Towards Exascale Computing for Lattice QCD

Masterarbeit in Physik

Martin Ueding
mu@martin-ueding.de

angefertigt im
Helmholtz-Institut für Strahlen- und Kernphysik

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Universität Bonn

August 2017

1. Gutachter: Professor Carsten Urbach
2. Gutachter: Professor Thomas Luu

Contents

1. Introduction

Physics currently describes nature in terms of four fundamental forces: Gravity, electromagnetism, the weak force, and the strong force. Einstein's theory of general relativity describes gravity in terms of differential geometry, the standard model of particle physics describes the other three forces in terms of quantum field theories.

This work concentrates on the strong force described by quantum chromodynamics (QCD). It is the force that binds the quarks to protons and neutrons and those to atomic nuclei. The amalgamations of quarks are called *hadrons*, pairs of a quark and anti-quark are called *mesons* and triplets of all quarks are called *baryons*. While electromagnetism knows a single charge, QCD works with three charges, whimsically labeled "red", "green", and "blue". In contrast to gravity and electromagnetism, the strong force is increasing for large distances, leading to *confinement*. Only objects with no net color charge can be free. Thus mesons contain quarks with color and anti-color, the baryons contain one of each color; both are color neutral. Individual quarks cannot be observed directly.

The emergent structure of QCD, manifest in the interaction of hadrons, can be described with *chiral perturbation theory* (χPT or ChPT), albeit with some numerical factors known as *low energy constants* (LEC) that must be derived from experiment. It is not directly possible to compute those LECs analytically from QCD. Large simulations of the underlying theory make it possible to estimate these LECs and see whether QCD really is the underlying theory of the hadrons.

The precision of these numerical experiments is limited by the available computing power. We will first look at the foundations of *Lattice QCD* and then at a particular simulation of the full theory. The later chapters cover the optimizations needed to make best use of the available computing resources. In the appendix, the reader will find rather technical information about the tools used in this thesis.

2. Lattice Gauge Theory

This chapter will introduce the needed theoretical foundations to simulate QCD on a computer. It is assumed that the reader has a basic understanding of "normal" continuum QCD and the path integral.

2.1. Quantum Chromodynamics

Quantum chromodynamics is the current theory of the strong force. It is a quantum field theory (QFT) describing the quarks as N_f flavors of spin-½ fermions and gluons as massless spin-1 force carriers. The gauge group is SU(3) as we observe $N_\mathrm{c} = 3$ colors in experiments.

All information about the theory is contained in the Lagrange density [1]

$$L = \frac{1}{2} \operatorname*{Tr}_{\text{color}} \left(\boldsymbol{G}_{\mu\nu} \boldsymbol{G}^{\mu\nu} \right) + \bar{\boldsymbol{\psi}} \left(\mathrm{i} \not{\boldsymbol{D}} - \boldsymbol{m} \right) \boldsymbol{\psi} \,, \tag{2.1}$$

with the gluon field strength tensor $\boldsymbol{G}_{\mu\nu}$, the spinor field vector of quark flavors $\boldsymbol{\psi}$, the gauge covariant derivative $\mathbf{D}$, and the mass matrix $\boldsymbol{m}$ in flavor space. Summation over repeated indices is implied. The various quantities are explained in the following list.

Gauge Field The gauge field $\boldsymbol{A}$ is a 1-form in the $\mathfrak{su}(N_\mathrm{c})$ algebra (adjoint representation) and also a Lorentz vector field with components A^a_μ where μ is a Lorentz index ($\mu = 0, \ldots, 3$) and a is an $\mathfrak{su}(N_\mathrm{c})$ algebra index ($a = 1, \ldots, N_\mathrm{c}^2 - 1$).

Field Strengh Tensor The field strength tensor $\boldsymbol{G}$ is the exterior derivative, $\boldsymbol{G} = \mathrm{d}\boldsymbol{A} + \boldsymbol{A} \wedge \boldsymbol{A}$ and therefore gauge algebra valued and a two-form over spacetime.

Covariant Derivative The covariant derivative $\mathbf{D}$ is the connection, in components it is $\mathrm{D}^{ij}_\mu = \partial_\mu \delta^{ij} + \mathrm{i} A^{ij}_\mu$ where $A^{ij}_\mu = A_{\mu a} \lambda^{ij}_a$ is the expression contracted with the $\mathrm{SU}(N_\mathrm{c})$ generators $\boldsymbol{\lambda}^a$ (fundamental representation). The "D slash"

operator $\not{\boldsymbol{D}}$ is defined as $\not{\boldsymbol{D}} = \mathrm{D}_\mu \boldsymbol{\gamma}^\mu$ where the γ-matrices are the generators of the Dirac algebra, equivalent to quaternions.

Spinor Field The spinor field $\boldsymbol{\psi}$ is a Grassmann number valued complex vector fields with color (i, j), spin-Lorentz indices (α, β), and flavor indices (f, f'). The adjoint field $\bar{\boldsymbol{\psi}}$ is defined as $\boldsymbol{\psi}^\dagger \boldsymbol{\gamma}^0$.

With all indices explicitly written out, the Lagrange density becomes

$$L(\boldsymbol{x}) = \frac{1}{2} G^{ab}_{\mu\nu}(\boldsymbol{x})\, G^{\mu\nu ba}(\boldsymbol{x}) + \bar{\psi}^i_{\alpha f}(\boldsymbol{x}) \Big[\mathrm{i}\gamma^\mu_{\alpha\beta} \Big[\partial_\mu \delta^{ij} - A^{ij}_\mu(\boldsymbol{x}) \Big] \delta_{ff'} - \delta^{ij} \delta_{\alpha\beta} m_{ff'} \Big] \psi^i_{\gamma f'}(\boldsymbol{x}) .$$

Observables can formally be computed using the path integral formalism [2]. The vacuum expectation value of an operator $\hat{O}$ in time ordering T is given by

$$\langle 0 | T \hat{O} | 0 \rangle = \frac{\int \mathscr{D}\boldsymbol{A}\, \mathscr{D}\bar{\boldsymbol{\psi}}\, \mathscr{D}\boldsymbol{\psi}\, O(\boldsymbol{A}, \boldsymbol{\psi}, \bar{\boldsymbol{\psi}}) \exp(\mathrm{i}S(\boldsymbol{A}, \boldsymbol{\psi}, \bar{\boldsymbol{\psi}}))}{\int \mathscr{D}\boldsymbol{A}\, \mathscr{D}\bar{\boldsymbol{\psi}}\, \mathscr{D}\boldsymbol{\psi} \exp(\mathrm{i}S(\boldsymbol{A}, \boldsymbol{\psi}, \bar{\boldsymbol{\psi}}))} , \tag{2.2}$$

where the action S is given by $S = \int \mathrm{d}^4 x\, L(\boldsymbol{x})$. The notation $\mathscr{D}\boldsymbol{\psi}$ means an integration over all possible Grassmann number valued fields $\boldsymbol{\psi}(\boldsymbol{x})$. Only for very simple models this functional integration can be evaluated analytically, one is the harmonic oscillator.

In theories with weak coupling, like quantum electrodynamics (QED) or the high-energy sector of QCD, one can expand the exponential in terms of the coupling constant and derive the Feynman rules. With low-energy QCD this is not possible, one cannot make any expansion as the coupling parameter g_s (here defined into $\boldsymbol{A}$ and therefore contained in $\not{\boldsymbol{D}}$ and $\boldsymbol{G}_{\mu\nu}$) is not small below the energy scale Λ_QCD.

2.2. Euclidean Time

Numerically performing the integration in Equation (2.2) is not sensible, the integration space has many dimensions and the integrand is oscillating on the whole domain. One performs the *Wick rotation* and introduces imaginary time τ such that the relationship with regular time t is $t = -\mathrm{i}\tau$. Then the exponential factor will change from $\exp(\mathrm{i}S)$ to $\exp(-S)$ due to the $\mathrm{d}x^0$ in the integral over the

Lagrange density. This effectively is a Boltzmann weight and one can rewrite the path integral expectation value as

$$\langle O \rangle = \frac{1}{Z} \int \mathscr{D}\boldsymbol{A}\, \mathscr{D}\bar{\boldsymbol{\psi}}\, \mathscr{D}\boldsymbol{\psi}\, O(\boldsymbol{A}, \boldsymbol{\psi}, \bar{\boldsymbol{\psi}}) \exp(-S(\boldsymbol{A}, \boldsymbol{\psi}, \bar{\boldsymbol{\psi}})) \tag{2.3}$$

with the partition function

$$Z := \int \mathscr{D}\boldsymbol{A}\, \mathscr{D}\bar{\boldsymbol{\psi}}\, \mathscr{D}\boldsymbol{\psi}\, \exp(-S(\boldsymbol{A}, \boldsymbol{\psi}, \bar{\boldsymbol{\psi}}))\,. \tag{2.4}$$

The region with contributing configurations is now localized around the minimal action (classical path). The action S is real and bounded from below, therefore the weight can be written as a probability density function

$$P(\boldsymbol{A}, \boldsymbol{\psi}, \bar{\boldsymbol{\psi}}) = \frac{\exp(-S(\boldsymbol{A}, \boldsymbol{\psi}, \bar{\boldsymbol{\psi}}))}{Z}\,. \tag{2.5}$$

Sampling of such a distribution over a high dimensional space is best done with Monte Carlo methods and will be covered in Section 2.4. Before doing so on a computer, we will have to discretize the theory.

2.3. Discretizations

The configuration y_i of the degrees of freedom must be stored in computer memory. Therefore it is needed to discretize space and time. An uniform hyper-cubic lattice with spacing a is introduced such that $\boldsymbol{x} = a\boldsymbol{n}$ with $\boldsymbol{n} \in \mathbf{Z}^4$. The lattice cannot be infinite on the computer, it has to have a finite extent. Usually all three spatial extents $L_\mathrm{s} \in \mathbf{N}^+$ are chosen the same, the temporal extent $L_\mathrm{t} \in \mathbf{N}^+$ is often chosen larger. The whole lattice has a volume $V = L_\mathrm{s}^3 \cdot L_\mathrm{t}$. Periodic boundary conditions are imposed to obtain translational invariance of the system and to remove surface effects. All boundaries are periodic, except for fermions in the time direction, there the boundary is anti-periodic in order to allow relating lattice measurements back to Minkowski space [3].

The discretization of QCD is not unique. Multiple discretizations are possible, they must only agree with QCD in the continuum limit. Discretization errors will scale with $\mathrm{O}(a)$ or a higher power. We may use this freedom to introduce additional

(a) Forward (b) Backward

Figure 2.1.: Gauge links U are the parallel transporters of the gauge field between next neighbor lattice sites. The backwards direction is given by the hermitian conjugate.

terms that scale with (positive) powers of the lattice spacing a to remove some of the discretization errors.

2.3.1. Gauge Field

A direct discretization of QCD would give spinor fields $\psi(n)$ and gauge fields $A_\mu(n)$ as the fundamental degrees of freedom. One cannot, however, construct gauge invariant quantities using A_μ on the lattice. Only by using the parallel transporter between lattice sites,

$$U_\mu(n) := U(n, n+\hat{\mu}) = \exp\left(\mathrm{i} \int_n^{n+\hat{\mu}} \mathrm{d}n' \, A_\mu(n') \right),$$

one can construct invariant quantities. These *gauge link* matrices U and the fermion fields ψ are the fundamental degrees of freedom in the lattice gauge theory. The links are defined between next neighbors in the lattice, as shown in Figure 2.1. The fermions are defined on the lattice points, although there are subtleties depending on the discretization scheme used.

The *plaquette* is the parallel transporter around a 1×1 square in the lattice:

$$P_{\mu\nu}(n) := U_\mu(n)\, U_\nu(n+\hat{\mu})\, U_\mu^\dagger(n+\hat{\nu})\, U_\nu^\dagger(n) \,. \tag{2.6}$$

The trace of the plaquette is invariant under gauge transformations. See Figure 2.2 for a graphical illustration of the plaquette. It corresponds to the gluon field strength tensor and is used in the gauge part of the action to order a^2:

$$S_g = \beta \sum_{n,\mu,\nu} \left[1 - \frac{1}{2} \operatorname*{Tr}_{\text{color}} \left(P_{\mu\nu}(n) + P_{\mu\nu}^\dagger(n) \right) \right].$$

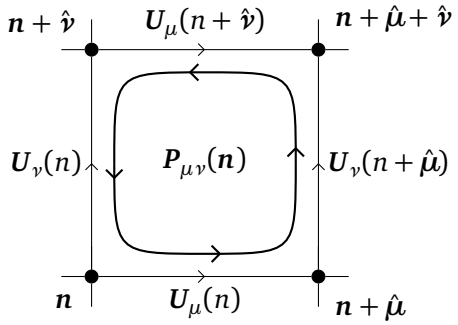


Figure 2.2.: The plaquette, see Equation (2.6), consisting of a square of four links.

One can lessen discretization errors by adding more plaquette-like quantities like rectangles to the action.

2.3.2. Wilson Fermions

The Dirac operator contains partial derivatives. On the lattice, those are expressed using difference quotients as introduced by Wilson [4]. We can introduce the forward and backward partial derivatives [5, p. 128],

$$\partial_\mu f(x) = \frac{f(x+a\hat{\mu}) - f(x)}{a}, \qquad \partial^*_\mu f(x) = \frac{f(x) - f(x-a\hat{\mu})}{a}.$$

With the gauge interaction, different points are gauged differently and the parallel transporter is needed to compare the two tangent spaces. The gauge covariant derivatives on the lattice are [5, p. 129]

$$\nabla_\mu \psi(x) = \frac{U_\mu(x)\psi(x+a\hat{\mu}) - \psi(x)}{a}$$
$$\nabla^*_\mu \psi(x) = \frac{\psi(x) - U^\dagger_\mu(x-a\hat{\mu})\psi(x-a\hat{\mu})}{a}.$$

Using these derivatives, we can build a symmetrized version of the gauge covariant derivatives to obtain the fermionic part of the action for a single flavor f,

$$\bar{\psi}_f \gamma_\mu \frac{\nabla_\mu + \nabla^*_\mu}{2} \psi_f .$$

The discretization of the fermions unfortunately is not this simple. Differential equations formulated using a symmetric discretized derivative might have a second solution that alternates its sign on the lattice sites.[1] In the continuum limit ($a \to 0$), those oscillating solutions are not sensible. For finite lattice spacing as used in the simulations, this becomes a problem. The other solution is a fermion *doubler* that has the same mass and is otherwise degenerate with the one fermion of interest. This happens for each dimension, so the simulation has 16 fermions where we only want a single one.

The No-Go-Theorem by Nielsen and Ninomiya [6] says that if one simulates a theory on a lattice, one will observe doublers iff the theory has (a) gauge invariance, (b) translational invariance, (c) locality, and (d) chiral symmetry in the massless limit. For doublers to be removed from the spectrum, at least one of the four assumptions of the theorem needs to be given up. The one which we could do without is the chiral symmetry. The Wilson term

$$-\bar{\psi}_f a \frac{r}{2} \nabla^*_\mu \nabla_\mu \psi_f$$

couples left-handed with right-handed particles and therefore breaks the chiral symmetry. By tuning the parameter r, called *Wilson parameter*, one can adjust the masses of the doublers. The choice in the literature is to set $r = 1$ and not write it out explicitly. In the continuum limit the doublers will become infinitely heavy and completely decouple. For finite lattice spacing the doublers will be heavy enough to not change the interesting part, the low lying spectrum. The breakage of chiral symmetry gives rise to new discretization errors that previously were forbidden by that symmetry.

Another disadvantage of this technique is the additive renormalization of the quark masses. In continuum QFT the renormalization is purely multiplicative. Here it will take a form like $Z(m_0 - m_{\mathrm{cr}})$. The critical mass is not known beforehand and has to be extracted from lattice data.

[1] This can be seen with a simple example given by Thomas Luu on a problem set from 2016. Take the differential equation $f'(x) = \lambda f(x)$ which has the solution $f(x) = f(0)\,\mathrm{e}^{\lambda x}$. Using a discretized derivative, this ODE becomes $f([n+1]a) - f([n-1]a) = 2a\lambda f(na)$ where a is the spacing and n labels the sites. Using the ansatz $f(na) = f(0)\,\mathrm{e}^{n\ln(g)}$ gives $g = \lambda a \pm \sqrt{(\lambda a)^2 + 1}$. Using $\ln(x+1) = x + \mathrm{O}(x^2)$ and $\ln(x-1) = \mathrm{i}\pi - x + \mathrm{O}(x^2)$ one can show that the extrapolated continuum solutions are $f(x) = f(0)\,\mathrm{e}^{\lambda x}$ and $f(x) = f(0)(-1)^n\,\mathrm{e}^{-\lambda x}$. At finite a, the two solutions are equivalent, in the limit $a \to 0$ the second solution is incorrect.

2.3.3. Reducing Discretization Errors

Due to finite computing resources we are bound to simulate with finite lattice spacing and extent. Discretization errors with $O(a)$ and $O(a^2)$ must be taken into account and a continuum extrapolation must be performed. Removing the $O(a)$ effects will make this extrapolation $a \to 0$ easier and allows for use of larger values of the lattice spacing a in the simulations.

The standard procedure is the *Symanzik improvement program* [5, 7] which adds higher order operators to the Lagrange density of the theory. The operators are of dimension five or higher and therefore scale as $O(a)$ and $O(a^2)$; they vanish in the continuum limit. One has to choose a linear combination of the operators such that the $O(a)$ coefficient vanishes. The $O(a^2)$ terms might have increased during this process. Also one pays with increased complexity of the discretized action to be simulated.

Clover Term In order to remove the $O(a)$ discretization errors, one includes the so-called "clover term" by Sheikholeslami and Wohlert [8]. This term reads

$$\mathrm{i}\frac{1}{4}c_{\mathrm{SW}}\bar{\boldsymbol{\psi}}_f \boldsymbol{\sigma}_{\mu\nu}\boldsymbol{F}_{\mu\nu}\boldsymbol{\psi}_f =: c_{\mathrm{SW}}\bar{\boldsymbol{\psi}}_f \boldsymbol{T}\boldsymbol{\psi}_f \,,$$

where c_{SW} is a parameter that needs to be tuned such that the $O(a)$ effects are properly cancelled. Figure 2.3 shows the discretized version of this term. In the tree level perturbation theory, the appropriate value for c_{SW} is 1.0. However, due to renormalization, the optimal value depends on the quark masses and the lattice spacing. Therefore one would need to run simulations with multiple lattice spacing and different clover coefficients. From extracted observables one would tune the coefficient until the lattice effects to not show up as $O(a)$ effects any more.

As one can see from the explicit lattice representation of $\boldsymbol{F}_{\mu\nu}$ [7, Eq. (6)], that term is antihermitian. Together with the imaginary unit, the whole clover term is hermitian. This means that the $\boldsymbol{\gamma}_5$ hermiticity of the kinetic operator ($\boldsymbol{M}^\dagger = \boldsymbol{\gamma}_5\boldsymbol{M}\boldsymbol{\gamma}_5$) is not spoiled by the addition of the clover term.

Twisted Mass As long as two quarks have the same mass, the theory has an SU(2) flavor symmetry. This is broken by different masses since the $\mathfrak{su}(2)$ generators (Pauli matrices) do not commute with the mass matrix any more. In a massless

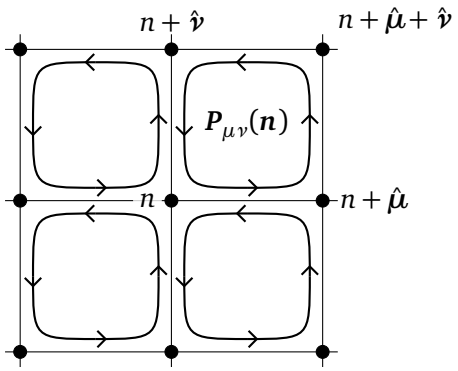


Figure 2.3.: Illustration of the clover term. Adapted from Jansen and Liu [7].

theory, there is full chiral symmetry. The addition of the Wilson term breaks this symmetry, but its orientation in flavor space can be chosen freely.

Automatic O(a) improvement can be obtained by using a twisted mass [5, 9, 10]. The idea is to introduce a mass that is the same for a doublet. Then one can rotate the spinor fields with a spin-flavor rotation, namely $\exp(\mathrm{i}\omega\gamma_5\tau^3/2)$ [5, p. 7] with *twist angle* ω, without changing physical observables. After the rotation, one introduces the Wilson term and then breaks the chiral symmetry.

In practice one sets $m_0 = m_{\mathrm{cr}}$ in order to get regular multiplicative renormalization. Then one introduces a new mass term, $\mathrm{i}\mu\gamma_5\tau^3$. This mass term is rotated in flavor space and is orthogonal to the Wilson term. Therefore one does not suffer from the O(a) discretization errors. This improvement is manifested in "correlation functions made of parity even multiplicatively renormalizable fields" [5, p. 35].

Due to the explicit two-flavor structure of τ^3, twisted mass can only describe a doublet of quarks. The addition of a *mass splitting term* like $-\epsilon\tau^1$ allows to simulate a non-degenerate doublet.

2.4. Generation of Configurations

2.4.1. Integrating out Fermion Fields

The Grassmann algebra of the fermionic fields cannot be implemented on the computer sensibly, they need to be removed. A Grassmann algebra with N numbers

can be represented with matrices with side length 2^N. A spinor field has $12V$ components (color, spin). For a small $V = 16^3 \times 32$ lattice this would be 1 572 864 components and therefore dimension in the order of $10^{473\,479}$. The matrices would be sparse and contain a limited amount of different non-zero entries. One example are the Dirac matrices which are sparse and only contain ±1 and ±i. Still, the needed bits vastly exceeds the number of baryons in the visible universe. An implementation of such a large Grassmann algebra using matrices is out of the question. A better approach is given by Creutz [11] but that would also not scale for the volumes that we are interested in.

A Gaussian integral with ordinary real numbers and a positive definite matrix $\boldsymbol{A}$ like $\int \mathrm{d}^n x\ \mathrm{d}^n y\ \exp(-x_i A_{ij} y_j)$ will give $1/\det(\boldsymbol{A})$. With Grassmann fields, the integral $\int \mathrm{d}^n \eta\ \mathrm{d}^n \theta\ \exp(-\eta_i A_{ij} \theta_j)$ will yield $\det(\boldsymbol{A})$. Integrating out the fermion fields $\bar{\boldsymbol{\psi}}$ and $\boldsymbol{\psi}$ will convert the $\exp(-\bar{\boldsymbol{\psi}} \boldsymbol{M} \boldsymbol{\psi})$ in the path integral into $\det(\boldsymbol{M})$. The Grassmann fields are integrated out and need no representation in the computer program. This is done for each flavor of fermions in the theory. A degenerate doublet of light quarks gives for instance $\det(\boldsymbol{M}_\mathrm{l})^2$.

One could evaluate this determinant in numerical simulations. The computation has cost of $\mathrm{O}(V^3)$ and is therefore very expensive. At the time where one did not have today's algorithms and computing power, one would simply set $\det(\boldsymbol{M}) = 1$ in order to perform the simulation at an acceptable speed on a reasonable lattice size. One calls this the *quenched approximation*. This is equivalent to removing the fermions from the action during the simulation. The fermions in the determinant are the sea fermions only. One can still compute correlation functions on the gauge configuration with valence fermions. Loop corrections are only computed for gluon loops, without sea fermions, no fermion loops are possible. Today there is a more efficient algorithm which will be introduced in Section 2.4.8. Computers have advanced as well, a simulation with quenched quarks can be written in a couple of weeks [12, 13] and reproduce 37 year old results like the ones from Creutz [14] on an average laptop.

Interestingly, the results obtained with the quenched approximation are not that far off the results with dynamical quarks. There are numerical instabilities introduced by this where extremal values in the other terms are not canceled by the determinant. This is another drawback of this approximation.

	Criterion			
Algorithm	Order	Exact	Ergodic	Year
Metropolis	V^4	yes	yes	1953
Langevin	V	no	yes	1981
Microcanonical	V	no	no	1982
Hybrid Monte Carlo	V	yes	yes	1987

Table 2.1.: Evolution of the algorithms towards an efficient, exact, and ergodic algorithm. [15, Table 1]

2.4.2. Development of Algorithms

With a suitable algorithm one can sample a probability distribution such that the more likely values are realized more often. This is called *importance sampling*. The expectation value of an operator then becomes the arithmetic mean of the N samples, $\langle O \rangle = \sum_{i=1}^{N} O_i / N$. In this section the basic building blocks of the hybrid Monte Carlo algorithm are introduced. The probability distribution that we want to sample is the weight in the path integral, given in Equation (2.5).

There are three important criteria for the algorithms used in lattice gauge theory. The algorithms developed can be categorized by these and it makes apparent why that algorithm was developed.

1. Computational complexity, the *order* in the volume V. As computer time is an expensive resource one always tries to make the most out of the allocated time. Scaling to larger volumes is needed for lighter quark masses. A high order in the volume makes this scaling prohibitively expensive.
2. Exactness. Any numerical algorithm can only be exact to machine precision. If an algorithm is free of systematic errors, it is called “exact”.
3. Ergodicity, the reversibility of time. This is required for the proof of the detailed balance condition that will be introduced below. Reversibility violations will introduce systematic errors.

