# Computational Methods for Quantum Chromodynamics


Dissertation

zur Erlangung des Doktorgrades der Naturwissenschaften (Dr. Rer. Nat.) der Fakultät für Physik der Universität Regensburg

Simon Heybrock

Regensburg, Januar 2013

Die Promotionsarbeit wurde angeleitet von Prof. Dr. Tilo Wettig.
Das Promotionsgesuch wurde am 18.10.2012 eingereicht.
Die Promotionskommission tagte am 28.11.2012.
Das Promotionskolloquium fand am 19.04.2013 statt.

**Prüfungsausschuss:**

| | |
|---|---|
| Vorsitzender | Prof. Dr. Rupert Huber |
| 1. Gutachter | Prof. Dr. Tilo Wettig |
| 2. Gutachter | Prof. Dr. Gunnar Bali |
| weiterer Prüfer | Prof. Dr. Thomas Niehaus |

# Contents

# Introduction

The experimental discovery of a multitude of hadrons and the attempt to understand their properties led particle physicists in the 1960s to propose the existence of fermions called quarks as the building blocks of hadrons. Further evidence required to postulate an additional three-valued quantum number, now known as *color charge*, to make the quark picture compliant with the Pauli exclusion principle. Ultimately, this led to the theory of the *strong interaction*, Quantum Chromodynamics (QCD), which describes the interaction between quarks and gluons.

QCD is a non-Abelian quantum field theory based on the gauge group $SU(3)$. Local $SU(3)$ symmetry (gauge invariance) leads to interactions between quarks and gluons and to gluon self-interactions, described by the Lagrangian

$$\mathcal{L}_{\mathrm{QCD}} = \bar{\psi}_q(i\gamma_\mu D^\mu - m_q)\psi_q - \frac{1}{4g^2}F^a_{\mu\nu}F^{\mu\nu}_a,$$

where a sum over repeated indices is implied. Here $g$ is the (bare) coupling constant, $m_q$ is the (bare) mass of quark flavor $q$, $\gamma_\mu$ are the gamma matrices, the field strength tensor $F^a_{\mu\nu}$ describes the gluons, and the $\psi_q$ and $\bar{\psi}_q$ are fermion fields. The factor $D^\mu$ in the first term is the *Dirac operator* which contains gluon fields $A^a_\mu$ and thus describes their interaction with the fermion fields. Since $F^a_{\mu\nu}$ contains terms up to second order in $A^a_\mu$, the second term does not only describe a kinetic gluon term, but also gluon self-interaction. For a more detailed definition we refer to Ref. [61]. The strength of the interaction depends on the energy scale. This is known as *running coupling*. Correlation functions are computed as a *path integral* (similar to partition functions) over all possible field configurations, weighted with a factor $\exp(-iS)$, where the action is

$$S = \int d^4x\, \mathcal{L}_{\mathrm{QCD}}.$$

The theory is formulated in Minkowski space and the integral is over the time component ($x^0$) and three space components ($x^i$, $i = 1, 2, 3$). Two fundamental properties of QCD were found: *asymptotic freedom* at high energies and *confinement* at low energies. In the former limit the strength of the coupling goes to zero. As a consequence, many quantities can be computed analytically in *pertubation theory*. However, in the low-energy limit the coupling constant becomes large, making a perturbative treatment impossible. Since many observations in nature correspond to the low-energy regime — e.g., the masses of all hadrons (foremost the neutron, proton, and pion) and the formation of atomic nuclei — this is a key obstacle for testing whether QCD is the correct theory of the strong interaction.

A non-perturbative definition of QCD is given by *lattice QCD*, introduced in 1974 by Kenneth Wilson [79], which puts QCD on a discrete space-time lattice. To preserve the essential $SU(3)$ gauge symmetry the algebra-valued gauge fields are replaced by parallel transporters, called link variables $U_{x,\mu}$, which lie in the group.

The Dirac operator needs to be discretized in a way that preserves as many symmetries of the continuum as possible. In the discretization found by Wilson, the *Wilson Dirac operator*, a stabilization term was introduced to remove unphysical doublers of the quark fields that are an artifact of the discretization. This term explicitly breaks the chiral symmetry. As a consequence, fine-tuning is necessary to obtain the correct results in the continuum limit, lattice spacing $a \to 0$. This problem was solved with the introduction of the *overlap Dirac operator*, which, however, is extremely expensive to compute numerically. Since by now the fine-tuning is under control, many groups use the Wilson operator (or the comparably cheap staggered operator, which has slightly better chiral symmetry properties at the cost of other disadvantages).

The lattice serves as a non-perturbative regularization scheme and allows a computation of correlation functions (path integrals). Since the weight factor $\exp(-iS)$ oscillates violently, it is not suitable for a numerical computation. Therefore, in contrast to the Minkowski metric typically used for continuum QCD, lattice QCD is formulated with an Euclidean metric, which yields the Boltzmann weight factor $\exp(-S)$. As a consequence, only Euclidean correlation functions can be computed. Thus, lattice QCD can be used to obtain the spectrum of the theory and (weak) matrix elements, whereas dynamical processes are not accessible.

Lattice QCD can be simulated on a computer with Monte Carlo methods, analogously to those used in statistical mechanics. For further details we refer to the books Refs. [13], [52], and [63]. The implementation on computers has fostered plenty of research and progress, combining QCD, numerical analysis, and computer science. At the beginning of the new millennium computers were strong enough and algorithms were good enough to give precision results for many quantities. And yet, when attending the Les Houches Summer School at the beginning of my PhD program three and a half years ago, Martin Lüscher, who is one of the key figures in lattice QCD, started his lecture with something along the lines of:[1]

> We are not there yet. Lattice QCD is not yet at a point where we can simply push a button to start a simulation, sit back, and wait for the result. Some fundamental algorithmic developments are necessary.

Progress has been made since then, but basically the above is still true. To comprehend the problem we consider how observables are computed. In general, the expectation value of an observable is given by an integral over all possible field configurations — the path integral. In lattice QCD this integral is approximated by a sum over a finite set of randomly chosen configurations. The error on the expectation value decreases with the square root of the set size, which in turn is proportional to the computing work. To make this approximation feasible *importance sampling* is used: the weight

[1] quoting from memory

$\exp(-S)$ of a configuration is absorbed into the probability distribution from which configurations are drawn. In this way mainly the configurations with a high weight are sampled, and thus a smaller set is sufficient for a good approximation of the integral.

Since dealing with fermions on a computer is not practical, the integration over the fermion fields is done analytically. The resulting determinant of the Dirac operator (the fermion determinant) is then rewritten as a Gaussian integral over pseudo-fermions, with the Dirac operator as covariance matrix. This Gaussian is an action term involving the inverse of the Dirac operator (multiplied by vectors from the left and from the right) and is part of the probability distribution for field configurations.

To generate configurations according to this distribution — given by the gauge action and the pseudo-fermion action term — typically a Monte Carlo method is used. Here, we discuss only the commonly used Hybrid Monte Carlo [19], which numerically integrates auxiliary Hamilton equations to evolve the configuration in phase space and thus obtains a new field configuration based on the previous one. The Hamiltonian forces are derivatives of the action with respect to the fields and thus also contain a term proportional to the inverse of the Dirac operator applied to a vector. This term is typically the most costly part of a lattice QCD computation.

With this in mind, the leading contribution to the cost $C$ of generating configurations can be roughly written as

$$C \propto N_{\mathrm{HMC}} \cdot N_{\mathrm{inversion}} \cdot C_{\mathrm{Dirac}}.$$

Here, $N_{\mathrm{HMC}}$ is the number of Hybrid Monte Carlo (HMC) steps needed to generate a stochastically independent configuration, $N_{\mathrm{inversion}}$ is the number of times we have to apply the Dirac operator for its iterative inversion, and $C_{\mathrm{Dirac}}$ is the cost of the Dirac operator. Each of these three major factors motivates a part of this thesis:

**Inverters and preconditioners** Inverters and preconditioners influence the factor $N_{\mathrm{inversion}}$. The inverse of the Dirac operator is typically computed by an iterative Krylov subspace inverter with an optional preconditioner. For a general review of these methods see Ref. [70]. Here we focus on two aspects which are important for QCD.

First, due to the large number of degrees of freedom, lattice QCD computations are done on parallel computing architectures. The computation power of a single node in a parallel architecture tends to be large compared to constraints by the bandwidth and latency of network connections between different (neighboring) nodes. Additionally, the trend goes towards computers with ever more nodes, making global operations (dot products) expensive. Therefore, to use the available computing resources efficiently, it is crucial to tailor algorithms to these limitations. This naturally leads to algorithms which require a lot of computation on the local subset of variables with little data exchange with neighboring nodes or global operations. In recent years the lattice QCD community started using such methods based on the Schwarz procedure, which involves iterative inversions on the local subset of variables, where no data exchange (neither global nor with neighbors) is necessary during such an inversion [43, 44].

Second, simulating at the physical point, i.e., with physical quark masses (pion mass), leads to an ill-conditioned Dirac operator. There are two aspects of the problem: for smaller mass (i.e., closer to the physical one) the eigenvalues are smaller, and additionally a smaller pion mass requires simulating in a larger volume, which in turn leads to more low modes [48]. This is one of the reasons why many current simulations are done with masses larger than the physical ones, which then must be carefully extrapolated to the physical point. This is an additional source of systematic uncertainty, so one would like to avoid it. The problems arising from the ill-conditioning are twofold: the computational cost will be large due to the slow convergence of the iterative inverters, and some of the commonly used inverters tend to fail completely in some cases (breakdowns occur or the convergence stagnates before the requested accuracy is reached). Algorithms based on the Schwarz procedure try to tackle these problems by introducing a coarse grid correction, for example the inexact deflation promoted in Ref. [47]. Other approaches are multigrid methods [1, 60, 24], which involve solves on (multiple levels of) coarser grids to improve convergence on the fine grid. Both methods are promising in the case of data analysis where large speedups can be achieved. However, due to the high setup cost, application for HMC trajectories is often not beneficial.

In Part I we study three inversion algorithms, or rather preconditioners for Krylov subspace inverters. As an extension of the commonly used Schwarz method with minimal overlap introduced in Refs. [43] and [44], Ch. 1 studies Schwarz methods with non-minimal overlap. Since the convergence of Schwarz methods improves with increasing overlap [72, 50], this is a possible remedy for the slow convergence of inverters at small quark masses. In Ch. 2 we discuss Schwarz methods with optimized boundary conditions [27]. The boundary conditions are responsible for the coupling of subdomains and are thus important for the convergence speed.[2] Apart from these two Schwarz-based methods, which rely on an overlapping domain decomposition, we introduce a Schur method for QCD in Ch. 3, which is based on a non-overlapping domain decomposition (see for example Refs. [72] and [50] for a general introduction of the method). The Schur method inverts the corresponding Schur complement with a preconditioned Krylov subspace method. The Schur complement and its preconditioner both involve inversions on local blocks and thus make the method suitable for a parallel computing architecture. This is in contrast to the Schwarz method where only the preconditioner comprises block inversions.

**Sign function and overlap operator** In the majority of QCD simulations the Wilson discretization or the staggered fermion formulation is used for the Dirac operator, in which case the cost of applying the operator, i.e., $C_{\text{Dirac}}$, is fixed. If we however use the overlap Dirac operator [54, 55], which is a discretization of the Dirac operator that respects chiral symmetry, we need to compute the sign function of a (complex) matrix (e.g., the Wilson Dirac operator) when applying the overlap operator. Since the kernel

[2] We use *convergence speed* synonymously with *rate of convergence*.

of this sign function is a huge matrix, Krylov subspace methods are employed, so $C_{\text{Dirac}}$ is variable and there is room for optimization.

In Part II we study two methods for computing the sign function of an operator efficiently. The nested Krylov subspace method for the sign function (Ch. 4) is a joint work with Jacques Bloch and is published as Ref. [7]. The method is based on a Krylov-Ritz method — which approximates the sign function using a projection on a Krylov subspace — and adds a further projection to a smaller, nested Krylov space in order to reduce the cost of computing the sign of the Ritz matrix. In Ch. 5, which is published as Ref. [36], a double-pass variant of multi-shift BiCGstab is introduced. In conjunction with a partial fraction expansion this can be used to compute the sign function, and thus provides a well performing alternative to the nested Krylov-Ritz method.

**Hybrid Monte Carlo** HMC [19] numerically integrates a set of Hamilton equations. Since the force term in these equations of motion contains the inverse of the Dirac operator, the expensive inverse (applied to a vector) has to be computed at every time step of the numerical integration. Thus, to minimize the cost, it is crucial to minimize the number of evaluation points without impairing the precision. A partially related aspect is the autocorrelation time, which is the auxiliary HMC time (roughly proportional to the number of evaluation points and thus the total cost) between two configurations that can be considered as independent. Given that only independent configurations can decrease the statistical error of a measurement, it is important to minimize autocorrelations.

We discuss optimizations of HMC in Part III. In Ch. 6 we introduce a modification of HMC based on a linear transformation which exposes the center-of-mass degrees of freedom of small blocks of lattice sites and gives us explicit control over their HMC parameters. Since different physical scales of a problem are typically governed by different effective interactions, it is intuitively clear that they have different optimal HMC parameters. The centers of mass form a coarse grid, i.e., correspond to a coarse scale, so after the linear transformation we can optimize HMC parameters on different scales. In Ch. 7 we briefly discuss application of the Schur complement method introduced in Ch. 3 to HMC. This is in analogy to the method introduced in Ref. [45] and bases on the improved condition number of the Schur complement, in order to reduce the magnitude of the forces that occur during an HMC trajectory.

For more detailed introductions we refer to the individual chapters. Conclusions are presented chapter-wise, and an additional global summary is given at the end, in Ch. 8.

# Part I

# Inverters and preconditioners

# Chapter 1

# Schwarz methods

## 1.1 Introduction

Almost ten years ago physicists working on lattice QCD started to use domain decomposition methods based on the Schwarz alternating procedure, which can be used as a direct inverter or as a preconditioner for a Krylov subspace inverter. The starting points are Refs. [43] and [44] that introduced the algorithm for lattice QCD, but in general the algorithm was already widely known in the partial differential equation community and is based on work from Ref. [65]. The method is very well suited for parallel computers and triggered a great deal of work on the topic. For general introductions we refer to the books Refs. [50] and [72].

For small quark masses, which lead to a bad condition number of the Dirac operator, Schwarz methods converge poorly or can even diverge [47]. Therefore, they are sometimes combined with other methods like inexact deflation (same reference), which however suffer from the high setup cost when used for computing forces in an HMC trajectory. This is our motivation for the present study. So far all QCD-related work on Schwarz methods we know of uses minimal overlap. Since the convergence of the method is known to improve with increasing overlap, we implement and study the Schwarz method with more overlap. The complexity of the application code increases drastically compared to the implementation with minimal overlap, which may be the reason why it has not yet been studied.

This chapter is organized as follows. We give a brief introduction to additive and multiplicative Schwarz methods in Sec. 1.2, the implementation is outlined in Sec. 1.3, and our numerical studies are given in Sec. 1.4. Finally, we give conclusions in Sec. 1.5.

## 1.2 Schwarz methods

For the purpose of this work we always consider discretizations on a hypercubic lattice without boundaries, e.g., closed to a torus. Furthermore, we assume that each lattice site couples only to its neighboring lattice points (we allow nearest-neighbor and all diagonal couplings, but not beyond that). This restriction is not necessary, but simplifies notation and derivations to a great extent, and is fulfilled in most lattice QCD simulations. For computing force terms for HMC (or propagators in data analysis) we

wish to solve a linear problem on a domain $\Omega$,

$$Au = f, \tag{1.1}$$

where in our case $\Omega$ is the whole (finite) space-time volume. Since $A$ is a large, sparse matrix, typically Krylov subspace methods are used to compute an approximation to $u = A^{-1}f$. A different method, introduced by Schwarz [65] and now named after him, is based on a domain decomposition. We start with a simple intuitive explanation and work out the details afterwards, in the remainder of this section.

We split the degrees of freedom in the system into two subsets, or domains, $\Omega_1$ and $\Omega_2$. After proper index reordering, the matrix $A$ has a block structure and we can write the linear system as

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}. \tag{1.2}$$

The matrices $A_{11}$ and $A_{22}$ describe coupling of sites within domain $\Omega_1$ and $\Omega_2$, respectively. $A_{12}$ and $A_{21}$ couple sites in domain $\Omega_1$ to sites in $\Omega_2$. We can write the matrix equation as two separate equations,

$$A_{11}u_1 + A_{12}u_2 = f_1, \tag{1.3a}$$
$$A_{22}u_2 + A_{21}u_1 = f_2. \tag{1.3b}$$

Let us assume we know the solution $u_2$ in domain $\Omega_2$. Then we can rewrite Eq. (1.3a) as

$$u_1 = A_{11}^{-1}(f_1 - A_{12}u_2), \tag{1.4}$$

i.e., the solution can be expressed via $u_2$ by an inversion of the local submatrix $A_{11}$. Of course, in general we do *not* know $u_2$. However, we can start with some approximation $\tilde{u}_2$ differing from the solution by the — potentially large — error $\varepsilon = \|u_2 - \tilde{u}_2\|$. We can then compute an approximation to the solution in $\Omega_1$,

$$\tilde{u}_1 = A_{11}^{-1}(f_1 - A_{12}\tilde{u}_2), \tag{1.5}$$

and use Eq. (1.3b) to find a new solution in domain $\Omega_2$, based on the approximate solution $\tilde{u}_1$,

$$\tilde{\tilde{u}}_2 = A_{22}^{-1}(f_2 - A_{21}\tilde{u}_1). \tag{1.6}$$

The corresponding error vectors are

$$u_1 - \tilde{u}_1 = A_{11}^{-1}(f_1 - A_{12}u_2) - A_{11}^{-1}(f_1 - A_{12}\tilde{u}_2) = A_{11}^{-1}A_{12}(u_2 - \tilde{u}_2), \tag{1.7a}$$
$$u_2 - \tilde{\tilde{u}}_2 = A_{22}^{-1}(f_2 - A_{21}u_1) - A_{22}^{-1}(f_2 - A_{21}\tilde{u}_1) = A_{22}^{-1}A_{21}(u_1 - \tilde{u}_1). \tag{1.7b}$$

If we combine these two equations we obtain

$$u_2 - \tilde{\tilde{u}}_2 = A_{22}^{-1}A_{21}A_{11}^{-1}A_{12}(u_2 - \tilde{u}_2). \tag{1.8}$$

We see that the new error $\varepsilon' = \|u_2 - \tilde{\tilde{u}}_2\|$ is smaller than $\varepsilon$ if the factor $A_{22}^{-1}A_{21}A_{11}^{-1}A_{12}$ reduces the norm of a vector it is multiplied with, i.e., if its spectral radius is smaller than 1. Intuitively this is the case, e.g., if the blocks $A_{11}$ and $A_{22}$ are "larger" than $A_{12}$ and $A_{21}$, i.e., if the matrix $A$ is dominated by diagonal terms. We can continue the alternating procedure, given by Eqs. (1.5) and (1.6), to further reduce the errors,

$$\tilde{u}_2 \Rightarrow \tilde{u}_1 \Rightarrow \tilde{\tilde{u}}_2 \Rightarrow \tilde{\tilde{u}}_1 \Rightarrow \cdots \Rightarrow u_2 \Rightarrow u_1, \tag{1.9}$$

which ultimately yields the exact solution, or a solution with sufficiently small error. This is the *Schwarz alternating procedure*. In practice, depending on the problem, the error does not necessarily decrease and this method often does not converge. Alternatively, since the Schwarz method can be written as block Gauss-Seidel or block Jacobi iteration, one can arrive at a similar convergence argument by considering the respective convergence bounds of these iterations.

We now turn to a more thorough description of Schwarz methods.

### 1.2.1 Overlapping domain decompositions

We wish to solve the problem

$$Au = f \quad \text{in } \Omega, \tag{1.10}$$

where $\Omega$ is a domain without boundaries. We consider an *overlapping* domain decomposition of $\Omega$ into $N$ domains $\Omega_i$, where the $\Omega_i$ do not include their boundaries $\partial\Omega_i$.[1] Note that in the discrete case the minimal value for the overlap of two domains is one lattice spacing (see Sec. 1.2.6 for the definition of overlap). It holds that

$$\Omega = \bigcup_{i=1}^{N} \Omega_i. \tag{1.11}$$

We can write down an equation for the degrees of freedom in domain $\Omega_i$,

$$\begin{pmatrix} A_{\Omega_i\Omega_i} & A_{\Omega_i,\Omega\setminus\Omega_i} \end{pmatrix} \begin{pmatrix} u_{\Omega_i} \\ u_{\Omega\setminus\Omega_i} \end{pmatrix} = \begin{pmatrix} A_{\Omega_i\Omega_i} & A_{\Omega_i,\partial\Omega_i} \end{pmatrix} \begin{pmatrix} u_{\Omega_i} \\ u_{\partial\Omega_i} \end{pmatrix} = \begin{pmatrix} f_{\Omega_i} \end{pmatrix}, \tag{1.12}$$

where $u_{\partial\Omega_i}$ is part of the solution in the domains $\Omega_j$, $j \neq i$, that are overlapping with $\Omega_i$.[2] This equation follows trivially from Eq. (1.10): the first vector component describes couplings inside domain $\Omega_i$, the second component couplings to everything else, i.e., $\Omega\setminus\Omega_i$, which is equivalent to couplings to the boundary $\partial\Omega_i$, since we assumed that the matrix $A$ contains only couplings to neighboring sites.

[1] Since we started from a discretized system, the grid on the domains is already defined and fixed. It is possible to have non-matching grids on overlapping domains, i.e., the grid points of two overlapping domains must not coincide. Since QCD does not exhibit any a priori inhomogeneities, we will not consider this case and restrict ourselves to matching grids.

[2] This is the equivalent to, e.g., Eq. (1.3a).

It is very important to realize that the set of equations for all $\Omega_i$ is *not* necessarily the same equation as Eq. (1.10). Since the domains are overlapping, some degrees of freedom may be contained in two or more domains and also their corresponding equations will be duplicated. That is, the solution at site $x$, $u_x$, can be a component of $u_{\Omega_i}$ and $u_{\Omega_j}$, with $i \neq j$. Similarly parts of $A$ and $f$ are also duplicated. As a consequence, Eq. (1.12) (for all $i$) describes a bigger linear system than Eq. (1.10).[3]

We define $A_{\Omega_i} = A_{\Omega_i\Omega_i}$. Then Eq. (1.12) becomes

$$A_{\Omega_i} u_{\Omega_i} + A_{\Omega_i,\partial\Omega_i} u_{\partial\Omega_i} = f_{\Omega_i}, \tag{1.13}$$

which yields

$$u_{\Omega_i} = A_{\Omega_i}^{-1} \left( f_{\Omega_i} - A_{\Omega_i,\partial\Omega_i} u_{\partial\Omega_i} \right). \tag{1.14}$$

Obviously the right-hand side contains the unknown $u_{\partial\Omega_i}$, so one resorts to an iterative solution. There are two main variants, *additive* and *multiplicative* Schwarz, discussed in the following.

### 1.2.2 Additive Schwarz

Starting with an initial guess $u^0$ for the solution, iterate *simultaneously for all domains*

$$u_{\Omega_i}^n = A_{\Omega_i}^{-1}(f_{\Omega_i} - A_{\Omega_i,\partial\Omega_i} u_{\partial\Omega_i}^n), \tag{1.15}$$

usually written as

$$u_{\Omega_i}^n = u_{\Omega_i}^{n-1} + A_{\Omega_i}^{-1}(f_{\Omega_i} - A_{\Omega_i} u_{\Omega_i}^{n-1} - A_{\Omega_i,\partial\Omega_i} u_{\partial\Omega_i}^n). \tag{1.16}$$

The latter form is often preferred, because $u_{\Omega_i}^n$ is written in terms of $u_{\Omega_i}^{n-1}$ plus a correction term. The unknown term $u_{\partial\Omega_i}^n$ is taken to be defined by the subdomain solutions of the neighboring domains,

$$u_{\partial\Omega_i}^n \leftarrow \bigcup_{\substack{j=1 \\ j\neq i}}^{N} u_{\Omega_j}^{n-1}, \tag{1.17}$$

where the union and assignment arrow are meant symbolically, since there is a freedom when defining this boundary condition: one can take any (mean) of the values on the right-hand side (at $x$) to define the left-hand side (at $x$).

### 1.2.3 Multiplicative Schwarz

Again, we start with an initial guess $u^0$ for the solution. Instead of iterating in parallel over all domains, we now iterate *sequentially*, using on the right-hand side the newest

[3] The systems are of equal size for minimal overlap.

available information about the solution,

$$u^n_{\partial\Omega_1} \leftarrow \bigcup_{j=2}^{N} u^{n-1}_{\Omega_j} \tag{1.18a}$$

$$u^n_{\Omega_1} = u^{n-1}_{\Omega_1} + A^{-1}_{\Omega_1}(f_{\Omega_1} - A_{\Omega_1} u^{n-1}_{\Omega_1} - A_{\Omega_1,\partial\Omega_1} u^n_{\partial\Omega_1}), \tag{1.18b}$$

$$\vdots$$

$$u^n_{\partial\Omega_i} \leftarrow \left(\bigcup_{j=1}^{i-1} u^n_{\Omega_j}\right) \cup \left(\bigcup_{j=i+1}^{N} u^{n-1}_{\Omega_j}\right) \tag{1.18c}$$

$$u^n_{\Omega_i} = u^{n-1}_{\Omega_i} + A^{-1}_{\Omega_i}(f_{\Omega_i} - A_{\Omega_i} u^{n-1}_{\Omega_i} - A_{\Omega_i,\partial\Omega_i} u^n_{\partial\Omega_i}), \tag{1.18d}$$

$$\vdots$$

$$u^n_{\partial\Omega_N} \leftarrow \bigcup_{j=1}^{N-1} u^n_{\Omega_j} \tag{1.18e}$$

$$u^n_{\Omega_N} = u^{n-1}_{\Omega_N} + A^{-1}_{\Omega_N}(f_{\Omega_N} - A_{\Omega_N} u^{n-1}_{\Omega_N} - A_{\Omega_N,\partial\Omega_N} u^n_{\partial\Omega_N}), \tag{1.18f}$$

where the union is taken such that the latest available values are used. Note that for the $A_{\Omega_i} u^{n-1}_{\Omega_i}$ term we need not update the solution in the overlapping regions because the term cancels exactly.[4] This is important, because in a parallel implementation of the algorithm it means that we do not have to exchange the whole overlapping region with the neighbors, but only the boundary, thus reducing the communication volume.

The sequential iteration above is obviously badly suited for a parallel algorithm, since the subdomain solutions have to be updated one after the other. However, in decompositions with many domains there are typically only a few overlapping neighboring domains and we can improve upon the pure sequential algorithm as follows.

We say that two domains $\Omega_i$ and $\Omega_j$ *couple* if $A$ has a nonzero coefficient $A_{kl}$ or $A_{lk}$ for at least one pair of sites $k \in \Omega_i$ and $l \in \Omega_j$. Domains that do not couple are *independent*. Independent domains can be treated in parallel, because their respective subdomain solutions do not depend on each other. Continuation of this reasoning leads to defining *coloring*. In an a priori arbitrary manner, we assign colors to sets of domains that do not couple, until each domain has a color. To maximise parallelism we try to minimize the number of colors $C$. As an example we consider a hypercubic domain decomposition. With nearest-neighbor coupling of sites and minimal overlap, only nearest-neighbor domains couple. That is, all even domains are independent and all odd domains are independent. Thus two colors are enough, yielding a chessboard-like coloring.[5] If there are also diagonal couplings or if the overlap is larger than one, the domains will also couple diagonally. In $D$ dimensions we then need $2^D$ colors.

The algorithm can then process all domains of one color at the same time, traversing through the colors sequentially. Since typically the number of colors is much smaller

[4] assuming an exact inversion of the subdomain matrix $A_{\Omega_i}$

[5] If the lattice is closed to a torus we have to assume an even number of domains in each direction.

than the number of domains, $C \ll N$, the parallelism is improved drastically. It is however by a factor $C$ worse than for the additive Schwarz algorithm.

We write down the version of iteration Eq. (1.18) with colors. To this end we define new domains $\Omega_c$, $c = 1, \ldots, C$, one for each color, where each $\Omega_c$ is the union of all original $\Omega_i$ with color $c$. The iteration then reads: for $c = 1, \ldots, C$ do

$$u^n_{\partial\Omega_c} \leftarrow \left( \bigcup_{d=1}^{c-1} u^n_{\Omega_d} \right) \cup \left( \bigcup_{d=c+1}^{C} u^{n-1}_{\Omega_d} \right) \tag{1.19a}$$

$$u^n_{\Omega_c} = u^{n-1}_{\Omega_c} + A^{-1}_{\Omega_c}(f_{\Omega_c} - A_{\Omega_c} u^{n-1}_{\Omega_c} - A_{\Omega_c,\partial\Omega_c} u^n_{\partial\Omega_c}), \tag{1.19b}$$

where the difference is that this iteration runs from 1 to $C$ instead of from 1 to $N$ as Eq. (1.18).

### 1.2.4 Hybrid Schwarz

To improve the parallelism of the multiplicative Schwarz method we can choose to use fewer colors than necessary for true multiplicativity. For the extreme case of just one color we would obtain additive Schwarz, which was already discussed earlier. Let us now consider a two-dimensional problem with quadratic domains. Multiplicative Schwarz would require four colors. If we use only two colors (black and white), we obtain a chessboard coloring. Black domains overlap with their diagonal neighbors, so they are treated in an additive way (the same holds for the white domains). Of course this will diminish the performance of the method. However, only a small part of the subdomain boundary is part of the diagonal neighbors — the largest fraction comes from nearest neighbors. Thus the advantage of the multiplicative Schwarz method over the additive method — the boundary contains updated values from solves of subdomains with another color — is maintained for a large part of the boundary.

The resulting algorithm is now partially additive (subdomain solves on all black (white) domains are done in parallel, even though some of the black (white) domains overlap) and partially multiplicative (subdomain solves in black and white domains alternate). From now on we label this algorithm as *hybrid* Schwarz method. The parallelism of the hybrid method is significantly better than that of the multiplicative method, at the cost of a convergence rate which is somewhere in between that of the additive and multiplicative — the details will depend on the problem to be solved. As a second advantage the communication pattern is simpler. For example, with two colors, nearest-neighbor communication is sufficient during the Schwarz method, for any overlap size.

### 1.2.5 Schwarz method as a preconditioner

The pure Schwarz method iterates over $n$ in Eq. (1.16) or Eq. (1.19) until the solution has reached the required accuracy. Often this iteration converges poorly or even diverges, e.g., for the Wilson operator at small quark masses.[6] As a remedy the

[6] See Sec. 4.2 for a definition of the Wilson operator.

Schwarz iteration can be used as a preconditioner for inverting the matrix $A$ with a Krylov subspace method. The solution to the original linear system

$$Au = f, \tag{1.20}$$

can be found with a Krylov subspace method for $A$, which iteratively approximates the solution $u$ via a polynomial in $A$. This approximation lies in the *Krylov subspace* of degree $k$ for $A$ and $f$, defined as

$$\mathcal{K}_k(A, f) = \mathrm{span}(f, Af, \ldots, A^{k-1}f). \tag{1.21}$$

Many variants of Krylov subspace methods exist, differing in the way of how the approximation in this space is found. Since these methods are well established and beyond the scope of this work, we refer to the review Ref. [70] or references therein for details. Instead of this direct approach, the system can be rewritten as

$$AMM^{-1}u = (AM)v = f, \quad \text{with } u = Mv, \tag{1.22}$$

where $M$ should be an approximation to $A^{-1}$. The solution $v$ of the new linear system is then approximated in the Krylov space $\mathcal{K}_{k'}(AM, f)$. Since for $M \approx A^{-1}$ we have $AM \approx 1$, the convergence of this Krylov method is much better. As preconditioner $M$ we can use one or a few Schwarz iterations with initial guess zero,[7] yielding a low precision approximation to $A^{-1}$. From now on we denote this number of Schwarz preconditioner iterations as $s$. The Krylov inverter for $AM$ will be referred to as *outer inverter*. The inversions for the subdomain problems in the Schwarz method (which are typically done iteratively by a Krylov inverter as well) are then referred to as *inner*, *local*, or *block inversions*.

### 1.2.6 A note on overlap

The definition of overlap can be misleading, especially from the viewpoint of a programmer working with lattices. For clarification, consider the two domains shown in Fig. 1.1. Their overlap is zero, and since we required that a domain does not include its boundary, there is a boundary plane separating the domains. In the discrete case this boundary is a layer of lattice sites. This is the case for the Schur complement algorithm, discussed in Ch. 3, but it is not possible for the Schwarz method. In Fig. 1.2 we show two domains with overlap. The overlap is such that it is exactly one lattice spacing in the discrete case. The boundary sites $\partial\Omega_1$ of $\Omega_1$ are now inside $\Omega_2$, and vice versa. Schwarz methods can be formulated in this case. Note that for distributing data in an algorithm overlap one means that each site is part of exactly one domain. From this viewpoint the domains are non-overlapping, and in some cases it is referred to as a domain decomposition without overlap — this seems to depend on the author.

[7] Using a nonzero initial guess would bring back all vector components which have already been approximated by previous iterations of the Krylov inverter, and would thus negate the previous computations.

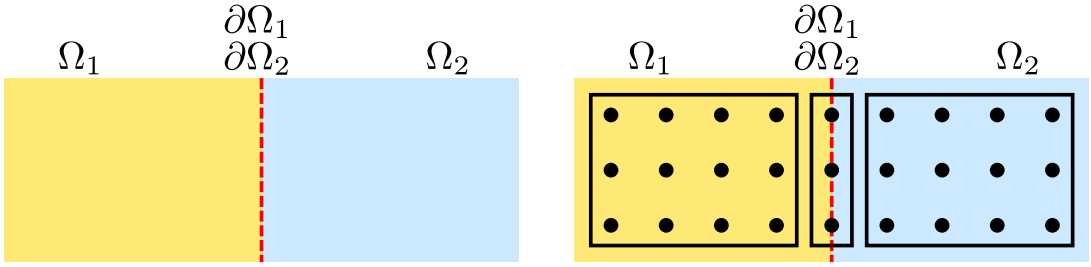



**Figure 1.1:** Two domains $\Omega_1$ and $\Omega_2$ with overlap zero. The continuum case is shown on the left. The boundaries of both domains coincide. On the right the discrete case is shown. The two domains are separated by a boundary layer of lattice sites, which do not belong to either of the domains.

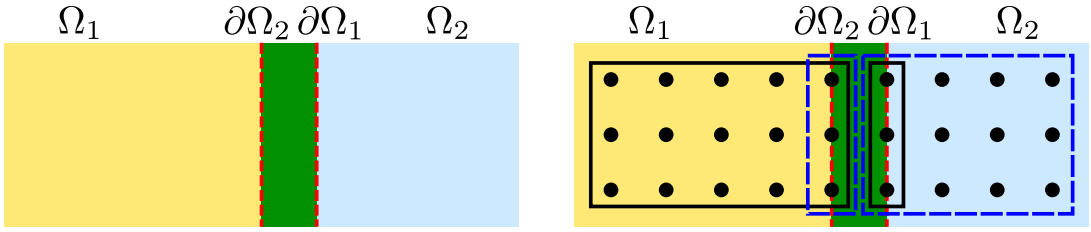



**Figure 1.2:** Two domains $\Omega_1$ and $\Omega_2$ with overlap. The overlapping region is shaded dark (green). The continuum case is shown on the left. The boundary of each domain lies inside the other domain. On the right the discrete case is shown. The sites belonging to either of $\Omega_1$ and $\partial\Omega_1$ are framed with a solid (black) line; $\Omega_2$ and $\partial\Omega_2$ are framed with a dashed (blue) line. The overlap is one lattice spacing, which is the minimal value for a discretized Schwarz method.

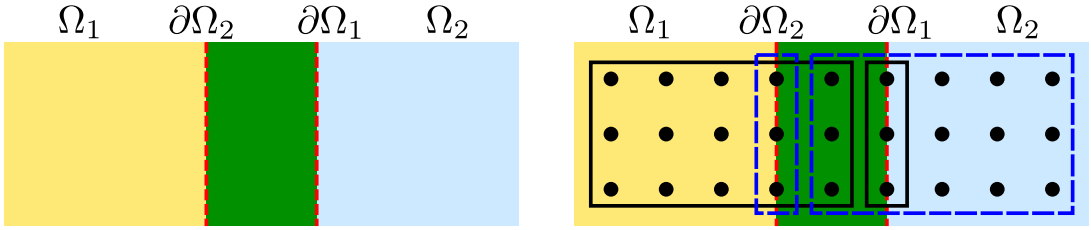



**Figure 1.3:** Same as Fig. 1.2 with an overlap of two lattice spacings. Note that there are now lattice sites belonging to both domains.

In the mathematical literature we are not aware of anyone using this definition. We adopt the mathematical point of view and refer to this as overlap one. In Fig. 1.3 we show two domains with an overlap of two lattice spacings. Now there is one layer of sites that is part of both domains (from the viewpoint of data distribution the overlap would be one site).

In this work we cover exclusively the discrete case. For Schwarz methods the overlap is then always a positive integer multiple of the lattice spacing. From now on we denote this overlap in units of the lattice spacing as $O$, so $O = 1, 2, 3, \ldots$.

| dimension | name | number | sharing neighbors |
|---|---|---|---|
| 0 | vertex | 8 | 3 nearest, 3 diagonal, 1 space-diagonal |
| 1 | edge | 12 | 2 nearest, 1 diagonal |
| 2 | face | 6 | 1 nearest |

**Table 1.1:** Boundary elements of a three-dimensional cube. Neighbor refers to the neighboring cube.

| dimension | name | number | sharing neighbors |
|---|---|---|---|
| 0 | vertex | 16 | 4 nearest, 6 diagonal, 4 space-diagonal, 1 hyperspace-diagonal |
| 1 | edge | 32 | 3 nearest, 3 diagonal, 1 space-diagonal |
| 2 | face | 24 | 2 nearest, 1 diagonal |
| 3 | cell | 8 | 1 nearest |

**Table 1.2:** Boundary elements of a four-dimensional hypercube. Neighbor refers to the neighboring hypercube.

## 1.3 Implementation

This section outlines our implementation of the Schwarz methods with arbitrary overlap. In general, the communication patterns of the Schwarz methods are more complex than for minimal overlap. Therefore we introduce some nomenclature for overlapping domain decompositions in four dimensions. Everyone is familiar with three-dimensional cubes, which have six two-dimensional faces, each shared with a nearest-neighbor cube (assuming we fill the three-dimensional space with cubes), twelve one-dimensional edges, each shared with two nearest neighbors and one diagonal neighbor, and eight vertices, each shared with three nearest neighbors, three diagonal neighbors, and one space-diagonal neighbor. We summarise this in Table 1.1 and give the corresponding notation for four dimensions in Table 1.2, for a four-dimensional hypercube.

### 1.3.1 Even/odd preconditioning

All lattice QCD simulations we know of use even/odd preconditioning, since it yields a speedup by a factor of about two (by reducing the iteration count for convergence to a given error) with relatively small additional programming effort. We briefly discuss this for domain decomposition algorithms. In this case there are two different ways to employ even/odd preconditioning: we can either first do the domain decomposition and then the even/odd preconditioning, or the other way round.

Even/odd preconditioning is based on splitting the lattice sites into two subsets, even and odd.[8] With a Schur complement of the resulting block matrix,

$$\hat{A} = A_{ee} - A_{eo} \left(A_{oo}\right)^{-1} A_{oe}, \tag{1.23a}$$

[8] A four-dimensional analogue to a chessboard, even sites are black fields, odd sites are white fields.

one can rewrite the original set of linear equations as

$$\hat{A}u_e = f_e - A_{eo}\,(A_{oo})^{-1}\,f_o, \tag{1.23b}$$
$$u_o = (A_{oo})^{-1}\,(f_o - A_{oe}u_e). \tag{1.23c}$$

The inversion of $A_{oo}$ is simple, since different odd sites do not couple if the operator has only nearest-neighbor interaction. The main computational work is now the iterative inversion of $\hat{A}$. With respect to the original system, the number of degrees of freedom is reduced to half, but the cost of $\hat{A}$ per site is about twice as high as that of $A$, so the effects cancel one another. However, the condition number is reduced and typically the inversion of $\hat{A}$ takes about half of the iterations it takes for $A$, so the overall gain in computational cost is a factor of two [48].

The first method is as in Ref. [44]. If we first perform a domain decomposition we can apply Eqs. (1.23) straightforwardly to $A_{\Omega_i}$ in each of the subdomain equations given in Eq. (1.13), after splitting the degrees of freedom inside the domain into an even subset and an odd subset. This will reduce the iteration counts of the Krylov inverters for the submatrices $A_{\Omega_i}$. Since these inversions are the major contribution to the total algorithmic cost, this yields an overall speedup of about two as well.

Alternatively, we can first do an even/odd preconditioning and invert $\hat{A}$ with the Schwarz method as preconditioner. This is however significantly more complex than for $A$. We can see from Eq. (1.23a) that $\hat{A}$ couples any sites that are two hops apart, i.e., diagonal neighbors and neighbors that are two hops in direction $\pm\mu$ for all $\mu$. This requires a large effort to implement the domain boundary conditions. The iteration count of the outer inverter will be reduced by a factor of two. An additional even/odd preconditioning of the subdomain equations is no longer (cheaply) possible in this case, because $\hat{A}_{oo}$ contains many off-diagonal terms and the inverse needed to form the Schur complement as in Eq. (1.23a) is not cheaply available.

Since we are not aware of any mention of the latter method in the literature, we implemented both types of even/odd preconditioning for a comparison. The results confirm the very similar cost of both methods, as expected. We conclude that it is not worth to deal with the complex code of the second method and simply use even/odd preconditioning for all subdomain equations from now on.

### 1.3.2 Data layout

We work with a lattice of size $L_\mu$, $\mu = 0, \ldots, 3$, closed to a torus. For the outer solver, i.e., for everything that is not part of the domain decomposition preconditioner, this lattice is distributed over a number of nodes. In direction $\mu$ there are $N_\mu^{\text{node}}$ nodes. We require that the local volume on each node is of equal size. Thus $N_\mu^{\text{node}}$ is a factor of $L_\mu$ and the local volume has side lengths

$$L_\mu^{\text{local}} = L_\mu / N_\mu^{\text{node}}. \tag{1.24}$$

The total number of nodes is

$$N^{\text{node}} = \prod_{\mu} N_{\mu}^{\text{node}}. \tag{1.25}$$

The local volume, i.e., the number of sites assigned to one node, is

$$V^{\text{local}} = \prod_{\mu} L_{\mu}^{\text{local}}. \tag{1.26}$$

For the sites in the local volume we choose to first store all even sites and then all odd sites, each of which are stored in linear order with $x$ as fastest running index and $t$ as slowest index. For specific implementations a different storage order can be of advantage.

In the domain decomposition preconditioner the lattice is covered by overlapping domains. In direction $\mu$ there are $N_{\mu}^{\text{domain}}$ domains. The side length of a domain is $L_{\mu}^{\text{domain}}$, which we give in terms of *sites inside the domain in direction* $\mu$.[9] We require all domains to be of equal size and to have a homogeneous distribution in each direction: the *spread* $S_{\mu}$ (distance between centers of nearest-neighbor domains) in direction $\mu$ is the same for all domains.[10] This yields the restriction that $N_{\mu}^{\text{domain}}$ is a factor of $L_{\mu}$. The minimal possible domain length is then

$$\left(L_{\mu}^{\text{domain}}\right)_{\text{min}} = L_{\mu}/N_{\mu}^{\text{domain}}. \tag{1.27}$$

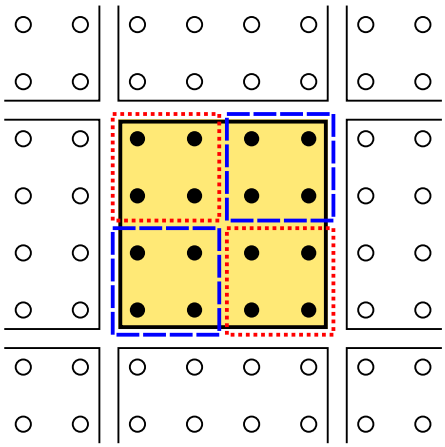
For multiplicative Schwarz we additionally require $N_{\mu}^{\text{domain}}$ to be even, to allow for a periodic coloring.

To avoid load imbalance or idle nodes in the Schwarz method, each node should have exactly one domain of each color (or exactly two, etc.). Then all nodes can work in parallel on each color, iterating sequentially over all colors. For the additive Schwarz method we have one color, so $N_{\mu}^{\text{domain}}$ must be an integer multiple of $N_{\mu}^{\text{node}}$. For multiplicative Schwarz with general overlap, we need 16 colors in four dimensions. Each node holds the corresponding 16 domains, arranged in a $2 \times 2 \times 2 \times 2$ layout as illustrated in Fig. 1.4. More generally, $N_{\mu}^{\text{domain}}$ must be an integer multiple of $2N_{\mu}^{\text{node}}$. With minimal overlap the $2 \times 2 \times 2 \times 2$ layout of domains can be exactly aligned with the boundaries of the local volume (see Fig. 1.4a). With more overlap there will be lattice sites contained in the local domains which are not part of the local volume — one or more shells of sites "enclosing" the local volume are contained in the local domains. For balancing the network load we try to make this overlapping part as symmetric as possible. Complete symmetry is only possible for odd overlap (as in Figs. 1.4a and 1.4c). For even overlap we choose to have one more layer of sites in forward direction (see Fig. 1.4b).[11]

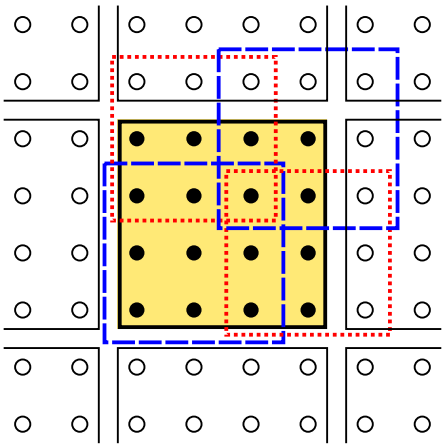
[9] The distance from boundary to boundary is by 1 larger than the number of sites, but we find it more convenient to use the length definition in terms of sites.

[10] For minimal overlap the spread is equal to the domain length, i.e., $S_{\mu} = (L_{\mu}^{\text{domain}})_{\text{min}}$.

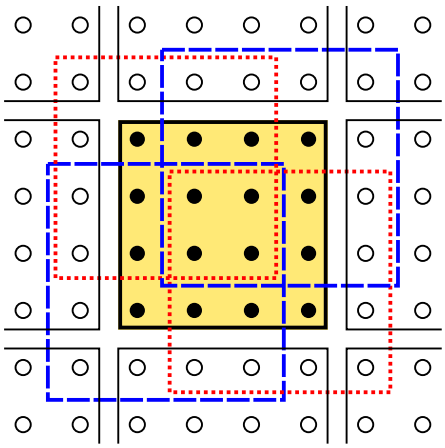
[11] Choosing backwards would be equivalent, this is just an implementation detail and has no influence.



**(a)** Overlap 1: domains do not exceed the area of the local volume.



**(b)** Overlap 2: domains exceed the area of the local volume by one site in forward direction.



**(c)** Overlap 3: domains exceed the area of the local volume by one site in all directions.

**Figure 1.4:** Illustration for the data layout of the Schwarz method. The filled dots framed with a shaded square depict the sites in the local volume which is of size $4 \times 4$. Sites on neighbors are empty dots. The local volume is covered by four domains ($S_\mu = 2$ for all $\mu$) associated with the node. The domains are drawn as dashed (blue) or dotted (red) frames (line type and color are just for clarity and have no other meaning). We show the case of overlap 1 in (a), overlap 2 in (b), and overlap 3 in (c).

The subdomain matrices are typically too large for a direct inversion. Therefore, they are inverted iteratively with a Krylov subspace inverter. We use even/odd preconditioning as explained in Sec. 1.3.1 for these inversions, so we first store all even sites of a domain, then all odd sites.[12] The sites are stored in linear order in memory, with $x$ as fastest index and $t$ as slowest index.

One may ask why the size of the subdomain matrices is not chosen such that a direct inversion can be done. There are two main reasons. First, the Schwarz methods with many small domains typically converge slowly. Second, even for very small domains the storage requirements can be prohibitive, making memory access a bottleneck, or

[12] Exception: when even/odd preconditioning is used for the outer inverter, we cannot use even/odd preconditioning for the inner inversion. In that case *all* sites are stored in linear order in memory. Usually this is however not the case, see Sec. 1.3.1.

| $S_\mu$ | domain size | $O$ | $V^{\text{domain}}$ | $\frac{V^{\text{domain}}}{V^{\text{domain}}_{\text{min}}}$ | $V^{\text{boundary}}$ | $\frac{V^{\text{boundary}}}{V^{\text{boundary}}_{\text{min}}}$ |
|---|---|---|---|---|---|---|
| 4 | $4 \times 4 \times 4 \times 4$ | 1 | 256 | 1 | 512 | 1 |
| | $5 \times 5 \times 5 \times 5$ | 2 | 625 | 2.44 | 1000 | 1.95 |
| | $6 \times 6 \times 6 \times 6$ | 3 | 1296 | 5.06 | 1728 | 3.38 |
| 6 | $6 \times 6 \times 6 \times 6$ | 1 | 1296 | 1 | 1728 | 1 |
| | $7 \times 7 \times 7 \times 7$ | 2 | 2401 | 1.85 | 2744 | 1.59 |
| | $8 \times 8 \times 8 \times 8$ | 3 | 4096 | 3.16 | 4096 | 2.37 |
| 8 | $8 \times 8 \times 8 \times 8$ | 1 | 4096 | 1 | 4096 | 1 |
| | $9 \times 9 \times 9 \times 9$ | 2 | 6561 | 1.60 | 5832 | 1.42 |
| | $10 \times 10 \times 10 \times 10$ | 3 | 10000 | 2.44 | 8000 | 1.95 |
| | $11 \times 11 \times 11 \times 11$ | 4 | 14641 | 3.57 | 10648 | 2.60 |

**Table 1.3:** Example of domain volumes and domain-boundary volumes for some typical domain sizes with varying overlap. Asymmetric domain sizes are not given here, but are of course also possible.

can even exhaust the memory resources. An example for this is given in Appendix A.

In Table 1.3 we give the volume of a domain (number of contained lattice sites) for typical domain sizes which are suitable for computers. For small spread $S_\mu$, which allows for a small minimal domain size, the volume grows rapidly if the overlap is increased. The boundary of a domain is then larger than its volume:[13] for example, for hypercubic domains of length $L$ the domain volume is $L^4$ compared to a boundary volume of $8 \cdot L^3$, i.e., only for $L > 8$ the boundary volume is smaller than the domain volume. For larger domains the growth is slower and the boundary volume is of the same order as the domain volume.

In Table 1.4 the storage requirement for the same set of domain configurations is given. We see a disadvantage of the Schwarz algorithm with overlap larger than one: not only does the domain volume increase due to the overlap, but additionally we need 16 colors instead of 2, which all have to be stored on the same node. The result is an algorithm with a severely increased memory footprint. Note however that this is the total amount of consumed memory. Since all colors are treated sequentially the amount of data needed *at the same time* is smaller by a factor of 16. This is crucial for processor cache usage and code optimization: from this point of view the algorithms with 16 colors are *not* much worse than those with 2 colors.

### 1.3.3 Data flow

We outline the data flow if the Schwarz method is used as a preconditioner. When the outer solver calls the Schwarz preconditioner, the data from the local volumes has to be mapped to the domains. For non-minimal overlap each node has to obtain the

[13] This may seem counterintuitive. The reason is the large space dimension of four: the domain length has to be kept small to keep the domain volume at a manageable value, which in turn yields a bad surface to volume ratio.

| $S_\mu$ | domain size | $O$ | $C$ | all local domains |
|---|---|---|---|---|
| 4 | $4 \times 4 \times 4 \times 4$ | 1 | 2 | 96 kB |
| | $4 \times 4 \times 4 \times 4$ | 1 | (16) | (768 kB) |
| | $5 \times 5 \times 5 \times 5$ | 2 | 16 | 1875 kB |
| | $6 \times 6 \times 6 \times 6$ | 3 | 16 | 3888 kB |
| 6 | $6 \times 6 \times 6 \times 6$ | 1 | 2 | 486 kB |
| | $6 \times 6 \times 6 \times 6$ | 1 | (16) | (3888 kB) |
| | $7 \times 7 \times 7 \times 7$ | 2 | 16 | 7203 kB |
| | $8 \times 8 \times 8 \times 8$ | 3 | 16 | 12288 kB |
| 8 | $8 \times 8 \times 8 \times 8$ | 1 | 2 | 1536 kB |
| | $8 \times 8 \times 8 \times 8$ | 1 | (16) | (12288 kB) |
| | $9 \times 9 \times 9 \times 9$ | 2 | 16 | 19683 kB |
| | $10 \times 10 \times 10 \times 10$ | 3 | 16 | 30000 kB |
| | $11 \times 11 \times 11 \times 11$ | 4 | 16 | 43923 kB |

**Table 1.4:** Storage requirements for multiplicative Schwarz algorithms for the double precision pseudo-fermion vectors (4 spin and 3 color degrees of freedom) of all local domains, computed as $12 \cdot V^{\text{domain}} \cdot C \cdot 16$ Byte. For minimal overlap $O = 1$ two domain colors are enough, but we also give the value for 16 colors in parentheses.

overlapping parts (for each domain) from the neighboring nodes. These are the sites marked as empty circles in Fig. 1.4 which lie inside one or more of the local domains. Given that the overlap is not only with nearest neighbors, this means sending to and receiving from all neighbors — nearest neighbors, diagonal neighbors, space-diagonal neighbors, and hyperspace-diagonal neighbors. The amount of data to be exchanged decreases with the dimension of these boundary elements of the local volume, e.g., in Fig. 1.4c with an overlap of $O = 3$ we must exchange four sites with each nearest neighbor and one site with each diagonal neighbor. Since this mapping is done only once per preconditioner call, the resulting performance impact is small.

The Schwarz iteration proceeds as in, e.g., Eq. (1.19). Local solving of the subdomain problems alternates with updating the approximation to the solution. In practice we only update the boundary values of the domain whose subdomain problem is to be solved next, as illustrated in Fig. 1.5. This involves obtaining data from other domains on the node and from domains on neighboring nodes. In two or more dimensions this includes communication with non-nearest neighbors. The exact domain which contains the latest approximation to the solution depends on the update order of the colors. Since it is straightforward but cumbersome to write this down in detail we leave it aside (but see also Sec. 1.3.4).

At the end of the Schwarz preconditioner call the data from the domains has to be mapped back to the local volume. We choose the simplest option, namely copying the part of the domain corresponding to minimal overlap back to the local volume. A more complex option would be to use the average over all overlapping domains. This would reduce the noise in the overlapping regions but requires an additional data exchange with all neighbors. The positive impact on the preconditioner quality can be expected

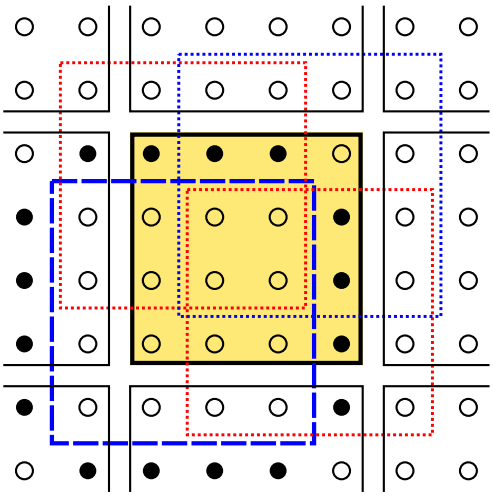


**Figure 1.5:** Illustration of domain boundaries. The boundary sites of the domain on the bottom left (framed with a dashed (blue) line) are represented as filled dots. The forward neighbors (towards right or top) are contained in the three other local domains. However, depending on the update order of the colors, parts of the sites might have more recent solution values on the neighboring nodes. The backward neighbors (towards left or bottom) are contained in domains that are assigned to neighboring nodes. A priori all backward neighbors are part of domains on the nearest-neighbor nodes, but diagonal data exchange can be necessary to obtain the latest solution values.

to be minor, which is why we opted for avoiding this communication overhead.

If the Schwarz method is used directly, i.e., not as a preconditioner, the given procedure corresponds to a complete solve.

### 1.3.4 Implementation variants of multiplicative Schwarz

In multiplicative Schwarz the boundary values of a subdomain are taken to be the most recent available. That is, after each update of a subdomain solution it has to be ensured that all neighboring domains are updated. This is trivial in the case of minimal overlap: since there is coupling only with nearest neighbors, the latest updated boundary values will automatically be used. For more overlap there is also diagonal coupling. Then we are forced to explicitly send the updated solution to a diagonal neighbor, e.g., as illustrated in Fig. 1.6. The Schwarz method updates the domains in turns, starting with $\Omega_1$, then $\Omega_2$. We consider what happens when $\Omega_3$ is reached. First, the current value of the solution on the boundaries must be obtained. $\partial\Omega_3$ is completely contained in $\Omega_1$ and $\Omega_4$, so communication with nearest-neighbor domains would be sufficient. However, since $\Omega_2$ was updated *after* $\Omega_1$ (and after $\Omega_4$, which has not yet been updated), there is a newer version of the solution available in the region indicated in the figure by the arrows (3) and (4). As a consequence, a *diagonal* data transfer from $\Omega_2$ to $\Omega_3$ is necessary.

Similarly, for intersections of 8 domains in three or more dimensions, a space-

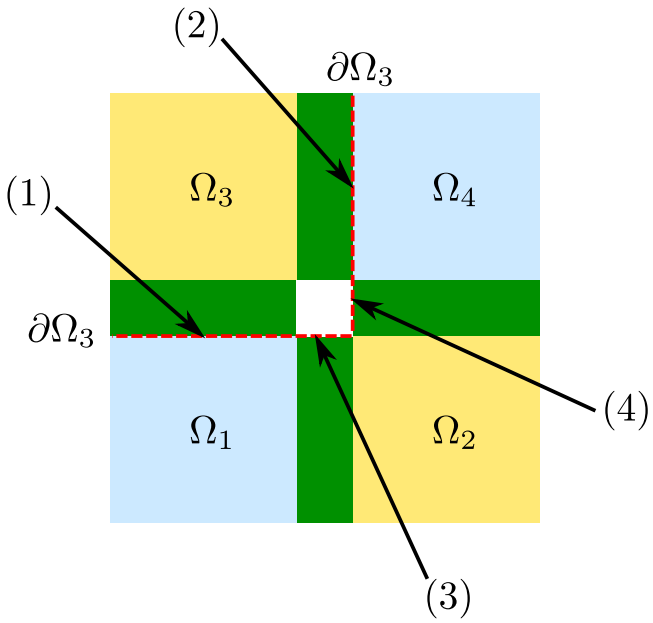



**Figure 1.6:** Four overlapping domains. Regions where two domains overlap are shaded dark (green). In the white center region all four domains overlap. The boundary of domain $\Omega_3$ is indicated with a dashed (red) line. The arrows point out different sections of the boundary, (1) is in the interior of $\Omega_1$, (2) is in $\Omega_4$, (3) is in $\Omega_1$ and $\Omega_2$, and (4) is in $\Omega_2$ and $\Omega_4$.

diagonal data exchange is necessary, and so is a hyperspace-diagonal exchange for intersections of 16 domains in four (or more) dimensions.

In our implementation we chose to limit the update of these boundary values to a certain level. This may simplify the implementation and avoid network communication with non-nearest neighbors. We label the respective variant of the multiplicative Schwarz method as *level 0* (updated solution only obtained from nearest neighbors), *level 1* (updated solution obtained from nearest and diagonal neighbors), *level 2* (updated solution obtained from nearest, diagonal, and space-diagonal neighbors), and so on. The consequences are studied in Sec. 1.4.

## 1.4 Numerical study of Schwarz methods with varying overlap

In this section we systematically study the performance of the Schwarz method. We first consider the method itself in Sec. 1.4.1 and then, based on the experience gained, go on to use it as a preconditioner in Sec. 1.4.2.

For our tests we use the Wilson Dirac operator with clover term (see Sec. 4.2 for a definition of the Wilson operator). The clover term was first introduced in Ref. [68], for further details and references see Ref. [38]. The implementation uses MPI for parallelization. Crucial parts of the code use hand-optimized SSE code. We use double precision throughout the computation. Often more than 80% of the floating-point operations are part of the preconditioner, where single precision can be used

without affecting the overall accuracy of the algorithm. When using single precision one can expect a speed improvement of about a factor of two.

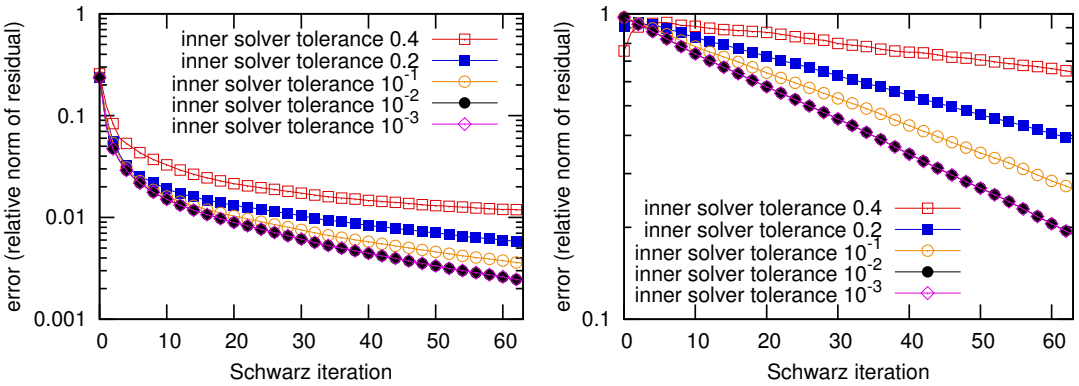
We base our benchmarks on $N_f = 2$ configurations generated by the QCDSF Collaboration. For details on these configurations see, e.g., Ref. [2]. We mainly work with $32^3 \times 64$ lattices at $\beta = 5.29$. This corresponds roughly to a lattice spacing of $a = 0.072$ fm and a lattice length of $L = 2.29$ fm. The configurations were generated at $\kappa = 0.13632$ corresponding to a pion mass of about 300 MeV, to be compared with a physical pion mass of roughly 140 MeV. For the purpose of our benchmarks we would like to test the inverter at lower pion masses, close to the physical point. Therefore we also invert the Dirac operator at a valence quark mass that is lower than the sea quark mass at which the configurations were generated, i.e., $\kappa_{\text{val}} > \kappa_{\text{sea}}$.[14] We use three values for $\kappa_{\text{val}}$, 0.13632, 0.13640, and 0.13645, corresponding to pion masses of roughly 300 MeV, 185 MeV, and 115 MeV, respectively. These pion masses are based on a rough extrapolation and do not take into account side effects like growing finite-volume effects. To push the Schwarz method to its limits we also use $\kappa_{\text{val}}$ up to 0.13660. The majority of the benchmarks was done on the iDataCool cluster at the University of Regensburg.

### 1.4.1 Performance of the Schwarz method

We start by studying the behavior of the multiplicative Schwarz method with minimal overlap ($O = 1$) and two colors. This corresponds to the method as introduced in Refs. [43] and [44]. We perform the subdomain solves with low precision. This is justified by Fig. 1.7, where we show the convergence rate depending on the requested relative norm $\varepsilon$ of the residual in the inner solver. For a quite large error $\varepsilon = 10^{-1}$ the negative influence on the overall convergence of the method is small, for $\varepsilon = 10^{-2}$ or lower it is invisible. A low precision solve needs much fewer iterations and will consequently yield a large overall reduction of the algorithmic cost. From now on we typically work with $\varepsilon = 10^{-1}$.

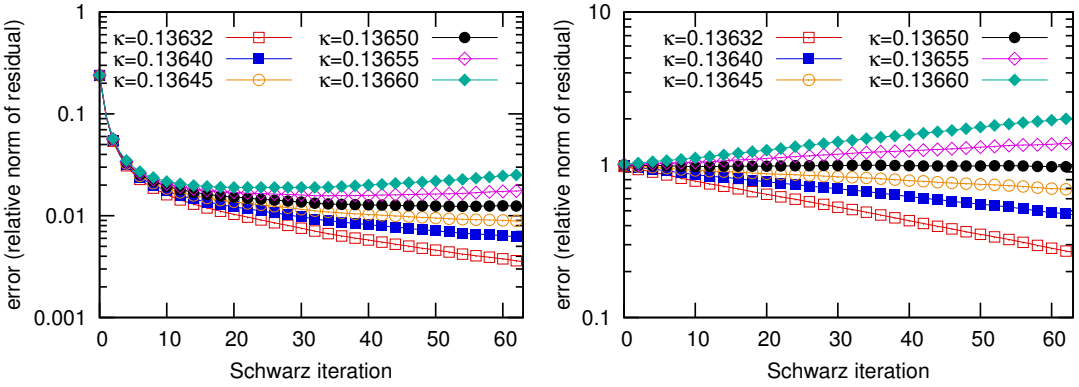
Let us note that we give two different subplots in Fig. 1.7. The Schwarz method shows a very quick initial convergence, but slows down afterwards (the same behavior can also be observed for Krylov subspace methods). The reason is the high part of the spectrum, which can easily be dealt with by inverters. For a proper estimation of the performance of the method we have to consider the main region of linear convergence, neglecting the initial region. For our tests we do this by considering the method in the context of a preconditioner for a Krylov inverter. The first preconditioner call shows the quick convergence behavior (Fig. 1.7a). In the second (and all further) calls the high part of the spectrum is already well approximated, so the method has to work harder to improve the approximation and consequently converges slower. As we intend to use the Schwarz method as a preconditioner which will typically be called many

[14] At the time when we started these benchmarks there were no configurations with lower pion mass available to us, therefore we had to take this artificial step.




**(a)** First preconditioner call **(b)** Second preconditioner call

**Figure 1.7:** Convergence of the multiplicative Schwarz method with minimal overlap for domains of size $8 \times 8 \times 4 \times 8$ with 2 colors. We show the dependence of the convergence on the requested tolerance (relative residual norm) of the inner solver. The two panes show the first and second call of the Schwarz preconditioner during an inversion with a Krylov method to illustrate the difference between the initial quick convergence in (a) and the subsequent linear convergence region towards the right of (a), in (b), and in all subsequent preconditioner calls.




