inversions of D_{oo} and communication with neighboring processes due to the couplings present in D_{eo} and D_{oe}. Limited numerical experiments suggest that the additional benefit of incorporating this part into the block diagonal preconditioner is moderate, so that we used the more simple-to-compute preconditioner which works exclusively with the block diagonal of D_{ee}. Further numerical experiments (see sect. 4.2) indicate that this block preconditioner gives a reduction by a factor of roughly 1.5 in the iteration count in some cases, with very little extra computational effort. It has no effect if there is no clover term, since then the diagonal blocks are all multiples of the identity.

4.1.0.2 Polynomial preconditioner

Right polynomial preconditioning for the system $D_c x = b$ amounts to fixing a polynomial q such that $q(D_c)$ is an approximation to D_c^{-1} and then solving

$$D_c q(D_c) y = b$$

for y using GMRES, transforming back after convergence via the back transformation $x_k = q(D_c) y_k$. This implies that we have

$$x_k \in x_0 + K_k(D_c q(D_c), r_0),$$

and x_k is such that the residual $r_k = b - D_c x_k$ is minimized over the space $x_0 + K_k(D_c q(D_c), r_0)$. If q has degree $\mu - 1$, we invest $k\mu$ matrix-vector multiplications to build the orthonormal basis of $K_k(D_c q(D_c), r_0)$, a space of dimension k, in the Arnoldi process. With the same effort in matrix-vector multiplications, we can as well build the $k\mu$-dimensional subspace $K_{\mu k}(D_c, r_0)$ which contains $K_k(D_c q(D_c), r_0)$. This shows that for the same amount of matrix-vector multiplications, the residual of the k-th GMRES iterate of the polynomially preconditioned system can never be smaller than that of the μk-th iterate of standard GMRES. In other words, polynomial preconditioning reduces the iteration count while possibly increasing the total number of matrix-vector products. Polynomial preconditioning can nevertheless be attractive for two reasons: The first is that the above trade-off can be reversed in the presence of restarts. Standard GMRES is slowed down due to restarts; so if the polynomial preconditioner is good enough to allow to perform preconditioned GMRES without restarts, we might end up with less matrix-vector multiplications. The second is that costs other than the matrix-vector products may become dominant. At iteration k, k inner products and k vector updates are performed in the Arnoldi process. In a parallel computing framework, these inner products, which require global communication, may become dominant for already quite small values of k, while very often matrix-vector products exhibit better potential for parallelization and display much more promising scalability profiles compared to inner products. This even holds if we use the less stable standard Gram-Schmidt orthogonalization within Arnoldi which allows to fuse all k inner products into one global reduction operation as explained after alg. 4.1.

We give an indicative example: assume that q has degree 3 and that we need 100 iterations with polynomially preconditioned GMRES. This amounts to 400 matrix-vector multiplications and 100 global reduction operations (for fused inner products). Assume further that with standard GMRES we need 200 iterations, i.e. 200 matrix vector multiplications and 200 global reductions. So the additional matrix-vector multiplications in polynomial preconditioning are more than com-

pensated for by savings in global reductions as soon as those take more time than two matrix-vector multiplications. Furthermore, the polynomially preconditioned method also saves on the vector operations related to the orthogonalization.

Several types of polynomial preconditioners have been suggested in the literature, based on Neumann series, Chebyshev polynomials or least squares polynomials [113], e.g. These approaches typically rely on detailed *a priori* information on the spectrum of the matrix and its boundaries. Recently, [31, 32], based on an idea from [33], showed that polynomial preconditioning with a polynomial obtained from a preliminary GMRES iteration can yield tremendous gains in efficiency when computing eigenpairs. We will use their way of adaptively constructing the polynomial for the preconditioner as we explain now.

In standard GMRES, an iterate $x_\mu \in x_0 + K_\mu(D_c, r_0)$ can be expressed as $x_0 + q_{\mu-1}(D_c)r_0$ with $q_{\mu-1}$ a polynomial of degree $\mu - 1$, and thus $r_\mu = b - D_c x_\mu = (I - D_c q_{\mu-1}(D_c))r_0 =: p_\mu(D_c)r_0$. Since r_μ is made as small as possible in GMRES, we can consider the polynomial $q_{\mu-1}$ to yield a good approximation $q_{\mu-1}(D_c)$ to D_c^{-1}. Strictly speaking, this interpretation only holds as far as the action on the vector r_0 is concerned, but we might expect this to hold for the action on just any vector if r_0 is not too special like a vector with random components, e.g.

As is explained in [32, 114], the polynomial $q_{\mu-1}$ can be constructed from the harmonic Ritz values of D_c with respect to the Krylov subspace $K_k(D_c, r_0)$. These are the eigenvalues θ_i of the matrix

$$(H_\mu + h_{\mu+1,\mu}^2 f e_\mu^T), \tag{4.4}$$

with H_μ and $h_{\mu+1,\mu}$ from the Arnoldi process, alg. 4.1, and $f = H_\mu^{-H} e_\mu$ [115]. The polynomial $p_\mu(t) = 1 - t q_{\mu-1}(t)$ with $r_\mu = p_\mu(D_c)r_0$ is then given as

$$p_\mu(t) = \prod_{i=1}^{\mu} \left(1 - \frac{t}{\theta_i}\right), \tag{4.5}$$

and since $q_{\mu-1}$ interpolates the values $\frac{1}{\theta_i}$ at the nodes θ_i, eq. 4.5 gives, after some algebraic manipulation, a representation for $q_{\mu-1}$ similar to the Newton interpolation polynomial formula as

$$q_{\mu-1}(t) = \sum_{i=1}^{\mu} \frac{1}{\theta_i} \prod_{j=1}^{i-1} \left(1 - \frac{t}{\theta_j}\right). \tag{4.6}$$

Here, by convention, the empty product is 1.

With this representation we can compute $q_{k-1}(D_c)v$ using $\mu - 1$ matrix-vector products by summing over accumulations of multiplications with $I - \frac{1}{\theta_j} D_c$.

Leja ordering

The representation in eq. 4.6 depends on the numbering which we choose for the harmonic Ritz values θ_i while, of course, mathematically q_μ is independent of the ordering. In numerical computation, however, the ordering matters due to different sensitivities to round-off errors. If D_c is not very well conditioned, the range of $1/\theta_i$ may cover several orders of magnitudes so that it is important not to have all the big or all the small values appear in succession [33]. An ordering choice that works well for a wide variety of cases is the Leja ordering [116]. An algorithm to Leja order harmonic Ritz values is given as alg. 4.2.

Algorithm 4.2: Leja ordering of harmonic Ritz values

Data: Set $\mathbb{K} = \{\theta_k\}_{k=1}^{\mu}$ of harmonic Ritz values.
Result: Set $\{\theta_k^L\}_{k=1}^{\mu}$ Leja ordered harmonic Ritz values.

Choose $\theta_0^L \in \mathbb{K}$ such that $|\theta_0^L| = \max\{|\theta| : \theta \in \mathbb{K}\}$
for $k = 2, \cdots, \mu$
  Remove θ_{k-1}^L from $\mathbb{K}$
  Determine $\theta_k^L \in \mathbb{K}$, such that;

$$\prod_{j=1}^{k-1} |\theta_k^L - \theta_j^L| = \max_{\theta \in \mathbb{K}} \prod_{j=1}^{k-1} |\theta - \theta_j^L|. \tag{4.7}$$

Alg. 4.3 summarizes the process of computing and applying the preconditioning polynomial q of degree $\mu - 1$.

Algorithm 4.3: Polynomialy-Preconditioned GMRES(m)

1. Construct the decomposition $D_c V_\mu = V_{\mu+1} \bar{H}_\mu$ by running μ steps of the Arnoldi process using a random initial vector v_0;
2. Compute the harmonic Ritz values θ_k of D_c as the eigenvalues of $H_\mu + h_{\mu+1,\mu}^2 f e_\mu^T$;
3. Leja order the obtained harmonic Ritz values θ_k;
4. Run GMRES(m) using the right preconditioner $q(D_c) = \sum_{i=1}^{\mu} \frac{1}{\theta_i} \left(I - \frac{1}{\theta_1} D_c\right) \cdots \left(I - \frac{1}{\theta_{i-1}} D_c\right)$.

For the new implementations in DD-αAMG involved in this work we merge the block diagonal (left) preconditioner with polynomial preconditioning. This means

that the preconditioning polynomial q is constructed using $B^{-1}D_c$ rather than D_c so that the overall preconditioned system takes the form (see also eq. 4.3)

$$\begin{aligned}(B^{-1}D_c)q(B^{-1}D_c)y &= B^{-1}b,\\ x &= q(B^{-1}D_c)y.\end{aligned} \tag{4.8}$$

4.1.0.3 Deflation with GCRO-DR

A typical algebraic multigrid solve will take 10-30 iterations. Depending on the cycling strategy (see sect. 3.3), each iteration will require one or more approximate solves on the coarsest level. In DD-αAMG as in other multigrid methods for lattice QCD, the cycling strategy is to use K-cycles [109]. This means that with just three levels we will have in the order of ten coarsest level solves per iteration, and this number increases as the number of levels increases. Even when using the preconditioning techniques presented so far, it usually happens that we need to perform several cycles of restarted GMRES to achieve the (relatively low) target accuracy required for these solves.

This is why acceleration through deflation appears as an attractive approach in our situation. The idea is to use information gathered in one cycle of restarted GMRES to obtain increasingly better approximations of small eigenmodes of D_c and to use those to augment the Krylov subspace for the next cycle. Then even better approximations are computed and used in the follwoing cycle or for the next system solve, etc. Effectively, this means that small eigenmodes are (approximately) deflated from the residuals, thus resulting in substantial acceleration of convergence.

Many deflation and augmentation techniques have been developed in the last 20 years [117, 118], and some of them have already been used in lattice QCD, typically eigCG [119] in the Hermitian case and GMRES-DR [120] for non-Hermitian problems. For Hermitian matrices, eigCG mimics the approximation of eigenpairs as done in Lanczos-based eigensolvers with a restart on the eigensolving part of the algorithm but in principle no restarts of CG itself. GMRES-DR, on the other hand, is used for non-Hermitian problems and, similarly to eigCG, it deflates approximations of low modes. When a sequence of linear systems is to be solved, eigCG can be employed in the Hermitian case but GMRES-DR in principle is not applicable in that case. Here, we propose to use GCRO-DR, the generalized conjugate residual method with inner orthogonalization and deflated restarts of [30].

GCRO-DR combines elements of GMRES-DR [120] and GCRO [121]: it takes deflation from GMRES-DR, and the inner/outer scheme with a minimization over arbitrary spaces from GCRO. It is particularly well suited to our situation where

we not only have to perform restarts but also repeatedly solve linear systems with the same matrix and different right hand sides.

A high level description of one cycle of GCRO-DR with a subspace dimension of m is as follows[26]:

1. Extract $k < m$ approximations to small eigenmodes of D_c from the current cycle.
2. Determine a basis $u_1, \ldots . u_k$ for the space spanned by these approximate eigenmodes, gathered as columns of $U_k = [u_1| \cdots |u_k]$ such that $C_k = D_c U_k$ has orthonormal columns $c_1, \ldots, c_k$.
3. With the current residual, perform $m-k$ steps of the Arnoldi process where you not only orthogonalize against the newly computed Arnoldi vectors, but also against $c_1, \ldots, c_k$. This yields the relation (by imposing $C_k = D_c U_k$ as well)

$$D_c[U_k \; V_{m-k}] = [C_k \; V_{m-k+1}]G_m \tag{4.9}$$

   where

$$G_m = \begin{bmatrix} I_k & B_k \\ 0 & \bar{H}_{m-k} \end{bmatrix} \text{ with } B_k = C_k^H D_c V_{m-k} \tag{4.10}$$

4. Obtain the new iterate by requiring the norm of its residual to be minimal over the space spanned by the columns of U_k and V_{m-k}. This amounts to solving a least squares problem with $G_m \in \mathbb{C}^{(m+1)\times m}$.

The very first cycle of GCRO-DR, where we do not yet have approximate eigenmodes available, is just a standard GMRES cycle of length m. In all subsequent cycles, including those for solving systems with further right hand sides, the first step above typically computes the small eigenmodes as the small harmonic Ritz vectors of the matrix G_m of the previous cycle. We have the option to stop updating the small approximate eigenmodes once they are sufficiently accurate.

A source of extra work in GCRO-DR compared to GMRES is the construction of the recycling vectors in U and C from the harmonic Ritz vectors. By the imposition of $C_k = D_c U_k$ and via a QR decomposition of a very small matrix (which is done redundantly in each process without the need for communications), C_k can be efficiently updated from U_k, so this extra work is relatively small. If the matrix changes from one linear system to the next, the application $D_c U_k$ is needed which implies more extra work, but this is not the case in our solves. Another source of extra work is the deflation of C_k in each Arnoldi iteration, which is again

[26]For the complete step-by-step GCRO-DR algorithm, see the appendix in [30].

relatively cheap as the dot products due to those deflations can be merged with the already existing dot products from Arnoldi into a single global reduction.

4.1.0.4 Communication hiding: pipelining

An additional way to reduce communication time in the coarsest-level solves, complementary to what we have discussed so far, is *communication hiding*, i.e. by overlapping global communication phases with local computations. In this direction, [35] introduces a *pipelined* version of the GMRES algorithm which loosens the data dependency between the application of the matrix vector products and the dot products within the Arnoldi process. This is achieved by lagging the generation of the data obtained from the computation of the matrix-vector products from its actual use in the classical Gram-Schmidt process, so that the proposal vector that gets orthogonalized in a given iteration is precomputed one or more iterations in the past. The number of "lagging" iterations is called the latency of the pipelining.

The method (with latency 1) requires the introduction of an extra set of vectors v_i^a, devised to store the matrix-vector products which are computed in advance. These "precomputed vectors" are related to the orthonormal vectors v_i from the Arnoldi process as $v_0^a = v_0$ and $v_j^a = (A - \sigma_j I)v_j$ for $j \geq 1$. Here, $\sigma_j \in \mathbb{C}$ can be chosen arbitrarily, and judicious choices contribute to maintain numerical stability. In our particular implementations we have chosen the σ_j coefficients to be zero since stability never was problematic due to the low relative tolerance required for the coarsest-level solves.

While this approach allows to hide global communications in Gram-Schmidt orthogonalizations behind local vector update operations, it does not so for the global communication required when normalizing the thus orthogonalized vector w_j to obtain the final orthonormal vector v_{j+1}.

In order to address this concern, it was observed in [35] that this communication can be avoided if instead we compute $\|Av_j\|_2^2$ and then use

$$h_{j+1,j}^2 = \|w_j\|_2^2 = \|Av_j\|_2^2 - \sum_{i=1}^{j} |h_{ij}|^2. \tag{4.11}$$

The computation of $\|Av_j\|_2^2$ can now be done within the same global reduction communication used for the inner products yielding the coefficients $h_{i,j}$ in the classical Gram-Schmidt orthogonalization.

It should be mentioned that a possible complication of this approach is that, due to numerical loss of orthogonality, we might get that $\|Av_j\|_2^2 - \sum_{i=1}^{j} |h_{i,j}|^2$ is not

positive, which results in a breakdown. Even when there is no such breakdown, the above re-arrangement of terms tends to make the method less stable numerically.

Algorithm 4.4: Latency 1 pipelined preconditioned GMRES

$v_0 \leftarrow r_0/\|r_0\|_2$
$v_0^a \leftarrow D_c M v_0$, where M is the polynomial preconditioner $q_{\mu-1}(B^{-1}D_c)$ (see sect. 4.1.0.2), with B^{-1} the block diagonal preconditioner from sect. 4.1.0.1
for $i = 1, \ldots, m$
  $w_{i-1} \leftarrow D_c M v_{i-1}^a$
  for $j = 0, \ldots, i-1$
    $h_{j,i-1} \leftarrow (v_j, v_{i-1}^a)$
  $t \leftarrow \|v_{i-1}^a\|_2^2 - \sum_{k=0}^{i-1} h_{k,i-1}^2$
  if $t < 0$ **then** breakdown
  $h_{i,i-1} \leftarrow \sqrt{t}$
  $v_i \leftarrow \left(v_{i-1}^a - \sum_{k=0}^{i-1} v_k h_{k,i-1}\right)/h_{i,i-1}$
  $v_i^a \leftarrow \left(w_{i-1} - \sum_{k=0}^{i-1} v_k^a h_{k,i-1}\right)/h_{i,i-1}$
$y \leftarrow \text{argmin}\left\|H_{m+1,m}y - \|r_0\|_2 e_0\right\|_2$
$x \leftarrow x_0 + MV_m y$

Alg. 4.4 summarizes the method. Mathematically, it is equivalent to standard GMRES. As compared to the latter, pipelined GMRES requires more memory to store the extra set of vectors v_i^a, and it requires more local computation in the form of additional AXPY operations. The advantage is that the global communications required to obtain the $h_{j,i-1}$ coefficients in the outer loop i and $\|v_{i-1}^a\|^2$ can be performed in parallel to the matrix-vector multiplication needed to compute w_{i-1}.

In order to extend the presented approach to GCRO-DR, which includes recycling and deflation, a new set of pre-computed vectors has to be introduced, specially aimed to store the matrix-vector plus preconditioner application on the recycling vectors. In alg. 4.5 we represent them by v_j^c. Fortunately, these vectors need to be re-computed only when U_k (and by association C_k) changes, which we typically do only for the first few linear systems in the sequence $D_c x_i = b_i$.

The whole coarsest-level solver

We have combined the block diagonal preconditioning with D_{ee}^{-1}, adaptive polynomial preconditioning, deflation and recycling via GCRO-DR, and pipelining into a single implementation for coarsest-level solves within DD-αAMG. The code was implemented in a modularized way such that the user can enable any combination

Algorithm 4.5: Latency 1 pipelined preconditioned GCRO-DR

$r = r_0 - C_k C_k^H r_0$
$v_0 \leftarrow r/\|r\|$
for $j = 0, \cdots, k$ **do** : $v_j^c = D_c M c_j$ **end**, where M is the polynomial preconditioner $q_{\mu-1}(B^{-1}D_c)$ (see sect. 4.1.0.2), with B^{-1} the block diagonal preconditioner from sect. 4.1.0.1
$v_0^a \leftarrow D_c M v_0$
for $i = 0, \cdots, m$
  $w_{i-1} \leftarrow D_c M v_{i-1}^a$
  for $j = 0, \cdots, k$ **do** : $b_{j,i-1} = (c_j, v_{i-1}^a)$ **end**
  for $j = 0, \cdots, i-1$ **do** : $h_{j,i-1} = (v_j, v_{i-1}^a)$ **end**
  $t \leftarrow \|v_{i-1}^a\|_2^2 - \sum_{k=0}^{i-1} h_{k,i-1}^2 - \sum_{k=0}^{i-1} b_{k,i-1}^2$
  if $t < 0$ **then** Breakdown
  $h_{i,i-1} \leftarrow \sqrt{t}$
  $v_i \leftarrow \left(v_{i-1}^a - \sum_{k=0}^{i-1} v_k h_{k,i-1} - \sum_{k=0}^{i-1} c_k b_{k,i-1}\right)/h_{i,i-1}$
  $v_i^a \leftarrow \left(w_{i-1} - \sum_{k=0}^{i-1} v_k^a h_{k,i-1} - \sum_{k=0}^{i-1} v_k^c b_{k,i-1}\right)/h_{i,i-1}$
$y \leftarrow \text{argmin}\left\|G_{m+1,m} y - \|r\|_2 e_{k+1}\right\|$, with $r = r_0 - C_k C_k^H r_0$
$x \leftarrow x_0 + \hat{Z} y$, with $\hat{Z} = [U \quad MV_{m-k}]$

of these four methods already during the compilation of the program. The implementation is ready to use and available at this GitHub repository. At runtime, the execution of the polynomial preconditioner and GCRO-DR is dynamic in the following sense: if the user enables the polynomial preconditioner at compile time with a degree of e.g. $d = 10$, but the number of iterations at the coarsest level is quite low e.g. 5, then we do not force the construction of the polynomial and the polynomial preconditioner is kept off. The block diagonal preconditioner and pipelining were not implemented like this, and if enabled at compile time they are always used during execution.

4.2 Numerical tests: Krylov based

All (Krylov-based) computations were performed on the JUWELS supercomputer from the Jülich Supercomputing Centre. In most of our tests on JUWELS, one process per node and 48 OpenMP threads[27] per process were used in the JUWELS

[27] The number of threads per MPI process can be varied in some cases and for example a value of 20 might be better in certain situations where the number of nodes is extremely large and the work per thread is very small such that the thread barriers start becoming significant. We take this into consideration in sect. 4.2.1.2.

cluster module. In Section 4.2.1 we present results for the clover-improved Wilson discretization using configuration D450r010n1 from the D450 ensemble of the CLS collaboration[28] [122]. Section 4.2.2 deals with twisted mass fermions where we use configuration conf.1000 of the cB211.072.64 ensemble of the Extended Twisted Mass Collaboration[29] [123]. In both cases the lattice is of size 128×64^3.

4.2.1 The clover-improved Wilson operator

With the inclusion of new algorithms at the coarsest level, new parameters for these algorithms need to be tuned, which we do in sect. 4.2.1.1. Once this is done, we test how the whole solver is affected by the numerical conditioning of the discretized operator by varying the mass parameter. Moreover, we perform some scaling tests of the whole solver.

The block diagonal preconditioner of sect. 4.1.0.1 is always used in all experiments here. It comes at very little extra computational cost, but can give a reduction of up to about 1.5 in the iteration count at the coarsest level, as can be seen from tab. 4.1. Furthermore, numerical tests in MATLAB indicate that there is not much difference in the reduction in the iteration count due to the use of the BDP when using the block diagonal of D_c instead of D_{ee}, hence we continue using D_{ee} here.

m_0	without BDP	with BDP
-0.3515	21	15
-0.35371847789	35	26
-0.354	40	28
-0.3545	52	37

Table 4.1: Effect of the block diagonal preconditioner (BDP) on coarsest-level solves in DD-αAMG, where we have the BDP as the only preconditioner of GMRES. The second and third columns are average number of iterations at the coarsest level in the solve phase. We have used configuration D450r010n1 here with different values of m_0.

4.2.1.1 Tuning parameters

The set of default parameters in our solves can be found in tab. 4.2 (see [25, 45] for more on these parameters).