Table 2.1 shows a summary of the algorithms presented here. We will look at the algorithms in detail in what follows.

2.4.3. Markov Chains

An effective means to obtain such a set of configurations distributed with respect to Equation (2.5) is a *Markov chain*. Elements y_i in this chain are related to their preceding element by a transition probability $P(y_i \to y_j) =: P_{ij}$. In order to maintain thermal equilibrium, we need to have *detailed balance*:

$$P(y_i \to y_j)\, \mathrm{e}^{-S_i} = P(y_j \to y_i)\, \mathrm{e}^{-S_j}\,. \tag{2.7}$$

The product of transition probability and occupation number is the total number of states changing from i to j. In the thermal equilibrium the occupation number must not change any more, therefore the number of configurations in one ensemble going from i to j must be exactly the same as the number of transitions back from j to i. The transition probability of the Markov chain needs to fulfill this detailed balance condition. There is a freedom in choosing the particular P_{ij}. Taking $P(y_i \to y_j) = \mathrm{e}^{-S_j}$ will fulfill detailed balance and is called the *heat bath algorithm*. For a general system it is very hard to sample the configuration space directly to obtain this transition probability.

2.4.4. Metropolis Algorithm

A more practical approach than the heat bath algorithm is the *Metropolis algorithm* [16]. It provides a computationally feasible way to update configurations such that detailed balance is maintained. An application to the harmonic oscillator is given by Creutz and Freedman [17].

The algorithm uses a transition probability of $P(y_i \to y_j) = \min(1, \mathrm{e}^{S_i - S_j})$. Inserting this into detailed balance condition, Equation (2.7), one directly finds that this algorithm fulfills it.[2] With this scheme one randomly picks a new configuration y_j and accepts it with the probability $\min(1, \mathrm{e}^{S_i - S_j})$. Otherwise it is rejected and the old configuration is added to the set of configurations another time. Then one repeats this procedure. The proposed configuration must be sampled in a way independent of the old configuration.

The generalization, the Metropolis-Hastings algorithm [18], allows to draw the proposed configuration from an instrumental distribution. In the acceptance step, the instrumental distribution needs to be included as well. The Metropolis

[2] Assume $S_i > S_j$. Then $\min(1, \mathrm{e}^{S_i - S_j}) = 1$ and $\min(1, \mathrm{e}^{S_j - S_i}) = \mathrm{e}^{S_j - S_i}$. Detailed balance is fulfilled: $1 \cdot \mathrm{e}^{-S_i} = \mathrm{e}^{S_j - S_i}\, \mathrm{e}^{-S_j}$.

algorithm is the special case where the instrumental density does not depend on the configuration, therefore the fraction just cancels in the acceptance probability. The heat bath has the instrumental distribution equal to the target density, therefore effectively removing the acceptance step.

In practice one wants to keep the acceptance rate around 70 %, therefore one samples close to the previous configuration y_i. Acceptance rates which are too high indicate that one does moves through phase space too slowly, therefore autocorrelation will be large. If the rate is too low, autocorrelation from duplicated configurations will be an issue.

The advantage of the Metropolis algorithm over the heat bath algorithm is that choosing the next candidate configuration is much easier. One does not have to sample according to e^{-S} which can be hard to impossible depending on S. Here one can just randomly choose a new candidate, at the cost of perhaps having to reject and try again. The acceptance step will also eliminate numerical errors like rounding or systematic effects from integrators. In order to maintain a high acceptance rate, one must ensure that changes in the action are not large. Choosing the new configuration not much different compared to the previous configuration will work, the price to pay are long autocorrelation times.

For a pure gauge theory this would be implemented by iterating through all the links and randomly choosing a new candidate for each link from $\mathrm{SU}(N_\mathrm{c})$. With a Metropolis step this new link may be accepted. One full sweep through all links would be one Monte Carlo iteration. When fermions are included, the fermionic determinant would have to be computed for each of the $\mathrm{O}(V)$ links in the lattice. The cost of a determinant is $\mathrm{O}(V^3)$, so in total the Metropolis algorithm for full QCD would need $\mathrm{O}(V^4)$ steps for one iteration [15, § 3.1]. This is too expensive to be useful on large lattices.

2.4.5. Pseudo Fermions

In order to make computation of the fermionic determinant cheaper, we use *pseudo fermions* [19, 20]. The idea is to move the determinant back into the exponential using a Gaussian integration with c-number fields instead of Grassmann fields. A positive operator is needed in order to interpret the exponential weight as a probability distribution. The fermionic operator $\boldsymbol{M}_\mathrm{l}$ is *not* positive definite in general. For the up and down quarks, we have almost equal masses and make the approximation that they are the same. Therefore we obtain $\det(\boldsymbol{M}_\mathrm{l})^2$ from

the Grassmann integration over the light quark fields ψ_u and ψ_d. Using that $\det(\gamma_5) = 1$, we can rewrite those two determinants like this:

$$\det(M_l)^2 = \det(M_l M_l) = \det(\gamma_5 M_l \gamma_5 M_l) = \det(M_l^\dagger M_l)\,.$$

Defining $Q_f^2 := M_f^\dagger M_f$ gives a positive operator as needed. In the action we introduce an isospin doublet of light quarks with a single (c-number valued) pseudo fermion field ϕ. That part of the action then is $\phi^*[Q_l^2]^{-1}\phi$, so the determinant is rewritten as $\int \mathcal{D}\phi^* \mathcal{D}\phi \exp(-\phi^*[Q_l^2]^{-1}\phi)$. The path integral of the bosonic fields ϕ will just produce $\det(Q^2)$ as desired. The cost of the pseudo fermions is that the expression $[Q^2]^{-1}\phi$ needs be computed during each evaluation of this monomial. This is done by solving the system $Q^2 x = \phi$, the major cost of lattice QCD calculations. Chapter 4 will focus on a library that provides a fast solver for this equation.

Just like the calculation of the determinant, the inversion of the operator is a lattice-global operation, but at a much reduced computational cost. Due to the construction, one can only simulate a degenerate doublet of quarks. For the light quarks, this is not a real limitation, for the strange and charm quarks, it is. Therefore a method for single quark fields is needed.

2.4.6. Rational Approximation

Integrating out a single strange quark in the action gives $\det(M_s)$. Pseudo fermions would give the same result if one simulates

$$\phi^* \sqrt{[Q^2]^{-1}}\phi$$

in the action. Just using $\phi^* Q^{-1}\phi$ is not a viable choice because Q^{-1} alone is not positive definite. In the case of twisted mass fermions, the operator M already is a two flavor operator that is diagonal in space. The down-flavor contains $-i\mu\gamma_5$ due to the τ^3, this minus sign could also be interpreted as complex conjugation. Therefore it is possible to interpret the two flavors of the twisted mass doublet as single-flavor operators M and $M^\dagger$ which naturally form Q^2. For the non-degenerate case $\epsilon \neq 0$, this is not possible, the square root of Q^2 (with Q a two-flavor operator) needs to be taken here as well in order to get a *single* non-degenerate doublet.

Taking the square root of a matrix operator must be implemented with some kind of expansion. One method is the polynomial approximation [21] where $\sqrt{A}$ is

approximated with a polynomial $P(\boldsymbol{A})$ which has a finite number of terms. An improved variant is the rational approximation [22] which will be introduced next.

The approximation for the square root needs to be sufficiently accurate over the range of eigenvalues of $\boldsymbol{Q}$. The first step therefore is to measure the eigenvalue spectrum with a thermalized configuration. For a simulation with new parameters this is a circular dependency (chicken–egg problem), one has to monitor the eigenvalue spectrum during the thermalization and adjust the eigenvalue bounds of the approximation as needed. The computation of all eigenvalues is expensive and also not needed, only the minimal and maximum eigenvalue are needed. In Chroma[3] one uses Ritz eigenvalues. These are computed as part of an Arnoldi iteration [23]. The Ritz eigenvalues always lie outside of the spectrum [24], giving some safety margin automatically.

Once the eigenvalue spectrum is known, one can use the Remez algorithm [25, 26] to compute a rational approximation in form of a partial fraction expansion,

$$\sqrt{[\boldsymbol{Q}^2]^{-1}} \approx \sum_i \frac{\alpha_i}{\boldsymbol{Q}^2 + \beta_i}\,.$$

The number of terms in the expansion can be chosen freely depending on the accuracy of the approximation needed. Chroma needs the GNU Multi Precision library [27] in order to compute the needed coefficients α_i and β_i.

The rational approximation can be used to improve another problem as well even for degenerate doublets using the "n-th root trick" by Clark and Kennedy [28]: When the quark masses are chosen smaller, their force contribution will increase. A single pseudo fermion field will introduce statistical noise which might trigger an instability in the integration. By rewriting $\det(\boldsymbol{A})$ as $\det(\boldsymbol{A}^{1/n})^n$ and using n independent pseudo fermion fields, the integration becomes more stable since individual statistical fluctuations are smaller. Similarly, one can choose to use independent pseudo fermion fields for the summands in the partial fraction expansion for a fermion field singlet.

The computation of the $(\boldsymbol{Q}^2+\beta_i)^{-1}\boldsymbol{\phi}$ are done by solving the equation $(\boldsymbol{Q}^2+\beta_i)x = \boldsymbol{\phi}$ for x. The only difference in the terms in the summands are the shifts β_i. With a *multi shift solver* [29] one can solve the equation once for the smallest shift and then quickly obtain the solution for the next larger shift. This makes approximations with over ten fractions still feasible to compute.

[3] Software package for lattice QCD simulations, will be introduced in Chapter 3.

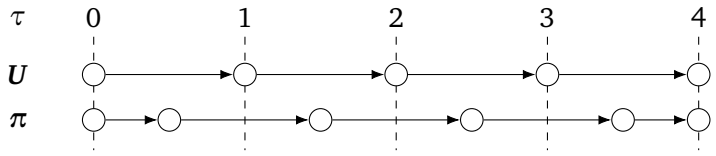


Figure 2.4.: Leap-frog integration. The coordinates $\boldsymbol{U}$ are evaluated on multiples of $\Delta\tau$ using Equation (2.9a), the momenta $\boldsymbol{\pi}$ are evaluated on the sites in between using Equation (2.9b). The very first and very last momentum step is done with a time step of $\Delta\tau/2$.

2.4.7. Molecular Dynamics

New configurations of gauge links can be obtained by evolving the four dimensional system as a whole in a fifth "time" dimension. Momenta $\boldsymbol{\pi}_{\boldsymbol{n}\mu}$ are introduced together with the guidance Hamiltonian

$$H_{\mathrm{MD}} = S + \frac{1}{2}\sum_{\boldsymbol{n}}\sum_{\mu}\mathrm{Re}\operatorname*{Tr}_{\mathrm{color}}(\boldsymbol{\pi}_{\boldsymbol{n}\mu}^2). \tag{2.8}$$

From Hamilton's equations one can obtain equations of motion along this unphysical fifth dimension. Using a numerical integration method one evolves one configuration of gauge links in the phase space along a hyperplane of constant H_{MD}. Integration schemes like Runge-Kutta do not conserve the energy and will lead to systematic errors. It is advantageous to use a symplectic integrator such that H_{MD} is conserved along the trajectory even though there are temporary fluctuations. This type of integrator also conserves phase space volume. The simplest such integrator is the *leap frog*. In that scheme, the time evolution is given by

$$\boldsymbol{U}_{\mu,\boldsymbol{n}}(\tau+\Delta\tau) = \exp\big(\mathrm{i}\Delta\tau\,\boldsymbol{\pi}_{\mu,\boldsymbol{n}}(\tau+\Delta\tau/2)\big)\boldsymbol{U}_{\mu,\boldsymbol{n}}(\tau), \tag{2.9a}$$

$$\boldsymbol{\pi}_{\mu,\boldsymbol{n}}(\tau+\Delta\tau) = \boldsymbol{\pi}_{\mu,\boldsymbol{n}}(\tau) + \Delta\tau\,\dot{\boldsymbol{\pi}}_{\mu,\boldsymbol{n}}(\tau). \tag{2.9b}$$

These equations keep $\boldsymbol{U}$ within $\mathrm{SU}(N_{\mathrm{c}})$ and are based on the ones given by Lippert [15, § 3.1]. The key element of the leap frog is the evaluation of the momenta in between the time slices. The gauge links are discretized with spacing $\Delta\tau$. In the first step, the momenta are only evolved by $\Delta\tau/2$ and then by $\Delta\tau$ afterwards. This is illustrated in Figure 2.4. The momentum derivative $\dot{\boldsymbol{\pi}}$ depends on the action used and includes force from fermions and gluons. Higher order symplectic integrators are available [30] and used in practice.

2.4.8. Hybrid Monte Carlo

A microcanonical molecular dynamics (MD) algorithm does not move through configuration space fast enough. It also only probes one hypersurface of constant energy within the Markov chain. One needs to have some way to leave the energy surface. The Langevin algorithm is a method to simulate statistical systems. It works by adding a random momentum from a Gaussian distribution and then solves the differential equations for a single time step. Momenta are refreshed and then point into a different direction in the phase space. The complexity scales as $O(V)$, but it is not exact [15, § 3.1]. A combination of the Langevin with the MD algorithm is a compromise that refreshes the momenta after a few MD integration steps. This algorithm is also not exact, needing an extrapolation to zero time stepping. Adding a global Metropolis step makes the algorithm exact. This is the *hybrid Monte Carlo* algorithm by Duane et al. [31]. The whole HMC algorithm goes as follows:

1. Create the pseudo fermion fields by sampling from a standard normal distribution and applying the kinetic operator once.
2. Sample the momenta randomly from $\mathfrak{su}(N_c)$.
3. Perform N_{MD} steps of molecular dynamics with time step $\Delta\tau$ such that $N_{MD}\,\Delta\tau \approx 1$ using the integration steps in Equation (2.9).
4. Accept the new configuration with probability $P = \min(1, e^{-\Delta H})$. ΔH is the change in the value of the guidance Hamiltonian before and after Step 3: $\Delta H = H_{after} - H_{before}$.

Besides the original paper [31] where the HMC has been applied to compact QED, there is a paper on the application to QCD [32] as well as an introduction for non-experts [15].

2.5. Optimization Tricks

The pseudo fermion fields are expensive to compute. If one tries to directly implement the above algorithms, one would need a much larger amount of computing time than with the tricks that will be introduced in this section. One has to use a large number of time steps in the molecular dynamics algorithm in order to get a good acceptance rate. Evaluating the fermion monomials is expensive, so reductions in the complexity will save vast amounts of computing time.

2.5.1. Even-Odd Preconditioning

Even-odd preconditioning is a technique by Degrand and Rossi [33] that allows to work with half of the lattice only. It also reduces the condition number of the equations that have to be solved. The mechanism will be introduced in the most general case for this work, the non-degenerate twisted mass (NDTM) doublet. The kinetic operator contains the following terms:

$$\boldsymbol{M} = -\frac{1}{2}\not{\boldsymbol{D}}_{\mathrm{W}} + 4 + m + \mathrm{i}\mu\boldsymbol{\gamma}_5\boldsymbol{\tau}^3 - \epsilon\boldsymbol{\tau}^1 + c_{\mathrm{SW}}\boldsymbol{T}\,,$$

where $\not{\boldsymbol{D}}_{\mathrm{W}}$ is the Wilson Dslash operator, $4+m=\alpha$ is the normal mass and the Wilson parameter ($4r=4$), μ is the twisted mass, ϵ is the mass splitting, and $\boldsymbol{T}$ the clover term. This quantity has intricate tensor structure: The Dslash connects neighboring lattice sites in space time; Dslash, $\boldsymbol{\gamma}_5$, and $\boldsymbol{T}$ act in spin space; Dslash and $\boldsymbol{T}$ act in color space; τ^3 and τ^1 act in flavor space.

The lattice can be divided into even and odd sites[4], such that all eight nearest neighbors of an even site are odd sites and vice versa. We can then split the kinetic operator M into two parts: A local part and a hopping part, which connects neighboring lattice sites. The local part only connects even with even sites (and odd with odd) and the hopping part connects even with odd sites (and vice versa). Writing this splitting explicitly in even-odd matrix form, it reads

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{M}_{\mathrm{ee}} & \boldsymbol{M}_{\mathrm{eo}} \\ \boldsymbol{M}_{\mathrm{oe}} & \boldsymbol{M}_{\mathrm{oo}} \end{pmatrix}_{\mathrm{EO}}.$$

We will indicate the space of the matrices in this section explicitly, otherwise it can become hard to distinguish even-odd from flavor or spin structure. In our case, we have $M_{\mathrm{eo}} = -\not{\boldsymbol{D}}_{\mathrm{eo}}/2$ and the remaining terms make up the local operator:

$$\boldsymbol{M}_{\mathrm{ee/oo}} = \alpha + \mathrm{i}\mu\boldsymbol{\gamma}_5\boldsymbol{\tau}^3 - \epsilon\boldsymbol{\tau}^1 + c_{\mathrm{SW}}\boldsymbol{T}_{\mathrm{ee/oo}}\,.$$

For the pseudo fermion fields, the kinetic operator $\boldsymbol{M}$ has to be inverted. One can use a LU decomposition [34, 35] to rewrite the matrix $\boldsymbol{M}$ as

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{M}_{\mathrm{ee}} & \boldsymbol{0} \\ \boldsymbol{M}_{\mathrm{oe}} & \boldsymbol{1} \end{pmatrix}_{\mathrm{EO}} \begin{pmatrix} \boldsymbol{1} & \boldsymbol{M}_{\mathrm{ee}}^{-1}\boldsymbol{M}_{\mathrm{eo}} \\ \boldsymbol{0} & \boldsymbol{M}_{\mathrm{oo}} - \boldsymbol{M}_{\mathrm{oe}}\boldsymbol{M}_{\mathrm{ee}}^{-1}\boldsymbol{M}_{\mathrm{eo}} \end{pmatrix}_{\mathrm{EO}}. \tag{2.10}$$

[4] The exact way of assigning even and odd is not important here, QPhiX uses $x+y+z+t \mod 2$ to assign the checkerboard. Then 0 is assigned as even and 1 as odd.

The lower right matrix element is usually defined as the odd-odd operator $\tilde{M}_{oo}$, namely $\tilde{M}_{oo} := M_{oo} - M_{oe}M_{ee}^{-1}M_{eo}$. The power of this decomposition is that the inverse of this product will be a product of similar looking matrices:

$$M^{-1} = \begin{pmatrix} \mathbf{1} & -M_{ee}^{-1}M_{eo}\tilde{M}_{oo}^{-1} \\ \mathbf{1} & \tilde{M}_{oo}^{-1} \end{pmatrix}_{EO} \begin{pmatrix} M_{ee}^{-1} & \mathbf{0} \\ -M_{oe}M_{ee}^{-1} & \mathbf{1} \end{pmatrix}_{EO} .$$

The parts that are local to the lattices sites are the two diagonal elements in even-odd space: The off-diagonal elements are just the Dslash operator. By virtue of the decomposition, we do not need to invert the whole operator at once, but rather only have to invert M_{ee} and then $\tilde{M}_{oo}$. Without the clover term ($c_{SW} = 0$), M_{ee} can be analytically inverted and is given by

$$M_{ee}^{-1} = \frac{\alpha - i\mu\gamma_5\tau^3 + \epsilon\tau^1}{\alpha^2 + \mu^2 - \epsilon^2} .$$

With the clover term ($c_{SW} \neq 0$), one has to use a numerical method for inversion. Since the clover term is diagonal in space-time (no derivatives included), inversion can be done independently for each lattice site. Inverting the 12×12 matrices (spin, color) is comparatively cheap and easily parallelizable.

The condition number of $\tilde{M}_{oo}$ is much reduced compared to the full operator. Also all matrix operations only need to act on half the volume which is an additional reduction in computational cost. The lattice extent is now restricted to an even number of sites in each direction. For finite temperature calculations, where the temperature is the inverse of the temporal lattice extent, this restriction can be severe. In these cases one needs to apply the full operator and cannot make use of this acceleration.

We conclude this section by writing out the flavor structure of the full preconditioned odd-odd operator. In general this is $\tilde{M}_{oo} = M_{oo} - M_{oe}M_{ee}^{-1}M_{eo}$. Plugging in the odd-odd and even-even term as well as the hopping matrix $M_{eo} = -\not{D}_{eo}/2$ gives

$$\tilde{M}_{oo} = \alpha + i\mu\gamma_5\tau^3 - \epsilon\tau^1 - \frac{1}{4}\not{D}_{oe}\frac{\alpha - i\mu\gamma_5\tau^3 + \epsilon\tau^1}{\alpha^2 + \mu^2 - \epsilon^2}\not{D}_{eo} .$$

For the flavor structure we will use the shorthand $d := \alpha^2 + \mu^2 - \epsilon^2$ for the

denominator and write

$$\tilde{M}_{\mathrm{oo}} = \begin{pmatrix} \alpha + \mathrm{i}\mu\gamma_5 - \frac{1}{4d}\not{D}_{\mathrm{oe}}(\alpha - \mathrm{i}\mu\gamma_5)\not{D}_{\mathrm{eo}} & -\epsilon - \frac{\epsilon}{4d}\not{D}_{\mathrm{oe}}\not{D}_{\mathrm{eo}} \\ -\epsilon - \frac{\epsilon}{4d}\not{D}_{\mathrm{oe}}\not{D}_{\mathrm{eo}} & \alpha - \mathrm{i}\mu\gamma_5 - \frac{1}{4d}\not{D}_{\mathrm{oe}}(\alpha + \mathrm{i}\mu\gamma_5)\not{D}_{\mathrm{eo}} \end{pmatrix}_{\mathrm{f}} . \tag{2.11}$$

2.5.2. Mass Preconditioning

The closer the quark masses come to their (small) physical value, the harder the equation $M^\dagger M x = \phi$, as needed in the fermionic action

$$S_{\mathrm{f}} = \phi_0^\dagger (M_0^\dagger M_0)^{-1} \phi_0 ,$$

becomes to solve with an algorithm like conjugate gradient [36] or BiCGStab [37] Additionally, the forces due to the fermions become larger. Therefore one needs to integrate with smaller time steps in order to keep the change in momentum $F \cdot \delta\tau$ approximately constant.

By introducing a second field of pseudo fermions one can distribute the force and condition number on both of them. This technique was proposed by Hasenbusch [38] and the added terms are often called "Hasenbusch terms". One introduces a second kinetic operator $M_1 := M_0 + \rho_1$ such that this new operator M_1 corresponds to a higher quark mass. This way the forces are smaller and the system is easier to solve. The precondition mass can also be chosen as a twisted mass, $M_1 = M_0 + \mathrm{i}\rho_1\gamma_5$ such that $M_1^\dagger M_1 = M_0^\dagger M_0 + \rho_1^2 = Q_0^2 + \rho_1^2$. The action is changed to the following:

$$S_{\mathrm{f}} = \phi_0^\dagger M_1^\dagger (M_0^\dagger M_0)^{-1} M_1 \phi_0 + \phi_1^\dagger (M_1^\dagger M_1)^{-1} \phi_1 .$$

The first term will be easier to solve because the right hand side is not just a pseudo fermion field but a preconditioned one with M_2. This reduces the forces such that one can choose the time stepping $\delta\tau$ a bit larger. The second term is a normal inversion but with a larger quark mass and therefore faster and more stable.

These terms are also called "determinant ratio" and "determinant" because one can also express this in the intermediate stage before the introduction of the pseudo fermions as

$$\det(Q_0^2) = \frac{\det(Q_0^2)}{\det(Q_0^2 + \rho_1^2)} \cdot \det(Q_0^2 + \rho_1^2) .$$

This technique can be combined with even-odd preconditioning. The determinant of the kinetic operator $\boldsymbol{M}$ can best be computed from the decomposition given in Equation (2.10). The even/odd preconditioning comes in by writing

$$\det(\boldsymbol{M}) = \det(\boldsymbol{M}_{\mathrm{ee}}) \det(\tilde{\boldsymbol{M}}_{\mathrm{oo}}) .$$

The determinant of $\tilde{\boldsymbol{Q}}$ has to be evaluated using the pseudo fermion approach. The determinant $\det(1 + \boldsymbol{T}_{\mathrm{ee}})$ does not need pseudo fermions, it can be evaluated directly via $\log\det(\boldsymbol{M}_{\mathrm{ee}})$ (also called "trace log") in the action. Each determinant ratio will feature one $\det(\boldsymbol{M}_{\mathrm{ee}})$ in the numerator and one in the denominator, these cancel cleanly and one only needs a single $\log\det(\boldsymbol{M}_{\mathrm{ee}})$ term in the action.

In the Chroma software [39] one uses a `CONSTDET`[5] monomial for the determinant ratios and the determinants because $\boldsymbol{M}_{\mathrm{ee}}$ does not depend on the pseudo fermions and will be treated separately. In case the Clover term is *not* employed, $\log\det(\boldsymbol{M}_{\mathrm{ee}})$ will just a constant number and does not affect the physics in any way. With the clover term, one has to add a single `LOGDET`[6] term for all mass preconditioning terms. This might be different in other lattice QCD implementations.