**(a)** First preconditioner call **(b)** Second preconditioner call

**Figure 1.8:** Convergence of the multiplicative Schwarz method with minimal overlap for domains of size $8 \times 8 \times 4 \times 8$ with 2 colors for inner tolerance $10^{-1}$. We show the dependence of the convergence on $\kappa$.

times, the total performance is dominated by the behavior shown in Fig. 1.7b. From now on we give the convergence results for the second call.[15]

In Fig. 1.8 we show the convergence rate of the Schwarz method for different values of the valence quark mass, by varying the value of $\kappa$ (the sea quark mass is the same in all cases, $\kappa_{\text{sea}} = 0.13632$). The method stagnates if $\kappa$ is too large and eventually diverges if we increase $\kappa$ further. Note that even for values of $\kappa$ where the method

[15] Alternatively one could drop the first few iterations with quick convergence in the first call and consider only the thus truncated convergence rate.

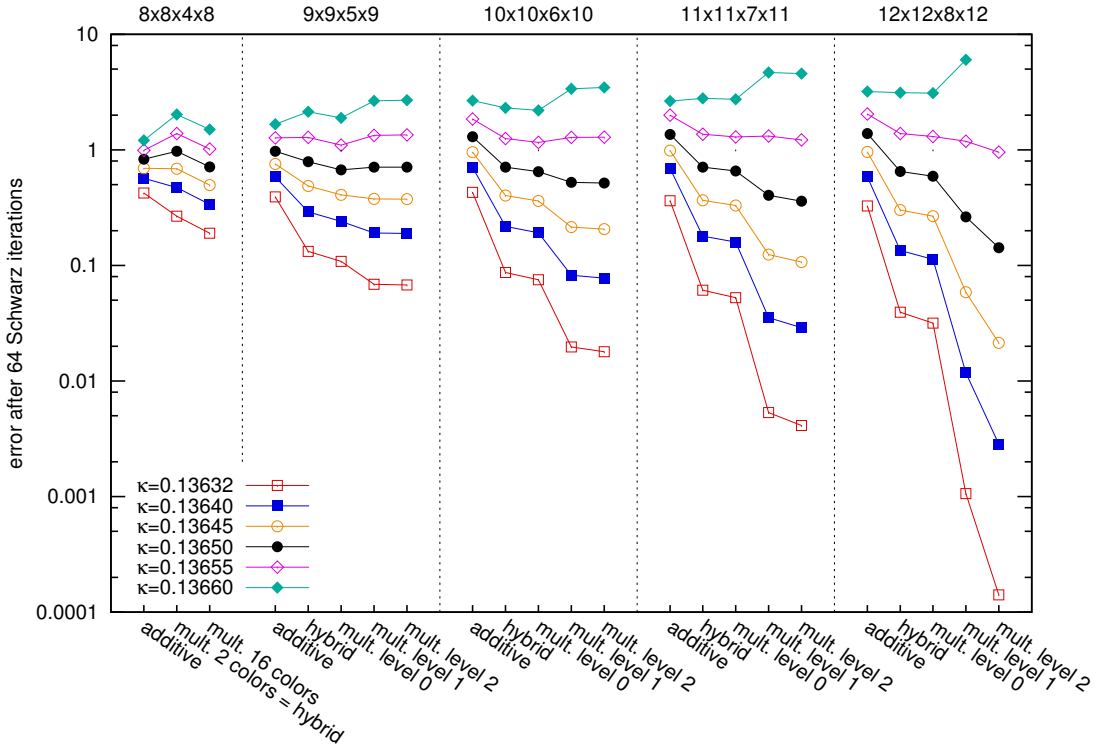



**Figure 1.9:** Error of the Schwarz method after 64 iterations. The spread, and thus the domain size for minimal overlap, is $8 \times 8 \times 4 \times 8$. Keeping the number of domains fixed, we increase the overlap in steps of one, yielding domains of size $9 \times 9 \times 5 \times 9$, $10 \times 10 \times 6 \times 10$, and so on (the value is indicated on top of the plot). We print the results for additive Schwarz, hybrid Schwarz, and multiplicative Schwarz (levels 0 to 2) for different values of $\kappa$.

diverges we still observe a quick initial convergence.

We now turn to the Schwarz method with more overlap, which is the new contribution of this study. We start by a comparison of the performance of the additive, multiplicative, and hybrid (with two colors) variants of the Schwarz method discussed in Secs. 1.2.2, 1.2.3, and 1.2.4, respectively. For the multiplicative Schwarz method with minimal overlap, which we use as a basis for comparison, two domain colors are enough, since only nearest-neighbor domains overlap. In this case the hybrid Schwarz method is completely equivalent to the multiplicative method. Furthermore, for minimal overlap the multiplicative Schwarz method is equivalent for all "levels", i.e., distinguishing is not necessary (see Sec. 1.3.4).

We show the error of the Schwarz method after a fixed number of iterations in Fig. 1.9, and observe the following. As expected, the convergence of the additive method is worse than that of the hybrid method, which in turn is worse than the multiplicative method. Increasing the level of the multiplicative method improves the convergence. With increasing overlap the convergence rate improves significantly, which fulfills our initial hopes. Increasing the level of the multiplicative method beyond

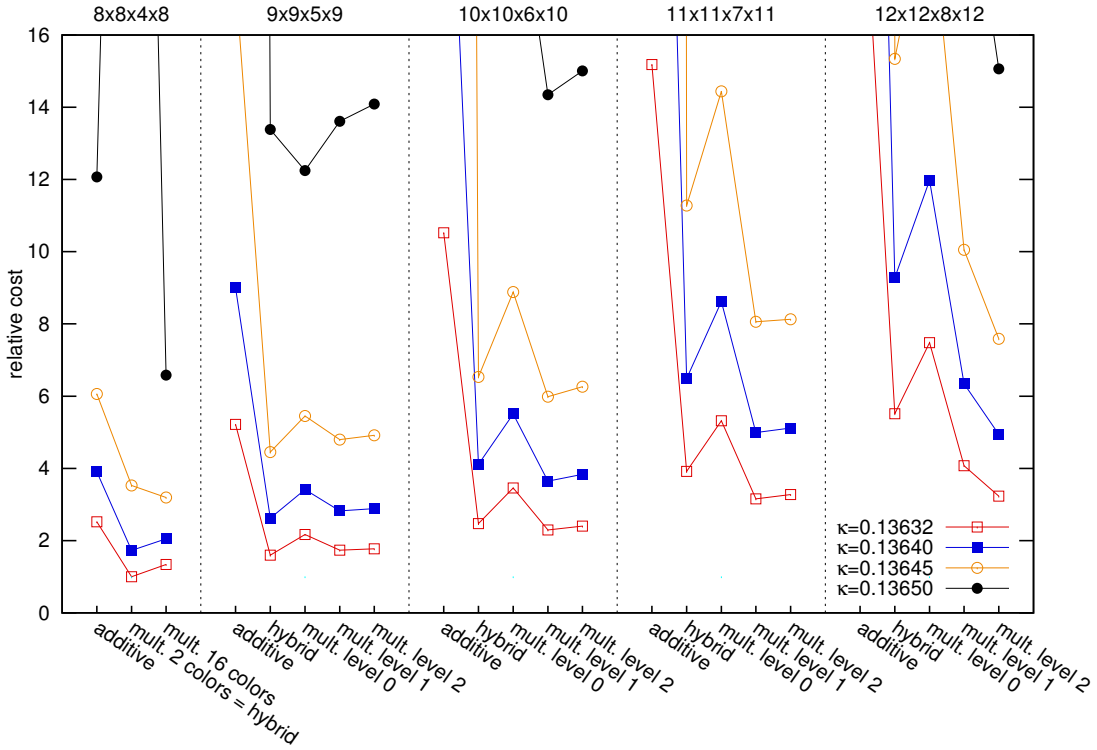
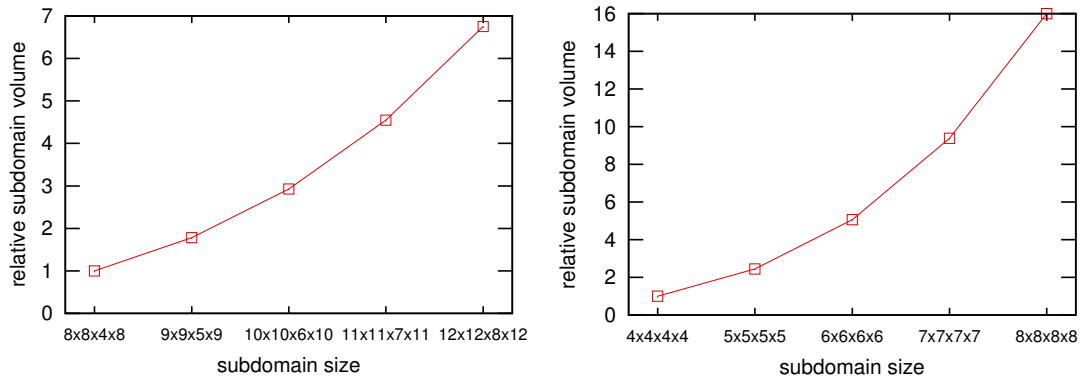



**Figure 1.10:** Cost of the Schwarz method as in Fig. 1.9 for convergence to a given error.




**(a)** Relative volume of subdomains with spread $8 \times 8 \times 4 \times 8$ for increasing overlap.

**(b)** Relative volume of subdomains with spread $4 \times 4 \times 4 \times 4$ for increasing overlap.

**Figure 1.11:** Dramatic increase of subdomain volume with increasing overlap. The effect is more severe for smaller spread.

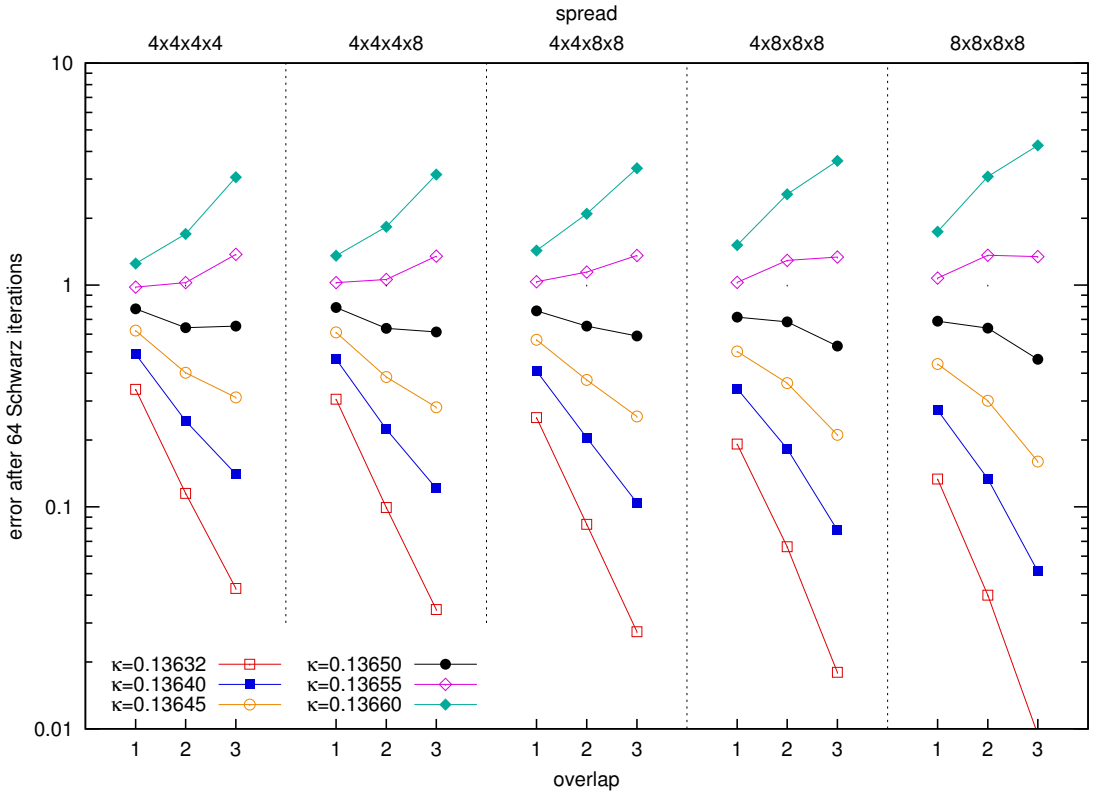



**Figure 1.12:** Error of the multiplicative Schwarz method with 16 colors after 64 iterations depending on the spread (given on top). We show results for overlap 1, 2, and 3 and different values of $\kappa$.

1 yields improvements only for large overlap. From now on we use level 2, unless stated otherwise. Finally, it is important to consider the dependence on $\kappa$ in relation to the overlap. Regardless of the overlap, stagnation (error $\approx 1$) occurs at the same value $\kappa = 0.13655$ and divergence (error growing $> 1$) at $\kappa = 0.13660$. In fact the divergence rate is increased by increased overlap. This means that by increasing the overlap we can increase the *rate* of convergence/divergence, however we *cannot* improve the Schwarz method if it already stagnates or diverges for minimal overlap.

These considerations of the error of the Schwarz method are not suitable as a measure for the performance as a stand-alone inverter (or preconditioner). Instead we consider the cost for obtaining a fixed error in Fig. 1.10. The given values are based on the approximation that the cost for applying the Wilson operator to a domain is proportional to the domain volume.[16] Since the method is well suited for parallel computers and given that it is usually possible to hide most of the network latency, we furthermore neglect the influence of the network. The starting point of our considerations, namely the multiplicative Schwarz method with minimal overlap and two colors,

[16] This is not entirely accurate, since the domain boundary cuts off some hopping terms.

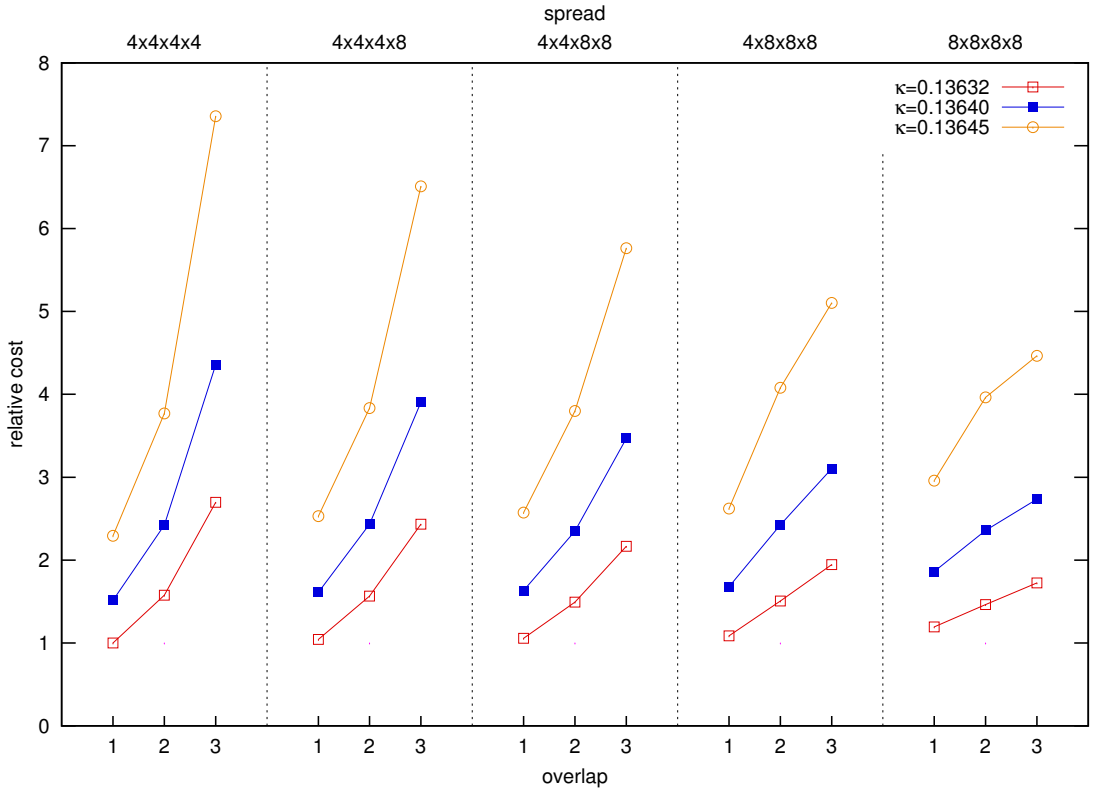



**Figure 1.13:** Approximate cost of the multiplicative Schwarz method (as in Fig. 1.12) for convergence to a given error, relative to the cost of the multiplicative Schwarz method with minimal overlap with 16 colors for spread $4 \times 4 \times 4 \times 4$.

is taken as a reference point. None of the other options shows a better performance, despite some significant improvements we observed in Fig. 1.9. The reason is mainly the quickly increasing volume of the subdomains if the overlap is increased, as can be seen in Fig. 1.11. As a secondary effect the number of inner iterations increases slightly if the overlap is increased.

Finally, we consider the effect of the domain spread. In Fig. 1.12 we give the error after 64 Schwarz iterations. Doubling the spread (and thus the minimal domain size) in every direction leads to more than a factor of two reduction in the error. In Fig. 1.13 we see that for minimal overlap increasing the domain size does not balance the higher cost of solving the subdomain problems on larger domains.[17] For overlap 2 or more, however, larger domains pay off.

### 1.4.2 Performance of the Schwarz method as preconditioner

We now turn to the Schwarz method as preconditioner (see Sec. 1.2.5). These considerations are important because, as we will see in the following, the performance

[17] The iteration count of the inner inverter increases with the domain volume.

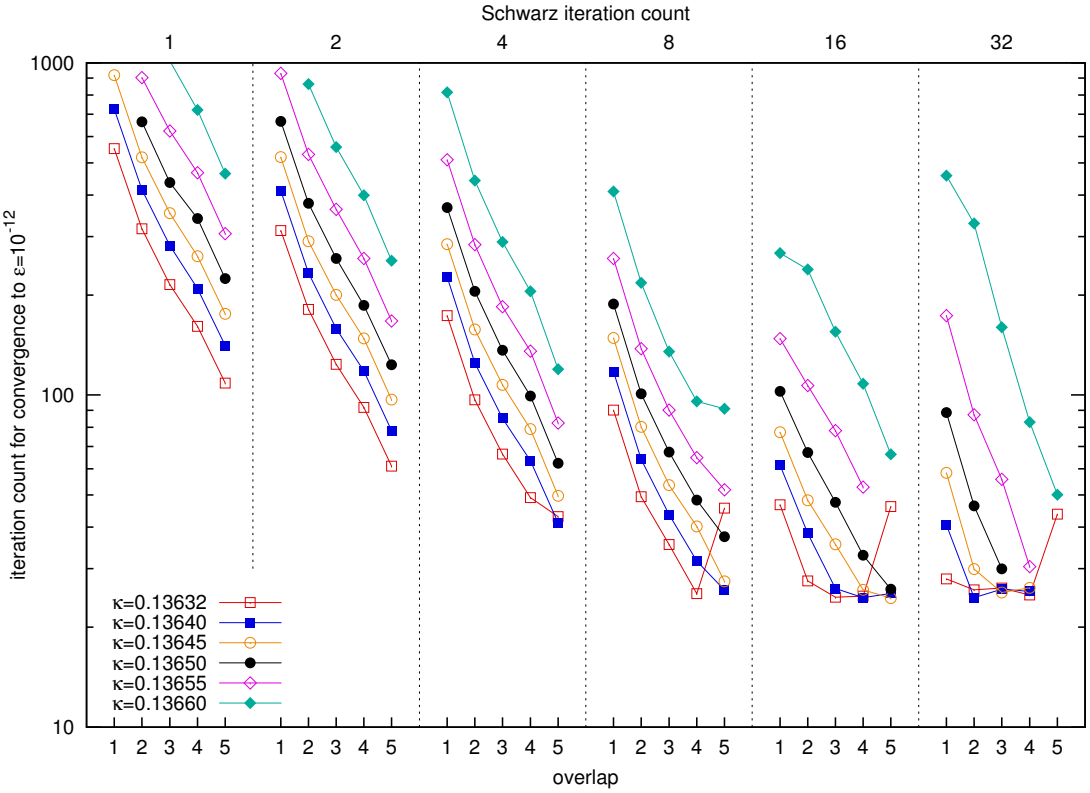



**Figure 1.14:** Iteration count of fGMRES-DR(24,8) with Schwarz preconditioner for convergence to a relative residual norm of $\varepsilon = 10^{-12}$. The iteration count $s$ of the Schwarz method is given on the top axis. The overlap $O$ is varied from 1 to 5, at a spread of $8 \times 8 \times 4 \times 8$. The corresponding domain sizes thus are $8 \times 8 \times 4 \times 8$, $9 \times 9 \times 5 \times 9$, $10 \times 10 \times 6 \times 10$, $11 \times 11 \times 7 \times 11$, and $12 \times 12 \times 8 \times 12$. We give results for different values of $\kappa$. Note that the stagnation at high iteration count $s$ is an artifact of our implementation: we check the error (and stop the iteration) only after a full GMRES cycle (32 and 24 iterations for the first and all further cycles, respectively).

of the Schwarz method as a stand-alone inverter does sometimes tell little about the performance as a preconditioner. Throughout this section we use a tolerance $\varepsilon = 10^{-1}$ for the inner inverter (the subdomain problems). As outer inverter we choose flexible GMRES with deflated restarts [25] (fGMRES-DR).

In Fig. 1.14 we give the iteration count of fGMRES-DR(24,8) (restart size 24, deflation size 8) for convergence to a relative residual norm $\varepsilon = 10^{-12}$. Increasing the overlap and/or the Schwarz iteration count $s$ decreases the GMRES iteration count. Note that this happens initially (for small $s$) also for $\kappa = 0.13660$. From Fig. 1.9 we see that the Schwarz method *diverges* in this case. Nevertheless, it works well as a preconditioner for few iterations, and only when increasing $s$ to 16 or 32 the outer iteration count grows. The reason is that the Schwarz method works as an iterative smoother: it reduces the contribution of the high part of the spectrum (which reduces the condition number and thus the iteration count of the outer inverter). For low

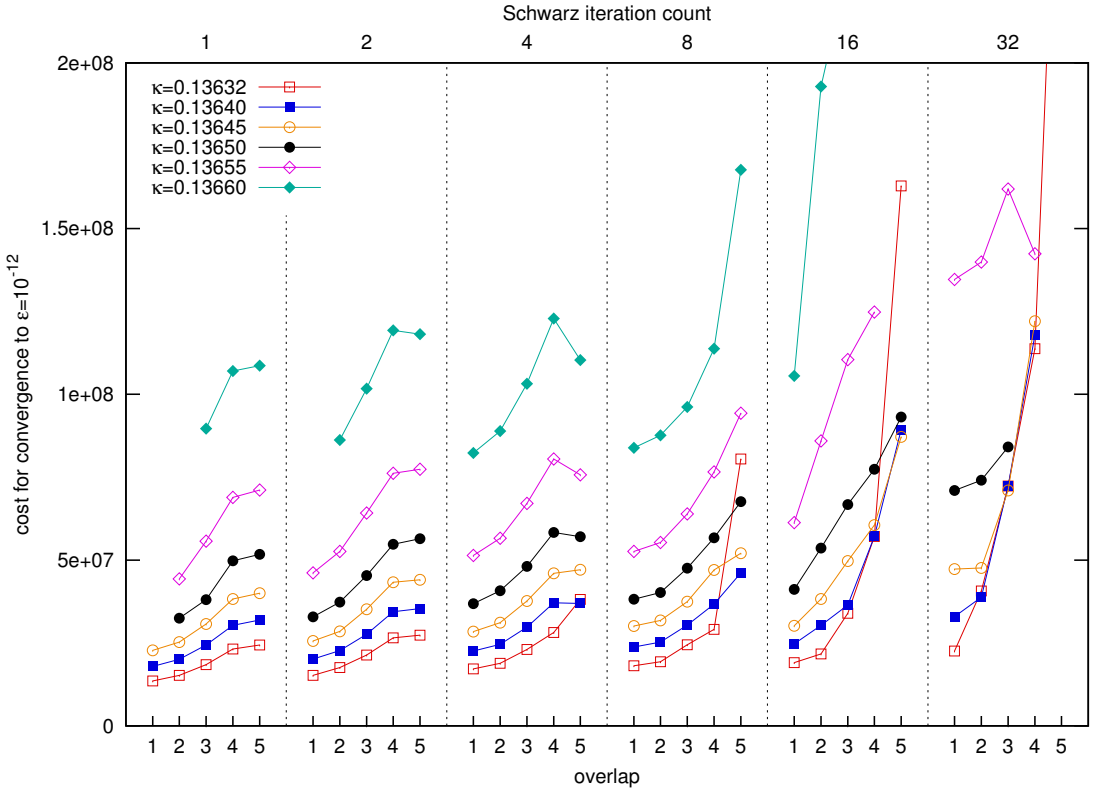



**Figure 1.15:** Approximate algorithmic cost for the Schwarz preconditioned fGMRES-DR inversion of Fig. 1.14. The absolute scale of the axis is not relevant.

$s$ the smoothing is incomplete, so increasing $s$ improves the convergence rate of the outer inverter. Once the Schwarz method has reduced the high-mode contributions to almost zero, a further increase of $s$ will mainly increase the work in the preconditioner (or increase the error introduced by a wrong approximation for the low modes) but will not improve the smoothing. Thus, the convergence rate of the outer inverter will only marginally improve (or even deteriorate), while the cost per iteration grows.

The outer iteration count is not a good measure for the algorithm performance. We give the approximate cost for the same algorithm parameters in Fig. 1.15 and a zoom to smaller $y$ values in Fig. 1.16. The cost grows with increasing overlap, due to an increased domain volume and slightly increased iteration count for solving the subdomain problems. However, the relative penalty for increasing the overlap by one is comparatively small. Increasing the Schwarz iteration count $s$ slightly deteriorates the performance. Both observations should not be treated as absolute statements however: an actual implementation for a specific computer can often favor an algorithm with fewer outer iterations, i.e., with higher $s$ or larger domains (more overlap). There are several reasons for this: cache effects, limitation of the network bandwidth, fewer global dot-products which may be costly in large parallel computers, and finally the possibility of single precision preconditioning which favors moving work from the

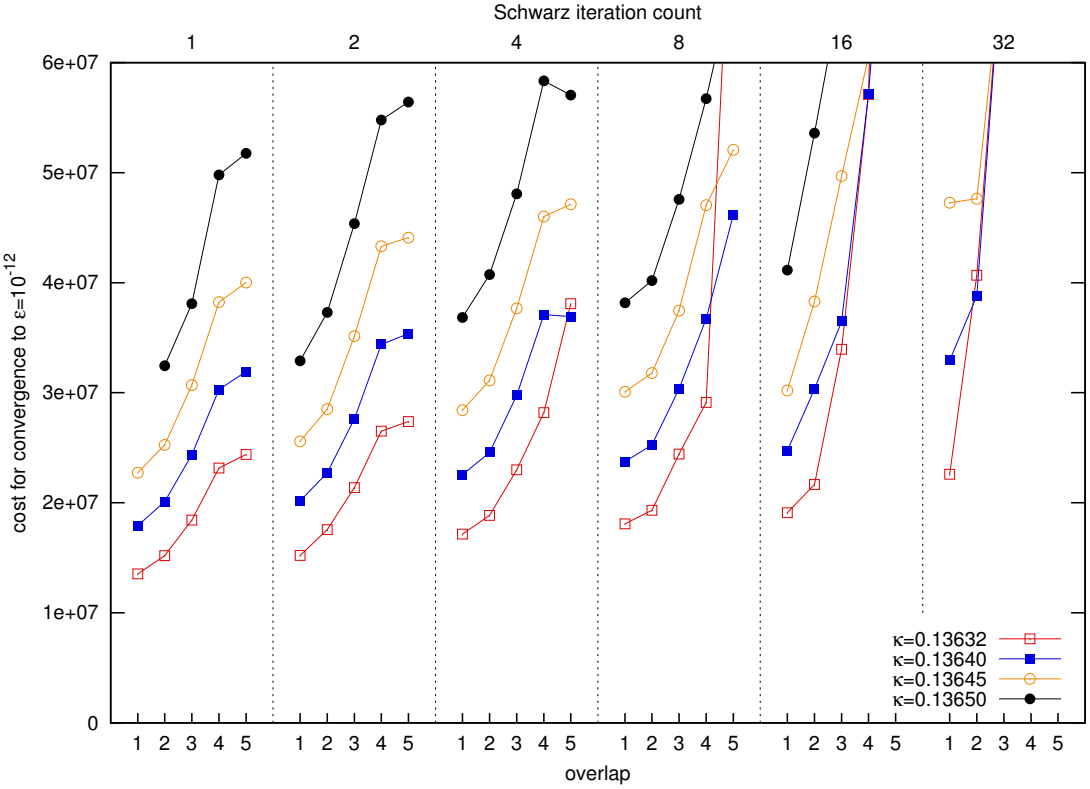



**Figure 1.16:** Same as Fig. 1.15, zoomed to smaller $y$ values.

outer (double precision) inverter into the preconditioner. Since the true performance depends strongly on the exact computing architecture and on how much effort is put into optimizing a specific algorithm for this very machine, we do not attempt to compare timings here.

So far all results we gave in this section were based on the same gauge configuration. To verify that the results are not a coincidence for this randomly selected configuration and to give a more general algorithmic cost estimate, we performed benchmarks for 16 different configurations. These configurations are all from the same Markov chain but separated by 50 or 100 trajectories, so we can expect the correlations between configurations to be low. We give an overview of the convergence cost in Fig. 1.17. The convergence speed depends on the configuration, however the differences are not large. For small $\kappa$ the variation is typically of the order of 50%. For large $\kappa$ we observe up to a factor of three difference between the two extremes of easiest and hardest solve.

In Fig. 1.18 we give a different plot of the same data points for $\kappa = 0.13632$ and $\kappa = 0.13655$. The behavior when increasing the overlap is consistent, up to small fluctuations: if at overlap $o_1$ an inversion for configuration $c_1$ is more expensive than an inversion for configuration $c_2$, it will also be more expensive at $o_2$. For the small $\kappa$ value minimal overlap is significantly cheaper than overlap $> 1$, for all configurations. For the larger value of $\kappa$ overlap 1 and overlap 2 are almost equal, in fact for a few

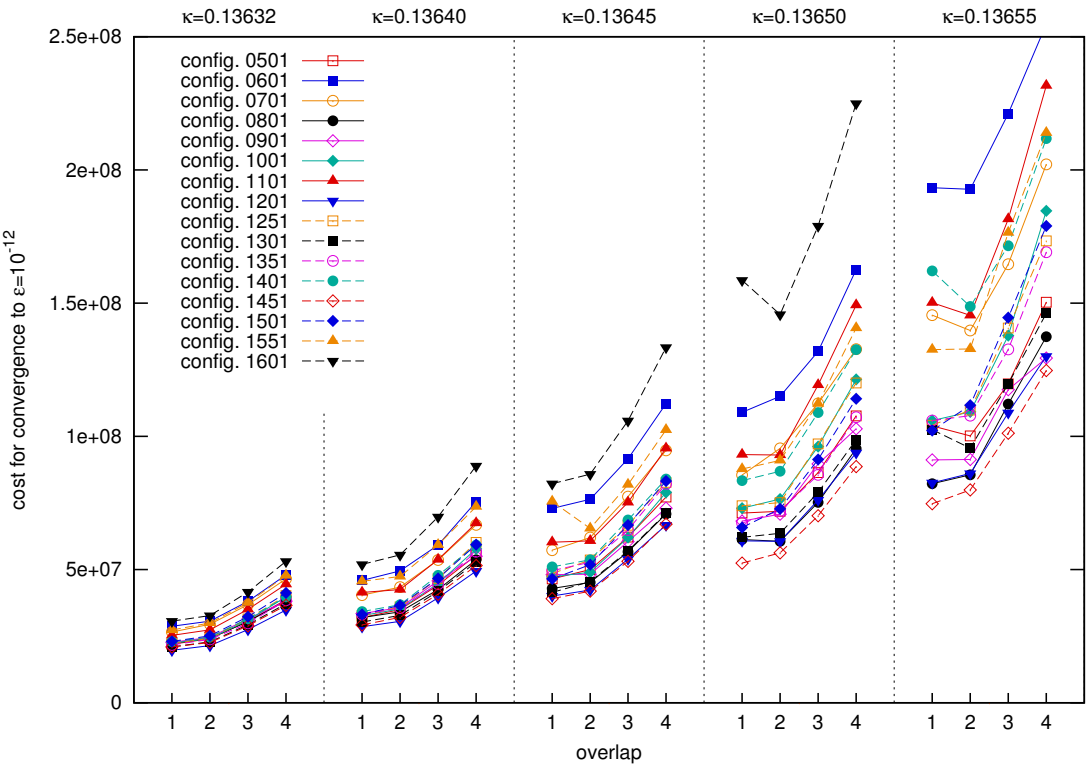



**Figure 1.17:** Overview of convergence cost of fGMRES-DR(24,8) with $s = 4$ iterations of the multiplicative Schwarz preconditioner for 16 different configurations, overlap size $O = 1$ to 4, and six different values of $\kappa$.

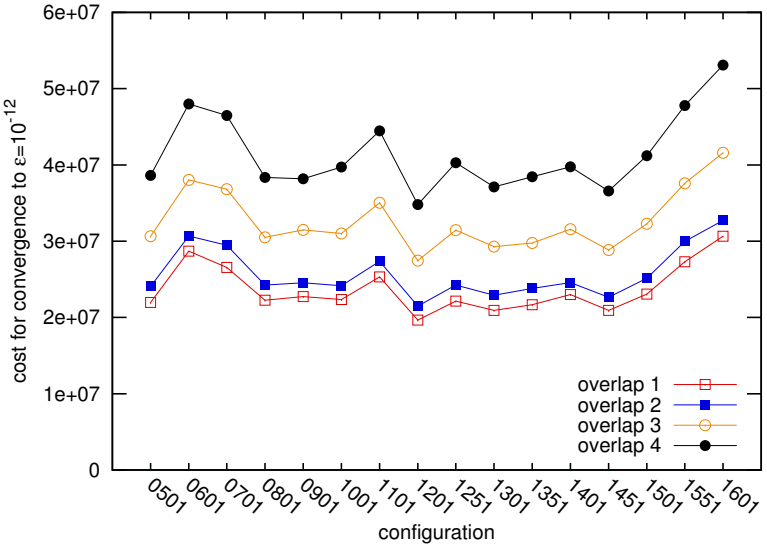
configurations overlap 2 performs better, which we had not observed before.

## 1.5 Conclusions

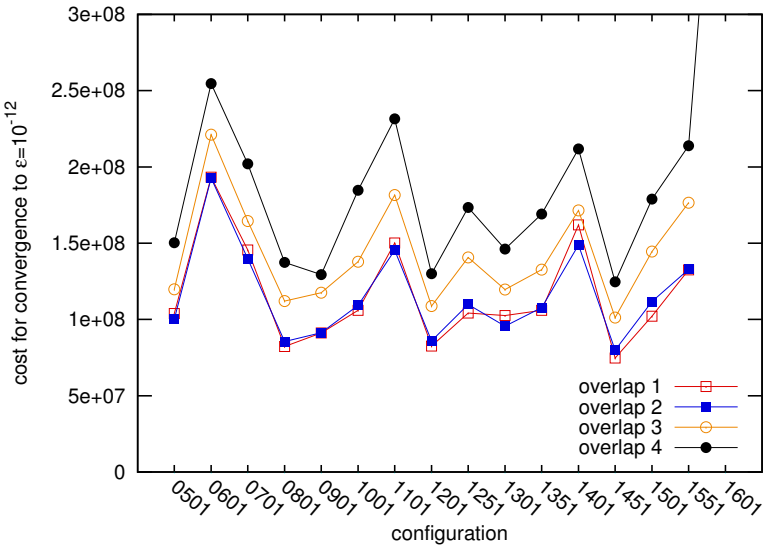
In this chapter we extended the overlapping domain decomposition methods based on the Schwarz method to non-minimal overlap. The method with minimal overlap has successfully been used in lattice QCD as a preconditioner for Krylov subspace inverters for several years.

We showed that the generalization to arbitrary overlap can be implemented and yields algorithmic costs comparably to, but usually slightly larger than, the minimal-overlap method. With non-minimal overlap the implementation is significantly more complex and communication with non-nearest neighbors becomes necessary. Regardless of the overlap size the method is well suited for mapping to a parallel computer, since it contains local inversions (which make up the largest part of the algorithmic cost), and thus reduces the impact of the network on the algorithm performance.

Just as for other linear problems, also for QCD increased overlap yields increased convergence speeds. The operation count to converge to a given error behaves in the




**(a)** $\kappa = 0.13632$




**(b)** $\kappa = 0.13655$

**Figure 1.18:** Convergence cost for 16 configurations with fGMRES-DR(24,8) with 4 iterations of the multiplicative Schwarz method (overlap 1 to 4). For a low value of $\kappa$ as in (a), minimal overlap is best for all tested configuration. For high $\kappa$ in (b), the relative difference between different overlap values is smaller. Overlap 1 and 2 are now of equal cost. For configuration 1601 the outer inverter does not converge. This could be fixed by increasing the restart size.

opposite way, however: increased overlap increases the volume and iteration count for the subdomain problems, which more than cancels the increased convergence speed.

We emphasize that this does not necessarily mean that minimal overlap is always to be favored. The true performance depends strongly on the computing architecture and on the work one puts into code optimization. In cases where local computations are cheap, it could pay off to put more effort into the preconditioner. Since this can also be achieved by simply increasing the Schwarz iteration count, it is not a priori clear whether increasing the overlap to achieve something similar can be of any advantage. Given that these considerations are strongly case dependent, we did not consider timings of the algorithms, but only provided a study of the relative operation counts, such that the most suitable algorithm can be chosen based on our results.

An important (negative) result is the behavior of the Schwarz method in case it stagnates or diverges: increased overlap does not cure the convergence problems at high $\kappa$. Larger overlap improves the convergence speed, but it cannot turn a non-converging method into a converging one.

Finally, we mention two possible optimizations which we consider for the Schur complement inverter in Ch. 3, and which could also improve the convergence of the Schwarz method. First, one can choose $\kappa$ in the preconditioner lower than the valence $\kappa$, to improve its convergence. Second, oblate domains can improve the convergence of the subdomain problems and could thus lead to a reduction of the total cost.

# Chapter 2

# Optimized Schwarz methods

## 2.1 Introduction

In this chapter we consider the class of so called *optimized Schwarz methods*, which modify the coupling between domains to achieve better convergence properties. For a thorough discussion we refer to Refs. [27] and [73]. In our context some other references are also of interest: Ref. [28] considers domain decompositions with cross-points, Ref. [17] considers the effect of discretization, and Ref. [14] studies a problem on a sphere, i.e., with closed boundaries.

As mentioned in the introduction to the Schwarz methods in Sec. 1.1 and as observed throughout Ch. 1, the convergence of these methods is poor for small quark masses. In Ch. 1 we used exclusively Dirichlet boundary conditions, which is the simplest option in terms of implementation but does not necessarily have the best convergence properties. Obviously there are many possible modifications to the boundary condition, e.g., by adding derivatives in the boundary plane or perpendicular to it, which offer potential for improved convergence.

The nomenclature generally used is *optimal Schwarz* for a method with the best possible choice of boundary conditions (which can be found analytically for some problems) and *optimized Schwarz* for some approximation to the optimal Schwarz method. Large parts of the available (mathematical) literature study the method in the continuum or with finite-element discretizations. Since this is not suitable for QCD (see Sec. 3.4.1), we go into some technical details for the application to finite-difference discretizations.

In the present chapter we first introduce and discuss optimized Schwarz methods for the Laplace operator in two dimensions in Sec. 2.2. This sets the stage for the application to QCD in Sec. 2.3. As mentioned above, an analytic study of the convergence is possible for some simple problems, but similar attempts for QCD fail quickly due to the gauge fields, so our analysis is purely numerical. A conclusion is given in Sec. 2.4.

## 2.2 Introductory study of optimized Schwarz methods

### 2.2.1 Boundary condition classification

We consider a problem on a domain $\Omega$ as in Sec. 1.2,

$$Au = f. \tag{2.1}$$

We cover $\Omega$ by two overlapping domains $\Omega_1$ and $\Omega_2$, with corresponding subdomain solutions $u_1$ and $u_2$. We put the boundary of $\Omega_1$ parallel to the $y$ direction at $x = b$, where we allow $y$ to have more than one component. Throughout this section we give the boundary condition only on this boundary. Those for $u_2$ or other boundaries of $u_1$ (e.g., when the domain is closed to a torus) follow by symmetry. The generalization to higher dimension is trivial if only two domains are used. Using more than two domains is also straightforward, but in order to maintain good convergence special care might be necessary on the corners of domains, where more than two domains overlap.

The subdomain equation for $u_1$ for the (additive or multiplicative) Schwarz method can then be written as [27]

$$Au_1^n = f_1 \quad \text{in } \Omega_1, \tag{2.2}$$

where $n$ is the index for the Schwarz iteration. Since $A$ contains derivatives, this subdomain problem is undefined without specification of a boundary condition, which specifies the values of $u_1^n$ on $\partial\Omega_1$. For clarity of notation we can write the above equation as in Eq. (1.13),

$$A_{\Omega_1} u_1^n + A_{\Omega_1,\partial\Omega_1} u_{\partial\Omega_1}^n = f_1, \tag{2.3}$$

where in our case we denote the boundary vector $u_{\partial\Omega_1}^n$ as $u_1^n(b, y)$ for arbitrary $y$. Since $u_1^n(b, y)$ is not inside $\Omega_1$ we need to define it via a boundary condition. There are several options for specifying the values on this boundary, classified as follows:

- Dirichlet boundary conditions,

$$u_1^n(b, y) = u_2^{n-1}(b, y) \quad \forall y, \tag{2.4}$$

  which was used throughout Ch. 1, e.g., in Eqs. (1.17) and (1.18). Implementation is trivial and yields a modification of the right-hand side of the subdomain equations.

- Neumann boundary conditions,

$$\partial_x u_1^n(b, y) = \partial_x u_2^{n-1}(b, y) \quad \forall y. \tag{2.5}$$

  Implementation yields a modification of the subdomain matrices and the right-hand side. To implement boundary conditions with first-order derivatives perpendicular to the boundary we need a minimal overlap of 2 (in the discrete case).

- Robin boundary conditions,

$$(\partial_x + p)u_1^n(b, y) = (\partial_x + p)u_2^{n-1}(b, y) \quad \forall y, \tag{2.6}$$

  for a parameter $p \in \mathbb{R}$. Implementation yields modification of the subdomain matrices and of the right-hand side of the subdomain equations. This boundary condition is also called *zeroth-order* (optimized), since it is of order zero along the boundary. Note that Neumann and Dirichlet boundary conditions can be obtained from Robin boundary conditions for $p = 0$ and $p \to \infty$, respectively.

- Robin boundary conditions with a term along the boundary for optimization,

$$(\partial_x + p - q\partial_y^2)u_1^n(b,y) = (\partial_x + p - q\partial_y^2)u_2^{n-1}(b,y) \quad \forall y, \tag{2.7}$$

  for parameters $p, q \in \mathbb{R}$. Implementation yields modification of the subdomain matrices and of the right-hand side of the subdomain equations. This boundary condition is also called *second-order* (optimized), since it is of order two in the boundary.[1] Higher orders are also possible but in this work we use at most second order.

For discretization of the underlying problem we also have to give a discrete version of the boundary conditions. The derivatives for Neumann or Robin boundary conditions need to be expressed via finite differences.

### 2.2.2 Discretization of boundary conditions

Let $A$ be a hypercubic nearest-neighbor discretization of the operator in Eq. (2.1), where we put the lattice spacing to 1 for simplicity. We assume a splitting into subdomains in the $x$ direction, i.e., the domain boundaries are hyperplanes perpendicular to the $x$-axis. Consider the forward boundary of a domain $\Omega_1$ in direction $\hat{e}_x$, at $x = b$.[2] The neighboring domain is labeled $\Omega_2$.[3] To write down the modifications due to the boundary conditions we have to use a more verbose notation.

We label sites in our lattice via their coordinates, $\{x, y\}$, where $y$ can have more than one component. We reorder the matrix indices of $A$ such that $x$ is the slowest index and define

$$A_{xx'} = \begin{pmatrix} a_{\{x,1\}\{x',1\}} & \cdots & a_{\{x,1\}\{x',n_y\}} \\ \vdots & \ddots & \vdots \\ a_{\{x,n_y\}\{x',1\}} & \cdots & a_{\{x,n_y\}\{x',n_y\}} \end{pmatrix}, \tag{2.8}$$

where $n_y$ is the number of sites in $y$ direction. Due to the nearest-neighbor coupling we have $A_{xx'} = 0$ for $x \notin \{x'-1, x', x'+1\}$ and $a_{\{x,y\}\{x',y'\}} = 0$ for $x \notin \{x'-1, x', x'+1\}$ or $y \notin \{y'-1, y', y'+1\}$. We can now write

$$A = \begin{pmatrix} A_{11} & \cdots & A_{1n_x} \\ \vdots & \ddots & \vdots \\ A_{n_x1} & \cdots & A_{n_xn_x} \end{pmatrix}, \tag{2.9}$$

[1] Usually only derivatives of even order are considered for symmetry reasons, so we did not include first-order terms in this list.

[2] That is, the last lattice site inside $\Omega_1$ is at $x = b - 1$.

[3] Possible other boundaries of the domain $\Omega_1$ are omitted in our notation for clarity. In practice they are treated in an analogous way.

and equivalently for the vectors $u$ and $f$. $n_x$ is the number of sites in $x$ direction. In this notation the subdomain problem for $\Omega_1$ reads

$$\begin{pmatrix} A_{11} & \cdots & A_{1,b} \\ \vdots & \ddots & \vdots \\ A_{b-1,1} & \cdots & A_{b-1,b} \end{pmatrix} \begin{pmatrix} u^n_{\Omega_1,1} \\ \vdots \\ u^n_{\Omega_1,b} \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_{b-1} \end{pmatrix}, \quad (2.10)$$

or, when taking into account the nearest-neighbor coupling,

$$\begin{pmatrix} A_{11} & A_{12} & & 0 & 0 \\ A_{21} & \ddots & \ddots & & \vdots \\ & \ddots & A_{b-2,b-2} & A_{b-2,b-1} & 0 \\ 0 & & A_{b-1,b-2} & A_{b-1,b-1} & A_{b-1,b} \end{pmatrix} \begin{pmatrix} u^n_{\Omega_1,1} \\ \vdots \\ u^n_{\Omega_1,b-1} \\ u^n_{\Omega_1,b} \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_{b-1} \end{pmatrix}, \quad (2.11)$$

which yields

$$\begin{pmatrix} A_{11} & A_{12} & & 0 \\ A_{21} & \ddots & \ddots & \\ & \ddots & A_{b-2,b-2} & A_{b-2,b-1} \\ 0 & & A_{b-1,b-2} & A_{b-1,b-1} \end{pmatrix} \begin{pmatrix} u^n_{\Omega_1,1} \\ \vdots \\ u^n_{\Omega_1,b-1} \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ A_{b-1,b}u^n_{\Omega_1,b} \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_{b-1} \end{pmatrix}. \quad (2.12)$$

Here $u^n_{\Omega_1,1}, \ldots, u^n_{\Omega_1,b-1}$ are unknowns and $u^n_{\Omega_1,b}$ is given by the boundary condition. Using a backward finite-difference discretization of the Robin boundary condition Eq. (2.6) we have

$$(1+p)u^n_{\Omega_1,b} - u^n_{\Omega_1,b-1} = (1+p)u^{n-1}_{\Omega_2,b} - u^{n-1}_{\Omega_2,b-1}. \quad (2.13)$$

We can substitute this into Eq. (2.12),

$$\begin{pmatrix} A_{11} & A_{12} & & 0 \\ A_{21} & \ddots & \ddots & \\ & \ddots & A_{b-2,b-2} & A_{b-2,b-1} \\ 0 & & A_{b-1,b-2} & A_{b-1,b-1} \end{pmatrix} \begin{pmatrix} u^n_{\Omega_1,1} \\ \vdots \\ u^n_{\Omega_1,b-1} \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ +A_{b-1,b}\left(\frac{1}{1+p}u^n_{\Omega_1,b-1} + u^{n-1}_{\Omega_2,b} - \frac{1}{1+p}u^{n-1}_{\Omega_2,b-1}\right) \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_{b-1} \end{pmatrix}. \quad (2.14)$$

The terms involving $u^{n-1}_{\Omega_2}$ are known and can be brought to the right-hand side. The

$u^n_{\Omega_1,b-1}$ term can be included in the subdomain matrix. This yields

$$\begin{pmatrix} A_{11} & A_{12} & & 0 \\ A_{21} & \ddots & \ddots & \\ & \ddots & A_{b-2,b-2} & A_{b-2,b-1} \\ 0 & & A_{b-1,b-2} & A_{b-1,b-1} + \frac{1}{1+p}A_{b-1,b} \end{pmatrix} \begin{pmatrix} u^n_{\Omega_1,1} \\ \vdots \\ u^n_{\Omega_1,b-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_{b-1} - A_{b-1,b}\left(u^{n-1}_{\Omega_2,b} - \frac{1}{1+p}u^{n-1}_{\Omega_2,b-1}\right) \end{pmatrix}. \quad (2.15)$$

For $p \to \infty$ we obtain Dirichlet boundary conditions, the subdomain matrix retains its original form (as a submatrix of $A$), and only the right-hand side $f$ is modified by the boundary condition. For $p = 0$ we obtain pure Neumann boundaries. In that case both the subdomain matrix and the right-hand side are modified, as for the more general Robin boundary condition.

If higher-order terms along the boundary are included, such as in Eq. (2.7), the terms proportional to the submatrix $A_{b-1,b}$ are modified with respect to Eq. (2.15). Since there are substantially different ways to discretize this type of boundary condition, we cover it in detail in the next section.

### 2.2.3 Discretization of boundary conditions with derivatives along the boundary

If there are boundary terms of order one or higher, e.g., as in Eq. (2.7),

$$(\partial_x + p - q\partial_y^2)u_1^n(b,y) = (\partial_x + p - q\partial_y^2)u_2^{n-1}(b,y), \quad (2.16)$$

there are two options for the discretization. We work for now in two dimensions where the boundary is one-dimensional. The generalization is straightforward. We use a backward derivative for $\partial_x$.

The first method is to place the $q$ term at the backward end of the derivative, which thus does not lie on the boundary,[4]

$$\begin{aligned} &u_1^n(b,y) - u_1^n(b-1,y) + pu_1^n(b-1,y) \\ &\qquad - q\big[u_1^n(b-1,y+1) - 2u_1^n(b-1,y) + u_1^n(b-1,y-1)\big] \\ &= u_2^{n-1}(b,y) - u_2^{n-1}(b-1,y) + pu_2^{n-1}(b-1,y) \\ &\qquad - q\big[u_2^{n-1}(b-1,y+1) - 2u_2^{n-1}(b-1,y) + u_2^{n-1}(b-1,y-1)\big]. \end{aligned} \quad (2.17)$$

We used a nearest-neighbor stencil for the second-order derivative in direction $y$. Note that the $q$ terms are placed at the last slice of domain $\Omega_1$, at $b-1$. This equation can trivially be solved for $u_1^n(b,y)$ to be used in the subdomain equation as in Eq. (2.15).

[4]Note that one can choose whether to put the boundary term proportional to $p$ at $b$ or at $b-1$: as we show in Sec. 2.3.1, this simply yields a different parametrization.

The second option places the $q$ terms on the boundary, i.e., on the forward end of the derivative.[5] We are not aware of any mention of this approach in the literature. The discrete boundary condition then reads

$$\begin{aligned} & u_1^n(b,y) - u_1^n(b-1,y) + pu_1^n(b,y) \\ & \qquad - q\big[u_1^n(b,y+1) - 2u_1^n(b,y) + u_1^n(b,y-1)\big] \\ & = u_2^{n-1}(b,y) - u_2^{n-1}(b-1,y) + pu_2^{n-1}(b,y) \\ & \qquad - q\big[u_2^{n-1}(b,y+1) - 2u_2^{n-1}(b,y) + u_2^{n-1}(b,y-1)\big]. \end{aligned} \tag{2.18}$$

Solving this equation for $u_1^n(b,y)$ is not trivial as it was in the first case. Due to the $q$ terms, equations for different $y$ are now coupled. To solve for $u_1^n(b,y)$, we write this equation in matrix form by defining a matrix $B$ of size $n_y \times n_y$, with $n_y$ the number of lattice sites in direction $y$,

$$B = \begin{pmatrix} 1+p+2q & -q & & \\ -q & \ddots & \ddots & \\ & \ddots & \ddots & -q \\ & & -q & 1+p+2q \end{pmatrix}, \tag{2.19}$$

under the assumption that the direction $y$ is not closed, otherwise there are coefficients $-q$ in the top-right and bottom-left corner of $B$. Then the boundary condition reads

$$Bu_1^n(b) - u_1^n(b-1) = Bu_2^{n-1}(b) - u_2^{n-1}(b-1) \tag{2.20a}$$

$$u_1^n(b) = B^{-1}u_1^n(b-1) + u_2^{n-1}(b) - B^{-1}u_2^{n-1}(b-1). \tag{2.20b}$$

That is, for solving for $u_1^n(b)$ we have to invert the matrix $B$, which has the size of the boundary. In the two-dimensional case $B$ is tridiagonal so this can be done cheaply. This second shape of the discrete boundary condition is strongly nonlocal due to the matrix inverse it involves. The consequence is that the subdomain problem is modified with a dense block matrix (even if $B$ is sparse, its inverse is not). If the boundary is large this might be a limiting factor for the algorithm performance.

We consider the Schwarz method for the two-dimensional Poisson equation,

$$(m - \nabla^2)u = f, \tag{2.21}$$

with mass term $m$ on a domain $\Omega$ in order to study some properties of optimized Schwarz methods. We use a nearest-neighbor stencil for the discrete Laplace operator. For a domain $\Omega$ with boundaries (i.e., for the Poisson equation on a square) we always use Dirichlet boundary conditions on the natural boundary (in contrast to the optimized boundary conditions on the artificial ones). We give the iteration count in the $p$-$q$ plane for the implementation without boundary solve (Eq. (2.17)) in Fig. 2.1. The method with boundary solve (Eq. (2.20b)) is given in Fig. 2.2. In both cases there

[5] The generalization to higher orders is done in almost the same way.

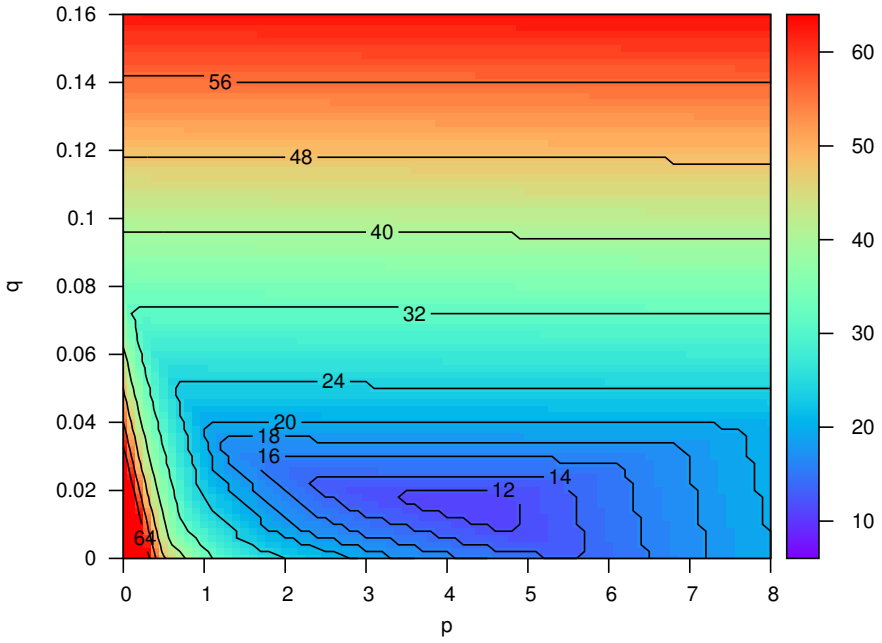



**Figure 2.1:** Schwarz iteration count for solving the Poisson equation (with $m = 1$) in a square region of size $31 \times 31$ with two domains with overlap 2 to a relative residual norm of $10^{-12}$. The boundary condition *without* boundary solve is used.

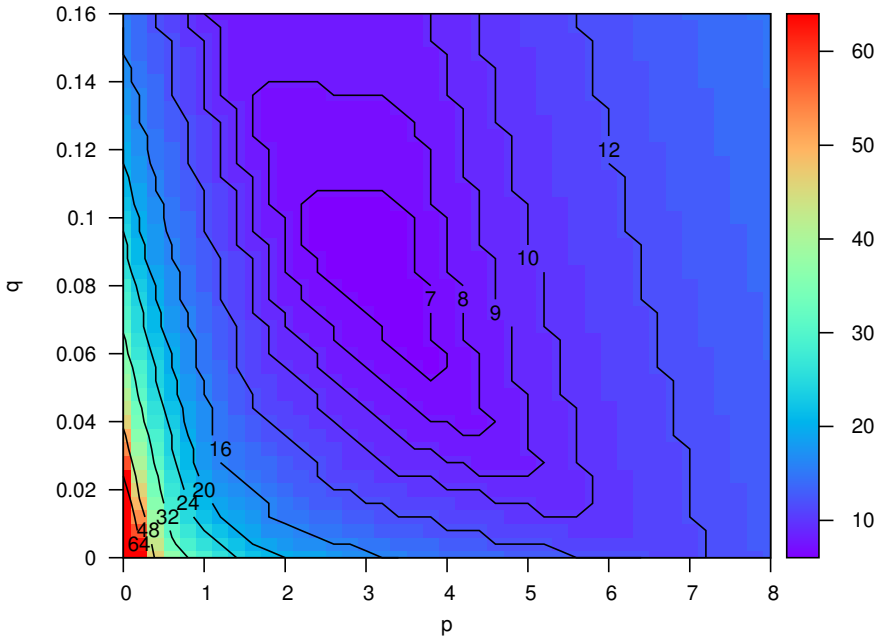



**Figure 2.2:** As Fig. 2.1 but with the boundary condition *with* boundary solve.

is a minimum at $p \neq 0$ and $q \neq 0$. For the method with boundary solve, the minimum is shifted to larger $q$ and the iteration count is everywhere lower than for the method without boundary solve, at the cost of higher computation effort per iteration due to the matrix-inverse in the boundary condition.

### 2.2.4 Topologies

Most publications on optimized Schwarz methods treat only open regions, for QCD however we need a torus. Naively one would not expect drastical changes of the algorithm behavior, but there seem to be subtleties. For one, the convergence proofs based on Fourier transformations in the direction of the boundary as in Ref. [27] fail for a torus — for the convergence factor defined there we obtain $\rho = 1$, i.e., there is no reduction of the error. In general it is unclear whether optimal Schwarz methods exist for domain decompositions with cycles (i.e., decompositions which are not stripes or are closed in the direction perpendicular to the stripes), see Ref. [29] for details. Here, we simply resort to a short numerical study for the Poisson equation with $m = 1$ for various cases for two domains.

#### Square

For a square, i.e., without closed boundaries, the iteration count is given in Fig. 2.1. There is a minimum at $p \neq 0$, $q \neq 0$.

#### Cylinder

For a cylinder the behavior is very similar to that for a square. There is a slight but insignificant difference between closing the cylinder in direction of the boundary or perpendicular to it.

#### Torus

We give the iteration count for a torus in Fig. 2.3. There is a minimum at $p \approx 1$, but $q$ has no effect (too large $q$ destroys the convergence). The explanation is related to the low modes of the problem, which depend on the topology. For a square the lowest eigenvalue is $2\pi^2 + m$ and the lowest mode is proportional to $\cos x \cos y$. Similarly, for a cylinder the lowest eigenvalue is $\pi^2 + m$ with a mode proportional to $\cos x$ or $\cos y$. For a torus, however, the lowest eigenvalue is $m$ and the lowest mode is a constant.[6] In basically all cases the convergence rate is dominated by the lowest mode. Thus we conjecture: $q \neq 0$ cannot help to approximate constant modes, else we would see a $q$ dependence. We can verify this by using a larger mass, $m > \pi^2$. The constant mode will still be the lowest mode, but its relative weight is reduced. Thus a better approximation to the other modes would help the overall convergence speed. Indeed we can then observe a $q$ dependence of the iteration count, with a minimum at $q > 0$.

[6] We used $m = 1$ in all figures given so far.

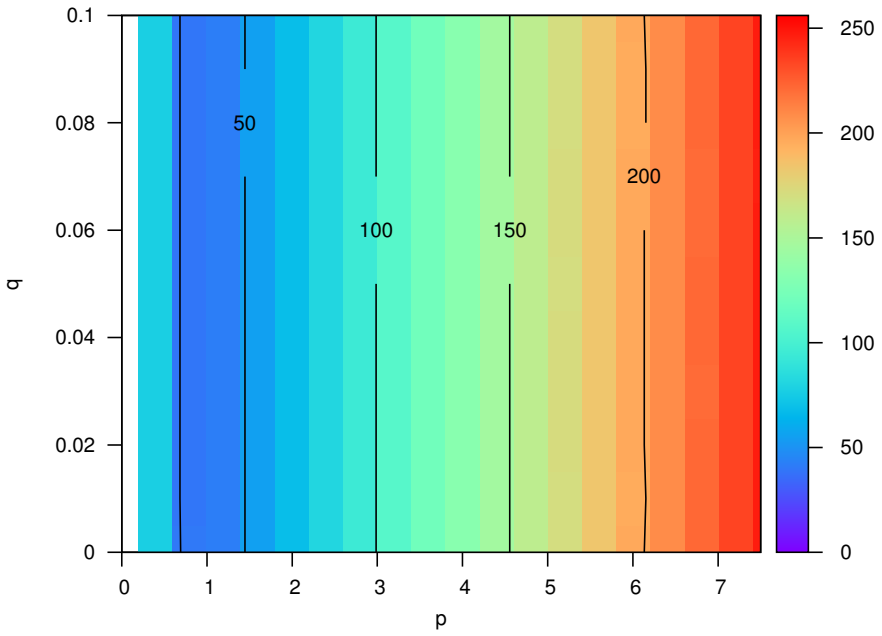



**Figure 2.3:** Schwarz iteration count for solving the Poisson equation ($m = 1$) on a torus of size $32 \times 32$ with two domains with overlap 2 to a relative residual norm of $10^{-12}$.

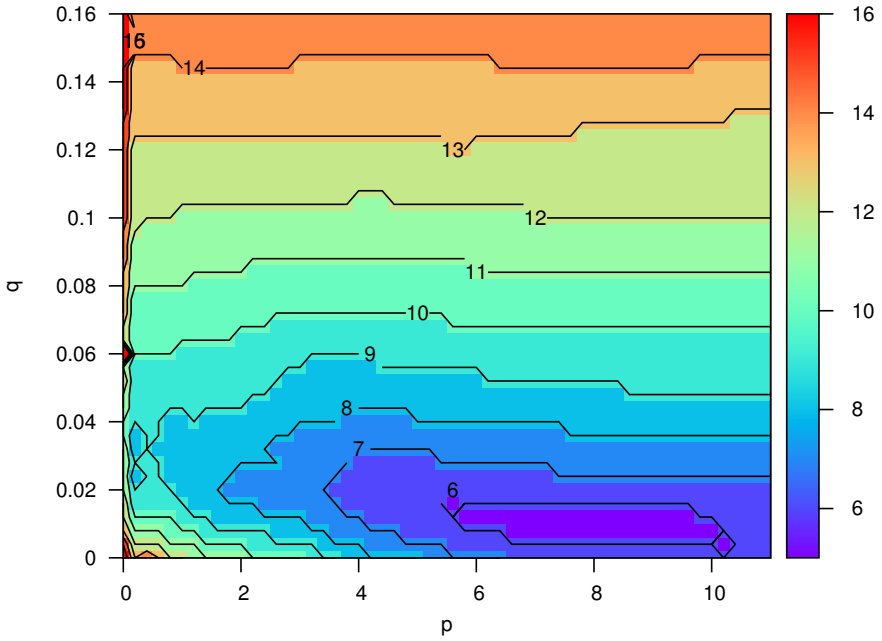



**Figure 2.4:** GCR iteration count, with Schwarz method (2 iterations) as preconditioner, for solving the Poisson equation ($m = 1$) on a torus of size $32 \times 32$ with two domains with overlap 2 to a relative residual norm of $10^{-12}$.

This leads us to conclude that the optimized Schwarz method does not solve problems due to low modes: zeroth-order optimization, which simply corresponds to Robin boundary conditions, gives a positive effect, but higher orders yield no improvement. Thus — as for many other inversion methods — we have to resort to coarse grids or (inexact) deflation methods.

In Fig. 2.4 we show the iteration count if the same Schwarz method is used as a preconditioner for the GCR inverter. We now see a $q$ dependence (even for $m = 1$), but $q \neq 0$ does not yield a significant advantage.

### 2.2.5 Domain splitting in two directions

For QCD we need to generalize the optimized Schwarz method to domain decompositions in more than one direction, as discussed, e.g., in Ref. [28]. For an isotropic problem the boundary conditions should be the same on all edges of a domain. The $\partial_x$ term from Secs. 2.2.2 and 2.2.3 is then a derivative normal to the domain surface. The $\partial_y$ terms lie along the boundary.

The corner points are a special case: since they couple to more than one site in neighboring domains, it can be necessary to adjust the parameters of the boundary conditions for these points to obtain good performance (in four dimensions these "corners" are faces, edges, and vertices). One option is to put $q = 0$ at the corners. This means that domain edges are independent from each other, and a boundary matrix in analogy to Eq. (2.19) would be block-diagonal.

As an example for such a domain decomposition we give the Schwarz iteration count in Fig. 2.5 in the $p$-$q$ plane, where we put $q = 0$ at the corners. As before, there is a minimum where $p \neq 0$ and $q \neq 0$.

### 2.2.6 Iterative inversions

An important practical point that is not mentioned in the papers cited in this chapter is the influence of the cost for solving the subdomain problems. The modified boundary conditions (see Eq. (2.15)) influence the condition number of the subdomain matrices. If we apply an iterative inverter to solve on the domains (which is the common case), this may have a severe effect on the total performance.