[28] Provided to us by Tomasz Korzec and Francesco Knechtli, both part of the physics department at Bergische Universität Wuppertal.

[29] Provided to us by Jacob Finkenrath, who is part of CaSToRC at the Cyprus Institute.

$\ell = 1$	restart length of FGMRES	10
	relative residual tolerance	10^{-9}
	number of test vectors	24
	size of lattice-blocks for aggregates	4^4
	pre-smoothing steps	0
	post-smoothing steps	3
	Minimal Residual iterations	4
	boostrap setup iterations	4
$\ell = 2$	restart length of FGMRES	5
	maximal restarts of FGMRES	2
	relative residual tolerance	10^{-1}
	number of test vectors	32
	size of lattice-blocks for aggregates	2^4
	pre-smoothing steps	0
	post-smoothing steps	2
	Minimal Residual iterations	4
	boostrap setup iterations	3
$\ell = 3$	restart length of GMRES	60
	maximal restarts of FGMRES	20
	relative residual tolerance	10^{-1}

Table 4.2: Base parameters in our DD-αAMG solves.

In addition to this set of parameters, there are two more that need to be tuned in order to minimize the total execution time of the solves. These parameters are:

- k: number of recycling vectors i.e. dimension of the recycling subspace in GCRO-DR.
- d: degree of the polynomial employed as polynomial preconditioner.
- u: the number of times we update the recycling subspace information represented by U and C. After u updates, we continue to use the last U and C in all further restarts and all further solves with new right hand sides.

The performance dependence on u shows initial gains for smaller values of u with only marginal progress for already only moderately large values. This is why we fixed $u = 10$ in all our experiments.

With a 128×64^3 lattice and with $u = 10$ fixed, fig. 4.1 displays the execution time for the solve phase in DD-αAMG for a cartesian product of (k, d) pairs. The choice $k = 0, d = 0$ (left upper) corresponds to no polynomial perconditioing and no deflation. We see that the choice $d = 4, k = 25$ gives more than a factor of 18 improvement over this case, and that choices for k, d in the neighborhood of this optimal pair affect the execution time only marginally. As a rule, smaller values of

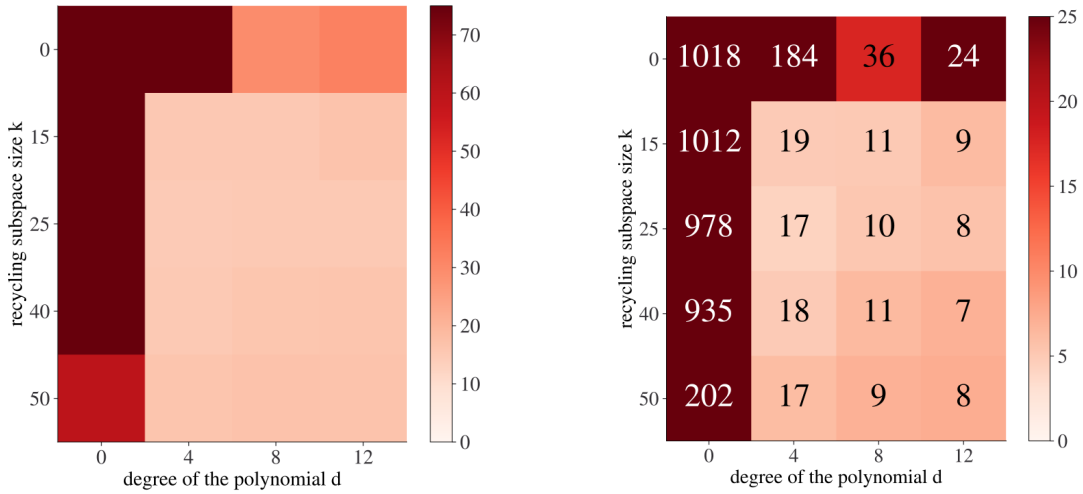

Figure 4.1: Tuning of the parameters k and d . The color of each square in the heatmap from the left represents the total execution time of the whole DD-αAMG solver, while the right corresponds to the time spent at the coarsest level. The configuration was for a lattice of size 128×64^3; we used 32 nodes with 48 OpenMP threads, each. All these computations were done for $m_0 = -0.355937$ (i.e. the most ill-conditioned case in fig. 4.2). The darkest boxes in the heatmap on the left all represent times larger than 200 seconds for $d = 0$ and around 92 seconds for $(k, d) = (0, 4)$. The numbers in the boxes on the right indicate the average number of iterations at the coarsest level during the whole multigrid solve.

k should be preferred as a low value of k reduces the risk of inducing instabilities (due to having deflation and Arnoldi dot products merged, see sect. 4.1.0.3). From fig. 4.1 we see that $d = 4$ and $d = 8$ are equally good in the particular tests that we have run.

4.2.1.2 Pipelining

N_{proc}	N_{thr}	with pipel.	without pipel.
128	20	5.02	4.56
256	20	3.18	2.98
512	10	2.7	2.6
1024	10	2.18	2.0

Table 4.3: Effect of pipelining on the whole DD-αAMG solver. We have used configuration D450r010n1 here with $m_0 = -0.355937$.

Table 4.3 gives a comparison of the total execution time for one DD-αAMG-

solve without and with pipelining in the preconditioned GCRO-DR solves on the coarsest level. As we see, pipelining always *increases* the execution time by up to 10%, even for larger number of processors.

N_{proc}	N_{thr}	pipel.	mvm	mvm-w.	glob-reds
256	20	OFF	0.638	0.127	0.235
512	10	OFF	0.687	0.210	0.259
1024	10	OFF	0.606	0.144	0.33
256	20	ON	0.964	0.331	0.0558
512	10	ON	0.901	0.294	0.0614
1024	10	ON	1.050	0.468	0.0896

Table 4.4: Execution times for parts of the coarse grid solves with and without pipelining. Times in last three columns are in seconds.

To understand this behavior better, we timed the relevant parts of the computation and communication on the coarsest level for our MPI implementation on JUWELS. The results are reported in Table 4.4. Here, for different numbers N_{proc} of processors and N_{th} of threads, we report three different timings: mvm refers to the time spent in one matrix-vector multiplication (arithmetic plus communications), mvm-w is the time processors spent in an MPI-wait for the communication related to the matrix-vector multiplication to be completed. Note that DD-αAMG uses a technique from [124] that aims to overlap computation and communication as much as possible for the nearest neighbor communication arising in the matrix-vector multiplication. Finally, glob-reds reports the time processors wait for the global reductions to be completed. We see that these wait times are indeed almost entirely suppressed in the pipelined version. However, we also see that we do not succeed to hide the communication for the global reductions behind the matrix-vector multiplication, since the time of the latter is increased when pipelining is turned on. We conclude that on JUWELS and with the MPI implementation in use, the communication for global reductions and for the matrix-vector multiplication compete for the same network resources, thus counteracting the intended hiding of communication. We anticipate that pipelining will pay off in situations where the matrix-vector multiplications present in the polynomial preconditioner can be done in a non-synchronized manner, so that the mvm-wait times are almost reduced to zero. We hypothesize that this can be achieved in a manner similar to what is called *communication avoiding* GMRES [125], whereby one exchanges vector components which belong to lattice sites up to a distance k in one go and then can evaluate polynomials up to degree k in A without any further communication. Implementing this approach is a major endeavor, though, and out of scope for this thesis.

m_0	old	new
−0.355770	19	19
−0.355815	23	19
−0.355850	26	20
−0.355895	29	20
−0.355937	30	20

Table 4.5: Number of iterations of the outermost FGMRES in DD-αAMG as m_0 moves down to more ill-conditioned cases.

4.2.1.3 Shifting m_0

With the number of processes fixed at 128, we now shift the mass parameter m_0 to see how much the new coarse grid solver improves upon the old when conditioning of the Wilson-Dirac matrix changes. Results are given in fig. 4.2. Pipelining is turned off for these tests and we put $k = 25$, $d = 4$ throughout. Our experiments show that the improvements due to the new coarse grid solver are close to marginal for the better conditioned matrices, but that they become very substantial in the most ill-conditioned cases. Actually, the times for the new solver are almost constant over the whole range for m_0, whereas the times for the old one increase drastically for the most ill-conditioned systems. The left dotted vertical line, located very close to -0.356, represents the location of m_{crit} i.e. the value of m_0 for which the Dirac operator becomes singular.

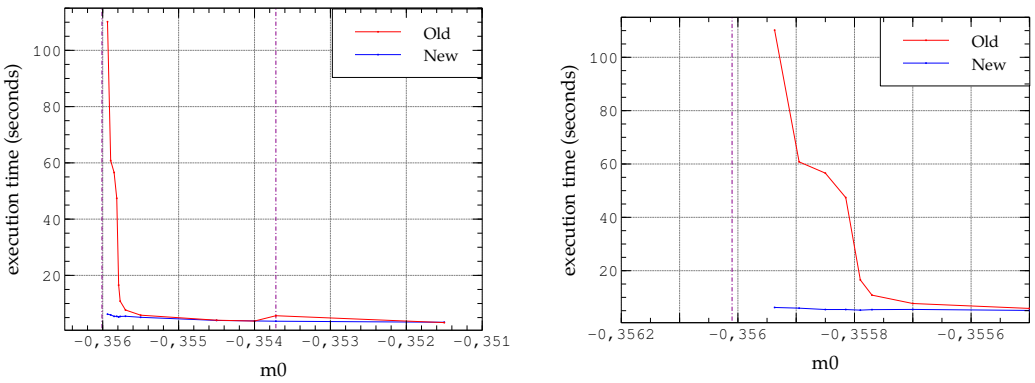

Figure 4.2: Total execution time of the solve phase in DD-αAMG as the system becomes more ill-conditioned (i.e. as m_0 becomes more negative). The vertical dashed line closest to -0.354 represents the value with which the Markov chain was generated and the vertical dashed line closest to -0.356 represents m_{crit}. The right plot zooms into the region where the old version of the solver does not perform well.

In Table 4.5 we summarily report an interesting observation regarding the setup of DD-αAMG. According to the bootstrap principle, the setup performs iterations in which the multigrid hierarchy is improved from one step to the next. In ill-conditioned situations, the solver on the coarsest level might stop at the prescribed maximum of possible iterations rather than because it has achieved the required accuracy. This affects the quality of the resulting final operator hierarchy. The table shows that for a given comparable effort for the setup, the one that uses the improved coarse grid solvers obtains a better overall method, since the coarsest systems are solved more accurately.

4.2.1.4 Strong scaling

Figure 4.3 reports a strong scaling test for both the old and the new coarse grid solves within DD-αAMG. We see that the new version improves scalability quite substantially, due to the better scalability of the coarse grid solve. This is to be attributed to the fact that the new coarse grid solver reduces the fraction of work spent in inner products and thus global reductions which start to dominate for large numbers of processors. Still, we do not see perfect scaling, one reason being that after some point the wait times occurring in the matrix-vector multiplications become perceptible.

We have mentioned before the need to tune the number of OpenMP threads when going to a very large number of processes. Indeed, as the work per OpenMP thread becomes quite small, then thread barriers start becoming problematic from a computational performance point of view. The results presented in fig. 4.3 already include this tuning: for 32 and 64 process we used 48 OpenMP threads per process, for 128 and 256 we switched to 20 threads, and finally for 512 and 1024 we rather used 10 threads. Pipelining has been kept off for these scaling tests.

We ran another strong scaling test with $m_0 = -0.35371847789$ i.e. the value of the mass parameter originally used for the generation of the ensemble. The results of this are shown in fig. 4.4. The coarsest-level improvements did not bring visible gains to the whole solver execution time, which is due to having a quite well-conditioned coarsest level. We also note that the new and old versions of the solver match in this case and for any other relatively well-conditioned value m_0, which gives consistency to the new implementation with respect to the old one.

4.2.2 The twisted mass operator

We now turn to the twisted mass discretization, eq. 2.13, where the parameter μ "shields" the spectrum away from 0 in the sense that the smallest singular

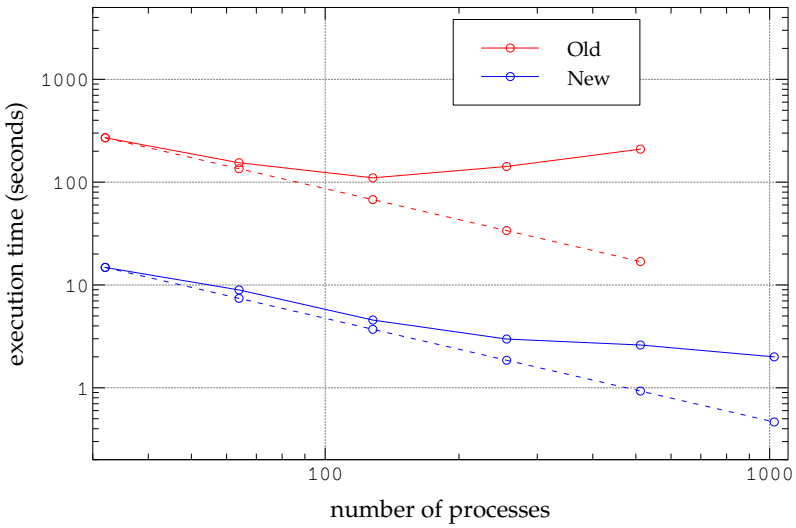

Figure 4.3: Strong scaling tests on Wilson fermions for the new coarsest-level additions. The solves were applied over a 128×64^3 lattice. *Old* means the previous version of DD-αAMG without the coarsest-level improvements introduced in this chapter, and the vertical axis represents the whole solve time. The dashed lines indicate how both cases would behave in case of perfect scaling. All these computations were done for $m_0 = -0.355937$ (i.e. the most ill-conditioned case in fig. 4.2).

value of D_{TM} is $\sqrt{\lambda_{sm}^2 + \mu^2}$ with λ_{sm} the smallest eigenvalue in absolute value of the symmetrized clover-improved Wilson Dirac operator D. This is, in general, algorithmically advantageous, but eigenvalues now have the tendency to cluster around the smallest ones [68].

There is an extension of DD-αAMG that operates on twisted mass fermions [28]. In that version, the twisted mass parameter μ remains, in principle, propagated without changes from one level to the next. On the coarsest level, the clustering phenomenon of small eigenvalues is particulalry pronounced, resulting in large iteration numbers of the solver at the coarsest level. A way to alleviate this is to use, instead of μ, a multiple $\mu_c = \delta \cdot \mu$ with a factor $\delta > 1.0$ on the coarsest level. As was shown in [27], this can decrease the required number of iterations substantially.

We used configuration conf.1000 of the cB211.072.64 ensemble of the Extended Twisted Mass Collaboration [123]. The lattice size is 128×64^3 and $\mu = 0.00072$. Different values of μ_c lead to a different spectrum at the coarsest level, and therefore for each different value of μ_c a new tuning of the new coarsest-level parameters u, k and d has to be performed. We tuned parameters in a similar way as we described before for Wilson fermions. For $\delta = 8.0$, $\delta = 16.0$ and $\delta = 20.0$, $u = 5$

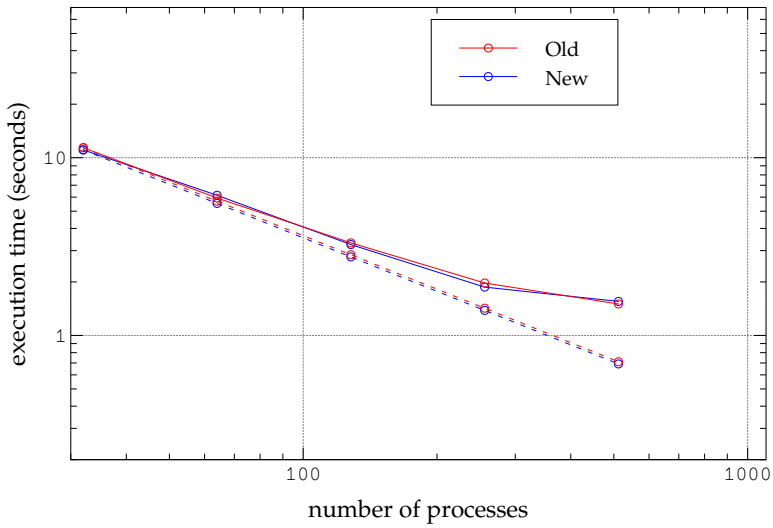

Figure 4.4: Strong scaling tests on Wilson fermions for the new coarsest-level additions. The solves were applied over a 128×64^3 lattice. *Old* means the previous version of DD-αAMG without the coarsest-level improvements discussed in this chapter, and the vertical axis represents the whole solve time. The dashed lines indicate how both cases would behave in case of perfect scaling. All these computations were done for $m_0 = -0.35371847789$.

was found to be sufficient, with $k = 35$ and $d = 2$ being optimal values. Also, we have used, in the twisted mass case in this section, the same DD-αAMG base parameters used in the Wilson case (see tab. 4.2).

We ran strong scaling tests[30] with the three different values of μ_c stated above. The results are shown in fig. 4.5. The left plot, which is for $\mu = 8.0$, shows the impact of the coarsest-level improvements on the overall performance of the solver. For example, the execution time for the coarsest level and for the whole solve are reduced by a factor of around 3 and 2, respectively, when using 512 nodes. Moreover, the scalability of the whole solver improves, as the coarsest level time now represents a smaller portion of the whole solver time, and the coarsest level itself scales better in part due to the polynomial preconditioner.

In the right plot of fig. 5.5 we compare the scaling of the whole solver for different values of μ_c. For the old coarsest-level solver, the larger μ_c, the better the scaling of the whole solver, which agrees with the findings in [27, 28]. This is because we need less iterations for one coarsest level solve when μ_c is increased, while at the same time the total number of iterations for the whole multigrid method is only marginally affected by the size of μ_c. With the new improvements, gains are

[30] We varied the number of OpenMP threads with the number of processes in the same way (and with the same values) as in the Wilson case.

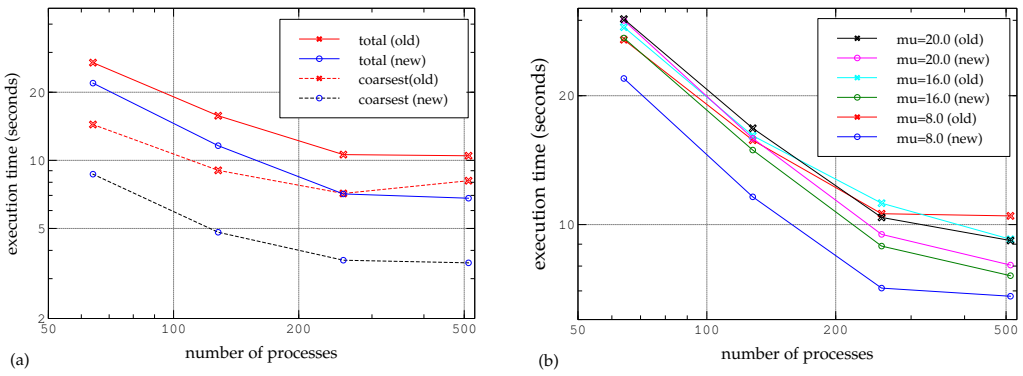

Figure 4.5: Strong scaling tests on twisted mass fermions for the new coarsest-level additions. *Left*: $\mu_c = 8.0$, comparing the previous version of DD-αAMG (*old*) with the one including coarsest-level improvements (*new*), and *total* representing the whole solve time. *Right*: strong scaling plus running over different values of μ_c, with only total (and not coarsest) times plotted.

in the order of 10%-40%, and they are more pronounced for larger numbers of processors and for smaller values of μ_c. With the improvements the dependence on μ_c is reversed: We need less time when μ_c becomes smaller. We explain this by an increase in the density of small modes of the coarsest-level operator, which renders GCRO-DR less effective, as it would require large values of d to resolve the many low modes of the operator.

This leads to the idea of exploring the coarsest-level improvements for even smaller values of μ_c. We did so for $\delta = 1$. In this case, the optimal values for the improved coarsest level solver were $u = 5$, $d = 10$ and $k = 80$. The improvements over the old coarsest level solver, which is plain restarted GMRES with a restart length of 100, are tremendous: The old solver needs on the order of 100,000 iterations to reduce the norm of the residual by a factor of 10^{-1}, while including the new features brings this number down to the order of 100. However, when we compare the fully improved coarsest level with $\delta = 1$ to plain restarted GMRES with $\delta = 8$, we perform only slightly better in the sense of overall execution time. Although $\delta = 1$ is not the typical value to use in conjunction with plain restarted GMRES, we have accomplished bringing δ back down to 1.0, removing thereafter this artificially introduced parameter. Furthermore, we have gained up to a factor of 2 in speedup by increasing δ and keeping our coarsest-level improvements on.

4.3 LU based improvements

The relatively low success of our improvements presented in sects. 4.1 and 4.2 (based on Krylov subspace methods) in dealing with the very hard coarsest-level

solves in the twisted mass discretization[31], has motivated us to use an alternative coarsest-level solver rather based on direct methods via the MUltifrontal Massively Parallel direct Solver (MUMPS[32]) package, which we discuss in the subsequent section.

This section is motivated by [126] and largely based on the master thesis project of Henning Leemhuis[33].

4.3.1 Direct solves via MUMPS

MUMPS [36, 37] is a package for solving sparse linear systems via approximate direct solves based on Gaussian elimination. It can deal in particular with non-symmetric complex matrices in single precision, which matches our systems at the coarsest level of our multigrid hierarchy. Just as in Gaussian elimination, the approximate inverse is pre-computed once in a setup phase, and then applied as a matrix-vector multiplication every time a coarsest-level solve is needed. More specifically, MUMPS does this via three steps:

- *Analysis*: this consists first of a preprocessing step which takes care of improving the quality of the linear system (through e.g. pre-defining an initial pivoting), and second the creation of an *assembly tree* which defines dependencies between the unknowns of the linear system and finds dense subproblems. In the former step, the sparsity pattern of the matrix of the system of equations (in our case D_c[34]) is used to find the best pivots to maintain sparsity of the factors L and U, and in the latter step data structure for the factorize and solve phases are created.
- *Factorization*: after the assembly tree is built and the pivot and data structures created in the analysis phase, the actual numerical factorization is computed in this phase.
- *Solve*: forward elimination and backward substitution are used to approximate the solution.