2.5.3. Multiple Time Scales

The action of an theory with $N_{\mathrm{f}} = 2 + 1$ flavors with clover term consists of various monomials, gauge, light determinant, clover log-det, the rational approximation. Each contributes to the total force on the gauge field. Their magnitudes are not all the same, though. The change is $\Delta\boldsymbol{U} = \pi\,\Delta\tau$ where π is the momentum. When momenta are smaller, they can be integrated with coarser time steps. Ideally the time steps are chosen such that each monomial contributes the same change to the gauge field.

This can be achieved by using different time scales for the monomials [40]. The monomials with the smallest force will be integrated on the outer time scale. The monomials contributing more are evaluated more often. The integration interval $\Delta\tau$ is split up into smaller intervals for these monomials. Figure 2.5 shows the evaluations of monomials on three time scales for one step on the outer time scale.

[5] `TWO_FLAVOR_EOPREC_CONSTDET_FERM_MONOMIAL` for the determinant and `TWO_FLAVOR_EOPREC_CONSTDET_RATIO_CONV_CONV_FERM_MONOMIAL` for the determinant ratio

[6] `N_FLAVOR_LOGDET_EVEN_EVEN_FERM_MONOMIAL`

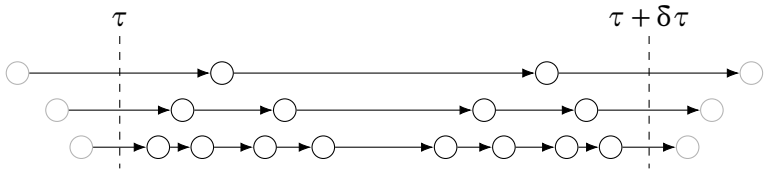


Figure 2.5.: Nesting of time scales with the second order Omelyan integrator. Each interval of size $\Delta\tau$ is split into n subintervals of size $\delta\tau$. Within each time interval, the integrator evaluates the force at $\tau + \lambda\,\delta\tau$ and $\tau+(1-\lambda)\,\delta\tau$. We have chosen the default value of $\lambda \approx 0.193$. A nested time scale will divide the time interval using the same prescription. The top row is the outermost time scale, then the first and second nesting.

Depicted is a second order integrator, so that the number of steps doubles with every additional time scale.

Fortunately the monomial with the largest force, the gauge monomial, is one of the cheapest to compute. Therefore one can gain performance by evaluating the more expensive monomials less often. The error in the energy, ΔH, will not change significantly, keeping the acceptance rate steady. Less evaluations of the expensive monomials mean less time per trajectory. Four time scales are easily reached for a production ensemble, but more time scales can also be useful.

3. Gauge Configuration Generation with Chroma

3.1. Software Overview

There are multiple software suites for lattice QCD available. Figure 3.1 provides an overview of the packages that are mentioned in this section. The suites differ in the discretizations implemented and also the targeted architectures. This work covers mostly the software from the USQCD collaboration. Their simulation package is called Chroma [39, 41] and requires a few modules:

- The QDP++ (QCD Data Parallel) library [42] provides data parallel algorithms that work with lattice data structures. It also provides a solver for the fermion matrix which is the performance critical part.

  QDP++ is implemented in C++ and uses expression templates. This way one can write high-level code with value semantics and still benefit from algorithmic optimizations below. The template expressions can be resolved to CPU or GPU code. The latter is done by the variant QDP-JIT (QDP Just In Time compilation) [43–45]. The expression templates with the JIT compiler will collect all instructions desired, upload all inputs to the GPU at once and only copy the end result back. This utilizes the limited bandwidth in the PCIe bus most efficiently.

- Communication over the network is done with the QMP (QCD Message Passing) library [46] which is a wrapper for MPI [47]. The 4D network topology and the location of the neighboring ranks is hidden in this library.

- The QMT (QCD Multi Threading) library [48] for threading has been replaced by OpenMP [49]. It provides similar facilities for parallelization with threads.

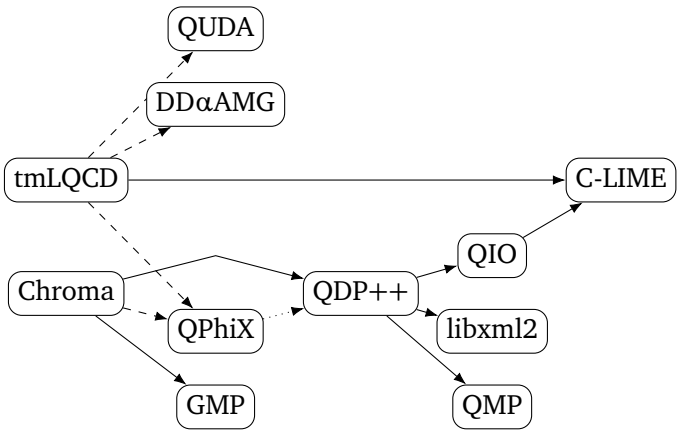


Figure 3.1.: Interplay of the various software packages. Solid arrows are denote dependencies. Dashed arrows denote optional dependencies. The dotted arrow indicates that QPhiX can be tested against QDP++ but does not use it for production.

- QPhiX[1] [50, 51] is a relatively new addition to the software suite and provides a faster implementation of the fermion matrix solver for the Intel Xeon Phi architecture and also for Intel Xeon. This library will be covered in more detail in Chapter 4.

- The LIME library [52] provides a file serialization format that is interleaves XML records for metadata and binary blobs for gauge configurations or propagators. This is wrapped in the QIO (QCD Input Output) library [53] which provides parallel IO to QDP.

The tmLQCD package [54, 55] provides similar facilities. It is written in C and only depends on the LIME [52] library and its MPI extension Lemon [56] for file serialization. The use of the same data format makes it easy to use gauge configurations generated with Chroma and tmLQCD with the other program. Compilation of tmLQCD is much easier than setting up Chroma due to this lack of a dependency hierarchy. It also features operators for different variants of twisted mass: Wilson, clover, non-degenerate. It can optionally interface with other solver libraries like DDαAMG which provides an adaptive multigrid solver or QUDA that provides solvers on NVIDIA accelerators. An interface to the QPhiX

[1] Usually pronounced “q-phi-ex”, sometimes “q-fix”.

solver has been developed by Bartosz Kostrzewa in the last months, my thesis contains contributions to this effort.

3.2. Simulation Setup

In the longer term, the σ resonance is to be analyzed. This is an application where twisted mass fermions suffer from unphysical excited state pollution, therefore plain Wilson fermions with a clover term are used. The states arise due to the following. Pions are the lightest hadrons that can be formed. The simplest scattering process that creates a σ from pions is $\pi_0\pi_0 \to \sigma$. This process is allowed by isospin coupling, $|1,0\rangle \otimes |1,0\rangle \ni |0,0\rangle$, and parity, $0^{-+} \cdot 0^{-+} \to 0^{++}$. With twisted mass, parity and isospin are explicitly broken such that the neutral pion is indistinguishable from the vacuum [5, p. 71]. This means that conventionally forbidden transitions would contribute in this framework. The pion–pion scattering data would be contaminated by states $\pi_0 X$ for all states X that would be allowed without twisted mass. This can be done recursively to add arbitrary amounts of neutral pions. These lower lying states need to be extracted and subtracted from the data before the scattering of interest becomes visible. The disadvantages of additional states outweigh the advantages of automatic $O(a)$ improvements. Therefore plain Wilson clover fermions are better suited for this particular task.

Both Chroma and tmLQCD can generate plain Wilson clover configurations. Since the group has lots of experience with tmLQCD, this is a good opportunity to test out Chroma and get some hands-on experience with it.

Finding a good set of parameters for a lattice QCD simulation is hard and requires a lot of tuning. Therefore we have started with values provided by the Budapest-Marseille-Wuppertal collaboration [57] and used their heaviest and coarsest parameters for a first test. Later on an ensemble with lighter quarks has also been used. The two sets of parameters are listed in Table 3.1.

Given their quoted lattice spacing of $a^{-1} = 1616(20)\,\mathrm{MeV}$, the pion masses are expected to be around 660 MeV and 270 MeV in physical units. These have to be measured once enough gauge configurations are generated. The measurement will be described in a following section. The heavier bare quark masses have the advantage that a smaller volume can be simulated. Temporal correlations functions of pions, the lightest hadronic state with mass M_π, behave like $\exp(-M_\pi t)$. The finite lattice extent in space L introduces finite size effects as terms of $\exp(-M_\pi L)$. The largest effect will come from the pion mass because that is the lightest hadronic

am_l	am_s	Lattice Size	aM_π	aM_K
−0.0960	−0.057	16×32^3	0.4115(6)	0.4749(6)
−0.1265	−0.057	32×96^3	0.169(1)	0.3500(7)

Table 3.1.: Selected parameters from the Budapest-Marseille-Wuppertal collaboration [57]. The coupling is $\beta = 3.3$ in both cases. They give an inverse lattice spacing of $a^{-1} = 1616(20)\,\mathrm{MeV}$.

state. One has to choose the lattice extent L such that $M_\pi L \geq 4$ in order to suppress these finite size effects by a good factor. This means that higher masses allow for smaller (and cheaper) lattices.

The gauge action used is the tree level improved action by Lüscher and Weisz [58, p. 347]. The coupling $\beta = 3.3$ has been used[2] with periodic boundary conditions. The quark fields are coupled to the gauge field after it had been stout-smeared [59] six times with strength[3] $\rho = 0.11$ in all four space-time dimensions. The clover term is included with a coefficient of $c_{SW} = 1.0$, the tree-level value. Due to the extensive smearing it is expected that this is a good approximation. Fermion fields have periodic boundary conditions in space and anti-periodic ones in time. Integration is performed using the minimal norm integrator which is called `LCM_STS_MIN_NORM_2` in Chroma's XML input file. The molecular dynamics trajectory length τ is always chosen as 1.0, the number of steps is varied to give an acceptance rate of roughly 70 % to 80 %.

Several stages of tuning were needed to thermalize the lattice. The simulation was started from a random (hot) configuration and subsequently cooled down. The action of the system has been decreasing in the beginning, therefore every update has been accepted. This allowed for coarse integration in the molecular dynamics part. Also running with a single time scale was possible. Once the energy had somewhat thermalized, the acceptance rate decreased. The addition of more integration steps in the molecular dynamics improved the acceptance rate but

[2] Chroma uses a different convention for the tree level improved action. A factor 5/3 from the Symanzik improvement program has been factored out. One has to supply $\beta_{\mathrm{Chroma}} = 5.5$ in the parameter file in order to get the correct result.

[3] When $\rho > 0.125$ is chosen, the system will behave in strange ways. The energy difference ΔH in the molecular dynamics will show large fluctuations and no thermalization. It might be that one hits an instability in the integrator and the systematic errors in the integrations pump energy into the system. On the outer time scale one needs thousands of integration steps in order to get some thermalization which will be incredibly slow.

Ensemble	Lattice Size	am_l	am_s
sWC_A2p1_Mpi660_L16T32	16×32^3	−0.0960	−0.057
sWC_A2p1_Mpi270_L24T96	24×96^3	−0.1265	−0.065
sWC_A2p1_Mpi270_L32T96	32×96^3	−0.1265	−0.065

Table 3.2.: The three ensembles of gauge configurations that are generated as part of this thesis.

made each configuration more expensive to compute. The addition of more mass preconditioning terms and distributing these on multiple time scales made the simulation more effective while maintaining a medium high acceptance rate.

The heavy ensemble has been named `sWC_A2p1_Mpi660_L16T32`. The "sWC" stand for the "stout Wilson clover". The letter "A" stands for the coarsest lattice spacing, finer lattice spacings will get letters "B" and so on. The "2p1" stand for $N_f = 2 + 1$, namely a degenerate doublet of light quarks and a single strange quark. The last two words stand for a pion mass of 660 MeV and the lattice size of $16^3 \times 32$.

After a sufficient number of gauge configurations have been generated, the pion and kaon masses have been measured and found to agree with the values quoted by the Budapest-Marseille-Wuppertal collaboration [57]. The measurement process will be covered in Section 3.7. Confident that the parameters have been set correctly and consistently, two lighter ensembles have been started on slightly larger lattices. Table 3.2 summarizes the lattices generated in this work. The two production ensembles had to be thermalized exactly like the test ensemble. After several weeks of tuning, four time scales, two mass preconditioning terms and a split rational approximation are used. The production setup is listed below, it shows the nesting of the time scales and the number of steps on each of them. For the $L = 32$ ensemble, the outer time scale uses 15 time steps instead of 12. Everything else is exactly the same.

- 12 Steps:
  - Light determinant ratio (masses −0.1265/−0.068 737 01)
  - Strange quark, smallest 4 shifts of the rational approximation
- 12-1 Steps:
  - Light determinant ratio (masses −0.068 737 01/0.508 892 9)
  - Strange quark, largest 10 shifts
- 12-1-1 Steps:
  - Light log-det

  - Strange log-det
  - Light determinant (mass 0.508 892 9)
- 12-1-1-1 Steps:
  - Gauge

For all monomials, the QPhiX conjugate gradient solver is used. For the molecular dynamics the (non-squared) residual of 10^{-6} is used whereas in the acceptance 10^{-9} is used for improved precision. A last solution predictor uses the previous solution as starting point for the next solve. This can introduce reversibility violations, hence also the stricter tolerance in the acceptance step. Edwards, Horváth, and Kennedy [60] state that a residual of 10^{-8} is sufficient to make the reversibility-violating effects negligible. Further investigations [61, 62] show that insufficient arithmetic precision leads to reversibility violations. Running a whole simulation of a larger lattice in single precision will introduce significant errors in observables [62]. Therefore we use double precision. In the future we might use a mixed precision solver to improve efficiency. Since the residual check is done in double precision, this will not have any effect. The restarts in the mixed precision solver will actually improve the accuracy of the solution.

3.3. Mass Parameter Tuning

The masses given by the FLAG collaboration [63, p. 30] for the pion and kaon with physical quarks but only QCD effects are $M_\pi^{\text{phys}} = 134.8(3)\,\text{MeV}$ and $M_\text{K}^{\text{phys}} = 494.2(3)\,\text{MeV}$. These are what we would expect in the limit of zero lattice spacing and physical quark masses.

The bare strange quark masses used by the Budapest-Marseille-Wuppertal collaboration [57] are such that the kaon does not have the physical value but rather 565.6(29) MeV, which is roughly 10 % too large. On the lattice with $\beta = 3.3$, the physical kaon mass in lattice units is $aM_\text{K}^{\text{phys}} = 0.306(4)$. A lower value for am_s has to be chosen such that the kaon mass obtains a value close to the physical value such that less quantities are unphysical in our simulation. Changing the bare strange quark mass will change the pion mass as well, though the effect will be small. Here only an estimation is done such that kaon mass is somewhat close to the physical value, we want to reduce the kaon mass around 10 % compared to the reference value.

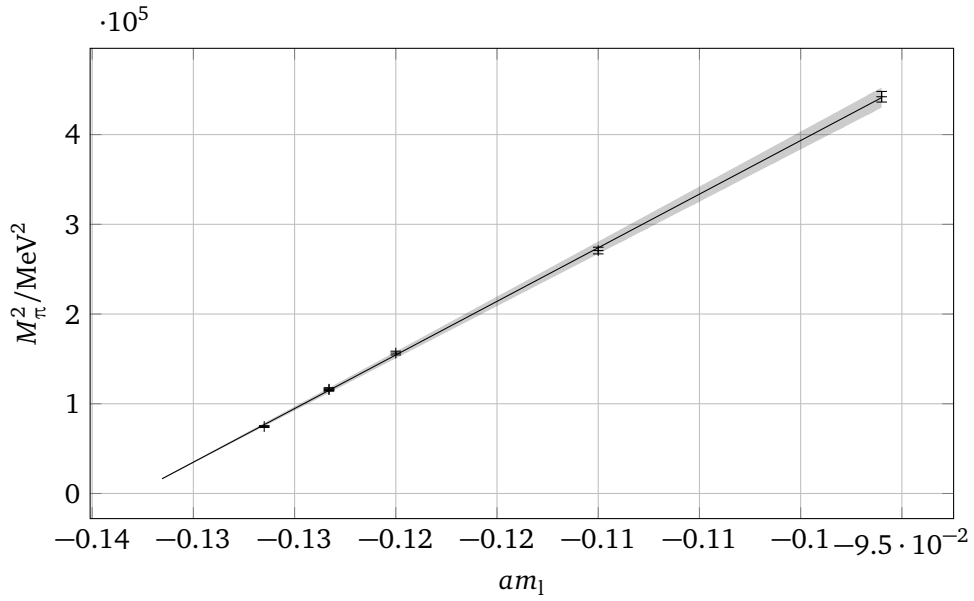


Figure 3.2.: Pion mass dependence on the light bare mass using the $\beta = 3.3$ data by the Budapest-Marseille-Wuppertal collaboration [57].

From the Gell-Mann-Oakes-Renner relation [64, p. 93] $M_\pi^2 = 2Bm_\mathrm{l}$, and the data given by the Budapest-Marseille-Wuppertal collaboration [57], the low energy constant B can be deduced. On the lattice with Wilson fermions there is the critical mass to consider, therefore the generalization [5, p. 17]

$$(aM_\pi)^2 = 2aB(am_\mathrm{l} - am_\mathrm{cr})$$

is needed. A fit using all bare light quark masses and measured pion masses by the Budapest-Marseille-Wuppertal collaboration [57] gives $aB = 2.288(9)$ and $am_\mathrm{cr} = -0.132\,92(6)$. The data, fit curve, and extrapolation to the physical pion mass are shown in Figure 3.2.

Chiral perturbation theory gives a similar relation for the mass of the kaon, namely $M_\mathrm{K}^2 = B(m_\mathrm{l} + m_\mathrm{s})$. The dominating summand will be the strange quark mass. The reduction of the kaon mass by roughly 10 % is achieved by reducing the (non-bare) strange quark mass to 90 % of its previous value:

$$(am_\mathrm{s})^\mathrm{new} = 0.9 \cdot [\underbrace{(am_\mathrm{s})^\mathrm{paper} - am_\mathrm{cr}}_{\text{quark mass}}] + am_\mathrm{cr}\,.$$

This results in a change in am_s from -0.057 to -0.065. The accuracy of this change can only be seen after generating a sufficient amount of gauge configurations and measuring the kaon mass. The calculation of the critical mass has only been possible since these lattices have already been generated by another group that has published the connection of bare parameters and meson masses. If one starts from scratch, one does not have this convenience.

Knowing the critical mass also enables finding of good parameters for the mass preconditioning. A rule of thumb says that the quark masses should be a factor 10 apart [65, p. 6][4]. Therefore the precondition mass with index i is computed as

$$(am_l)_i = 10^i \cdot [am_l - am_{cr}] + am_{cr}\,.$$

The sequence of masses starts with the target value and can be extended for any number of preconditioning terms needed; the first elements are -0.1265, -0.0687 and 0.5089.

3.4. Performance Tuning

Besides the physical parameters (bare masses am, coupling β) and the numerical parameters (preconditioning, time scales), there are options that have an impact on the performance. These include the choice of compiler, compilation flags and the choice of modules from the USQCD software repository. Then one has to find a suitable number of threads for each MPI rank and the number of ranks per node. For the strong scaling, one has to make a good choice of the 4D domain decomposition. The lattice is divided into equal blocks and distributed to the MPI ranks. One has to choose the mapping from these blocks to the MPI ranks to reflect the network topology.

The first test runs have been done on JUQUEEN [67]. The default compiler, IBM XLC, does not support C++11 and therefore could not be used. Via the module system a newer version of GCC is available, but that does not support the QPX intrinsics for that architecture. A modified version of Clang with intrinsics support is available and can compile Chroma. The performance is not what we have hoped, therefore an attempt was made to use the Bagel library [68]. It seems that Chroma only supports version 1 of Bagel whereas that version only supports Blue Gene/P

[4] Reference found through Reference [66].

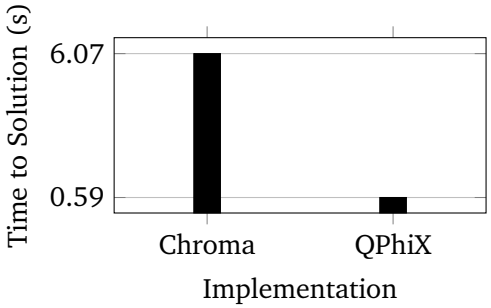


Figure 3.3.: Performance of the Chroma BiCGStab solver compared against the QPhiX implementation.

machines. JUQUEEN is a Blue Gene/Q machine that needs Bagel in version 3. The interfaces have changed, so this is a dead end.

The second machine that I was granted computing time on, JURECA [69], is an Intel Xeon Haswell machine with dual socket E5-2680 v3 @ 2.50 GHz, 12 cores each. The performance with the default solvers in Chroma was also below our hopes. The QPhiX library provides solvers optimized for Intel Xeon Phi architectures and also provides huge benefits on the Intel Xeon platform. Figure 3.3 illustrates the speedup of ten that has been observed with the BiCGStab solver on 81 nodes of JURECA, "Hyper-Threading" was not used. Both programs have been compiled with the Intel C++ compiler in version 2017. The needed number of iterations was 265 ± 3 on an almost thermalized $24^3 \times 96$ lattice. The data was generated for a previous talk [70]. Chroma together with QPhiX provides a good performance to generate gauge configurations.

Since JURECA is a dual socket machine, the performance is improved by using one MPI rank per socket. This allocates the memory for the two CPUs on different memory banks and therefore it is closer to the cores that use it. Communication between the sockets is explicitly over MPI and not implicitly via shared memory among threads. Using simultaneous multithreading (SMT, Intel calls it "Hyper-Threading") made the performance worse, so we just use 12 threads per MPI rank. Highly optimized code (like QPhiX) will not exhibit a lot of memory latency and is memory bandwidth bound, there is no benefit from more threads, it just creates more overhead for this application. Pinning of the threads to the cores gives an additional benefit. This can be tuned on a single node until one has the best single node performance.

The strong scaling onto multiple nodes depends on the domain decomposition. The 4D lattice has to be distributed onto the number of MPI ranks. Chroma uses the notion of four "geometry" factors which denote the number of ranks in the direction. The lattice size that each MPI rank is assigned then is the quotient of the global lattice size and the geometry factor. The Dslash operator is a nearest neighbor operator, therefore the boundaries need to be communicated between adjacent MPI ranks. Choosing the local lattice size to have the similar extents in every direction should give the least surface area possible. This should limit communication to the necessary minimum and allow for optimal scaling. QPhiX uses a data structure where each direction has a unique role. The x (and a bit of y) direction is used for SIMD vectors, y and z are divided into blocks for cache prefetching. The data layout of QPhiX is described in more detail in Chapter 4. This complicated layout adds more constraints to the domain decomposition and our experience is that choosing the geometry factors in increasing order gives the best performance.

On every machine one should run an exhaustive test of all domain decompositions to get the optimal configuration to run the simulation with. Table 3.3 shows all allowed geometries for a $64^3 \times 128$ lattice on two nodes and the most extreme on 128 nodes. One can see clearly the great variation between the decompositions. The number of jobs to run is large but each test only takes a few seconds. Overall core-hours invested are well spent given potential significant speedups. Aggregating the best geometry for a particular lattice and a given number of nodes, one obtains the strong scaling behavior as shown in Figure 3.4. The larger lattices scale much better because the ratio of computation to communication is much higher. Curiously the medium lattice seems to perform better than the larger lattice. The most likely explanation is that the data fits into the caches earlier and therefore more memory bandwidth is provided. For the large lattice, the speedup from 64 to 128 nodes looks like impossibly good scaling, which likely is the same cache effect.

In double precision, the good scaling stops much later because of the additional computational work. We have doubled the number of nodes as long as the speedup has been a factor 1.5 or more. On JURECA, the $L = 24$ ensemble ran on 64 nodes and needed around 8 minutes per configuration. The $L = 32$ ensemble ran on 128 nodes and consumed between 13 and 16 minutes per configuration. This results in around 200 and 750 core-hours per trajectory, respectively.

		Decomposition				
Nodes	Ranks	X	Y	Z	T	Performance / GF/s
2	2	2	1	1	1	204.3(11)
	4	4	1	1	1	215.3(1)
	4	2	2	1	1	260.2(2)
	4	2	1	2	1	270.4(2)
	2	1	2	1	1	273.2(9)
	4	2	1	1	2	275.3(2)
	2	1	1	2	1	337.0(37)
	4	1	4	1	1	351.9(2)
	4	1	2	2	1	367.7(3)
	2	1	1	1	2	368.1(48)
	4	1	2	1	2	377.3(3)
	4	1	1	4	1	379.9(3)
	4	1	1	2	2	393.4(4)
	4	1	1	1	4	410.2(3)
128	128	4	1	2	16	5367.2(837)
	256	4	8	2	4	10 811.9(2523)
	128	1	4	8	4	11 766.5(6436)
	256	1	2	8	16	19 172.0(542)

Table 3.3.: Spread in Dslash performance on JURECA with 2 nodes and 128 nodes. It can be seen that using two MPI ranks per node is advantageous. Also the correct domain decomposition is crucial for performance, there is a significant factor between the best and worst decomposition. On two nodes, all decompositions are shown; on 128 nodes only best and worst for each rank number are shown. This is a $64^3 \times 128$ lattice using single precision. Block sizes are 8, padding in xy is 1 and 0 for xyz. The SoA length is set to 4.

