Observables relevant for the understanding of the structure of baryons were determined by means of Monte Carlo simulations of lattice Quantum Chromodynamics (QCD) using 2+1 dynamical quark flavours. Special emphasis was placed on how these observables change when flavour symmetry is broken in comparison to choosing equal masses for the two light and the strange quark. The first two moments of unpolarised, longitudinally, and transversely polarised parton distribution functions were calculated for the nucleon and hyperons.

Modern lattice QCD simulations require petaflop computing and beyond, a regime of computing power we just reach today. Heterogeneous multicore computing is getting increasingly important in high performance computing and allows for deploying multiple types of processing elements within a single workflow. In this work new design concepts were developed for an active library (QDP++) exploiting the compute power of a heterogeneous multicore processor (IBM PowerXCell 8i processor). It was possible to run a QDP++ based physics application (Chroma) on an IBM BladeCenter QS22.

Universitätsverlag Regensburg

ISBN 978-3-86845-082-8

9 783868 450828

gefördert von:

AΦ Alumni der physikalischen Fakultät der Universität Regensburg e.V.

UR Universität Regensburg

Dissertationsreihe Physik - Band 23

Frank Winter

Frank Winter

Investigation of Hadron Matter using Lattice QCD and Implementation of Lattice QCD Applications on Heterogeneous Multicore Acceleration Processors

Universitätsverlag Regensburg

23 Dissertationsreihe Physik

Frank Winter

Investigation of Hadron Matter using Lattice QCD and Implementation of Lattice QCD Applications on Heterogeneous Multicore Acceleration Processors

Investigation of Hadron Matter using Lattice QCD and Implementation of Lattice QCD Applications on Heterogeneous Multicore Acceleration Processors

Dissertation zur Erlangung des Doktorgrades der Naturwissenschaften (Dr. rer. nat.)
der Fakultät für Physik der Universität Regensburg
vorgelegt von

Frank Winter

aus Mainz

2011

Die Arbeit wurde von Prof. Dr. A. Schäfer angeleitet.
Das Promotionsgesuch wurde am 21.01.2011 eingereicht.

Prüfungsausschuss:	Vorsitzender:	Prof. Dr. R. Huber
	1. Gutachter:	Prof. Dr. A. Schäfer
	2. Gutachter:	Dr. D. Pleiter
	weiterer Prüfer:	Prof. Dr. T. Niehaus

Dissertationsreihe der Fakultät für Physik der Universität Regensburg, Band 23

Herausgegeben vom Präsidium des Alumnivereins der Physikalischen Fakultät:
Klaus Richter, Andreas Schäfer, Werner Wegscheider, Dieter Weiss

Frank Winter

Investigation of Hadron Matter using Lattice QCD and Implementation of Lattice QCD Applications on Heterogeneous Multicore Acceleration Processors

Universitätsverlag Regensburg

Bibliografische Informationen der Deutschen Bibliothek.
Die Deutsche Bibliothek verzeichnet diese Publikation
in der Deutschen Nationalbibliografie. Detailierte bibliografische Daten
sind im Internet über http://dnb.ddb.de abrufbar.

1. Auflage 2011
© 2011 Universitätsverlag, Regensburg
Leibnizstraße 13, 93055 Regensburg

Konzeption: Thomas Geiger
Umschlagentwurf: Franz Stadler, Designcooperative Nittenau eG
Layout: Frank Winter
Druck: Docupoint, Magdeburg
ISBN: 978-3-86845-082-8

Investigation of Hadron Matter using Lattice QCD

and

Implementation of Lattice QCD Applications on Heterogeneous Multicore Acceleration Processors

Dissertation

zur Erlangung des Doktorgrades

der Naturwissenschaften

(Dr. rer. nat.)

der Fakultät für Physik der Universität Regensburg

vorgelegt

von

Frank Winter

aus

Mainz

2011

Die Arbeit wurde von Prof. Dr. A. Schäfer angeleitet.
Das Promotionsgesuch wurde am 21.01.2011 eingereicht.

Prüfungsausschuss:	Vorsitzender:	Prof. Dr. R. Huber
	1. Gutachter:	Prof. Dr. A. Schäfer
	2. Gutachter:	Dr. D. Pleiter
	weiterer Prüfer:	Prof. Dr. T. Niehaus

Contents

Part I

Investigation of Hadron Matter using Lattice Methods

Chapter 1

Introduction

In 1968 deep inelastic electron scattering experiments at the Stanford Linear Accelerator Center (SLAC) discovered quarks as fundamental constituents in the nucleon and played an essential role in establishing QCD as the theory of the strong interactions. Subsequent efforts in understanding the structure of the proton in terms of quarks interacting through the exchange of gluons has opened up a variety of experimental and theoretical studies differing from those encountered in any other known systems.

The discovery of asymptotic freedom of non-Abelian gauge theories in 1973 led to the development of Quantum Chromodynamics (QCD) [1, 2], the theory of the strong force between quarks.

QCD is formulated in terms of quarks and gluons which is believed are the basic degrees of freedom that make up hadronic matter. It has been very successful in predicting phenomena involving large momentum transfers and at short distances. In this regime the coupling constant is small and the path integral approach leads to an intuitive tool to carry out perturbation theory. However, at the energy scale of the hadronic world, i.e. at scales ≈ 1 GeV where the coupling constant is of order unity perturbative methods fail. Thus, one cannot calculate the masses of mesons and baryons from QCD with perturbative methods even if one is given the coupling constant and the masses of the quarks.

In the low-energy regime Lattice QCD provides a non-perturbative tool for calculating the hadronic spectrum and the matrix elements of any operator within hadronic states from first principles. To this date, Lattice QCD represents the only known and working approach to quantitatively study the non-perturbative aspects of QCD from first principles.

In Lattice QCD the basic degrees of freedom, i.e. the fermionic and gluonic fields, are formulated on a discrete Euclidean spacetime lattice and the path integral is carried out numerically by Monte Carlo integration. It retains the fundamental character of QCD since no new parameters or field variables are introduced in this discretisation. The lattice spacing in a quantum field theory, serves as the ultraviolet regulator that must eventually be taken to zero keeping physical quantities, like the renormalised couplings or mass spectrum, fixed. The only tunable input parameters in these simulations are the gauge coupling constant and the bare masses of the quarks for each of the N_f quark flavours. Within the context of the standard model they have to be fixed in terms of an equal number of experimental quantities. Typically, the $N_f + 1$ parameters of the theory, i.e. QCD coupling and quark masses are matched to reproduce $N_f + 1$ hadron masses. Thereafter all predictions of Lattice QCD (after extrapolation to the continuum) have to match experimental data if QCD is the correct theory of the strong interaction, and in this sense Lattice QCD is a first principles approach. Lattice QCD is believed to provide reliable results from simulations at finite lattice spacing since the contributions from lattice artifacts are believed to be under control.

The formulation of QCD on a discrete spacetime lattice acts as a non-perturbative regularisation scheme. At finite values of the lattice spacing, which provides an ultraviolet cutoff, the infinities seen in perturbative QCD do not exist. Furthermore, renormalised physical quantities have a finite, well behaved continuum limit, i.e. taking the lattice spacing to zero.

Quantising QCD with the Feynman path-integral and formulating the theory on a Euclidean spacetime lattice permits to simulate the theory on computers using methods analogous to those used for Statistical Mechanics systems. These simulations allow for calculating correlation functions of hadronic operators and matrix elements of suitable operators between hadronic states in terms of the fundamental quark and gluon degrees of freedom.

In the past and still today, lattice simulations are subject to a number of limitations. The simulations are extremely expensive, reaching the need for petaflop computing and beyond, a regime of computing power just reached today. Costs of dynamical fermion simulations typically rise approximately with some power of the lattice extent and powers of the inverse lattice spacing and the inverse light quark mass. Therefore, for a long time the sea quarks were treated as infinitely heavy, i.e. the so-called quenched approximation, what was indeed a crude approximation given that the up and down quarks have masses of only a few MeV. Also due to the affinity to simulate even numbers of mass-degenerate dynamical quark flavours in the past only the lightest quark doublet, the up and down quarks, were taken into consideration. This saved the needs for computational resources to simulate heavier quarks like the strange quark. Also the masses of the quarks as used in the simulations have been unphysically large which reduces the overall computational cost.

Due to breakthroughs in algorithmic design and machine development, such as the use of improved actions which reduce lattice artefacts, today, these simulations are performed in increasingly physical conditions: Besides the up and down quarks, also the strange quark and lately also the charm degree of freedom are included in simulations, the quarks masses are chosen to be closer to their physical values, lattices sizes are set larger to reduce finite size effects, and the lattice spacing is taken to be small such that a better controlled continuum limit can be performed. The conditions are getting continuously more physical, so studying the low-energy limit of QCD should agree increasingly well with experiment and the predictions should get more trustworthy.

However, numerical simulations of Lattice QCD are based on a Monte Carlo integration of the Euclidean path integral, consequently, the measurements have statistical errors in addition to the systematic errors. Judgement of the quality of lattice calculations requires to understand the origin of these errors.

When investigating the structure of hadrons with lattice methods one has to consider the following requirements concerning the simulation parameters. First, the quark masses and the lattice extent should be sufficiently large so that not only a reasonable fraction of the hadron under investigation, but also other relevant degrees of freedom, in particular the virtual pions, which are essential for the hadron structure, fit into the lattice volume. Second, the lattice spacing should be small enough, i.e. the coupling large enough so that the internal structure of the hadron can be resolved and discretisation effects are kept under control.

Many fundamental properties of the hadron structure are encoded by the parton distribution functions (PDFs). They encode essential information about the distribution of momentum and spin of quarks and gluons inside hadrons and have in general an interpretation as probability densities as a function of the momentum fraction carried by the particular constituent.

PDFs are universal, i.e. process independent, non-perturbative objects. They are defined in terms of (forward or off-forward) hadron matrix elements of QCD quark and gluon operators. These matrix elements can, in turn, be written in the form of QCD path integrals, which makes them directly amenable to lattice methods.

Major facilities like CERN, DESY, JLAB, SLAC, and FNAL operate large scale hadron experiments for the generation of data on the quark structure of matter. At large energy scales lepton-hadron deeply inelastic scattering (DIS) processes give access in particular to the structure of the hadron. In electron-proton DIS, the electron probes the structure of the proton and provides access to the quark PDFs of the nucleon over a wide range of momentum fraction. At high energies the electron not only probes the valence quarks of the proton but also the QCD vacuum structure, and the "quark sea" consisting of all possible flavours of quarks and anti-quarks and a high density of gluons. However, since the experiments have limited kinematical coverage, the quark PDFs are only known in a limited range of momentum fraction and in order to retrieve information for a larger range of momentum fraction global PDF analyses are required.

It turns out that experimentally, a rather large number of different processes must be studied in order to access the structure of hadrons in great detail in terms of PDFs. Further challenges arise in studies of polarised distribution functions, which in general demand a preparation of polarised beams and targets.

When calculating PDFs with Lattice QCD many of the above mentioned difficulties are absent. Since Lattice simulations are carried out with the full Dirac structure of QCD all polarisations of the hadron are accessible – no separate simulations are necessary for different polarisations. Also the cost of Lattice QCD calculations is small compared to the overall cost of experiments.

However, the lattice approach to hadron structure has also some disadvantages. The full momentum-dependence of PDFs, cannot be studied directly on the lattice. These are defined via bi-local operators on the light-cone which cannot be studied with Lattice QCD which is formulated in Euclidean space. One can, however, relate moments of PDFs with lattice operators trough Mellin transformations. Only the lowest moments of the distribution functions corresponding to matrix elements of local operators, can so far be reliably computed. Calculation of higher moments suffer from increasingly bad statistics. Also operator mixing tends to be an issue with higher moments. So far, calculations have not been performed beyond the fourth moment of the distribution functions. Clearly, for a reconstruction of the momentum-dependence of the PDFs this is not sufficient.

Introducing a hypercubic lattice breaks the continuum space symmetries. The continuous symmetry group of the continuum theory, the Poincaré group, is reduced to a discrete group, the hypercubic group. As a consequence, even the local lattice vector current is not conserved and has to be renormalised. Lattice operators corresponding to higher moments also require renormalisation, and special care has to be taken to properly account for possible operator mixing, particularly with operators of lower dimensions.

This work investigates the baryon structure using gauge configurations generated with $N_f = 2+1$ dynamical flavours of $\mathcal{O}(a)$-improved Wilson fermions and the Symanzik improved gluon action. With the strange quark mass as an additional dynamical degree of freedom in this work's simulations the need is avoided for a partially quenched approximation when investigating the properties of particles containing a strange quark, e.g. the hyperons. In this work the quark masses are chosen by first finding the flavour SU(3) symmetric point where the light (up and down) quarks and the strange degree of freedom are mass-degenerate and then vary the individual quark masses while keeping the singlet quark mass constant. Simulations are performed on lattice volumes of $24^3 \times 48$ with lattice spacing, $a = 0.078(3)$fm.

This work focuses on the first ($n = 1$) and second ($n = 2$) moments of the nucleon and hyperon (Σ^+ and Ξ^0) unpolarised, longitudinally and transversely polarised PDFs. These include the baryon axial and tensor charges and quark momentum fractions.

The axial charge of the nucleon is important as it governs neutron β-decay and also provides a quantitative measure of spontaneous chiral symmetry breaking. It is also related to the first moment of the longitudinally polarised parton distribution functions, $g_A = \Delta u - \Delta d$.

Theoretical and experimental studies are carried out since many years. The Particle Data Group (PDG) world average is $g_A^N = 1.2694(28)$. Hence it is an important quantity to study on the lattice, and since it is relatively clean to calculate (zero momentum, isovector), it serves as a milestone for lattice simulations of nucleon structure.

While there has been much work on the (experimentally well-known) nucleon axial charge, there have only been a handful of lattice investigations of the axial charge of the other octet baryons, which are relatively poorly known experimentally. These constants are important since at leading order of SU(3) heavy baryon chiral perturbation theory (ChPT), these coupling constants are linear combinations of the universal coupling constants D and F, which enter the chiral expansion of every baryonic quantity.

Much of our knowledge about QCD and the structure of the nucleon has been derived from deep inelastic scattering experiments where cross sections are determined by its structure functions. Through the operator product expansion, the first moments of these structure functions are directly related to the momentum fractions carried by the quarks and gluons in the, e.g., nucleon, $\langle x \rangle_{q,g}$. While the quark momentum fractions of the nucleon and pion have received much attention for many years, there have to date been no investigations of the flavour SU(3) symmetry breaking effects of the quark momentum fractions of the hyperons. The obvious question that arises in this context is: "How is the momentum of the hyperon distributed amongst its light and strange quark constituents?"

The isospin symmetry between the proton and the neutron originates from the SU(2) symmetry between the up and down quarks, which are isospin doublets with isospin $I = 1/2$ and isospin three-components $I_3 = \pm 1/2$, respectively. This symmetry states that the up quark distribution in the proton is equal to the down quark distribution in the neutron. Since this work's simulations include varying the light and strange quark masses starting from the flavour SU(3) symmetric point in this work it was possible to predict the degree of isospin symmetry violation in the parton distribution functions of the nucleon by determining the quark momentum fractions of the octet baryons.

Chapter 2

Continuum QCD

QCD is a non-Abelian gauge field theory with SU(3) as the gauge group. The fundamental degrees of freedom are the quark and gluon fields. While the quark fields describe massive spin $\frac{1}{2}$ fermions which carry colour charge, the gluon fields describe the massless spin 1 gauge bosons mediating the colour force.

The dynamics of quarks and gluons are described by the QCD Lagrangian[1]

$$\mathcal{L} = -\frac{1}{4} F^a_{\mu\nu} F^{\mu\nu a} + \sum_{f=1}^{N_f} \overline{\psi}^f (i\gamma^\mu D_\mu - m_0^f)\psi^f \tag{2.1}$$

where ψ, $\overline{\psi}$ denote the Dirac 4-spinor quark fields with flavour index f, m_0^f are the bare quark masses, and N_f denotes the number of quark flavours. The adjoint vector $\overline{\psi}(x)$ is defined by $\overline{\psi}(x) = \psi^\dagger(x)\gamma_0$ where γ_0 is the γ-matrix related to the time direction. To reflect the Fermi-Dirac statistic the components of the fermion fields ψ, $\overline{\psi}$ are total anti-commuting Grassman variables.

For convenience we have dropped in Eq. (2.1) the quark and gluon field dependence on the position $x = (x_0, x_1, x_2, x_3)$, spin and colour indices and assume matrix-vector notation. The gauge covariant derivative is given as

$$D_\mu = \partial_\mu - ig_0 A_\mu \tag{2.2}$$

[1]It is not quite the most general Lagrangian that can be written when demanding a theory with a local SU(3) gauge invariance. A term $\propto {}^*\!FF$ can also be added.

	B	Q	I	I_3	S
u	1/3	2/3	1/2	+1/2	0
d	1/3	-1/3	1/2	-1/2	0
s	1/3	-1/3	0	0	-1

Table 2.1 – Quark quantum numbers of the light quarks with the baryon number B, the electric charge Q, the isospin I, and the strangeness S.

where g_0 is the gauge coupling constant, and the field strength tensor is defined in terms of the gluon fields as

$$F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu - ig_0[A_\mu, A_\nu]. \tag{2.3}$$

The field strength tensor and the gauge fields A_μ are elements of the Lie algebra of the SU(N_c) group where N_c indicates the number of colours[2]. Usually the gauge fields are expressed as

$$A_\mu(x) = \sum_{i=1}^{N_c^2-1} A_\mu^{(i)}(x) T_i \tag{2.4}$$

where the $A_\mu^{(i)}(x)$ are real numbers and the T_i are the generators of the SU(N_c) group. They are traceless, complex, and hermitian $N_c \times N_c$ matrices which obey

$$\mathrm{tr}[T_j T_k] = \frac{1}{2}\delta_{j,k} \tag{2.5}$$

$$[T_j, T_k] = if_{jkl} T_l \tag{2.6}$$

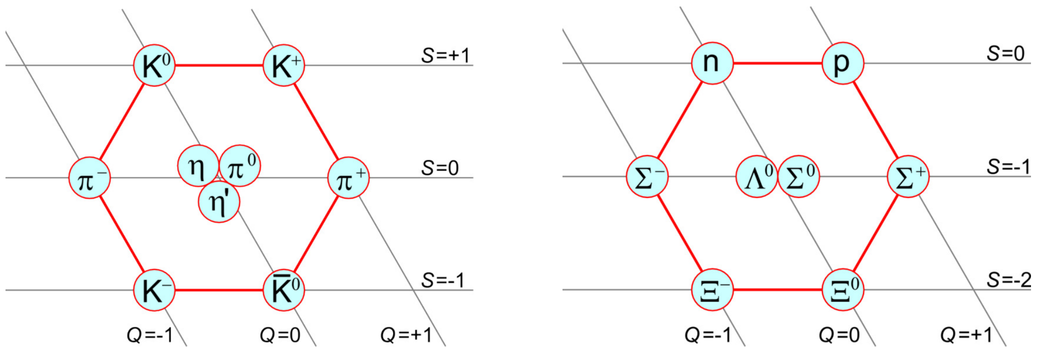
with the complete anti-symmetric coefficients f_{jkl}, the so-called *structure constants* of the group. The Lagrangian in Eq. (2.1) is invariant under local gauge transformations of the fermion and gauge fields

$$\begin{aligned} \psi(x) &\to \psi'(x) = \Omega(x)\psi(x) \\ \overline{\psi}(x) &\to \overline{\psi}'(x) = \overline{\psi}(x)\Omega(x)^\dagger \\ A_\mu(x) &\to A'_\mu(x) = \Omega(x)A_\mu(x)\Omega(x)^\dagger + i(\partial_\mu\Omega(x))\Omega(x)^\dagger \end{aligned} \tag{2.7}$$

with

$$\Omega(x) = \exp[-\theta_i(x) T_i] \in \mathrm{SU(N_c)} \tag{2.8}$$

[2]QCD can be formulated with any number of colours, but nature uses $N_c = 3$.



(a) Meson octet plus singlet **(b)** Baryon octet

Figure 2.1 – Fundamental representation of the SU(3) group for combining a quark and anti-quark pair (meson) and three quarks (baryon). Picture source: [3]

where the real numbers $\theta_i(x)$ can be chosen independently at every spacetime point.

The bare parameters of the theory are the gauge coupling constant g_0 and the bare quark masses m_0^f.

Quarks come in six flavours: These are the u, d, s, c, b, and t quarks. The masses of the u and d quarks are just a few MeV whereas the s quark has a mass of about 100 MeV - these are the light quarks. The heavy quarks c, b, and t all have masses over 1 GeV. The quantum numbers for the light quarks are given in Tab. 2.1.

At small energy scales, i.e. energies smaller than $\approx$ 1 GeV, quarks are confined into hadrons. Baryons represent bound states of three quarks and mesons consist of a quark and anti-quark pair.

Most matter that surrounds us (and also most matter created in accelerators) consists of the lightest quarks. The masses of the light quarks are small compared to the scale of the strong force. Consider for example the mass of the proton. Most of the contribution ($\approx$ 99%) comes from the kinetic and potential energy of the massless gluons and light quarks confined in the proton. The light quarks are within a good approximation degenerate in their mass. The QCD Lagrangian is (approximately) invariant under permutations of the quark flavour indices, i.e. it exhibits a flavour SU(3) symmetry.

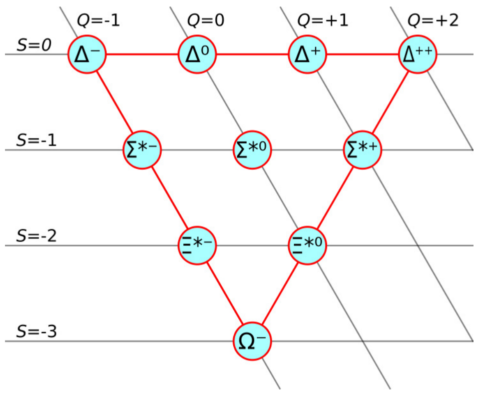


Figure 2.2 – Baryon decuplet. Picture source: [3]

The fundamental representation of the SU(3) group for combining a light quark and anti-quark pair decouples into an octet and singlet:

$$\mathbf{3} \otimes \overline{\mathbf{3}} = \mathbf{8} \oplus \mathbf{1} \tag{2.9}$$

Thus, the mesons are grouped into an octet and a singlet, see Fig. 2.1a. The mesons fall onto lines of constant charge, the diagonal lines $Q = -1, 0, +1$ and constant strangeness, the horizontal lines $S = -1, 0, +1$. The fundamental representation for baryons which consist of three light quarks decouples into ($J = 1/2$) octets and ($J = 3/2$) decuplets

$$\mathbf{3} \otimes \mathbf{3} \otimes \mathbf{3} = \mathbf{10} \oplus \mathbf{8} \oplus \mathbf{8} \oplus \mathbf{1}. \tag{2.10}$$

Fig. 2.1b (2.2) depicts the baryon octet (decuplet). Whereas the baryons of the octet fall onto lines of constant charge, the diagonal lines $Q = -1, 0, +1$ and constant strangeness, the horizontal lines $S = 0, -1, -2$ the baryons of the decuplet fall onto lines of constant charge, the diagonal lines $Q = -1, 0, +1, +2$ and constant strangeness, the horizontal lines $S = 0, -1, -2, -3$.

The non-Abelian term in the field strength tensor leads to gluon self-interactions. On the one hand the gluons are the carriers of the force and on the other hand they are colour charged particles, thus they interact with themselves.

The Lagrange description of QCD can be quantised by the path integral formulation. It is convenient to define matrix elements in terms of the path integral formulation

$$\langle 0|\Omega|0\rangle = \frac{1}{Z} \int \mathcal{D}[A, \psi, \overline{\psi}]\, \Omega\, e^{-i \int \mathrm{d}^4x\, \mathcal{L}} \tag{2.11}$$

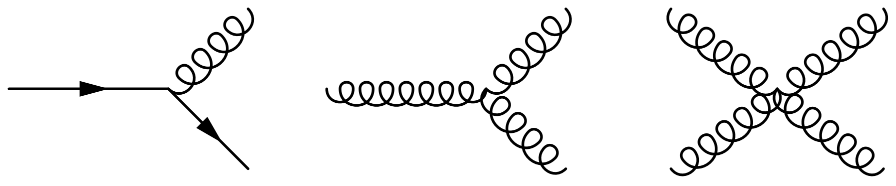

(a) Quark-gluon vertex (b) 3-gluon vertex (c) 4-gluon vertex

Figure 2.3 – The fundamental vertices of QCD. Quarks are represented by solid lines and gauge particle by wavy lines.

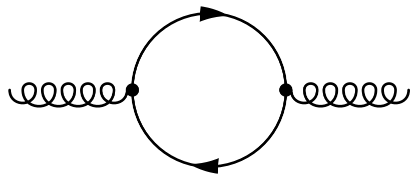


Figure 2.4 – Loop diagram leading to ultraviolet divergence.

with

$$Z = \int \mathcal{D}[A, \psi, \overline{\psi}] \; e^{-i \int \mathrm{d}^4 x \, \mathcal{L}} \tag{2.12}$$

where the operator Ω on the lhs acts on states of the Hilbert space and Ω on the rhs is a functional of the quark and gluon fields. The integral in the exponent is carried out over 4 dimensions of Minkowski spacetime.

The path integral can be carried out in the regime of a sufficiently small strong coupling constant

$$\alpha_s = \frac{g^2}{4\pi} \ll 1 \tag{2.13}$$

by perturbative methods using the Feynman rules. In this expansion, at leading order only tree-level diagrams composed out of the fundamental vertices of QCD are involved. Fig. 2.3 depicts the fundamental vertices.

In the perturbative expansion at orders beyond the tree-level, loop diagrams have to be taken into account. Loop diagrams represent integrations over infinite internal momenta and lead to ultraviolet (UV) divergences.

The UV divergences can be removed by regularisation, i.e. by introducing an ultraviolet momentum cut-off Λ_{UV}. The most convenient procedure is dimensional regularisation where the integrals are carried out with a dimension slightly smaller than 4, i.e. with dimension $d = 4 - 2\epsilon$. To keep the action dimensionless an additional mass parameter μ is introduced and the coupling constant is redefined to $g = \mu^{\epsilon} g_0$. Thus the coupling constant carries a mass dimension. It is convenient to choose μ of the order of the characteristic energy scale of the process of interest. The UV divergences are then absorbed by redefining the bare constants of the theory, i.e. the coupling constant g_0 and the bare quark masses m_0^f. This leads to the dependence of the gauge coupling constant g on the energy scale μ, the so-called running of the coupling constant $g(\mu)$.

The renormalisation group equations enable us to calculate the renormalised coupling constant to all orders of perturbation theory taking into account the most significant term at each order. The Gell-Mann–Law or so-called β function encodes the dependence of the coupling constant on the energy scale or distance:

$$\beta(\alpha_s) = -\frac{d\alpha_s(\mu)}{d \ln \mu^2} \tag{2.14}$$

For QED the β function is positive, thus the electric charge decreases at large distances due to screening of the QED vacuum. For non-Abelian gauge theories, like QCD, this is not the case. Gross et. al. showed that non-Abelian gauge theories are asymptotically free (if the number of flavours N_f is not too large) [1, 2]. For these theories the β function is negative, i.e. the coupling constant decreases for small distances or high energies. In the case of QCD with SU(3) as the colour gauge group, the β function reads

$$\beta(\alpha_s) = -\alpha_s \left(b_0 \frac{\alpha_s}{4\pi} + b_1 \left(\frac{\alpha_s}{4\pi}\right)^2 + ... \right) \tag{2.15}$$

with

$$b_0 = \frac{11 - 2\frac{N_f}{3}}{4\pi} > 0, \qquad \text{for } N_f \leq 16. \tag{2.16}$$

The β function of QCD is known to four-loop order [4]. Upon changing the scale, it can be shown that the coupling constant obeys the differential equation

$$\alpha_s(\mu^2) = \frac{1}{b_0 \ln \frac{\mu}{\Lambda_{\mathrm{QCD}}}}. \tag{2.17}$$

This equation can be integrated with Λ_{QCD} as the integration constant. The constant Λ_{QCD} sets the scale for all relevant dimensionful quantities.

Chapter 3

Phenomenology

A brief introduction to the phenomenology concerning this work is given while following in parts the notation of [5].

3.1 Pion Decay

The pion decay constant f_π represents an important constant of nature since it sets the scale of the chiral symmetry expansion and in addition determines the rate for the pion semi-leptonic decays.

The scattering matrix element of pion decay, see Fig. 3.1,

$$\pi^- \to \mu^- + \overline{\nu}_\mu \tag{3.1}$$

involves a leptonic and a hadronic matrix element

$$T_{fi} = (-ig_W)\cos\theta_c \overline{u}_\mu(\vec{k}_\mu)\gamma_\lambda(1-\gamma_5)u_\nu(\vec{k}_\nu) \times \frac{i}{m_W^2}\langle 0|J_h^\lambda(0)|\pi^+(\vec{p})\rangle \tag{3.2}$$

where we assume the mass of the W boson to be large compared to the momentum transfer. The relation to the Fermi constant is $G_F/\sqrt{2} = g_W^2/m_W^2$. We use the continuum normalisation of the states

$$\langle p'|p\rangle = (2\pi)^3 2E\delta^3(\vec{p}' - \vec{p}). \tag{3.3}$$

The hadronic part involves the weak current

$$J_h^\mu = \overline{u}\gamma^\mu(1-\gamma_5)d. \tag{3.4}$$

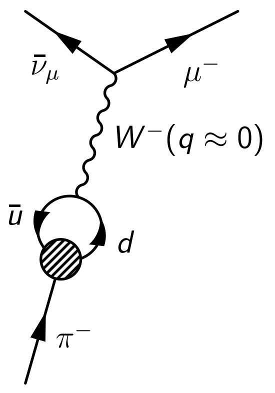


Figure 3.1 – Pion decay: The momentum transfer is taken to be small.

Since the pion is a pseudoscalar under parity the vector part vanishes and we find

$$\langle 0|\overline{u}\gamma^\mu\gamma_5 d|\pi^+(\mathbf{p})\rangle = -f_\pi p^\mu \tag{3.5}$$

with f_π the pion decay constant.

The experimental value of the pion decay constant is [6]

$$f_{\pi^\pm} = 130.7 \pm 0.1 \pm 0.36\text{MeV}. \tag{3.6}$$

The first error comes from $|V_{ud}|$ and the second from matching energy scales.

3.2 Beta Decay

In beta decay a neutron disintegrates into a proton, electron and electron antineutrino, see Fig. 3.2,

$$n \to p + e^- + \overline{\nu}_e. \tag{3.7}$$

This process is well described by the charged weak current model purely vector-axialvector interaction [7, 8, 9]. Studying the rate at which this process occurs and the angular correlations among the decay products provides insight into this basic semileptonic decay.

Neutron β decay is viewed as the conversion of a d quark into an u quark through the emission of a virtual W^- gauge boson:

$$d \to u + e^- + \overline{\nu}_e \tag{3.8}$$

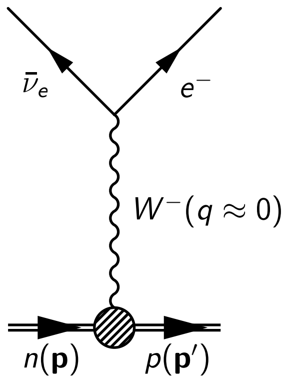


Figure 3.2 – Neutron beta-decay. A neutron disintegrates into a proton. The momentum transfer is taken to be small.

The mass difference between the neutron and proton is small, $(m_n - m_p)c^2 \approx 1.293$ MeV [10], particularly in comparison with their masses which are of order 1 GeV. We assume the momentum transfer $\mathbf{q} = \mathbf{p}' - \mathbf{p}$ to be small.

With the assumption that neutron decay is point-like, there is no change in total orbital angular momentum, and one can consider the selection rules for allowed transitions between initial and final states:

Fermi Decays: A decay in which the change in spin and isospin is zero ($|\Delta J| = 0$ and $|\Delta I| = 0$) and no parity change occurs is referred to as a Fermi decay. Fermi decays arise from vector currents.

Gamow–Teller Decays: If the electron and antineutrino spins are aligned with total spin 1, the proton couples to three possible spin states determined by Pauli spin matrices. These decays can change the spin and isospin by 0 or 1 ($|\Delta J| \in \{0, 1\}$ and $|\Delta I| \in \{0, 1\}$, but with $J_i = 0 \rightarrow J_f = 0$ forbidden, i for initial and f for final) and are known as Gamow–Teller decays. Gamow–Teller decays arise from axial-vector currents.

The beta decay is a mixed Fermi/Gamow-Teller decay.

The ratio at which Fermi and Gamow-Teller transitions occur is not precisely 3 : 1. There is a small deviation that is a measure of the difference in the coupling strengths of the two decays. The ratio is parametrised by a factor such that $3\lambda^2 : 1$, where λ is defined by

$$\lambda = \frac{g_A^N}{g_V^N} \tag{3.9}$$

with the nucleon axial charge g_A^N and vector charge g_V^N. Usually this ratio is determined under the assumption of conserved vector currents (CVC), which implies $g_V = 1$. The weak vector current is assumed to be conserved with a universal coupling constant in an analogy to the electro-magnetic vector current. This assumption is the CVC hypothesis. The value of g_A^N is determined experimentally to be 1.2695 ± 0.0029 [10].

The deviation of g_A^N from 1, the axial charge of a point-like particle, can be attributed, according to the Adler-Weisberger sum rule [11, 12], to the differences between the $\pi^+ N$ and $\pi^- N$ cross sections in pion-nucleon scattering. The value of g_A^N is a sensitive probe of the inner dynamics of the nucleon.

The matrix element for beta decay can be described in field theory, and with the assumption that only vector and axial-vector currents are involved, one can construct a matrix element describing neutron decay as a four-fermion interaction composed of hadronic and leptonic matrix elements

$$T_{fi} = (-ig_W)\cos_c \overline{u}_e(\vec{p}_\mu)\gamma_\mu(1-\gamma_5)u_{\nu_e}(\vec{p}_\nu) \times \frac{i}{m_W^2}\langle p(\vec{p})|J_h^\mu(0)|n(\vec{p})\rangle. \tag{3.10}$$

The leptonic portion of the matrix element can be calculated in a straightforward manner. The hadronic part reads

$$T_{fi} \propto \langle\, p(\mathbf{p})\,|\,J_h^\mu(0)\,|\,n(\mathbf{p})\,\rangle \tag{3.11}$$

with the weak current $J_h^\mu(0)$ as defined in Eq. (3.4). We determine g_A^N and g_V^N through

$$\langle p|\overline{u}\gamma_\mu d|n\rangle = g_V^N \overline{u}_p\gamma_\mu u_n \tag{3.12}$$

$$\langle p|\overline{u}\gamma_\mu\gamma_5 d|n\rangle = g_A^N \overline{u}_p\gamma_\mu\gamma_5 u_n \tag{3.13}$$

where $\overline{u}_B$, u_B are nucleon spinors with $B = n, p$. With current algebra we find, see proof in App. A of [13],

$$\langle p|\overline{u}\gamma_\mu\gamma_5 d|n\rangle = \langle p(\mathbf{p}), \mathbf{s}|A_\mu^{(u)} - A_\mu^{(d)}|p(\mathbf{p}), \mathbf{s}\rangle \tag{3.14}$$

$$= 2s_\mu g_A^N \tag{3.15}$$

where we have made explicit the spin dependence in Eq. (3.15) and introduced the spin vector s_μ which satisfies $s^2 = -m_p$ and with

$$A_\mu^{(q)} = \overline{q}\gamma_\mu\gamma_5 q. \tag{3.16}$$

Thus a measurement of g_A^N is equivalent to the measurement of the non-singlet proton matrix element.

Transition	a	b
$n \to p$	1	1
$\Sigma^- \to \Sigma^0$	$\sqrt{2}$	0
$\Xi^- \to \Xi^0$	-1	1

Table 3.1 – Clebsch-Gordan coefficients in nucleon and hyperon decays.

3.3 Baryon Matrix Element

According to Cabibbo Theory [14] the baryon matrix element for the decay

$$B_1 \to B_2 l \nu \tag{3.17}$$

at finite momentum can be written as

$$\langle B_2 | J^h_\mu | B_1 \rangle = C \overline{u}_{B_2}(p_2) \left[g_V(q^2)\gamma_\mu + g_A(q^2)\gamma_\mu\gamma_5 + ... \right] u_{B_1}(p_1) \tag{3.18}$$

where the ellipsis refers to terms proportional to the induced tensor and pseudoscalar form factors which are not relevant for our current discussion. If we assume exact flavour SU(3) symmetry, the vector and axial-vector form factors $g_V(q^2)$ and $g_A(q^2)$ can be written as

$$g_V(q^2) = aF_1(q^2) + bD_1(q^2) \tag{3.19}$$

$$g_A(q^2) = aF_2(q^2) + bD_2(q^2) \tag{3.20}$$

where the $F_i(q^2)$ and $D_i(q^2)$ with $i = 1, 2$ are different functions of q^2 for each of the two form factors. The constants a and b are generalised Clebsch-Gordan coefficients whose values are given in Tab. 3.1. This allows us to write the following relations between the nucleon and hyperon axial charges [15, 16]

$$g_A^N = F + D \tag{3.21}$$

$$g_A^\Sigma = \sqrt{2}F \tag{3.22}$$

$$g_A^\Xi = D - F \tag{3.23}$$

where we defined $F = F_2(0)$ and $D = D_2(0)$.

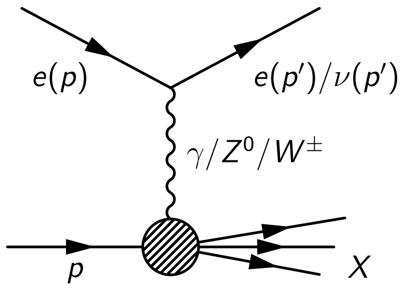


Figure 3.3 – Schematic diagram of a neutral current (exchange boson γ or Z^0) and charged current (exchange boson $W^\pm$) deep inelastic electron-proton scattering process. Momentum transfer is $q = p' - p$.

3.4 Deep Inelastic Scattering

In Deep Inelastic Scattering (DIS) processes the internal structure of hadrons (particularly the baryons, such as protons and neutrons) is probed with a leptonic, electrically charged scattering particle, i.e. electrons, muons – also DIS processes with neutrinos as the structure probing particle are carried out. It provided the first convincing evidence of the reality of quarks.

In scattering processes, see Fig. 3.3,

$$lN \to lX \tag{3.24}$$

$$\nu N \to \mu^- X \tag{3.25}$$

the momentum transfer from the incoming lepton l (usually an electron) is large enough to destroy the nucleon N (usually a proton). The final state X of this process can be anything. The hadronic part of the scattering matrix element is found to be

$$T_{fi} \propto \langle X | J^\mu(\mathbf{q}) | p \rangle \tag{3.26}$$

with the vector current

$$J^\mu = \frac{2}{3}\bar{u}\gamma^\mu u - \frac{1}{3}\bar{d}\gamma^\mu d + \left[-\frac{1}{3}\bar{s}\gamma^\mu s\right] + ... \tag{3.27}$$

The scattering process of an incoming electron with momentum p_e and an outgoing electron with momentum p'_e is described by the following kinematic variables:

- The momentum transfer

$$\Delta = q = p' - p, \qquad t = \Delta^2, \tag{3.28}$$

- the energy loss of the electron in laboratory frame

$$\nu = \frac{p \cdot q}{m_N} = E_e - E'_e, \tag{3.29}$$

- the inelasticity, i.e. the fractional energy loss of electron in laboratory frame

$$y = \frac{\nu}{E_e} = \frac{E_e - E'_e}{E_e}, \tag{3.30}$$

- the virtuality of the exchanged boson

$$Q^2 = -q^2, \tag{3.31}$$

- and the Bjorken scaling variable

$$x = \frac{Q^2}{2m_N \nu}. \tag{3.32}$$

The so-called forward limit refers to $p' = p$.

For virtualities

$$Q^2 > 1\text{GeV}^2 \tag{3.33}$$

the Compton wavelength of the exchanged boson is smaller than the dimension of the proton and the exchanged boson is able to probe the internal structure of the proton. In this energy regime if in addition the invariant mass of the hadronic final state

$$M_X^2 = (p+q)^2 \tag{3.34}$$

is much larger than the invariant mass of the proton, the process is called deep inelastic scattering (DIS).

In case of a single boson exchange, the double-differential cross section for DIS electron-proton scattering reads

$$\left.\frac{d^2\sigma}{d\Omega dE'_e}\right|_{N\,\text{lab frame}} = \frac{\alpha^2}{m_N Q^4} \frac{E'_e}{E_e} L_{\mu\nu} W^{\mu\nu} \tag{3.35}$$

with the leptonic tensor

$$L_{\mu\nu} = k'_\mu k_\nu + k'_\nu k_\mu - g_{\mu\nu} k' \cdot k + i\epsilon_{\mu\nu\rho\sigma} s_e^\rho q^\sigma \tag{3.36}$$

where k and k' are the four-momenta of the incoming and the scattered electron, and $\epsilon_{\mu\nu\rho\sigma}$ is the completely anti-symmetric tensor and $g_{\mu\nu}$ is the metric tensor in Minkowski space and where we have neglected terms of order $\mathcal{O}(m_e^2)$.

In case of summing over the final states X, i.e. considering inclusive DIS processes, the hadronic tensor is given by

$$W^{\mu\nu} = \frac{1}{4\pi}\int \mathrm{d}^4x\, e^{-iq\cdot x}\langle p|\left[J^\mu(x), J^\nu(0)\right]|p\rangle \tag{3.37}$$

$$= W_S^{\mu\nu} + iW_A^{\mu\nu} \tag{3.38}$$

where in the second line we have split the tensor into a symmetric and anti-symmetric piece. With current conservation and using parity and time reversal invariance, the general Lorentz decomposition of the symmetric piece of the tensor is given by

$$\begin{aligned} W_S^{\mu\nu} = &\left(-g^{\mu\nu} + \frac{q^\mu q^\nu}{q^2}\right) F_1(x, Q^2) + \\ &\frac{1}{m_N\nu}\left(p_\mu - \frac{p\cdot q}{q^2}q_\mu\right)\left(p_\nu - \frac{p\cdot q}{q^2}q_\nu\right) F_2(x, Q^2) \end{aligned} \tag{3.39}$$

and the anti-symmetric piece is given by

$$\begin{aligned} W_A^{\mu\nu} = &\frac{1}{m_N\nu}\epsilon^{\mu\nu\rho\sigma} q_\rho s_\sigma g_1(x, Q^2) + \\ &\frac{1}{m_N\nu}\epsilon^{\mu\nu\rho\sigma} q_\rho \left(s_\sigma - \frac{q\cdot s}{m_N\nu}p_\nu\right) g_2(x, Q^2) \end{aligned} \tag{3.40}$$

with the unpolarised structure functions F_1 and F_2 and the polarised structure functions g_1 and g_2 depending on the Bjorken scaling variable x and the virtuality Q^2. The quark-parton model makes approximate predictions for the structure functions.

3.5 The Quark-Parton Model

A simple physical picture of DIS processes is provided by the quark-parton model where at high energies the nucleon can be considered as a collection of free on-shell particles, i.e. partons, each carrying a fraction ξ of the nucleon momentum [17, 18, 19, 20]. In this picture $q_\sigma(\xi)$ and $\overline{q}_\sigma(\xi)$ are the parton distributions and σ the possible spin projections. Thus the momentum of a parton is given by

$$p_\xi = \xi p. \tag{3.41}$$

To avoid a variable invariant mass we work in the infinite momentum frame. We express the structure functions as

$$F_1^{(q)} = \frac{e_q^2}{2}\delta(\xi - x) \tag{3.42}$$

$$F_2^{(q)} = e_q^2 \xi \delta(\xi - x) \tag{3.43}$$

$$g_1^{(q)} = \frac{e_q^2}{2}\sigma_N \sigma_q \delta(\xi - x) \tag{3.44}$$

$$g_2^{(q)} = 0 \tag{3.45}$$

with e_q the normalised charge of the parton. Multiplying our results with $q_\sigma(\xi)$ and integrating over ξ gives

$$F_1(x) = \sum_q \frac{e_q^2}{2}\left(q(x) + \overline{q}(x)\right) \tag{3.46}$$

$$F_2(x) = \sum_q e_q^2 x \left(q(x) + \overline{q}(x)\right) \tag{3.47}$$

$$g_1(x) = \sum_q \frac{e_q^2}{2}\left(\Delta q(x) + \Delta \overline{q}(x)\right) \tag{3.48}$$

$$g_2(x) = 0 \tag{3.49}$$

with

$$q(x) = q_\uparrow(x) + q_\downarrow(x) \tag{3.50}$$

for the unpolarised case and

$$\Delta q(x) = q_\uparrow(x) - q_\downarrow(x) \tag{3.51}$$

for the polarised case. Thus, in the parton model the structure functions depend only on x. This is the so-called Bjorken-scaling and is direct evidence of a substructure of the nucleon. Radiative QCD corrections allow also for a dependence on Q^2. These scaling violations are a direct test of QCD.

3.6 Generalised Parton Distributions

The parton densities one can extract from DIS processes encode the distribution of longitudinal momentum and polarisation carried by quarks, antiquarks and gluons within a fast

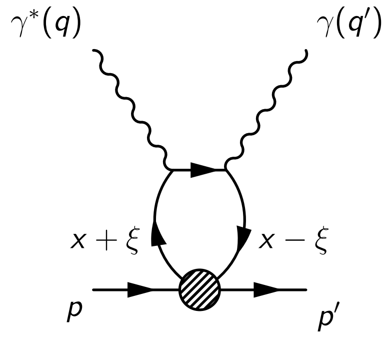


Figure 3.4 – Handbag diagram. ξ denotes the longitudinal momentum transfer.

moving hadron. They have had a high impact on our physical picture of hadron structure. Important pieces of information are missed out in these quantities, in particular how partons are distributed in the plane transverse to the direction in which the hadron is moving, or how important their orbital angular momentum is in making up the total spin of a nucleon. It has become clear that appropriate exclusive scattering processes may provide such information, encoded in generalised parton distributions (GPDs) [21, 22, 23, 24]. Reviews about GPDs are found in the literature [25, 26, 27].

The simple factorisation of dynamics into short- and long-distance parts is not only valid for the forward Compton amplitude, but also for the more general case where there is a finite momentum transfer to the target, provided at least one of the photon virtualities is large. A particular case is where the final photon is on-shell, so that it can appear in a physical state. To be more precise, one has to take the limit of large initial photon virtuality Q^2, and the invariant momentum transfer $t = (p' - p)^2$ remaining fixed. One then speaks of deeply virtual Compton scattering (DVCS), as shown in Fig. 3.4, which can be accessed in the exclusive process $ep \to e\gamma p$. The long-distance part, represented by the lower blob, is called a generalised parton distribution.

The production of a real photon requires a finite transfer of longitudinal momentum, where "longitudinal" refers to the direction of the initial proton momentum in a frame where both p and p' move fast, e.g. the centre of momentum frame of the $\gamma^* p$ collision. Proton and parton momenta now are no longer the same in the initial and final states. Therefore a GPD no longer represents a squared amplitude (and thus a probability), but rather the interference between amplitudes describing different quantum fluctuations of a nucleon.

3.7 Operator Product Expansion

For renormalisable quantum field theories the Operator Product Expansion (OPE) can be carried out to all orders with perturbation theory. The coefficient functions appearing in the OPE gain perturbative corrections which are constrained by the renormalisation group Callan-Symanzik equations [28, 29, 30, 31, 32].

In order to obtain predictions for the structure functions in terms of parton distribution functions (PDF) to leading order in the strong coupling, we apply the OPE method.

In general for a set of operators $O_i(x)$ the OPE has the form [33, 34]

$$\lim_{x\to 0} O_i(x)O_j(0) = \sum_k E_{ijk} O_{ijk}(0) \tag{3.52}$$

where E_{ijk} are the Wilson coefficients. We define the twist t of an operator as its dimension minus spin. By dimensional analysis it becomes clear that the rhs of Eq. (3.52) is an expansion in inverse powers of Q^2, with expansion powers being given by $t-2$. Listing all operators with a certain twist gives terms of $\mathcal{O}((Q^2)^{-(t-2)})$. The leading order terms are of twist $t = 2$ and are given from the symmetrised, traceless parts of the quark bilinear operators

$$O_q^{\mu_1\cdots\mu_n} = i^{n-1}\overline{q}\gamma^{\mu_1}\overleftrightarrow{D}^{\mu_2}\cdots\overleftrightarrow{D}^{\mu_n}q \tag{3.53}$$

$$O_{5q}^{\mu_1\cdots\mu_n} = i^{n-1}\overline{q}\gamma^{\mu_1}\gamma_5\overleftrightarrow{D}^{\mu_2}\cdots\overleftrightarrow{D}^{\mu_n}q \tag{3.54}$$

where $\overleftrightarrow{D} = \frac{1}{2}(\overrightarrow{D} - \overleftarrow{D})$ and the symmetrised, traceless part of an expression or operator SO is defined as

$$SO^{\mu_1\cdots\mu_n} = O^{\mu_1\cdots\mu_n} - \mathrm{tr} \tag{3.55}$$

where the trace terms are such that

$$\eta_{\mu_i\mu_j}O^{\mu_1\cdots\mu_i\cdots\mu_j\cdots\mu_n} = 0. \tag{3.56}$$

The OPE as carried out above is valid for forward γN Compton scattering within a small region $\frac{1}{x} \approx \frac{1}{Q} \to 0$. This is not the physical region for DIS processes. However, analyticity (in the $\frac{1}{x}$ plane) and dispersion relations connect to the discontinuity region. Finally the optical theorem relates the forward Compton scattering amplitude structure functions to the DIS structure functions.

Using the optical theorem to relate the inclusive $\gamma^* p$ cross section to the imaginary part of the forward Compton amplitude $\gamma^* p \to \gamma^* p$.

We arrive for the unpolarised nucleon structure functions at

$$2\int_0^1 dx x^{n-1} F_1(x, Q^2) = \sum_f E^{(f)}_{F_1,n} v^{(f)}_n(\mu) + \mathcal{O}(1/Q^2) \tag{3.57}$$

$$\int_0^1 dx x^{n-2} F_2(x, Q^2) = \sum_f E^{(f)}_{F_2,n} v^{(f)}_n(\mu) + \mathcal{O}(1/Q^2) \tag{3.58}$$

where n is even and starts at 2, and f are the quark flavours and v_n comes from the nucleon matrix element

$$\langle \mathbf{p}, \mathbf{s}| O^{\{\mu_1 \cdots \mu_n\}} - \mathrm{tr} |\mathbf{p}, \mathbf{s}\rangle = v^{(f)}_n \mathcal{S} \overline{u}(\mathbf{p}, \mathbf{s}) \gamma^{\mu_1} p^{\mu_2} \cdots p^{\mu_n} u(\mathbf{p}, \mathbf{s}) \tag{3.59}$$

$$= 2 v^{(f)}_n [p^{\mu_2} \cdots p^{\mu_n} - \mathrm{tr}]. \tag{3.60}$$

The moments in Eqs. (3.57) and (3.58) have parton interpretation, being powers of the fraction of the nucleon momentum carried by the parton

$$v^{(q)}_n(\mu) = \langle x^{n-1}\rangle^{(q)}(\mu) \tag{3.61}$$

$$= \int_0^1 dx x^{n-1} [q(x, \mu) + (-1)^n \overline{q}(x, \mu)] \tag{3.62}$$

in some scheme $\mathcal{S}$ and at scale μ.

3.8 Moments of Parton Distribution Functions

We give a brief introduction to the calculation of moments of parton distribution functions (PDFs) on the lattice. For more detailed discussions see [35, 36, 37, 38, 39].

In DIS processes one can measure the quark light-cone distributions in the nucleon. These distributions characterise the key bound state properties of the nucleon. At the leading-twist level, there are three types of quark distribution in the nucleon:

- Quark density distribution $q(x)$
- Quark helicity distribution $\Delta q(x)$
- Quark transversity distribution $\delta q(x)$

In QCD, all these parton distributions can be written as the matrix elements of bi-local operators. For instance, the quark light-cone distribution operator is

$$\mathcal{O}(x) = \int \frac{d\lambda}{4\pi} e^{i\lambda x} \overline{\psi}(\frac{-\lambda n}{2}) n \mathcal{P} e^{-ig\int_{-\lambda/2}^{\lambda} d\alpha n\cdot A(\alpha n)} \psi(\frac{\lambda n}{2}), \tag{3.63}$$

where n is a unit vector along the light-cone and $\lambda = p^+x^-$. Expanding $\mathcal{O}(x)$ in local operators via the OPE generates the tower of twist-two operators

$$O_q^{\mu_1\cdots\mu_n} = \overline{q}\gamma^{\{\mu_1} i\overleftrightarrow{D}^{\mu 2} \cdots i\overleftrightarrow{D}^{\mu_n\}} q \tag{3.64}$$

whose matrix elements can be calculated in Lattice QCD. For the quark helicity distribution Δq and the transversity distribution δq the towers of twist-two operators read

$$O_{5q}^{\mu_1\cdots\mu_n} = \overline{q}\gamma_5\gamma^{\{\mu_1} i\overleftrightarrow{D}^{\mu 2} \cdots i\overleftrightarrow{D}^{\mu_n\}} q \tag{3.65}$$

$$O_{Tq}^{\sigma\mu_1\cdots\mu_n} = \overline{q}\gamma_5\sigma^{\{\mu_1} i\overleftrightarrow{D}^{\mu 2} \cdots i\overleftrightarrow{D}^{\mu_n\}} q. \tag{3.66}$$

The quark density distribution $q(x)$ specifying the probability of finding a quark carrying a fraction x of the nucleon's momentum in the light cone frame is measured by the diagonal nucleon matrix element

$$\langle P|\mathcal{O}(x)|P\rangle = q(x) \tag{3.67}$$

and the $(n-1)^{\text{th}}$ moment of the quark distributions are specified by the diagonal matrix elements

$$\langle P|O_q^{\mu_1\cdots\mu_n}|P\rangle \propto \int dx x^{n-1} q(x) \tag{3.68}$$

$$\langle P|O_{5q}^{\mu_1\cdots\mu_n}|P\rangle \propto \int dx x^{n-1} \Delta q(x) \tag{3.69}$$

$$\langle P|O_{Tq}^{\sigma\mu_1\cdots\mu_n}|P\rangle \propto \int dx x^{n-1} \delta q(x). \tag{3.70}$$

3.9 Euclidean Operators

Minkowski $\mathcal{M}$ space has the signature (1,-1,-1,-1). Minkowski and Euclidean components of a 4-vector are related by

$$\psi_4 = i\psi^{(\mathcal{M})0} = i\psi_0^{(\mathcal{M})} \tag{3.71}$$

$$\psi_i = \psi^{(\mathcal{M})i} = -\psi_i^{(\mathcal{M})}. \tag{3.72}$$

γ	1	γ_5	γ_4	γ_i	$\gamma_4\gamma_5$	$\gamma_i\gamma_5$
η_γ	+1	-1	+1	+i	-1	-i

Table 3.2 – Shown are the values of the coefficient η for the possible combinations of γ-matrices.

Covariant derivatives are defined in Minkowski space as

$$D^{(\mathcal{M})\mu} = \partial^{(\mathcal{M})\mu} - igA^{(\mathcal{M})\mu} \tag{3.73}$$

and are related to their Euclidean counterparts

$$D_\mu = \frac{\partial}{\partial x_\mu} + igA_\mu \tag{3.74}$$

by

$$D_4 = -iD^{(\mathcal{M})0} \tag{3.75}$$

$$D_i = -D^{(\mathcal{M})i}. \tag{3.76}$$

The general operator in Minkowski $\mathcal{M}$ space is defined as

$$O_\gamma^{(\mathcal{M})\mu_1\cdots\mu_n} = i^n \overline{\psi}\gamma^{(\mathcal{M})} \overleftrightarrow{D}^{(\mathcal{M})\mu_1} \ldots \overleftrightarrow{D}^{(\mathcal{M})\mu_n}\psi \tag{3.77}$$

and is related to its Euclidean counterpart

$$O^\gamma_{\mu_1\cdots\mu_n} = \overline{\psi}\gamma \overleftrightarrow{D}^{\mu_1} \ldots \overleftrightarrow{D}^{\mu_n}\psi \tag{3.78}$$

by

$$O_\gamma^{(\mathcal{M})\mu_1\cdots\mu_n} = \eta_\gamma (-1)^{n_4} (-i)^{n_{123}} O^\gamma_{\mu_1\cdots\mu_n} \tag{3.79}$$

where n_4 is the number of time-like indices and n_{123} is the number of spatial indices and the value of the coefficient η is shown in Tab. 3.2.

3.10 Sum Rules

The momentum fraction $\langle x \rangle$ obeys the sum rule

$$1 = \sum_q \langle x \rangle_{q,\mu^2} + \langle x \rangle_{g,\mu^2} \tag{3.80}$$

where the sum runs over all relevant quark flavours and $\langle x\rangle_g$ is the fraction of momentum coming from gluons. In the large μ limit perturbative calculations yield

$$\lim_{\mu\to\infty}\langle x\rangle_{q,\mu^2} = \frac{3}{16+3N_f} \tag{3.81}$$
$$\lim_{\mu\to\infty}\langle x\rangle_{g,\mu^2} = \frac{16}{16+3N_f}. \tag{3.82}$$

For $N_f = 4$ these limits are found to be $\lim_{\mu\to\infty}\langle x\rangle_{q,\mu^2} = 3/28 \approx 10\%$ and $\lim_{\mu\to\infty}\langle x\rangle_{g,\mu^2} = 4/7 \approx 60\%$. Thus the gluons play a substantial role.

The nucleon spin is exactly 1/2 due to rotational symmetry. It obeys the sum rule [40]

$$\frac{1}{2} = \frac{1}{2}\sum_q \langle 1\rangle_{\Delta q,\mu^2} + \sum_q L_{q,\mu^2} + J_{g,\mu^2} \tag{3.83}$$

which relates the nucleon spin to the contributions from quark helicity $\langle 1\rangle_{\Delta q}$ and orbital angular momentum L_q and a net contribution from the gluons J_g.

European Muon Collaboration (EMC) experiments revealed that very little of a proton's spin is carried by its quarks. This was a very curious and unexpected experimental result and led to the "proton spin crisis" [41].

The asymptotic evolution of the total quark contribution $J_q = \frac{1}{2}\sum_q\langle 1\rangle_{\Delta q} + \sum_q L_q$ and the total gluon contribution J_g is given by

$$\lim_{\mu\to\infty} J_{q,\mu^2} = \frac{1}{2}\frac{3N_f}{16+3N_f} \tag{3.84}$$
$$\lim_{\mu\to\infty} J_{g,\mu^2} = \frac{16}{16+3N_f}. \tag{3.85}$$

Where again for $N_f = 4$ these limits were found to be $\lim_{\mu\to\infty} J_q = 2/7 \approx 0.60 \times 1/2$ and $\lim_{\mu\to\infty} J_g = 3/14 \approx 0.40 \times 1/2$. Thus again the gluons are playing a substantial role.

Chapter 4

QCD on the Lattice

This brief introduction follows the notation of [42].

4.1 Continuum Action

In the path integral description of QCD the continuum action

$$S_{\mathcal{M}}^{\mathrm{QCD}}[\psi,\overline{\psi},A] = -\frac{1}{4}\int \mathrm{d}^4x\, F^{\mu\nu}F_{\mu\nu} + \sum_{f=1}^{N_f}\int \mathrm{d}^4x\, \overline{\psi}^{(f)}\big(i\gamma^\mu D_\mu - m_0^{(f)}\big)\psi^{(f)} \tag{4.1}$$

$$= S_G^{\mathrm{QCD}} + S_F^{\mathrm{QCD}} \tag{4.2}$$

is divided into the gluonic term S_G^{QCD} containing only the gluon fields and a fermionic part S_F^{QCD} containing the quark and the gluon fields.

As QCD is a relativistic theory it is most conveniently formulated in Minkowski space. However, for a numerical treatment of the path integral it is advantageous to perform the Wick rotation to Euclidean space, where $x^0 \to ix_4$ where x_4 is the time component. This replaces the highly oscillating term $\exp[iS_{\mathcal{M}}^{\mathrm{QCD}}]$ in the path integral by an exponentially damping term $\exp[-S_E^{\mathrm{QCD}}]$, i.e. the Boltzmann factor. From now on we work in Euclidean space and drop the subscript E in the action S_E^{QCD}.

4.2 Introduction of the Lattice

We consider QCD in a four-dimensional (4D) hypercubic volume $V = L^3 \times T$ with the spatial extent L and the temporal extent T. In order to evaluate the path integral numerically we introduce a hypercubic 4D lattice $x = (x_1, x_2, x_3, x_4)$ within this volume. Even though the discretisation in the temporal dimension originates conceptually completely different than for the spatial dimension, i.e. it comes from introducing finite time evolution steps, we further assume the same lattice spacing a between the lattice points in all four dimensions. We label the lattice points with four-dimensional vectors

$$\Lambda = \{n = (n_1, n_2, n_3, n_4) | n_1, n_2, n_3 = 0, 1, \dots, N-1;\ n_4 = 0, 1, \dots, N_T - 1\} \tag{4.3}$$

with $N = L/a$ and $N_T = T/a$. Thus the total number of lattice sites is $|\Lambda| = N^3 N_T$. This allows us to reach each spacetime point in the lattice by writing

$$x = an. \tag{4.4}$$

We replace the fermionic fields of the continuum by fermionic fields located at the lattice points

$$\psi(x) \to \psi(n) \tag{4.5}$$

$$\overline{\psi}(x) \to \overline{\psi}(n) \tag{4.6}$$

with $n \in \Lambda$. We stress that our notation for the continuum and lattice version only differ in the argument of the fermionic fields.

4.3 Wilson Gauge Action

To formulate the gluonic fields on the lattice we first define the parallel transporter in the continuum

$$U(y_1, y_2) = \mathcal{P} \exp \left[i g_0 \int_{y_1}^{y_2} \mathrm{d}x_\mu A_\mu(x) \right] \tag{4.7}$$

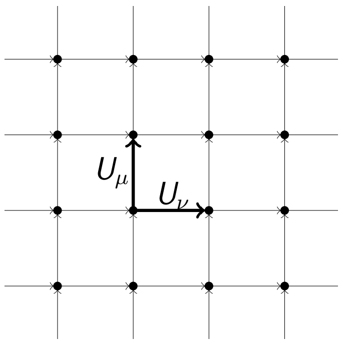


Figure 4.1 – The hypercubic lattice: The links connecting two neighbouring lattice sites are assigned the parallel transporters U_μ and U_ν.

with the path ordering operator $\mathcal{P}$. We define the gauge fields on the lattice with the help of the parallel transporter

$$U_\mu(n) = U(an, an + a\hat{\mu}) \tag{4.8}$$

$$= \exp[iaA_\mu(n)] \tag{4.9}$$

with $\mu = 1, .., 4$ and $\hat{\mu}$ the unit vector pointing in direction μ. The lattice version of the gauge fields are no longer elements of the Lie algebra of the group $\mathrm{SU}(N_c)$ but are elements of the group

$$U_\mu(n) = \exp\left(i \sum_{i=1}^{N_c^2-1} A_\mu^{(i)}(n) T_i\right) \tag{4.10}$$

and are located at the links connecting two neighbouring lattice points, see Fig. 4.1.

We further define the parallel transporter in the negative direction

$$U_{-\mu}(n) = U_\mu(n - \hat{\mu})^\dagger. \tag{4.11}$$

Instead of the term parallel transporter in the following we will call them shortly gauge fields.

As in the continuum the lattice version of the theory must be invariant under local gauge transformations. On the lattice we implement the gauge transformation by choosing elements of the special unitary group in N_c dimension for each lattice site independently

$$\Omega(n) \in \mathrm{SU}(N_c). \tag{4.12}$$

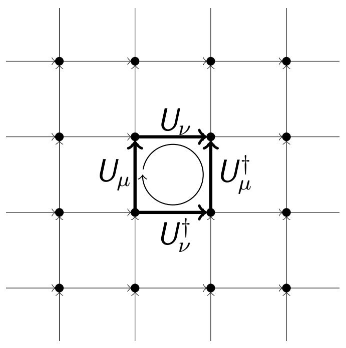


Figure 4.2 – The smallest closed loop on the hyper-cubic lattice is the plaquette which is traversed in the picture clock-wise: $U_{\mu\nu}(n) = U_\mu(n)U_\nu(n+\hat{\mu})U_\mu(n+\hat{\nu})^\dagger U_\nu(n)^\dagger$.

The gauge fields $U_\mu(n)$ transform under local gauge transformations as

$$\begin{aligned} U_\mu(n) &\to U'_\mu(n) = \Omega(n)U_\mu(n)\Omega(n+\hat{\mu})^\dagger \\ U_{-\mu(n)} &\to U'_{-\mu}(n) = \Omega(n)U_{-\mu}(n)\Omega(n-\hat{\mu})^\dagger . \end{aligned} \tag{4.13}$$

With this definition it follows directly that the product along any closed loop on the lattice is gauge invariant.

On the lattice the following gauge invariant objects can be constructed: First, so-called strings, path-ordered products of links that either have a fermion on one end and an antifermion at the other, or, in case of periodic boundary conditions, wind around the lattice. If a string goes around the lattice in temporal direction it is called a Polyakov line (or loop), otherwise it is a so-called Wilson line. The second class of gauge-invariant objects consists of closed Wilson loops. Also we can use the total anti-symmetric epsilon tensor ϵ^{abc} to construct anti-symmetric colour combinations for, e.g. the nucleon.

The smallest closed loops are the so-called plaquettes

$$\begin{aligned} U_{\mu\nu}(n) &= U_\mu(n)U_\nu(n+\hat{\mu})U_{-\mu}(n+\hat{\mu}+\hat{\nu})U_{-\nu}(n+\hat{\nu}) \\ &= U_\mu(n)U_\nu(n+\hat{\mu})U_\mu(n+\hat{\nu})^\dagger U_\nu(n)^\dagger . \end{aligned} \tag{4.14}$$

Fig. 4.2 depicts a plaquette. Summing over all plaquettes, counting each plaquette just in one orientation, yields

$$\frac{1}{g^2}\sum_{n\in\Lambda}\sum_{\mu<\nu}\left[1-\frac{1}{2}\left(U_{\mu\nu}(n)+U_{\mu\nu}(n)^\dagger\right)\right] = \frac{1}{4}a^4\sum_{n\in\Lambda}\sum_{\mu,\nu}F_{\mu\nu}(n)F_{\mu\nu}(n) \ +\mathcal{O}(a^2) \tag{4.15}$$

where $F_{\mu\nu}(n)$ is the discretised version of the field strength tensor

$$\begin{aligned} F_{\mu\nu}(n) &= \frac{1}{a}\big(A_\nu(n+\hat{\mu}) - A_\nu(n)\big) - \big(A_\mu(n+\hat{\nu}) - A_\mu(n)\big) \\ &\quad + ig_0[A_\mu(n), A_\nu(n)] \\ &= F_{\mu\nu}(x) + \mathcal{O}(a). \end{aligned} \tag{4.16}$$

In the continuum limit $a \to 0$ we find the field strength tensor approaching the continuum version

$$F_{\mu\nu}(n) \xrightarrow{a\to 0} F_{\mu\nu}(x). \tag{4.17}$$

This allows us to take the trace of the left side of Eq. (4.15) as the discretised gauge action. This action was first formulated in this form by Wilson [43].

For SU(N_c) the Wilson gauge action reads

$$S_G[U] = \beta \sum_{n\in\Lambda}\sum_{\mu<\nu}\big[1 - \frac{1}{N}\mathrm{Re}\ \mathrm{tr}U_{\mu\nu}(n)\big] \tag{4.18}$$

with $\beta = 2N_c/g_0^2$. The discretisation error of this gauge action is of order $\mathcal{O}(a^2)$.