In our experience adding only (positive) $p$ improves the condition, since it adds terms to the diagonal of the subdomain matrix. Using $q \neq 0$, which adds off-diagonal terms, often makes the condition number worse. We consider the effect for the $2 \times 2$ domain decomposition of the previous example: in Fig. 2.5 we gave the Schwarz iteration count. The corresponding average iteration count on the subdomain problems is given in Fig. 2.6. In Fig. 2.7 we give the combined iteration count. We see that the strong minimum for $q \neq 0$ of Fig. 2.5 is weakened in the more realistic cost estimation: in the combined iteration count plot, the minimum almost intersects the $p$-axis. Nevertheless, in this case the Robin boundary conditions, i.e., the zeroth-order optimized boundary conditions, are superior to the Dirichlet boundary conditions.

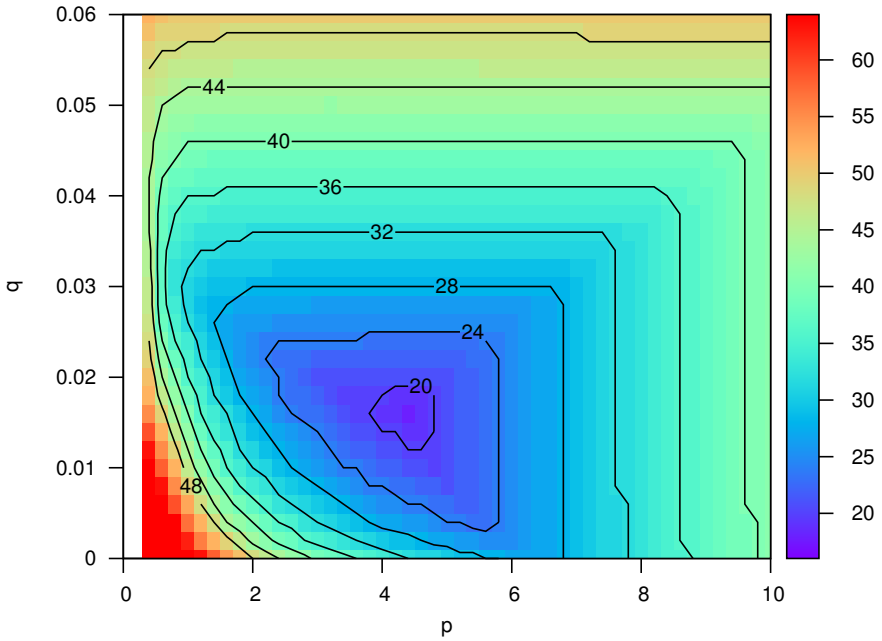



**Figure 2.5:** Schwarz iteration count for solving the Poisson equation ($m = 1$) to a relative residual norm of $10^{-12}$. The problem is on a square of size $63 \times 63$ with four domains of overlap 2, arranged as $2 \times 2$.

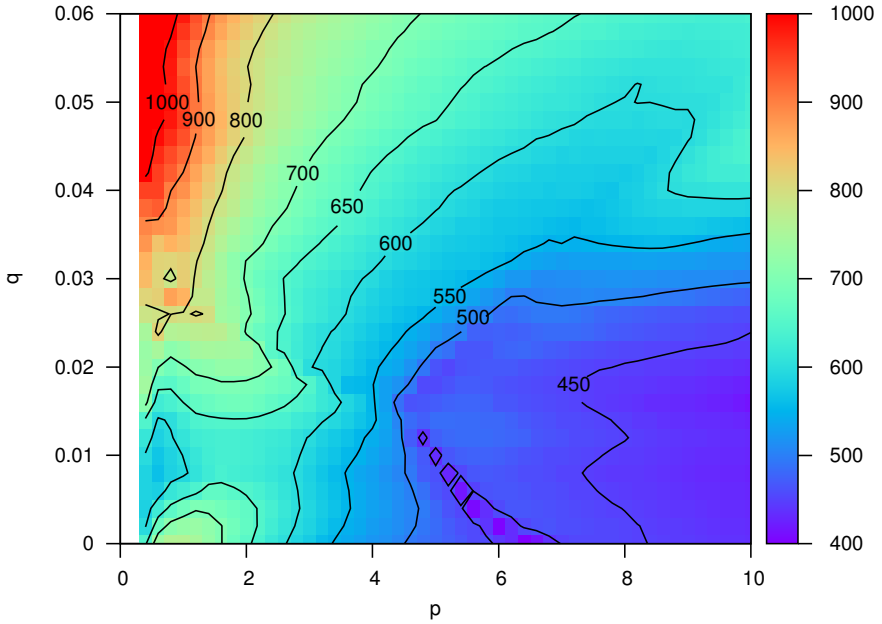



**Figure 2.6:** Average subdomain iteration count for the problem of Fig. 2.5.

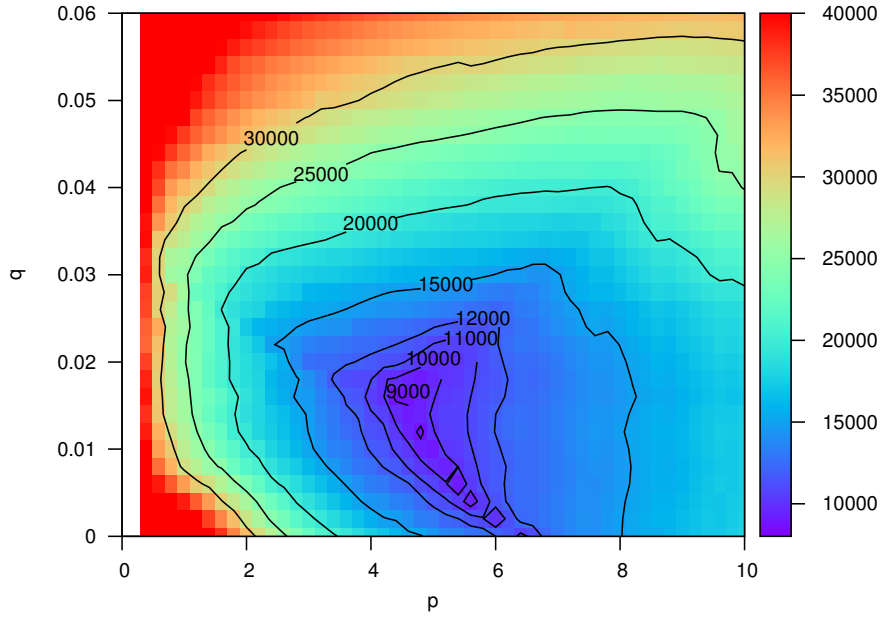



**Figure 2.7:** Combined iteration count for the problem of Fig. 2.5. This data gives a better cost estimate.

## 2.3 Optimized Schwarz methods for QCD

In this section we implement and test the optimized Schwarz methods for QCD. We derive the necessary algorithmic modifications for the zeroth-order optimization in Sec. 2.3.1. The second-order case then follows in analogy to Sec. 2.2.3. The results are analysed in Sec. 2.3.2. Since the zeroth-order method does not give a worthwhile improvement, we did not expect a drastic improvement when adding the second-order. The expectation is confirmed by our results.

### 2.3.1 Boundary conditions for gauge theories

Consider the forward boundary of a domain $\Omega_1$ in direction $\hat{\mu}$, at $x = b$. The neighboring domain is labeled $\Omega_2$. The commonly used Dirichlet boundary condition reads

$$\psi_1^n(b) = \psi_2^{n-1}(b), \tag{2.22}$$

where the subscripts $i = 1, 2$ denote the domain and $n$ is the iteration index. The simplest Robin boundary condition would be

$$(\partial_\mu + p)\psi_1^n(b) = (\partial_\mu + p)\psi_2^{n-1}(b), \tag{2.23}$$

$p \in \mathbb{R}$, but since we are working with a gauge theory, replacing $\partial_\mu$ by the covariant derivative $D_\mu$ seems to be the most natural way,

$$(D_\mu + p)\psi_1^n(b) = (D_\mu + p)\psi_2^{n-1}(b). \tag{2.24}$$

On the lattice $D_\mu$ is replaced by its discrete version [46],

$$\nabla_\mu \psi(x) = U(x,\mu)\psi(x+\hat{\mu}) - \psi(x), \tag{2.25a}$$
$$\nabla^*_\mu \psi(x) = \psi(x) - U^\dagger(x-\hat{\mu},\mu)\psi(x-\hat{\mu}), \tag{2.25b}$$

the forward and backward finite-difference discretization of the gauge-covariant derivative.[7] For the boundary $\partial\Omega_1$ at $b$ the last site *inside* the domain $\Omega_1$ is at $x = b - \hat{\mu}$. Using the backward derivative the boundary condition is[8]

$$(\nabla^*_\mu + p')\psi_1^n(b) = (\nabla^*_\mu + p')\psi_2^{n-1}(b), \tag{2.26}$$

or explicitly

$$\begin{aligned}(1+p')\psi_1^n(b) - U^\dagger(b-\hat{\mu},\mu)\psi_1^n(b-\hat{\mu}) \\ = (1+p')\psi_2^{n-1}(b) - U^\dagger(b-\hat{\mu},\mu)\psi_2^{n-1}(b-\hat{\mu}).\end{aligned} \tag{2.27}$$

The application of the Dirac operator on a subdomain vector reads[9]

$$\begin{aligned}\eta_1^n(x) &= D_{\mathrm{W}}\psi_1^n(x) \\ &= \psi_1^n(x) - \kappa\sum_{\nu=0}^{3}\Big[U(x,\nu)(1-\gamma_\nu)\psi_1^n(x+\hat{\nu}) \\ &\qquad + U^\dagger(x-\hat{\nu},\nu)(1+\gamma_\nu)\psi_1^n(x-\hat{\nu})\Big].\end{aligned} \tag{2.28}$$

[7] $x$ is now a discrete index

[8] We renamed $p$ to $p'$, for reasons that will become clear later.

[9] In our tests we include the Clover improvement, but as this has no influence on these derivations it is omitted here.

If $x$ has nearest neighbors on the boundary, e.g., for $x = b - \hat{\mu}$, we use the boundary condition to eliminate degrees of freedom outside $\Omega_1$,

$$\begin{aligned}
\eta_1^n(b-\hat{\mu}) &= \psi_1^n(b-\hat{\mu}) - \kappa \sum_{\nu=0}^{3} \Big[ U(b-\hat{\mu},\nu)(1-\gamma_\nu)\psi_1^n(b-\hat{\mu}+\hat{\nu}) \\
&\qquad\qquad + U^\dagger(b-\hat{\mu}-\hat{\nu},\nu)(1+\gamma_\nu)\psi_1^n(b-\hat{\mu}-\hat{\nu}) \Big] \\
&= \psi_1^n(b-\hat{\mu}) - \kappa \sum_{\substack{\nu=0 \\ \nu\neq\mu}}^{3} \Big[ \dots \Big] \\
&\quad - \kappa \Big[ U(b-\hat{\mu},\mu)(1-\gamma_\mu)\psi_1^n(b) + U^\dagger(b-2\hat{\mu},\mu)(1+\gamma_\mu)\psi_1^n(b-2\hat{\mu}) \Big] \\
&= \psi_1^n(b-\hat{\mu}) - \kappa \sum_{\substack{\nu=0 \\ \nu\neq\mu}}^{3} \Big[ \dots \Big] \\
&\quad - \kappa \Big[ (1-\gamma_\mu) \Big( U(b-\hat{\mu},\mu)\psi_2^{n-1}(b) - \tfrac{1}{1+p'}\psi_2^{n-1}(b-\hat{\mu}) + \tfrac{1}{1+p'}\psi_1^n(b-\hat{\mu}) \Big) \\
&\qquad + U^\dagger(b-2\hat{\mu},\mu)(1+\gamma_\mu)\psi_1^n(b-2\hat{\mu}) \Big],
\end{aligned} \tag{2.29}$$

where we used Eq. (2.27) in the last step, which gives the new term in the big parentheses. The terms $U(b-\hat{\mu},\mu)\psi_2^{n-1}(b)$ and $\frac{1}{1+p'}\psi_2^{n-1}(b-\hat{\mu})$ are known and can be brought to the other side of the equation. The term $\frac{1}{1+p'}\psi_1^n(b-\hat{\mu})$ gives a modification of the subdomain matrix.

There seems to be an ambiguity of the discretization, since we can choose whether we put the $p$ term at the forward or backward end of the finite-difference derivative. This is only apparently so — we can show that both choices are equivalent. Consider a boundary condition in analogy to Eq. (2.26), where we now put a $p$ term at $b-\hat{\mu}$ and use the forward derivative at $b-\hat{\mu}$ to ensure gauge invariance,[10]

$$(\nabla_\mu + p)\psi_1^n(b-\hat{\mu}) = (\nabla_\mu + p)\psi_2^{n-1}(b-\hat{\mu}), \tag{2.30}$$

or explicitly

$$\begin{aligned}
&U(b-\hat{\mu},\mu)\psi_1^n(b) + (p-1)\psi_1^n(b-\hat{\mu}) \\
&\qquad\qquad = U(b-\hat{\mu},\mu)\psi_2^{n-1}(b) + (p-1)\psi_2^{n-1}(b-\hat{\mu}).
\end{aligned} \tag{2.31}$$

[10] Alternatively one can use a backward derivative as in Eq. (2.26) and add a factor $U^\dagger(b-\hat{\mu},\mu)$ to the $p$ term to ensure gauge invariance.

We can continue the derivation as above and obtain

$$\begin{aligned}\eta_1^n(b-\hat{\mu}) = \psi_1^n(b-\hat{\mu}) - \kappa \sum_{\substack{\nu=0 \\ \nu\neq\mu}}^{3} \Big[\dots\Big] - \kappa\Big[(1-\gamma_\mu)\Big(&(p-1)\psi_2^{n-1}(b-\hat{\mu}) \\ &+ U(b-\hat{\mu},\mu)\psi_2^{n-1}(b) - (p-1)\psi_1^n(b-\hat{\mu})\Big) \\ &+ U^\dagger(b-2\hat{\mu},\mu)(1+\gamma_\mu)\psi_1^n(b-2\hat{\mu})\Big].\end{aligned} \tag{2.32}$$

We see that this is equivalent to the backward discretization for

$$p' = \frac{p}{1-p}. \tag{2.33}$$

From now on we prefer the version with forward discretization, because we can simply put $p = 1$ to obtain a pure Dirichlet boundary condition at $x = b$. For the backward version we would have to use $p' \to \infty$. For $p = 0$ we obtain Neumann boundaries.

Second-order (or higher-order) boundary terms can be derived as in Sec. 2.2.3. As for the Neumann term we choose to replace the standard derivatives by covariant derivatives.

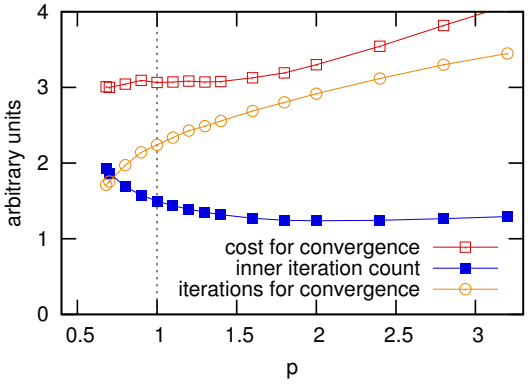
To close this section, let us note that there are some other options for choosing the boundary conditions:

- use non-covariant derivative,
- add terms with spin projections $(1 \pm \gamma_\mu)$, motivated by the shape of the Dirac operator.

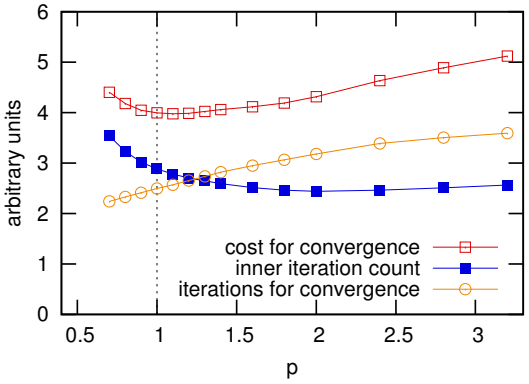
This may break gauge invariance, but since it is part of the preconditioner, it cannot influence the final result. A quick preliminary test showed neither a positive nor a negative influence of these modifications.

### 2.3.2 Results

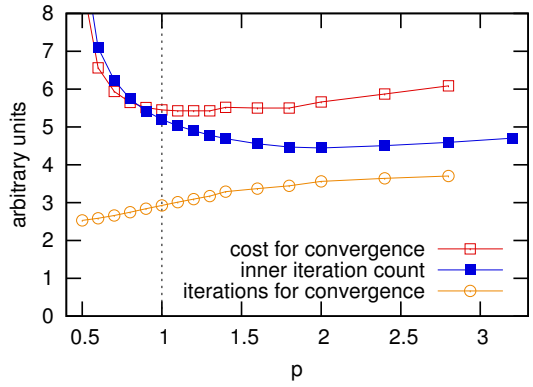
We now discuss our benchmarks of the optimized Schwarz methods for QCD. Throughout this section we use a lattice size of $32^3 \times 64$ with gauge configurations generated at $\kappa = 0.13632$ as in Sec. 1.4, and the multiplicative Schwarz method with 16 colors and domain spread $8 \times 8 \times 4 \times 8$. In Figs. 2.8 and 2.9 we show the convergence behavior of the Schwarz method depending on $p$ for $\varepsilon = 10^{-4}$. Let us consider the iterations needed for convergence to a given relative residual norm. Going from $p = 1$ towards $p = 0$ reduces this iteration count significantly. However, at $p$ around 0.6 the Schwarz method diverges. Choosing $p > 1$ gives a modest increase in the iteration count. As usual we also have to consider the inner iteration count to evaluate the algorithm performance. In this case $p < 1$ gives a strongly rising iteration count, and thus cancels the positive effect on the Schwarz iteration count. The total cost for convergence, approximated as the product of Schwarz iteration count and inner iteration count, has a minimum at or near $p = 1$ in all cases.




**(a)** Overlap 2




**(b)** Overlap 3



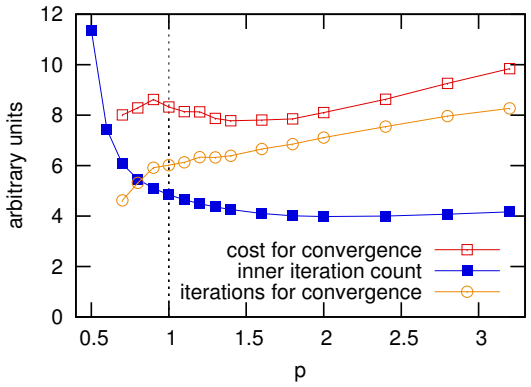

**(c)** Overlap 4

**Figure 2.8:** Convergence behavior of the optimized Schwarz method depending on the boundary parameter $p$ at $\kappa = 0.13632$. $p = 1$ corresponds to the usual Dirichlet boundary condition. We give the Schwarz iteration count for convergence to a fixed residual norm (empty circles, yellow), the inner iteration count (filled squares, blue), and the product of the two, which is a measure for the total cost of convergence (empty squares, red). Values are scaled to lie in the same plot range.

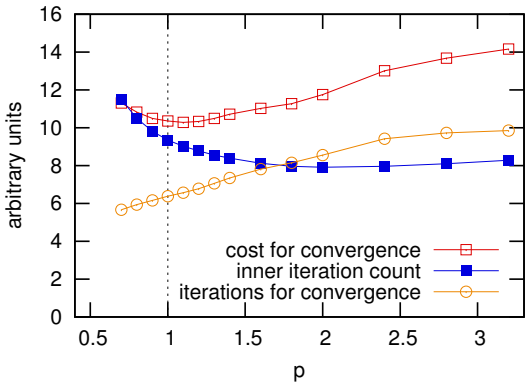
We also tested the optimized Schwarz method for complex-valued boundary parameter, $p \in \mathbb{C}$. The convergence degrades with growing imaginary part and we could not find a parameter range where the optimum is not on the real axis. Therefore we do not consider this case any further.

Next, we use the optimized Schwarz method as a preconditioner for flexible GMRES with deflated restarts (fGMRES-DR). The result is given in Figs. 2.10 and 2.11, where we plot the cost for convergence to a given relative norm of the residual. There are no results for $p \lessapprox 0.6$ because the preconditioner is divergent for this parameter region and keeps fGMRES-DR from converging. We observe a marginal improvement over Dirichlet boundaries for $p < 1$ for the minimal overlap of 2. For more overlap the rising inner iteration count shifts the minimum, in some cases even to $p > 1$, but the overall effect is marginal.

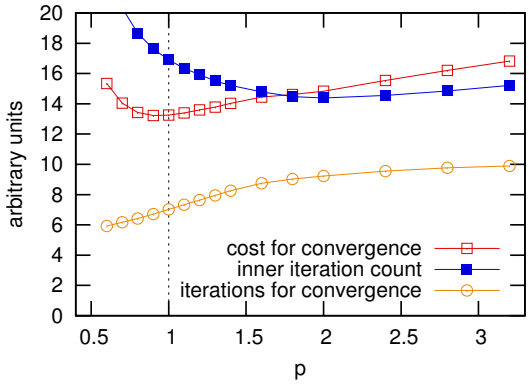
As seen above, for $p$ around 0.6 the Schwarz preconditioner quickly becomes divergent, which is also reflected in the drastically rising iteration count of the subdomain solves, as could be seen in, e.g., Fig. 2.9a. This is due to the modification of the subdomain matrices by the boundary term, given in detail in Eq. (2.32), however it is unclear why boundary conditions with Neumann character are ill-conditioned. One possible explanation is that the gauge field is very noisy, making derivative terms a




**(a)** Overlap 2




**(b)** Overlap 3




**(c)** Overlap 4

**Figure 2.9:** As Fig. 2.8 but at $\kappa = 0.13645$.

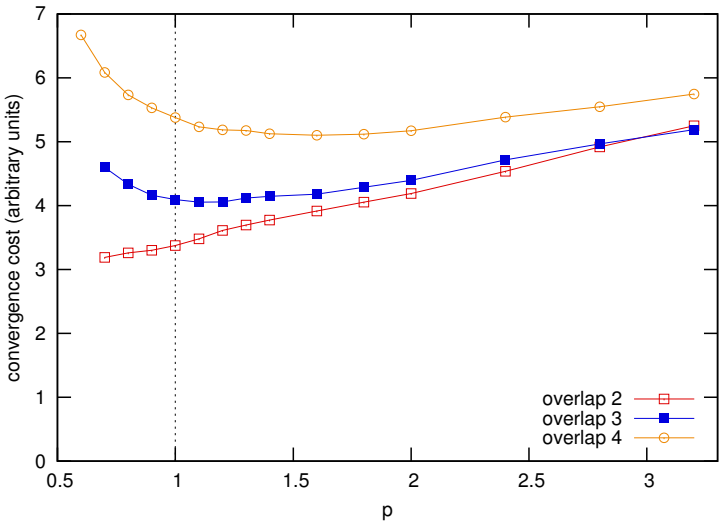



**Figure 2.10:** Approximate convergence cost for fGMRES-DR(24,8) with 4 iterations of the optimized Schwarz preconditioner at $\kappa = 0.13632$. We show the dependence on the boundary condition parameter $p$ and give results for different overlap size.

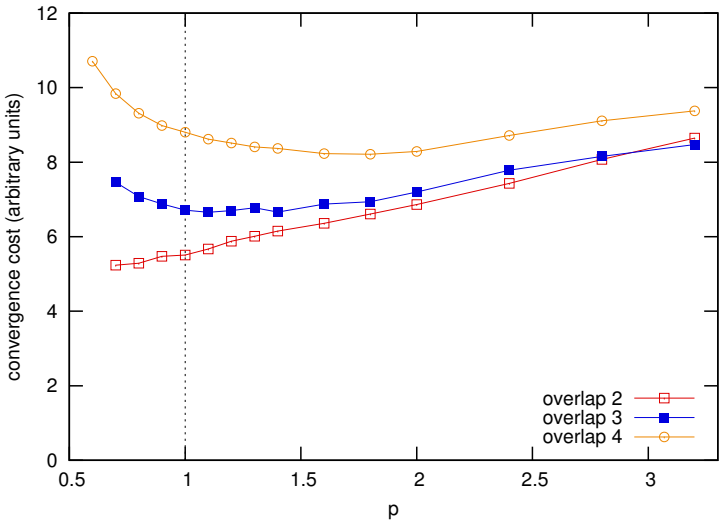



**Figure 2.11:** As Fig. 2.10 but at $\kappa = 0.13645$.

bad choice for the boundary condition. It is plausible that after fixing the gauge to a smooth gauge better results might be obtained [15, 39], but we did not verify this.

In summary we cannot observe a benefit of using zeroth-order optimized boundary conditions for QCD. While we obtained improvements for some random gauge configurations on tiny lattices, the tests presented here for realistic cases do not reproduce these improvements.

For the Poisson equation the gain by adding a second-order term was smaller than for the first-order term (see Sec. 2.2), so we do not expect a significant improvement in the case of QCD. We implemented a second-order version with covariant derivatives and the $q$ term placed on the backward end of the derivative normal to the domain boundary. In Fig. 2.12 we give the number of Schwarz iterations for convergence and the average number of inner iterations.[11] Negative $q$ gives a slight improvement of the Schwarz iteration count, however it can lead to a divergent inner iteration, especially for $p < 1$. A cost estimation based on the product of these two iteration counts is given in Fig. 2.13. There is a small improvement of up to 4% for $p < 1$ and $0 \lessapprox q \lessapprox 0.1$. Given that there is additional algorithmic cost for the second order (which is not taken into account in this plot), this improvement is negligible.

It is however remarkable that this improvement is *not* to due a reduced Schwarz iteration count, but on the contrary due to the combination of an increased Schwarz iteration count with an improved inner iteration count. This turns the approach of optimized Schwarz methods upside down: (in the case of QCD) one should not strive for minimization of the Schwarz iteration count, but instead find parameters which optimize the condition number of the subdomain matrices, without too much negative influence on the Schwarz iteration.

[11] Note that the configuration differs from the one used for the zeroth-order method, so the results cannot be compared exactly.

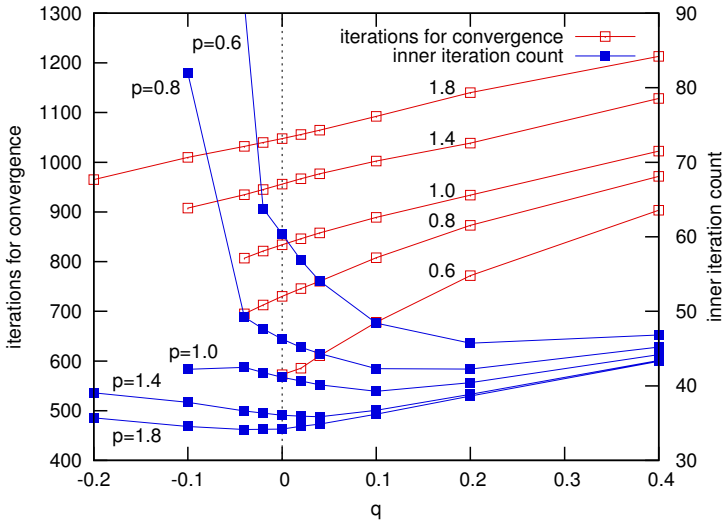



**Figure 2.12:** Convergence behavior of the second-order optimized Schwarz method with overlap $O = 2$ depending on the boundary parameters $p$ and $q$ at $\kappa = 0.13632$. Note that the $y$-axes do not start at 0. The parameter set ($p = 1$, $q = 0$) corresponds to the usual Dirichlet boundary condition. The zeroth-order case is marked by the dotted line at $q = 0$. We give the Schwarz iteration count for convergence to a fixed residual norm (empty squares, red) and the average inner iteration count (filled squares, blue). All curves are cut off at negative $q$ because the iteration diverges below a critical value for $p$ and $q$.

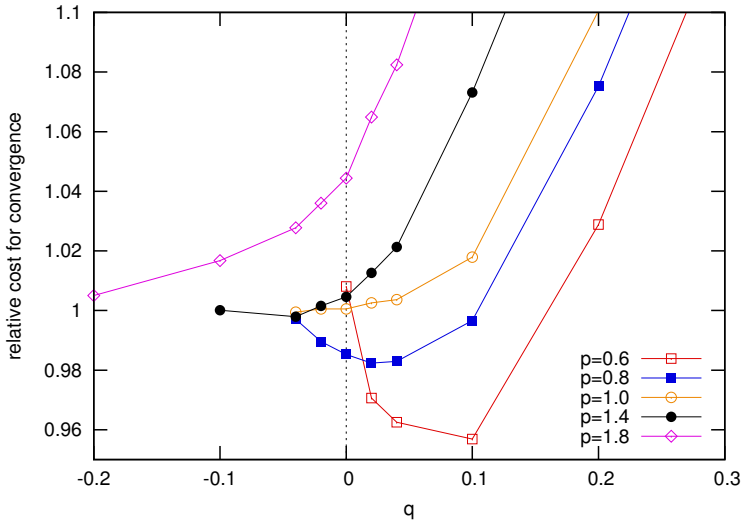



**Figure 2.13:** Product of the Schwarz iteration count and the inner iteration count from Fig. 2.12, relative to Dirichlet boundary conditions ($p = 1$, $q = 0$). This is a measure for the total cost of convergence. Note the small scale of the $y$-axis.

## 2.4 Conclusions

This chapter provided a brief introduction to optimized Schwarz methods, which modify the coupling between subdomains to improve the convergence. We transferred the method to QCD, and performed a numerical analysis for the zeroth-order and second-order optimization. The former corresponds to optimal Robin boundary conditions.

For zeroth-order, we observe improvements of the convergence speed of the preconditioner if we increase the Neumann character of the boundary condition (by choosing $p < 1$). However there are two issues: (1) The method becomes divergent if $p$ is chosen too small, i.e., thorough testing is necessary to guarantee a stable method for any gauge field configurations. (2) The iteration count for solving the subdomain problems increases if $p$ is decreased. This often more than cancels the positive effects of increased Schwarz convergence speed. This important practical result is often not mentioned in the mathematical literature which tends to be interested only in the overall convergence speed of the Schwarz method, not taking into account the cost for solving the subdomain problems.

For second-order the findings are similar. There are small improvements, which are however overcompensated by the increased inner iteration count.

Given these results we conclude that it is not worth to use non-Dirichlet boundary conditions for QCD, unless an improvement is found (see below): while small speedups can be observed for both zeroth-order and second-order, the effect is too small to justify the additional code complexity and the increased overlap (at least $O = 2$), which is necessary to implement derivatives perpendicular to the boundary.

Finally, let us note that the tested version of the second-order method for QCD is not necessarily ideal. There are some options which we plan to investigate in future work: (1) $q$ can be adjusted independently on cells, faces, edges, and corners. (2) A discretization with boundary solve as discussed in Sec. 2.2.3 could be used.

# Chapter 3

# Schur complement inverter for non-overlapping subdomains

## 3.1 Introduction

In this chapter we present a Schur complement algorithm for inverting the Wilson Dirac operator, based on a decomposition of the physical volume into non-overlapping subdomains — a method well known in the partial differential equation (PDE) community.[1] This method is substantially different from the commonly used Schwarz algorithms in QCD and thus it is worth to study whether it can provide a remedy for the issues discussed in the Introduction and for Schwarz methods (Ch. 1 and Ch. 2): slow convergence, stagnation, or divergence due to low modes.

Computation of such a Schur complement involves local inversions, making the algorithm a good candidate for parallel computers. Additionally, the condition number of the Schur complement is typically better than that of the original matrix (see Ref. [81]), but there are no known bounds in the non-Hermitian case which are applicable to the Dirac operator. The better condition number comes at the expense of dealing with the Schur complement, which is a full matrix, instead of the Dirac operator, which is very sparse. Up to now there were no investigations of such Schur complement methods for lattice QCD, and the vast majority of research in the PDE community focuses on two- and three-dimensional problems. This work aims at filling this blank spot, i.e., a domain decomposition based Schur complement method in four dimensions for gauge theories, by introducing a basic algorithm and a preconditioner. As introductory books we can recommend the rather elementary Ref. [72] and the more thorough and mathematical Ref. [50].

We introduce some language required for describing algorithms throughout this chapter. Large-scale linear problems are solved on parallel computers which are built from many boards connected via a network. Typically, each of these boards has memory shared among several CPUs with several (or many) cores each. This heterogeneous structure makes a comprehensive description cumbersome. Therefore we simplify the notation by defining a parallel computer as a set of *nodes* connected via a network. Here a *node* may be one core, one CPU, or all CPUs on a board. This is a choice of the programmer. The only important point is that data transfers within a node should be very fast, i.e., via a shared cache or main memory, and should not involve network

[1] These methods are also known as *iterative substructuring* algorithms.

transfers.[2] *Local variables* denotes a subset of all variables which is stored on a given node of the parallel architecture. *Local operations* denote operations that involve only *local variables*, i.e., they can be done without network communication. *Global operations* involve all nodes of the machine. In particular these are dot-products and norms, which involve all degrees of freedom. *Neighbor communication* involves only a few neighboring nodes.

This chapter is organized as follows. We introduce the Schur complement method in Sec. 3.2 and the Neumann-Neumann preconditioner in Sec. 3.3. Technical details necessary for the implementation are covered in Sec. 3.4. We briefly mention coarse grids in Sec. 3.5 before giving a detailed analysis of our benchmarks and numerical studies in Sec. 3.6. We draw conclusions in Sec. 3.7.

## 3.2 Schur complement

We seek the solution of the linear system

$$Au = f, \tag{3.1}$$

where $f$ is a given source vector, $A$ is a complex square matrix which, e.g., represents a finite-difference discretization of the Dirac operator, and $u$ is the solution vector. Splitting the degrees of freedom into two sets $I$ and $B$ (the names will get a meaning later), we write $A$ in block form,

$$A = \begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix}, \tag{3.2}$$

where $A_{II}$ and $A_{BB}$ represent couplings within a set, while $A_{IB}$ and $A_{BI}$ represent coupling from one set to the other. Then the *Schur complement* (see for example Ref. [81]) of $A_{II}$ is defined as

$$S = A_{BB} - A_{BI}A_{II}^{-1}A_{IB}. \tag{3.3}$$

Using the Schur complement $S$ we can write

$$A^{-1} = \begin{pmatrix} A_{II}^{-1} + A_{II}^{-1}A_{IB}S^{-1}A_{BI}A_{II}^{-1} & -A_{II}^{-1}A_{IB}S^{-1} \\ -S^{-1}A_{BI}A_{II}^{-1} & S^{-1} \end{pmatrix}. \tag{3.4}$$

That is, the inverse of $A$ can be expressed via $A_{II}^{-1}$ and $S^{-1}$. In principle the choice of the two index subsets is arbitrary. Due to the structure of the Wilson Dirac operator,[3] which involves only couplings to nearest neighbors in the four-dimensional lattice, a natural choice is to decompose the physical volume into non-overlapping blocks.[4] Arbitrary geometries are possible, however we restrict our descriptions and

[2] Ethernet, Infiniband, or other.
[3] see Ch. 4.2
[4] Typically one would put one block on one node.

numerical tests to hyper-rectangles of uniform size.[5] The structure can be imagined as a four-dimensional soap foam, where the soap is the interface, containing the boundary degrees of freedom, and the air inside the bubbles corresponds to the interior degrees of freedom of the blocks. Throughout this chapter we label all points $x_\mu$ with

$$\text{index} = \begin{cases} B & \text{if} \quad x_\mu \mod l_\mu = 0 \quad \text{for any} \quad \mu \in 0,1,2,3, \\ I & \text{otherwise,} \end{cases} \tag{3.5}$$

corresponding to points on the boundary and interior points, respectively. $l_\mu$ is the length of a block in direction $\mu$. Note that there are 12 internal degrees of freedom per space-time point (4 spin times 3 color degrees of freedom), which however play no explicit role in our decompositions. Since the block interior is separated from the interior of neighboring blocks by the boundary layer, the operator will contain a block-diagonal part for the interior degrees of freedom. After proper index reordering we obtain

$$A = \begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix} = \begin{pmatrix} A_{I^{(1)}I^{(1)}} & & & A_{I^{(1)}B} \\ & \ddots & & \vdots \\ & & A_{I^{(N)}I^{(N)}} & A_{I^{(N)}B} \\ A_{BI^{(1)}} & \dots & A_{BI^{(N)}} & A_{BB} \end{pmatrix}, \tag{3.6}$$

where the upper index $(i)$, running over all blocks, denotes degrees of freedom in block $i$ of set $I$. The block-diagonal structure of $A_{II}$ makes an inversion of this part cheaper and allows for a *local* computation of $A_{II}^{-1}$ if each sub-block $A_{I^{(i)}I^{(i)}}$ is local on a node of the parallel architecture. Therefore we can efficiently compute the Schur complement $S$ of the block $A_{II}$, given in Eq. (3.3).

As mentioned in the introduction, the Schur complement is a full matrix,[6] which makes a direct computation unfeasible: the interior unknowns are eliminated, so the number of unknowns in the Schur complement system is smaller than in the original system, however,[7] the size of the boundary will still be very large. Therefore, $S$ cannot be assembled but has to be inverted iteratively. From (3.4) we see that the solution vector $u = A^{-1}f$ can be found by computing in a first step

$$u_B = S^{-1}g_B = S^{-1}(f_B - A_{BI}A_{II}^{-1}f_I). \tag{3.7}$$

Then the solution for the interior degrees of freedom is given by

$$u_I = A_{II}^{-1}(f_I - A_{IB}u_B). \tag{3.8}$$

$S$ is inverted iteratively by a Krylov subspace solver. This iteration will be referred to as *outer iteration*. In each outer iteration the Schur complement must be computed, which involves an inversion of $A_{II}$, referred to as *local inversion* or *block inversion*.

[5] The gauge field has no a priori jumps or large discontinuities which would motivate nonuniform blocking.

[6] in contrast to the Wilson Dirac operator, which is sparse

[7] especially in four dimensions

## 3.3 Neumann-Neumann preconditioner for the Schur complement

An iterative inversion of the Schur complement $S$ is expensive, since the computation of $S$ involves inversions. Therefore, we consider employing preconditioners to reduce the number of outer iterations. We focus on the Neumann-Neumann preconditioner, introduced below. For details we refer to Ref. [50], which also serves as a basis for our notation and for most of this section. A preconditioner should approximate the inverse of the operator. To make use of the computing architecture — which is fast for computations on the local node, but slow if (global) communication is involved — we construct a preconditioner built from inversions on the local degrees of freedom.

Denote by $B^{(i)}$ the points on the boundary of block $(i)$ and by $I^{(i)}$ the points in its interior. The submatrix of $S$ corresponding to block $(i)$ is given by

$$S_{B^{(i)}B^{(i)}} = A_{B^{(i)}B^{(i)}} - A_{B^{(i)}I^{(i)}} A^{-1}_{I^{(i)}I^{(i)}} A_{I^{(i)}B^{(i)}} - \sum_{k\neq i} A_{B^{(i)}I^{(k)}} A^{-1}_{I^{(k)}I^{(k)}} A_{I^{(k)}B^{(i)}}, \quad (3.9)$$

where the sum contributes only if block $(k)$ is sharing a surface cell with the block $(i)$, i.e., if it is one of the eight nearest neighbors in four dimensions (assuming an operator with only nearest-neighbor coupling). An inverse of the submatrix would yield an approximation of the inverse of $S$ on the *local* degrees of freedom, which could serve as a preconditioner. The problem is that the computation of this submatrix involves terms which are *not local*, due to the sum over $k$, and additionally involves one inversion for each neighboring block as well (not only for the local block). Generally, a rather radical step is taken by cutting off all contributions outside of the block $(i)$. Roughly speaking, it amounts to dropping the sum over the neighbors and a modification of the boundary terms. The result is labeled *subdomain Schur complement* for block $i$, which is defined as

$$S^{(i)} = A^{(i)}_{BB} - A^{(i)}_{BI} {A^{(i)}_{II}}^{-1} A^{(i)}_{IB}, \quad (3.10)$$

and can be computed locally. A detailed definition and discussion is given in Sec. 3.4. The Neumann-Neumann preconditioner is then defined as

$$M = D \left( \sum_{(i)} R_i^T {S^{(i)}}^{-1} R_i \right) D, \quad (3.11)$$

where $D$ is a diagonal scaling matrix with $D^{-1}_{jj}$ = number of blocks sharing boundary point $j$. This takes into account that the degrees of freedom on the boundary are shared by two or more subdomain Schur complements. $R_i$ is a rectangular matrix, restricting the degrees of freedom to the boundary of block $(i)$.

Optionally, one can go one step further and employ a block-Jacobi method to improve the approximation of the preconditioner to $A$: use $u^0_B = 0$ as starting guess and iterate for $n = 1, \ldots, j$

$$u^n_B \leftarrow u^{n-1}_B + M(\tilde{f}_B - S u^{n-1}_B). \quad (3.12)$$

That is, we can implicitly define the Jacobi preconditioner via $u_B^j = M_{\text{Jacobi}}^{(j)} \tilde{f}_B$. Note that $M_{\text{Jacobi}}^{(1)} = M$. When this iteration is used as preconditioner, $\tilde{f}_B$ is the current iteration vector of the outer inverter, not to be confused with the source vector $f_B$ of the outer inversion. Note that $M_{\text{Jacobi}}^{(j)}$ requires $j$ applications of ${S^{(i)}}^{-1}$ to a vector but only $j-1$ applications of $S$ to a vector, because the starting guess is zero.

## 3.4 Technical details

### 3.4.1 Finite-element methods and subdomain matrices

In the previous section we used the subdomain Schur complement $S^{(i)}$ to define the Neumann-Neumann preconditioner. However, we omitted some important details in its definition, which we discuss here. Schur complement methods are typically formulated for finite-element methods (FEM) — an ample discussion can be found for example in Ch. 3 of Ref. [50]. QCD, however, is formulated as a finite-difference method.[8] For the purpose of the following discussion we have to give a brief introduction to finite-element methods.

We follow the outline given in Sec. 3.1 of Ref. [50]. An overview is given in Fig. 3.1. Consider a system on a domain $\Omega$ without boundaries, e.g., a torus as used in most lattice QCD simulations, described by a linear system

$$\mathcal{A}u(x) = f(x) \quad \text{in } \Omega, \tag{3.13}$$

with some operator $\mathcal{A}$. Suppose we are given a triangulation $\mathcal{T}(\Omega)$ of $\Omega$, with vertices at $x_i$, where $i = 1, \ldots, n$. We can then seek a solution $\hat{u}(x) \approx u(x)$ in the space of piecewise linear functions defined on $\mathcal{T}(\Omega)$. With piecewise linear basis functions $\{\phi_i(x)\}$, where

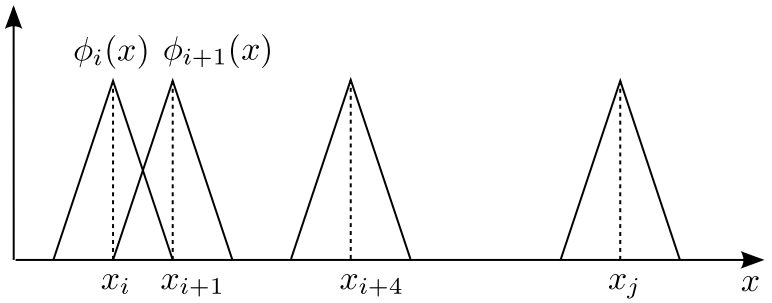
$$\phi_j(x_i) = \delta_{ij}, \tag{3.14}$$

we can represent the solution $\hat{u}(x)$ in this basis,

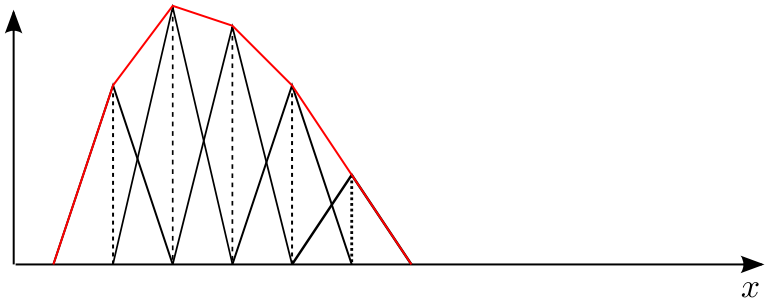
$$\hat{u}(x) = \sum_i \hat{u}(x_i)\phi_i(x). \tag{3.15}$$

The functions $\phi_i(x)$ are equal to one at $x_i$ and go to zero linearly along edges of the triangulation when going to a direct neighbor, i.e., the support of $\phi_i(x)$ is the polygon formed by all direct neighbors of the point $x_i$. They are called finite elements. With these basis functions we can write down a matrix representation of the linear system.

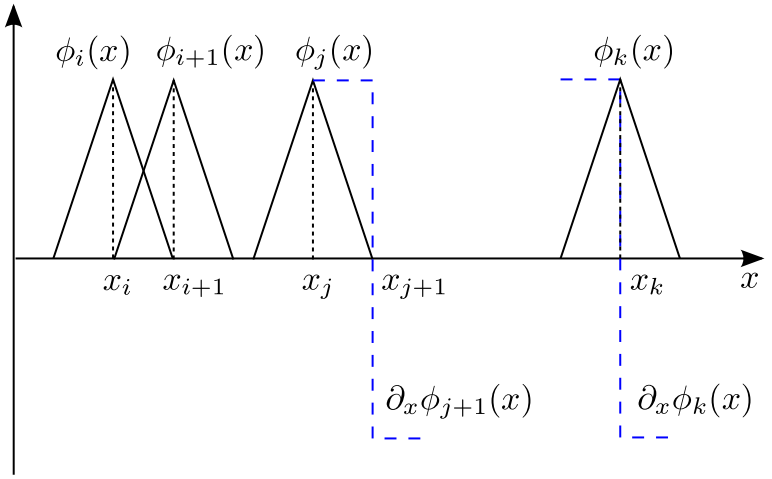
[8] Maintaining gauge invariance of the algorithm basically prohibits a formulation in terms of finite elements, but this is an ongoing research topic, see, e.g., Ref. [10] on a solution for Abelian gauge theories.




**(a)** Piecewise linear basis functions (finite elements). The support of $\phi_i(x)$ is the interval $(x_{i-1}, x_{i+1})$.




**(b)** Piecewise linear function as sum over finite elements




**(c)** Discretization of an operator is found via $A_{ij} = \int_\Omega dx\, \phi_i(x)\mathcal{A}\phi_j(x)$. Consider, e.g., an operator $\mathcal{A} = c + \partial_x$. The constant term yields contributions for $A_{ii}$ and $A_{i,i\pm1}$, since an element overlaps with itself and its nearest neighbor, as can be seen in the sketch, for $\phi_i(x)$ and $\phi_{i+1}(x)$. The derivative of an element yields a step function, plotted dashed (blue). For $A_{j,j+1}$ this yields the area of the right half of the triangle of $\phi_j$. We also see that the contribution of $\partial_x$ to $A_{kk}$ is zero, since the left and right half of the triangle $\phi_k$ cancel. All these statements are valid for any $i$, $j$, and $k$.

**Figure 3.1:** Crash course in finite-element discretization for a one-dimensional problem.

Defining

$$A_{ij} = \int_\Omega dx\, \phi_i(x) \mathcal{A} \phi_j(x), \tag{3.16a}$$

$$f_i = \int_\Omega dx\, \phi_i(x) f(x), \tag{3.16b}$$

$$u_i = \hat{u}(x_i), \tag{3.16c}$$

we obtain a linear system

$$Au = f. \tag{3.17}$$

We partition the system into blocks via non-overlapping subdomains $\Omega_i$, aligned with the triangulation $\mathcal{T}(\Omega)$, with

$$\Omega = \cup_i \overline{\Omega}_i, \tag{3.18a}$$

$$\Omega_i \cap \Omega_j = \emptyset, \quad \text{for } i \neq j. \tag{3.18b}$$

Due to the alignment we have two types of nodes $x_i$, those on the boundary and those in the interior of the subdomains. Collecting these in the indices $I$ for interior points and $B$ for boundary points, the linear system can be written in block form,

$$\begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix} \begin{pmatrix} u_I \\ u_B \end{pmatrix} = \begin{pmatrix} f_I \\ f_B \end{pmatrix}. \tag{3.19}$$

Subdomain matrices $A^{(i)}$ for domain $\Omega_i$ are defined as

$$A^{(i)}_{jk} = \int_{\Omega_i} dx\, \phi_j(x) \mathcal{A} \phi_k(x), \tag{3.20a}$$

$$f^{(i)}_j = \int_{\Omega_i} dx\, \phi_i(x) f(x). \tag{3.20b}$$

In contrast to the definitions in Eq. (3.16), the integration is now only over the domain $\Omega_i$. It is constructed such that

$$A_{jk} = \sum_i A^{(i)}_{jk}, \tag{3.21a}$$

$$f_j = \sum_i f^{(i)}_j. \tag{3.21b}$$

Note that $A^{(i)}$ is *not* a submatrix of $A$. The difference is illustrated in Fig. 3.2. Suppose that the basis functions $\phi_i(x)$ that lie on the boundary are symmetric with respect to the boundary. Then the integration in the definition of the subdomain matrices, Eq. (3.20), covers only a specific fraction of the support of these finite elements. Nearest-neighbor terms $A^{(i)}_{jk}$ are created by $\phi_j$ and $\phi_k$ offset by one lattice spacing in one

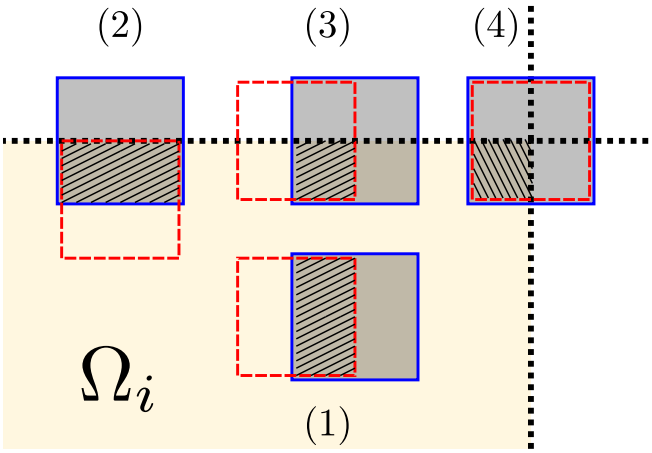



**Figure 3.2:** This sketch shows the domain $\Omega_i$ with three neighboring domains, the boundaries are indicated by the dotted lines. Solid (blue) and dashed (red) squares depict the support of basis functions $\phi_n$. In practice their shape will be more complex, but for the purpose of this illustration this simplification is sufficient. (1) shows two neighboring squares inside $\Omega_i$, the overlapping area is shaded. If one of the squares is centered on the boundary the overlapping area is the same, as shown in (2). In (3) both squares lie on the boundary, then the overlapping area *inside* $\Omega_i$ is reduced to 1/2 of the original overlap, assuming the basis functions are symmetric with respect to the boundary. Generalizing to three dimensions the squares are replaced by cubes. In (4) we show two such cubes lying on the edge of $\Omega_i$ sticking out of the drawing plane. Since they are offset by half of the edge length in this direction, the overlapping volume is reduced to 1/4 of the original overlap of two cubes completely inside $\Omega_i$.

direction. Such a term describes a *link* between these two neighboring points. A hypercubic subdomain[9] in four dimensions has surface *cells* (three-dimensional), *faces* (two-dimensional), *edges* (one-dimensional), and *vertices* (zero-dimensional). The number of subdomains sharing a link is dictated by the finite element lying on the object[10] of highest dimension. For links that do not lie inside the boundary[11], the matrix elements are unchanged. For both elements on a cell (or one on a cell and one on a face), the link is shared between two subdomains, so the integration area is reduced to 1/2. Since we assumed the elements to be symmetric with respect to the boundary, the corresponding matrix element is also reduced to 1/2. Similarly, for both elements on a face (or one on a face and one on an edge), it is reduced to 1/4. For both element on an edge (or one on an edge and one on a vertex), it is reduced to 1/8. Since a vertex is zero-dimensional there cannot be two neighboring elements on it, so there is no corresponding matrix element.

As for $A$, we can split the indices of $A^{(i)}$ into those in $\Omega_i$ labeled $I$, and those in

[9] not necessarily with the same length in each dimension
[10] interior, cell, face, edge, vertex
[11] That is, not both ends are in $\partial\Omega_i$.

$\partial\Omega_i$ labeled $B$, yielding

$$A^{(i)} = \begin{pmatrix} A_{II}^{(i)} & A_{IB}^{(i)} \\ A_{BI}^{(i)} & A_{BB}^{(i)} \end{pmatrix}. \tag{3.22}$$

It should be noted that $A_{BB}^{(i)}$ *differs* from the submatrix $A_{B^{(i)}B^{(i)}}$ of $A$ — the latter submatrix does not have the scaled boundary terms. The other submatrices of $A^{(i)}$ are unchanged with respect to submatrices of $A$: we have $A_{II}^{(i)} = A_{I^{(i)}I^{(i)}}$, $A_{IB}^{(i)} = A_{I^{(i)}B^{(i)}}$, and $A_{BI}^{(i)} = A_{B^{(i)}I^{(i)}}$. This is used to define the *subdomain Schur complement*,

$$S^{(i)} = A_{BB}^{(i)} - A_{BI}^{(i)} {A_{II}^{(i)}}^{-1} A_{IB}^{(i)}, \tag{3.23}$$

which we used in the definition of the Neumann-Neumann preconditioner in Sec. 3.3.

The action of ${S^{(i)}}^{-1}$ does not require $S^{(i)}$ explicitly. Instead it can be calculated via the relation

$${S^{(i)}}^{-1} v_B^{(i)} = \begin{pmatrix} 0 & I \end{pmatrix} {A^{(i)}}^{-1} \begin{pmatrix} 0 \\ I \end{pmatrix} v_B^{(i)}, \tag{3.24}$$

for a given vector $v_B^{(i)}$, zero matrix 0, and unit matrix $I$ of appropriate size. Note that ${A^{(i)}}^{-1}$ can be computed locally, after an initial data exchange with neighbors.

The motive for this discussion of the finite-element method is the implementation of the Schur complement algorithm for QCD or gauge theories in general. These theories are discretized with finite-difference methods, so we have to adapt the definition of the algorithms. We cannot use finite elements for discretization, because this usually breaks gauge invariance. Nevertheless we define the subdomain matrices from Eq. (3.20) in the same way — with scaled terms for links that lie in the boundary. In this case we cannot argue with the reduced area of integration, but we can still think of a gauge link $U_{x,\mu}$ as being shared among two or more subdomains. Numerical experiments confirmed this — without the scaling term of the boundary links the algorithm is still convergent in some cases, but in general the convergence rate is much worse.

### 3.4.2 Data layout

Having defined the algorithm, we now have to map it to the nodes of a parallel computing architecture. The simplest option — applied for many algorithms — is to cut the lattice into hypercubic blocks, and assign one block to each node. Sites on one node are stored linearly in memory, using, e.g., $x$ as fastest index and $t$ as slowest index. This layout is not expedient for the Schur complement algorithm. We still cut the volume into hypercubic blocks, but the ordering of sites will be modified, and we have sites which are (temporarily) stored on more than one node. There are three requirements which lead to a different storing order, illustrated also in Fig. 3.3:

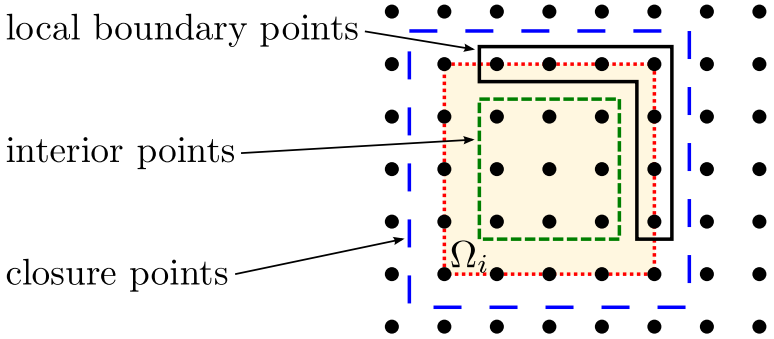



**Figure 3.3:** Sketch of a subdomain with indication of some subsets of lattice sites. The subdomain $\Omega_i$ is shaded, its boundary $\partial\Omega_i$ is marked with a dotted (red) line. Interior sites are framed with short dashes (green). The boundary sites which are assigned to node $(i)$ are framed with a solid (black) line. Sites in the closure $\overline{\Omega}_i$, i.e., in $\Omega_i$ or $\partial\Omega_i$ are framed with long dashes (blue).

1. Inversion of $A_{II}^{(i)}$, which is part of the Schur complement and is also needed for eliminating the interior degrees of freedom and constructing the solution for the interior degrees of freedom.
2. The outer inverter (for inverting $S$) deals only with the degrees of freedom on the boundary. Locally this involves all points which are forward neighbors of the local interior block.[12]
3. Inversion of $S^{(i)}$, which corresponds to inversion of $A^{(i)}$. This involves all local interior points and all points on the boundary of the domain $\Omega_i$, i.e., not only those in forward direction.

Each of these has optimal performance if the involved sites are stored without gaps, which also simplifies the implementation of the respective inverters. A memory layout has to match these requirements. The one we use is based on three contiguous blocks, given in Table 3.1. It first stores the local interior sites, then the local (forward) subset of the boundary, and finally (copies) of the remaining boundary sites (which are local for neighboring nodes). The requirements listed above are fulfilled, one by one:

1. Inversion of $A_{II}^{(i)}$ works on the first block of memory.
2. Inversion of $S$ uses the second block of memory.
3. For inversion of $A^{(i)}$, the last block of memory is filled with copies of the second block from the corresponding neighbors. Then it uses all three blocks of sites in the memory layout.

Thus all three inverters can work on a single vector without gaps. However, since we have split interior and boundary sites, some hopping terms in $A^{(i)}$ require access to

[12] This is a choice. All backward neighbors, or combinations of forward and backward neighbors are also possible. This has no influence on the convergence and is merely relevant for the implementation.

| memory offset | site type | subtype |
|---|---|---|
| 0 | interior sites | – |
| $V_I$ | forward boundary sites | 4 forward cells<br>6 forward faces<br>4 forward edges<br>1 forward vertex |
| $V_I + V_B$ | remaining boundary sites | 4 remaining cells<br>18 remaining faces<br>28 remaining edges<br>15 remaining vertices |

**Table 3.1:** Sketch of the memory layout on one node. Here $V_I$ is the volume of the local interior block and $V_B$ is the volume of the local part of the boundary. The forward and remaining boundary sites are each stored contiguously but separately. The sites within each boundary block were reordered, such that cells, faces, edges, and vertices are each contiguous in memory, which simplifies implementation of communication with the various neighbors.

memory in different blocks. For example, for the coupling of a site on the surface layer of the interior with a point on the forward boundary we need to access the first and second block. This cannot be avoided, but a similar problem also occurs with any other algorithm for links which are not in the direction of the fastest lattice index.

### 3.4.3 Communication

Just like other inverters, the Schur complement method requires communication with nearest neighbors when applying the operator, and some global communication for dot-products. However, if we consider the local subset of boundary sites and the sites involved in $A^{(i)}$, we see that the latter involves points which are neither stored locally, nor on the nearest neighbors: for example, in Fig. 3.3 the bottom left site on $\partial\Omega_i$ is contained neither in the local boundary points of the left neighbor of $\Omega_i$ nor in those of the bottom neighbor. That is, the communication is not restricted to nearest neighbors — it also involves diagonal neighbors, space-diagonal neighbors, and hyperspace-diagonal neighbors (apart from global communication for dot-products). This is a drawback of the Schur complement algorithm, which complicates implementation. There will also be some additional overhead, however the amount of data exchanged with non-nearest neighbors is rather small, so the total impact on the algorithm performance can be expected to be small.

Both the Schur complement and the Neumann-Neumann preconditioner contain the inverse of *local* submatrices. These inversions are typically the most costly part of the algorithm, i.e., large parts of the algorithm are not hampered by network latency and bandwidth limitations. This crucial locality, which is also present in many other domain-decomposition methods, was one of the reasons for studying Schur complement methods.

### 3.4.4 Algorithms

Since the Schur method with Neumann-Neumann preconditioner consists of many parts, we give schematic algorithm listings for clarity. Alg. 3.4.1 gives the global algorithm structure, for solving Eq. (3.7) with a preconditioned Krylov subspace method to obtain the solution on the subdomain boundaries, and for using Eq. (3.8) to find the solution for the interior degrees of freedom. The Krylov inverter needs the application of the Schur complement from Eq. (3.3) to an arbitrary vector, which is given in Alg. 3.4.2. Additionally the Krylov inverter also applies the preconditioner from Eq. (3.11) or Eq. (3.12) to arbitrary vectors. The corresponding algorithm is given in Alg. 3.4.3.

**Algorithm 3.4.1** Structure of inversions via the Schur complement

**Require:** vectors $f_B$, $f_I$, i.e., $f$

1: compute $g_B = f_B - D_{BI} D_{II}^{-1} f_I$ (iteratively invert $D_{II}$)
2: use Krylov inverter to compute boundary solution $u_B = S^{-1} g_B$, using $M^{(j)}_{\text{Jacobi}}$ as preconditioner
3: compute interior solution $u_I = D_{II}^{-1}(f_I - D_{IB} u_B)$ (iteratively invert $D_{II}$)
4: **return** vectors $u_B$ and $u_I$, i.e., $u$

**Algorithm 3.4.2** Applying the Schur complement

**Require:** vector $v_B$

1: compute right-hand side $A_{IB} v_B$ for interior inversion
2: iteratively invert $A_{II}^{-1}$ on each node with Krylov inverter, yielding $A_{II}^{-1} A_{IB} v_B$
3: compute/assemble $w_B = (A_{BB} - A_{BI} A_{II}^{-1} A_{IB}) v_B$
4: **return** vector $w_B$

**Algorithm 3.4.3** Applying the Jacobi preconditioner $M^{(j)}_{\text{Jacobi}}$

**Require:** right-hand side $\tilde{f}_B$

1: put $u_B^0 = 0$
2: **for** $n = 1$ **to** $j$ **do**
3: compute $D(\tilde{f}_B - S u_B^{n-1})$
4: apply $R_i$ to it, which amounts to distributing the d.o.f. in $D(\tilde{f}_B - S u_B^{n-1})$ to all nodes that share these d.o.f., to obtain a vector $v_i$ on each node
5: compute $v_i' = {S^{(i)}}^{-1} v_i$ on each node, using a Krylov inverter for $A^{(i)}$.
6: $D(\sum_i R_i^T v_i')$: average the contributions from nodes in overlapping regions
7: compute $u_B^n$ by adding the new contribution to $u_B^{n-1}$
8: **end for**
9: **return** vector $u_B^j$

## 3.5 Coarse grids and deflation

Just as for direct Krylov inverters or for the Schwarz method, the convergence speed of the Schur complement inverter can be significantly improved by deflating the problem or by adding a coarse grid. Whether this is cost-effective or not depends on several factors (multiple right-hand sides, computing architecture), and is outside the scope of this thesis. We briefly mention and discuss three options.

1. Projecting out global low modes, e.g., as in Ch. 4, works in this case as well and can provide a moderate speedup, provided that low modes are available. As discussed in Refs. [47] and [48] the increasing number of low modes when going to larger volumes can make this method too expensive, however. But even if the number of low modes can still be managed, finding the low modes is typically more expensive than a single inversion, so this is only an option for problems with many right-hand sides.
2. A remedy for the rising number of low modes is the *inexact deflation*, introduced for lattice QCD in Ref. [47] for the Schwarz method. The method uses an approximation to the space of low modes and locally projects each domain on this space. The resulting coarse-grid formulation of the problem is used as a preconditioner for the full problem. Leaving aside the cost of finding the approximation to the space of low modes, this method works very well. We suppose that similar results can be obtained for the Schur complement inverter. However, as we will see that the Schwarz methods perform better than the Schur method, and as we have no reason to assume that inexact deflation gives a bigger advantage to the Schur method than it does to the Schwarz method, we have not tested this (in view of the considerable implementation effort).
3. We have tested a piecewise-constant coarse-grid correction, based on projecting each block to one point. Combined with the Neumann-Neumann preconditioner, this method is known as *balancing* Neumann-Neumann preconditioner, described, e.g., in Refs. [72] and [50]. The additional algorithmic cost is significant but the achieved speedup is not. The reason is that piecewise constant functions are not a good approximation for the low modes of the Dirac operator. One therefore has to resort to methods that explicitly involve information about the low modes, like the inexact deflation mentioned above.

## 3.6 Numerical results

### 3.6.1 Implementation and simulation details

For our tests we use the Wilson Dirac operator $D_{\mathrm{W}}$ with clover term.[13] The Clover term was first introduced in Ref. [68], for further details and references see Ref. [38]. The implementation uses MPI for parallelization. Crucial parts of the code use hand-optimized SSE code. Even-odd preconditioning is not used on the outer level, since it

[13] In our notation we keep $A$, i.e., $A = D_{\mathrm{W}}$.

makes the handling of the block structure very cumbersome. Instead, we use even-odd preconditioning for inversions of the local blocks. Since this part of the computation takes most of the time, we expect an improvement similar to that obtained by even-odd preconditioning for a complete inversion. We use double precision throughout the computation. Typically more than 80% of the floating-point operations are part of the preconditioner, where single precision can be used without affecting the overall accuracy of the algorithm. When using single precision one can expect a speed improvement of about a factor of two.

As in Sec. 1.4, we base our benchmarks on the $N_f = 2$ configurations generated by the QCDSF Collaboration [2], and use the same test setup as for the Schwarz methods. In particular we work with valence quark masses different from sea quark masses, with $\kappa_{\text{val}} = 0.13632$, 0.13640, and 0.13645 for the $32^3 \times 64$ configurations at $\beta = 5.29$.