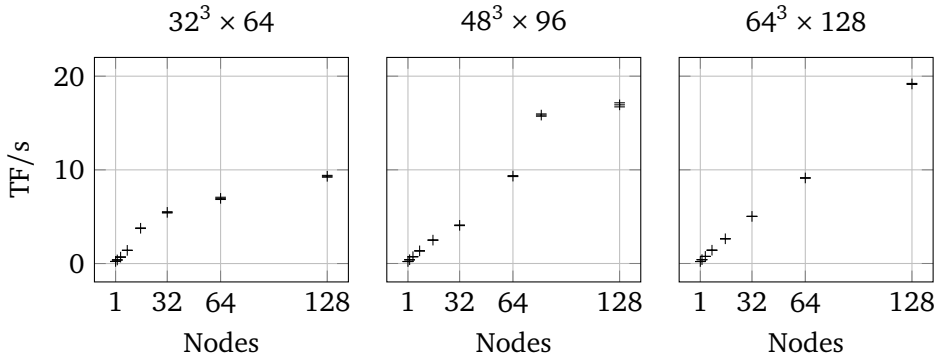


Figure 3.4.: Strong scaling on JURECA with three different lattice sites. Performance is shown in 10^{12} floating point operations per second and is proportional to the speed-up. Both axes are linearly scaled, perfect scaling would be a straight line. Over-perfect scaling likely is a cache effect.

3.5. Second Replica

The generation of configurations in one ensemble cannot be parallelized in the sense that different nodes generate different configurations. As seen in the previous section, the maximum number of nodes that can efficiently be used for a particular ensemble is also limited, therefore the actual time spend per configuration is rather fixed.

The best one can do is a factor of two by creating a second branch in the Markov chain. Once the lattice has deemed to be thermalized, one starts a second simulation with exactly the same physical parameters based on one particular gauge configuration, number 500 has been used in the $L = 24$ ensemble. The random seeds are altered and then the two *replicas* evolve into different directions of the phase space. Exploiting time reversal allows to join the replicas at the branching point to yield a longer Markov chain. Figure 3.5 illustrates this process and the naming of “forward” and “backward” ensemble. Once configuration generation is done, one can rename the configurations starting from zero in a continuous fashion.

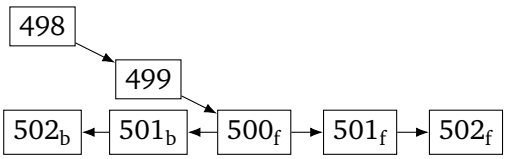


Figure 3.5.: Preliminary naming of the configurations of the forward and backward replica that are branched from configuration 500.

3.6. Autocorrelation Time

The Markov process inherently has autocorrelation in it because we generate configurations based on previous ones. It might take a couple of updates until there is no correlation noticeable any more. The number of correlations that feel the history is called *integrated autocorrelation time*. The autocorrelation A of an observable O is defined by

$$A(t-t_0) = \langle [O(t) - \langle O \rangle][O(t_0) - \langle O \rangle] \rangle \,.$$

This is normalized to give $\Gamma(t) = A(t)/A(0)$. From this the integrated autocorrelation time

$$\tau_{\text{int}} = \frac{1}{2}\Gamma(0) + \sum_{t=0}^{W} \Gamma(t)$$

can be computed. It would be appealing to choose the cutoff W very large in order to get the best estimate. Since Γ has statistical noise associated with it, one would end up summing a lot of random noise and effectively yielding no sensible value. Therefore the cutoff should be chosen such that $\Gamma(W)$ becomes zero in a sensible fashion. A robust method of computing the cutoff W has been developed by the ALPHA collaboration [71] and is also implemented in R-hadron [72] as `uwerrprimary`.

Figure 3.6 shows how the integrated autocorrelation time of the plaquette is computed for the $L = 24$ lattice. The result is $\tau_{\text{int}} = 7.68(193)$. Therefore in the later analysis of this ensemble, we use a spacing of at least 8 configurations in order to remove the autocorrelation among the configurations. Other observables might have a different autocorrelation times, one would have to perform a similar analysis for each observable to make sure that one has removed the autocorrelation with a sufficient spacing.

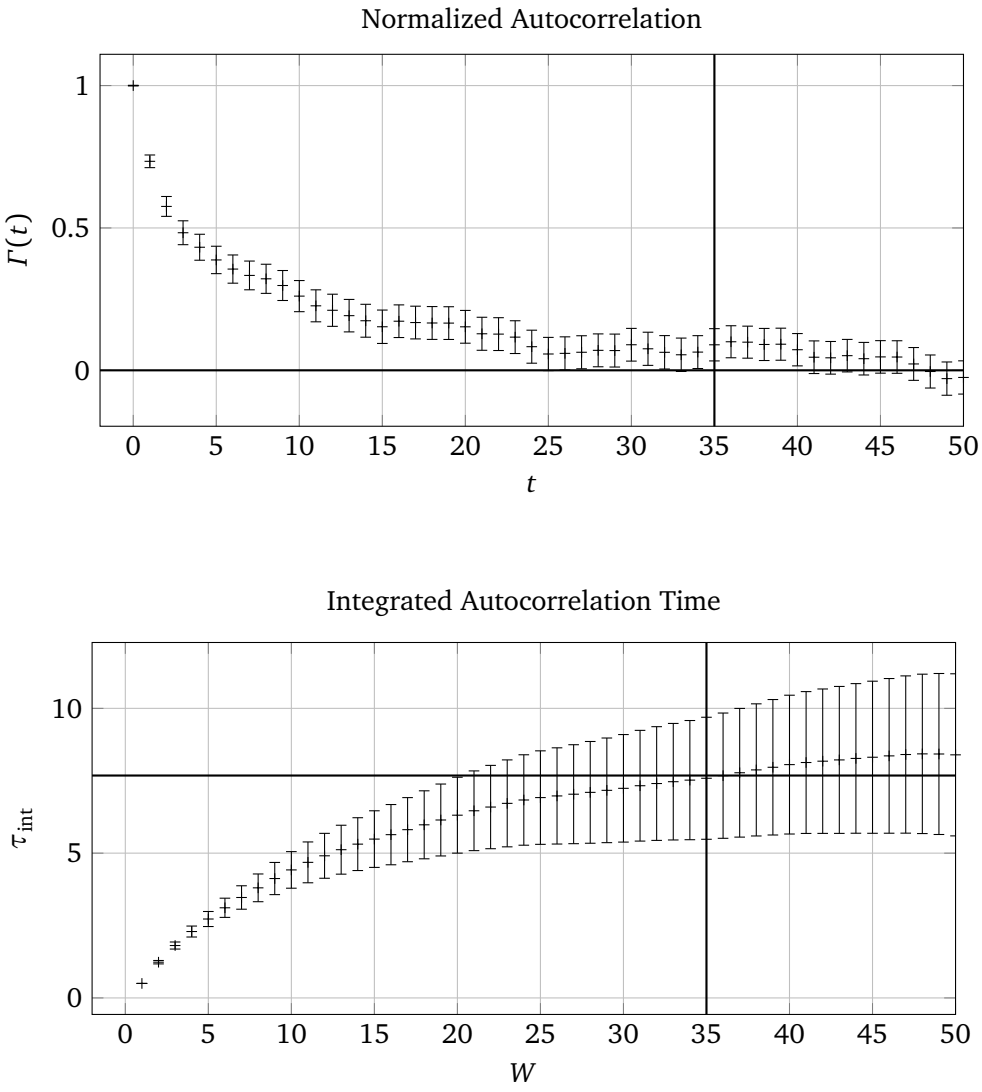


Figure 3.6.: Calculation of the integrated autocorrelation time using the algorithm by the ALPHA collaboration [71]. The lines show optimal cutoff W and zero in the upper plot, optimal cutoff and corresponding integrated autocorrelation time in the lower plot.

An alternative method is the *blocking* where one just takes every nth measurement into account or averages them beforehand. The value of n where the error on the result does not change any more also is an estimate for the integrated autocorrelation time.

3.7. Meson Mass Extraction

The mass of the pion and kaon can be computed quickly and compared to the values given by the Budapest-Marseille-Wuppertal collaboration [57]. For this first analysis, the configurations with numbers 400 to 649 of the `sWC_A2p1_Mpi660_L16T32` ensemble will be used. The last change in the simulation parameters has been at number 300. The choice allows for a sufficient number of configurations and a relaxation time of $\tau = 100$ in molecular dynamics units.

3.7.1. Correlators

Correlation functions of the respective meson operators O contain the effective mass. One can see this quickly by taking some energy eigenstates $|n\rangle$ and write

$$\begin{aligned} C(t) &= \langle 0|O(t)\,O^\dagger(0)|0\rangle = \langle 0|\mathrm{e}^{Ht} O \mathrm{e}^{-Ht} O^\dagger|0\rangle \\ &= \sum_n \langle 0|\mathrm{e}^{Ht} O \mathrm{e}^{-Ht}|n\rangle\,\langle n|O^\dagger|0\rangle \\ &= \sum_n \mathrm{e}^{-[E_n - E_0]t}\,|\langle 0|O|n\rangle|^2\,. \end{aligned}$$

In general, the operator O excites multiple energy eigenstates, just like plucking of a string excites multiple harmonic modes. For $t \to \infty$ only the lowest energy will dominate the correlation function. By fitting a falling exponential to the correlation, one can extract the lowest lying excited state.

For the pion, the operator is $\bar{\psi}_1\gamma_5\psi_1$. The correlation between different time slices is given by

$$C_\pi(t) = \langle [\bar{\psi}_1\gamma_5\psi_1](t)\,[\bar{\psi}_1\gamma_5\psi_1](0)^\dagger\rangle\,.$$

The kaon has one of the light quark fields replaced by a strange quark field, ψ_s, such that the correlator then is given by $\langle [\bar{\psi}_1\gamma_5\psi_s](t)\,[\bar{\psi}_1\gamma_5\psi_s](0)^\dagger\rangle$.

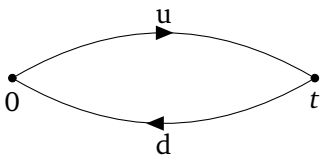


Figure 3.7.: Quark line diagram for a single pion. A up-quark and an anti-down-quark propagate from time 0 to time t.

All three pions have the same mass in pure QCD in the isospin limit. Therefore it is sufficient to look at the π^+ only; it is the easiest case because there are two distinct quark flavors and no self-loops can occur. The meson operator is $\bar{\mathrm{d}}\gamma_5\mathrm{u}$. The whole correlation function can be written as a single Wick contraction:

$$\begin{aligned} C_\pi(t) &= \langle \bar{\mathrm{d}}(t)\gamma_5\mathrm{u}(t)\,[\bar{\mathrm{d}}(0)\gamma_5\mathrm{u}](0)]^\dagger\rangle = \langle \bar{\mathrm{u}}(t)\gamma_5\mathrm{d}(t)\,\bar{\mathrm{d}}(0)\gamma_5\mathrm{u}(0)\rangle \\ &= -\langle \mathrm{u}(0)\bar{\mathrm{u}}(t)\gamma_5\,\mathrm{d}(t)\bar{\mathrm{d}}(0)\gamma_5\rangle = -\langle S_\mathrm{u}(0,t)\gamma_5\,S_\mathrm{d}(t,0)\gamma_5\rangle\,. \end{aligned}$$

When the up and down quark masses are degenerate, one can use the γ_5 hermiticity of the propagator and rewrite the correlator as

$$C_\pi(t) = -\langle S_\mathrm{u}(0,t)\,S_\mathrm{u}^\dagger(0,t)\rangle\,.$$

This allows to re-use one propagator for the other degenerate quark flavor. Figure 3.7 shows the corresponding quark line diagram for this contraction. Chroma can compute those propagators from the gauge configuration. One has to create a source, potentially smear it, build a propagator (invert), and then contract the propagators with the Γ structures of interest. Chroma calls the contractions "hadron spectrum" and will do all 16 of the Γ structures. Section A.1.2 contains a pointer to a complete input XML file that has been used to compute the pion and kaon propagator.

Every state that is probed by the pion or kaon operator will give a contribution of the form $\exp(-Mt) + \exp(-M[T-t])$, where T is the temporal extent of the lattice and M is the effective mass of that state. In the region $t \approx T/2$ the state with the smallest mass (ground state) will be the largest contribution, therefore it can be extracted by a fit to data points around the center of the correlator. Due to the periodic boundary conditions in time and the symmetries of $C(t)$, the correlator will by symmetric around the point $T/2$. It is common practice to fold the correlator around that point and average the (numerically equal) values. In a plot, it will look like a falling exponential curve with slight corrections from the

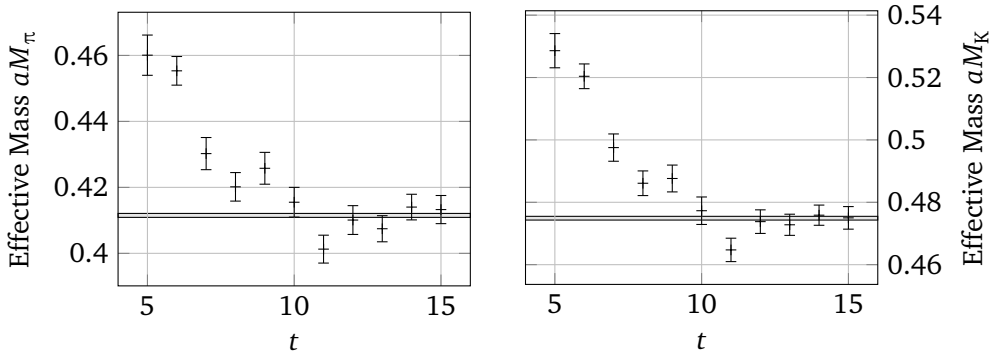


Figure 3.8.: Effective mass of the pseudoscalar mesons on the $L = 16$ lattice, computed with Equation (3.1). The values for $t < 5$ are rather large and obfuscate the plateau. The shaded band denotes the one standard error confidence interval around the reference results $aM_\pi^{[57]} = 0.4115(6)$ and $aM_K^{[57]} = 0.4749(6)$ by the Budapest-Marseille-Wuppertal collaboration [57].

cosh-like behavior.

3.7.2. Effective Mass

A suitable fit range can be found with the effective mass which is the negative logarithmic derivative of the correlator. In the region where the ground state becomes the dominant contribution, a plateau is expected. The discretized variant of this quantity,

$$aM_\pi(t) = \operatorname{arcosh}\left(\frac{C(t-1)+C(t+1)}{2C(t)}\right), \tag{3.1}$$

is shown in Figure 3.8 for the $L = 16$ lattice. The figure also shows the reference values $aM_\pi^{[57]} = 0.4115(6)$ and $aM_K^{[57]} = 0.4749(6)$ that were obtained by the Budapest-Marseille-Wuppertal collaboration [57].

An alternative definition of the effective mass is given by the exponential definition [73, p. 61] which is solved numerically for M_π:

$$\frac{C(t)}{C(t+1)} = \frac{\exp(-M_\pi t)+\exp(-M_\pi[T-t])}{\exp(-M_\pi[t+1])+\exp(-M_\pi[T-[t+1]])}.$$

Both definition have been applied to the $L = 24$ ensemble. Correlation functions have been computed using Chroma's point sources on configurations 501 to 2269 with a spacing of 8 on the forward replica. Figure 3.9 shows the effective mass with both definitions. One can see that the exponential solve gives a much better signal.

3.7.3. Correlated Fit

The correlator shows correlation in the time variable, therefore a simple least-squares fit would underestimate the error. We will quickly review the simple least-squares method to expand it for autocorrelation. Given are data points y_i with a corresponding explanatory variable x_i and a model $f(x, \lambda)$ with parameters λ_j. The likelihood function $f(x, y, \lambda) = \exp(-\chi^2)$ with

$$\chi^2 = \sum_i \frac{(y_i - f(x_i, \lambda))^2}{\sigma_i^2}$$

shall be maximized. Each fraction in the sum is the deviation of the data from the model (the residual) in terms of the uncertainty of that data point. Minimizing χ^2 is an equivalent procedure and usually used directly. This expression can be written in matrix form after introduction of the residual vector $r_i := y_i - f(x_i, \lambda)$ and the covariance matrix $\hat{C}_{ij} = \sigma_i^2 \delta_{ij}$. Then the quantity to minimize is given as $\chi^2 = r_i (\hat{C}^{-1})_{ij} r_j$, with summation over repeated indices implied. Uncorrelated fit models have a diagonal covariance matrix. Correlation is encoded in the non-diagonal elements. The covariance matrix of the correlators is computed as

$$\hat{C}_{ij} = \left\langle (C(i) - \bar{C})(C(j) - \bar{C}) \right\rangle .$$

The "bracket" average is to be taken over all the configurations (250 in this case) where the correlator has been evaluated. The "bar" average is to be taken over all time values t of the correlator on one configuration. If there was no correlation within the correlator, the quantity $\langle C(i)\, C(j) \rangle$ would only be nonzero if $i = j$ and the usual (diagonal) standard deviation is recovered.

The inverse covariance matrix $\hat{C}^{-1}$ is real and symmetric. Therefore a Cholesky decomposition can factor it into $L^{\mathrm{T}} L$ where L is a lower triangular matrix. The χ^2

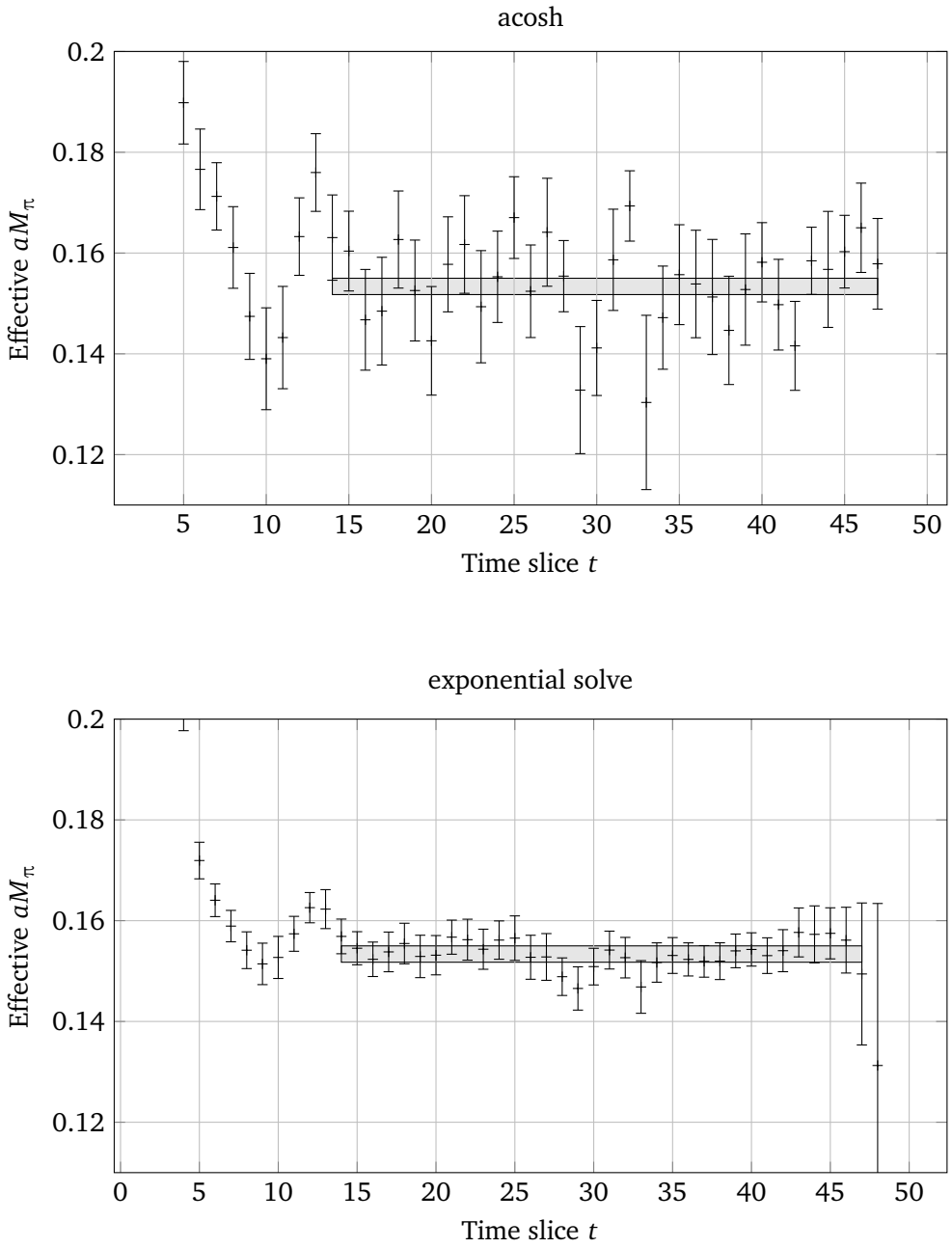


Figure 3.9.: Effective pion mass from point sources on the $L = 24$ ensemble using 221 configurations. The shaded box shows the one standard error band of an uncorrelated fit to the correlators using the R-hadron package.

can be rewritten by defining new residual vectors:

$$\chi^2 = \boldsymbol{r}^{\mathrm{T}}\hat{\boldsymbol{C}}^{-1}\boldsymbol{r} = \underbrace{\boldsymbol{r}^{\mathrm{T}}\boldsymbol{L}^{\mathrm{T}}}_{\boldsymbol{s}^{\mathrm{T}}}\underbrace{\boldsymbol{L}\boldsymbol{r}}_{\boldsymbol{s}} = |\boldsymbol{s}|^2 \,.$$

This redefined form can be used in regular least-squares routines as for instance shipped with SciPy [74] or R [75].

Performing a correlated fit starting at $t = 7$ as described with the aforementioned cosh-like model to the $L = 16$ ensemble, effective masses of $aM_\pi = 0.4229(52)$ and $aM_{\mathrm{K}} = 0.4832(52)$ are obtained. These compare nicely[5] to the corresponding reference results $aM_\pi^{[57]} = 0.4115(6)$ and $aM_{\mathrm{K}}^{[57]} = 0.4749(6)$ by the Budapest-Marseille-Wuppertal collaboration [57].

Using the `fit.effectivemass` function from R package "hadron" [72], Bartosz Kostrzewa has set up a script that basically does the same steps. The difference is that R-hadron is more comprehensive than my code and it also has the exponential solve definition of the effective mass. Using this script I have performed an uncorrelated fit to the $L = 24$ ensemble from time slice 14 to 47 because the data shows a plateau there. The covariance matrix could not be estimated properly using the limited amount of noisy point source correlation functions available, therefore an uncorrelated fit was performed.

It can already be seen by eye from the plots shown in Figure 3.9 that our mass of 0.1533(16) is significantly lower than the 0.169(1) quoted by the Budapest-Marseille-Wuppertal collaboration [57]. This indicates that the altered bare strange quark mass has a significant effect on the pion mass. Our error is likely underestimated, still the difference is very large in terms of our error.

[5] The errors on the masses in this analysis as well as the ones in the reference are assumed to be Gaussian. In order to check the null hypotheses, that computed masses come from the same underlying distribution, a two-tailed Z-test can be performed. The test statistic is computed as

$$Z = \frac{X - Y}{\sqrt{s_X^2 + s_Y^2}} \,,$$

where s_X is the standard error of the variable X. The scores are $z_\pi = 0.0278$ and $z_{\mathrm{K}} = 0.0174$. One can say that masses for the pion agree to $0.028\,\sigma$, where σ is the combined standard error from the above denominator. A probability for the equality of the underlying distribution can also be computed: The masses obtained here could be equally likely larger or smaller than the reference, therefore this is a two-tailed case. The p value therefore is given by $p = \mathrm{erf}(|z|) + 1 - \mathrm{erf}(-|z|)$ where erf is the cumulative density function of the normal distribution, the error function. Therefore the probability that the simulation in this work reproduces the reference mass is $p_\pi = 0.978$ and $p_{\mathrm{K}} = 0.986$, respectively.

The error estimation is done with the bootstrap method: From set $\mathbf{S}$ of the N configurations, take N configurations with replacement, this yields the set $\mathbf{S}_1^*$. The exact same analysis can be done with the N configurations of the set $\mathbf{S}_1^*$, yielding $(aM_\pi)_1^*$ and a kaon mass. This is repeated until $\mathbf{S}_{3N}^*$ is obtained. The mean of all effective masses $(aM_\pi)_i^*$ lies close to the value aM_π which is obtained with the actual data. The standard deviation of the $(aM_\pi)_i^*$ gives an estimate for the statistical error in the quantity aM_π.