MUMPS allows for a hybrid MPI+OpenMP execution, which is in accordance with the parallel programming model of DD-αAMG. This will be useful in future numerical tests which fall beyond the reach of this thesis.

A clear disadvantage of direct methods such as Gaussian elimination is that there is no mechanism for tuning e.g. the relative residual tolerance in our solves, which

[31] We have been able to bring the μ_c factor down to 1.0 but at no further gain in performance.
[32] http://MUMPS-solver.org/
[33] Who is now a new member of our group as a PhD researcher.
[34] We will use D_c from hereon to refer to the matrix to be factorized by MUMPS.

might allow us to save unnecessary extra computational work. MUMPS offers such a mechanism, via a *block low-rank* (BLR) approximation. When using the BLR method, MUMPS partitions the matrix D_c in $p \times p$ blocks and maps it to an approximation $\widetilde{D}_c$:

$$\widetilde{D}_c = \begin{pmatrix} \widetilde{D}_{c,1,1} & \widetilde{D}_{c,1,2} & \dots & \widetilde{D}_{c,1,p} \\ \widetilde{D}_{c,2,1} & \ddots & & \vdots \\ \vdots & & & \\ \widetilde{D}_{c,p,1} & \dots & & \widetilde{D}_{c,p,p} \end{pmatrix} \tag{4.12}$$

A low-rank approximation can be applied to each off-diagonal block in the form $\widetilde{D}_{c,i,j} = X_{i,j} Y_{i,j}^H$, with this compression such that $\|\widetilde{D}_c - D_c\| < \epsilon$ in some norm, and the value of ϵ allows us to approximately tune the desired tolerance in our solves with D_c.

Since the matrix $\widetilde{D}_c$ consists of several local independent compression steps, the results of a matrix multiplication $\widetilde{D}_c$ can deviate from the result of $D_c x$ by more than ϵ. Any global relative residual norm $\|D_c x - \widetilde{D}_c x\|_2 / \|D_c x\|_2$ can still be larger than ϵ.

4.4 Numerical tests: LU based

As introduced in sect. 4.1, the coarsest level in DD-αAMG comes with an odd-even factorization. As implemented in DD-αAMG, all matrix-vector multiplications are highly optimized and the application of any matrix is encoded in a function that receives a vector and returns the result of the matrix multiplication, but the matrix itself is never stored explicitly. In order to use MUMPS, we need to store the coarsest-level matrix D_c in an explicit sparse format e.g. CRS to provide MUMPS with the actual matrix. To do this, we first turned off odd-even preconditioning at the coarsest level and then stored D_c by means of three arrays (rows, columns, values) representing the matrix in sparse format. This array is accepted by MUMPS.

We ran our MUMPS comparison tests[35] on a single node Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz with 56 cores and 1.5 TB or RAM. We ran with 32 processes and 1 OpenMP thread per process. We tested again with the 128×64^3 twisted mass matrix from sect. 4.2.2. We found that the optimal number of levels when using restarted GMRES as the coarsest-level solver is three, while when

[35] These tests were performed in collaborative work with Henning Leemhuis as part of his master thesis project.

using MUMPS it is better to use four[36]. Tab. 4.6 lists the parameters used when solving at the coarsest level with MUMPS i.e. with four levels, with the unlisted ones having the same value as in tab. 4.2. When using three levels (i.e. plain restarted GMRES), the DD-αAMG parameters are almost the same as in tab. 4.2, except for the ones listed in tab. 4.6 for $\ell = 1$ and $\ell = 2$.

$\ell = 1$	relative residual tolerance	10^{-10}
	boostrap setup iterations	5
	post-smoothing steps	3
$\ell = 2$	boostrap setup iterations	4
$\ell = 3$	restart length of FGMRES	5
	maximal restarts of FGMRES	2
	relative residual tolerance	10^{-1}
	number of test vectors	32
	size of lattice-blocks for aggregates	2^4
	pre-smoothing steps	0
	post-smoothing steps	3
	Minimal Residual iterations	4
	boostrap setup iterations	4
$\ell = 4$	restart length of GMRES	60
	maximal restarts of FGMRES	20
	relative residual tolerance	10^{-1}

Table 4.6: Base parameters in our DD-αAMG solves, with MUMPS.

The results of these tests are displayed in tab. 4.7, where the times are all in seconds. The second column corresponds to plain restarted GMRES as the coarsest-level solver, while in the third column we have used MUMPS as a preconditioner to GMRES. The time t_ℓ is to be interpreted as the time seen from level ℓ, e.g. t_3 is the time spent at level three only, t_2 the time spend in the combination of levels two and three, and so on. The times when using MUMPS are an upper bound in the sense that odd-even preconditioning has been turned off entirely[37] when using MUMPS i.e. it is not even used in the blocks solves in SAP, which if enabled would give us even lower execution times in column three.

[36]The main reasons to prefer having four levels over three (i.e. 16^4 versus 8^4 coarsest-level lattices) when using MUMPS are, first, a substantial reduction in the volume of data communicated among processes when performing the approximate direct solves, and second a large reduction in arithmetic work done in the factorisation phase of MUMPS.

[37]We have done so for simplicity, i.e. some extra coding is required to have odd-even preconditioning in SAP but not at the coarsest level. This will be of course enabled in a later version of our code.

t_ℓ	**Without MUMPS** ($\delta = 8.0$)	**With MUMPS** ($\delta = 1.0$)
t_0	2012.8	1347.9
t_1	1515.7	920.3
t_2	1236.5	450.0
t_3	-	176.1

Table 4.7: Execution times for the comparison of MUMPS versus no MUMPS in coarsest-level solves in a twisted mass gauge configuration with a lattice size of 128×64^3. The times are in seconds.

Numerically, we see the iteration count of GMRES at the coarsest level going from ~ 400 without MUMPS to 1 when using MUMPS. Also, due to the small coarsest lattice when using MUMPS, the setup times for MUMPS to analyze and factorize the matrix D_c are relatively small, leading to a reduction in the setup phase of DD-αAMG time from 11401.2 to 7243.6 seconds.

Coarsest-level solves via MUMPS are certainly promising on a single node, as we have seen above, and they seem to be the way to go when dealing with the twisted mass discretization. This motivates us to further explore this approach in the future.

4.5 Outlook on coarsets-level computations

There are further improvements to be done for both approaches presented in this chapter, i.e. Krylov and LU based. They are all fundamentally important in the context of large-scale computing, and particularly relevant as current supercomputers dive more into the exascale. We briefly discuss these now.

Krylov based

An important outcome of the discussion in sect. 4.2.1.2 is the need of a communication-avoiding scheme in our coarsest-level implementations: as we increase the number of nodes in our executions, nearest-neighbor communications become a two-fold problem, first in the sense of its lack of scalability, and second as they interfere with the global communications trying to be hidden by pipelining. We will implement a communication-avoiding method, which we expect to have a nice interplay with both pipelining and the polynomial preconditioner.

A less pressing (but still relevant) improvement consists of evaluating, implementing and testing the extraction of the actual block-diagonal of D_c, to be used in

the block diagonal preconditioner (see sect. 4.1.0.1) instead of D_{ee}. Our MATLAB tests, on relatively small lattices, indicate so far that there is no significant algorithmic gain when using D_c instead of D_{ee}. We would like to further test this on more realistic lattices.

LU based

From sect. 4.4, see in particular tab. 4.7 in there, MUMPS seems to be a better way of solving the coarsest-level for twisted-mass fermions, compared to Krylov-based methods. However, our tests in sect. 4.4 are on a single node, and the BLR approximate direct solver of MUMPS scales quite badly[38] as we increase the number of compute nodes. Although our Krylov-based methods suffer also as we increase the number of nodes, the effect is more dramatic in the MUMPS solver that we have used here.

The concept of *agglomeration* can be introduced in multigrid methods[39], to improve the scalability of the overall solver. The core idea behind agglomeration is quite simple: run coarser levels on less nodes than finer ones. In particular, one can choose to apply agglomeration at the coarsest-level only. In the particular case of using MUMPS at the coarsest-level in DD-αAMG, agglomeration might allow us to scale well up to a large number of nodes, without the different stages of MUMPS having to necessarily scale well.

Furthermore, we will apply agglomeration in combination with our Krylov-based methods as well, in particular for twisted-mass solves, which might bring an additional performance in that case.

Our upcoming work involving agglomeration and MUMPS in DD-αAMG will continue to be in collaboration with Henning Leemhuis, as one of the topics in his PhD research.

All the DD-αAMG improvements, associated to the twisted mass discretization, will be directly useful in hybrid Monte Carlo computations performed by the Extended Twisted Mass Collaboration. As for the improvements related to the clover-improved Wilson operator, we plan to run a comparison of the current state of the solver in OpenQCD, against our latest DD-αAMG, in lattice QCD computations via distillation, this in the context of the PhD work of Juan Urrea (under the supervision of Prof. Dr. Francesco Knechtli at Bergische Universität Wuppertal).

[38] Neither its factorisation phase nor its solve phase scale well.
[39] See [126] for a discussion on agglomeration and examples of its use.

Chapter 5

Hybrid GPU/CPU DD-αAMG

With the currently fast and ongoing evolution of Graphic Processing Units (GPUs) and High Performance Computing hardware in general, computational science applications are seeing much faster and energy-efficient implementations for some of its most demanding large-scale computations. Such applications go from Machine Learning [127] to collisions in molecular dynamics [128], and lattice QCD is also one of them. One of the main efforts in using heterogeneous computing for lattice QCD computations is the QUDA library[40] [129], through which one can perform calculations in lattice QCD on GPUs via NVIDIA's CUDA platform. The smoother in the multigrid solver within the QUDA library can be chosen from multiple options, the best one for scalability being either an additive or multiplicative SAP, where the SAP domain-decomposition blocks are of the size of the local lattice i.e. the lattice size per MPI process. In sect. 5.3.1 we show how having the SAP domain decomposition match the process decomposition is not the best option for efficiency.

Furthermore, the multigrid solver in the QUDA library offloads operations at all levels of the multigrid hierarchy to be computed on GPUs. The efficiency problems at the coarsest level described in chapter 4 can be particularly apparent when using GPUs, in two ways. First, as finest-level operations are computed faster on GPUs, the coarsest level is more exposed. Second, if all levels are computed on GPUs, then scalability issues are coming not only from large communication times on coarser levels, but also due to the very little data being used for computations on them[41]. Our contribution in this chapter consists of a hybrid solver where

[40] See https://github.com/lattice/quda

[41] The less the number of lattice sites involved in GPU computations, the sooner scalability issues will appear – this happens faster on GPUs than on CPUs due to the lack of a cache memory hierarchy like the one present on CPUs.

some finest-level operations of DD-αAMG are done on GPUs and coarser levels run on CPUs.

The remainder of this chapter is organized as follows. A short reminder on the role that smoothers play in DD-αAMG is given in sect. 5.1. Some details regarding offloading smoothers to GPUs are presented in sect. 5.2. Then, results of numerical tests on the finest-level smoother alone and the GPU-boosted DD-αAMG solver as a whole are shown in sect. 5.3, where we talk about aggressive coarsening as a way to use GPU resources better and to improve the scalability of hybrid GPU+CPU solvers in lattice QCD. All of the implementations for this chapter were done within the DD-αAMG library for clover-improved Wilson fermions [45].

The nature of this chapter is different compared to previous ones. Here the main focus is on implementation aspects at a low level e.g. cache efficiency on GPUs and CPUs, GPU hardware, etc., although we still rely on the conceptual grounds of AMG introduced in chapter 3.

5.1 SAP in DD-αAMG

Let us now briefly recall the general algorithmic structure of DD-αAMG, with special emphasis on the smoother at the finest level.

One iteration of the K-cycle employed by DD-αAMG (see sect. 3.3.4.2) has the structure given in fig. 5.1, where R and P stand for restriction and prolongation steps, the blocks labeled with A correspond to Arnoldi operations not detailed further in the figure, the S blocks are the smoother and the lowest-level black boxes are the coarsest-level solves.

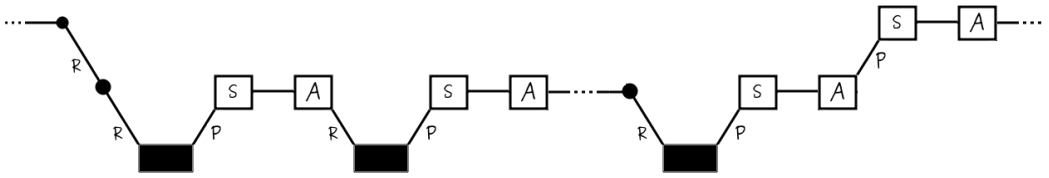

Figure 5.1: One step of DD-αAMG's 3-level multigrid (MG).

The smoother (sect. 3.3.4.2) is red-black block Gauss-Seidel i.e. multiplicative SAP, which on CPUs uses caches in an advantageous manner, matching perfectly the hybrid distributed+shared programming model (e.g. MPI+OpenMP) used by DD-αAMG and scales very well with the number of processors on large-scale machines.

The lowest-level black boxes in fig. 5.1 are performed via GMRES in the original version of DD-αAMG, and can be boosted via the algorithmic combinations described in chapter 4.

5.2 Schwarz Alternating Procedure on GPUs

5.2.1 Domain Decomposition: GPUs vs CPUs

The SAP algorithm used in DD-αAMG behaves differently, from an efficiency point of view, if it is implemented either on CPUs or GPUs (although, its algorithmic properties remain the same). Furthermore, the blocks in SAP are arranged in a red-black fashion (see sect. 4.1) and the solves with the Schur complement are performed by a few steps of the minimal residual (MR) algorithm [34], which is mathematically equivalent to GMRES(1) i.e., restarted GMRES with a cycle length of 1. MR allows us to perform solves with non-symmetric block-matrices with minimum memory requirements [34].

5.2.1.1 CPUs

When plain (non-blocked) Gauss-Seidel with coloring is used, one of the main problems with implementing it "naively" (i.e. no domain-decomposition) is that it does not make good use of cache. This is one of the main motivations behind preferring SAP over for example plain Gauss-Seidel when we are trying to develop highly performing scientific codes [130]. Another motivation for preferring SAP is of course the expected faster convergence due to more frequent local updates [131].

Let us illustrate this cache-friendliness with some numbers[42]. If our block-lattice has dimensions 4^4 then a single vector living in that lattice has a size of $12\times4^4 = 3072 = 24$ KB (in single precision). Due to symmetries and sparsity, the data for the corresponding block-matrix does not have a size of $3072\times3072 = 9437184 = 72$ MB, but rather $9\times42\times4^4 = 756$ KB. MR, as implemented in DD-αAMG, reads/writes roughly 10 vectors and the block-matrix during its execution, which amounts to a total memory of $10\cdot(24\text{ KB}) + (756\text{ KB}) = 996$ KB. The machine where we perform our numerical experiments has an L2 cache of size 1024K, which is enough to contain all the data associated with a whole single block solve.

We present how SAP is realized in a bit more detail in alg. 5.1: launch asynchronous communications (line 2), then perform all the computations related to

[42] All of these numbers are associated to the finest-level, where the matrix is D, as defined in eq. 2.11.

the blocks of color $c = 0$ (i.e. the first color, let us call this red) *not involved* in the communications launched in line 2, and then in the second iteration of the for loop in c compute with the black blocks also not involved in communications. These initial computations correspond to lines 7 and 9. Then, after waiting for the communications to finish in line 10, all those blocks that have not been updated yet are processed in the second for loop in c.

Algorithm 5.1: SAP on CPUs

```
   Data: Block (array of domain-decomposition blocks), s (SAP iterations),
         n_b (number of domain-decomposition blocks)
 1 for k = 1, ..., s
 2     for c = 0, 1
 3         StartGhostExchanges()
 4         for i = 1, ..., n_b
 5             if Block[i].color==c & Block[i].no_comm then
 6                 // some boundary-related operations on a single block
 7                 BoundaryOpsCPU(Block[i])
 8                 // minimal residual on a single block
 9                 MR(Block[i])
10     for c = 0, 1
11         WaitGhostExchanges()
12         for i = 1, ..., n_b
13             if Block[i].color==c & Block[i].comm then
14                 // some boundary-related operations on a single block
15                 BoundaryOpsCPU(Block[i])
16                 // minimal residual on a single block
17                 MR(Block[i])
18     // some final operations
19     FinalOps()
```

What is important at this point is to notice that MR in alg. 5.1 is executed on a single block every time it is called, and that the execution of each block is completely independent of that of all the other blocks of the same color. This is an interesting fact if we think about shared memory, because it implies that we can use different OpenMP threads to process for example all the blocks of a certain color not involved in communications (this is the way it is actually implemented in DD-αAMG). An ideal scenario would be if we could process e.g. all of those blocks of a certain color not involved in communications at the same time, which is unfortunately not possible on CPUs, but it becomes a good feature when we turn to use GPUs. On CPUs we could try to get close to this approach (of

"fusing" blocks) as much as possible by using vectorization, but still the inherent serial nature of CPUs makes it impossible.

But although we have to go block-by-block on our per-thread computations in SAP, the fact that we use blocks small enough to fit in L2 or L3 cache makes the CPU implementations in DD-αAMG highly well-performing.

5.2.1.2 GPUs

In its CPU implementation, DD-αAMG makes use of MPI and OpenMP for distributed and shared memory manipulations. At the "lowest" level i.e. for the execution of the fundamental operations $(+,-,*,/)$ it uses SIMD [132] through SSE for vectorization[43].

When we switch to use GPUs, we keep using MPI for distributed memory but we use SIMT (Single Instruction Multiple Threads [38]) for the fundamental operations.

To get the most performance out of our GPU computations, we have to launch as many tasks as possible in a single CUDA kernel. By doing this, while some groups of threads are retrieving memory from *device RAM*, other threads are computing their tasks. In this way, reading from device RAM can be hidden behind computations quite well by massively launching tasks concurrently.

Alg. 5.2 shows the approach that we follow when offloading parts of SAP to GPUs within DD-αAMG.

Using GPUs for HPC applications has pros and cons. In particular, in the context of SAP, one disadvantage is that cache-friendliness as we had it for CPUs is gone. To see this, let us go back to the numbers given in Sect. 5.2.1.1: a single lattice block requires 324 KB. In some of our numerical experiments, we have used Quadro P6000 GPUs[44], which have 3 MB of L2 cache and although this is enough to store all the data associated with a single lattice block, we do not compute one such block at a time, but rather a group of them concurrently as can be seen from alg. 5.2.

For example, if we have a local lattice (i.e. per GPU) of dimensions 16^4 then, with blocks of dimension 4^4 we get 256 such blocks. This implies that at some point in SAP we will have to launch over 64 lattice blocks concurrently[45], for a total

[43] The use of vectorization is optional i.e. it can be disabled, and its use is of course dependent on whether the compiler and the hardware allow it.

[44] See `https://www.techpowerup.com/gpu-specs/quadro-p6000.c2865`.

[45] The list `BlocksList`, in alg. 5.2, is filled-up four times: two colors and with/without communications. In this 16^4 local lattice that we are describing here, two of those lists will have over 64 blocks, and the other two therefore less than 64.

Algorithm 5.2: SAP on GPUs

Data: Block (array of domain-decomposition blocks), s (SAP iterations), n_b (number of domain-decomposition blocks)

```
1  for k = 1, ..., s
2      StartGhostExchanges()
3      for c = 0, 1
4          BlocksList = []
5          for i = 1, ..., n_b
6              if Block[i].color==c & Block[i].no_comm then
7                  BlocksList.append(Block[i])
8          // some boundary-related operations on a set of blocks
9          BoundaryOpsCPU(BlocksList)
10         // minimal residual on a set of blocks
11         MR(BlocksList)
12     WaitGhostExchanges()
13     for c = 0, 1
14         BlocksList = []
15         for i = 1, ..., n_b
16             if Block[i].color==c & Block[i].comm then
17                 BlocksList.append(Block[i])
18         // some boundary-related operations on a set of blocks
19         BoundaryOpsCPU(BlocksList)
20         // minimal residual on a set of blocks
21         MR(BlocksList)
22     // some final operations
23     FinalOps()
```

memory larger than 64×(996 KB) = 62.25 MB. In conclusion, doing things the CPU-way (alg. 5.1) is no longer a possibility if we want our code to run efficiently on GPUs.

5.2.2 SAP in DD-αAMG on GPUs: implementation details

5.2.2.1 Switching for loops: blocks fusing

The use of GPUs presents, in our case, an important trade-off. We loose cache friendliness, but we can take advantage of a related SAP feature: the independence of domain-decomposition blocks. This enables the launch of all of the concurrent

domain-decomposition block computations at the same time on GPUs. This independence was of course present as well in alg. 5.1, but on GPUs we have the possibility to actually launch all of those blocks at the same time.

Both the main advantage and the main downside from using GPUs with respect to CPUs come from within the call to MR in alg. 5.2:

- *Downside*: we have to perform all the operations within MR sequentially i.e. we take a set of domain-decomposition blocks and run the first operation wihtin MR, then for the same blocks the next operation and so on. Although we had the same sequential nature on CPUs, cache friendliness then made things fast, because after the few first statements within MR all of the block's information was already loaded to cache, and therefore the operations became very fast. However, on GPUs we have to reload everything to cache every time we call a new statement within MR.
- *Advantage*: the obvious advantage is that we get the chance to run each statement within MR for all of the domain-decomposition blocks in parallel, which gives us very large tasks to run on GPUs and therefore we can get highly performing executions.

A final downside of using GPUs with respect to CPUs is, again, related to cache friendliness: when we compute MR we have an extra sequential layer, which is related to the physical dimension of our problem and appears due to *coupling terms*. These terms are those in eq. 2.7 appart from the identity. In other words, some of the statements within MR consist of eight operations that have to be run one after the other (4×2 due to physical spatial dimension and possible directions $+$ and $-$). A first problem with having so many sequential calls on GPUs (even with each call being a large task) is that this demands a great deal of hardware synchronization which decreases concurrency, and a second problem is that when we launch many GPU tasks we start accumulating overhead due to setup times.

Even with all the issues that appear when switching to GPUs, the fact that we can launch very large tasks corresponding to many domain-decomposition blocks at the same time, and without large dot products, outweighs the problems just described, and we get significant speedups in our computations (see sect. 5.3).