4.4 Naive Discretisation of Fermion Fields

In order to formulate a discretised version of the fermionic action on the lattice we need to make sure that it is invariant under local gauge transformation, compare to Eqs. (4.13),

$$\begin{aligned} \psi(n) &\to \psi'(n) = \Omega(n)\psi(n) \\ \overline{\psi}(n) &\to \overline{\psi}'(n) = \overline{\psi}(n)\Omega(n)^\dagger \end{aligned} \tag{4.19}$$

where we have dropped the flavour index. The mass term of the continuum fermion action is already invariant under this gauge transformation. It remains to change the partial derivative into a gauge invariant form. We identify the term

$$\overline{\psi}'(n)U'_\mu\psi'(n+\hat{\mu}) = \overline{\psi}(n)\Omega(n)^\dagger U'_\mu(n)\Omega(n+\hat{\mu})\psi(n+\hat{\mu}) \tag{4.20}$$

$$= \overline{\psi}(n)U_\mu\psi(n+\hat{\mu}) \tag{4.21}$$

to be gauge invariant where we used Eqs. (4.13). We use this term to discretise the partial derivative and arrive at the so-called naive discretisation of the fermion action

$$S_F = a^4 \sum_{n\in\Lambda} \overline{\psi}(n)\Big(\sum_{\mu=1}^{4} \gamma_\mu \frac{U_\mu(n)\psi(n+\hat{\mu}) - U_{-\mu}(n)\psi(n-\hat{\mu})}{2a} + m_0\psi(n)\Big) \tag{4.22}$$

$$= a^4 \sum_{n,m\in\Lambda} \overline{\psi}(n) D_{\text{naiv}}(n|m)\psi(m) \tag{4.23}$$

where the action is a functional of the fermion and gauge fields, i.e. $S_F = S_F[\psi, \overline{\psi}, U]$. In Eq. (4.23) we introduced the naive Dirac operator D_{naiv} on the lattice

$$D_{\text{naiv}}(n|m) = \frac{1}{2a}\sum_{\mu=1}^{4}\Big(\gamma_\mu U_\mu(n)\delta_{n+\hat{\mu},m} - \gamma_\mu U_{-\mu}(n)\delta_{n-\hat{\mu},m}\Big) + m_0\delta_{n,m} \tag{4.24}$$

and assumed matrix-vector notation. The Dirac Operator D is often called the fermion matrix M.

We will now show that the naive discretisation of the fermion action leads to the so-called fermion doubling problem. We calculate the Fourier transform of the Dirac operator on the lattice for the free case ($U_\mu = 1$) analytically. A detailed calculation can be found in the App. A.1. The Fourier transformation reads

$$\widetilde{D}_{\text{naiv}}(p|q) = \delta_{p,q}\Big(m_0 + \frac{i}{a}\sum_{\mu=1}^{4}\gamma_\mu \sin(p_\mu a)\Big). \tag{4.25}$$

In momentum space this operator is diagonal, which in turn leads to a straight-forward calculation of its inverse by calculating the inverse of the bracket of the rhs of Eq. (4.25)

$$\widetilde{D}_{\text{naiv}}(p|q)(p)^{-1} = \frac{m_0 - ia^{-1}\sum\limits_{\mu}\gamma_\mu \sin(p_\mu a)}{m_0^2 + a^{-2}\sum\limits_{\mu}\sin(p_\mu a)^2}. \tag{4.26}$$

The inverse of the Dirac operator in position space is proportional to this result, so we can make statements on the poles of this operator. For massless fermions ($m_0 = 0$) the naive Dirac operator has poles at

$$p = (0,0,0,0), \tag{4.27}$$

$$(\pi/a, 0, 0, 0), \ldots, (\pi/a, \pi/a, \pi/a, \pi/a). \tag{4.28}$$

The first pole describes the single fermion which is also described by the continuum operator. The other 15 poles at the corners of the Brillouin zone lead to unwanted contributions. This problem is known as the fermion doubling problem.

4.5 Nielsen and Ninomiya Theorem

Nielsen and Ninomiya formulated the so-called no-go theorem [44, 45]. It states that for every formulation of fermions on the lattice the following four conditions are not fulfilled all at the same time:

Locality: The Dirac operator $D(x)$ is local. Locality means that the coupling between the fields vanishes exponentially $\exp(-\mu|x|)$.

Continuum limit: The Fourier transform of the massless ($m_0 = 0$) Dirac operator fulfills $\widetilde{D}(p) = i\gamma_\mu p_\mu + \mathcal{O}(ap^2)$, i.e. we reach the right continuum limit.

No doublers: $\widetilde{D}(p)$ in the massless case is invertible, i.e. we have no doublers.

Chiral symmetry: D is γ_5-hermitian, i.e. $\gamma_5 D + D\gamma_5 = 0$ – we have chiral symmetry.

To overcome the fermion doubling problem, Wilson suggested to introduce an additional term, the so-called Wilson term to the fermionic action [46].

4.6 Wilson Fermion Action

We introduce an additional term, the Wilson term, to the Dirac operator in Fourier space, see Eq. (4.25),

$$\widetilde{D}^W(p|q) = \delta_{p,q}\left(m_0 + \frac{i}{a}\sum_{\mu=1}^{4}\gamma_\mu \sin(p_\mu a) + \frac{r}{a}\sum_{\mu=1}^{4}\left(1-\cos(p_\mu a)\right)\right) \tag{4.29}$$

where r is the so-called Wilson parameter. In the following we will set $r = 1$. This term vanishes for the physical pole $p = (0, 0, 0, 0)$. For each of the other (unphysical) poles contributing to the fermion doubling problem it provides an extra contribution $2/a$ for each element equal to π/a. These contributions become in the continuum limit infinitely heavy and decouple from the theory. In order to find the operator in position space we apply the inverse Fourier transformation and we get the Wilson Dirac operator in position space

$$D^W(n|m) = \big(m_0 + \frac{4}{a}\big)\delta_{n,m} - \sum_{\mu=1}^{4} \frac{(1-\gamma_\mu)U_\mu(n)\delta_{n+\hat{\mu},m} + (1+\gamma_\mu)U_{-\mu}(n)\delta_{n-\hat{\mu},m}}{2a}. \quad (4.30)$$

After rescaling the fermionic fields

$$\psi \rightarrow \sqrt{m_0 + 4/a}\,\psi \quad (4.31)$$

$$\overline{\psi} \rightarrow \sqrt{m_0 + 4/a}\,\overline{\psi} \quad (4.32)$$

we can rewrite this operator as

$$D^W = 1 - \kappa H \quad (4.33)$$

with the so-called hopping parameter, or Wilson quark mass parameter,

$$\kappa = \frac{1}{2(am_0 + 4)} \quad (4.34)$$

and the hopping matrix

$$H(n|m) = \sum_{\mu=1}^{4} \big[(1-\gamma_\mu)U_\mu(n)\delta_{n+\hat{\mu},m} + (1+\gamma_\mu)U_{-\mu}(n)\delta_{n-\hat{\mu},m}\big]. \quad (4.35)$$

The inverse of the Dirac operator D^{-1} and its determinant $\det[D]$ are expanded in powers of κ. For the quark propagator one can use the geometric series

$$D^{-1} = (1-\kappa H)^{-1} = \sum_{j=0}^{\infty} \kappa^j H^j. \quad (4.36)$$

The series converges for $\kappa||H|| < 1$. The norm of the hopping term obeys $||H|| \leq 8$ and thus the series converges for $\kappa < 1/8$.

The mass of a Wilson quark is given by

$$am_0 = \frac{1}{2\kappa} - 4 = \frac{1}{2\kappa} - \frac{1}{2\kappa_c}. \quad (4.37)$$

It vanishes in the free case for $\kappa = \kappa_c = 1/8$. We define $am_0 = 1/2\kappa - 1/2\kappa_c$ also for the interacting theory, so that κ_c becomes dependent on the lattice spacing a. As a consequence the quark mass receives not only multiplicative but also additive renormalisation.

To determine κ_c one can use the chiral relation $m_\pi^2 \propto m_0$, calculate m_π^2 as a function of $1/2\kappa$ and extrapolate to zero and use Eq. (4.37). Another way is to calculate the quark mass using the PCAC relation based on the axial vector Ward identity as a function of $1/2\kappa$ and again extrapolate to zero.

The quark mass is determined by the hopping expansion parameter κ. The larger κ the smaller the quark mass. In terms of the Wilson quark mass parameter κ a decreasing quark mass is equivalent to κ approaching κ_c, the critical value where the quark mass vanishes. However, for a given set of simulation parameters (β, κ) the critical hopping parameter κ_c is uniquely defined only as a statistical average over the whole gauge field ensemble, while its value on individual configurations fluctuates. From this, a complication arises: If the fluctuating value of κ_c gets close to κ, the quark matrix may become singular. This problem becomes increasingly severe with decreasing quark mass, β and L.

Adding the Wilson term to the fermion action eliminates the fermion doublers, but it breaks explicitly chiral symmetry of the action.

4.7 Discretisation Errors

Formulating the theory on the lattice introduces discretisation errors. To estimate these errors we take the so-called naive continuum limit.

We expand the gauge fields in Eq. (4.8) for small a

$$U_\mu(n) = 1 + iaA_\mu(n) + \mathcal{O}(a^2) \tag{4.38}$$

$$U_{-\mu}(n) = 1 - iaA_\mu(n-\hat{\mu}) + \mathcal{O}(a^2) \tag{4.39}$$

and find the fermionic action in Eq. (4.22) splitting into a free part S_{F_0} including the mass term and an interaction part S_{F_I}

$$S_F[\psi, \overline{\psi}, U] = S_{F_0}[\psi, \overline{\psi}] + S_{F_I}[\psi, \overline{\psi}, A] \tag{4.40}$$

where the interaction term is

$$S_{F_I}[\psi, \overline{\psi}, A] = ia^4 \sum_{n\in\Lambda} \sum_{\mu=1}^{4} \overline{\psi}(n)\gamma_\mu \frac{1}{2}\left(A_\mu(n)\psi(n+\hat{\mu}) - A_\mu(n-\hat{\mu})\psi(n-\hat{\mu})\right) \tag{4.41}$$

$$= ia^4 \sum_{n\in\Lambda} \sum_{\mu=1}^{4} \overline{\psi}(n)\gamma_\mu A_\mu(n)\psi(n) + \mathcal{O}(a) \tag{4.42}$$

where we used $\psi(n \pm \hat{\mu}) = \psi(n) + \mathcal{O}(a)$ and $A_\mu(n-\hat{\mu}) = A_\mu(n) + \mathcal{O}(a)$. Thus, the Wilson action has discretisation errors of order $\mathcal{O}(a)$.

4.8 Integration with Monte Carlo Methods

The introduction of the lattice has reduced the dimensionality of the path integral to be finite. But still, the integral is far too high dimensional to be carried out, even numerically, exactly. It is convenient to treat the fermionic and gluonic parts separately.

With the action split into a fermionic and gluonic part, see Eq. (4.2), we write the matrix element in the path integral formalism, compare to Eq. (2.11), as

$$\langle O\rangle = \frac{1}{Z}\int \mathcal{D}[\psi,\overline{\psi}]\mathcal{D}[U]e^{S_F[\psi,\overline{\psi},U]-S_G[U]}O \tag{4.43}$$

$$= \langle\langle O\rangle_F\rangle_G \tag{4.44}$$

where $O = O[\psi,\overline{\psi},U]$ is a combination of quark and gluon fields. We separated the fermionic part

$$\langle O\rangle_F = \frac{1}{Z_F[U]}\int \mathcal{D}[\psi,\overline{\psi}]e^{-S_F[\psi,\overline{\psi},U]}O[\psi,\overline{\psi},U] \tag{4.45}$$

with

$$Z_F[U] = \int \mathcal{D}[\psi,\overline{\psi}]e^{-S_F[\psi,\overline{\psi},U]} \tag{4.46}$$

from the gluonic part

$$\langle B\rangle_G = \frac{1}{Z_G}\int \mathcal{D}[U]e^{-S_G[U]}Z_F[U]B[U] \tag{4.47}$$

with

$$Z_G = \int \mathcal{D}[U]e^{-S_G[U]}. \tag{4.48}$$

The fermion part in Eq. (4.45) can be integrated exactly due to the Grassman nature of the integration variables. The quark fields obey to the Fermi statistic and are represented by total-anticommuting Grassman variables. In the denominator of Eq. (4.45), the Gaussian integrals can be carried out exactly with the Matthews-Salam formula [47, 48]

$$Z_F[U] = \int \mathcal{D}[\psi,\overline{\psi}]\exp\Bigg[\sum_{n,m\in\Lambda}\overline{\psi}(n)D(n|m)\psi(m)\Bigg]. \tag{4.49}$$

A possible minus sign in the exponent can be absorbed by a redefinition of the Dirac operator D. We determine the partition function as the fermion determinant

$$Z_F[U] = \det(D). \tag{4.50}$$

We suppose now, that O involves a number of fermion fields

$$\langle O\rangle_F = \langle\psi_{i_1}\overline{\psi}_{j_1} \dots \psi_{i_n}\overline{\psi}_{j_n}\rangle_F. \tag{4.51}$$

To calculate the nominator of the expectation value we use Wick's theorem to integrate out the fermionic fields

$$\langle\psi_{i_1}\overline{\psi}_{j_1} \dots \psi_{i_n}\overline{\psi}_{j_n}\rangle_F = \frac{1}{Z_F}\int \mathcal{D}[\psi,\overline{\psi}]\psi_{i_1}\overline{\psi}_{j_1} \dots \psi_{i_n}\overline{\psi}_{j_n} e^{\sum\overline{\psi}(n)D(n|m)\psi(m)} \tag{4.52}$$

$$= (-1)^n \sum_{P(1,\dots,n)} \operatorname{sign}(P) D^{-1}_{i_1,j_{P_1}} \cdots D^{-1}_{i_n,j_{P_n}}. \tag{4.53}$$

The sum in the first line runs over all $n, m \in \Lambda$ and $P(1,\dots,n)$ is the set of all permutations of the numbers from 1 to n. This procedure of integrating out the fermion fields is also called carrying out the fermion contractions.

The gluon part of the integral is carried out by means of Monte Carlo methods. A set of gauge configurations $\{U_n \,|1 \leq n \leq N_{\text{conf}}\}$ is generated where N_{conf} is the number of gauge configurations and where each U_n is sampled according to the probability distribution density

$$\mathrm{d}P(U) = \frac{e^{-S_G[U]}\mathcal{D}[U]}{\int \mathcal{D}[U]e^{-S_G[u]}}, \tag{4.54}$$

the so-called Gibbs measure. Sampling the gauge configurations according to this measure ensures that the individual contributions to the path integral are given a different importance determined by the Boltzmann factor $\exp(-S_G)$.

Given the gauge configurations U_n sampled with the Gibbs measure, we can approximate the path integral by

$$\langle B\rangle_G \approx \frac{1}{N_{\text{conf}}}\sum_{n=1}^{N_{\text{conf}}} B[U_n] \tag{4.55}$$

where the sum converges quickly since the gauge configurations are sampled according to the Boltzmann factor. Due to probability theory we approach the exact path integral value in the limit $N_{\text{conf}} \to \infty$.

4.8.1 Markov Chains

In order to find the gauge configurations we start from an arbitrary configuration U_0 and construct a stochastic sequence U_n of configurations that follow an equilibrium distribution

$P(U)$. This is done with a so-called Markov process

$$U_0 \to U_1 \to U_2 \to \dots . \tag{4.56}$$

The change of a field configuration is called an update or a Monte Carlo step. A Markov process is characterised by a conditional transition probability

$$P(U_n = U'|U_{n-1} = U) = T(U'|U) \tag{4.57}$$

which determines the transition probability to go to configuration U' if starting from U. The transition probability satisfies the following relations

$$0 \leq T(U'|U) \leq 1 \qquad \text{and} \qquad \sum_{U'} T(U'|U) = 1. \tag{4.58}$$

In equilibrium the probability for hopping out of a configuration should be same as hopping into it. The corresponding balance equation reads

$$\sum_{U} T(U'|U)P(U) = \sum_{U} T(U|U')P(U'). \tag{4.59}$$

Although other solutions to this equation are known, most algorithms implement the so-called detailed balance condition

$$T(U'|U)P(U) = T(U|U')P(U'), \tag{4.60}$$

a sufficient but not necessary condition. The transition probability can be written as

$$T(U'|U) = p_c(U \to U')p_A(U \to U') \tag{4.61}$$

where p_c is the probability to choose a candidate configuration U' and p_A is the acceptance rate.

For example for quenched QCD where one drops the fermion determinant ($\det D = 1$) we may use over-relaxation [49] to propose a new configuration and use the Metropolis acceptance probability

$$p_A = \min(1, e^{-\Delta S}) \tag{4.62}$$

where ΔS denotes the difference of the action for the proposed configuration to the original one.

State	J^{PC}	Γ	Particles
Scalar	0^{++}	1, γ_4	f_0,a_0,K_0^*,...
Pseudoscalar	0^{-+}	γ_5, $\gamma_4\gamma_5$	$\pi^\pm$, π^0, η, $K^\pm$, K^0, ...
Vector	1^{--}	γ_i, $\gamma_4\gamma_i$	$\rho^\pm$, ρ^0, ω, K^*, ϕ, ...
Axial vector	1^{++}	$\gamma_i\gamma_5$	a_1, f_1, ...
Tensor	1^{+-}	$\gamma_i\gamma_j$	h_1, b_1, ...

Table 4.1 – Quantum numbers of the most commonly used meson interpolators according to the general form 4.63.

Simulations that include the fermion determinant (dynamical QCD) typically use the Hybrid Monte Carlo (HMC) algorithm [50]. In the standard HMC algorithm the Dirac operator appears quadratically, thus only an even number of flavours can be simulated. This restriction can be overcome in the case of a positive fermion determinant with the polynomial [51, 52] or rational HMC [53, 54].

4.9 Hadron Interpolator

In order to calculate correlators and matrix elements the first step is the identification of the correct hadron interpolators O, $\overline{O}$ such that they feature the same quantum numbers, i.e. possess the same symmetries as the hadron states we are interested in. In particular we have to ensure that our interpolators transform in the correct way under parity $\mathcal{P}$, charge $\mathcal{C}$ transformation and have correct total spin J. The symmetry transformations implemented on the lattice are shown in App. A.2.

In the following we will use the symbol M for meson interpolators, and B for baryon interpolators. The meson interpolators are most commonly used in the form

$$M_\Gamma(n) = c_{f_1 f_2}\overline{\psi}^{(f_1)}(n)\Gamma\psi^{(f_2)}(n) \tag{4.63}$$

where Γ represents a combination of γ-matrices. Tab. 4.1 lists the quantum numbers and the according combination of γ-matrices of typically used meson interpolators. It is common practice to instead of using a flavour index f attached to the fermionic field denoted by

$\psi^{(f)}$, $\overline{\psi}^{(f)}$ to use a different symbol for each of the flavours. Thus, the spinors u, $\overline{u}$, d, $\overline{d}$, s, $\overline{s}$ denote the quarks and anti-quarks of the three light flavours. The matrix c denotes the corresponding flavour matrix.

We consider meson interpolators involving u and d quarks (and their anti-quarks) and set $c_{ud} = 1$. For example, the pion interpolator of the iso-triplet π^- is given by

$$M(n) = \overline{u}(n)\gamma_5 d(n). \tag{4.64}$$

It features the correct quantum numbers $J^{PC} = 0^{-+}$ which can be shown by applying the charge and parity transformations and it has the correct spin number. The corresponding interpolator for π^+ is obtained by interchanging the quark flavours $u \leftrightarrow d$.

Baryons are composed out of three quarks. The proton p and the neutron n are the $I_3 = \pm 1/2$ components of an isospin-doublet ($I = 1/2$). Their masses are almost degenerate, i.e. $m_p = 938.27$ MeV and $m_n = 939.57$ MeV, stating that isospin is a good symmetry. Considering the electric charges of the neutron and proton we find that the proton must be a *uud*-state and the neutron a *ddu*-state. Due to nearly exact isospin-symmetry of these two particles it is common practice to write down just one interpolator for both, i.e. the baryon interpolator

$$B(n) = \epsilon_{abc} u(n)_a \big(u(n)_b^T C\gamma_5 d(n)_c\big) \tag{4.65}$$

$$\overline{B}(n) = \epsilon_{abc} \big(\overline{u}(n)_a C\gamma_5 \overline{d}(n)_b^T\big)\overline{u}(n)_c. \tag{4.66}$$

The baryon interpolators have a free Dirac index, i.e. $B = B_\alpha$ and $\overline{B} = \overline{B}_\alpha$. The term in parentheses combines the u and the d quark into a so-called diquark making use of the charge conjugation matrix C and γ_5. The diquark contracts the Dirac indices and thus represents an ordinary number in spin structure. The interpolator for the $I_3 = -1/2$ case is obtained by interchanging the constituents, i.e. $u \leftrightarrow d$. Since the diquark has the correct isospin symmetry it remains to show that our interpolator transforms correctly under parity.

Under parity $\mathcal{P}$ our nucleon interpolator transforms like, see App. A.2,

$$B^{\mathcal{P}}(\mathbf{n}, n_4) = \gamma_4 B(-\mathbf{n}, n_4). \tag{4.67}$$

As we will see later, the change of the spatial vector $\mathbf{n}$ into $-\mathbf{n}$ is irrelevant, because we carry out a sum over all spatial components. Our interpolator couples to states with positive and negative parity. Thus we need the parity projector

$$P_\pm = \frac{1}{2}(1 \pm \gamma_4) \tag{4.68}$$

to project to states with positive $P = +1$ or negative $P = -1$ parity

$$B_\pm(n) = \frac{1}{2}\big(B(n) \pm B^{\mathcal{P}}(n)\big). \tag{4.69}$$

4.10 Momentum Projection

In order to study hadronic correlation functions with finite initial and final momentum a momentum projection to the interpolators is implemented. The interpolators are represented in momentum space by a Fourier transformation

$$\widetilde{O}(\mathbf{p}, n_t) = \frac{1}{\sqrt{|\Lambda_3|}} \sum_{n\in\Lambda_3} O(\mathbf{n}, n_t) e^{-ia\mathbf{n}\cdot\mathbf{p}} \tag{4.70}$$

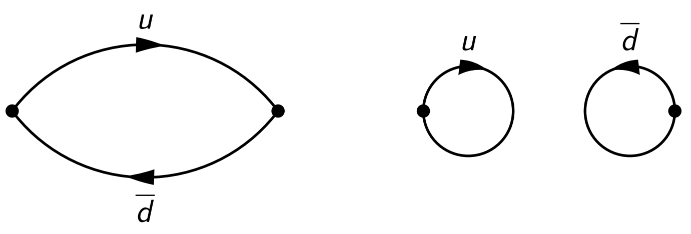
where Λ_3 are the labels of lattice point in the spatial plane. The hadron interpolator O stands for one of our previously defined meson or baryon interpolators M or B. If we prepare in such a way the initial state and the final state with with momentum p and p' we arrive at the Euclidean hadron correlators

$$\langle \overline{\widetilde{O}}(\mathbf{p}', n_t), \widetilde{O}(\mathbf{p}, 0)\rangle = \frac{1}{\sqrt{|\Lambda_3|}} \sum_{n,m\in\Lambda_3} e^{-ia\mathbf{n}\cdot\mathbf{p}'} e^{-ia\mathbf{m}\cdot\mathbf{p}} \langle \overline{O}(\mathbf{n}, n_t), O(\mathbf{m}, 0)\rangle. \tag{4.71}$$

We drop the tilde symbol in the following if the argument is of momentum type.

4.11 Hadron Correlators

In order to evaluate the hadron correlators we write down the correlation function in terms of hadron interpolators and carry out the fermion contractions. For a meson of the form



(a) Connected diagram **(b)** Disconnected diagram

Figure 4.3 – The 2-point quark correlation function for a meson. The quark line connected term (left) and the disconnected term (right).

as shown in Eq. (4.63) with momentum projection we find

$$C_{\Gamma_1\Gamma_2}(\mathbf{p},t) = \left\langle M_{\Gamma_1}(\mathbf{p},t)\overline{M}_{\Gamma_2}(\mathbf{p},0)\right\rangle \tag{4.72}$$

$$= -\frac{1}{\sqrt{|\Lambda_3|}} \sum_{\substack{\mathbf{n},\mathbf{m}\in\Lambda_3 \\ f_1,f_2}} |c_{f_1f_2}|^2 e^{-ia\mathbf{p}\cdot(\mathbf{n}-\mathbf{m})} \Bigg[\left\langle \mathrm{tr}\left[D_{f_2}^{-1}(n,t|m,0)\gamma_4\Gamma_2^\dagger\gamma_4 D_{f_1}^{-1}(m,0|n,t)\Gamma_1\right]\right\rangle_G \tag{4.73}$$

$$- \delta_{f_1,f_2}\left\langle \mathrm{tr}\left[D_f^{-1}(n,t|n,t)\Gamma_1\right]\mathrm{tr}\left[D_f^{-1}(m,0|m,0)\gamma_4\Gamma_2^\dagger\gamma_4\right]\right\rangle_G \Bigg]$$

where the trace runs over Dirac and colour indices. The first term is the quark line connected term, while the second term represents the disconnected term, see Fig. 4.3. The disconnected term is computationally expensive to calculate since it involves the evaluation of the propagator at every point on the t plane. However, if we restrict ourselves to flavour non-singlets particles, then the second term is not present.

The Green's function or quark propagator is defined as

$$D^{-1}(\mathbf{n},t|\mathbf{m},0) = \langle\psi(\mathbf{n},t)\overline{\psi}(\mathbf{m},0)\rangle_F. \tag{4.74}$$

Since we are averaging over many gauge configurations, we can use translational invariance to move every source from $(\mathbf{n},0)$ to $(\mathbf{0},0)$. For flavour non-singlet mesons with u and d quarks we arrive at

$$C_{\Gamma_1\Gamma_2}(\mathbf{p},t) = -\sum_{\mathbf{n}} e^{-ia\mathbf{p}\cdot\mathbf{n}} \left\langle \left[\Gamma_1 D_d^{-1}(\mathbf{n},t|0)\gamma_4\Gamma_2^\dagger\gamma_4\right]^{ab}_{\alpha\beta}\left[\left(\gamma_5 D_u^{-1}(\mathbf{n},t|0)\gamma_5\right)^*\right]^{ab}_{\alpha\beta}\right\rangle_G \tag{4.75}$$

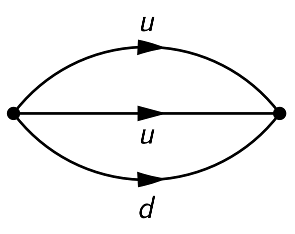


Figure 4.4 – The 2-point quark correlation function for a baryon. Only quark line connected terms are present.

where we now made explicit the colour and Dirac indices and where we made use of the γ_5-hermiticity of the Dirac operator in order to obtain the backward running propagator

$$\gamma_5 D^{-1}\gamma_5 = D^{-1\dagger}. \tag{4.76}$$

A similar calculation for the correlation functions using our nucleon interpolators $B_\pm$ gives

$$C_\Gamma(\mathbf{p},t) = \sum_{\alpha\beta}\Gamma_{\beta\alpha}\left\langle B_\alpha(\mathbf{p},t)\overline{B}_\beta(\mathbf{p},0)\right\rangle \tag{4.77}$$

$$\begin{aligned} &= \sum_{\mathbf{n}} e^{-ia\mathbf{p}\cdot\mathbf{n}}\epsilon_{abc}\epsilon_{a'b'c'}\times \\ &\left\langle \mathrm{tr}\left[\Gamma D_u^{-1}(\mathbf{n},t|0)^{aa'}\right]\mathrm{tr}\left[\widetilde{D}_d^{-1}(\mathbf{n},t|0)^{bb'}D_u^{-1}(\mathbf{n},t|0)^{cc'}\right]\right. \\ &\left. + \,\mathrm{tr}\left[\Gamma D_u^{-1}(\mathbf{n},t|0)^{aa'}\widetilde{D}_d^{-1}(\mathbf{n},t|0)^{bb'}D_u^{-1}(\mathbf{n},t|0)^{cc'}\right]\right\rangle_G \end{aligned} \tag{4.78}$$

where the trace runs over the Dirac index and we have defined

$$\widetilde{X} = (C\gamma_5 X\gamma_5 C)^T \tag{4.79}$$

with C the charge conjugation matrix.

This result states that for the nucleon correlation function only connected parts contribute, see Fig. 4.4.

4.12 Symanzik Improvement Program

It was pointed out that when discretising the continuum gluonic and fermionic actions we introduce discretisation errors. In the following we will introduce an additional term to the Wilson fermion action in order to improve the discretisation errors from $\mathcal{O}(a)$ to $\mathcal{O}(a^2)$.

Following [55, 56, 57, 58, 59] the discretised action can for small energies be written as an effective action

$$S_{\text{eff}} = \int \mathrm{d}^4x \left(L^{(0)}(x) + aL^{(1)}(x) + a^2L^{(2)}(x) + \dots \right). \tag{4.80}$$

$L^{(0)}$ is the usual continuum QCD Lagrangian and $L^{(k)}$ are correction terms. Since we are interested in $\mathcal{O}(a)$ improvement we limit ourself to terms of order $L^{(1)}$ and neglect all other terms with higher orders of a. Compared to $L^{(0)}$ which is a dimension-4 operator, the $L^{(1)}$ must be of dimension 5. The leading order correction term $L^{(1)}$ can be written as a linear combination of these operators

$$L_1^{(1)}(x) = \overline{\psi}(x)\sigma_{\mu\nu}F_{\mu\nu}(x)\psi(x) \tag{4.81}$$

$$L_2^{(1)}(x) = \overline{\psi}(x)\overrightarrow{D}_\mu(x)\overrightarrow{D}_\mu(x)\psi(x) + \overline{\psi}(x)\overleftarrow{D}_\mu(x)\overleftarrow{D}_\mu(x)\psi(x) \tag{4.82}$$

$$L_3^{(1)}(x) = m_0\,\mathrm{tr}[F_{\mu\nu}(x)F_{\mu\nu}(x)] \tag{4.83}$$

$$L_4^{(1)}(x) = m_0\left(\overline{\psi}(x)\gamma_\mu\overrightarrow{D}_\mu(x)\psi(x) - \overline{\psi}(x)\gamma_\mu\overleftarrow{D}_\mu(x)\psi(x)\right) \tag{4.84}$$

$$L_5^{(1)}(x) = m_0^2\overline{\psi}(x)\psi(x) \tag{4.85}$$

with $\sigma_{\mu\nu} = [\gamma_\mu, \gamma_\nu]/2i$ and $\overrightarrow{D}$ ($\overleftarrow{D}$) denoting the covariant derivative acting to the right (left) side. The number of operators can be reduced by applying the field equation $(\gamma_\mu D_\mu + m_0)\psi = 0$ which results in the two relations

$$L_1^{(1)} - L_2^{(1)} + 2L_5^{(1)} = 0\ , \qquad L_4^{(1)} + 2L_5^{(1)} = 0. \tag{4.86}$$

This reduces the number of operators and we are left with $L_1^{(1)}$, $L_3^{(1)}$, and $L_5^{(1)}$. Two of these, i.e. $L_3^{(1)}$ and $L_5^{(1)}$ are already linearly present in the action and thus can be absorbed in a redefinition of the bare parameters m_0 and g_0. Thus it is sufficient to work with $L_1^{(1)}$ and we write the $\mathcal{O}(a)$ improved Wilson fermion action

$$S_F^I = S_F^W + c_{\text{sw}}a^5 \sum_{n\in\Lambda}\sum_{\mu<\nu} \overline{\psi}(n)\frac{1}{2}\sigma_{\mu\nu}F_{\mu\nu}(n)\psi(n) \tag{4.87}$$

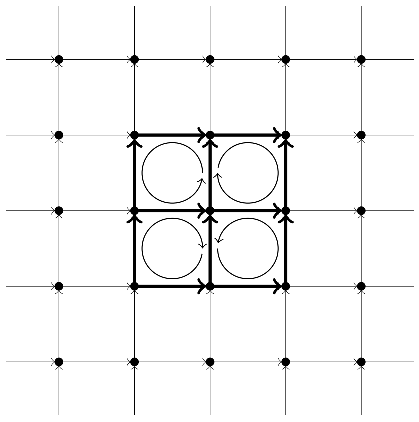


Figure 4.5 – The clover term

with the *Sheikholeslami-Wohlert coefficient* c_{sw} and the lattice version of the field strength tensor $F_{\mu\nu}(n)$. A widely used version is

$$F_{\mu\nu}(n) = \frac{-i}{8a^2}(Q_{\mu\nu}(n) - Q_{\nu\mu}(n)) \tag{4.88}$$

where $Q_{\mu\nu}(n)$ is the so-called *clover term*

$$Q_{\mu\nu}(n) = U_{\mu,\nu}(n) + U_{\nu,-\mu}(n) + U_{-\mu,-\nu}(n) + U_{-\nu,\mu}(n). \tag{4.89}$$

It is the sum of the four plaquettes $U_{\mu\nu}(n)$ around lattice point n in the $\mu - \nu$ plane, see Fig. 4.5 and compare to the field strength tensor in Eq. (4.16) used in the Wilson gauge action.

The improvement of the lattice fermion action is sufficient to reduce discretisation errors to $\mathcal{O}(a^2)$ for on-shell quantities such as hadron masses. In order to achieve this level of improvement also for matrix elements we need to improve the hadron interpolators as well. We introduce the pseudoscalar density P^a

$$P^a(n) = \frac{1}{2}\overline{\psi}(n)\gamma_5\tau^a\psi(n), \tag{4.90}$$

with τ^a is one of the Pauli matrices acting in $N_f = 2$ flavour space. Further we introduce the vector fields

$$\begin{aligned} V_\mu(n) &= \frac{1}{2}\overline{\psi}(n)\gamma_\mu\psi(n) \\ V_\mu^a(n) &= \frac{1}{2}\overline{\psi}(n)\gamma_\mu\tau^a\psi(n), \end{aligned} \tag{4.91}$$

and axial vector fields

$$\begin{aligned} A_\mu(n) &= \frac{1}{2}\overline{\psi}(n)\gamma_\mu\gamma_5\psi(n) \\ A_\mu^a(n) &= \frac{1}{2}\overline{\psi}(n)\gamma_\mu\gamma_5\tau^a\psi(n). \end{aligned} \tag{4.92}$$

The improvement of the isovector axial current A_μ^a and the pseudoscalar density P^a are carried out by selecting suitable dimension-4 operators and absorbing linear dependent operators in the renormalisation factors.

The renormalised and improved interpolators for the isovector axial current A_μ^a and the pseudoscalar density P^a read

$$A_\mu^{(r)a}(n) = Z_A(1 + b_A am)\big(A_\mu^a(n) + c_A a\widehat{\partial}_\mu P^a(n)\big) \tag{4.93}$$

$$P^{(r)a}(n) = Z_P(1 + b_P am)P^a(n) \tag{4.94}$$

where b_A and b_P are real parameters and $\widehat{\partial}_\mu$ denotes the symmetric difference operator

$$\widehat{\partial}_\mu f(n) = \frac{f(n+\hat{\mu}) - f(n-\hat{\mu})}{2a} \tag{4.95}$$

with the yet to be determined improvement coefficients b_A and c_A for the interpolators and c_{sw} for the fermion action.

The next step is to find a suitable basis of possible irrelevant operators with the same symmetry properties as our operators resulting from the operator product expansion after Euclideanisation in Eq. (3.79). A possible way to find candidates for improved operators [57] is to make general covariant transformations or rotations on the fermion fields [60],

$$\psi = \psi + a(\epsilon\gamma_\mu\overrightarrow{D}^\mu + \eta m_q)\psi + \mathcal{O}(a^2) \tag{4.96}$$

$$\overline{\psi} = \psi + a(-\epsilon\gamma_\mu\overrightarrow{D} + \eta m_q)\psi + \mathcal{O}(a^2) \tag{4.97}$$

together with

$$\overline{\chi}\overleftarrow{D}_\mu\psi + \overline{\chi}\overrightarrow{D}_\mu\psi \to \partial_\mu(\overline{\chi}\psi) + \mathcal{O}(a^2) \tag{4.98}$$

where ∂_μ can be taken to be the simple discretisation

$$(\partial_\mu f)(x) = \frac{1}{2}[f(x + a\hat{\mu}) - f(x - a\hat{\mu})]. \tag{4.99}$$

A suitable basis for the one-link operator

$$O_{\mu\nu}^{DV} = \overline{\psi}\gamma_\mu\overleftrightarrow{D}_\nu\psi \tag{4.100}$$

is

$$O_{\mu\nu}^{(\mathrm{impr})DV} = [1 + ac_0 m_q]\overline{\psi}\gamma_\mu \overleftrightarrow{D}_\nu \psi + iac_1\overline{\psi}\sigma_{\mu\lambda}\overleftrightarrow{D}_{[\nu}\overleftrightarrow{D}_{\lambda]}\psi \\ - ac_2\overline{\psi}\overleftrightarrow{D}_{\{\mu}\overleftrightarrow{D}_{\nu\}}\psi \\ + \frac{1}{2}iac_3\partial_\lambda\left(\overline{\psi}\sigma_{\mu\lambda}\overleftrightarrow{D}_\nu\psi\right) \tag{4.101}$$

where we made use of the following abbreviations

$$a_{[\mu}b_{\nu]} = a_\mu b_\nu - a_\nu b_\mu \tag{4.102}$$

$$a_{\{\mu}b_{\nu\}} = a_\mu b_\nu + a_\nu b_\mu. \tag{4.103}$$

4.13 Propagator Calculation

Instead of calculating the full quark propagator $D^{-1}(n|m)$ which is a $(N^3 \times N_T \times N_c \times 4)^2$ matrix, we make use of translational invariance and restrict the calculation to just one column. This represents the propagation from one lattice site n_0 to all other lattice sites n.

$$D^{-1}(n|n_0) = \sum_{m\in\Lambda} D^{-1}(n|m)S_0^{(n_0)}(m) \tag{4.104}$$

with the so-called *point source*

$$S_0^{(n_0)}(m) = \delta(m - n_0). \tag{4.105}$$

In order to obtain the quark propagator the fermion matrix must be inverted 12 times for each quark flavour – once for each combination of Dirac and colour indices. In order to calculate the quark propagator one has to solve the linear system of equations

$$Dx = b \tag{4.106}$$

where D is the Dirac operator, x is the quark propagator we want to find and b is our prepared source.

Calculating the quark propagator is a numerically demanding procedure. Typically most of the compute time is spent with calculating the quark propagators.

The Dirac operator $D(n|m)$ is a large sparse matrix, i.e. it is a matrix with most elements set to zero. Sparse linear systems of equations are traditionally solved by iterative methods. The iterative methods applied are mostly preconditioned Krylov (sub)space solvers.

The idea behind Krylov space solvers is to generate a sequence of approximate solutions x_n, so that the corresponding residuals

$$r_n = b - Dx_n, \qquad n = 0, 1, 2, ... \tag{4.107}$$

converge to the zero vector. Here, convergence means that after a finite number of steps the true solution was approximated within a given accuracy. The residuals can also be calculated recursively

$$r_n = r_{n-1} - Dr_{n-1}. \tag{4.108}$$

Given a non-singular matrix D and a non-zero vector r_0, the n-th Krylov (sub)space $\mathcal{K}_n(D, r_0)$ generated by D from r_0 is

$$\mathcal{K}_n(D, r_0) = \mathrm{span}(r_0, Dr_0, D^2 r_0, \ldots, D^n r_0). \tag{4.109}$$

The n-th residual lies in the n-th Krylov space

$$r_n \in \mathcal{K}_n(D, r_0). \tag{4.110}$$

Clearly $\mathcal{K}_0 \subseteq \mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq ...$ and the dimension increases in each step by one until the equal sign holds. There is a positive (integer) number $\nu = \nu(r_0, D)$ called the grade of r_0 with respect to D such that

$$\dim \mathcal{K}_n(D, r_0) = \begin{cases} n & \text{if} \quad n \leq \nu \\ \nu & \text{if} \quad n > \nu. \end{cases} \tag{4.111}$$

Thus $\mathcal{K}_\nu(D, r_0)$ is the smallest D-invariant subspace that contains r_0 and our true solution x lies in

$$x \in x_0 + \mathcal{K}_\nu(D, r_0). \tag{4.112}$$

Krylov space methods for solving equations like $Dx = b$ have the special feature that the matrix D needs only be given as an operator: For any vector x one must be able to compute Dx and so D may be given as a function or operator and must not be available in full matrix representation. The operation Dx is commonly referred to as the matrix times vector operation.

Krylov space solvers often converge very slowly. In practice, Krylov space solvers are therefore nearly always applied with preconditioning.

We will now introduce the so-called *even-odd preconditioning* [61]. The original equation $Dx = b$ is replaced by

$$\underbrace{V_1^{-1} D V_2^{-1}}_{D'} \underbrace{V_2 x}_{x'} = \underbrace{V_1^{-1} b}_{b'} \tag{4.113}$$

where we have introduced the so-called preconditioning matrices V_1 and V_2. We then apply the Krylov space solver to the preconditioned system of linear equations

$$D' x' = b'. \tag{4.114}$$

For the determination of the matrices V_1 and V_2 we split the Dirac operator D which includes the clover term into two parts: One part which is diagonal in position space and another part which connects two neighbouring lattice sites. The lattice sites can be divided into *even* and *odd* sites, depending on whether adding together the coordinates (n_1, n_2, n_3, n_4) results in an even or odd number. By labelling the lattice sites in such a way that first all odd sites appear and then the even sites, the Dirac operator can be written as

$$D = \begin{pmatrix} X & \kappa D_{oe} \\ \kappa D_{eo} & X \end{pmatrix} \tag{4.115}$$

where X is diagonal in position space

$$X(n|m) = \left[1 + \frac{i \kappa c_{sw}}{2} \sigma_{\mu\nu} F_{\mu\nu} \right] \delta_{n,m}. \tag{4.116}$$

Using

$$V_1 = \begin{pmatrix} X & 0 \\ \kappa D_{eo} & X \end{pmatrix}, \qquad V_2 = \begin{pmatrix} X & \kappa D_{oe} \\ 0 & X \end{pmatrix} \tag{4.117}$$

we find

$$D' = \begin{pmatrix} X^{-1} & 0 \\ 0 & X^{-1}(1 - \kappa^2 D_{eo} X^{-1} D_{oe} X^{-1}) \end{pmatrix}. \tag{4.118}$$

It is sufficient to first consider the even lattice sites and solve

$$\left(X - \kappa^2 D_{eo} X^{-1} D_{oe} \right) x_e = b_e - \kappa D_{eo} X^{-1} b_0. \tag{4.119}$$

With the solution for the even sites x_e the solution for the odd sites is given by

$$X_o = X^{-1}(b_0 - \kappa D_{oe} x_e). \tag{4.120}$$

With even-odd preconditioning a matrix of half the original size must be inverted, but we have to apply D_{oe} as well as D_{eo}.

The most important improvement factor in terms of computational cost is that the off-diagonal elements are now suppressed by κ^2 rather than κ.

Some typically used inverters in Lattice QCD simulations are the conjugate gradient (CG), minimal residual (MR) or BiCGStab solvers [62, 63]. CG solvers can only be applied to systems involving hermitian matrices. The Dirac operator D is not hermitian, but CG can still be used by first multiplying D by γ_5 and then applying the solver or by solving the normal equation

$$D^\dagger D x = D^\dagger x. \tag{4.121}$$

CG while being reliable converges slower than MR or BiCGStab.

4.14 Quark and Gluonic Smearing

Each interpolator with the correct quantum numbers has an overlap not only with the physical state but also with excited states. To get a clean and strong signal the overlap with the ground state can be improved by going over to more physical wave functions than for example point sources would lead to. We used point source when calculating the quark propagator. As hadrons are not point objects this is not an ideal thing to do. To achieve a good signal we would like to have a very good overlap with the meson (or baryon) wavefunction. However as we do not know the hadron wavefunction this is not possible. The amplitudes in the correlation functions might be very small when using point sources. To help this situation we shall employ an improvement: We applying the smearing algorithm

$$S^{n_0} = M S_0^{n_0} \tag{4.122}$$

with

$$M = \sum_{n=0}^{N_J} \kappa_J^n H_J^n \tag{4.123}$$

and H_J being the spatial part of the Wilson term

$$H_J(\mathbf{n},\mathbf{m}) = \sum_{j=1}^{3} \left(U_j(\mathbf{n}, n_t)\delta(\mathbf{n}+\hat{j},\mathbf{m}) + U_j(\mathbf{n}-\hat{j}, n_t)^\dagger \delta(\mathbf{n}-\hat{j},\mathbf{m}) \right). \tag{4.124}$$

This smearing prescription is known as *Jacobi smearing*. It has two degrees of freedom, i.e. the positive real parameter κ_J which controls the coarseness of the iteration, and the number of smearing iterations N_J which increases the size of the smeared object roughly like a random walk. Physically we wish to smear until our source occupies a reasonable fraction of the size of the hadron. A suitable measure of the root mean square (rms) radius is given by

$$r_{\text{rms}} = \frac{\sum\limits_{\mathbf{n}\in\Lambda_3} |a\mathbf{n}|^2 |S^{n_0}(\mathbf{n})|^2}{\sum\limits_{\mathbf{n}\in\Lambda_3} |S^{n_0}(\mathbf{n})|^2}. \tag{4.125}$$

Typical values for the rms radius are in the range of $0.25\text{fm} \leq r_{\text{rms}} \leq 0.45\text{fm}$.

4.15 Two-Point Correlation Functions

The calculation of the correlation functions on the quark level was carried out in Sec. 4.11. We now turn to the correlation functions using the Hamilton formalism which allows us to extract matrix elements and energy levels.

The Euclidean correlation function of two operators O_1 and O_2 is defined as

$$\langle O_2(t) O_1(0) \rangle_T = \frac{1}{Z_T} \text{tr}\left[e^{-(T-t)\widehat{H}} \widehat{O}_2 e^{-t\widehat{H}} \widehat{O}_1 \right] \tag{4.126}$$

with the normalisation factor

$$Z_T = \text{tr}\left[e^{-T\widehat{H}} \right] \tag{4.127}$$

and $\widehat{H}$ being the Hamiltonian of the system. The time t is the actual time distance of interest and T is a formal maximal distance, which will eventually be taken to infinity. Using the defining relation of the trace of an operator

$$\text{tr}[\widehat{O}] = \sum_n \langle n|\widehat{O}|n\rangle \tag{4.128}$$

with a natural choice of eigenstates $|n\rangle$ of $\widehat{H}$ which satisfy the eigenvalue equation

$$\widehat{H}|n\rangle = E_n|n\rangle \tag{4.129}$$

and inserting the unity operator as a complete set of states

$$1 = \sum_n |n\rangle\langle n| \tag{4.130}$$

and expanding the exponential function in the denominator we arrive at the Euclidean correlation function

$$\langle O_2(t)O_1(0)\rangle_T = \frac{\sum_{m,n}\langle m|\widehat{O}_2|n\rangle\langle n|\widehat{O}_1|m\rangle e^{-t\Delta E_n}e^{-(T-t)\Delta E_m}}{1+e^{-T\Delta E_1}+e^{-T\Delta E_2}+\ldots} \tag{4.131}$$

where we have factored out e^{-TE_0} with E_0 denoting the energy eigenvalue of the vacuum $|0\rangle$ and introduced

$$\Delta E_n = E_n - E_0. \tag{4.132}$$

Thus all involved energies are energy differences to the vacuum energy and only these differences can be measured in experiment. For convenience we now rename ΔE_n to E_n. From this expression we see that matrix elements as well as the energy values of the system are accessible and thus can be calculated. By fitting the amplitudes we can determine the matrix elements and by fitting the exponentials we can determine the energy states of the system.

Even though we might only be interested in the ground state energy of the system, our measurements also include excited states. Assume we want to compute the energy levels of a hadronic system with an operator that creates a proton from the vacuum $\widehat{O}_p(0)$ and the operator $\widehat{O}_p^\dagger(t)$ annihilates the proton at a later time t. The states $\langle p|$ with quantum numbers corresponding to the proton have non-vanishing overlap with the proton operator. But also excited states $\langle p'|$ of the proton will have non-vanishing overlap. The total contribution is

$$\lim_{T\to\infty}\langle O_p(t)O_p^\dagger(0)\rangle_T = |\langle p|\widehat{O}_p^\dagger|0\rangle|^2 e^{-tE_p} + |\langle p'|\widehat{O}_p^\dagger|0\rangle|^2 e^{-tE_{p'}} + \ldots. \tag{4.133}$$

The energies of the excited states are larger than the energy of the ground state

$$E_{p'} > E_p. \tag{4.134}$$

Thus we can extract the ground state if we go to a sufficiently large time t where the exited states get exponentially suppressed. In order to extract the matrix elements and energy levels from the nominator the denominator of Eq. (4.131) should be roughly unity. To accomplish this we choose the temporal extent of the lattice large in comparison to the time t

$$0 \ll t \ll T. \tag{4.135}$$

Due to the finite extent and the periodicity of the lattice the states are also propagating in negative time direction, e.g. the anti-particle of the pions propagate in negative time direction. In case of the nucleon it is its parity partner with a different mass that propagates in negative time direction.

For the meson correlation function we find for the ground state

$$\begin{aligned} C_{\Gamma_1\Gamma_2}(\mathbf{p},t) &= \frac{1}{2E_\mathbf{p}}\langle 0|\widehat{M}_{\Gamma_2}|M(\mathbf{p})\rangle\langle M(\mathbf{p})|\widehat{M}_{\Gamma_1}|0\rangle\left[e^{-E_\mathbf{p}t}+\tau_{\Gamma_1}\tau_{\Gamma_2}e^{-E_\mathbf{p}(T-t)}\right] && (4.136)\\ &= A_{\Gamma_1\Gamma_2}\left[e^{-E_\mathbf{p}t}+\tau_{\Gamma_1}\tau_{\Gamma_2}e^{-E_\mathbf{p}(T-t)}\right] && (4.137)\end{aligned}$$

where the τ factors determine how the meson operator behaves under time reversal, i.e. whether the two-point function is symmetric or antisymmetric with respect to $t \to T-t$, and are given by $\gamma_4\Gamma^\dagger\gamma_4 = \tau\Gamma$. Possible values are $\tau_{\Gamma_1}, \tau_{\Gamma_1} = \pm 1$.

With the baryon interpolators in Eq. (4.65) and (4.66) we find for the baryon correlation function

$$\begin{aligned} C_\Gamma(\mathbf{p},t) =& \frac{\sqrt{Z\overline{Z}}}{2E_\mathbf{p}}\sum_\mathbf{s}\left[\overline{u}(\mathbf{p},\mathbf{s})\Gamma u(\mathbf{p},\mathbf{s})e^{-E_\mathbf{p}t}+\overline{v}(\mathbf{p},\mathbf{s})\Gamma v(\mathbf{p},\mathbf{s})e^{-E_\mathbf{p}(T-t)}\right]+\\ &\frac{\sqrt{Z'\overline{Z}'}}{2E'_\mathbf{p}}\sum_\mathbf{s}\left[\overline{v}'(\mathbf{p},\mathbf{s})\Gamma v'(\mathbf{p},\mathbf{s})e^{-E'_\mathbf{p}t}+\overline{u}'(\mathbf{p},\mathbf{s})\Gamma u'(\mathbf{p},\mathbf{s})e^{-E'_\mathbf{p}(T-t)}\right] \end{aligned} \qquad (4.138)$$

where we defined the overlaps

$$\begin{aligned} \langle 0|B_\alpha(0)|N(\mathbf{p},\mathbf{s})\rangle &= \sqrt{Z}u_\alpha(\mathbf{p},\mathbf{s}) && (4.139)\\ \langle 0|\overline{B}_\alpha(0)|N(\mathbf{p},\mathbf{s})\rangle &= \sqrt{\overline{Z}}\overline{v}_\alpha(\mathbf{p},\mathbf{s}) && (4.140)\end{aligned}$$

and

$$\begin{aligned} \langle N(\mathbf{p},\mathbf{s})|B_\alpha(0)|0\rangle &= \sqrt{Z}v_\alpha(\mathbf{p},\mathbf{s}) && (4.141)\\ \langle N(\mathbf{p},\mathbf{s})|\overline{B}_\alpha(0)|0\rangle &= \sqrt{\overline{Z}}\overline{u}_\alpha(\mathbf{p},\mathbf{s}). && (4.142)\end{aligned}$$

The first term of Eq. (4.138) corresponds to the nucleon $J^P = \frac{1}{2}^+$ state with energy $E_\mathbf{p}$. The anti-particle of the nucleon has negative parity, $P=-1$, so the second term must accord to an excited state, i.e. the anti-particle of which has positiv parity $P=+1$. The lowest parity partner, i.e. $J^P = \frac{1}{2}^-$, is nearly mass degenerate with the nucleon and so we must keep this additional state with energy $E'_\mathbf{p}$ as well. To suppress the unwanted negative parity states we choose the projection operator Γ for an unpolarised stationary nucleon as

$$\Gamma_u = \begin{cases} \frac{1}{2}(1+\gamma_4) & \text{for} \quad 0 \ll t \ll T/2 \\ \frac{1}{2}(1-\gamma_4) & \text{for} \quad T/2 \ll t \ll T\ . \end{cases} \tag{4.143}$$

Using the sum rules over spinors we arrive at

$$C_{\frac{1}{2}(1+\gamma_4)}(\mathbf{p},t) = \sqrt{Z\overline{Z}}\left(\frac{E_\mathbf{p}+m}{E_\mathbf{p}}\right)e^{-E_\mathbf{p}t}, \qquad 0 \ll t \ll T/2 \tag{4.144}$$

and

$$C_{\frac{1}{2}(1-\gamma_4)}(\mathbf{p},t) = -\sqrt{Z\overline{Z}}\left(\frac{E_\mathbf{p}+m}{E_\mathbf{p}}\right)e^{-E_\mathbf{p}(T-t)}, \qquad T/2 \ll t \ll T. \tag{4.145}$$

4.16 Pion Decay Constant

The pion decay constant in Eq. (3.5) can be determined from two-point correlation functions on the lattice. In Euclidean space at zero three-momentum we define

$$\langle 0|A_4^{(r)a}|\pi\rangle = m_\pi f_\pi \tag{4.146}$$

with the renormalised and improved operator $A_\mu^{(r)a}(n)$ defined in Eq. (4.93). By computing the correlation function $C_{A_4P}^{LS}$ and C_{PP}^{SS}, see Eq. (4.137), where we use the notation (S) for a smeared operator and (L) for a local operator we find the matrix element of A_4 to be

$$\langle 0|A_4^{(r)a}|\pi\rangle = -\frac{\sqrt{2m_\pi}C_{A_4P}^{LS}}{\sqrt{C_{PP}^{SS}}} \times 2\kappa \tag{4.147}$$

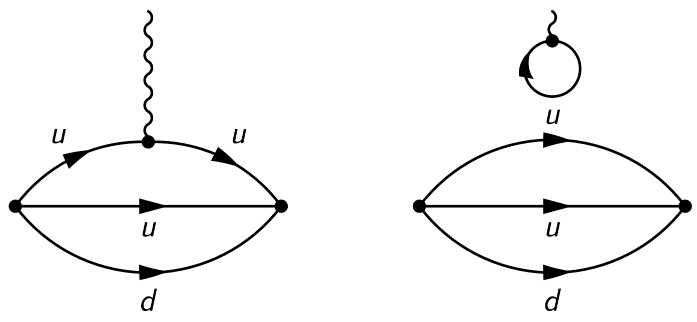
and for the matrix element of the improvement term we obtain from the correlation function C_{PP}^{LS} and $C_{A_4P}^{LS}$

$$\frac{\langle 0|a\partial_4 P|\pi\rangle}{\langle 0|A_4|\pi\rangle} = \sinh am_\pi \frac{C_{PP}^{LS}}{C_{A_4P}^{LS}}. \tag{4.148}$$

4.17 Three-Point Correlation Functions

Matrix elements can be calculated with the help of three-point functions. We restrict ourself to baryon three-point functions

$$C_\Gamma(t,\tau,\mathbf{p},\mathbf{q},O) = \sum_{\alpha\beta}\Gamma_{\beta\alpha}\langle B_\alpha(t,\mathbf{p}')|O(\tau,\mathbf{q})|\overline{B}_\beta(0,\mathbf{p})\rangle \tag{4.149}$$



(a) Connected diagram **(b)** Disconnected diagram

Figure 4.6 – The three-point correlation function for baryons.

with the momentum transfer

$$q = p' - p. \tag{4.150}$$

The operator O is given by

$$O(\tau, \mathbf{q}) = \sum_{\substack{\mathbf{z}\in\Lambda_3\\ v,w\in\Lambda}} e^{ia\mathbf{q}\cdot\mathbf{z}}\overline{\psi}^a_\gamma(v)O^{ab}_{\gamma\delta}(v,w,\mathbf{z},\tau)\psi^b_\delta(w) \tag{4.151}$$

where we sum over the spatial plane $\mathbf{z}$ only, and the whole space v, w. After applying Wick's theorem, i.e. carrying out the quark contractions into quark propagators and a bit of algebra [64] we are left with a *quark line connected* and a *quark line disconnected* term. Here we consider the quark line connected term only. Fig. 4.6a depicts the connected term of the baryon three-point function whereas Fig. 4.6b depicts the disconnected term. The correlation function finally reads

$$C^\psi_\Gamma(t,\tau,\mathbf{p},\mathbf{q},O) = -V_3\sum_{\mathbf{y},v,w} e^{ia\mathbf{q}\cdot\mathbf{y}}\langle \mathrm{tr}_{DC}[\Sigma^\psi_\Gamma((0|v),\mathbf{p},t)O^\psi(v,w,\mathbf{y},\tau)S^\psi(w|0)]\rangle_G \tag{4.152}$$

where we introduced the short notation

$$S = D^{-1} \tag{4.153}$$

with the sequential propagator

$$\Sigma^\psi_\Gamma((0|v),\mathbf{p},t) = \sum_{\mathbf{x}} S^\psi_\Gamma((\mathbf{x},t|0),\mathbf{p})S^\psi((\mathbf{x},t)|v) \tag{4.154}$$

with the part for the u quark

$$\begin{aligned} S^{(u)a'a}_\Gamma((x|0),\mathbf{p}) =& e^{-ia\mathbf{p}\cdot\mathbf{x}}\epsilon_{abc}\epsilon_{a'b'c'}\times \\ & \Big[\widetilde{S}^{(d)bb'}(x|0)S^{(u)cc'}(x|0)\Gamma + \mathrm{tr}_D[\widetilde{S}^{(d)bb'}(x|0)S^{(u)cc'}(x|0)]\Gamma + \\ & \Gamma S^{(u)bb'}(x|0)\widetilde{S}^{(d)cc'}(x|0) + \mathrm{tr}_D[\Gamma S^{(u)bb'}(x|0)]\widetilde{S}^{(d)cc'}(x|0)\Big] \end{aligned} \tag{4.155}$$

and for the d quark

$$\begin{aligned} S_\Gamma^{(d)a'a}((x|0),\mathbf{p}) =& e^{-ia\mathbf{p}\cdot\mathbf{x}}\epsilon_{abc}\epsilon_{a'b'c'}\times \\ & \left[\widetilde{S}^{(u)bb'}(x|0)\widetilde{\Gamma}\widetilde{S}^{(u)cc'}(x|0)+\mathrm{tr}_D[\Gamma S^{(u)bb'}(x|0)\widetilde{S}^{(u)cc'}(x|0)]\right] \end{aligned} \tag{4.156}$$

with the tilde operating in Dirac space

$$\widetilde{X}=(C\gamma_5 X\gamma_5 C)^T. \tag{4.157}$$

The sequential propagator can be computed by an additional inversion of the Dirac operator D for each choice of the final momentum $\mathbf{p}$

$$\sum_{v\in V} D(v'|v)\gamma_5\Sigma_\Gamma^{(\psi)^\dagger}((0,v),\mathbf{p},t)=\gamma_5 S_\Gamma^{(\psi)^\dagger}((v',t|0),\mathbf{p})\delta_{v_0',t}. \tag{4.158}$$

Calculating three point functions is a two step procedure: First, the calculation of the quark propagator from point 0 to any point x. Then a second inversion is made with the source given in Eq. (4.155) and (4.156) depending whether the inserted operator consists of u or d quarks.

A change of the sink properties requires the computation of new sequential propagators, and so simulating several final momenta, different field interpolators, or applying different smearing procedures for the sink rapidly becomes rather expensive. The advantage of this approach, however, is that it allows also for the insertion of different operators, and thus the calculation of several matrix elements and also any momentum insertion – ideal for moments of PDFs and form factors.

4.17.1 Matrix Elements

In order to extract the matrix elements one considers ratios of three-point to two-point functions. At time τ we insert an operator $O(\tau)$ into the three-point baryon correlator and insert at time τ a complete set of states. For a large enough separation distance only the ground state of the baryons contribute

$$\begin{aligned} &\langle 0|\widehat{B}(t,\mathbf{p}')O(\tau)\widehat{\overline{B}}(0,\mathbf{p})|0\rangle \\ =&\langle 0|B(t,\mathbf{p}')|N(\mathbf{p}')\rangle\frac{e^{-E_{p'}(t-\tau)}}{2E_{p'}}\langle N(\mathbf{p}')|O(\tau)|N(\mathbf{p})\rangle\frac{e^{-E_p\tau}}{2E_p}\langle N(\mathbf{p})|\overline{B}(0,\mathbf{p})|0\rangle \\ =&\sqrt{Z(\mathbf{p}')\overline{Z}(\mathbf{p})}F(\Gamma,\mathcal{J})e^{-E_{\mathbf{p}'}(t-\tau)}e^{-E_{\mathbf{p}}\tau} \end{aligned} \tag{4.159}$$

where we used the overlaps defined in Eqs. (4.139) to (4.142) and with

$$\langle N, \mathbf{p}', \mathbf{s}'|O|N, \mathbf{p}, \mathbf{s}\rangle = \overline{u}(\mathbf{p}', \mathbf{s}')\mathcal{J}_O(\mathbf{q})u(\mathbf{p}, \mathbf{s}) \tag{4.160}$$

we find

$$F(\Gamma, \mathcal{J}) = \frac{1}{4}\mathrm{tr}_D\Gamma\big(\gamma_4 - i\frac{\mathbf{p}'\cdot\gamma}{E_{\mathbf{p}'}} + \frac{m}{E_{p'}}\big)\mathcal{J}\big(\gamma_4 - i\frac{\mathbf{p}\cdot\gamma}{E_{\mathbf{p}}} + \frac{m}{E_{\mathbf{p}}}\big) \tag{4.161}$$

and for the two-point function we find

$$\langle 0|B(t, \mathbf{p})\overline{B}(0, \mathbf{p})|0\rangle = \langle 0|B(t, \mathbf{p})|N(\mathbf{p})\rangle\frac{e^{-E_p t}}{2E_p}\langle N(\mathbf{p})|\overline{B}(0, \mathbf{p})|0\rangle \tag{4.162}$$

where we neglected the periodicity of the lattice. When taking the ratio the unknown factors cancel

$$\begin{aligned} R(t, \tau, \mathbf{p}, \mathbf{q}) &= \frac{C_\Gamma(t, \tau, \mathbf{p}, \mathbf{q}, O)}{C_{\Gamma_u}(t, \mathbf{p})} && (4.163)\\ &= \frac{E_\mathbf{p}}{E_\mathbf{p} + m}F(\Gamma, \mathcal{J}_O(0)), \qquad 0 \ll \tau \ll t \ll \frac{1}{2}T && (4.164)\\ &\propto \langle N|O|N\rangle. && (4.165) \end{aligned}$$

with the three-point correlation function $C_\Gamma(t, \tau, \mathbf{p}, \mathbf{q})$ as defined in Eq. (4.152). The relation in Eq. (4.163) is only valid for zero momentum transfer $q = 0$. For the gamma matrix in the unpolarised case we take $\Gamma_u = P_+$ as defined in Eq. (4.68).

In order to determine the matrix element we seek for a region $0 \ll \tau \ll t \ll \frac{1}{2}T$ where the ratio is constant.

Chapter 5

Preparing the Simulations

5.1 The SLiNC Action

The QCDSF Collaboration carries out investigations of baryon structure using configurations generated with $N_f = 2+1$ dynamical flavours of $\mathcal{O}(a)$-improved Wilson fermions. With the strange quark mass as an additional dynamical degree of freedom in the simulations needs are avoided for a partially quenched approximation when investigating the properties of particles containing a strange quark, e.g. the hyperons.

The fermion action elaborated is the $N_f = 2+1$ flavour Stout Link Non-perturbative Clover (SLiNC) fermion action with non-perturbative $O(a)$ improvement [65]

$$\begin{aligned} S_F = \sum_{n\in\Lambda} \Bigg[\kappa \sum_{\mu} \Bigg(& \overline{q}(n)(\gamma_\mu - 1)\widetilde{U}_\mu(n) q(n+\hat{\mu}) \\ & - \overline{q}(n)(\gamma_\mu + 1)\widetilde{U}^\dagger_\mu(n-\hat{\mu}) q(n-\hat{\mu}) \Bigg) \\ & + \overline{q}(n)q(n) - \frac{1}{2}\kappa a c_{sw} \sum_{\mu\nu} \overline{q}(n)\sigma_{\mu\nu}F_{\mu\nu}(n)q(n) \Bigg] \end{aligned} \tag{5.1}$$

with

$$\sigma_{\mu\nu} = [\gamma_\mu, \gamma_\nu] \tag{5.2}$$

and $F_{\mu\nu}$ as defined in Eq. (4.88).

The Dirac kinetic term and Wilson mass term in Eq. (5.1) employ one level of stout smeared links, so-called fat links, which we denote by $\widetilde{U}$ and introduce below in Sec. 5.2. Smearing helps at present lattice spacings by smoothing out fluctuations in the gauge fields slightly.

For the gluonic part of the action we utilise the Symanzik tree-level gluon action

$$\begin{aligned} S_G = \frac{6}{g_0^2} \Bigg[c_0 \sum_{\text{plaquette}} \frac{1}{3} \mathrm{Re}\,\mathrm{Tr}(1 - U_{\text{plaquette}}) + \\ c_1 \sum_{\text{rectangle}} \frac{1}{3} \mathrm{Re}\,\mathrm{Tr}(1 - U_{\text{rectangle}}) \Bigg] \end{aligned} \tag{5.3}$$

with

$$c_0 = 20/12, \quad c_1 = -1/12, \text{and} \quad \beta = 10/g_0^2. \tag{5.4}$$

The first sum is carried out over all plaquettes as introduced in Sec. 4.3. The second sum runs over all rectangles. The rectangles are built in the same fashion as the plaquettes, but extended to consist of the outline of two neighbouring plaquettes in the same plane.

The standard action as proposed by Wilson is obtained by setting $c_0 = 1$ and $c_1 = 0$. However, to avoid a nearby first-order phase transition when using the standard Wilson action, as pointed out in literature [66], we use the above mentioned values for c_0 and c_1.

5.2 Stout Link Smearing

A method of smearing link variables which is analytic, and hence differentiable, everywhere in the finite complex plane can be defined as follows [67]. Let $C_\mu(n)$ denote the sum of the perpendicular staples which have one end point located at lattice site n and the other at the neighbouring site $n + \hat{\mu}$

$$C_\mu(n) = \sum_{\nu \neq \mu} \left(U_\nu(n) U_\mu(n+\hat{\nu}) U_\nu^\dagger(n+\hat{\mu}) + U_\nu^\dagger(n-\hat{\nu}) U_\mu(n-\hat{\nu}) U_\nu(n-\hat{\nu}+\hat{\mu}) \right). \tag{5.5}$$

The matrix

$$Q_\mu(n) = \frac{i}{2} \left(\Omega_\mu^\dagger(n) - \Omega_\mu(n) \right) - \frac{i}{2N_c} \mathrm{tr} \left(\Omega_\mu^\dagger(n) - \Omega_\mu(n) \right) \tag{5.6}$$

with

$$\Omega_\mu(n) = C_\mu U_\mu^\dagger(n), \tag{5.7}$$

where no summation over μ is meant here, is hermitian and traceless. Thus the matrix $\exp[iQ_\mu(n)]$ is an element of the $\mathrm{SU}(N_c)$ group. The link variable after an applied smearing step is defined as

$$U_\mu^{(k+1)}(n) = \exp[iQ_\mu(n)] U_\mu^k(n) \tag{5.8}$$

which is also an element of the gauge group.

5.3 Tuning the Mass

Our simulations include $N_f = 2 + 1$ flavours of dynamical quarks. By $2 + 1$ flavours we mean here two mass degenerate up-down, $m_l^{(r)}$, light quarks and one strange, $m_s^{(r)}$, quark. Simulating at the physical masses $m_l^{(r)*}$ and $m_s^{(r)*}$ is computationally very difficult (the superscript * ($^{(r)}$) denotes physical (renormalised) quantities). We simulate at larger masses and extrapolate to the physical masses.