We introduce the notation used in this section. Denote by $o$ the iteration count of the outer inverter and by $j$ the Jacobi iteration count of the preconditioner. Application of $S$ to a vector requires applying the inverse of $A_{II}$, which we do iteratively to precision $\varepsilon_{A_{II}^{-1}}$. Here, precision denotes the requested relative residual norm of the Krylov inverter — we use BiCGstab in our code. The corresponding BiCGstab iteration count is $n_{A_{II}}$. Typically $\varepsilon_{A_{II}^{-1}}$ should be close to machine precision, or at least significantly lower than the target precision of the outer inversion. We usually work with $\varepsilon_{A_{II}^{-1}} = 10^{-14}$. In the preconditioner as in Eq. (3.12) we also apply $S$ (for $j > 1$), but there the precision can be lower. We denote the corresponding precision as $\varepsilon^{\text{prec.}}_{A_{II}^{-1}}$ and the iteration count as $n^{\text{prec.}}_{A_{II}}$. Finally, we also need ${S^{(i)}}^{-1}$ in the preconditioner, with precision $\varepsilon_{{S^{(i)}}^{-1}}$. We compute it as in Eq. (3.24) via $A^{(i)}$ and use the corresponding residual norm of the $A^{(i)}$ inversion to define the precision. Thus the actual error for ${S^{(i)}}^{-1}$ is slightly lower, since it involves only the degrees of freedom on the boundary of block $(i)$, while $A^{(i)}$ includes the interior as well. The corresponding iteration count is denoted by $n_{A^{(i)}}$. For both types of block inversions in the preconditioner we use BiCGstab. We use the notation $\text{NNp}(j, \varepsilon_{{S^{(i)}}^{-1}}, \varepsilon^{\text{prec.}}_{A_{II}^{-1}})$ to refer to the Neumann-Neumann preconditioner with the parameters given in parentheses. We denote by *block size* the size of the local volume. The size of the interior block is then one less than the block size in each dimension.

For a fixed physical setting ($\kappa$, $\beta$, and the lattice volume), the performance of the algorithm depends on several tunable parameters. We discuss these in detail in Sec. 3.6.2 to Sec. 3.6.5. In Sec. 3.6.6 we comment on an improvement using algorithms for multiple right-hand sides and in Sec. 3.6.7 we propose to use a lower $\kappa$ in the preconditioner. A study for a larger set of configurations is given in Sec. 3.6.8 and a comparison with the Schwarz methods in Sec. 3.6.9.

### 3.6.2 Parameter tuning part 1: Block volume and geometry

The volume and geometry of the local blocks (respectively their interior) has a large influence on the algorithm performance. The volume of the blocks is bounded from

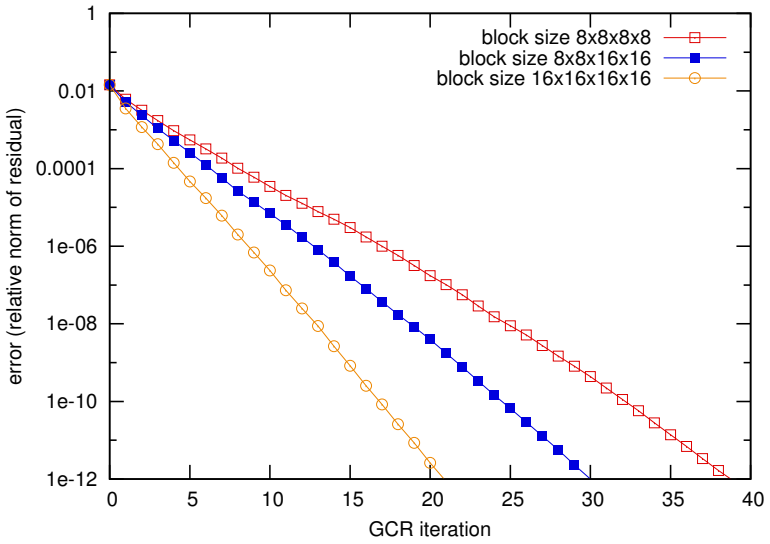



**Figure 3.4:** Convergence depending on block size for a $32^3 \times 64$ lattice at $\kappa = 0.13632$. We use GCR with the Neumann-Neumann preconditioner with 16 iterations, $\varepsilon_{S^{(i)-1}} = 10^{-1}$, and $\varepsilon^{\text{prec.}}_{A_{II}^{-1}} = 10^{-2}$.

above by the number of nodes in the machine, unless one block is shared among several nodes, which is unfavorable due to the loss of locality of block inversions. Smaller block volumes increase the outer iteration count, because the condition number of the preconditioned operator deteriorates. A proof for this statement does not exist for the Dirac operator, however it is intuitively clear or can be inferred from the behavior of many other operators. The example of a $32^3 \times 64$ lattice at $\kappa = 0.13632$, given in Fig. 3.4 (with the GCR algorithm as outer inverter), confirms the expectations. Increasing the block side lengths by a factor of two decreases the outer convergence speed by a factor of almost two.

As a very relevant side effect of the block size, we have to consider the iteration count for computing $A_{II}^{-1}$ and ${S^{(i)}}^{-1}$. Naively we can argue that the condition number is determined by the lowest Fourier mode fitting into the local block, i.e., the blocking cuts off the low part of the spectrum. The modes of the Dirac operator are far from being Fourier modes, but the general behavior is very similar. Our experiments confirm a strong dependence on the block volume. We also refer the reader to Ref. [45] on Schwarz methods. There it is argued that the domain boundaries provide an infrared cutoff and therefore the domains should be smaller than roughly 1 fm, to avoid contributions of low modes. The Schwarz algorithm is quite different from the Schur complement algorithm, however the block inversions of $A_{II}$ are identical, so we can expect to have the same limitation in our case. We give an example for the same configuration as above in Table 3.2. As discussed above, larger block sizes yield a convergence of GCR in fewer iterations $o$. However, the iteration counts $n_{A_{II}}$ and $n^{\text{prec.}}_{A_{II}}$ increase drastically. The increase of $n_{A^{(i)}}$ is weaker — the only possible explanation

| block size | $j$ | $o$ | $n_{A_{II}}$ | $n_{A_{II}}^{\text{prec.}}$ | $n_{A^{(i)}}$ | Dirac operator hop |
|---|---|---|---|---|---|---|
| $8 \times 8 \times 8 \times 8$ | 16 | 40 | 58.5 | 10.8 | 17.5 | 333k [0.937] |
| $8 \times 8 \times 16 \times 16$ | 16 | 31 | 85.3 | 14.4 | 19.2 | 295k [0.902] |
| $16 \times 16 \times 16 \times 16$ | 16 | 22 | 165.5 | 25.2 | 22.3 | 297k [0.831] |

**Table 3.2:** Iteration count and number of computed Dirac operator hopping terms per site, depending on the block size, for convergence of GCR with NNp(16,$10^{-1}$,$10^{-2}$) to an error of $10^{-12}$. The numbers in square brackets give the fraction of these operations that are part of the preconditioner.

is the modified boundary term. As a rough measure for the total resulting cost of the algorithm we give the count of computed hopping terms of the Dirac operator per site for convergence. In all cases it is about $3 \cdot 10^5$, so the decreased outer iteration count is approximately canceled by the increased iteration count of the inner inversions. We can conclude that in the studied range of block sizes we can choose it according to the available machine size, without significant negative impact.[14]

For the block geometry one can choose between the extremes of hypercubic blocks and oblate blocks (where the block length in one or more directions is very small, so the blocks are "thin"). Hypercubic blocks yield the best surface to volume ratio, minimizing the amount of communication, and yield the best convergence of the outer inverter, as shown in Fig. 3.5. The reason is probably that using many thin blocks along one direction leads to slower exchange of information between the blocks — the preconditioner takes longer to propagate information among the blocks and thus loses some of its efficiency. However, the convergence of the block inversions can be significantly better for flat blocks. The reason is the improved condition number for blocks with small extent in one or more directions.

We give an example of the spectrum of the block interior in Fig. 3.6, where, e.g., the top pane shows almost cubic blocks (interior size $19 \times 9 \times 9 \times 15$), "flat" blocks (one direction with small extent, $19 \times 19 \times 19 \times 3$), and "long" blocks (two directions with small extent, $40 \times 40 \times 4 \times 3$). The high part of the spectrum is almost independent of the block shape. The low part shows a large (relative) change away from zero when going to one or even two directions with small extent. For some Krylov inverters the bounds for convergence are given in terms of the condition number (e.g., conjugate gradient on the normal equations), for others the absolute values of the largest and smallest eigenvalues are important, e.g., for GMRES [42]. In this case this would mean an improvement by more than a factor of two for the flat blocks and more than a factor of three for the long blocks.

A comparison of the three panes shows that one small block side length $l_\mu$ is more effective than decreasing the overall volume for near-cubic blocks: from top to bottom the volume is decreased by about a factor of four, which yields a factor of two improvement of the magnitude of the smallest eigenvalue for near-cubic blocks (empty squares, red). But a simple change of the block geometry yields a larger change: in the top

[14] The optimal values for other parameters will however depend on the chosen block size.

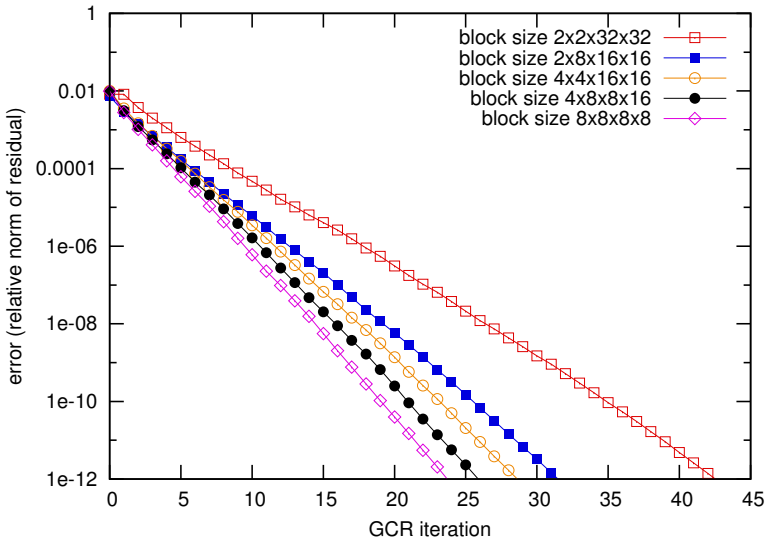



**Figure 3.5:** Convergence depending on block aspect ratio for a $32^3 \times 64$ lattice at $\kappa = 0.13632$. We use GCR with NNp(32, 0.4, $10^{-2}$).

pane, going from these near-cubic blocks to flat blocks (filled squares, blue), improves the magnitude almost by a factor of three. A cutoff in one direction seems sufficient as an overall infrared cutoff which removes eigenvalues with small absolute value, so we may speculate that the structures responsible for low modes are four-dimensional. If they had only three or less dimensions, some of them would be aligned with our blocks, making the cutoff inefficient. We would then see some low modes, which are not removed by the blocks. It is obvious from the plot that such significant outliers do not exist in our case. Note also that for the blocks with one or two thin directions, the other directions have a rather big extent — sometimes bigger than the limit of 1 fm given earlier.[15]

The resulting decrease in the iteration count for the block inversions is displayed in Table 3.3 for NNp(32,0.4,$10^{-2}$). Along with the *increase* of the GCR iteration count $o$ we see a considerable *decrease* of the block iteration count. The effect is largest for the inversion of $A_{II}$ (for both, $S$ in the outer inverter and $S$ in the preconditioner). The inversion of $A^{(i)}$ profits less from the asymmetric blocks, which can only be due to the modified boundary. A more detailed and complete overview can be found in Table 3.4.

### 3.6.3 Parameter tuning part 2: Choice of the Krylov inverter for outer inversion

The choice of the algorithm for the outer inversion is crucial. The decisive factor is the preconditioning we use. The Neumann-Neumann preconditioner $M$ contains

[15] The lattice spacing is 0.072, so for a block length of 19 sites we have 1.37 fm, 40 sites correspond to 2.88 fm.

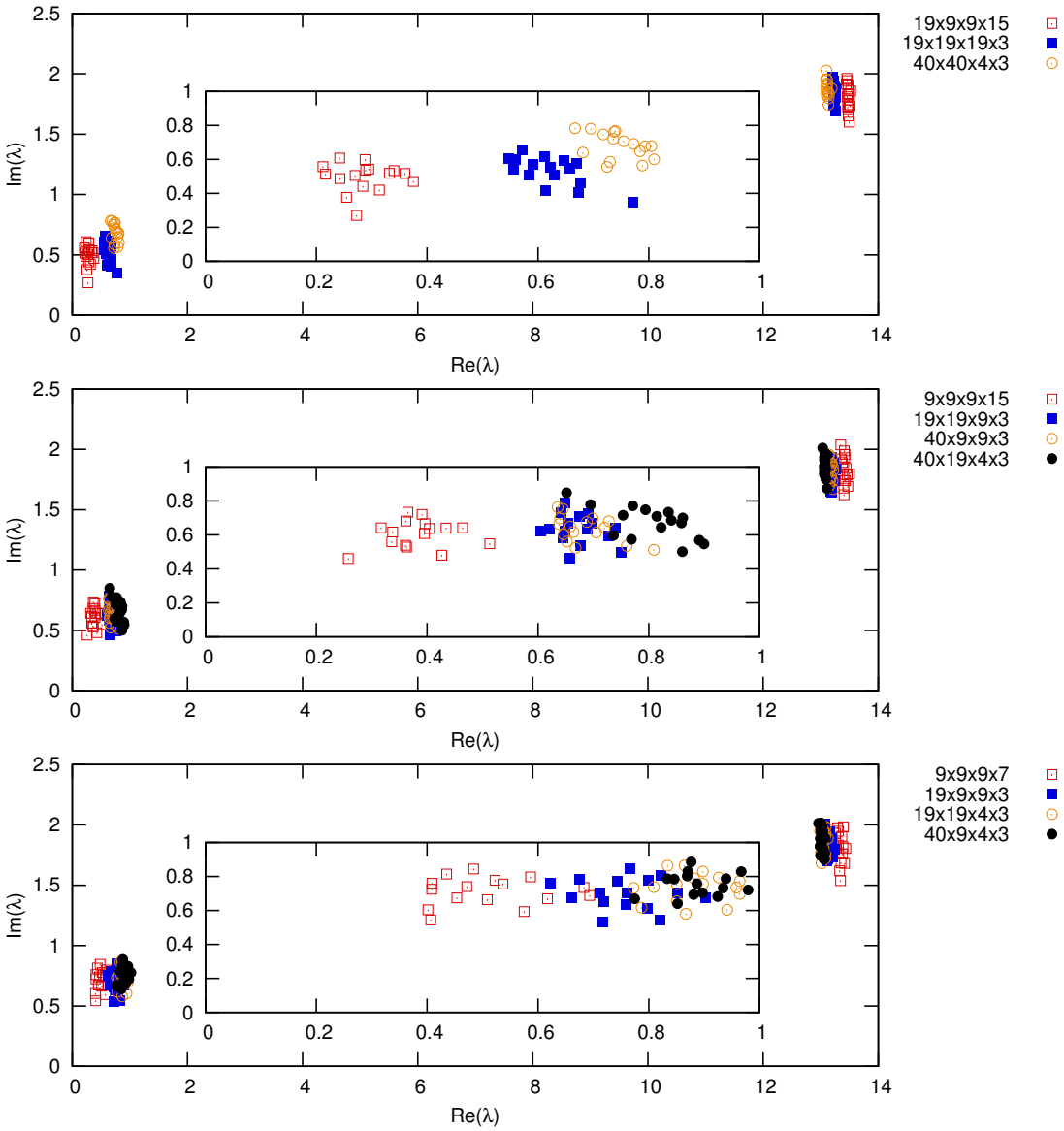



**Figure 3.6:** Spectra of one interior block of a $40^3 \times 64$ lattice with $\kappa = 0.13632$ with large (top), medium (center), and small (bottom) block sizes. The low part is magnified for clarity in the inset part of the plots. The spectra of all other blocks are qualitatively identical. The interior block side lengths are odd in many cases because a local block contains one layer of boundary sites: the side length $l_\mu$ of the block size must be a factor of the lattice side length $L_\mu$, so the interior block side length is $L_\mu/l_\mu - 1$. If the interior block length is equal to $L_\mu$ there is no blocking in direction $\mu$. Only the largest and smallest eigenvalues were computed.

| block size | $j$ | $o$ | $n_{A_{II}}$ | $n^{\text{prec.}}_{A_{II}}$ | $n^{\text{prec.}}_{A^{(i)}}$ | Dirac operator hop |
|---|---|---|---|---|---|---|
| $8 \times 8 \times 8 \times 8$ | 32 | 25 | 59.6 | 10.5 | 8.3 | 240k [0.941] |
| $4 \times 8 \times 8 \times 16$ | 32 | 27 | 45.6 | 7.6 | 7.1 | 214k [0.952] |
| $4 \times 4 \times 16 \times 16$ | 32 | 30 | 37.5 | 6.1 | 6.7 | 216k [0.962] |
| $2 \times 8 \times 16 \times 16$ | 32 | 33 | 24.4 | 4.0 | 6.3 | 212k [0.981] |
| $2 \times 2 \times 32 \times 32$ | 32 | 44 | 17.0 | 3.0 | 5.8 | 284k [0.993] |

**Table 3.3:** Iteration count and number of computed Dirac operator hopping terms per site, depending on the block aspect, for convergence of GCR with NNp(32,0.4,$10^{-2}$) to an error of $10^{-12}$, for a $32^3 \times 64$ configuration at $\kappa = 0.13632$. The numbers in square brackets give the fraction of these operations that are part of the preconditioner.

iterative and inexact matrix inversions. It therefore depends on the right-hand side, which depends on the iteration $k$, so implicitly $M$ depends on $k$. We emphasize this in our notation by writing $M_k$ instead of $M$. This variable preconditioning requires modifications of the inversion algorithms. For details we refer to the literature on *flexible* Krylov methods (also known as *variable* or *non-stationary* preconditioning). Here we give a short introduction, following Ref. [70]. We omit many details which depend on the specific algorithm since we are only interested in the general motivation. Given a linear system of $n$ equations

$$Ax = b, \tag{3.25}$$

with an initial guess $x_0$ and the corresponding initial residual $r_0 = b - Ax_0$, the Krylov subspace of dimension $m$ is

$$\mathcal{K}_m(A, r_0) = \text{span}(r_0, Ar_0, \ldots, A^{m-1}r_0), \tag{3.26}$$

defined by $A$ and $r_0$. The solution $x$ to the linear system can then be approximated as $x_m \in x_0 + \mathcal{K}_m(A, r_0)$. This can be found via an orthonormal basis $\{v_1, \ldots, v_m\}$ of $\mathcal{K}_m(A, r_0)$, which we construct iteratively by computing $Av_k$ and orthonormalizing it with respect to the previous vectors $v_1, \ldots, v_k$. This gives the famous Arnoldi relation

$$v_{k+1}h_{k+1,k} = Av_k - \sum_{j=1}^{k} v_j h_{jk}, \tag{3.27}$$

which can be written in matrix form as

$$\begin{aligned} AV_m &= V_{m+1}H_{m+1,m} \\ &= V_m H_m + h_{m+1,m} v_{m+1} e_m^T, \end{aligned} \tag{3.28}$$

where $V_m = (v_1, \ldots, v_m)$, $H_{m+1,m}$ is the upper Hessenberg matrix containing the coefficients $h_{jk}$, and $H_m$ contains the first $m$ rows of $H_{m+1,m}$. The importance of this equation becomes clear when considering, e.g., the GMRES algorithm, which finds $x_m$ based on a minimal-residual condition

$$\|r_m\| = \min_{x \in x_0 + \mathcal{K}_m} \|b - Ax\|. \tag{3.29}$$

| | | | | | | Wilson Clover operator | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| block size | $j$ | $o$ | $n_{A_{II}}$ | $n^{\text{prec.}}_{A_{II}}$ | $n^{\text{prec.}}_{A^{(i)}}$ | diagonal | hop | axpy | local dotp | dotp | comm. vol. |
| $2 \times 2 \times 32 \times 32$ | 8 | 177 | 17.0 | 3.0 | 5.6 | 44k [0.963] | 282k [0.975] | 123k [0.916] | 83k [0.963] | 4160 | 4316 [0.915] |
| $2 \times 2 \times 32 \times 32$ | 16 | 88 | 17.1 | 3.0 | 5.5 | 43k [0.980] | 276k [0.987] | 117k [0.955] | 81k [0.980] | 2068 | 4292 [0.957] |
| $2 \times 2 \times 32 \times 32$ | 32 | 44 | 17.0 | 3.0 | 5.8 | 44k [0.990] | 284k [0.993] | 118k [0.978] | 83k [0.990] | 1034 | 4292 [0.978] |
| $2 \times 8 \times 16 \times 16$ | 8 | 128 | 24.5 | 4.0 | 6.1 | 31k [0.918] | 210k [0.932] | 89k [0.878] | 60k [0.916] | 3008 | 2542 [0.924] |
| $2 \times 8 \times 16 \times 16$ | 16 | 63 | 24.4 | 4.0 | 6.3 | 30k [0.957] | 205k [0.964] | 84k [0.935] | 58k [0.956] | 1480 | 2503 [0.962] |
| $2 \times 8 \times 16 \times 16$ | 32 | 33 | 24.4 | 4.0 | 6.3 | 31k [0.977] | 212k [0.981] | 86k [0.965] | 60k [0.976] | 776 | 2622 [0.981] |
| $4 \times 4 \times 16 \times 16$ | 8 | 107 | 37.6 | 6.1 | 6.8 | 31k [0.865] | 211k [0.873] | 87k [0.834] | 59k [0.862] | 2514 | 1748 [0.923] |
| $4 \times 4 \times 16 \times 16$ | 16 | 52 | 37.6 | 6.1 | 6.9 | 29k [0.926] | 197k [0.931] | 80k [0.908] | 55k [0.924] | 1222 | 1699 [0.961] |
| $4 \times 4 \times 16 \times 16$ | 32 | 30 | 37.5 | 6.1 | 6.7 | 31k [0.959] | 216k [0.962] | 86k [0.949] | 60k [0.958] | 705 | 1961 [0.980] |
| $4 \times 8 \times 8 \times 16$ | 8 | 95 | 45.9 | 7.7 | 6.9 | 29k [0.833] | 203k [0.839] | 83k [0.804] | 57k [0.828] | 2232 | 1417 [0.924] |
| $4 \times 8 \times 8 \times 16$ | 16 | 47 | 46.2 | 7.7 | 6.9 | 27k [0.906] | 190k [0.910] | 76k [0.889] | 53k [0.904] | 1104 | 1402 [0.961] |
| $4 \times 8 \times 8 \times 16$ | 32 | 27 | 45.6 | 7.6 | 7.1 | 31k [0.949] | 214k [0.952] | 85k [0.940] | 59k [0.948] | 634 | 1611 [0.980] |
| $8 \times 8 \times 8 \times 8$ | 2 | 436 | 59.0 | 11.3 | 8.5 | 61k [0.504] | 425k [0.512] | 183k [0.470] | 120k [0.499] | 10246 | 1457 [0.700] |
| $8 \times 8 \times 8 \times 8$ | 4 | 184 | 59.2 | 11.0 | 8.3 | 41k [0.682] | 286k [0.688] | 119k [0.651] | 80k [0.677] | 4324 | 1230 [0.850] |
| $8 \times 8 \times 8 \times 8$ | 8 | 87 | 58.7 | 10.8 | 8.2 | 33k [0.812] | 234k [0.816] | 95k [0.789] | 65k [0.808] | 2044 | 1163 [0.924] |
| $8 \times 8 \times 8 \times 8$ | 16 | 43 | 59.2 | 10.8 | 9.0 | 32k [0.898] | 227k [0.901] | 92k [0.886] | 63k [0.896] | 1010 | 1150 [0.962] |
| $8 \times 8 \times 8 \times 8$ | 32 | 25 | 59.6 | 10.5 | 8.3 | 34k [0.940] | 240k [0.941] | 96k [0.932] | 66k [0.938] | 588 | 1337 [0.981] |

**Table 3.4:** Parameters and results for various block aspect ratios for a $32^3 \times 64$ configuration at $\kappa = 0.13632$, for convergence of GCR with NNp($j$,0.4,$10^{-2}$) to an error of $10^{-12}$. All listed decompositions use 512 blocks. Crucial parts of the algorithmic cost are given to allow for a comparison: the Wilson Clover Dirac operator, vector-vector operations (axpy, local and global dot-products), and communication volume. The quoted numbers are per site.

Assuming without loss of generality $x_0 = 0$, i.e., $r_0 = b$, the solution of this equation can be expressed via the orthonormal basis of the Krylov subspace, written as

$$x_m = V_m y_m. \tag{3.30}$$

We use the Arnoldi relation Eq. (3.28) and $V_{m+1}e_1 = v_1 = b/\beta$, where $\beta = \|b\|$. Then

$$\begin{aligned} r_m &= b - Ax_m = b - AV_m y_m \\ &= \beta v_1 - V_{m+1} H_{m+1,m} y_m \\ &= V_{m+1}(\beta e_1 - H_{m+1,m} y_m). \end{aligned} \tag{3.31}$$

Since $V_{m+1}$ has orthonormal columns, the least-squares problem can be rewritten as

$$\|r_m\| = \min_{y \in \mathbb{R}^m} \|\beta e_1 - H_{m+1,m} y_m\|, \tag{3.32}$$

i.e, the Arnoldi relation allows to replace the original minimization problem — with vectors of length $n$ — by a problem with vectors of length $m$, where typically $m \ll n$.

We now introduce a preconditioner to illustrate the arising problem. We rewrite the original system of equations as

$$AMu = b, \tag{3.33a}$$

$$x = Mu. \tag{3.33b}$$

We seek a solution $u_m$ in the space

$$\mathcal{K}_m(AM, b) = \operatorname{span}(b, (AM)b, \ldots, (AM)^{m-1}b), \tag{3.34}$$

and write

$$x_m = Mu_m = MV_m y_m. \tag{3.35}$$

If the preconditioner is implemented in an inexact way, e.g., due to a low-precision iterative inversion, it is not a fixed matrix, but depends on the right-hand side, i.e., we are dealing with terms of the form $M_k v_k$. The equation for constructing an orthonormal basis becomes

$$v_{k+1} h_{k+1,k} = AM_k v_k - \sum_{j=1}^{k} v_j h_{jk}. \tag{3.36}$$

We can try to continue naively, as in Eq. (3.31), using Eq. (3.35) for $x_m$,

$$r_m = b - Ax_m = b - AMV_m y_m, \tag{3.37}$$

but fail immediately, because we cannot use the Eq. (3.36) to replace $AMV_m$ by $V_{m+1}H_{m+1,m}$, as we did earlier.[16] The reason is that Eq. (3.36) contains $M_k$ in place

[16] For a fixed preconditioner this *is* possible, and everything follows as in the case without preconditioner.

of $M$ and obviously $MV_m \neq (M_1 v_1, \ldots, M_m v_m)$. However, it is possible to write down a *modified* Arnoldi relation. Defining $z_k = M_k v_k$ we obtain from Eq. (3.36)

$$AZ_m = V_{m+1} H_{m+1,m}. \tag{3.38}$$

We then write the solution as[17]

$$x_m = Z_m y_m, \tag{3.39}$$

and can derive the minimal residual condition similar to Eq. (3.31),

$$\begin{aligned} r_m &= b - Ax_m = b - AZ_m y_m \\ &= \beta v_1 - V_{m+1} H_{m+1,m} y_m \\ &= V_{m+1}(\beta e_1 - H_{m+1,m} y_m). \end{aligned} \tag{3.40}$$

The bottom-line is that any Krylov inverter has to be modified for use with variable preconditioning. Along with some modifications of the algorithm this means that not only $V_m$ but also $Z_m$ has to be stored.

We consider three options for the outer inversion algorithm.

**Flexible BiCGstab** [78] has the advantage of short recurrences without restarts. Due to the short recurrence a non-stationary preconditioner with low-precision inversions leads to a quick loss of orthogonality in the generated Krylov space. As a consequence the algorithm converges slowly or not at all. At the cost of increasing the precision of the inversions in the preconditioner we can achieve good convergence of the outer BiCGstab, but this proved to be inefficient in terms of overall computation time.

**GCR** [77, 21] works with a flexible preconditioner without modification (even without a preconditioner two sets of vectors have to be stored). It enforces an orthonormal basis of the Krylov space by explicit orthonormalization of all basis vectors (long recurrence) and we can use a low precision in the preconditioner. The overall convergence can be slow such that CPU time and memory requirements due to the long recurrence force us to restart the iteration.[18] The resulting loss of orthogonality at a restart often has a severe negative impact on the convergence. As a remedy we can either increase the work in the preconditioner (see Sec. 3.6.4) or use deflated restarts (see next item).

**Flexible GMRES with deflated restarts** [25] (fGMRES-DR) circumvents the information loss at the restart. At the end of a cycle, before the restart, a set of

[17] Note that the range of $Z_m$ is in general not a Krylov subspace (in contrast to the range of $V_m$), see section 10 of Ref. [70] for details.

[18] At a restart the basis vectors are dropped and the algorithm is restarted, using the residual obtained before the restart as a starting guess. Since the old basis was dropped, orthogonality cannot be maintained. This is especially severe for low modes which are hard to find: all the effort GCR spent to find them is lost at a restart.

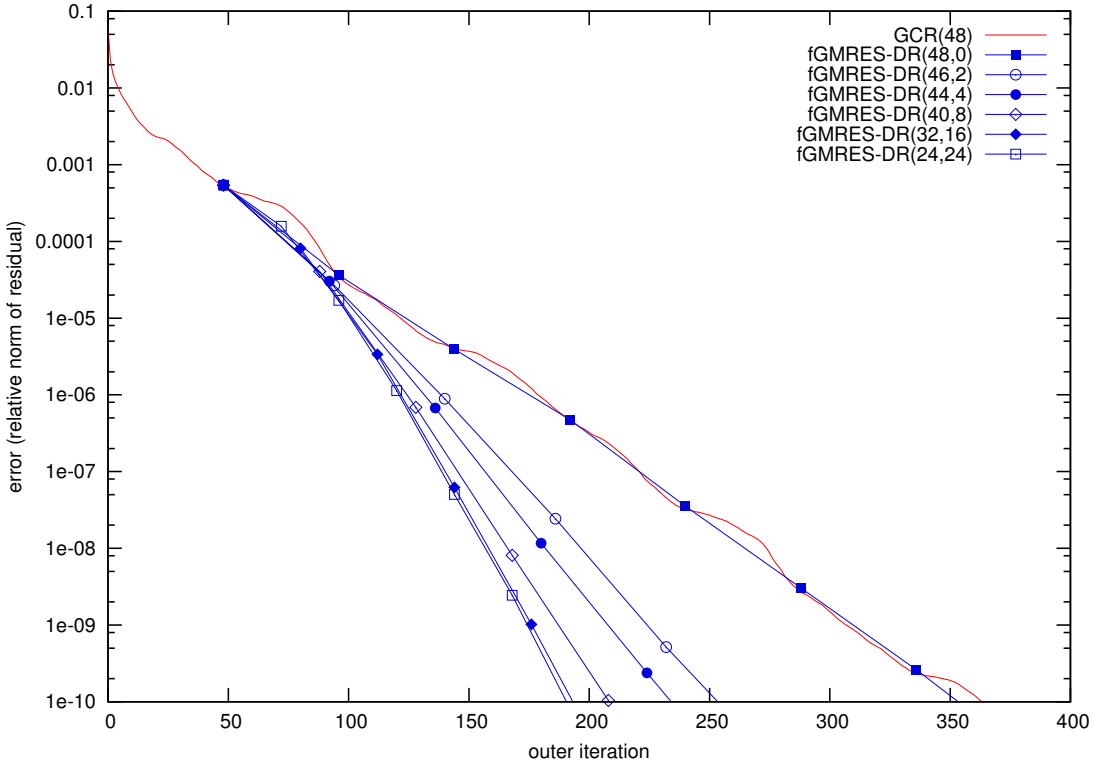



**Figure 3.7:** Convergence of GCR and flexible GMRES with deflated restarts (fGMRES-DR) for a $32^3 \times 64$ configuration at $\kappa_{\text{sea}} = 0.13632$ with $\kappa_{\text{val}} = 0.13645$. We use the Neumann-Neumann preconditioner with $8^4$ blocks and parameters NNp(4, $10^{-1}$, $10^{-2}$). For GCR($m$) and fGMRES-DR($m$,$k$) $m$ gives the number of new vectors generated before a restart and $k$ gives the size of the deflation space.

Krylov vectors corresponding to the smallest Ritz values is computed and taken to the next cycle. This can efficiently alleviate the negative effect of the restart. The disadvantage of the deflated restarts is an increased cost, mainly due to an increased number of vector-vector operations like axpy and dot-products: without deflation the average number of vectors involved in the Gram-Schmidt orthogonalization per newly constructed vector is about half of the basis size $\tilde{m}$ at a restart, i.e., $\tilde{m}/2$. With deflation of $k$ vectors and maximal basis size $\tilde{m}$ (so $m = \tilde{m} - k$ *new* vectors are generated before a restart) the average number of vectors in the Gram-Schmidt process is $(\tilde{m} + k)/2$, since there will always be orthogonalization with respect to the deflation space.

We give a comparison of GCR and fGMRES-DR in Fig. 3.7, with a varying size of the deflation space. The total size of the basis (size of deflation space plus number of iterations before a restart, $\tilde{m} = k+m$) is kept constant, which is reasonable, since this value is the decisive factor for memory consumption (however not for the orthonormalization cost). Without deflated restarts, i.e., when the size of the deflation space is zero, fGMRES-DR is equivalent to GCR, and both converge at the same speed. We can see

the slowdown of the GCR convergence after restarts, due to the loss of (especially) the approximation of the low modes. Keeping a few vectors with the smallest eigenvalues after a restart (size of the deflation space $> 0$) gives a significant improvement, because the slowdown in convergence is milder. Using beyond 16 deflation vectors does not yield an additional speedup.

To conclude, fGMRES-DR shows better performance than GCR, however there is an additional cost involved for the maintenance of the deflation space. We postpone a conclusion to Sec. 3.6.4, which has a big influence on the optimal algorithm choice.

### 3.6.4 Parameter tuning part 3: Iteration count of the preconditioner

We consider the algorithm performance depending on the Jacobi iteration count $j$. An initial consideration suggests that $j = 1$ gives an optimal total iteration count, but we will see afterwards that this is not so for the algorithm performance in practice. A decrease of the iteration count in the Jacobi method from $j$ to $j'$ increases the number of iterations in the outer Krylov space from $o$ to $o'$.[19] The fixed polynomial approximation built by the Jacobi iteration is inferior to the optimal Krylov space polynomial found by using GCR or GMRES. As a consequence, the product of the iteration counts will decrease, i.e., $o'j' < oj$ (assuming there is no restart), favoring a low or minimal ($j = 1$) Jacobi iteration count. Naively the total work depends only on $oj$, but in fact this is not so due to two crucial points:

- As discussed in Sec. 3.6.5, the iterations in the preconditioner can (and should) be done in low precision, i.e., they are much cheaper than outer iterations.
- For large $o$, memory limitations and exploding cost for the basis orthogonalization typically require restarting the outer iteration. The restarted GCR algorithm often shows poor performance, which could be avoided by increasing $j$ or by paying for some additional overhead to use deflated restarts with, e.g., fGMRES-DR. One example of this negative effect can be seen in Fig. 3.8: for $j = 16$, the restart after 48 GCR iterations stops the super-linear convergence.[20] After the restart it takes many iterations to get back to the quick convergence. For small $j$ we never enter the region of super-linear convergence, because the restart comes too early.

Furthermore, the Jacobi iteration does not rely on global dot-products and thus reduces negative effects of network latencies on parallel machines.

We give an example of a numerical test in Fig. 3.8, which shows the convergence of the GCR algorithm depending on $j$. More detailed data can be found in Table 3.5. For small $j$ (in this case $j = 1, 2, 4, 8$) the convergence is very slow and we need to restart the GCR algorithm many times. For $j = 8$ the convergence is still slow, but we start to see an acceleration of the convergence slightly before the restart. This super-linear

[19] Under the assumption that the preconditioner does not diverge.

[20] Super-linear convergence is a property of Krylov methods, but restarts can destroy this favorable feature.

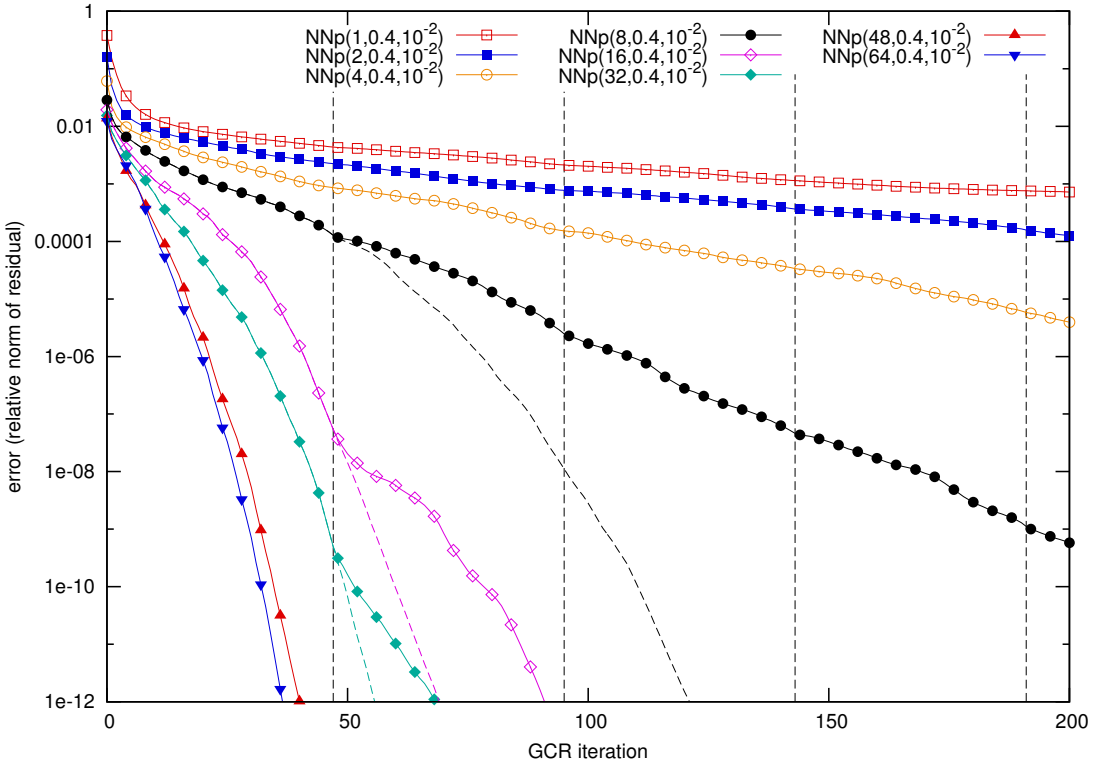



**Figure 3.8:** GCR convergence depending on iteration count $j$ of the Neumann-Neumann preconditioner with $8^4$ blocks. The configuration is $32^3 \times 64$ at $\kappa_{\text{sea}} = 0.13632$ with $\kappa_{\text{val}} = 0.13645$. Dots on the lines are drawn only on every 4th data point for clarity. The dashed vertical lines indicate the restarts of the GCR algorithm, after 48 iterations. For $j = 8, 16, 32$ the dashed lines show the convergence without restart, which were obtained on a machine with more memory.

convergence of GCR is however heavily disturbed by the restarts. For $j = 16$ the super-linear convergence sets in sufficiently long before a restart, which leads to a big overall speedup — compared to $j = 8$, the increase of $j$ by a factor of two led to a decrease of $o$ by a factor of three. However we still need to restart the algorithm once, so even larger $j$ might be beneficial. For $j = 32$ the convergence is faster, but not by a factor of two — the convergence of the preconditioner is not good enough, due to the suboptimal polynomial — and we still have to do one restart. We can further increase $j$ to 48 which makes the convergence fast enough for GCR without restart, and the overall performance is better than for $j = 32$. However, it is significantly *worse* than for $j = 16$. A further increase to $j = 64$ confirms that the Jacobi iteration is nearly stagnating — the difference to $j = 48$ is only marginal. The plot also shows the convergence without restarts for $j = 8, 16, 32$, which can be obtained on machines with more memory. We see the dramatic influence of the restarts for $j = 8$. In the other cases the effect is smaller.

Our conclusions, which also confirm our expectations given earlier, are:

- In this and many other cases $j$ can be chosen large enough to allow for GCR without restarts, however this is not automatically optimal.
- Large $j$ can be severely hampered by the suboptimal Jacobi polynomial. Increasing $j$ is beneficial mainly if the number of GCR restarts is reduced significantly.
- Such a significant reduction can be achieved by choosing $j$ such that we enter the region of super-linear convergence long before a restart.[21]
- Choosing $j$ even larger does not pay off.
- For this case with a restart after 48 iterations, $j = 16$ is optimal.[22]

A comparison of the results in this section with Sec. 3.6.3 raises the question whether one should use fGMRES-DR to alleviate the negative effects of the restarts, or just increase the work in the preconditioner by increasing $j$, such that no or very few restarts are necessary, without suffering too much from the bad Jacobi polynomial approximation. As usual the answer depends a lot on the problem. If the problems are very "hard", i.e., such that the convergence of GCR is poor, increasing $j$ sufficiently might not be feasible. Ultimately the convergence of the preconditioner stagnates, so increasing $j$ only increases the cost but not the convergence speed. Thus we cannot make the preconditioner better, resulting in the necessity of more GCR iterations. This will require to use an algorithm with restarts and fGMRES-DR will pay off.

### 3.6.5 Parameter tuning part 4: Precision of block inversions in the preconditioner

1. *Precision of the inversions of $A_{II}$ in the preconditioner*
   For $j > 1$ we have to apply $S$ (which involves $A_{II}^{-1}$) in the preconditioner. Typically this is a major contributor to the total algorithmic cost, so one should carefully optimize this part. The stopping criterium of the iterative inversions of $A_{II}^{-1}$ is based on the relative residual norm.[23] In the preconditioner it is not necessary to iterate until reaching (nearly) machine precision. It turns out that surprisingly low accuracy is sufficient: at $\varepsilon^{\text{prec.}}_{A_{II}^{-1}} = 10^{-2}$ the outer iteration count is almost the same as for $\varepsilon^{\text{prec.}}_{A_{II}^{-1}} = 10^{-14}$, but with a large reduction in the total cost. Going to larger error ($\varepsilon^{\text{prec.}}_{A_{II}^{-1}} = 10^{-1}$ or larger) will significantly worsen the outer iteration count and therefore does not pay off in our experience.

2. *Precision of the inversions of $S^{(i)}$ in the preconditioner*
   The preconditioner is based on the inversion of the subdomain Schur complements $S^{(i)}$. Here as well, an exact solve is not necessary. The negative effect

[21] As a consequence the optimal $j$ depends on the restart size.

[22] We often test only power-of-two values for $j$. The true optimum will in general lie in between, but a full optimization for every case is not feasible in practice. The optimum depends on the gauge configuration, so all one can do is to guess or optimize $j$ for one or a few configurations, assuming that similar values are also nearly optimal for all other configurations.

[23] Stopping at a fixed iteration count ($n^{\text{prec.}}_{A_{II}}$ in this case) is also common, but not used in this analysis.

$\kappa_{\text{val}} = 0.13632$

| $j$ | $o$ | Dirac hop | axpy | local dotp | dotp | communication |
|---|---|---|---|---|---|---|
| 2 | 298 | 408k [0.651] | 175k [0.619] | 115k [0.641] | 7003 | 996 [0.700] |
| 4 | 165 | 390k [0.795] | 163k [0.772] | 109k [0.788] | 3878 | 1103 [0.850] |
| 8 | 85 | 368k [0.885] | 152k [0.870] | 103k [0.881] | 1998 | 1137 [0.924] |
| 16 | 40 | 333k [0.937] | 137k [0.929] | 93k [0.935] | 940 | 1070 [0.962] |
| 32 | 24 | 382k [0.965] | 156k [0.960] | 106k [0.963] | 564 | 1284 [0.981] |

$\kappa_{\text{val}} = 0.13640$

| $j$ | $o$ | Dirac hop | axpy | local dotp | dotp | communication |
|---|---|---|---|---|---|---|
| 2 | 548 | 756k [0.654] | 324k [0.622] | 213k [0.645] | 12878 | 1831 [0.700] |
| 4 | 284 | 671k [0.797] | 282k [0.773] | 188k [0.790] | 6674 | 1898 [0.850] |
| 8 | 142 | 620k [0.888] | 257k [0.874] | 173k [0.884] | 3337 | 1898 [0.925] |
| 16 | 61 | 514k [0.940] | 211k [0.932] | 143k [0.937] | 1434 | 1631 [0.962] |
| 32 | 39 | 624k [0.967] | 255k [0.962] | 173k [0.966] | 916 | 2085 [0.981] |

$\kappa_{\text{val}} = 0.13645$

| $j$ | $o$ | Dirac hop | axpy | local dotp | dotp | communication |
|---|---|---|---|---|---|---|
| 2 | 1282 | 1761k [0.653] | 755k [0.621] | 496k [0.644] | 30127 | 4283 [0.700] |
| 4 | 609 | 1445k [0.799] | 606k [0.775] | 404k [0.792] | 14312 | 4069 [0.850] |
| 8 | 279 | 1221k [0.890] | 505k [0.875] | 340k [0.886] | 6556 | 3728 [0.925] |
| 16 | 92 | 778k [0.941] | 319k [0.933] | 216k [0.938] | 2162 | 2459 [0.962] |
| 32 | 70 | 1122k [0.968] | 458k [0.964] | 312k [0.967] | 1645 | 3742 [0.981] |
| 48 | 42 | 1014k [0.978] | 413k [0.975] | 281k [0.977] | 987 | 3368 [0.987] |
| 64 | 38 | 1215k [0.983] | 495k [0.981] | 337k [0.983] | 893 | 4062 [0.990] |

**Table 3.5:** GCR iteration count $o$ for three different valence quark masses, depending on the iteration count $j$ of the Neumann-Neumann preconditioner. The algorithm parameters are as in Fig. 3.8. Additional columns give the resulting algorithmic cost.

on the outer iteration count is smaller than discussed in item 1. We find that requesting a relative residual norm of $\varepsilon_{S^{(i)-1}} = 10^{-1}$ or even $\varepsilon_{S^{(i)-1}} = 0.2$ and $\varepsilon_{S^{(i)-1}} = 0.4$ works and pays off in the end.

Note that we observed some instabilities of the Neumann Neumann preconditioner, partially related to low precision. See Sec. 3.6.8 for details.

### 3.6.6 Multiple right-hand sides block inversions

For computing the Schur complement $S$ and the inverse of the subdomain Schur complements $S^{(i)}$ we need the inverse of $A_{II}$ and $A^{(i)}$. These matrices are rather small but still too large for an efficient handling of an explicit inverse, so the inversions are done iteratively for each right-hand side. The condition number is quite good (low modes

are spread out over large fractions of the volume and thus do not exist on the small blocks), so the convergence of the employed Krylov inverters is quick.[24] Nevertheless these inversions are the major contribution to the total algorithmic cost. As the matrices are fixed for all outer iterations and only the right-hand sides change we are dealing with a *sequential multiple right-hand sides problem*. We can try to improve the total performance by sharing or reusing information from solves with different right-hand sides.

Typical methods deflate the matrix (by using information about the small eigenvectors) or augment the Krylov space for the current right-hand side with information gained while solving the linear system with an earlier right-hand side. We tested a method of the latter type proposed in Ref. [53]. There, GMRES with deflated restarts is used for solving with the first right-hand side to obtain a deflation basis. These approximate eigenvectors are then used to augment the Krylov space for all other right-hand sides. This improves the iteration count, but in our case only by 10% to 20%, which does not balance the additional cost of GMRES-DR over BiCGstab. One reason for the small effect becomes apparent when we consider again Fig. 3.6: the eigenvalues of the blocks are already well separated from zero and lie in a dense bulk. Deflating a few of them has no large impact on the condition number, so the $\mathcal{O}(10)$ approximate eigenvectors we obtain from GMRES-DR will not lead to a big effect on the inverter.

We conclude that multiple right-hand sides methods are not useful for the Schur complement algorithm, unless new ideas lead to a significant improvement over the tested Ansatz.

### 3.6.7 Lowered kappa preconditioner

As a possible modification of the preconditioner we consider using a different $\kappa = \kappa_{\text{prec}}$ in the Neumann-Neumann preconditioner, keeping the target $\kappa$ only in the non-preconditioner part. The correct physical result will not be changed, since a modification in the preconditioner will not change the result of the outer inversion — only the path of convergence is modified. The motivation for this modification is the rather bad convergence of the Jacobi iteration at high $\kappa$ close to $\kappa_{\text{phys}}$, due to low modes of the Dirac operator. If we use a slightly smaller $\kappa_{\text{prec}}$, these low modes will be lifted and the convergence rate will improve. Since the low modes are approximated badly by the preconditioner anyway, the shift does not have a severe negative influence on the result of the preconditioner. For the high modes the changed $\kappa_{\text{prec}}$ will only lead to a small relative change. We can hope that the result is still close enough to the desired one, such that the preconditioner not only converges but still serves as a preconditioner for the right problem, i.e., for $\kappa$ of the outer inverter.

Our tests indicate that this method works, giving a slight improvement at no additional cost. We give examples for three different values of $\kappa_{\text{val}}$ in Figs. 3.9, 3.10, and 3.11. We observe a minimum in the number of, e.g., computed Dirac operator diag-

[24] The blocking serves as an infrared cutoff, as discussed in Sec. 3.6.2.

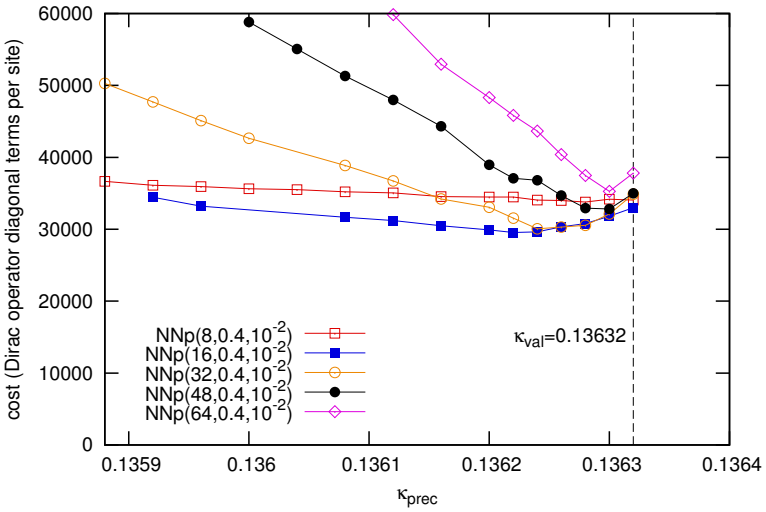



**Figure 3.9:** Influence of modified $\kappa = \kappa_{\mathrm{prec}}$ in the preconditioner with $8^4$ blocks and GCR as outer inverter. The configuration is $32^3 \times 64$ at $\kappa_{\mathrm{val}} = \kappa_{\mathrm{sea}} = 0.13632$. Choosing $\kappa_{\mathrm{prec}} < \kappa_{\mathrm{val}}$ can give a slight improvement.

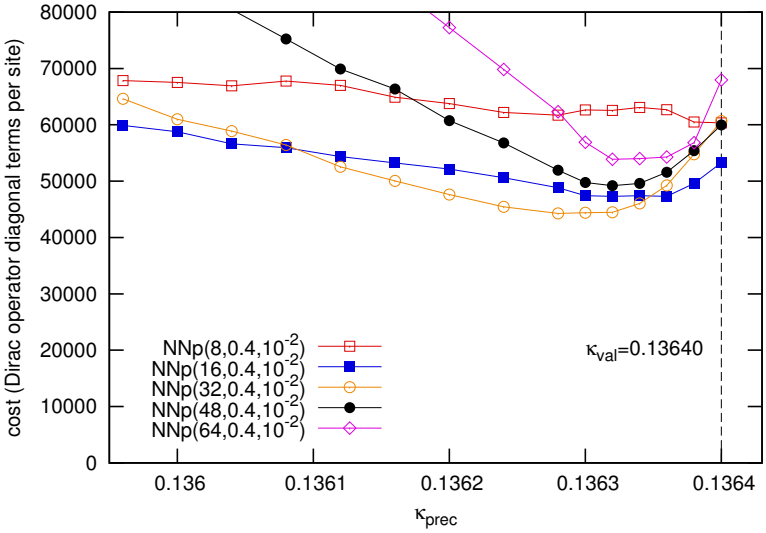



**Figure 3.10:** As Fig. 3.9 but at $\kappa_{\mathrm{val}} = 0.13640$. Choosing $\kappa_{\mathrm{prec}} < \kappa_{\mathrm{val}}$ gives a considerable improvement in a rather wide region. For small iteration count $j$ of the Neumann-Neumann preconditioner the effect is negligible or non-existent. Choosing $\kappa_{\mathrm{prec}} < \kappa_{\mathrm{val}}$ changes the optimal $j$. For $\kappa_{\mathrm{prec}} = \kappa_{\mathrm{val}}$ $j = 16$ is optimal, while at the observed minimum in the plot at $\kappa_{\mathrm{prec}} = 0.1363$ $j = 32$ is optimal. The gain is about 17%.

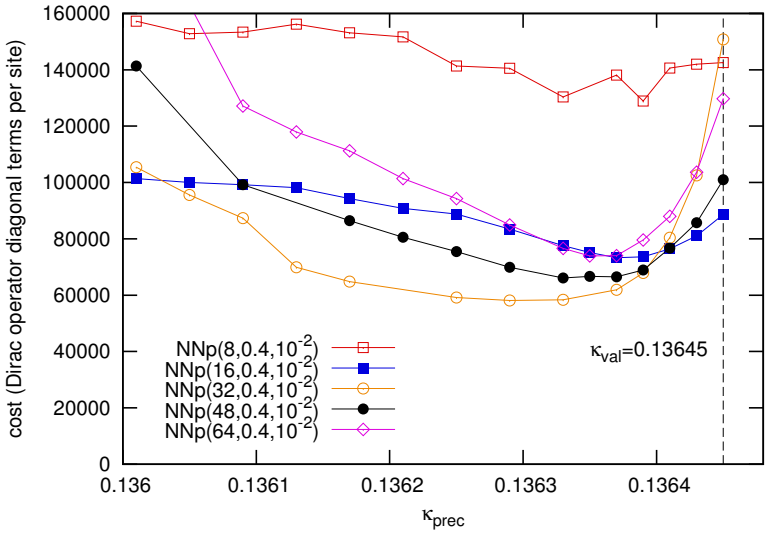



**Figure 3.11:** As Fig. 3.9 but at $\kappa_{\text{val}} = 0.13645$, which is near the physical value. Choosing $\kappa_{\text{prec}} < \kappa_{\text{val}}$ gives a considerable improvement in a wide region. For small iteration count $j$ of the Neumann-Neumann preconditioner the effect is negligible or non-existent. For $j = 32$ we observe an improvement by more than a factor of two. However, choosing $\kappa_{\text{prec}} < \kappa_{\text{val}}$ changes the optimal $j$. For $\kappa_{\text{prec}} = \kappa_{\text{val}}$ $j = 16$ is optimal, while at the observed minimum in the plot at $\kappa_{\text{prec}} = 0.13629$ $j = 32$ is optimal. The gain is about 34%.

onal terms at $\kappa_{\text{prec}} < \kappa_{\text{val}}$. This is due to a reduced outer iteration count, stemming from an improved convergence speed of the preconditioner. The change for the inner iterations (not given in the plot) is marginal, which agrees with the expectation: given that the low modes are cut off by the blocks, the small shift in the spectrum of the blocks away from zero has no effect. The change of the condition number is minor, resulting in only a small change for the block convergence.

At $\kappa_{\text{val}} = 0.13632$ the improvement is only minor, indicating that the convergence of the preconditioner is still quite good in this case, but when going to lighter quarks, e.g., to $\kappa_{\text{val}} = 0.13645$, the gain is 34%. If the shift of $\kappa_{\text{prec}}$ is too large the iteration count starts rising again, because ultimately the preconditioner will converge to the wrong point, keeping the outer inverter from converging. It is interesting to note that the optimal number of iterations $j$ in the Neumann-Neumann preconditioner depends on $\kappa_{\text{prec}}$, favoring higher $j$ for lower values. This can be explained by the bad convergence of the Jacobi iteration in the preconditioner at high $\kappa_{\text{prec}}$. After a small number of iterations the divergence of the Jacobi iteration outweighs the gain by the inherent smoothing (or further smoothing is inefficient, even if the iteration does not diverge), so any further work in the preconditioner is wasted and should rather be invested in more iterations of the outer inverter (making each of these iterations cheaper, with a low $j$). If we now lower $\kappa_{\text{prec}}$ this stagnation occurs later, at a larger $j$. Since in general iterations in the preconditioner are cheaper, it is beneficial to do more iterations, lowering the outer iteration count.

We can conclude that lowering $\kappa$ in the preconditioner can give a worthwhile improvement in the total algorithmic cost of up to 30% or more. We see these big benefits only for small quark masses near or at the physical values. The implementation is trivial and no additional cost is added to the algorithm. The parameter $\kappa_{\text{prec}}$ must be optimized, but this is not critical, as the wide minimum makes the tuning easy.

In cases where the preconditioner diverges at $\kappa_{\text{prec}} = \kappa_{\text{val}}$ (leading to a stagnation of the outer inverter) the use of $\kappa_{\text{prec}} < \kappa_{\text{val}}$ can make the outer inverter convergent. Thus, lowering $\kappa_{\text{prec}}$ would not only yield a speedup, but would rescue an algorithm which would otherwise fail to converge. However, it is a priori not clear if such a sweet spot exists: in principle the preconditioner could be divergent for all values of $\kappa_{\text{prec}}$ close to $\kappa_{\text{val}}$, and converge only for $\kappa_{\text{prec}} \ll \kappa_{\text{val}}$, where it converges to a point far from the correct one and thus does not serve as a good preconditioner for the outer inversion.

### 3.6.8 Dependence on configuration

The benchmarks done so far were only for a single gauge configuration. To ensure a realistic evaluation of the algorithm performance we performed tests on 16 configurations from the same Markov chain. The configurations are separated by 50 or 100 trajectories, so we can expect that the correlations are low. For this test we use fGMRES-DR(24,8) with NNp(4,$10^{-1}$,$10^{-2}$), unless stated otherwise. We give detailed results for various $\kappa$, since the behavior is partially erratic and worse than expected from the tests in the previous sections.

In Fig. 3.12 we show the convergence of fGMRES-DR for $\kappa = 0.13632$. The convergence speed is similar for all but configuration 1451. For this configuration the outer inverter stagnates. The reason is that the Neumann-Neumann preconditioner diverges in some cases. Increasing the precision of the local inversions in the preconditioner makes the divergence worse, leading to an earlier stagnation of GMRES. For a different source (right-hand side) the stagnation occurs as well, but later. Before we attempt an interpretation we first consider other values of $\kappa$.

For $\kappa = 0.13640$, shown in Fig. 3.13, all inversions converge, including configuration 1451. For $\kappa = 0.13645$, shown in Fig. 3.14, there are two configuration that lead to stagnation of GMRES, 1351 and 1451. In both cases, however, increasing the precision in the preconditioner leads to convergence of GMRES. That is, the stagnation is an artifact of a bad approximation in the preconditioner. For $\kappa = 0.13650$, shown in Fig. 3.15, the behavior is very similar to that for $\kappa = 0.13645$.

The conclusion is that the Neumann-Neumann preconditioner is unstable: for some gauge configurations it can diverge, even for comparably "easy" values of $\kappa$ (0.13632 in our case). This is not due to a low precision approximation on the blocks, and must therefore be a property of the Schur complement arising from the non-overlapping domain decomposition, or from the local approximation done in the Neumann-Neumann preconditioner as introduced in Sec. 3.4. There are two possible explanations:

1. The subdomain Schur complements $S^{(i)}$ used in the Neumann-Neumann precon-

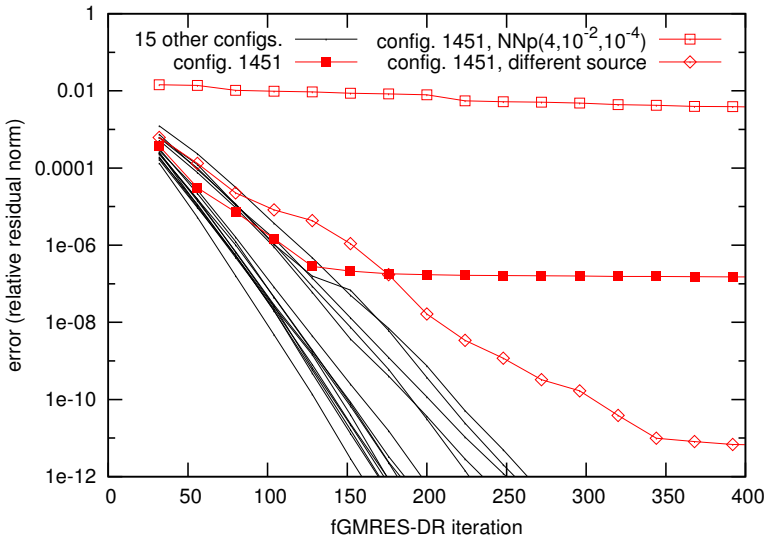



**Figure 3.12:** Convergence of fGMRES-DR(24,8) for the Schur complement, preconditioned with NNp(4,$10^{-1}$,$10^{-2}$), for 16 different configurations at $\kappa = 0.13632$.

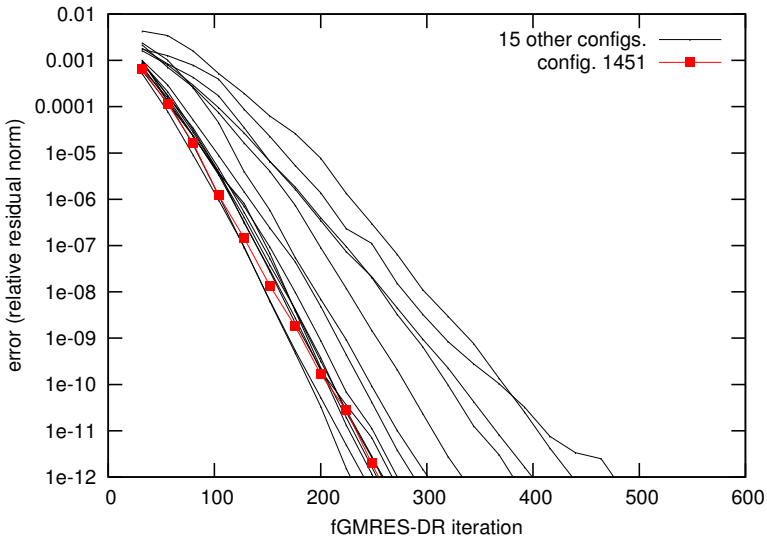



**Figure 3.13:** Convergence of fGMRES-DR(24,8) for the Schur complement, preconditioned with NNp(4,$10^{-1}$,$10^{-2}$), for 16 different configurations at $\kappa = 0.13640$.

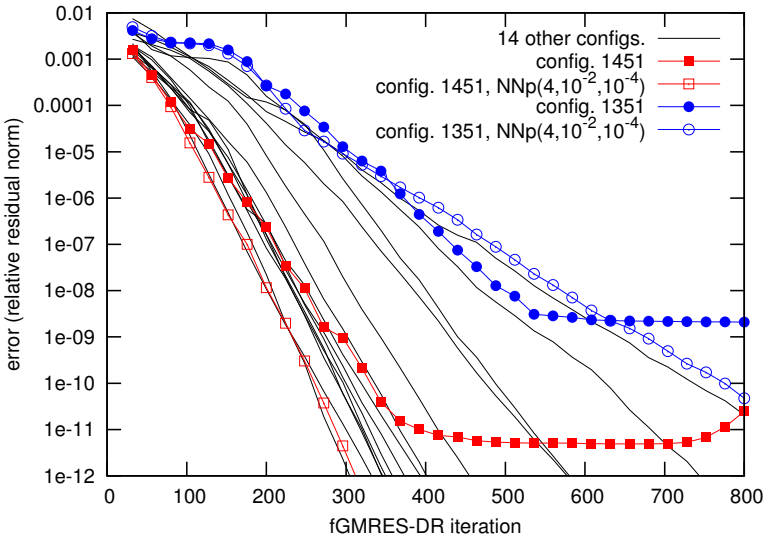



**Figure 3.14:** Convergence of fGMRES-DR(24,8) for the Schur complement, preconditioned with NNp(4,$10^{-1}$,$10^{-2}$), for 16 different configurations at $\kappa = 0.13645$.

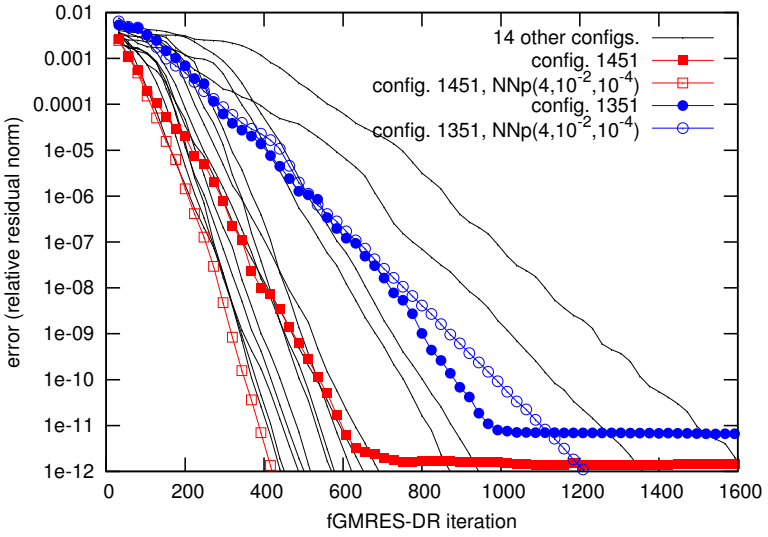



**Figure 3.15:** Convergence of fGMRES-DR(24,8) for the Schur complement, preconditioned with NNp(4,$10^{-1}$,$10^{-2}$), for 16 different configurations at $\kappa = 0.13650$.