5.2.2.2 CUDA threads, Domain-Decomposition blocks and memory re-arrangements

Before going into the results of our numerical experiments, we will first describe in this section the mapping that we employed to assign work to our CUDA threads and GPU cores, and how these map to our data and operations within SAP.

Mapping CUDA threads to a set of domain-decomposition blocks

When writing the actual implementation of the CUDA kernels that are in charge of GPU computations when we offload parts from the CPUs into the GPUs, there are multiple decisions to make in terms of hardware, concurrency and memory mappings. In fig. 5.2 we show the hardware mapping of our choosing, which allowed us to avoid frequent on-the-fly memory re-arrangements due to having frequent switchings from CPU to GPU and viceversa.

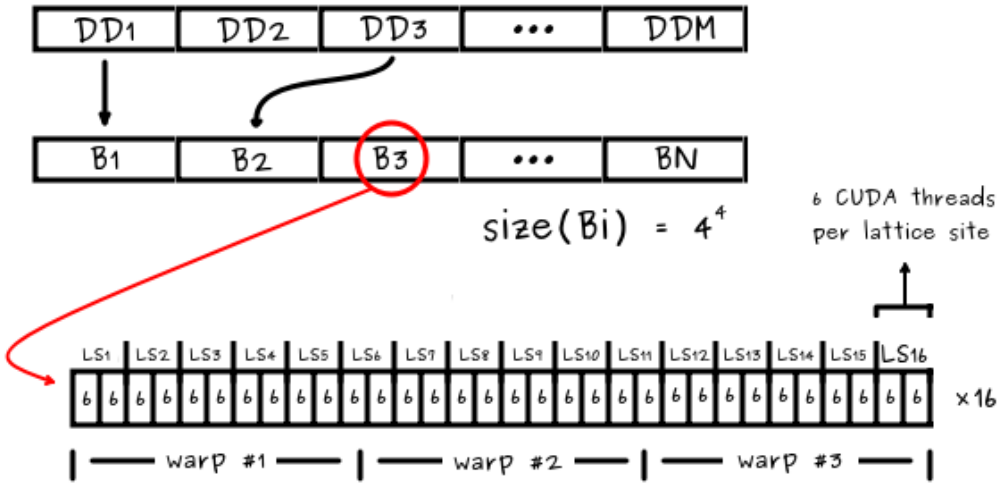

Figure 5.2: Illustration of how we mapped the domain-decomposition blocks to be computed to the CUDA threads used for such computations. Note that $M \geq N$ i.e. $\{\mathrm{B}i\}$ is a subset of $\{\mathrm{DD}i\}$. The illustration here is for the case when our domain-decomposition block size is 4^4 and the CUDA block size is 96 with 6 CUDA threads per lattice site.

With the hardware mapping illustrated in fig. 5.2, we kept our previous CPU memory layouts *almost* intact, except for the memory re-arrangements associated to fusing sites for hopping terms discussed soon.

In fig. 5.2, the set of domain-decomposition blocks $\{\mathrm{DD}i\}$ consists of all the blocks in our system of equations. However, the set $\{\mathrm{B}i\}$ is that of the blocks to be computed in a particular situation (e.g. those of red color involved in communications; see alg. 5.2). The mapping in fig. 5.2 is quite particular and interesting for multiple reasons. If we choose to use a single CUDA thread per lattice site, there is not much room for decisions to be made, but if we want to use more than one CUDA thread per lattice site then we have to be careful with our mapping. For example, if we want to use two CUDA threads per lattice site and due to having a physical GPU hardware warp being of size 32, then we can allocate 16 lattice

sites per warp. The issue with having two CUDA threads per lattice site is that it requires quite a lot of GPU hardware resources per CUDA thread (and the same applies when using a single CUDA thread per lattice site, even worse).

If we want to use six threads per lattice site, the mapping becomes tricky: due to the size of a warp, we have to associate 96 CUDA threads to 16 lattice sites. Although this cuts our warps in a sort of "fractional" manner, the amount of resources per CUDA thread is much less and we increase concurrency greatly which allows the GPU to hide device RAM accesses much better.

In our implementations we have enabled the option to chose between 2 and 6 CUDA threads per lattice site, and correspondingly CUDA block sizes that are multiples of 32 and 96, respectively.

Memory re-arrangements and fusing sites for hopping terms

The coupling terms discussed in sect. 5.2.2.1 are problematic not only due to their sequential nature but also because of their crossed memory accesses. We illustrate these terms (in a reduced space of 2D) in fig. 5.3 for a 2^4 domain-decomposition block.

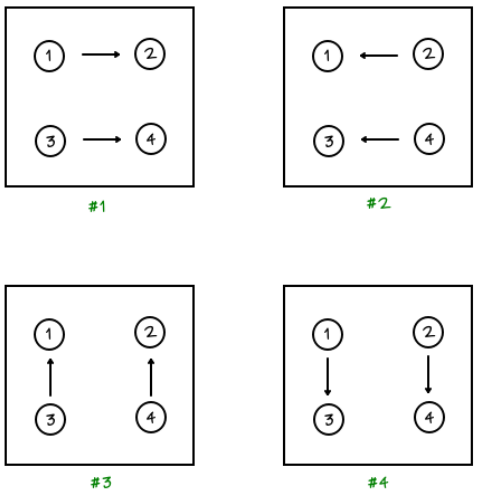

Figure 5.3: Four hopping terms in 2D.

Suppose we have a matrix application that acts following the stages and arrows as indicated in fig. 5.3, e.g. in stage #2 the lattice site labeled as 3 will depend on the current value of the lattice site labeled as 4. Furthermore, there is a sub-matrix $D_{3\leftarrow 4}$ that will give us the rule on how to update site 3 from 4.

When we run over all of the stages and operations in fig. 5.3 it will be inevitable to do crossed memory accesses of the information corresponding to the lattice sites, no matter how we store them in memory. But we can save a significant amount of execution time due to crossed memory accesses of the matrix elements $D_{i\leftarrow j}$; the way to do this is by storing a submatrix associated to stage #1, another one associated to stage #2, and so on. This is precisely what we have done in our GPU computations i.e. we have fused all of the necessary lattice sites for each of the eight possible directions.

To summarize, the crossed memory accesses of the vectors (i.e. the lattice sites) are unavoidable, but we have avoided crossed accesses of the matrix elements. Even better, due to the hardware mapping depicted in fig. 5.2 our matrix memory accesses are almost all done in half-warps which is an optimal reading from device RAM, and due to having domain-decomposition blocks the crossed readings of lattice sites do not address locations which are too far in device RAM.

Further memory optimizations

Besides global memory (i.e. device RAM) we have also used `shared` and `constant` memories in the following way:

- `shared`: when performing matrix-vector multiplications e.g. within MR, we used shared memory to collaboratively (i.e. as done by a group of CUDA threads) load the local matrix into shared memory. For some of those matrix-vector operations, we need buffers. We used shared memory as well to store such buffers to improve locality (e.g. when we have crossed accesses as in hopping terms).
- `constant`: the only way in which we have used this memory so far is to store the data associated to the γ-matrices (i.e. the matrices coming from the Clifford algebra, see def. 2.1).

5.3 Numerical tests

5.3.1 SAP on GPUs

For the numerical tests of SAP on the finest level, to be reported in this section, we use configurations for which we do not specify where they come from nor detail their parameters. Those details are not relevant due to the purely implementational nature of this sub-section. For the tests in sect. 5.3.2, which consist of calls to the whole multigrid solver, we provide specific details for the matrices involved.

We ran our first numerical experiments with 32^4 and 48^4 local lattices (i.e. lattice size per GPU) with two processes and one GPU per MPI process, and the results are shown in tab. 5.1. We have focused here on times for SAP at the finest level only, which is the part of the code that we have enabled to use GPUs on.

L^4	**Speedup Comp**	**Speedup Tot**
32^4	33.54	22.11
48^4	41.71	26.18

Table 5.1: Two types of speedup for the smoother on GPUs, one taking into account only computations and the second one (last column) including times for transferring data from the CPU to the GPU and viceversa. The first column indicates the size of the local lattice. NVIDIA Quadro P6000 GPUs were used.

The second column in tab. 5.1 gives us the speedups that we obtain if we compare the whole SAP on CPUs versus the compute part of SAP on GPUs. This is not a realistic speedup due to the current state of our code: in our current implementation, every time we call SAP on GPUs we have to send data to the GPUs, compute, and then retrieve data back to the CPUs[46]. Therefore, a more realistic speedup in our case is the one displayed in the third column of tab. 5.1, in which case we also take into account the data transfers CPU-to-GPU and viceversa.

It is important to note that speedups of $\sim$30 are to be expected, as we are comparing against CPU code that has been optimized for cache friendliness and with vectorization enabled. More specifically: if we compare only one iteration of MR as called in alg. 5.1 versus the same MR in alg. 5.2, on CPUs and GPUs, then we will see a huge speedup due to GPU usage, of e.g. roughly 80 or even 100. But this speedup decreases as soon as we perform multiple MR iterations. This is, again, because when computing on CPUs after only one iteration of MR everything we need has been loaded to cache e.g. L3, but in the case of GPUs we have to reload data every time we call a statement in MR.

Varying the SAP block size

For a 64×32^3 lattice, we have run our GPU implementation of SAP with different sizes of the SAP blocks, and the results have been tabulated in tab. 5.2. As we can see, smaller SAP blocks lead to better performance. Conversely, we can

[46] Once we have more portions of our code ported to GPUs e.g. the whole finest level, then we won't have to take into account these data transfers.

conclude that taking each domain decomposition block to be of the size of the local volume, i.e. 32^4, will not be the best performing choice.

SAP block	Time per SAP call (seconds)
4^4	0.1444
8^4	0.1562
16^4	0.2316

Table 5.2: Time per SAP call versus domain decomposition block size, on a lattice of size 64×32^3 with two processes and one GPU per MPI process. NVIDIA Quadro P6000 were used.

5.3.2 Hybrid GPU+CPU DD-αAMG solver

We have fully ported the SAP smoother in DD-αAMG from C to CUDA C. Hence, the current state of our solver[47] is hybrid: the smoother runs fully on MPI+CUDA and the remaining parts of the solver on MPI+OpenMP+SSE.

In fig. 5.4, two aggregation schemes are shown for DD-αAMG solves with a lattice of size 128×64^3 and three levels. The left-column scheme in that figure corresponds to a coarsening as usually employed in DD-αAMG. This traditional coarsening performs well when running on CPUs, and from chapter 4 we know its scalability is relatively poor when the number of nodes is relatively large and the coarsest level represents a quite large portion of the execution time[48]. Using our GPU-boosted smoother in such an aggregation scheme does not provide substantial gains, as the smoother does not represent much of the overall solve time; this is an observation that we will support with results from computational tests in this section. Furthermore, porting the whole solver to run on GPUs is also not a good idea, as the second and third levels will scale particularly bad on GPUs.

A better way to make good use of our smoother running on GPUs is to opt for a more aggressive coarsening, as displayed in the right column of fig. 5.4. By switching to this alternative aggregation scheme we redirect most of the execution time to the smoother at the finest level, with the crucial advantage that the total time spent at coarser levels is (even after turning GPU usage on) a relatively small percentage of the whole solve time.

[47] Which we have made available on GitHub: https://github.com/Gustavroot/DDalphaAMG
[48] This can be alleviated in many cases via our coarsest-level improvements from chapter 4.

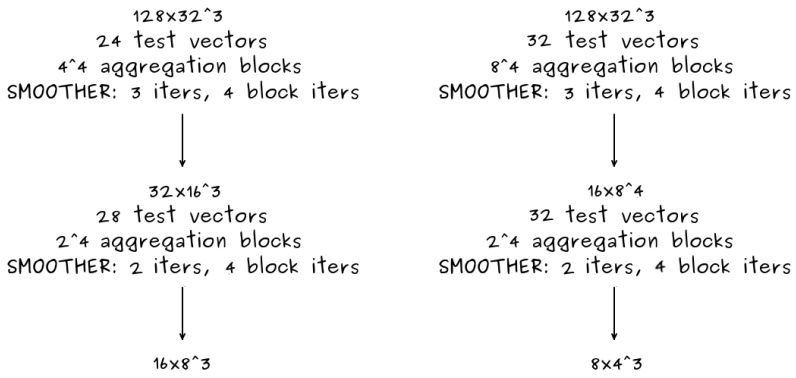

Figure 5.4: *Left*: CPU coarsening. *Right*: GPU coarsening.

We ran numerical experiments using both coarsening schemes presented in fig. 5.4, with the left one running fully on CPUs and the right one in a hybrid GPU+CPU manner. We have used the same clover-improved Wilson-Dirac configuration as in chapter 4. The DD-αAMG parameters are almost the same as in tab. 4.6, except for aggressive coarsening. In this latter case the parameters changed are as depicted on the right in fig. 5.4. The results are presented in fig. 5.5. We have performed tests with up to 256 GPUs. Both the CPU and hybrid solvers run with the coarsest-level improvements from chapter 4 already included[49]. The runs were performed on the booster module of the JUWELS supercomputer from the Jülich Supercomputing Centre[50].

Although the old (CPU) and new (GPU) results from fig. 5.5 are almost equivalent from the point of view of overall execution time, they are quite different when we take a closer look at the execution time spent on the individual components that make up the multigrid hierarchy. We do so in tab. 5.3, where GPU1 means running our hybrid solver with the coarsening on the right in fig. 5.4 (i.e. aggressive coarsening), and for GPU2 we have run the hybrid solver with the non-aggressive coarsening (see left in fig. 5.4). It is clear from this table that GPU1 is a much better alternative than GPU2, as we want to minimize the coarse-grid time due to its bad scalability and due to better possible performance improvements of the finest level via GPUs.

From the timings presented in tab. 5.3 it is clear that further improving the

[49] Although the impact of those coarsest-level improvements is not big here, as we are using the same mass parameter as in fig. 4.4

[50] Runs in the booster module have been done via the project cecy00 with title *Transverse momentum dependent soft function in lattice QCD.*

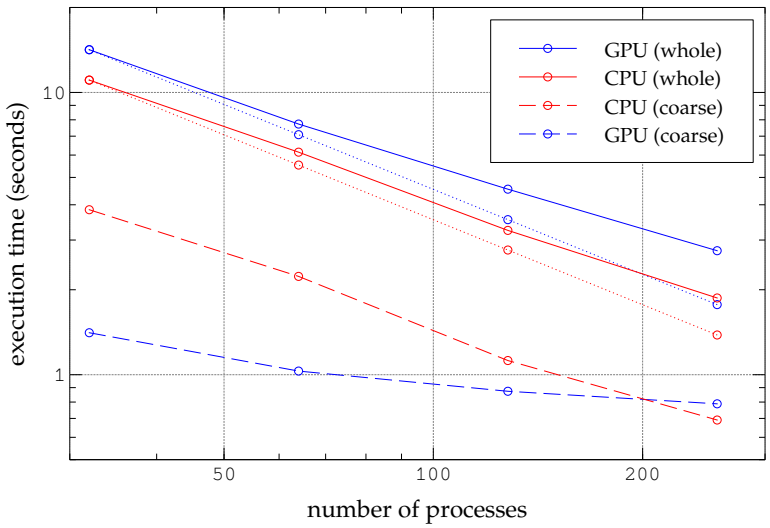

Figure 5.5: Strong scaling for $m_0 = -0.35371847789$ of the *old* version of DD-αAMG (before the GPU improvements) and the *new* (running on GPUs) version. The solid lines represent total execution time of a whole solve, and the dashed lines the time spent on coarser levels. The dotted line exemplifies how perfect scaling would look like in the hybrid solver. For CPU executions, 1 MPI process corresponds to 1 node, and for GPU executions we associate 4 MPI processes to each node with 1 GPU per MPI process.

	CPU	**GPU1**	**GPU2**
total solve time	6.14	7.73	7.7414
fine grid time	3.91 (64%)	6.7 (87%)	3.5367 (45.7%)
coarse grid time	2.23 (36%)	1.03 (13%)	4.2047 (54.3%)
smoother at $\ell = 1$	2.85	1.98	0.934
P and R at $\ell = 1$	0.62	2.89	1.59

Table 5.3: More detailed timings of some multigrid components in DD-αAMG, corresponding to the run with 64 processes from fig. 5.5. Times here are in seconds. Coarse grid time represents in this case the total time spent at $\ell = 2$ and $\ell = 3$ combined. The columns labeled as CPU and GPU1 correspond to the data displayed in fig. 5.5.

hybrid GPU+CPU implementation seems to be very promising. In particular, improvements via which we can possibly enhance the GPU version, are:

- Port the whole finest level, and not only the smoother at that level. The finest level is very rich in floating point operations, and communications are nicely hidden behind those. By merging the smoother at the finest level with all the other operations there, we can avoid frequent and large data

transfers from CPU to GPU and viceversa.

- Explore the use of lower precisions for the smoother at $\ell = 1$. Currently, our smoother runs in single precision, which we can further lower to half and with this take advantage of the good performance that GPUs offer at that precision.
- By using Tensor Cores[51] at different precisions throughout our implementations we can further improve the performance of the smoother at the finest level.

A clear suggestion from tab. 5.3 is that we keep coarser levels (i.e. $\ell > 1$) on CPUs, and only port the finest level from C to CUDA C, provided we do aggressive coarsening. Furthermore, another very interesting and useful consequence of the use of aggressive coarsening here is that the time spent on interpolation and restriction at $\ell = 1$ represents a large portion of the total execution time, which will be very beneficial for performance gains when we port the whole level $\ell = 1$ to be computed on GPUs.

GPU implementations tend to scale relatively poorly when the number of GPUs is increased too much. We expect this to happen also in our hybrid GPU+CPU code at some point. With this in mind, fig. 5.5 seems to suggest that a good strategy to optimize the use of computational resources might be to stay at 64 GPUs or less (for the lattice configuration under use here) and improve the finest level as much as possible via the advantages that GPUs provide.

5.4 Outlook on GPU implementations

From sect. 5.3.2 we can conclude that our approach of aggressive coarsening plus offloading the finest level onto GPUs, shows promise; most of the execution time is now being spent at the finest level. Furthermore, the total time spent at coarser levels in the case illustrated in tab. 5.3 is now around 1 second, which is a very low price to pay for such a large (128×64^3 in this case) lattice.

Based on the results of this chapter, we will then continue this work in the following directions, which we list in ascending order of priority, to improve our hybrid solver:

1. Usage of Tensor Cores.
2. Multiple precisions: we will employ different precisions on different parts of the finest level, e.g. the smoother in half precision.

[51] https://www.nvidia.com/en-us/data-center/tensor-cores/

3. Block solver: extension of our solver to act on multiple right hand sides "at once". The development of this will be within the context of the PhD work of Liam Burke (under supervision of Prof. Dr. Kirk M. Soodhalter at Trinity College Dublin), and the resulting code will then be used in lattice QCD computations via distillation, this in the context of the PhD work of Juan Urrea (under the supervision of Prof. Dr. Francesco Knechtli at Bergische Universität Wuppertal).
4. Coarsest level: when the coarsest level starts becoming representative again, the improvements from chapter 4, and furthermore the future work described in sect. 4.5, will be of immediate use here.

In developing the points above, we hew to the following constraints in our upcoming GPU developments for the hybrid DD-αAMG: stay at a relatively low number of nodes, e.g. 32 or 64; keep coarser levels (i.e. $\ell > 1$) running on CPUs; instead of attempting to have good strong scaling, try to reduce the execution time at the finest level as much as possible, via an offloading of finest-level operations onto GPUs. We will additionally perform numerical experiments to compare our solver with other alternatives out there, e.g. QUDA.

Chapter 6

Multigrid Multilevel Monte Carlo

In this chapter we develop and test a new method for the computation of the trace of a matrix function $f(A)$. Sects. 6.1 and 6.2.1 are largely based on our paper on multilevel Monte-Carlo for trace computations [1]. The application of our method to the Wilson operator, presented in sect. 6.2.2, comes from collaborative work with Jose Jiménez, which resulted in his M.Sc. thesis [133], hence that section here is partially based on his thesis.

As mentioned in sect. 2.4, the computation of disconnected diagrams in lattice QCD requires the extraction of the trace of $f(A) = \Gamma A^{-1}$. We consider here first the situation where one is interested in the trace $\mathrm{tr}(f(A))$ of a matrix function $f(A)$, in general. Here, $f(A) \in \mathbb{C}^{n\times n}$ is the matrix obtained from $A \in \mathbb{C}^{n\times n}$ and the function $f : z \in D \subseteq \mathbb{C} \to f(z) \in \mathbb{C}$ is the usual operator in the theoretic sense; see [134], e.g. Our focus is on the inverse A^{-1}, i.e. $f(z) = z^{-1}$. Computing the trace is an important task arising in many applications. The trace of the inverse is required, for example, in the study of fractals [135], in generalized cross-validation and its applications [136, 137]. In network analysis, the Estrada index—a total centrality measure for networks—is defined as the trace of the exponential of the adjacency matrix A of a graph [138, 139] and an analogous measure is given by the trace of the resolvent $(\rho I - A)^{-1}$ [140, Section 8.1]. For Hermitian positive definite matrices A, one can compute the log-determinant $\log(\det(A))$ as the trace of the logarithm of A. The log-determinant is needed in machine learning and related fields [141, 142]. Further applications are discussed in [143–145]. The particular application that we tackle in this chapter is the one described in sect. 2.4 i.e. the trace of the inverse of the discretized Dirac operator [146]. As simulation methods get more and more precise, these contributions become increasingly important.

It is usually unfeasible to compute the diagonal entries $f(A)_{ii}$ directly as $e_i^H f(A) e_i$,

e_i the ith canonical unit vector, and then obtain the trace by summation. For example, for the inverse this would mean that we have to solve n linear systems, which is prohibitive for large values of n.

One large class of methods which aims at circumventing this cost barrier are deterministic approximation techniques. Probing methods, for example, approximate

$$\mathrm{tr}(f(A)) \approx \sum_{i=1}^{N} w_i^H f(A) w_i, \tag{6.1}$$

where the vectors w_i are carefully chosen sums of canonical unit vectors and N is not too large. Various approaches have been suggested and explored in order to keep N small while at the same time achieving good accuracy in (6.1). This includes approaches based on graph colorings; see [147–149] e.g., and the hierarchical probing techniques from [41, 150]. In order for probing with such vectors to yield good results, the matrix $f(A)$ should expose a decay of the moduli of its entries when we move away from the diagonal, since the sizes of the entries farther away from the diagonal determine the accuracy of the approximation. Recent theoretical results in this direction were given in [151]. Lanczos techniques represent another deterministic approximation approach and are investigated in [152–154], e.g. Without giving details let us just mention that in order to improve their accuracy, deterministic approximation techniques can be combined with the stochastic techniques presented in the sequel; see [145], e.g.