We shall now introduce our strategy for choosing the paths in the $m_l^{(r)}$-$m_s^{(r)}$ plane for carrying out the simulations and how we approach the physical point. A natural starting point for these paths is the flavour SU(3) symmetric point $m_l^{(r)} = m_s^{(r)} = m_{\text{sym}}^{(r)(0)}$ denoted by the superscript ^{0}.

We choose the path in the $m_l^{(r)}$-$m_s^{(r)}$ plane such that the singlet quark mass is kept fixed [68],

$$\overline{m}^{(r)} = \frac{1}{3}(2m_l^{(r)} + m_s^{(r)}) = \text{constant}. \tag{5.9}$$

Following this path both the kaon and η are lighter than their physical values along the entire trajectory, i.e. they both approach their physical mass values from below. One hopes that flavour SU(3) chiral perturbation theory works better since masses are kept small [68].

We extend our measurements beyond the symmetric point with heavy up-down quarks and a lighter strange quark.

5.4 Extrapolating Flavour Singlet Quantities

A flavour singlet quantity $X_S(m_u^{(r)}, m_d^{(r)}, m_s^{(r)})$ is an object that is invariant under quark permutation symmetry between u, d, and s. Flavour singlet quantities are flat at a point on the flavour SU(3) symmetric line and hence allow simpler extrapolations to the physical point. This may be shown by considering small changes about a point on the flavour symmetric line. Taylor expanding X_S about a point on the symmetric line where flavour

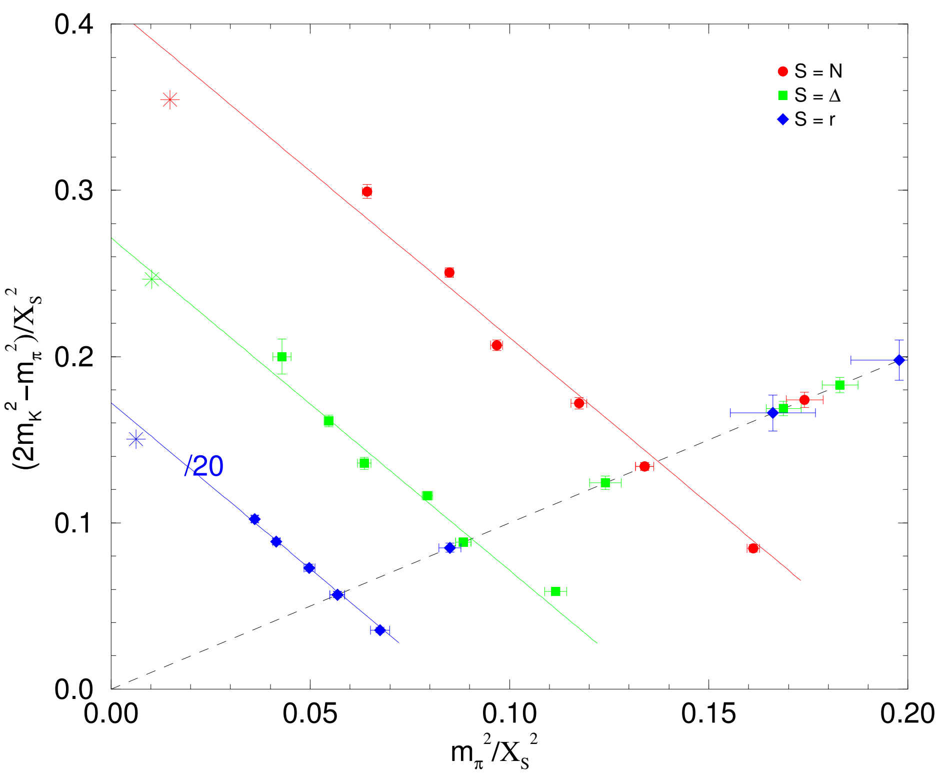


Figure 5.1 – The SU(3) flavour symmetric line ($y = x$) is the dashed line. For convenience the results for $S = r$ have been divided by a factor of 20. The experimental points using the three singlet quantities, X_S , $S = N, \Delta, r$, are shown as stars.

SU(3) holds gives

$$X_S\left(\overline{m}^{(r)(0)} + \delta m_l^{(r)}, \overline{m}^{(r)(0)} + \delta m_l^{(r)}, \overline{m}^{(r)(0)} + \delta m_s^{(r)}\right) = X_{S\,\mathrm{sym}}^{(0)} + \left.\frac{\partial X_S}{\partial m_u^{(r)}}\right|_{\mathrm{sym}}^{(0)} \delta m_l^{(r)} + \left.\frac{\partial X_S}{\partial m_d^{(r)}}\right|_{\mathrm{sym}}^{(0)} \delta m_l^{(r)} + \left.\frac{\partial X_S}{\partial m_s^{(r)}}\right|_{\mathrm{sym}}^{(0)} \delta m_s^{(r)} + \mathcal{O}((\delta m_q^{(r)})^2). \tag{5.10}$$

But on the symmetric line we have

$$\left.\frac{\partial X_S}{\partial m_u^{(r)}}\right|_{\mathrm{sym}} = \left.\frac{\partial X_S}{\partial m_d^{(r)}}\right|_{\mathrm{sym}} = \left.\frac{\partial X_S}{\partial m_s^{(r)}}\right|_{\mathrm{sym}}, \tag{5.11}$$

and on our chosen trajectory $\overline{m}^{(r)} = \text{constant}$,

$$2\delta m_l^{(r)} + \delta m_s^{(r)} = 0, \tag{5.12}$$

which together imply that

$$X_S\left(\overline{m}^{(r)(0)} + \delta m_l^{(r)}, \overline{m}^{(r)(0)} + \delta m_l^{(r)}, \overline{m}^{(r)(0)} + \delta m_s^{(r)}\right) = X_{S\,\mathrm{sym}}^{(0)} + \mathcal{O}((\delta m_q^{(r)})^2). \tag{5.13}$$

Thus, the effect at first order of changing the strange quark mass is cancelled by the change in the light quark mass, so X_S has a stationary point on the flavour SU(3) symmetric line. This result can be verified by considering leading order (LO) together with next to leading order (NLO) flavour SU(3) chiral perturbation theory ChPT [68].

For X_S we take the centre of mass of the baryon octet or decuplet [10],

$$X_N = \frac{1}{3}(m_N + m_\Sigma + m_\Xi) = 1.150\,\text{GeV} \tag{5.14}$$

$$X_\Delta = \frac{1}{3}(2m_\Delta + m_\Omega) = 1.379\,\text{GeV}. \tag{5.15}$$

In order to determine the starting point on the symmetric line, we relate the known physical point to the initial point via

$$\left.\frac{\frac{1}{3}(2m_K^2 + m_\pi^2)}{X_S^2}\right|^{*} = \left.\frac{m_\pi^2}{X_S^2}\right|^{(0)}_{\text{sym}} \tag{5.16}$$

where we choose $S = N, \Delta$ respectively. Simulations along the flavour symmetric line and using Eq. (5.16) are sufficient to determine the initial point.

Fig. 5.1 shows the paths in the $m_l^{(r)}$-$m_s^{(r)}$ plane for carrying out the simulations and how we approach the physical point. The dashed flavour SU(3) symmetric line is the starting point of the path towards the physical point.

5.5 Clover Fermions

The singlet (S) and non-singlet (NS) quark masses renormalise differently if the fermionic action is not invariant under chiral symmetry. The here applied clover $\mathcal{O}(a)$ improved Wilson fermion action does not exhibit chiral symmetry. For the renormalised quark mass we find [69]

$$m_q^{(r)} = Z_m^{NS}(m_q - \overline{m}) + Z_m^S\overline{m} \tag{5.17}$$

$$= Z_m^{NS}(m_q + \alpha_Z\overline{m}) \tag{5.18}$$

where $q = l, s$ and

$$\alpha_Z = \frac{Z_m^S - Z_m^{NS}}{Z_m^{NS}}. \tag{5.19}$$

κ_l	κ_s	$m_\pi[GeV]$	$m_K[GeV]$	$N \times N_T$	$m_\pi L$	N_{meas}
0.120830	0.121040	0.481	0.420	24x48	4.63	2500
0.120900	0.120900	0.443	0.443	24x48	4.28	4000
0.120950	0.120800	0.414	0.459	24x48	3.99	2500
0.121000	0.120700	0.377	0.473	24x48	3.63	2500
0.121040	0.120620	0.350	0.485	24x48	3.37	2500

Table 5.1 – Simulation parameters for $N_f = 2+1$ dynamical fermions with two mass-degenerate light quarks and one strange quark. The simulation parameter β was chosen to $\beta = 5.50$ which corresponds to a lattice spacing of $a = 0.083(3)$fm.

The bare quark mass was defined in Eq. 4.37 where $\kappa_{\text{sym},c}$ is defined by the vanishing of the quark mass along the symmetric line. From LO ChPT we find

$$\frac{1}{3}\left(2(am_K)^2 + (am\pi)^2\right) \propto \frac{2}{9}(1+\alpha_Z)a\overline{m}. \tag{5.20}$$

With $a\overline{m} =$ constant we find

$$\kappa_s = \frac{1}{\frac{3}{\kappa_{\text{sym}}^{(0)}} - \frac{2}{\kappa_l}} \tag{5.21}$$

where $\kappa_{\text{sym}}^{(0)}$ is the appropriate κ on the flavour SU(3) symmetric line. Tuning the mass is not trivial and one might miss the physical point.

Now, given κ_l we can find the corresponding κ_s. After some experimentation we choose the κ_l and κ_s as shown in Tab. 5.1. Note that it is possible to choose the κ_l and κ_s values such that $m_l > m_s$. In this strange world, we expect to see an inversion of the particle spectrum, with for example the nucleon being the heaviest octet particle.

5.6 Lattice Spacing

We determine the lattice spacing a utilising the relation

$$a = aX_S/X_S^{\text{exp}} \tag{5.22}$$

where X_S^{exp} is the experimental value and aX_S is the quantity measured on the lattice. One would hope that whatever scale is used, i.e. $S \in \{N, \Delta, \rho, r_0, \pi\}$, to get the same lattice

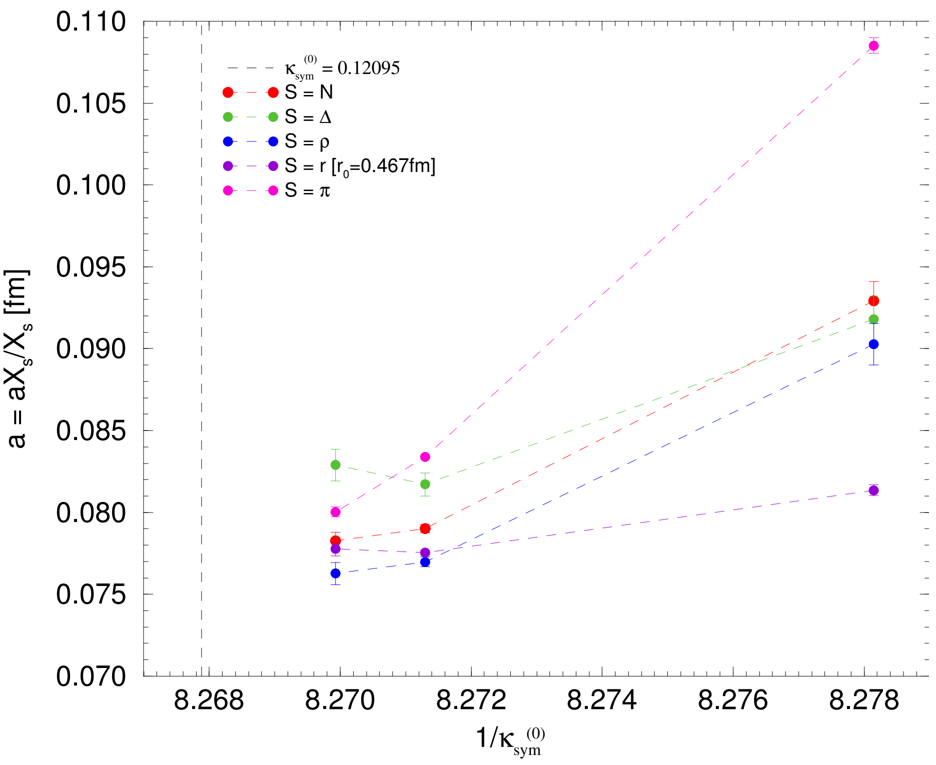


Figure 5.2 – Determining the lattice spacing a with the relation $a = aX_S/X_S^{\text{exp}}$. The gluonic quantity $X_r = 0.467$ fm is the QCDSF $N_f = 2$ result. From phenomenology one finds $r_0 = 0.5$ fm.

spacing. Fig. 5.2 shows a determination of the lattice spacing a using some different X_S where only the largest available volumes were used. The data points should fall onto the same curve, but there is some variation. This variation might come from finite size effects.

When using X_N, i.e. the centre of mass of the baryon octet, this results at the symmetric point $\kappa_{\text{symm}} = 0.12090$ in a lattice spacing $a = 0.083(3)$. But another X_S would result in a little different lattice spacing. Given the uncertainties one somehow has to make a choice and we use $a = 0.083(3)$ for the further analysis.

Chapter 6

Discussion of Errors

Numerical simulations of QCD include unavoidable statistical and systematic errors. In this chapter we briefly introduce the different types of errors one is confronted with in Lattice QCD simulations.

6.1 Statistical Errors

In Lattice QCD, observables are calculated by means of Monte Carlo integration. The expectation value (central value) of an observable is calculated by taking the mean value of the observable calculated over the whole set of gauge configurations in an ensemble. We consider a sample of N_{meas} measurements X_1, X_2, ..., $X_{N_{\text{meas}}}$ of a quantity X. The expectation value (central value) is defined as

$$\langle X \rangle_{N_{\text{meas}}} = \frac{1}{N_{\text{meas}}} \sum_{i=1}^{N_{\text{meas}}} X_i. \tag{6.1}$$

If the configurations are statistically independent, we expect the statistical error of the full ensemble to be proportional to $1/\sqrt{N_{\text{meas}}}$.

6.2 Autocorrelation Times

This section is included to be complete on the description of errors one faces in Lattice QCD – we did not carry out a study of autocorrelation times.

In Monte Carlo simulations the configurations are correlated. It is convenient to analyse the autocorrelation in the observables one is interested in. This serves two purposes: First, the exponential autocorrelation time is related to the length of the thermalisation phase of the Markov chain, i.e. the time that the system needs to converge from its initial state to equilibrium. In order to avoid systematic errors in the final results due to an initialisation bias one should discard the data from the first measurements. Second, if we consider an observable X, a run of length T_{MC} (in Monte Carlo time measured in trajectories), contains only $T_{\text{MC}}/2\tau_{\text{int}}^X$ effectively independent data points with τ_{int} the so-called integrated autocorrelation time.

A suitable estimator of the true autocorrelation function for a finite time-series X_t, $t = 1, \dots, T_{\text{MC}}$, is given by

$$C^X(t) = \frac{1}{T_{\text{MC}} - 1} \sum_{s=1}^{T_{\text{MC}}-t} (X_s - \langle X \rangle_L)(X_{s+t} - \langle X \rangle_R), \tag{6.2}$$

where the "left" and "right" mean-value estimators are defined as

$$\langle X \rangle_L = \frac{1}{T_{\text{MC}} - t} \sum_{r=1}^{T_{\text{MC}}-t} X_r \tag{6.3}$$

$$\langle X \rangle_R = \frac{1}{T_{\text{MC}} - t} \sum_{r=1}^{T_{\text{MC}}-t} X_{r+t} \tag{6.4}$$

with the normalised autocorrelation function

$$\rho^X(t) = C^X(t)/C^X(0) \tag{6.5}$$

one can determine the exponential autocorrelation times τ_{exp}^X with a fit to

$$\tau_{\text{exp}}^X = \frac{t}{-\log|\rho^X(t)|} \qquad t \to \infty. \tag{6.6}$$

The integrated autocorrelation times can be measure with the help of Sokal's windowing procedure [70] and can be estimated with

$$\tau_{\text{int}}^X = \frac{1}{2} + \sum_{t=1}^{T_{\text{cut}}} \rho^X(t) \tag{6.7}$$

with the variable cut-off T_{cut}.

6.3 Binning

Using binning the integrated autocorrelation time can also be estimated via the variance ratio. We bin the time series into $N_{bs} \leq N$ bins of $N_b = N/N_{bs}$ measurements each. It is convenient to choose the values of N and N_{bs} so that N is a multiple of N_{bs}. The binned data are the averages

$$\langle X \rangle_j^{N_b} = \frac{1}{N_b} \sum_{i=1+(j-1)N_b}^{jN_b} X_i, \qquad \text{for} \quad j = 1, \dots, N_{bs}. \tag{6.8}$$

For $N_b > \tau_{\text{exp}}$ the autocorrelations are essentially reduced to those between nearest neighbour bins and even these approach zero under further increase of the binsize.

6.4 Fitting and Error Determination

In order to determine hadronic observables we have to fit our models to the observed data. Fitting is done by minimising a χ^2 function, where χ^2 is a measure of the deviation of the data from the fit model $f_{a_1 \dots a_M, t}$ with $a_1 \dots a_M$ being the fit parameters to determine. Ideally the measurement data follows a Gaussian distribution and is statistically independent. In this case we expect χ^2 to be roughly equal to the number of degrees of freedom, i.e. $\chi^2/\text{dof} = 1$.

For each of the N_{meas} configurations we calculate correlation functions $y_{i,t}$ with the time slice index t and the configuration index i. We calculate the mean values μ_t and the standard deviation σ_t. One typically defines χ^2 as follows

$$\chi^2_{a1 \dots a_M} = \sum_t \frac{(f_{a_1 \dots a_M, t} - \mu_t)^2}{\sigma_t^2}. \tag{6.9}$$

This definition does not take into account that the measurement data is correlated for different t for the same i. Thus we expect to underestimate χ^2. Introducing the covariance matrix

$$c_{t,t'} = \frac{1}{N(N-1)} \sum_{i=1}^{N} (y_{i,t} - \mu_t)(y_{i,t'} - \mu_{t'}) \tag{6.10}$$

where we set $N = N_{\text{meas}}$ we arrive at the definition

$$\chi^2_{a_1 \dots a_M} = \sum_{t,t'} \frac{(f_{a_1 \dots a_M, t} - \mu_t)(f_{a_1 \dots a_M, t'} - \mu_{t'})}{c_{t,t'}}. \tag{6.11}$$

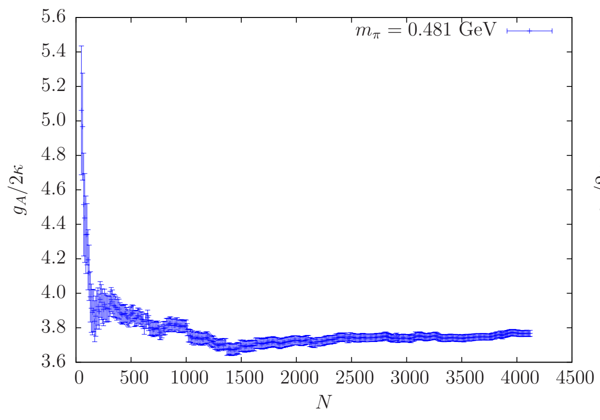
If the data is uncorrelated, $c_{t,t'} = \sigma_t^2 \delta_{t,t'}$. Taking into account the covariance matrix is generally more reliable to estimate χ^2. However, small eigenvalues of this matrix can lead to a not reliable determination of the inverse. Not only this changes the determination of χ^2 but it might lead to incorrect results for the fit parameters $a_1 \dots a_M$. To prevent from this we used singular value decomposition to calculate the inverse [71].

The aforementioned problem occurs when the number of configurations is small. How many configurations are necessary depends on the observable, or, put differently from the signal-to-noise ratio. We carried out the fits with the diagonal covariance matrix as well as with the full covariance matrix. We take deviations of the two results as a measure for the quality of our data. Due to the better stability of the results with the diagonal covariance matrix, we only use these for further analysis.

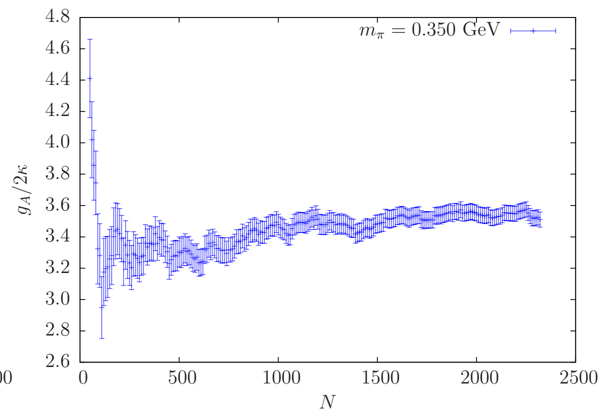
Typically, the signal-to-noise ratio gets worse the farther away from the source we are measuring. However, in the vicinity of the source we have to take into account excited states. To determine the fit interval appropriately we systematically shortened the interval until the fit parameters a_i did not change anymore within statistical errors. Typically we have chosen the fit interval to be symmetric.

Since the fit parameters a_i were obtained using correlated data, the determination of the error for these parameters via χ^2 is unreliable. For the error determination we used the bootstrap method [72]. The bootstrap method foresees to choose N normal-distributed, integer-valued numbers from the interval $1, \dots, N$ and to generate in this way a new ensemble and to apply the fit procedure to the newly created ensemble. By repeating this procedure we get a statistical distribution for the fit parameters a_i. This allows the determination of the standard deviation for each of the parameters.

Typically, we generated 200 bootstrap ensembles. As the central value of a fit parameter a_i we give the fit result using the original ensemble. As the error of the fit parameter we give the standard deviation obtained by the bootstrap method.



(a) At the Symmetric point.



(b) For the lightest pion mass.

Figure 6.1 – Running average plots. g_A for the nucleon (u part).

6.5 Running of the average

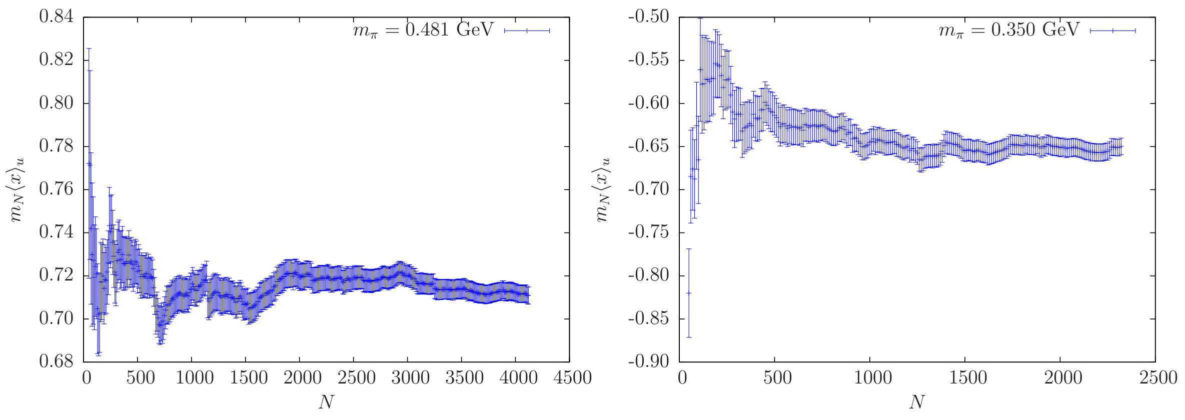
Given our sample of N_{meas} measurements X_1, X_2, ..., $X_{N_{\text{meas}}}$ of a quantity X we carry out the determination of the central value and the error analysis taking into account only the first N measurements where we repeat the analysis for various values of $N \leq N_{\text{meas}}$. This gives us the opportunity to study the convergence behaviour of the quantity.

Fig. 6.1a (6.1b) shows the running average of the axial charge g_A for the nucleon (u part) at the symmetric point (lightest pion mass) as a function of the number of measurements N. Early in Monte Carlo time the value fluctuates heavily indicating that N is not large enough in order to determine the quantity reliably. Towards larger N the fluctuations stay within statistical errors.

Fig. 6.2a (6.1b) shows a similar plot for the running average of the momentum fraction $\langle x \rangle$ for the nucleon (u part) at the symmetric point (lightest pion mass) as a function of the number of measurements N. Again, fluctuations are getting smaller towards larger N.

6.6 Discretisation Effects

We now discuss the systematic errors involved in Lattice QCD. Discretisation effects due to the finite lattice spacing was already discussed in Sec. 5.6.



(a) At the symmetric point. **(b)** For the lightest pion mass.

Figure 6.2 – Running average plot. $\langle x\rangle_u$ for the nucleon (u part).

The tree-level Symanzik gauge action and Wilson Clover fermion action has discretisation errors of order $\mathcal{O}(a^2)$. To determine the discretisation errors one would need to simulate with various values of β leaving all other parameters fixed, e.g. simulating at the same quark masses for different lattice spacings a, i.e. different values of β, and then extrapolate to the continuum limit $a \to 0$.

All quantities, i.e. masses, decay constants, etc. are subject to discretisation effects. Ali Khan et al. studied the nucleon axial charge g_A^N with Wilson Clover fermions with $N_f = 2$ dynamical flavours with various lattice sizes [73]. They observed that the "large volume" results for nucleon axial charge g_A^N obtained at the smallest quark masses for all studied β values lie very close together indicating that discretisation effects are small for this quantity. Also, recent work with $N_f = 2$ simulations found discretisation effects to be small for the nucleon axial charge g_A^N [74].

6.7 Finite Size Effects

In any numerical lattice simulation, the lattice volume is necessarily finite. In order to control potential finite size effects (FSE) on measurable quantities such as masses or decay constants one either has to ensure that they are negligible by making the lattice large enough, or eliminate them by extrapolation to the infinite volume. In either case one has to compare results from different lattice volumes, with all other parameters kept fixed.

In a lattice simulation the physical box-size can be enlarged either by increasing the number of spatial lattice sites at fixed lattice spacing a, or by increasing a for fixed N, N_T. This, however, also increases the associated discretisation errors.

In order to keep discretisation effects and FSE small the two requirements of sufficiently large spatial extend La and small lattice spacing a read

$$a \ll 1/m_\pi \ll L. \tag{6.12}$$

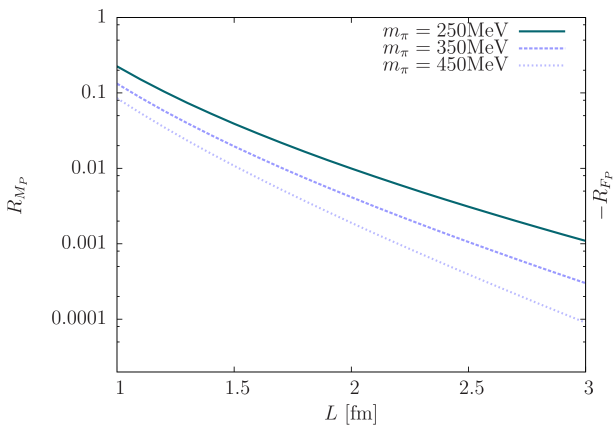
At this stage of discussion we used a single volume to calculate the moments of PDFs. A check for finite size effects is a next step in the discussion. However, we show results for the pion decay constant for two volumes.

6.8 Chiral Effective Field Theory

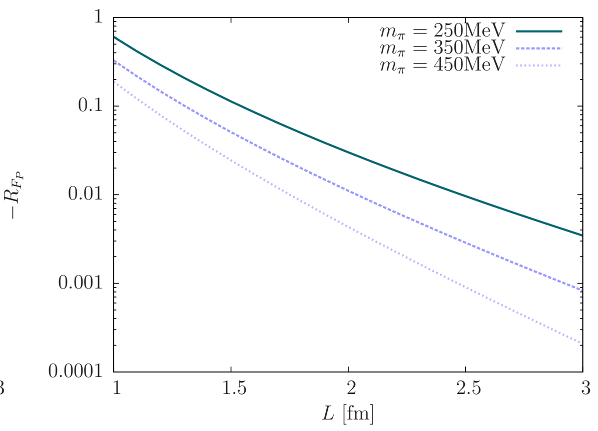
In order to remove discretisation effects one has to take the thermodynamic limit ($L \to \infty$), and the continuum limit ($a \to 0$). The extrapolation to small (physical) quark masses, the so-called chiral limit, then should yield reliable results. Chiral effective field theory (ChEFT) can help to perform these extrapolations.

ChEFT describes low-energy QCD by means of an effective field theory based on pion, nucleon, etc. degrees of freedom taking into account the constraints imposed by (spontaneously broken) chiral symmetry. If the volume is not too small, the finite size effects originate from virtual pions propagating across lattice boundaries. The pion mass has to be small to render ChPT valid. For $m_\pi L \gg 1$ finite size effects are expected to be small.

There are several ways to treat baryons in ChEFT. Ali Khan et al., e.g., applied the (non-relativistic) small scale expansion (SSE) [75], which uses explicit pion, nucleon and $\Delta(1232)$ degrees of freedom and calculated the quark-mass dependence of g_A on finite volumes [73]. Their results show a shift of the axial charge g_A in small volumes towards lower values. This behaviour was confirmed by their $N_f = 2$ dynamical flavour lattice simulation results [73]. Also ongoing efforts with $N_f = 2$ simulations confirm the dropping of g_A in small volumes [74].



(a) Pion mass m_π.



(b) Pion decay constant f_π.

Figure 6.3 – Finite size effects according to the resummed Lüscher formula. Relative shift calculated up to NNLO correction terms.

n	1	2	3	4	5	6	7	8	9	10
$m(n)$	6	12	8	6	24	24	0	12	30	24
n	11	12	13	14	15	16	17	18	19	20
$m(n)$	24	8	24	48	0	6	48	36	24	24

Table 6.1 – The multiplicities $m(n)$ for $n \leq 20$.

6.9 FSE Corrections for Pion Decay Constant

The Lüscher formula represents a convenient and powerful way to calculate corrections to finite size effects for some observables, like e.g. masses [76]. It estimates finite volume effects to subleading (in the chiral counting) order. The formula gives the finite volume shift $M_P(L) - M_P$ of a particle P in terms of the infinite volume πP forward scattering amplitude. For this amplitude the ChPT expression at a certain loop order is used.

A resummed version of the Lüscher formulae for masses and decay constants appeared in [77]. The asymptotic expression for the shift in the pion mass and pion decay constant were given to 3-loop order.

To predict the shifts $M_P(L) - M_P$ and $F_P(L) - F_P$ in a lattice calculation with a known box length L, and thus to correct the data for this systematic effect, we need an explicit representation of the amplitudes $\mathcal{F}_P(\nu)$ and $\mathcal{N}_P(\nu)$, respectively. The resummed Lüscher formula for the relative finite size shift of the pseudoscalar decay constant

$$R_{F_P} = \frac{F_P(L) - F_P}{F_P} \tag{6.13}$$

$$= +\frac{M_\pi}{16\pi^2 \lambda_P F_P} \sum_{n=1}^{\infty} \frac{m(n)}{\sqrt{n}} \int_{-\infty}^{\infty} dy \mathcal{N}_P(iy) e^{-\sqrt{n(1+y^2)}\lambda_\pi} + O\left(e^{-\overline{M}L}\right) \tag{6.14}$$

with $\overline{M} = M_\pi(\sqrt{3}+1)/\sqrt{2}$ and $\lambda_P = m_P L$ and the multiplicities $m(n)$ as given in Tab. 6.1 predicts a negative shift in the pion decay constant for a finite lattice size compared to the infinite volume value. Whereas the according formula for the pseudoscalar meson mass

$$R_{M_P} = \frac{M_P(L) - M_P}{M_P} \tag{6.15}$$

$$= -\frac{M_\pi}{32\pi^2 \lambda_P M_P} \sum_{n=1}^{\infty} \frac{m(n)}{\sqrt{n}} \int_{-\infty}^{\infty} dy \mathcal{F}_P(iy) e^{-\sqrt{n(1+y^2)}\lambda_\pi} + O\left(e^{-\overline{M}L}\right) \tag{6.16}$$

predicts a positive shift.

Fig. 6.3b (6.3a) shows the relative shift for the pion mass (decay constant) where we calculated the corrections until next-to-next-to-leading order (NNLO) and summed up to $n = 20$.

6.10 Disconnected Distributions

Another possible contribution, which has not been incorporated in our lattice computations performed up to now, comes from disconnected diagrams. We refer back to Fig. 4.3b (4.6b) which depicts diagrammatically the disconnected contribution of the two (three) point function.

Chapter 7

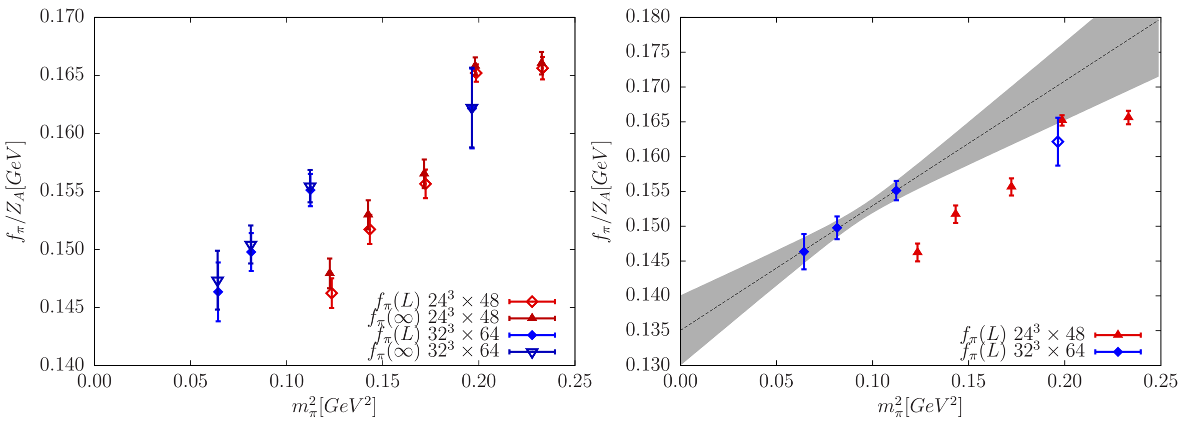
Pion Decay Constant

This work determines the pion decay constant f_π using Eq. (4.146) and (4.147). Operator improvement is not included, since neither a perturbative nor a non-perturbative determination of the operator improvement coefficient c_A is available, see Eq. (4.93). However, for $N_f = 2$ this coefficient is know to be small [78].

Fig. 7.1a shows this work's results for the unrenormalised pion decay constant f_π/Z_A. Shown are the results for lattice sizes $24^3 \times 48$ and $32^3 \times 64$. The measurement data is listed in Tab. B.1 and Tab. B.3 in the appendix.

For smaller pion masses, the data for the $32^3 \times 64$ lattice shows a significant shift towards larger f_π compared to the $24^3 \times 48$ lattice. The direction of the shift is in accordance with the predictions of the resummed Lüscher formula, see Sec. 6.9. In this work the finite size effect corrections were calculated according to the resummed Lüscher formula, using the unrenormalised pion decay constant until NNLO correction terms. The corrected quantities are listed in Tab. B.2 (Tab. B.4) for the $24^3 \times 48$ ($32^3 \times 64$) lattice and included in Fig. 7.1a.

The resummed Lüscher formula does not seem to explain the observed shift. Even though the FSE corrections show a trend in the right directions, i.e. f_π gets shifted towards larger values for the $24^3 \times 48$ lattice, the predictions of ChPT do not explain the large discrepancies observed in this work's lattice measurement. However, this work applies the resummed Lüscher formula for relatively large pion masses where it is not sure if ChPT is valid.



(a) Comparison: Two lattices sizes. **(b)** Linear extrapolation to physical point.

Figure 7.1 – Unrenormalised pion decay constant f_π/Z_A. Left panel: Diamonds (measured quantity), triangles (with FSE corrections). Right panel: Extrapolation with filled symbols.

It is not obvious what is happening at the symmetric point. FSE should lower f_π for a smaller volume. Instead a larger value is seen for a smaller volume, although the statistics on the $32^3 \times 64$ lattice is still small so this may improve after more measurements.

Extrapolation formulae obtained from ChPT predict logarithmic behaviour, but with the actual data this can not be fitted. The best that can be done at the moment is a linear two-parameter fit to the physical pion mass

$$f_\pi(m_\pi) = a_0 + a_1 m_\pi^2. \tag{7.1}$$

This work makes use of the measurement data of the $32^3 \times 64$ lattice to carry out the two-parameter fit not taking into account the symmetric point. The fit results are detailed in Tab. C.1 and depicted in Fig. 7.1b. One finds

$$f_\pi(m_\pi^{\text{phys}})/Z_A = 0.1383(51). \tag{7.2}$$

A comparison with the experimental value of f_π, see Eq. (3.6), allows to give an estimate of the renormalisation constant

$$Z_A^{\text{est}} \approx 0.940(35). \tag{7.3}$$

Chapter 8

$n = 1$ Moment of Polarised PDF

Form factors are fundamental hadronic observables that probe the structure of hadrons. A new generation of experiments using polarised beams and targets are currently under way at major facilities in order to measure the spin structure of the nucleon and at higher values of the momentum transfer. The nucleon form factors connected to the axial vector current are more difficult to measure and therefore less accurately known than the electromagnetic form factors. The nucleon matrix element of the axial vector current is written in terms of two Lorenz invariant form factors, the axial form factor and the induced pseudo-scalar form factor depending on the momentum transfer squared. The nucleon axial charge, defined as the axial vector current at zero momentum transfer which can be determined from β-decay, is known to a high precision. The momentum transfer-dependence of the axial vector form factor has been studied from neutrino scattering [79] and pion electro-production [80, 81] processes.

The axial charge is defined as the axial vector form factor at zero four-momentum transfer, $g_A^B = G_A(0)$, which is obtained from the matrix element for the baryon, B, see Sec 3.2,

$$\langle B(p', s')|A_\mu^{u-d}|B(p, s)\rangle = \overline{u}_B(p', s') \left[\gamma_\mu \gamma_5 G_A(q^2) + \gamma_5 \frac{q_\mu}{2m_N} G_P(q^2)\right] u_B(p, s) \ , \qquad (8.1)$$

where $q = p' - p$ denotes the 4-momentum transfer and $u_B(p, s)$ is the spinor for the baryon, B, with momentum p and spin vector s and G_P is the induced pseudoscalar form factor. The isovector axial current is defined as $A_\mu^{u-d} = \overline{u}\gamma_\mu\gamma_5 u - \overline{d}\gamma_\mu\gamma_5 d$ where u and d denote the up and down quark fields, respectively. We work in the limit of exact isospin invariance, i.e. u and d quarks are assumed to be degenerate in mass. The states are

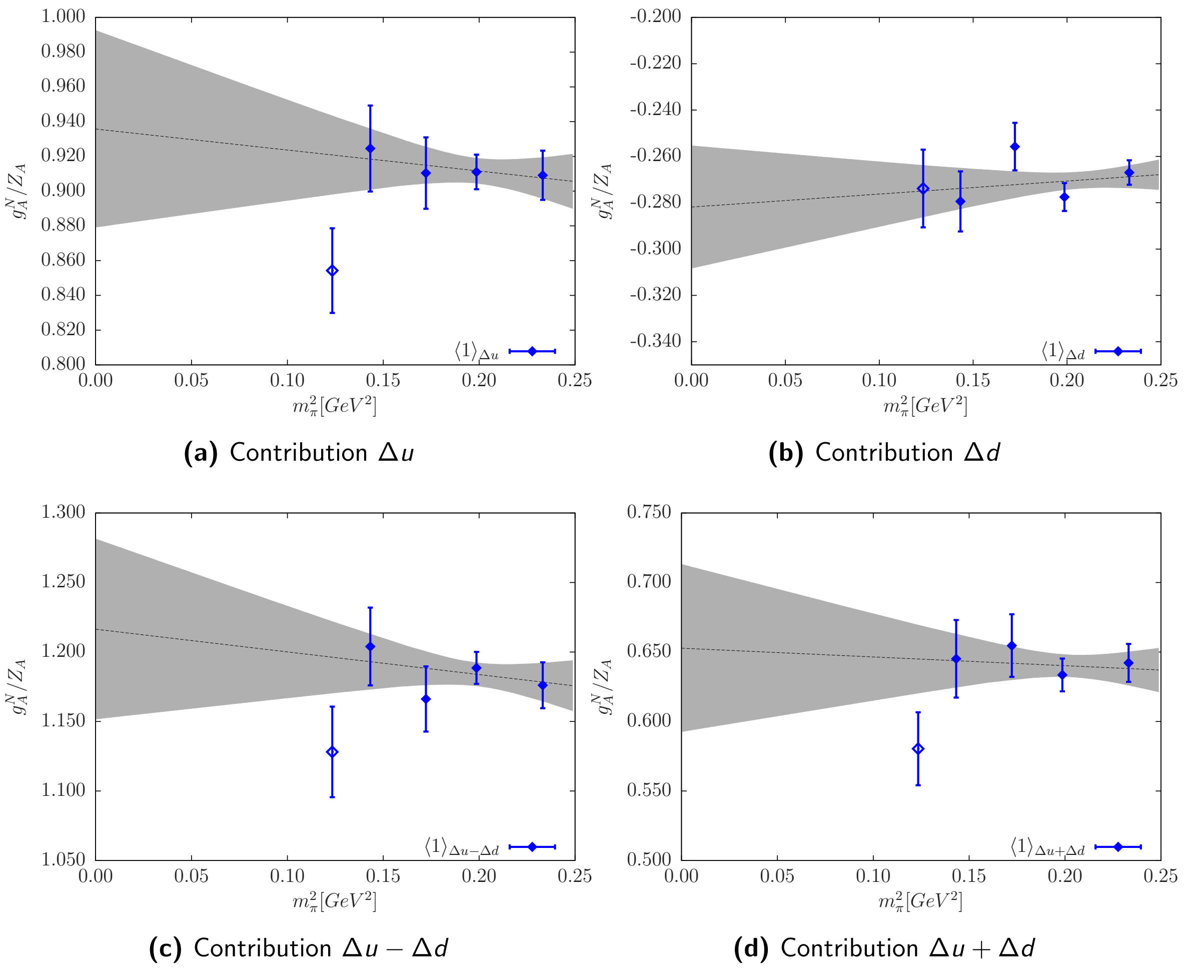


Figure 8.1 – First ($n = 1$) moment of polarised PDF $\langle 1\rangle^N_{\Delta q}$. Data points with open symbols were not included in the extrapolation.

normalised according to $\langle p', s'|p, s\rangle = (2\pi)^3 2p^0\delta(\mathbf{p} - \mathbf{p}')\delta_{ss'}$, we take $s^2 = -m_B^2$ and m_B is the baryon mass. Thus the axial charge is given by the forward matrix element

$$\langle B(p, s)|A_\mu^{u-d}|B(p, s)\rangle = 2g_A^B s_\mu. \tag{8.2}$$

In parton model language, the forward matrix elements of the axial current are related to the fraction of the spin of the baryon carried by the quarks. Denoting by $\langle 1\rangle^B_{\Delta q}$ the contribution of the quark, q, to the spin of the baryon, B, we find

$$\langle B(p, s)|\overline{q}\gamma_\mu\gamma_5 q|B(p, s)\rangle = 2\langle 1\rangle^B_{\Delta q} s_\mu. \tag{8.3}$$

Thus for the nucleon we write $g_A^N = \langle 1\rangle^N_{\Delta u} - \langle 1\rangle^N_{\Delta d}$.

8.1 Nucleon

Figs. 8.1a - 8.1d show the Δu and Δd contribution to the axial charge for the nucleon. Since we do not have a nonperturbative determination of the renormalisation constant Z_A we show the unrenormalised quantities. Later in this section we will use Z_A determined in the previous section from f_π, however since as we discussed, there are some remaining systematic uncertainties that need to be understood before we arrive at a reliable determination of f_π. The individual contributions $\langle 1 \rangle_{\Delta u}$ and $\langle 1 \rangle_{\Delta d}$, and the combined contributions $\langle 1 \rangle_{\Delta u - \Delta d}$ and $\langle 1 \rangle_{\Delta u + \Delta d}$ are displayed separately. We give the measurement data for each of the contributions and its error obtained by a bootstrap analysis in Tab. B.5.

At the smallest pion mass the Δu contribution of the quantity is significantly smaller than for the heavier pion masses. This is clearly seen in Fig. 8.1a. Finite size effects of the nucleon axial charge have been seen in earlier work with $N_f = 2$ flavours of dynamical Wilson Clover fermions, see Sec. 6.6. In earlier work with $N_f = 2$ flavours of dynamical $\mathcal{O}(a)$ improved Wilson fermions [73] significant finite size effects were observed. FSE corrections independent of N_f were calculated via ChPT. The extrapolation with FSE corrections via ChPT carried out in their discussion results in $g_A^N = 1.15(12)$ with a fit to the lightest pion masses. Also they find discretisation effects to be small. We argue that the smaller value of our nucleon axial charge at the lightest pion masses is due to finite size effects and we expect discretisation effects to be small.

With the current data we can not do better than a first approximation with a linear two-parameter fit to extrapolate to the physical point

$$\langle 1 \rangle^B_{Deltaq} = a_0 + a_1 m_\pi^2. \tag{8.4}$$

The fit results and extrapolations to the physical point for each of the contributions are given separately in the upper part of Tab. C.2. For the nucleon we find:

$$\begin{aligned}
\langle 1 \rangle^N_{\Delta u}/Z_A = &\quad 0.934(57) \\
\langle 1 \rangle^N_{\Delta d}/Z_A = &\quad \text{-}0.281(27) \\
\langle 1 \rangle^N_{\Delta u - \Delta d}/Z_A = &\quad 1.213(65) \\
\langle 1 \rangle^N_{\Delta u + \Delta d}/Z_A = &\quad 0.652(61)
\end{aligned}$$

Baryon	u-quark	d-quark
N	κ_l	κ_l
Σ	κ_l	κ_s
Ξ	κ_s	κ_l

Table 8.1 – Matching of the hopping parameters for the light quark mass κ_l and the strange quark mass κ_s to the flavours in the nucleon interpolators.

Since these are unrenormalised quantities we can not compare to experiment, but we can use the estimate from Eq. (7.3) and thus give a rough estimate: $g_A^N = 1.141(75)$.

From experiment the nucleon axial charge g_A^N is know very precisely from neutron beta decay. A recent review [82] gives $g_A^N = 1.2750(9)$ and the PDG [10] world average is $g_A^N = 1.2694(28)$. Thus we underestimate the nucleon axial charge g_A^N.

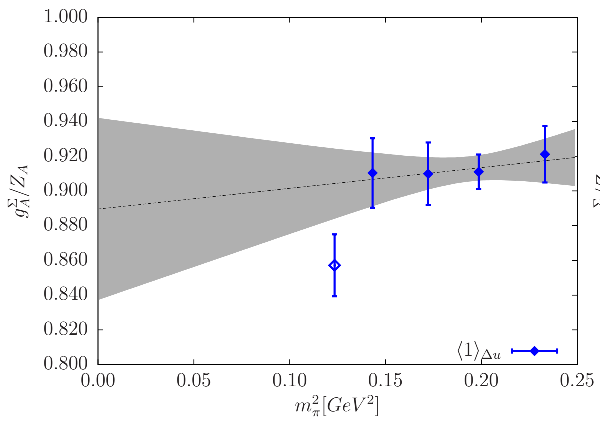
Much effort was already put into the determination of g_A^N with full QCD lattice simulations [83, 84, 85, 86, 74, 87, 88, 89]. All groups consistently underestimate the experimental value.

Reasons for this discrepancy are still not yet completely understood, although it is likely to be a combination of finite size effects and chiral non-analytic behaviour close to the physical point.

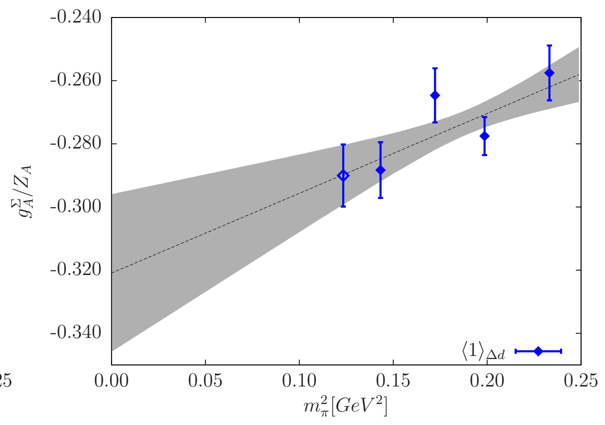
8.2 Hyperons

Since there is currently no experimental data for the axial charges of the other octect baryons, and theoretical predictions are rather imprecise, Lattice QCD gives us the opportunity to make predictions for $\langle 1 \rangle^B_{Deltaq}$ the other octect baryons, $B = \Sigma^+, \Xi^0$.

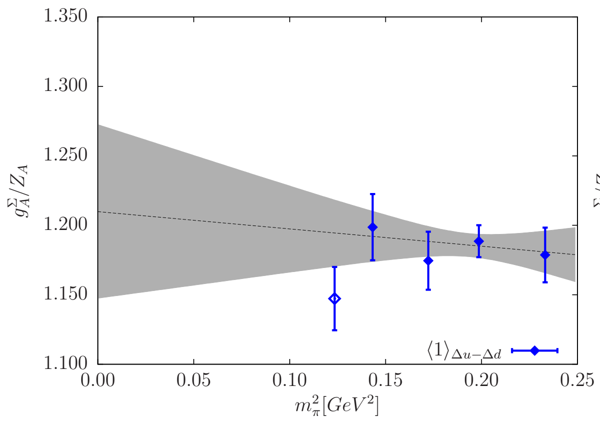
We reuse the proton interpolator, Eq. (4.65) and (4.66), for the Σ and Ξ. We map the simulation parameters κ_l and κ_s, see Sec. 5.3, to the u and d quark of the proton interpolator according to Tab. 8.1.



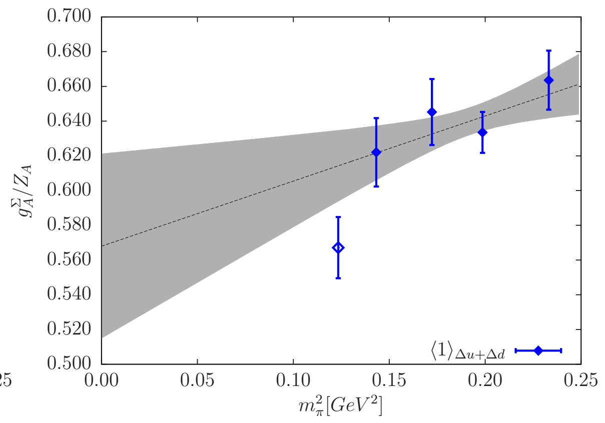
(a) Contribution Δu



(b) Contribution Δd



(c) Contribution $\Delta u - \Delta d$



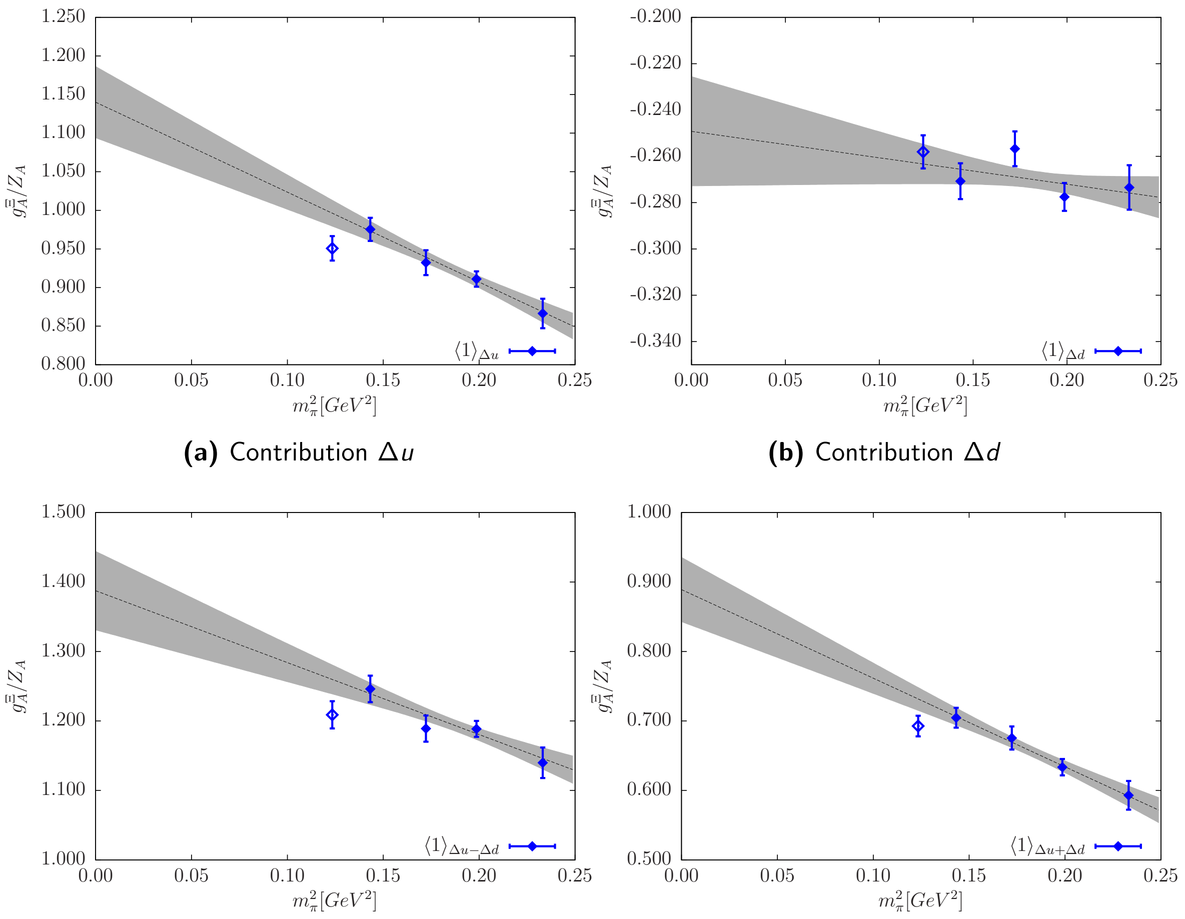
(d) Contribution $\Delta u + \Delta d$

Figure 8.2 – First ($n = 1$) moment of polarised PDF $\langle 1 \rangle^{\Sigma}_{\Delta q}$.

Fig. 8.2a - 8.2d (8.3a - 8.3d) show the unrenormalised axial charge for the Σ (Ξ). Again we display the individual contributions in separate figures. The measurement data for the individual contributions for the Σ (Ξ) is given in Tab. B.6 (B.7).

Like for the nucleon, the Δu-part has dropped significantly for the lightest pion mass. This might again be caused by finite size effects.

We carry out a first (linear) extrapolation to the physical point using Eq. 8.4. The fit results and extrapolations to the physical point for each of the contributions are given separately in the middle (lower) part of Tab. C.2 for the Σ (Ξ). For the Σ the axial charge of the



(a) Contribution Δu **(b)** Contribution Δd

(c) Contribution $\Delta u - \Delta d$ **(d)** Contribution $\Delta u + \Delta d$

Figure 8.3 – First ($n = 1$) moment of polarised PDF $\langle 1\rangle^{\Xi}_{\Delta q}$.

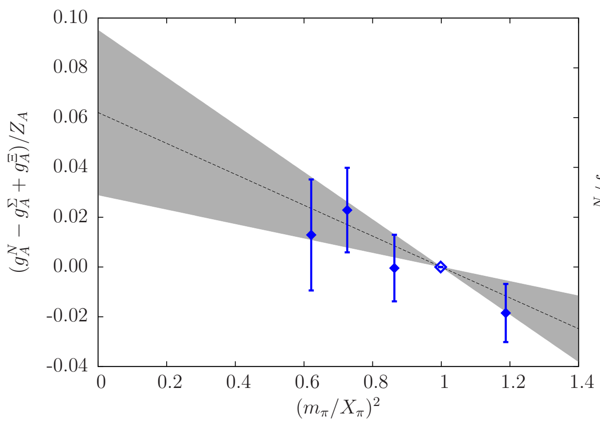
unpolarised quark distributions at the physical point are found to be:

$$\begin{aligned}
\langle 1\rangle^{\Sigma}_{\Delta u}/Z_A &= \quad 0.892(53)\\
\langle 1\rangle^{\Sigma}_{\Delta d}/Z_A &= \quad \text{-}0.316(25)\\
\langle 1\rangle^{\Sigma}_{\Delta u-\Delta d}/Z_A &= \quad 1.208(63)\\
\langle 1\rangle^{\Sigma}_{\Delta u+\Delta d}/Z_A &= \quad 0.575(53)
\end{aligned}$$

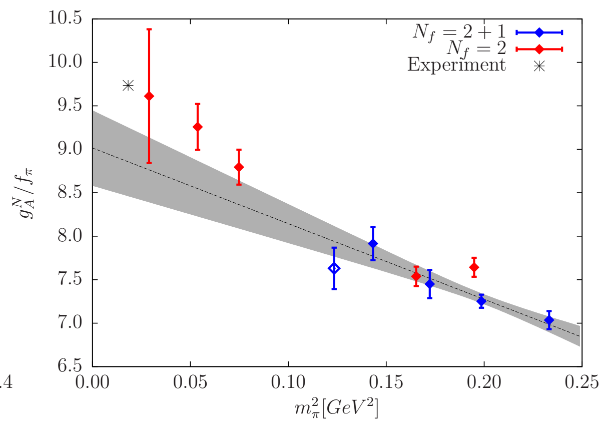
Whereas for the Ξ we find:

$$\begin{aligned}
\langle 1\rangle^{\Xi}_{\Delta u}/Z_A &= \quad 1.119(47)\\
\langle 1\rangle^{\Xi}_{\Delta d}/Z_A &= \quad \text{-}0.251(24)\\
\langle 1\rangle^{\Xi}_{\Delta u-\Delta d}/Z_A &= \quad 1.369(57)\\
\langle 1\rangle^{\Xi}_{\Delta u+\Delta d}/Z_A &= \quad 0.866(47)
\end{aligned}$$

In the context of hyperons the axial charges are also important to learn about the role of flavour SU(3) symmetry breaking. In particular, in the case of conserved flavour SU(3)



(a) $\delta_{SU(3)} = (g_A^N - g_A^\Sigma + g_A^\Xi)/Z_A$



(b) g_A^N/f_π

Figure 8.4 – Left: SU(3) symmetry breaking term. Right: Nucleon axial charge over pion decay constant.

symmetry the axial charges of the N, Σ, and Ξ ground states are connected by the following simple relations, see Sec. 3.3:

$$g_A^N = F + D \tag{8.5}$$

$$g_A^\Sigma = 2F \tag{8.6}$$

$$g_A^\Xi = F - D. \tag{8.7}$$

These follow through SU(3) Clebsch-Gordan coefficients in the decomposition of the axial form factor into the functions F and D relating to the baryon octet components in SU(3) [90].

One way to test flavour SU(3) symmetry breaking in the axial couplings is to study the quantity $\delta_{SU(3)}$, defined as

$$\delta_{SU(3)} = g_A^N - g_A^\Sigma + g_A^\Xi. \tag{8.8}$$

Fig. 8.4a shows the unrenormalised quantity $\delta_{SU(3)}/Z_A$ as a function of m_π/X_π with the flavour singlet quantity

$$X_\pi^2 = \frac{1}{3}(2m_K^2 + m_\pi^2). \tag{8.9}$$

The measurement data is given in Tab. B.8. We see a very mild SU(3) symmetry breaking towards lighter pion masses.

We carried out a linear one-parameter fit anchored at the symmetric point. The fit results are given in Tab. C.3. An extrapolation to the physical point gives

$$\delta_{SU(3)}/Z_A = \quad 0.052(34)$$

which indicates a very small SU(3) symmetry breaking. This is the unrenormalised quantity; if multiplied with our estimate Z_A^{est} we obtain

$$\delta_{SU(3)}/Z_A \times Z_A^{\mathrm{est}} = 0.049(32).$$

Three attempts have been carried out to determine the axial charges for the Σ and Ξ using different theoretical approaches: Chiral perturbation theory [91], the large-N_c limit [92, 93], and the quark model [94]. In the attempt using chiral perturbation theory the one-loop corrections due to flavour SU(3) symmetry breaking were calculated. The axial charge for the Σ and Ξ were predicted to $0.70 \leq g_A^{\Sigma} \leq 1.10$ and $0.18 \leq -g_A^{\Xi} \leq 0.36$. Using the large-$N_c$ approach the following ranges were predicted: $0.60 \leq g_A^{\Sigma} \leq 0.72$ and $0.26 \leq -g_A^{\Xi} \leq 0.30$. Both approaches give very loose bounds on the values of these coupling constants.

An earlier lattice calculation also determined the hyperon axial charges [95]. They used $N_f = 2 + 1$ dynamical flavours of (improved) staggered fermions (asqtad) [96, 97, 98] for the sea quarks and domain-wall fermions (DWF) [99, 100, 101, 102] for the valence sector. The pion mass was varied from roughly 750 MeV down to 350 MeV in a lattice box size of 2.6 fm. The axial coupling constant was expanded in terms of the flavour SU(3) breaking parameter $x = (m_K^2 - m_\pi^2)/4\pi^2 f_\pi^2$ as follows

$$g_A^N = D^{\mathrm{symm}} + F^{\mathrm{symm}} + \sum_n C_N^{(n)} x^n \tag{8.10}$$

$$g_A^{\Sigma} = 2F^{\mathrm{symm}} + \sum_n C_{\Sigma}^{(n)} x^n \tag{8.11}$$

$$g_A^{\Xi} = F^{\mathrm{symm}} - D^{\mathrm{symm}} + \sum_n C_{\Xi}^{(n)} x^n \tag{8.12}$$

with the superscript $^{\mathrm{symm}}$ denoting the quantity at the symmetric point, i.e. independent of x and the lattice data was found to prefer the constraint

$$C_N^{(1)} - C_{\Sigma}^{(1)} + C_{\Xi}^{(1)} = 0. \tag{8.13}$$

With a combined fit for $n = 1$ the axial charges of the hyperons were determined to be $g_A^\Sigma = 0.900(42)$ and $g_A^\Xi = -0.277(15)$. (Due to another definition of g_A^Σ this numerical value is different by a factor of 2 from the one quoted in the paper [95].) These results are in accordance with our results.

A more recent lattice calculation [103] determined the hyperon axial charges to be $g_A^\Sigma = 0.970(30)$ and $g_A^\Xi = -0.299(14)$. (Due to another definition of g_A^Σ (g_A^Ξ) this numerical value is different by a factor of $\sqrt{2}$ (by sign opposite) from the one quoted in the paper [95].) Again, these results are in accordance with our results.

8.3 Ratios

Since we do not have the renormalisation constant yet we now consider ratios were this constant cancels.

The first ratio we consider is the axial charge of the nucleon g_A^N over the pion decay constant $f_{\pi^\pm}$. We plot this ratio in Fig. 8.4b. Since the renormalisation constants cancel in the ratio, we are able to compare our results to the experimental value [10] (star) and to $N_f = 2$ results [74]. Except for the lightest pion mass, which is possibly due to FSE, the measurements show a trend towards the experimental value and agree very well with the $N_f = 2$ results.

Next we make use of the SU(3) constants introduced in Eqs. (8.5)-(8.7) and consider the following ratios:

$$\frac{g_A^\Sigma}{g_A^N} = \frac{2F}{F+D} \tag{8.14}$$

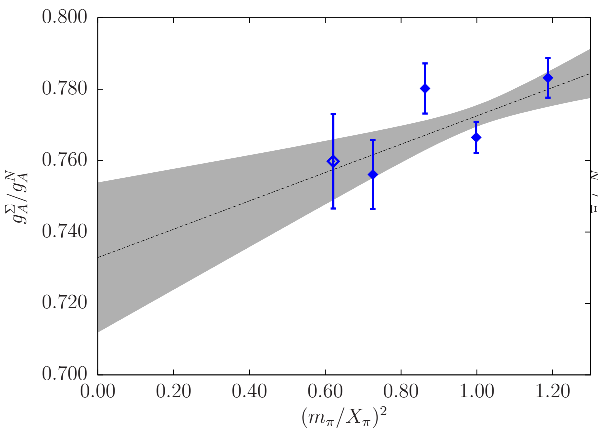
$$\frac{g_A^\Xi}{g_A^N} = \frac{F-D}{F+D} \tag{8.15}$$

and

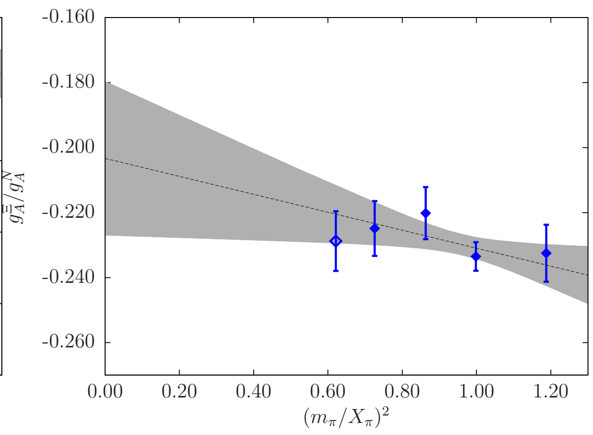
$$\frac{g_A^N - g_A^\Xi}{g_A^\Sigma} = \frac{D}{F} \tag{8.16}$$

$$\frac{g_A^N + g_A^\Xi}{g_A^\Sigma} = 1. \tag{8.17}$$

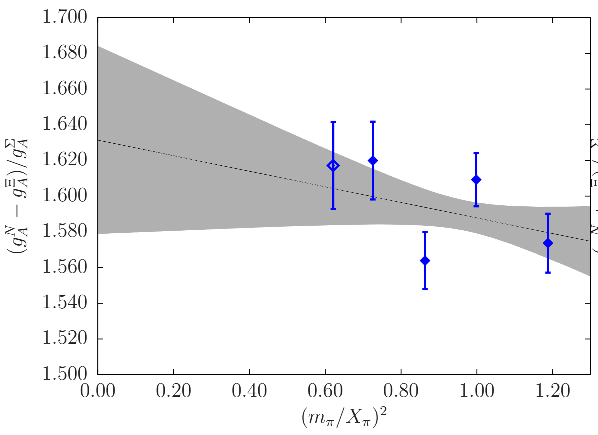
We plot these ratios in Figs. 8.5a - 8.5d as a function of $(m_\pi/X_\pi)^2$. The measurement data is given in Tab. B.22 - B.25.



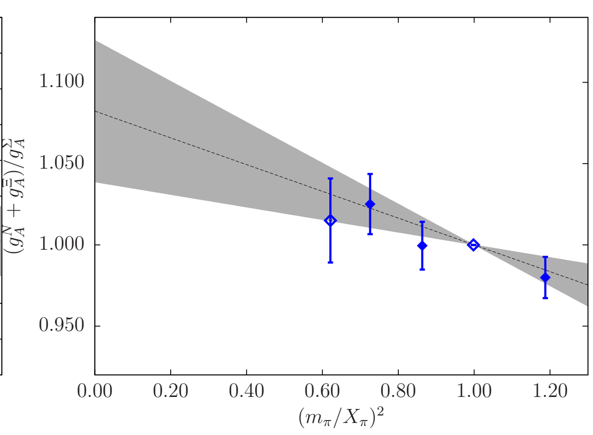
(a) g_A^Σ/g_A^N



(b) g_A^Ξ/g_A^N



(c) $(g_A^N - g_A^\Xi)/g_A^\Sigma$



(d) $(g_A^N + g_A^\Xi)/g_A^\Sigma$

Figure 8.5 – Ratios of axial charges.