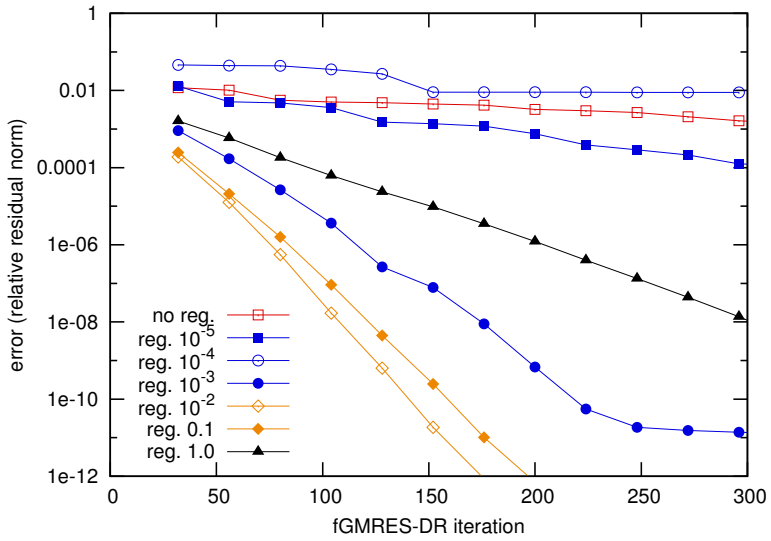



**Figure 3.16:** Convergence of fGMRES-DR(24,8) for the Schur complement, preconditioned with NNp(4,$10^{-2}$,$10^{-4}$), for configuration 1451 at $\kappa = 0.13632$. We show results for different regularization parameters in the inversion of the subdomain Schur complements $S^{(i)}$.

ditioner can be singular, see, e.g., Sec. 3.7.1 of Ref. [50]. In our case we did not observe instabilities or rising iteration counts for these inversion. We check the influence of a regularization parameter $\delta$ for $S^{(i)}$ — a constant term added on the diagonal — which would remove potential singularities.[25] The result is given in Fig. 3.16.[26] Without regularization ($\delta = 0$) GMRES stagnates at a relative residual norm of $\varepsilon \approx 0.01$, and similarly for small regularization $\delta = 10^{-5}$ and $\delta = 10^{-4}$. For $\delta = 10^{-3}$ GMRES converges initially, but stagnates at $\varepsilon = 10^{-11}$, so $\delta$ is still too small. At $\delta = 10^{-2}$ and $\delta = 0.1$ we obtain a convergent outer inverter, at an iteration count similar to that of the "well-behaved" configurations in Fig. 3.12, so the regularization works. However, if we further increase $\delta$ to 1.0 the convergence is significantly slowed down. The conclusion is that the regularization works well, however it is unclear whether tuning $\delta$ is feasible in practice, since the window where it yields good results is rather small. Presumably a pseudo-inverse would yield better results, but these tend to be more expensive to compute than an regular inversion with a Krylov method.

2. The spectrum of the Schur complement $S$ differs from that of the original operator: usually it has a *better* condition number, but since we are not aware of known bounds for the Wilson operator, we cannot exclude a deterioration. This could lead to a divergence of the Jacobi iteration in the preconditioner. As a consequence we cannot use $j > 1$ if we demand stable behavior. For

[25] Note that this is different from lowering $\kappa$ in the preconditioner, described in Sec. 3.6.7: there not only $S^{(i)}$ is modified, but also $S$ in the preconditioner.

[26] To simplify the implementation of the regularization we do not use even/odd preconditioning for the inverse of $S^{(i)}$ in this plot. This should have no significant influence on the outer inverter.

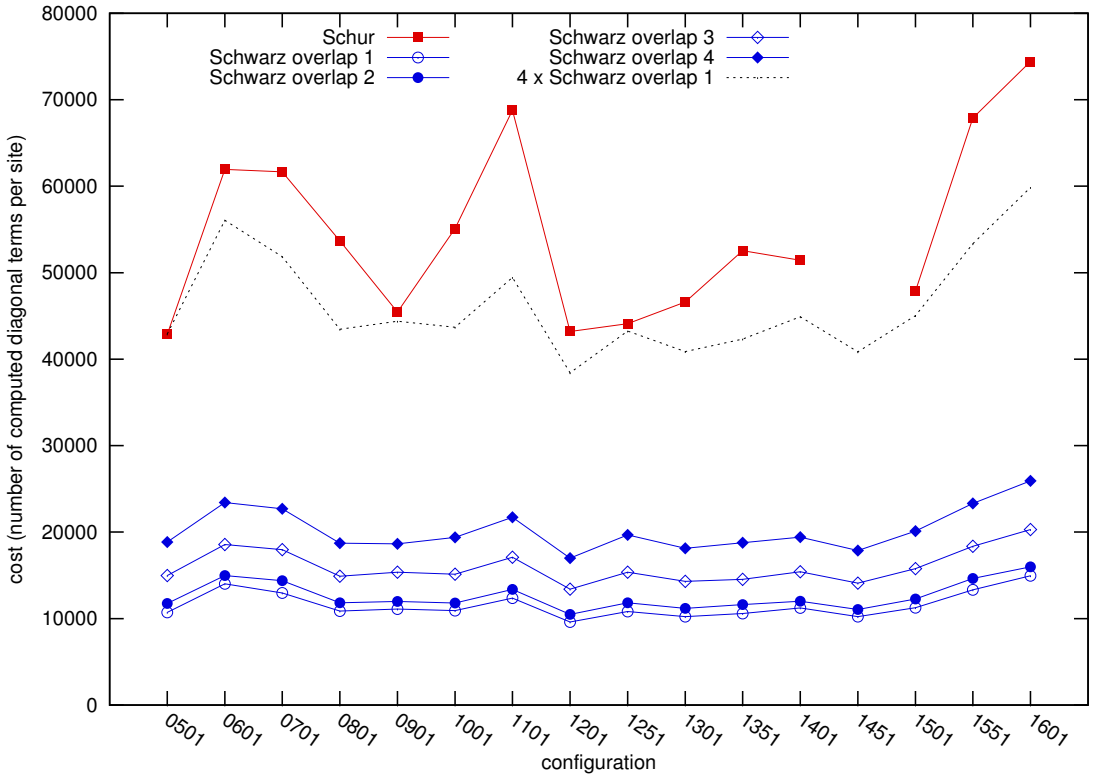



**Figure 3.17:** Comparison of the Schur method with the Schwarz method at $\kappa = 0.13632$. We use fGMRES-DR(24,8), Schur method with NNp(4,$10^{-1}$,$10^{-2}$), Schwarz method with $s = 4$ iterations and precision $\varepsilon = 10^{-1}$ for the subdomain problems. To visualize and compare the configuration dependence of both methods the dotted line gives the result of the Schwarz method with minimal overlap scaled by an (arbitrary) factor of 4.

$j = 1$ fGMRES-DR converges nicely for all configurations in our example at $\kappa = 0.13632$. However at large $\kappa$ the convergence becomes so slow that the frequent restarts prohibit convergence. Additionally, for $j = 1$ a big fraction of work is *outside* the preconditioner, where we have to compute $S$ to high precision:[27] since $S$ contains a local inversion it cannot be considered cheap, so $j = 1$ is very unfavorable (this is in contrast to an inversion of the Wilson operator in the case of Schwarz methods: since the operator itself is cheap, even for only one Schwarz iteration in the preconditioner a large fraction of the computation is part of the preconditioner).

### 3.6.9 Comparison with Schwarz methods

A generic comparison of the algorithm performance of the Schur complement algorithm and other inverters is impossible. The true performance will strongly depend on the

[27] Since we use $\varepsilon_{A_{II}^{-1}} \ll \varepsilon^{\text{prec.}}_{A_{II}^{-1}}$ a preconditioner iteration is much cheaper than an outer iteration.

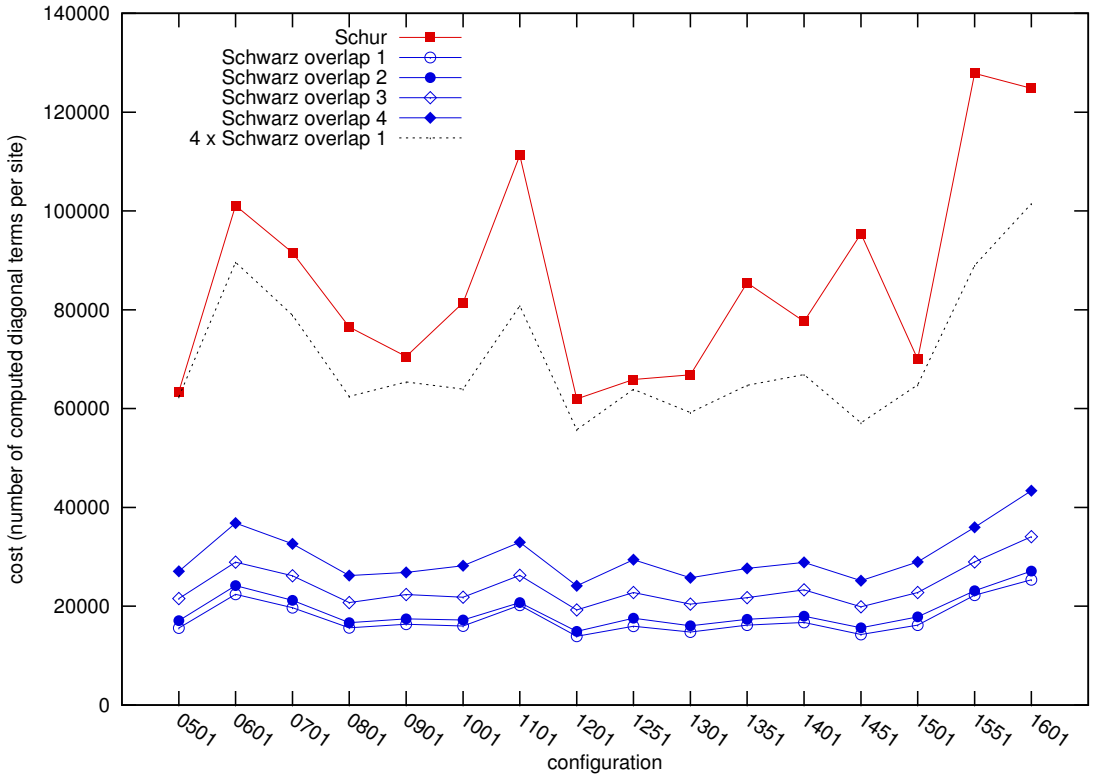



**Figure 3.18:** As Fig. 3.17, but at $\kappa = 0.13640$.

physical parameters and the target architecture. Here we want to compare with the Schwarz methods from Ch. 1. In that case a comparison is more feasible, since both algorithms are based on domain decomposition and involve local inversions. Thus also the amount of communication and global dot-products is of similar order and we can base a comparison on the number of calls to the Wilson operator.

We use data obtained for 16 configurations, as discussed in Sec. 1.4.2 for the Schwarz method and in Sec. 3.6.8 for the Schur method. In Fig. 3.17 to Fig. 3.20 we plot the number of computed diagonal Wilson Clover terms (per site) for convergence to a relative residual norm of $\varepsilon = 10^{-12}$ for $\kappa = 0.13632$, $\kappa = 0.13640$, $\kappa = 0.13645$, and $\kappa = 0.13650$.[28] The number of Wilson hopping terms is roughly proportional to the number of diagonal terms.[29]

We observe several things: (1) Typically the Schur method is 4 times as expensive as the Schwarz method. (2) The cost of the Schwarz method and the Schur method are correlated: cheap/expensive configurations for the Schwarz method typically are also cheap/expensive for the Schur method. This is the expected behavior, given that the spectra of the Schur complement and the original operator are related. (3) For the Schur complement there are larger relative differences in the inversion cost for different

[28] Some data points are missing since there was no convergence with the default parameter choice.
[29] Since the boundaries cut off some of the links there is no exact proportionality.

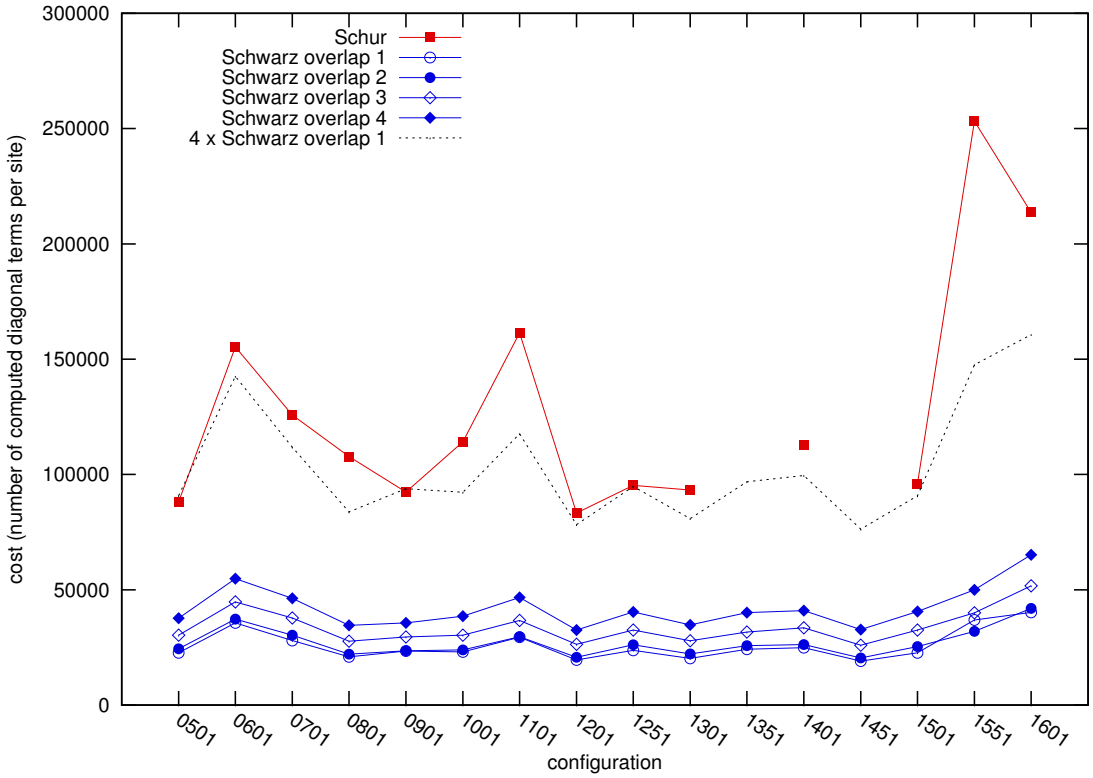



**Figure 3.19:** As Fig. 3.17, but at $\kappa = 0.13645$.

configurations than for the Schwarz method, i.e., the Schur method appears to be less stable.

## 3.7 Conclusions

In this chapter we have introduced and discussed an inverter for the Dirac operator based on the Schur complement of a non-overlapping domain decomposition of the four-dimensional volume. The method originates from finite-element discretizations and we showed how it can be employed for the finite-difference discretization of the Dirac operator. We also introduced and discussed the Neumann-Neumann preconditioner. We gave an ample study of the convergence of this Schur complement inverter with the preconditioner. The algorithm performance depends on parameters, but our systematic study showed that this dependence is quite well understood and under control, so a choice of near-optimal parameters should not be a big obstacle in practice.

However, the convergence of the preconditioner is not stable for all configurations: sporadically some configurations lead to strong divergences for certain values of $\kappa$. These divergences are in contrast to those where a slow divergence sets in when $\kappa$ is increased (independently of the configuration), as observed in both the Neumann-Neumann preconditioner and the Schwarz methods (see, e.g., Fig. 1.8b). These "sys-

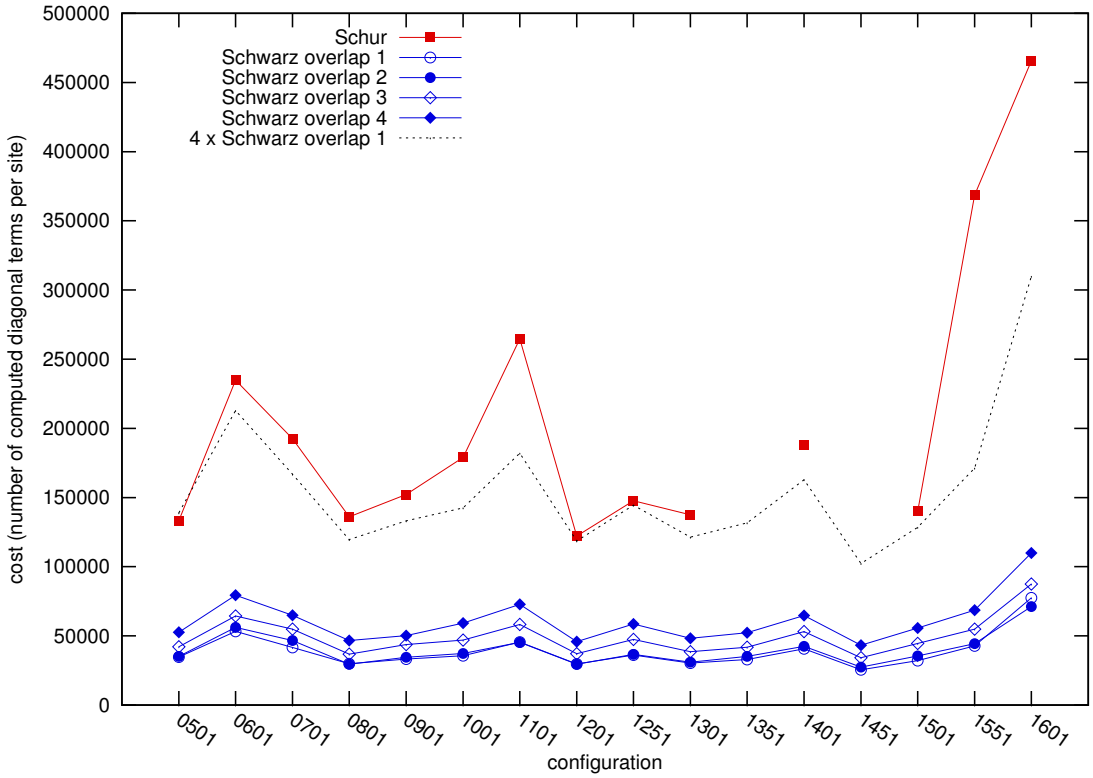



**Figure 3.20:** As Fig. 3.17, but at $\kappa = 0.13650$.

tematic" divergences have to be distinguished from the aforementioned strong divergence. As a consequence the Schur method is unpredictable (at least with our current knowledge): while the behavior seems systematic and well controlled for most configurations, there are cases where it completely fails. We thus have to conclude that the Schur method with Neumann-Neumann preconditioner is not viable for QCD, unless we understand and control the divergences, e.g., by a suitable pseudo-inverse of the subdomain Schur complements.

Apart from that, like other domain decomposition algorithms, the Schur method is well-suited for parallel computers due to the local inversions of the blocks, which make up the major part of the algorithm. This is similar to the Schwarz method. Other algorithms such as conjugate gradient severely suffer from limitations by the network.

The raw algorithmic cost of the Schur method — neglecting any differences in possible code optimization for a specific machine — is considerably higher than for the Schwarz method and for Krylov inversions without preconditioners. The latter might still be outperformed, because of the aforementioned better suitability of the Schur method for parallel computers. The former, however, is very similar in that respect, and we are forced to conclude that the Schur method is inferior to Schwarz methods. Additionally, the implementation is more complex and difficult than for the Schwarz method.

To close, we mention three options, which might improve the performance of the Schur method, and discuss why we did not consider them in more detail.

1. The reason for the slow convergence of the algorithm is the condition number of the Schur complement $S$. While it is improved with respect to the Dirac operator, it is still far from satisfactory. One can try to use *very large* blocks and hope that the condition number of $S$ is better — in a preliminary test we did *not* see a sufficient improvement, however. Since the blocks are very large the block inverters will converge slowly, so one can apply the block decomposition recursively, i.e., invert each of the blocks with the Schur method as well. Such a recursive algorithm could also be worthwhile for the arising computing architectures with more than one level, like the Intel MIC architecture. For example, in a cluster with many MICs, each MIC could hold one large block, and each core or thread on a MIC holds one of the small blocks that make up the large blocks. The same approach is of course also possible for the Schwarz method.

2. As discussed in Sec. 3.6.6, the main contributors to the algorithmic cost are the block inversions, which are repeated many times with the same matrix but different right-hand sides. We believe this is the most promising starting point for an improvement. However, if such an improvement is found it will automatically work for the Schwarz methods as well, which also suffer from expensive block inversions. That is, the Schur method would still be inferior.

3. Coarse grids or inexact deflation are certainly a way to improve the Schur method. Here as well, however, the Schwarz methods (or multigrid methods) profit to a similar extent.

# Part II

# Sign function and overlap operator

*This chapter is verbatim as published with Jacques Bloch in Comput. Phys. Commun., Volume 182, Issue 4, April 2011, Pages 878–889*

# Chapter 4

# A Nested Krylov subspace method to compute the sign function of large complex matrices

*We present an acceleration of the well-established Krylov-Ritz methods to compute the sign function of large complex matrices, as needed in lattice QCD simulations involving the overlap Dirac operator at both zero and nonzero baryon density. Krylov-Ritz methods approximate the sign function using a projection on a Krylov subspace. To achieve a high accuracy this subspace must be taken quite large, which makes the method too costly. The new idea is to make a further projection on an even smaller, nested Krylov subspace. If additionally an intermediate preconditioning step is applied, this projection can be performed without affecting the accuracy of the approximation, and a substantial gain in efficiency is achieved for both Hermitian and non-Hermitian matrices. The numerical efficiency of the method is demonstrated on lattice configurations of sizes ranging from $4^4$ to $10^4$, and the new results are compared with those obtained with rational approximation methods.*

## 4.1 Introduction

In quantum chromodynamics (QCD) some physical observables rely on the chiral properties of the theory. To study such observables in a lattice formulation of QCD it is important to discretize the Dirac operator such that it respects the corresponding chiral symmetry. This is most faithfully achieved using the overlap Dirac operator [54, 55]. To study QCD at nonzero baryon density the overlap formulation was recently extended to include a quark chemical potential [8, 9]. A major ingredient in the overlap operator, which makes its use very challenging, is the computation of the sign function of a complex matrix, which is Hermitian at zero baryon density, but becomes non-Hermitian when a quark chemical potential is introduced.

The search for efficient numerical methods to compute the sign function for the large sparse matrices encountered in this context is an ongoing field of research. Typically, Krylov subspace methods are employed to evaluate the operation of a matrix function on an arbitrary vector. We distinguish two main variants: the Krylov-Ritz approximation, which evaluates the function via a projection on the Krylov subspace, and the rational approximation, where the function is first approximated by a partial

fraction expansion, which is then efficiently solved using a multi-shift Krylov subspace inverter.

In the Hermitian case efficient rational approximation methods for the sign function have been devised [57, 76] and are currently being used in large scale lattice simulations. The current method of choice uses the Zolotarev partial fraction expansion [76, 12, 40], which yields the optimal rational approximation to the sign function over a real interval [82], in conjunction with a multi-shift conjugate gradient inversion. For non-Hermitian matrices, which occur in the presence of a quark chemical potential, Krylov subspace approximations to the sign function are relatively new and still under development. Recently, partial fraction expansion methods using the Neuberger expansion [57] with non-Hermitian multi-shift inverters were proposed [4].

The Krylov-Ritz approximation, which we discuss in this paper, is based on the construction of a Krylov basis and its accompanying Ritz matrix. Depending on the algorithm used to construct the basis we distinguish between the Lanczos approximation in the Hermitian case [76], and the Arnoldi approximation [6] or two-sided Lanczos approximation [5] in the non-Hermitian case. The latter clearly yields the more efficient function approximation for non-Hermitian matrices [5]. In the Krylov-Ritz approximation the large complex matrix is projected on the Krylov subspace, and its sign function is approximated by lifting the sign function of its projected image (Ritz matrix) back to the original space. The latter sign function is computed to high accuracy using the spectral definition of a matrix function or using a matrix-iterative method. When a large Krylov subspace is needed to reach the desired accuracy, the computation of this matrix sign function becomes a bottleneck for the algorithm.

Herein we will introduce an enhancement of the Krylov-Ritz approximation method which substantially reduces the cost of this internal sign computation and boosts the efficiency of the overall method, such that it competes with, and even surpasses, the rational function approximation in both the Hermitian and non-Hermitian case. The dramatic reduction in computation time is achieved by projecting the Ritz matrix on an even smaller, nested Krylov subspace, after performing a suitable preconditioning step first. The desired sign function is then computed via the sign function of the inner Ritz matrix, which yields the same accuracy as the original Krylov-Ritz approximation.

The outline of the paper is as follows. In Sec. 4.2 we introduce the overlap operator and the matrix sign function. In Sec. 4.3 we show how the matrix function of large matrices is computed using Krylov-Ritz approximation methods. In Sec. 4.4 we introduce the nested Krylov subspace method, which substantially enhances the efficiency of the Krylov-Ritz approximation to the sign function. We study its convergence properties and present numerical results for various lattice sizes, including a comparison with rational approximation methods. Finally, our conclusions are given in Sec. 4.5. For completeness we have added some algorithms in Appendix C.

## 4.2 Overlap operator and the matrix sign function

Our motivation to develop numerical algorithms to compute the matrix sign function of large, sparse, complex matrices comes from its application in lattice quantum chromodynamics (LQCD). The overlap formulation of the Dirac operator [54, 55], which ensures that chiral symmetry is preserved in LQCD, is given in terms of the matrix sign function [58], and its definition in the presence of a quark chemical potential $\mu$ [8] is given by

$$D_{\mathrm{ov}}(\mu) = \mathbb{1} + \gamma_5 \operatorname{sgn}(\gamma_5 D_{\mathrm{w}}(\mu)), \tag{4.1}$$

where $\mathbb{1}$ denotes the identity matrix, $\gamma_5 = \gamma_1\gamma_2\gamma_3\gamma_4$ with $\gamma_1, \ldots, \gamma_4$ the Dirac gamma matrices in Euclidean space, sgn is the matrix sign function, and

$$D_{\mathrm{w}}(\mu) = \mathbb{1} - \kappa \sum_{i=1}^{3} (T_i^+ + T_i^-) - \kappa(e^{\mu} T_4^+ + e^{-\mu} T_4^-) \tag{4.2}$$

is the Wilson Dirac operator at nonzero chemical potential [35] with $(T_\nu^\pm)_{yx} = (\mathbb{1} \pm \gamma_\nu) U_{x,\pm\nu} \delta_{y,x\pm\hat{\nu}}$, $\kappa = 1/(8 + 2m_{\mathrm{w}})$, $m_{\mathrm{w}} \in (-2, 0)$ and $U_{x,\pm\nu} \in \mathrm{SU}(3)$, where $U_{x,-\nu} = U^\dagger_{x-\hat{\nu},+\nu}$. The exponential factors $e^{\pm\mu}$ implement the quark chemical potential on the lattice. For $\mu = 0$ the argument of the sign function is Hermitian, while for $\mu \neq 0$ it is non-Hermitian. To compute the overlap operator we need to define the matrix sign function for a general complex matrix $A$ of dimension $n$. A generic matrix function $f(A)$ can be defined by

$$f(A) = \frac{1}{2\pi i} \oint_\Gamma f(z)(zI - A)^{-1} dz, \tag{4.3}$$

where $\Gamma$ is a collection of contours in $\mathbb{C}$ such that $f$ is analytic inside and on $\Gamma$ and such that $\Gamma$ encloses the spectrum of $A$. If $A$ is diagonalizable, i.e., $A = U\Lambda U^{-1}$, with diagonal eigenvalue matrix $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ and $U \in \mathrm{Gl}(n, \mathbb{C})$, then this general definition can be simplified to the well-known spectral form

$$f(A) = U f(\Lambda) U^{-1}, \tag{4.4}$$

with

$$f(\Lambda) = \operatorname{diag}\left(f(\lambda_1), \ldots, f(\lambda_n)\right). \tag{4.5}$$

If $A$ cannot be diagonalized, a spectral definition of $f(A)$ can still be derived using the Jordan decomposition [30]. For simplicity, but without loss of generality, we assume diagonalizability in the following. For Hermitian $A$ the eigenvalues are real and their sign is defined by $\operatorname{sgn}(x) = \pm 1$ for $x \gtrless 0$ with $x \in \mathbb{R}$, such that Eq. (4.4) readily defines the matrix sign function. For non-Hermitian $A$ the eigenvalues are complex and require a definition of $\operatorname{sgn}(z)$ for $z \in \mathbb{C}$. The sign function needs to satisfy $(\operatorname{sgn}(z))^2 = 1$ and reproduce the usual $\operatorname{sgn}(x)$ for real $x$. We define

$$\operatorname{sgn}(z) = \frac{z}{\sqrt{z^2}} = \operatorname{sgn}\left(\operatorname{Re}(z)\right), \tag{4.6}$$

where the cut of the square root is chosen along the negative real axis. This choice, although not unique, gives the correct physical result for the overlap Dirac operator in Eq. (4.1) (see Ref. [9]).

## 4.3 Krylov-Ritz approximations for matrix functions

Since we aim at problems with large matrices, as is the case in LQCD, memory and computing power limitations require sophisticated methods to deal with the sign function. For a matrix $A$ of large dimension $n$ the common approach is not to compute $f(A)$ but rather its action on a vector, i.e., $y = f(A)x$, which is needed by iterative inverters to compute $f(A)^{-1}b$ or by iterative eigenvalues solvers for $f(A)$. The Krylov-Ritz method approximates the resulting vector in the Krylov subspace

$$\mathcal{K}_k(A, x) \equiv \text{span}(x, Ax, A^2x, \ldots, A^{k-1}x) \tag{4.7}$$

of $\mathbb{C}^n$, implicitly making a polynomial approximation of degree $k-1$ to $f(A)$. The optimal approximation to $y$ in this subspace is its orthogonal projection $y_k^\perp$. For $V_k = (v_1, \ldots, v_k)$, where the $v_i$ form an orthonormal basis of $\mathcal{K}_k(A, x)$, an orthogonal projector is given by $P = V_k V_k^\dagger$, and we have

$$y = f(A)x \approx y_k^\perp = Pf(A)x. \tag{4.8}$$

However, to compute this projection on the Krylov subspace we already need $y$, which is the quantity we wanted to determine in the first place. Thus, we need to replace this exact projection by an approximation. To reduce the large dimensionality of the problem one typically projects $A$ on the Krylov subspace using $A_k \equiv PAP$. The projected matrix $A_k$ has dimension $n$ but rank at most $k$. The $k$-dimensional image of the projected matrix $A_k$ is defined by the matrix $H_k = V_k^\dagger A V_k$, which is often referred to as Ritz matrix. The components of $H_k$ are the projection coefficients of $A_k$ in the basis $V_k$, as $A_k$ and $H_k$ are related by $A_k = V_k H_k V_k^\dagger$ (in analogy to the vector case).

The Krylov-Ritz approximation [26, 64] to $f(A)$ consists in taking the function of the Ritz matrix $H_k$ and lifting it back to the full $n$-dimensional space,

$$f(A) \approx V_k f(H_k) V_k^\dagger. \tag{4.9}$$

This approximation actually replaces the polynomial interpolating $f$ at the eigenvalues of $A$ by the polynomial interpolating $f$ at the eigenvalues of $H_k$, also called Ritz values [64]. Substituting the approximation (4.9) in $f(A)x$ yields

$$y \approx V_k f(H_k) V_k^\dagger x = |x| V_k f(H_k) e_1^{(k)}, \tag{4.10}$$

where we choose $v_1$ collinear with $x$, i.e., $v_1 = V_k e_1^{(k)} \equiv x/|x|$, with $e_1^{(k)}$ the first unit vector of $\mathbb{C}^k$. To evaluate the approximation (4.10) we do not need to perform the matrix multiplications of Eq. (4.9) explicitly. First, one computes the function $f(H_k)$ of the $k$-dimensional Ritz matrix to high accuracy, using the spectral definition

(4.4) or a matrix-iterative method. Then, the final approximation is simply a linear combination of the basis vectors $v_i$, with coefficients given by the first column of $f(H_k)$ multiplied with $|x|$.

The Krylov-Ritz approximation described above uses an orthonormal basis of the Krylov subspace $\mathcal{K}_k(A, x)$. For the Hermitian case such a basis can efficiently be constructed using the Lanczos algorithm, which we listed in C.1 for completeness. It generates an orthonormal basis and a tridiagonal symmetric $H_k$ using a three-term recurrence relation. The non-Hermitian case is more laborious as the construction of an orthonormal basis is typically performed using the Arnoldi algorithm, which suffers from long recurrences as each basis vector has to be orthogonalized with respect to all the previous ones. The two-sided Lanczos algorithm is a suitable alternative [5] which uses two three-term recurrences to construct bases $V_k = (v_1, \dots, v_k)$ and $W_k = (w_1, \dots w_k)$ of the right, respectively left, Krylov subspaces $\mathcal{K}_k(A, x)$ and $\mathcal{K}_k(A^\dagger, x)$, which are biorthonormal, i.e., $v_i^\dagger w_j = \delta_{ij}$ (see C.2 for a listing of the algorithm). The lack of orthogonality of the basis $V_k$ prevents the construction of the orthogonal projector needed for the Krylov-Ritz function approximation (4.9). Nevertheless, the biorthonormality between $V_k$ and $W_k$ can be used to construct an oblique projector $P = V_k W_k^\dagger$ on the right Krylov subspace. The oblique projection of $A$ is $A_k = PAP$ and its $k$-dimensional image is defined by $H_k = W_k^\dagger A V_k$, which we call two-sided Ritz matrix, such that $A_k = V_k H_k W_k^\dagger$. The matrix $H_k$ generated by the two-sided Lanczos algorithm is tridiagonal. The two-sided Krylov-Ritz approximation to $f(A)$ then consists in taking the matrix function of $H_k$ and lifting it back to the original space,

$$f(A) \approx V_k f(H_k) W_k^\dagger. \tag{4.11}$$

After applying this approximation of $f(A)$ to $x$ we find an expression which is similar to Eq. (4.10),

$$y \approx V_k f(H_k) W_k^\dagger x = |x| V_k f(H_k) e_1^{(k)}, \tag{4.12}$$

where the last step assumes that $v_1 = V_k e_1^{(k)} \equiv x/|x|$. The price paid to achieve short recurrences in the non-Hermitian case is the loss of orthogonality of the projection on the Krylov subspace, which translates in a somewhat lower accuracy of the two-sided Lanczos approximation compared to the Arnoldi approximation, for equal Krylov subspace sizes. Nevertheless, the large gain in speed makes it by far the more efficient method [5].

In the case where $f$ is the sign function, the approximations (4.10) and (4.12) require the computation of $\mathrm{sgn}(H_k)$. Although it could be computed directly with the spectral definition (4.4), matrix-iterative methods are often cheaper for medium sized matrices. We choose to employ the Roberts-Higham iteration (RHi) [62]: Set $S_0 = H_k$ and compute

$$S_{n+1} = \frac{1}{2}(S_n + S_n^{-1}). \tag{4.13}$$

This iteration converges quadratically to $\operatorname{sgn}(H_k)$, if the sign function for complex arguments is defined by Eq. (4.6). The matrix inversion scales like $k^3$ and so will the RHi. For the QCD application considered here, typically 7 to 10 iterations are necessary to converge within machine precision [6, 5].

The hope is that the Krylov-Ritz approximations (4.10) and (4.12) are accurate for $k \ll n$. The method is known to work very well as long as no eigenvalues are close to a function discontinuity. However, for the sign function this method suffers from the sign discontinuity along the imaginary axis. If $A$ has eigenvalues close to this discontinuity the approximating polynomial must steeply change from $-1$ to $+1$ over a small interval to give an accurate approximation. This cannot be achieved with a low order polynomial, i.e., the Krylov subspace must be large, which makes the algorithm expensive. The common solution to this problem is to use deflation, where the contribution of the eigencomponents associated to these critical eigenvalues to the sign function is computed exactly.[1] The Krylov subspace approximation is then performed in a deflated space, i.e., the subspace where the directions along the critical eigenvectors have been removed. We refer to the literature for details [6].

The convergence of the Krylov-Ritz approximations to the matrix sign function is illustrated in Fig. 4.1: the Lanczos approximation for the Hermitian case on the left, and the two-sided Lanczos approximation for the non-Hermitian case on the right. The accuracy of the approximation cannot be determined by comparing to the exact value $\operatorname{sgn}(A)x$, as its evaluation by direct methods is too costly if A is large. To obtain an estimate for the error, we compute $\tilde{x} \approx \operatorname{sgn}(A)^2 x$ (by applying the Krylov-Ritz approximation twice in succession), which should equal $x$ if the approximation to the sign function were exact, and then take $\varepsilon = |\tilde{x} - x|/2|x|$ as a measure for the error. This error estimate proved to be consistent with the true error obtained by comparing the approximation to the exact solution for $4^4$ and $6^4$ lattices, and will therefore be used for all lattice sizes. Here, and in all subsequent tests, we choose the test vector $x = (1, \ldots, 1)$. As expected, the accuracy improves with increasing Krylov subspace size $k$, and a larger deflation gap $\Delta$, corresponding to a higher number of deflated eigenvectors, leads to a faster convergence. For a given accuracy and equal deflation gap, the subspace size $k$ required for non-Hermitian $A$ is larger than for Hermitian $A$.

To analyze the efficiency of the algorithm we briefly sketch the three major contributions to the total CPU time. For each matrix $A$ the deflation requires the computation of the critical eigenvalues and the corresponding eigenvectors. The time needed by the rest of the algorithm strongly depends on the eigenvalue gap, as the Krylov subspace size can be reduced if the deflation gap is increased. As mentioned at the beginning of this section, the product $f(A)x$ is usually needed for many source vectors $x$, e.g.,

[1] In practice we deflated the eigenvalues with smallest modulus $|\lambda|$ instead of those with smallest absolute real part $|\operatorname{Re}\lambda|$, as the former are more efficiently determined numerically, and both choices yield almost identical deflations for the operator $\gamma_5 D_{\mathrm{w}}(\mu)$ of Eq. (4.1). The reason for this is that, as long as the chemical potential $\mu$ is not unusually large, the spectrum looks like a very narrow bow-tie shaped strip along the real axis, and the sets of eigenvalues with smallest absolute real parts and smallest magnitudes will nearly coincide. In the following we therefore define the deflation gap $\Delta$ as the largest deflated eigenvalue in magnitude, i.e., $\Delta = \max|\lambda_{\mathrm{defl}}|$.

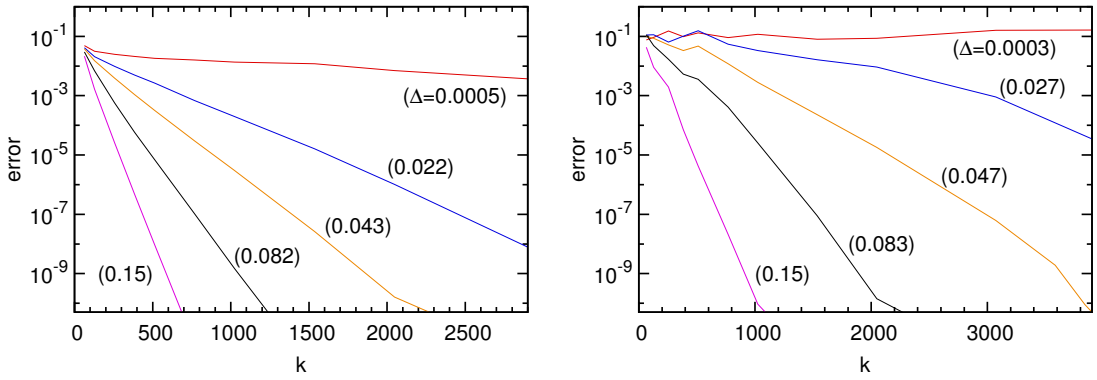



**Figure 4.1:** Accuracy of the Krylov subspace approximation for $y = \mathrm{sgn}(A)x$, where $A$ is $\gamma_5 D_\mathrm{w}(\mu)$ for a $6^4$ lattice (for a lattice volume $V$ the matrix $\gamma_5 D_\mathrm{w}$ has dimension $12V$, such that $\dim(A) = 15552$ here). Left pane: Hermitian case ($\mu = 0$) using the Lanczos method, right pane: non-Hermitian case with chemical potential $\mu = 0.3$ using the two-sided Lanczos method. The relative error $\varepsilon$ is shown as a function of the Krylov subspace size $k$ for different deflation gaps $\Delta$ (given in parenthesis).

as part of an iterative inversion. In this case the expensive deflation of $A$ only needs to be performed once in an initialization step, while the Krylov subspace part of the algorithm will be repeated for each new vector $x$. For this reason we assume from now on that an initial deflation has been performed and we will concentrate on the efficiency of the Krylov subspace part of the algorithm. We discern two main components in the Krylov-Ritz method: the construction of the Krylov basis using the Lanczos or two-sided Lanczos algorithms, where the computation time grows linearly with the subspace size $k$, and the RHi to compute $\mathrm{sgn}(H_k)$, which scales as $k^3$. Figure 4.2 illustrates these last two contributions. For high accuracy the Krylov subspace becomes large such that the cost of the RHi dominates the total CPU time of the Krylov-Ritz approximation and the method becomes too costly. In the following, the implementation of the Krylov-Ritz approximation for which $\mathrm{sgn}(H_k)$ is computed using Eq. (4.13) will be referred to as *non-nested method*. In the next section we will present a *nested* Krylov subspace method, which drastically reduces the cost to compute $\mathrm{sgn}(H_k)e_1^{(k)}$ and vastly improves the overall efficiency of the Krylov-Ritz approximation.

## 4.4 Nested Krylov subspace method for the sign function

### 4.4.1 Nesting and preconditioning

We introduce a new method which speeds up the expensive computation of the vector $\mathrm{sgn}(H_k)e_1^{(k)}$ required in the Krylov-Ritz approximations (4.10) and (4.12) to $\mathrm{sgn}(A)x$. The idea is to approximate this matrix-vector product by a further Krylov-Ritz approximation, using a second, nested Krylov subspace (specified below) of size $\ell \ll k$,

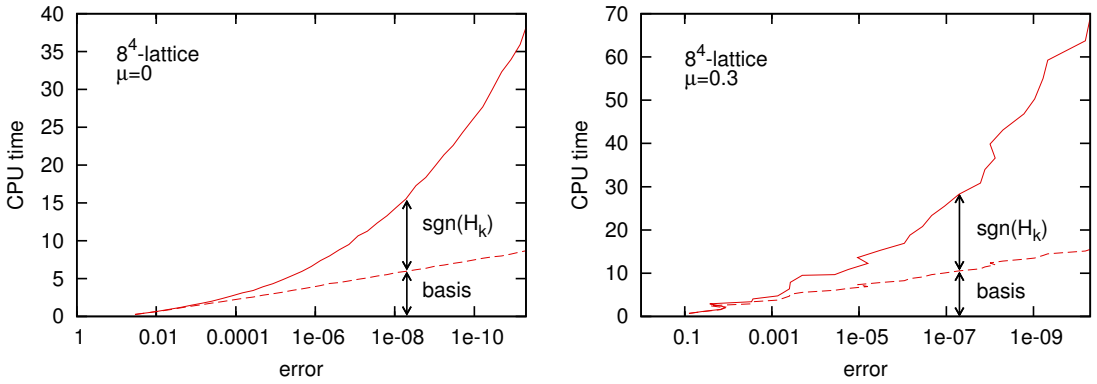



**Figure 4.2:** CPU time $t$ (in seconds) versus accuracy for an $8^4$ lattice configuration in the Hermitian case with deflation gap $\Delta = 0.055$ (left) and the non-Hermitian case with $\mu = 0.3$ and deflation gap $\Delta = 0.107$ (right). The full line shows the total time required to compute $\mathrm{sgn}(A)x$, while the dashed line gives the time needed to construct the Krylov basis. The difference between both lines represents the time taken by the RHi to compute $\mathrm{sgn}(H_k)$. The irregular convergence pattern for the non-Hermitian case is a well-known feature of the two-sided Lanczos algorithm.

i.e.,

$$\mathrm{sgn}(H_k)e_1^{(k)} = V_\ell \,\mathrm{sgn}(H_\ell)e_1^{(\ell)}, \tag{4.14}$$

where $V_\ell$ is the matrix containing the basis vectors of the inner Krylov subspace, constructed with the Lanczos or two-sided Lanczos method, and $H_\ell$ is the inner Ritz or two-sided Ritz matrix. The $\mathrm{sgn}(H_\ell)$ is computed using the RHi on the inner Ritz matrix $H_\ell$. After substituting this result in Eq. (4.10) and (4.12) we get the nested approximation

$$y \approx |x|\, V_k V_\ell \,\mathrm{sgn}(H_\ell)e_1^{(\ell)} \tag{4.15}$$

to $\mathrm{sgn}(A)x$. By introducing an additional Krylov subspace, the number of operations necessary to compute $\mathrm{sgn}(H_k)e_1^{(k)}$ is reduced from $\mathcal{O}(k^3)$ in the non-nested method to $\mathcal{O}(\ell^3) + \mathcal{O}(k\ell)$. If $\ell \ll k$ this will very much improve the efficiency of the Krylov-Ritz approximation.

The obvious choice for the inner Krylov subspace is $\mathcal{K}_\ell(H_k, e_1^{(k)})$. However, it is easy to see that approximations in this Krylov subspace will not improve the efficiency of the method. The Ritz matrix $H_\ell$ of the Krylov subspace $\mathcal{K}_\ell(H_k, e_1^{(k)})$ will only contain information coming from the $\ell \times \ell$ upper left corner of $H_k$, because of the tridiagonal nature of $H_k$ and the sparseness of the source vector $e_1^{(k)}$. This will effectively cut down the size of the outer Krylov subspace from $k$ to $\ell$, which will substantially worsen the accuracy of the approximation if $\ell$ is chosen much smaller than $k$. Nonetheless, the nested Krylov subspace method can be made to work efficiently if we perform an initial

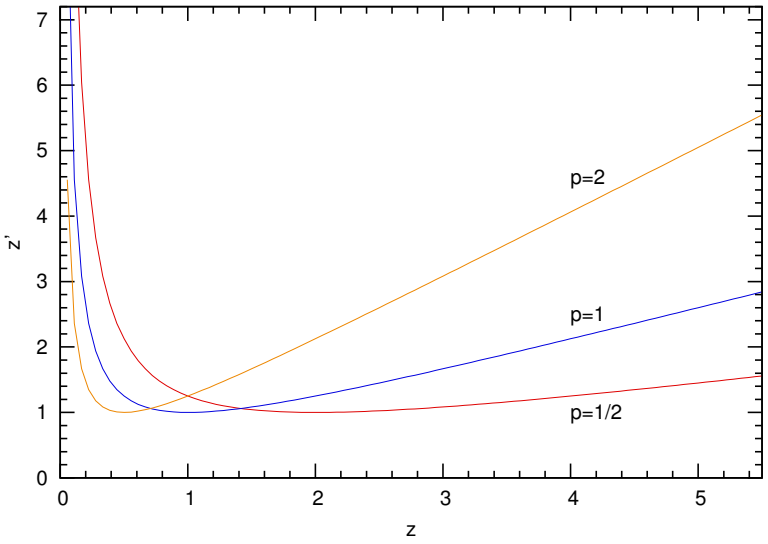



**Figure 4.3:** Mapping of the preconditioning step $z' = (pz + 1/pz)/2$ for positive real eigenvalues and various values of $p$.

preconditioning step on the tridiagonal Ritz matrix, replacing[2]

$$H_k \to H_k' = \frac{1}{2}\left[pH_k + (pH_k)^{-1}\right], \tag{4.16}$$

with $p$ a positive real number, and construct the approximation to $\mathrm{sgn}(H_k)e_1^{(k)}$ in the Krylov subspace $\mathcal{K}_\ell(H_k', e_1^{(k)})$. This alternate Krylov subspace can be used to compute $\mathrm{sgn}(H_k)e_1^{(k)}$ because the transformation leaves the sign unchanged. To show this, we note that both matrices have identical eigenvectors, as a matrix and its inverse share the same eigenvectors, and that the sign of their eigenvalues satisfies

$$\begin{aligned}\mathrm{sgn}\,\frac{1}{2}\left(pz + \frac{1}{pz}\right) &= \mathrm{sgn}\,\mathrm{Re}\left(pz + \frac{1}{pz}\right) = \mathrm{sgn}\,\mathrm{Re}\left(pz + \frac{pz^*}{|pz|^2}\right) \\ &= \mathrm{sgn}\left[\underbrace{\left(1 + \frac{1}{|pz|^2}\right)}_{>0}\mathrm{Re}(pz)\right] = \mathrm{sgn}\,(z)\,,\end{aligned} \tag{4.17}$$

where we used the definition (4.6). Hence, $\mathrm{sgn}(H_k') = \mathrm{sgn}(H_k)$ according to Eq. (4.4).[3]

As $H_k$ is tridiagonal the cost of its inversion, required in (4.16), is only of $\mathcal{O}(k)$. Moreover, as the transformation increases the relative gap between the spectrum and

[2]The factor 1/2 is chosen for convenience. For $p = 1$ the transformation actually mimics the first step of the RHi (4.13).

[3]If $H_k$ is not diagonalizable, the equality can be shown by applying Eq. (4.17) to the integration variable in the integral representation (4.3).

the singularity along the imaginary axis (see below), we expect a clear gain in efficiency for the inner Krylov-Ritz approximation, characterized by $\ell \ll k$.

For a Hermitian matrix the transformation induced by the preconditioning step is illustrated in Fig. 4.3 for real positive eigenvalues (for negative values the graph would be reflected with respect to the origin). The factor $p$ is chosen to optimize the effect of the transformation on the relative distance to the imaginary axis, which in the Hermitian case corresponds to a minimization of the condition number. We examine the condition number for the Hermitian case, assuming that the spectral support of $H_k$ is similar to that of the original matrix $A$, after deflation. As can be seen from Fig. 4.3, after transformation the smallest eigenvalue (in absolute value) is $z'_{\min} = 1$, while the largest will be given by the transform of either the smallest or largest eigenvalues of $H_k$. The smallest condition number will be achieved when both values are identical, i.e., for $p$ satisfying[4]

$$\frac{1}{2}\left(pz_{\min} + \frac{1}{pz_{\min}}\right) \stackrel{!}{=} \frac{1}{2}\left(pz_{\max} + \frac{1}{pz_{\max}}\right) \quad \Rightarrow \quad p_{\text{opt}} = \sqrt{\frac{1}{z_{\min} z_{\max}}}, \tag{4.18}$$

where $z_{\min} = \min|z|$ and $z_{\max} = \max|z|$, for $z$ in the spectrum of $H_k$, and the largest transformed eigenvalue will be

$$z'_{\max} = \frac{1}{2}\left(\sqrt{\frac{z_{\max}}{z_{\min}}} + \sqrt{\frac{z_{\min}}{z_{\max}}}\right) \approx \frac{1}{2}\sqrt{\frac{z_{\max}}{z_{\min}}}. \tag{4.19}$$

In the Hermitian case, the transformation (4.16) therefore reduces the condition number $C$ by a factor

$$\mathcal{F} = \frac{C}{C'} = \frac{z_{\max}}{z_{\min}} \Bigg/ \frac{1}{2}\left(\sqrt{\frac{z_{\max}}{z_{\min}}} + \sqrt{\frac{z_{\min}}{z_{\max}}}\right) \approx 2\sqrt{\frac{z_{\max}}{z_{\min}}}. \tag{4.20}$$

The effect of the preconditioning of the Ritz matrix for a typical spectrum of $\gamma_5 D_{\text{w}}$ in lattice QCD is illustrated in Fig. 4.4 for the Hermitian case. The top and bottom graphs depict the spectra of $H_k$ and $H'_k$, respectively. The spectrum of the original Ritz matrix has only a small gap at zero, while the gap for the transformed matrix is large. In this example, the condition number is almost improved by a factor 20. In general, the value of $z_{\max}$ for $\gamma_5 D_{\text{w}}$ varies only slightly with the choice of the simulation parameters and $\mathcal{F}$ will mainly depend on the deflation gap.

For the non-Hermitian case, let us assume that the complex spectrum is contained in the circles $C(-m, r) \cup C(m, r)$, with real center $m > 0$ and radius $r < m$. The optimal $p$, maximizing the relative distance from the imaginary axis for the transformed spectrum, is still given by Eq. (4.18) which now simplifies to $p_{\text{opt}} = (m^2 - r^2)^{-1/2}$. For this choice the transformed eigenvalues are contained in the circles $C(-m', r') \cup C(m', r')$ with center $m' = (m_s + 1/m_s)/2$ and radius $r' = (m_s - 1/m_s)/2$, where $m_s \equiv p_{\text{opt}} m$. This is illustrated in the left panel of Fig. 4.5, where we show the transformation of

[4] In practice $p_{\text{opt}}$ is only known approximately, as it is computed from spectral information of A instead of $H_k$. However, this has no significant impact on the performance of the nested method.

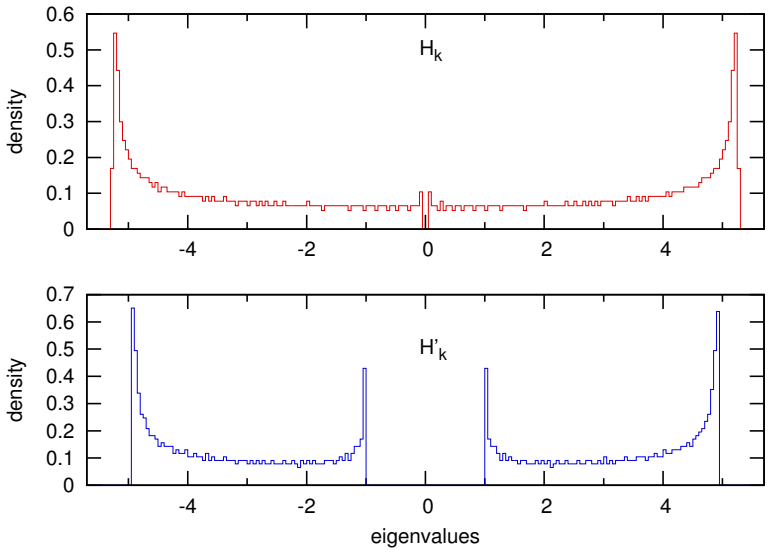



**Figure 4.4:** Upper pane: density of eigenvalues of $H_k$ in the Hermitian case for an $8^4$ lattice with $k = 1536$. The spectrum has a narrow deflation gap $\Delta = 0.055$. The optimal $p$-factor (4.18) for the transformation (4.16) is $p_{\text{opt}} \approx 1.86$ (using $z_{\min} = \Delta$ and $z_{\max} = 5.26$). The lower pane shows the corresponding eigenvalue density of the transformed matrix $H'_k$, where the condition number is improved by a factor $\mathcal{F} = 19.3$ (see Eq. (4.20)).

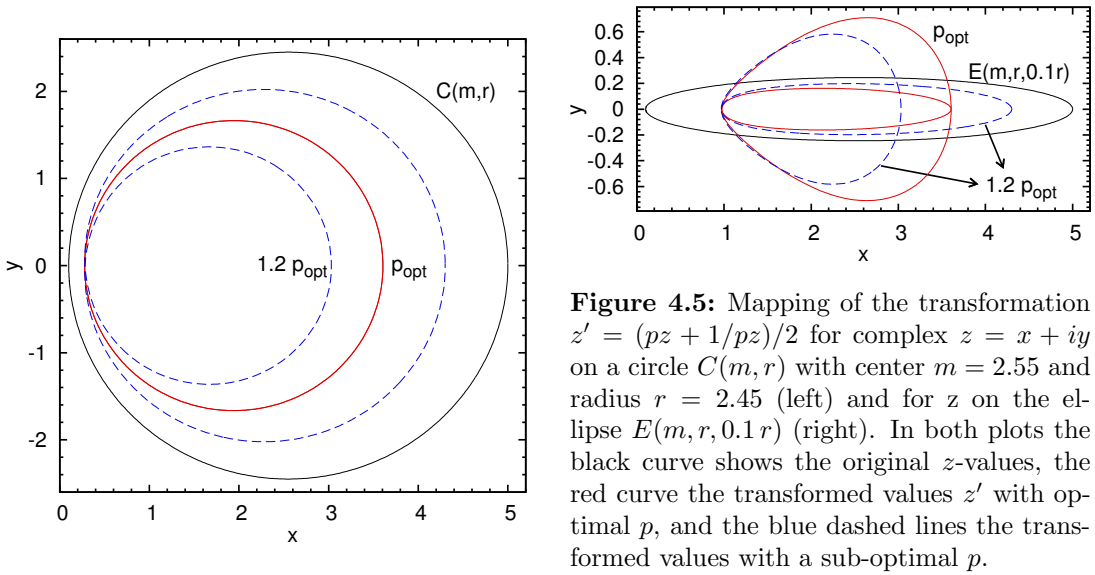



**Figure 4.5:** Mapping of the transformation $z' = (pz + 1/pz)/2$ for complex $z = x + iy$ on a circle $C(m, r)$ with center $m = 2.55$ and radius $r = 2.45$ (left) and for z on the ellipse $E(m, r, 0.1\,r)$ (right). In both plots the black curve shows the original $z$-values, the red curve the transformed values $z'$ with optimal $p$, and the blue dashed lines the transformed values with a sub-optimal $p$.

a circle $C(m, r)$ for the optimal and a sub-optimal value of $p$. For sub-optimal $p$ the transformation yields an inner and an outer circle-like contour, which merge into the circle $C(m', r')$ when $p \to p_{\text{opt}}$. For $p_{\text{opt}}$ the relative distance from the imaginary axis will be maximal and we expect the transformation (4.16) to work best. The gain in efficiency will however not be as large as for the Hermitian case. This can be quantified by the relative distance to the imaginary axis, in analogy to the calculation performed above for the Hermitian case. For the original spectrum we define the relative distance as

$$d \equiv \frac{\min |\operatorname{Re} z|}{\max |\operatorname{Re} z|} = \frac{m - r}{m + r} \tag{4.21}$$

and for the transformed spectrum

$$d' \equiv \frac{\min |\operatorname{Re} z'|}{\max |\operatorname{Re} z'|} = \frac{m' - r'}{m' + r'} = \frac{1}{m_s^2} = \frac{m^2 - r^2}{m^2}. \tag{4.22}$$

The improvement factor due to the transformation is given by the ratio of these distances, yielding

$$\mathcal{F} = \frac{d'}{d} = \left(\frac{m + r}{m}\right)^2 = \left(2 - \frac{\Delta}{m}\right)^2, \tag{4.23}$$

where we wrote $r = m - \Delta$, with $\Delta$ the deflation gap. When $\Delta \ll m$ we will have $\mathcal{F} \approx 4$. For the example shown in the left plot of Fig. 4.5 the transformation generates an improvement by a factor $\mathcal{F} = 3.84$, as computed with Eq. (4.23).

In lattice QCD at nonzero baryon density $\gamma_5 D_{\text{w}}$ is usually weakly non-Hermitian and, after deflation, the spectra are contained in ellipses $E(-m, a, b) \cup E(+m, a, b)$, with center $m \in \mathbb{R}^+$ and major and minor axes $a$ and $b$ along the real and imaginary axes, respectively. The transformation (4.16) of an ellipse $E(m, a, b)$ with aspect ratio $a/b = 10$ is illustrated in the right panel of Fig. 4.5. For such a narrow ellipse the transformed spectrum is qualitatively similar to the Hermitian case, as all the eigenvalues are transformed to the right of $z' = 1$, i.e., away from the imaginary axis, such that the high efficiency of the transformation is still guaranteed. The optimal value $p_{\text{opt}}$ is again determined by (4.18) with $p_{\text{opt}} = (m^2 - a^2)^{-1/2}$, as it maximizes the relative distance from the imaginary axis. The transformation is illustrated for a realistic test case of lattice QCD in Fig. 4.6, where the eigenvalues and transformed eigenvalues of the Ritz matrix for $\gamma_5 D_{\text{w}}(\mu)$ are shown for $\mu = 0.3$.

As we will see below the preconditioning step significantly speeds up the Krylov-Ritz approximation in its application to lattice QCD at zero and nonzero chemical potential.

### 4.4.2 Convergence

In this section we investigate the convergence properties of the nested method. The method was implemented to compute the sign function of $\gamma_5 D_{\text{w}}(\mu)$ needed by the

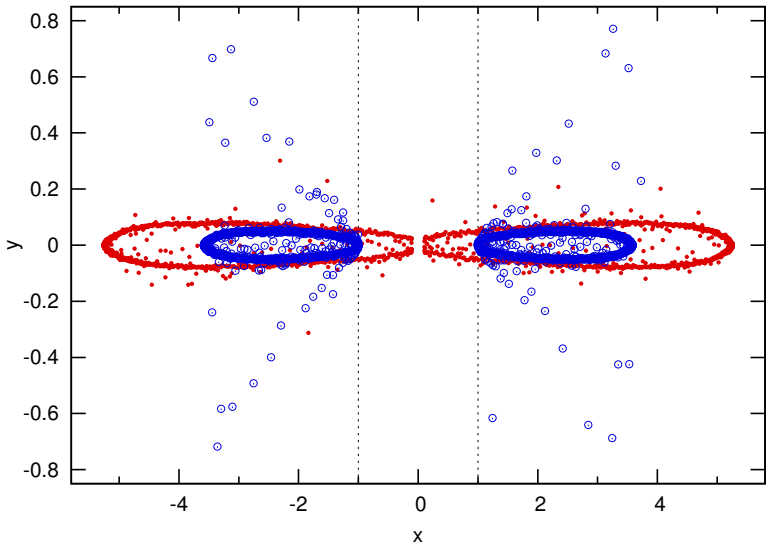



**Figure 4.6:** Red dots: spectrum of $H_k$ in the non-Hermitian case for an $8^4$ lattice with $k = 1580$, $\mu = 0.3$ and deflation gap $\Delta = 0.107$. The optimal $p$-factor (4.18) for the transformation (4.16) is $p_{\text{opt}} \approx 1.335$ (using $z_{\min} = \Delta$ and $z_{\max} = 5.243$). Blue circles: the corresponding spectrum of the transformed matrix $H'_k$. As desired, the transformed eigenvalues are well away from the imaginary axis (the vertical lines at $x = \pm 1$ serve to guide the eye). Note the different scales on the $x$ and $y$ axes.

overlap operator (4.1), for both the Hermitian and the non-Hermitian case. Whenever the matrix has eigenvalues close to the imaginary axis, these critical eigenvalues are first deflated to open up a deflation gap, necessary to keep the Krylov subspace within a reasonable size (see Sec. (4.3)). Our implementation uses Chroma [20] to compute the Wilson operator. The linear algebra is performed with BLAS and LAPACK routines. To ensure the efficiency of the nested method a judicious implementation of the preconditioning step (4.16), used to construct the inner Krylov subspace, is needed. Explicitly inverting the tridiagonal matrix $H_k$ to form the full matrix $H'_k$, then constructing the basis of the inner Krylov subspace by successive full matrix-vector multiplications would make a rather inefficient algorithm. To construct the inner Krylov subspace we do not need to construct the full matrix $H'_k$ explicitly, but only have to apply $H'_k$ to $\ell - 1$ vectors of $\mathbb{C}^k$ (in the non-Hermitian case ${H'_k}^\dagger$ is also needed). These products are best computed using the $LU$ decomposition of $H_k$, which is $\mathcal{O}(k)$ and thus especially efficient for tridiagonal matrices. A detailed listing of the algorithm is given in C.3.

The overall accuracy of the nested approximation (4.15) depends on the parameters $k$ and $\ell$, defining the sizes of the outer and inner Krylov subspaces, respectively. For $\ell \to k$ the solution of the nested method will converge to that of the non-nested method with Krylov subspace size $k$ and accuracy $\varepsilon_k$, so its total error will also converge to $\varepsilon_k$. To investigate the accuracy of the nested algorithm, our strategy is to fix the outer Krylov subspace size $k$, corresponding to a certain desired accuracy, and vary

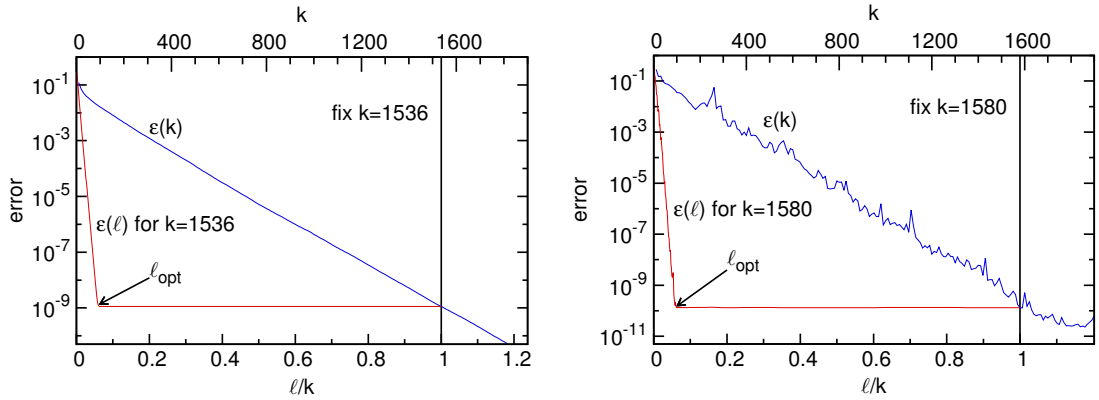



**Figure 4.7:** Accuracy $\varepsilon$ of the nested method for an $8^4$ lattice configuration. Hermitian case with deflation gap $\Delta = 0.055$ (left) and non-Hermitian case with $\mu = 0.3$ and deflation gap $\Delta = 0.107$ (right). $\varepsilon(k)$ shows how the error of the non-nested method decreases with growing Krylov subspace (blue line). The vertical line fixes the size $k$ of the outer Krylov space used in the nested method. $\varepsilon(\ell)$ shows the accuracy of the nested method, for fixed $k$, as a function of the size $\ell$ of the inner Krylov subspace (red line). The rapid convergence illustrates the efficiency of the nested method. The smallest value of $\ell$ for which optimal convergence is reached is denoted by $\ell_{\text{opt}}$. Note that we always restrict ourselves to even Krylov subspace sizes, as odd values systematically give a somewhat worse accuracy because of spurious near-zero eigenvalues occurring in the Ritz matrix.

the inner Krylov subspace size $\ell$. We show the convergence results for an $8^4$ lattice configuration in Fig. 4.7, for both the Hermitian and non-Hermitian case. As expected the nested method reaches the accuracy of the non-nested method when its size is large enough. Surprisingly however, this happens for $\ell \ll k$, as the convergence of the inner Ritz approximation seems to be extremely fast. The smallest value of $\ell$ for which optimal convergence is reached will be called $\ell_{\text{opt}}$. The fast convergence is closely related to the large improvement in condition number discussed in the previous section. We also showed in Eq. (4.20) how the improvement of the condition number, due to the preconditioning of the Ritz matrix $H_k$, depends on the deflation gap. A smaller gap will yield a larger improvement, and vice-versa. This in turn will influence the convergence rate of the nested method. Figure 4.8 verifies that the result $\ell_{\text{opt}} \ll k$ remains valid for different deflation gaps. The figure also illustrates that the somewhat larger reduction in condition number achieved for a smaller gap yields an accordingly smaller ratio $\ell_{\text{opt}}/k$ (approximately proportional to the ratio of the respective improvement factors $\mathcal{F}$). This is an additional advantage as the size reduction is largest when the outer subspace is large. In all cases, the inner Krylov subspace can be taken much smaller than the outer subspace, such that the efficiency of the Krylov-Ritz method is substantially boosted, as will be shown in the benchmarks below.