In this chapter, we deal with the other big class of methods which aim at breaking the cost barrier using *stochastic estimation*. In general, they work for any matrix and, at least in principle, do not require a decay away from the diagonal. Our goal was to develop a multilevel Monte-Carlo method to estimate $\mathrm{tr}(f(A))$ stochastically, which we have accomplished. Our approach can be regarded as a variance reduction technique applied to the classical stochastic "Hutchinson" estimator [155]

$$\mathrm{tr}(f(A)) \approx \frac{1}{N} \sum_{n=1}^{N} (x^{(n)})^H f(A) x^{(n)}, \tag{6.2}$$

where the components of the random vectors $x^{(n)}$ obey an appropriate probability distribution. The variance of the estimator in eq. 6.2 decreases only like $\frac{1}{N}$, which makes the method too costly when higher precisions are to be achieved. The multilevel approach presented here aims at curing this by working with representations of A at different levels. On the higher numbered levels, evaluating $f(A)$ becomes increasingly cheap, while on the lower levels, which are more costly to evaluate, the variance is small. Our focus here is on the trace of the matrix inverse, where we can evaluate $A^{-1}x$ using a fast solver. We just note that for a general matrix function $f(A)$, stochastic trace estimation techniques can be combined with the Lanczos process to approximately evaluate the quadratic forms

$x^H f(A)x$; see, e.g. [153, 154].

This chapter is organized as follows: in sect. 6.1.1 we recall the general framework of multilevel Monte-Carlo estimators. In sect. 6.1.2 we then discuss Hutchinson's method for stochastically estimating the trace before turning to our new multilevel approach in sect. 6.1.3. This section also contains a comparison to known approaches based on deflation as a motivation of why the new multilevel method should provide additional efficiency. Several numerical results are presented in sects. 6.2.1, 6.2.2 and 6.2.3, for the Schwinger, Wilson and twisted mass operators, respectively.

6.1 Stochastic trace estimation and multilevel Monte Carlo

We establish the full theoretical foundation to develop a multigrid multilevel Monte Carlo method, including stochastic estimation via Huthinson's method and the multilevel Monte Carlo method, ultimately combining the two of them by assuming the existence of a multigrid hierarchy for the given problem matrix.

6.1.1 Multilevel Monte-Carlo

We discuss the basics of the multilevel Monte-Carlo approach as a variance reduction technique. We place ourselves in a general setting, thereby closely following [39].

Assume that we are given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with sample space Ω, sigma-algebra $\mathcal{F} \subseteq \Omega$ and probability measure $\mathbb{P} : \mathcal{F} \to [0, 1]$. For a given random variable $f : \Omega \to \mathbb{C}$, the *standard Monte-Carlo* approach estimates its expected value $\mathbb{E}[f]$ as the arithmetic mean

$$\mathbb{E}[f] \approx \frac{1}{M} \sum_{m=1}^{M} f(\omega^{(m)}), \tag{6.3}$$

where the $\omega^{(m)}$ are independent events coming from $(\Omega, \mathcal{F}, \mathbb{P})$. The variance of this estimator is $\frac{1}{M}\mathbb{V}[f]$, so the root mean square deviation has order $\mathcal{O}(M^{-1/2})$. This indicates that the number M of events has to increase quadratically with the accuracy required which is why, typically, higher accuracies require very high computational effort in this type of Monte-Carlo estimation.

The idea of *multilevel Monte-Carlo* is to split the random variable f as a sum

$$f = \sum_{\ell=1}^{L} g_\ell, \tag{6.4}$$

where the random variables $g_\ell : \Omega \to \mathbb{C}$ are regarded as contributions "at level ℓ" to f. This gives

$$\mathbb{E}[f] = \sum_{\ell=1}^{L} \mathbb{E}[g_\ell],$$

and an unbiased estimator for $\mathbb{E}[f]$ is obtained as

$$\mathbb{E}[f] \approx \sum_{\ell=1}^{L} \frac{1}{M_\ell} \sum_{m=1}^{M_\ell} g_\ell(\omega^{(m,\ell)}),$$

where the $\omega^{(m,\ell)}$ denote the independent events on each level. The variance of this estimator is

$$\sum_{\ell=1}^{L} \frac{1}{M_\ell} \mathbb{V}[g_\ell].$$

The idea is that we are able to find a multilevel decomposition of the form in eq. 6.4 in which the cost C_ℓ to evaluate g_ℓ is low when the variance $V_\ell := \mathbb{V}[g_\ell]$ is high and vice versa. As is explained in [39], the solution to the minimization problem which minimizes the total cost subject to achieving a given target variance ϵ^2

$$\text{minimize } \sum_{l=1}^{L} M_\ell C_\ell \quad \text{s.t. } \sum_{\ell=1}^{L} \frac{1}{M_\ell} V_\ell = \epsilon^2$$

gives $M_\ell = \mu\sqrt{V_\ell / C_\ell}$. Here, the Lagrangian multiplier μ satisfies $\mu = \epsilon^{-2} \sum_{\ell=1}^{L} \sqrt{V_\ell / C_\ell}$, and the corresponding minimal total cost is

$$C = \epsilon^{-2} \left(\sum_{\ell=1}^{L} \sqrt{V_\ell C_\ell} \right)^2.$$

The typical situation is that the contributions g_ℓ on level ℓ are given as differences $f_\ell - f_{\ell+1}$ of approximations f_ℓ to f on the various levels, i.e. we have

$$f = \sum_{\ell=1}^{L-1} \underbrace{(f_\ell - f_{\ell+1})}_{=g_\ell} + \underbrace{f_L}_{=g_L} \quad \text{with } f_1 = f. \tag{6.5}$$

If we assume that the cost $\hat{C}_\ell$ to evaluate f_ℓ decreases rapidly with the level ℓ, the cost C_ℓ for evaluating the differences $g_\ell = f_\ell - f_{\ell+1}$ is well approximated by $\hat{C}_\ell$. The ratio of the total cost encountered when reducing the variance to a given value between multilevel Monte-Carlo (with optimal choice of N_ℓ) and standard Monte-Carlo, see eq. 6.3, is then approximately given by

$$\left(\sum_{\ell=1}^{L} \sqrt{V_\ell \hat{C}_\ell}\right)^2 \Big/ \left(\mathbb{V}[f]\hat{C}_1\right).$$

This is the basic quantitative relation indicating how the costs $\hat{C}_\ell$ to evaluate the f_ℓ and the variances V_ℓ of the differences $f_\ell - f_{\ell+1}$ have to relate in order for the multilevel approach to be more efficient than standard Monte-Carlo estimation of f.

If the computations are being performed on a single compute node, then communications will (most likely) not play a big role in the overall execution time. Thus, the computational effort (i.e. execution time) can be modeled well by e.g. the number of nonzero elements in the matrices involved in the calculations. This is for example the case for the Schwinger matrix that we consider in sect. 6.2.1. In the main case under study here, i.e. matrices coming from lattice QCD discretizations, the situation is more difficult than this: as discussed in chapter 4, global reductions and nearest-neighbor communications induce a non-linear behaviour in terms of execution time as we increase the number of nodes, which is manifested more strongly for coarser levels. This renders the straighforward connection between execution time and arithmetic work made above not realistic. Furthermore, the assumption made on the rapid decrease in the cost at a certain level ℓ as we move down in the multigrid hierarchy is also invalid. The alternative is thus, when running on large-scale machines[52], to optimize the total computational effort in terms of the execution time at the various levels, rather than modeling via the arithmetic work (we have used the latter in the Schwinger case).

6.1.2 Stochastic estimation of the trace of a matrix

We now assume that we are given, in an indirect manner, a matrix $A = (a_{ij}) \in \mathbb{C}^{n\times n}$ for which we want to compute the trace

$$\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii}.$$

[52] Although our numerical tests in sects. 6.2.2 and 6.2.3 are on a single node, more realistic scenarios will later involve many nodes.

Our basic assumption is that the entries a_{ii} of A are neither available directly nor obtainable at decent computational cost. This is typically the case when A arises as a function of a large (and sparse) matrix, the most common case being the matrix inverse.

In a seminal paper [155], Hutchinson suggested to use a stochastic estimator to approximate $\mathrm{tr}(A)$. The following theorem summarizes his result together with the generalizations on the admissible probability spaces; see [156, 157], e.g.

Theorem 6.1.
Let $\mathbb{P} : \Omega \to [0,1]$ be a probability measure on a sample space Ω and assume that the components x_i of the vector $x \in \mathbb{C}^n$ are random variables depending on $\omega \in \Omega$ satisfying

$$\mathbb{E}[x_i] = 0 \quad \textit{and} \quad \mathbb{E}[\overline{x}_i x_j] = \delta_{ij} \quad \textit{(where } \delta_{ij} \textit{ is the Kronecker delta).} \tag{6.6}$$

Then

$$\mathbb{E}[x^H A x] = \mathrm{tr}(A) \textit{ and } \mathbb{V}[x^H A x] = \sum_{\substack{i,j,k,p=1 \\ i\neq j, k\neq p}}^{n} \overline{a}_{ij} a_{kp} \mathbb{E}[\overline{x}_i x_j \overline{x}_k x_p].$$

In particular, if the probability space is such that each component x_i is independent from x_j for $i \neq j$, then

$$\mathbb{V}[x^H A x] = \sum_{\substack{i,j \\ i\neq j}}^{n} \overline{a}_{ij} a_{ij} + \sum_{\substack{i,j \\ i\neq j}}^{n} \overline{a}_{ij} a_{ji} \mathbb{E}[x_i^2] \mathbb{E}[\overline{x}_j^2].$$

Proof. The proof is simple, but we repeat it here because the literature often treats only the real and not the general complex case. We have

$$\mathbb{E}[x^H A x] = \sum_{i=1}^{n} a_{ii} \mathbb{E}(\overline{x}_i x_i) + \sum_{i,j=1, i\neq j}^{n} a_{ij} \mathbb{E}(\overline{x}_i x_j) = \mathrm{tr}(A),$$

where the last inequality follows from eq. 6.6. Similarly

$$\begin{aligned}
\mathbb{V}[x^H A x] &= \mathbb{E}\left[\overline{(x^H A x - \mathrm{tr}(A))}(x^H A x - \mathrm{tr}(A))\right] \\
&= \mathbb{E}\big[\big(\sum_{\substack{i,j=1 \\ i\neq j}}^{n} x_i \overline{a}_{ij} \overline{x}_j\big)\big(\sum_{\substack{k,p=1 \\ k\neq p}}^{n} \overline{x}_k a_{kp} x_p\big)\big] \\
&= \mathbb{E}\big[\sum_{\substack{i,j,k,p=1 \\ i\neq j, k\neq p}}^{n} \overline{a}_{ij} a_{kp} x_i \overline{x}_j \overline{x}_k x_p\big] = \sum_{\substack{i,j,k,p=1 \\ i\neq j, k\neq p}}^{n} \overline{a}_{ij} a_{kp} \mathbb{E}[x_i \overline{x}_j \overline{x}_k x_p].
\end{aligned} \tag{6.7}$$

Since the components x_i are assumed to be independent, we have $\mathbb{E}[\overline{x}_i x_j \overline{x}_k x_p] = 0$ except when $i = j, k = p$ (which does not occur in eq. 6.7) or $i = k, j = p$ or $i = p, j = k$. This gives

$$\sum_{\substack{i,j,k,p=1 \\ i\neq j, k\neq p}}^{n} \overline{a}_{ij} a_{kp} \mathbb{E}[x_i \overline{x}_j \overline{x}_k x_p] = \sum_{\substack{i,j \\ i\neq j}}^{n} \overline{a}_{ij} a_{ij} \mathbb{E}[x_i \overline{x}_j \overline{x}_i x_j] + \sum_{\substack{i,j \\ i\neq j}}^{n} \overline{a}_{ij} a_{ji} \mathbb{E}[x_i \overline{x}_j \overline{x}_j x_i],$$

and in the first sum $\mathbb{E}[x_i \overline{x}_j \overline{x}_i x_i] = \mathbb{E}[\overline{x}_i x_i]\mathbb{E}[\overline{x}_j x_j] = 1$ by assumption, whereas in the second sum we have $\mathbb{E}[x_i \overline{x}_j \overline{x}_j x_i] = \mathbb{E}[x_i^2]\mathbb{E}[\overline{x}_j^2]$. □

Note that as a definition for the variance of a complex random variable y we used $\mathbb{E}[\overline{(y - \mathbb{E}(y)}(y - \mathbb{E}[y]]$ rather than $\mathbb{E}[(y - \mathbb{E}[y])^2]$ to keep it real and non-negative.

Standard choices for the probability spaces are to take x with identically and independently distributed (i.i.d.) components as

$$x_i \in \{-1, 1\} \text{ with equal probability } \tfrac{1}{2}, \tag{6.8}$$
$$x_i \in \{-1, 1, -i, i\} \text{ with equal probability } \tfrac{1}{4}, \tag{6.9}$$
$$x_i = \exp(i\theta) \text{ with } \theta \text{ uniformly distributed in } [0, 2\pi], \tag{6.10}$$
$$x_i \text{ is } N(0, 1) \text{ normally distributed.} \tag{6.11}$$

Corollary 6.2.
If the components x_i are i.i.d. with the distribution in eq. 6.8 or eq. 6.11, then

$$\mathbb{V}[x^H A x] = \frac{1}{2} \|\text{offdiag}(A + A^T)\|_F^2,$$

where $\|\cdot\|_F$ denotes the Frobenius norm and offdiag *the offdiagonal part of a matrix. If the components are i.i.d. with the distribution in eq. 6.9 or 6.10, then*

$$\mathbb{V}[x^H A x] = \|\text{offdiag}(A)\|_F^2.$$

Proof. For the distributions in eqs. 6.8 and 6.11, the components x_i have only real values and $\mathbb{E}[x_i^2] = 1$. Therefore

$$\begin{aligned}
\sum_{\substack{i,j \\ i\neq j}}^{n} \overline{a}_{ij} a_{ij} + \sum_{\substack{i,j \\ i\neq j}}^{n} \overline{a}_{ij} a_{ji} \mathbb{E}[x_i^2]\mathbb{E}[\overline{x}_j^2] &= \sum_{\substack{i,j \\ i\neq j}}^{n} \overline{a}_{ij} a_{ij} + \sum_{\substack{i,j \\ i\neq j}}^{n} \overline{a}_{ij} a_{ji} \\
&= \frac{1}{2} \sum_{\substack{i,j \\ i\neq j}}^{n} (\overline{a_{ij} + a_{ji}})(a_{ij} + a_{ji}) \\
&= \frac{1}{2} \|\text{offdiag}(A + A^T)\|_F^2.
\end{aligned}$$

For the distributions in eqs. 6.9 and 6.10 we have $\mathbb{E}[x_i^2] = 0$, and thus

$$\sum_{\substack{i,j \\ i \neq j}}^{n} \overline{a}_{ij} a_{ij} + \sum_{\substack{i,j \\ i \neq j}}^{n} \overline{a}_{ij} a_{ji} \mathbb{E}[x_i^2]\mathbb{E}[\overline{x}_j^2] = \sum_{\substack{i,j \\ i \neq j}}^{n} \overline{a}_{ij} a_{ij} = \|\text{offdiag}(A)\|_F^2.$$

□

In a practical situation where we approximate $\text{tr}(A)$ by averaging over N samples we can compute the sample root mean square deviation along with the averages and rely on the law of large numbers to assess the probability that the computed mean lies within the σ, 2σ or 3σ interval. Several results on Hutchinson's method have been formulated which go beyond this asymptotic aspects by giving tail or concentration bounds; see [158–160], e.g. For the sake of illustration we here report a summary of these results as given in [143]. In our numerical examples, we will simply work with the sample root mean square deviation to assess accuracy.

Theorem 6.3.
Let the distribution for the i.i.d. components of the random vectors $x^{(m)}$ be sub-Gaussian, and let $\epsilon, \delta \in (0,1)$. Then for $M = \mathcal{O}(\log(1/\delta)/\epsilon^2)$ we have that the probability for

$$\left| \frac{1}{M} \sum_{m=1}^{M} (x^{(m)})^H A x^{(m)} - \text{tr}(A) \right| \leq \epsilon \|A\|_F \tag{6.12}$$

is $\geq 1 - \delta$.

Note that if A is symmetric positive semidefinite with λ_i denoting its (non-negative) eigenvalues, then

$$\|A\|_F = \left(\sum_{i=1}^{n} \lambda_i^2 \right)^{1/2} \leq \sum_{i=1}^{n} \lambda_i = \text{tr}(A),$$

implying that eq. 6.12 yields a (probabilistic) relative error bound for the trace. Also note that the real distributions in eqs. 6.8 and 6.11 are sub-Gaussian [143].

6.1.3 Multilevel Monte-Carlo for the trace of the inverse

We now turn to the situation where we want to estimate $\text{tr}(A^{-1})$ for a large and sparse matrix A. Direct application of theor. 6.1 shows that an unbiased estimator for $\text{tr}(A^{-1})$ is given by

$$\frac{1}{M} \sum_{m=1}^{M} x^{(m)H} A^{-1} x^{(m)} \approx \text{tr}(A^{-1}), \tag{6.13}$$

where the vectors $x^{(m)}$ are independent random variables satisfying eq. 6.6, and that its variance is

$$\frac{1}{M}\|\text{offdiag}(A^{-1} + A^{-T})\|_F^2 \quad \text{or} \quad \frac{1}{M}\|\text{offdiag}(A^{-1})\|_F^2,$$

depending on whether the components of $x^{(m)}$ satisfy eqs. 6.8, 6.11 or eqs. 6.9, 6.10, respectively.

Each time we add a sample m to eq. 6.13 we have to solve a linear system with matrix A and right hand side $x^{(m)}$, and the cost for solving these linear systems determines the cost for each stochastic estimate. For a large class of matrices, multigrid methods represent particularly efficient linear solvers. We assume that this is the case for our matrix A and now describe how to derive a multilevel Monte-Carlo method for the approximation of $\text{tr}(A^{-1})$ which uses the multigrid hierarchy not only for the linear solver, but also to obtain a good representation as in eq. 6.5 required for a multilevel Monte-Carlo approach.

6.1.3.1 Derivation of a multilevel Monte-Carlo method

Multigrid methods rely on the interplay between a smoothing iteration and a coarse grid correction which are applied alternatingly. In the geometric interpretation, where we view components of vectors as representing a continuous function on a discrete grid, the smoother has the property that it makes the error of the current iterate smooth, i.e. varying slowly from one grid point to the next. Such error can be represented accurately by a coarser grid, and the coarse grid correction solves for this coarse error on the coarse grid using a coarse grid representation of the matrix. The solution is then interpolated back to the original "fine" grid and applied as a correction to the iterate. The principle can be applied recursively using a sequence of coarser grids with corresponding operators, the solves on the coarsest grid being obtained by direct factorization.

To obtain a multilevel Monte-Carlo decomposition we discard the smoother and only consider the coarse grid operators and the intergrid transfer operators. The coarse grid operators

$$A_\ell \in \mathbb{C}^{n_\ell \times n_\ell}, \ell = 1, \ldots, L,$$

the prolongation and restriction operators

$$P_\ell \in \mathbb{C}^{n_\ell \times n_{\ell+1}},\ R_\ell \in \mathbb{C}^{n_{\ell+1} \times n_\ell},\ \ell = 1, \ldots, L-1,$$

and the coarse system matrices

$$A_{\ell+1} = R_\ell A_\ell P_\ell,\ \ell = 1, \ldots, L-1,$$

have all been introduced, in general, in chapter 3.

Using the accumulated prolongation and restriction operators

$$\hat{P}_\ell = P_1 \cdots P_{\ell-1} \in \mathbb{C}^{n\times n_\ell}, \hat{R}_\ell = R_{\ell-1} \cdots R_1 \in \mathbb{C}^{n_\ell \times n},\ \ell = 1, \ldots, L,$$

where we put $\hat{R}_1 = \hat{P}_1 = I \in \mathbb{C}^{n\times n}$ by convention, we regard $\hat{P}_\ell A_\ell^{-1} \hat{R}_\ell$ as the approximation to A^{-1} at level ℓ. We thus obtain a multilevel decomposition for the trace as

$$\mathrm{tr}(A^{-1}) = \sum_{\ell=1}^{L-1} \mathrm{tr}\left(\hat{P}_\ell A_\ell^{-1} \hat{R}_\ell - \hat{P}_{\ell+1} A_{\ell+1}^{-1} \hat{R}_{\ell+1}\right) + \mathrm{tr}(\hat{P}_L A_L^{-1} \hat{R}_L). \tag{6.14}$$

This gives

$$\mathrm{tr}(A^{-1}) = \sum_{\ell=1}^{L-1} \mathbb{E}\left[(x^\ell)^H \left(\hat{P}_\ell A_\ell^{-1} \hat{R}_\ell - \hat{P}_{\ell+1} A_{\ell+1}^{-1} \hat{R}_{\ell+1}\right) x^\ell\right] + \mathbb{E}\left[(x^L)^H \hat{P}_L A_L^{-1} \hat{R}_L x^L\right],$$

with the components of $x^\ell \in \mathbb{C}^n$ being i.i.d. stochastic variables satisfying eq. 6.6. The unbiased multilevel Monte-Carlo estimator is then

$$\begin{aligned}\mathrm{tr}(A^{-1}) \approx & \sum_{\ell=1}^{L-1} \frac{1}{M_\ell} \sum_{m=1}^{M_\ell} \left((x^{(m,\ell)})^H \hat{P}_\ell A_\ell^{-1} \hat{R}_\ell x^{(m,\ell)} - (x^{(m,\ell)})^H \hat{P}_{\ell+1} A_{\ell+1}^{-1} \hat{R}_{\ell+1} x^{(m,\ell)}\right) \\ & + \frac{1}{M_L} \sum_{i=1}^{M_L} (x^{(m,L)})^H \hat{P}_L A_L^{-1} \hat{R}_L x^{(m,L)},\end{aligned}$$

where the vectors $x^{(m,\ell)} \in \mathbb{C}^n$ are stochastically independent samples of the random variable $x \in \mathbb{C}^n$ satisfying eq. 6.6.