We use a linear extrapolation to obtain preliminary predictions at the physical quark masses. The fit results are displayed in Tab. C.4 and Tab. C.5. For the ratio $(g_A^N + g_A^\Xi)/g_A^\Sigma$ we use a one-parameter fit with a fit function anchored at the symmetric point whereas for the other ratios we use a two-parameter linear fit ansatz. We obtain for the combinations of D and F at the physical point the following values:

$$\frac{g_A^\Sigma}{g_A^N} = \frac{2F}{F+D} = 0.740(21) \tag{8.18}$$

$$\frac{g_A^\Xi}{g_A^N} = \frac{F-D}{F+D} = -0.208(24) \tag{8.19}$$

and

$$\frac{g_A^N - g_A^\Xi}{g_A^\Sigma} = \frac{D}{F} = 1.624(53) \qquad (8.20)$$

$$\frac{g_A^N + g_A^\Xi}{g_A^\Sigma} = 1.068(44) \qquad (8.21)$$

These preliminary results are in agreement with earlier lattice [95, 103] and quark model [94] determinations.

From a fit to the experimental data taking model independent leading SU(3) breaking contributions to the axial current matrix elements into account Savage and Walden [91] found the following values: $F = 0.47(7)$ and $D = 0.79(10)$. Combining the central values the following ratios are obtained: $g_A^\Sigma/g_A^N = 0.75$, $g_A^\Xi/g_A^N = -0.25$, and $(g_A^N - g_A^\Xi)/g_A^\Sigma = 1.68$. Thus, our results are also in good accordance with their results.

Chapter 9

$n = 2$ Moment of Unpolarised PDF

The second ($n = 2$) moment of a baryon's, B, unpolarised quark distribution function, $q(x)$ gives the total fraction of the baryon's momentum carried by the quark, q, $\langle x \rangle_q^B$. On the lattice this work calculates moments of the quark distribution functions $q(x)$

$$\langle x^{n-1} \rangle_q^B = \int_0^1 dx\, x^{n-1}(q^B(x) + (-1)^n \overline{q}^B(x)) \tag{9.1}$$

where x is the momentum fraction of the baryon B carried by the quarks by calculating the matrix elements of local twist-2 operators

$$\langle B(p)| \left[\mathcal{O}_q^{\mu_1 \cdots \mu_n} - \text{tr}\right] |B(p)\rangle = 2\langle x^{n-1} \rangle_q^B [p^{\mu_1 \cdots \mu_n} - \text{tr}], \tag{9.2}$$

where

$$\mathcal{O}_q^{\mu_1 \cdots \mu_n} = i^{n-1} \overline{q} \gamma^{\mu_1} \overset{\leftrightarrow}{D}^{\mu_2} \cdots \overset{\leftrightarrow}{D}^{\mu_n} q. \tag{9.3}$$

In order to calculate the quark momentum fractions this work considers only the second moment, $\langle x \rangle_q$, for which one uses the standard local operator

$$\mathcal{O}_q^{\langle x \rangle} = \mathcal{O}_q^{44} - \frac{1}{3}\left(\mathcal{O}_q^{11} + \mathcal{O}_q^{22} + \mathcal{O}_q^{33}\right) \tag{9.4}$$

where $\overset{\leftrightarrow}{D} = (\vec{D} - \overset{\leftarrow}{D})/2$ is the forward/backward covariant derivative.

This is not the only possible choice of lattice operator to calculate the momentum fractions. The operator $\overline{\psi}\gamma_i \overleftrightarrow{D_4} \psi$ might be used as well. However to extract the matrix element using this operator requires a non-zero baryon momentum $p_i \neq 0$. Lattice calculations with $p \neq 0$ are more noisy. This work's lattice simulations were carried out at zero baryon momentum, thus one extracts the matrix elements using Eq. (9.2).

Collaboration	$\langle x \rangle_{u-d}$
ABMK	0.1646 ± 0.0027
BBG	0.1603 ± 0.0041
JR	0.1496 ± 0.0062
MSTW	0.1501 ± 0.0048
AMP06	0.1676 ± 0.0058

Table 9.1 – Phenomenological values for $\langle x \rangle_{u-d}$ at $\mu = 2$ GeV for the nucleon.

The matrix element in Eq. (9.2) is obtained on the lattice by considering the ratio

$$R(t, \tau, \vec{p}) = \frac{C_{3\text{pt}}(t, \tau, \vec{p})}{C_{2\text{pt}}(t, \vec{p})} = -\frac{E_{\vec{p}}^2 + \frac{1}{3}\vec{p}^{\,2}}{E_{\vec{p}}} \langle x \rangle \tag{9.5}$$

where $C_{2\text{pt}}$ and $C_{3\text{pt}}$ are lattice two and three-point functions, respectively, with total momentum $\vec{p}$ (in the simulation considered here only $\vec{p} = 0$). The operator $\mathcal{O}_q^{\langle x \rangle}$ from Eq. (9.4) is inserted into the three-point function $C_{3\text{pt}}(t, \tau, \vec{p})$ at time τ between the baryon source located at time $t = 0$ and sink at time t.

The state-of-the-art results for the particular quantity of interest to us, $\langle x \rangle_{u-d}$, are collected in Tab. 9.1 [104, 105, 106, 107, 108, 109].

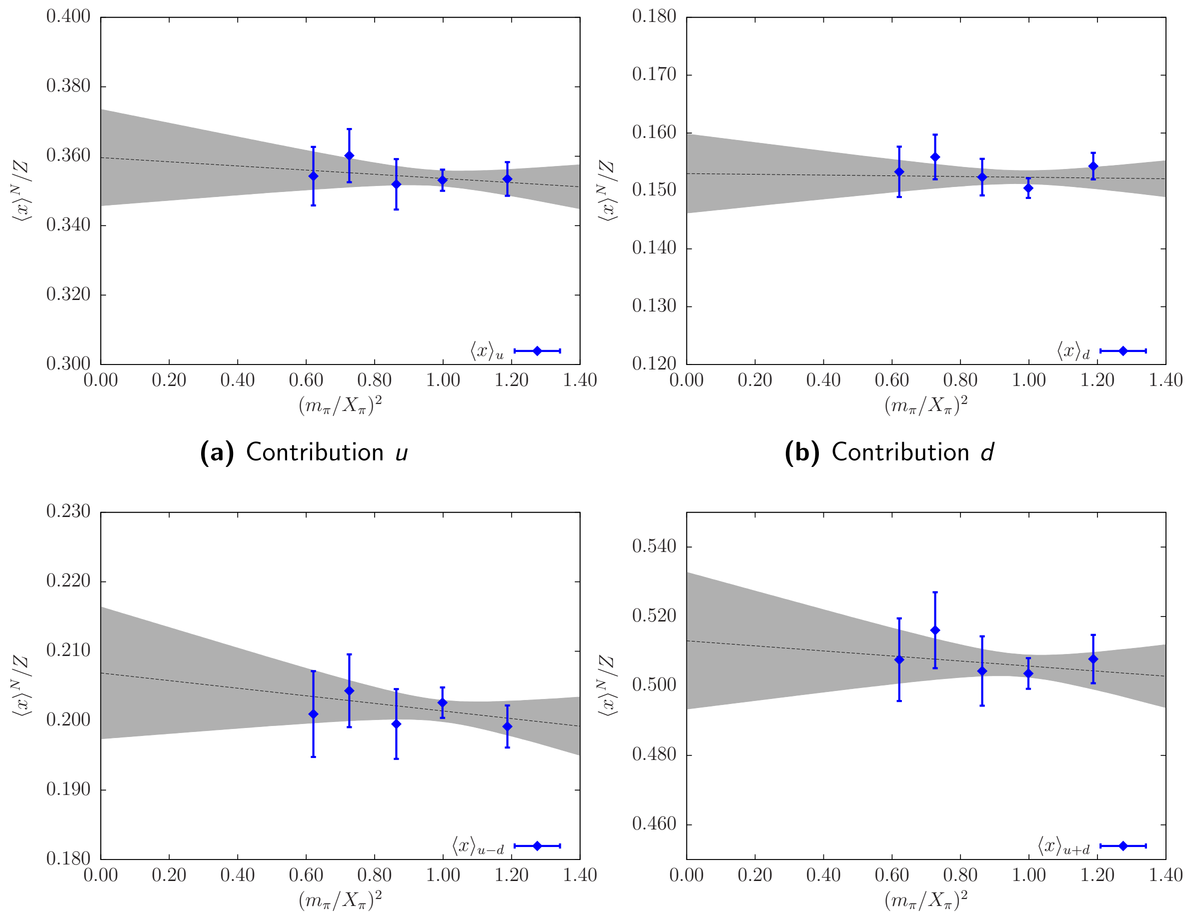
9.1 Nucleon

Figs. 9.1a - 9.1d show $\langle x \rangle_q$ for the nucleon. Since the renormalisation constant Z is not available the unrenormalised quantities are shown. The individual contributions $\langle x \rangle_u$ and $\langle x \rangle_d$, and the combined contributions $\langle x \rangle_{u-d}$ and $\langle x \rangle_{u+d}$ are displayed separately. This work lists the measurement data for each of the contributions in Tab. B.9.

Again this work tries a first approximation with a linear two-parameter fit to the physical point

$$\langle x \rangle_q^B / Z = a_0 + a_1 m_\pi^2. \tag{9.6}$$

The fit results and extrapolations to the physical point for each of the contributions are given separately in the upper part of Tab. C.6. For the nucleon the momentum fractions



(a) Contribution u **(b)** Contribution d

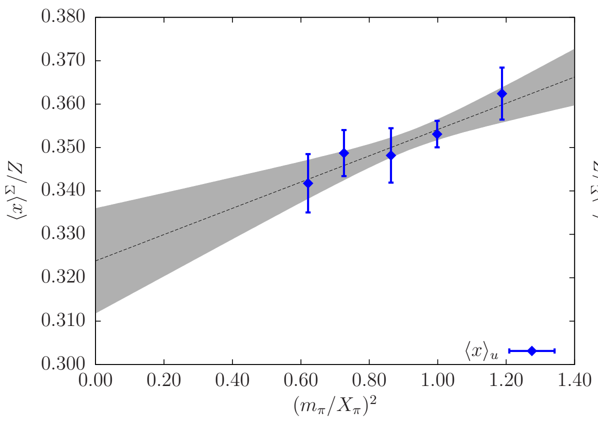
(c) Contribution $u-d$ **(d)** Contribution $u+d$

Figure 9.1 – Second ($n=2$) moment of unpolarised PDF $\langle x\rangle_q^N$.

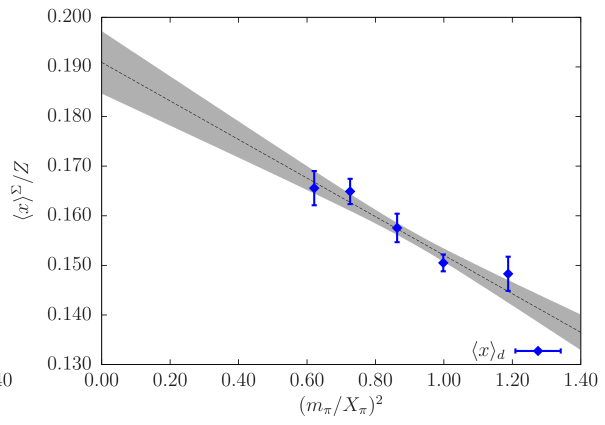
of the unpolarised quark distribution at the physical point are found to be:

$$
\begin{aligned}
\langle x\rangle_u^N/Z &= \quad 0.359(14)\\
\langle x\rangle_d^N/Z &= \quad 0.1529(70)\\
\langle x\rangle_{u-d}^N/Z &= \quad 0.2059(97)\\
\langle x\rangle_{u+d}^N/Z &= \quad 0.512(20)
\end{aligned}
$$

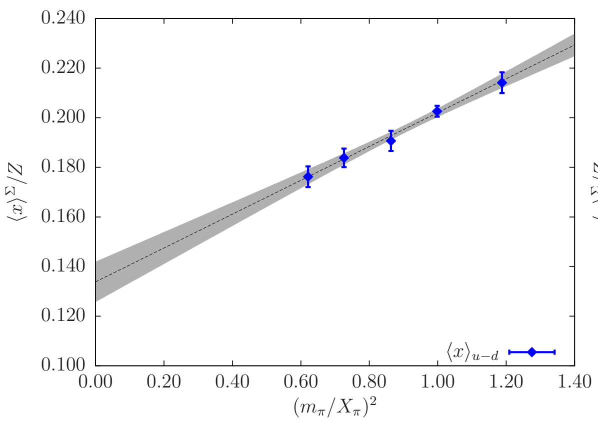
Although this work's results only cover a small mass range, the general behaviour is similar to that seen in other lattice simulations, i.e. very flat. When looking at the world lattice results for the quark momentum fraction $\langle x\rangle_{u-d}$ [110, 111, 84, 83, 112] the most striking feature that one observes is that, despite the community's efforts to calculate with many actions, several lattice spacings and volumes and a broad range of pion masses, the calculations still overestimate the experimental measurement by at least 30% and maybe as much as a factor of 2. The spread amongst the groups obviously suggests some systematic variations.



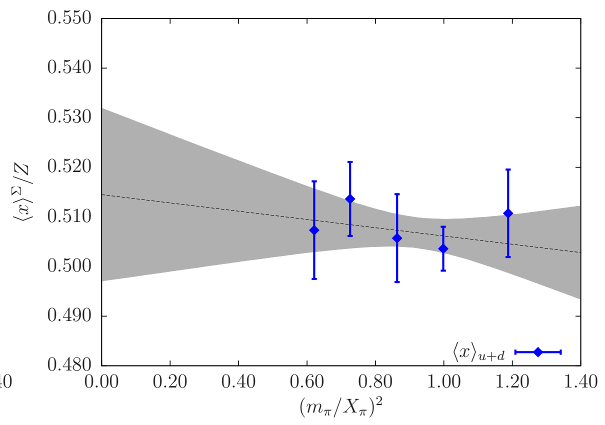
(a) Contribution u



(b) Contribution d



(c) Contribution $u - d$



(d) Contribution $u + d$

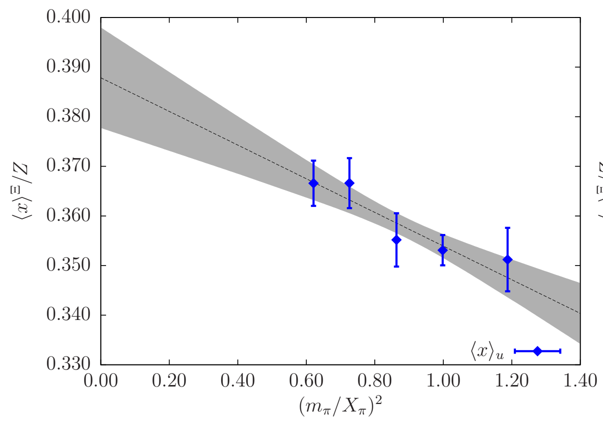
Figure 9.2 – Second ($n = 2$) moment of unpolarised PDF $\langle x \rangle^{\Sigma}_{q}$.

9.2 Hyperons

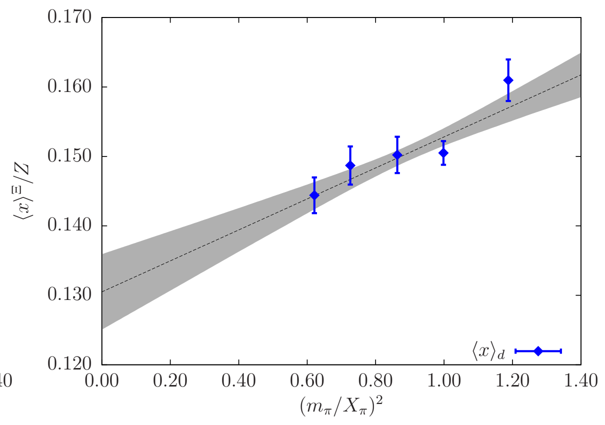
Figs. 9.2a - 9.2d (9.3a - 9.3d) show $\langle x \rangle_q$ for the Σ (Ξ). Again this work shows the unrenormalised quantity and the individual contributions $\langle x \rangle_u$ and $\langle x \rangle_d$, and the combined contributions $\langle x \rangle_{u-d}$ and $\langle x \rangle_{u+d}$ are displayed separately. The measurement data for each of the contributions is given in Tab. B.10 and B.11.

As in the previous section for the axial charges, the renormalisation constant for the momentum fractions Z has not yet been determined. As a consequence in this work no quantitative predictions for the quark momentum fractions of the hyperons is given.

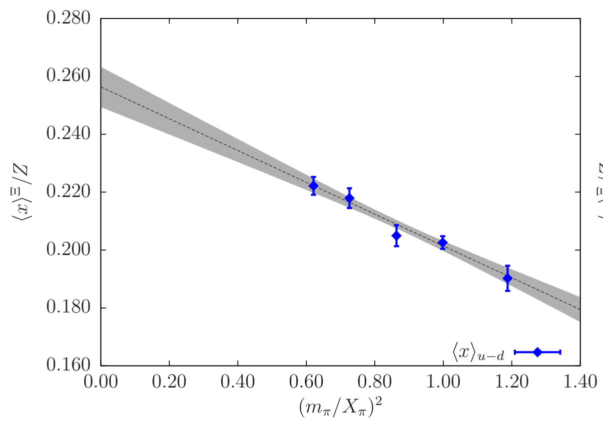
A first (linear) extrapolation to the physical point is carried out using Eq. 9.6. The fit results and extrapolations to the physical point for each of the contributions are given



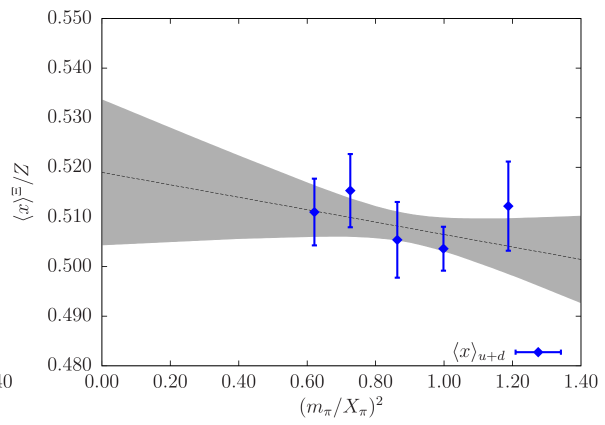
(a) Contribution u



(b) Contribution d



(c) Combined contribution $u-d$



(d) Combined contribution $u+d$

Figure 9.3 – Second ($n=2$) moment of unpolarised PDF $\langle x\rangle^{\Xi}_{q}$.

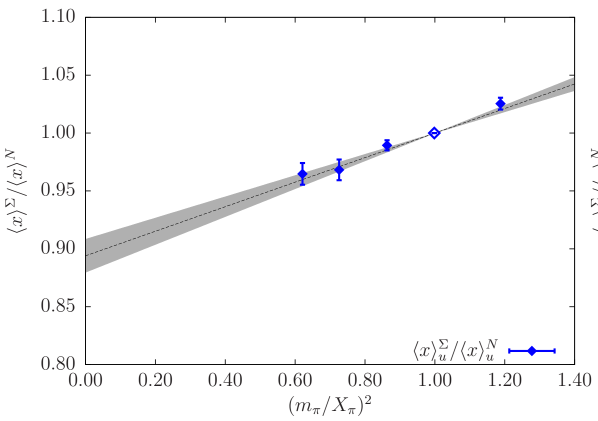
separately in the middle (lower) part of Tab. C.6 for the Σ (Ξ). For the Σ:

$$\begin{aligned}
\langle x\rangle^{\Sigma}_{u}/Z &= 0.329(12)\\
\langle x\rangle^{\Sigma}_{d}/Z &= 0.1844(64)\\
\langle x\rangle^{\Sigma}_{u-d}/Z &= 0.1453(82)\\
\langle x\rangle^{\Sigma}_{u+d}/Z &= 0.513(18)
\end{aligned}$$

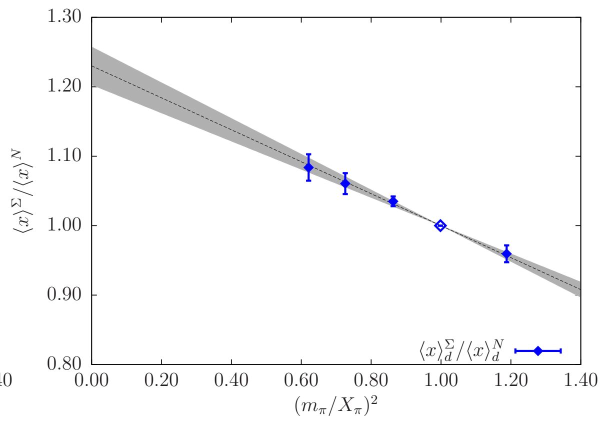
For the Ξ:

$$\begin{aligned}
\langle x\rangle^{\Xi}_{u}/Z &= 0.382(10)\\
\langle x\rangle^{\Xi}_{d}/Z &= 0.1343(55)\\
\langle x\rangle^{\Xi}_{u-d}/Z &= 0.2471(70)\\
\langle x\rangle^{\Xi}_{u+d}/Z &= 0.517(15)
\end{aligned}$$

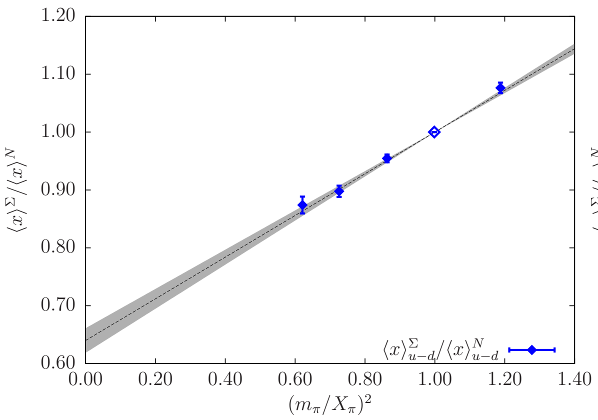
This work studies the ratios of the momentum fractions of the unpolarised quark distribution where the normalisation constant cancels.



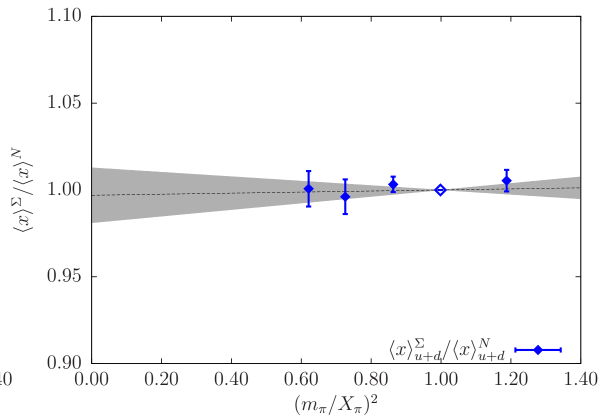
(a) Contribution u



(b) Contribution d



(c) Contribution $u - d$



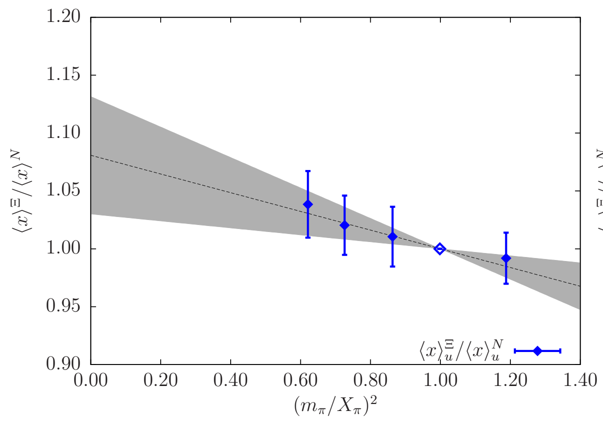
(d) Contribution $u + d$

Figure 9.4 – Ratio $\langle x\rangle^{\Sigma}_{q}/\langle x\rangle^{N}_{q}$

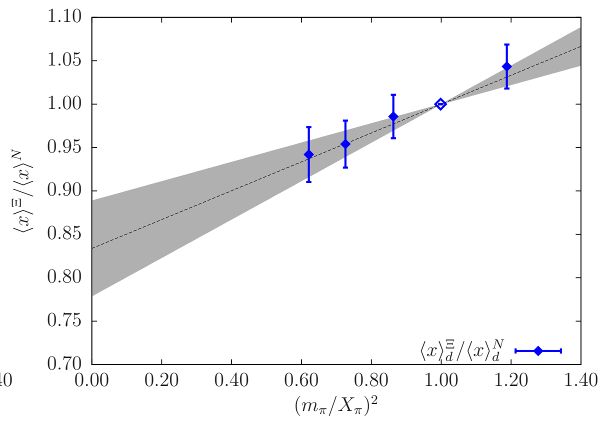
Figs. 9.4a - 9.4d (9.5a - 9.5d) show the ratios $\langle x\rangle^{B}_{q}/\langle x\rangle^{N}_{q}$ with $B = \Sigma$ (Ξ). The strange quark in the Σ takes a larger fraction of the total momentum compared to the down quark in the proton. Likewise do the strange quarks in the Ξ compared to the up quarks in the proton. The total contribution for both hyperons are relatively flat.

9.3 Isospin-Symmetry Breaking

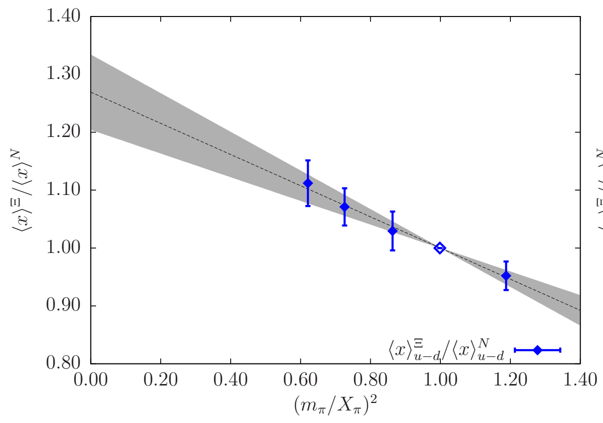
Isospin symmetry is related to the invariance of the QCD Hamiltonian under rotations about the 2-axis in isospace, turning u quarks to d and protons to neutrons. Extensive studies in nuclear systems have shown that it is an excellent symmetry [113], typically accurate to a fraction of a percent (e.g. $m_n - m_p \sim 0.1\%$). There has been extensive theoretical work on the effect of the $u - d$ mass difference on parton distribution functions, where isospin



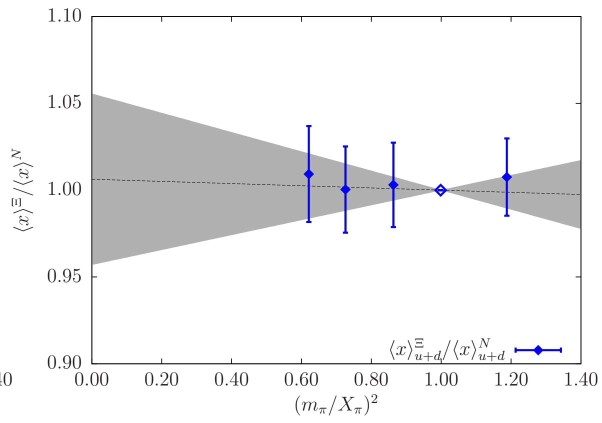
(a) Contribution u



(b) Contribution d



(c) Contribution $u-d$



(d) Contribution $u+d$

Figure 9.5 – Ratio $\langle x\rangle^{\Xi}_{q}/\langle x\rangle^{N}_{q}$

symmetry implies [114, 115]:

$$u^p(x,Q^2) = d^n(x,Q^2), \quad d^p(x,Q^2) = u^n(x,Q^2)\ . \tag{9.7}$$

Within the MIT bag model, Sather [116] and Rodionov *et* al. [117] found that charge symmetry violation (CSV) in the singly represented valence sector, $\delta d(x) \equiv d^p(x) - u^n(x)$, could be as large as 5% in the intermediate to large range of Bjorken x. Furthermore, these authors also found that $\delta u(x) \equiv u^p(x) - d^n(x)$ was similar in magnitude but of opposite sign.

The isospin symmetry breaking arising from the $u-d$ mass difference was deduced by studying the second moments of the parton distribution functions. The sign and magnitude of the effect found in this work are consistent both with the estimates based on the MIT bag model [118] and with the best fit global determination of Ref. [119]. However,

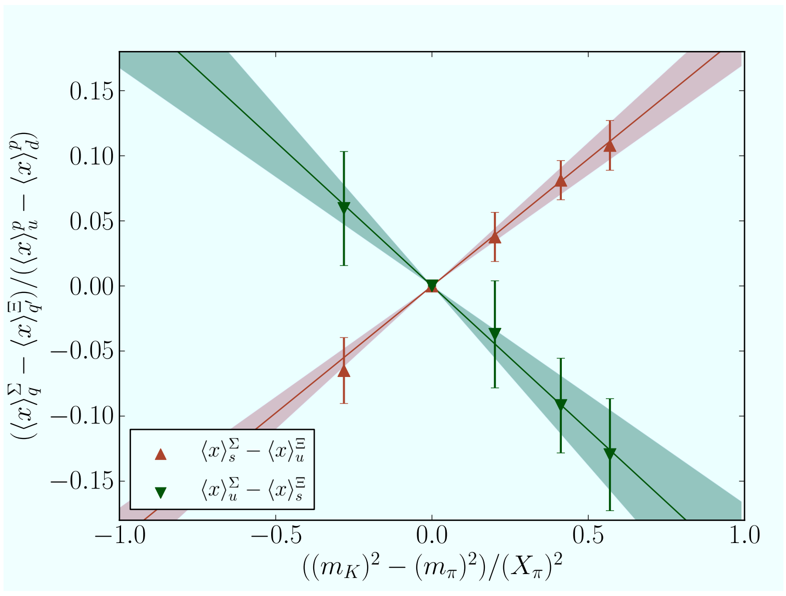


Figure 9.6 – The difference between the doubly and singly represented quarks in the Σ and Ξ as a function of the strange/light quark mass difference. δu and δd was deduced from the slopes of these curves, respectively (c.f. Eqs. (9.18) and (9.19)).

the uncertainties in this work are considerably smaller than those derived from the global analysis.

Because of valence quark normalisation, the first moments of $\delta u^-(x)$ and $\delta d^-(x)$ must vanish. Hence the second moment (labeled δq^-) is the first place where isospin symmetry breaking can be visible in the valence quark distributions,

$$\delta u^- = \int_0^1 dx\, x(u^{p-}(x) - d^{n-}(x)) = \langle x\rangle_{u^-}^p - \langle x\rangle_{d^-}^n\,, \tag{9.8}$$

$$\delta d^- = \int_0^1 dx\, x(d^{p-}(x) - u^{n-}(x)) = \langle x\rangle_{d^-}^p - \langle x\rangle_{u^-}^n\,. \tag{9.9}$$

As detailed below, these isospin symmetry breaking momentum fractions are related to the hyperon moments by

$$\delta u^- \sim \langle x\rangle_{u^-}^\Sigma - \langle x\rangle_{s^-}^\Xi \tag{9.10}$$

$$\delta d^- \sim \langle x\rangle_{s^-}^\Sigma - \langle x\rangle_{u^-}^\Xi\,, \tag{9.11}$$

in the limit where the strange and light quarks have almost equal mass.

The operators used for determining the quark momentum fractions need to be renormalised, preferably using a nonperturbative method such as RI′-MOM [120, 121, 122]. Here, only ratios of matrix elements are required and hence these renormalisation factors cancel.

At this stage, is is pointed out that one caveat of the calculations is that the second ($n = 2$) moment as calculated on the lattice is a C-even moment, while we actually require the C-odd moments in Eqs. (9.8) and (9.9). Secondly, it is well known that lattice results for the second moment of the iso-vector nucleon PDFs, $\langle x\rangle_{u-d}$, do not agree well with experiment (see e.g. [123]). Based on chiral perturbation theory it is expected that finite size effects and chiral corrections are potentially large [124, 125, 126, 127], but this has so far not been confirmed by lattice calculations. This discrepancy may also be due to a mismatch of lattice nucleon matrix elements and perturbative Wilson coefficients. However, ratios of moments of PDFs are considered, in which such effects cancel out. For example, one finds $\langle x\rangle^p_u/\langle x\rangle^p_d \approx 2.3$ in good agreement with $\langle x\rangle^p_{u^-}/\langle x\rangle^p_{d^-} = 2.40(6)$ found in [105]. Is seems encouraging that lattice results for the ratio $\langle x\rangle_{(u-d)}/\langle x\rangle_{(\Delta u-\Delta d)}$ agree well with experiment [128].

Fig. 9.4a (9.5b) shows the ratio of the $u(s)$-quark momentum fraction of the $\Sigma(\Xi)$ baryon to the momentum fraction of the u in the proton. They are also given in Tab. B.12, as a function of m_π^2, normalised with the singlet quantity defined in Eq. 8.9. Here one sees the strong effect of the decrease (increase) in the light (strange) quark momentum fractions as one approaches the physical point. In particular, one sees that the heavier strange quark in the Ξ^0 carries a larger momentum fraction than the up quark in the proton. It was also noticed that the up quark in the Σ^+ has a smaller momentum fraction than the up quark in the proton. This is a purely environmental effect since the only difference between these two measurements is the mass of the spectator quark (s in Σ^+, d in p). This implies that the momentum fraction of the strange quark in the Σ should be larger than that of the down quark in the proton, which is exactly what is seen in Fig. 9.4b and 9.5a.

To infer the level of isospin symmetry breaking relevant to the nucleon, one only needs to consider small perturbations about the isospin symmetric point. For instance, one might write

$$\delta u = m_\delta\left(-\frac{\partial\langle x\rangle^p_u}{\partial m_u} + \frac{\partial\langle x\rangle^p_u}{\partial m_d}\right) + \mathcal{O}(m_\delta^2)\,, \tag{9.12}$$

where $m_\delta \equiv (m_d - m_u)$, making use of isospin symmetry by equating $\partial\langle x\rangle_d^n/\partial m_d = \partial\langle x\rangle_u^p/\partial m_u$ and $\partial\langle x\rangle_d^n/\partial m_u = \partial\langle x\rangle_u^p/\partial m_d$. A similar expression holds for δd.

Near the flavour SU(3) symmetric point, it should be noted that the up quark in the proton is equivalent to an up quark in a Σ^+ or a strange quark in a Ξ^0, which are collectively described as the "doubly-represented" quark [129].

The local derivatives required for δu can be obtained by varying the masses of the up and down quarks independently. Within the present calculation, it should be noted that the difference $\langle x\rangle_s^\Xi - \langle x\rangle_u^p$ measures precisely the variation of the doubly-represented quark matrix element with respect to the doubly-represented quark mass (while holding the singly-represented quark mass fixed). Similar variations allow us to evaluate the other required derivatives, where one writes

$$\frac{\partial\langle x\rangle_u^p}{\partial m_u} \simeq \frac{\langle x\rangle_s^{\Xi^0} - \langle x\rangle_u^p}{m_s - m_l}, \; \frac{\partial\langle x\rangle_u^p}{\partial m_d} \simeq \frac{\langle x\rangle_u^{\Sigma^+} - \langle x\rangle_u^p}{m_s - m_l}, \tag{9.13}$$

$$\frac{\partial\langle x\rangle_d^p}{\partial m_u} \simeq \frac{\langle x\rangle_u^{\Xi^0} - \langle x\rangle_d^p}{m_s - m_l}, \; \frac{\partial\langle x\rangle_d^p}{\partial m_d} \simeq \frac{\langle x\rangle_s^{\Sigma^+} - \langle x\rangle_d^p}{m_s - m_l}. \tag{9.14}$$

With these expressions and Eq. (9.12), one obtains the relevant combinations for this work's determination of isospin symmetry breaking in the nucleon

$$\delta u = m_\delta \frac{\langle x\rangle_u^{\Sigma^+} - \langle x\rangle_s^{\Xi^0}}{m_s - m_l}, \; \delta d = m_\delta \frac{\langle x\rangle_s^{\Sigma^+} - \langle x\rangle_u^{\Xi^0}}{m_s - m_l}. \tag{9.15}$$

By invoking the Gell-Mann–Oakes–Renner relation and normalising to the total nucleon isovector quark momentum fraction, one writes

$$\frac{\delta u}{\langle x\rangle_{u-d}^p} = \frac{m_\delta}{\overline{m}_q} \frac{(\langle x\rangle_u^{\Sigma^+} - \langle x\rangle_s^{\Xi^0})/\langle x\rangle_{u-d}^p}{(m_K^2 - m_\pi^2)/X_\pi^2}, \tag{9.16}$$

$$\frac{\delta d}{\langle x\rangle_{u-d}^p} = \frac{m_\delta}{\overline{m}_q} \frac{(\langle x\rangle_s^{\Sigma^+} - \langle x\rangle_u^{\Xi^0})/\langle x\rangle_{u-d}^p}{(m_K^2 - m_\pi^2)/X_\pi^2}. \tag{9.17}$$

Written in this way, the fractional isospin symmetry breaking terms are just the slopes of the curves shown in Fig. 9.6 (evaluated at the symmetry point) multiplied by the ratio $m_\delta/\overline{m}_q$. By fitting the slopes, one obtains

$$\frac{\delta u}{\langle x\rangle_{u-d}^p} = \frac{m_\delta}{\overline{m}_q}(-0.221 \pm 0.054) \tag{9.18}$$

$$\frac{\delta d}{\langle x\rangle_{u-d}^p} = \frac{m_\delta}{\overline{m}_q}(0.195 \pm 0.025) \tag{9.19}$$

Chiral perturbation theory yields the quark mass ratio $m_\delta/\overline{m}_q = 0.066(7)$ [130] and the isovector momentum fraction is experimentally determined to be $\langle x\rangle^p_{u-d} \simeq 0.158$ at $4\,\mathrm{GeV}^2$. Substituting these values into Eqs. (9.18) and (9.19) yields the first lattice QCD estimates of the isospin symmetry breaking momentum fractions

$$\delta u = -0.0023(6), \quad \delta d = 0.0020(3). \tag{9.20}$$

The first observation this work makes is that these results are roughly equal in magnitude and have opposite sign. These values are slightly larger than, but within errors in agreement with, the phenomenological predictions of [117, 131], where within the MIT bag model (at a scale $Q^2 \simeq 4\,\mathrm{GeV}^2$) they found $\delta u^- = -0.0014$ and $\delta d^- = 0.0015$. They are also consistent with the best-fit values of the phenomenological analysis of MRST [119], $\delta u^- = -\delta d^- = -0.002^{+0.009}_{-0.006}$ (90% CL).

While this work's result gives a very clear indication of the sign and magnitude of the isospin symmetry breaking in these moments, the usual caveats are added: a precise quantitative determination will require a detailed study of the finite-volume and discretisation effects as well as a controlled chiral extrapolation. It is noted that "disconnected" insertions have not yet been calculated, however since focus was laid on differences of baryons, these contributions should cancel. Lastly, is was estimated the isospin symmetry breaking associated only with the $u - d$ mass difference. It is important to also find a method to investigate the isospin symmetry breaking induced by electromagnetic effects which is expected [119, 132] to be of a similar size.

Chapter 10

$n = 1$ Moment of Tensor GPDs

Quark helicity flip GPDs are defined as a form factor decomposition of a non-forward nucleon matrix element of a bi-local light cone quark operator involving the $\sigma^{\mu\nu}$-tensor. For the quark part one defines [133]

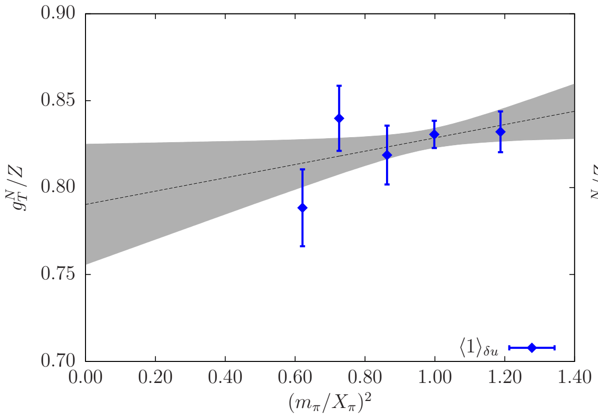
$$\begin{aligned}\langle p', \Lambda'| \int \frac{d\lambda}{4\pi} e^{i\lambda x} \overline{\psi}(-\lambda n/2) i\sigma^{\mu\nu} \psi(\lambda n/2) |p, \Lambda\rangle \\ = \overline{U}(p', \Lambda') \Big(i\sigma^{\mu\nu} H_T(x, \xi, t) + \frac{\gamma^{[\mu} \Delta^{\nu]}}{2m} E_T(x, \xi, t) \\ + \frac{\overline{P}^{[\mu} \Delta^{\nu]}}{m^2} \widetilde{H}_T(x, \xi, t) + \frac{\gamma^{[\mu} \overline{P}^{\nu]}}{m} \widetilde{E}_T(x, \xi, t) \Big) U(p, \Lambda) \qquad (10.1)\end{aligned}$$

with $\overline{P} = (P' + P)/2$ and n is a light-like vector. Here we dropped for simplicity the dependence on the resolution scale Q^2 as well as the gauge links rendering the bi-local operators gauge invariant. The first of these tensor GPDs, $H_T(x, \xi, t)$, is called generalised transversity, because it reproduces the transversity distribution in the forward limit $H_T(x, 0, 0) = \delta q(x) = h_1(x)$. Integrating $H_T(x, \xi, t)$ over x gives the tensor form factor

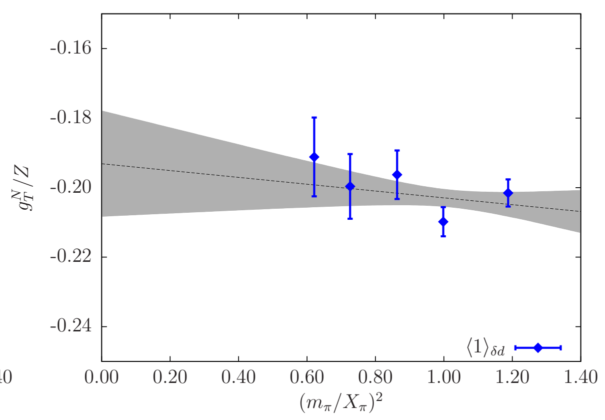
$$\int_{-1}^{1} dx H_T(x, \xi, t) \Big|_{\xi=0} = g_T(t) \qquad (10.2)$$

whose forward limit is known as the tensor charge, g_T.

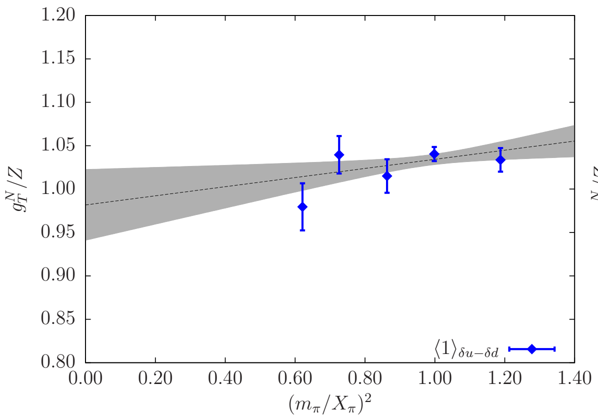
Typically lattice calculations of moments of GPDs do not take into account the computationally expensive quark-line disconnected contributions, see Sec. 4.17. Since the tensor operators flip the quark helicity, these disconnected diagrams do not contribute in the continuum theory for vanishing quark masses. Therefore, we expect only small contributions



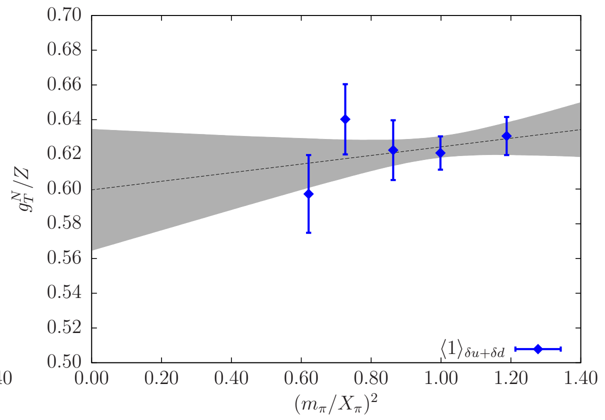
(a) Contribution δu



(b) Contribution δd



(c) Contribution $\delta u - \delta d$



(d) Contribution $\delta u + \delta d$

Figure 10.1 – First ($n = 1$) moment of transversity PDF $\langle 1\rangle^N_{\delta q}$.

for the disconnected graphs in our calculation. This expectation is supported by numerical results with Lattice QCD [134].

On the lattice, it is not possible to deal directly with matrix elements of bi-local light-cone operators. The lhs of Eq. (10.1) is transfered to Mellin space by integrating over x, i.e. $\int_{-1}^{1} dx x^{n-1}$. For the nucleon matrix elements one gets the local tensor operators

$$\mathcal{O}_T^{\mu\nu\mu_1\dots\mu_{n-1}}(0) = \overline{\psi}(0) i\sigma^{\mu\{\nu} i\overleftrightarrow{D}^{\mu_1} \dots i\overleftrightarrow{D}^{\mu_{n-1}\}}\psi(0) \tag{10.3}$$

which are in turn parameterised in terms of the tensor generalised form factors (GFFs) A_{Tni}, B_{Tni}, $\widetilde{A}_{Tni}$, $\widetilde{B}_{Tni}$. Here and in the following, $\overleftrightarrow{D} = 1/2(\overrightarrow{D} - \overleftarrow{D})$ and $\{\cdots\}$ indicates symmetrisation of indices and subtraction of traces. Parameterisation for arbitrary n can be found in [135, 136].

For $n = 1$ we have

$$\langle P', \Lambda'|\overline{\psi}(0) i\sigma^{\mu\nu}\psi(0)|P,\Lambda\rangle = \overline{U}(P',\Lambda')\Big(i\sigma^{\mu\nu}A_{T10}(t) + \frac{\overline{P}^{[\mu}\Delta^{\nu]}}{m^2}\widetilde{A}_{T10}(t) + \frac{\gamma^{[\mu}\Delta^{\nu]}}{2m}B_{T10}(t)\Big) U(P,\Lambda) \tag{10.4}$$

where we choose $(\mu, \nu) = (3, 4)$. The inclusion of an additional term $\propto \gamma^{[\mu}\overline{P}^{\nu]} = \gamma^\mu\overline{P}^\nu - \gamma^\nu\overline{P}^\mu$ is forbidden by time reversal symmetry [133] and $A_{T10}(t)$ is identified as the tensor form factor $g_T(t)$ from Eq. (10.2).

On the lattice we extract the matrix elements from ratios of nucleon two- and three-point correlation functions as given in Eq. (4.163). To extract $\langle 1\rangle_{\delta q}$ we use

$$R^{\mu\nu} = \frac{C_{\mathcal{O}}^{\mathrm{3pt}\,\mu\nu}\left(\tau, P' = (m,\vec{0}), P = (m,\vec{0})\right)}{C^{\mathrm{2pt}}\left(\tau_{\mathrm{snk}}, P = (m,\vec{0})\right)} \tag{10.5}$$

$$= \frac{1}{(2\kappa_l)(2\kappa_s)}\frac{m}{4}\langle 1\rangle_{\delta q} \tag{10.6}$$

where $\mu\nu$ represents the operator $\overline{\psi}i\sigma^{\mu\nu}\psi$ and C^{2pt} and $C_{\mathcal{O}}^{\mathrm{3pt}\,\mu\nu}$ are lattice two and three-point functions, respectively, with total momentum $\vec{p}$ (in our simulation we consider only $\vec{p} = 0$). The operator $\mathcal{O}_T^{\mu\nu}$ from Eq. (10.3) is inserted into the three-point function $C_{\mathcal{O}}^{\mathrm{3pt}\,\mu\nu}$ at time τ between the baryon source located at time $t = 0$ and sink at time t.

10.1 Nucleon

Figs. 10.1a - 10.1d show the $n = 1$ moment of the transversity PDF. Since we do not have the renormalisation constant Z we show the unrenormalised quantities. The individual contributions $\langle 1\rangle_{\delta u}$ and $\langle 1\rangle_{\delta d}$, and the combined contributions $\langle 1\rangle_{\delta u-\delta d}$ and $\langle 1\rangle_{\delta u+\delta d}$ are displayed separately. We give the measurement data for each of the contributions in Tab. B.13.

Again we try a first approximation with a linear two-parameter fit to the physical point

$$\langle 1\rangle_{\delta q}^{B}/Z = a_0 + a_1 m_\pi^2. \tag{10.7}$$

The fit results and extrapolations to the physical point for each of the contributions are given separately in the upper part of Tab. C.7. The forward moments at the physical point

B	f_T^{theor}	δu	δd	$\delta u - \delta d$	$\delta u + \delta d$
N	0.168(3) MeV [78]	1.03(12)	-0.248(34)	1.28(14)	0.774(97)
Σ	0.168(3) MeV [78]	1.04(12)	-0.275(33)	1.31(14)	0.774(95)
Ξ	0.168(3) MeV [78]	1.15(12)	-0.255(33)	1.39(15)	0.912(98)
N	0.140(5) MeV [137]	0.86(10)	-0.207(29)	1.06(13)	0.645(83)
Σ	0.140(5) MeV [137]	0.86(10)	-0.229(29)	1.09(13)	0.645(82)
Ξ	0.140(5) MeV [137]	0.95(10)	-0.212(29)	1.16(13)	0.760(85)

Table 10.1 – Estimate of the $n = 1$ moment of the tensor GPD $g_T^{\text{est}} = \frac{Z_V g_T}{Z_V f_T} f_T^{\text{theor}}$ for N, Σ, and Ξ.

are found to be:

$$\begin{aligned}
\langle 1 \rangle^N_{\delta u}/Z_T &= 0.797(35) \\
\langle 1 \rangle^N_{\delta d}/Z_T &= -0.195(15) \\
\langle 1 \rangle^N_{\delta u - \delta d}/Z_T &= 0.991(42) \\
\langle 1 \rangle^N_{\delta u + \delta d}/Z_T &= 0.604(35)
\end{aligned}$$

Since these are unrenormalised quantity we can not compare directly to experiment, neither we can compare to other lattice results. Instead we take the ratio of the tensor charge over the coupling constant of the vector meson to the tensor current. In this ratio the renormalisation constant cancels and with previous lattice results we can give an estimate of the tensor charge

$$g_T^{\text{est}} = \frac{Z_T g_T}{Z_T f_T} f_T^{\text{theor}}. \tag{10.8}$$

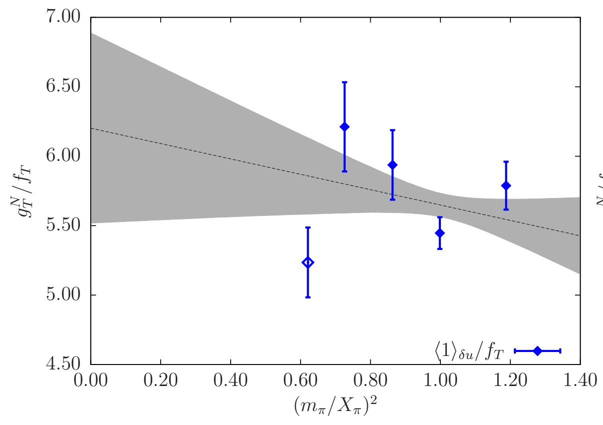
We determine the coupling f_T of the light vector mesons to the tensor current. The coupling f_T can only be determined theoretically. On the lattice we calculate

$$e(\lambda)_i m_V f_T = \langle 0 | T_{\mu\nu} | V, \lambda \rangle \tag{10.9}$$

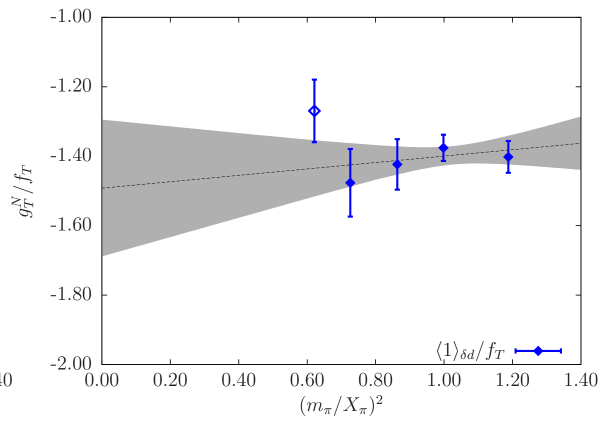
with the tensor current

$$T_{\mu\nu} = \overline{\psi}_{f_2} \sigma_{\mu\nu} \psi_{f_1}. \tag{10.10}$$

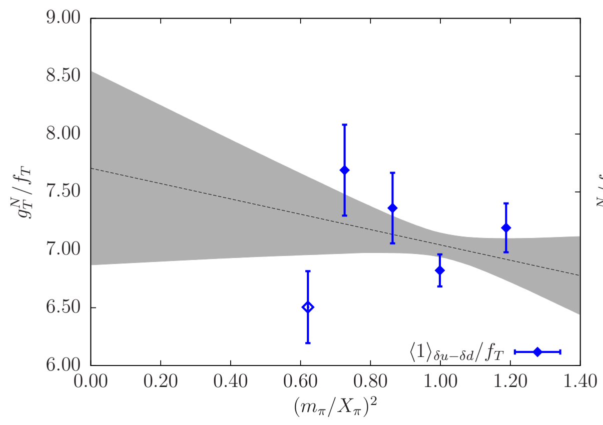
We note here that this coupling does not include the improvement terms.



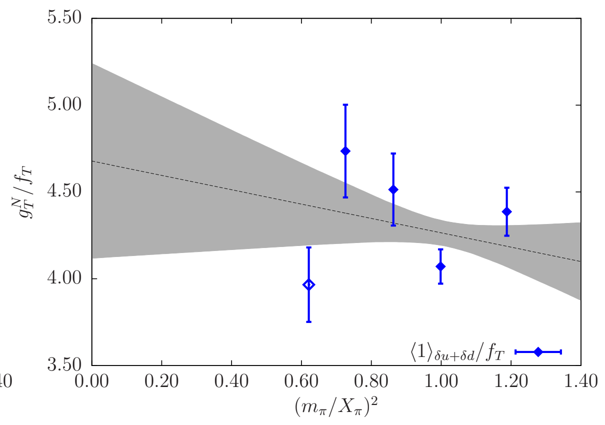
(a) Contribution δu



(b) Contribution δd



(c) Contribution $\delta u - \delta d$



(d) Contribution $\delta u + \delta d$

Figure 10.2 – Ratio $\langle 1\rangle^N_{\delta q}/f_T$. The renormalisation constant cancels. We use f_T of the light meson.

Figs. 10.2a - 10.2d show the ratio g_T/f_T for the nucleon. We use f_T of the light meson. The individual contributions δu and δd, and the combined contributions $\delta u - \delta d$ and $\delta u + \delta d$ are displayed separately.

The measurement data for each of the contributions is given in Tab. B.16. The fit results and extrapolations to the physical point for each of the contributions are given separately in the upper part of Tab. C.8.

The QCDSF/UKQCD Collaboration used an $N_f = 2$ flavour, non-perturbatively $\mathcal{O}(a)$ improved Wilson Clover action and determine the vector meson coupling constant to $f_T = 168(3)$ MeV [78]. The UKQCD/RBC Collaboration used the Iwasaki gauge action and simulated $N_f = 2+1$ flavours of domain wall fermions and determine the ratio f_T/f_V and together with the experimental value for f_V they arrive at $f_T = 140(5)$ MeV [137]. Since the results for both collaborations do not agree we use either of both values to derive an estimate for g_T.

Tab. 10.1 shows the estimates for g_T for the nucleon using the above mentioned theoretical determinations of f_T.

Our results for $N_f = 2 + 1$ simulations are quite similar to earlier $N_f = 2$ results, i.e. g_T is flat as a function of m_π^2 [138, 74].

We now compare the nonsinglet combination $\delta u - \delta d$ of g_T and g_A, because in the non-relativistic limit one expects that both quantities agree. Our estimate for the nucleon axial charge is $g_A = 1.141(75)$, see Sec 8.1. Comparing to our estimates for the nucleon tensor charge as shown in Tab. 10.1 we find good agreement indicating that the quarks are not very relativistic. This is not surprising since our simulations are carried out at rather large quark masses. This result is supported by an earlier lattice calculation [138].

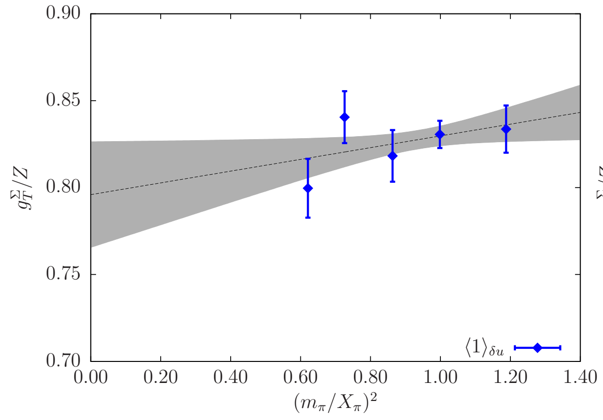
10.2 Hyperons

Figs. 10.3a - 10.3d (10.4a - 10.4d) show the unrenormalised tensor charge for the Σ (Ξ). Again we display the individual contributions in separate figures. The measurement data for the individual contributions for the Σ (Ξ) is given in Tab. B.14 (B.15).

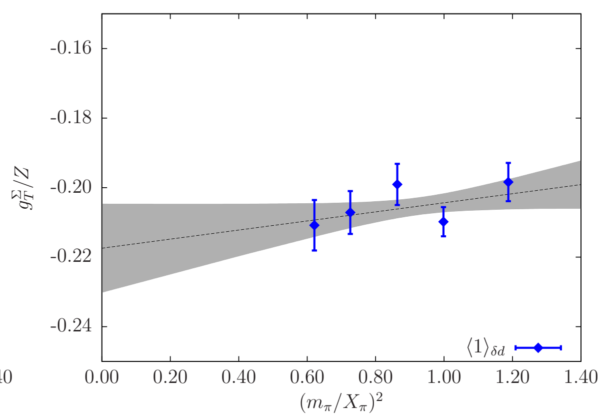
We carry out a first (linear) extrapolation to the physical point using Eq. 10.7. The fit results and extrapolations to the physical point for each of the contributions are given separately in the middle (lower) part of Tab. C.7 for the Σ (Ξ).

For the Σ we find:

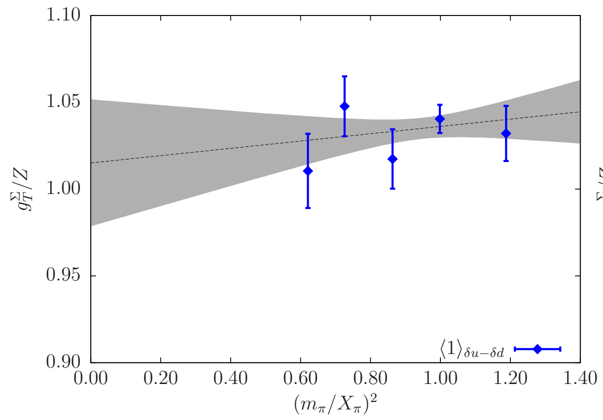
$$\begin{aligned}
\langle 1\rangle^{\Sigma}_{\delta u}/Z_T &= \quad 0.802(31) \\
\langle 1\rangle^{\Sigma}_{\delta d}/Z_T &= \quad -0.215(13) \\
\langle 1\rangle^{\Sigma}_{\delta u-\delta d}/Z_T &= \quad 1.019(37) \\
\langle 1\rangle^{\Sigma}_{\delta u+\delta d}/Z_T &= \quad 0.582(29)
\end{aligned}$$



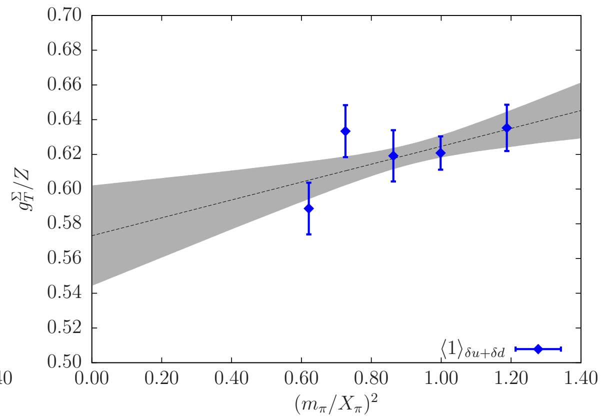
(a) Contribution δu



(b) Contribution δd



(c) Contribution $\delta u - \delta d$



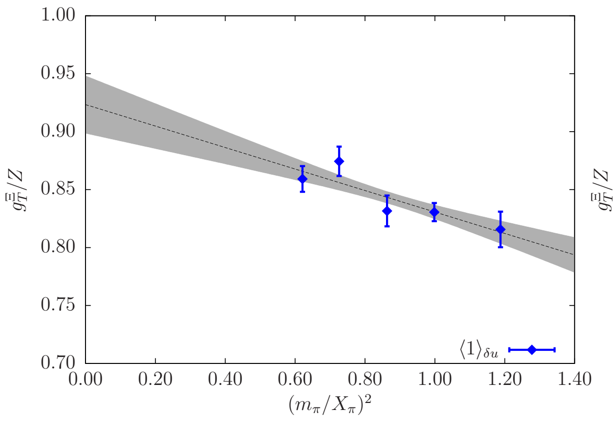
(d) Contribution $\delta u + \delta d$

Figure 10.3 – First ($n = 1$) moment of transversity PDF $\langle 1\rangle^\Sigma_{\delta q}$.

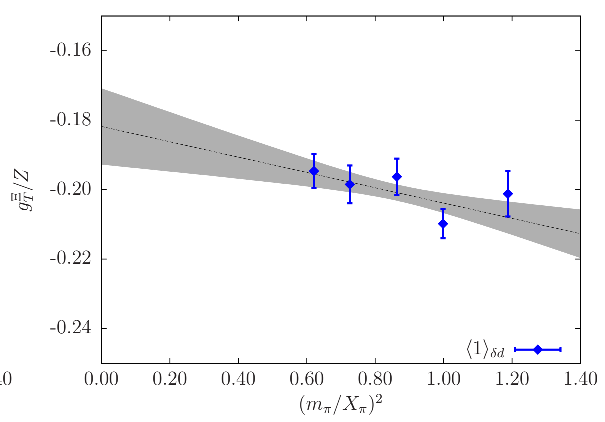
Whereas for the Ξ we find:

$$\begin{aligned}
\langle 1\rangle^{\Xi}_{\delta u}/Z_T &= \quad 0.908(25)\\
\langle 1\rangle^{\Xi}_{\delta d}/Z_T &= \quad \text{-}0.186(11)\\
\langle 1\rangle^{\Xi}_{\delta u-\delta d}/Z_T &= \quad 1.091(30)\\
\langle 1\rangle^{\Xi}_{\delta u+\delta d}/Z_T &= \quad 0.723(25)
\end{aligned}$$

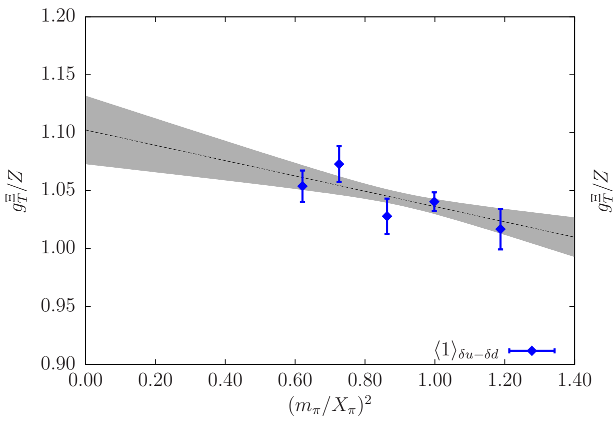
Figs. 10.5a - 10.5d (10.6a - 10.6d) show the ratio g_T/f_T for the Σ (Ξ). The individual contributions δu and δd, and the combined contributions $\delta u - \delta d$ and $\delta u + \delta d$ are displayed separately. The measurement data for each of the contributions is given in Tab. B.17 for the Σ and in Tab. B.18 for the Ξ. The fit results and extrapolations to the physical point for each of the contributions are given separately in the middle (lower) part of Tab. C.8 for the Σ (Ξ).



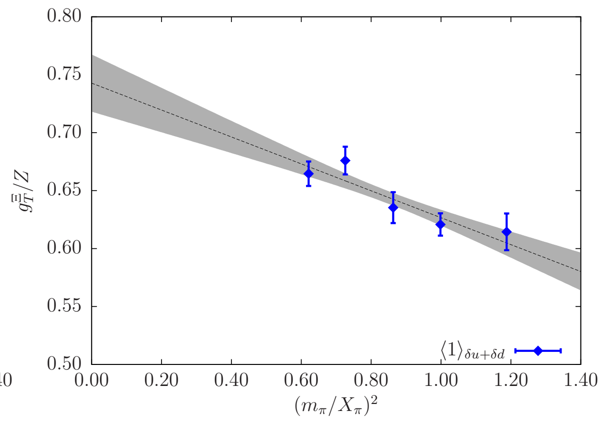
(a) Contribution δu



(b) Contribution δd



(c) Contribution $\delta u - \delta d$



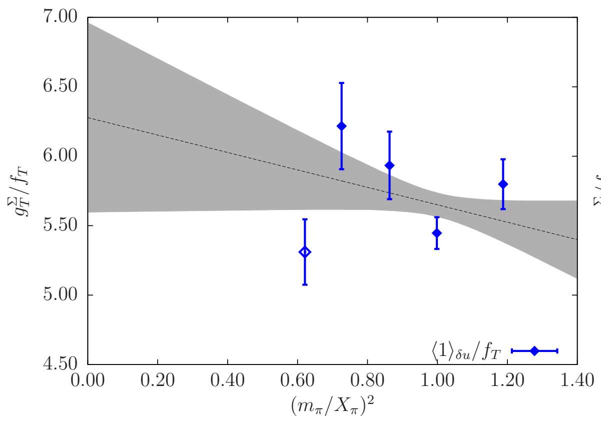
(d) Contribution $\delta u + \delta d$

Figure 10.4 – First ($n = 1$) moment of transversity PDF $\langle 1 \rangle^{\Xi}_{\delta q}$.

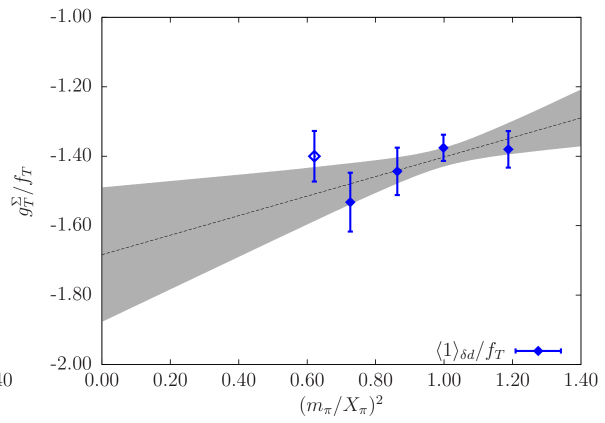
For these quantities nothing is known neither from experiment nor from theory. These are the first predictions.

Tab. 10.1 shows the estimates for $\langle 1 \rangle^{B}_{\delta q}$ for $B = \Sigma, \Xi$.

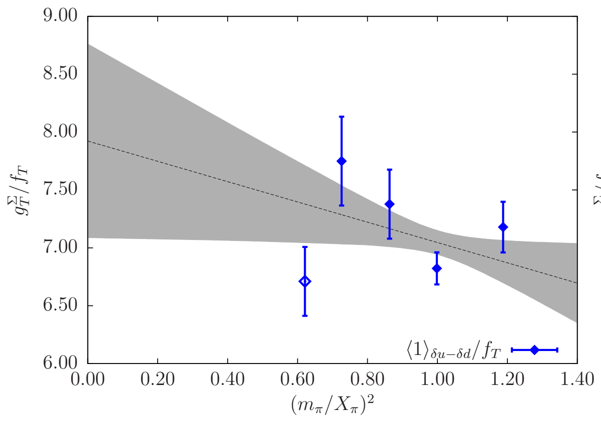
Whereas the contributions coming from the light quarks in the hyperons do not differ much from the according contributions in the nucleon, the strange quark contributions differ. The non-singlet combination $\delta u - \delta d$ is slightly increasing when considering the N, Σ, and Ξ.