In the analysis here, the fit ranges are chosen by eye such that the effective mass agrees with a constant within the statistical errors. A more thorough approach would not fix the starting point of the fit but rather perform all possible fits. Each fit would have an associated χ^2 value characterizing the quality of the fit. Via the cummulative density function of the χ^2 distribution, a so called p-value can be computed[6]. It gives the probability that another data sample would give a χ^2 that is greater or equal than the one in question, *if* the data is indeed explained by the model. A high p-value implies that the data can be explained by a single mass state, therefore that the fit range is sufficiently narrow around $T/2$. The effective mass extracted from each fit can be added to a p-weighted distribution. The quantiles of that distribution then give the best value and an estimation of the systematic error due to the fit range selection. This preliminary mass extraction is just a check whether the simulation yields the expected results. Later on more advanced analysis techniques will be used.

3.8. Scale Setting

Although one would expect to know the lattice spacing in physical units when running a simulation, the lattice spacing has to be measured afterwards. Renormalization effects change the lattice spacing. One has to measure a physical quantity in lattice units and relate that to the experimentally known value in physical units. One possibility is to measure aM_Ω, a dimensionless quantity. Using the experimentally known physical value of M_Ω in MeV, one can extract a in MeV or fm.

The complication with this approach is that it is only valid for physical masses and in the continuum limit, $a \to 0$. Tuning the bare mass parameters such that the mesons obtain physical values will increase the condition numbers in the inversions and make it much more compute intensive, at least with algorithms like

[6] $p = 1 - F_{\chi^2}(\chi^2)$

conjugate gradient. Reducing the lattice spacing will have to go with an increase in lattice points, although $a = 0$ of course can never be reached in a simulation. One therefore performs the simulation at a few different masses and a few different lattice spacings. Extrapolating both to the physical point then allows to compare the lattice measurements with experimental measurements.

Setting the scale for a single simulation without the extrapolation will have significant systematic errors. Using an observable which has a lesser dependence on the lattice spacing and quark masses is an improvement. The Wilson flow has been suggested as a scale by Lüscher [76]. From the gauge fields $\boldsymbol{A}_\mu$ a field $\boldsymbol{B}_\mu \in \mathrm{SU}(N_\mathrm{c})$ is derived by solving the defining equations of the flow are

$$\dot{\boldsymbol{B}}_\mu = \mathrm{D}_\nu \boldsymbol{G}_{\nu\mu}\,, \qquad \boldsymbol{B}_\mu(0) = \boldsymbol{A}_\mu\,,$$

$$\boldsymbol{G}_{\mu\nu} = \partial_\mu \boldsymbol{B}_\nu - \partial_\nu \boldsymbol{B}_\mu + [\boldsymbol{B}_\mu, \boldsymbol{B}_\nu]\,, \qquad \mathrm{D}_\mu = \partial_\mu + [\boldsymbol{B}_\mu, \cdot\,]\,.$$

They are used to integrate the gauge field into a new flow direction which is independent of physical or molecular dynamics time. Chroma has this measurement implemented, an example input file can be found through Section A.1.3. The implementation can leverage the stout smearing because the derivative of the lattice is exactly the exponent in the smearing. Infinitesimal stout smearing steps are combined with a Runge-Kutta integrator to yield a high-order integration of the differential equation.

On each configuration the gauge action $E(t) \propto \langle \boldsymbol{G}_{\mu\nu} \boldsymbol{G}_{\mu\nu} \rangle$ is computed using the clover definition of the gluon field strength tensor. The resulting quantity $\langle E(t) \rangle$ is shown in Figure 3.10 for a particular configuration.

The quantity $t^2 \langle E(t) \rangle$ will grow linearly with the flow time, as can be seen in Figure 3.11a. If the configuration is not yet thermalized, it might have a maximum and vanish for very large flow times. In that case it does not make much sense to set the scale, therefore this is not a problem. The proposed scale setting [76] works as follows:

1. Integrate the flow equations and compute $\langle E(t) \rangle$.
2. Compute $t^2 \langle E(t) \rangle$.
3. Find the time t_0/a^2 where $t_0^2 \langle E(t_0) \rangle = 0.3$.

Another improvement has been suggested by the Budapest-Marseille-Wuppertal collaboration [77]. Here the scale setting quantity is not $t^2 \langle E(t) \rangle$ but $t^2 \frac{\mathrm{d}}{\mathrm{d}t} \langle E(t) \rangle$. It shows a similar linear increase with t as shown in Figure 3.11b. They quote the continuum value as $w_0 = 0.1755(18)(04)\,\mathrm{fm}$ which they obtained by computing

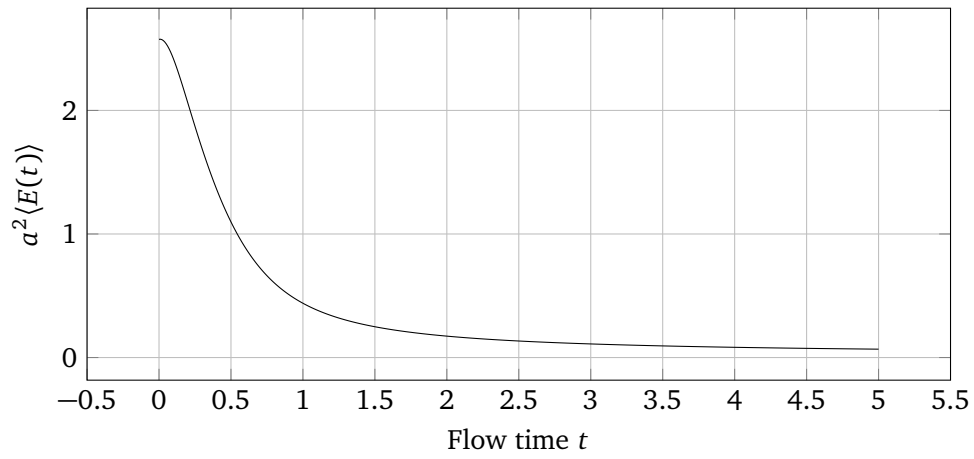
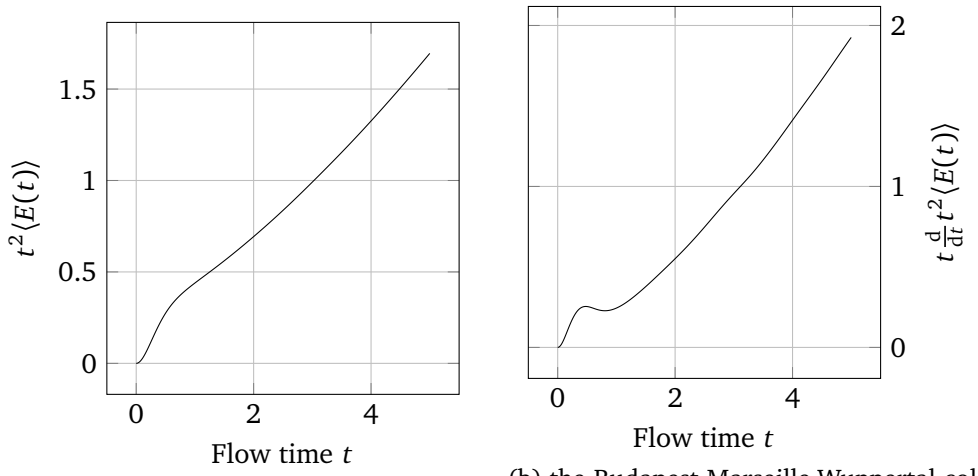


Figure 3.10.: Expectation value of the gauge action density versus flow time. Computed on configuration 622 of the small and heavy ensemble.



(a) Lüscher [76].

(b) the Budapest-Marseille-Wuppertal collaboration [77].

Figure 3.11.: Derived quantities for scale setting.

the quantity w_0/M_Ω on ensembles with different lattice spacings and quark masses and extrapolated to the continuum limit with physical quark masses. Therefore it is not directly comparable with our simulations at finite lattice spacing and unphysical quark masses. However, this quantity shows only mild dependence on the lattice spacing and the quark masses. It is expected that one can compute a/w_0 from the available configurations and then get an estimation of a that is good within perhaps 15 %.

The scale w_0/a has been computed for some of the early configurations in the small and medium ensemble. Using the continuum value of w_0 one can compute the value a^{-1} for each configuration, shown in Figure 3.12. The resulting values of very roughly 1470 MeV and 1250 MeV are much smaller than the quoted value of 1616(20) MeV [57] for our ensemble. The deviation is likely a systematic effect because we did not extrapolate our w_0/a to the continuum. The chiral perturbation equation given in their paper needs more different mass parameters in order to get a good estimate of the fit parameters. For our analysis we only use the published value because its accuracy will be superior to the calculated values.

3.9. Perambulators and Mass Extraction

The preliminary analysis of the pion and kaon mass in Section 3.7 was done using point sources and one inversion per color and spin component. There was no smearing used besides the six iterations of stout smearing that are part of our fermion–gauge interaction. The gauge configurations contain various energy states which will contaminate the signal in the pion mass measurement. For the lightest mesons, we would like to suppress the high frequency modes in order to get a better signal of the ground state.

3.9.1. Laplace Heaviside Smearing

The Laplacian Heaviside (LapH) quark smearing scheme [78] in the stochastic variant [79] has been used in the more advanced analysis of the data. This section will follow Morningstar et al. [79]. In summary, the method reduces the gauge configuration to its lowest eigenmodes in terms of the Laplace operator. First one smears the spatial gauge links. The authors use stout smearing, we have used

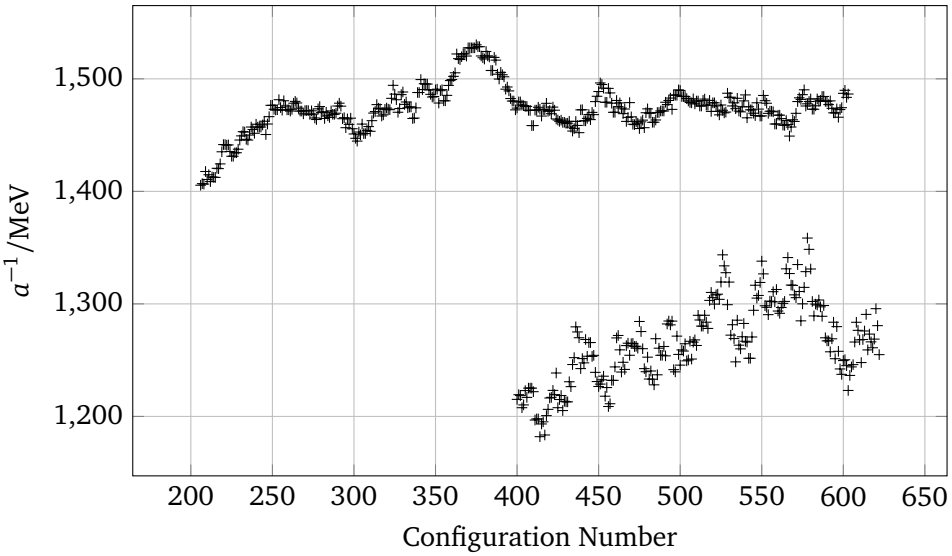


Figure 3.12.: Computed scale using the w_0 scale for two ensembles. The top curve corresponds to the `sWC_A2p1_Mpi270_L24T96` ensemble, configurations before 500 are not considered thermalized, large changes and trends are expected. The bottom curve is from the `sWC_A2p1_Mpi660_L16T32` ensemble and shows longer autocorrelation time, larger fluctuations and a significantly lower value of a^{-1}. No error bars are shown because the w_0/a measurement on each configuration has no error estimate and the error on w_0 itself is quoted at the percent level.

HYP smearing [80] instead.[7] Then the eigenvectors and eigenvalues (all real and negative) of the gauge covariant discreticed Laplacian operator are computed on these gauge configurations. Let $\boldsymbol{V}_\triangle$ be the matrix of eigenvectors and $\boldsymbol{\Lambda}_\triangle$ the matrix of eigenvalues. Then the Laplacian using the smeared fields can be written as $\tilde{\triangle} = \boldsymbol{V}_\triangle \boldsymbol{\Lambda}_\triangle \boldsymbol{V}_\triangle^\dagger$. We are only interested in the lowest eigenmodes, therefore we only keep the most negative eigenvalues by constructing a smearing operator $\mathscr{S}$ (script "S") such that

$$\mathscr{S} = \boldsymbol{V}_\triangle \, \Theta(\sigma_\mathrm{s}^2 + \boldsymbol{\Lambda}_\triangle) \, \boldsymbol{V}_\triangle^\dagger \, .$$

The Heaviside step function Θ will yield a diagonal matrix with ones where eigenvalues have smaller magnitude than the threshold σ_s^2 and zeros otherwise. We can discard the unused entries from $\boldsymbol{V}_\triangle$ and name this non-square matrix $\boldsymbol{V}_\mathrm{s}$. It maps from the LapH space to the physical space:

$$\boldsymbol{V}_\mathrm{s} : \text{Eigenvectors} \otimes \text{Time} \rightarrow \text{Space} \otimes \text{Time} \otimes \text{Color} \, .$$

The Laplacian does not act along the temporal direction and therefore is diagonal in time. There are independent eigenvectors for each time slice. Therefore one could also state the above as $\boldsymbol{V}_\mathrm{s}(t)$: Eigenvectors $\rightarrow$ Space $\otimes$ Color. The original authors have observed that the number of vectors that are selected by the Heaviside function varies by one or two when the number of eigenvectors N_v is greater than 60. Therefore one can fix the number of eigenvectors for all time slices without losing or gaining many eigenvectors. This allows to store $\boldsymbol{V}_\mathrm{s}$ as a rectangular array of $N_\mathrm{v} L_\mathrm{t}$ columns and $L_\mathrm{s}^3 L_\mathrm{t} N_\mathrm{c}$ rows. The smearing operator can be written as $\mathscr{S} = \boldsymbol{V}_\mathrm{s} \boldsymbol{V}_\mathrm{s}^\dagger$. The authors state an optimal value of $\sigma_\mathrm{s}^2 \approx 0.33$ which is supposed to be insensitive to the choice of hadron operator and parameters like quark mass and lattice spacing.

Instead of using the actual propagator $\boldsymbol{M}^{-1}$, one uses the "quark line" $\mathscr{Q}$ (script "Q") which is smeared:

$$\mathscr{Q} = \mathscr{S} \boldsymbol{\Omega}^{-1} \mathscr{S} = \boldsymbol{V}_\mathrm{s} (\boldsymbol{V}_\mathrm{s}^\dagger \boldsymbol{\Omega}^{-1} \boldsymbol{V}_\mathrm{s}) \boldsymbol{V}_\mathrm{s}^\dagger \, .$$

$\boldsymbol{\Omega}$ is $\boldsymbol{\gamma}_4 \boldsymbol{M}$ which is stated as useful to ensure hermiticity in baryon correlation matrices. Instead of storing the full tensor $\boldsymbol{\Omega}^{-1}$, one only needs to store the expression in parentheses. For the $24^3 \times 96$ lattice, the full one-to-all propagator would need

[7] Apparently the exact nature of the smearing is not important, it just needs to sufficiently remove the high-frequency modes. In the development of the software used (peram_gen), a cross-checked implementation of HYP smearing was already available, so this was used.

13 824 spatial volume elements. This is reduced to $N_\mathrm{v} = 120$ eigenvectors, this is a saving factor of around 115. Still the Dirac structure is preserved such that all sorts of correlation functions can be computed with a given set of quark lines.

3.9.2. Stoachstic LapH (sLapH)

The LapH method introduced so far allows storing less data and making less inversions. However, the number of needed inversions is still quite high. If one would do all these inversions, the limiting factor would be the gauge noise introduced by the way that the configurations are generated. Therefore Morningstar et al. [79] claim that it is sufficient to compute stochastic inverses. The following will be a summary of their stochastical Laplace-Heaviside method, "sLapH".

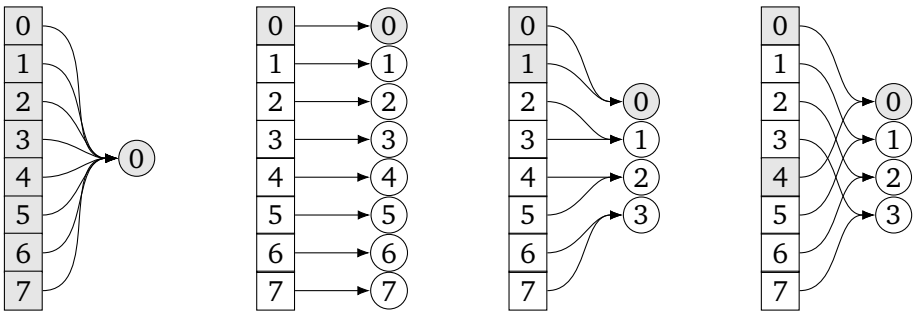
The inverse of the large matrix $\boldsymbol{\Omega}$ can be stochastically inverted by taking standard normal random noise vectors $\boldsymbol{\eta}$ such that their expectation value vanishes, $E(\eta_i) = 0$, and have no correlation, $E(\eta_i \eta_j) = \delta_{ij}$. For each such noise vector $\boldsymbol{\eta}^r$ one solves the equation $\boldsymbol{\Omega} \boldsymbol{X}^r = \boldsymbol{\eta}^r$. An estimate of $\boldsymbol{\Omega}^{-1}$ is then given by

$$\Omega^{-1}_{ij} \approx \frac{1}{N_\mathrm{R}} \sum_{r=1}^{N_\mathrm{R}} X^r_i \eta^{r*}_j , \tag{3.2}$$

where N_R is the number of random noise vectors used. The estimate is not a good one because the variances are very large.

Dilution of the noise vectors will reduce the variance in the propagator $\boldsymbol{\Omega}^{-1}$. The dilution process can be expressed in terms of a complete set of orthogonal projection operators $\boldsymbol{P}^{(b)}$. These are diagonal matrices that only contain ones and zeros, effectively selecting certain components from vectors into the diluted element b.

Figure 3.13 illustrates the four dilution schemes. Projecting the N elements of a space into a single element is called "no dilution" (Figure 3.13a) because the resulting vectors still contain all their components. Projecting each vector component onto a separate element is named "full dilution" (Figure 3.13b) as each component is separated. Projecting adjacent components onto one element such that J elements remain is called "block-J" dilution (Figure 3.13c). A mapping based on the modulo is called "interlace-J" dilution (Figure 3.13d). Unfortunately people either mean that the result has J blocks *or* that J source blocks are averaged into a



(a) no dilution (b) full dilution (c) block-4 (d) interlace-4

Figure 3.13.: The dilution schemes as introduced by Morningstar et al. [79]. Mapping from original space (squares) to the dilution blocks (circles). The original space has eight elements in this illustration.

single result block when they say "block-J". Therefore it is best to be unambigious and call it "block dilution into J blocks" or "block dilution with block size J".

Inserting two full sets of such projection operators $\sum_b \boldsymbol{P}^{(b)}$ into the estimate of Ω_{ij}^{-1} from Equation (3.2) gives

$$\Omega_{ij}^{-1} \approx \frac{1}{N_\mathrm{R}} \sum_{r=1}^{N_\mathrm{R}} \sum_{b,b'} (P_{ik}^{(b)} X_k^r)(P_{jl}^{(b')} \eta_l^{r*}) \overset{1}{=} \frac{1}{N_\mathrm{R}} \sum_{r=1}^{N_\mathrm{R}} \sum_{b,b'} (P_{ii}^{(b)} X_i^r)(P_{jj}^{(b')} \eta_j^{r*})$$
$$\overset{2}{=}: \frac{1}{N_\mathrm{R}} \sum_{r=1}^{N_\mathrm{R}} \sum_{b,b'} X_i^{r[b]} \eta_j^{r[b']*} \overset{3}{\approx} \frac{1}{N_\mathrm{R}} \sum_{r=1}^{N_\mathrm{R}} \sum_{b} X_i^{r[b]} \eta_j^{r[b]*} .$$

In the additional steps, the following has been done:

1. The projection operators are always diagonal matrices containing only 1 and 0 entries. Therefore one can directly remove one index in each parentheses to make it easier to reason about the expression.

2. In the original paper the authors introduce a shorthand notation for the projected vectors. This is also introduced here in this step. Since the projectors are diagonal, the only difference between the original vector and the diluted one is that certain elements are replaced with zeros.

3. This last step is the most important one and gives rise to the power of dilution. Only terms that correspond do the same projector b is kept, mixed terms

are discarded. In the case of no dilution this does not change anything because $b = b' = 1$ was the only available dilution element. For full dilution, the projectors are δ_{ib} and $\delta_{jb'}$. By setting $b = b'$, we effectively insert a δ_{ij} which lets all off-diagonal components of $\boldsymbol{\Omega}^{-1}$ vanish exactly. The block and interlace dilution schemes are a mixture of both, preserving a few off-diagonal elements of the propagator $\boldsymbol{\Omega}^{-1}$.

The stochastical estimation of the propagator is best done with diluted random noise vectors in the LapH space. This lets one rewrite the quark line introduced above in terms of a standard normal noise vector in LapH space $\boldsymbol{\rho}$ with $E(\boldsymbol{\rho}) = \mathbf{0}$ and $E(\boldsymbol{\rho}\boldsymbol{\rho}^\dagger) = \mathbf{1}$:

$$\begin{aligned}\mathscr{Q} &= \mathscr{S}\boldsymbol{\Omega}^{-1}\mathscr{S} \overset{1}{=} \mathscr{S}\boldsymbol{\Omega}^{-1}\boldsymbol{V}_\mathrm{s}\boldsymbol{V}_\mathrm{s}^\dagger \overset{2}{=} \sum_b \mathscr{S}\boldsymbol{\Omega}^{-1}\boldsymbol{V}_\mathrm{s}\boldsymbol{P}^{(b)}\boldsymbol{P}^{(b)\dagger}\boldsymbol{V}_\mathrm{s}^\dagger \\ &\overset{3}{=} \sum_b \mathscr{S}\boldsymbol{\Omega}^{-1}\boldsymbol{V}_\mathrm{s}\boldsymbol{P}^{(b)}E(\boldsymbol{\rho}\boldsymbol{\rho}^\dagger)\boldsymbol{P}^{(b)\dagger}\boldsymbol{V}_\mathrm{s}^\dagger \\ &\overset{4}{=} \sum_b E\big(\underbrace{\mathscr{S}\boldsymbol{\Omega}^{-1}\boldsymbol{V}_\mathrm{s}\boldsymbol{P}^{(b)}\boldsymbol{\rho}}_{\boldsymbol{\varphi}^{[b]}(\boldsymbol{\rho})}\underbrace{(\boldsymbol{V}_\mathrm{s}\boldsymbol{P}^{(b)}\boldsymbol{\rho})^\dagger}_{\boldsymbol{\varrho}^{[b]}(\boldsymbol{\rho})}\big).\end{aligned}$$

The various steps do the following:

1. The second dilution operator $\mathscr{S}$ is expanded in terms of its definition.
2. A unity in form of a full set of projection operators is inserted.
3. Another unity in form of the expectation value is inserted.
4. The terms are regrouped in a suggestive way, the expectation value expanded around the whole expression.

The quantities $\boldsymbol{\varrho}^{[b]}(\boldsymbol{\rho})$ and $\boldsymbol{\varphi}^{[b]}(\boldsymbol{\rho})$ are called displaced-smeared-diluated quark source and sink vectors. In the original papers the authors also include a displacement which is not used in this work here.

In order to store the minimal amount of data, one stores the *eigensystem* $\boldsymbol{V}_\mathrm{s}$, the *random vectors* $\boldsymbol{\rho}^r$ and the *perambulators* $\boldsymbol{V}_\mathrm{s}^\dagger\boldsymbol{\Omega}^{-1}\boldsymbol{V}_\mathrm{s}\boldsymbol{P}^{(b)}\boldsymbol{\rho}$. One does not store the smeared-diluted quark sink $\boldsymbol{\varphi}^{[b]}(\boldsymbol{\rho})$ because it is a vector in the physical space (and not LapH space) and therefore much larger.

Quark lines can then be build up from these parts and are given by

$$\mathscr{Q}_{uv} \approx \frac{1}{N_\mathrm{R}}\sum_{r=1}^{N_\mathrm{R}} \boldsymbol{\varrho}_u^{[b]}(\boldsymbol{\rho})\boldsymbol{\varphi}_v^{[b]}(\boldsymbol{\rho})^*.$$

The pion correlation function is build up from two quark lines and is given by

$$C_{l\bar{l}}(t) = c^{(l)}_{\alpha\beta} c^{(\bar{l})*}_{\bar{\alpha}\bar{\beta}} \sum_{\boldsymbol{x}\bar{\boldsymbol{x}}} \mathrm{e}^{-\mathrm{i}\boldsymbol{p}\cdot(\boldsymbol{x}-\bar{\boldsymbol{x}})} \left\langle -\mathcal{Q}^{(\bar{A}A)}_{\bar{\alpha}\bar{\alpha};a\alpha} \mathcal{Q}^{(B\bar{B})}_{a\beta;\bar{\alpha}\bar{\beta}} + \mathcal{Q}^{(BA)}_{a\beta;\bar{\alpha}\bar{\alpha}} \mathcal{Q}^{(\bar{A}\bar{B})}_{\bar{\alpha}\bar{\alpha};a\beta} \right\rangle ,$$

where c is combined group subduction coefficient and a Dirac ($\boldsymbol{\Gamma}$) structure and further

- u, v are compound indices for space, time, color and spin;
- B is the source flavor;
- A is the sink flavor;
- α, β are Dirac spin indices; and
- l, $\bar{l}$ are compound indices for momentum $\boldsymbol{p}$, symmetry group irrep Λ, row λ of the irrep, and symmetry channel.