The following remarks collect some important observations about this stochastic estimator.

Remark 6.4.
Computationally, the estimator requires to solve systems of the form $A_\ell y^{(m,\ell)} = z$ with $z = \hat{R}_\ell x^{(m,\ell)}$. Since the matrices A_ℓ arise from the multigrid hierarchy, we directly have a multigrid method available for these systems by restricting the method for A to the levels $\ell, \ldots, L$.

Remark 6.5.
Since for any two matrices $B = (b_{ij}) \in \mathbb{C}^{n\times m}$ and $C = (c_{kl}) \in \mathbb{C}^{m\times n}$, the trace of their product does not depend on the order,

$$\mathrm{tr}(BC) = \sum_{i=1}^{n} \sum_{j=1}^{m} b_{ij} c_{ji} = \sum_{j=1}^{m} \sum_{i=1}^{n} c_{ji} b_{ij} = \mathrm{tr}(CB), \tag{6.15}$$

we have

$$\mathrm{tr}(\hat{P}_L A_L^{-1} \hat{R}_L) = \mathrm{tr}(A_L^{-1} \hat{P}_L \hat{R}_L).$$

So, instead of estimating the contribution $\mathrm{tr}(\hat{P}_L A_L^{-1} \hat{R}_L)$ *in eq. 6.14 stochastically, we can also compute it directly by inverting the matrix* $A_L \in \mathbb{C}^{n_L \times n_L}$ *and computing the product* $A_L^{-1} \hat{R}_L \hat{P}_L$*. Note that* $\hat{R}_L$ *and* $\hat{P}_L$ *are usually sparse with a maximum of* d*, say, non-zero entries per row. The arithmetic work for* $A_L^{-1} \hat{R}_L \hat{P}_L$ *is thus of order* $\mathcal{O}(dn_L^2)$ *for the product* $\hat{R}_L \hat{P}_L$ *plus* $\mathcal{O}(n_L^3)$ *for the inversion of* A_L *and the product* $A_L^{-1}(\hat{R}_L \hat{P}_L)$*. Since the variance of* $x^H \hat{P}_L A_L^{-1} \hat{R}_L x$ *is presumably large, this direct computation can be much more efficient than a stochastic estimation, even when we aim at only quite low precision in the stochastic estimate.*

The direct inversions suggested in remark 6.4 are of important use when the coarsest-level is small enough and communications over many nodes is not an issue. This is the case for the numerical experiments with the Schwinger model in sect. 6.2.1, where we run on a single node and the coarsest-level is indeed small enough. This is not the case, though, for the tests presented in sects. 6.2.2 and 6.2.3 in the context of lattice QCD, where although we run on a single node, the coarsest-level matrices are too large to invert directly.

Remark 6.6.
There are situations where $\hat{R}_\ell \hat{P}_\ell = I \in \mathbb{C}^{n_\ell \times n_\ell}$*, for example in aggregation based multigrid methods, where the columns of* P_ℓ *are orthonormal and* $R_\ell = P_\ell^H$*, see [108, 161]. Then*

$$\mathrm{tr}(\hat{P}_\ell A_\ell^{-1} \hat{R}_\ell) \;=\; \mathrm{tr}(A_\ell^{-1} \hat{R}_\ell \hat{P}_\ell) \;=\; \mathrm{tr}(A_\ell^{-1}),$$

and

$$\mathrm{tr}(\hat{P}_{\ell+1} A_{\ell+1}^{-1} \hat{R}_{\ell+1}) \;=\; \mathrm{tr}(\hat{P}_\ell P_\ell A_{\ell+1}^{-1} R_\ell \hat{R}_\ell) \;=\; \mathrm{tr}(P_\ell A_{\ell+1}^{-1} R_\ell \hat{R}_\ell \hat{P}_\ell) \;=\; \mathrm{tr}(P_\ell A_{\ell+1}^{-1} R_\ell).$$

This means that instead of the multilevel decomposition in eq. 6.14 we can use

$$\mathrm{tr}(A) = \sum_{\ell=1}^{L-1} \mathrm{tr}\left(A_\ell^{-1} - P_\ell A_{\ell+1}^{-1} R_\ell\right) + \mathrm{tr}(A_L^{-1}),$$

in which the stochastic estimation on level ℓ *now involves random vectors from* $\mathbb{C}^{n_\ell}$ *instead of* $\mathbb{C}^n$*.*

6.1.3.2 Discussion of the multilevel Monte-Carlo method

A profound analysis of the proposed multilevel Monte-Carlo method must take the approximation properties of the representation of the matrix at the various

levels into account. This is highly problem dependent, and although we know the properties of the specific problems at hand (i.e. Schwinger and lattice QCD), formulating such an analysis for the multilevel Monte-Carlo method introduced here, applied to these particular problems, remains a difficult task. So, here we only provide a discussion of heuristics on why the proposed approach has the potential to yield efficient multilevel Monte-Carlo schemes.

To simplify the discussion to follow, let us assume that the variance of the estimator at level ℓ is given by the square of the Frobenius norm of the off-diagonal part. This is the case, for example, if the components are i.i.d. with the distribution in eq. 6.9 or 6.10; see coroll. 6.2. This Frobenius norm can be related to the singular values of A. Recall that the singular value decomposition of a non-singular matrix A is

$$\begin{aligned} A &= U\Sigma V^H \quad \text{with } U,\Sigma,V \in \mathbb{C}^{n\times n}, U^HU = V^HV = I, \\ &\quad U = [u_1|\cdots|u_n],\ V = [v_1|\ldots|v_n], \\ &\quad \Sigma = \operatorname{diag}(\sigma_1,\ldots,\sigma_n), 0 < \sigma_1 \leq \cdots \leq \sigma_n, \end{aligned} \tag{6.16}$$

with left singular vectors u_i, right singular vectors v_i and positive singular values σ_i which we ordered by increasing value for convenience here. In the following we base all our discussion on singular values and vectors. It is therefore worthwhile to mention that in the case of a Hermitian matrix A this discussion simplifies in the sense that then the singular values are the moduli of the eigenvalues, and left and right singular vectors are identical and coincide with the eigenvectors.

If $A \in \mathbb{C}^{n\times n}$ has singular values $\sigma_i, i = 1,\ldots,n$, then

$$\|\operatorname{offdiag}(A)\|_F^2 = \sum_{i=1}^{n} \sigma_i^2 - \sum_{i=^1}^{n} |a_{ii}|^2, \tag{6.17}$$

since $\|A\|_F^2 = \sum_{i=1}^n \sigma_i^2$; see, e.g. [83]. For the trace of the inverse A^{-1} we thus have

$$\|\operatorname{offdiag}(A^{-1})\|_F^2 = \sum_{i=1}^{n} \sigma_i^{-2} - \sum_{i=^1}^{n} |(A^{-1})_{ii}|^2. \tag{6.18}$$

since the reciprocals of the singular values of A, are the singular values of A^{-1}. Therefore, in a simplified manner—disregarding the second term in eq. 6.18—we hypothesize that the small singular values of A are those who contribute most to the variance for the Hutchinson estimator, see eq. 6.13, for $\operatorname{tr}(A^{-1})$. In high performance computing practice, *deflation* has thus become a common tool, see [40, 148, 162, 163], e.g., to reduce the variance: One precomputes the k, say, smallest singular values $\sigma_1,\ldots,\sigma_k$ of A in the singular value decomposition, see eq. 6.16, together with their left singular vectors $u_1,\ldots,u_k$. With the orthogonal

projector

$$\Pi = U_k U_k^H, \text{ where } U_k = [u_1|\cdots|u_k], \tag{6.19}$$

we now have $A^{-1} = A^{-1}(I - \Pi) + A^{-1}\Pi$ with

$$A^{-1}(I - \Pi) = \sum_{i=k+1}^{n} v_i \sigma_i^{-1} u_i^H, \quad A^{-1}\Pi = \sum_{i=1}^{k} A^{-1} u_i u_i^H. \tag{6.20}$$

This shows that in $A^{-1}(I - \Pi)$ we have deflated the small singular values of A, so that we can expect a reduction of the variance when estimating the trace of this part stochastically. The trace of the second part is equal to the sum $\sum_{i=1}^{k} u_i^H A^{-1} u_i$ (see eq. 6.15), and $A^{-1}u_i = \sigma_i^{-1} v_i$. So the second part can be computed directly from the singular triplets computed for the deflation. If A is Hermitian, the deflation approach simplifies and amounts to precomputing the k smallest eigenpairs. We refer to the results in [40] for a more in-depth analysis and discussion about the heuristics just presented.

The deflation approach is still quite costly, since one has to precompute the singular values and vectors, and if the size of the matrix increases it is likely that we have to increase k to maintain the same reduction in the variance. Approximate deflation has thus been put forward as an alternative [164, 165], where one can use larger values for k while at the same time allowing that the contribution of the small singular values to the variance is eliminated only approximately. One then replaces Π by a more general projector of the form

$$\Pi = \hat{U}_k (\hat{V}_k^H A \hat{U}_k)^{-1} \hat{V}_k^H A, \;\; \hat{U}_k, \hat{V}_k \in \mathbb{C}^{n \times k}$$

where now $\hat{U}_k$ and $\hat{V}_k$ can be regarded as containing approximate left and right singular vectors, respectively, as their columns. Actually, it is sufficient that their range is spanned by such approximations to left and right singular vectors, since the construction of Π is invariant under transformations $\hat{U} \to \hat{U}B_U, \hat{V} \to \hat{V}B_V$ with non-singular matrices $B_U, B_V \in \mathbb{C}^{k \times k}$. In the decomposition $A^{-1} = A^{-1}(I - \Pi) + A^{-1}\Pi$ we now have, again using eq. 6.15,

$$\begin{aligned} \operatorname{tr}(A^{-1}(I - \Pi)) &= \operatorname{tr}(A^{-1}) - \operatorname{tr}(\hat{U}_k (\hat{V}_k^H A \hat{U}_k)^{-1} \hat{V}_k^H), \\ \operatorname{tr}(A^{-1}\Pi) &= \operatorname{tr}(\hat{U}_k (\hat{V}_k^H A \hat{U}_k)^{-1} \hat{V}_k^H). \end{aligned}$$

If k is relatively small, the second trace can be computed directly as in the exact deflation approach. If we take larger values for k, we can estimate it stochastically. The inexact deflation approach then becomes a two-level Monte-Carlo method.

If we look at our multilevel Monte-Carlo decomposition in eq. 6.4 with just two levels, then it differs from inexact deflation in that the value for k is now very large, namely the grid size at level 2 which usually is $\mathcal{O}(n)$. The matrix $\hat{U}_k$ spanning

the approximate singular vectors is replaced by the prolongation operator P_1, and $\hat{V}_k^H$ corresponds to the restriction operator R_1. The multigrid construction principle should ensure that the range of P_1 contains good approximations to $\mathcal{O}(n)$ left singular vectors belonging to small singular values, and similarly for R_1^H with respect to right singular vectors. This is why the variance reduction can be expected to be efficient. We thus have a large value of k—proportional to n—which targets at a high reduction of the variance of the first term. The second term involves the second level matrix representation, which is still of large size, and its trace estimator will, in addition, still have large variance. This is the reason why we extend the approach to involve many levels, ideally until a level L where we can compute the trace directly, so that we do not suffer from a potentially high variance of a stochastic estimator. In the numerical results to be reported in sect. 6.2, the variance is exposed via the number of stochastic estimates required to obtain a given target accuracy. In all examples, the number of stochastic estimates is small on the finer levels and increases substantially on the coarser levels. So the numerical examples experimentally confirm the above theoretical motivation.

To conclude this discussion, we note that several other techniques for variance reduction have been suggested which can also be regarded as two-level Monte-Carlo techniques. For example, [166, 167] take a decomposition $A^{-1} - p(A) + p(A)$ with an appropriately chosen polynomial $p(A)$ and then estimates $\mathrm{tr}(A^{-1} - p(A))$ stochastically. The "truncated solver" method of [168] follows a related idea by subtracting an approximation to the inverse. A similar decomposition with p being a truncated Chebyshev series approximation was considered in [144, 169, 170], for example, in which case $\mathrm{tr}(A^{-1} - p(A))$ is actually neglected. The work then resides in the stochastic estimation of $\mathrm{tr}(p(A))$, thus avoiding to solve linear systems.

Finally, we refer to [143] for a recent further variance reduction technique for Hutchinson's method, enhancing it by using vectors of the form $A^{-1}v$ with random vectors v.

6.2 Numerical tests

In this section, we perform numerical experiments to test the performance of the method introduced in sect. 6.1.3. We first test the method on the Schwinger operator in sect. 6.2.1, and then we proceed to further test it with two different lattice QCD discretizations in sects. 6.2.2 and 6.2.3. The numerical tests in sect. 6.2.1 are the same ones described in ref. [1], and the ones in sect. 6.2.2 the same as in ref. [133].

The function for which we compute the trace is $\mathrm{tr}(f(A)) = A^{-1}$ for all the three matrices used in this section.

6.2.1 Schwinger model

In the Schwinger case, the improvements of the multilevel approach compared to "plain" Hutchinson (eq. 6.13) are tremendous and typically reach two orders of magnitude or more. This is why we compare against *deflated Hutchinson*, where we deflate the n_{defl} smallest eigenpairs of the matrix A. With $U \in \mathbb{C}^{n \times n_{\mathrm{defl}}}$ holding the respective eigenvectors in its columns, we use the projector $\Pi = I - UU^H$ as in eq. 6.19, resulting in the decomposition eq. 6.20. Therein we estimate $\mathrm{tr}(A^{-1}(I - \Pi))$ with the Hutchinson estimator whereas $\mathrm{tr}(A^{-1}\Pi) = \sum_{i=1}^{n_{\mathrm{defl}}} \lambda_i^{-1}$ is obtained directly from the deflated eigenpairs. We always performed a rough scan to determine a number n_{defl} of deflated eigenpairs which is close to time-optimal. The deflated Hutchinson approach usually gains at least one order of magnitude in time and arithmetic cost over plain Hutchinson.

All our Schwinger computations were done on a single thread of an Intel Xeon Processor E5-2699 v4, with a Python implementation. We aimed in this case at a relative accuracy of $\epsilon = 10^{-3}$. This is done as follows: We first perform five stochastic estimates, take their mean and subtract their sample root mean square deviation, giving the value τ. In the deflated Hutchinson method we now perform stochastic estimates with sampling vectors $x^{(n)}$ as long as their sample root mean square deviation

$$\sqrt{\frac{1}{M}\sum_{m=1}^{M}\left((x^{(m)})^H A^{-1} x^{(m)} - \bar{t}_M\right)^2}, \quad \text{where } \bar{t}_M = \frac{1}{M}\sum_{n=1}^{M}(x^{(m)})^H A^{-1} x^{(m)}$$

exceeds $\epsilon\tau$. For the multilevel Monte-Carlo method we have to prescribe a value for the sample root mean square deviations ρ_ℓ for the stochastic estimation of each of the traces

$$\mathrm{tr}\left(\hat{P}_\ell A_\ell^{-1}\hat{R}_\ell - \hat{P}_{\ell+1}A_{\ell+1}^{-1}\hat{R}_{\ell+1}\right), \quad \ell = 1, \ldots, L-1, \tag{6.21}$$

from eq. 6.14, while we always compute the last term $\mathrm{tr}(\hat{P}_L A_L^{-1}\hat{R}_L)$ in eq. 6.14 non-stochastically as $\mathrm{tr}(A_L^{-1}\hat{R}_L\hat{P}_L)$, inverting A_L explicitly. The requirement is to have

$$\sum_{\ell=1}^{L-1}\rho_\ell^2 = (\epsilon\tau)^2,$$

so the obvious approach is to put $\rho_\ell = \epsilon\tau/\sqrt{L-1}$ for all ℓ. It might be advantageous, though, to allow for a larger value of ρ_ℓ on those level differences where

the cost is high, and we do so in this section for the Schwinger model. To prevent a possible unlucky severe under-estimation of the exact mean square deviation by the sample mean square deviation, we always perform at least five stochastic estimates for each ℓ in eq. 6.21.

For each stochastic estimate for eq. 6.21 we have to solve linear systems with the matrices A_ℓ and $A_{\ell+1}$. This is done using a multigrid method based on the same prolongations P_ℓ, restrictions R_ℓ and coarse grid operators A_ℓ that we use to obtain our multilevel decomposition in eq. 6.14. However, when multigrid is used as a solver, we use the full hierarchy going down to coarse grids of very small sizes, whereas in the multilevel decomposition (eq. 6.14) used in multilevel Monte-Carlo we might stop at an earlier level; we do not do this in our computations, though, because the number of levels is relatively small in our examples.

For the Schwinger case in this section, we report mainly two quantities. The first is the number of stochastic estimates that are performed at each level difference in eq. 6.21 for multilevel Monte-Carlo together with the number of stochastic estimates in deflated Hutchinson (which always requires linear solves at the finest level). These numbers may be interpreted as illustrating how multilevel Monte-Carlo moves the higher variances to the coarser level differences. As a second quantity, we report the approximate arithmetic cost for both methods, deflated Hutchinson and multilevel Monte-Carlo, which we obtain using the following cost model[53] for the computation of a quantity[54] $x^H P_\ell A_\ell^{-1} R_\ell x$: We only consider the matrix-vector products occurring in this computation. These arise through multiplications with P_ℓ and R_ℓ and through the matrix-vector multiplications that we perform in the multigrid solver that we use to compute $A_\ell^{-1} y$. For every matrix-vector product of the generic form Bx we assume a cost of $\mathrm{nnz}(B)$, the number of nonzeros in B. In this manner, one unit in the cost model roughly corresponds to a multiplication plus an addition. This applies to the computation of residuals, of prolongations and restrictions and the coarsest grid solve in the multigrid solver as well as to the "global" restrictions and prolongations $\hat{R}_\ell, \hat{P}_\ell$ used in each stochastic estimate in multilevel Monte-Carlo. For the latter method, we also count the cost for the direct computation of the trace at the coarsest level, which involves the inversion of the coarsest grid matrix and additional matrix-matrix products. This cost model thus only neglects vector-vector and scalar operations and is thus considered sufficiently accurate for our purposes.

[53] This cost model applies here for the Schwinger case only. For this particular problem, we have used V-cycles in its corresponding multigrid solver, instead of the K-cycles currently under use in DD-αAMG. We do not have such a cost model for the solves in DD-αAMG, which involves a relatively complicated formulation due to the combination of K-cycles and communications when we go to a large number of nodes.

[54] For both problems here, Schwinger and lattice QCD, we used a standard aggregation (see def. 3.17), which thanks to remark 6.6 allows us to avoid accumulated interpolation and restriction operators $\hat{P}_\ell$ and $\hat{R}_\ell$ and use P_ℓ and R_ℓ instead.

We used a Schwinger matrix arising from a thermalized configuration within a Markov process. This guarantees that the random gauge links obey a Boltzmann distribution with a given temperature parameter. The matrix belongs to an $N \times N$ lattice with $N = 128$, and is thus of size $2N^2 \times 2N^2 = 32,768 \times 32,768$.

The multigrid hierarchy for the Schwinger matrix was obtained through the aggregation based approach described in chapter 3: at each level, the operator represents a periodic nearest neighbor coupling an a 2-dimensional lattice of decreasing size. At each lattice site we have several, d say, degrees of freedom (dofs), i.e. variables belonging to a lattice site are vectors of length d. When going from one level to the next, we subdivide the lattice into small sublattices—the aggregates. Each aggregate becomes a single lattice site on the next level. The corresponding restriction operator is obtained by computing (quite inexact) approximations to the d smallest eigenvectors, the components of which are assembled over the aggregates and orthogonalized. This gives restriction operators which are orthonormal, and since we take the prolongations to be their adjoints, we are in the simplified situation of remark 6.6 for estimating the traces of the differences in multilevel Monte-Carlo.

The Schwinger matrix is not Hermitian, and from lemma 2.12 its eigenvalues come in complex conjugate pairs. This spin symmetry can be preserved on the coarser levels if one doubles the dofs, as was explained in chapter 3 (see in particular def. 3.17 in there).

We built a multigrid hierarchy with four levels. For the aggregates, at all levels we always aggregated 4×4 sublattices into one lattice site on the next level, and we started with 2 dofs for the second level and 4 for all remaining levels. Those dofs are then doubled because we implemented the spin structure preserving approach. Tab. 6.1 summarizes the most important information on the multigrid hierarchy. It also shows the five different (negative) values for the mass m that we used in our experiments. These values are physically meaningful, and for all of them the spectrum of S^N is contained in the right half plane. As m becomes smaller, S^N becomes more ill-conditioned, so the cost for each stochastic estimate increases. When solving linear systems at the various levels, we used one V-cycle of multigrid with two steps of Gauss-Seidel pre- and post-smoothing as a preconditioner for flexible GMRES [34]. Our implementation was done in Python[55], and the relative tolerance for the solves at each level was set to 10^{-7}.

Fig. 6.1 shows our results. We tuned the required sample root mean square deviation ρ_ℓ at each level due to the observation that this time the sample root mean square deviation is comparably small on the last level difference. The values we chose, independently of the mass parameter m, are $\rho_1 = \sqrt{0.4}\epsilon\tau$, $\rho_2 = \sqrt{0.55}\epsilon\tau$ and $\rho_3 = \sqrt{0.05}\epsilon\tau$ for all masses.

[55]The code can be found in the GitHub repository https://github.com/Gustavroot/MLMCTraceComputer

Schwinger model						
N		$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$	L
128	n_ℓ	$2 \cdot 128^2$	$4 \cdot 32^2$	$8 \cdot 8^2$	$8 \cdot 2^2$	4
	$\mathrm{nnz}(S_\ell^N)$	2.94e5	1.64e5	2.46e4	1024	
m		-0.1320	-0.1325	-0.1329	-0.1332	-0.1333
n_{defl}		384	384	512	512	512

Table 6.1: Parameters and quantities for the numerical experiments with the Schwinger operator.