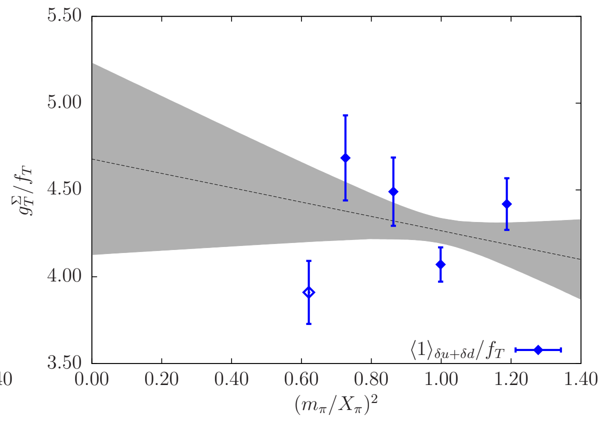
(a) Contribution δu



(b) Contribution δd

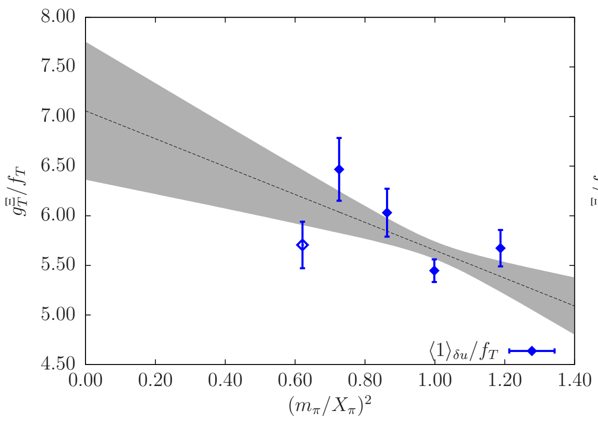


(c) Contribution $\delta u - \delta d$

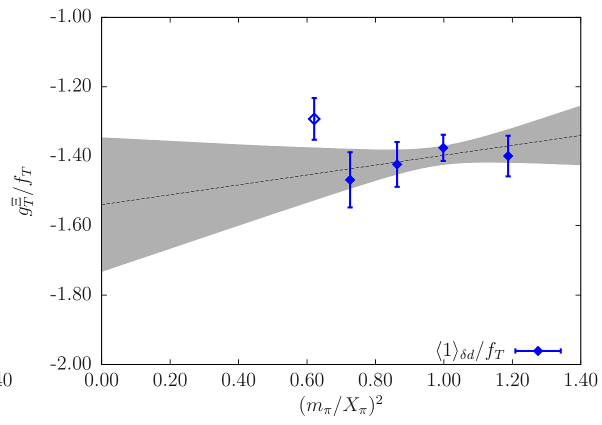


(d) Contribution $\delta u + \delta d$

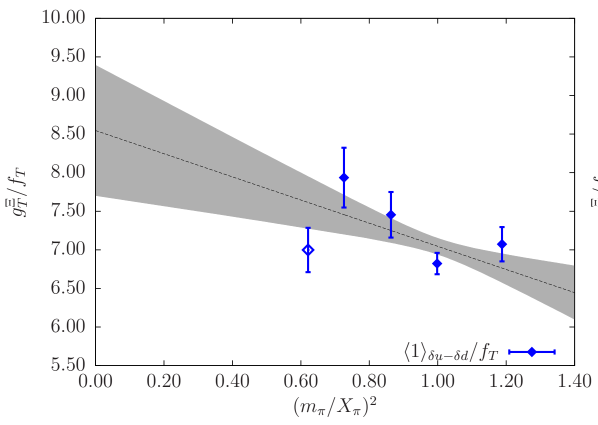
Figure 10.5 – Ratio $\langle 1\rangle^{\Sigma}_{\delta q}/f_T$. The renormalisation constant cancels. We use f_T of the light meson.



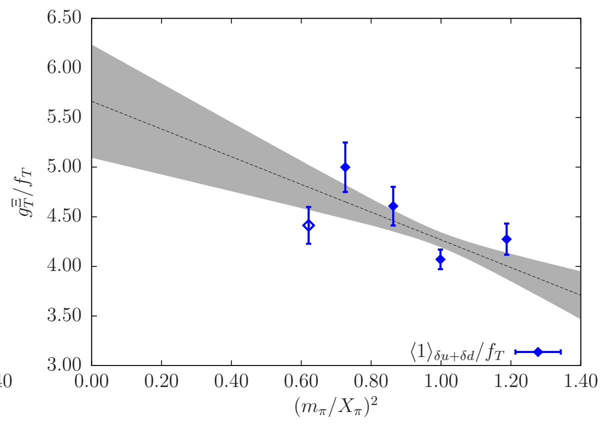
(a) Contribution δu



(b) Contribution δd



(c) Contribution $\delta u - \delta d$



(d) Contribution $\delta u + \delta d$

Figure 10.6 – Ratio $\langle 1\rangle_{\delta q}^{\Xi}/f_T$ for Ξ. The renormalisation constant cancels. We use f_T of the light meson.

Chapter 11

$n = 2$ Moment of Tensor GPDs

Carrying out a similar calculation as performed in Chap. 10 for $n = 2$ one finds

$$\begin{aligned}
A_{[\mu\nu]} S_{\{\nu\mu_1\}} \langle P', \Lambda' | \overline{\psi}(0) i\sigma^{\mu\nu} i\overset{\leftrightarrow}{D}^{\mu_1} \psi(0) | P, \Lambda \rangle &= A_{[\mu\nu]} S_{\{\nu\mu_1\}} \overline{U}(P', \Lambda') \\
\times \Bigg(i\sigma^{\mu\nu} \overline{P}^{\mu_1} A_{T20}(t) &+ \frac{\overline{P}^{[\mu} \Delta^{\nu]}}{m^2} \overline{P}^{\mu_1} \widetilde{A}_{T20}(t) \\
+ \frac{\gamma^{[\mu} \Delta^{\nu]}}{2m} \overline{P}^{\mu_1} B_{T20}(t) &+ \frac{\gamma^{[\mu} \overline{P}^{\nu]}}{m} \Delta^{\mu_1} \widetilde{B}_{T21}(t) \Bigg) U(P, \Lambda)
\end{aligned} \tag{11.1}$$

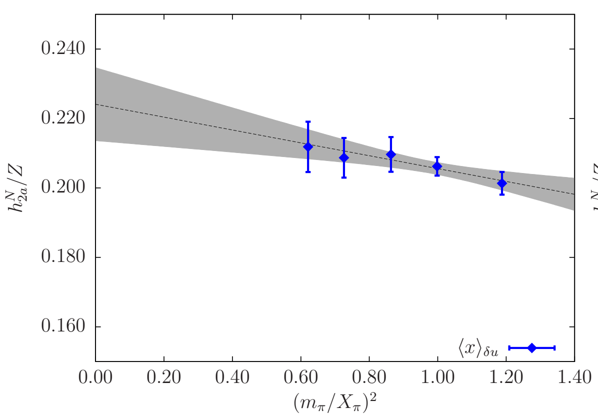
up to trace terms, where $A_{[\mu\nu]}$ and $S_{\{\mu\nu\}}$ denote anti-symmetrisation and symmetrisation of (μ, ν), respectively.

On the lattice this work extracts the matrix elements from ratios of nucleon two- and three-point correlation functions as given in Eqs. (4.163). To extract the forward matrix element from which one can determine $\langle x \rangle_{\delta q}$, this work uses

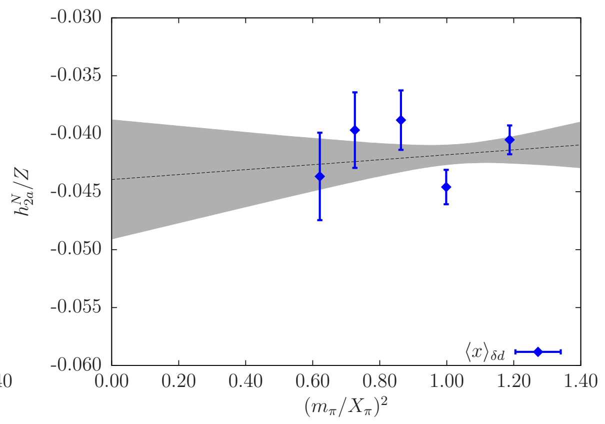
$$R^{2\{34\}} = \frac{C_{\mathcal{O}}^{3\text{pt}\,2\{34\}} \left(\tau, P' = (m, \vec{0}), P = (m, \vec{0}) \right)}{C^{2\text{pt}} \left(\tau_{\text{snk}}, P = (m, \vec{0}) \right)} \tag{11.2}$$

$$= \frac{1}{(2\kappa_l)(2\kappa_s)} \frac{m}{4} \langle x \rangle_{\delta q} \tag{11.3}$$

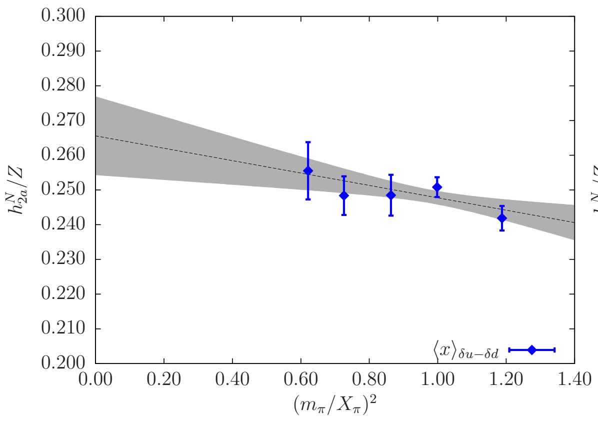
where $2\{34\}$ represents the operator $\overline{\psi} \sigma^{2\{3} \overset{\leftrightarrow}{D}^{4\}} \psi$ and the baryon is polarised in the z-direction. $\langle x \rangle_{\delta q}$ is sometimes called h_2.



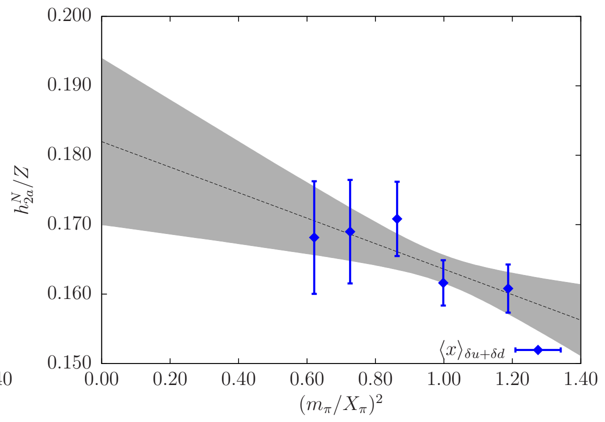
(a) Contribution δu



(b) Contribution δd



(c) Contribution $\delta u - \delta d$



(d) Contribution $\delta u + \delta d$

Figure 11.1 – Second ($n = 2$) moment of transversity PDF $\langle x\rangle_{\delta q}^{N}$.

11.1 Nucleon

Figs. 11.1a - 11.1d show the $n = 2$ moment of nucleon helicity GPD. Since the renormalisation constant Z_{h_2} is not available the unrenormalised quantities are shown. The individual contributions $\langle x\rangle_{\delta u}$ and $\langle x\rangle_{\delta d}$, and the combined contributions $\langle x\rangle_{\delta u-\delta d}$ and $\langle x\rangle_{\delta u+\delta d}$ are displayed separately. This work gives the measurement data for each of the contributions in Tab. B.27.

This work carries out a first approximation with a linear two-parameter fit to the physical point

$$\langle x\rangle_{\delta q}^{B}/Z = a_0 + a_1 m_\pi^2. \tag{11.4}$$

The fit results and extrapolations to the physical point for each of the contributions are given separately in the upper part of Tab. C.10. For the nucleon:

$$\begin{aligned}\langle x\rangle^N_{\delta u}/Z =&\ \ 0.221(11)\\ \langle x\rangle^N_{\delta d}/Z =&\ \ \text{-}0.0436(52)\\ \langle x\rangle^N_{\delta u-\delta d}/Z =&\ \ 0.263(11)\\ \langle x\rangle^N_{\delta u+\delta d}/Z =&\ \ 0.179(12)\end{aligned}$$

To date there has only been one other calculation of this by the QCDSF/UKQCD Collaboration using $N_f = 2$ flavours of $\mathcal{O}(a)$-improved Wilson fermions [139]. Although it is possible to directly compare this work's results to their result due to the missing renormalisation factors one can, however, observe the behaviour of the results as a function of m_π^2 to be similar to their results. In this work little dependence of $\langle x\rangle_{\delta q}$ on the pion mass m_π^2 is observed.

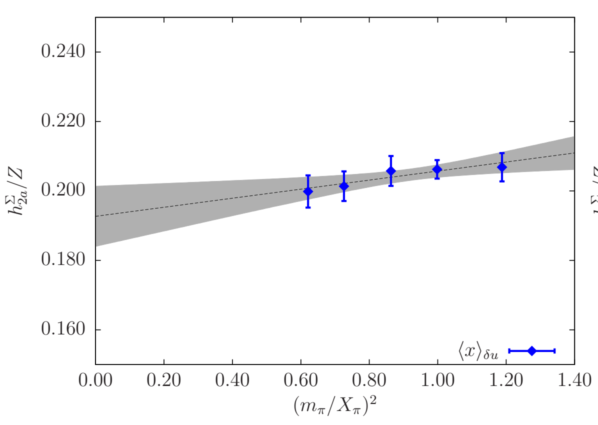
This work compares the ratio $\langle x\rangle^N_{\delta u}/\langle x\rangle^N_{\delta d}$ since here the renormalisation constant cancels. It was obtained $\langle x\rangle^N_{\delta u}/\langle x\rangle^N_{\delta d} = 5.07(65)$ whereas the $N_f = 2$ results [139] yield $\langle x\rangle^N_{\delta u}/\langle x\rangle^N_{\delta d} = 5.15(23)$. Thus, this work's results are in good agreement with their results.

11.2 Hyperons

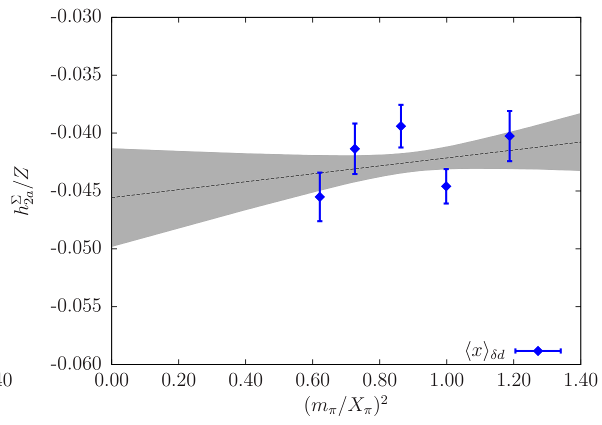
Figs. 11.2a - 11.2d (11.3a - 11.3d) show the unrenormalised $n = 2$ moment of the transversity distribution for the Σ (Ξ). Again this work displays the individual contributions in separate figures. The measurement data for the individual contributions for the Σ (Ξ) is given in Tab. B.28 (B.29).

A first (linear) extrapolation was carried out to the physical point using Eq. 11.4. The fit results and extrapolations to the physical point for each of the contributions are given separately in the middle (lower) part of Tab. C.10 for the Σ (Ξ). For the Σ it was found:

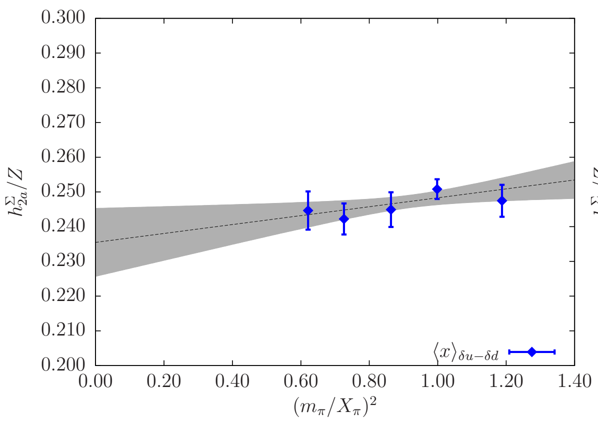
$$\begin{aligned}\langle x\rangle^\Sigma_{\delta u}/Z =&\ \ 0.1949(88)\\ \langle x\rangle^\Sigma_{\delta d}/Z =&\ \ \text{-}0.0450(43)\\ \langle x\rangle^\Sigma_{\delta u-\delta d}/Z =&\ \ 0.2376(100)\\ \langle x\rangle^\Sigma_{\delta u+\delta d}/Z =&\ \ 0.1479(95)\end{aligned}$$



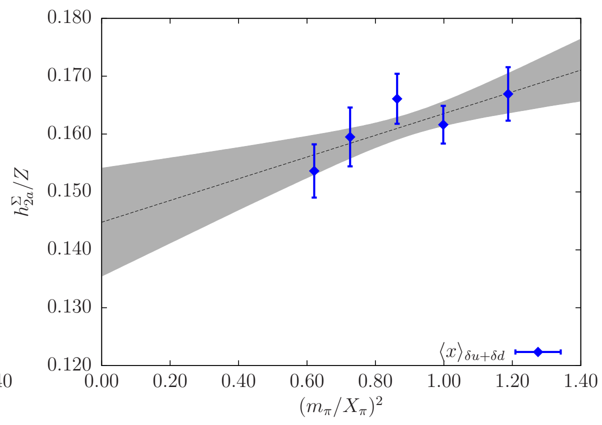
(a) Contribution δu



(b) Contribution δd



(c) Contribution $\delta u - \delta d$



(d) Contribution $\delta u + \delta d$

Figure 11.2 – Second ($n = 2$) moment of transversity PDF $\langle x\rangle_{\delta q}^{\Sigma}$.

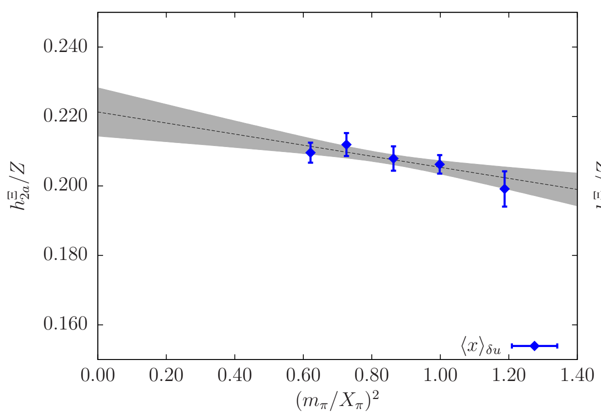
Whereas for the Ξ is was found:

$$\begin{aligned}
\langle x\rangle_{\delta u}^{\Xi}/Z =& \quad 0.2186(71) \\
\langle x\rangle_{\delta d}^{\Xi}/Z =& \quad \text{-}0.0352(34) \\
\langle x\rangle_{\delta u-\delta d}^{\Xi}/Z =& \quad 0.2550(79) \\
\langle x\rangle_{\delta u+\delta d}^{\Xi}/Z =& \quad 0.1852(79)
\end{aligned}$$

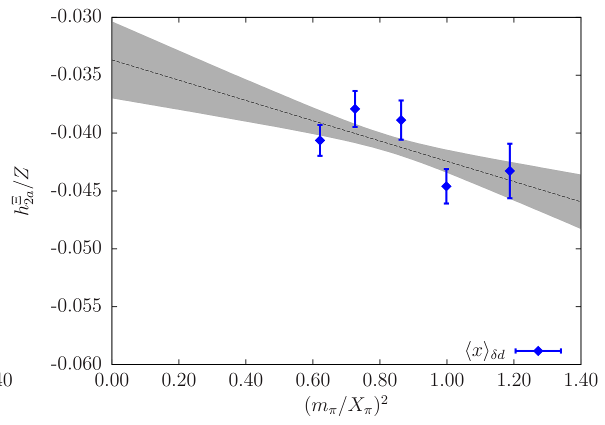
Again the individual quark sectors of the octet baryons are compared. It was found that the contribution from the up-quark in the Σ has a different slope to the up-quark in the N. However the individual quark contributions in N and Ξ are very similar.

11.3 Ratios

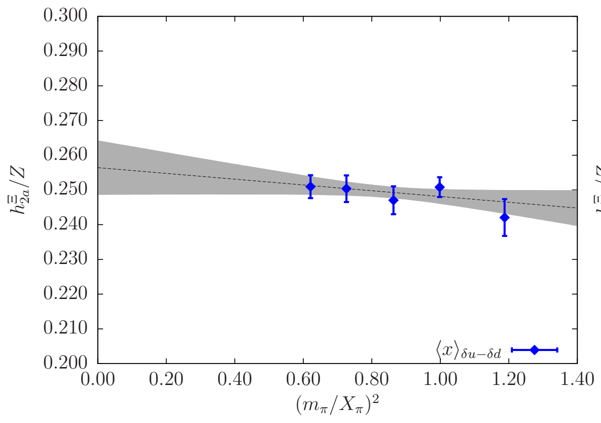
Figs. 11.4a - 11.4d (11.5a - 11.5d) show the ratios $\langle x\rangle_{\delta q}^{B}/\langle x\rangle_{\delta q}^{N}$ with $B = \Sigma$ (Ξ).



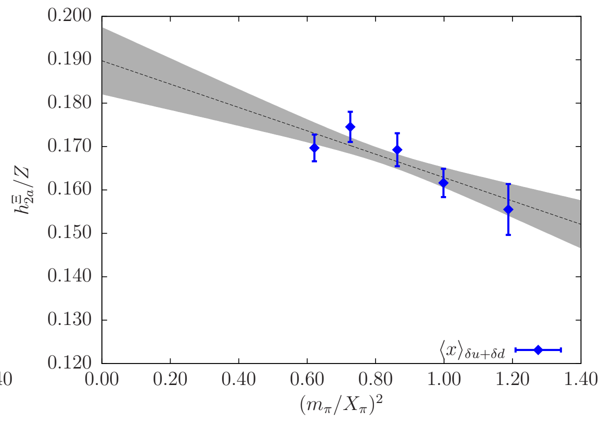
(a) Contribution δu



(b) Contribution δd



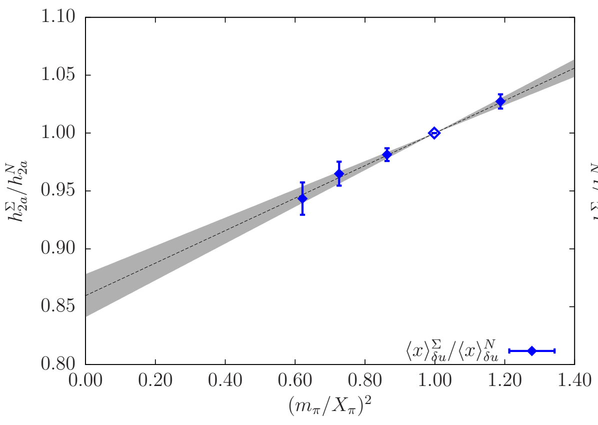
(c) Contribution $\delta u - \delta d$



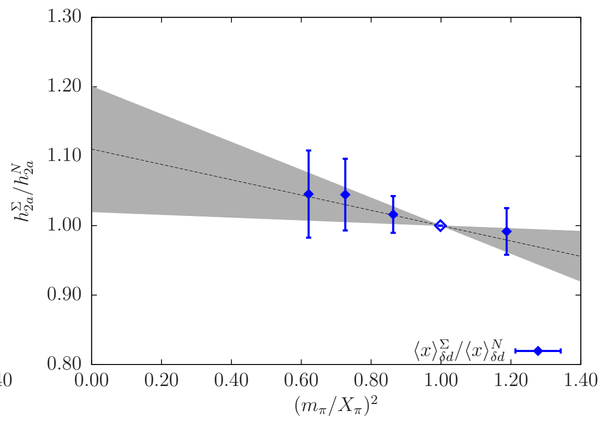
(d) Contribution $\delta u + \delta d$

Figure 11.3 – Second ($n = 2$) moment of transversity PDF $\langle x\rangle_{\delta q}^{\Xi}$.

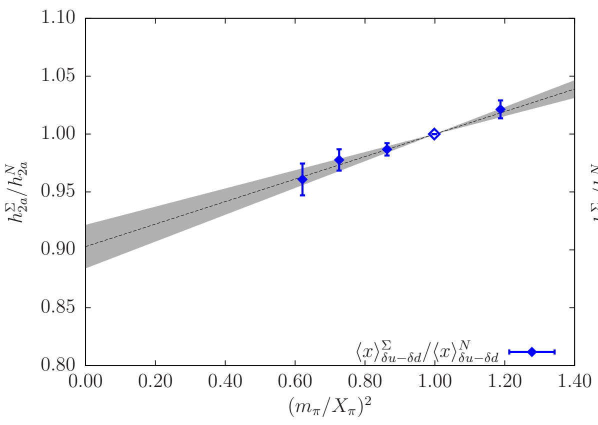
In particular interest is that unlike the unpolarised results presented in Sec. 9.1 and 9.2 there appears to be a difference in the total quark contribution $\langle x\rangle_{\delta u+\delta d}$ between the proton and Σ as seen in Fig. 11.4d.



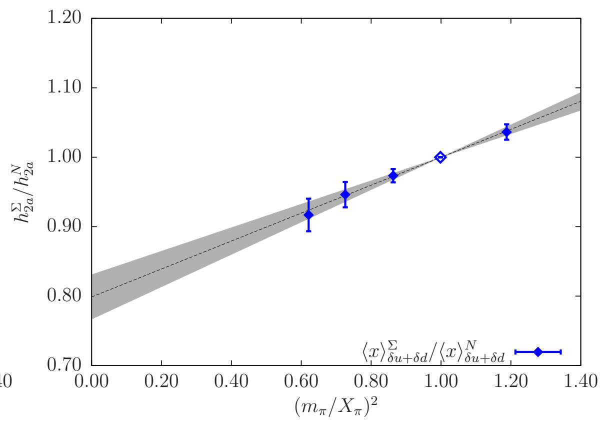
(a) Contribution δu



(b) Contribution δd

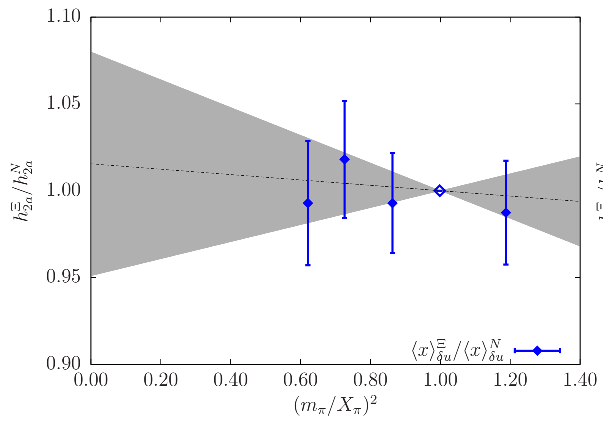


(c) Contribution $\delta u - \delta d$

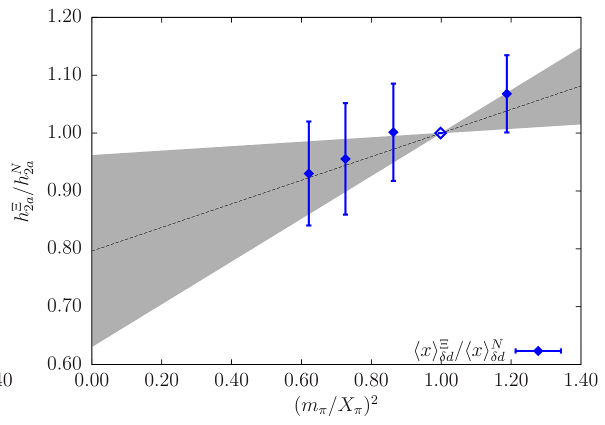


(d) Contribution $\delta d + \delta d$

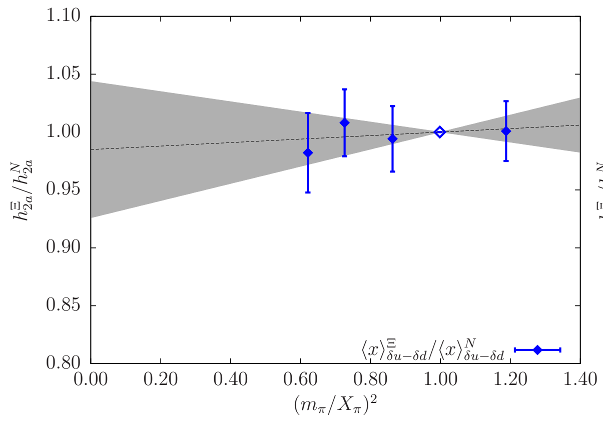
Figure 11.4 – Ratio $\langle x\rangle^{\Sigma}_{\delta q}/\langle x\rangle^{N}_{q}$



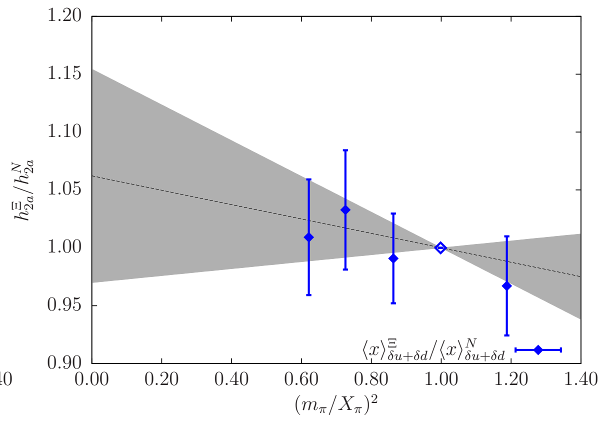
(a) Contribution δu



(b) Contribution δd



(c) Contribution $\delta u - \delta d$



(d) Contribution $\delta d + \delta d$

Figure 11.5 – Ratio $\langle x\rangle_{\delta q}^{\Xi}/\langle x\rangle_{q}^{N}$

Chapter 12

$n = 2$ Moment of Polarised PDF

12.1 Nucleon

Figs. 12.1a - 12.1d show the results for the $n = 2$ moment of the polarised PDF of the nucleon. Since the renormalisation constant Z is not available the unrenormalised quantities are shown. The individual contributions $\langle x \rangle_{\Delta u}$ and $\langle x \rangle_{\Delta d}$, and the combined contributions $\langle x \rangle_{\Delta u - \Delta d}$ and $\langle x \rangle_{\Delta u + \Delta d}$ are displayed separately. This work gives the measurement data for each of the contributions and its error obtained by a bootstrap analysis in Tab. B.19.

The $n = 2$ moment of polarised PDF is sometimes called a_1.

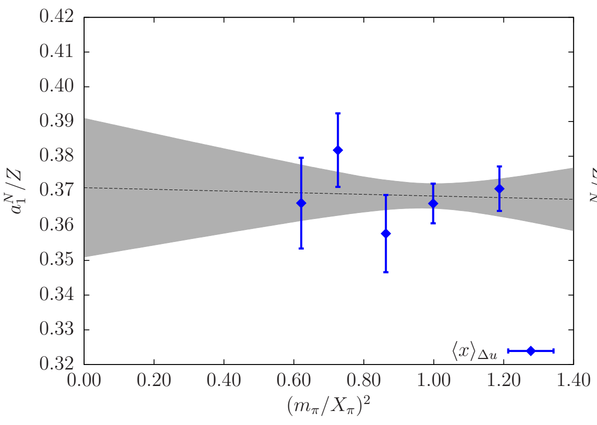
A first approximation with a linear two-parameter fit to the physical point is given

$$\langle x \rangle^{B}_{\Delta q}/Z = a_0 + a_1 m_\pi^2. \tag{12.1}$$

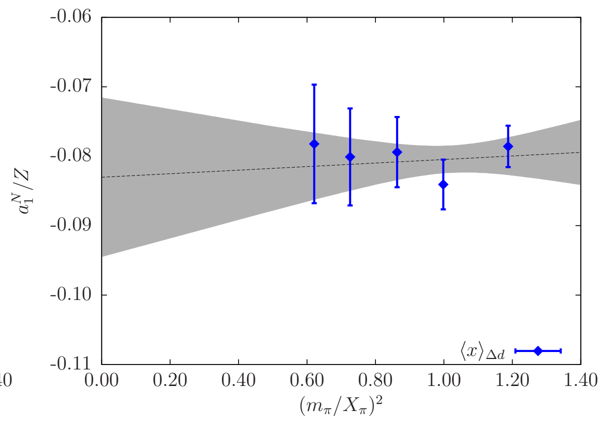
The fit results and extrapolations to the physical point for each of the contributions are given separately in the upper part of Table C.9. For the nucleon it was found:

$$\begin{aligned}
\langle x \rangle^{N}_{\Delta u}/Z = &\quad 0.371(20) \\
\langle x \rangle^{N}_{\Delta d}/Z = &\quad \text{-}0.083(12) \\
\langle x \rangle^{N}_{\Delta u - \Delta d}/Z = &\quad 0.451(23) \\
\langle x \rangle^{N}_{\Delta u + \Delta d}/Z = &\quad 0.287(24)
\end{aligned}$$

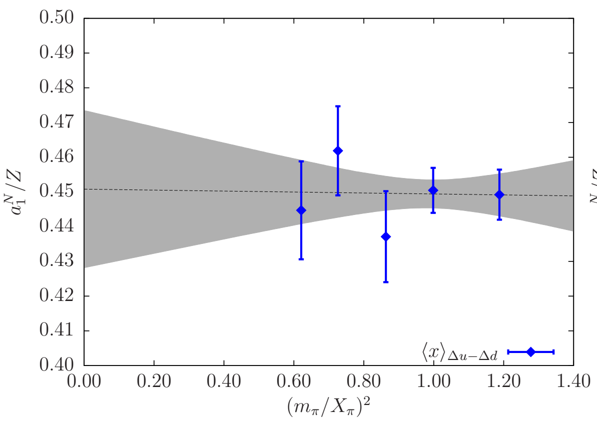
To date there has only been one other calculation of this by the QCDSF/UKQCD Collaboration using $N_f = 2$ $\mathcal{O}(a)$-improved Wilson fermions [139]. This work's results cannot



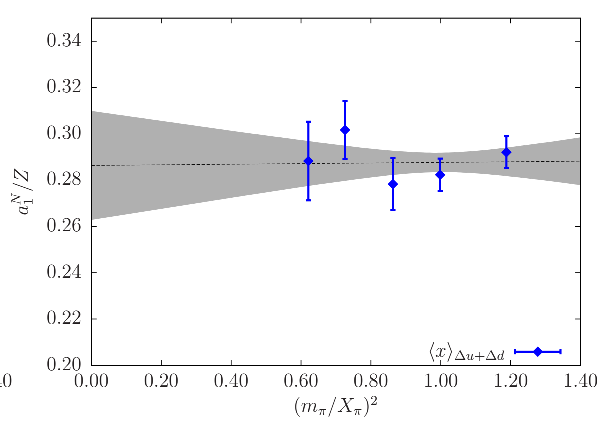
(a) Contribution Δu



(b) Contribution Δd



(c) Combined contribution $\Delta u - \Delta d$



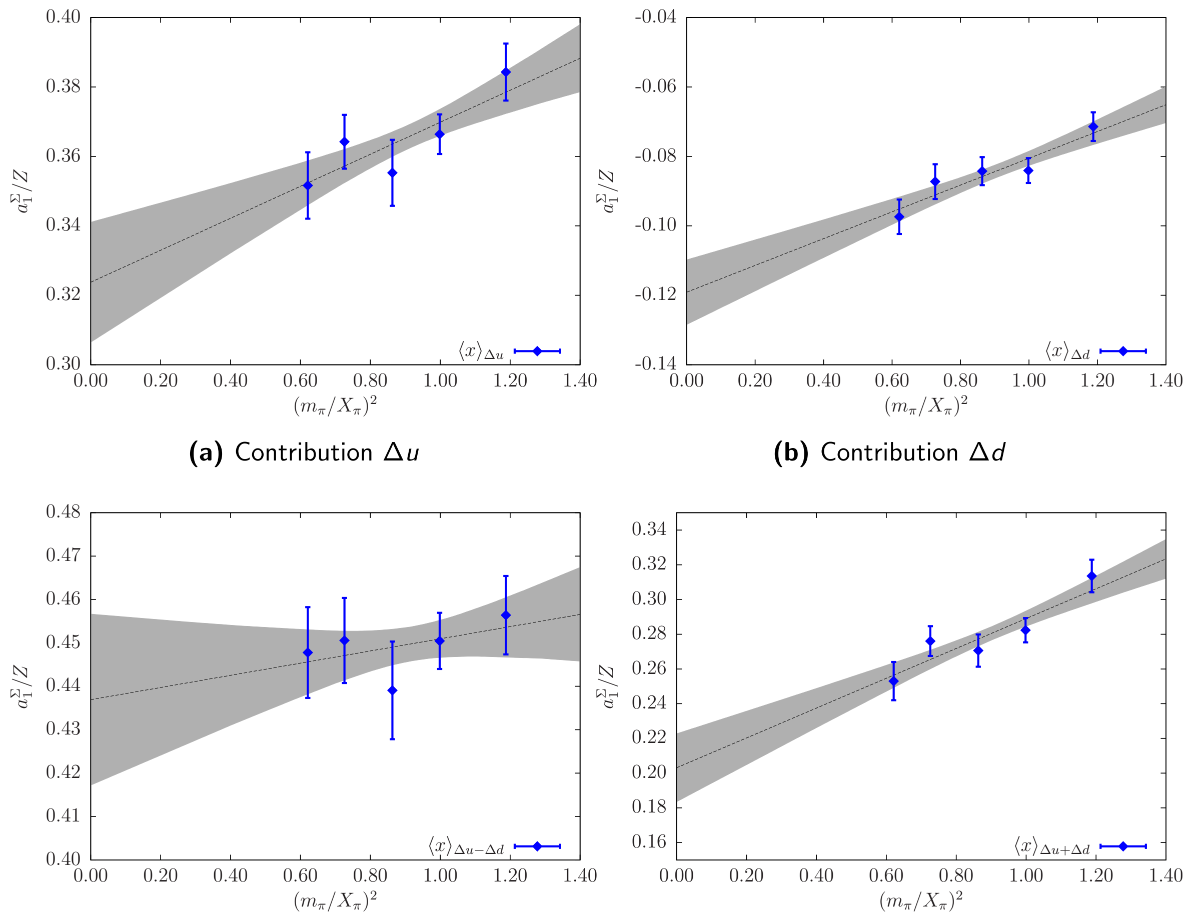
(d) Combined contribution $\Delta u + \Delta d$

Figure 12.1 – Second ($n = 2$) moment of polarised PDF $\langle x\rangle^N_{\Delta q}$.

directly be compared due to the missing renormalisation factors. However the behaviour of this work's results as a function of m_π^2 is similar to their results. Little dependence of $\langle x\rangle_{\Delta q}$ is seen on the pion mass m_π^2.

12.2 Hyperons

Figs. 12.2a - 12.2d (12.3a - 12.3d) show the unrenormalised polarised momentum fractions for the Σ (Ξ). The individual contributions are displayed in separate figures. The measurement data for the individual contributions for the Σ (Ξ) is given in Tab. B.20 (B.21).



(a) Contribution Δu

(b) Contribution Δd

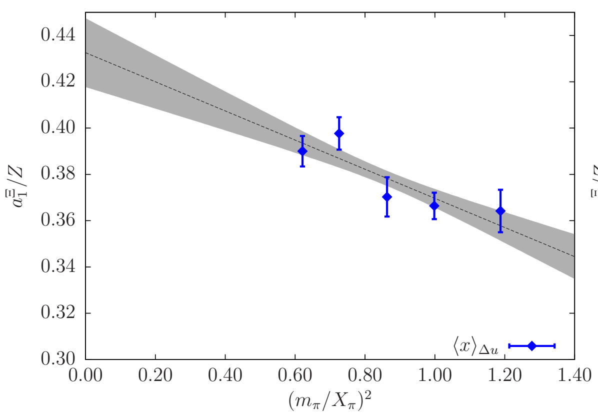
(c) Contribution $\Delta u - \Delta d$

(d) Contribution $\Delta u + \Delta d$

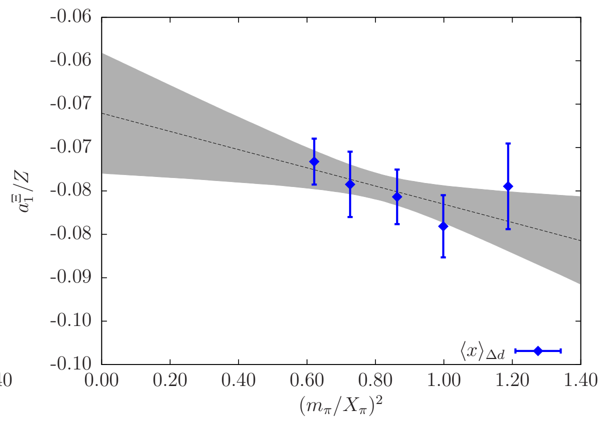
Figure 12.2 – Second ($n = 2$) moment of polarised PDF $\langle x \rangle^{\Sigma}_{\Delta q}$.

This work carried out a first (linear) extrapolation to the physical point using Eq. 12.1. The fit results and extrapolations to the physical point for each of the contributions are given separately in the middle (lower) part of Tab. C.9 for the Σ (Ξ). For the Σ it was found:

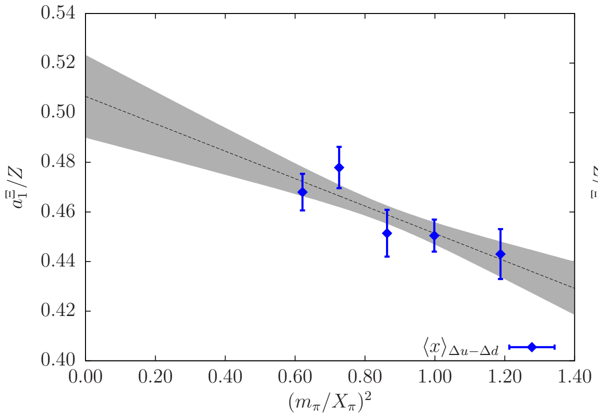
$$
\begin{aligned}
\langle x \rangle^{\Sigma}_{\Delta u}/Z &= \quad 0.332(18) \\
\langle x \rangle^{\Sigma}_{\Delta d}/Z &= \quad \text{-}0.1126(95) \\
\langle x \rangle^{\Sigma}_{\Delta u-\Delta d}/Z &= \quad 0.439(20) \\
\langle x \rangle^{\Sigma}_{\Delta u+\Delta d}/Z &= \quad 0.218(20)
\end{aligned}
$$



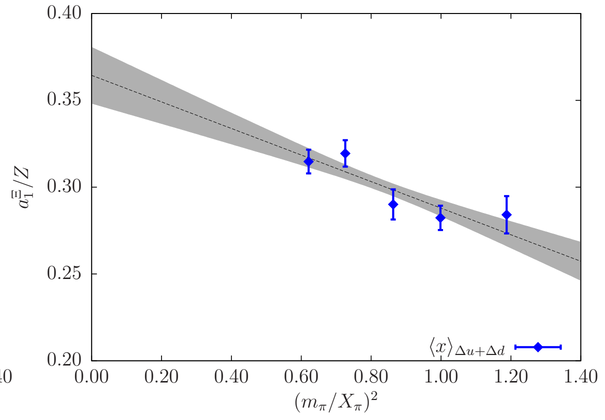
(a) Contribution Δu



(b) Contribution Δd



(c) Contribution $\Delta u - \Delta d$



(d) Contribution $\Delta u + \Delta d$

Figure 12.3 – Second ($n = 2$) moment of polarised PDF $\langle x \rangle^{\Xi}_{\Delta q}$.

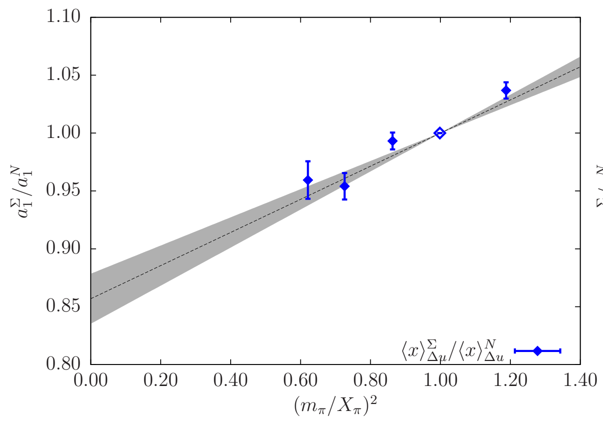
Whereas for the Ξ it was found:

$$\begin{aligned}
\langle x \rangle^{\Xi}_{\Delta u}/Z &= \quad 0.422(15) \\
\langle x \rangle^{\Xi}_{\Delta d}/Z &= \quad \text{-}0.0728(71) \\
\langle x \rangle^{\Xi}_{\Delta u - \Delta d}/Z &= \quad 0.497(17) \\
\langle x \rangle^{\Xi}_{\Delta u + \Delta d}/Z &= \quad 0.351(17)
\end{aligned}$$

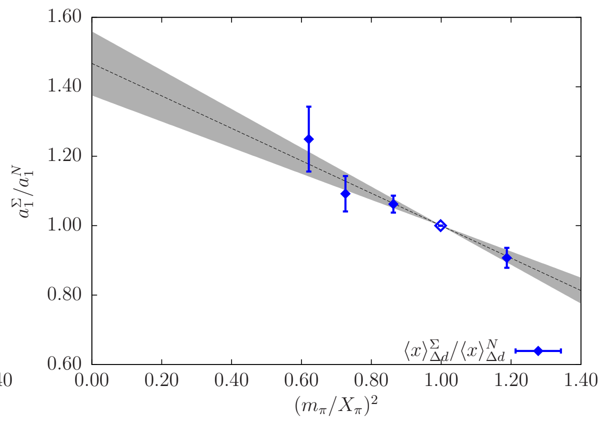
Thus a similar result was found to the unpolarised results in Chap. 9. That is, $\langle x \rangle^{\Sigma}_{\Delta u}$ decreases while $\langle x \rangle^{\Sigma}_{\Delta s}$ increases (in magnitude) as one moves towards the physical point, and similarly for $\langle x \rangle^{\Xi}_{\Delta u}$ and $\langle x \rangle^{\Xi}_{\Delta s}$.

12.3 Ratios

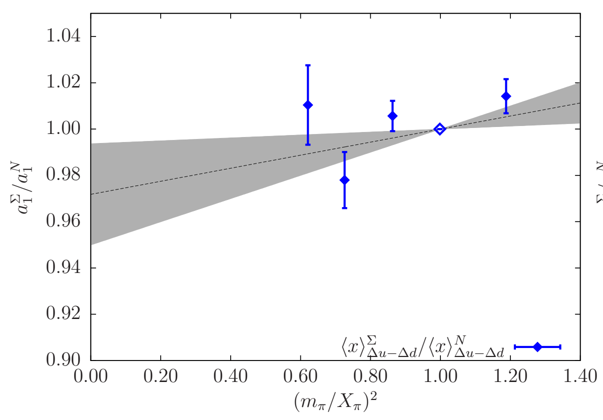
Figs. 12.4a - 12.4d (12.5a - 12.5d) show the ratios $\langle x \rangle^{B}_{\Delta q}/\langle x \rangle^{N}_{\Delta q}$ with $B = \Sigma$ (Ξ).



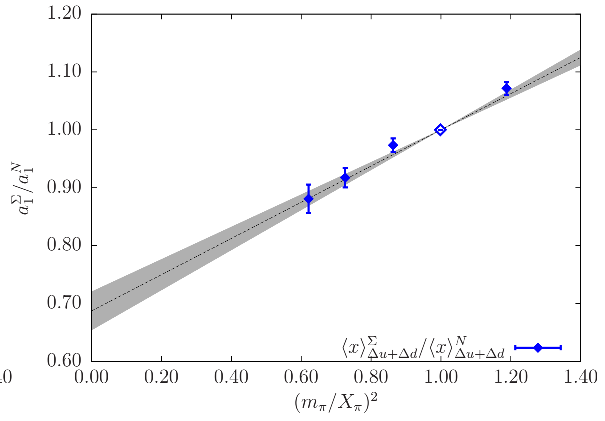
(a) Contribution Δu



(b) Contribution Δd



(c) Contribution $\Delta u - \Delta d$

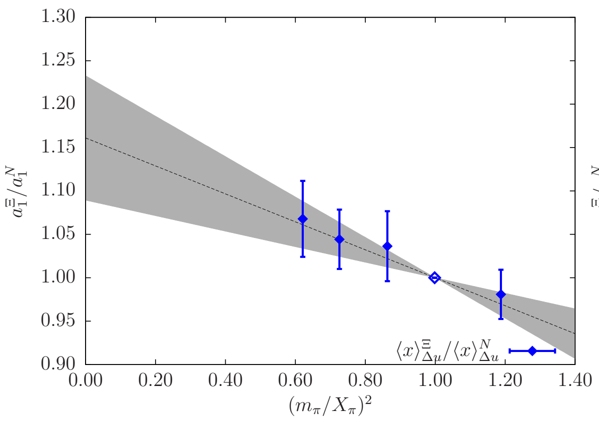


(d) Contribution $\Delta u + \Delta d$

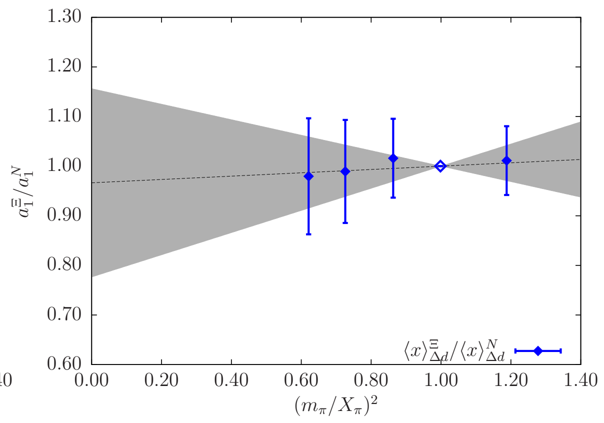
Figure 12.4 – Ratio $\langle x\rangle^\Sigma_{\Delta q}/\langle x\rangle^N_{\Delta q}$

The ratios highlight the difference between the individual quark sectors that was discussed above, i.e. the momentum fraction of longitudinally polarised up-quarks in a longitudinally polarised Σ is smaller than that in the proton. And similarly, the momentum fraction of the longitudinally polarised strange-quark in a longitudinally polarised Σ is larger than that of the down-quark in the proton.

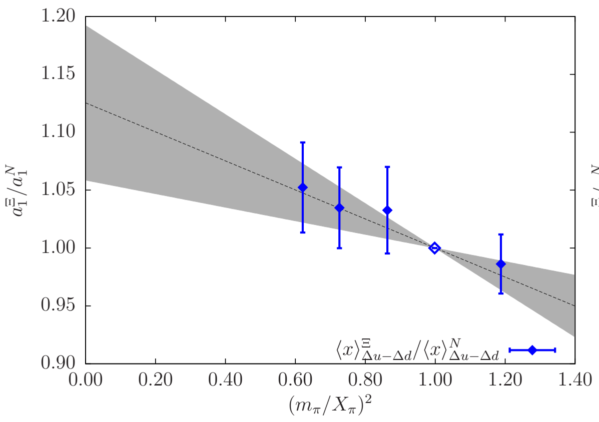
Interestingly, however, is that Fig. 12.4d shows us that the total polarised quark contribution in the Σ is smaller than the corresponding contributions in the nucleon. Fig. 12.5d shows a similar trend for the Ξ but opposite in sign, i.e. larger than the corresponding contributions in the nucleon, but the effect is smaller.



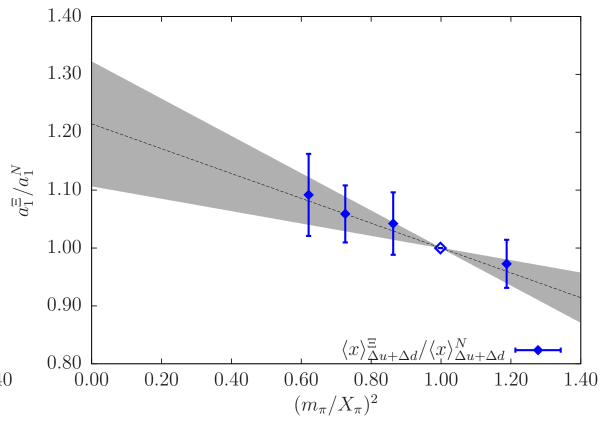
(a) Contribution Δu



(b) Contribution Δd



(c) Contribution $\Delta u - \Delta d$



(d) Contribution $\Delta u + \Delta d$

Figure 12.5 – Ratio $\langle x\rangle^{\Xi}_{\Delta q}/\langle x\rangle^{N}_{\Delta q}$

Chapter 13

Conclusion and Outlook

Baryon structure functions were investigated by means of Monte Carlo simulations of Lattice QCD with $N_f = 2+1$ dynamical quark flavours. The first two moments of unpolarised, longitudinally, and transversely polarised PDFs were calculated for the nucleon and hyperons which include an additional degree of freedom, the strange quark, which makes them amenable to studies with $N_f = 2+1$ flavours.

The QCDSF Collaboration carries out investigations of baryon structure using configurations generated with $N_f = 2+1$ dynamical flavours of $\mathcal{O}(a)$-improved Wilson fermions. The fermion action elaborated is the $N_f = 2+1$ flavour Stout Link Non-perturbative Clover (SLiNC) fermion action with non-perturbative $O(a)$ improvement whereas the Symanzik tree-level action serves as the gluonic part. With the strange quark mass as an additional dynamical degree of freedom in the simulations needs are avoided for a partially quenched approximation when investigating the properties of particles containing a strange quark.

In this work the moments of PDFs were computed from forward baryon matrix elements of the flavour-nonsinglet twist-2 operators with up to one derivative at zero momentum transfer. The required bare matrix elements are extracted from ratios of bare three-point over two-point functions. In general there are quark-line disconnected contributions, which are computationally expensive to evaluate. Since our simulations exhibit exact isospin invariance, for certain flavour combinations the disconnected contributions cancel, e.g. for the baryon axial charge.

Additionally the operators must be improved and renormalised. At the moment operator improvement is not included, since neither a perturbative nor a non-perturbative determination of the operator improvement coefficients are available. However, the effect of operator improvement is expected to be small. An obvious feature that is currently lacking is a determination of the renormalisation constants for the local operators considered here. These calculations are now underway and will allow to make more quantitative predictions in the near future. For comparison with phenomenology the author is currently limited to look at ratios where the renormalisation constants cancel.

The Lattice QCD suite Chroma supports calculating baryonic and mesonic three-point and two-point correlation functions. In order to calculate higher moments of PDFs in this work the corresponding derivative operators were implemented.

In this work the three-point and two-point correlation functions were calculated utilising the QCDSF $N_f = 2 + 1$ flavour SLiNC configurations. The numerical calculations have been performed on the Nehalem Cluster (JuRoPa) at NIC (Jülich, Germany), and the SGI ICE 8200 at HLRN (Berlin-Hannover, Germany).

In this work the pion decay constant and tensor decay constant were calculated in order to circumvent the need for renormalisation constants and to be able to give estimates for the baryon axial charge and the tensor charge.

The nucleon axial charge (unrenormalised) was determined and using an estimate for the renormalisation constant from the pion decay constant it was possible to compare the quantity to experiment and results from other studies. In this work the experimental value was underestimated. This is in accordance with what other collaborations are seeing. Reasons for this discrepancy are still not yet completely understood, although it is likely to be a combination of finite size effects and chiral non-analytic behaviour close to the physical point. Our results for the hyperon axial charges agree well with earlier lattice results and show a hint of flavour SU(3)-symmetry breaking effects.

Since the tensor operators flip the quark helicity, the disconnected diagrams do not contribute in the continuum theory for vanishing quark masses. Therefore, only small contributions for the disconnected graphs are expected. The baryon tensor charge (unrenormalised) was determined and again applying an estimate for the renormalisation obtained from theoretical predictions of the tensor decay constant it was possible to give an estimate of the quantity. Also, the nucleon tensor charge showed little quark mass dependence. This observation was also made by earlier calculations. For the tensor charge of the hyperons nothing is known neither from experiment nor from theory; this work gave the first predictions on these quantities. Whereas the contributions coming from the light quarks in the hyperons do not differ much from the according contributions in the nucleon, the strange quark contributions differ. The non-singlet combination $\delta u - \delta d$ is slightly increasing when considering the N, Σ, and Ξ.

In this work the quark momentum fractions for the unpolarised distribution functions were calculated. Ratios of the quark momentum fractions for different baryons were studied. The quark momentum fractions of the octet hyperons show strong flavour SU(3)-symmetry breaking effects, with the heavier strange quark contributing a larger fraction to the total baryon momentum than the light quarks. By examining the flavour SU(3)-breaking effects in these momentum fractions, it was possible to extract the first QCD determination of the size and sign of isospin-symmetry violations in the parton distribution functions in the nucleon, δu and δd. These results are roughly equal in magnitude and have opposite sign. They are are in excellent agreement with earlier phenomenological calculations.

To date there exists only one other calculation of the $n = 2$ moment of the transversally polarised quark distribution function. Although at the moment it is not possible to directly compare this work's results due to missing renormalisation factors, it was possible, however, to observe the behaviour of the results as a function of the pion mass to be similar to their results – little quark mass dependence was seen. Comparing the ratio of individual contributions $\langle x \rangle^N_{\delta u} / \langle x \rangle^N_{\delta d}$ to their result is possible since here the renormalisation constant cancels. In this work this ratio was found to be in good agreement with their results.

For the $n = 2$ moment of the longitudinally polarised quark distribution function in this work a similar result was found as for the unpolarised case. The light quark distribution in the Σ decreases while the strange quark distribution increases (in magnitude) as one moves towards the physical point, and similarly for the light and strange quark distribution in the Ξ. Interestingly this work unveiled that the total polarised quark contribution in the Σ is smaller than the corresponding contributions in the nucleon. A similar trend for the

Ξ is observed but opposite in sign; it is larger than the corresponding contributions in the nucleon.

At this stage of discussion of moments of PDFs a single lattice volume was used and a check for finite size effects (FSE) is the next step in these simulations. In earlier work moments of PDFs were found to be (partly) very sensitive to finite size effects, e.g. the baryon axial charge. Future investigations with simulations incorporating larger lattice sizes which will allow for dedicated FSE studies.

In the near future $N_f = 2 + 1$ flavour simulations closer to the physical point we be undertaken. This gives the opportunity to carry out extrapolations to the physical point in a controlled way with ChPT.

Part II

Implementation of Lattice QCD Applications on Heterogeneous Multicore Acceleration Processors

Chapter 14

Introduction

14.1 Lattice QCD and HPC

Quantum Chromodynamics (QCD), the theory of the strong force acting between quarks and gluons, is a non-Abelian gauge theory with SU(3) as the gauge group. The gluons, the gauge bosons of the theory, not only interact with the quarks but also with themselves. At high energies, analytic calculations can be performed with perturbative tools reaching high accuracy. However, for low-energy systems, such as baryons, the quark-gluon interaction and the gluon self-interaction are very strong. In this regime the coupling constant grows larger rendering perturbative tools unusable.

To study QCD at low energies, i.e. the study of hadron masses, structures of nucleons, decays of particles, one has to rely on non-perturbative methods, i.e. numerical techniques. The Feynman path integral is formulated on a discretised space and time with a four-dimensional (4D) Cartesian lattice and evaluated numerically by Monte Carlo integration. This lattice, gives the study of QCD with this technique its name, Lattice QCD. When the lattice spacing a is made sufficiently small, QCD simulations could in principle yield precise answers for a wide variety of physical phenomena.

An essential ingredient to the evaluation of the Feynman path integral is the quark propagator, which is mainly the inverse of the Dirac operator, the fermion matrix. The Dirac operator is a huge sparse matrix of dimensionality of the order of number of lattice points in the lattice. Typically the inversion of this large sparse matrix dominates the overall computation time of Lattice QCD simulations.

In order to generate an ensemble of gauge configurations the fermion matrix has to be inverted several times. The cost of generating a decent sized ensemble of gauge configurations for $N_f = 2$ flavour dynamical Wilson quarks is roughly [140]

$$k \left[\frac{20\text{MeV}}{\overline{m}}\right]^{c_m} \left[\frac{L}{3\text{fm}}\right]^{c_L} \left[\frac{0.1\text{fm}}{a}\right]^{c_a} \text{ tera-flops} \times \text{ years} \tag{14.1}$$

with the renormalised quark mass $\overline{m}$ at a scale of 2 GeV in the $\overline{\text{MS}}$-scheme. Typical values for the exponents in this formula are $c_m = 1-2$, $c_L = 4-5$, and $c_a = 4-6$. These values have a large uncertainty. The prefactor k is typically $\mathcal{O}(1)$ for Wilson fermions.

In the past, as a result of insufficiently available computational power Lattice QCD simulations were only feasible at unphysically large pion masses, coarse and small lattices. Today, with the availability of powerful parallel computer system simulations near the physical point with ever finer and larger lattices are possible.

Reaching the physical point where the value of the pion mass reaches its physical value is very difficult even with today's machines – petascale computers are required. Lattice QCD has received much attention as a "grand challenge" problem in scientific computing.

The Lattice QCD community has a long tradition not only in building cluster systems out of commodity hardware components specific for their needs but also in designing and building their own application-optimised HPC machines. With only nearest-neighbour communication patterns in a typical formulation of the Dirac operator it is an attractive target for application specific computing. Custom microprocessors and application-specific network processors were developed and employed in highly-scalable Lattice QCD machines. Among those, the apeNEXT and QCDOC machines [141, 142] and the latest application-optimised HPC machine, QPACE [143], a new type of massively parallel computer.

14.2 Commodity Clusters

A cluster is a parallel computer system comprising an integrated collection of independent nodes each of which is a system in its own right capable of independent operation and derived from products developed and marketed for other stand-alone purposes. Moreover, a commodity cluster is a cluster in which the network as well as the compute nodes are commercial products available in the market.

Today, clusters typically comprise two levels of parallelism: Parallelism within a node (intra-node parallelism) and parallelism across nodes (inter-node parallelism). Intra-node parallelism typically uses a shared memory model within a node: One (or more) multi-core processors usually configured as Symmetric Multi-Processing (SMP) processors. Inter-node parallelism involves all off-node communication to neighbouring or distant nodes utilising the node card's network components.

Commodity clusters have provided an exceptional opportunity in performance to cost, flexibility in configuration and expansion, rapid tracking of technology advances, direct use of a wide range of available and often open source software, portability between clusters, and a wide array of choices of component types and characteristics. In addition, commodity clusters have provided scaling between the very small (a few nodes) to the very large (approaching $\mathcal{O}(10^5)$ processors). Over the last decade not only the evolution of microprocessors developed and marketed for stand-alone purposes showed a reliably increased clock speed but also the network components available in the market of procurement showed substantially improved performance. This directly reflected in the aggregate performance of commodity clusters. Upgrading a commodity clusters and utilising the software at hand without the need for modifying it resulted in a doubling of the performance every two years. This was the time of when message passing started to be the successful approach for inter-node communications. Libraries that implemented the Message Passing Interface (MPI) were the most prominent ones [144]. However, in 2004, when clock speeds began to stall the problems of commodity computing became more salient, especially the memory wall or divergence problem. Commodity clusters with node architectures featuring single-core microprocessors were not delivering any more the previously seen performance increase.

Now in its 40th year, Moore's Law, that predicts a doubling of the number of transistors in a single microprocessor every 18 month, is still going strong. But unfortunately, ever-increasing transistor density no longer delivers comparable improvements in application performance. Adding transistors also adds wire delays and speed-to-memory issues. More aggressive single-core designs also inevitably lead to greater complexity and heat. Finally, scalar processors themselves have a fundamental limitation: a design based on serial execution, which makes it extremely difficult to extract more Instruction-Level Parallelism (ILP) from application codes.

The memory wall or von Neumann bottleneck represents the divergence of the number of processor clock cycles needed for carrying out an arithmetic operation to the clock cycles needed to transfer the operands off-chip, i.e. to DRAM, to an SMP configured neighbouring processor or off-node. In some recent system designs, i.e. the Cell Broadband Engine, the bandwidth from an off-chip memory device to the register file located close to the floating point units is limited to 1 byte per processor clock cycle. On the other hand the peak floating point performance of the core is 8 floating point operations (single precision) per clock cycle. There is clear evidence that in near future processor architectures this ratio will further diverge, e.g. for Graphic Processors Units (GPUs).

CPU vendors started to provide more of their key product on the same die. Before 2004 data from the Top 500 list [145], a ranking of supercomputers by their compute power, shows that flop performance improved at a factor of 1.8 per year, with 1.4 from a faster clock and wider floating point units, i.e. AMD's in 1999 introduced 3DNow! and Intel's in 2000 introduced Streaming SIMD Extensions (SSE), and 1.3 from simply having a bigger machine. Plotting machine size against time shows a clear inflection point around 2004 after which machines have mainly improved performance and kept on trend by using more cores for processing. Computer companies are increasing on-chip parallelism to improve performance – the traditional doubling of clock speeds every 2 years is being replaced by a doubling of cores or other parallelism mechanisms. A broader transition to multicore started with Intel's release of its first dual-core Xeon in 2005. For many applications (especially those requiring heavy floating-point operations), multi-core processing provided performance gains coping with the traditional doubling in the performance every 18 month. This opened up the field of multi-threaded programming models and launched the development of concurrent programming libraries like OpenMP, POSIX Threads, and GNU Portable Threads [146].

Modern system designs have already dismissed the conventional balance of bytes per flops of 1:1. Instead, new architectures comprise between one order of magnitude more Floating-Point Units (FPU) than ever existed before. An entirely different set of balance requirements drives today's and future's architectures based on bandwidth, overhead time, and latency tolerance. If no change in structure and evolution is undertaken in the future flops are free and memory accesses dominate the overall cost.

While in the short term, Moore's law is expected to apply unabated and commodity components will dominate system designs, eventually, Moore's law will flat line due to atomic and quantum effects and conventional components will provide low efficiency such that little gain will be achieved for larger systems.

14.3 Supercomputing

Supercomputers have played an important role for decades in advancing the state-of-the-art in high performance computing and communications. Innovations in communications hardware, network protocols, and network operating systems often arise from supercomputing research and development projects. Furthermore, supercomputers are systems capable of solving certain types of important scientific and engineering problems, known as "grand challenge" problems.

Companies developing supercomputers, like IBM or Cray, utilise node architectures either featuring custom or mainstream commercial multi-core microprocessors typically accompanied with low latency, highly-scalable interconnects which might be supported by network co-processors.

IBM's Blue Gene series, currently in its 3rd technology iteration, is being developed with the approach to building large scale supercomputers taking a large number of relatively simple processing cores and to connect these via a proprietary highly-scalable, low latency network. This has the advantage of creating a high aggregate memory bandwidth (as each processor is directly connected to its own memory) whilst maintaining low power consumption because of the relatively low clock frequency. The next-generation prototype of IBM's Blue Gene series, Blue Gene/Q, has topped the latest iteration (Nov. 2010) of the Green 500 list [147], a ranking of supercomputers by their compute power efficiency. In 2011/12 Lawrence Livermore National Laboratory will deploy a 20 peta-flops Blue Gene/Q Sequoia system.

Cray's XT supercomputers is a series of massively parallel supercomputer. The Jaguar system at DOE/SC/Oak Ridge National Laboratory is a Cray XT5 installation employing a massive array of six-core AMD Opterons with a peak performance of 2.3 peta-flops. They are interconnected with Cray's proprietary connection network with high bandwidth and low latency.

14.4 Heterogeneous Multi-Core Processors

Heterogeneous computing is the strategy of deploying multiple types of processing elements within a single workflow, and allowing each to perform the tasks to which it is best suited. This model can employ the specialised processors to accelerate operations by orders of magnitude faster than what scalar processors can achieve, while expanding the applicability of conventional microprocessor architectures. Because many HPC applications include both code that could benefit from acceleration and code that is better suited for conventional processing, no one type of processor is best for all computations. Heterogeneous processing allows for the right processor type for each operation within a given application.