In the whole analysis from gauge configuration generation to an effective mass, a lot of intermediate steps need to be done. Figure 3.14 shows a graph with the steps and intermediate results. The gauge configurations are generated with Chroma and then also stored as stout smeared configurations. The eigensystems are generated using “LapH_EigSys” [81]. The perambulators are then generated using “peram_gen” [82]. There are two implementations of the contraction code, implementation W [83] and L. These produce correlation functions, one can also directly create correlation functions without the LapH technique using Chroma or tmLQCD. Correlation functions can be then be analysed with various packages, like implementation H [72], U [84] and L.

3.9.3. Improved Meson Masses

The pion mass that was shown in Figure 3.9 has been obtained from correlation functions that were calculated using point sources in Chroma. In the variant with the *exponential solve* definition of the effective mass, the points still fluctuate around the fit band significantly.

A much better signal is obtained by extracting the correlation functions from the perambulators. The excited states will be suppressed to reduce noise and give a better plateau. Liuming Liu has done the necessary contractions to yield the correlation functions. The corresponding effective mass in the *exponential solve* definition is shown in Figure 3.15 where you can directly compare with the point source data from Figure 3.9.

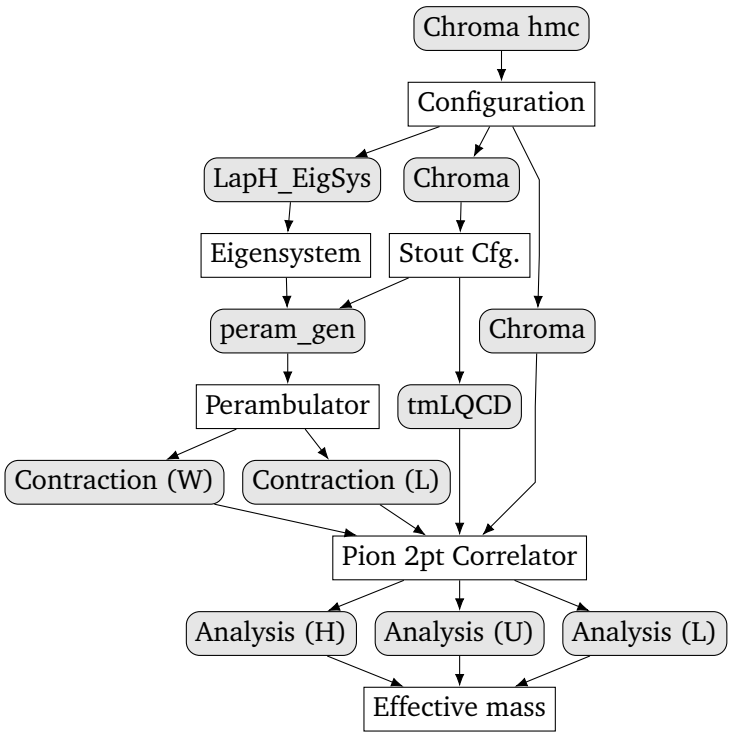


Figure 3.14.: Various ways to extract observables from the generated gauge configurations. Rectangles represent data, filled rounded rectangles represent programs that work with the data.

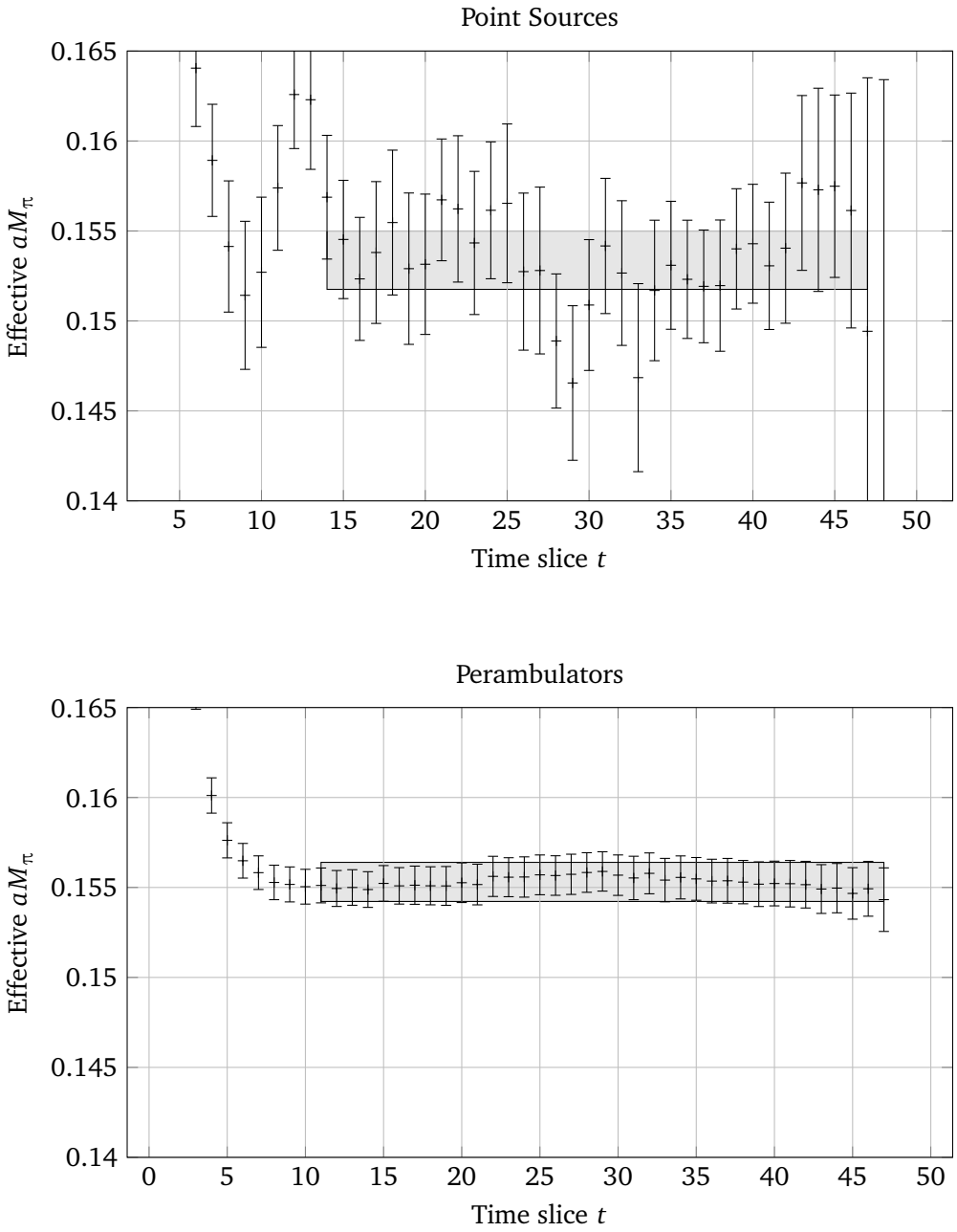


Figure 3.15.: Effective pion mass on the $L = 24$ ensemble with the *exponential solve* definition. The upper plot shows the same data as Figure 3.9 that has been generated using point sources on 221 configurations. The data in the lower plot is generated using perambulators on 186 configurations. The shaded box shows the one standard error band of an uncorrelated fit to the correlators using the R-hadron package. One can see the effectiveness of the excited state removal.

Due to the drastic reduction in noise compare to the point sources, the fits can start at $t = 11$ (or even earlier) as opposed to $t = 14$. The effective mass from the point source is $aM_\pi = 0.1534(16)$, whereas a similar fit using the correlation functions from the perambulators gives 0.1553(10). The errors are likely incorrect since the fits are uncorrelated. Even with the excited state removal, the correlation matrix cannot be estimated properly and fails to be inverted.

3.9.4. Scattering Length

Beyond the masses of the pions, one can also look at pion scattering on the lattice. Due to the imaginary time, extraction of scattering quantities is a bit different from usual quantum field theories in Minkowski space. One measures a two-point function which is the pion–pion correlation function from above. The exponential decay of the correlation is driven by the particle mass m. For the scattering one also computes four-point correlation functions of the form $\langle \pi(t)\,\pi(t)\,\pi(0)^\dagger\,\pi(0)^\dagger \rangle$. The exponential decay factor will be the *interacting* energy $E_{4\mathrm{pt}}$ of the two particles. The energy difference, the binding energy, will depend on the lattice volume L as described by Lüscher [85]. His formalism relates the two energies to the scattering length as

$$2m - E_{4\mathrm{pt}} = \frac{4\pi a_0}{mL^3}\left[1 + c_1\frac{a_0}{L} + c_2\frac{a_0^2}{L^2}\right] + \mathrm{O}(L^{-6}), \tag{3.3}$$

where $c_1 = -2.937\,297$ and $c_2 = 6.375\,183$. For given two-point and four-point energies, the equation is solved numerically for a_0, which is the S-wave scattering length.

The scattering length can be determined in different isospin channels, two pions can couple to $I = 0, 1, 2$. Liuming Liu has run her $I = 0$ analysis [86] on the new gauge configurations and determined the S-wave scattering length to be

$$M_\pi a_0^{I=0} = 1.10(36).$$

The large error likely stems from the small number of configurations that went into this analysis. In the future we plan to generate more configurations and do a more thorough analysis for the scattering. This new data point can be related to the existing data [86] and is shown in Figure 3.16. The M_π/f_π value has been estimated using chiral perturbation theory, therefore it is not as accurate as the measured values on the other two points.

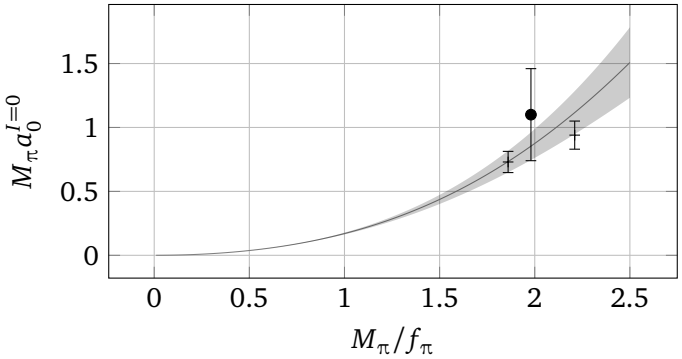


Figure 3.16.: New data point (marked with circle) together with the previous data and fit from Liu et al. [86].

4. Extending QPhiX

The most expensive part of a lattice QCD simulation is the inversion of the fermion matrix $\boldsymbol{M}$ in the HMC. The inversion has to be done for each time step in the molecular dynamics step. There are somewhere between 10 to 100 time steps for each Markov chain step. For a typical simulation we want to have in the order of 5000 Markov updates. The computation of correlation functions needs more inversions, each point source propagator needs twelve inversions, perambulators even more.

A lattice spinor $\boldsymbol{\psi}$ defined on a lattice has spin and color elements on each site. These are complex numbers, which each take 16 B to store in double precision.[1] On a modest $32^3 \times 96$ lattice, the needed memory for a spinor therefore is

$$32^3 \cdot 96 \cdot 4 \cdot 3 \cdot 16\,\mathrm{B} = 576\,\mathrm{MiB}\,.$$

The inverse matrix $\boldsymbol{M}^{-1}$ would then require

$$(32^3 \cdot 96 \cdot 4 \cdot 3)^2 \cdot 16\,\mathrm{B} = 20.25\,\mathrm{PiB}\,.$$

For a $64^3 \times 128$ lattice, this would be 2.25 EiB. Both sizes exceed the size of any computer memory by far, it is unfeasible to store the inverse matrix. This excludes algorithms like Gaussian elimination where the inverse matrix has to be stored during the computation.

The force computation in the molecular dynamics does not really require $\boldsymbol{M}^{-1}$, only the result of $\boldsymbol{M}^{-1}\psi$ has to be computed. Therefore it is sufficient to compute the spinor $\boldsymbol{\chi}$ defined by the equation $\boldsymbol{M}\boldsymbol{\chi} = \boldsymbol{\psi}$. Krylov subspace solvers like conjugate gradient [36] can provide a solution without having to store $\boldsymbol{M}^{-1}$. They only need memory for a few spinors, which is readily available. This class of solvers works by repeatedly applying the matrix to the vector. Therefore one needs to have a fast matrix–vector multiplication.

[1] In the following, the unit "B" stands for "byte" and 1 MiB for 1024^2 B, which is approximately $1000^2\,\mathrm{B} = 1\,\mathrm{MB}$. Similarly $1\,\mathrm{PiB} = 1024^5$ B and $1\,\mathrm{EiB} = 1024^6$ B.

The kinetic operator $\boldsymbol{M}$ consists of a local and a hopping part. The local part contains the mass, twisted mass, and the clover term. The hopping part stems from the discretized derivative which links nearest-neighbor lattice sites. In the operator $\boldsymbol{M}^\dagger\boldsymbol{M}$, the hopping will link next-to-nearest-neighbors together. Thinking about the spacetime structure of M, it will consist only of a handful of diagonals (from the bulk) and some non-zero values on the edges (from the boundary conditions). It is a very sparse matrix, therefore the multiplication $\boldsymbol{M}\chi$ really is a *stencil* operation. This name is given to update procedures where the new value of an array element only depends on the value of the neighboring values.

Sparse matrix multiplications suffer from low *arithmetic intensity*. This is the ratio of floating point operations done and the number of bytes transferred. Modern CPUs with high clock speeds require a large arithmetic intensity in order to be saturated with computations. The sparse multiplication $\boldsymbol{M}\chi$ will be memory bandwidth bound on current high performance platforms. Therefore a lot of optimization is needed to gain performance.

The QPhiX library [87] by the USQCD collaboration provides fast solvers that can be used in HMC packages like Chroma or tmLQCD. In the following sections the library will be introduced and added features described. For a more detailed introduction to QPhiX see the master thesis of Peter Labus [88].

4.1. Code Generator

QPhiX targets *Intel Xeon* processors and *Intel Xeon Phi* coprocessors. This type of hardware is currently very common in high performance systems. The "JURECA" system in Jülich (Germany) and the "Hazel Hen" system at HLRS in Stuttgart (Germany), use *Intel Xeon E5-2680 v3 Haswell* [89] processors, two per mainboard. These support the AVX2 instruction set architecture (ISA).

The A2 partition of the "Marconi" system at CINECA in Casalecchio di Reno (Italy), uses *Intel Xeon Phi 7250* [90] processors. They are also called *Knights Landing* (KNL) and are the successors of the *Knights Corner* (KNC) PCIe cards. The KNL processors support the AVX512 instruction set architecture. Other KNL machines are "DEEP-ER" in Jülich, "Frioul" at CINES in Montpellier (France) and "Cori" at NERSC in Berkeley (USA). The "JURECA" system will be extended with a KNL booster soon. An introduction to the KNL architecture can be found in the talk by Duran [91].

Both architectures have rather long SIMD vector lengths (256 bit and 512 bit) and a two-digit core number (12 and 72). Both support SMT, the Xeon has two hyperthreads, the KNL has four hardware threads. Both Xeon and Xeon Phi use the x86 architecture, they can run "normal" programs. The Xeon Phi is Intel's competitor to NVIDIA's Tesla PCIe cards. The KNL is a bootable CPU and therefore does not suffer from the host–device transfer limitations that the PCIe cards exhibit. Also one has access to the whole host memory due to the homogeneous architecture. Another selling point of the KNL is the ability to run any x86 code directly. This should attract scientist with large legacy code bases that cannot be rewritten using CUDA in a feasible way. In reality, obtaining significant performance on KNL needs a lot of programming effort which seems comparable to the writing of GPU code. Therefore it does not seem to make a large difference in terms of effort for lattice QCD applications.

For best performance, it is desirable to exploit the specific ISA and make use of the full SIMD length. Each ISA has different intrinsic functions that the C++ compiler will translate to special machine instructions, an overview of Intel intrinsics can be found in Reference [92]. There is no simple way of writing a single code that is then compiled into the correct intrinsic functions. Compilers might unroll loops into SIMD instructions, but they tend to be conservative. OpenMP supports a `simd` directive, but that is not sufficient. The data structures have to be tailored for the SIMD length such that similar elements are packed closely together and allocations are *aligned* properly. Alignment means that memory addresses are cleanly divisible by the vector length.

In QPhiX these problems are addressed via a code generator. This is a program that generates C++ code files containing the stencils (also called kernels). These are then wrapped with higher level code that uses those kernels to build Krylov solvers. The generated code does not contain loops over matrix indices. It is a flat stream of instructions that use the intrinsic functions for a particular ISA.

In the code generator, the kernels are composed from C++ objects that represent the instructions. The CPU has instructions for adding and subtracting vectors, there are instructions like $a\boldsymbol{x} + \boldsymbol{y}$ (axpy or *fused multiply add*). Also there are *prefetches* that can load a certain region of memory into the L1 or L2 cache. The registers in the CPU are also represented by C++ objects. A kernel then consists of a list of these instruction objects, working on register objects. This makes the kernel code readable and portable. To generate the fastest possible axpy operation, one writes the following in the code generator:

```
fmaddFVec(ivector, Ret, A, X, Y, mask);
```

Architecture	Generated Code
Scalar	`ret = (a * x) + y`
AVX	`ret = _mm256_add_pd(_mm256_mul_pd(a, b), c)`
AVX2	`ret = _mm256_fmadd_pd(a, x, y)`
AVX512	`ret = _mm512_fmadd_pd(a, x, y)`

Table 4.1.: Generated code for the supported architectures to perform the axpy operation.

The `ivector` is an `std::vector` of instruction objects which will be written to a file at the end. The next four parameters are C++ objects representing the variables to be used in the computation. The `mask` is an optional parameter that can exclude certain elements of the SIMD vector from the computation. For each ISA, there are different bits of C++ generated.

The various generated lines are listed in Table 4.1. On the scalar architecture we cannot use any special functions and generate a regular operation. AVX supports 256 bit intrinsics, so we can use a SIMD multiplication and addition. AVX2 introduced the fused-multiply-add, therefore only a single instruction needs to be generated. For the AVX512 target, the same instruction objects generates a 512 bit fused multiply add of packed doubles.

Loops in the code generator are evaluated during the generation of the kernels. The generated code will not contain any loops over color and spin indices, but rather a block of 12 similar looking instructions next to each other. This reduces runtime overhead and improves the performance for compute bound parts of the kernel. Having the complete stream of instructions available in the code generator also means that one knows exactly when each variable is accessed. In a second code generation step, explicit software prefetches to the L1 and L2 cache can be interspersed with the computations, allowing to reduce the effects of memory latency. This was needed for the KNC architecture, the KNL architecture has much better hardware prefetches so that we do not utilize the software prefetches any more.

The code generator allows writing kernel functions in an abstract way but yet obtain ISA specific intrinsics code. Once all elementary instructions have been implemented for one ISA, all kernels can be generated for that ISA without any change to the code describing the kernel. This make adoption of a new architecture much easier while still retaining great performance.

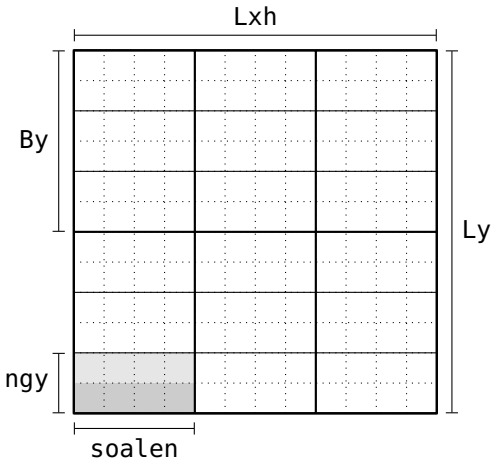


Figure 4.1.: Blocked data layout used in QPhiX. The original lattice (dotted grid) is divided into tiles (thin grid) and blocks (thick grid). Adapted from Joó et al. [87].

4.2. Data Layout

In order to use SIMD capabilities in a program, the data layout has to be chosen in terms of SIMD vectors. Different architectures have different SIMD lengths, therefore the data layout has to be sufficiently flexible. On an AVX512 platform in single precision, we need $512/32 = 16$ elements in a SIMD vector. The SIMD vector length in terms of the chosen floating point type is called `veclen` in the program. For AVX, it takes the value 8. In double precision, the length is halved.

The SIMD vector index must be the fastest index of the data structure. For a spinor field, the natural choice of the fastest index would either be the spin or color index. However, the range of that index is always fixed and does not give the needed flexibility. Therefore QPhiX uses a data layout where part of the x-dimension is used as the fastest index.

Figure 4.1 illustrates the partitioning of the lattice which works as follows. The x-dimension is divided into chunks of length `soalen`, which is a divisor of `veclen`. In this example, we have $\texttt{soalen} = 4$ and $\texttt{veclen} = 8$. In the figure, two chunks of a single vector are shown as differently shaded areas. A number `ngy` of chunks is combined into a *tile* which has $\texttt{soalen} \cdot \texttt{ngy} = \texttt{veclen}$ elements. This tile is then used as a SIMD vector. In the figure the tiles are delimited with thin solid

lines.

This combination of x and y direction in a single SIMD vector is called *vector folding*. The "Dslash" operation connects nearest neighboring lattice sites together, therefore it appears advantageous to load neighbors in both x and y with a single SIMD vector. Pushing this concept further to include z and t could be done; sending and receiving lattice sites to a neighboring process will require unpacking and packing in all directions then. QPhiX only has x and y directions partially folded, therefore communication along z and t can be done with full SIMD vectors without repacking. This explains the dependence of the performance on the domain decomposition as shown in Table 3.3.

The `soalen` parameter needs to be tuned for the particular machine to find the best performance. A number of tiles is grouped together to form a *block*, as indicated with thick solid lines. The blocks are loaded into the cache one at a time. Depending on the cache sizes, different block sizes provide optimal performance. The z-direction is divided into blocks of length `Bz`. The time direction is not divided, each block contains the full time extent.

The four-spinor fields have a slightly different memory layout. Their fastest index does not loop over `veclen` but only `soalen` elements. Multiple different spinors are loaded at the same time and then are ordered with permute and shuffle instructions. Effectively this means that the x and y coordinates are split into three indices of the data structure.

The definitions of the data structures is done in the `Types` class. The spinor and gauge fields are defined as follows:

```
typedef FT FourSpinorBlock[3][4][2][soalen];
typedef FT TwoSpinorBlock[3][2][2][veclen];
typedef FT SU3MatrixBlock[8][(compress12 ? 2 : 3)][3][2][veclen];
```

The type `FT` is the floating point type. The indices for the spinor blocks are color, spin, complex, and the SIMD index.

The gauge field has one direction index with range 4. As introduced in Figure 2.1, there are four forward gauge links $\boldsymbol{U}_\mu(\boldsymbol{n})$ and four backward gauge links $\boldsymbol{U}_\mu(\boldsymbol{n})^\dagger$. The link from $\boldsymbol{n}$ to $\boldsymbol{n}+\hat{\boldsymbol{\mu}}$ is stored on lattice site $\boldsymbol{n}$. However, the link from $\boldsymbol{n}$ to $\boldsymbol{n}-\hat{\boldsymbol{\mu}}$ is stored on lattice site $\boldsymbol{n}-\hat{\boldsymbol{\mu}}$. The Dslash stencil operation needs to have all eight gauge links pointing away from a given lattice site. The gauge field does not change during the solving of $\boldsymbol{M}\boldsymbol{\psi}=\boldsymbol{\chi}$, therefore memory latency impact is reduced by copying the gauge field from $\boldsymbol{n}-\hat{\boldsymbol{\mu}}$ to $\boldsymbol{n}$. In total, there are then eight gauge

fields on each lattice site, hence the range 8 of the first index. QPhiX does not use hermitian conjugated backward links, rather the multiplication routines conjugate it on the fly. Therefore the gauge field is simply duplicated and distributed to other lattice sites.

The next two indices for the gauge field are the color indices. Since the arithmetic intensity is low, additional compute cycles are "free" while waiting for data from memory. A SU(3) matrix only contains eight real degrees of freedom. Storing the full matrix naively requires nine complex numbers, that are 18 real numbers. The matrix exponential or other decompression techniques that are needed to retrieve the matrix from only eight parameters are too expensive. It will require more compute cycles than are available for free, becoming the new bottleneck. On GPUs, the needed arithmetic intensity is higher, therefore it might make sense to use these there. For KNL, a good compromise is the compression to 12 real parameters. The last row of the matrix is discarded and reproduced on the fly. The second last index spans real and imaginary part, the last index is the SIMD index, as usual.