We also verified that the convergence curves are fairly insensitive to the choice of the source vector and lattice configuration. The fast convergence property of the nested method is generic, regardless of the simulation details, for both the Hermitian and non-

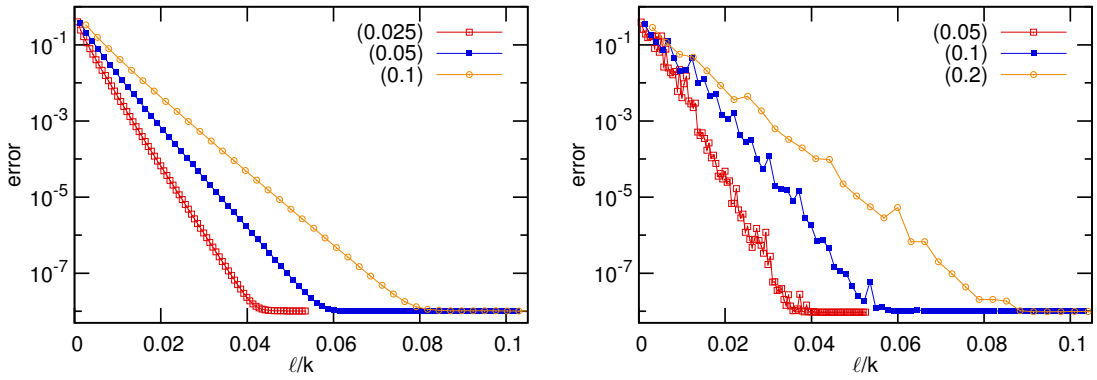



**Figure 4.8:** Convergence of the nested method for an $8^4$ lattice configuration as a function of the relative inner Krylov subspace size $\ell/k$, for various deflation gaps (given in parenthesis). For each gap the value of $k$ is chosen such that an accuracy of $10^{-8}$ is achieved. Left: Hermitian case with $k = 2806, 1462$ and $758$ for deflation gap $\Delta = 0.025, 0.05$ and $0.1$. Right: non-Hermitian case with $\mu = 0.3$ and $k = 3808, 1456$ and $634$ for $\Delta = 0.05, 0.1$ and $0.2$. Again, the irregular convergence pattern for the non-Hermitian case is characteristic for the two-sided Lanczos algorithm.

Hermitian case, even though the precise value of $\ell_{\text{opt}}$ depends on the lattice size, the simulation parameters, the deflation gap and the desired overall accuracy (determined by $k$).

### 4.4.3 Benchmarks

With the fast convergence ($\ell_{\text{opt}} \ll k$) discussed in the previous section, we can expect a substantial gain in computation time when using the nested method. The total CPU time consumed by the nested method is illustrated in Fig. 4.9 for the Hermitian case (left) and the non-Hermitian case (right). The size of the outer Krylov subspace is kept fixed, such that its construction gives a constant contribution to the run time, depicted by the horizontal dashed line. The contribution to the CPU time which varies with $\ell$ mainly comes from the computation of $\text{sgn}(H_\ell)$ with the RHi and is proportional to $\ell^3$. For $\ell \approx k$ the total run time of the nested method is about equal to that of the non-nested method. However, as illustrated by the $\varepsilon(\ell)$ curve (red line) and discussed in Sec. 4.4.2, $\ell$ can be chosen much smaller while preserving the accuracy of the non-nested method. The central result, illustrated by the vertical band in Fig. 4.9, is that there exists an interval in $\ell$ for which the accuracy is still optimal, but the CPU time needed to compute $\text{sgn}(H_\ell)$ with the RHi is negligible compared to the time required to construct the outer Krylov subspace. There is therefore no need to make a compromise between run time and accuracy, as both can be optimized simultaneously. The error in this range is the minimal error achievable with the given size of the outer Krylov subspace, while the run time is completely dominated by the cost for building the basis in that subspace. The nested method is able to quench the CPU time needed for the computation of $\text{sgn}(H_k)e_1^{(k)}$ without affecting the accuracy

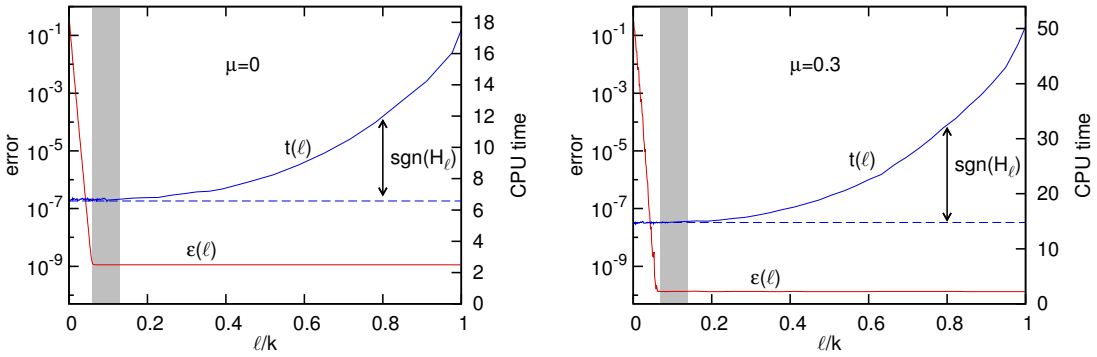



**Figure 4.9:** Error and CPU usage of the nested method for lattice size $8^4$. Hermitian case with deflation gap $\Delta = 0.055$ and fixed $k = 1536$ (left), and non-Hermitian case with $\mu = 0.3$, $\Delta = 0.107$ and $k = 1580$ (right). $\varepsilon(\ell)$ shows the accuracy versus inner Krylov subspace size $\ell$ (red line). $t(\ell)$ shows the total CPU time in seconds (solid blue line), while the horizontal dashed line measures the time needed to construct the basis in the outer Krylov subspace. The difference between both lines corresponds to the time taken by the RHi to compute $\mathrm{sgn}(H_\ell)$. The vertical band highlights the operational window of the nested method, i.e., the region in $\ell$ where the accuracy is optimal, but the CPU-time used to compute $\mathrm{sgn}(H_\ell)$ is negligible.

of the Krylov-Ritz approximation.

To evaluate the nested method further, we compare it to state-of-the-art rational approximation methods. In the Hermitian case the Zolotarev rational approximation, evaluated with a multi-shift conjugate gradient inverter [76], is routinely used in lattice simulations. In the non-Hermitian case, i.e., simulations at nonzero baryon density, overlap fermions are not yet commonly used because of their high cost, but recently an efficient algorithm was presented, which evaluates the Neuberger rational approximation using a multi-shift restarted FOM inverter [4]. In Fig. 4.10 we compare the results obtained with the nested Krylov subspace and rational approximation methods, and show how the CPU time varies as a function of the achieved accuracy for various lattice sizes. In all cases the Hermitian and non-Hermitian versions of the nested method perform better than the rational approximation method. The volume dependence of the run time for a fixed accuracy $\varepsilon$ can be extracted from Fig. 4.10 and is displayed for $\varepsilon = 10^{-8}$ in Fig. 4.11. Fits to the nested method results show a volume dependence which is slightly steeper than linear, i.e., proportional to $V^{1.2}$ for the Hermitian case and $V^{1.3}$ for the non-Hermitian case. The comparisons clearly demonstrate the good efficiency of the nested method.

### 4.4.4 Note on the memory usage

In the numerical tests we observed that, for a fixed deflation gap, the Krylov subspace size needed to achieve a certain accuracy is almost independent of the lattice volume in the Lanczos approximation and only grows slowly with the volume in the two-sided Lanczos approximation. Therefore, the memory consumed by the Krylov basis $V_k$ is roughly proportional to the lattice volume. For large lattice sizes this storage

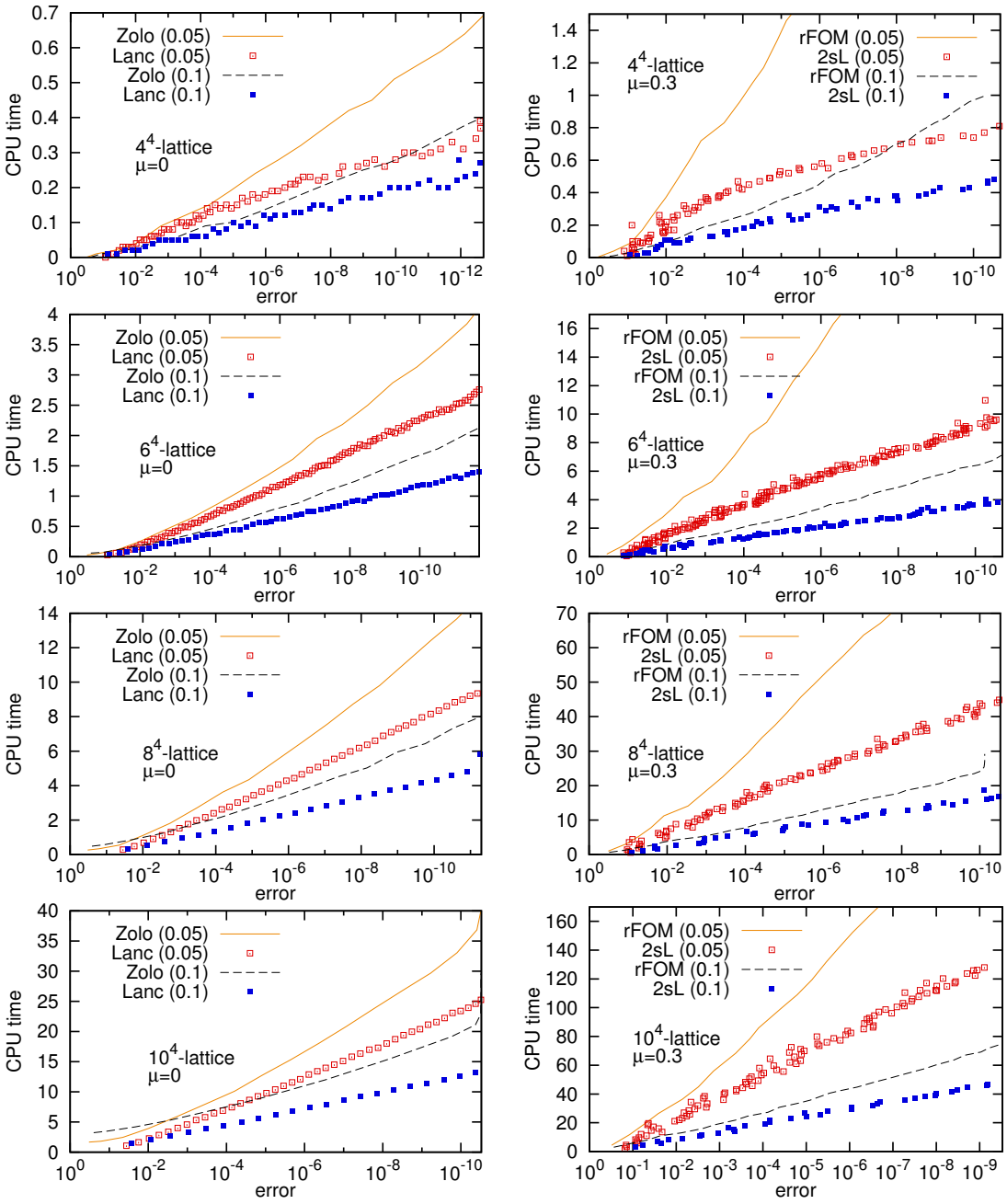



**Figure 4.10:** Comparison of the nested Krylov subspace method (squares) with rational approximation methods (lines) for lattices of sizes $4^4$, $6^4$, $8^4$ and $10^4$ for two different deflation gaps (given in parenthesis). Left: Hermitian case comparing the nested Lanczos approximation (Lanc) with the Zolotarev approximation (Zolo), evaluated using the Chroma QCD library. Right: non-Hermitian case with $\mu = 0.3$ comparing the nested two-sided Lanczos method (2sL) with the Neuberger approximation evaluated with a restarted FOM algorithm (rFOM). The timings were measured on a single 2.4 GHz Intel Core 2 core with 8 GB of memory.

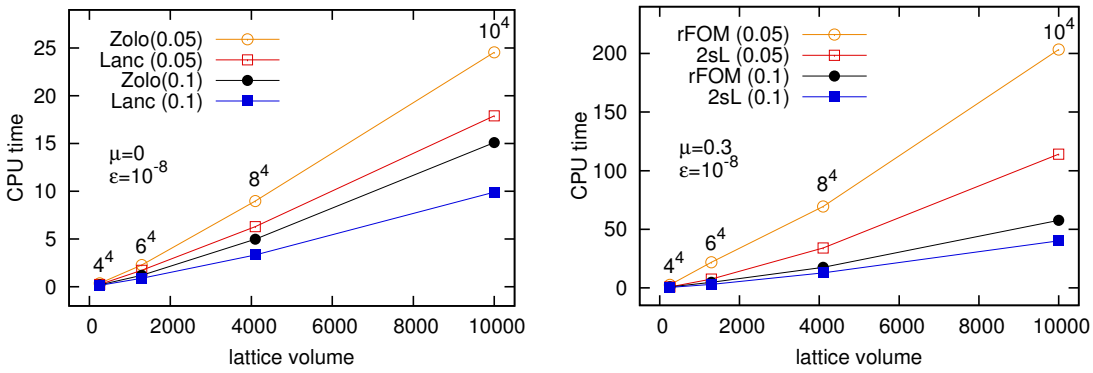



**Figure 4.11:** Volume dependence of the run times for the nested method and rational approximation methods for the Hermitian (left) and non-Hermitian (right) case. The data are taken from Fig. 4.10 at an accuracy of $\varepsilon = 10^{-8}$.

requirement might become too large to run the Krylov-Ritz approximation on a single node.

One solution, which only requires little storage, is to implement a *double-pass* version of the algorithm, which is possible due to the use of short recurrences. In double-pass mode only the two most recently generated basis vectors are stored during the construction of the outer Krylov subspace basis. In the first pass the matrix $H_k$ is built and the product $\mathrm{sgn}(H_k)e_1^{(k)}$ is computed with Eq. (4.14). In the second pass the basis vectors of the outer Krylov subspace are generated again and immediately added in a linear combination, whose coefficients were computed in the first pass. The drawback of this variant is that the Krylov basis is constructed twice, such that the corresponding CPU time will be doubled.

The more efficient solution is to parallelize the single-pass version of the algorithm, such that the memory requirement gets distributed over the available nodes. Benchmarks on larger volumes, using such a parallel implementation, are currently being performed.

### 4.4.5 Multi-level nesting

In principle, if the inner Krylov subspace in Eq. (4.15) is still too large for an efficient application of the RHi on the inner Ritz matrix, the nested method could be applied recursively.[5] In this case we rename $k$ to $k_0$, $\ell$ to $k_1$, and add more recursively nested levels $k_i$ as necessary. Except for the deepest level, the matrix-vector product $\mathrm{sgn}(H_{k_i})e_1^{(k_i)}$ required at level $i$ will be computed with a Krylov-Ritz approximation (4.14) in the nested Krylov subspace $\mathcal{K}_{k_{i+1}}(H'_{k_i}, e_1^{(k_i)})$, where $H'_{k_i}$ is defined by Eq. (4.16) on $H_{k_i}$ and typically $k_{i+1} \ll k_i$. At the deepest level the sign function of the Ritz matrix will be evaluated with the RHi. This multi-level nesting is illustrated in Fig. 4.12, where we show the convergence curves for 1, 2, 3, 4 and 5 nested levels

[5] Note that for all cases considered in the current study a single level of nesting was sufficient.

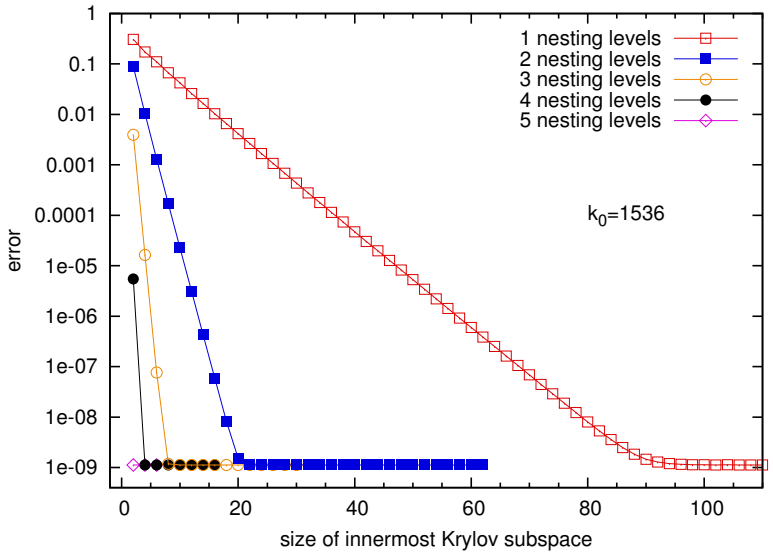



**Figure 4.12:** Accuracy $\varepsilon$ of the nested method with 1, 2, 3, 4 and 5 nesting levels for lattice size $8^4$ in the Hermitian case with deflation gap $\Delta = 0.055$ and $k_0 = 1536$. We plot the dependence of $\varepsilon$ on the size $k_i$, $i = 1, \ldots, 5$, of the innermost Krylov space. The convergence curves are labelled with the number of levels in the method. For the $i$-level method, the outer levels $k_j$, $j = 1, \ldots, i-1$ are fixed to a value in their convergence region.

as a function of the size of the innermost Krylov subspace, with the sizes of all outer levels kept fixed to some value inside their convergence region (the convergence curves do not depend on the precise choice of the outer $k_i$'s). As before, the convergence criterion is set by the size $k_0$ of the outer Krylov subspace. Each additional level lowers the size of the Krylov subspace. In the case depicted in Fig. 4.12 the optimal Krylov subspace sizes, i.e. where convergence is reached, for the successive levels decreases from $1536 \to 90 \to 20 \to 8 \to 4 \to 2$. The improvement is most dramatic for the first nested level, but fast convergence is exhibited at all levels[6]. This can be related to the quadratically convergent RHi, as the preconditioning step at each level mimics a step of the RHi and compresses the spectrum more and more towards $\pm 1$. Moreover, the judicious choice of $p$ at each level improves the convergence even more. It is intriguing to note that, in the example of Fig. 4.12, the sign of a matrix of dimension $n = 49152$ can be evaluated to an accuracy of $10^{-9}$ by computing the sign of a $2 \times 2$ matrix, which is then lifted back to the original $n$-dimensional space through linear combinations of Krylov vectors. This emphasizes again the power of Krylov subspace methods.

[6]For each level the $p$-factor for Eq. (4.16) is computed using Eq. (4.18), using appropriately approximated boundaries for the spectrum of the Ritz matrix of the previous level. Note that the factor $p$ converges to 1 as more levels are introduced, and the preconditioning step converges to the RHi.

## 4.5 Conclusions

In this paper we have presented a nested Krylov subspace method which boosts the Krylov-Ritz approximations used to compute the sign function of both Hermitian and non-Hermitian matrices. The Krylov-Ritz approximation projects the matrix on a Krylov subspace in which it computes the sign function exactly, before lifting it back to the original space. Its standard implementation suffers from the CPU intensive computation of the sign of the Ritz matrix, which goes like the cube of the Krylov subspace size. By making an additional projection on a much smaller Krylov subspace, the nested method significantly reduces the total computation time of the Krylov-Ritz approximation, without affecting its accuracy. Numerical tests showed that the nested method works equally well for Hermitian and non-Hermitian matrices and is more efficient than state-of-the-art rational approximation methods. Moreover, it exhibits a good, close to linear, volume scaling. We are currently investigating the efficiency of the nested method for larger lattice volumes using a parallel implementation of the algorithm.

To end, we comment on the relation between the nested method and the extended Krylov subspace methods introduced in Ref. [18]. An extended Krylov space is defined as

$$\mathcal{K}_k(A, A^{-1}, x) = \mathrm{span}(x, Ax, A^{-1}x, A^2x, A^{-2}x, \ldots, A^{k-1}x, A^{-k+1}x), \tag{4.24}$$

and an approximation in that subspace approximates $f(A)$ by the sum $Q(A) = \sum_{-k+1}^{k-1} c_i A^i$. In the nested method we construct the $\ell$-dimensional Krylov subspace $\mathcal{K}_\ell(H'_k, e_1^{(k)})$, which forms an $\ell$-dimensional subspace of the $(2\ell-1)$-dimensional extended Krylov subspace $\mathcal{K}_\ell(pH_k, (pH_k)^{-1}, e_1^{(k)})$. The nested method implicitly fixes the coefficients of the positive and negative powers of $Q(H_k)$ to be equal, $c_{-i} = c_i$, which follows from the use of the property $\mathrm{sgn}(H_k) = \mathrm{sgn}(H_k + H_k^{-1})$. Hence, the nested method implicitly truncates the size of the extended Krylov subspace.

Approximations for the sign function in extended Krylov subspaces have been briefly considered recently [41], however not in combination with the nesting of Krylov subspaces, i.e. the extended subspace is constructed for the original matrix $A$, not for the Ritz matrix $H_k$. Evidently this is not feasible in the application to lattice QCD as the inversion of the $\gamma_5$-Wilson Dirac operator is too expensive in order to construct extended Krylov subspaces.

To conclude, we briefly consider the application of the nested method to other matrix functions. The method presented in Sec. 4.4 requires a transformation which leaves the matrix function invariant, similar to Eq. (4.16) for the sign function. If such a transformation is not known, the nested method could be adapted by using an extended Krylov subspace method at the inner level. This is also a topic of work in progress.

*This chapter is verbatim as published in*
*Proceedings of Science,*
*Lattice 2010 (322), 2010*

# Chapter 5

# Double-pass variants for multi-shift BiCGstab($\ell$)

*In analogy to Neuberger's double-pass algorithm for the Conjugate Gradient inversion with multi-shifts we introduce a double-pass variant for BiCGstab($\ell$). One possible application is the overlap operator of QCD at nonzero chemical potential, where the kernel of the sign function is non-Hermitian. The sign function can be replaced by a partial fraction expansion, requiring multi-shift inversions. We compare the performance of the new method with other available algorithms, namely partial fraction expansions with restarted FOM inversions and the Krylov-Ritz method using nested Krylov subspaces.*

## 5.1 Introduction and Motivation

In this contribution we present double-pass variants for the multi-shift inverter BiCGstab($\ell$)[1], which, in some cases, can perform better than the conventional single-pass. The method is an analogue to Neuberger's double-pass Conjugate Gradient (CG) method [59, 11]. The use of BiCGstab($\ell$) instead of CG can be a speed advantage (for Hermitian matrices) or necessary (for non-Hermitian matrices). One possible application is the computation of quark propagators for a set of distinct masses. Here, however, we focus on computing the overlap operator of QCD. At nonzero quark chemical potential, $\mu \neq 0$, it is defined as

$$D_{\mathrm{ov}}(\mu) = \mathbb{1} + \gamma_5 \operatorname{sgn}\left(\gamma_5 D_{\mathrm{w}}(\mu)\right), \tag{5.1}$$

where $D_{\mathrm{w}}(\mu)$ is the (Wilson) Dirac operator with chemical potential.[2] For $\mu \neq 0$ the matrix $\gamma_5 D_{\mathrm{w}}(\mu)$ is non-Hermitian. One way to compute the sign function of such a matrix, acting on a given vector $b$, is via a partial fraction expansion (PFE),

$$f(A)b \approx \sum_{s=1}^{N_s} \frac{\omega_s}{A + \sigma_s} b, \tag{5.2}$$

where we are especially interested in the case of $A = (\gamma_5 D_{\mathrm{w}})^2$ with $f(A) = 1/\sqrt{A}$, since $\operatorname{sgn} z = z/\sqrt{z^2}$. The vectors $(A + \sigma_s)^{-1} b$ for a set of shifts $\{\sigma_s\}$ can be approximated

[1] $\ell$ is the degree of the minimal-residual polynomial in the algorithm
[2] See Sec. 4.2 for a definition of the Wilson operator.

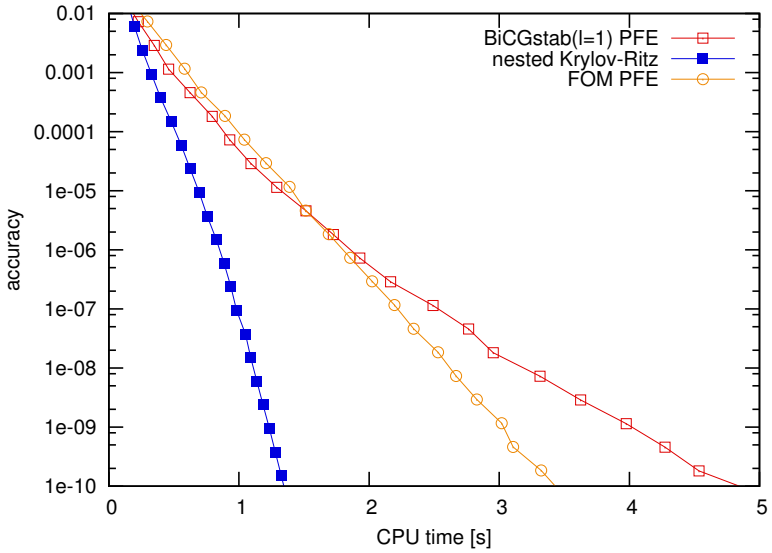



**Figure 5.1:** Accuracy versus computation time for the overlap operator on a $4^3 \times 8$ lattice, $\beta = 5.32$, $\mu = 0.05$, with the Neuberger PFE of the sign function. For each PFE data point the number of poles $N_s$ is chosen minimal for the desired accuracy, as in Ref. [4]. In this plot it ranges from 10 (accuracy 0.01) to 58 (accuracy $10^{-10}$). The 4 eigenvalues smallest in magnitude are deflated in advance.

by iterative inverters which find solutions in a Krylov subspace, defined as

$$\mathcal{K}_k(A, b) = \text{span}(b, Ab, \ldots, A^{k-1}b). \tag{5.3}$$

A crucial feature of Krylov subspaces is their shift-invariance, $\mathcal{K}_k(A+\sigma_s, b) = \mathcal{K}_k(A, b)$, which allows for so called *multi-shift inversions*, where one Krylov subspace suffices to compute $(A + \sigma_s)^{-1}b$ for a set $\{\sigma_s\}$ with little overhead per additional shift. We will refer to methods employing Eq. (5.2) as PFE methods.

## 5.2 Double-pass algorithm

As a starting point, we consider established algorithms to compute the sign function of a non-Hermitian matrix, (i) the Krylov-Ritz method with nested Krylov subspaces [7] (see Ch. 4), (ii) PFEs with FOM inversions [4], and (iii) PFEs with BiCGstab($\ell$) as inverter. The latter has so far not been considered in the context of the sign function. For details on the BiCGstab($\ell$) method see Ref. [71], a version with shifts was introduced in Ref. [23]. Benchmark results are given in Fig. 5.1. The nested Krylov-Ritz algorithm outperforms both PFE methods, which is somewhat surprising since all rely on a similar Krylov subspace.[3] We can gain more insight by analysing the bad performance of BiCGstab($\ell$) in this case:

[3] Note however that the employed single-pass (nested) Krylov-Ritz method requires a huge amount of memory.

- Denote by $N_s$ the number of shifts and by $M_s$ the number of outer iterations of the BiCGstab($\ell$) algorithm until the system with shift $\sigma_s$ is converged. Then, the multi-shift version has $\sum_{s=1}^{N_s} M_s l(0.5l + 4.5)$ axpy operations ($y \leftarrow \alpha x + y$, for scalar $\alpha$ and vectors $x$ and $y$) more than the BiCGstab($\ell$) algorithm without shifts.
- BiCGstab($\ell$) requires $2l + 5$ vectors and the multi-shift version has $N_s(l + 1)$ additional shift vectors, where typically $N_s = \mathcal{O}(10)$. These figures should be seen in relation to the typical cache size of current processors ($\mathcal{O}$(1 MByte)) and the size of a vector, e.g., 48 kByte (local volume $4^4$) or 768 kByte (local volume $8^4$) for double precision. In a typical case not all shift vectors fit into cache and the access to main memory can become the bottleneck of the algorithm.

To tackle these performance restraints one can try a *double-pass* approach in analogy to Neuberger's double-pass algorithm for a multi-shift CG inversion. Schematically the idea is as follows: the quantity computed in Eq. (5.2) and approximated in a Krylov subspace is

$$\sum_{s=1}^{N_s} \omega_s (A + \sigma_s)^{-1} b \approx \sum_{s=1}^{N_s} \omega_s \sum_{n=1}^{N} w_s^{(n)}, \tag{5.4}$$

where $N$ is the number of iterations in the inverter and $w_s^{(n)}$ is a vector for shift $s$ in iteration $n$. To remove $s$-dependent vectors one could try to swap the sums over $s$ and $n$, however $w_s^{(n)}$ is given by a recursion relation,

$$w_s^{(n)} = \alpha_s^{(n)} w_s^{(n-1)} + \beta_s^{(n)} v^{(n)} = \sum_{i=1}^{n} \gamma_{s,i}^{(n)} v^{(i)}, \tag{5.5}$$

where $v^{(n)}$ is an unshifted iteration vector. In the last step the recursion of the vectors $w_s^{(n)}$ was resolved. By combining Eqs. (5.4) and (5.5) and summing over $s$ (and $n$), all vectors depending on $s$ are removed from the algorithm. However, the coefficients $\gamma_i = \sum_{s,n} \gamma_{s,i}^{(n)}$ are not known until the end of the iteration. There are two options

1. (*double-pass*): Follow Neuberger's approach by running the algorithm once to obtain $\gamma_i$. In a second pass generate the vectors $v^{(i)}$ again and compute $\sum_i \gamma_i v^{(i)}$.
2. (*pseudo-double-pass*): Compute the coefficients $\gamma_i$ as in double-pass, but store all $v^{(i)}$ during the first pass instead of recomputing them in a second pass.

Both methods remove all $s$-dependent vectors from the algorithm and hence reduce the number of operations and the number of vectors to be held in cache. In our case, to obtain the coefficients corresponding to the (schematic) coefficients $\gamma_i$, the recursion has to be solved for the BiCGstab($\ell$) algorithm. The result is given in Sec. 5.4.

## 5.3 Cost analysis and benchmarks

The number of operations (scalar ones are omitted) and vectors of the multi-shift BiCGstab($\ell$) algorithms are given in Table 5.1 and Table 5.2. The number of vectors

| method | #Mv | #axpy | #dot-products |
|---|---|---|---|
| 1-pass | $2Ml$ | $Ml(1.5l + 5.5) + \sum_{s=1}^{N_s} M_s l(0.5l + 4.5)$ | $Ml(0.5l + 3.5)$ |
| 2-pass | $4Ml$ | $Ml(1.5l + 5.5) + Ml(1.5l + 4.5) + 2Ml$ | $Ml(0.5l + 3.5)$ |
| pseudo-2-pass | $2Ml$ | $Ml(1.5l + 5.5) + 2Ml$ | $Ml(0.5l + 3.5)$ |

**Table 5.1:** Operation count for variants of BiCGstab(ℓ). $M$ denotes the number of outer iterations of the algorithm and the dimension of the Krylov space is $2Ml$.

| method | #vectors |
|---|---|
| 1-pass | $2l + 5 + N_s(l + 1)$ |
| 2-pass | $2l + 5$ |
| pseudo-2-pass | $2l + 5 + 2Ml$ |

**Table 5.2:** Number of required vectors for variants of BiCGstab(ℓ).

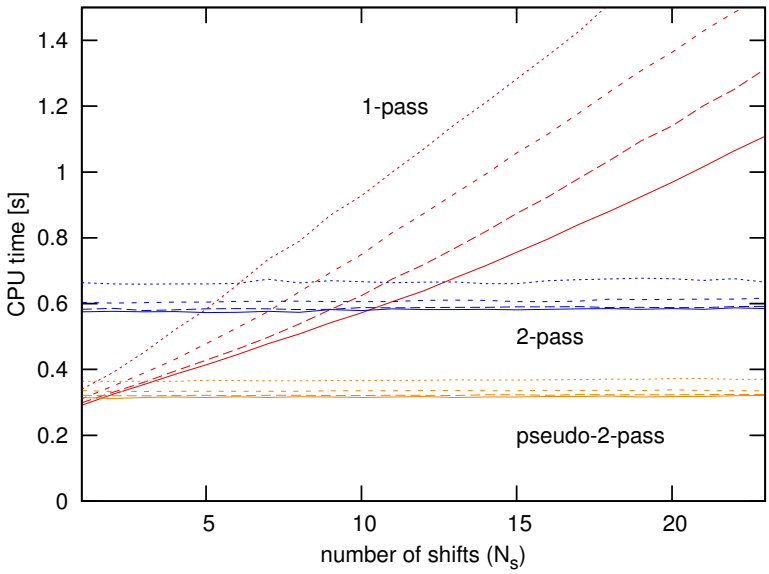



**Figure 5.2:** Computation time versus $N_s$ for fixed $Ml = 256$ for a $4^3 \times 8$ lattice (Wilson Dirac operator) for all three BiCGstab(ℓ) variants. Results are given for $l = 1, 2, 4, 8$ with lines solid to dotted.

alone is not always meaningful: in single-pass a considerable subset[4] of the $N_s(l + 1)$ vectors is accessed in each iteration of the algorithm. In pseudo-double-pass each of the $2Ml$ vectors is written and read exactly once in total. That is, the access pattern of pseudo-double-pass requires less memory access than single-pass, even though many more vectors are involved.

As a naive test of the figures given in the tables, we consider the algorithm runtime for fixed $Ml$ with a varying number of shifts, given in Fig. 5.2. The double-pass and pseudo-double-pass runtime is largely independent of $N_s$. The pure operation count

[4] depending on ℓ, and on the removal of converged systems from the iteration

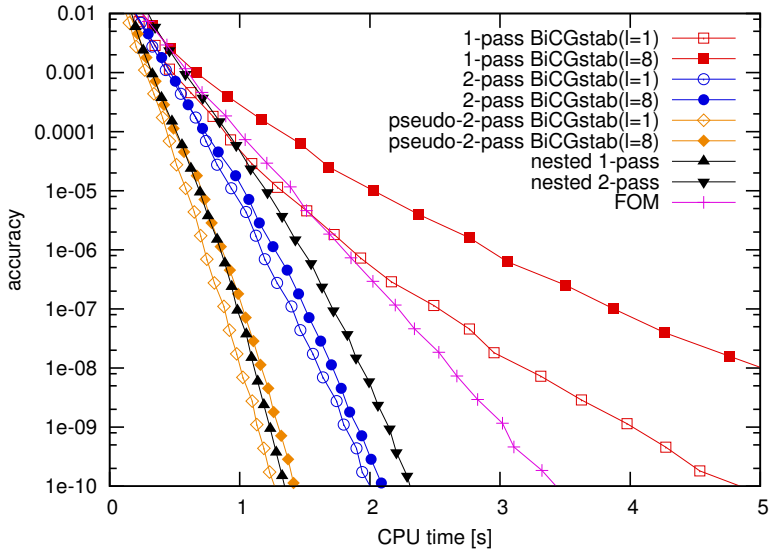



**Figure 5.3:** Accuracy versus computation time for the overlap operator ($4^3 \times 8$ lattice with $\beta = 5.32$, $\mu = 0.05$). The timings are averaged over 200 independent gauge configurations. Note that an extreme case with little deflation (4 eigenvalues smallest in magnitude) was chosen for this plot where many poles are required (as before $N_s$ is scaled from 10 to 58), yielding a large speed advantage of the double-pass algorithms. When less poles are needed (e.g., for small $\mu$ when the Zolotarev PFE can be used instead of the Neuberger expansion) the performance difference is often smaller.

of double-pass would yield an almost doubled computation time compared to pseudo-double-pass. In practice, however, it is less since no (or less) main memory access is required. An effect of the cache size can be seen from the single-pass $l = 1$ curve. The slope changes in the vicinity of $N_s = 10$, which is consistent with the cache size of 4 MByte and the size of a vector of 96 kByte. As a further observation, the relative performance loss for large $\ell$ is much smaller in the double-pass methods compared to single-pass, which is not surprising since the number of required vectors increases with $\ell$.[5]

As a more realistic benchmark, we compute the overlap operator for given configurations in Fig. 5.3. The double-pass and pseudo-double-pass BiCGstab($\ell$) algorithms perform as well as or even better than the nested double-pass and single-pass algorithms, respectively. Note that also the respective memory requirements are similar. In double-pass the performance does not degrade for $l > 1$ as it does for single pass. To explore differences between the nested Krylov-Ritz method and the BiCGstab($\ell$) methods a series of benchmarks was performed, where both the number of deflated eigenvectors and the chemical potential $\mu$ were varied. The tests indicate that BiCGstab($\ell$) profits more from deflation than the Krylov-Ritz method does. On the other hand, for large $\mu$ BiCGstab($\ell$) tends to stagnate earlier than Krylov-Ritz.

[5] Since we work at fixed $Ml$ this plot does not tell which $\ell$ is optimal, because the convergence rate depends on $\ell$.

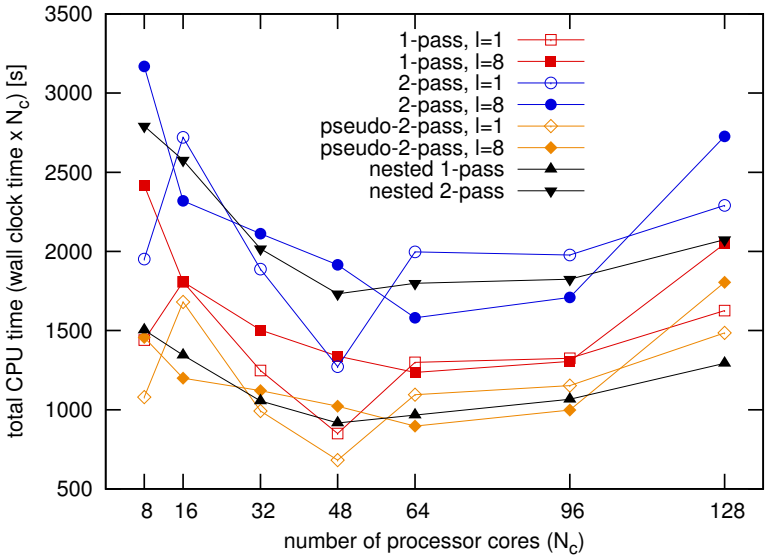



**Figure 5.4:** Total computation time versus number of cores for the nested Krylov-Ritz method and BiCGstab($\ell$) for $l = 1$ and 8. The simulation uses a $12^3 \times 24$ lattice, $\beta = 5.71$, $\mu = 0.016667$ with 44 deflated eigenvalues. The accuracy is $10^{-10}$, obtained with $N_s = 16$ shifts. The dimension of the Krylov subspace is about 2000 for BiCGstab($\ell = 1, 8$) as well as for nested Krylov-Ritz. The benchmarks where done on an Opteron 2354 Cluster (2.2 GHz, 2 quad-core processors with 16 GByte RAM per node, Infiniband network). Lines are drawn to guide the eye.

Finally, we give results of a larger-scale simulation in Fig. 5.4. As before, pseudo-double-pass BiCGstab($\ell$) is the algorithm performing best. The optimal $\ell$ depends on the number of cores $N_c$. Due to memory limitations for small $N_c$ and network limitations for large $N_c$, there is an optimal $N_c$ minimizing the total CPU time.

## 5.4 Algorithm details

We follow the notation in Ref. [23] where also a listing of BiCGstab($\ell$) is given. Upper indices $mj$ denote iteration $j$ of the BiCG part and outer iteration $m$. Define the coefficients

$$A_{mj} = \sum_s \omega_s \frac{1}{(\vartheta^s \varphi^s)^{mj}} \sum_{j'=j}^{l-1} (\alpha^s)^{mj'} \left( \prod_{k=j+1}^{j'} (-\beta^s)^{mk} \right), \tag{5.6a}$$

$$\begin{aligned} B_m^s = \sum_{n=m+1}^{M-1} & \left\{ \sum_{j=0}^{l-1} (\alpha^s)^{nj} \left( \prod_{k=0}^{j} (-\beta^s)^{nk} \right) \right\} \\ & \times \left\{ \prod_{k=m+1}^{n-1} \left( \sum_{j=0}^{l} \frac{-\gamma_j^k}{(\psi^s)^k} \sigma_s^j (-1)^{l-j} \right) \prod_{k'=0}^{l-1} (\beta^s)^{kk'} \right\}, \end{aligned} \tag{5.6b}$$

$$D^s_{mij} = \frac{-\gamma_i^m}{(\psi^s)^m} \frac{1}{(\vartheta^s \varphi^s)^{mj}} \prod_{k=j+1}^{l-1} (-\beta^s)^{mk}, \tag{5.6c}$$

$$E^s_{mj} = \frac{-1}{(\psi^s)^m} \left( \sum_{i=j+1}^{l} \gamma_i^m \sigma_s^{i-j-1} (-1)^{l-i} \right) \prod_{k=j+1}^{l-1} (\beta^s)^{mk}, \tag{5.6d}$$

$$F^s_{mj} = \frac{1 - (\alpha^s)^{mj} \sigma_s}{(\alpha^s)^{mj} (\vartheta^s \varphi^s)^{mj}}, \tag{5.6e}$$

$$G^s_{mj} = -\frac{1}{(\alpha^s)^{mj} (\vartheta^s \varphi^s_{\text{new}})^{mj}}, \tag{5.6f}$$

where $\gamma_0 = -1$, and a matrix

$$(M^m)_{jk} = -\sum_{q=k}^{j-1} \alpha^{mq} \left( \prod_{p=k+1}^{q} (-\beta^{mp}) \right), \qquad j,k = 0, \ldots, l-1. \tag{5.7}$$

Then the contribution to $\sum_s \omega_s x^s$ from the BiCG part is given by

$$\begin{aligned} \mathbf{x}_{\text{BiCG}} = \sum_{m=0}^{M-1} \sum_{j=0}^{l-1} \Bigg\{ & \sum_{p=j}^{l-1} \sum_{k=j}^{p} \left[ A_{mp} ((M^m)^j)_{pk} + \sum_{i=0}^{j} \left( \sum_s \omega_s B^s_m D^s_{mip} \right) ((M^m)^{j-i})_{pk} \right] \\ & \times \left[ (\mathbf{r}_j)^{mj} - \sum_{q=j}^{k-1} \alpha^{mq} \left( \prod_{p'=j+1}^{q} (-\beta^{mp'}) \right) (\mathbf{u}_{j+1})^{mj} \right] \\ & + \left( \sum_s \omega_s B^s_m E^s_{mj} F^s_{mj} \right) (\mathbf{r}_j)^{mj} \\ & + \left( \sum_s \omega_s B^s_m E^s_{mj} G^s_{mj} \right) \left( (\mathbf{r}_j)^{mj} - \alpha^{mj} (\mathbf{u}_{j+1})^{mj} \right) \Bigg\}. \end{aligned} \tag{5.8}$$

All operations involving the vectors $\mathbf{u}_i^s$ can be removed from the original algorithm. The final result is given by adding $\mathbf{x}_{\text{BiCG}}$ to the contributions of the seed system and the MR-part of the algorithm, $\sum_s \omega_s \mathbf{x}^s_{MR}$, which is computed trivially. For reference an implementation of the algorithms is provided online at `http://sourceforge.net/projects/bicgstabell2p/`.

## 5.5 Conclusions

We have presented an extension of the double-pass trick from Conjugate Gradient to the more general BiCGstab($\ell$). While initially PFE methods looked inferior to the nested Krylov-Ritz method in the non-Hermitian case, our new (pseudo-)double-pass BiCGstab($\ell$) is a method with similar performance. Our benchmarks concentrated on

the overlap operator where pseudo-double-pass performs as well or even better than the nested Krylov-Ritz method on the tested architectures. Current supercomputers might have enough main memory such that pseudo-double-pass is feasible, but this will depend on details of the simulation. Large values of $\ell$ yield less overhead in the double-pass methods compared to single-pass. This could boost the application of the algorithm in problems where $l > 1$ is crucial for convergence. We plan to investigate the efficiency of the double-pass BiCGstab($\ell$) algorithms for other functions aside from the inverse square root.

As a closing remark let us mention that a pseudo-double-pass method can also be used instead of the usual (double-pass) Conjugate Gradient method. This extension seems trivial, though we are not aware of any mention in the literature.

# Part III

# Hybrid Monte Carlo

# Chapter 6

# Hybrid Monte Carlo with coarse momentum field

## 6.1 Introduction

In dynamical simulations of lattice QCD, i.e., with dynamical fermions, the major cost factor is the computation of the fermionic force terms during a Hybrid Monte Carlo (HMC) trajectory. HMC [19] consists of a Hamiltonian time evolution followed by a Metropolis accept-reject step. The Hamilton equations of motion are integrated numerically for a fixed trajectory length with a finite step size. The equations of motion contain a force term derived from the action of the system, giving rise to a term called fermionic force, roughly speaking the inverse Dirac operator multiplied by a pseudo-fermion vector, which has to be computed at every time step. Techniques for these costly inversions have been covered in Part I. We will now focus on the time evolution in the HMC algorithm.

Since the number of force computations needed for the numerical integration of the Hamilton equations multiplies the large cost of the fermion matrix inversion, it is worth considering how to minimize this factor. A large error in the integration leads to low acceptance rates in the Metropolis step of the HMC, making the algorithm inefficient. Therefore it is necessary to decrease the size of the time steps until the integration error is insignificant. The situation is especially dramatic when approaching the physical point by lowering the quark masses, decreasing the lattice spacing, and increasing the volume: not only does the cost for inverting the Dirac operator increase quickly, but additionally the integration of the Hamilton equations is less stable, requiring smaller step sizes. Furthermore the autocorrelation times increase in this limit, so there is at least a triple penalty for simulating near or at parameters corresponding to nature, see for example Ref. [66]. In this chapter we investigate modifications of the HMC algorithm which can allow for a faster evolution in the configuration space, thus reducing the autocorrelation times. For readers who are unfamiliar with HMC we give an introduction to it in Sec. 6.1.1.

We briefly mention two related methods for improving the performance of HMC. The first method is multiple time-scale integration [67]. The force due to the action is split up in several terms. If the terms differ in their magnitude, one can use large steps for the small force terms and small steps for the large force terms. If the small force terms are expensive to compute this can considerably improve the performance, since

large steps for these terms results in fewer steps per trajectory. A second method is the mass preconditioning known as Hasenbusch trick [34, 33]. One artificially introduces additional force terms, rewriting the fermion determinant of, e.g., the Wilson Dirac operator $D_{\mathrm{w}}$. Consider two degenerate flavors of mass $m$ and write $M_m = D_{\mathrm{w}} + m$. Then the partition function can be written with a term $\det M_m^\dagger M_m$. The Hasenbusch trick rewrites this as

$$\det M_m^\dagger M_m = \det \left( M_{m'}^\dagger M_{m'} (M_{m'}^\dagger M_{m'})^{-1} M_m^\dagger M_m \right), \tag{6.1}$$

where we choose $m' > m$ and introduce pseudo-fermions for each term,

$$\det M_m^\dagger M_m \propto \int \mathcal{D}\chi_1 \mathcal{D}\chi_1^* \mathcal{D}\chi_2 \mathcal{D}\chi_2^* \exp(-|M_{m'}^{-1}\chi_1|^2) \exp(-|M_{m'} M_m^{-1} \chi_2|^2). \tag{6.2}$$

One can then argue that the condition number of the two new terms is improved, which allows for cheaper inversions and larger step sizes in HMC. Larger steps are possible because the force terms are smaller if the condition number is smaller. The Hasenbusch trick can be combined with multiple time-scale integration.

Aside from these two methods which are used in QCD, there are many more. In Ref. [81], the effect of linear transformations is discussed, which is similar to the modified HMC introduced in this chapter. Fourier acceleration techniques are studied in Ref. [3] and in the series of papers Refs. [15], [39], and [16], which are related to generalized HMC [22, 75]. A multigrid Monte Carlo method is studied in Ref. [31]. All these methods are problematic for gauge theories, which is without doubt a reason why they are not in widespread use in the QCD community.

The remainder of this chapter is organized as follows. After some introductory comments on HMC in Sec. 6.1.1 and autocorrelations in Sec. 6.1.2, we introduce the HMC algorithm with coarse momentum field (cm-HMC) in Sec. 6.2. Results for the $xy$-model with the cm-HMC algorithm are presented in Sec. 6.3. Finally, we give a detailed discussion on the application of cm-HMC and generalized HMC to QCD (and why it fails) in Sec. 6.4. Conclusions are given in Sec. 6.5.

### 6.1.1 Hybrid Monte Carlo

For future reference we describe the Hybrid Monte Carlo (HMC) algorithm, as proposed in Ref. [19]. We consider a theory for a real field $\phi$. This may be a quantum field theory, or a statistical physics problem. Our goal is the computation of expectation values of some observable $\mathcal{O}(\phi)$. For a thermodynamical quantum field theory with action $S(\phi)$ expectation values are computed by

$$\langle \mathcal{O}(\phi) \rangle = \frac{1}{Z} \int \mathcal{D}\phi \exp(-S(\phi)) \mathcal{O}(\phi), \tag{6.3}$$

with the partition function

$$Z = \int \mathcal{D}\phi \exp(-S(\phi)). \tag{6.4}$$

For a statistical physics system observables in the canonical ensemble are given by

$$\langle \mathcal{O}(\phi) \rangle = \frac{1}{Z} \int \mathcal{D}\phi \exp(-\beta H(\phi)) \mathcal{O}(\phi), \tag{6.5}$$

with the partition function

$$Z = \int \mathcal{D}\phi \exp(-\beta H(\phi)). \tag{6.6}$$

Since this is identical to the quantum field theory when we define $S(\phi) = \beta H(\phi)$, we can, without loss of generality, consider only the quantum field theory defined by the action $S(\phi)$. For simulations on a computer the integral in Eq. (6.3) is replaced by a finite sum over $N$ configurations and the space-time is discretized. For realistic problems the number of possible configurations is huge, so it is inevitable to use *importance sampling*, i.e., absorb the Boltzmann factor $\exp(-S(\phi))$ into the distribution of configurations,

$$\langle \mathcal{O}(\phi) \rangle \approx \sum_{n=1}^{N} \mathcal{O}(\phi_n), \tag{6.7}$$

where $\phi$ is distributed according to

$$P(\phi) \propto \exp(-S(\phi)). \tag{6.8}$$

Our task is now to generate configurations $\phi_n$ distributed according to the probability $P(\phi)$. This is done by devising a Markov process, which generates a new configuration $\phi'$ from a given configuration $\phi$ with probability $P(\phi \to \phi')$. If the algorithm for this transition fulfills certain conditions, see, e.g., Ref. [48], one can prove that the series of $\phi_n$ converges to a set of configurations distributed according to the Boltzmann distribution. Often the stronger *detailed-balance* requirement

$$P(\phi) P(\phi \to \phi') = P(\phi') P(\phi' \to \phi) \tag{6.9}$$

is used. Together with the requirement for *ergodicity* this is sufficient (but not necessary) for convergence to the fixed point distribution $P(\phi)$. Hybrid Monte Carlo is an example of an algorithm which fulfills these conditions. It is defined as follows.

Introduce an auxiliary time parameter $\tau$ and a set of conjugate momenta $\pi(\tau)$ with a Hamiltonian

$$\mathcal{H}_{\mathrm{MD}}(\phi, \pi) = \frac{1}{2}\pi^2 + S_{\mathrm{MD}}(\phi) \tag{6.10}$$

where $S_{\mathrm{MD}}$ is a priori an arbitrary action. This yields equations of motion

$$\dot{\phi} = \frac{\delta \mathcal{H}_{\mathrm{MD}}}{\delta \pi} = \pi, \tag{6.11a}$$

$$\dot{\pi} = -\frac{\delta \mathcal{H}_{\mathrm{MD}}}{\delta \phi} = -\frac{\delta S_{\mathrm{MD}}}{\delta \phi}. \tag{6.11b}$$

Usually one chooses $S_{\mathrm{MD}} = S$ yielding a special Hamiltonian $\mathcal{H}$. A new configuration $\phi'$ is generated by evolving the system $(\phi, \pi)$ for a fixed time $\tau_0$ according to the Hamilton equations. The initial momentum $\pi$ must be chosen at random from a Gaussian distribution

$$P(\pi) \propto \exp\left(\frac{\pi^2}{2}\right). \tag{6.12}$$

After the time evolution, also known as molecular dynamics, we accept the candidate configuration $\phi'$ with probability

$$P((\phi, \pi) \to (\phi', \pi')) = \min(1, \exp(-\delta\mathcal{H})), \tag{6.13}$$

with $\delta\mathcal{H} = \mathcal{H}(\phi', \pi') - \mathcal{H}(\phi, \pi)$, where it is crucial to use the special Hamiltonian $\mathcal{H}$ and *not* $\mathcal{H}_{\mathrm{MD}}$. If accepted, we add the new configuration $\phi'$ to the Markov chain, otherwise $\phi$ is added (again). This finishes one update step, and we start the next step, from the latest $\phi$ in the chain: we draw a new initial momentum $\pi$, do a time evolution of $(\phi, \pi)$, and a Metropolis step.

An analytical integration of the Hamilton equations is not possible in practice. Therefore one resorts to a numerical integration. One typical choice for an integrator is the *leapfrog* algorithm. Start by an initial half-step

$$\pi(\Delta\tau/2) = \pi(0) - \Delta\tau/2 \cdot \frac{\delta S_{\mathrm{MD}}(0)}{\delta\phi}, \tag{6.14a}$$

then do $N_{\mathrm{step}}$ steps in $\phi$ and $N_{\mathrm{step}} - 1$ steps in $\pi$ of the form

$$\phi(\tau + \Delta\tau) = \phi(\tau) + \Delta\tau \cdot \pi(\tau + \Delta\tau/2) \tag{6.14b}$$

$$\pi(\tau + \Delta\tau/2) = \pi(\tau - \Delta\tau/2) - \Delta\tau \cdot \frac{\delta S_{\mathrm{MD}}(\tau)}{\delta\phi}, \tag{6.14c}$$

and a final half-step

$$\pi(\tau_0) = \pi(\tau_0 - \Delta\tau/2) - \Delta\tau/2 \cdot \frac{\delta S_{\mathrm{MD}}(\tau_0)}{\delta\phi}, \tag{6.14d}$$

where the total covered time difference — the *trajectory length* — is given by

$$\tau_0 = \Delta\tau \cdot N_{\mathrm{step}}. \tag{6.15}$$

#### Remark on a mass parameter in the HMC Hamiltonian

From the physical point of view it is not obvious why the auxiliary Hamiltonian for HMC, Eq. (6.10), is chosen in this form. For a field $\phi$ with action $S(\phi)$ it is usually defined as

$$\mathcal{H}(\phi, \pi) = \frac{1}{2}\pi^2 + S(\phi). \tag{6.16}$$

One may ask why the kinetic term does not contain an auxiliary mass $m$. The reason is that in the HMC algorithm this is a redundant parameter. In view of the modification to HMC discussed in this chapter it is worth considering this in more detail. To see the redundancy, introduce the mass parameter in the Hamiltonian, and rename $\pi$ to $\pi'$ (since the probability distribution is modified to $\exp(-\pi'^2/2m)$ to ensure detailed balance),

$$\mathcal{H}_m(\phi_m, \pi'_m) = \frac{1}{2m}\pi'^2_m + S(\phi_m), \tag{6.17}$$

where we have added an index $m$ to $\phi$ and its conjugate momentum field to emphasize that they obey the equations of motion of $\mathcal{H}_m$,

$$\dot{\phi}_m = \frac{\delta H_m}{\delta \pi'_m} = \frac{\pi'_m}{m}, \tag{6.18a}$$

$$\dot{\pi}'_m = -\frac{\delta H_m}{\delta \phi_m} = -\frac{\delta S(\phi_m)}{\delta \phi_m}, \tag{6.18b}$$

and the corresponding equations of the leapfrog integration of these equations, with a step size $\Delta\tau'$

$$\phi_m(\tau + \Delta\tau') = \phi_m(\tau) + \pi'_m(\tau + \tfrac{1}{2}\Delta\tau')\frac{1}{m}\Delta\tau', \tag{6.19a}$$

$$\pi'_m(\tau + \tfrac{1}{2}\Delta\tau') = \pi'_m(\tau - \tfrac{1}{2}\Delta\tau') - \frac{\delta S(\tau)}{\delta \phi_m}\Delta\tau', \tag{6.19b}$$

with adequate initial and final half-steps for the momentum field. Now define $\Delta\tau = \Delta\tau'/\sqrt{m}$ and $\pi_m = \pi'_m/\sqrt{m}$. We obtain

$$\phi_m(\tau + \sqrt{m}\Delta\tau) = \phi_m(\tau) + \pi_m(\tau + \tfrac{1}{2}\sqrt{m}\Delta\tau)\Delta\tau, \tag{6.20a}$$

$$\pi_m(\tau + \tfrac{1}{2}\sqrt{m}\Delta\tau) = \pi_m(\tau - \tfrac{1}{2}\sqrt{m}\Delta\tau) - \frac{\delta S(\tau)}{\delta \phi_m}\Delta\tau. \tag{6.20b}$$

We see that these equations are equivalent to those obtained from the original Hamiltonian if we identify $\phi_m(n \cdot \sqrt{m}\Delta\tau) = \phi_m(n \cdot \Delta\tau')$ with $\phi(n \cdot \Delta\tau)$ and similarly for $\pi_m$ and $\pi$. To conclude, the integration of the equations of motion for $H_m$ (Eq. (6.18)) with the two parameters $\{\Delta\tau', m\}$ is equivalent to the integration of the equations of motion of the original Hamiltonian (Eq. (6.14)) with one parameter $\Delta\tau = \Delta\tau'/\sqrt{m}$. That is, giving a large mass to the field[1] corresponds to choosing smaller time steps in the leapfrog integration.

### 6.1.2 Autocorrelations

The correlations in a set of measurements of an observable $A$ can be quantified by autocorrelation functions. A rather general analysis of autocorrelations can be found,

[1] This is of course unrelated to any physical mass of the field. We mean the effective mass of the field in the auxiliary Hamiltonian time evolution.

e.g., in Ref. [80]. Here we only introduce the most common basics, which are sufficient for our purposes. The autocovariance function is given by

$$C_A(t) \equiv \left\langle \big(A(t') - \langle A\rangle\big)\,\big(A(t'+t) - \langle A\rangle\big)\right\rangle = \left\langle A(t')A(t'+t)\right\rangle - \langle A\rangle^2, \tag{6.21}$$

for (discrete) time differences $t$ and an arbitrary time $t'$ in a Markov chain, where $\langle . \rangle$ denotes the mean. In practice the true mean is not accessible, so it is approximated by the average over the $N$ values in the set of measurements,

$$C_A(t) \approx \frac{1}{N-t}\sum_{t'=0}^{N-t} A(t')A(t'+t) - \left(\frac{1}{N-t}\sum_{t'=0}^{N-t} A(t')\right)\left(\frac{1}{N-t}\sum_{t'=0}^{N-t} A(t'+t)\right). \tag{6.22}$$

The autocorrelation function is the normalized autocovariance function,

$$\rho_A(t) \equiv \frac{C_A(t)}{C_A(0)}, \tag{6.23}$$

where

$$C_A(0) = \sigma_A^2. \tag{6.24}$$

The *integrated autocorrelation time* for the observable $A$ is then defined by

$$\tau_{\mathrm{int},A} = \frac{1}{2} + \sum_{t=1}^{t_{\mathrm{cutoff}}} \rho_A(t), \tag{6.25}$$

where $t_{\mathrm{cutoff}}$ is a suitable maximal correlation time taken into account. In praxis this is chosen to neglect noise effects from the autocovariance function for large times $t$. Due to autocorrelations the effective number of independent measurements in the initial set of $N$ measurements is $N/(2\tau_{\mathrm{int},A})$ [49].

We want to study in detail the autocorrelation times for different algorithms or depending on certain parameters. For a meaningful analysis this requires error estimates for autocorrelation times or, in other words, errors of the errors on the observables. A method which is suitable for use in practice is given in Ref. [80]. Throughout this chapter we use this method via the analysis software provided with this publication.

It has been shown, e.g., in Ref. [51], that the autocorrelation time decreases with increased trajectory length.[2] To avoid confusions, we should point out that this is not meant measurement-wise: a standard analysis code, analysing a series of $N$ consecutive measurements would return an autocorrelation time in units of measurements, since the underlying trajectory length may not be known to the software or the person doing the data analysis. This is however not what really matters when considering the cost of generating the configurations. Longer trajectories cost more, since more steps have to be taken when integrating the equations of motions of HMC numerically. Therefore,

[2] Usually one trajectory corresponds to one measurement.

autocorrelations should be measured in the same units as the time parameter $\tau$ in a HMC trajectory. When optimizing the trajectory length for an HMC run, one should consider $\tau_{\text{int}}$ in the same time units as the trajectory length, assuming the same step size,

$$\tau_{\text{int}} = \tau_0 \left[ \frac{1}{2} + \sum_{t=1}^{t_{\text{cutoff}}} \rho_A(t) \right], \tag{6.26}$$

where $\tau_0$ is the trajectory length. For our purposes of studying HMC this is the definition we will use from now on, unless stated otherwise. Note that we do not need the factor $\tau_0$ when computing the error on the expectation value of $A$. The next complication is that one can use different step sizes in different simulations. If the trajectory length is the same, the number of steps will differ and therefore the cost will also differ. In simulations with dynamical fermions we need one inversion per step and one additional inversion at the end of the trajectory. Since typically there are many steps the cost of the trajectory is roughly proportional to the number of steps. For comparing the cost of HMC runs with arbitrary parameters it is useful to define an integrated autocorrelation time in units of steps,

$$\tau_{\text{int}}^{\text{step}} = N_{\text{step}} \left[ \frac{1}{2} + \sum_{t=1}^{t_{\text{cutoff}}} \rho_A(t) \right], \tag{6.27}$$

where $N_{\text{step}}$ is the number of steps per trajectory.

Praxis shows that $\tau_{\text{int}}$ goes down with increasing trajectory length. This positive effect does not continue indefinitely. The reason is not entirely clear to us, but we try to briefly sketch an explanation: when comparing to a Metropolis algorithm, the shorter autocorrelations in HMC are based on the *directed movement* instead of the random-walk behavior [56]. For large evolution times the trajectories will eventually turn back and return to the starting point (see Sec. 6.2.2 for an example with illustrating figures), however this happens only in systems with low dimensionality. For high-dimensional systems, such as a field theory, this does not happen.[3] However, even if trajectories do not "turn back", we still reach a point where this directed movement does not significantly increase the distance to the starting point, or decrease the autocorrelations.[4] An intuitive picture could be that the trajectory might "thermalize"

[3] Due to finite time and finite-precision arithmetics.

[4] It is hard to quantify this. One option is given in Ref. [37], where the dot-product between the current momentum and the vector from the start of the trajectory to the current position is used. Our experiments with the $xy$-model showed that this quantity declines quickly towards zero after a peak at medium trajectory length. However, it never gets smaller than zero for the trajectory lengths we are able to handle on the computer, which means that the trajectory does not loop back and return to the starting point. That is, after a certain molecular-dynamics time has passed, the distance of the new (current) configuration from the old configuration neither increases nor decreases (where we have to mention that this naive measure of distance is not necessarily closely related to autocorrelations of observables). We do not understand this behavior and it thus requires further investigation.

in the noisy background potential (the action), and we return to a random-walk-like behavior, losing the benefits of HMC.

A second reason, which might in some cases explain why $\tau_{\text{int}}$ grows for too long trajectories, is the finite precision of calculation on computers. Consider for example solving the Hamilton equations numerically with a leapfrog scheme. In exact arithmetics the error of the numerical integration is bounded independently of the trajectory length for a given step size. On a computer this bound could be violated if the integration time $\tau$ is too large and the errors will grow rapidly with increasing $\tau$. As a consequence, the acceptance rate decreases, which in turn increases $\tau_{\text{int}}$. However, in our experience this does happen only for extremely long trajectories, far beyond the optimal trajectory length found in practice.

This section motivates a search for optimization: if we manage to reduce the errors in the numerical integration, we can use larger steps, which ultimately means reduced error bars on our observables, as we can construct a longer Markov chain at the same cost.

## 6.2 Hybrid Monte Carlo with coarse momentum field

### 6.2.1 Coarse momentum field with modified mass

We introduce a modification of the HMC algorithm which belongs to the class of linear transformations. A pedagogical introduction to the effect of linear transformations on the performance of HMC is given in Ref. [81]. Following the discussion given there, suppose we have an estimate $\Sigma$ for the covariance matrix of $\phi$, where $\phi$ is roughly Gaussian distributed according to $S(\phi)$. One can then transform the variables such that their covariance matrix is close to the identity (i.e., their distribution is roughly rotationally invariant), by putting $\phi' = L^{-1}\phi$, where $L$ is a lower triangular factor of the Cholesky decomposition of the covariance matrix, $\Sigma = LL^T$. With the auxiliary momenta $\pi$ for the transformed variables $\phi'$, the kinetic energy term $T(\pi) = \pi^2/2$ yields good HMC performance. The alternative, equivalent, way is to keep $\phi$ but instead use a different kinetic term, $T(\pi) = \pi^T\Sigma\pi/2$. This means the momentum variables now have covariance $\Sigma^{-1}$. On small model problems this method yields close to optimal performance of HMC (see Sec. 6.2.2 for an example and an explanation). In a real problem however, the dimension of $\Sigma$ is too large to handle. That is, we cannot explicitly find $\Sigma$, or if we can, dealing with it in the equations of motion is infeasible. Often an approximation can be sufficient to yield considerable performance improvements.

We opt for changing the kinetic term, since it is simpler and has a similar structure for all systems we might want to simulate. We introduce a variable transformation which is local and cheap, so it is feasible for parallel algorithms.[5] Additionally it uses some — albeit rather trivial — physical insight: it employs blocks of the lattice, arguing (or hoping) that variables that are far apart couple more weakly. This is

[5] Memory locality is crucial for high-performance computing.

accomplished by a coarse grid, formed by the mean values of small hypercubic sub-blocks of the lattice.