Traditionally, there have been two primary barriers to widespread adoption of heterogeneous architectures: the programming complexity required to distribute workloads across multiple processors and the additional effort required if those processors are of different types.

The most prominent examples of heterogeneous efforts in HPC include the IBM PowerXCell 8i processor and the rapidly growing Graphic Processing Units (GPU) and GPGPU computing community supported by NVIDIA and AMD. Intel recently announced its massively multi-core chip, Many Integrated Core (MIC), architecture and disclosed it to act as a multi-core accelerator for the system's CPU. Thus the whole system (CPU and MIC) can be considered a heterogeneous platform.

Application Programming Interfaces (API) based on the Programming language C, such as CUDA released by NVIDIA, have opened up GPU/GPGPU computing to a much wider audience [148]. AMD offers a similar Software Development Kit (SDK) + API for their ATI-based GPUs, which is called FireStream SDK. Typically these SDKs and APIs are bound to the company's GPGPUs. There are efforts to standardise parallel programming of heterogeneous systems, e.g. OpenCL [149].

The PowerXCell 8i processor, an enhanced version of the Cell processor used in the Playstation 3, comprises 1 general purpose core and 8 accelerator cores. The general purpose core, the PowerPC Processing Element (PPE), is a standard PowerPC core that can, e.g. be used for running the operating system Linux. The PPE is usually used as the application controller which distributes the payload of computation to the 8 accelerator cores, the Synergistic Processing Elements (SPE).

Each SPE comprises it own 16 byte wide Single Instruction Multiple Data (SIMD) FPU, supporting a Fused Multiply-Add (FMA) operation. The accelerator core features a so-called Local Store (LS), a 256 kilo-bytes on-chip memory from which up to one 16 byte-wide word can be loaded or stored to or from the register file per clock cycle. The register file is 128 entry, 16 byte wide and is thus exceptionally large.

The FPUs on the PowerXCell 8i processor perform up to 8 double-precision FMA operation on a SIMD vector per clock cycle, resulting in a peak performance of 102 double-precision giga-flops at a core clock speed of 3.2 GHz.

The processing elements as well as the memory interface controller are interconnected via the Element Interconnect Bus (EIB) featuring a very high bandwidth. Memory transfers between an SPE's LS and any other processing element or main memory is carried out by means of asynchronous Direct Memory Access (DMA) transfers – SPE computation executes in parallel to memory transfers.

While being impressive in absolute numbers for application developers the high floating-point performance of the Cell processor is difficult to exploit to a satisfactory fraction. This is due to various reasons. The Cell processor comprises two levels of parallelism – FPUs operating on SIMD vectors and the multi-core parallelism. Typically, application development for parallel system is considered laborious. Also, the memory hierarchy of the Cell processor is non-trivial and the LS size is very limited when considering typical scientific code bases.

14.5 HPC Challenges

Based on the current rate of performance improvement an exa-scale (10^{18}) system, i.e. a system with an aggregate peak performance of 1 exa-flops, would be expected to be available around 2018. Today, FPUs consume a very small fraction of the area in modern chip designs and a much smaller fraction of the power consumption. On modern systems, a double-precision FMA operation consumes roughly 100 pJ (pico Joule) [150]. By contrast, reading a double precision operand from DRAM costs around 2000 pJ per operand – where a FMA requires 3 operands to be read. If one extrapolates the improvement in technology with the current rate this ratio gets even worse, i.e. in 2018 the FMA operation would consume 10 pJ and reading an operand would cost 1000 pJ. Taken 10 pJ per FMA operation as a basis a system which is capable to perform 1 exa-FMA operations per second consumes 10 mega-watt. This only includes the energy needed for carrying out the floating-point operations. Accounting also for the energy needed for reading the operands from DRAM one quickly sums up to the power a nuclear plant provides. The primary design constraints for future HPC systems is power consumption.

When seeking for the biggest energy consumers in microprocessors it becomes quickly apparent that it is the moving of data itself. There are mainly two types of data movement: On-chip and off-chip communications. While on-chip communications involve data transfers inside the chip, like traffic between the cache hierarchy and the register file or the register file and floating point pipelines, off-chip communications include transfers that go across chip boundaries and access, e.g. the main memory and also include internode communications like MPI transfers.

One possibility to increase locality of data flow is introducing multi-level cache hierarchies to microprocessors. This replaces parts of the off-chip communications with on-chip transfers to close cache levels and reduces the overall power consumption. Another possibility is to group together computation and memory hierarchy in the microprocessor into functional clusters or hierarchies of these to further exploit locality of data accesses. This trend is already applied in the system design of GPUs, where several cores are grouped together into functional units.

Automatically managed caches, i.e. cache-coherent models, involve a substantial administrative logic and thus are expensive in terms of power consumption. These caches virtualise the data location of on-chip and off-chip memory, and are therefore invisible to current programming models. However, the cost of moving data off-chip is substantial, thus virtualising the data location in this manner wastes energy and might substantially reduce performance. One strategy to reduce off-chip communication is to employ more explicit software management of memory, i.e. explicit managed caches. As a result applications and algorithms will need to change and include more code for managing these newly introduced memories. In particular, they will need to manage locality to achieve high performance.

The Cell processor is an example for a microprocessor featuring explicitly managed caches. The LS of the accelerator cores are subject to explicit memory management. It is due to the application designer to ensure data and code availability in LS when required.

The cache-coherent model and SMP are likely to not form part of the HPC path. Instead explicitly managed caches, *System-on-a-Chip*, *SMP on a Chip*, and *Processor in Memory* will become important elements that bring the memory closer to the logic or vice versa. Programming language designers must consider how to enable expression of data locality without the cache-coherent model.

14.6 Application-Optimised HPC

An application-optimised HPC machine is a computer cluster of highly replicated basic components typically with custom designed network components. The machine features hardware characteristics especially well suited for specific types of applications or domains of problems.

The nature of Lattice QCD implies that the minimal design parameters for an ideal Lattice QCD machine can be more restrictive compared with those of general-purpose parallel machines: Typically, the implementation of the Dirac operator involves nearest-neighbour communication patterns only. This makes computers with a torus network connecting the processing nodes an obvious choice. Typically the only common non-nearest-neighbour communications are broadcast and global reduction operations. Both communication and memory access patterns are deterministic and amenable to both software and hardware prefetching.

These key simplifications to hardware requirements led to the development of a number of specialised machines, application-optimised HPC systems, in the last decades.

The earliest Lattice QCD machines featured a two-dimensional periodic mesh for the internode connection. They were designed and built at Columbia University in the 1980s. The memory on each node was mapped into the address space of the four neighbouring nodes, a method that has also been employed over the generations of the (initially Italian) Array Processor Experiment (APE) machines [151].

Machines of hundreds of nodes were built and achieved good performance through processing nodes made up of a microprocessor and an external FPU. The largest installation was deployed in 1989 which featured a maximum peak performance of 16 giga-flops in double precision.

A later development was the QCDSP (QCD on digital Signal Processors) machine. The semantics for accessing the data of a neighbouring node has been decoupled from the CPU [142]. DMA engines and an asynchronous message-passing library were used to give a simple but efficient overlapping of communication and computation with complete hiding of the internode communication latency. Also the QCDSP system received an increased dimensionality of the processor grid from 2 to 4. Each processor node featured increased compute power. One compute node of the QCDSP system comprised a Digital Signal Processor (DSP). The additional component of the QCDSP processing node is a custom Application-Specific Integrated Circuit (ASIC), designed by the Columbia Lattice QCD group, supplying the DSP with single-cycle access to DRAM. The internode network featured a low latency for memory-to-memory transfers for neighbouring nodes. The largest installation was set up in 1998 with a peak performance of 600 giga-flops in double precision.

The QCDOC system represented an obvious path for improvement by employing components reflecting much greater transistor density and higher clock speeds [141] – the IBM System-on-a-Chip (SoC) technology. The QCDOC ASIC combines all of the features of the QCDSP node, in addition to many more, on a single chip, and it provides 20 times the performance at about twice the cost. The processor grid was extended to be 6D. The largest and latest installation was deployed in 2005 and featured 10 tera-flops peak performance in double precision.

14.7 QPACE

QPACE (QCD PArallel computer based on CEll processors) is a novel parallel computer which has been developed to be primarily used for Lattice QCD simulations [143]. The compute power is provided by the IBM PowerXCell 8i processor. The PowerXCell 8i processor supersedes the Cell processor with support for high-performance double precision operations, IEEE-compliant rounding, and a DDR2 memory interface. The QPACE nodes are interconnected by a custom, application optimised 3D torus network implemented on a Field Programmable Gate Array (FPGA). To achieve the very high packaging density of 26 tera-flops per rack a new water cooling concept has been developed and successfully realised. There are two installations of 4 QPACE racks each deployed in 2010 and which provide an aggregate peak performance of 208 tera-flops in double precision.

The network processor of a QPACE node features low-latency and high-bandwidth connects to its 6 nearest neighbours within a 3D torus. A main feature of the torus network is the ability to send and receive messages directly from the SPE of one node to an SPE of a neighbouring node without support from the PPE and without copying the data to main memory.

It remains a difficult task to exploit the high peak performance of this machine. Application developers are not only confronted with the programming challenges found for the Cell processor, but for QPACE another level of parallelism is introduced – the inter-node connect.

14.8 HPC Software Challenges

Scientific productivity on recently emerging peta-scale and future exa-scale systems is widely attributed to the system balance in terms of processor, memory, network capabilities and the software stack. Next generations of these systems are likely to be composed with node architecture with 16 or more cores on single or multiple sockets, deeper memory hierarchies and a complex interconnection network infrastructure. Hence, the development of scalable applications on these systems most likely requires application and library developers to account for hierarchical programming models of memory, computation and network activity. Even the current generations of the Cray XT and IBM Blue Gene series offer several multi-core processors per node card, multiple levels of unified and shared caches and a regular communication topology along with support for distributed computing (MPI) and hybrid (MPI and SMP) programming models. As a result, it has become extremely challenging to sustain performance efficiencies on these systems.

Typically in application development in the scientific domain where application performance, portability, and production of maintainable code bases are primary objectives, a general purpose language with object-oriented features like C++ is one of the first choices. C++ provides many benefits for HPC application development including: register near data structures, code inlining facilities, and meta-programming methods.

However, tradition C++ libraries, typically collections of compiled subroutines, quickly reach their limitations in applicability in the HPC domain as long as high-performance code is required. Compiled subroutines represent static program code and do not feature dynamic code generation that meta-programming methods can offer.

Active libraries, mostly enabled by meta-programming methods, on the other hand have proven to provide domain-specific abstractions and the know-how needed to optimise them [152]. They combine the benefits of built-in language abstractions, i.e. convenient syntax and efficient code, with those of library-level abstractions.

The Portable Expression Template Engine (PETE) pioneered the use of expression template techniques for parallel physics computations [153, 154]. It achieves an exceptional level of abstraction without sacrificing performance.

QCD Data Parallel (QDP++) builds on top of PETE and extends its concept by providing domain-specific abstractions suited for quantum field theory. The Chroma package, a suite for Lattice QCD calculations, in turn builds on top QDP++ and achieves high portability and efficiency on desktop workstations, commodity clusters, and several supercomputers and application-optimised HPC systems [155].

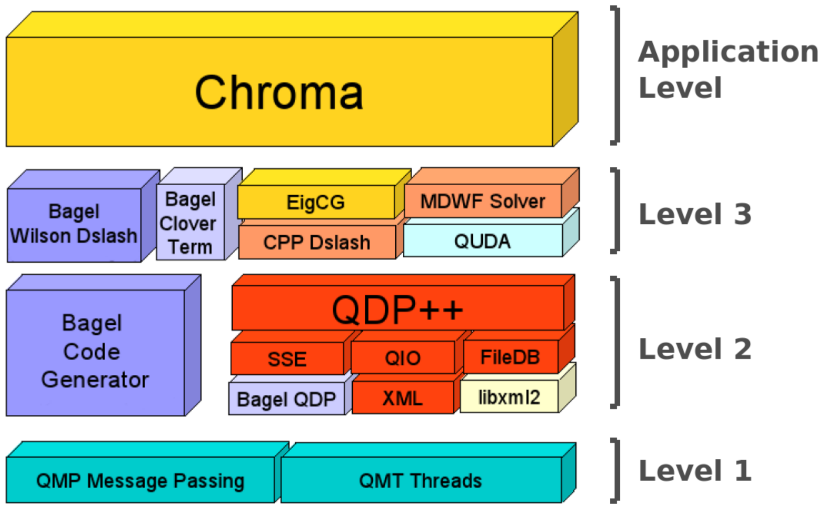


Figure 14.1 – Software components of the SciDAC software hierarchy involved in Chroma.

However, the level of abstractions reached by those active libraries to date are most likely not sufficient for future machines. Especially heterogeneous multi-core processors with two- (or more) level memory hierarchies are not supported by such high-level active libraries like QDP++. Library developers are required to also account for the different memory models found in these processors.

14.9 SciDAC Software Hierarchy

The U.S. Lattice community started a project through the U.S. SciDAC (Scientific Discovery through Advanced Computing) initiative to standardise a set of software components in order to allow the effective exploitation of computing resources for Lattice QCD [156].

Fig. 14.1 depicts the software components of the SciDAC software hierarchy involved in Chroma. The main software components this works targets at are the Chroma application suite and all involved software components: QDP++, QMP, and QMT which will be briefly introduced.

Level 1 – QCD Message Passing (QMP), QCD Multithreading (QMT) QMP provides a message passing API for Lattice QCD calculations similar to MPI. QMP was designed to take advantage of the specialised communication hardware of the supported architectures. QMT was introduced to cope with the upcoming SMP processors. These are homogeneous multi-core processors, i.e. all cores are of the same type, and the load and store instructions target, possibly through a hierarchy of cache levels, at the main memory. On clusters of these processors a performance gain can be achieved if using QMP for inter-node communication, i.e. the off-node communication, and QMT for intra-node communication, i.e. communication within one processor [157].

Level 2 – QCD Data Parallel (QDP++) QDP++ provides lattice-wide data types and operations for applications in quantum field theory. It is the key software package in the Chroma suite and for now its description is postponed to a later, separate section.

Level 3 – Special optimised software This level provides interfaces to optimised and machine-dependent functions. Two different types of level 3 optimisation functions are differentiated: Level 3a and 3b. Whereas level 3a implementations access the data objects through QDP++ data types, level 3b implementations rely on a specific data layout and access the raw data directly. Level 3a implementations lead in general to more portable codes whereas level 3b implementations offer greater freedom on carrying out more aggressive optimisations.

At level 2 and 3 optimised kernels are found which were generated by the BAGEL assembler generator [158].

Application Level – Chroma At this level the application Chroma and the Chroma library are implemented – no machine dependent code is found at this level. Application developers concentrate on implementing the algorithmic structure of the program and make use of the lattice wide data types and operators offered by QDP++ and Level 3 implementations.

Chroma is a collection of Lattice QCD applications that was originally developed to serve the needs of the LHPC and UKQCD Collaborations. The Chroma package itself and all other SciDAC software components are open source software and can be modified freely. Chroma has now a large user base around the world which partly contributed new functionality and or extended existing ones. It includes spectroscopy, decay constant, nucleon form factor and structure function moment calculations. The code contains chiral fermion actions, Wilson, Domain Wall and Overlap fermion operators and numerous inverters.

14.10 QDP++

QDP++ is a major component of the USQCD/SciDAC software stack. It provides a data-parallel programming environment suitable for essentially any kind of Lattice QCD application.

The interface provides a level of abstraction such that high-level user code can be run unchanged on a single processor node or a collection of nodes with parallel communications. Architectural dependencies are hidden below the interface.

As the double-plus in it's name suggests it is implemented in C++ – to distinguish it from an independent C-implementation, QDP-C.

Traditional overloading of C++ arithmetic operators typically involves creating and copying of temporaries of the instantiated type. However, in order to achieve a high performance movement or replication of data must be minimised, especially when lattice wide data objects are involved.

The expression templates technique can be used to evaluate vector and matrix expressions in a single pass without temporaries, i.e. with a higher performance [159]. PETE is an extensible implementation of the expression template technique [153]. QDP++ makes use of PETE and thus takes advantage of the intuitive form of constructing expressions by using overloaded arithmetic operators and on the other hand avoids the creation of lattice temporaries.

14.11 QCD Applications on Accelerators

According to Amdahl's Law the overall speedup of a computer program including a fraction P (in term of execution time) that was sped up by a factor S is

$$S_{\text{total}} = \frac{1}{(1-P)+\frac{P}{S}}. \tag{14.2}$$

Consider the following example: An unoptimised computer program's execution time is T. One might be able to optimised the program parts which make up, say $P = 50\%$, of the execution time. The optimised program parts gain a speedup factor of $S = 2$, i.e. they execute with twice of the performance. Thus, the overall speedup factor is $S_{\text{total}} = 4/3$ and the new program execution time is $T_{\text{new}} = T/S_{\text{total}}$.

This law also predicts the maximum speedup factor given a fixed portion of the program that is subject to optimisation. For example, consider the optimisation achievable by parallelisation. If one is able to parallelise the program perfectly, i.e. it executes after parallelisation with negligible execution time, so that the speed-up factor grows over all finite bounds $S \to \infty$, then in the above example the overall speedup factor is limited to $S_{\text{total}} = 2$.

On the Cell processor, the execution time of program parts executed on the accelerators compared to when executed on the general purpose core can be substantially smaller. It is difficult to make general estimates for the speedup factor, but S_{Cell} might be in the range $\mathcal{O}(10)$ to even $\mathcal{O}(1000)$. Thus the fraction in the denominator in (14.2) can become negligible and the overall speedup factor depends only on P.

Lattice QCD programs typically spend the majority of the execution time in the inverter. Let us assume a realistic value of $P = 80\%$. Normally the inverter gets optimised for the target hardware. Thus the maximal overall speedup factor would be around $S_{\text{total}} = 5$.

On the Cell processor the program parts not implemented for execution on the accelerators execute with such a poor performance, that even with the speedup factor applied the performance is still not acceptable. As a result, to achieve a high overall performance on the Cell processor a large fraction of the involved program parts needs to be ported to the accelerators, the SPEs.

14.12 New design concepts for QDP++

In this work new design concepts were developed on how to implement an active C++ library like QDP++ on accelerator type of processors like the IBM PowerXCell 8i processor. Not only is a proof-of-concept provided, but it was possible to run a QDP++ based physics application (Chroma) with reasonable performance on the IBM PowerXCell 8i Processor.

Developing these design concepts significantly extents beyond usual code porting activities. This work pursues a new strategy leveraging PETE.

The expression template technique, on which PETE is based on, is implemented by means of meta-programming methods and as such by definition resolved during translation. Only after resolving the expression templates the functions are available in an assembled form that one can address for building for the SPEs.

At run-time of the main application the involved PETE expressions are written into a database. This work includes a code-generator which processes the database and generates program code fractions for the accelerator core. The newly generated code fractions rely on the functionality of QDP++. In order to account for the hardware characteristics of the accelerator this work provides an optimised version of QDP++ for this processor type. All involved PETE expressions are available as compiled functions, optimised for the accelerator. This work proposes to compile all such functions and to bundle them into a library and to link the main application against it. In this way all program parts of the main application involving PETE expressions execute on the accelerators.

A brief outline follows of the modifications and optimisations carried out in order to achieve an implementation of QDP++ suitable for the accelerator cores – in case of the Cell processor, the SPEs:

Support for the SIMD organisation was added. A change in the data organisation ensured that complex numbers always fit into one processor register. To reduce memory bandwidth requirements additional attributes were introduced to QDP++ data types which describe the data object's access pattern. The standard QDP++ memory allocator was replaced by a pool memory based implementation. This ensures a high-performance management of the SPE's LS. A major modification was adding support for accessing main memory via DMA transfers. In this way the SPEs access main memory with asynchronous DMA transfers which execute in parallel to SPE computation. With multi-buffering algorithms memory latencies and transfer overheads could be (partially) hidden by computation. Parallelisation on several SPEs was achieved by assigning each SPE to a disjoint subset of the problem, i.e. each SPE operates on a different part of the data vector. In order to find the optimal problem size and alignment settings for each SPE compile-time calculations were carried out. Also, to cope with the limited size of the SPE's LS for code and data this work employes SPE code overlay techniques.

IBM has discontinued the development of the Cell processor. Other accelerator-based architectures are getting more important targets in the HPC market. The design concepts developed in this work are not bound to specific microprocessor. It is retargetable and similar challenges are likely to be encountered when targeting to other accelerator based architectures, like e.g. GPUs.

Chapter 15

The IBM PowerXCell 8i Processor

15.1 Overview

The IBM PowerXCell 8i Processor is the enhanced version of the Cell Broadband Engine (Cell/B.E.) Processor. Both are collectively called Cell Broadband Engine Architecture (CBEA) Processors. The CBEA was developed jointly by Sony, Toshiba, and IBM and extends the 64 bit PowerPC Architecture. The Cell processor is the first implementation of a multiprocessor family conforming to the CBEA. The IBM PowerXCell 8i Processor which also conforms to the CBEA provides a Double Data Rate 2 (DDR2) memory interface and improved double-precision floating-point performance and additional double-precision instructions. In the following when speaking of the Cell processor the enhanced version is meant since this is our target hardware.

Although the Cell processor is initially intended for applications in multimedia consumer-electronics devices, the CBEA has been designed to enable fundamental advances in processor performance. Especially applications in scientific fields take advantage of the IBM PowerXCell 8i Processor with its outstanding floating-point performance.

Both CBEA processors have 9-cores: One PowerPC Processor Element (PPE) and eight Synergistic Processing Elements (SPEs). The two types of cores differ substantially in architecture and functionality thus they are heterogeneous multi-core processors.

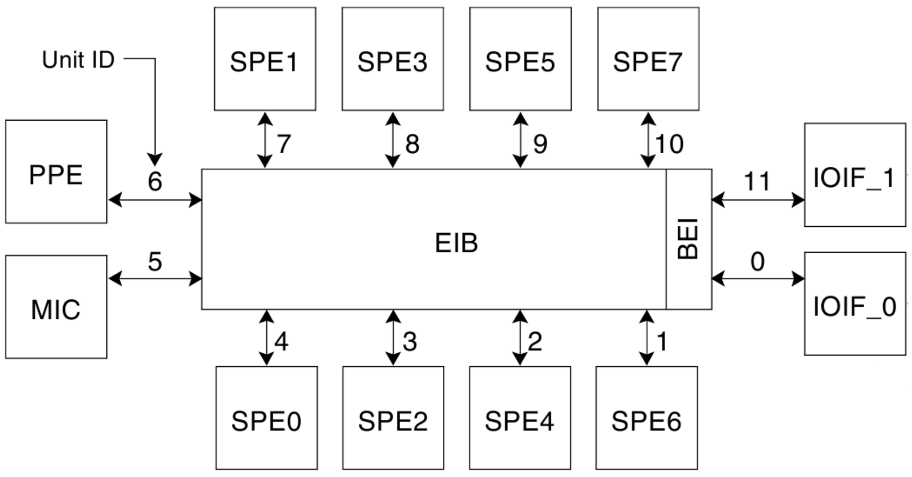


Figure 15.1 – Cell Broadband Engine Architecture. The Element Interconnect Bus (EIB) connects the Synergistic Processor Elements (SPEs) and the PowerPC Processor Elements (PPE) and the Memory Interface Controller (MIC) and Input/Output Interfaces (IOIF) via the Cell Broadband Engine Interface (BEI). Picture source: [160]

The first type of core, the PPE, is the general purpose core and complies with the 64 bit PowerPC Architecture and is able to run 32 bit and 64 bit operating systems and applications.

The second type of core, the SPE, is the accelerator core and is optimised for running compute-intensive Single-Instruction, Multiple-Data (SIMD) applications. Each SPE has its own execution units, floating-point pipelines, and local memory for instructions and data. One CBEA Processor comprises 8 SPEs.

The original machine design foresees the general purpose core, the PPE, to run the top-level, or control thread of an application whereas the accelerator cores, the SPEs, are considered to provide the floating-point performance for the application.

Fig. 15.1 shows a high-level block diagram of the CBEA Processor hardware. The PowerPC Processor Element and the eight Synergistic Processing Element are interconnected and connected to the on-chip memory and I/O controllers by the Element Interconnect Bus (EIB).

15.2 PowerPC Processor Element

The PPE is the processor element in CBEA processors running the operating system and acting as the controller for the SPEs. It has a 64 bit Reduced Instruction Set Computing (RISC) processor that implements the PowerPC architecture. It executes two hardware threads and features SIMD vector multimedia extensions. The GNU C++ Compiler for the PPE supports C/C++ intrinsics for SIMD vector multimedia extensions. Besides running the operating system and managing SPE threads, its intention is control processing and managing system resources. The PPE consists of two main units, the PowerPC Processor Unit (PPU) and the PowerPC Processor Storage Subsystem (PPSS). It features 32 kilo-bytes level-1 instruction and data caches and a 512 kilo-bytes level-2 unified (instruction and data) cache.

15.3 Synergistic Processor Element

The Synergistic Processor Element (SPE) is a RISC processor with 128 bit SIMD organisation and execute a 32 bit instruction set. It is especially well suited for applications with high floating point performance requirements. The aggregate floating point performance of the SPEs in one CBEA Processor is more than 100 giga-flops in double precision.

The SPE consists of two main units, the Synergistic Processor Unit (SPU) and the Memory Flow Controller (MFC) (see below). Fig. 15.2 shows a block diagram of the SPE.

15.3.1 Synergistic Processor Unit

The Synergistic Processor Unit (SPU) comprises the 256 kilo-bytes Local Storage (LS) which holds the program code and data. The register file of the SPU contains 128 registers each 128 bit wide. All SPU load and store instructions move data directly (no cache level involved) between the register file and the LS.

The SPU features four execution units, a Direct Memory Access (DMA) interface, and a channel interface for communicating with its MFC, the PPE, other SPEs and other devices.

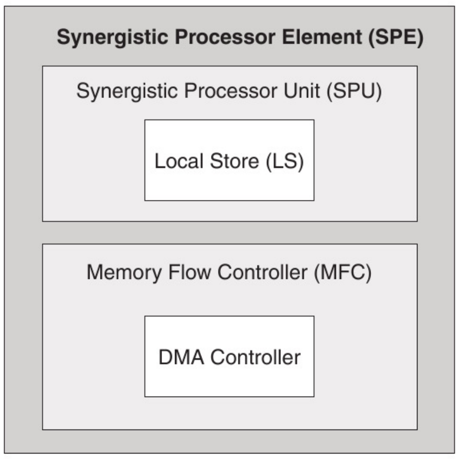

Synergistic Processor Element (SPE)
Synergistic Processor Unit (SPU)
Local Store (LS)
Memory Flow Controller (MFC)
DMA Controller

Figure 15.2 – Block diagram of the Synergistic Processor Element (SPE). The SPE consists of the Synergistic Processor Unit (SPU) which includes the Local Storage (LS) and the Memory Flow Controller (MFC) containing the DMA Controller. Picture source: [160]

Table 15.1 – LS-access arbitration priority (in descending order) and transfer sizes.

access type	bandwidth/cycle	max. occupancy
DMA-access	128 bytes	1/8
DMA-List-access	128 bytes	1/4
SPU load and store	16 bytes	1
SPU instruction prefetch	128 bytes	1

The SPU accesses the Main Storage (MS) by requesting DMA transfers from its MFC. The DMA controller executes the DMA transfers in parallel to SPU program execution, it executes so-called asynchronous DMA transfers.

The SPU supports dual-issue of instructions on its two execution pipelines. The pipelines are referred to as the even and the odd pipeline. The SPU can issue and complete up to two instructions per cycle, one on each of the two execution pipelines. Whether an instruction goes to the odd or even pipeline depends on the instruction type.

Local Storage

The Local Storage (LS) a is 256 kilo-bytes memory located in the SPE. It holds all instructions and data used by the SPU program and is protected with Error-Correcting Code (ECC) code.

A single local memory port is shared by several SPU elements, i.e. the instruction fetch mechanism, the processor's memory instructions, and the DMA transfer mechanism. Exactly one read or write operation in one clock cycle can be performed. Competition to LS from multiple sources is solved by arbitration.

Tab. 15.1 details the arbitration priorities. DMA transfers always have highest priority but their maximal impact on LS is limited. These operations occupy, at most, one of every eight cycles (one of sixteen for DMA reads, and one of sixteen for DMA writes) to the LS. Thus, the impact of DMA reads and writes on LS availability for loads, stores, and instruction fetches is small.

SPU instruction execution flow is most efficient either with no branches or with correctly predicted branches. A branch instruction might disrupt the sequential flow. Correctly predicted branches execute in one cycle, but a not correctly predicted branch results a penalty of 18 to 19 clock cycles, depending on the address of the branch target. Thus mispredicted branches can seriously degrade program performance. Branch instructions also restrict a compiler's ability to optimally schedule instructions by creating a barrier on instruction reordering. The Synergistic Processor Unit Instruction Set include *branch hints* to predict in advance the destination of a nearby branch.

Floating-Point Support

Single (double) precision floating-point operations are performed in a 4 (2) vector SIMD fashion. The data formats for floating-point operations are those defined by the Institute of Electrical and Electronics Engineers (IEEE) Standard 754, but the results calculated by single-precision instructions deviate from this standard. This deviation in single precision must be taken into account by application programmers, e.g. when carrying out large sums.

The Cell processor is capable of performing two double-precision floating-point operations per cycle with a 9-clock-cycle latency. The SPU instruction set provides a fused multiply-add operation which results in 4 double-precision floating-point operations per SPU per clock cycle peak performance.

Memory Flow Controller

The MFC provides the SPE's interface between LS and MS and other SPE's LS and system devices. It contains a DMA controller which executes data transfers between LS and the destination or source storage area.

Software on the SPE, the PPE, and other SPEs and devices use MFC commands to initiate DMA transfers, query DMA status, perform MFC synchronisation and interprocessor-communication via mailboxes and signal-notification. The MFC commands implementing DMA transfers between the LS and MS are called DMA commands. The MFC maintains two separate command queues, an SPU command queue for commands from the MFC's associated SPU, and a proxy command queue for commands from the PPE and other SPEs and devices.

The MFC supports out-of-order execution of DMA commands. This enables the MFC to reorder execution of DMA commands to achieve a higher aggregate bandwidth.

Tag-Group Identifiers DMA commands are tagged with one of 32 tag-group identifiers. This enables the program to determine the status of issued DMA commands. A synchronisation command can be issued to the MFC to wait for the completion of queued commands in a tag-group.

Fence and Barrier Option Control over the execution order of DMA commands within a tag-group can be taken by programs with the fence or barrier option. A fenced DMA command is not executed until all previously issued DMA commands within the same tag-group have been performed. DMA commands issued after the fenced DMA command might be executed before the fenced DMA command. A DMA command with the barrier option and all the DMA commands issued after the barrier command are not executed until all previously issued commands in the same tag group have been performed.

Inbound and Outbound Transfer The terms *inbound* and *outbound* for DMA transfers are defined from the SPU point of view. An *inbound* transfer is a DMA transfer from main storage to the local storage. An *outbound* transfer is a DMA transfer from local storage to main storage.

Direct Memory Access Controller

The MFC's Direct Memory Access Controller (DMAC) implements the DMA transfers. Programs running on the associated SPU, the PPE, or another SPE or device, enqueue DMA commands to the DMAC command queue. SPU computation continues after issuing a DMA transfer command to the queue. This queue takes up to 16 DMA commands, which are eligible by the DMAC for execution. The DMAC executes DMA commands from the queue autonomously, even in parallel, which allows up to 16 DMA transfers being executed in parallel.

DMA List Transfers

A whole set of DMA transfers can be issued to the MFC at once via a DMA list transfer command. The list elements are stored sequentially in LS and are passed to the MFC through a pointer to the first list element together with its size and a DMA list command. The list specifies a sequence of DMA transfers between a single area of LS and possibly discontinuous areas in main storage. DMA list commands can only be issued by programs running on the associated SPE, but the PPE or other devices (including other SPEs) can create and store the lists in an SPE's LS. DMA lists can be used to implement scatter-gather functions between main storage and the LS.

SPU Decrementer

The SPU features a 32 bit decrementer which counts down at a fixed ratio to the processor clock. An SPU decrementer is accessed through two channels: The SPU Write Decrementer

Channel and the SPU Read Decrementer Channel. The SPU decrementer can be used to measure the execution time of an SPU program. In this work the decrementer speed was measured on a IBM PowerXCell 8i Processor on a QS22 Cell Blade to be 1/120 counts per clock cycle.

SPU Channels

The SPU communicates with its MFC and with all other processor elements (PPE or other SPUs) and devices through its SPU channels. Channels are unidirectional interfaces for sending and receiving variable-size (up to 32 bit) messages, and for sending commands (such as DMA transfer commands) to the SPE's associated MFC. SPE software accesses channels with special channel-read and channel-write instructions. These instructions are used to initiate MFC commands, query DMA and SPU status, send mailbox and signal-notification messages, and access auxiliary resources such as the SPE's decrementer.

SPU Mailboxes

SPU Mailboxes are facilities to send and receive short (up to 32 bit) messages from an to other processor elements, i.e. other SPUs, the PPE or other devices.

There are two different SPU mailboxes implemented with MFC channels.

- SPU Read Inbound Mailbox (SPU receives a message)
- SPU Write Outbound Mailbox (SPU sends a message)

These mailbox services send and receive 32 bit messages. The PPE also has access to these mailboxes by reading and writing to MFC memory mapped IO registers, see App. D.4 for a more detailed description.

```
allocate $B$
**for all** vector sites **do**
  initiate DMA transfer $B$
  wait DMA transfer $B$
  calculate $B$
**end for**
```

Algorithm 1 – Sequential processing of a vector object residing in main memory. No overlap of DMA transfers and computation takes place.

15.4 Programming the SPU

15.4.1 C-language intrinsics

The SPU instruction set provides C-language intrinsics to allow for access to certain SPU functionality. The code for the intrinsic is usually inserted inline, avoiding the overhead of a function call. Using intrinsic results often in code with a higher performance than using the equivalent inline assembly. This it due to the optimiser's built-in knowledge of how intrinsics behave, so some optimisations are available that are not available when inline assembly is used. Also, the optimiser can expand the intrinsic differently, align buffers differently, or make other adjustments depending on the context and arguments of the call.

For example to access the fused multiply-add the SPU instruction set provides the C-language intrinsics **spu_madd()** which can be used in the following way:

```
vector double a,b,c,d;
d = spu_madd(a,b,c)
```

which multiplies **a** and **b** and adds **c** to the result and stores it into d. Using this intrinsic translates directly into the corresponding machine instruction.

15.4.2 Double-Buffering

To hide memory latencies and DMA transfer overheads double-buffering techniques can be used. Let us consider an algorithm that sequentially performs computation on all elements of a vector. Alg. 1 allocates one buffer B in LS and then iterates over the whole vector by first transferring a part of it and then carrying out the calculation. This sequence has no overlap between data transfer and computation, i.e. either the DMA transfer is carried out or the calculation.

The performance can be improved by allocating two LS buffers, B_0 and B_1, and overlapping computation and DMA transfers. This technique is known as double-buffering, see Sec. 18.4.1.

15.4.3 Dual-Issue

In order to achieve a high dual-issue rate the compiler chooses adequate instructions – if possible – reorders them in an appropriate way. Thereby it is limited to code sequences between branches and is limited by data dependencies.

Unrolling loops and thus interleaving computation with control structures typically enlarges code portions between branches and thus gives the compiler more freedom in reordering instructions. Typically this helps to improve the dual-issue rate.

15.4.4 SPU Code Overlays

The CBEA development tools offer code overlay techniques to overcome physical limitations of the LS.

Code overlays are program segments which are stored in main storage and are loaded into the SPU's LS on demand. When the SPU program branches to code which currently not resides in LS but in an overlay segment in MS, then this segment is copied into the LS and thereafter the branch is executed. This transfer overwrites possibly another overlay segment which is not required by the program at this time.

When using code overlays the LS is divided into a root segment, which is loaded during the whole execution time of the program, and one or more overlay regions, where overlay segments are loaded when needed. Any given overlay segment will always be loaded into the same region. A region may contain more than one overlay segment, but a segment will never cross its region's boundary. A segment is the smallest unit which can be loaded

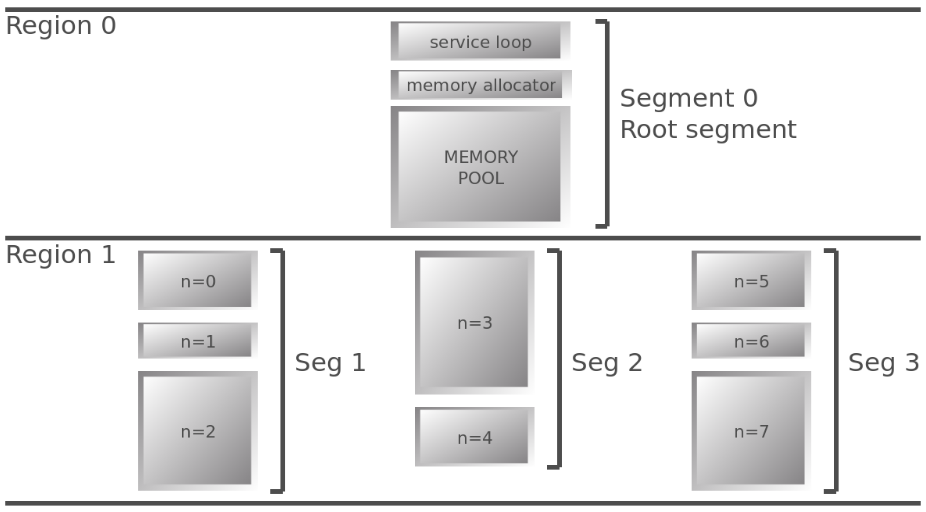


Figure 15.3 – The overlay structure of an SPU program. The root segment 0 which contains program parts that are frequently used is located in overlay region 0 and loaded at any time. Less frequently used functions, here numbered with index *n*, are grouped into overlay segments (Seg. 1 to 3) and placed into the same overlay region. They are loaded into LS at demand.

as a logical entity during execution. Segments can contain any set of program sections, uninitialised or initialised data.

The code overlay technique is completely implemented in the SPU linker. The linker can map two or more code segments to the same physical address in LS. It generates small fractions of code, so-called call-stubs and generates associated tables for overlay management. At execution time when a call is made from an executing segment to another segment that is not loaded into local storage, the code overlay manager residing in the root segment transfers the code from main storage into local storage and issues the branch to the function's start.

Branch instructions in the original code that branch to code located in overlay segments are replaced by branches to the call-stubs. The overlay manager then determines from the tables whether the function's code needs to be loaded.

To convert an existing SPU program to an program that uses code overlays a linker script must be created which specifies which segments of the program is subject to overlays. Lst. 15.1 shows an example of such a linker script. The linker prepares the required segments so that they are loadable on demand and also adds the call-stubs to the code.

Listing 15.1 – Example of a linker script for an SPU program with code overlays. Two overlay regions are defined each of which contains two segments

```
OVERLAY {
 .segment1 {./sc.o(.text)}
 .segment4 {./sg.o(.text)}
}
OVERLAY {
 .segment2 {./sd.o(.text) ./se.o(.text)}
 .segment3 {./sf.o(.text)}
}
```

The code overlay technique uses inbound DMA transfers. As a result, when using data buffers in overlay segments one must consider the scope of the data. Data in an overlay segment might get overwritten when exiting the function and returning to it at a later time. To avoid this situation all data sections can be kept in the root segment which is never used as a overlay region. If the data size is too large to be stored in the root segment, then sections for transient data may be included in overlay regions. In this case the call-graph of the program must be analysed and the overlay structure created in such a way that branching does not lead to overwriting necessary data.

Program parts which are frequently used should be placed into the root segment whereas less frequently used program parts are typically placed into overlay segments. Fig. 15.3 displays an example of an overlay structure using four overlay segments and two overlay regions.

The size of an segment is the sum of its code and data sections. The size of an overlay region is the size of its largest segment. The memory requirement for the whole program can be calculated by summing the sizes of all overlay regions and adding extra storage requirements for management code and tables.

15.4.5 Integer Multiplication

On the SPU 32 bit integer multiplication is achieved with 4 16 bit integer multiplication. If the algorithm does not require 32 bit integers it is recommended to use 16 bit integers to avoiding 32 bit integer multiplies.

15.4.6 Scalar Data Types

Load and store instructions of data types smaller than a SIMD vector, e.g. scalar types, require additional rotate instructions and have long latencies. For program parts that are executed frequently it is recommended to avoid the usage of scalar types. Instead it is recommended to use a whole SIMD vector even if just a scalar type is needed. This seems to be a waste of local storage. But scalar types require the compiler to generate additional bit shuffle operations for every load and store that is performed.

Chapter 16

QCD Data Parallel

16.1 Overview

QDP++ provides a data-parallel programming environment suitable for essentially any kind of lattice QCD application and is the basis for a suite of LQCD applications called Chroma [155]. The interface provides a level of abstraction such that high-level user code can be run unchanged on a single processor node or a collection of nodes with parallel communications. Architectural dependencies are hidden below the interface.

The core functionality of QDP++ is provided by the Portable Expression Template Engine (PETE) [153] which in turn is an extensible implementation of the C++ expression template technique.

Function and class templates together with function and operator overloading offer the possibility to represent expressions as C++ types. This technique is commonly referred to as Expression Templates and was first introduced by Todd Veldhuizen [159] and David Vandevoorde.

Expression templates offer the possibility to pass expressions as function arguments. The compiler inlines the expression into the function body. Typically, this results in faster and more convenient code than C-style callback functions.

Expression templates can also be used to evaluate vector expressions in a single loop without instantiating temporaries of the whole vector class which typically results in a higher performance. PETE is an extensible implementation of the expression template technique.

This work gives a comprehensive introduction to expression templates and employs a vector class making use of this technique.

This chapter is organised as follows. Sec. 16.2 briefly introduces the QDP++ lattice-wide data types and operations. Sec. 16.3 introduces the necessary C++ language elements which are essential for expression templates. Sec. 16.4 details on the fundamental concepts of expression templates. Sec. 16.5 shows the traditional approach to a C++ vector class. Sec. 16.6 applies the expression template technique to the vector class and shows how it is used in modern Lattice QCD application libraries like QDP++.

16.2 Lattice-Wide Data Types and Operations

QDP++ models the tensor product structure of Lattice QCD objects through a series of nested templates. The indices of lattice fermion fields might (but don't have to) follow the following structure:

$$\text{Site} \otimes \text{Dirac} \otimes \text{Color} \otimes \text{Complex} \tag{16.1}$$

In QDP++ one models this type by the following (nested) C++ templated type:

```
OLattice< PSpinVector< PColorVector< RComplex < Real >, Nc >, Ns > >
```

where **Real** is the type of the complex components, defined as either float or double and N_c and N_s are the numbers of spin and colour components defined before compile-time of QDP++. QDP++ operations iterate through the data type structure from the outer to inner most level. The outer most level, i.e. the lattice site index, is evaluated by PETE. Then the inner levels are traversed from the Dirac index down to the complex index. Most commonly used data types are predefined and type aliases are provided, e.g.:

```
typedef OScalar< PScalar< PScalar< RScalar<REAL> > > > Real;
typedef OLattice< PScalar< PScalar< RScalar<REAL> > > > LatticeReal;
typedef OLattice< PSpinMatrix< PColorMatrix< RComplex<REAL>, Nc>, Ns> >
        LatticePropagator;
```

```
typedef OLattice< PSpinVector< PColorVector< RComplex<REAL>, Nc>, Ns> >
        LatticeFermion;
```

QDP++ provides lattice-wide expressions with overloaded C++ arithmetic operators. The expression

$$\psi_0 = \kappa\psi_1 + \psi_2 \tag{16.2}$$

with the fermion fields ψ_0, ψ_1, and ψ_2 and the real number κ is implemented making use of the lattice wide data types and operations offered by QDP++ as

```
LatticeFermion psi0, psi1, psi2;
Real      kappa;

psi0 = kappa * psi1 + psi2;
```

Since the formulae maintain their original form when implemented with QDP++ this offers a very intuitive way of generating the expressions to the application developer.

Traditional overloading of C++ arithmetic operators typically involves creating and copying of temporaries of the instantiated lattice types. The expression **psi1 + psi2** triggers the execution of **operator+** which creates a temporary vector, and loops over the vector index to add the elements of the two vectors **psi1** and **psi2** and to store each element in the temporary vector. However, in order to achieve a high performance software the movement or replication of data must be minimised, especially when lattice wide data objects are involved.

The Expression Templates (ET) technique uses C++ recursively defined templates for transforming C++ statements or expressions into other statements with the same effect but possibly with a higher performance [159]. It is a C++ technique for passing expressions as function arguments. The expressions are in turn inlined into the function's body, which enables the possibility to implement vector operations, like e.g. the execution of **operator+** on vector types while avoiding temporaries of vector objects. This avoids allocation of vector instances and can result in code with a higher performance. However, at the level of the vector element a temporary instance is still created, leaving the level of optimisation achieved of the final software to some extent to the compiler's ability to detecting these temporaries and eliminating them.

The Portable Expression Template Engine (PETE) is an extensible implementation of the expression template technique [153]. It is part of the Parallel Object-Oriented Methods and Applications (POOMA) collection of templated C++ classes [154].

16.3 C++ Templates

In computer science, polymorphism is a programming language feature that allows different data types to be handled using a uniform interface. The concept of parametric polymorphism applies to both, data types and functions. A function that evaluates to or is being applied to values of different types is known as a polymorphic function. A data type that is of a generalised type is called a polymorphic data type.

The C++ programming language defines *templates* as the mechanism for parametrising a class or a function with a type or a list of types. This allows for *generic programming*, a style of computer programming in which algorithms are written in terms of arbitrary types, i.e. to separate the implementation of algorithms from data types.

The following example demonstrates a definition of a class with a template parameter.

```
template < typename T, int N >
class PVector
{
public:
  PVector() {}

private:
  T F[N];
};

void foo(){
  PVector< double , 3 >                vec0;
  PVector< PVector< double , 4 > , 3 > vec1;
}
```

A class template **PVector** is defined that represents a primitive vector class which stores an array of N elements of an arbitrary type T.

Template parameters are either typename template parameters or non-type template parameters. The above given class template is parametrised with a typename parameter (**typename T**) and a non-type parameter (**int N**). And the instance **vec0** is a vector of 3 built-in types **double**.

The class template instantiation requires all template parameters to be specified or determinable by the compiler.

Since a template instantiation of a class template is a fully qualified type, it can be used, even recursively, as a template typename parameter as demonstrated in case for the instance **vec1**.

16.3.1 Default Arguments for Template Parameters

Template parameters may have default arguments. The compiler applies the default values to all template parameters for which no specification was found, i.e. default arguments have to lowest priority of all template parameter specifications. The set of default template arguments accumulates over all declarations of a given template. The following example demonstrates this:

```
template < class T=double, int N=3 > class PVector;

void foo(){
  PVector<>       d3;
  PVector<float> f3;
}
```

The type of **d3** is **PVector⟨double,3⟩**, type of **f3** is **PVector⟨float,3⟩**.

The scope of a template parameter starts from the point of its declaration to the end of its template definition. This implies the possible usage of the name of a template parameter in other template parameter declarations and their default arguments. See the following example:

```
template<typename T> class A;
template<typename T, typename U = A<T> > class B;
```

In the second line the parameter **T** is in scope when defining a default value for parameter U.

16.3.2 Explicit Specialisation

Instantiating a template with a given set of template arguments causes the compiler to generate a new class definition based on those template arguments. This behaviour can be overridden. For a given set of template arguments the specification of the function or class template can be given explicitly.

```
template<>
class PVector<int,3>
{
  // special implementation
};
```

The **template**⟨⟩ prefix indicates that the following template declaration takes no template parameters. The declaration name is the name of a previously declared template. Note that it is possible to forward-declare an explicit specialisation thus the declaration body is optional, at least until the specialisation is referenced.

16.3.3 Partial Specialisation

A partial specialisation is a generalisation of explicit specialisations. An explicit specialisation only has a template argument list. A partial specialisation has both a template argument list and a template parameter list. The compiler uses the partial specialisation if its template argument list matches a subset of the template arguments of a template instantiation. The compiler will then generate a new definition from the partial specialisation with the rest of the unmatched template arguments of the template instantiation. To continue the above example:

```
template< typename T >
class PVector<T,3>
{
};
```

The GNU C++ Compiler allows for partial specialisation either of class templates or function templates.

16.3.4 Dependent and Qualified Names

Template meta-programming requires to distinguish between *dependent/non-dependent* and *qualified/unqualified* names and qualifiers.

Dependent and non-dependent names

In the C++ language a name (including qualifiers) gets the quality of being *dependent* if the name depends on a template parameter, i.e. a (possibly not yet) fully qualified C++ type.

When using **typedef** statements, this quality is passed unchanged to the newly defined type. The following example illustrates this:

```
template <typename T>
class A {
   int i;
   vector<int> vi;
   vector<int>::iterator vitr;

   T t;
   vector<T> vt;
   typedef typename vector<T>::iterator viter;
};
```

The first 3 class members have non-dependent names whereas the last 3 have dependent names.

Qualified and Unqualified Names

In the C++ language a name gets the quality of being *qualified* if the name is preceded with a scope qualifiers. This also includes *dependent* scope qualifiers. The following example illustrates this:

```
using namespace std;
template< typename T >
void bar( T t )
{
  int i;
  typedef typename T::type local;
  cout << std::endl;
}
```

The names **i** and **cout** are *unqualified* names whereas **type** and **endl** are *qualified* names. In this example **local** is a *qualified dependent* name.

The typename keyword

Two usages of the **typename** keyword exists: As a name qualifier and as a template parameter.

As a name qualifier the **typename** keyword specifies a name to be treated as a type. Its use is mandatory when followed by a *qualified dependent* name.

```
  template<typename T>
2 class A
  {
4   typedef T::x y;
            T::x z;
6   typedef char C;
            A::C d;
8 };
```

The statements in line 4, 5 and 7 are ill-formed and generate a compiler error. In line 4, the name **T::x** is a *qualified dependent* name, it must be preceded with the keyword **typename**. In line 5, the use of the name **T::x** is ambiguous; it might refer to a class member or a type. In line 7, again the name **A::C** is a *qualified dependent* name, it must be preceded with the keyword **typename**. Here the corrected version:

```
template<typename T>
class A
{
  typedef typename T::x y;
  typename T::x z;
  typedef  char C;
  typename A::C d;
};
```

The second usage of the keyword **typename** occurs when specifying a template parameter. The keyword **typename** is synonymic to the keyword **class** in template parameter declarations. These two statements are identical:

```
template< class T >    class A;
template< typename T > class A;
```

16.4 Expression Templates

Function and class templates together with function and operator overloading offer the possibility to represent expressions as C++ types. This technique is commonly referred to as Expression Templates and was first introduced by Todd Veldhuizen [159] and David Vandevoorde.

Expression templates offer the possibility to pass expressions as function arguments. The compiler inlines the expression into the function body. Typically, this results in faster and more convenient code than C-style callback functions.

This work presents the fundamental concept of expression templates with the help of developing a basic example. A function which carries out numerical integration is developed. The function takes the algebraic expression and the integration boundaries as its arguments.

In C, this problem is usually solved passing pointers to a callback function containing the expression. The following example demonstrates this:

```
double integrate( double (*func)(double) , double xmin , double ymax)
{
  // code
}

double myfunc(double x)
{
  return (x/(1.+x));
}

integrate( myfunc , 0. , 10. );
```

The problem with callback functions is that repeated calls generate a lot of overhead, especially if the expression which the function evaluates is short. The compiler is not able to inline the function (to eliminate the function call) since it is referred to via a pointer, i.e. dereferenced at run time.

The technique of expression templates allows expressions to be passed to functions as an argument and inlined into the function body. The following listing demonstrates the final usage of the function with expression templates.

```
Variable x;
double result = integrate( ( 1. - x ) / ( 1. + x ) , 0. , 10. );
```

The function **integrate** is defined as a function template and the compiler produces a function instance which contains the according expression inlined into the function's body.

Therefore the expression must be parsed at compile time, and stored as nested template arguments of an C++ type representing the expression.

A C++ type is required that represents an expression and that provides a public access function to evaluate the expression for a specific value. Certainly the expression can hold a number of different types. In the integration example at least three different types are required: A type representing the variable, a type representing a constant, and some type of compound object like addition or division.

At first sight the concept of polymorphism of the C++ language looks like a good candidate to meet the requirements. Defining a class for the expression with a virtual class member function to evaluate the expression and deriving classes for a variable, constants and compound objects. But, whenever a class declares virtual functions or is derived directly or indirectly from a class which declares virtual functions, the compiler adds an extra hidden member variable which points to the virtual table (so called vtable).

The virtual table of a class is an array of pointers to the virtual functions. The entries in the virtual table are updated at run time to correspond to the according function address determined by the run time type information (RTTI) system – runtime binding take place. Again the function call overhead would have to be paid and the net profit over the C function call-back implementation is zero.

To obtain code with a high performance the more appropriate approach to polymorphism is to use template instantiation. Let us look at the following classes:

```
struct Foo1 {
  void bar() {
    // implementation
  }
};

struct Foo2 {
  void bar() {
    // implementation
  }
};
```

The class **Foo1** and **Foo2** do have common interfaces but do not have a common base class. On can wrap these classes using a class template that has the same public interface (or parts of it):

```
template<typename F>
struct BaseFoo {
  const F& f_;
  BaseFoo(F& f) : f_(f) {}

  void bar() {
    f_.bar();
  }
};
```

This definition ensures that the class template **BaseFoo** can only be instantiated for classes which have the **bar()** method in their public interface. The C++ syntax allows to define the data member **f_** of a constant reference type since it is initialised during construction. This form of polymorphism does not require any virtual tables and is thus called (template driven) static polymorphism.

The second form of static polymorphism is shown in the next example:

```
template< typename T , typename C >
class Base
{
public:
  inline const T& elem() const
  {
    return static_cast<const C*>(this)->elem();
  }
};
```

```
template< typename T >
class Derived: public Base< T , Derived< T > >
{
  inline const T& elem() const
  {
    return F;
  }
private:
  T F;
};
```

The derived class is given as a template parameter to the base class. The member function of the derived class is accessed through a **static_cast**$\langle\rangle$ operation of the **this** pointer of the base to a pointer to the derived class.

static_cast can perform conversions between pointers to related classes, not only from the derived class to its base, but also from a base class to its derived. This ensures that at least the classes are compatible if the proper object is converted, but no safety check is performed during runtime to check if the object being converted is in fact a full object of the destination type. Therefore, one has to ensure that the conversion is safe. On the other hand, the overhead of the type-safety checks of **dynamic_cast** is avoided.

This technique allows for inline function calls since it avoids function pointers and lookups in the virtual table. Static polymorphism will be applied later when developing a C++ class library for a vector classes which uses expression templates.

Template driven static polymorphism is used to define a class template that encapsulates an expression. Calling **operator()** starts evaluation of the expression. Its implementation calls the corresponding operator of the interfaced class.

```
template<typename E>
struct Expr {
  Expr(E e) : e_(e) {}
  double operator() (double d) {
    return e_(d);
  }
  E e_;
};
```

The class template is parametrised by one typename template parameter **E** which in this example represents either a variable, constant or some type of compound expression type. The former ones are implemented as follows:

```
struct Constant {
  Constant(double d) : d_(d) { }
  Constant(int d) : d_(d) { }
  double operator() (double) {
    return d_;
  }

  double d_;
};

struct Variable {
  double operator() (double d) {
    return d;
  }
};
```

Note that the class template **Constant** contains a not explicit constructor resulting in an implicit conversion rule which is exactly what is wanted here to have the compiler transform the expression into the C++ type.

The compound type should represent a binary operation, for example an addition, and implement the same public access operator like the interface class **Expr**. It furthermore should be constructable from any two objects of the templated type **Expr** and another templated type representing the operation and is therefore a class template:

```
template<typename E1, typename E2, typename Op>
struct BinaryExpr {
  BinaryExpr(Expr<E1> l, Expr<E2> r) : l_(l), r_(r) {
  }

  double operator() (double d) {
    return Op::apply(l_(d), r_(d));
  }

  Expr<E1> l_;
  Expr<E2> r_;
};
```

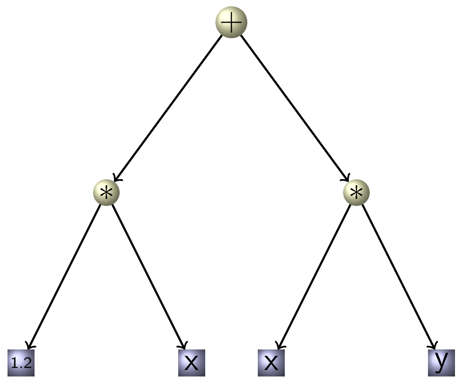


Figure 16.1 – Tree representation of the expression $1.2*x+x*y$. Where x and y denote vectors and the numerical constant is a scalar. The inner nodes (circles) of the tree represent binary expressions. The leaf nodes (blocks) represent the data objects.

The class template **BinaryExpr** is parametrised with three typename template parameters. The first two parameters represent the two constituents of the binary expression. They can be of type **BinaryExpr**, i.e. recursive expressions are supported and thus the expression is referred to as an expression tree.

More generally spoken the template parameter **E1** and **E2** represent either *leafs* or *nodes* of the expression tree. Fig. 16.1 shows a graphical representation of an expression tree.

The third parameter represents the actual operation for this binary expression. An instance of it will never be created. As a consequence the member function to be called from **BinaryExpr** has to be declared **static**. Since the **operator()** cannot be declared static, an **apply()** member function is used instead.

```
struct Add {
  static double apply(double l, double r) {
    return l+r;
  }
};
```

Taking a closer look at the algebraic expression that should be represented by the expression type

```
int main()
{
  Variable x;
  evaluate( ( 1.0 - x ) / ( 1.0 + x ) , 0.0, 10.0 );
  return 0;
}
```

one sees that operator overloading for **BinaryExpr** is required. The involved operators are overloaded as follows:

```
template<typename E1, typename E2>
Expr<BinaryExpr<Expr<E1>, Expr<E2>, Add> >
operator+ (E1 e1, E2 e2)
{
  typedef BinaryExpr<Expr<E1>, Expr<E2>, Add> ExprType;
  return Expr<ExprType>( ExprType(Expr<E1>(e1), Expr<E2>(e2)) );
}
```

```
template<typename E1>
Expr<BinaryExpr<Expr<E1>, Expr<Constant>, Add> >
operator+ (E1 e1, double d)
{
  typedef BinaryExpr<Expr<E1>, Expr<Constant>, Add> ExprType;
  return Expr<ExprType>( ExprType(Expr<E1>(e1),
                                  Expr<Constant>(Constant(d))) );
}
```

The operator returns an instance of type **Expr** parametrised with the template typename parameter **BinaryExpr**. Which is in turn parametrised with **Expr**⟨**E1**⟩, **Expr**⟨**E2**⟩ and **Add**. The former two encapsulate the expressions involved in the operation and the latter gives the type of the operation.

The operator must return a constructed object of the return type which is in this case **BinaryExpr**⟨**Expr**⟨**E1**⟩**, Expr**⟨**E2**⟩**, Add**⟩ aliased by a **typedef** statement to **ExprType**. An instance of the type **ExprType** is constructed with instances of **Expr**⟨**E1**⟩ and **Expr**⟨**E2**⟩ constructed with the instances **e1** and **e2**.

It is mandatory to overload this operator with all combinations of **Expr**⟨**Constant**⟩ and **template**⟨**typename E**⟩ **Expr**⟨**E**⟩ occurring in the expression. Since the returned object is of type **Expr** another **BinaryExpr** can be recursively instantiated with it.

This closes the introduction on expression templates and leads to applying this technique to a vector class.

16.5 Object-Oriented C++ Vector Class

First the traditional implementation of C++ vector classes is recalled. Typically a vector class is implemented the following way using the object-oriented language part of C++:

```
template < typename T >
class Vector
{
public :
  Vector();
  Vector(int size);
  Vector(const Vector & v);
  virtual ~Vector();

  Vector   operator+=( const Vector & right );
  Vector   operator*=( const Vector & right );
  Vector & operator=( const Vector & right);

  inline T& elem(int i);
  inline const T& elem(int i) const;
  inline const int& size() const;

protected :
  int size_;
  T * F;
};
```

This defines the vector class with dynamical memory allocation. The global arithmetic operators are overloaded to provide for a intuitive API. The global **operator+** is overloaded in the following way:

```
template<class T>
Vector<T> operator+(Vector<T> & lhs, Vector<T> & rhs)
{
  Vector<T> tmp(lhs.size_);

  // loop over vector index and add element-wise

  return tmp;
}
```

Now one is able to use the vector class like:

```
int main()
```

```
{
  const int sz = 1000;
  Vector<double> d(sz), v1(sz), v2(sz);

  d = v1 + v2;
}
```

Unfortunately this naive approach leads not to optimal performance: The expression **v1+v2** triggers execution of **operator+** which creates a temporary vector, and loops over the vector index to add the elements of the two vectors **v1** and **v2** and to store each element in the temporary vector. Finally, the call to the assignment operator executes a second loop. Thus, this simple statement is equivalent to:

```
  Vector<double> temp_1;
  for ( int i = 0 ; i < sz ; ++i )
    temp_1.elem(i) = v1.elem(i) + v2.elem(i));

  for ( int i = 0 ; i < sz ; ++i )
    d.elem(i) = temp_1.elem(i));
```

Clearly, if this program was written in C instead of C++, the two loops can be combined, and the temporary vector eliminated:

```
  Vector<double> d(sz), v1(sz), v2(sz);
  for ( int i = 0 ; i < sz ; ++i )
    d.elem(i) = v1.elem(i) + v2.elem(i);
```

However, this implementation dismisses the possibility to use C++ classes and global operators. This is a substantial disadvantage in large software projects.

16.6 Portable Expression Template Engine

Since the traditional approach to a vector class in C++ does not lead to optimal performance, this work shows now how to extend the vector class using the expression templates technique. The final class should feature:

- during evaluation no creation of temporary vector objects necessary,
- single loop evaluation,

- different leafs types, e.g. scalars, vectors.

First, the class definition will be introduced that stores the vector data and interfaces access member functions. Since the library should operate with both vectors and scalars polymorphism is required to handle both in a uniform way. Here static polymorphism is used and the base class is introduced later. For convenience only the class definition for the vector type is given explicitly – the implementation of the class that represents scalar types follows in a similar manner.

```
template<class T>
class OLattice: public QDPType<T, OLattice<T> >
{
public:
  OLattice( int size ): size_(size) { F = new T[size]; }
  ~OLattice() { delete[] F; }

public:
  inline       T& elem(int i)       {return F[i];}
  inline const T& elem(int i) const {return F[i];}

  inline const int& size() const { return size_; };

private:
  T *F;
  int size_;
};
```

This implementation follows closely the one in QDP++.

The class template **OLattice** is parametrised by one typename template parameter which represents the type of objects to be stored. The constructor takes one integral argument giving the number of elements in the vector. Heap memory is reserved during construction of an **OLattice** instance and is deleted when the instance goes out of scope.

The member function **elem(int i)** returns a modifiable reference to the i-th element of the vector thus providing the access interface to the vector. In order to give the compiler more freedom to carry out optimisations one overloads the member function **elem(int i)** returning a constant reference type. Since C++ does not allow overloading on the return type, the overloaded function is declared **const**.

The base class is defined as

```
template< typename T , typename C >
class QDPType
{
public:

  const T& elem(int i) const
                        {return static_cast<const C*>(this)->elem(i);}
        T& elem(int i)
                        {return static_cast<const C*>(this)->elem(i);}

  const T& elem() const {return static_cast<const C*>(this)->elem();}
        T& elem()       {return static_cast<const C*>(this)->elem();}
};
```

The class template **QDPType** is parametrised by two typename template parameters. The first parameter represents the type of objects to be stored, i.e. the same type as used in the definition of **OLattice**. The second parameter is the container type itself which enables static polymorphism.

Notice, that **QDPType** implements calls to access functions that might not be implemented in the derived class. The access function for the scalar type **elem()** has no counterpart in **OLattice**. Since **static_cast** turns off type checking at compile time one must make sure when using the class to call the correct access function.

Similar to the previous section a class template is required that represents the binary operation:

```
template<class Op, class Left , class Right>
class BinaryNode
{
public:

  inline
  BinaryNode(const Left &l , const Right &r) : left_m(l), right_m(r)
  {}

  inline
  const Op &
  operation() const { return op_m; }

  inline
  typename DeReference<Left >::Return_t
  left() const { return DeReference<Left >::apply(left_m); }
  inline
  typename DeReference<Right>::Return_t
  right() const { return DeReference<Right>::apply(right_m); }

private:
  Op    op_m;
  Left  left_m;
  Right right_m;
};
```

The class template **BinaryNode** is parametrised by three typename template parameters. The first determines the operation. On construction it is not necessary to pass an instance of the class representing the operation since the operation is not templated and can be constructed by the default constructor at construction time of **BinaryNode**. The instance of the operation is accessible through the member function **operation()** which returns a reference to the instance. The other two typename template parameters represent the subexpressions of the binary operation. They can be accessed via the member functions **left()** and **right()** which return dereferenced types of the objects.

These class templates **Reference** and **DeReference** wrap types in a reference type and a dereferenced type. There is no essential need for these wrappers to be present – just for the sake of clarity writing **Reference⟨A⟩** is preferred to writing **A&**.

```
template<class T>
struct Reference
{
  typedef T Type_t;

  inline
  Reference(const T &reference)
    : reference_m(reference)
  { }

  inline
  const T &reference() const
  {
    return reference_m;
  }

  const T &reference_m;
};
```

```
template<class T>
struct DeReference
{
  typedef const T& Return_t;
  static inline Return_t apply(const T &a) { return a; }
};
```

In order to have the compiler create a type **BinaryNode** out of the addition of two **QDPType** the **operator+** must be overloaded. It takes two instances of type **QDPType** as its arguments and returns an instance of type **BinaryNode** parametrised with the addition operation type **OpAdd**.

```
template<class T1,class C1,class T2,class C2>

inline typename MakeReturn<
  BinaryNode<
    OpAdd,
    typename CreateLeaf<QDPType<T1,C1> >::Leaf_t,
    typename CreateLeaf<QDPType<T2,C2> >::Leaf_t
    >,
  typename BinaryReturn<C1,C2,OpAdd>::Type_t
  >::Expression_t

operator+(const QDPType<T1,C1> & l,const QDPType<T2,C2> & r)
{
  typedef BinaryNode<
    OpAdd,
    typename CreateLeaf<QDPType<T1,C1> >::Leaf_t,
    typename CreateLeaf<QDPType<T2,C2> >::Leaf_t
    > Tree_t;
  typedef typename BinaryReturn<C1,C2,OpAdd>::Type_t Container_t;
  return MakeReturn<
    Tree_t , Container_t
    >::make(Tree_t(
                   CreateLeaf<QDPType<T1,C1> >::make(l),
                   CreateLeaf<QDPType<T2,C2> >::make(r)
                  )
             );
}
```

This function template is parametrised with four typename template parameters in order to parametrise the two **QDPType** arguments involved in the operation. The return type is basically of type **BinaryNode** parametrised with the operation type **OpAdd** and the two subexpressions. Here additional wrappers are involved: **CreateLeaf⟨T⟩::Leaf_t** returns a reference to the type **T** and **MakeReturn⟨T,C⟩::Expression_t** is a wrapper which returns **QDPExpr⟨T,C⟩**, a type that represents an expression:

```
template<class T>
struct CreateLeaf
{
};

template<class T, class C>
struct CreateLeaf<QDPType<T,C> >
{
  typedef QDPType<T,C> Inp_t;
  typedef Reference<Inp_t> Leaf_t;

  inline static
  Leaf_t make(const Inp_t &a) { return Leaf_t(a); }
};

template<class T, class C>
struct MakeReturn
{
  typedef QDPExpr<T, C>  Expression_t;
  inline static Expression_t make(const T &a) { return Expression_t(a); }
};
```

The container type **C** passed as a template argument to **MakeReturn⟨T,C⟩** is evaluated at compile-time by a so-called *trait-class template* **BinaryReturn⟨C1,C2,OpAdd⟩**.