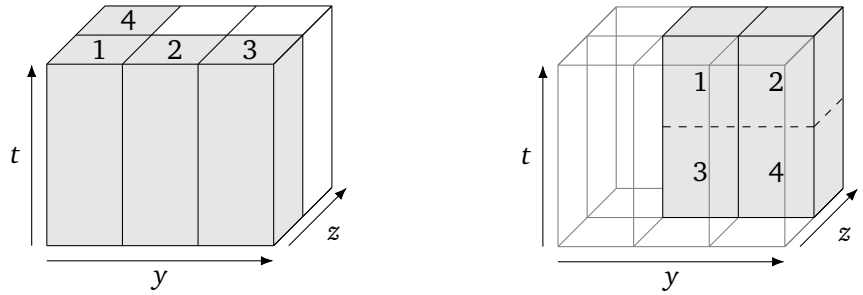
Twelve parameter gauge compression works as follows. One discards the last row vector of the matrix

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} \rightarrow \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{pmatrix}$$

to save memory bandwidth and then reproduces this with a vector product inside the kernels, $\boldsymbol{c} = (\boldsymbol{a} \times \boldsymbol{b})^*$. This is the gauge compression available in QPhiX.

The four parameters describing the data layout, floating point type (`typename FT`), vector length (`int veclen`), soalen (`int soalen`), and a flag for gauge compression (`bool compress12`), are used as C++ template parameters throughout the codebase. Their naming unfortunately is not completely consistent, their order is the always same, though.

As hinted at earlier, multiple slices in the y direction are grouped into blocks. The same is done in the z direction also. These blocks in y and z are distributed evenly to the available cores. As long as there are at least as many blocks as threads, every threads gets one block assigned to it as indicated with shading in Figure 4.2a. Blocks are assigned with the full x and t extent. At some point, there might be fewer blocks than cores. The t direction is split such that there are at least as many blocks as cores again, see Figure 4.2b. In order to prevent splitting into



(a) More blocks than threads (b) Less blocks than threads

Figure 4.2.: Parallelization in 3.5 dimensions shown with four threads and six blocks. Adapted from Joó et al. [87].

miniscule work packages, the splitting is halted after a few iterations and cores are left idling.

Between rounds of distributing blocks to cores, one can enable a thread barrier that synchronizes all threads. Usually barriers are known to decrease performance because they force threads to idle. Contrary to intuition they can give a slight performance increase on certain architectures because they synchronize memory accesses. This improves data sharing and therefore bandwidth usage because the threads do not drift apart too much. On the KNL platform we do not see any benefit, they are disabled by default.

4.3. Implementing Non-Degenerate Twisted Mass

QPhiX is written exclusively with even-odd preconditioning. It provides a solution to $\tilde{\boldsymbol{M}}_{\mathrm{oo}}\psi = \chi$, given $\boldsymbol{M}_{\mathrm{oo}}$ and $\boldsymbol{M}_{\mathrm{ee}}^{-1}$ as external arrays.[2] In the chapter on lattice theory we have introduced the preconditioned operator in Equation (2.11) with $d := \alpha^2 + \mu^2 - \epsilon^2$:

$$\tilde{\boldsymbol{M}}_{\mathrm{oo}} = \begin{pmatrix} \alpha + \mathrm{i}\mu\gamma_5 - \frac{1}{4d}\not{\boldsymbol{D}}_{\mathrm{oe}}(\alpha - \mathrm{i}\mu\gamma_5)\not{\boldsymbol{D}}_{\mathrm{eo}} & -\epsilon - \frac{\epsilon}{4d}\not{\boldsymbol{D}}_{\mathrm{oe}}\not{\boldsymbol{D}}_{\mathrm{eo}} \\ -\epsilon - \frac{\epsilon}{4d}\not{\boldsymbol{D}}_{\mathrm{oe}}\not{\boldsymbol{D}}_{\mathrm{eo}} & \alpha - \mathrm{i}\mu\gamma_5 - \frac{1}{4d}\not{\boldsymbol{D}}_{\mathrm{oe}}(\alpha + \mathrm{i}\mu\gamma_5)\not{\boldsymbol{D}}_{\mathrm{eo}} \end{pmatrix}_{\mathrm{f}}.$$

[2] In the American notation one uses $\boldsymbol{A}$ instead of $\boldsymbol{M}_{\mathrm{oo}}$ and $\boldsymbol{A}^{-1}$ instead of $\boldsymbol{M}_{\mathrm{ee}}^{-1}$. Within QPhiX, they are usually called `clov` and `invclov`.

Previously, QPhiX worked only with single flavor fermion fields. Therefore only the upper left part of this matrix has been implemented,

$$\tilde{M}^{uu}_{oo} = \alpha + i\mu\gamma_5 - \frac{1}{4d}\not{D}_{oe}(\alpha - i\mu\gamma_5)\not{D}_{eo}\,.$$

The subscript indices are the checkerboard indices (even and odd), the superscript indices are the flavor indices (up and down; alternatively charm and strange). In the general case (potentially with clover term) this would be written as

$$\tilde{M}^{uu}_{oo} = M^{uu}_{oo} - \frac{1}{4}\not{D}_{oe}(M^{-1}_{ee})^{uu}\not{D}_{eo}\,.$$

The application of this operator to a spinor field needs two applications of the hopping matrix D which connects neighboring lattice sites. In QPhiX, these two applications are implemented in two separate steps such that only an implementation of the hopping matrix is needed and only a one element thick boundary needs to be communicated between the MPI ranks. The first part is the "Dslash" operation[3] which computes $\psi^u_e = (M^{-1}_{ee})^{uu}\not{D}_{eo}\chi^u_o$. Then the "achimbdpsi" ($\alpha\chi - \beta\not{D}\psi$) operation uses this intermediate result and computes $M^{uu}_{oo}\chi^u_o - \beta\not{D}_{oe}\psi^u_e = \tilde{M}^{uu}_{oo}\chi^u_o$.

The original authors of QPhiX, Joó et al. [87], have implemented Wilson and Wilson clover fermions. This has been extended to Wilson twisted mass fermions by Schröck, Simula, and Strelchenko [93]. My colleague Peter Labus [88] has further extended QPhiX to include the twisted mass clover case $\mu \neq 0$. As part of this this work I have extended QPhiX to the non-degenerate case ($\epsilon \neq 0$) with help from my colleague Bartosz Kostrzewa.

In order to reuse as much of the existing code as possible, we express the non-degenerate operations in terms of the degenerate ones which are already implemented in QPhiX. We will first treat the non-degenerate Wilson twisted mass case without a clover term. The application of the even-odd preconditioned operator on a two-flavor spinor field is explicitly given by

$$\begin{aligned}(\tilde{M}_{oo}\chi_o)^u &= \left[\alpha + i\mu\gamma_5 - \frac{1}{4d}\not{D}_{oe}(\alpha - i\mu\gamma_5)\not{D}_{eo}\right]\chi^u_o - \left[\epsilon + \frac{\epsilon}{4d}\not{D}_{oe}\not{D}_{eo}\right]\chi^d_o \\ (\tilde{M}_{oo}\chi_o)^d &= \left[\alpha - i\mu\gamma_5 - \frac{1}{4d}\not{D}_{oe}(\alpha + i\mu\gamma_5)\not{D}_{eo}\right]\chi^d_o - \left[\epsilon + \frac{\epsilon}{4d}\not{D}_{oe}\not{D}_{eo}\right]\chi^u_o\,.\end{aligned}$$

[3] The naming convention is a bit unfortunate. The "Dslash" operation in QPhiX always means a hopping matrix with M^{-1}_{ee} included, so the operation is $M^{-1}_{ee}\not{D}_{eo}$, whereas "Dslash" is also often used as another name for the hopping matrix alone.

The first bracket contains the "achimbdpsi" operation in the Wilson twisted mass case. For the down-flavor, the sign of the twisted mass μ is flipped. The second bracket can be interpreted as a Wilson "achimbdpsi" operation with appropriately set α and β. The two intermediate results would need to be subtracted from each other to give the final results. This method needs eight applications of the hopping matrix and therefore is more expensive than needed.

Instead, the following variant has been chosen:

1. First the Wilson "Dslash", which is just the hopping matrix, is applied to both flavors of the spinor, $\not{D}_{\mathrm{eo}}\boldsymbol{\chi}_{\mathrm{o}}^{\mathrm{u}}$ and $\not{D}_{\mathrm{eo}}\boldsymbol{\chi}_{\mathrm{o}}^{\mathrm{d}}$.
2. A newly written routine multiplies the intermediate results with $\alpha \mp \mathrm{i}\mu\boldsymbol{\gamma}_5$ and stores that in another temporary spinor.
3. This spinor is then used as input in the Wilson twisted mass "achimbdpsi" operation to yield the first brackets indicated above.
4. The result from step "1" is used in the Wilson "achimbdpsi" operation to yield the second brackets.
5. A BLAS routine subtracts the two parts in each flavor and stores the results into the final spinors.

This approach only needs six applications of the hopping matrix and should be faster because it needs fewer memory accesses. The gauge field is still streamed from memory once for each flavor. It might be faster to generate kernels that directly do the "Dslash" operation on two flavors at once. This way, the gauge (and clover) fields need to be streamed through only once. This would need new two flavor kernels that need to be generated in the code generator. Also the data structures would need to be changed in order to hold multiple flavors. If the performance of the current implementation is not sufficient, these optimizations could be implemented and tested against the each other. These significant changes are beyond the scope of this work and might be taken up in a future work.

The Wilson twisted mass clover operator works analogously. Since the clover term cannot be inverted analytically, one cannot decompose it as nicely. The most explicit form of the operator is given by

$$\tilde{\boldsymbol{M}}_{\mathrm{oo}} = \begin{pmatrix} \alpha + \mathrm{i}\mu\boldsymbol{\gamma}_5 + c_{\mathrm{SW}}\boldsymbol{T}_{\mathrm{oo}} \\ \alpha - \mathrm{i}\mu\boldsymbol{\gamma}_5 + c_{\mathrm{SW}}\boldsymbol{T}_{\mathrm{oo}} \end{pmatrix}_{\mathrm{f}} - \epsilon\boldsymbol{\tau}^1 - \frac{1}{4}\not{D}_{\mathrm{oe}}\boldsymbol{M}_{\mathrm{ee}}^{-1}\not{D}_{\mathrm{eo}}\,.$$

The matrix M_{ee}^{-1} is non-trivial in flavor, all four submatrices in flavor space will potentially be non-zero. We can regroup the terms such that the first term can be

evaluated with previously existing functionality and a the second term needs new routines.[4] We obtain

$$\begin{aligned}(\tilde{M}_{oo}\chi_o)^u &= \left[\alpha + i\mu\gamma_5 + c_{SW}T_{oo} - \frac{1}{4}\not{D}_{oe}(M_{ee}^{-1})^{uu}\not{D}_{eo}\right]\chi_o^u \\ &\quad - \left[\epsilon + \frac{1}{4}\not{D}_{oe}(M_{ee}^{-1})^{ud}\not{D}_{eo}\right]\chi_o^d \\ (\tilde{M}_{oo}\chi_o)^d &= \left[\alpha - i\mu\gamma_5 + c_{SW}T_{oo} - \frac{1}{4}\not{D}_{oe}(M_{ee}^{-1})^{dd}\not{D}_{eo}\right]\chi_o^d \\ &\quad - \left[\epsilon + \frac{1}{4}\not{D}_{oe}(M_{ee}^{-1})^{du}\not{D}_{eo}\right]\chi_o^u .\end{aligned}$$

While the first bracket is the Wilson twisted mass clover "achimbdpsi", the second term is a mixture of non-clover and clover operations. One could use the same "achimbdpsi" operation again with $M_{ee}^{ud} = M_{ee}^{ud} = \epsilon$. The data structure for that matrix has $2 \cdot 6^2$ c-number entries (two spin blocks with half-spin and color), totaling in 144 floating point numbers *per lattice site*.

The solvers must be adapted to the two flavor operator. Instead of single spinor fields they take arrays of N_f spinor fields. The clover term must now be passed as an N_f^2 matrix. Internally, the solver use various BLAS routines like "axpy" ($y := ax + y$) or "norm2" ($|x|^2$). These routines have been generalized to N_f flavors by calling the existing implementation multiple times. The square norm needs a reduction in the end. The results from each flavor are computed separately and the single flavor results are then added up to give the multi flavor result. The number of flavors is passed as a C++ template argument, the compiler should be able to easily unroll the loop and inline the single flavor implementation.

[4] As an intermediate step, one can just write the flavor structure of τ^1 and M_{ee}^{-1} explicitly and apply that to the two-flavor spinor field. The resulting flavor components can be written as

$$\begin{aligned}(\tilde{M}_{oo}\chi_o)^u &= [\alpha + i\mu\gamma_5 + c_{SW}T_{oo}]\chi_o^u - \epsilon\chi_o^d - \frac{1}{4}\not{D}_{oe}(M_{ee}^{-1})^{uu}\not{D}_{eo}\chi_o^u - \frac{1}{4}\not{D}_{oe}(M_{ee}^{-1})^{ud}\not{D}_{eo}\chi_o^d \\ (\tilde{M}_{oo}\chi_o)^d &= [\alpha - i\mu\gamma_5 + c_{SW}T_{oo}]\chi_o^d - \epsilon\chi_o^u - \frac{1}{4}\not{D}_{oe}(M_{ee}^{-1})^{dd}\not{D}_{eo}\chi_o^d - \frac{1}{4}\not{D}_{oe}(M_{ee}^{-1})^{du}\not{D}_{eo}\chi_o^u .\end{aligned}$$

4.4. Twisted Boundary Conditions

The twelve parameter gauge compression only works for SU(3) matrices. Twisted boundary conditions work by multiplying every single element of the gauge field with a complex phase. The matrices are then contained in U(3) and have nine real parameters. Twelve parameter gauge compression will destroy this phase. When using twisted boundary conditions, the only possibility had been to disable the gauge compression. This would work but give less performance.

QPhiX has been extended to support compression for twisted boundary conditions as part of this work. The gauge field itself is not changed in general, it is only changed when multiplied to a spinor. QPhiX only multiplies the upper half-spinor with the gauge field and then reconstructs the lower half-spinor from the upper components. This halves the message sizes when spinors are transmitted over the network and reduces the number of operations. The complex phase is added after the gauge–half-spinor multiplication, the two spin components of the half-spinor are multiplied with the phase. This costs a few more floating point operations. The arithmetic intensity is still way below saturation such that this is essentially free. In order to let user only pay this cost when they are interesting in twisted boundary conditions, there are kernels that have twisted boundary conditions enabled and disabled. In total there are 16 combinations of twisted and simple boundary conditions.

4.5. Interface to tmLQCD

QPhiX has been developed as a library to be primarily used within Chroma. The tmLQCD software [54] has its own Krylov solvers implemented. Benchmarks have shown that on architectures like KNL, QPhiX delivers more performance than the tmLQCD solvers. The additions of non-degenerate twisted mass and twisted boundary conditions make QPhiX feature complete for tmLQCD.

An interface was initiated by Schröck [94] and then further pursued by my colleagues Bartosz Kostrzewa and Peter Labus. Now one can use the QPhiX solvers in tmLQCD. For the heavy quarks, this gives a speed-up of around 2 on AVX2, larger speed-ups are expected on AVX512 which at the time of writing still has a minor bug and therefore cannot be tested yet.

For very light quark masses the CG and BiCGStab solvers are inferior to multigrid solvers like implemented in DDαAMG. With the bindings to the solver libraries, one can use different solvers for different monomials which is a significant performance advantage.

In order to expose all kinds of operators that are implemented in QPhiX, an abstract base class was introduced on the tmLQCD side. Eventually this structure could be moved into the QPhiX codebase. tmLQCD is a pure C code, so the C++ functions of QPhiX had to be wrapped into C functions. Template parameter must be expressed as simple C types, there is a lot of template instantiation code in the interface.

Another issue faced along the way are different parameterizations of the theory. tmLQCD uses the κ-parameterization whereas QPhiX does not. Functions that convert spinors and gauge fields between the two programs had to be developed and tested. Data layout of QPhiX is fundamentally different from tmLQCD, a lot of index manipulation is needed to transfer the data correctly. The designated SIMD direction in QPhiX is the x-direction, in tmLQCD this is the z-direction. An interchange of these directions as well as a compensating change in the $\boldsymbol{\gamma}$-matrices has alleviated this problem. The order of MPI directions in QPhiX is $tyzx$ whereas tmLQCD has $txyz$. Fortunately the QMP library supports a dimension mapping which can easily compensate for this difference.

4.6. Twisted Preconditioning Mass

The twisted mass clover in QPhiX is implemented such that the twisted mass μ is added to the clover term $\boldsymbol{T}$ and stored in a single data structure. tmLQCD instead only stores the clover term and adds the twisted mass on the fly.

A similarly independent twisted mass ρ has been added to the QPhiX operators. The $\mathrm{i}\rho\boldsymbol{\gamma}^5$ has been implemented in the "achimbdpsi" kernel function where it will unconditionally add this to the output spinor field. Instead of making this yet another template parameter, it was decided that these four flops will not be noticed in the over thousand flops needed for the whole operator application. If we should ever have an architecture where we are not bound by memory access, this decision can be revisited. On current platforms we are far from saturation, on KNL by a factor of at least 8.

Incidentally, the structure of this term is exactly the same as the twisted mass itself, at least in the degenerate case. Therefore the implementations of plain Wilson and

twisted mass Wilson “achimbdpsi” become equivalent from the code and one could think of using only this twisted preconditioning mass in the degenerate cases. This would allow for more code re-use.

4.7. Software Engineering

QPhiX has around 40 000 lines of code but provides a solver only. It has to be used in packages like Chroma or tmLQCD. These have 500 000 and 140 000 lines of code, respectively. This means that there is a lot of complexity that needs to be maintained over the timespan of decades. It must be ensured that the code is working correctly and does not stop working without anyone noticing.

The operations implemented in QPhiX are already implemented in the QDP++ library. The existing implementation is tested and has been used to produce results, this is the reference implementation for QPhiX. There are a lot of unit tests that test the QPhiX functions against their QDP++ equivalents. Incidentally, around a third of source lines are for testing.

Unit tests are only useful if they are run often. Projects often suffer from commits that do not even compile cleanly or fail the tests. For QPhiX we therefore use *continuous testing* using Travis CI [95]. Every git commit that is pushed to GitHub will be compiled from scratch using GCC and OpenMPI. Then all test cases run through. If any of that fails, the commit is marked and the developer can fix it quickly. This way it is possible to provide a stable `devel` branch.

A project under heavy development often suffers from the buildup of low quality code. It is important to refactor code every once in a while to make addition of features sustainable. Several thousand lines of code were redundant and could be deleted by mild refactoring steps. Addition of new features like twisted boundary conditions or non-degenerate doublets then only added a manageable amount of new code.

Incremental compilation time has a big impact on developer productivity. If a minor change causes a ten minute rebuild of the project, a lot of potential focus is lost. In QPhiX the root cause for long compilation is the use of C++ templates which make QPhiX mostly a header-only library. All the kernels have been included in the headers as well, resulting in recompilation of millions of lines of generated code for every single test program. This issue has been alleviated by the use of *explicit template instantiation* where template functions are compiled into a static

library for a certain set of template parameters. This has reduced the amount of code that needs to be recompiled significantly, saving a lot of time in incremental builds. Also the RAM requirement for each compiler process has gone down from 2000 MiB to 300 MiB. Incremental builds of the test cases now takes a couple ten seconds instead ten minutes.

The addition of twisted boundary conditions has made it necessary to build 16 times as many kernels. Without the changes to the build system, incremental compilation time would be in the order of hours, needing in the order of 30 GiB of memory. This would make automated testing on Travis CI impossible. Now it is still possible to compile the 20 million lines of code in a reasonable amount of time.

New features are implemented in a separate git branch. In order to make the feature available for all users, it needs to be merged into the main branch, usually `devel` or `master`. While developing a feature, one should regularly merge changes on the main branch into the feature branch to ensure that the new feature stays compatible with the remaining code. Unfortunately, a lot of work had been done in such feature branches with no effort to merge them back into the main branch. Therefore these features are not available, at least not usable together with other features. After a while the main branch has advanced in a way that it becomes incompatible with the proposed features. If one eventually gets the feature into the main branch, a lot of changes might be needed to actually get the feature in. Therefore in this work we have tried to work together with the upstream authors to get features merged in quickly.

5. Conclusion & Outlook

During the course of this work, experience with the Chroma software has been gathered, further HMC setups can now be developed rather quickly. The new Wilson clover ensembles have been preliminarily analyzed and the pion mass was extracted. These configurations will be analyzed further in the future, especially with the σ resonance in mind.

QPhiX has gained a couple new features, most importantly the non-degenerate twisted mass operator. With the interface to tmLQCD it is now possible to perform HMC simulations with very light quarks (using DDαAMG) and more efficient charm and strange quarks (using QPhiX).

The implementation of the non-degenerate doublet in QPhiX can be made faster, still. The communication strategy was modified earlier this year, the changes were only implemented for non-twisted mass fermions. These modifications could be brought to the twisted mass case. Also there are more applications of the hopping matrix than strictly needed, this could be alleviated with a custom kernel code.

A. XML File for Chroma

Chroma uses XML files for input of the parameters and output of the measurements. In this chapter, a couple of full examples as well as a couple of hints to construct these files will be given. A broad overview of Chroma XML files is given by Joó [96].

For parsing the output XML, good experiences have been made with XML libraries supporting XPath. Using `grep` on XML is not recommended.

In order to compile Chroma, have a look into my compilation script[1] that supports JURECA and Marconi A2 and Hazel Hen. Typical error messages that have been observed in the course of this thesis have been published on the Chroma wiki[2].

A.1. Full Examples

On my website, there is a section with supplementary materials. Including these text files in this thesis would make it around twenty pages longer without it being useful.

`http://martin-ueding.de/studies/msc_physics/physics900/index.html`

A.1.1. HMC

One supplement is a full parameter file for the `hmc` program of Chroma, named `hmc.xml`. It is the one used in the simulations in this thesis. It should cover a lot of interesting points as it employs

- $N_\mathrm{f} = 2 + 1$ Wilson clover quark flavors;

[1] `https://github.com/martin-ueding/chroma-auxiliary-scripts/tree/master/compilation`

[2] `https://github.com/JeffersonLab/chroma/wiki/Compiling-Chroma`

- mass preconditioning terms on different time scales;
- a rational approximation that is split into two time scales; and
- different solver residuals for the molecular dynamics and the acceptance step.

A.1.2. Meson Correlators

The file `meson.xml` is a XML input for `chroma` which will read in a gauge configuration and compute pion and kaon correlators from them. After `chroma` has been run, there will be a file containing the correlation data. There will be 16 blocks which each contain a `gamma_value` tag. Also for every momentum used, the correlator will be given as a sequence of real and imaginary parts. The number of those inner blocks depends on the number of time slices of the gauge field.

```
<Point_Point_Wilson_Mesons>
  <elem>
    <gamma_value>0</gamma_value>
    <momenta>
      <elem>
        <sink_mom_num>0</sink_mom_num>
        <sink_mom>0 0 0</sink_mom>
        <mesprop>
          <elem>
            <re>0.535827303605439</re>
            <im>0</im>
          </elem>
          <elem>
            <re>-0.0651788529561225</re>
            <im>0</im>
          </elem>
          ...
        </mesprop>
      </elem>
    </momenta>
  </elem>
  ...
</Point_Point_Wilson_Mesons>
```

Chroma always computes all 16 possible Γ structures, the $\bar{\psi}\Gamma_i\psi$ operators. Their numbering system comes from a binary and corresponds to the Dirac matrices γ_i used. For instance for the pion we need $\Gamma = \gamma_5$. Here the definition is $\gamma_5 = \gamma_1\gamma_2\gamma_3\gamma_4$. We use the matrices 4321 which gets encoded into the binary mask

1111_2. Converted to decimal, that is 15_{10}. Therefore the pion (or kaon) correlator is in the block with `<gamma_value>15</gamma_value>`.

A complete table of these structures can be found in the Chroma repository at `docs/notes/chroma_gamma_matrices.tex`.

A.1.3. Wilson Flow

Computing the Wilson flow is a single measurement that can be done with the `chroma` program using a file like `wflow.xml`.

The integer in `nstep` gives the number of integration steps, `wtime` is the total flow time. The step size is the quotient of those numbers. `t_dir` specifies the time direction (usually 4). Chroma will output two averages, one of the temporal and another of the spatial components. By setting `t_dir` to 5, one can average isotropically over all space and time directions.

The output will contain all the details of the configuration and one block with three tags:

```
<wilson_flow_results>
  <wflow_step>0 0.01 ... </wflow_step>
  <wflow_gact4i>-0 -0 ... </wflow_gact4i>
  <wflow_gactij>2.57482770984263 2.57562346045517 ...</wflow_gactij>
</wilson_flow_results>
```

The tag `wflow_step` contains a list of the time steps used in the flow direction. The lists in `wflow_gact4i` and `wflow_gactij` contain the gauge action densities averaged across the F_{4i} components (temporal) and the F_{ij} components (spatial), respectively.