To introduce our method, we start with an example of a one-dimensional system with $N$ real degrees of freedom $\phi_n$, $n = 0, \ldots, N-1$, with $N$ even, governed by some action $S(\phi)$. The usual kinetic term for HMC is given by

$$T(\pi) = \frac{1}{2}\pi^T\pi. \tag{6.28}$$

Considering the momenta of two neighboring points, $\pi_{2n}$ and $\pi_{2n+1}$, we define a *coarse* vector $\pi_c$ with $\pi_{c,n} = (\pi_{2n} + \pi_{2n+1})/\sqrt{2}$ and a *fine* vector $\pi_f$ with $\pi_{f,n} = (\pi_{2n} - \pi_{2n+1})/\sqrt{2}$ for $n = 0, \ldots, N/2-1$, which corresponds to considering the system as being built from blocks of two points each. The coarse vector describes the mean values (or centers of mass) of the blocks and the fine vector the differences within each block. The kinetic energy for the two points in a block is

$$\begin{aligned} T(\pi_{2n}, \pi_{2n+1}) &= \frac{\pi_{2n}^2}{2} + \frac{\pi_{2n+1}^2}{2} \\ &= \frac{1}{2}\left[\left(\frac{1}{\sqrt{2}}(\pi_{c,n} + \pi_{f,n})\right)^2 + \left(\frac{1}{\sqrt{2}}(\pi_{c,n} - \pi_{f,n})\right)^2\right] \\ &= \frac{1}{2}\left[\frac{1}{2}(\pi_{c,n}^2 + \pi_{f,n}^2 + 2\pi_{c,n}\pi_{f,n}) + \frac{1}{2}(\pi_{c,n}^2 + \pi_{f,n}^2 - 2\pi_{c,n}\pi_{f,n})\right] \\ &= \frac{\pi_{c,n}^2}{2} + \frac{\pi_{f,n}^2}{2}. \end{aligned} \tag{6.29}$$

We see that the local quadratic kinetic energy for the two sites can be rewritten in terms of the mean values of the field and the difference of the fields, where both are quadratic as well. In fact this is simply an orthogonal basis rotation of the variables $\pi_{2n}, \pi_{2n+1}$. The total kinetic energy in this formulation is

$$T(\pi) = \frac{\pi_c^T\pi_c}{2} + \frac{\pi_f^T\pi_f}{2}. \tag{6.30}$$

The main step of our algorithm is to adjust the masses of the transformed degrees of freedom, $\pi_c$ and $\pi_f$, independently. We modify the kinetic term,

$$T'(\pi) = \frac{\pi_c^T\pi_c}{2m_c} + \frac{\pi_f^T\pi_f}{2m_f}, \tag{6.31}$$

where $m_c = m_f = 1$ corresponds to the original $T$. In the original basis this kinetic term is not diagonal,

$$T'(\pi) = \sum_{n=0}^{N/2-1}\left[\frac{m_c + m_f}{4m_cm_f}(\pi_{2n}^2 + \pi_{2n+1}^2) + \frac{m_f - m_c}{2m_cm_f}(\pi_{2n}\pi_{2n+1})\right]. \tag{6.32}$$

By choosing the mass $m_c \neq m_f$, we can achieve a different behavior during the time evolution, where the coarse degrees of freedom react more strongly or more weakly to the forces due to the potential. As discussed in Sec. 6.1.1, the mass parameter is a redundant parameter and simply yields a scaling of the step size. Consequently, we can fix one of the two new mass parameters to an arbitrary value, and thus put $m_f = 1$, which leaves us with two free parameters $m_c$ and $\Delta\tau$ (apart from the trajectory length). Since $\pi_c$ describes the center-of-mass momenta of the blocks of lattice points (up to $\sqrt{2}$), it gives a coarsened description of our system.

Typically the low-energy contributions to a system are the most problematic ones for an HMC time evolution, i.e., they yield the longest autocorrelations [66]. Often there is a (loose) connection between low-energy modes and low-frequency modes. Low-frequency modes can be well described by the coarse grid. Via $m_c$ we can now explicitly control the HMC time evolution of these low-frequency contributions and speed it up, which hopefully results in reduced autocorrelation times. This is only possible due to the scale separation we obtain by the block transformation. Without the transformation a change of the HMC parameters will affect all frequencies in the same way. Given that the HMC trajectory is a classical time evolution in some potential, we see that lighter fields will be affected more by the Hamiltonian force. Therefore, choosing $m_c < 1$ should yield a stronger effect of HMC on coarse scales. Alternatively, we can arrive at the same conclusion by recalling the discussion of the mass dependence of the trajectory length in Sec. 6.1.1 — a smaller mass will yield a larger effective step size.

The time evolution follows the now modified equations of motion,

$$\dot{\phi}_n = \frac{\delta H'}{\delta \pi_n} = \frac{1}{\sqrt{2}} \frac{\pi_{c,n/2}}{m_c} + \frac{(-1)^n}{\sqrt{2}} \frac{\pi_{f,n/2}}{m_f}, \tag{6.33a}$$

$$\dot{\pi}_n = -\frac{\delta H'}{\delta \phi_n} = -\frac{\delta S(\phi)}{\delta \phi_n}, \tag{6.33b}$$

where only the equation for $\phi$ has changed, and collapses to the usual $\dot{\phi} = \pi$ for $m_c = m_f = 1$. The indices $n/2$ for $\pi_c$ and $\pi_f$ are rounded down to the nearest integer. In order to satisfy detailed balance, a change of the Hamiltonian requires to change also the momentum distribution from which the momenta are drawn at the beginning of each trajectory. The distribution is now given by

$$P'(\pi) = \exp(-T'(\pi)) = \exp\left(-\frac{\pi_c^T \pi_c}{2m_c} - \frac{\pi_f^T \pi_f}{2m_f}\right). \tag{6.34}$$

Generating random numbers according to this distribution is unproblematic, since we can draw random numbers from a Gaussian distribution in the transformed basis and then do a reverse transformation to go back to the original basis. From now on we name this algorithm with modified Hamiltonian and momentum probability distribution the *coarse-momentum Hybrid Monte Carlo* (cm-HMC).

For completeness, we can write down a more general formulation, using a general Gaussian, which is equivalent to the linear transformations described in Ref. [56].

Defining the mass matrix $M$, we can write the down the usual kinetic term $T$ and the momentum field distribution $P$ for a vector of momenta $\pi$,

$$M = \mathrm{diag}\left(\frac{1}{m}, \ldots, \frac{1}{m}\right), \tag{6.35a}$$

$$T(\pi) = \frac{1}{2}\pi^T M \pi, \tag{6.35b}$$

$$P(\pi) = \exp(-T(\pi)) = \exp(-\tfrac{1}{2}\pi^T M \pi). \tag{6.35c}$$

Since $P$ is simply an exponential of $T$, we work only with $P$ in the following derivation. We insert an orthogonal rotation $Q$, and use the fact that $M$ is proportional to the identity matrix,

$$\begin{aligned} P(\pi) &= \exp(-\tfrac{1}{2}\pi^T Q^T Q M Q^T Q \pi) \\ &= \exp(-\tfrac{1}{2}\pi^T Q^T M Q \pi) \\ &= \exp(-\tfrac{1}{2}\pi'^T M \pi'), \end{aligned} \tag{6.36}$$

where $\pi' = Q\pi$. We want to modify masses in the new basis. Here it is just a modification on the diagonal of $M$, but if we transform back to the original basis, it is not. For example, put

$$M' = \mathrm{diag}\left(\frac{1}{m_c}, \ldots, \frac{1}{m_c}, \frac{1}{m} \ldots, \frac{1}{m}\right), \tag{6.37}$$

to give some of the degrees of freedom in the new basis a different mass. Then define a new momentum distribution,

$$\begin{aligned} P'(\pi) &= \exp(-\tfrac{1}{2}\pi'^T M' \pi') \\ &= \exp(-\tfrac{1}{2}\pi'^T Q Q^T M' Q Q^T \pi') \\ &= \exp(-\tfrac{1}{2}\pi^T (Q^T M' Q) \pi), \end{aligned} \tag{6.38}$$

where it is important to note that $Q^T M' Q$ is *not diagonal*. A priori $Q$ is an arbitrary orthogonal transformation, but of course practicality of the method puts strong limitations. In the following we will restrict ourselves to simple blocking transformations as in Eq. (6.31) and higher-dimensional analogues. Comparing with the linear transformations discussed in the beginning of this section, we see that our method is equivalent to it if we put $\Sigma = Q^T M' Q$. Therefore cm-HMC is a special case of a known method, however we are not aware of a systematic study or use of this or related transformations. In the following we will show the feasibility of the method and study its benefits.

### 6.2.2 Coarse-momentum HMC showcase

For illustration we discuss a trivial example, similar to one given in Ref. [56], where the introduced modification of the momentum field can yield optimal results. Consider a

system with two degrees of freedom, $\phi_x$ and $\phi_y$, summarized in a vector $\phi = (\phi_x, \phi_y)^T$. We choose a quadratic but correlated energy for the system. To define the energy we introduce two new variables,

$$\phi_c = \frac{1}{\sqrt{2}}(\phi_x + \phi_y), \tag{6.39a}$$

$$\phi_f = \frac{1}{\sqrt{2}}(\phi_x - \phi_y). \tag{6.39b}$$

Then we define the action

$$S(\phi) = \frac{\phi_c^2}{2\sigma_c^2} + \frac{\phi_f^2}{2\sigma_f^2}, \tag{6.40}$$

where for the following we put $\sigma_c = 10$ and $\sigma_f = 1$. The probability for a field configuration $\phi$ is then

$$P(\phi) \propto \exp(-S(\phi)), \tag{6.41}$$

and we can use HMC to obtain an ensemble of such configurations. As usual, the acceptance rate decreases if $\Delta\tau$ is too large and quickly goes to zero, independently of the trajectory length. The acceptance rate is plotted in Fig. 6.1. In this simple system we can directly understand why this happens, by looking at Fig. 6.2. The system can be imagined as a mass point in an asymmetric two-dimensional potential well. In HMC we integrate the trajectory numerically, starting at a given point with a nonzero step size. The plot shows two trajectories with the same set of initial field values ($\phi$ and the corresponding auxiliary momentum $\pi$), but with different step sizes. For the larger step size the trajectory is erratic and actually far from the more precise result with smaller step size. Due to the Metropolis step, HMC can still deal with this, but if we choose $\Delta\tau$ only a little larger the accept rate is too low. We now gain insight into two crucial points. First, the maximal feasible step size is given by the most constrained direction of the potential: the trajectory has to perform quick oscillations in this direction. Second, from the figure we can estimate the autocorrelation time. For example, the correlation of $\phi_x$ is of the order of the time it takes for a trajectory to get from one end of the elongated potential well to the other end. These two points are in conflict: the most constrained direction of the potential fixes a maximal step size and the least constrained direction fixes a minimal trajectory length for uncorrelated (or weakly correlated) measurements. If the ratio of these two scales is small we have to pay a lot of computing time for long trajectories with small steps.

In a simple case like this there is a solution for this problem by transforming the $\phi$ variables with the lower triangular factor of the Cholesky decomposition of the covariance matrix in the action, as suggested in Ref. [56]. The block modification of the momentum field — when putting $m_c \neq 1$ in the cm-HMC algorithm — is equivalent, but takes a different viewpoint. By construction of our model problem the coarse degree of freedom $\phi_c$, corresponding to the coarse momentum field component $\pi_c$, is the least constrained direction of the potential. Changing $m_c$ changes the width of

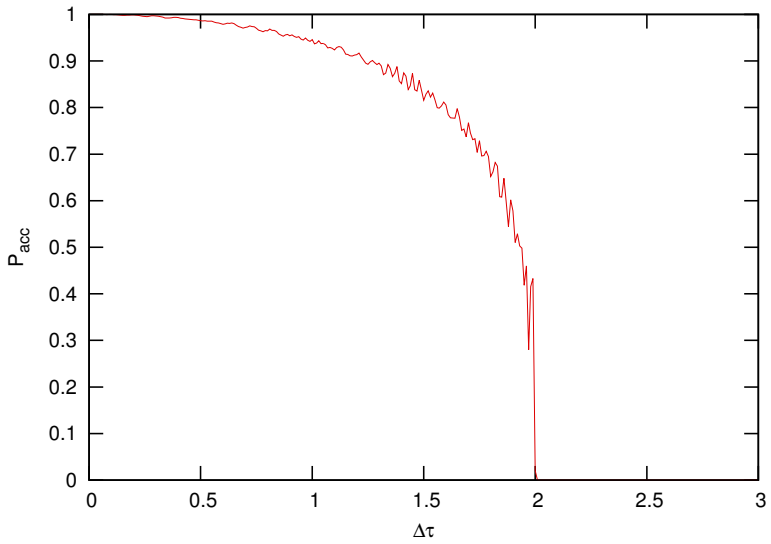



**Figure 6.1:** Acceptance rate

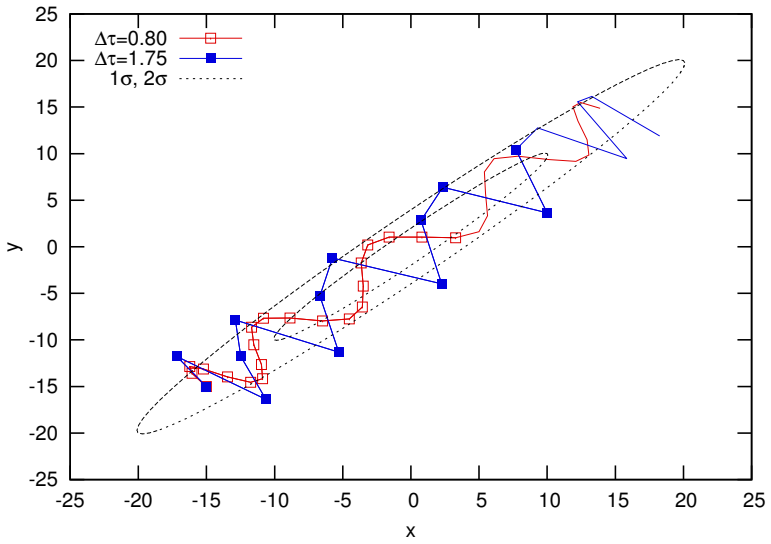



**Figure 6.2:** Examples for a trajectory for two different step sizes. The section of the trajectory drawn with symbols (empty or filled squares) indicates the (measured) trajectory length required for minimal $\tau_{\mathrm{int}}^{\mathrm{step}}$.

the momentum distribution in this direction and, importantly, also the corresponding equation of motion. A small $m_c$ leads to larger steps respectively longer trajectories. In Fig. 6.3 we see that the step size along the short axis is unchanged, compared to Fig. 6.2. The motion along the long axis is much quicker now — due to $m_c = 0.1$ the effective step size is larger by a factor $\sqrt{0.1}$. Since the potential is very wide along this axis this does *not* lead to large integration errors, as it would along the short axis.

In this model problem there is a optimal choice for the mass parameter, $m_c = 1/\sigma_c^2 = 0.01$. With this modified mass our particle experiences the potential as spherically symmetric, which leads to trajectories which are ellipses, pictured in Fig. 6.4. Due to the symmetry we can have large steps with short trajectories, which leads to minimized autocorrelation times. This can be understood in terms of the general formulation at the end of Sec. 6.2.1. Defining

$$Q = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}, \tag{6.42}$$

and

$$\begin{pmatrix} \phi_c \\ \phi_f \end{pmatrix} = Q \cdot \begin{pmatrix} \phi_x \\ \phi_y \end{pmatrix}, \tag{6.43}$$

we see that the energy $E(\phi)$ can be written as

$$E(\phi) = \frac{1}{2}\phi^T Q^T M Q \phi = \frac{1}{2}\phi^T \Sigma \phi, \tag{6.44}$$

with the covariance matrix $\Sigma = Q^T M Q$, where $M = \mathrm{diag}(1/\sigma_c^2, 1/\sigma_f^2)$. That is, we can transform the momenta with $Q$ and then use a kinetic term with $M$ as the covariance matrix. This yields the aforementioned optimal choice for the mass parameter, $m_c = 1/\sigma_c^2 = 0.01$, which is just one of the diagonal terms of the matrix $M$.

### 6.2.3 A model problem: the $xy$-model

We introduce the classical $xy$-model, which we will use for testing the cm-HMC algorithm.[6] It describes a system of $N$ interacting classical spins. Each spin is described by an angle. The interaction energy between two spins $\vec{s}_x$ and $\vec{s}_y$ depends on $\vec{s}_x \cdot \vec{s}_y$. When the system is placed in a background field there will be an additional energy term proportional to the cosine of the angle of each spin. The Hamiltonian is given by

$$H(\theta_1, \ldots, \theta_N) = -\sum_{i \neq j} J_{ij} \cos(\theta_i - \theta_j) - \sum_j h_j \cos(\theta_j). \tag{6.45}$$

Here $h_j$ is the strength of the magnetic field at spin $j$. $J_{ij}$ is the strength of the interaction between spins $i$ and $j$. The arrangement of spins can be arbitrary, but usually one considers a hypercubic arrangement in $D$ dimensions (which can be closed

[6] Note that good cluster algorithms exists for the $xy$-model.

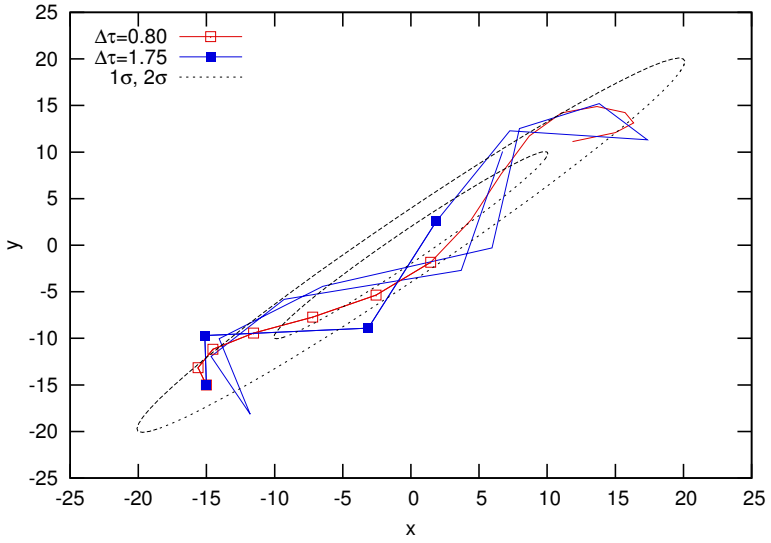



**Figure 6.3:** As Fig. 6.2, but with cm-HMC at $m_c = 0.1$.

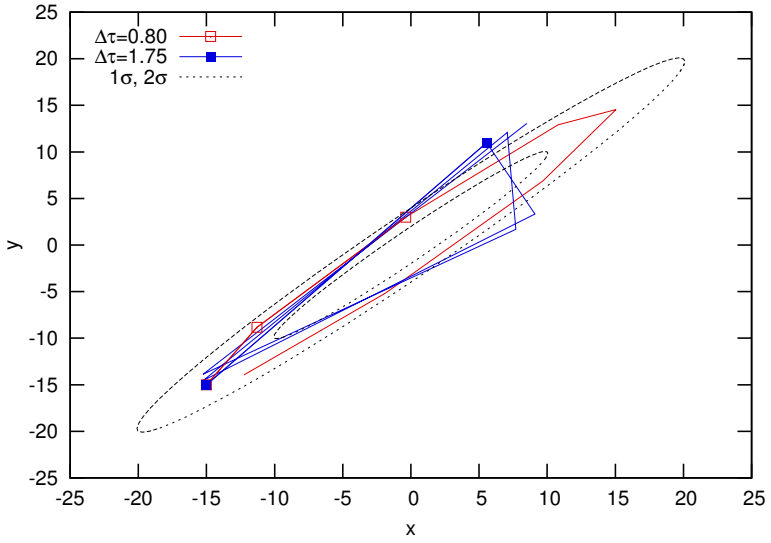



**Figure 6.4:** As Fig. 6.2, but with cm-HMC at $m_c = 0.01$.

to a torus). Often $J_{ij}$ is put to zero except for the nearest neighbors, where it is assumed to be the same for any two nearest neighbors $i$ and $j$. We can then write

$$H(\theta_1, \ldots, \theta_N) = -\sum_{<ij>} J\cos(\theta_i - \theta_j) - \sum_j h_j \cos(\theta_j), \tag{6.46}$$

where $< ij >$ stands for nearest neighbors. As usual, the probability for a configuration $(\theta_1, \ldots, \theta_N)$ is given by the Boltzmann distribution

$$P(\theta_1, \ldots, \theta_N) = \frac{e^{-\beta H(\theta_1, \ldots, \theta_N)}}{Z} \tag{6.47}$$

with the inverse temperature $\beta$ and the partition function $Z$.

For future reference we derive the HMC equations of motion for the $xy$-model as in Sec. 6.1.1. First we introduce conjugate momenta $\pi$ corresponding to $\theta$ and write down an auxiliary Hamiltonian,

$$\mathcal{H}_{\mathrm{MD}}(\theta, \pi) = \tfrac{1}{2}\pi^2 + \beta H_{\mathrm{MD}}(\theta). \tag{6.48}$$

In principle $H_{\mathrm{MD}}$ could be chosen arbitrarily for the guiding Hamiltonian, as long as the correct $H$ is used for the Metropolis accept-reject step, but we do not consider this case here: we use $H_{\mathrm{MD}} = H$, i.e., $\mathcal{H}_{\mathrm{MD}} = \mathcal{H}$. The Hamilton equations for $\mathcal{H}$ read

$$\dot{\theta} = \frac{\delta \mathcal{H}}{\delta \pi} = \pi \tag{6.49a}$$

$$\dot{\pi} = -\frac{\delta \mathcal{H}}{\delta \theta} = -\beta \frac{\delta H(\theta)}{\delta \theta}, \tag{6.49b}$$

and should be understood spin-wise. In this case, with the Hamiltonian Eq. (6.45),

$$\dot{\theta}_k = \frac{\delta \mathcal{H}(\theta, \pi)}{\delta \pi_k} = \pi_k \tag{6.50a}$$

$$\begin{aligned}
\dot{\pi}_k &= -\frac{\delta \mathcal{H}(\theta, \pi)}{\delta \theta_k} = -\beta \frac{\delta H(\theta)}{\delta \theta_k} \\
&= -\beta \frac{\delta}{\delta \theta_k} \left[ -\sum_{i \neq j} J_{ij} \cos(\theta_i - \theta_j) - \sum_j h_j \cos(\theta_j) \right] \\
&= \beta \sum_{i \neq j} J_{ij} \sin(\theta_i - \theta_j)(\delta_{ik} - \delta_{jk}) - \beta \sum_j h_j \sin(\theta_j)\delta_{jk} \\
&= \beta \sum_{\substack{j \\ j \neq k}} (J_{kj} + J_{jk}) \sin(\theta_k - \theta_j) - \beta h_k \sin(\theta_k).
\end{aligned} \tag{6.50b}$$

These equations can be integrated numerically, e.g., with the leapfrog algorithm.

### Transformation in three dimensions

The $xy$-model in three dimensions has a phase transition from a magnetized phase to a disordered phase. A simple numerical study of the phase diagram is given in Fig. 6.5. Near the phase transition long autocorrelation times are observed, as expected — an example is given in Fig. 6.6. It is therefore interesting to study our algorithm in this case.

We use $2 \times 2 \times 2$ blocks and label the points in a block with indices $1, \ldots, 8$, in directions $x$, $y$, and $z$, where $x$ is the fastest index and $z$ is the slowest. We define the coarse momentum field as the (scaled) mean of all momenta in a block,

$$\pi_c = \frac{1}{\sqrt{8}} \sum_{i=1}^{8} \pi_i. \tag{6.51}$$

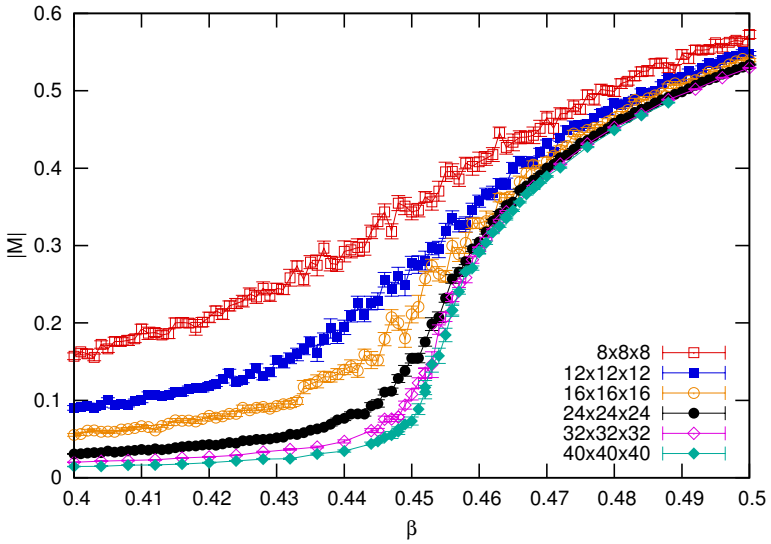



**Figure 6.5:** Phase diagram for the $xy$-model in three dimensions. The plot shows the large finite-volume effects for small lattice sizes. In this work we are not interested in concrete results for the $xy$-model, which has been extensively studied by many authors. We will merely use some simple observables as a test-bench for autocorrelation experiments.

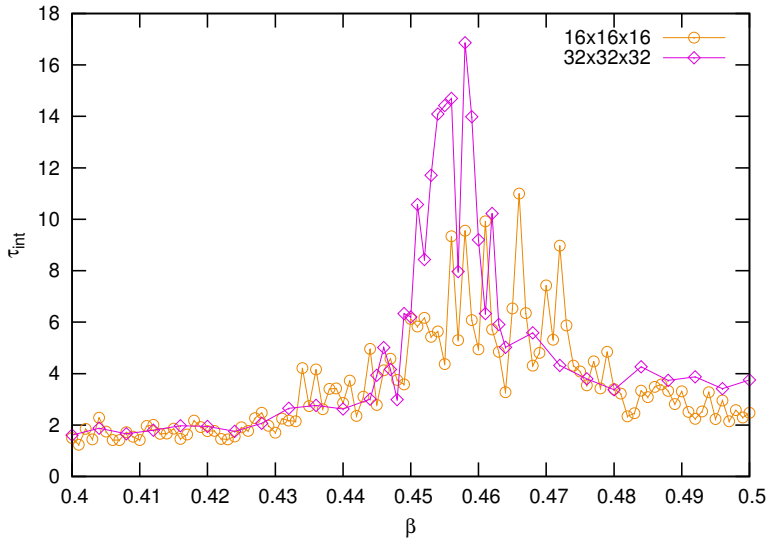



**Figure 6.6:** Autocorrelations for the $xy$-model in three dimensions. There is a lot of noise due to the short Markov chains, but nevertheless we observe a clear peak at the phase transition, that gets higher if the lattice size is increased.

$\pi_c$ is the first component of the vector $\pi'$, obtained from the original momentum vector $\pi = (\pi_1, \ldots, \pi_8)^T$ by an orthogonal transformation $Q$,

$$\pi' = \begin{pmatrix} \pi_c \\ \pi_z \\ \pi_u \\ \pi_d \\ \pi_{ul} \\ \pi_{ur} \\ \pi_{dl} \\ \pi_{dr} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} \\ \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & \frac{1}{\sqrt{8}} & -\frac{1}{\sqrt{8}} & -\frac{1}{\sqrt{8}} & -\frac{1}{\sqrt{8}} & -\frac{1}{\sqrt{8}} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \\ \pi_5 \\ \pi_6 \\ \pi_7 \\ \pi_8 \end{pmatrix} = Q\pi. \tag{6.52}$$

All components of $\pi'$ except for $\pi_c$ are a special choice which we found convenient for the implementation on a computer. Any other choice connected to this one via an orthogonal rotation $Q'$, which leaves $\pi_c$ invariant, is equivalent (if only the mass of the coarse component corresponding to $\pi_c$ is chosen different from 1). In cm-HMC the mass of the coarse component $\pi_c$ is modified to optimize the HMC time evolution of the coarse field components — in this case the center-of-mass degrees of freedom of the $2 \times 2 \times 2$ blocks.

### 6.2.4 Cost analysis

We evaluate the cost of the new algorithm. Consider a single level algorithm in $D$ dimensions, i.e., a blocking of $2^D$ lattice sites by an orthogonal transformation. The operations we need and their multiplicity can be seen in columns one and two of Table 6.1, where multiplicity is per trajectory and $N_{\text{step}}$ is the number of leapfrog steps per trajectory. The number of operations to transform from the original basis to the coarse basis is given by

$$2 \cdot \frac{1}{2^D} \left( 2^D + \sum_{i=1}^{D} 2^i 2^{D-i} \right) = 2(D+1), \tag{6.53}$$

per site. The denominator is the volume of one block, the nominator is the number of floating-point operations for a transformation of the form as in Eq. (6.52), where the overall factor of 2 counts one addition and one multiplication. This may be slightly different depending on the transformation. The sum arises from recursively cutting a $2^D$ hypercube in halves in each dimension, each new degree of freedom giving the difference of two halves on a specific level, as can be seen when considering the matrix in Eq. (6.52). The cost for the reverse transformation is the same. Depending on the number of degrees of freedom per site there is an additional factor, but this is the same for all operations, so we can omit it. Since the blocking is local we can expect that optimization of memory access is possible and there is only negligible performance penalty due to this. In Table 6.1 columns three and four list these additional operations

| operation | multiplicity | additional operations | additional cost |
|---|---|---|---|
| generate momentum | 1 | 1 transform, $1/2^D$ scale | $2(D+1)+1/2^D$ |
| momentum force | $N_{\text{step}}$ | 2 transform, $1/2^D$ scale | $4(D+1)+1/2^D$ |
| energy | 1 | 1 transform, $1/2^D$ scale | $2(D+1)+1/2^D$ |

**Table 6.1:** Operation count and cost of cm-HMC per site. Here "scale" is the scaling with $m_c$ and "transform" is the orthogonal basis transformation (respectively its inverse).

and their cost. *Additional* refers to a comparison with the original algorithm, and the cost is given as floating-point operations per site.

We have to view this additional cost in relation with the cost of the original algorithm. Typically, the most expensive part is the $\delta S/\delta\phi$ term, which has multiplicity $N_{\text{step}}$ for the resulting force term during the trajectory and 1 for the energy computation. In a simulation with dynamical fermions this involves matrix inversions, so the cost of our algorithm would be completely negligible. But even in the simple example of the $xy$-model the relative additional cost is small: the force term of the $xy$-model in $D$ dimensions requires $D$ sin(.) calls per site. Sine is an expensive function on many CPUs, so one can resort to an optimized implementation, e.g., the *sleef* library for elementary functions [69]. Using the SIMD vector units, sleef requires 94 instructions per sin(.). In 3 dimensions this amounts to an additional cost for the modified algorithm of about 6%, which is more than compensated for by the reduced autocorrelation time presented in Sec. 6.3.

### 6.2.5 Recursive algorithm

The transformation we have discussed so far builds a coarse grid from the centers of mass of blocks of size $2^D$. A block side length of 2 is a special choice, side lengths of $3, 4, \ldots$ are obviously also possible. Instead, however, for having maximum control while at the same time keeping the transformations simple, we can employ the $2^D$ blocking recursively, as illustrated in Fig. 6.7. Starting with a $L^D$ lattice we can perform a first level of blocking with blocks of size $2^D$. This yields a first-level coarse grid (the mean values on all blocks) of size $(L/2)^D$. We can block this again with blocks of size $2^D$. The resulting second-level coarse grid is of size $(L/4)^D$. Each point in this second-level coarse grid corresponds to the mean of a $4^D$ block of the original lattice. We can continue this procedure until the resulting $n$th-level coarse grid is too small to allow further blocking. In practice one would usually stop earlier, because too many levels of blocking destroy the locality of the algorithm. The virtue of recursive blocking over single-level blocking with larger blocks is that the former gives control of HMC parameters on each level (one coarse mass parameter per level), while the latter has only the coarse mass for the large blocks as free parameter. That is, the recursive algorithm allows for fine-grained control on many different scales.

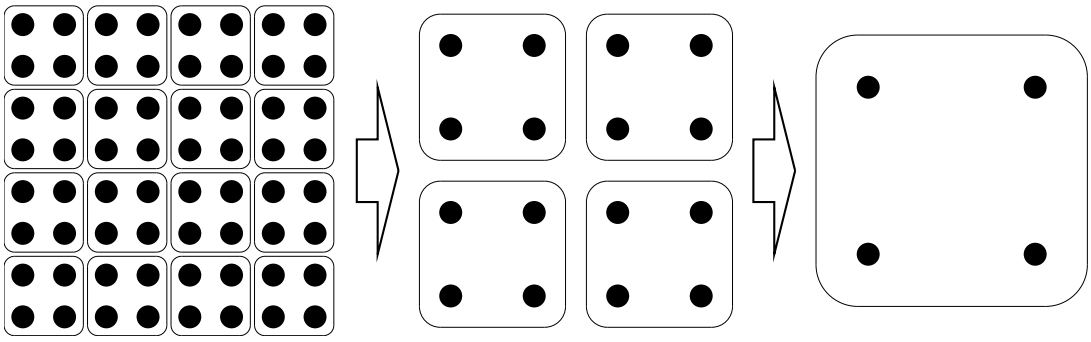


**Figure 6.7:** Illustration of a recursive blocking in two dimensions. Each dot on the leftmost grid represents a lattice point in the original lattice. After each blocking transformation we draw only the coarse degree of freedom, so in the center each dot denotes the center of mass of four dots in a $2 \times 2$ arrangement of the original lattice on the left, as indicated by the framing. Similarly, on the rightmost lattice a dot denotes the center of mass of four dots in the center lattice.

## 6.3 Results

In this section we study in detail the performance of the cm-HMC algorithm. A cost analysis was given in Sec. 6.2.4. We outline our testing procedure in Sec. 6.3.1 and go on to present results for the 3d $xy$-model for a single level algorithm in Sec. 6.3.2 and evaluate a recursive algorithm in Sec. 6.3.3. A cross-check for another integrator is given in Sec. 6.3.4. Throughout this section we use the parameters $J = 1$ and $h = 0$ for the $xy$-model as given in Eq. (6.46).

### 6.3.1 Parameter determination

First, we have to determine the optimal parameters for the original HMC algorithm. We start by choosing a reasonably small $\Delta\tau$ and determine the optimal trajectory length $\tau_0 = N_{\text{step}}\Delta\tau$ (where $N_{\text{step}}$ is the number of steps) for fixed $\Delta\tau$. Then we fix the trajectory length and increase $\Delta\tau$ to find the minimum. Typically the minimum is located at $\Delta\tau$ slightly less than a critical value $\Delta\tau_{\text{critical}}$, where $\tau_{\text{int}}$ starts to deteriorate quickly. This procedure is not perfect, since by increasing $\Delta\tau$ the optimal trajectory length decreases slightly, however this effect is minor and we can expect to be close to the optimum with the outlined procedure. If necessary the method could be iterated to improve the result.

We outline the method for the $xy$-model in three dimensions for a $12^3$ lattice with $\beta = 0.445$. In some preliminary runs we determined that $\Delta\tau = 0.08$ is well below the critical value where trajectories become unstable. Therefore we fix this value for now and find the optimal trajectory length, i.e., the length where the integrated autocorrelation time is minimal. The result in Fig. 6.8 shows a minimum near $\tau_0 = 4$. Next we keep $\tau_0 = 4$ fixed and adjust $\Delta\tau$ in Fig. 6.9. The minimum is at $\Delta\tau = 0.25$ with $\tau_{\text{int}}^{\text{step}} \approx 256$.

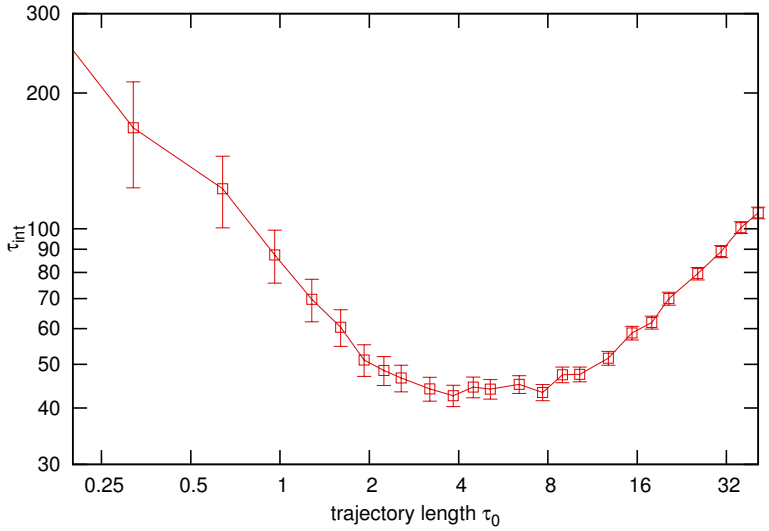



**Figure 6.8:** Lattice size $12^3$, $\beta = 0.445$. Integrated autocorrelation time depending on the trajectory length for fixed $\Delta\tau = 0.08$.

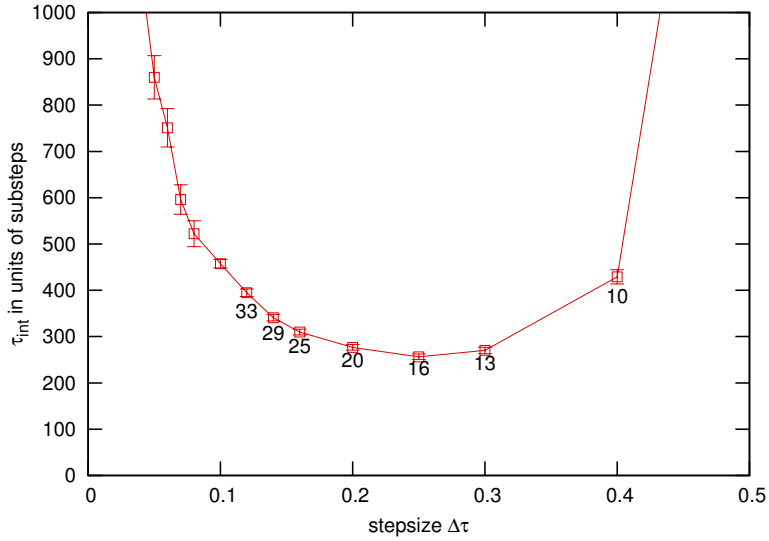



**Figure 6.9:** Lattice size $12^3$, $\beta = 0.445$. Integrated autocorrelation time depending on the step size for fixed trajectory length $\tau_0 = 4$. $\tau_{\text{int}}$ is given in units of steps to allow for identification of the cheapest $\Delta\tau$ in terms of computer time. The labels next to the data points indicate the number of steps per trajectory.

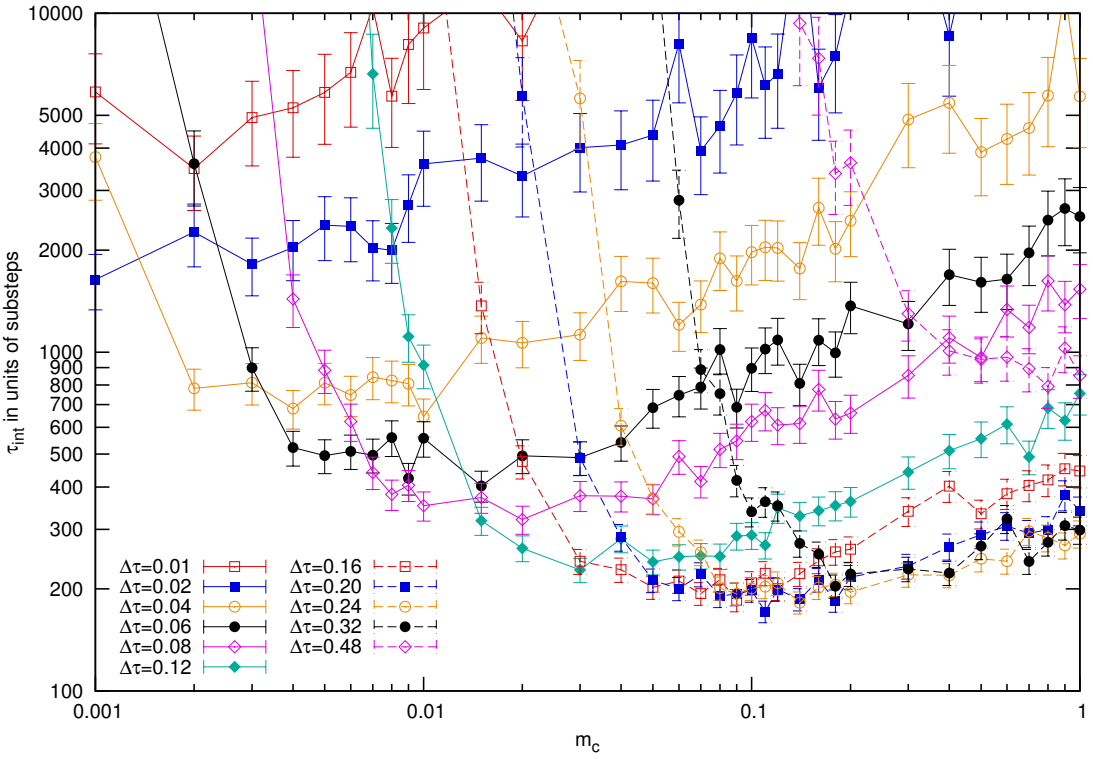



**Figure 6.10:** Lattice size $12^3$, $\beta = 0.445$. Integrated autocorrelation time depending on the coarse mass parameter, for various values of $\Delta\tau$ with a fixed number of 8 substeps per trajectory.

Our procedure for the optimization of the cm-HMC algorithm parameters is slightly different. The reason is that now we cannot define a unique trajectory length: while some degrees of freedom have the original step size, the coarse degrees of freedom see an effective step depending on the mass, but both have the same number of steps per trajectory. Instead, we first try to find the optimal value for $m_c$. To do so, we fix the trajectory length $\tau_0$ and the step size $\Delta\tau$ at sub-critical values and then vary $m_c$.

Figure 6.10 shows results for a $12^3$ lattice with $\beta = 0.445$ which is near the critical point for this lattice size, so we expect large autocorrelation times. We plot $\tau_{\text{int}}$ for a range of step sizes $\Delta\tau$ versus the coarse mass $m_c$. We notice that (1) $\tau_{\text{int}}$ has a minimum for $m_c \neq 1$, i.e., we may save computing time by using the algorithm, (2) as expected, when choosing $\Delta\tau$ too large or too small, $\tau_{\text{int}}$ quickly deteriorates.

If we plot the same data differently, the plot becomes more instructive: instead of $\tau_{\text{int}}$ vs. $m_c$ we can use $\tau_{\text{int}}$ vs. the *coarse effective step size*, $\Delta\tau_c = \Delta\tau/\sqrt{m_c}$, as shown in Fig. 6.11. The result is surprising at first, but easily explained. Lines for different $\Delta\tau$ now lie on top of each other in many regions, unless $\Delta\tau$ is too large or too small. We can understand this behavior by keeping the discussion in Sec. 6.2.2 in mind.

- If $\Delta\tau$ is too small, the correlations of the coarse degrees of freedom are smaller than those of the degrees of freedom in the complementary space. We see a

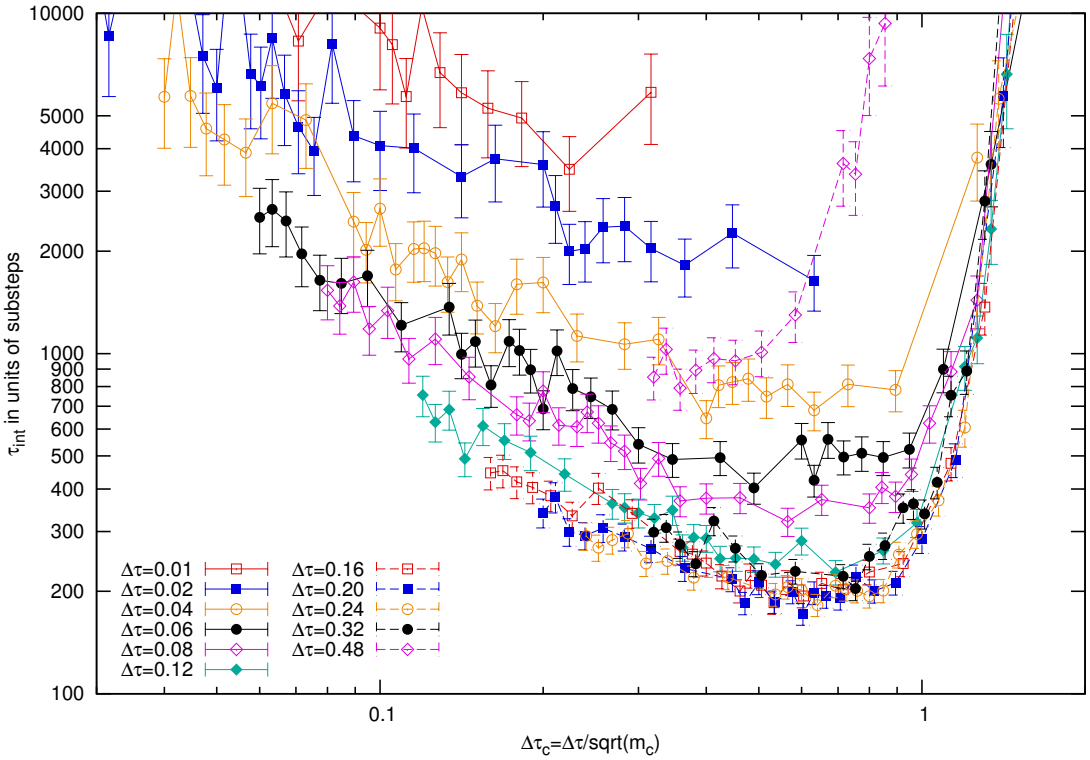



**Figure 6.11:** As Fig. 6.10, but plotted against $\Delta\tau_c = \Delta\tau/\sqrt{m_c}$.

minimum in $\Delta\tau_c$, but the minimal autocorrelation time is dominated by autocorrelations in the complementary space ($\Delta\tau = 0.01, \ldots, 0.08$).

- For medium $\Delta\tau$, $\tau_{\text{int}}^{\text{step}}$ is almost independent of $\Delta\tau$ and we see only a $\Delta\tau_c$ dependence. The autocorrelation times are thus dominated by autocorrelations of the coarse degrees of freedom ($\Delta\tau = 0.12, \ldots, 0.32$).

- If $\Delta\tau$ is too large, the acceptance rate decreases due to a large error introduced in the complement of the coarse subspace ($\Delta\tau = 0.48, \ldots$).

We can conclude from this plot that the optimal $m_c$ does depend on $\Delta\tau$, but the dependence is simple. We have to keep the effective step size for the coarse degrees of freedom at the optimal point by fixing $\Delta\tau_c = \Delta\tau/\sqrt{m_c}$. It is crucial to choose both $\Delta\tau$ and $\Delta\tau_c$ near the minimum, else good performance cannot be obtained. Now we can continue as we did for optimizing the parameters of the original algorithm: we pick a fixed sub-critical value for $\Delta\tau$ and then adjust the trajectory length to find the optimal value, see Fig. 6.12. In a final step we also fix the trajectory length and adjust $\Delta\tau$, see Fig. 6.13. The optimal point is $\Delta\tau = 0.2$, $N_s = 16$, and $m_c = 0.1$ with $\tau_{\text{int}}^{\text{step}} \approx 178$. This is 30% better than the original algorithm.

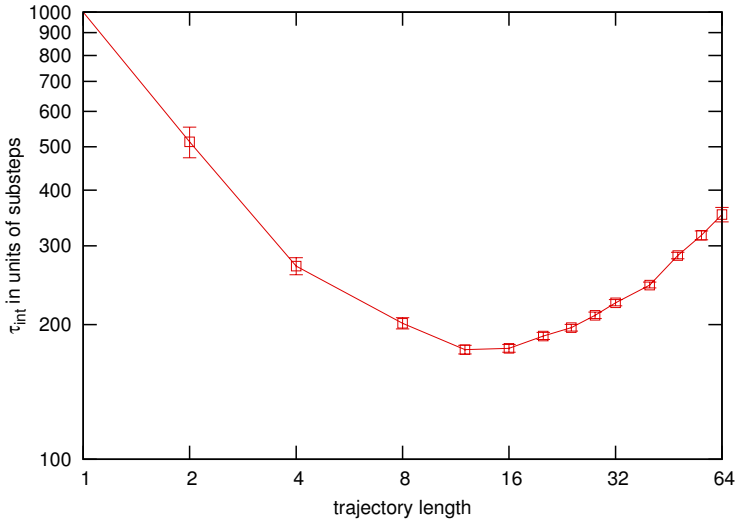



**Figure 6.12:** Lattice size $12^3$, $\beta = 0.445$. Integrated autocorrelation time depending on the trajectory length. We keep $\Delta\tau = 0.2$ and $m_c = 0.1$ fixed, i.e., $\Delta\tau_c \approx 0.632$.

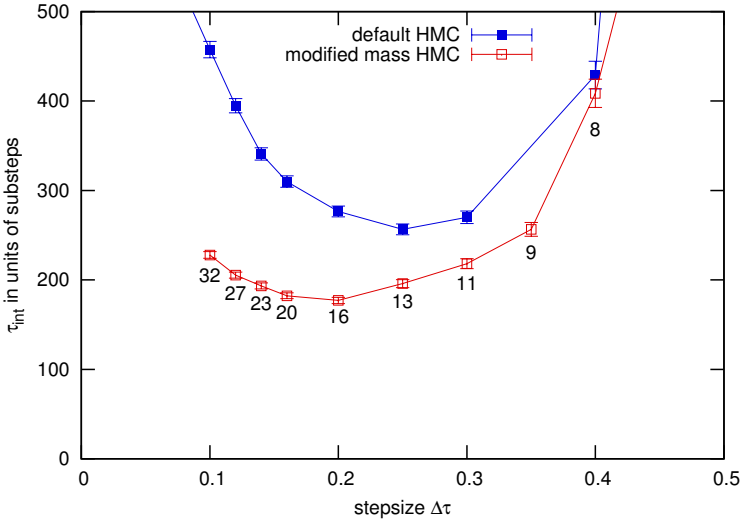



**Figure 6.13:** Lattice size $12^3$, $\beta = 0.445$. Integrated autocorrelation time depending on the step size for fixed trajectory length $\tau_0 = 3.2$ and $\tau_{0,c} = \tau_0/\sqrt{0.1}$. $\tau_{\text{int}}$ is given in units of steps for allowing identification of the cheapest $\Delta\tau$ in terms of computer time. The labels next to the data points indicate the number of steps per trajectory.

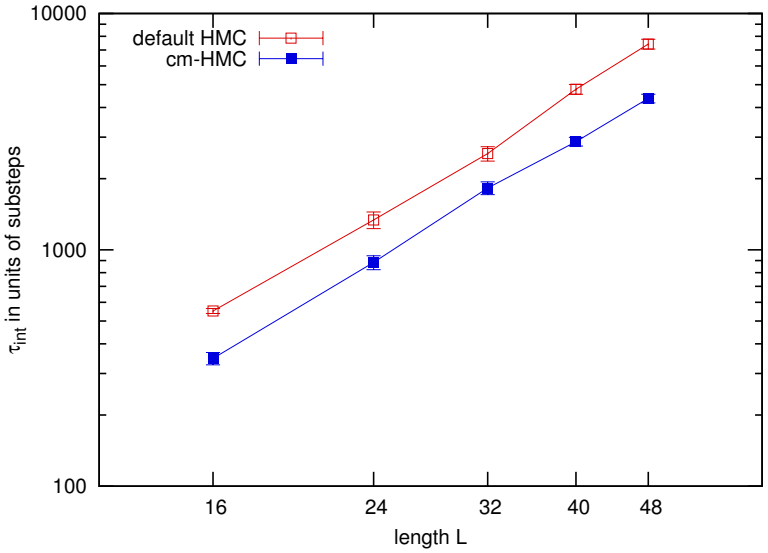



**Figure 6.14:** Optimal integrated autocorrelation time in units of steps. The error bars give the statistical errors. Systematic errors due to the minimization procedure are not included.

### 6.3.2 Volume dependence

In the next step we study the performance of the two algorithms for increasing lattice sizes. As common point we choose the maximum of the magnetic susceptibility for each lattice. We repeat the same optimization procedure as outlined above for each lattice size. The resulting optimal $\tau_{\text{int}}^{\text{step}}$ is given in Fig. 6.14. More detailed data for Fig. 6.14 are given in Table 6.2 and Table 6.3. We observe that $\tau_{\text{int}}^{\text{step}}$ grows with increasing $L$. This is partially due to the smaller required step size, but there is an additional inherent grow. In other words there is a double penalty for simulations at larger lattices.

It should be noted that in some studies certain parameters of HMC are kept fixed for all lattice sizes. For example, Ref. [66] investigates the critical slowing down of the square of the topological charge in QCD as a function of the lattice spacing at a fixed trajectory length $\tau = 4$ (the step size $\Delta\tau$ is varied). This limitation seems arbitrary (but is probably used simply because simulations in QCD are very expensive) and will result in a suboptimal performance of HMC. We advocate optimizing the HMC parameters for each lattice size and only then consider the critical slowing down. Our results support this: in the vicinity of a continuous phase transition we expect a behavior according to a power law,

$$\tau_{\text{int}}^{\text{step}}(L) = a \cdot L^b, \tag{6.54}$$

for parameters $a, b \in \mathbb{R}$. We can try to fit the optimal integrated autocorrelation time to this. For the default HMC algorithm the fit to $a \cdot L^b$ yields a $\chi^2$/d.o.f. of 0.92, and for cm-HMC $\chi^2$/d.o.f. $= 0.27$. Since $\chi^2$/d.o.f. is of order 1, the data points agree well with the power law. From the small $\chi^2$/d.o.f. we can furthermore conclude that the

| lattice size | $\beta$ | $\tau_{0,\mathrm{opt}}$ | $\Delta\tau_{\mathrm{opt}}$ | $\tau^{\mathrm{step}}_{\mathrm{int,opt}}$ |
|---|---|---|---|---|
| 12 | 0.448 | | | |
| 16 | 0.45 | 6 | 0.2 | 551(14) |
| 24 | 0.452 | 10 | 0.2 | 1340(110) |
| 32 | 0.453 | 16 | 0.12 | 2600(200) |
| 40 | 0.4535 | 16 | 0.11 | 4780(230) |
| 48 | 0.454 | 32 | 0.09 | 7460(360) |

**Table 6.2:** Parameters of the simulations with HMC at the critical point. The optimal $\tau_0$ is a rough estimate. $\tau^{\mathrm{step}}_{\mathrm{int,opt}}$ is given in units of steps, the error includes only statistical errors.

| lattice size | $\beta$ | $\Delta\tau_{c,\mathrm{opt}}$ | $\tau_{0,\mathrm{opt}}$ | $\Delta\tau_{\mathrm{opt}}$ | $\tau^{\mathrm{step}}_{\mathrm{int,opt}}$ |
|---|---|---|---|---|---|
| 16 | 0.45 | 0.46 | 3 | 0.12 | 347(21) |
| 24 | 0.452 | 0.38 | 5 | 0.1 | 890(60) |
| 32 | 0.453 | 0.3 | 12 | 0.1 | 1839(120) |
| 40 | 0.4535 | 0.26 | 8 | 0.07 | 2870(120) |
| 48 | 0.454 | 0.25 | 10 | 0.055 | 4370(180) |

**Table 6.3:** As Table 6.2, but for the cm-HMC algorithm.

systematic errors due to the optimization procedure are either small or of similar size in all cases.

The fitted exponent of the two algorithms is 2.35(4) and 2.30(3) respectively, i.e., identical within errors. The coefficient $a$ is smaller for the cm-HMC algorithm, 0.59(7) instead of 0.8(1) for HMC. The conclusion is that our algorithm does not improve the critical exponent, but only has a smaller coefficient. While this result is good when considering a single lattice size it is not satisfactory in the big picture. Without improving the exponent we cannot expect to simulate at much larger lattices than before.

### 6.3.3 Recursive method

In order to optimize the parameters of an algorithm with more than one coarse level, we extend the optimization procedure of the single-level algorithm. We label the mass on the $i$-th level by $m_c^{(i)}$. In an algorithm with $C$ coarse levels, start with $j = C$,

1. fix a suitable (below the critical value where the acceptance rate drops) $\Delta\tau$, fix the trajectory length, fix $m_c^{(i)} = 1.0$ for $i = 1, \ldots, j-1$, find the optimal value for $m_c^{(j)}$,
2. fix $\Delta\tau_c^{(j)}$ (at its optimal value), if $j > 1$, decrease $j$ by one and go to (1),
3. find the optimal trajectory length, keeping $\Delta\tau$ and $\Delta\tau_c^{(i)}$ for $i = 1, \ldots, C$ fixed,
4. find the optimal $\Delta\tau$, keeping the trajectory length fixed (at its optimal value) and $\Delta\tau_c^{(i)}$ for $i = 1, \ldots, C$ fixed.

| algorithm | $\Delta\tau^{(3)}_{c,\mathrm{opt}}$ | $\Delta\tau^{(2)}_{c,\mathrm{opt}}$ | $\Delta\tau^{(1)}_{c,\mathrm{opt}}$ | $\tau_{0,\mathrm{opt}}$ | $\Delta\tau_{\mathrm{opt}}$ | $\tau_{\mathrm{int,opt}}$ |
|---|---|---|---|---|---|---|
| default | - | - | - | 10 | 0.2 | 1340(110) |
| 1 coarse level | - | - | 0.38 | 5 | 0.1 | 890(60) |
| 2 coarse levels | - | 0.7 | 0.30 | 4 | 0.1 | 710(50) |
| 3 coarse levels | 1.0 | 0.5 | 0.28 | 5 | 0.09 | 706(41) |

**Table 6.4:** Optimal parameters for the recursive cm-HMC algorithm

| algorithm | $\Delta\tau_{c,\mathrm{opt}}$ | $\tau_{0,\mathrm{opt}}$ | $\Delta\tau_{\mathrm{opt}}$ | $\tau_{\mathrm{int,opt}}$ |
|---|---|---|---|---|
| default HMC, leapfrog | | 6 | 0.2 | 551(14) |
| default HMC, 2MN | | 8 | 0.5 | 350(20) |
| cm-HMC, leapfrog | 0.46 | 3 | 0.12 | 347(21) |
| cm-HMC, 2MN | 0.8 | 3 | 0.4 | 263(21) |

**Table 6.5:** Comparison of optimal parameters for the cm-HMC algorithm between leapfrog integrator and 2MN integrator

Said briefly, find all optimal masses starting at the coarsest level, then find the optimal trajectory length, finally determine the optimal $\Delta\tau$. We have carried out this procedure for one, two, and three levels for the $24^3$ lattice with $\beta = 0.452$. The optimal parameters are given in Table 6.4. Adding a second levels yields a 20% improvement. Using more than two coarse levels brings no significant improvement, at least in this case. The situation might be different for larger volumes. It is interesting to note that for $C$ coarse levels the value for $\Delta\tau^{(i)}_{c,\mathrm{opt}}$ always lies in the open interval of $(\Delta\tau^{(i)}_{c,\mathrm{opt}}, \Delta\tau^{(i+1)}_{c,\mathrm{opt}})$ for $C+1$ coarse levels.

### 6.3.4 Other integrators

As a consistency check we repeat the benchmarks for another integrator, the second-order minimum norm integrator (2MN) as presented in Ref. [74]. A comparison for the $16^3$ lattice is given in Table 6.5. As expected, the 2MN integrator performs better than leapfrog. The point we are really interested in is the relative improvement of the cm-HMC algorithm. We obtain a relative improvement of 37(4)% for the leapfrog algorithm and 25(7)% for the 2MN integrator. The conclusion is that the improvement of the autocorrelation times is not an artifact of the leapfrog algorithm, but is also present for other integrators. The size of the effect looks similar in both tested cases, although the error bars do not allow for a definite statement.

## 6.4 Coarse-momentum HMC for QCD

In this section we discuss the application of the coarse-momentum HMC algorithm introduced in Sec. 6.2 to QCD or gauge theories in general. The cm-HMC algorithm is obtained from HMC by a block-transformation of the momenta and modification of the kinetic term. Many other variants can be found by considering other and more general

kinetic terms. A further option is the generalized HMC algorithm (GHMC) [75]. GHMC does not modify the HMC Hamiltonian but instead changes the equations of motion in a particular way. For gauge theories the mentioned options all suffer from the same issue, which we discuss below. The fundamental problem is that the modifications break the gauge covariance of the HMC algorithm. We show that this can be fixed, but the ensuing equations of motion cannot be solved properly for discrete auxiliary time. As a consequence this section does not contain any final results, except for a local version of the GHMC method, discussed in Sec. 6.4.2. We think that nevertheless the following discussion is useful and provides insight and a possible starting point for future work.

The mentioned modifications of HMC fall in two main classes: modifications of the kinetic term (e.g., cm-HMC) which we discuss in Sec. 6.4.1, and modifications of the equations of motion, discussed in Sec. 6.4.2.

### 6.4.1 HMC with modified kinetic term

Using the cm-HMC algorithm for gauge theories like QCD was our initial motivation. At first glance the generalization seems trivial, and a naive application of the block transformation to the HMC Hamiltonian of QCD is straightforward. However, it violates the gauge covariance of the HMC algorithm, as we will explain in the following. The usual kinetic term for HMC is

$$T(H) = \frac{1}{2} \sum_{x,\mu} \operatorname{Tr} H_{x,\mu}^{\dagger} H_{x,\mu}, \tag{6.55}$$

where $H_{x,\mu}$ are the traceless-Hermitian momentum variables for the gauge field $U_{x,\mu}$.[7] A more general kinetic term, which is still quadratic in $H$, is given by

$$T(H) = \frac{1}{2} \sum_{x,y,\mu} \operatorname{Tr} H_{x,\mu}^{\dagger} A_{xy,\mu} H_{y,\mu}, \tag{6.56}$$

for some matrix $A_\mu$ to be specified. The cm-HMC is obtained for a block-diagonal $A_\mu = Q^T D_\mu Q$ where $Q$ is an orthogonal rotation matrix, acting as a block transformation similar to Eq. (6.52), and $D_\mu$ is a diagonal matrix. We first study cm-HMC for gauge theories and afterwards consider also other options for $A_\mu$.

#### cm-HMC for gauge theories

The block transformation introduced in Sec. 6.2 combines variables in a $2^D$ block. Considering, e.g., $2^4$ blocking for an $SU(N_c)$ gauge field, the transformed coarse mo-

[7] To stay in the group the $H_{x,\mu}$ must be algebra valued, i.e., traceless-Hermitian.

mentum variable $H^c_{2x,\mu}$ is given by[8]

$$\begin{aligned} H^c_{2x,\mu} = \frac{1}{4}H_{2x,\mu} + \frac{1}{4}\sum_{\nu=1}^{4} H_{2x+\hat{\nu},\mu} + \frac{1}{4}\sum_{\nu=1}^{3}\sum_{\rho=\nu+1}^{4} H_{2x+\hat{\nu}+\hat{\rho},\mu} \\ + \frac{1}{4}\sum_{\nu=1}^{2}\sum_{\rho=\nu+1}^{3}\sum_{\theta=\rho+1}^{4} H_{2x+\hat{\nu}+\hat{\rho}+\hat{\theta},\mu} + \frac{1}{4}H_{2x+\hat{1}+\hat{2}+\hat{3}+\hat{4},\mu}. \end{aligned} \tag{6.57}$$

This is an extension to four dimensions of Eq. (6.51). Under a gauge transformation $G$ we have $H_{x,\mu} \to H^G_{x,\mu} = G_x H_{x,\mu} G^\dagger_x$. For $H^c_{2x,\mu}$, all summands on the right-hand side transform differently, thus $H^c_{2x,\mu}$ has no well-defined transformation behavior under $G$. The consequence is that we cannot use $H^c_{2x,\mu}$ to write down a kinetic term for the HMC Hamiltonian, since any such term would yield equations of motion violating the gauge covariance of HMC — a gauge transformation would not commute with an HMC update. Generally speaking, denoting the time evolution of fields $\Phi$ due to the HMC Hamiltonian $\mathcal{H}(\Phi)$ by $\mathcal{T}_\mathcal{H}$ we require (see Ref. [15])

$$G\left(\mathcal{T}_\mathcal{H}(\Phi)\right) = \mathcal{T}_\mathcal{H}\left(G(\Phi)\right). \tag{6.58}$$

This ensures that physical quantities measured on an ensemble of configurations generated by HMC stay invariant if a gauge transformation is applied to the whole ensemble. According to Ref. [16], this guarantees that no terms breaking local gauge invariance, such as gluon mass terms, can appear in the equilibrium action. Therefore it is crucial to preserve gauge covariance of the Monte Carlo algorithm.