A *trait class template* provides a way of associating information with a compile-time entity, i.e. a type, integral constant, or address. A key feature of trait class templates is that they are *non-intrusive*. They allow for associating information with arbitrary types, without intrusion to the type including built-in types. Typically, traits are specified for a particular type by (partially) specialising the traits template.

In the case of an addition of two types of **QDPType⟨T,C⟩** with the same container type a specialisation of **BinaryReturn⟨T1,T2,Op⟩** is not necessary since the default implementation already returns the correct type.

```
template<class T1, class T2, class Op>
struct BinaryReturn
{
  typedef typename Promote<T1, T2>::Type_t Type_t;
};
```

In **BinaryReturn** another trait class, i.e. **Promote**$\langle$**T1,T2**$\rangle$ is used which in its default implementation just defines a **typedef** of **T1** to **Type_t**. This secondary trait class is necessary in order to promote an operation where a **float** and a **double** are involved to the one with the higher precision.

The object returned from the overloaded operator is required to be a constructed instance of the return type. In order to construct the object of type **QDPExpr** the trait class **MakeReturn** interfaces the member function **make** which just calls the constructor of **QDPExpr**. The instance of **QDPExpr** does not offer much functionality. It just stores an expression upon construction and interfaces a public member function **expression()** and a type interface **Expression_t** to access its type and instance.

```
template<class T, class C>
class QDPExpr
{
public:
  typedef T Expression_t;
  QDPExpr(const T& expr) : expr_m(expr) { }
  const Expression_t& expression() const { return expr_m; }
private:
  T expr_m;
};
```

The last class template required is the operation **OpAdd** itself. It makes use of the previous defined trait class **BinaryReturn** to be generically usable in the vector class even if **OLattice** was instantiated with different types.

```
struct OpAdd
{
  template<class T1, class T2>
  inline typename BinaryReturn<T1, T2, OpAdd >::Type_t
  operator()(const T1 &a, const T2 &b) const
  {
    return (a + b);
  }
};
```

One is safe now to use **operator()** since only constructed instances of **OpAdd** will be used.

Now the C++ compiler will create a type **QDPExpr** upon finding an algebraic expression involving instances of type **OLattice**. The class template **OLattice** does not provide an overloaded assignment operator yet. The one automatically generated by the compiler is not suitable since instances of **OLattice** are not of Plain Old Data (POD) types. A POD type is a C++ type that has an equivalent in C, and that uses the same rules as C uses for initialisation, copying, layout, and addressing. Compiler generated operators only work on the POD part of classes, which is not enough since here dynamical memory allocation is used which needs special care.

Evaluating the Expression

To evaluate the expression an **evaluate** function template is required. Instances of **evaluate** are called from the assignment operator of **OLattice** in order to provide a convenient API. The compiler should trigger evaluation upon translating, e.g.

```
  OLattice a(1000),b(1000),c(1000);
  c = a + b;
```

In order to avoid calls from every leaf type one carries out the call to **evaluate** from the base class and instead call from the derived class the member function **assign** of the base class:

```
template<class T>
class OLattice: public QDPType<T, OLattice<T> >
{
  // ...
  template<class T1,class C1>
  inline
  OLattice& operator=(const QDPExpr<T1,C1>& rhs)
    {
      return this->assign(rhs);
    }
  // ...
}
```

And one extends the base class:

```
template<class T, class C>
class QDPType
{
  // ...
  template<class T1,class C1>
  inline
  C& assign(const QDPExpr<T1,C1>& rhs)
  {
    C* me = static_cast<C*>(this);
    evaluate(*me,OpAssign(),rhs);
    return *me;
  }
  // ...
```

The assignment operator of the class **OLattice** takes an instance of **QDPExpr** and calls the **assign** member function of the base class. The **evaluate** function is called to actually execute the assignment.

```
template< class T, class T1, class Op, class RHS >
void evaluate(OLattice<T>& dest,
              const Op& op,
              const QDPExpr<RHS, OLattice<T1> >& rhs )
{
  for(int i=0; i < dest.size(); ++i)
    {
      op(dest.elem(i), forEach(rhs, EvalLeaf1(i), OpCombine()));
    }
}
```

The function template **evaluate** is parametrised with the destination type, the operator and the expression to evaluate. The function body loops through all vector elements and calls the operators **operator()** with the destination element and the return object of **forEach**. The last argument passed to **forEach** is an instance of a type that determines how to combine the expression at the tree nodes. In order to combine the children of a node according to the operator which is stored in the node the **evaluate** function passes an instance of **OpCombine**.

Calling the templated function **forEach** triggers evaluation. The type **QDPExpr** was constructed recursively by wrapping types around types. It is a **struct** which contains other **struct**s and so on, i.e. a nested **struct**. Now, one has to unwrap this struct in the same recursive manner as it was wrapped before. Unwrapping of **QDPExpr** is started by calling an instance of the function template **forEach**.

```
template<class Expr, class FTag, class CTag>
inline typename ForEach<Expr,FTag,CTag>::Type_t
```

```
forEach(const Expr &e, const FTag &f, const CTag &c)
{
  return ForEach<Expr, FTag, CTag>::apply(e, f, c);
}
```

The class template **ForEach** is parametrised with 3 typename parameters with the following meaning:

- **Expr** is of type **QDPExpr** and represents the part of the expression yet to traverse
- **FTag** is the *functor tag* which is a type that specifies which function to call at the leafs of the expression
- **CTag** is the already introduced *combine tag*

The default implementation of **ForEach**, i.e. the ones used at the leafs of the expression tree reads:

```
template<class Expr, class FTag, class CTag>
struct ForEach
{
  typedef typename LeafFunctor<Expr, FTag>::Type_t Type_t;
  inline static
  Type_t apply(const Expr &expr, const FTag &f, const CTag &)
  {
    return LeafFunctor<Expr, FTag>::apply(expr, f);
  }
};
```

The class template **ForEach** is used for two purposes:

- return type construction and
- return value evaluation

The type interface **ForEach::Type_t** constructs the return type by retrieving the return types from the children and combining them using the type interface of the trait class **Combine::Type_d**. The member function **ForEach::apply** evaluates the return value by retrieving the return values from the children and combining them using the member function of the trait class **Combine::combine**.

The class template **ForEach** is specialised for every different type occurring in the nested **struct**. As an example its specialisation is given for **BinaryOperation**:

```
template<class Op, class A, class B, class FTag, class CTag>
struct ForEach<BinaryNode<Op, A, B>, FTag, CTag >
{
  typedef typename ForEach<A, FTag, CTag>::Type_t TypeA_t;
  typedef typename ForEach<B, FTag, CTag>::Type_t TypeB_t;
  typedef typename Combine2<TypeA_t, TypeB_t, Op, CTag>::Type_t Type_t;
  inline static
  Type_t apply(const BinaryNode<Op, A, B> &expr, const FTag &f,
               const CTag &c)
  {
    return Combine2<TypeA_t, TypeB_t, Op, CTag>::
      combine(ForEach<A, FTag, CTag>::apply(expr.left(), f, c),
              ForEach<B, FTag, CTag>::apply(expr.right(), f, c),
              expr.operation(), c);
  }
};
```

The class template **ForEach** is specialised by a templated **BinaryNode**⟨**Op, A, B**⟩ which gives access to the children **A** and **B** and the operator **Op** of this type **BinaryNode**. For type construction of **ForEach::Type_t** the compiler retrieves (recursively) the type interfaces of the class template **ForEach** this time with the child **A** and **B** as the expression type. The type is then determined by the type interface of the trait class **Combine2::Type_t**.

In a similar manner the return value is recursively evaluated. Here the member function **ForEach::apply** is called passing the child types as template arguments. The return value is then evaluated by the member function of the trait class **Combine2::combine_t**.

The **evaluate** function passes an instance of **EvalLeaf1** as the *functor tag* to the **forEach** function.

```
struct EvalLeaf1
{
  int i1_m;
  inline EvalLeaf1(int i1) : i1_m(i1) { }
  inline int val1() const { return i1_m; }
};
```

The constructor of the **EvalLeaf** instance takes the element number of the vector to be returned.

The class template **LeafFunctor** is parametrised with two typename parameters. The first specifies the type of the leaf, which in the here considered example is the type **OLattice**. The second specifies the *leaf tag* or *functor tag* as introduced earlier. This class template must be specialised with every type occurring at the leafs of the expression tree.

```
template<class LeafType, class LeafTag>
struct LeafFunctor {};

template<class T, class C>
struct LeafFunctor<QDPType<T,C>, EvalLeaf1>
{
  typedef Reference<T> Type_t;
  inline static Type_t apply(const QDPType<T,C> &a, const EvalLeaf1 &f)
    {
      return Type_t(a.elem(f.val1()));
    }
};
```

Now the compiler is able to inline expressions involving instances of **OLattice** and to evaluate them with just one loop. Statements like

```
{
  OLattice<double> l0,l1,l2;
  l0 = l1 + l2;
}
```

are evaluated in an efficient way.

The here given implementation is not the best possible implementation. Some design choices had been made to demonstrate the concepts in a more pedagogical way. No safety checks on memory handling are implemented, no alignment requirements are taken into account.

Chapter 17

New Design Concepts for QDP++

17.1 Overview

This work introduces new design concepts on how to implement an active C++ library like QDP++ on accelerator type of processors like the IBM PowerXCell 8i processor. This significantly extents beyond usual code porting activities – this work leverages the Portable Expression Template Engine (PETE).

The expression template technique, on which PETE is based on, is implemented by means of meta-programming methods and as such by definition resolved during compile-time. Only after resolving the expression templates the functions are available in an assembled form that one can address for building for the accelerator core.

The evaluation of PETE expressions triggers execution of the **evaluate** function. This functions iterates over the lattice wide data type and evaluates the result object of vector or scalar type, see Sec. 16.6. The **evaluate** function is implemented as a function template which is instantiated for each PETE expression the compiler finds in the application code. The purpose of the new design concepts is to execute the **evaluate** function on the accelerator cores rather than on the general purpose core.

The most general and clean way of an implementation that addresses template instantiations (here the **evaluate** function) is to modify the compiler at this stage. A compiler modification generates C++ code fragments for each resolved **evaluate** function. Then,

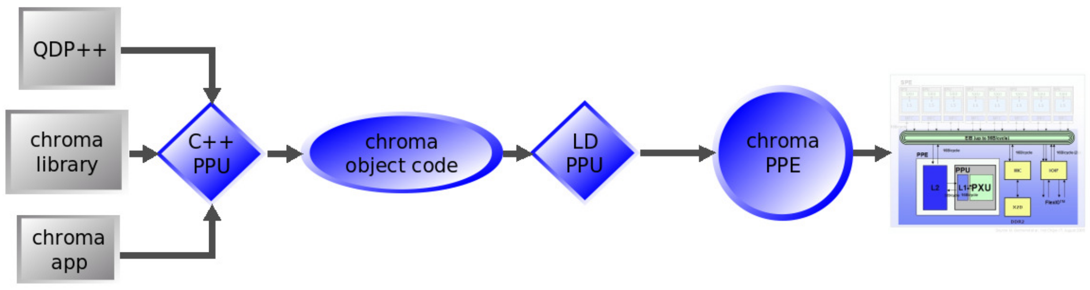


Figure 17.1 – A Chroma build for the PPE is possible with the standard build setup. The Chroma library and Chroma application are compiled on top of QDP++ using the PPU C++ compiler. The object code is then linked using the PPU linker. The resulting executable makes use only of the PPE.

the compiler for the accelerator core generates the executable functions out of these code fragments.

Changing the C++ compiler is an elaborate task and one first asks for a solution that avoids this step. Luckily, there is such a solution that does not require modifying the compiler. This solution is by no means less general, i.e. it addresses all PETE expressions, but at the same time is a lot more easier to implement.

This work proposes to write at run-time of the main application the involved PETE expressions into a database. Then a code-generator reads the database and generates program code fractions for the accelerator cores. The newly generated code fractions rely on the functionality of QDP++. In order to account for the hardware characteristics of the accelerator this work provides an optimised version of QDP++ for this processor type. In this way all involved PETE expressions are available as compiled functions, optimised for the accelerator. This work proposes to pack all such functions into a library and link the main application against it. Thus all program parts of the main application involving PETE expressions execute on the accelerators.

17.2 QDP++ on IBM PowerXCell 8i Processor

QDP++ is an active library which offers large parts of its functionality at compile-time. Chroma builds on top of QDP++ and derives from QDP++'s portability and efficiency.

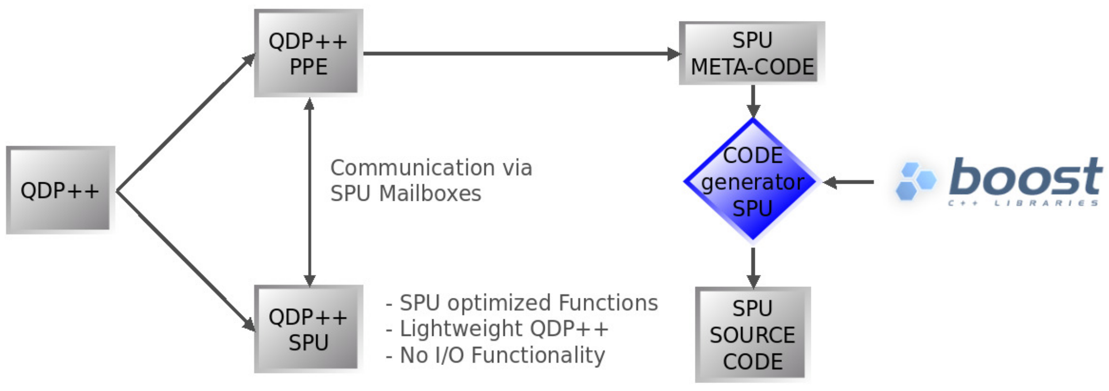


Figure 17.2 – New build process components enabling a Chroma build for the Cell processor. QDP++ for the PPE generates SPU meta-code which is interpreted by the SPU code generator which generates an SPU version of the code. QDP++ for SPU is a light-weight implementation of QDP++.

Targeting a build of Chroma for the PPE is possible, but results in an executable that runs exclusively on the PPE. It does not make any usage of the SPUs floating-point pipelines nor its DMA engines resulting in a poor performance.

On the other hand, targeting a build of Chroma for the SPU is impossible. Considering the huge code base ($\mathcal{O}(10^6)$ lines of C++ code) and taking a look at the SPU's limited local storage (LS) it becomes quickly apparent that the LS is just too small – not mentioning the SIMD organisation and the absence of input and output routines.

The only practical way is to target a Chroma build for the PPE and to extract the **evaluate** functions used in Chroma and build them separately for the SPU. In this way the PPE remains the application controller and the accelerators execute the compute-intensive parts.

Fig. 17.1 depicts the standard procedure for building Chroma, here targeting for the PPE. The Chroma application and library is compiled on top of QDP++ using the PPU C++ compiler. The object code is then linked with the PPU linker. This results in an executable which makes use of the PPE only.

17.3 Evaluate Pretty Function

The pretty-function is a C-style string variable available at run-time containing the function's name, return type and arguments. If the function is templated, the pretty-function contains the fully resolved templatised function type and arguments. It is accessed via the compiler's built-in variable **__PRETTY_FUNCTION__**.

This work modifies QDP++ in such a way that the **evaluate** function streams out its pretty-function into a database. For convenience only the program part is shown involving the **__PRETTY_FUNCTION__**:

```
template<class T, class T1, class Op, class RHS>
void evaluate(OLattice<T>& dest, const Op& op,
              const QDPExpr<RHS,OLattice<T1> >& rhs,
              const Subset& s)
{
  // ..
  some_stream << string(__PRETTY_FUNCTION__) << "\n";
  //
}
```

During execution of the main application on the PPE this results in populating a meta-code database containing a description of all **evaluate** functions called.

For example, a lattice wide function for multiplying two SU(3) colour matrices M' and M'' and storing the result in M

$$M_{i,j} = (M' \times M'')_{i,j}, \qquad \{i,1,3\},\{j,1,3\} \tag{17.1}$$

is implemented like this using QDP++

```
{
  LatticeColorMatrix c0,c1,c2;

  c0 = c1 * c2;
}
```

At compile-time the **evaluate** function gets instantiated and the pretty-function for this particular QDP++ functions reads

```
void evaluate(OLattice<T1>&, const Op&,
              const QDPExpr<RHS, OLattice<T2> >&, const Subset&)
[with T1 = PScalar<PColorMatrix<RComplex<double>, 3> >,
      T2 = PScalar<PColorMatrix<RComplex<double>, 3> >,
      Op = OpAssign,
     RHS = BinaryNode<OpMultiply,
           OLattice<PScalar<PColorMatrix<RComplex<double>, 3> > > >,
           OLattice<PScalar<PColorMatrix<RComplex<double>, 3> > > > > ]
```

As one can see the templatised function is given in a completely instantiated form.

After execution a whole bunch of such pretty-functions are stored in the database which are referred to as the SPU meta-code. In this way the set of functions to be built is limited to a minimum. The modified version of QDP++ is in the following referred to as QDP++ for PPE.

This work implements the SPU code-generator which reads the SPU meta-code and generates SPU C++ functions for each of the pretty-functions. This step is quite straight-forward and the program is not printed here.

Since the SPU's LS is very limited in size compile-time calculations are carried out in order to balance out data size versus code size, e.g. by means of adjusting the memory size available to the memory allocator and carrying out loop-unrolling.

Even the set of QDP++ functions was already limited to a minimum, the remaining set most likely still does not fit into the SPU's LS. Typically a build of Chroma involves roughly 100 different QDP++ functions. In this work it was found that the SPU's LS is capable to hold 3-6 of these functions depending on their size.

As a consequence this work makes use of code overlays, see Sec. 15.4.4. A subset of the function set are placed into overlay segments which get loaded on demand at run-time by the overlay manager. The main function which calls out to the individual functions resides in the root segment along with the memory allocator and the memory pool. The SPU code-generator produces the final SPU program with the QDP++ functions placed into these overlay regions. Then the code-generator produces a linker script that directs the SPU linker to link the final SPU accordingly. This work implemented the SPU code-generator in the Perl Programming Language [161].

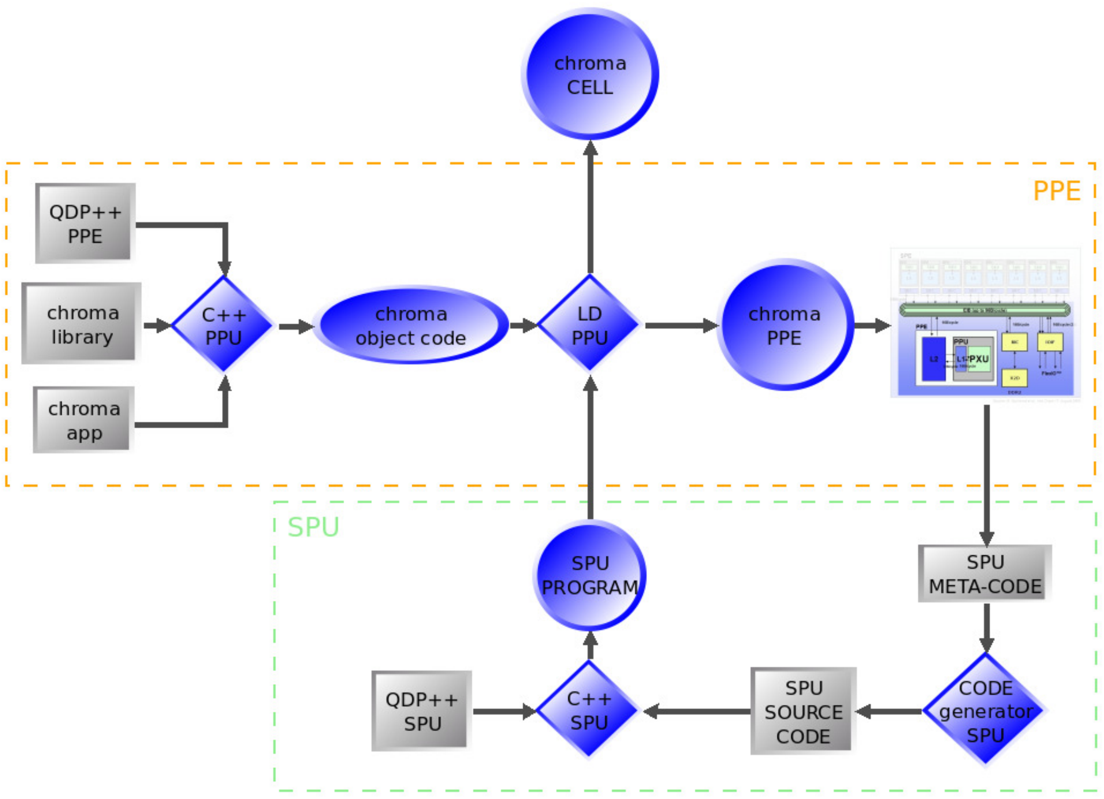


Figure 17.3 – The new build process of Chroma for the Cell processor: Following the horizontal chain of step from left to right matches the original build procedure on the PPE. The meta-code is processed by the code-generator and the SPU program is compiled on top of QDP++ for SPU. Finally the Chroma object code is linked again – this time also against the SPU program.

In order to achieve a good performance on the SPU a modified version of QDP++ for the SPU is provided. It takes advantage of the architecture of this processor with all input and output routines removed and support for SIMD organisation added. A major modification was adding support for accessing main memory via DMA transfers. The implementation details of this modified version of QDP++ are given in a later chapter. The set of SPU functions is then compiled with the SPU C++ compiler on top of QDP++ for the SPU. Finally all compiled SPU functions are bundled into the SPU library.

Fig. 17.2 depicts the overview of the modified and new components of the build process. Additionally a database is added to the SPU library that describes which QDP++ functions are available in the library.

17.4 New Chroma Build Environment

Building Chroma for the Cell processor follows a two-step compilation process.

Firstly Chroma is build following the original build procedure. The only change compared to the original build procedure is the replacement of the original QDP++ with the modified QDP++ for the PPE. This results in an application for the PPE, first without support for the accelerators. During execution the SPU meta-code is generated, describing the set of QDP++ functions used.

Secondly the SPU code-generator processes the SPU meta-code and produces SPU program fractions. Compilation of the SPU program fractions on top of the modified version of QDP++ for the SPU results in a set of SPU functions. This set of SPU functions are bundled into the SPU library. Finally the Chroma object code is linked again – this time also against the SPU library. Fig. 17.3 visualises the new build procedure.

Note that it is not necessary to recompile the Chroma library or Chroma application again. Only the linking step has to be carried out again. This saves development time since compiling Chroma is a time consuming step.

The resulting executable of Chroma is targeted for the Cell processor which executes all available QDP++ functions on the SPUs. The main control thread of the application remains on the PPE. Whenever a QDP++ function is called, the PPE queries the database of the SPU program for availability of the specific QDP++ function in the SPU program. If the function is available, the PPE indicates the SPU to execute that function and the PPE remains idle until the SPU has finished the calculation and the PPE continues the program execution.

If the function is not available as an SPU version the PPE executes the PPE implementation. Optionally further SPU meta-code for this function can be generated. In that way the missing function can be build afterwards and made available for future runs – the set of QDP++ functions that execute on the SPUs gets more and more complete.

Chapter 18

Implementation Details

18.1 Data access mode in QDP++ expressions

Typically in Lattice QCD applications the data objects get rather large. Consider for example the size of a vector that stores a quark propagator: $N^3 \times N_T \times 4^2 \times 3^2 \times$ [precision], e.g., on a lattice $48^3 \times 96$ in double precision this is roughly 12 giga-bytes. These data objects are too large to be kept in SPU's LS – even if the software was built for QPACE, i.e. after the parallelisation step. In order to avoid the physical limitation in size of the LS the data objects, i.e. the instances of our vectors class, are stored in MS.

In a QDP++ expression each data object (or leaf object) is either accessed *read-only*, *read-write*, or *write-only*. A typical expression, here taken from the Jacobi smearing routine of the Chroma code base, is

$$\psi(n) = \psi_0(n) + \kappa\, \psi_{\text{smear}} \tag{18.1}$$

the QDP++ analog implementation is

```
void smear( LatticeFermion & psi )
{
  Real              kappa;
  LatticeFermion    psi_smear, psi_0;

  psi = psi_0 + kappa * psi_smear;
}
```

where a **LatticeFermion** is a data structure for a spin-colour-vector located at every lattice site. In this work concerning the data access modes of the data objects the following was found: The three instances **psi_0**, **psi_smear** and **kappa** are accessed *read-only*. The instance **psi** is accessed *write-only*.

This work introduces the following nomenclature concerning QDP++ functions: A QDP++ functions consists of a QDP++ expression $E_{\text{QDP++}}$, an operator O, and a destination vector v_{dest}. Thus the QDP++ function can be written in the general form:

$$O(\, v_{\text{dest}} \,,\, E_{\text{QDP++}} \,). \tag{18.2}$$

If the operator O is of the form **operator=**, **operator+=**, or **operator*=** one might write:

$$v_{\text{dest}} = E_{\text{QDP++}} \tag{18.3}$$

where the assignment operator is to be replaced by the corresponding operator. In the above example the destination vector was set to $v_{\text{dest}} = \psi$, the QDP++ expression to $E_{\text{QDP++}} = \psi_0(n) + \kappa\, \psi_{\text{smear}}$, and the operator to $O =$**operator=**.

The type of the operator O determines the data access mode to the destination vector v_{dest}: The **operator=** requires **write-only** access to the destination object. The **operator+=** or **pokeSpinVector()** requires *read-write* data access. The **pokeSpinVector()** operation allows to partly modify an object of spin-vector type, i.e. to update a particular element while keeping the rest of the vector invariant. The **operator+=** requires first to read the vector, then to modify it, and lastly to store it back again.

In QDP++ operators are represented by classes. The majority of operators, e.g. **operator=**, **operator+=**, or calculation of traces of matrices, do not require additional data members in the associated class. However, the **pokeSpinVector()** operator requires an additional data member. It stores an integer which determines the spin component to keep fixed. This information is only available at run-time. Thus a concept is needed that enables operators to send information to the SPUs.

To interfere as little as possible with the existing QDP++ design a base class **BaseOp** is introduced. It is virtual and exposes access functions which associate the data access mode to the operator and provides means to send (PPU) and receive (SPU) data. Default implementations for these access functions are provided that suit the majority of operators in order to alter only a few operator definitions. The SPU implementation of the base class is given as follows:

```
struct BaseOp
{
  virtual void recvInfo()                  { }
  virtual bool getReadAccessMode() const { return false; }
  virtual     ~BaseOp()                    { }
};
```

The member function **recvInfo()** triggers on the SPU receiving of data from the counterpart operator on the PPU. The default implementation does nothing since most of the operators do not have data members. The member function **getReadAccessMode()** exposes the data access type of the operator. Since every operator needs at least write access, this function determines whether the operator requires additional **read-access**. The default implementation states no read access.

The implementation of sending and receiving the data members is only executed once per QDP++ expression and must therefore not feature the highest performance possible. SPU mailboxes were found to be an adequate solution since they transfer integer numbers between processor elements and most of the data members to transmit are integer valued.

The implementation details and an example of such operators are detailed in App. D.1.

The data access mode of the **leafs** in QDP++ expressions are considered as *read-only*. This introduces a limitation.

In C/C++ the assignment operators (also plus-assignment, etc.) return r-values. They can be used safely in further expression construction. Consider the following example:

```
{
  LatticeFermion      f0, f1, f2;

  f0 = ( f1 += f2 );
}
```

Here **f1** is altered and then **f0** is altered. The implementation described here does not support using r-values emerging from any assignment operator on the right side of any assignment operator. At this stage a compromise had to be made in order to allow the code to execute with a higher performance. This limitation leads to a significant reduction of data that needs to be transferred between MS an LS. The vectors appearing on the right hand side of the operator only have to be transferred from MS to LS and do not have to be transferred back again and thus resulting in a higher performance.

Also the introduced limitation can be overcome by sequential execution of the involved commands:

```
{
  LatticeFermion      f0, f1, f2;

  f1 += f2;
  f0 = f1;
}
```

The so far examined parts of Chroma do not make use of such constructions. Future changes to the Chroma code base might include these type of constructions and care must be taken to detect them.

18.2 QDP++ Memory Allocator

The performance of memory accesses for a given machine architecture can depend heavily on memory alignment and location. To get control over memory alignment and where memory is reserved QDP++ implements its own memory allocators. The QDP++ default memory allocator uses the operator **new** to reserve memory regions in the heap. In order to satisfy alignment constraints, for a particular request a memory region is reserved in heap that is slightly larger than the requested memory size and returns an aligned pointer within this region. This is the implementation found in the original QDP++ code.

The GNU C++ Compiler implements the operator **new** as a thin layer around the C heap allocation functions which are usually optimised for infrequent allocation of large memory blocks. This approach works well for the lattice objects which are typically large objects and are allocated infrequently.

However, on the SPU this approach is not optimal for two reasons.

First, the GNU C++ SPU Compiler implementation of the **operator new** is not in a mature state with respect to satisfying alignment requests and using the **operator delete** for releasing the reserved memory region.

Second, the LS of the SPU is very limited in size and shortages in memory are very likely to occur. For large programs the usage of code overlay techniques is necessary. Here, code

QDP++ type	T	size(T_{single})	size(T_{double})
LatticeBool	**bool**	1	1
LatticeInteger	**int**	4	4
LatticeReal	real number	4	8
LatticeComplex	complex number	8	16
LatticeColorMatrix	colour matrix	72	144
LatticePropagator	spin-colour matrix	1152	2304

Table 18.1 – Commonly used QDP++ lattice types with the primitive type T (the type of the lattice site or link) and its size in single and double precision in units of bytes.

segments in overlay regions should be as large as possible and at the same time if use is made of dynamical memory allocation enough space must be left available on the heap. With dynamical allocation the memory shortages can only be detected at run-time which makes the development process tedious. In this sense using the stack is advantageous since memory shortages can be detected at link-time. However, not all shortages can be detected by the linker since it is not able to predict precisely the stack usage.

For these reasons in this work the usage of dynamical heap allocation is avoided and instead implemented a custom stack pool-based memory allocator. A common approach to custom stack pool-based memory allocator is to allocate a large block of memory (the memory pool) in stack space, possibly at the startup of the program. The custom allocator serves individual allocation requests by returning an aligned pointer that refers to a memory address in the memory pool. Additionally it maintains a book-keeping index of memory pool usage. In this way memory alignment requirements are under control. Deallocation of memory is carried out by updating the book-keeping index.

With memory pool-based allocators allocation of memory is mainly carried out by updating an array and incrementing a counter. This is faster than using the standard implementation which uses the C heap allocation functions.

The memory pool size P is a software configuration parameter. Reasonable values vary in the range from around 20 kilo-bytes to some 64 kilo-bytes constraining accordingly the maximum size of an overlay region and with this the maximum size of a function.

18.3 SPU Parallelisation

The processing of different vector elements is typically independent from each other. As a consequence this work can take advantage by trivially parallelising the problem to several SPUs, i.e. each SPU is assigned to a different part of the data vector. Let the number of elements in the vector be N_v and the number of SPUs be N_{SPU} (the maximum number of SPUs for one IBM PowerXCell 8i Processor processor is $N_{\mathrm{SPU}} = 8$). Then the number of elements to be processed per SPU is

$$N_{sv} = \frac{N_v}{N_{\mathrm{SPU}}}. \tag{18.4}$$

The data vectors are then divided into equally sized parts

$$v = (\underbrace{v_0, v_1, \dots, v_{N_{sv}-1}}_{w_0}, \underbrace{v_{N_{sv}}, \dots, v_{2N_{sv}-1}}_{w_1}, \dots, \underbrace{v_{N_v-Nsv}, \dots, v_{N_v-2}, v_{N_v-1}}_{w_{N_{\mathrm{SPU}}-1}}) \tag{18.5}$$

where the vectors w_i are assigned to SPU number i.

18.4 Memory Transfer Latencies

The SPU uses asynchronous DMA transfers to move data between MS and the LS. This offers the possibility to hide memory latencies and transfer overhead by moving data in parallel with SPU computation. In order to achieve this at least two buffers are needed. One serving as destination or source for the DMA transfers and another one for computation, i.e. for double-buffering.

Evaluation of QDP++ expressions are especially well suited for this double buffering technique since the calculations for each lattice site are typically independent. High saturation of the memory bandwidth between MS and LS can only be achieved by issuing the DMA transfers on 128 bytes boundaries. That is, the source and destination addresses and transfer sizes are multiples of 128 bytes.

The minority of the involved lattice site types meet the requirement of being a multiple of 128 bytes in size. Tab. 18.1 lists some of the commonly used QDP++ lattice types among their primitive types and their sizes in single and double precision. As a consequence the

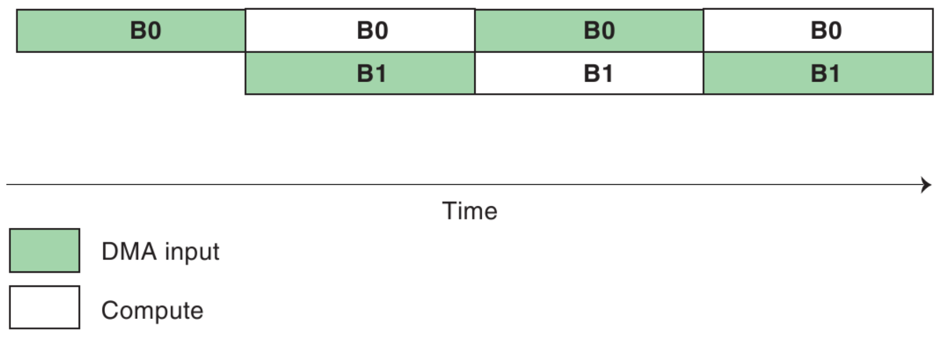


Figure 18.1 – Double-buffering: Two buffers B_0 and B_1 are used to overlap DMA transfers and SPU computation. Picture source: [160]

lattice sites of a vector v are grouped together into transfer sets such that – if LS size permits – the size of the set is a multiple of 128 bytes.

$$w = \big(\underbrace{v_0, v_1, v_2, v_3}_{\times N_s}, \underbrace{v_4, v_5, v_6, v_7}_{\times N_s}, \dots, \underbrace{v_{N_{sv}-4}, v_{N_{sv}-3}, v_{N_{sv}-2}, v_{N_{sv}-1}}_{\times N_s}\big) \tag{18.6}$$

If the LS size is not sufficient these sets are chosen such that at least the sets meet the alignment requirements for DMA transfers, i.e. being a multiple of 16 bytes in size.

Once the set is transferred into LS via a DMA transfer it is being processed. If the transfer set is larger than 16 kilo-byte (which is the largest possible DMA transfer supported by the Cell B.E.) then a DMA list transfer is issued.

In order to transfer the data from MS to LS, for each data object in the QDP++ expression E_{QDP++} double-buffering is used. The destination vector v_0 is accessed either *write-only* or *read-write* and the data is transferred either using a double-buffering for write-only access or shared input-output buffering for read-write access.

18.4.1 Double-Buffering

Two buffers B_0 and B_1 equal in size are required for double-buffering. These are allocated using the custom memory allocator at start-up of the function. Additionally two tag-group identifiers T_0 and T_1 are needed. Tag-group T_0 is applied to all transfers involving B_0 and tag-group T_1 is applied to all transfers involving B_1.

```
allocate $B_0$
allocate $B_1$
$i \leftarrow 0$
$i' \leftarrow 1$
initiate DMA transfer $B_1$
**for all** vector sites **do**
   **if** not last vector site **then**
      initiate DMA transfer $B_i$
   **end if**
   wait DMA transfer $B_{i'}$
   calculate $B_{i'}$
   swap($i$,$i'$)
**end for**
```

Algorithm 2 – Double-buffering: Computation and DMA transfers execute in parallel.

Alg. 2 and detail the double-buffering technique. The program allocates the buffers and starts the first DMA transfer, then enters the loop. The loop starts the next DMA transfer and waits for the first one to complete. When the first DMA transfer has completed, the code executes the calculation function. The program then toggles the buffer index and loops again to start the next DMA transfer. The process repeats until all the vector sites have been transferred and processed, see Fig. 18.1.

18.4.2 Shared Input-Output Buffering

If processing requires both transferring of the data from MS to LS and transferring it back after computation, then shared input-output buffering is used. As for double-buffering, again two buffers B_0 and B_1 of equal size are required. And again two tag-group identifiers T_0 and T_1 are needed. Tag-group T_0 is applied to all transfers involving B_0 and tag-group T_1 is applied to all transfers involving B_1.

In contrast to double buffering an ordering dependency is introduced when sharing buffers for both input and output. Previous outbound transfers need to complete before subsequent incoming transfers can be initiated on the same buffer. In order to ensure this, the inbound

```
allocate $B_0$
allocate $B_1$
$i \leftarrow 0$
$i' \leftarrow 1$
initiate inbound DMA transfer $B_1$
**for all** vector sites **do**
  **if** not first vector site **then**
    initiate outbound DMA transfer $B_i$
  **end if**
  **if** not last vector site **then**
    initiate fenced inbound DMA transfer $B_i$
  **end if**
  wait DMA transfer $B_{i'}$
  calculate $B_{i'}$
  swap($i$,$i'$)
**end for**
initiate outbound DMA transfer $B_i$
wait DMA transfer $B_i$
```

Algorithm 3 – Shared Input-Output-Buffering. Includes both transferring of the data from MS to LS and back.

DMA transfers will be issued with the *fence* option. The fence attribute causes DMA commands to be locally ordered with respect to all previously issued commands within the same tag-group.

Alg. 3 details the shared input-output-buffering technique. First, the buffers are allocated using the custom allocator and the first inbound DMA transfer is issued. When entering the loop for the first time the second inbound DMA transfer is initiated. During the first loop iteration the fence option has no effect. The program then waits for the previous transfer to complete and calculation is started. The indices are swapped and program execution enters the loop again. From the second iteration on, the output DMA transfer is initiated for every loop iteration. To ensure local ordering of the subsequent inbound transfer in respect to the previously issued outbound transfer the inbound transfer is issued with the fence option. The program then waits again on the previous inbound transfer to complete and calculation is started. This processing repeats until all the vector sites have been transferred and processed. A last outbound transfer is initiated to store the last transfer set to MS.

18.5 Data Alignment and Transfer Sizes

Next, the size of the transfer set is determined. The size of the transfer set should be chosen reasonably taking into account the available memory space and the sizes of the involved primitive types.

For example, an expression involving a primitive data type of 1 byte in size (like **bool**) should be ideally grouped together with 128 elements. A data type that occupies just one byte per lattice site is, e.g. the **LatticeBool**, which stores one boolean variable at each lattice point. In general, a data type that is preceded with **Lattice** provides storage for the corresponding data type for each lattice point.

Consider the following example:

```
  {
2   LatticeBool        b;
    LatticeInteger     i0,i1,i2;
4   LatticeColorMatrix c0,c1,c2;

6   i0 = b ? i1 : i2;
    c0 = b ? c1 : c2;
8 }
```

Line 6 shows an example which typically makes no problems: All involved objects are roughly of the same small size. The size of a primitive of a **LatticeInteger** (**LatticeBool**) is 4 (1) byte. It is possible to group $N_s = 128$ elements together so the DMA transfers execute with a high performance.

Line 7 shows an example which is a bit more tricky: A small primitive type (**LatticeBool**) is accompanied by larger primitive types, i.e. **LatticeColorMatrix**. The size of the primitive type of a **LatticeColorMatrix** is 144 bytes in double precision, see Tab. 18.1. Since double buffers are needed for each object it depends on the poolsize P whether a group of

$N_s = 128$ elements is possible or not. Here, one would require $N_s = 128$ in order to get multiples of 128 bytes for the **bool** type. Since N_s is the same for all objects involved, it remains to be checked whether this choice still suits the memory requirements when taking into account all involved objects. Since the **ColorMatrix** appears 3 times (**c0**, **c1**, and **c2**), for $N_s = 128$ one requires roughly $3 \times 144 \times 2 \times N_s = 110$ kilo-bytes. It depends on the poolsize P whether this choice of N_s is suitable. If the poolsize is not sufficient, then one might reduce the number of elements in the group to $N_s = 16$.

The best choice of the number of elements N_s in the transfer set depends on

- the poolsize P
- the sizes of the primitive type involved in the QDP++ expression and the destination vector.

The memory pool is required to hold double-buffers for the transfer set assigned to the result vector and additional double-buffers for each of the constituents occurring in the QDP++ expression.

These buffers B_i^n were introduced previously, see Sec. 18.4.1 but now carry a new index n which indicates the number of the vector in MS.

Alg. 4 shows pseudo code of a general QDP++ function call on the PPE. First, the involved data objects are defined. Then an expression is constructed with storing the data objects at the expression leafs. Then the call to the operator is issued.

The vectors v_n reside in MS, where v_0 is the destination vector. Evaluation of site $v_0(i)$ requires access to sites $v_n(i)\ \forall\, 1 \leq n \leq N$. The vectors v_n are divided into equally sized, non-overlapping, continuous subsets, i.e. the transfer sets containing N_S elements, see Eq. (18.6).

Alg. 5 determines the number of elements N_S. This algorithm introduces a new parameter $B_{\text{min}}^{\text{DMA}}$ which specifies the (desired) minimum DMA transfer size. The **for**-loop and the nested **while**-loop determine the number of elements N_S taking into account the parameter $B_{\text{min}}^{\text{DMA}}$. At the same time it accounts for the total size of all double-buffers that need to be allocated. Then the algorithm verifies whether all buffers fit in the memory pool. If the

```
$L(T_0)\, v_0$
$L(T_1)\, v_1$
$L(T_2)\, v_2$
$\vdots$
$L(T_N)\, v_N$
$O\ op$
$op(v_0, E_{\mathrm{QDP++}}(v_1, v_2, \dots, v_N))$
```

Algorithm 4 – General form of a QDP++ function call. The operator O is carried out for the expression $E_{\mathrm{QDP++}}$ containing N leafs ($v_1 \dots v_N$), where v_0 is the destination vector. The instances v_i are of lattice type $L(T_i)$ with the primitive type T_i.

memory pool is not sufficiently large the value of N_S is adjusted accordingly, leading to a smaller DMA transfer size than originally requested.

The poolsize P is known before compile-time. But the sizes of the primitive types are known only at compile-time. Execution of Alg. 5 is deferred until compile-time. It is implemented using the Boost Meta-Programming Library (MPL) [162].

The number of elements N_s determines the size of the buffers B_i^n

$$\mathrm{size}(B_i^n) = N_s \times \mathrm{size}(T_n)\,, \qquad i \in \{0, 1\}. \tag{18.7}$$

The following shows an example:

```
{
  LatticePropagator   g0,g1,g2;
  LatticeColorMatrix  u;

  g0 -= g1 + u * g2;
}
```

The size of an instance of the primitive data type of the vector type **LatticePropagator** is 2304 bytes in double precision, see Tab. 18.1. Parameter settings of $B_{\mathrm{min}}^{\mathrm{DMA}} = 2048$ bytes and $P = 132$ kilo-bytes results in $N_S = 8$. In this example the poolsize P is sufficiently large to hold buffers that are all a multiple of 128 bytes in size.

$B_{\mathrm{min}}^{\mathrm{DMA}}$ is a software configuration parameter. It is a candidate for self-tuning approaches like in ATLAS [163] or FFTW3 [164].

```
$N_S \leftarrow 0$
$t \leftarrow 0$
**for** $n = 0$ to $N$ **do**
  $t \leftarrow t + 2 \times \text{size}(T_n)$
  $c \leftarrow 1$
  **while** $c \times \text{size}(T_n) \bmod \text{DMA}_{\text{min}} \neq 0$ **do**
    $c \leftarrow c \times 2$
  **end while**
  $N_S \leftarrow \max(N_S, c)$
**end for**
**while** $t \times N_S > P$ **do**
  $N_S \leftarrow N_S/2$
**end while**
```

Algorithm 5 – Determining the number of elements N_S of the transfer set. Since size(T_n) is only known at compile-time this algorithm is executed at compile-time using Boost MPL.

18.6 SPU Code Overlays

Applications like Chroma make use of hundreds of different QDP++ functions each of which results in a different SPU function. The sum of the code sizes of all SPU functions plus data most likely exceed the LS size.

The physical limitation on code size for the SPU can be overcome by using code overlays included in the IBM Cell development tools.

The SPU program executes a service loops that waits for commands from the PPE. The arriving command from the PPE indicates a particular SPU function to be executed and the service loop branches to that particular function. After the function has completed SPU execution continues in the service loop waiting for the next command.

Programs with such call graph structures are well suited for implementation with code overlays. The service loop remains in the root segment, while the worker functions remain in code segments placed in overlay regions.

Listing 18.1 – Linker script example for the auto overlay manager. Linker symbols for the same functions (pr1027.o) are placed into different overlay segments.

```
SECTIONS
{
 OVERLAY :
 {
  .ovly2 {
  {
   .....
   :pr1026.o (.text._Z13function_1026v)
   :pr1026.o (.text._ZN3QDP8evaluateILi64ENS_11PSpinVectorINS_12PColor
              VectorINS_8RComplexIdEELi3EEELi2EEES6_NS_8OpAssignENS_9
              UnaryNodeINS_10OpIdentityENS_9ReferenceINS_7QDPTypeIS6_NS_
              8OLatticeIS6_EEEEEEEEEvRNSC_IT0_EERKT2_RKNS_7QDPExprIT3_
              NSC_IT1_EEEERKNS_6SubsetE)
   :pr1027.o (.text._Z13function_1027v)
  }
  .ovly3 {
   :pr1027.o (.text._ZN3QDP8evaluateILi128ENS_11PSpinVectorINS_12PColor
              VectorINS_8RComplexIdEELi3EEELi2EEES6_NS_8OpAssignENS_10
              BinaryNodeINS_13OpAdjMultiplyENS_9UnaryNodeINS_10OpIdenti
              tyENS_9ReferenceINS_7QDPTypeINS_7PScalarINS_12PColorMatrix
              IS4_Li3EEEEENS_8OLatticeISH_EEEEEEEENSA_INS_21FnSpinProjec
              tDir0PlusENSC_INSD_INS1_IS5_Li4EEENSI_ISO_EEEEEEEEEEEEvRNS
              I_IT0_EERKT2_RKNS_7QDPExprIT3_NSI_IT1_EEEERKNS_6SubsetE)
   :pr1028.o (.text._Z13function_1028v)
   :pr1028.o (.text._ZN3QDP8evaluateILi64ENS_11PSpinVectorINS_12PColor
              VectorINS_8RComplexIdEELi3EEELi2EEES6_NS_8OpAssignENS_9
              UnaryNodeINS_22FnSpinProjectDir1MinusENS_9ReferenceINS_7
              QDPTypeINS1_IS5_Li4EEENS_8OLatticeISC_EEEEEEEEEvRNSD_IT0_
              EERKT2_RKNS_7QDPExprIT3_NSD_IT1_EEEERKNS_6SubsetE)
   .....
  }
 }
}
INSERT AFTER .text;
```

Listing 18.2 – Post processed linker script. Linker symbols for function pr1027.o are placed into the same (newly created) overlay segment.

```
.ovly2 {
{
 .....
 :pr1026.o (.text._Z13function_1026v)
 :pr1026.o (.text._ZN3QDP8evaluateILi64ENS_11PSpinVectorINS_12
            PColorVectorINS_8RComplexIdEELi3EEELi2EEES6_NS_8
            OpAssignENS_9UnaryNodeINS_10OpIdentityENS_9ReferenceINS_7
            QDPTypeIS6_NS_8OLatticeIS6_EEEEEEEEEvRNSC_IT0_EERKT2_
            RKNS_7QDPExprIT3_NSC_IT1_EEEERKNS_6SubsetE)
}
.ovly3 {
 :pr1028.o (.text._Z13function_1028v)
 :pr1028.o (.text._ZN3QDP8evaluateILi64ENS_11PSpinVectorINS_12
            PColorVectorINS_8RComplexIdEELi3EEELi2EEES6_NS_8
            OpAssignENS_9UnaryNodeINS_22FnSpinProjectDir1MinusENS_9
            ReferenceINS_7QDPTypeINS1_IS5_Li4EEENS_8OLatticeISC_
            EEEEEEEEEvRNSD_IT0_EERKT2_RKNS_7QDPExprIT3_NSD_IT1_
            EEEERKNS_6SubsetE)
 .....
}
.....
.ovly15 {
 :pr1027.o (.text._Z13function_1027v)
 :pr1027.o (.text._ZN3QDP8evaluateILi128ENS_11PSpinVectorINS_12
            PColorVectorINS_8RComplexIdEELi3EEELi2EEES6_NS_8
            OpAssignENS_10BinaryNodeINS_13OpAdjMultiplyENS_9
            UnaryNodeINS_10OpIdentityENS_9ReferenceINS_7QDPType
            INS_7PScalarINS_12PColorMatrixIS4_Li3EEEEENS_8
            OLatticeISH_EEEEEEEEENSA_INS_21FnSpinProjectDir0Plus
            ENSC_INSD_INS1_IS5_Li4EEENSI_ISO_EEEEEEEEEEEEvRNSI_
            IT0_EERKT2_RKNS_7QDPExprIT3_NSI_IT1_EEEERKNS_6SubsetE)
}
```

Since memory allocation occurs in each of the worker functions, the memory allocator and the memory pool remain in the root segment.

Fig. 15.3 depicts an overview of the SPU program call graph and overlay structure.

The amount of functions that can be placed into the same overlay segment is mainly determined by the memory poolsize. To take advantage of automatic grouping of several worker functions into the same overlay segment the auto overlay manager, see Sec. 15.4.4, is used. This gives a first, coarse placement of the individual functions.

Some larger functions are split by the compiler into several smaller ones. The auto overlay manager might not be able to place these functions into the same overlay segment, see an original linker script in Lst. 18.1. Using this script results in a not optimal performance since overlay segments have to be switched during function execution.

A post processing tool was developed that alters these linker scripts. The tool examines the script for linker symbols that are split across multiple overlay segments. If such segment crossing symbols are found they are moved into a newly created overlay segment, see the post processed linker script in Lst. 18.2.

After post-processing the linker is invoked again. This ensures that all linker symbols of one SPU function remain in the same overlay segment.

18.7 Single Instruction Multiple Data

The SPU features SIMD floating point pipelines. The peak performance is reached issuing a fused multiply-add instruction per machine cycle. This results in 4 floating point operations in double precision per SPU.

Computations in a real-world application can hardly be broken down into instructions made up exclusively of multiply-add operations. The peak performance will rarely be reached and serves more as a reference point for benchmark measurements.

The sustained performance is defined as the ratio of the measured performance over the peak performance.

However, application programs should saturate the floating point pipelines as much as possible to obtain a high sustained performance.

In order to achieve this, the program data primitives must be broken down into smaller data fractions in such a way that the basic operations can be performed taking advantage of the SIMD floating point pipelines.

QDP++ data types are constructed in a nested manner. It therefore offers the possibility to manually override the generic construction of types at any level of type construction by means of class template specialisations.

A convenient choice is to specialise at the level of complex numbers. In double precision these fit exactly into one SPU processor register.

```
template<>
class RComplex<REAL64>
{
public:
  RComplex() {}
  ~RComplex() {}

private:
  vector double F;
};
```

The explicit class template specialisation defines the QDP++ type for a complex number in double precision on the SPU. Making this specialisation explicit gives it priority over all other template definitions the compiler can find for this class.

To access the real and imaginary part of the complex numbers the class provides the member access functions:

```
T& RComplex<T>::real();
T& RComplex<T>::imag();
```

Arithmetic operations (e.g. complex multiplication, etc.) make use of these member functions. In this way, the generic implementations of the arithmetic operations result in many bit shuffle operations, since this allows scalar usage of vector types.

To avoid unnecessary bit shuffle operations basic operations on complex numbers are specialised. App. D.2 details an example of the implementation of arithmetic operations with complex numbers.

18.8 Loop-Unrolling

Our implementation foresees unrolling of many loops of QDP++. Loop-unrolling can lead to a significant performance improvement in SPU programs. Since then computational and controlling instructions occur in larger sequence between branch instructions the compiler has more freedom to reorder instructions, e.g. in order to achieve a good dual-issue rate or hiding floating point pipeline latencies.

18.8.1 Operations on Primitive Types

The generic implementation of operations on QDP++ primitive types, like matrix-vector multiplication or matrix-matrix addition involve loops over vector lengths or matrix dimensions. The loop count is known at compile-time.

Generic operations on primitive types can be divided into two disjoint sets:

1. Operations containing loops that the compiler can possibly unroll
2. Operations containing loops that the compiler is not able to unroll under no circumstances

For example, the generic implementation of **operator+** for two matrices belongs to the first set.

```
template<typename T,int N>
Matrix<T,N>
inline operator+ ( Matrix<T,N>& l , Matrix<T,N>& r )
{
  typename Matrix<T,N> ret;

  for(int i=0; i < N; ++i)
    for(int j=0; j < N; ++j)
      ret.elem(i,j) = l.elem(i,j) + r.elem(i,j);

  return d;
};
```

The loop count is known at compile-time, and every loop iteration is independent from each other. The loop-unrolling facility of the compiler is able to unroll the loops under the assumption that instruction inlining constraints would still be met.

An example for an operation belonging to the second set is the **operator*** for two matrices.

```
template<typename T, int N>
Matrix<T,N>
inline operator* ( Matrix<T,N>& l , Vector<T,N>& r )
{
  typename Matrix<T,N> ret;

  for(int i=0; i < N; ++i)
  {
    ret.elem(i) = l.elem(i,0) * r.elem(0);
    for(int j=1; j < N; ++j)
      ret.elem(i) += l.elem(i,j) * r.elem(j);
  }

  return d;
};
```

Even though the loop count is known at compile-time, the compiler is not able to unroll this loop. The loop iterations are not independent from each other.

The compiler does not know that repeated application of the **operator+=** equals to one application of **operator=** with an expression of a sequence of **operator+** on the right hand side. There is no syntax element in C++ that assigns semantic properties to operators.

18.8.2 The Evaluation Loop

Evaluation of the destination vector involves a loop over the elements of the transfer set, i.e. the evaluation loop:

```
for $i = 0$ to $N_S - 1$ do
    $op(v_0(i), f(v_1(i), v_2(i), \dots, v_N(i)))$
end for
```

If the body of the evaluation loop is short unrolling the evaluation loop can result in performance gains. The loop count N_s is determined at compile-time making it in principle possible to unroll the loop. However, unrolling this loops entirely quickly results in very large code. Due to inlining constraints the compiler often decides to not unroll this loop. One can either

```
**Require:** $N_S \bmod N_L = 0$
  $d \leftarrow 0$
  **while** $d < N_S$ **do**
    **for** $i = 0$ to $N_L - 1$ **do**
      $op(v_0(i+d), f(v_1(i+d), v_2(i+d), \dots, v_N(i+d)))$
    **end for**
    $d \leftarrow d + N_L$
  **end while**
```

Algorithm 6 – Forced Loop-Unrolling of the evaluation loop, partly implemented with Boost MPL. The **while**-loop is a run-time loop, and the **for**-loop is a compile-time loop. d represents the number of already processed vector elements.

- use the compiler's loop-unroll option and trust the compiler's heuristics to advantageously unroll the loop, or
- force the loop-unrolling by meta-programming techniques, or
- determine optimal settings using autotuning techniques.

To carry out a benchmark analysis in a systematic way so that the impact of unrolling this loop can be studied in detail, control must be taken of unrolling this loop.

To force the compiler to unroll the evaluation loop the Boost Meta-Programming Library is used.

Alg. 6 shows the loop-unrolling as implemented by us. In order to get fine-control over the loop-unrolling process a new parameter N_L is introduced. The parameter N_L specifies the number of loop iterations to unroll.

The **while**-loop is a run-time loop and the **for**-loop is a compile-time loop. Book-keeping of the already processed elements in the internal variable d was introduced to avoid 32 bit integer multiplication in the SPU and to replace it with repeated summation.

18.8.3 Primitive Type Assignment Operators

The assignment operators in the class template definitions for the QDP++ primitive types are not defined by default. This is safe since the primitive types are POD types. The compiler generated assignment operators work properly for POD types. Even though the check for self-assignment could be saved for performance issues.

The C++ compiler decides on basis of the POD size of a class either to

- generate a copy loop, or
- issue a call-out to the **memcpy** function of the Standard C Library

to copy the POD portion of the class to assign.

The default threshold for the GNU C++ Compiler is 8 kilo-bytes. The compiler generated assignment operator calls **memcpy** if the POD part of the class is equal or larger than this threshold.

When using code overlays and working with compiler generated assignment operators one must take care of placing the instance of **memcpy** in the root overlay region. Also one has to take into account that every branch involves an additional function stub to be called in the overlay manager resulting in a call overhead. Lastly branches in the program flow introduce at least one not correctly predicted branch resulting in a massive performance drop.

In summary it is best to avoid compiler generated assignment operators and to implement them explicitly. Refer to App. D.3 which details on our implementation.

18.9 Inlining the Operation Functions

By declaring a function inline, the C++ compiler is directed to make calls to that function faster. One way the compiler can achieve this is to integrate that function's code into the code of its callers. This makes execution faster by eliminating the function-call overhead. In addition, if any of the actual argument values are constant, their known values may permit simplifications at compile time such that not all of the inline function's code needs to be included. The effect on code size is less predictable. Object code may be larger or smaller with function inlining, depending on the sizes of the functions to inline and from where and how many times they are called.

The compiler's tree inliner is controlled by the inline option arguments given at compiler invocation time.

In QDP++ most of the operations are implemented as inline functions. If all inlining constraints are met the compiler does not create a separate function, but inserts the function's body into the caller's body, i.e. inlines the function.

When evaluating QDP++ expressions several operations are involved in a nested manner. For example a matrix times vector operation includes multiple multiplications and additions of complex numbers. Multiplication of complex numbers in turn involve multiplications and additions of real numbers. This process quickly results in a large number of instructions.

On most machine architectures, so on the SPU, it is advantageous to inline all those operations into the caller's functions body. But the default values of the tree inliner parameters are too restrictive to allow this. The compiler fails to inline the majority of the functions which were defined inline.

App. E.1 lists the GNU C++ SPU Compiler inline options used to inline most of the functions into the evaluation loop.

Some function calls can not be inlined. These are for example the multiplication of a spin matrix with gamma matrices. The number stating which gamma matrix to use for the multiplication is only known at run-time. In QDP++ the multiplication with gamma matrices is implemented with function pointers which can under no circumstances be inlined. Each multiplication with a specific set of gamma matrices remains as a separate function.

When working with code overlays one has to place functions which carry out multiplications with gamma matrices into the same overlay region as the function calling out to them.

Chapter 19

Benchmark Results

19.1 Overview

A set of QDP++ functions were used for benchmark measurements. Calls to these functions emerged during the execution of Chroma, especially during

- propagator calculation, i.e. inversion with the conjugate gradient method,
- calculation of the hadronic spectrum, and
- source and sink smearing routines.

These calculations require roughly 60 different QDP++ functions f_n to be called. Refer to Tab. F.1 which shows a list of the QDP++ functions. The explicit mathematical form of the individual functions can not easily be recovered. The only source of information is the meta-code of the functions. In the meta-code the individual functions are defined in a recursive templated manner which is not easy to read for humans. However, it is possible to extract the original form by hand which was carried out for individual functions as shown below to study their behaviour. Extracting the mathematical form in an automated manner is not implemented.

The execution time of a function includes the time needed for transferring the data and for computing. Since the mathematical form of the function is not accessible in a systematic manner in this work the number of floating-point operations necessary for a QDP++

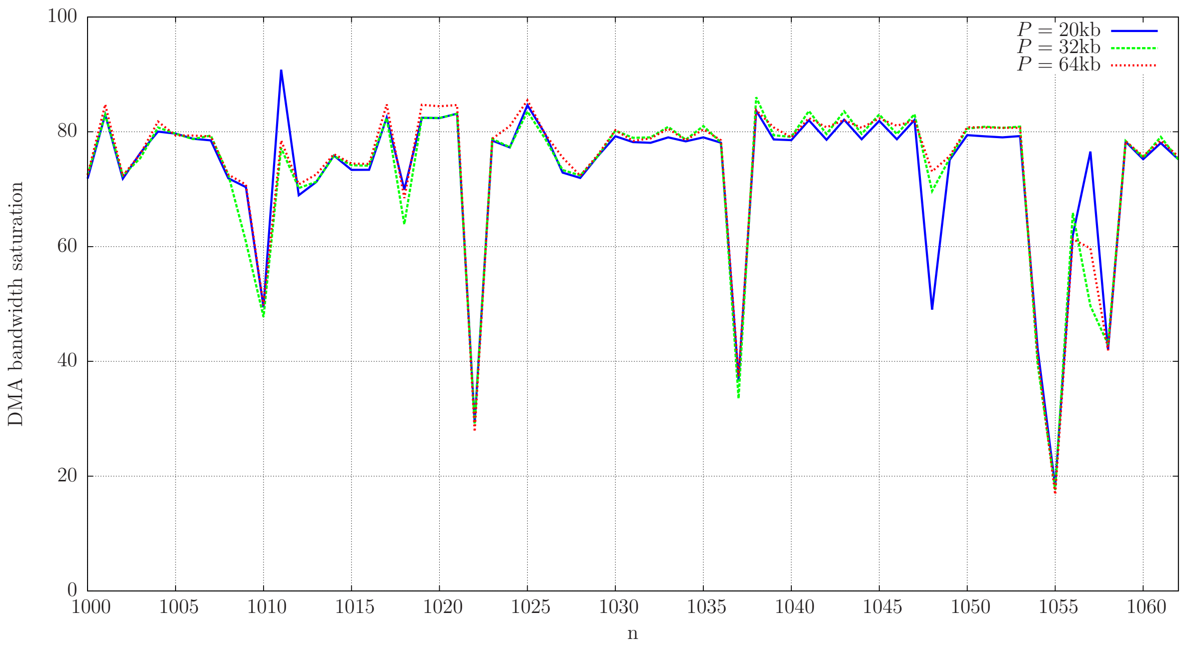


Figure 19.1 – Memory bandwidth saturation for all investigated QDP++ functions f_n. Different benchmark measurements are shown for different values of the poolsize P. Only DMA transfers are carried out, SPU computation is switched off in this benchmark test.

function is not determined. As a consequence the time required to transfer the data is taken as a point of reference for benchmark measurements.

In order to carry out the benchmarks, for each function f_n the amount of data $B(n)$ to be transferred was determined. This refers to the total amount of data and includes data that must be transferred from MS to LS and vice versa. Furthermore the execution time $t_{\text{exe}}(n)$ for each function was measured. The sustained bandwidth saturation

$$S_{\text{DMA}}(n) = \frac{B(n)}{t_{\text{exe}}(n)} \beta_{\text{peak}}^{-1} \tag{19.1}$$

was determined for each function where β_{peak} is the maximum bandwidth achievable with the Cell Broadband Engine and is limited to $\lambda_{\text{peak}} = 8$ bytes per cycle.

The best overall performance of the software was found for QDP++ specialisations turned on and with adjusted compiler options concerning optimisations. In order to be able to study the impact of both features, the benchmark measurements were performed by either turning off QDP++ specialisations or by applying the default compiler options. This work always compares the achieved performance to the maximum performance achieved by both features applied.

The benchmark measurements are organised as follows: To be able to study to what extend SPU computation is hidden by DMA transfers, the first benchmark concentrates on processing with computation switched off. Next the impact of the QDP++ specialisations on the performance is studied. Then the effect of using compiler optimisation facilities is examined. Last benchmark measurement focuses on unrolling the evaluation loop.

All benchmark measurements are carried out on a QS22 Cell Blade at Jülich Supercomputing Centre (JSC). One blade comprises 2 IBM PowerXCell 8i Processors. The Non-Uniform Memory Access (NUMA) memory model is used throughout. With NUMA controlled execution, a Linux process can be tied to a specific processor. Addressable memory to the process can be assigned to a specific physical memory domain. That is, the processor can access its local memory rather than accessing, memory local to another processor or memory shared between processors. The benchmark measurements were carried out using 1 IBM PowerXCell 8i Processor with 8 SPUs.

19.2 DMA Transfers

The first benchmark measurements are carried out with computation switched off. Included is only transferring the read-only data from MS to LS and storing back the result vector.

The measurement is carried out for different sizes P of the memory pool. Here the values 20, 32, and 64 kilo-bytes are chosen. The stack based memory pool resides in the root overlay region, which in turn limits the maximum size of the individual functions. A value larger than 64 kilo-bytes was not possible, since the size of the largest function plus the memory pool plus code in the root segment would then exceed the LS size.

Fig. 19.1 depicts the benchmark results for this measurements. Most of the functions show a good saturation of around 80% of the DMA bandwidth. Variation of the memory poolsize does not have a large impact on the overall DMA bandwidth saturation. This is due to the fact that mostly the DMA memory alignment constraints for obtaining good DMA bandwidth saturation are met by grouping together several lattice sites for processing.

Some isolated functions show a sudden drop in the memory bandwidth saturation. Like for example for $n = 1022$ or $n = 1037$ and $n = 1054, 1055$ DMA bandwidth saturation of

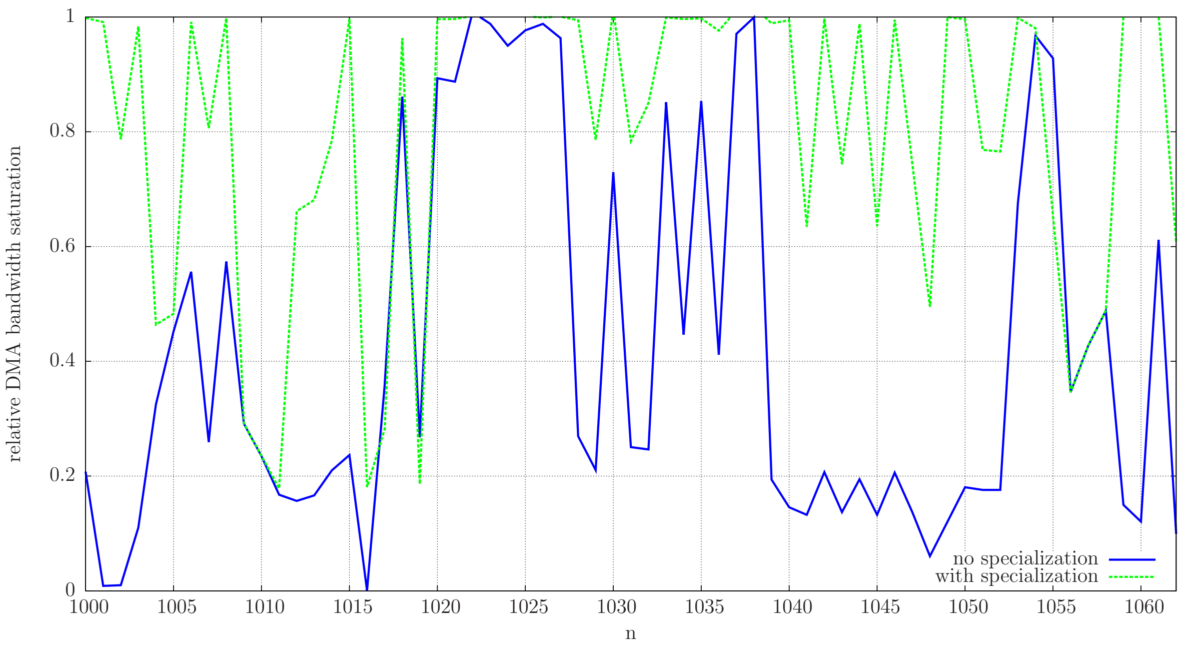


Figure 19.2 – Impact of QDP++ specialisations in comparison to generic implementations.

less than 40% is measured. For these functions only a few bytes per lattice site have to be transferred.

For example, function $n = 1055$ implements

```
{
  LatticeBool     b;
  LatticeInteger  i;
  int             i0, i1;

  b = i & i0 > i1;
}
```

and requires 5 bytes to be transferred. This is 1 byte for the destination site, and 4 bytes for the right hand side of the expression. DMA bandwidth saturation would be reached when only spending 5 cycles at one lattice site. Taking into account the management overhead for issuing DMA transfer commands, traversing (possibly recursively) the expression tree and switching DMA buffer indices, the total amount of clock cycles is larger than the minimum number required taking only into account the number of bytes to be transferred.