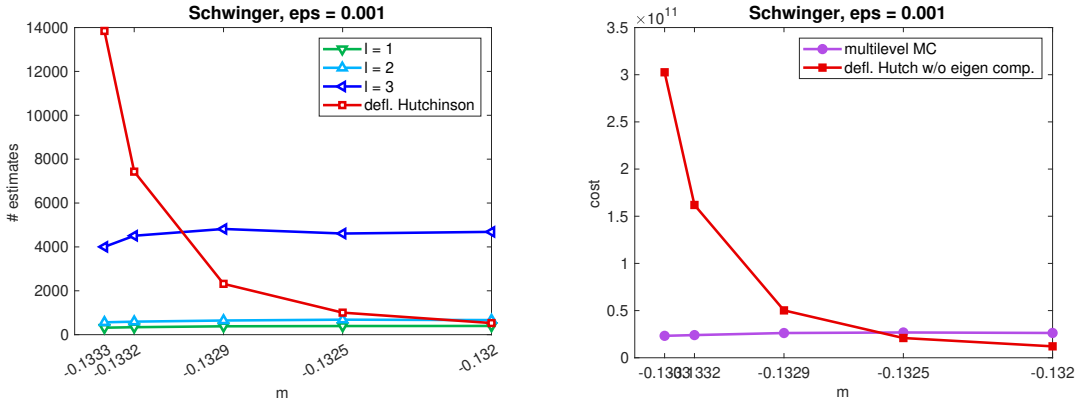

Figure 6.1: Multilevel Monte-Carlo and deflated Hutchinson for the Schwinger matrix: no of stochastic estimates on each level difference eq. 6.21 and total cost for different masses m.

We compared against deflated Hutchinson with a time-optimal number of deflated eigenpairs, and we did not count the cost for the eigenpair computation. The figure shows that multilevel Monte-Carlo becomes increasingly efficient over deflated Hutchinson as the masses become smaller, ending up in a one order of magnitude improvement in cost for the smallest. Interestingly, we also see that the number of stochastic estimates to be performed on each level in multilevel Monte-Carlo depends on the masses only very mildly, whereas the number of stochastic estimates increase rapidly in deflated Hutchinson.

Further optimizations may be achieved by skipping some of the levels[56]. While all levels are generally needed in the multigrid solver, we could, for example, skip every other level in the multilevel Monte-Carlo decomposition eq. 6.14. First experiments in this Schwinger setting and with $\epsilon = 10^{-3}$ show that this can indeed pay off: The second level matrix A_2 in the Schwinger example is such that the work for solving a system with A_2 is comparable to that for solving a system with A_1. So, if we skip level 2 or levels 2 and 3 together, although we need more stochastic estimates than what we need when we use all levels, we have less overall

[56]This idea of skipping levels was suggested by an anonymous referee of our paper [1].

work. This work is comparable in both cases and about 40% less than the total work when using all levels.

6.2.2 LQCD I: clover-improved Wilson-Dirac operator

We turn now from the relatively simple Schwinger model to lattice QCD, and test our multilevel Monte-Carlo method for computing the trace of the inverse of the 4D Dirac operator, in the particular case of the clover-improved Wilson-Dirac discretization[57].

We have performed the same numerical tests as in the Schwinger case, now with a lattice QCD gauge configuration for a lattice of size[58] 64×32^3. We used three multigrid levels with aggregation blocks of size 4^4 from the finest level to the first coarse one, and of size 2^4 from the first coarse level to the coarsest. Furthermore, for our multilevel Monte-Carlo trace computation we tuned the stopping criteria for the sample root mean square deviations to be $\rho_1 = \sqrt{0.95}\epsilon\tau$ and $\rho_2 = \sqrt{0.04}\epsilon\tau$ for the first two difference levels. Unlike in the Schwinger case, we compute the trace at the coarsest level stochastically rather than directly, and we tuned $\rho_3 = \sqrt{0.01}\epsilon\tau$ for that level. The trace at the coarsest level is computed via (non-deflated) Hutchinson, and unlike in our Schwinger numerical experiments, we compare the overall multilevel Monte-Carlo against non-deflated Hutchinson. We set $\epsilon = 10^{-4}$, and τ is obtained again by averaging over five estimates.

To implement both our multilevel Monte Carlo and the Hutchinson method, we used DD-αAMG whenever solving a linear system was needed at different levels in the multilevel hierarchy. Our code can be found at this GitHub repository, and all of our multilevel Monte Carlo computations for the Wilson operator were done on 32 processes and 1 OpenMP thread of a single Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz node. As in the Schwinger case, we have shifted the value of the mass parameter m_0 to values more negative than the one originally employed in the construction of the corresponding Markov chain, to reach more ill-conditioned situations. Due to this shift to harder matrices, we have seen the need to enable our coarsest-level improvements from chapter 4 and we found that, without enabling them, the execution times were prohibitively large when going to our smallest values of m_0 and we would not have been able to perform our multilevel Monte Carlo tests in a reasonable amount of time. Some of the parameters in the multigrid hierarchy in DD-αAMG change with respect to the default values (see tab. 4.2) when we want to have a good multigrid hierarchy

[57] This comes from collaborative work with Jose Jiménez and my involvement in the supervision of his master thesis project.

[58] This configuration was provided by the lattice QCD group at the University of Regensburg via the Collaborative Research Centre SFB-TRR55, with parameters $m_0 = -0.332159624413$ and $c_{sw} = 1.9192$ [171].

for multilevel Monte Carlo computations; in tab. 6.2 we list the changed ones. The parameters have been changed, compared to the base ones in tab. 4.2, to improve the quality of the operators at levels $\ell = 2$ and $\ell = 3$ as they will be more similar now to the finest-level one, in the sense of having a more similar near kernel (see sect. 3.3.1 for more on the concept of near kernel, and sects. 6.1.3.1 and 6.1.3.2 where we have discussed why it is important to have "similar" operators when going from one level to the next in the multigrid hierarchy employed in our multilevel Monte Carlo method).

As discussed before, we present execution times in this section rather than results in terms of a cost model. We do so mainly because a cost model for DD-αAMG would be quite involved, mostly due to the mixture of K-cycles with different types of communications (i.e. global and nearest-neighbors).

$\ell = 1$	number of test vectors	28
	boostrap setup iterations	7
$\ell = 2$	boostrap setup iterations	6

Table 6.2: Parameters in DD-αAMG for multilevel Monte Carlo.

As in fig. 6.1 for Schwinger, we present our results in the Wilson case in fig. 6.2, where we see a similar behaviour: the two difference levels are not affected by changes in the value of m_0. The coarsest level does get affected by changes in m_0, leading to values in the number of estimates as large as in Hutchinson. But due to the coarsest-level's small size, the overall execution time is then considerably smaller in the multilevel Monte Carlo case, compared to the Hutchinson case, by a factor of around 5 for the most ill-conditioned m_0.

6.2.3 LQCD II: twisted mass operator

We apply now our multigrid multilevel Monte Carlo method to twisted mass matrices. The numerical tests are the same as for Wilson, but in this case we have used a 96×48^3 lattice[59]. The coarsest-level solves are done now via MUMPS. Similar to tab. 4.6, the DD-αAMG parameters used in the run, changed with respect to tab. 4.2, can be found in tab. 6.3. At the coarsest level, we have set $\delta = 1.0$.

For our multilevel Monte-Carlo trace computation we have tuned the stopping criteria for the sample root mean square deviations to be $\rho_1 = \sqrt{0.45}\epsilon\tau$, $\rho_2 =$

[59] From the Extended Twisted Mass Collaboration, provided to us by Jacob Finkenrath, who is part of CaSToRC at the Cyprus Institute. The main parameters of this configuration are $\kappa = 0.137290$, $c_{sw} = 1.57551$ and $\mu = 0.0009$.

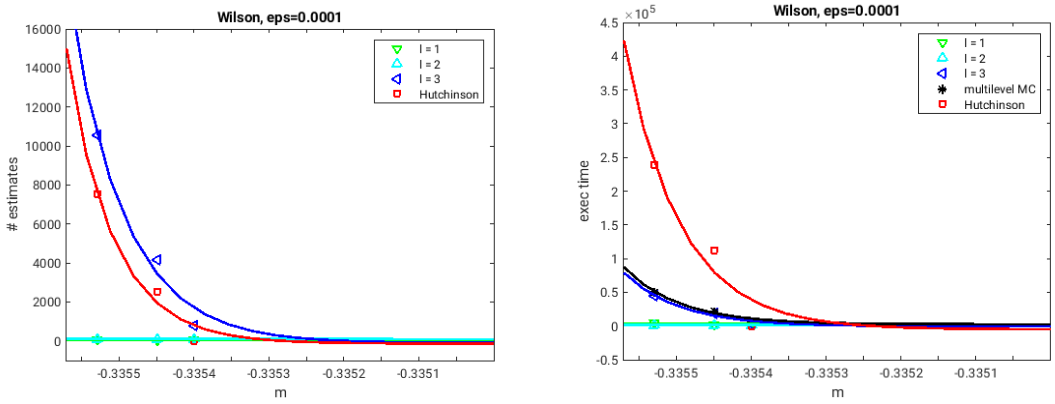

Figure 6.2: Multilevel Monte-Carlo and "plain" Hutchinson for the clover-improved Wilson-Dirac matrix: no of stochastic estimates on each level difference eq. 6.21 and total execution time for different masses m. Contrary to fig. 6.1 where we have used connecting lines in the plots, we have fitted here the data with exponentials. For $\ell = 1$, the values on the *left* plot are around 40 and on the *right* plot around 2000.0. The corresponding (approximate) values for $\ell = 2$ are 90 and 1000.0.

$\sqrt{0.45}\epsilon\tau$, $\rho_3 = \sqrt{0.09}\epsilon\tau$ and $\rho_4 = \sqrt{0.01}\epsilon\tau$. The trace at the coarsest level is computed via (non-deflated) Hutchinson, and we compare here the overall multilevel Monte-Carlo against non-deflated Hutchinson. We have set $\epsilon = 2.0 \cdot 10^{-5}$, and τ is obtained again by averaging over five estimates.

The computations were done on a single node Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz with 56 cores and 1.5 TB or RAM. We ran with 54 processes and 1 OpenMP thread per process. Tab. 6.4 shows the results of our run.

The last row in tab. 6.4 displays the total computational gain of our multilevel method over (non-deflated) Hutchinson, with a speedup of a bit over a factor of 5. But, more importantly, we see a tremendous algorithmic gain when comparing the first difference level in the multilevel Monte Carlo method versus Hutchinson: we see a reduction in the number of estimates of the former of around 18 times smaller than the latter. This algorithmic gain is spoiled mostly by the expensive execution time associated to the solves of the operator at the second level: the time for solving with the operator at the finest level is only a factor of 2.4 more expensive than solving at the second level, which is the main element bringing down the algorithmic factor of 18 down the computational factor of 5. This interference of the second-level operator motivates us again to skip levels in multilevel Monte Carlo: in the Schwinger model, the second-level operator contained, roughly, the same number of nonzero entries as the finest-level one, and skipping that level in the multilevel Monte Carlo computation represented improvements in the computational results, with relatively small increase in the number of estimates.

$\ell = 1$	number of test vectors	32
	post-smoothing steps	4
	boostrap setup iterations	8
	number of test vectors	36
	post-smoothing steps	3
$\ell = 2$	boostrap setup iterations	7
$\ell = 3$	restart length of FGMRES	5
	maximal restarts of FGMRES	2
	relative residual tolerance	10^{-1}
	number of test vectors	36
	size of lattice-blocks for aggregates	2^4
	pre-smoothing steps	0
	post-smoothing steps	3
	Minimal Residual iterations	4
	boostrap setup iterations	6
$\ell = 4$	restart length of GMRES	60
	maximal restarts of FGMRES	20
	relative residual tolerance	10^{-1}

Table 6.3: Base parameters in our DD-αAMG solves, for multilevel Monte Carlo with MUMPS.

The results in tab. 6.4 seem to indicate that skipping the second level might be a good idea again in the twisted mass case. We might see more algorithmic loss in twisted mass than with the Schwinger model, but the combination of our method with deflation might overcome this issue (this is future work, though; see sect. 6.3).

6.3 Outlook on multigrid multilevel Monte Carlo

The results in sect. 6.2.1 are remarkable, and they have motivated us to further perform the tests in sects. 6.2.2 and 6.2.3. The performance gains in the latter two sections, i.e. in the context of lattice QCD, although not compared yet to deflated Hutchinson, show the great potential of our multigrid multilevel Monte Carlo method when applied to lattice QCD matrices.

Due to each level difference (and level, the coarsest) in our multigrid multilevel Monte Carlo being computed via Hutchinson, the method is very adaptable, in the sense that we can now further enhance each level difference, via different already-existing methods typically used for the computation of tr $f(A)$ in lattice QCD. Two of these methods are deflation [40] and hierarchical probing [41]. We

	measurement	Mult. MC	Hutchinson
$\ell = 1$	# estimates	63	1164
	time (seconds)	17,010.0	221,160.0
$\ell = 2$	# estimates	172	-
	time (seconds)	15,480.0	-
$\ell = 3$	# estimates	903	-
	time (seconds)	9,030.0	-
$\ell = 4$	# estimates	7715	-
	time (seconds)	1,543.0	-
total	# estimates	-	1164
total	time (seconds)	43,063.0	221,160.0

Table 6.4: Results for the application of multigrid multilevel Monte Carlo in the twisted mass case. For multilevel Monte Carlo, $\ell = 1$, $\ell = 2$ and $\ell = 3$ represent difference levels, whereas $\ell = 4$ is the coarsest level. For the Hutchinson method there are no difference levels, hence $\ell = 1$ is the only option.

will pursue this line of research further, still in the context of both Wilson and twisted mass fermions, and we will combine our multigrid multilevel Monte Carlo method with deflation and hierarchical probing, within the PhD research work of Jose Jiménez. Deflation, in particular, is of importance in combination with our method: when taking difference levels, and due to the approximate nature of the test vectors and furthermore of P and R due to local coherence, there might be some outlying eigenvectors which might be easy to pick and deflate. Our method is not restricted, though, to be combined with deflation and probing only, hence many other (algorithmic) doors have been opened with the introduction of this new technique, and we will also explore some of them.

Another interesting feature of the method introduced here, but this time of a rather more computational nature, is the possibility to use agglomeration, as discussed in sect. 4.5, in a dynamic way at every level: we can run the first difference level on the total number of nodes, but then we can re-factor the code to run the second level difference in an agglomerated/batched way, i.e. we can run this second difference level on for example a quarter of the total number of nodes, and each subset of processes can run a certain number of estimates (i.e. a *batch*) before all the subsets synchronize to check for convergence. This process can be repeated recursively and coarser difference levels can be computed on less nodes i.e. with an even more aggressive agglomeration. We will investigate this in future work.

List of Figures

List of Tables

List of Algorithms & Scripts

Bibliography

[1] Andreas Frommer, Mostafa Nasr Khalil, and Gustavo Ramirez-Hidalgo. A multilevel approach to variance reduction in the stochastic estimation of the trace of a matrix. *arXiv preprint arXiv:2108.11281*, 2021. Accepted in the SIAM Journal on Scientific Computing.

[2] Jesus Espinoza-Valverde, Andreas Frommer, Gustavo Ramirez-Hidalgo, and Matthias Rottmann. Coarsest-level improvements in multigrid for lattice QCD on large-scale computers. *arXiv preprint arXiv:2205.09104*, 2022.

[3] John Campbell, Joey Huston, and Frank Krauss. *The Black Book of Quantum Chromodynamics: a Primer for the LHC Era*. Oxford University Press, 2018.

[4] Walter Greiner, Stefan Schramm, and Eckart Stein. *Quantum Chromodynamics*. Springer Science & Business Media, 2007.

[5] Kenneth G Wilson. Confinement of quarks. *Physical review D*, 10(8):2445, 1974.

[6] Tanja Bergrath, Maria Ramalho, Richard Kenway, et al. PRACE scientific annual report 2012. Technical report, PRACE, 2012. `http://www.prace-ri.eu/IMG/pdf/PRACE_Scientific_Annual_Report_2012.pdf`, p. 32.

[7] Martyn Guest, Giovanni Aloisio, Richard Kenway, et al. The scientific case for HPC in Europe 2012 - 2020. Technical report, PRACE, October 2012. `http://www.prace-ri.eu/PRACE-The-Scientific-Case-for-HPC`, p. 75.

[8] S. Dürr, Z. Fodor, J. Frison, C. Hoelbling, R. Hoffmann, S. D. Katz, S. Krieg, T. Kurth, L. Lellouch, T. Lippert, K. K. Szabo, and G. Vulvert. Ab initio determination of light hadron masses. *Science*, 322(5905):

1224–1227, 2008. ISSN 0036-8075. doi: 10.1126/science.1163233. URL http://science.sciencemag.org/content/322/5905/1224.

[9] Thomas A. DeGrand and Pietro Rossi. Conditioning techniques for dynamical fermions. *Comput. Phys. Commun.*, 60:211–214, 1990. doi: 10.1016/0010-4655(90)90006-M.

[10] S. Fischer, A. Frommer, U. Glassner, T. Lippert, G. Ritzenhofer, and K. Schilling. A parallel SSOR preconditioner for lattice QCD. *Comput. Phys. Commun.*, 98:20–34, 1996. doi: 10.1016/0010-4655(96)00089-6.

[11] Martin Lüscher. Local coherence and deflation of the low quark modes in lattice QCD. *JHEP*, 2007(07):081, 2007. URL http://stacks.iop.org/1126-6708/2007/i=07/a=081.

[12] Andreas Frommer, Andrea Nobile, and Paul Zingler. Deflation and flexible SAP-preconditioning of GMRES in lattice QCD simulations. 4 2012. arXiv:1204.5463.

[13] Martin Lüscher. Solution of the Dirac equation in lattice QCD using a domain decomposition method. *Comput. Phys. Commun.*, 156:209–220, 2004. doi: 10.1016/S0010-4655(03)00486-7.

[14] R Ben-Av, Achi Brandt, M Harmatz, E Katznelson, PG Lauwers, Shay Solomon, and K Wolowesky. Fermion simulations using parallel transported multigrid. *Physics Letters B*, 253(1-2):185–192, 1991.

[15] Richard C Brower, K Moriarty, E Myers, and Claudio Rebbi. The multigrid method for fermion calculations in quantum chromodynamics. *Multigrid Methods: Theory, Applications, and Supercomputing, SF McCormick, ed*, 110:85–100, 1987.

[16] Thomas Kalkreuter. Multigrid methods for propagators in lattice gauge theories. *Journal of computational and applied mathematics*, 63(1-3):57–68, 1995.

[17] Jeroen C Vink. Multigrid inversion of staggered and Wilson fermion operators with SU(2) gauge fields in two dimensions. *Physics Letters,(Section) B;(Netherlands)*, 272(1/2), 1991.

[18] J. C. Osborn, R. Babich, J. Brannick, R. C. Brower, M. A. Clark, S. D. Cohen, and C. Rebbi. Multigrid solver for clover fermions. *PoS*, LATTICE2010:037, 2010. doi: 10.22323/1.105.0037.

[19] Richard C Brower, MA Clark, Alexei Strelchenko, and Evan Weinberg. Multigrid for staggered lattice fermions. *arXiv preprint arXiv:1801.07823*, 2018.

[20] Saul D Cohen, RC Brower, MA Clark, and JC Osborn. Multigrid algorithms for domain-wall fermions. *arXiv preprint arXiv:1205.2933*, 2012.

[21] J.C. Osborn. QOPQDP software. `https://github.com/usqcd-software/qopqdp`.

[22] OpenQCD. `https://luscher.web.cern.ch/luscher/openQCD/`. 2012.

[23] QUDA: A library for QCD on GPUs. `http://lattice.github.io/quda/`. Accessed: 2022-04-11.

[24] Andreas Frommer, Karsten Kahl, Stefan Krieg, Björn Leder, and Matthias Rottmann. Adaptive aggregation-based domain decomposition multigrid for the lattice Wilson-Dirac operator. *SIAM journal on scientific computing*, 36(4):A1581–A1608, 2014.

[25] Andreas Frommer, K Kahl, S Krieg, B Leder, and M Rottmann. An adaptive aggregation based domain decomposition multilevel method for the lattice Wilson-Dirac operator: Multilevel results. *arXiv preprint arXiv:1307.6101*, 2013.

[26] M. Rottmann, A Strebel, S. Heybrock, S. Bacchio, B. Leder, and I Kanamori. DD-αAMG software, Wilson. `https://github.com/DDalphaAMG/DDalphaAMG`, .

[27] Constantia Alexandrou, Simone Bacchio, Jacob Finkenrath, Andreas Frommer, Karsten Kahl, and Matthias Rottmann. Adaptive aggregation-based domain decomposition multigrid for twisted mass fermions. *Physical Review D*, 94(11):114509, 2016.

[28] Simone G Bacchio. *Simulating Maximally Twisted Fermions at the Physical Point with Multigrid Methods*. PhD thesis, 2019.

[29] M. Rottmann, A Strebel, S. Heybrock, S. Bacchio, B. Leder, and I Kanamori. DD-αAMG software, twisted mass. `https://github.com/sbacchio/DDalphaAMG`, .

[30] M. Parks, E. De Sturler, G. Mackey, D. Johnson, and S. Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM J. Sci. Comput.*, 28(5):1651–1674, 2006.

[31] M. Embree, J. Loe, and R. Morgan. Polynomial preconditioned Arnoldi. *arXiv preprint arXiv:1806.08020*, 2018.

[32] J. Loe and R. Morgan. New polynomial preconditioned GMRES. *arXiv preprint arXiv:1911.07065*, 2019.

[33] N. Nachtigal, L. Reichel, and L. Trefethen. A hybrid GMRES algorithm for nonsymmetric linear systems. *SIAM J. Matrix Anal. Appl.*, 13(3):796–825, 1992.

[34] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[35] P. Ghysels, T. Ashby, K. Meerbergen, and W. Vanroose. Hiding global communication latency in the GMRES algorithm on massively parallel machines. *SIAM J. Sci. Comput.*, 35(1):C48–C71, 2013.

[36] Patrick R Amestoy, Alfredo Buttari, Jean-Yves L'excellent, and Theo Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Transactions on Mathematical Software (TOMS)*, 45(1):1–26, 2019.

[37] Patrick R Amestoy, Iain S Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[38] J. Cheng, M. Grossman, and Ty. McKercher. *Professional CUDA C Programming*. John Wiley & Sons, Inc., 2014.