We can fix the gauge covariance if we modify the definition of $H^c_{x,\mu}$ in Eq. (6.57) by inserting chains of link variables, e.g.,

$$\begin{aligned} H^c_{2x,\mu} = \frac{1}{4}H_{2x,\mu} + \frac{1}{4}\sum_{\nu=1}^{4} U_{2x,\nu} H_{2x+\hat{\nu},\mu} U^\dagger_{2x,\nu} \\ + \frac{1}{4}\sum_{\nu=1}^{3}\sum_{\rho=\nu+1}^{4} U_{2x,\nu} U_{2x+\hat{\nu},\rho} H_{2x+\hat{\nu}+\hat{\rho},\mu} U^\dagger_{2x+\hat{\nu},\rho} U^\dagger_{2x,\nu} \\ + \frac{1}{4}\sum_{\nu=1}^{2}\sum_{\rho=\nu+1}^{3}\sum_{\theta=\rho+1}^{4} U_{2x,\nu} U_{2x+\hat{\nu},\rho} U_{2x+\hat{\nu}+\hat{\rho},\theta} H_{2x+\hat{\nu}+\hat{\rho}+\hat{\theta},\mu} \\ \times U^\dagger_{2x+\hat{\nu}+\hat{\rho},\theta} U^\dagger_{2x+\hat{\nu},\rho} U^\dagger_{2x,\nu} \\ + \frac{1}{4} U_{2x,1} U_{2x+\hat{1},2} U_{2x+\hat{1}+\hat{2},3} U_{2x+\hat{1}+\hat{2}+\hat{3},4} H_{2x+\hat{1}+\hat{2}+\hat{3}+\hat{4},\mu} \\ \times U^\dagger_{2x+\hat{1}+\hat{2}+\hat{3},4} U^\dagger_{2x+\hat{1}+\hat{2},3} U^\dagger_{2x+\hat{1},2} U^\dagger_{2x,1}. \end{aligned} \tag{6.59}$$

Since we have $U_{x,\nu} \to G_x U_{x,\nu} G^\dagger_{x+\hat{\nu}}$ under $G$, it is obvious that now also $H^c_{2x,\mu} \to G_{2x} H^c_{2x,\mu} G^\dagger_{2x}$. The definition of all other transformed momentum variables can be

[8] The spatial index $2x$ indicates that $H^c$ is defined only at lattice sites where all components of $x$ are even.

altered similarly to Eq. (6.59). With these we can then construct a kinetic term for the Hamiltonian which is gauge invariant, e.g.,

$$T(H,U) \propto \operatorname{Tr}\left((H^c_{2x,\mu})^\dagger H^c_{2x,\mu}\right) + \cdots\,, \tag{6.60}$$

where after a gauge transformation all matrices $G_{2x}$ cancel either directly or after using the cyclic invariance of the trace. Similarly to the more detailed discussion in Sec. 6.4.2 one can derive equations of motion in the continuum. However, we fail when attempting to discretize these equations in a reversible way. The Hamiltonian is not separable, i.e., $\mathcal{H}(H,U) \neq T(H) + V(U)$ since the new kinetic term depends on the gauge fields *and* momentum fields, so the standard constructions for a symplectic integration scheme fail. Therefore we have to conclude that cm-HMC is not feasible for QCD, unless future work allows to tackle the modified equations of motion.

#### Generalized kinetic term — nearest-neighbor stencil

Another option for a near-local kinetic term is a stencil of the form

$$A_{xy,\mu} = \delta_{xy} + \sum_{\nu=1}^{4} \alpha_{\mu\nu}\left(\delta_{x,y+\hat{\nu}} + \delta_{x,y-\hat{\nu}}\right), \tag{6.61}$$

for variables $\alpha_{\mu\nu} \in \mathbb{R}$. The kinetic term would then read

$$T(H) = \frac{1}{2}\sum_{x,\mu}\left(\operatorname{Tr} H^\dagger_{x,\mu} H_{x,\mu} + \sum_{\pm\nu} \alpha_{\mu\nu} \operatorname{Tr} H^\dagger_{x,\mu} H_{x+\hat{\nu},\mu}\right), \tag{6.62}$$

where the term proportional to $\alpha$ is not gauge invariant. As for the block transformation, we can insert link variables to fix this,

$$T(H,U) = \frac{1}{2}\sum_{x,\mu}\left(\operatorname{Tr} H^\dagger_{x,\mu} H_{x,\mu} + \sum_{\pm\nu} \alpha_{\mu\nu} \operatorname{Tr} H^\dagger_{x,\mu} U_{x,\nu} H_{x+\hat{\nu},\mu} U^\dagger_{x,\nu}\right). \tag{6.63}$$

Again, derivation of the HMC equations of motion for continuous time is straightforward as detailed in Sec. 6.4.2. However, despite the simple structure, we could not find a symplectic integrator for the numerical integration, because the Hamiltonian $\mathcal{H}(H,U)$ is not separable.

We have therefore also excluded another simple modification of the kinetic term. All other — more complicated — kinetic terms will suffer from the same problem. With this and the exclusion of cm-HMC above we can conclude that any nonlocal modification of the kinetic term is not feasible in (non-Abelian) gauge theories, unless a solution is found.

### 6.4.2 Generalized Hybrid Monte Carlo for QCD

In Ref. [75] a generalization of Hybrid Monte Carlo is introduced. The Hamiltonian is not modified, but an arbitrary matrix is incorporated in the equations of motion, without destroying the crucial energy conservation, $\dot{\mathcal{H}} = 0$, and reversibility. In the simplest case of a real field $\phi$ with auxiliary conjugate momentum $\pi$ the Hamiltonian reads

$$\mathcal{H} = \frac{1}{2}\pi^2 + V(\phi). \tag{6.64}$$

The *generalized* equations of motion then are

$$\dot{\phi}_i = A_{ij}\pi_j, \tag{6.65a}$$

$$\dot{\pi}_i = -(A^T)_{ij}\frac{\delta V}{\delta \phi_j} = -A_{ji}\frac{\delta V}{\delta \phi_j}, \tag{6.65b}$$

where $A$ is an arbitrary matrix and a sum over repeated indices is implied. The usual Hamilton equations are recovered when $A$ is the unit matrix. It is simple to show that these equations conserve energy, by considering $\dot{\mathcal{H}}$,

$$\begin{aligned} \frac{d\mathcal{H}}{dt} &= \frac{\partial \mathcal{H}}{\partial \pi_i}\frac{d\pi_i}{dt} + \frac{\partial \mathcal{H}}{\partial \phi_i}\frac{d\phi_i}{dt} \\ &= -\pi_i A_{ji}\frac{\delta V}{\delta \phi_j} + \frac{\delta V}{\delta \phi_i}A_{ij}\pi_j \\ &= 0. \end{aligned} \tag{6.66}$$

Time reversibility can easily be checked [75].

Without loss of generality we now consider the pure $SU(N_c)$ gauge action $S(U)$. Fermions can be added without changing the following discussion and results. The HMC Hamiltonian is given by

$$\mathcal{H} = \frac{1}{2}\sum_{x,\mu} \operatorname{Tr} H_{x,\mu}^\dagger H_{x,\mu} + S(U), \tag{6.67}$$

with the Wilson gauge action

$$S(U) = \frac{2}{g_0^2}\sum_x \sum_{\mu<\nu} \operatorname{Tr}\left[1 - \frac{1}{2}\left(U_{\mu\nu}(x) + U_{\mu\nu}^\dagger(x)\right)\right], \tag{6.68}$$

where $g_0$ is the gauge coupling and the plaquette $U_{\mu\nu}$ is defined as

$$U_{\mu\nu}(x) = U_\mu(x)U_\nu(x+\hat{\mu})U_\mu^\dagger(x+\hat{\nu})U_\nu^\dagger(x). \tag{6.69}$$

Since we have two indices, $\mu$ and $x$, there are two special choices for the matrix $A_{ij}$ in the generalized equations of motion (to be derived below), where the indices are $i = (x, \mu)$,

$$A^{\text{nonlocal}}_{x,\mu;y,\nu} = A_{xy,\mu}\delta_{\mu\nu}, \tag{6.70a}$$

$$A^{\text{local}}_{x,\mu;y,\nu} = \delta_{xy}A_{\mu\nu}. \tag{6.70b}$$

After proper index reordering both choices are block-diagonal. More general choices are possible, however we will show in the following that any nonlocal $A$ requires a further modification of the equations of motion to keep gauge covariance, which can then no longer be integrated numerically. Therefore considering a more general $A$ suffers from the same problem and is not feasible. Consequently we afterwards study a local $A$ which does not break gauge invariance.

### Nonlocal $A$

As indicated above, we require the update algorithm to be gauge covariant, but for nonlocal $A$ the molecular-dynamics time evolution described by Eqs. (6.65) is not. In order to find a generalization, we consider pure Wilson $SU(N_c)$ gauge theory and derive the equations of motion in analogy to Ref. [32]. The usual equations of motion are given by

$$\dot{U}_{x,\mu} = iH_{x,\mu}U_{x,\mu}, \tag{6.71a}$$

$$i\dot{H}_{x,\mu} = \frac{\beta}{6}\left[(U_{x,\mu}V_{x,\mu} - V_{x,\mu}^\dagger U_{x,\mu}^\dagger) - \frac{1}{N_c}\operatorname{Tr}(U_{x,\mu}V_{x,\mu} - V_{x,\mu}^\dagger U_{x,\mu}^\dagger)\right], \tag{6.71b}$$

where $V_{x,\mu}$ is the usual notation for the sum of the staples of the link $U_{x,\mu}$. In analogy to Eq. (6.65a) we want to introduce a nonlocal sum over momenta $H_{x,\mu}$. The equation of motion is invariant under a local gauge transformation $G$,

$$\begin{aligned}\dot{U}_{x,\mu}^G &= G_x\dot{U}_{x,\mu}G_{x+\hat{\mu}}^\dagger \\ &= iG_xH_{x,\mu}U_{x,\mu}G_{x+\hat{\mu}}^\dagger \\ &= iG_xH_{x,\mu}G_x^\dagger G_xU_{x,\mu}G_{x+\hat{\mu}}^\dagger \\ &= iH_{x,\mu}^GU_{x,\mu}^G.\end{aligned} \tag{6.72}$$

We see that the invariance is based on the insertion/cancellation of $G_x^\dagger G_x$ in between $H_{x,\mu}$ and $U_{x,\mu}$. If we want to replace $H_{x,\mu}$ by a sum, $\sum_y A_{xy,\mu}H_{y,\mu}$, each summand transforms differently under $G$ and the cancellation does not work.[9] We can fix this by inserting path-ordered products of links, $S_{xy} = U_{x,\nu}\cdot U_{x+\hat{\nu},\rho}\cdot\dots\cdot U_{y-\hat{\sigma},\sigma}$, which connect $x$ with $y$ and transform as $S_{xy}^G = G_xS_{xy}G_y^\dagger$. In general $S_{xy}$ can be an average over many such paths with the same start and endpoint. It is reasonable to choose $S_{xy}^\dagger = S_{yx}$. Now we can write down a generalization of Eq. (6.71a),

$$\dot{U}_{x,\mu} = i\left(\sum_y A_{xy,\mu}S_{xy}H_{y,\mu}S_{xy}^\dagger\right)U_{x,\mu}, \tag{6.73}$$

where the matrix elements $A_{xy,\mu}$ are arbitrary complex constants. For symmetry reasons it makes sense to choose $A_{xy,\mu} = A_{yx,\mu}$. The $A_{xy,\mu}$ are entries of a matrix $A_\mu$,

[9] Replacing not only $H_{x,\mu}$ but $H_{x,\mu}U_{x,\mu}$ would not work either: in $\sum_Y H_{y,\nu}U_{y,\nu}$ only the $x = y$ term transforms as required.

so $A_\mu^T = A_\mu$. We can check the gauge invariance of the equation,

$$\begin{aligned}
\dot{U}^G_{x,\mu} &= G_x \dot{U}_{x,\mu} G^\dagger_{x+\hat{\mu}} \\
&= iG_x \left( \sum_y A_{xy,\mu} S_{xy} H_{y,\mu} S^\dagger_{xy} \right) U_{x,\mu} G^\dagger_{x+\hat{\mu}} \\
&= iG_x \left( \sum_y A_{xy,\mu} S_{xy} G^\dagger_y G_y H_{y,\mu} G^\dagger_y G_y S^\dagger_{xy} \right) G^\dagger_x G_x U_{x,\mu} G^\dagger_{x+\hat{\mu}} \\
&= i \left( \sum_y A_{xy,\mu} S^G_{xy} H^G_{y,\mu} (S^G_{xy})^\dagger \right) U^G_{x,\mu},
\end{aligned} \tag{6.74}$$

which is consistent with Eq. (6.73). Introducing the vector $s_x = (S_{x1}, \ldots, S_{xV})$, matrices $H_\mu = \mathrm{diag}(H_{1,\mu}, \ldots, H_{V,\mu})$ and $A_{x,\mu} = \mathrm{diag}(A_{x1,\mu} I_{N_c \times N_c}, \ldots, A_{xV,\mu} I_{N_c \times N_c})$, the equation of motion takes a more compact form,

$$\dot{U}_{x,\mu} = i \left( s_x A_{x,\mu} H_\mu s^\dagger_x \right) U_{x,\mu}, \tag{6.75}$$

where no summation over $x$ and $\mu$ is implied, and the term in parentheses is a product of four vectors/matrices. In the following we sometimes drop the index $\mu$. Next we have to derive the second equation of motion. It follows by requiring energy conservation, $\dot{\mathcal{H}} = 0$, and using the new equation of motion, Eq. (6.73),

$$\begin{aligned}
\dot{\mathcal{H}} &= \sum_{x,\mu} \mathrm{Tr} \left[ \dot{H}_x H_x + \frac{\beta}{6} \left( i \dot{U}_x V_x - i V^\dagger_x \dot{U}^\dagger_x \right) \right] \\
&= \sum_{x,\mu} \mathrm{Tr} \left[ \dot{H}_x H_x + i \frac{\beta}{6} \left( s_x A_x H s^\dagger_x U_x V_x - \mathrm{h.c.} \right) \right] \\
&= \sum_{x,\mu} \mathrm{Tr} \left[ \dot{H}_x H_x \right] + \sum_{x,\mu} \mathrm{Tr} \left[ i \frac{\beta}{6} \sum_y \left( A_{xy} S_{xy} H_y S^\dagger_{xy} U_x V_x - \mathrm{h.c.} \right) \right] \\
&= \sum_{x,\mu} \mathrm{Tr} \left[ \dot{H}_x H_x \right] + \sum_{x,\mu} \mathrm{Tr} \left[ i \frac{\beta}{6} \sum_y \left( A_{xy} H_y S^\dagger_{xy} U_x V_x S_{xy} - \mathrm{h.c.} \right) \right] \\
&= \sum_{\mu} \mathrm{Tr} \left[ \sum_x \dot{H}_x H_x + i \frac{\beta}{6} \sum_{xy} \left( A_{xy} H_y S^\dagger_{xy} U_x V_x S_{xy} - \mathrm{h.c.} \right) \right].
\end{aligned} \tag{6.76}$$

Renaming the summation indices in the second sum we obtain

$$\begin{aligned}
\dot{\mathcal{H}} &= \sum_{\mu} \mathrm{Tr} \left[ \sum_x \dot{H}_x H_x + i \frac{\beta}{6} \sum_{xy} \left( A_{yx} H_x S^\dagger_{yx} U_y V_y S_{yx} - \mathrm{h.c.} \right) \right] \\
&= \sum_{x,\mu} \mathrm{Tr} \left[ H_x \left( \dot{H}_x + i \frac{\beta}{6} \sum_y \left( A_{xy} S_{xy} U_y V_y S^\dagger_{xy} - \mathrm{h.c.} \right) \right) \right] \equiv 0.
\end{aligned} \tag{6.77}$$

Following the reasoning in Ref. [32], this leads to an equation for $H_{x,\mu}$,

$$\begin{aligned} i\dot{H}_x &= \frac{\beta}{6}\left[\sum_y \left(A_{xy}S_{xy}U_yV_yS_{xy}^\dagger - \text{h.c.}\right) - \frac{1}{N_c}\operatorname{Tr}\left(\sum_y A_{xy}S_{xy}U_yV_yS_{xy}^\dagger - \text{h.c.}\right)\right] \\ &= \frac{\beta}{6}\left[\sum_y \left(A_{xy}S_{xy}U_yV_yS_{xy}^\dagger - \text{h.c.}\right) - \frac{1}{N_c}\sum_y A_{xy}\operatorname{Tr}\left(S_{xy}^\dagger S_{xy}U_yV_y - \text{h.c.}\right)\right] \\ &= \frac{\beta}{6}\sum_y\left[\left(A_{xy}S_{xy}U_yV_yS_{xy}^\dagger - \text{h.c.}\right) - \frac{1}{N_c}A_{xy}\operatorname{Tr}\left(U_yV_y - \text{h.c.}\right)\right]. \end{aligned} \tag{6.78}$$

We can now summarize the equations of motion for the generalized HMC, which take a more compact form by defining the matrices $U_\mu = \operatorname{diag}(U_{1,\mu},\ldots,U_{V,\mu})$ and $V_\mu = \operatorname{diag}(V_{1,\mu},\ldots,V_{V,\mu})$:

$$\dot{U}_{x,\mu} = i\left(s_xA_{x,\mu}H_\mu s_x^\dagger\right)U_{x,\mu}, \tag{6.79a}$$

$$i\dot{H}_{x,\mu} = \frac{\beta}{6}\left[\left(s_xA_{x,\mu}U_\mu V_\mu s_x^\dagger - \text{h.c.}\right) - \frac{1}{N_c}\operatorname{Tr}\left(A_{x,\mu}U_\mu V_\mu - \text{h.c.}\right)\right]. \tag{6.79b}$$

Before studying the discretization of these equations, and why it fails in this case, we discuss two open details. First, we have not defined how $S_{xy}$ should be defined. A priori there is no strict requirement. Choosing the shortest possible path seems to be the most reasonable option. If there is more than one such path we average over all paths of equal length. Second, the matrix $A_{xy,\mu}$ needs to be chosen. For isotropic systems, as in the case for most lattice QCD simulations, choosing $A_{xy,\mu} = A_{xy,\nu} = A_{xy}$ for any $\mu$ and $\nu$ seems sensible. Furthermore, requiring translation invariance $A_{xy} = A_{x+\hat{\mu},y+\hat{\mu}}$ appears natural. Fixing the shape further is more difficult. Requiring locality (e.g., choosing a simple stencil), or an exponential decline with distance are obvious choices.

When discretizing the equations of motion as usual, the equation for $U$ must ensure that we stay in the group. In the standard HMC this means that

$$U_{x,\mu}(\tau+\Delta\tau) = e^{iH_{x,\mu}(\tau+\Delta\tau/2)\Delta\tau}U_{x,\mu}(\tau), \tag{6.80}$$

where the exponent is traceless and anti-Hermitian.[10] Solving the generalized equation in an exact way is problematic, since the $s_x$ contain $U_{x,\mu}$, i.e., the equation is nonlinear, but one could still attempt a numerical solution. But the generalized method fails on a more fundamental level: we need a symplectic integrator to ensure energy conservation and time reversibility for the Hamiltonian time evolution. Reversibility is required for proving that HMC satisfies the detailed-balance condition [19] and (approximate) energy conservation is necessary for acceptance probabilities significantly larger than zero. Symplectic integrators are readily found for separable Hamiltonians,

$$\mathcal{H}(p,q) = T(p) + V(q). \tag{6.81}$$

[10] An approximation with a Taylor expansion is usually not sufficient, unless the expansion is of high order, since the result will not be in $SU(N_c)$.

If $\mathcal{H}(p,q)$ does not take this form, it is easy to see that a construction of a symplectic integrator like the leapfrog algorithm in Sec. 6.1.1 fails. In our GHMC algorithm we did not modify the Hamiltonian, so it is still separable, however the modified equations of motion do not reflect this fact: the force terms contain both $H$ and $U$, as for an inseparable Hamiltonian. Constructing symplectic integrators for inseparable Hamiltonians is difficult and the general case is an open problem. Instead of finding a general solution for Eq. (6.79) we can try to consider a simple case, where $A_{xy,\mu}$ includes only nearest neighbors, similar to the modification of the kinetic term in Sec. 6.4.1, but the equations do not simplify sufficiently to suggest a solution. The consequence is that this generalized HMC for now reached a dead end:

1. The naive generalization violates gauge covariance of HMC.
2. We can fix the gauge covariance of the time evolution.
3. A reversible discretization of the fixed time-evolution equations is hard (unsolved).
4. An irreversible discretization breaks detailed balance of HMC.

### Local $A$

We have seen above that any $A$ that combines objects that transform differently under local gauge transformations ultimately leads to equations of motion that cannot be handled. This leaves us one more option for using GHMC for QCD: in 4 dimensions at each site $x$ there are $D = 4$ matrices $H_{x,\mu}$, which all transform as $H_{x,\mu} \to G_x H_{x,\mu} G_x^\dagger$. Obviously any sum of these matrices follows the same transformation law. Recalling the usual equations of motion,

$$\dot{U}_{x,\mu} = iH_{x,\mu}U_{x,\mu}, \tag{6.82a}$$

$$i\dot{H}_{x,\mu} = \frac{\beta}{6}\left[(U_{x,\mu}V_{x,\mu} - V_{x,\mu}^\dagger U_{x,\mu}^\dagger) - \frac{1}{N_c}\operatorname{Tr}(U_{x,\mu}V_{x,\mu} - V_{x,\mu}^\dagger U_{x,\mu}^\dagger)\right], \tag{6.82b}$$

we can write down the local generalization,

$$\dot{U}_{x,\mu} = i\left(H_{x,\mu} + \alpha\sum_{\nu\neq\mu} H_{x,\nu}\right)U_{x,\mu}, \tag{6.83a}$$

$$\begin{aligned} i\dot{H}_{x,\mu} = {} & \frac{\beta}{6}\left[(U_{x,\mu}V_{x,\mu} - V_{x,\mu}^\dagger U_{x,\mu}^\dagger) - \frac{1}{N_c}\operatorname{Tr}(U_{x,\mu}V_{x,\mu} - V_{x,\mu}^\dagger U_{x,\mu}^\dagger)\right] \\ & + \alpha\frac{\beta}{6}\sum_{\nu\neq\mu}\left[(U_{x,\nu}V_{x,\nu} - V_{x,\nu}^\dagger U_{x,\nu}^\dagger) - \frac{1}{N_c}\operatorname{Tr}(U_{x,\nu}V_{x,\nu} - V_{x,\nu}^\dagger U_{x,\nu}^\dagger)\right]. \end{aligned} \tag{6.83b}$$

For $\alpha = 0$ the original equations are recovered. A priori $\alpha$ could depend on $x$, $\mu$, and $\nu$ but for symmetry reasons we choose it is a constant.

We make a test of this algorithm for a quenched simulation on a $16^4$ lattice at $\beta = 5.789$. As for $\alpha = 0$, energy is also approximately conserved for $\alpha \neq 0$, if the

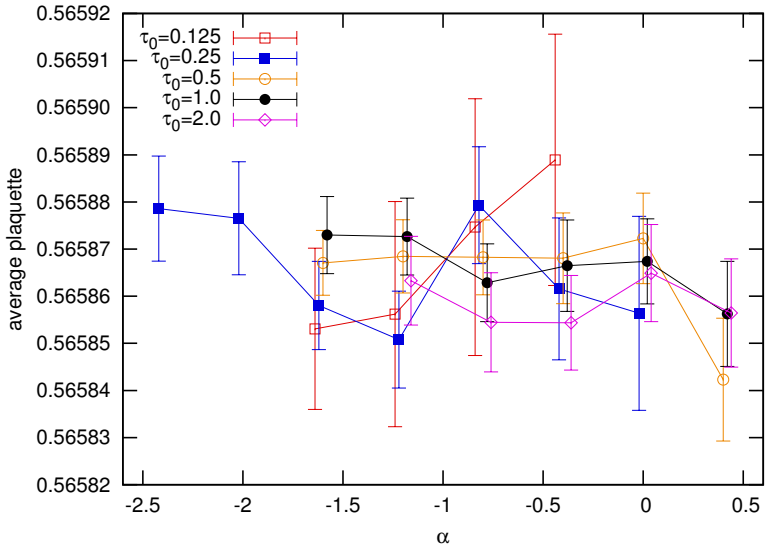
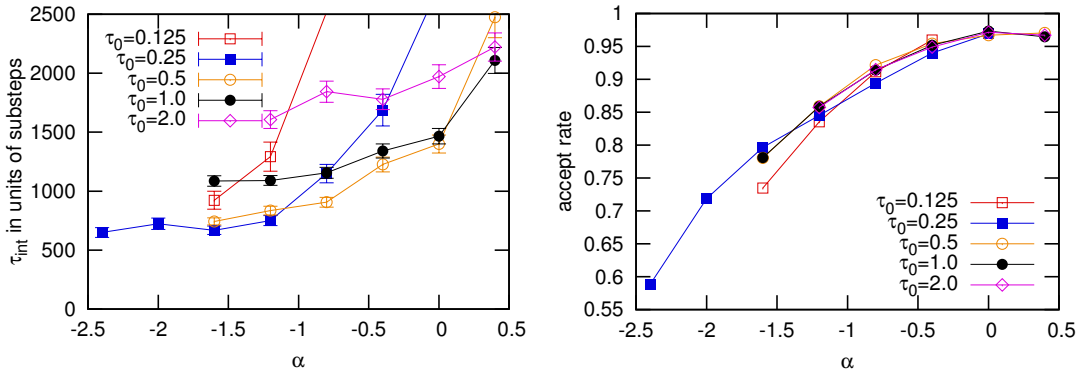



**Figure 6.15:** Average plaquette depending on $\alpha$ and the trajectory length. Small horizontal offsets were added to the data points for better visibility. All data samples agree within errors, as they should.

step size is not too large. As consistency check we measure the error in $\mathcal{H}$ after integrating one trajectory forward, reverse the time, and integrate backwards again. As required we have $\Delta\mathcal{H} = 0$ close to machine precision, i.e., reversibility is also confirmed numerically. Of course all observables must be unchanged within statistical errors when introducing $\alpha \neq 0$. As an example we measured the average plaquette, given in Fig. 6.15. The $\alpha = 0$ results correspond to the unmodified algorithm. We observe no significant deviation of the $\alpha \neq 0$ results from this reference value.

To determine the influence of $\alpha$ on the integrated autocorrelation time $\tau_{\text{int}}$, we start by performing a series of HMC runs for different values of $\alpha$. Since $\tau_{\text{int}}$ depends strongly on the trajectory length $\tau_0$ we give in Fig. 6.16a results for a whole range of $\tau_0$ at fixed step size $\Delta\tau = 0.005$. At $\alpha = 0$ we have $\tau_{\text{int}}^{\text{step}} \approx 1400$, which decreases to $\tau_{\text{int}}^{\text{step}} \approx 700$ at $\alpha = -1.6$. However, a change of $\alpha$ also changes the accept rate of HMC, as can be seen in Fig. 6.16b. Since at negative $\alpha$ the accept rate is worse, we can expect that the maximal allowed step size is smaller than for $\alpha = 0$. In Fig. 6.17 we show the result of a series of HMC runs for the "optimal" $\alpha = -1.6$ and $\alpha = 0$ for varying step size. The relative difference in $\tau_{\text{int}}^{\text{step}}$ is smaller after this additional optimization step, since — with respect to Fig. 6.16a — the step size for $\alpha = 0$ could be increased more than for $\alpha = -1.6$, at the respective optimal trajectory length. The reason is that adding contributions from all directions in the equation of motion Eq. (6.83b) increases the average and maximal magnitude of the force. In a final step we find the optimal $\tau_0$, which was so far only roughly approximated.[11] We keep the optimal $\Delta\tau$ found in Fig. 6.17, $\Delta\tau(\alpha = 0) = 0.0165$ and $\Delta\tau(\alpha = -1.6) = 0.0062$, and vary $\tau_0$. The

[11] In the optimization in Fig. 6.16a there are factors of two between $\tau_0$ for different runs, which turns out to be too coarse to give a definite determination of the optimal $\alpha$.




**(a)** Integrated autocorrelation time **(b)** Accept rate of HMC

**Figure 6.16:** Integrated autocorrelation time and accept rate depending on $\alpha$ for various trajectory lengths. The step size $\Delta\tau$ is fixed at 0.005.

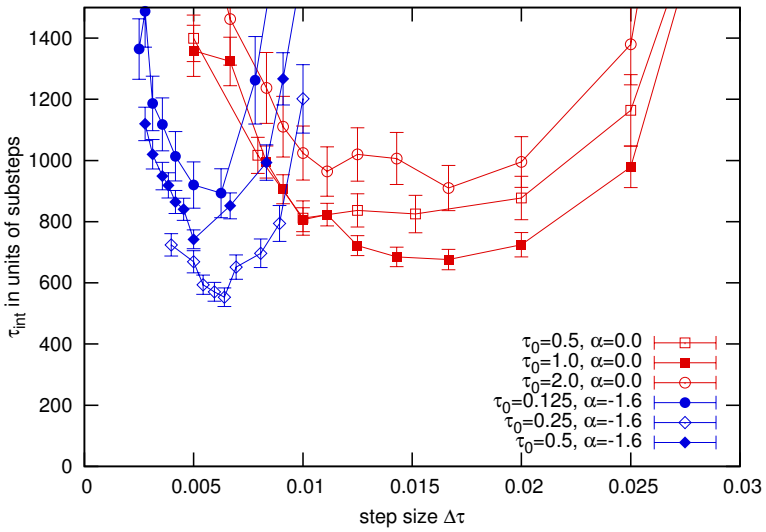



**Figure 6.17:** Integrated autocorrelation time depending on the step size $\Delta\tau$ for trajectory lengths near the optimal length. We give data for the unmodified algorithm ($\alpha = 0$) and at an value of $\alpha = -1.6$ which is the minimum observed in Fig. 6.16a.

result is given in Fig. 6.18. The value of $\tau_{\text{int}}$ at the minimum can be determined by a quadratic fit around the minimum. At $\alpha = 0$ we obtain $\tau_{\text{int}}^{\text{opt}} = 598(11)$ at $\tau_0 = 0.82$.[12] For $\alpha = -1.6$ the fit yields $\tau_{\text{int}}^{\text{opt}} = 594(22)$ at $\tau_0 = 0.28$. In both cases the reduced $\chi^2$ is close to 1, so the fit quality is good.

The conclusion is that we cannot observe any positive (or negative) influence of the local generalized HMC on the integrated autocorrelation time.

[12] See Appendix B on how the error at the minimum of a quadratic fit is obtained.

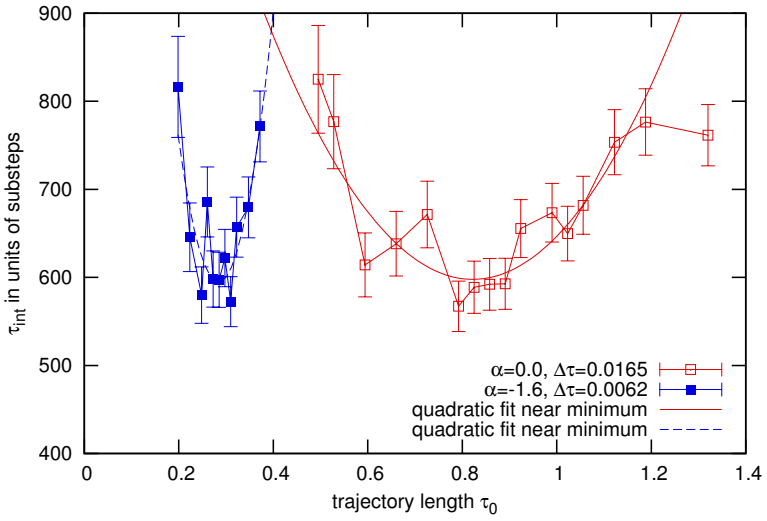



**Figure 6.18:** Integrated autocorrelation time depending on the trajectory length $\tau_0$ for the optimal step sizes $\Delta\tau$ found in Fig. 6.17. We give data for the unmodified algorithm ($\alpha = 0$) and at an value of $\alpha = -1.6$ which is the minimum observed in Fig. 6.16a.

## 6.5 Conclusions

In this chapter we introduced a cheap and simple modification of Hybrid Monte Carlo, based on a linear transformation of the auxiliary momentum variables. The centers of mass of small blocks of lattice sites can be considered as a coarse field. The auxiliary mass of this field can be adjusted independently of the mass of the remaining degrees of freedom. This gives and additional parameter for HMC, which can be used to tune the algorithm such that lower autocorrelation times are obtained.

The presented study of the $xy$-model shows good results with the new algorithm, however, the method does not change the critical exponent of the autocorrelation time. Therefore, the problem of long autocorrelation times is not solved but only moved to slightly larger lattices.

For gauge theories the naive algorithm breaks gauge invariance of HMC, so modifications become necessary. These, however, yield equations of motion that cannot be solved numerically in a time-reversible way. The conclusion is that the cm-HMC method (or any other nontrivial linear transformation of momentum or field variables) is not applicable to QCD or gauge theories in general. The only exception is a local modification which uses the fact that at each site there is one momentum matrix for each dimension, so all of them transform identically under local gauge transformations. We used this for a local version of generalized HMC, but no improvement of the autocorrelation time is obtained.

# Chapter 7

# Schur complement with non-overlapping subdomains for HMC

In Ref. [45] a modification of HMC based on the Schur complements of arbitrary matrix decompositions is described. We give a very short review and point out that an identical approach can easily be taken for the non-overlapping domain decomposition that we described extensively in Ch. 3.

For two flavors the analytic integration of the fermionic part of the action yields the fermion determinant $\det(MM)$. Using $\gamma_5$-hermiticity one rewrites it as

$$\det(MM) = \det(M\gamma_5\gamma_5 M) = \det(\gamma_5 M^\dagger \gamma_5 M) = \det(M^\dagger M). \tag{7.1}$$

Then one can introduce pseudo-fermions, which leads to

$$\det(M^\dagger M) = \int \mathcal{D}\chi^* \mathcal{D}\chi \exp\left(-\chi^* (M^\dagger M)^{-1} \chi\right). \tag{7.2}$$

Before introducing pseudo-fermions, we can alternatively rewrite $\det(M)$ and $\det(M^\dagger)$, using their Schur complements. For a matrix

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \tag{7.3}$$

the determinant can be factorized according to the well known formula

$$\det(M) = \det\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \det(D)\det(A - BD^{-1}C). \tag{7.4}$$

One can then introduce *separate* pseudo-fermions for *each* of these determinants. The advantage is an improved condition number of the matrix $D$ and its Schur complement (compared to the condition number of $M$), and the option to use multiple time-scale integration, as discussed in detailed in Ref. [45]. There, two possible decompositions are discussed. (1) Even/odd preconditioning, where $A$ contains all even lattice sites and $D$ contains all odd lattice sites. (2) Schwarz preconditioning decomposes the lattice into hypercubic blocks. $A$ then contains all even blocks and $D$ all odd blocks, so both $A$ and $D$ are block-diagonal (if the operator has only nearest-neighbor coupling there is no coupling between two blocks of the same color). For symmetry reasons, one can also pull a factor of $A$ out of the determinant in formula Eq. (7.4),

$$\det(D)\det(A - BD^{-1}C) = \det(D)\det(A)\det(1 - A^{-1}BD^{-1}C), \tag{7.5}$$

since $A$ and $D$ are equivalent in the two mentioned decompositions.

The same procedure is possible when splitting the lattice into boundary and interior sites, i.e., for a non-overlapping domain decomposition as in Ch. 3. There we can write

$$\det(M) = \det \begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix} = \det(A_{II}) \det(A_{BB} - A_{BI} A_{II}^{-1} A_{IB}), \tag{7.6}$$

and introduce separate pseudo-fermions for the interior (for $\det(A_{II})$) and boundary (for $\det(A_{BB} - A_{BI} A_{II}^{-1} A_{IB})$),

$$\begin{aligned} \det(M^\dagger M) &= \det(A_{II}) \det(A_{BB} - A_{BI} A_{II}^{-1} A_{IB}) \\ &\quad \times \det(A_{II}^\dagger) \det(A_{BB}^\dagger - A_{IB}^\dagger {A_{II}^\dagger}^{-1} A_{BI}^\dagger) \\ &= \det(A_{II}) \det(S) \det(A_{II}^\dagger) \det(S^\dagger) \\ &= \int \mathcal{D}\chi_I^* \mathcal{D}\chi_I \mathcal{D}\chi_B^* \mathcal{D}\chi_B \exp\left( -\chi_I^* (A_{II}^\dagger A_{II})^{-1} \chi_I - \chi_B^* (S^\dagger S)^{-1} \chi_B \right). \end{aligned} \tag{7.7}$$

Pulling the factor $A_{BB}$ out of the determinant does not seem of advantage a priori in this case, because $A_{II}$ and $A_{BB}$ describe very different structures. As mentioned above, the advantages of this method are an improved condition number of the matrix $A_{II}$ and its Schur complement, and the option to use multiple time-scale integration. Furthermore, with the Neumann-Neumann preconditioner we have a good preconditioner for the Schur complement, which we can use for its inversion in each HMC step.

# Chapter 8

# Conclusions

In this thesis we studied various aspects of computer simulations of field theories in general or QCD in particular. We focused on three major factors which contribute to the cost of simulation algorithms.

In Part I we covered three variants of preconditioners for Krylov subspace inverters which had not been applied to QCD so far: Schwarz methods with non-minimal overlap, optimized Schwarz methods, and a Schur complement method based on a non-overlapping domain decomposition. We found that neither of the latter two performs better than the Schwarz method with minimal overlap and Dirichlet boundary conditions, as commonly used in QCD. We extended this method to non-minimal overlap, which is also well-known, but had previously not been used for QCD. As expected from other linear problems, this extension yields a better convergence rate, but the algorithmic cost for convergence usually rises with increasing overlap if the number of processors is kept fixed.

However, the strong scaling behavior of QCD, i.e., how the time-to-solution varies with the number of processors for a fixed global lattice size, gives a practical upper bound for the number of processors: decreasing the local (or subdomain) volume below a certain limit makes it impossible to hide network latencies and thus increases the time-to-solution. Here, the Schwarz method with non-minimal overlap could come to the rescue. Assume that we use the additive Schwarz method with minimal overlap distributed to $N$ processors. If the number of processors and subdomains is increased to $N'$, we can at the same time decrease the spread and keep the subdomain volume constant. This implies an increasing overlap. As the additive Schwarz method uses only nearest-neighbor communication and as we kept the subdomain volume constant, the wall-clock time per iteration will be the same in the two cases. Since the error reduction per iteration is typically larger for more overlap, the method with non-minimal overlap (on $N'$ processors) converges faster (in terms of wall-clock time) than the method with minimal overlap (on $N$ processors). As a consequence, we can scale QCD to a larger number of processors and still obtain a reduction of the wall-clock time.

Since the inversion algorithms typically contain also global operations like dot-products, a case-by-case study for specific architectures has to show if this scaling is applicable in practice.

In Part II we introduced two new methods for computing the sign function of complex matrices. Both of them perform similarly well and can yield a considerable

speed improvement over existing algorithms. With minor modifications, the methods can also be transferred to other matrix functions. Just as most inverters, these two methods use Krylov subspaces and thus are severely hampered by low modes. In our benchmarks we used exact deflation techniques to deal with this. It is unclear whether these are feasible for very large lattices and small quark masses.

In view of the good results with inexact deflation methods for inversions, one would like to be able use them also for the sign function, but to our current knowledge this is not possible.

Part III discussed a modification of HMC based on a linear transformation which exposes different physical length scales and thus permits a better tuning of the HMC parameters. This allows for the use of fewer steps at the same trajectory length, i.e., it reduces the autocorrelation time (if measured in units of integration steps). In spite of its simple nature, we could however not carry the method over to gauge theories: preservation of the gauge invariance of the algorithm leads to discretized equations that cannot be properly solved.

In summary, this thesis added a few more puzzle pieces to the big picture of the efficient simulation of lattice field theories. All pieces encounter similar obstacles: different scales in the underlying problem yield a bad algorithm performance and not even dedicated techniques like preconditioners with local inversions can resolve this issue entirely. Presumably, methods relying on low modes can provide remedy, but these are barely out off their infancy. An improvement of these methods is likely to have a positive influence on different simulation parts: inversions, Hamiltonian time evolution and the overlap operator. Only a comprehensive understanding of all involved and related algorithms can bring us there.

# Chapter 9

# Acknowledgements

I would finally like to thank those people who made this work possible.

I am grateful to Tilo Wettig for guiding my steps in this field and for giving me the opportunity to do my research at the University of Regensburg. In particular, I want to emphasize his trust in me to work independently and without deadlines for individual subprojects. The resulting freedom allowed — and forced — me to acquire ample knowledge and comprehension of the field and to learn how to conduct research on my own.

I also thank Falk Bruckmann who supervised me in the initial phase of my PhD. Though I eventually took a different research path, working with Falk on supersymmetry on the lattice taught me interesting aspects of computer simulations of field theories and raised my interest in the topics covered in this work.

A really big thanks goes to my office mate Jacques Bloch. Countless discussions with him — partially on joint work like the nested Krylov methods, but mainly on work I did on my own — were of tremendous help. With his vast experience in the field he could provide deeper insights, especially related to numerics and algorithms. His answers to hundreds of small questions I had during the past years also helped me a lot. Furthermore, his encouraging, partially grim, humor made it easier for me to go on despite many negative results in various of my projects.

Some people provided their help for individual chapters of this work. In particular I thank Andreas Frommer and Axel Klawonn for advice and discussions on the Schur complement inverters for non-overlapping domain decompositions. Jacques Bloch and me would like to thank Andreas Frommer and Tilo Wettig for useful discussions on the nested Krylov subspace methods for the sign function. Furthermore, I want to thank Florian Gruber, a friend and PhD colleague, for the enjoyable time spent together in this department and many on and off-topic discussions. In particular I want to point out the many coffee breaks, during one of which a discussion on topological objects gave me the initial idea for the cm-HMC algorithm.

Last but not least, I am grateful to my family. My parents, who supported me and my interest in physics without questioning it, even in phases where it was hard, because I made no progress and failed to explain what it is that I am actually doing. My girlfriend Iulia, who gave a lot of support in the phase of writing this thesis, with advice drawn from experience with her own thesis.

# Appendix A

# Memory consumption for domain decompositions

We consider a QCD problem on a parallel computer and assume that the local volume $V$ is $8^4$ (the subvolume assigned to a processor or to a core of a processor). This is a typical value for many current machines and lattice sizes. For simplicity we do not distinguish between cores and processors.

We study the storage requirements for a domain decomposition algorithm, where the subdomain problems can be solved either iteratively or directly. All numbers given below assume double precision, i.e., 16 Byte per complex value. The gauge fields on the $8^4$ subvolume need

$$\begin{aligned} V_{\text{gauge}} &= V \cdot N_c^2 \cdot N_d \cdot 16 \text{ Byte} \\ &= 8^4 \cdot 3^2 \cdot 4 \cdot 16 \text{ Byte} = 2304 \text{ kByte}. \end{aligned} \tag{A.1}$$

A spinor needs

$$\begin{aligned} V_{\text{spinor}} &= V \cdot N_c \cdot N_s \cdot 16 \text{ Byte} \\ &= 8^4 \cdot 3 \cdot 4 \cdot 16 \text{ Byte} = 768 \text{ kByte}. \end{aligned} \tag{A.2}$$

Typical iterative methods need 3 to 5 spinors, so the total memory requirement for an iterative inversion of the $8^4$ subdomain matrix is, e.g.,

$$V_{\text{iterative}} = 5 \cdot 768 \text{ kByte} + 2304 \text{ kByte} = 6 \text{ MByte}. \tag{A.3}$$

This is of similar order as the typical cache size of a few megabytes in current processors.

A direct inversion of the local $8^4$ is definitely not feasible due to the required computing power and memory size: storing a full matrix corresponding to the $8^4$ subvolume needs

$$V_{\text{full}} = (V \cdot N_c \cdot N_s)^2 \cdot 16 \text{ Byte} = 36 \text{ GByte} \tag{A.4}$$

of memory. Instead we can consider using the smallest possible domains (excluding side lengths of 1): $2^4$. Storing such a matrix requires

$$V_{2^4} = (2^4 \cdot 3 \cdot 4)^2 \cdot 16 \text{ Byte} = 576 \text{ kByte}, \tag{A.5}$$

but storing all $4^4$ full submatrices which make up the local volume requires

$$V_{\text{direct}} = 4^4 \cdot V_{2^4} = 144 \text{ MByte}. \tag{A.6}$$

While this could be handled on current computers, it is by a factor 24 more than for an iterative inversion, far beyond the available cache size, and will thus make memory access a severe bottleneck.

# Appendix B

# Propagation of error: quadratic case

When studying measurements describing a one-dimensional function that exhibits a minimum, the value and statistical errors at the minimum are of particular interest. If the underlying function is not known we can try to fit the data with a parabola near the minimum, i.e., with

$$f(x) = c_1 x^2 + c_2 x + c_3. \tag{B.1}$$

This three-parameter fit leads to a variance-covariance matrix

$$\Sigma_c = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{12} & \sigma_2^2 & \sigma_{23} \\ \sigma_{13} & \sigma_{23} & \sigma_3^2 \end{pmatrix}, \tag{B.2}$$

where $\sigma_{ij}$ is the covariance of $c_i$ and $c_j$. We rewrite the function $f$ as

$$f = \sum_{i=1}^{3} c_i x_i = \mathbf{c} \cdot \mathbf{x} \tag{B.3}$$

with

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x^2 \\ x \\ 1 \end{pmatrix}. \tag{B.4}$$

The square of the standard deviation on $f$ is then given by

$$\begin{aligned} \sigma_f^2 &= \mathbf{x}^T \Sigma_c \mathbf{x} \\ &= \begin{pmatrix} x^2 & x & 1 \end{pmatrix} \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{12} & \sigma_2^2 & \sigma_{23} \\ \sigma_{13} & \sigma_{23} & \sigma_3^2 \end{pmatrix} \begin{pmatrix} x^2 \\ x \\ 1 \end{pmatrix} \\ &= \sigma_1^2 x^4 + 2\sigma_{12} x^3 + (2\sigma_{13} + \sigma_2^2) x^2 + 2\sigma_{23} x + \sigma_3^2. \end{aligned} \tag{B.5}$$

This gives the error on $f(x)$ for any $x$. The variance at the minimum $x_0 = -c_2/(2c_1)$ is

$$\sigma_f^2(x_0) = \sigma_1^2 \frac{c_2^4}{16 c_1^4} - 2\sigma_{12} \frac{c_2^3}{8 c_1^3} + (2\sigma_{13} + \sigma_2^2) \frac{c_2^2}{4 c_1^2} - 2\sigma_{23} \frac{c_2}{2 c_1} + \sigma_3^2. \tag{B.6}$$

# Appendix C

# Algorithms for nested Krylov subspace methods

## C.1 Lanczos algorithm

$v_1 \leftarrow \frac{x}{|x|}$
$r \leftarrow Av_1$
**for** $j = 1$ to $k$ **do**
    $H(j,j) \leftarrow v_j^\dagger r$
    $r \leftarrow r - H(j,j)v_j$
    **if** $j = k$ **then**
        stop
    **end if**
    $\beta \leftarrow \sqrt{r^\dagger r}$
    $H(j,j+1) \leftarrow \beta$
    $H(j+1,j) \leftarrow \beta$
    $v_{j+1} \leftarrow \frac{1}{\beta} r$
    $r \leftarrow Av_{j+1}$
    $r \leftarrow r - \beta v_j$
**end for**

All $H(i,j)$ not assigned above are zero. Consequently $H$ is tridiagonal and symmetric. The $v_j$ are the column vectors of the matrix $V_k$.

## C.2 Two-sided Lanczos algorithm

$v_1 \leftarrow \frac{x}{|x|}$
$w_1 \leftarrow v_1$
$r \leftarrow Av_1$
$l \leftarrow A^\dagger w_1$
**for** $j = 1$ to $k$ **do**
    $H(j,j) \leftarrow w_j^\dagger r$
    $r \leftarrow r - H(j,j)v_j$
    $l \leftarrow l - (H(j,j))^* w_j$
    **if** $j = k$ **then**
        stop

**end if**
$\delta \leftarrow r^\dagger l$
**if** $\delta = 0$ **then**
serious breakdown, stop
**end if**
$\beta \leftarrow \sqrt{\delta}$
$H(j+1, j) \leftarrow \beta$
$\gamma \leftarrow \frac{\delta^*}{\beta}$
$H(j, j+1) \leftarrow \gamma$
$v_{j+1} \leftarrow \frac{1}{\beta} r$
$w_{j+1} \leftarrow \frac{1}{\gamma^*} l$
$r \leftarrow A v_{j+1}$
$l \leftarrow A^\dagger w_{j+1}$
$r \leftarrow r - \gamma v_j$
$l \leftarrow l - \beta^* w_j$
**end for**

The $v_j$ and $w_j$ are the column vectors of the matrices $V_k$ and $W_k$, respectively. All $H(i, j)$ not assigned above are zero. Consequently $H$ is tridiagonal, but not symmetric as in the Hermitian case. The coefficients $\beta$ and $\gamma$ are, non-uniquely, chosen to satisfy the biorthonormality condition

$$w_j^\dagger v_i = \delta_{ij}. \tag{C.1}$$

There are potential problems in the two-sided Lanczos process, namely serious breakdowns and near breakdowns, where $\delta \leftarrow r^\dagger l = 0$, respectively $\approx 0$, however, these were not encountered in our numerical tests.

## C.3 Nested algorithm

Given a (non-)Hermitian matrix $A$, a source vector $x$ and the critical eigenvectors $r_i$ (left and right eigenvectors $l_i$ and $r_i$), with eigenvalues $\lambda_i$, $i = 1, \ldots, m$, do:

1. Apply Left-Right deflation (see Ref.[6]) to construct $x_\ominus$, where the components of the source vector $x$ along the eigenvectors $r_i$ have been removed:

$$x_\ominus = x - \sum_{i=1}^{m} \langle l_i, x \rangle r_i,$$

   where $l_i = r_i$ for Hermitian $A$.

2. Run the (two-sided) Lanczos algorithm from C.1 (C.2) with $A$ and $x_\ominus$ to obtain $V_k$ and $H_k$.

3. Perform an $LU$ decomposition of $pH_k$, e.g., with the LAPACK routine dgttrf (zgttrf). This yields a lower triangular matrix $L$ with unit diagonal and one

sub-diagonal, and an upper triangular matrix $U$ with one diagonal and two super-diagonals. All other entries of $L$ and $U$ are zero.

4. Run the (two-sided) Lanczos algorithm with $H'_k = (pH_k + (pH_k)^{-1})/2$ and source vector $e_1^{(k)}$ to construct the Krylov basis $V_\ell$ and the Ritz matrix $H_\ell$. To do so apply $H'_k$ to each Krylov vector $v$:
   (a) Compute $(pH_k)^{-1}v$ using a sparse LU back substitution, e.g., with the LAPACK routine dgttrs (zgttrs).
   (b) Compute $(pH_k)v$ and add to the result of (a). This tridiagonal multiply and add can be done efficiently using the BLAS band-matrix-vector multiplication routine dsbmv (zgbmv).
5. Run the RHi (or any other suitable method to compute the sign function) on $H_\ell$ to obtain $\operatorname{sgn}(H_\ell)$.
6. The final approximation is then given by

$$\operatorname{sgn}(A)x \approx \sum_{i=1}^{m} \operatorname{sgn}(\lambda_i)\langle l_i, x\rangle r_i + |x_\ominus| V_k V_\ell \operatorname{sgn}(H_\ell) e_1^{(\ell)}.$$

Note that steps (3-5) are done in real arithmetic in the Hermitian case.

# References


[1] R. Babich, J. Brannick, R. Brower, M. Clark, T. Manteuffel, et al., *Adaptive multigrid algorithm for the lattice Wilson-Dirac operator*, *Phys. Rev. Lett.* **105** (2010) 201602, [arXiv:1005.3043].

[2] G. Bali, P. Bruns, S. Collins, M. Deka, B. Gläßle, et al., *Nucleon mass and sigma term from lattice QCD with two light fermion flavors*, *Nucl. Phys.* **B866** (2013) 1–25, [arXiv:1206.7034].

[3] G. Batrouni, G. Katz, A. S. Kronfeld, G. Lepage, B. Svetitsky, et al., *Langevin Simulations of Lattice Field Theories*, *Phys. Rev.* **D32** (1985) 2736.

[4] J. C. R. Bloch, T. Breu, A. Frommer, S. Heybrock, K. Schäfer, et al., *Short-recurrence Krylov subspace methods for the overlap Dirac operator at nonzero chemical potential*, *Comput. Phys. Commun.* **181** (2010) 1378–1387, [arXiv:0910.1048].

[5] J. C. R. Bloch, T. Breu, and T. Wettig, *Comparing iterative methods to compute the overlap Dirac operator at nonzero chemical potential*, *PoS* **LATTICE2008** (2008) 027, [arXiv:0810.4228].

[6] J. C. R. Bloch, A. Frommer, B. Lang, and T. Wettig, *An iterative method to compute the sign function of a non-Hermitian matrix and its application to the overlap Dirac operator at nonzero chemical potential*, *Comput. Phys. Commun.* **177** (2007) 933–943, [arXiv:0704.3486].

[7] J. C. R. Bloch and S. Heybrock, *A nested Krylov subspace method to compute the sign function of large complex matrices*, *Comput. Phys. Commun.* **182** (2011) 878–889, [arXiv:0912.4457].

[8] J. C. R. Bloch and T. Wettig, *Overlap Dirac operator at nonzero chemical potential and random matrix theory*, *Phys. Rev. Lett.* **97** (2006) 012003, [hep-lat/0604020].

[9] J. C. R. Bloch and T. Wettig, *Domain-wall and overlap fermions at nonzero quark chemical potential*, *Phys. Rev.* **D76** (2007) 114511, [arXiv:0709.4630].

[10] J. Brannick, C. Ketelsen, T. Manteuffel, and S. McCormick, *Least-squares finite element methods for quantum electrodynamics*, *SIAM J. Sci. Comput.* **32** (Feb., 2010) 398–417.

[11] T.-W. Chiu and T.-H. Hsieh, *A note on Neuberger's double pass algorithm*, *Phys. Rev.* **E68** (2003) 066704, [hep-lat/0306025].

[12] T.-W. Chiu, T.-H. Hsieh, C.-H. Huang, and T.-R. Huang, *A note on the Zolotarev optimal rational approximation for the overlap Dirac operator*, *Phys. Rev.* **D66** (2002) 114502, [hep-lat/0206007].

[13] M. Creutz, *Quarks, Gluons and Lattices*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, 1985.

[14] J. Côté, M. Gander, L. Laayouni, and S. Loisel, *Comparison of the Dirichlet-Neumann and optimal Schwarz method on the sphere*, in *Domain Decomposition Methods in Science and Engineering* (T. J. Barth, M. Griebel, D. E. Keyes, R. M. Nieminen, D. Roose, T. Schlick, R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, and J. Xu, eds.), vol. 40 of *Lecture Notes in Computational Science and Engineering*, pp. 235–242. Springer Berlin Heidelberg, 2005.

[15] C. Davies, G. Batrouni, G. Katz, A. S. Kronfeld, G. Lepage, et al., *Fourier acceleration in lattice gauge theories. I. Landau gauge fixing*, *Phys. Rev.* **D37** (1988) 1581.

[16] C. Davies, G. Batrouni, G. Katz, A. S. Kronfeld, G. Lepage, et al., *Fourier acceleration in lattice gauge theories. III. Updating field configurations*, *Phys. Rev.* **D41** (1990) 1953.

[17] V. Dolean and M. J. Gander, *Can the Discretization Modify the Performance of Schwarz Methods?*, in *Domain Decomposition Methods in Science and Engineering XIX* (Y. Huang, R. Kornhuber, O. Widlund, and J. Xu, eds.), vol. 78 of *Lecture Notes in Computational Science and Engineering*, pp. 117–124. Springer Berlin Heidelberg, 2011.

[18] V. Druskin and L. Knizhnerman, *Extended Krylov subspaces: Approximation of the matrix square root and related functions*, *SIAM J. Matrix Anal. Appl.* **19** (1998) 755–771.

[19] S. Duane, A. D. Kennedy, B. Pendleton, and D. Roweth, *Hybrid Monte Carlo*, *Phys. Lett.* **B195** (1987) 216–222.

[20] **SciDAC** Collaboration, R. G. Edwards and B. Joo, *The Chroma software system for lattice QCD*, *Nucl. Phys. Proc. Suppl.* **140** (2005) 832, [hep-lat/0409003].

[21] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, *SIAM Journal on Numerical Analysis* **20** (Apr., 1983) 345–357.

[22] A. L. Ferreira and R. Toral, *Hybrid Monte Carlo method for conserved-order-parameter systems*, *Phys. Rev. E* **47** (June, 1993) 3848–3851.

[23] A. Frommer, *BiCGstab(ℓ) for families of shifted linear systems*, *Computing* **70** (2003), no. 2 87–109.

[24] A. Frommer, K. Kahl, S. Krieg, B. Leder, and M. Rottmann, *Aggregation-based Multilevel Methods for Lattice QCD*, *PoS* **LATTICE2011** (2011) 046, [arXiv:1202.2462].

[25] A. Frommer, A. Nobile, and P. Zingler, *Deflation and Flexible SAP-Preconditioning of GMRES in Lattice QCD Simulation*, arXiv:1204.5463.

[26] E. Gallopoulos and Y. Saad, *On the parallel solution of parabolic equations*, in *Proceedings of the International Conference on Supercomputing 1989, Heraklion, Crete, June 5-9, 1989* (R. D. Groot, ed.), pp. 17–28, ACM press, 1989.

[27] M. Gander, *Optimized Schwarz methods*, *SIAM Journal on Numerical Analysis* **44** (2006), no. 2 699–731.

[28] M. Gander and F. Kwok, *Best Robin parameters for optimized Schwarz methods at cross points*, *SIAM Journal on Scientific Computing* **34** (2012), no. 4 A1849–A1879.

[29] M. J. Gander and F. Kwok, *Optimal interface conditions for an arbitrary decomposition into subdomains*, in *Domain Decomposition Methods in Science and Engineering XIX* (Y. Huang, R. Kornhuber, O. Widlund, and J. Xu, eds.), vol. 78 of *Lecture Notes in Computational Science and Engineering*, pp. 101–108. Springer Berlin Heidelberg, 2011.

[30] G. Golub and C. V. Loan, *Matrix Computations*. The John Hopkins University Press, 1989.

[31] J. Goodman and A. D. Sokal, *Multigrid Monte Carlo method. Conceptual foundations*, *Phys. Rev. D* **40** (Sep, 1989) 2035–2071.

[32] S. A. Gottlieb, W. Liu, D. Toussaint, R. Renken, and R. Sugar, *Hybrid Molecular Dynamics Algorithms for the Numerical Simulation of Quantum Chromodynamics*, *Phys.Rev.* **D35** (1987) 2531–2542.

[33] M. Hasenbusch and K. Jansen, *Speeding up lattice QCD simulations with clover improved Wilson fermions*, *Nucl. Phys.* **B659** (2003) 299–320, [hep-lat/0211042].

[34] M. Hasenbusch, *Speeding up the Hybrid Monte Carlo algorithm for dynamical fermions*, *Phys. Lett.* **B519** (2001) 177–182, [hep-lat/0107019].

[35] P. Hasenfratz and F. Karsch, *Chemical potential on the lattice*, *Phys. Lett.* **B125** (1983) 308.

[36] S. Heybrock, *Double-pass variants for multi-shift BiCGstab(ℓ)*, *PoS* **LATTICE2010** (2010) 322, [arXiv:1010.2592].

[37] M. D. Hoffman and A. Gelman, *The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo*, arXiv:1111.4246.

[38] **ALPHA** Collaboration, K. Jansen and R. Sommer, *$O(\alpha)$ improvement of lattice QCD with two flavors of Wilson quarks*, *Nucl. Phys.* **B530** (1998) 185–203, [hep-lat/9803017].

[39] G. Katz, G. Batrouni, C. Davies, A. S. Kronfeld, P. Lepage, et al., *Fourier acceleration in lattice gauge theories. II. Matrix inversion and the quark propagator*, *Phys. Rev.* **D37** (1988) 1589.

[40] A. D. Kennedy, *Fast evaluation of Zolotarev coefficients*, hep-lat/0402038.

[41] L. Knizhnerman and V. Simoncini, *A new investigation of the extended Krylov subspace method for matrix function evaluations*, *Numer. Linear Algebra Appl.* **17** (2010), no. 4 615–638.

[42] J. Liesen and P. Tichý, *Convergence analysis of Krylov subspace methods*, *GAMM-Mitt.* **27** (2004), no. 2 153–173.

[43] M. Lüscher, *Lattice QCD and the Schwarz alternating procedure*, *JHEP* **0305** (2003) 052, [hep-lat/0304007].

[44] M. Lüscher, *Solution of the Dirac equation in lattice QCD using a domain decomposition method*, *Comput. Phys. Commun.* **156** (2004) 209–220, [hep-lat/0310048].

[45] M. Lüscher, *Schwarz-preconditioned HMC algorithm for two-flavour lattice QCD*, *Comput. Phys. Commun.* **165** (2005) 199–220, [hep-lat/0409106].

[46] M. Lüscher, *Implementation of the lattice Dirac operator*, *distributed with the DD-HMC software package* (2006).

[47] M. Lüscher, *Local coherence and deflation of the low quark modes in lattice QCD*, *JHEP* **0707** (June, 2007) 081, [arXiv:0706.2298].

[48] M. Lüscher, *Computational strategies in lattice QCD*, in *Modern Perspectives in Lattice QCD: Quantum Field Theory and High Performance Computing* (L. Lellouch, R. Sommer, B. Svetitsky, A. Vladikas, and L. F. Cugliandolo, eds.), vol. Les Houches 2009, Session XCIII, pp. 331–399, 2009.

[49] N. Madras and A. D. Sokal, *The Pivot algorithm: A highly efficient Monte Carlo method for the self-avoiding walk*, *J. Statist. Phys.* **50** (1988) 109–186.

[50] T. P. A. Mathew, *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations*. Springer-Verlag Berlin Heidelberg, 2008.

[51] H. B. Meyer, H. Simma, R. Sommer, M. Della Morte, O. Witzel, et al., *Exploring the HMC trajectory-length dependence of autocorrelation times in lattice QCD*, *Comput. Phys. Commun.* **176** (2007) 91–97, [hep-lat/0606004].

[52] I. Montvay and G. Münster, *Quantum Fields on a Lattice*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, 1997.

[53] R. B. Morgan and W. Wilcox, *Deflated iterative methods for linear equations with multiple right-hand sides*, `math-ph/0405053`.

[54] R. Narayanan and H. Neuberger, *Chiral determinant as an overlap of two vacua*, *Nucl. Phys.* **B412** (1994) 574–606, [`hep-lat/9307006`].

[55] R. Narayanan and H. Neuberger, *A Construction of lattice chiral gauge theories*, *Nucl. Phys.* **B443** (1995) 305–385, [`hep-th/9411108`].

[56] R. Neil, *MCMC using Hamiltonian dynamics*, in *Handbook of Markov Chain Monte Carlo* (S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, eds.), Handbooks of Modern Statistical Methods, ch. 5, pp. 113–162. Chapman and Hall/CRC, 2011.

[57] H. Neuberger, *A practical implementation of the overlap-Dirac operator*, *Phys. Rev. Lett.* **81** (1998) 4060–4062, [`hep-lat/9806025`].

[58] H. Neuberger, *Exactly massless quarks on the lattice*, *Phys. Lett.* **B417** (1998) 141–144, [`hep-lat/9707022`].

[59] H. Neuberger, *Minimizing storage in implementations of the overlap lattice Dirac operator*, *Int. J. Mod. Phys.* **C10** (1999) 1051–1058, [`hep-lat/9811019`].

[60] J. Osborn, R. Babich, J. Brannick, R. Brower, M. Clark, et al., *Multigrid solver for clover fermions*, *PoS* **LATTICE2010** (2010) 037, [`arXiv:1011.2775`].

[61] M. Peskin and D. Schroeder, *An Introduction To Quantum Field Theory*. The Advanced Book Program. Basic Books, 1995.

[62] J. Roberts, *Linear model reduction and solution of the algebraic Riccati equation by use of the sign functions*, *Internat. J. Control* **32** (1980) 677–687.

[63] H. Rothe, *Lattice Gauge Theories: An Introduction (Third Edition)*. World Scientific Lecture Notes in Physics. World Scientific, 2005.

[64] Y. Saad, *Analysis of some Krylov subspace approximations to the matrix exponential operator*, *SIAM J. Numer. Anal.* **29** (1992), no. 1 209–228.

[65] H. A. Schwarz, *Über einen Grenzübergang durch alternierendes Verfahren*, in *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, vol. 15, pp. 272–286. 1870.

[66] S. Schäfer, R. Sommer, and F. Virotta, *Critical slowing down and error analysis in lattice QCD simulations*, *Nucl. Phys.* **B845** (Sept., 2011) 93–119, [`arXiv:1009.5228`].

[67] J. Sexton and D. Weingarten, *Hamiltonian evolution for the Hybrid Monte Carlo algorithm*, *Nucl. Phys.* **B380** (1992), no. 3 665–678.

[68] B. Sheikholeslami and R. Wohlert, *Improved Continuum Limit Lattice Action for QCD with Wilson Fermions*, *Nucl. Phys.* **B259** (1985) 572.

[69] N. Shibata, *Efficient evaluation methods of elementary functions suitable for SIMD computation.*, *Computer Science - R&D* **25** (2010), no. 1-2 25–32.

[70] V. Simoncini and D. B. Szyld, *Recent computational developments in Krylov subspace methods for linear systems*, *Numerical Linear Algebra with Applications* **14** (2007) 1–59.

[71] G. L. G. Sleijpen and D. R. Fokkema, *BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum*, *Electronic Transactions on Numerical Analysis* **1** (1993) 11–32.

[72] B. Smith, P. Bjorstad, and W. Gropp, *Domain Decomposition — Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 2004.

[73] A. St-Cyr, M. Gander, and S. Thomas, *Optimized multiplicative, additive, and restricted additive Schwarz preconditioning*, *SIAM Journal on Scientific Computing* **29** (2007), no. 6 2402–2425.

[74] T. Takaishi and P. de Forcrand, *Testing and tuning new symplectic integrators for Hybrid Monte Carlo algorithm in lattice QCD*, *Phys. Rev.* **E73** (2006) 036706, [hep-lat/0505020].

[75] R. Toral and A. Ferreira, *Generalized Hybrid Monte Carlo*, hep-lat/9409014.

[76] J. van den Eshof, A. Frommer, T. Lippert, K. Schilling, and H. A. van der Vorst, *Numerical methods for the QCD overlap operator. I: Sign-function and error bounds*, *Comput. Phys. Commun.* **146** (2002) 203–224, [hep-lat/0202025].

[77] P. K. W. Vinsome, *ORTHOMIN — an iterative method for solving sparse sets of simultaneous linear equations*, in *Proceedings Fourth SPE Symposium on Reservoir Simulation, Los Angeles*, pp. 149–160, 1976.

[78] J. A. Vogel, *Flexible BiCG and flexible BiCGstab for nonsymmetric linear systems*, *Applied Mathematics and Computation* **188** (2007), no. 1 226–233.

[79] K. G. Wilson, *Confinement of Quarks*, *Phys. Rev.* **D10** (1974) 2445–2459.

[80] **ALPHA** Collaboration, U. Wolff, *Monte Carlo errors with less errors*, *Comput. Phys. Commun.* **156** (2004) 143–153, [hep-lat/0306017].

[81] F. Zhang, ed., *The Schur complement and its applications.* Numerical Methods and Algorithms. Springer-Verlag, 2005.

[82] E. I. Zolotarev, *Application of elliptic functions to the question of functions deviating least and most from zero*, *Zap. Imp. Akad. Nauk. St. Petersburg* **30** (1877) 5.