A.2. Finding XML Parameters

There seems to be no central documentation for all the parameters that can be supplied in the input XML file. One can go through the C++ source code to find the possible parameters. The parameters for a certain entity are usually read in the same file as the entity is defined. The entities are loaded into a factory, indexed by the `const std::string name` attribute. Using `grep` to find it in the codebase will yield the filename:

```
$ git grep REMEZ
lib/update/molecdyn/monomial/remez_rat_approx.cc:    const std::string name =
    "REMEZ";
```

Most of the these C++ source files contain a `Params` struct or a `read` function. The following snippet is what to look for:

```
Params::Params(XMLReader &xml, const std::string &path) {
    XMLReader paramtop(xml, path);

    read(paramtop, "numPower", numPower);
    read(paramtop, "denPower", denPower);
    read(paramtop, "lowerMin", lowerMin);
    read(paramtop, "upperMax", upperMax);
    read(paramtop, "degree", degree);

    if (paramtop.count("digitPrecision") != 0)
        read(paramtop, "digitPrecision", digitPrecision);
    else
        digitPrecision = 50;
}
```

From this the parameters can be extracted. The parameter `digitPrecision` is optional and has a default of 50. The resulting XML could be the following:

```
<ratApproxType>REMEZ</ratApproxType>
<lowerMin>1.0e-3</lowerMin>
<upperMax>33</upperMax>
<numPower>-1</numPower>
<denPower>4</denPower>
<degree>16</degree>
```

Another example are the parameters of the QPhiX clover multi-shift inverter. The file where the `std::string name` is defined can be found with `grep` as well:

```
$ git grep QPHIX_CLOVER_MULTI_SHIFT_INVERTER
lib/actions/ferm/invert/qphix/multi_syssolver_mdagm_cg_clover_qphix_w.cc:
    const std::string name("QPHIX_CLOVER_MULTI_SHIFT_INVERTER");
```

The parameters are not defined in that class, though. Traversing the C++ inheritance, one finds the parameters in the file `lib/actions/ferm/invert/qphix/multi_syssolver_qphix_clover_params.cc`. In there, the parameters can be found:

```
MultiSysSolverQPhiXCloverParams::MultiSysSolverQPhiXCloverParams(
    // ...
```

```
    read(paramtop, "MaxIter", MaxIter);
    // ...

    if (paramtop.count("Verbose") > 0) {
        read(paramtop, "Verbose", VerboseP);
    } else {
        VerboseP = false;
    }
    // ...
}
```

In case a name is used that is not known, Chroma will output an error message and print all the possible entities that are available in the given context (monomials, solvers, ...). Parameters that are not needed are silently ignored, this can be irritating at times.

This section has also been published on the Chroma wiki[3].

A.3. Available Names

On the Chroma Wiki, there is a page[4] that lists all the `const std::string name` variables found within the Chroma library. They are extracted and formatted with a small script[5].

[3] https://github.com/JeffersonLab/chroma/wiki/Creating-XML-Input-Files
[4] https://github.com/JeffersonLab/chroma/wiki/Available-XML-Entities
[5] https://github.com/martin-ueding/chroma-auxiliary-scripts/tree/master/extract-names

Acknowledgments

I greatly appreciate that Carsten has given me the opportunity to write this master thesis in his group and supervised me along the way. Working as part of his group is a very pleasant experience and I am looking forward to continue. A lot of software development could be done as part of this thesis which has given me various insights into high performance programming.

Bartek has helped me with countless discussions about physics, programming and running simulations. Also I appreciate that he has set up and run a few jobs for me and provided an analysis script for R-hadron.

The help from Christian, Christopher, Liuming, and Markus regarding the perambulator and contraction generation has been invaluable; I also thank them for all the discussions we had about those concepts.

In my first months on the QPhiX codebase, Peter has showed me around, answered a lot of questions and helped me understand the complex data structures and the code generation.

Carsten and Tom have read a first draft of my thesis and I appreciate all the hints they have given me regarding the text.

Many thanks go to Liuming who has performed the analysis of the gauge configurations using their existing code and provided me with the scattering length from the generated gauge configurations.

Computer time on Hazel Hen, JURECA, and JUQUEEN has been granted by Carsten, Silvano Simula has given me access to Marconi. Without the millions of core-hours I was able to use, this thesis would not have been possible.

Bálint Joó has given advice on working with Chroma and composing input files. He also gave me write access to the QPhiX repository and allowed me to make major changes to the codebase. The credit of trust is greatly appreciated.

Giorgio Amati (CINECA) and Gabriel Hautreux (CINES) have run QPhiX benchmarks on the KNL machines in their institutes to help diagnose performance

issues that we have faced. This helped us understand the performance differences between the various machines.

Also I would like to thank CINECA, Intel, and Lenovo for inviting me to the “KNL Workshop” in Bologna and especially Lenovo for paying for flight, accomodation, dinner, and fancy transportation. The Bonn Cologne Graduate School (BCGS) has payed for a C++ course at the Höchstleistungsrechenzentrum Stuttgart, thank you very much for this trip. Carsten and the university have granted me a standing desk which is also much appreciated.

The Travis CI GmbH runs the QPhiX unit tests for every commit for free, we appreciate this gift to the free software community.

Without Alisa, Christian, Markus, Peter, and everyone from the theory department who joined for lunch, the days at the office would not have been nearly as pleasant.

Many thanks go to my parents without whose support this work and the whole course of my studies would have been significantly harder.

Finally I want to thank Frederike for a lot of things. Her support helped me very much and I am sorry for all the rants about the dark sides of C++ that I brought home.

List of Figures

Bibliography

[1] H. Fritzsch, M. Gell-Mann, and H. Leutwyler. “Advantages of the Color Octet Gluon Picture”. In: *Phys. Lett.* 47B (1973), pp. 365–368. DOI: 10.1016/0370-2693(73)90625-4.

[2] R. P. Feynman and A. R. Hibbs. *Quantum mechanics and path integrals*. International series in pure and applied physics. New York, NY: McGraw-Hill, 1965. URL: https://cds.cern.ch/record/100771.

[3] M. Lüscher. “Construction of a selfadjoint, strictly positive transfer matrix for euclidean lattice gauge theories”. In: *Communications in Mathematical Physics* 54.3 (1977), pp. 283–292.

[4] K. G. Wilson. “Confinement of quarks”. In: *Phys. Rev. D* 10 (8 Oct. 1974), pp. 2445–2459. DOI: 10.1103/PhysRevD.10.2445. URL: https://link.aps.org/doi/10.1103/PhysRevD.10.2445.

[5] A. Shindler. “Twisted mass lattice QCD”. In: *Phys. Rept.* 461 (2008), pp. 37–110. DOI: 10.1016/j.physrep.2008.03.001. arXiv: 0707.4093 [hep-lat].

[6] H. Nielsen and M. Ninomiya. “A no-go theorem for regularizing chiral fermions”. In: *Physics Letters B* 105.2 (1981), pp. 219–223. ISSN: 0370-2693. DOI: 10.1016/0370-2693(81)91026-1.

[7] K. Jansen and C. Liu. “Implementation of Symanzik’s improvement program for simulations of dynamical Wilson fermions in lattice QCD”. In: *Comput. Phys. Commun.* 99 (1997), pp. 221–234. DOI: 10.1016/S0010-4655(96)00128-2. arXiv: hep-lat/9603008 [hep-lat].

[8] B. Sheikholeslami and R. Wohlert. “Improved continuum limit lattice action for QCD with wilson fermions”. In: *Nuclear Physics B* 259.4 (1985), pp. 572–596. ISSN: 0550-3213. DOI: 10.1016/0550-3213(85)90002-1.

[9] R. Frezzotti et al. “A Local formulation of lattice QCD without unphysical fermion zero modes”. In: *Nucl. Phys. Proc. Suppl.* 83 (2000), pp. 941–946. DOI: 10.1016/S0920-5632(00)91852-8. arXiv: hep-lat/9909003 [hep-lat].

[10] R. Frezzotti. “Twisted mass lattice QCD”. In: *Nucl. Phys. Proc. Suppl.* 140 (2005). [,134(2004)], pp. 134–140. DOI: 10.1016/j.nuclphysbps.2004.11.329. arXiv: hep-lat/0409138 [hep-lat].

[11] M. Creutz. “Grassmann integrals by machine”. In: *Nuclear Physics B-Proceedings Supplements* 73.1-3 (1999), pp. 819–821.

[12] M. Ueding. *su2-hmc Repository*. 2016. URL: https://github.com/martin-ueding/su2-hmc.

[13] C. Urbach. *su2 Repository*. URL: https://github.com/urbach/su2.

[14] M. Creutz. “Monte Carlo study of quantized SU(2) gauge theory”. In: *Phys. Rev. D* 21 (8 Apr. 1980), pp. 2308–2315. DOI: 10.1103/PhysRevD.21.2308.

[15] T. Lippert. “The Hybrid Monte Carlo algorithm for quantum chromodynamics”. In: *Lect. Notes Phys.* 508.122 (1998). DOI: 10.1007/BFb0106881. arXiv: hep-lat/9712019 [hep-lat].

[16] N. Metropolis et al. “Equation of state calculations by fast computing machines”. In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[17] M. Creutz and B. Freedman. “A Statistical Approach to Quantum Mechanics”. In: *Annals of Physics* 132.1981 (Nov. 1980), pp. 427–462. URL: http://www.itkp.uni-bonn.de/~urbach/bnd/cf.pdf.

[18] W. K. Hastings. “Monte Carlo sampling methods using Markov chains and their applications”. In: *Biometrika* 57.1 (1970), pp. 97–109.

[19] F. Fucito et al. “A Proposal for Monte Carlo Simulations of Fermionic Systems”. In: *Nucl. Phys.* B180 (1981). [,586(1980)], p. 369. DOI: 10.1016/0550-3213(81)90055-9.

[20] D. H. Weingarten and D. N. Petcher. “Monte Carlo Integration for Lattice Gauge Theories with Fermions”. In: *Phys. Lett.* 99B (1981), pp. 333–338. DOI: 10.1016/0370-2693(81)90112-X.

[21] M. Lüscher. “A New approach to the problem of dynamical quarks in numerical simulations of lattice QCD”. In: *Nucl. Phys.* B418 (1994), pp. 637–

648. DOI: 10.1016/0550-3213(94)90533-9. arXiv: hep-lat/9311007 [hep-lat].

[22] A. D. Kennedy, I. Horvath, and S. Sint. "A New exact method for dynamical fermion computations with nonlocal actions". In: *Nucl. Phys. Proc. Suppl.* 73 (1999), pp. 834–836. DOI: 10.1016/S0920-5632(99)85217-7. arXiv: hep-lat/9809092 [hep-lat].

[23] Z. Bai. *Krylov subspace projection methods*. 2009. URL: http://www.cs.ucdavis.edu/~bai/Winter09/krylov.pdf.

[24] S. Blackford. *An Introduction to Iterative Projection Methods*. Nov. 2000. URL: http://www.netlib.org/utk/people/JackDongarra/etemplates/node82.html.

[25] R. M. Aarts et al. *Remez Algorithm*. URL: http://mathworld.wolfram.com/RemezAlgorithm.html.

[26] J. Maddock et al. *The Remez Method*. 2006. URL: http://www.boost.org/doc/libs/1_46_1/libs/math/doc/sf_and_dist/html/math_toolkit/backgrounders/remez.html.

[27] Free Software Foundation. *The GNU Multiple Precision Arithmetic Library*. 2000. URL: https://gmplib.org/.

[28] M. A. Clark and A. D. Kennedy. "Accelerating dynamical fermion computations using the rational hybrid Monte Carlo (RHMC) algorithm with multiple pseudofermion fields". In: *Phys. Rev. Lett.* 98 (2007), p. 051601. DOI: 10.1103/PhysRevLett.98.051601. arXiv: hep-lat/0608015 [hep-lat].

[29] B. Jegerlehner. "Krylov space solvers for shifted linear systems". In: (1996). arXiv: hep-lat/9612014 [hep-lat].

[30] I. P. Omelyan, I. M. Mryglod, and R. Folk. "New optimized algorithms for molecular dynamics simulations". In: *Condensed Matter Physics* 5.3(31) (2002), pp. 369–390. URL: http://www.icmp.lviv.ua/journal/zbirnyk.31/001/art01.pdf.

[31] S. Duane et al. "Hybrid Monte Carlo". In: *Phys. Lett.* B195 (1987), pp. 216–222. DOI: 10.1016/0370-2693(87)91197-X.

[32] K. Bitar et al. "Hybrid Monte Carlo and Quantum Chromodynamics". In: *Nucl. Phys.* B313 (1989), pp. 377–392. DOI: 10.1016/0550-3213(89)90324-6.

[33] T. A. Degrand and P. Rossi. “Conditioning techniques for dynamical fermions”. In: *Computer Physics Communications* 60.2 (1990), pp. 211–214. ISSN: 0010-4655. DOI: 10.1016/0010-4655(90)90006-M.

[34] E. W. Weisstein. *LU Decomposition*. From MathWorld–A Wolfram Web Resource. URL: http://mathworld.wolfram.com/LUDecomposition.html.

[35] Hellen. *What is the name of this matrix decomposition?* accessed on 2017-07-31). URL: https://math.stackexchange.com/q/2377744.

[36] M. R. Hestenes and E. Stiefel. “Methods of conjugate gradients for solving linear systems”. In: *Journal of Research of the National Bureau of Standards* 49.6 (1952). DOI: 10.6028/jres.049.044.

[37] H. A. Van der Vorst. “Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems”. In: *SIAM Journal on scientific and Statistical Computing* 13.2 (1992), pp. 631–644.

[38] M. Hasenbusch. “Speeding up the HMC algorithm: Some new results”. In: *PoS* LAT2005 (2006), p. 116. arXiv: hep-lat/0509080 [hep-lat].

[39] *SciDAC, LHPC, UKQCD collaboration*: R. G. Edwards and B. Joó. “The Chroma software system for lattice QCD”. In: *Nucl. Phys. Proc. Suppl.* 140 (2005), p. 832. DOI: 10.1016/j.nuclphysbps.2004.11.254. arXiv: hep-lat/0409003 [hep-lat].

[40] D. H. Weingarten and J. C. Sexton. “Hamiltonian evolution for the hybrid Monte Carlo algorithm”. In: *Nucl. Phys. Proc. Suppl.* 26 (1992). [,613(1991)], pp. 613–616. DOI: 10.1016/0920-5632(92)90349-W.

[41] R. G. Edwards, B. Joó, and F. Winter. *Chroma Repository*. URL: https://github.com/JeffersonLab/chroma.

[42] R. G. Edwards, B. Joó, and T. Kurth. *QDP++ Repository*. URL: https://github.com/usqcd-software/qdpxx.

[43] F. Winter. “Accelerating QDP++/Chroma on GPUs”. In: *PoS* (2011). arXiv: 1111.5596 [hep-lat].

[44] F. Winter. “Gauge Field Generation on Large-Scale GPU-Enabled Systems”. In: *PoS* LATTICE2012 (2012), p. 185. arXiv: 1212.0785 [hep-lat].

[45] F. Winter. *QDP JIT Repository*. URL: https://github.com/fwinter/qdp-jit.

[46] J. Chen et al. *QMP Repository*. URL: https://github.com/usqcd-software/qmp.

[47] MPI Forum. *MPI: A Message-Passing Interface Standard*. URL: http://mpi-forum.org/.

[48] R. G. Edwards and B. Joó. *qmt Repository*. URL: https://github.com/usqcd-software/qmt.

[49] *OpenMP*. URL: http://www.openmp.org/.

[50] B. Joó et al. "Lattice qcd on intel® xeon phitm coprocessors". In: *International Supercomputing Conference*. Springer. 2013, pp. 40–54.

[51] B. Joó et al. *QPhiX Repository*. URL: https://github.com/JeffersonLab/qphix.

[52] R. G. Edwards et al. *C-LIME Repository*. URL: https://github.com/usqcd-software/c-lime.

[53] R. G. Edwards et al. *qio Repository*. URL: https://github.com/usqcd-software/qio.

[54] K. Jansen and C. Urbach. "tmLQCD: A Program suite to simulate Wilson Twisted mass Lattice QCD". In: *Comput. Phys. Commun.* 180 (2009), pp. 2717–2738. DOI: 10.1016/j.cpc.2009.05.016. arXiv: 0905.3331 [hep-lat].

[55] C. Urbach et al. *tmLQCD Repository*. URL: https://github.com/etmc/tmLQCD.

[56] *ETM collaboration*: A. Deuzeman et al. "Lemon: An MPI parallel I/O library for data encapsulation using LIME". In: *Computer Physics Communications* 183.6 (2012), pp. 1321–1335.

[57] *Budapest-Marseille-Wuppertal collaboration*: S. Dürr et al. "The ratio FK/Fpi in QCD". In: *Phys. Rev.* D81 (2010), p. 054507. DOI: 10.1103/PhysRevD.81.054507. arXiv: 1001.4692 [hep-lat].

[58] M. Lüscher and P. Weisz. "Efficient numerical techniques for perturbative lattice gauge theory computations". In: *Nuclear Physics B* 266.2 (1986), pp. 309–356.

[59] C. Morningstar and M. J. Peardon. "Analytic smearing of SU(3) link variables in lattice QCD". In: *Phys. Rev.* D69 (2004), p. 054501. DOI: 10.1103/PhysRevD.69.054501. arXiv: hep-lat/0311018 [hep-lat].

[60] R. Edwards, I. Horváth, and A. Kennedy. "Instabilities and non-reversibility of molecular dynamics trajectories". In: *Nuclear Physics B* 484.1 (1997), pp. 375–399. ISSN: 0550-3213. DOI: 10.1016/S0550-3213(96)00618-9. arXiv: hep-lat/9606004 [hep-lat].

[61] C. Liu, A. Jaster, and K. Jansen. "Liapunov exponents and the reversibility of molecular dynamics algorithms". In: *Nucl. Phys.* B524 (1998), pp. 603–617. DOI: 10.1016/S0550-3213(98)00174-6. arXiv: hep-lat/9708017 [hep-lat].

[62] C. Urbach. "Untersuchung der Reversibilitätsverletzung beim Hybrid-Monte-Carlo-Algorithmus". Diplomarbeit. Freie Universität Berlin, 2002. URL: https://www.itkp.uni-bonn.de/~urbach/diplom.pdf.

[63] *FLAG collaboration*: S. Aoki et al. "Review of lattice results concerning low-energy particle physics". In: *Eur. Phys. J.* C77.2 (2017), p. 112. DOI: 10.1140/epjc/s10052-016-4509-7. arXiv: 1607.00299 [hep-lat].

[64] S. Scherer. "Introduction to chiral perturbation theory". In: *Adv. Nucl. Phys.* 27 (2003), p. 277. arXiv: hep-ph/0210398 [hep-ph].

[65] M. Lüscher and S. Schaefer. "Lattice QCD with open boundary conditions and twisted-mass reweighting". In: *Comput. Phys. Commun.* 184 (2013), pp. 519–528. DOI: 10.1016/j.cpc.2012.10.003. arXiv: 1206.2809 [hep-lat].

[66] B. Kostrzewa. "Maximally Twisted Mass Lattice QCD at the Physical Pion Mass". PhD thesis. Humboldt-Universität zu Berlin, 2016. URL: https://edoc.hu-berlin.de/handle/18452/18385.

[67] Jülich Supercomputing Centre. "JUQUEEN: IBM Blue Gene/Q Supercomputer System at the Jülich Supercomputing Centre". In: *Journal of large-scale research facilities* 1.A1 (2015). DOI: 10.17815/jlsrf-1-18.

[68] P. A. Boyle. "The BAGEL assembler generation library". In: *Computer Physics Communications* 180.12 (2009), pp. 2739–2748.

[69] Jülich Supercomputing Centre. "JURECA: General-purpose supercomputer at Jülich Supercomputing Centre". In: *Journal of large-scale research facilities* 2.A62 (2016). DOI: 10.17815/jlsrf-2-121.

[70] P. Labus and M. Ueding. *Lattice QCD Stencils on Intel Xeon Phis*. Talk given at Cineca, Bologna, Italy on 2017-03-23. Mar. 2017.

[71] *ALPHA collaboration*: U. Wolff. "Monte Carlo errors with less errors". In: *Comput. Phys. Commun.* 156 (2004). [Erratum: Comput. Phys. Commun.176,383(2007)], pp. 143–153. DOI: 10.1016/S0010-4655(03)00467-3;10.1016/j.cpc.2006.12.001. arXiv: hep-lat/0306017 [hep-lat].

[72] C. Urbach, B. Kostrzewa, and S. Reker. *hadron Repository*. URL: https://github.com/etmc/hadron.

[73] K. Ottnad. "Properties of pseudoscalar flavor singlet mesons from lattice QCD". PhD thesis. Uni Bonn, Dec. 2013. URL: https://inspirehep.net/record/1351701/files/3506.pdf.

[74] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001–. URL: http://www.scipy.org.

[75] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org.

[76] M. Lüscher. "Properties and uses of the Wilson flow in lattice QCD". In: *JHEP* 08 (2010). [Erratum: JHEP03,092(2014)], p. 071. DOI: 10.1007/JHEP08(2010)071;10.1007/JHEP03(2014)092. arXiv: 1006.4518 [hep-lat].

[77] *Budapest-Marseille-Wuppertal collaboration*: S. Borsanyi et al. "High-precision scale setting in lattice QCD". In: *JHEP* 09 (2012), p. 010. DOI: 10.1007/JHEP09(2012)010. arXiv: 1203.4469 [hep-lat].

[78] *Hadron Spectrum collaboration*: M. Peardon et al. "A Novel quark-field creation operator construction for hadronic physics in lattice QCD". In: *Phys. Rev.* D80 (2009), p. 054506. DOI: 10.1103/PhysRevD.80.054506. arXiv: 0905.2160 [hep-lat].

[79] C. Morningstar et al. "Improved stochastic estimation of quark propagation with Laplacian Heaviside smearing in lattice QCD". In: *Phys. Rev.* D83

(2011), p. 114505. DOI: 10.1103/PhysRevD.83.114505. arXiv: 1104.3870 [hep-lat].

[80] A. Hasenfratz and F. Knechtli. "Flavor symmetry and the static potential with hypercubic blocking". In: *Phys. Rev.* D64 (2001), p. 034504. DOI: 10.1103/PhysRevD.64.034504. arXiv: hep-lat/0103029 [hep-lat].

[81] C. Helmes and C. Jost. *LapH_EigSys Repository*. URL: https://github.com/chelmes/LapH_EigSys.

[82] B. Knippschild and B. Kostrzewa. *peram_gen Repository*. URL: https://github.com/knippsch/peram_gen.

[83] B. Knippschild and M. Werner. *sLapH Contraction Repository*. URL: https://github.com/maowerner/sLapH-contractions.

[84] M. Ueding. *mu-correlators Repository*. 2015. URL: https://github.com/martin-ueding/mu-correlators.

[85] M. Lüscher. "Volume dependence of the energy spectrum in massive quantum field theories". In: *Communications in Mathematical Physics* 105.2 (1986), pp. 153–188.

[86] L. Liu et al. "Isospin-0 $\pi\pi$ s-wave scattering length from twisted mass lattice QCD". In: (2016). arXiv: 1612.02061 [hep-lat].

[87] B. Joó et al. "Lattice QCD on Intel® Xeon PhiTM Coprocessors". In: *Supercomputing: 28th International Supercomputing Conference, ISC 2013, Leipzig, Germany, June 16-20, 2013. Proceedings*. Berlin, Heidelberg: Springer, 2013, pp. 40–54. ISBN: 978-3-642-38750-0. DOI: 10.1007/978-3-642-38750-0_4. URL: http://www.opencirrus.intel-research.net/publications/QCD_ISC13_Full_Final.pdf.

[88] P. Labus. "Lattice Quantum Chromodynamics on Intel Xeon Phi based supercomputers". MA thesis. 2016.

[89] Intel. *Intel® Xeon® Processor E5-2680 v3*. URL: https://ark.intel.com/products/81908/Intel-Xeon-Processor-E5-2680-v3-30M-Cache-2_50-GHz.

[90] Intel. *Intel® Xeon Phi™ Processor 7250*. URL: https://ark.intel.com/products/94035/Intel-Xeon-Phi-Processor-7250-16GB-1_40-GHz-68-core.

[91] A. Duran. *Introduction to the Intel Xeon Phi 2nd generation architecture (codenamed Knights Landing)*. 2017. URL: http://www.lenovoevents.co.uk/pre/Introduction_to_the_second_generation_Intel_Xeon_Phi_architecture.pdf.

[92] Intel. *Intrinsics Guide*. URL: https://software.intel.com/sites/landingpage/IntrinsicsGuide/.

[93] M. Schröck, S. Simula, and A. Strelchenko. "Accelerating Twisted Mass LQCD with QPhiX". In: *PoS* LATTICE2015 (2016), p. 030. arXiv: 1510.08879 [hep-lat].

[94] M. Schröck. *QUDA and QPhiX interfaces for tmLQCD*. May 2015. URL: http://physik.uni-graz.at/~msk/talks/schroeck-etmc2015.pdf.

[95] Travis CI GmbH. *QPhiX on Travis CI*. URL: https://travis-ci.org/JeffersonLab/qphix.

[96] B. Joó. *Guide To Writing Chroma XML Files*. 2008. URL: http://usqcd.jlab.org/usqcd-docs/chroma/HackLatt08/Presentations/XMLFileConstructionGuide.pdf.

[97] D. E. Knuth, T. Larrabee, and P. M. Roberts. *Mathematical Writing*. 1987. URL: http://jmlr.csail.mit.edu/reviewing-papers/knuth_mathematical_writing.pdf.

Declaration

I hereby declare that the work presented here was formulated by myself and that no sources or tools other than those cited were used.

Bonn, 2017-08-28 Signature _____.