[39] Michael B Giles. Multilevel Monte Carlo methods. *Acta Numer.*, 24:259–328, 2015. doi: 10.1017/S096249291500001X.

[40] Arjun Singh Gambhir, Andreas Stathopoulos, and Kostas Orginos. Deflation as a method of variance reduction for estimating the trace of a matrix inverse. *SIAM J. on Sci. Comput.*, 39(2):A532–A558, 2017. doi: 10.1137/16M1066361.

[41] Andreas Stathopoulos, Jesse Laeuchli, and Kostas Orginos. Hierarchical probing for estimating the trace of the matrix inverse on toroidal lattices. *SIAM J. Sci. Comput.*, 35(5):299–322, 2013.

[42] Mark Thomson. *Modern Particle Physics*. Cambridge University Press, 2013.

[43] Christof Gattringer and Christian Lang. *Quantum Chromodynamics on the Lattice: an Introductory Presentation*, volume 788. Springer Science & Business Media, 2009.

[44] Arjun Singh Gambhir. *Disconnected Diagrams in Lattice QCD*. PhD thesis, College of William and Mary, 2017.

[45] Matthias Rottmann. *Adaptive Domain Decomposition Multigrid for Lattice QCD*. PhD thesis, Wuppertal U., 2016.

[46] Karsten Kahl. *Adaptive Algebraic Multigrid for Lattice QCD computations*. PhD thesis, Universität Wuppertal, Fakultät für Mathematik und Naturwissenschaften, 2018.

[47] Aron Beekman, Louk Rademaker, and Jasper van Wezel. An introduction to spontaneous symmetry breaking. *SciPost Physics Lecture Notes*, page 011, 2019.

[48] Giovanni Costa and Gianluigi Fogli. *Symmetries and Group Theory in Particle Physics: An Introduction to Space-time and Internal Symmetries*, volume 823. Springer Science & Business Media, 2012.

[49] H Fritzsch. The history of quantum chromodynamics. *International Journal of Modern Physics A*, 34(01):1930001, 2019.

[50] Leslie E Ballentine. *Quantum Mechanics: a Modern Development*. World Scientific Publishing Company, 2014.

[51] Jun John Sakurai and Jim Napolitano. *Modern Quantum Mechanics; 2nd ed.* Addison-Wesley, San Francisco, CA, 2011. URL `https://cds.cern.ch/record/1341875`.

[52] Leonard Susskind and Art Friedman. *Special Relativity and Classical Field Theory*. Penguin UK, 2017.

[53] Michael Peskin. *An Introduction to Quantum Field Theory*. CRC press, 2018.

[54] Steven Weinberg. *The Quantum Theory of Fields*, volume 2. Cambridge university press, 1995.

[55] Charles Kittel and Herbert Kroemer. *Thermal Physics*, volume 9690. Wiley New York, 1970.

[56] Richard Phillips Feynman. Space-time approach to non-relativistic quantum mechanics. *Feynman's Thesis—A New Approach To Quantum Theory*, pages 71–109, 2005.

[57] Richard P Feynman, Albert R Hibbs, and Daniel F Styer. *Quantum Mechanics and Path Integrals*. Courier Corporation, 2010.

[58] Philip J Davis and Philip Rabinowitz. *Methods of Numerical Integration*. Courier Corporation, 2007.

[59] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987.

[60] Herbert Goldstein. *Classical Mechanics*. Addison-Wesley, 1980.

[61] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

[62] Michael Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

[63] Taylor Ryan Haar. *Optimisations to Hybrid Monte Carlo for Lattice QCD*. PhD thesis, 2019.

[64] W.K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970. doi: 10.1093/biomet/57.1.97.

[65] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953. doi: 10.1063/1.1699114.

[66] Paul Adrien Maurice Dirac. The quantum theory of the electron. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 117(778):610–624, 1928.

[67] H Blaine Lawson and Marie-Louise Michelsohn. *Spin Geometry (PMS-38), Volume 38*. Princeton university press, 2016.

[68] Roberto Frezzotti, Pietro Antonio Grassi, Stefan Sint, and Peter Weisz. A local formulation of lattice QCD without unphysical fermion zero modes. *Nuclear Physics B-Proceedings Supplements*, 83:941–946, 2000.

[69] John Kogut and Leonard Susskind. Hamiltonian formulation of Wilson's lattice gauge theories. *Physical Review D*, 11(2):395, 1975.

[70] Gordon D Smith, Gordon D Smith, and Gordon Dennis Smith Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford university press, 1985.

[71] Leonard Susskind. Lattice fermions. *Physical Review D*, 16(10):3031, 1977.

[72] Kenneth G. Wilson. *Quarks and Strings on a Lattice*. Springer US, 1977.

[73] István Montvay and Gernot Münster. *Quantum Fields on a Lattice*. Cambridge University Press, 1997.

[74] Bijan Sheikholeslami and Ralf Wohlert. Improved continuum limit lattice action for QCD with Wilson fermions. *Nuclear Physics B*, 259(4):572–596, 1985.

[75] Roberto Frezzotti, Pietro Antonio Grassi, Stefan Sint, Peter Weisz, Alpha Collaboration, et al. Lattice QCD with a chirally twisted mass term. *Journal of High Energy Physics*, 2001(08):058, 2001.

[76] Roberto Frezzotti, Stefan Sint, and Peter Weisz. O(a) improved twisted mass lattice QCD. *Journal of High Energy Physics*, 2001(07):048, 2001.

[77] Julian Schwinger. Gauge invariance and mass. II. *Physical Review*, 128(5): 2425, 1962.

[78] Robert Link. *The Schwinger Model*. PhD thesis, University of British Columbia, 1986.

[79] Lloyd N Trefethen and David Bau III. *Numerical Linear Algebra*, volume 50. Siam, 1997.

[80] William L Briggs, Van Emden Henson, and Steve F McCormick. *A Multigrid Tutorial*. SIAM, 2000.

[81] Artur Strebel. *Advanced Applications for Algebraic Multigrid Methods in Lattice QCD*. PhD thesis, Universität Wuppertal, Fakultät für Mathematik und Naturwissenschaften, 2020.

[82] James W Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[83] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[84] William Kahan. Lecture notes on the status of IEEE standard 754 for binary floating-point arithmetic. *http://http. cs. berkeley. edu/˜ wkahan/ieee754status/ieee. ps*, 1996.

[85] James R Bunch and Beresford N Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8(4):639–655, 1971.

[86] Andrew Tanenbaum. *Modern Operating Systems*. Pearson Education, Inc.,, 2009.

[87] Kyle Gallivan, William Jalby, and Ulrike Meier. The use of BLAS3 in linear algebra on a parallel processor with a hierarchical memory. *SIAM Journal on Scientific and Statistical Computing*, 8(6):1079–1084, 1987.

[88] H. Schwarz. Gesammelte mathematische Abhandlungen. *Vierteljahrschrift Naturforsch. Ges. Zürich*, pages 272–286, 1870.

[89] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, 1996.

[90] Yousef Saad and Martin H Schultz. GMRES: A gneralized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[91] Mongi Benhamadou et al. On the FOM algorithm for the resolution of the linear systems Ax= b. *Advances in Linear Algebra & Matrix Theory*, 4(03): 156, 2014.

[92] Walter Edwin Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics*, 9(1): 17–29, 1951.

[93] Wayne Joubert. On the convergence behavior of the restarted GMRES algorithm for solving nonsymmetric linear systems. *Numerical linear algebra with applications*, 1(5):427–447, 1994.

[94] Ilya Zavorin, DianneP O'Leary, and Howard Elman. Complete stagnation of GMRES. *Linear Algebra and its Applications*, 367:165–183, 2003.

[95] Mark Embree. The tortoise and the hare restart GMRES. *SIAM review*, 45 (2):259–266, 2003.

[96] Guido Cossu, Peter Boyle, Norman Christ, Chulwoo Jung, Andreas Jüttner, and Francesco Sanfilippo. Testing algorithms for critical slowing down. In *EPJ Web of Conferences*, volume 175, page 02008. EDP Sciences, 2018.

[97] Wolfgang Hackbusch. *Multi-grid Methods and Applications*, volume 4. Springer Science & Business Media, 2013.

[98] John W Ruge and Klaus Stüben. Algebraic multigrid. In *Multigrid methods*, pages 73–130. SIAM, 1987.

[99] John David Jackson. *Classical electrodynamics*, volume 31999. Wiley New York, 1977.

[100] Eric Poisson and Clifford M Will. *Gravity: Newtonian, post-Newtonian, Relativistic*. Cambridge University Press, 2014.

[101] Pieter Wesseling and Cornelis W Oosterlee. Geometric multigrid with applications to computational fluid dynamics. *Journal of computational and applied mathematics*, 128(1-2):311–334, 2001.

[102] Suha Kayum, Michel Cancelliere, Marcin Rogowski, and Ahmed Al-Zawawi. Application of algebraic multigrid in fully implicit massive reservoir simulations. In *SPE Europec featured at 81st EAGE Conference and Exhibition*. OnePetro, 2019.

[103] Mark F Adams. Algebraic multigrid methods for constrained linear systems with applications to contact problems in solid mechanics. *Numerical linear algebra with applications*, 11(2-3):141–153, 2004.

[104] Ronald Babich, James Brannick, Richard C Brower, MA Clark, Thomas A Manteuffel, SF McCormick, JC Osborn, and C Rebbi. Adaptive multigrid algorithm for the lattice Wilson-Dirac operator. *Physical review letters*, 105 (20):201602, 2010.

[105] James Brannick, Richard C Brower, MA Clark, James C Osborn, and Claudio Rebbi. Adaptive multigrid algorithm for lattice QCD. *Physical review letters*, 100(4):041601, 2008.

[106] A. Frommer, K. Kahl, S. Krieg, B. Leder, and M. Rottmann. Aggregation-based multilevel methods for lattice QCD. *Proceedings of Science*, LATTICE2011:046, 2011. `http://pos.sissa.it`.

[107] Dietrich Braess. Towards algebraic multigrid for elliptic problems of second order. *Computing*, 55(4):379–393, 1995.

[108] Marian Brezina, R Falgout, Scott MacLachlan, T Manteuffel, S McCormick, and J Ruge. Adaptive smoothed aggregation (αSA) multigrid. *SIAM review*, 47(2):317–346, 2005.

[109] Yvan Notay and Panayot S Vassilevski. Recursive Krylov-based multigrid cycles. *Numerical Linear Algebra with Applications*, 15(5):473–487, 2008.

[110] Achi Brandt, James Brannick, Karsten Kahl, and Irene Livshits. Bootstrap amg. *SIAM Journal on Scientific Computing*, 33(2):612–632, 2011.

[111] M. Clark, B. Joó, A. Strelchenko, M. Cheng, A. Gambhir, and R. Brower. Accelerating lattice QCD multigrid on GPUs using fine-grained parallelization. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 795–806. IEEE, 2016.

[112] Henning Leemhuis. Approximate direct solves on the coarsest level of multigrid. Master's thesis, Bergische Universität Wuppertal, Germany, 2022.

[113] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Statist. Comput.*, 6(4):865–881, 1985.

[114] R. Morgan. Computing interior eigenvalues of large matrices. *Linear Algebra Appl.*, 154:289–309, 1991.

[115] R. Morgan and M. Zeng. A harmonic restarted Arnoldi algorithm for calculating eigenvalues and determining multiplicity. *Linear Algebra Appl.*, 415 (1):96–113, 2006.

[116] D. Calvetti and L. Reichel. On the evaluation of polynomial coefficients. *Numerical Algorithms*, 33(1-4):153–161, 2003.

[117] O. Coulaud, L. Giraud, P. Ramet, and X. Vasseur. Deflation and augmentation techniques in Krylov subspace methods for the solution of linear systems. *arXiv preprint arXiv:1303.5692*, 2013.

[118] K. Soodhalter, E. de Sturler, and M. Kilmer. A survey of subspace recycling iterative methods. *GAMM-Mitteilungen*, 43(4):e202000016, 2020.

[119] A. Stathopoulos and K. Orginos. Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics. *SIAM J. Sci. Comput.*, 32(1):439–462, 2010.

[120] R. Morgan. GMRES with deflated restarting. *SIAM J. Sci. Comput.*, 24 (1):20–37, 2002.

[121] Eric de Sturler. Nested Krylov methods based on GCR. *Journal of Computational and Applied Mathematics*, 67(1):15–41, 1996.

[122] Daniel Mohler, Stefan Schaefer, and Jakob Simeth. CLS 2+1 flavor simulations at physical light- and strange-quark masses. In *EPJ Web of Conferences*, volume 175, page 02010. EDP Sciences, 2018.

[123] Constantia Alexandrou, Simone Bacchio, Panagiotis Charalambous, Petros Dimopoulos, Jacob Finkenrath, Roberto Frezzotti, Kyriakos Hadjiyiannakou, Karl Jansen, Giannis Koutsou, Bartosz Kostrzewa, et al. Simulating twisted mass fermions at physical light, strange, and charm quark masses. *Physical Review D*, 98(5):054518, 2018.

[124] Stefan Krieg and Thomas Lippert. Tuning lattice QCD to petascale on Blue Gene. In *P, NIC Symposium*, volume 2010, pages 155–164, 2010.

[125] Mark Hoemmen. *Communication-Avoiding Krylov Subspace Methods*. PhD thesis, University of California, Berkeley, 2010.

[126] Philippe Leleux. *Hybrid Direct and Interactive Solvers for Sparse Indefinite and Overdetermined Systems on Future Exascale Architectures*. PhD thesis, 2021.

[127] D. Steinkraus, I. Buck, and P. Y. Simard. Using GPUs for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 1115–1120 Vol. 2, 2005.

[128] C. Yang, Q. Wu, J. Chen, and Z. Ge. GPU acceleration of high-speed collision molecular dynamics simulation. In *2009 Ninth IEEE International Conference on Computer and Information Technology*, volume 2, pages 254–259, 2009.

[129] M.A. Clark, R. Babich, K. Barros, R.C. Brower, and C. Rebbi. Solving lattice QCD systems of equations using mixed precision solvers on GPUs. *Computer Physics Communications*, 181(9):1517–1528, Sep 2010. ISSN 0010-4655. doi: 10.1016/j.cpc.2010.05.002. URL `http://dx.doi.org/10.1016/j.cpc.2010.05.002`.

[130] Y. Nakamura, K. Ishikawa, Y. Kuramashi, T. Sakurai, and H. Tadano. Modified block BiCGStab for lattice QCD. *Comput. Phys. Commun.*, 183 (1):34–37, 2012.

[131] M. Lüscher. Lattice QCD and the Schwarz alternating procedure. *JHEP*, 183:p. 052, 2003.

[132] J. Cardoso, J. Coutinho, and P. Diniz. *Embedded Computing for High Performance Computing*. Elsevier Inc., 2017. ISBN 9781417642595. URL `http://books.google.com/books?id=W-xMPgAACAAJ`.

[133] Jose Jiménez. A Block Trace estimator and its Application to Lattice QCD. Master's thesis, Bergische Universität Wuppertal, Germany, 2022.

[134] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008. ISBN 0898716462, 9780898716467.

[135] B. Sapoval, Th. Gobron, and A. Margolina. Vibrations of fractal drums. *Phys. Rev. Lett.*, 67:2974–2977, Nov 1991.

[136] Gene H. Golub and Urs von Matt. Generalized cross-validation for large scale problems. *J. Comput. Graph. Statist.*, 6:1–34, 1995.

[137] Gene H Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

[138] Ernesto Estrada and Desmond J Higham. Network properties revealed through matrix functions. *SIAM Rev.*, 52(4):696–714, 2010.

[139] Yuval Ginosar, Ivan Gutman, Toufik Mansour, and Matthias Schork. Estrada index and Chebyshev polynomials. *Chem. Phys. Lett.*, 454:145–147, 2008.

[140] Ernesto Estrada. *The Structure of Complex Networks: Theory and Applications*. Oxford University Press, Inc., New York, 2011.

[141] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[142] Havard Rue and Leonhard Held. *Gaussian Markov Random Fields: Theory And Applications*. CRC press, 2005.

[143] Raphael A Meyer, Cameron Musco, Christopher Musco, and David P Woodruff. Hutch++: Optimal stochastic trace estimation. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 142–155. SIAM, 2021.

[144] Shashanka Ubaru and Yousef Saad. Applications of trace estimation techniques. In *International Conference on High Performance Computing in Science and Engineering*, pages 19–33. Springer, 2017.

[145] Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of $\mathrm{tr}(f(A))$ via stochastic Lanczos quadrature. *SIAM J. Matrix Anal. Appl.*, 38(4): 1075–1099, 2017.

[146] J. Sexton and D. Weingarten. Systematic expansion for full QCD based on the valence approximation, 1994.

[147] Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad. An estimator for the diagonal of a matrix. *Applied Numerical Mathematics*, 57:1214–1229, 11 2007.

[148] Eric Endress, Carlos Pena, and Karthee Sivalingam. Variance Reduction with Practical All-to-all Lattice Propagators. *Comput. Phys. Commun.*, 195:35–48, 2015. doi: 10.1016/j.cpc.2015.04.017.

[149] Jok M. Tang and Yousef Saad. A probing method for computing the diagonal of a matrix inverse. *Numer. Linear Algebra Appl.*, 19(3):485–501, 2012.

[150] Jesse Laeuchli and Andreas Stathopoulos. Extending hierarchical probing for computing the trace of matrix inverses. *SIAM J. Sci. Comput.*, 42(3): A1459–A1485, 2020.

[151] Andreas Frommer, Claudia Schimmel, and Marcel Schweitzer. Analysis of probing techniques for sparse approximation and trace estimation of decaying matrix functions. *SIAM J. Matrix Anal. Appl.*, 42:1290–1318, 2021. doi: https://doi.org/10.1137/20M1364461.

[152] A. H. Bentbib, M. El Ghomari, K. Jbilou, and L. Reichel. Shifted extended global Lanczos processes for trace estimation with application to network analysis. *Calcolo*, 58(1):Paper No. 4, 35, 2021. ISSN 0008-0624. doi: 10.1007/s10092-020-00395-1.

[153] Jie Chen and Yousef Saad. A posteriori error estimate for computing $\mathrm{tr}(f(A))$ by using the Lanczos method. *Numer. Linear Algebra Appl.*, 25 (5):e2170, 20, 2018. ISSN 1070-5325. doi: 10.1002/nla.2170.

[154] Lin Lin, Yousef Saad, and Chao Yang. Approximating spectral densities of large matrices. *SIAM Rev.*, 58(1):34–65, 2016. ISSN 0036-1445. doi: 10.1137/130934283.

[155] M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Comm. Statist. Simulation Comput.*, 19 (2):433–450, 1990. ISSN 0361-0918. doi: 10.1080/03610919008812864.

[156] S.J. Dong and K.F. Liu. Stochastic estimation with Z_2 noise. *Phys. Lett. B*, 328:130–136, 1994.

[157] Walter Wilcox. Noise methods for flavor singlet quantities. 1999.

[158] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM*, 58 (2), 2011.

[159] Alice Cortinovis and Daniel Kressner. On randomized trace estimates for indefinite matrices with an application to determinants, 2020.

[160] Farbod Roosta Khorasani and Uri Ascher. Improved bounds on sample size for implicit matrix trace estimators. *Foundations of Computational Mathematics*, 15(5):1187–1212, 2015.

[161] Dietrich Braess. Towards algebraic multigrid for elliptic problems of second order. *Computing*, 55(4):379–393, 1995. doi: {10.1007/BF02238488}.

[162] Thomas A. DeGrand and Stefan Schaefer. Improving Meson Two Point Functions in Lattice QCD. *Comput. Phys. Commun.*, 159:185–191, 2004. doi: 10.1016/j.cpc.2004.02.006.

[163] Leonardo Giusti, P. Hernandez, M. Laine, P. Weisz, and H. Wittig. Low-energy Couplings of QCD from Current Correlators near the Chiral Limit. *JHEP*, 04:013, 2004. doi: 10.1088/1126-6708/2004/04/013.

[164] Gunnar Bali, Sara Collins, Andreas Frommer, Karsten Kahl, Issaku Kanamori, Benjamin Müller, Matthias Rottmann, and Jakob Simeth. (Approximate) low-mode averaging with a new multigrid eigensolver, 2015.

[165] Eloy Romero, Andreas Stathopoulos, and Kostas Orginos. Multigrid deflation for lattice QCD. *J. Comput. Phys.*, 409:109356, May 2020. ISSN 0021-9991. doi: 10.1016/j.jcp.2020.109356.

[166] Suman Baral, Travis Whyte, Walter Wilcox, and Ronald B. Morgan. Disconnected loop subtraction methods in lattice QCD. *Comput. Phys. Commun.*, 241:64–79, 2019. ISSN 0010-4655.

[167] Quan Liu, Walter Wilcox, and Ron Morgan. Polynomial subtraction method for disconnected quark loops, 2014.

[168] C. Alexandrou, M. Constantinou, V. Drach, K. Hadjiyiannakou, K. Jansen, G. Koutsou, A. Strelchenko, and A. Vaquero. Evaluation of disconnected quark loops for hadron structure using GPUs. *Comput. Phys. Commun.*, 185 (5):1370–1382, May 2014. ISSN 0010-4655. doi: 10.1016/j.cpc.2014.01.009.

[169] Insu Han, Dmitry Malioutov, Haim Avron, and Jinwoo Shin. Approximating spectral sums of large-scale matrices using stochastic Chebyshev approximations. *SIAM J. Sci. Comput.*, 39(4):A1558–A1585, 2017. ISSN 1064-8275. doi: 10.1137/16M1078148.

[170] Insu Han, Dmitry Malioutov, and Jinwoo Shin. Large-scale log-determinant computation through stochastic Chebyshev expansions. In *International Conference on Machine Learning*, pages 908–917, 2015.

[171] Gunnar S. Bali, Sara Collins, Benjamin Gläßle, Meinulf Göckeler, Johannes Najjar, Rudolf H. Rödl, Andreas Schäfer, Rainer W. Schiel, André Sternbeck, and Wolfgang Söldner. The moment $\langle x \rangle_{u-d}$ of the nucleon from $n_f = 2$ lattice QCD down to nearly physical quark masses. *Phys. Rev.*, D90(7): 074510, 2014. doi: 10.1103/PhysRevD.90.074510.