This overhead is present for all functions but significantly carries weight only for those functions with a small amount of bytes to be transferred.

n	QDP++ function	index range
1006	$M_{i,j} = M'_{i,j}$	$\{i,1,3\}\{j,1,3\}$
1007	$M_{i,j} = (M' \times M'')_{i,j}$	$\{i,1,3\}\{j,1,3\}$

Table 19.1 – Example of two QDP++ functions.

19.3 Specialisations

The next benchmark measurement switches on SPU computation. The whole processing is carried out, DMA transfers and computation. This benchmark focuses on the impact of QDP++ specialisations.

Fig. 19.2 depicts the benchmark results for this measurements. The curves shown represent the relative bandwidth saturation

$$S^{\text{rel}}_{\text{DMA}}(n) = \frac{S_{\text{DMA}}(n)}{S^{\text{no comp}}_{\text{DMA}}(n)} \tag{19.2}$$

with $S^{\text{no comp}}_{\text{DMA}}(n)$ from the previous benchmark measurement with computation switched off, i.e. for measurement with poolsize $P = 64$ kilo-bytes. Significant performance improvements are observed when including the specialisations of the QDP++ primitive types and operations.

Tab. 19.1 shows two of the measured functions which are discussed in more detail in the following.

Discussion of function f_{1006}: This function assigns a vector of colour matrices to another vector of colour matrices, see Tab. 19.1. SPU computation is perfectly hidden by DMA transfers when QDP++ specialisations are included – DMA saturation is as large as with computation switched off. When QDP++ specialisations is not included the generic assignment routine gets called. This routine involves a nested loop over the matrix dimensions and gets not unrolled by the compiler due to inlining constraints. This results in a performance drop to around 40%.

An SPU Timing analysis was carried out for function f_{1006}. The output of the SPU Timing Tool for the program part relevant for the assignment of the colour matrix is given in App. G.1.1 with template specialisation and in App. G.1.2 without specialisation. Whereas the assignment in the former case is carried out within less than 20 machine cycles the program part generated from the generic C++ code requires more than 400 machine cycles to do the same job. The reason for the inefficient code the compiler produces out of the generic code is that when it assigns element-wise the matrix it has to resolve the next template level, i.e. the complex numbers. The real and imaginary parts get assigned separately, i.e. sub quad-words get assigned. This results in many load, shuffle, and store operations. On the other hand the program version with template specialisation takes advantage out of assigning a whole complex number with just one load and store operation.

Function $n = 1007$ represents the multiplication of two colour matrices and assigning the result to the destination vector. SPU computation is not completely hidden by DMA transfers – even with included QDP++ specialisations. The nested loops involved in colour matrix multiplication were unrolled in the specialisations. Optimisations beyond the level of loop unrolling are not carried out. This results in a better performance (around 65%) than the generic operation results in (around 20%).

The SPU Timing analysis for function f_{1007} is given in App. G.2.1 with template specialisation and in App. G.2.2 without specialisation. The former case does not contain any loop and executes in roughly 300 machine cycles. The necessary 27 complex multiplications are executed sequentially where each time the optimised function for complex multiplication was inlined.

The generic C++ code involves 3 nested loops to carry out the matrix multiplication. As seen in the assembler code the compiler was able to unroll the inner-most loop leaving the 2 out-most loops unrolled. The inner-most loop body contains 3 complex multiplications and 2 complex additions. For better visibility the rotate operations are emphasised, most of which result from accessing the real and imaginary parts in complex multiplication, i.e. by accessing sub-quad words. The inner-most loop body executes in roughly 70 machine cycles. In order to multiply the two colour matrices the inner loop body is executed 9 times.

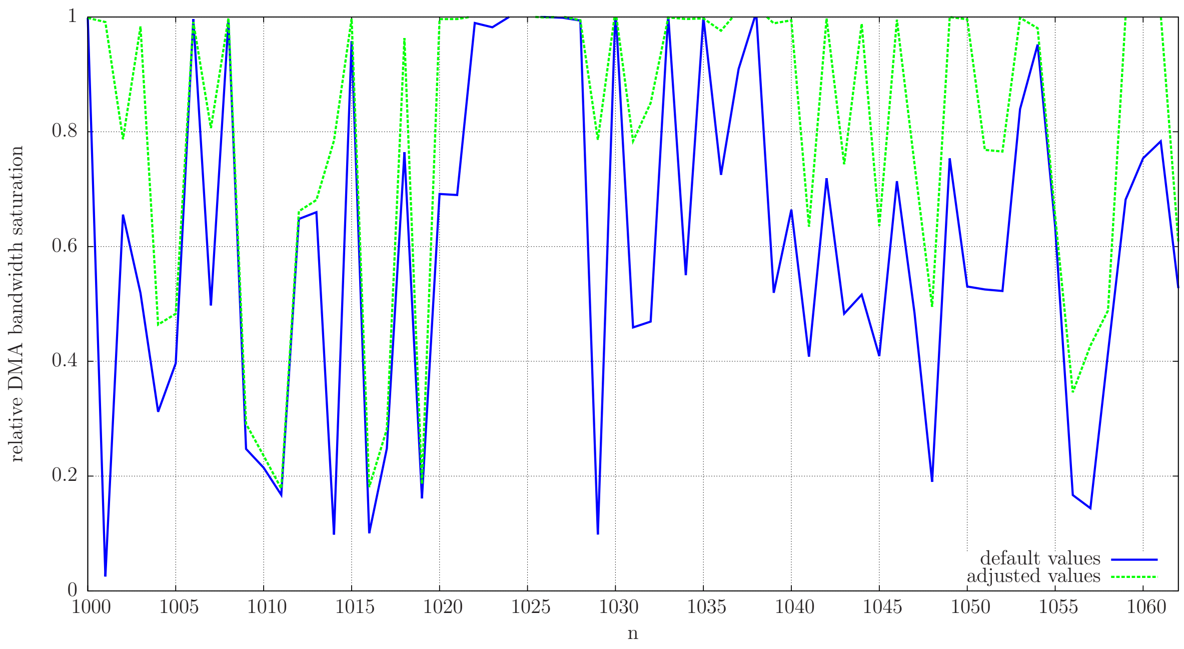


Figure 19.3 – Impact of GNU C++ SPU Compiler optimisation options, i.e. tree inliner and compiler loop-unrolling controlled by adjusted compiler options in comparison to default values.

19.4 Compiler Optimisations

The SPU GNU C++ Compiler was used to build the SPU executables. The next benchmark measurement concentrates on the optimisations the compiler is able to carry out. This includes the tree inliner and the loop-unrolling facilities. Here, the QDP++ specialisations are included.

The compiler options for controlling the inlining facility are adjusted in such a way that one instance of the body of the evaluation loop is completely inlined. This does not mean that all loops in the generic implementation of the operations are unrolled as well.

The compiler options used here are given in App. E.1.

Fig. 19.3 depicts the benchmark results for this measurements. Again, the relative DMA bandwidth saturation is shown normalised by the benchmark results for switched off computation.

For most functions a significant performance improvement is observed when using the adjusted options for the tree inliner.

For example $n = 1014$ represents the following operation

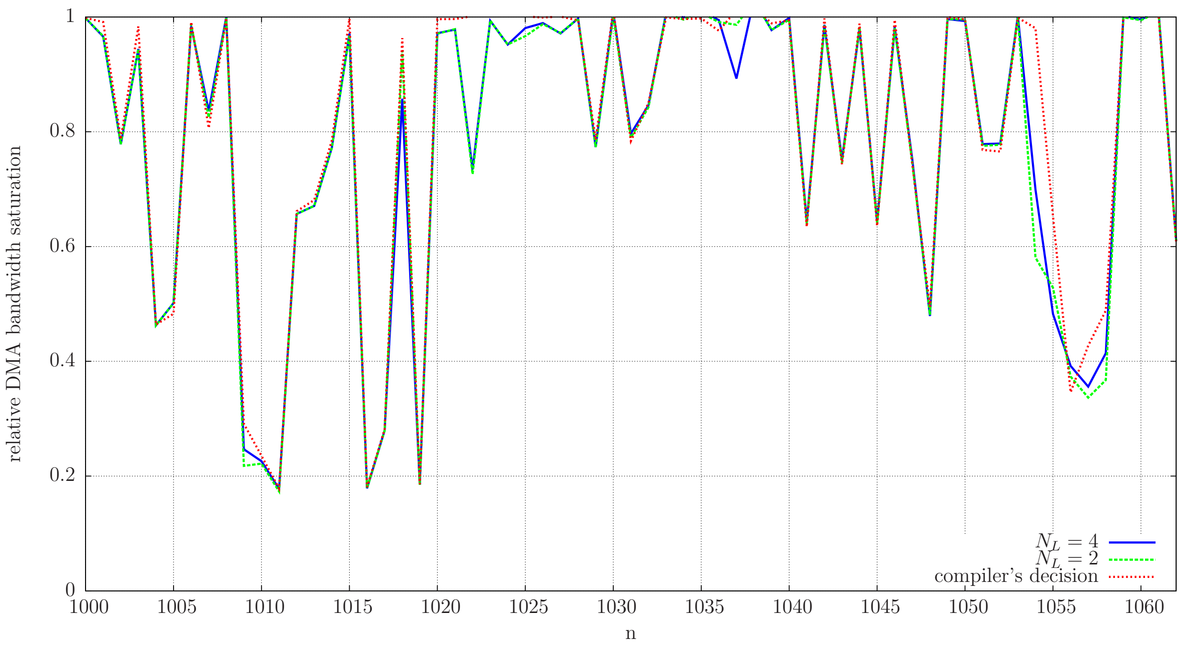


Figure 19.4 – Forced evaluation loop unroll with $N_L = 2$ and $N_L = 4$ in comparison to compiler's decision.

n	QDP++ function	index range
1014	$M^{\mathrm{SC}}_{i,j,k,l} += (M'^{\mathrm{C}} \times M''^{\mathrm{SC}} + M'''^{\mathrm{SC}})_{i,j,k,l}$	$\{i,1,3\}\{j,1,3\}\{k,1,4\}\{l,1,4\}$

The superscripts ^{C} and ^{SC} stand for colour and spin-colour respectively. A substantial performance gain is recognised from around 7% with the default values for the compiler options to around 60% DMA saturation with optimised values applied. If inlining is constrained too restrictively the compiler fails to inline functions like for example the complex multiplication and issues branches to it instead. These branches come with an overhead. And naturally loop branches are at least once not correctly predicted, i.e. when the last operation in a loop is carried out, thus resulting in a performance penalty. The compiler "predicts" the loop iteration count to be larger than one.

19.5 Evaluation Loop

Next a benchmark measurement was carried out to be able to study the impact of unrolling the evaluation. Taking into account the limited LS size, the unrolling is done partially controlled by the parameter N_L, see Alg. 6 on page 224.

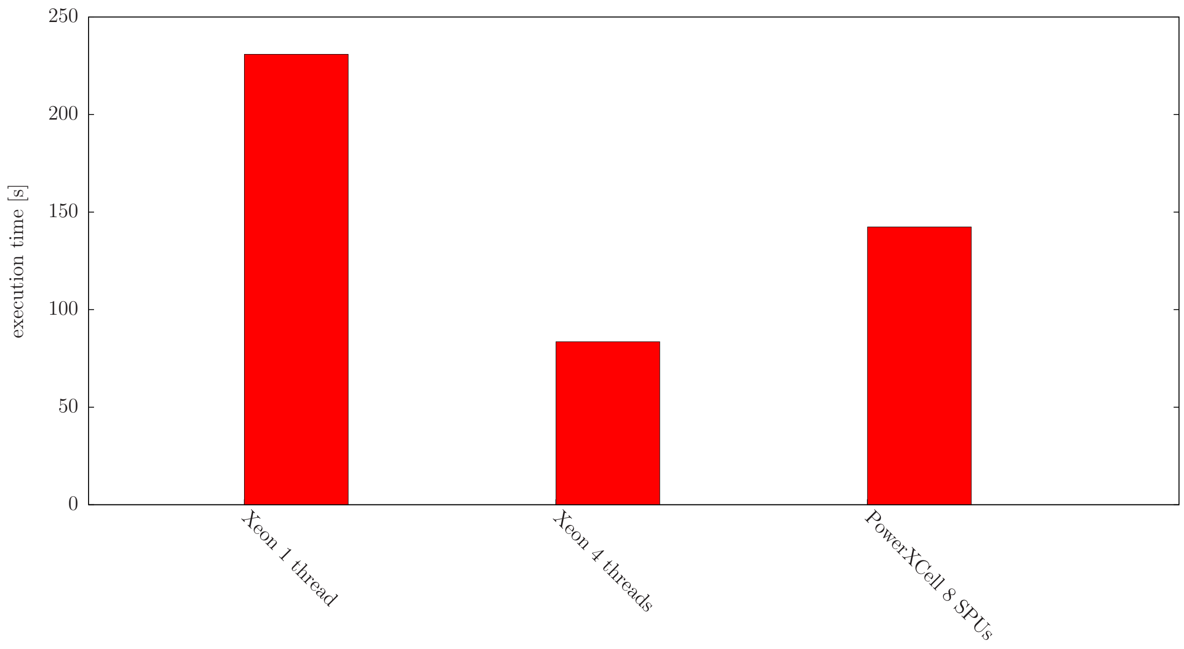


Figure 19.5 – Comparison of the execution time for calculating the hadronic spectrum for a commodity CPU and the IBM PowerXCell 8i Processor.

Alg. 6 is applied for the evaluation loop with loop unrolling parameter $N_L = 2$ and $N_L = 4$.

Fig. 19.4 depicts the benchmark results for this measurements. Again, the relative DMA bandwidth saturation is shown normalised by the benchmark results for switched off computation.

No significant performance gains or losses are detected when unrolling the evaluation. It seems to be safe to leave the decision whether to unroll the evaluation loop to the compiler and to only force it to inline all functions into the evaluation loop's body.

19.6 Chroma on Cell vs. Commodity CPU

This work compares the performance of a Chroma build with the implementation of QDP++ for the IBM PowerXCell 8i Processor with a build for a commodity processor. The systems under considerations were an IBM PowerXCell 8i Processor on a Blade Center QS22 using 8 SPUs and an Intel Xeon Processor 5130 (4M Cache, 2.00 GHz, 1333 MHz FSB) with 2 cores and 2 hardware threads. The build for the commodity CPU utilises the SciDAC component for multi-threading, QMT, to make use of all possible 4 threads. In order to

System	execution time [s]
IBM PowerXCell 8i Processor	142.4
Intel Xeon (no QMT)	230.8
Intel Xeon (with QMT)	83.5

Table 19.2 – Comparison of the execution time for calculating the hadronic spectrum for a commodity CPU and the IBM PowerXCell 8i Processor.

detect the impact of utilising multi-threading this work measures the execution time when not making use of the QMT library. The hadronic spectrum was calculated (mesonic and baryonic two-point functions) for the same gauge configuration, i.e. a $16^3 \times 32$ lattice, on both systems and measured the total execution, i.e. only the Chroma measurement for calculation of the hadronic spectrum was taken into account. The builds for both systems were configured to use double precision floating point numbers.

Fig. 19.5 and Tab. 19.2 show the execution time for the three setups. When using all available hardware threads the Intel Xeon outperforms the IBM PowerXCell 8i Processor nearly by a factor of 2 even when all SPUs are active.

Chapter 20

Conclusion and Outlook

In this work new design concept were successfully developed on how to implement an active C++ library like QDP++ on a accelerator type of processors like the IBM PowerXCell 8i processor. A proof-of-concept has been provided. Furthermore it was possible to run a QDP++ based physics application (Chroma) with reasonable performance on the IBM PowerXCell 8i Processor.

This work significantly extents beyond usual code porting activities: A new strategy was pursued leveraging the Portable Expression Template Engine (PETE).

The new design concepts include to write out at run-time of the main application the involved PETE expressions into a database. Furthermore it includes a code-generator which processes the database and generates program code fractions for the accelerator core. The newly generated code fractions rely on the functionality of QDP++. In order to meet the hardware characteristics of the accelerator this work provides an optimised version of QDP++ for this processor type. All involved PETE expressions are available as compiled functions, optimised for the accelerator.

As a last step this work proposes to bundle all functions into a library and link the main application against it. In this way all program parts of the main application involving PETE expressions execute on the accelerators.

This work briefly outlined the modifications and optimisations carried out in order to achieve a version of QDP++ suitable for the accelerator cores – in case of the Cell processor, the SPEs.

Support for the SIMD organisation was added. A change in the data organisation ensured that complex numbers always fit into one processor register. To reduce memory bandwidth requirements additional attributes were introduced to QDP++ data types which describe the data object's access pattern. The standard QDP++ memory allocator was replaced by a stack memory based implementation. A major modification was adding support for accessing main memory via DMA transfers. In this way the SPEs access main memory with asynchronous DMA transfers which execute in parallel to SPE computation. With multi-buffering algorithms memory latencies and transfer overheads could be (partially) hidden by computation. Parallelisation on several SPEs was achieved by assigning each SPE to a disjoint part of the problem, i.e. each SPE operates on a different part of the data vector. In order to find the optimal problem size and alignment settings for each SPE compile-time calculations were carried out. To cope with the limited size of the LS SPE code overlay techniques were employed.

The employed optimisation techniques lead to SPE code with a reasonable performance. Clearly, more aggressive optimisation techniques can be applied. Providing further special implementations not only at the level of complex numbers but, e.g. for colour SU(3) matrix operations, would increase the performance of individual operations even more.

On the other hand roughly half of the QDP++ functions considered for benchmarking already execute with a good saturation of the available memory bandwidth. Further improvement in terms of number of floating-point operations would not necessarily result in an improved overall performance.

Since information on the used QDP++ functions is not fully available at compile-time, a first build and execution of the main application is required – first without support for the accelerator cores. The build process can be improved if some parts of it move into the compiler. This eliminate the need for the first build and consequently it is not necessary to execute the application prior to production use. On the other hand this gives rise to other problems: First, all QDP++ functions found in the code base of the main application are built – in case of Chroma this is an enormous set. Building an SPE version for all of them results in an enormous SPE binary. Even with code overlays for each reachable function a small function stub calling out to the function is needed. The sum of all these stubs plus data regions most likely exceeds the SPE's LS and one has to leverage other code overlay techniques to overcome this problem. Second, the build time of the main application increases substantially.

This work applied the new design concepts and implemented a version of QDP++ for the IBM PowerXCell 8i Processor. The performance of a physics application (Chroma) was compared on the Cell processor with the performance of Chroma built with the original QDP++ on a commodity processor. The systems under considerations were an IBM PowerXCell 8i Processor on a Blade Center QS22 using 8 SPEs and an Intel Xeon Processor with 2 cores and 2 hardware threads. In this work the hadronic spectrum (mesonic and baryonic two-point functions) was calculated for a given gauge configuration, i.e. a $16^3 \times 32$ lattice, on both systems and the total execution was measured. A reasonable execution time was measured on the IBM PowerXCell 8i Processor in comparison to the commodity processor.

In order to achieve a good performance on the QPACE computer (Sec. 14.7) accelerated code is necessary in all compute-intensive parts of the code (Sec. 14.11). This work demonstrated that a good performance of code parts usually not subject to optimisation can be achieved. A hand-tuned optimised compute kernels for the QPACE computer exists, e.g. an inverter with excellent performance [165].

However, in order to aim for Chroma on QPACE the development of design concepts is required that extents beyond the usual porting activities, i.e. providing hand-tuned optimised compute kernels. The developed design concepts are good candidates to provide the necessary software idioms for a complete Chroma build on QPACE. For certain compute-tasks, e.g. the inversion of the fermion matrix, hand-tuned optimised compute kernels are still required. The QDP++ API allows for a seamless integration of these kernels.

The development of the Cell processor has stopped. Other accelerator-based architectures are getting more important targets in the HPC market. These include the GPGPUs and Intel's recently announced Many Integrated Core (MIC) Chips, and AMD's Fusion. A GPU-based system is current holding the first position in the latest iteration (Nov. 2010) of the Top 500 list [145].

The design concept developed in this work is not bound to specific microprocessor. It is retargetable and similar challenges are likely to be encountered when targeting to other accelerator based architectures.

Appendices

Appendix A

Appendix

A.1 Fourier Transformation of Dirac Operator

For the following calculation we set $L = L_T$ and use periodic boundary conditions. First we calculate the Fourier transform of the naive Dirac operator in momentum space

$$\widetilde{D}_{\text{naiv}}(p|q) = \frac{1}{|\Lambda|} \sum_{n,m \in \Lambda} e^{-ip\cdot na} D_{\text{naiv}}(n|m) e^{iq\cdot ma} \tag{A.1}$$

$$= \frac{1}{|\Lambda|} \sum_{n \in \Lambda} e^{-i(p-q)\cdot na} \left(\sum_{\mu=1}^{4} \gamma_\mu \frac{e^{+iq_\mu a} - e^{-iq_\mu a}}{2a} + m_0 \right) \tag{A.2}$$

$$= \delta_{pq} \left(m_0 + \frac{i}{a} \sum_{\mu=1}^{4} \gamma_\mu \sin(p_\mu a) \right). \tag{A.3}$$

Thus the Dirac operator is diagonal in momentum space, which in turn leads to a straightforward calculation of its inverse:

$$\widetilde{D}(p)^{-1} = \frac{m - ia^{-1} \sum_\mu \gamma_\mu \sin(p_\mu a)}{m^2 + a^{-2} \sum_\mu \sin(p_\mu a)^2}. \tag{A.4}$$

A.2 Symmetry Transformation of Hadron Interpolators

On the lattice the parity transformation $\mathcal{P}$ is implemented as

$$\psi(\mathbf{n}, n_4) \xrightarrow{\mathcal{P}} \gamma_4 \psi(-\mathbf{n}, n_4) \tag{A.5}$$

$$\overline{\psi}(\mathbf{n}, n_4) \xrightarrow{\mathcal{P}} \overline{\psi}(-\mathbf{n}, n_4)\gamma_4 \tag{A.6}$$

$$U_i(\mathbf{n}, n_4) \xrightarrow{\mathcal{P}} U_i(-\mathbf{n} - \hat{i}, n_4)^\dagger \tag{A.7}$$

$$U_4(\mathbf{n}, n_4) \xrightarrow{\mathcal{P}} U_4(-\mathbf{n}, n_4). \tag{A.8}$$

Whereas the implementation of the charge conjugation transformation $\mathcal{C}$ which transforms a particle into its anti-particle follows

$$\psi(n) \xrightarrow{\mathcal{C}} C^{-1}\overline{\psi}(n)^T \tag{A.9}$$

$$\overline{\psi}(n) \xrightarrow{\mathcal{C}} -\psi(n)^T C \tag{A.10}$$

$$U_\mu(n) \xrightarrow{\mathcal{C}} \left(U_\mu(n)^\dagger\right)^T \tag{A.11}$$

with the superscript ^{T} denoting the transpose and with the charge conjugation matrix C acting only on the Dirac spin indices and is defined to obey the relation

$$C\gamma_\mu C^{-1} = -\gamma_\mu^T \tag{A.12}$$

The explicit form of C depends on the representation of the γ-matrices used.

Under parity $\mathcal{P}$ our nucleon interpolator transforms like

$$B^{\mathcal{P}}(\mathbf{n}, n_4) = \epsilon_{abc}\gamma_4 u(-\mathbf{n}, n_4)_a \left(u(-\mathbf{n}, n_4)_b^T \gamma_4^T C\gamma_5\gamma_4 d(-\mathbf{n}, n_4)_c\right) \tag{A.13}$$

$$= \epsilon_{abc}\gamma_4 u(-\mathbf{n}, n_4)_a \left(u(-\mathbf{n}, n_4)_b^T C\gamma_5 d(-\mathbf{n}, n_4)_c\right) \tag{A.14}$$

$$= \gamma_4 B(-\mathbf{n}, n_4) \tag{A.15}$$

where we used $\gamma_\mu^T C = -C\gamma_\mu$.

Appendix B

Measurement Data

B.1 Pion Decay Constant

The following tables list the measurement data. If the table is split into an upper and lower part by a horizontal line, the data in the lower part was not used for fitting.

Table B.1 – Pion decay constant f_π for lattice size $24^3 \times 48$.

$m_\pi^2 [GeV^2]$	$f_\pi [GeV]$
0.233	0.16562(97)
0.199	0.16520(75)
0.172	0.1557(12)
0.143	0.1517(13)
0.123	0.1462(13)

Table B.2 – FSE corrected pion decay constant f_π for lattice size $24^3 \times 48$.

$m_\pi^2 [GeV^2]$	$f_\pi [GeV]$
0.233	0.16605(97)
0.198	0.16580(75)
0.172	0.1565(12)
0.142	0.1530(13)
0.122	0.1479(13)

Table B.3 – Pion decay constant f_π for lattice size $32^3 \times 64$.

$m_\pi^2[GeV^2]$	$f_\pi[GeV]$
0.112	0.1551(14)
0.082	0.1498(16)
0.064	0.1463(25)
0.196	0.1621(34)

Table B.4 – FSE corrected pion decay constant f_π for lattice size $32^3 \times 64$.

$m_\pi^2[GeV^2]$	$f_\pi[GeV]$
0.112	0.1555(14)
0.081	0.1504(16)
0.064	0.1474(25)
0.196	0.1622(34)

B.2 $n = 1$ Moment of Polarised PDF

Table B.5 – $\langle 1 \rangle_{\Delta q}$ for N

$m_\pi^2[GeV^2]$	u	d	$u-d$	$u+d$
0.233	0.909(14)	-0.2670(53)	1.176(16)	0.642(14)
0.199	0.9110(100)	-0.2775(60)	1.189(12)	0.633(12)
0.172	0.910(21)	-0.256(10)	1.166(23)	0.655(23)
0.143	0.925(25)	-0.279(13)	1.204(28)	0.645(28)
0.123	0.854(24)	-0.274(17)	1.128(33)	0.580(26)

Table B.6 – $\langle 1 \rangle_{\Delta q}$ for Σ

$m_\pi^2[GeV^2]$	u	d	$u-d$	$u+d$
0.233	0.921(16)	-0.2575(87)	1.179(20)	0.664(17)
0.199	0.9110(100)	-0.2775(60)	1.189(12)	0.633(12)
0.172	0.910(18)	-0.2646(86)	1.175(21)	0.645(19)
0.143	0.910(20)	-0.2883(88)	1.199(24)	0.622(20)
0.123	0.857(18)	-0.2900(98)	1.147(23)	0.567(18)

Table B.7 – $\langle 1 \rangle_{\Delta q}$ for N

$m_\pi^2[GeV^2]$	u	d	$u-d$	$u+d$
0.233	0.866(19)	-0.2734(96)	1.140(22)	0.593(21)
0.199	0.9110(100)	-0.2775(60)	1.189(12)	0.633(12)
0.172	0.932(16)	-0.2567(75)	1.189(19)	0.675(17)
0.143	0.975(15)	-0.2708(77)	1.246(19)	0.705(14)
0.123	0.951(16)	-0.2581(71)	1.209(20)	0.693(15)

Table B.8 – $\delta_{SU(3)} = g_A^N - g_A^\Sigma + g_A^\Xi$

$(m_\pi/X_\pi)^2$	$\delta_{SU(3)}$
1.188	-0.018(12)
0.863	-0.000(13)
0.726	0.023(17)
0.621	0.013(22)
0.998	0.000(00)

B.3 $n = 2$ Moment of Unpolarised PDF

Table B.9 – $\langle x \rangle_q$ for N

$m_\pi^2[GeV^2]$	u	d	$u-d$	$u+d$
0.233	0.3535(49)	0.1543(23)	0.1992(30)	0.5077(70)
0.199	0.3531(31)	0.1505(17)	0.2026(22)	0.5036(44)
0.172	0.3519(73)	0.1524(32)	0.1995(50)	0.5043(100)
0.143	0.3602(77)	0.1559(39)	0.2043(52)	0.516(11)
0.123	0.3543(84)	0.1533(44)	0.2010(62)	0.508(12)

Table B.10 – $\langle x \rangle_q$ for Σ

$m_\pi^2[GeV^2]$	u	d	$u-d$	$u+d$
0.233	0.3624(60)	0.1483(34)	0.2141(42)	0.5107(88)
0.199	0.3531(31)	0.1505(17)	0.2026(22)	0.5036(44)
0.172	0.3482(63)	0.1575(29)	0.1906(41)	0.5057(89)
0.143	0.3487(53)	0.1649(25)	0.1838(37)	0.5136(75)
0.123	0.3418(67)	0.1656(35)	0.1762(42)	0.5073(98)

Table B.11 – $\langle x \rangle_q$ for Ξ

$m_\pi^2[GeV^2]$	u	d	$u-d$	$u+d$
0.233	0.3512(64)	0.1610(30)	0.1902(43)	0.5122(90)
0.199	0.3531(31)	0.1505(17)	0.2026(22)	0.5036(44)
0.172	0.3552(54)	0.1502(26)	0.2050(36)	0.5054(76)
0.143	0.3666(50)	0.1487(28)	0.2179(34)	0.5153(74)
0.123	0.3666(46)	0.1444(26)	0.2222(31)	0.5110(67)

Table B.12 – Pion and kaon masses on $24^3 \times 48$ lattices with lattice spacing, $a = 0.083(3)$fm [68], where the error on the lattice spacing has been included in the errors for m_π and m_K. The last four columns contain our results for ratios of the hyperon quark momentum fractions.

m_π [MeV]	m_K [MeV]	$\langle x \rangle_u^\Sigma / \langle x \rangle_u^p$	$\langle x \rangle_s^\Sigma / \langle x \rangle_d^p$	$\langle x \rangle_s^\Xi / \langle x \rangle_u^p$	$\langle x \rangle_u^\Xi / \langle x \rangle_d^p$
460(17)	401(15)	1.0263(51)	0.960(12)	0.993(23)	1.044(28)
423(15)	423(15)	1.0	1.0	1.0	1.0
395(14)	438(16)	0.9888(44)	1.0344(70)	1.010(25)	0.985(24)
360(13)	451(16)	0.9670(83)	1.059(14)	1.019(26)	0.953(29)
334(12)	463(17)	0.9631(94)	1.082(18)	1.037(29)	0.940(30)

B.4 $n = 1$ Moment of Tensor GPD

Table B.13 – $\langle 1 \rangle_{\delta q}$ for N

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	0.832(12)	-0.2015(39)	1.034(14)	0.631(11)
0.998	0.8306(78)	-0.2098(42)	1.0404(81)	0.6208(96)
0.863	0.819(17)	-0.1963(70)	1.015(19)	0.622(17)
0.726	0.840(19)	-0.1996(93)	1.040(22)	0.640(20)
0.621	0.788(22)	-0.191(11)	0.980(27)	0.597(22)

Table B.14 – $\langle 1 \rangle_{\delta q}$ for Σ

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	0.834(14)	-0.1984(55)	1.032(16)	0.635(13)
0.998	0.8306(78)	-0.2098(42)	1.0404(81)	0.6208(96)
0.863	0.818(15)	-0.1991(59)	1.017(17)	0.619(15)
0.726	0.841(15)	-0.2072(62)	1.048(17)	0.633(15)
0.621	0.800(17)	-0.2108(73)	1.010(21)	0.589(15)

Table B.15 – $\langle 1 \rangle_{\delta q}$ for Ξ

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	0.816(15)	-0.2012(66)	1.017(18)	0.614(16)
0.998	0.8306(78)	-0.2098(42)	1.0404(81)	0.6208(96)
0.863	0.832(13)	-0.1963(52)	1.028(15)	0.635(13)
0.726	0.874(13)	-0.1985(54)	1.073(15)	0.676(12)
0.621	0.859(11)	-0.1946(49)	1.054(14)	0.665(11)

B.5 Ratio $n = 1$ Moment of Tensor GPD over f_T

Table B.16 – $\langle 1 \rangle_{\delta q}/f_T$ for N

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	5.79(17)	-1.402(46)	7.19(21)	4.39(14)
0.998	5.45(11)	-1.376(38)	6.82(14)	4.071(99)
0.863	5.94(25)	-1.423(73)	7.36(30)	4.51(21)
0.726	6.21(32)	-1.477(97)	7.69(39)	4.74(27)
0.621	5.24(25)	-1.270(90)	6.50(31)	3.97(21)

Table B.17 – $\langle 1 \rangle_{\delta q}/f_T$ for Σ

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	5.80(18)	-1.380(53)	7.18(22)	4.42(15)
0.998	5.45(11)	-1.376(38)	6.82(14)	4.071(99)
0.863	5.93(24)	-1.444(68)	7.38(30)	4.49(20)
0.726	6.22(31)	-1.532(85)	7.75(38)	4.68(25)
0.621	5.31(24)	-1.400(73)	6.71(30)	3.91(18)

Table B.18 – $\langle 1 \rangle_{\delta q}/f_T$ for Ξ

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	5.67(18)	-1.399(59)	7.07(22)	4.27(16)
0.998	5.45(11)	-1.376(38)	6.82(14)	4.071(99)
0.863	6.03(24)	-1.424(65)	7.45(30)	4.61(19)
0.726	6.47(32)	-1.468(79)	7.94(39)	5.00(25)
0.621	5.71(23)	-1.293(60)	7.00(29)	4.41(19)

B.6 $n = 2$ Moment of Polarised PDF

Table B.19 – $\langle x \rangle_{\Delta q}$ for N

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	0.3706(64)	-0.0786(30)	0.4492(72)	0.2921(69)
0.998	0.3664(57)	-0.0841(36)	0.4505(65)	0.2823(70)
0.863	0.358(11)	-0.0794(51)	0.437(13)	0.278(11)
0.726	0.382(11)	-0.0801(70)	0.462(13)	0.302(13)
0.621	0.366(13)	-0.0782(85)	0.445(14)	0.288(17)

Table B.20 – $\langle x \rangle_{\Delta q}$ for Σ

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	0.3843(82)	-0.0714(41)	0.4564(90)	0.3135(94)
0.998	0.3664(57)	-0.0841(36)	0.4505(65)	0.2823(70)
0.863	0.3553(95)	-0.0842(40)	0.439(11)	0.2706(93)
0.726	0.3642(77)	-0.0873(50)	0.4506(98)	0.2761(86)
0.621	0.3516(96)	-0.0974(50)	0.448(10)	0.253(11)

Table B.21 – $\langle x \rangle_{\Delta q}$ for Ξ

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	0.3642(92)	-0.0795(49)	0.443(10)	0.284(11)
0.998	0.3664(57)	-0.0841(36)	0.4505(65)	0.2823(70)
0.863	0.3703(85)	-0.0807(31)	0.4514(95)	0.2901(87)
0.726	0.3977(70)	-0.0792(38)	0.4779(83)	0.3194(76)
0.621	0.3900(66)	-0.0766(26)	0.4680(74)	0.3147(68)

B.7 Ratios of Axial Charge

Table B.22 – Measurement data for g_A^Σ/g_A^N for lattice size $24^3 \times 48$.

$(m_\pi/X_\pi)^2$	g_A^Σ/g_A^N
1.188	0.7832(56)
0.998	0.7665(44)
0.863	0.7802(70)
0.726	0.7561(96)
0.621	0.760(13)

Table B.23 – Measurement data for g_A^Ξ/g_A^N for lattice size $24^3 \times 48$.

$(m_\pi/X_\pi)^2$	g_A^Ξ/g_A^N
1.188	-0.2325(88)
0.998	-0.2335(44)
0.863	-0.2202(80)
0.726	-0.2249(84)
0.621	-0.2288(92)

Table B.24 – Measurement data for $(g_A^N - g_A^\Xi)/g_A^\Sigma$ for lattice size $24^3 \times 48$.

$(m_\pi/X_\pi)^2$	$(g_A^N - g_A^\Xi)/g_A^\Sigma$
1.188	1.574(17)
0.998	1.609(15)
0.863	1.564(16)
0.726	1.620(22)
0.621	1.617(24)

Table B.25 – Measurement data for $(g_A^N + g_A^{\Xi})/g_A^{\Sigma}$ for lattice size $24^3 \times 48$.

$(m_\pi/X_\pi)^2$	$(g_A^N + g_A^{\Xi})/g_A^{\Sigma}$
1.188	0.980(13)
0.863	1.000(15)
0.726	1.025(18)
0.621	1.015(26)
0.998	1.000(00)

Table B.26 – Measurement data for g_A^N/f_π for lattice size $24^3 \times 48$.

$m_\pi^2[GeV^2]$	g_A^N/f_π
0.233	7.04(10)
0.199	7.252(75)
0.172	7.45(16)
0.143	7.92(19)
0.123	7.63(24)

B.8 $n = 2$ Moment of Tensor GPD

Table B.27 – $\langle x \rangle_{\delta q}$ for N

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	0.2013(33)	-0.0405(12)	0.2419(35)	0.1608(35)
0.998	0.2062(27)	-0.0446(15)	0.2508(29)	0.1616(33)
0.863	0.2097(50)	-0.0388(26)	0.2485(59)	0.1708(54)
0.726	0.2087(57)	-0.0397(33)	0.2484(56)	0.1690(75)
0.621	0.2118(73)	-0.0437(38)	0.2555(82)	0.1681(81)

Table B.28 – $\langle x \rangle_{\delta q}$ for Σ

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	0.2068(41)	-0.0403(22)	0.2475(46)	0.1669(46)
0.998	0.2062(27)	-0.0446(15)	0.2508(29)	0.1616(33)
0.863	0.2058(43)	-0.0394(18)	0.2449(50)	0.1661(43)
0.726	0.2014(43)	-0.0414(22)	0.2422(45)	0.1595(51)
0.621	0.1998(46)	-0.0455(21)	0.2447(55)	0.1536(46)

Table B.29 – $\langle x \rangle_{\delta q}$ for Ξ

$(m_\pi/X_\pi)^2$	u	d	$u-d$	$u+d$
1.188	0.1991(51)	-0.0433(24)	0.2421(53)	0.1555(59)
0.998	0.2062(27)	-0.0446(15)	0.2508(29)	0.1616(33)
0.863	0.2079(35)	-0.0389(17)	0.2470(40)	0.1693(38)
0.726	0.2119(33)	-0.0379(16)	0.2504(38)	0.1745(35)
0.621	0.2096(29)	-0.0406(13)	0.2510(33)	0.1697(31)

Appendix C

Fit Results

C.1 Pion Decay Constant

Table C.1 – Fit result for the pion decay constant with the fit ansatz $f_\pi = a_0 + a_1 m_\pi^2$. The fit is based on data shown in Table B.3. The extrapolated quantity and resulting estimate for the renormalisation constant are shown in the last two columns.

a_0	a_1	χ^2/dof	quality	$f_\pi({m_\pi^{\mathrm{phys}}}^2)/Z_A$	Z_A^{est}
0.1350(50)	0.179(52)	0.01	0.90	0.1383(51)	0.940(35)

C.2 $n = 1$ Moment of Polarised PDF

Table C.2 – Fit results for $\langle 1\rangle^B_{\Delta q}$ with the fit ansatz: $\langle 1\rangle_{\Delta q} = a_0 + a_1 m_\pi^2$. The extrapolation to the physical point is given in the last column.

B	q	a_0	a_1	χ^2/dof	quality	$\langle 1\rangle_{\Delta q}/Z_A$
N	Δu	0.936(57)	-0.12(28)	0.13	0.88	0.934(57)
N	Δd	-0.282(27)	0.06(13)	4.14	0.02	-0.281(27)
N	$\Delta u - \Delta d$	1.216(65)	-0.16(32)	1.22	0.30	1.213(65)
N	$\Delta u + \Delta d$	0.653(60)	-0.06(30)	0.74	0.48	0.652(61)
Σ	Δu	0.890(52)	0.12(27)	0.14	0.87	0.892(53)
Σ	Δd	-0.321(25)	0.25(13)	3.93	0.02	-0.316(25)
Σ	$\Delta u - \Delta d$	1.210(63)	-0.12(32)	0.62	0.54	1.208(63)
Σ	$\Delta u + \Delta d$	0.568(53)	0.37(27)	1.25	0.29	0.575(53)
Ξ	Δu	1.140(46)	-1.17(25)	0.29	0.75	1.119(47)
Ξ	Δd	-0.249(24)	-0.11(13)	4.00	0.02	-0.251(24)
Ξ	$\Delta u - \Delta d$	1.387(57)	-1.04(30)	1.70	0.18	1.369(57)
Ξ	$\Delta u + \Delta d$	0.889(46)	-1.28(25)	0.19	0.83	0.866(47)

Table C.3 – Fit result for SU(3) symmetry breaking term using the fit ansatz: $\delta_{SU(3)} = a_0(m_\pi/X_\pi - 1)$ The extrapolation to the physical point is given in the last column.

Ratio	a_0	χ^2/dof	quality	value
$(g_A^N - g_A^\Sigma + g_A^\Xi)/Z_A$	-0.062(33)	1.13	0.33	0.052(34)

Table C.4 – Fit result for ratios of baryon axial charge using the fit ansatz: $R = a_0 + a_1(m_\pi/X_\pi)$ with R one of the ratios given in the first column. The extrapolation to the physical point is given in the last column.

Ratio	a_0	a_1	χ^2/dof	quality	value
g_A^Σ/g_A^N	0.733(21)	0.040(21)	5.99	0.00	0.740(21)
g_A^Ξ/g_A^N	-0.203(24)	-0.028(24)	1.32	0.27	-0.208(24)
$(g_A^N - g_A^\Xi)/g_A^\Sigma$	1.631(53)	-0.044(54)	6.52	0.00	1.624(53)

Table C.5 – Fit result for the ratio $(g_A^N + g_A^{\Xi})/g_A^{\Sigma}$ with fit ansatz $R = a_0(m_\pi/X_\pi - 1) + 1$. The extrapolation to the physical point is given in the last column.

Ratio	a_0	χ^2/dof	quality	value
$(g_A^N + g_A^{\Xi})/g_A^{\Sigma}$	-0.082(44)	0.79	0.45	1.068(44)

C.3 $n = 2$ Moment of Unpolarised PDF

Table C.6 – Fit results for $\langle x \rangle_q^B$ with the fit ansatz: $\langle x \rangle_q^B = a_0 + a_1 m_\pi^2$. The extrapolation to the physical point is given in the last column.

B	q	a_0	a_1	χ^2/dof	quality	$\langle x \rangle_q / Z_{v2b}$
N	u	0.360(14)	-0.006(14)	0.64	0.59	0.359(14)
N	d	0.1530(69)	-0.0006(69)	2.77	0.04	0.1529(70)
N	$u-d$	0.2069(95)	-0.0055(95)	0.97	0.41	0.2059(97)
N	$u+d$	0.513(20)	-0.007(20)	1.11	0.34	0.512(20)
Σ	u	0.324(12)	0.030(13)	0.69	0.56	0.329(12)
Σ	d	0.1909(63)	-0.0389(68)	2.83	0.04	0.1844(64)
Σ	$u-d$	0.1338(80)	0.0682(87)	0.40	0.76	0.1453(82)
Σ	$u+d$	0.514(17)	-0.008(19)	1.38	0.25	0.513(18)
Ξ	u	0.388(10)	-0.034(11)	1.26	0.29	0.382(10)
Ξ	d	0.1305(54)	0.0223(59)	4.11	0.01	0.1343(55)
Ξ	$u-d$	0.2564(69)	-0.0550(77)	1.67	0.17	0.2471(70)
Ξ	$u+d$	0.519(15)	-0.013(16)	1.90	0.13	0.517(15)

C.4 $n = 1$ Moment of Tensor GPD

Table C.7 – Fit results for $\langle 1\rangle^{B}_{\delta q}$ with the fit ansatz: $\langle 1\rangle_{\delta q} = a_0 + a_1 m_\pi^2$. The extrapolation to the physical point is given in the last column.

B	q	a_0	a_1	χ^2/dof	quality	$\langle 1\rangle_{\delta q}/Z$
N	u	0.790(35)	0.038(35)	2.94	0.03	0.797(35)
N	d	-0.193(15)	-0.010(15)	4.47	0.00	-0.195(15)
N	$u-d$	0.982(41)	0.053(41)	4.05	0.01	0.991(42)
N	$u+d$	0.600(35)	0.025(35)	2.05	0.10	0.604(35)
Σ	u	0.796(30)	0.034(32)	3.11	0.03	0.802(31)
Σ	d	-0.217(13)	0.013(14)	3.56	0.01	-0.215(13)
Σ	$u-d$	1.015(37)	0.021(38)	3.09	0.03	1.019(37)
Σ	$u+d$	0.573(29)	0.051(31)	3.71	0.01	0.582(29)
Ξ	u	0.923(25)	-0.093(28)	3.24	0.02	0.908(25)
Ξ	d	-0.182(11)	-0.022(12)	3.93	0.01	-0.186(11)
Ξ	$u-d$	1.102(29)	-0.066(32)	3.46	0.02	1.091(30)
Ξ	$u+d$	0.743(24)	-0.116(28)	3.51	0.01	0.723(25)

C.5 Ratio $n = 1$ Moment of Tensor GPD over f_T

Table C.8 – Fit results for $\langle 1 \rangle^{B}_{\delta q}/f_T$ with the fit ansatz: $\langle 1 \rangle_{\delta q}/f_T = a_0 + a_1 m_\pi^2$. The extrapolation to the physical point is given in the last column.

B	q	a_0	a_1	χ^2/dof	quality	$\langle 1 \rangle_{\delta q}/f_T$
N	u	6.20(69)	-0.55(67)	7.51	0.00	6.11(70)
N	d	-1.49(20)	0.09(19)	0.89	0.41	-1.48(20)
N	$u-d$	7.70(84)	-0.66(82)	6.17	0.00	7.59(85)
N	$u+d$	4.68(56)	-0.41(55)	8.60	0.00	4.61(57)
Σ	u	6.28(68)	-0.63(68)	7.70	0.00	6.17(69)
Σ	d	-1.68(19)	0.28(19)	1.24	0.29	-1.64(20)
Σ	$u-d$	7.92(84)	-0.88(83)	6.46	0.00	7.77(85)
Σ	$u+d$	4.68(55)	-0.41(55)	8.61	0.00	4.61(56)
Ξ	u	7.06(69)	-1.41(69)	8.18	0.00	6.82(70)
Ξ	d	-1.54(19)	0.14(20)	0.74	0.47	-1.52(20)
Ξ	$u-d$	8.55(85)	-1.50(84)	6.59	0.00	8.29(86)
Ξ	$u+d$	5.66(57)	-1.40(57)	9.51	0.00	5.43(58)

C.6 $n = 2$ Moment of Polarised PDF

Table C.9 – Fit results for $\langle x \rangle^{B}_{\Delta q}$ with the fit ansatz: $\langle x \rangle_{\Delta q} = a_0 + a_1 m_\pi^2$. The extrapolation to the physical point is given in the last column.

B	q	a_0	a_1	χ^2/dof	quality	$\langle x \rangle_{\Delta q}/Z$
N	u	0.371(20)	-0.002(20)	2.76	0.04	0.371(20)
N	d	-0.083(11)	0.003(11)	1.47	0.22	-0.083(12)
N	$u-d$	0.451(23)	-0.001(23)	1.95	0.12	0.451(23)
N	$u+d$	0.286(23)	0.001(23)	2.92	0.03	0.287(24)
Σ	u	0.324(17)	0.046(19)	2.43	0.06	0.332(18)
Σ	d	-0.1191(93)	0.0386(100)	2.09	0.10	-0.1126(95)
Σ	$u-d$	0.437(20)	0.014(21)	1.05	0.37	0.439(20)
Σ	$u+d$	0.203(20)	0.086(21)	3.82	0.01	0.218(20)
Ξ	u	0.433(15)	-0.063(17)	4.35	0.00	0.422(15)
Ξ	d	-0.0710(69)	-0.0105(83)	1.36	0.25	-0.0728(71)
Ξ	$u-d$	0.507(17)	-0.055(19)	2.91	0.03	0.497(17)
Ξ	$u+d$	0.364(16)	-0.076(19)	4.56	0.00	0.351(17)

C.7 $n = 2$ Moment of Tensor GPD

Table C.10 – Fit results for $\langle x \rangle^B_{\delta q}$ with the fit ansatz: $\langle x \rangle_{\delta q} = a_0 + a_1 m_\pi^2$. The extrapolation to the physical point is given in the last column.

B	q	a_0	a_1	χ^2/dof	quality	$\langle x \rangle_{\delta q}/Z$
N	u	0.224(11)	-0.019(11)	0.33	0.80	0.221(11)
N	d	-0.0440(52)	0.0021(49)	6.42	0.00	-0.0436(52)
N	$u-d$	0.266(11)	-0.018(11)	2.34	0.07	0.263(11)
N	$u+d$	0.182(12)	-0.018(12)	1.29	0.28	0.179(12)
Σ	u	0.1927(87)	0.0130(93)	0.40	0.75	0.1949(88)
Σ	d	-0.0456(42)	0.0034(46)	7.68	0.00	-0.0450(43)
Σ	$u-d$	0.2355(98)	0.013(10)	1.77	0.15	0.2376(100)
Σ	$u+d$	0.1448(94)	0.019(10)	2.16	0.09	0.1479(95)
Ξ	u	0.2213(70)	-0.0159(81)	1.34	0.26	0.2186(71)
Ξ	d	-0.0337(33)	-0.0088(39)	7.33	0.00	-0.0352(34)
Ξ	$u-d$	0.2564(78)	-0.0083(89)	1.92	0.12	0.2550(79)
Ξ	$u+d$	0.1898(77)	-0.0269(91)	3.55	0.01	0.1852(79)

Appendix D

Implementation Details

D.1 Operator Extension

PPU implementation

```
// PPU
struct BaseOp
{
  virtual void sendInfo()          const { }
  virtual bool getReadAccessMode() const { return false; }
  virtual      ~BaseOp(){}
};

struct FnPokeSpinVector: public BaseOp
{
  // ...

  FnPokeSpinVector(int _row): row(_row) {}

  bool getReadAccessMode() const
  {
    return true;
  }

  void sendInfo() const
  {
    for ( unsigned int spu = 0 ;  spu < getNSPU() ; ++spu )
      {
        spu_Mailbox_write_uint( spu , row );
      }
  }
private:
  int row;
};
```

SPU implementation

```
// SPU
struct BaseOp
{
  virtual void recvInfo()                { }
  virtual bool getReadAccessMode() const { return false; }
  virtual      ~BaseOp()                 { }
};

struct FnPokeSpinVector: public BaseOp
{
  // ...

  FnPokeSpinVector(int _row): row(_row) {}

  bool getReadAccessMode()
  {
    return true;
  }

  void recvInfo()
  {
    row = spu_readch( SPU_RdInMbox );
  }

private:
  int row;
};
```

D.2 Arithmetical Operations with Complex Numbers

Complex multiplication. (analog: adjoint-multiply, multiply-adjoint)

```
template<>
inline BinaryReturn<RComplex<REAL64>,
                    RComplex<REAL64>, OpMultiply >::Type_t
operator*(const RComplex<REAL64>& l, const RComplex<REAL64>& r)
{
  typedef BinaryReturn<RComplex<REAL64>,
                       RComplex<REAL64>, OpMultiply >::Type_t  Ret_t;
  typedef vector double T;
  const vector unsigned char swap  =
                        { 8,9,10,11,12,13,14,15,0,1,2,3,4,5,6,7 };
  const vector unsigned char alter =
                { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                  0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F};

  T c = spu_mul( l.F , r.F );
  c   = spu_sub( c , spu_shuffle( c , c , swap ) );
  T d = spu_mul( r.F , spu_shuffle( l.F , l.F , swap ) );
  d   = spu_add( d , spu_shuffle( d , d , swap ) );

  return Ret_t( spu_shuffle( c , d , alter ) );
}
```

Inner Product

```
template<>
inline BinaryReturn<RComplex<REAL64>,
                    RComplex<REAL64>, FnLocalInnerProduct >::Type_t
localInnerProduct(const RComplex<REAL64>& l,
                  const RComplex<REAL64>& r)
{
  typedef BinaryReturn<RComplex<REAL64>,
                       RComplex<REAL64>,
                       FnLocalInnerProduct >::Type_t  Ret_t;

  typedef vector double T;
  const vector unsigned char swap  =
                       { 8,9,10,11,12,13,14,15,0,1,2,3,4,5,6,7 };
  const vector unsigned char alter =
               { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F };

  T c = spu_mul( l.F , r.F );
  c   = spu_add( c , spu_shuffle( c , c , swap ) );
  T d = spu_mul( r.F , spu_shuffle( l.F , l.F , swap ) );
  d   = spu_sub( d , spu_shuffle( d , d , swap ) );

  return Ret_t( spu_shuffle( c , d , alter ) );
}
```

Times I (analog: times -I, complex conjugate)

```
template<>
inline UnaryReturn<RComplex<REAL64>, FnTimesI >::Type_t
timesI(const RComplex<REAL64>& s1)
{
  typedef UnaryReturn<RComplex<REAL64>, FnTimesI >::Type_t   Ret_t;

  const vector unsigned char swap  =
                     { 8,9,10,11,12,13,14,15,0,1,2,3,4,5,6,7 };
  const vector double help         = { -1.0 , 1.0 };

  Ret_t d( spu_mul( spu_shuffle( s1.F , s1.F , swap ) , help ) );
  return d;
}
```

Addition, Subtraction analog

```
template<>
inline BinaryReturn<RComplex<REAL64>,
                    RComplex<REAL64>, OpAdd>::Type_t
operator+(const RComplex<REAL64>& l, const RComplex<REAL64>& r)
{
  typedef BinaryReturn<RComplex<REAL64>,
                       RComplex<REAL64>, OpAdd>::Type_t  Ret_t;
  return Ret_t( spu_add( l.F , r.F ) );
}
```

D.3 Assignment Operators

Assignment Operators

```
template <class T, template<class,int> class C>
class PMatrix<T,3,C>
{
  // ...

  typedef C<T,3>  CC;

  template<class T1>
  inline
  CC& operator=(const C<T1,3>& rhs)
  {
    elem(0,0) = rhs.elem(0,0);
    elem(0,1) = rhs.elem(0,1);
    elem(0,2) = rhs.elem(0,2);
    elem(1,0) = rhs.elem(1,0);
    elem(1,1) = rhs.elem(1,1);
    elem(1,2) = rhs.elem(1,2);
    elem(2,0) = rhs.elem(2,0);
    elem(2,1) = rhs.elem(2,1);
    elem(2,2) = rhs.elem(2,2);
    return static_cast<CC&>(*this);
  }
};
```

D.4 Mailboxes

There are two different SPU mailboxes implemented with MFC channels.

- SPU Read Inbound Mailbox (SPU receives a message)
- SPU Write Outbound Mailbox (SPU sends a message)

These services are available at the SPE and the PPE. However, usage of the mailboxes differ if used from the SPE or from the PPE.

D.4.1 SPE side

The SPU Read Inbound Mailbox is used by reading the SPU Read Inbound Mailbox Channel. If the SPU Read Inbound Mailbox Channel has a message, the value read from the mailbox channel is the oldest message written to the mailbox. If the SPU Read Inbound Mailbox has no message and SPU software reads from the channel, the SPU will stall on the read. The SPU remains stalled until the PPE or other devices write a message to the mailbox by writing to the MMIO address of the mailbox.

```
{
  unsigned int msg;
  msg = spu_readch( SPU_RdInMbox );
}
```

The SPU Write Outbound Mailbox is used by writing to the SPU Write Outbound Mailbox Channel. This write-channel instruction will return immediately if there is sufficient space in the SPU write outbound mailbox queue to hold the message value. If there is insufficient space, the write-channel instruction will stall the SPU until the PPE or any other device reads from this mailbox.

```
{
  unsigned int msg;
  spu_writech( SPU_WrOutMbox , msg );
}
```

D.4.2 PPE side

Using mailboxes on the PPE is a bit more involved. The PPE cannot use directly SPU MFC channels, so it has to read and write to MMIO mapped MFC registers instead. These accesses are non-blocking. In order to avoid unwanted overwriting of data and reading of not yet available data PPE software queries MMIO mapped status registers of the SPU's MFC.

Before PPE software can read data from one of the SPU Write Outbound Mailboxes, it must first read the Mailbox Status Register to determine that unread data is present in the SPU Write Outbound Mailbox. If no data is available PPE software waits until data is available.

```
unsigned int spu_Mailbox_read_uint( int spu )
{
  do {
  } while ( ((ps[ spu ]->SPU_Mbox_Stat) & 0x000000FF) == 0);
  return ps[ spu ]->SPU_Out_Mbox;
}
```

Where **ps** is the *problem state* of the SPU, which is mapped at SPU initialisation time. Mapping the problem state is part of the standard procedure of SPU initialisation supported by the libspe2.

Using the SPU Read Inbound Mailbox from the PPE follows similar pattern. Since on the PPE writing to a full SPU Read Inbound Mailbox will not stall, PPE software first has to verify that the mailbox is not full. The fields of the SPU Mailbox Status Register can be queried in order to check for a full mailbox.

```
void spu_Mailbox_write_uint( int spu , unsigned int msg )
{
  unsigned int mb_status, slots;
  do {
    mb_status = ps[ spu ]->SPU_Mbox_Stat;
    slots = (mb_status & 0x0000FF00) >> 8;
  } while (slots == 0);
  ps[ spu ]->SPU_In_Mbox = msg;
}
```

Appendix E

The GNU C++ Compiler

E.1 Inline Parameters

Several parameters control the tree inliner used in GNU C++ Compiler. Given is an extract from the GNU C++ Compiler inline option description, the default values and the values used in order to enable the tree inliner to inline all functions into the evaluate loop.

The version number of the GNU C++ Compiler used here is 4.3.2.

max-inline-insns-single

This number sets the maximum number of instructions (counted in internal representation of GNU C++ Compiler) in a single function that the tree inliner will consider for inlining. This only affects functions declared inline and methods implemented in a class declaration (C++). The default value is 450.

Value used 6000.

large-function-insns

The limit specifying really large functions. For functions larger than this limit after inlining inlining is constrained by **–param large-function-growth**. This parameter is useful primarily to avoid extreme compilation time caused by non-linear algorithms used by the backend. The default value is 2700.

Value used 40000.

large-unit-insns

The limit specifying large translation unit. Growth caused by inlining of units larger than this limit is limited by **–param inline-unit-growth**. For small units this might be too tight (consider unit consisting of function A that is inline and B that just calls A three time. If B is small relative to A, the growth of unit is 300% and yet such inlining is very sane. For very large units consisting of small inlineable functions however the overall unit growth limit is needed to avoid exponential explosion of code size. Thus for smaller units, the size is increased to **–param large-unit-insns** before applying **–param inline-unit-growth**. The default is 10000

Value used 40000.

large-stack-frame

The limit specifying large stack frames. While inlining the algorithm is trying to avoid that the stack frame grows beyond this limit. Default value is 256 bytes.

Value used 1024.

large-stack-frame-growth

Specifies maximal growth of large stack frames caused by inlining in percents. The default value is 1000 which limits large stack frame growth to 11 times the original size.

Value used 10000.

Appendix F

Benchmark Measurements

F.1 QDP++ Functions

Table F.1 – SPU functions used for benchmark measurements. N_S number of elements in one transfer set, T_0 destination type, T_i leaf types in QDP++ expression $E_{\text{QDP++}}$.

n	N_S	size(T_0)	size(T_i)
1000	4	2304	2304
1001	4	16	2304 2304
1002	4	2304	2304
1003	32	192	192 192
1004	128	72	72
1005	64	144	72
1006	64	144	144
1007	64	144	144 144
1008	4	2304	2304
1009	512	8	4
1010	256	8	8
1011	256	16	8 8
1012	4	2304	144 2304
1013	2	2304	144 2304 2304
1014	2	2304	144 2304 2304

1015	4	2304	2304 2304
1016	4	2304	2304 2304
1017	2	16	2304 2304 2304 2304
1018	128	16	16
1019	4	16	2304 2304
1020	4	16	2304 2304
1021	4	16	2304 2304
1022	512	4	4
1023	64	192	48
1024	8	144	2304
1025	64	48	192
1026	64	144	48
1027	8	2304	144
1028	4	2304	2304
1029	2	2304	144 2304 2304
1030	128	144	4
1031	64	144	144 144
1032	64	144	144 144
1033	64	144	144
1034	64	144	144
1035	64	144	144
1036	64	144	144
1037	256	8	8
1038	128	48	144
1039	64	96	192
1040	64	96	96
1041	64	96	144 192
1042	64	96	192
1043	64	96	144 192
1044	64	96	192
1045	64	96	144 192

1046	64	96	192
1047	64	96	144 192
1048	16	192	144 96 96 144 96 96 144 96 96 144 96 96
1049	32	192	192 192
1050	32	192	192
1051	32	192	192
1052	32	192	192
1053	32	192	192
1054	512	4	4
1055	2048	1	4
1056	128	16	16 16
1057	128	16	16
1058	512	8	4
1059	32	192	192
1060	32	192	192 192
1061	64	144	144 144
1062	32	192	144 192

Appendix G

SPU Timing Analysis

G.1 Matrix Assignment

G.1.1 With Template Specialisation

The program version with template specialisation assigns a whole complex number with one load and store operation.

```
1 01                                            6789 stqx    $7,$21,$9
1 012                                            789 lqd     $75,912($sp)
1 ---345678                                       -- stqd    $75,16($79)
1     456789                                         lqd     $2,928($sp)
1      -----012345                                   stqd    $2,32($79)
1            123456                                  lqd     $3,944($sp)
1             -----789012                            stqd    $3,48($79)
1                   890123                           lqd     $12,960($sp)
1                    -----456789                     stqd    $12,64($79)
1                          567890                    lqd     $6,976($sp)
1                           -----123456              stqd    $6,80($79)
1                                 234567             lqd     $5,992($sp)
1                                  -----890123       stqd    $5,96($79)
1                                        901234      lqd     $80,1008($sp)
1 0                                       -----56789 stqd    $80,112($79)
1 01                                            6789 lqd     $78,1024($sp)
1 --234567                                       --- stqd    $78,128($79)
```

G.1.2 Without Template Specialisation

The SPU C++ Compiler produces inefficient code out of the generic assignment function. When the compiler assigns the matrix element-wise it has to resolve the next template level, i.e. the complex numbers. The real and imaginary parts get assigned separately which results in many load, shuffle, and store operations. The shuffle and rotate operations are emphasised.

```
 1 1            0                                        lnop
   0             12                                      il        $51,896
 3 1              234567                                 lqd       $82,1104($sp)
   0               34                                    a $83,$51,$sp
 5 1                4                                    lnop
                                                         L53:
 7 0                 56                                  ai        $13,$83,13
   1                  678901                             lqd       $45,0($83)
 9 0                   78                                a $44,$83,$111
   1                    890123                           lqd       $47,0($82)
11 0                     90                              a $6,$83,$112
   1                      0123                           cbd       $15,0($82)
13 0                       12                            ai        $51,$44,13
   1                        2345                         cbx       $18,$82,$111
15 0                         34                          ai        $58,$6,13
   1                          4567                       cbx       $20,$82,$112
17 0                           56                        a $8,$83,$96
   1                            6789                     cbx       $71,$82,$96
19 0                             7                       nop       127
   1                              8901                   rotqby    $46,$45,$13
21 0                               90                    ai        $68,$8,13
   1                                0123                 cbx       $28,$82,$86
23 0                                 12                  a $17,$83,$86
   1                                  2345               cbx       $30,$82,$95
25 0                                   34                a $49,$83,$113
   1                                    4567             cbx       $31,$82,$113
27 0                                     5               nop       127
   1                                      6789           shufb     $41,$46,$47,$15
29 0                                       78            ai        $24,$17,13
   1                                        8901         cbx       $34,$82,$114
31 0                                         90          ai        $9,$49,13
   1                                          0123       cbx       $39,$82,$115
33 0                                           12        a $50,$83,$114
   1                                            2345     cbx       $40,$82,$85
35 1                                             345678  stqd      $41,0($82)
   1                                              456789 lqx       $19,$83,$111
37 0                                               5     nop       127
   1  01                                            6789 lqx       $21,$82,$111
39 0                                                 78  ai        $37,$50,13
   1  01                                              89 cbx       $42,$82,$116
41 0  0                                                9 a $16,$83,$115
   1  0123                                               cbx       $43,$82,$117
43 0   12                                                a $52,$83,$85
   1    2345                                             cbx       $44,$82,$118
45 0     34                                              ai        $36,$16,13
   1      4567                                           cbx       $45,$82,$119
47 0       5                                             nop       127
   1        6789                                         rotqby    $53,$19,$51
49 0         78                                          ai        $11,$52,13
   1          8901                                       cbx       $46,$82,$120
51 0           90                                        a $54,$83,$116
   1            0123                                     cbx       $47,$82,$121
53 0             12                                      a $55,$83,$117
   1              2345                                   cbx       $49,$82,$122
55 0               3                                     nop       127
   1                4567                                 shufb     $22,$53,$21,$18
57 0                 56                                  ai        $4,$54,13
   1                  6789                               cbd       $50,0($82)
59 0                   78                                ai        $14,$55,13
   1                    8901                             cbx       $51,$82,$123
61 0                     90                              a $56,$83,$118
```

```
    1                      0123                           cbx       $52,$82,$124
 63 1                       123456                        stqx      $22,$82,$111
    1                        234567                       lqx       $59,$83,$112
 65 0                         3                           nop       127
    1                          456789                     lqx       $62,$82,$112
 67 0                           56                        ai        $12,$56,13
    1                            6789                     cbx       $53,$82,$125
 69 0                             78                      a $57,$83,$119
    1                              8901                   cbx       $54,$82,$126
 71 0                               90                    a $60,$83,$120
    1                                0123                 cbx       $55,$82,$127
 73 0                                 12                  ai        $13,$57,13
    1                                  2345               cbx       $56,$82,$81
 75 0                                   3                 nop       127
    1                                    4567             rotqby    $61,$59,$58
 77 0                                     56              ai        $15,$60,13
    1                                      6789           cbd       $57,7($82)
 79 0                                       78            a $63,$83,$121
    1                                        8901         cbx       $58,$82,$97
 81 0                                         90          a $64,$83,$122
    1                                          0123       cbx       $59,$82,$109
 83 0                                           1         nop       127
    1                                            2345     shufb     $65,$61,$62,$20
 85 0                                             34      ai        $6,$63,13
    1                                              4567   cbx       $60,$82,$110
 87 0                                               56    ai        $17,$64,13
    1                                                6789 cbx       $61,$82,$99
 89 0                                                 78  ai        $41,$83,13
    1  01                                              89 cbx       $62,$82,$100
 91 1  01234                                            9 stqx      $65,$82,$112
    1  012345                                             lqx       $69,$83,$96
 93 0   1                                                 nop       127
    1    234567                                           lqx       $73,$82,$96
 95 0     34                                              a $66,$83,$123
    1      4567                                           cbx       $63,$82,$101
 97 0       56                                            a $67,$83,$124
    1        6789                                         cbx       $64,$82,$102
 99 0         78                                          ai        $16,$66,13
    1          8901                                       cbx       $65,$82,$103
101 0           90                                        ori       $66,$50,0
    1            0123                                     rotqby    $72,$69,$68
103 0             12                                      ai        $19,$67,13
    1              2345                                   cbx       $67,$82,$104
105 0               34                                    a $70,$83,$125
    1                4567                                 cbx       $68,$82,$105
107 0                 56                                  a $48,$83,$102
    1                  6789                               cbx       $69,$82,$106
109 0                   7                                 nop       127
    1                    8901                             shufb     $75,$72,$73,$71
111 0                     90                              ai        $18,$70,13
    1                      0123                           cbx       $70,$82,$98
113 0                       12                            a $74,$83,$126
    1                        2345                         cbx       $71,$82,$107
115 0                         34                          a $76,$83,$127
    1                          4567                       cbx       $72,$82,$108
117 1                           567890                    stqx      $75,$82,$96
    1                            678901                   lqx       $78,$83,$86
119 1                             789012                  lqx       $29,$82,$86
    1                              890123                 stqd      $30,1392($sp)
121 0                               90                    ai        $21,$74,13
    1                                0123                 cbx       $73,$82,$87
123 0                                 12                  ai        $8,$76,13
    1                                  2345               cbx       $74,$82,$88
125 0                                   34                a $77,$83,$81
    1                                    4567             cbx       $75,$82,$89
```

```
127 0                                     5               nop       127
    1                                      6789           rotqby    $2,$78,$24
129 0                                       78            ai        $22,$77,13
    1                                        8901         cbx       $76,$82,$90
131 0                                         90          ai        $79,$83,23
    1                                          0123       cbx       $77,$82,$91
133 0                                           12        a $80,$83,$97
    1                                            2345     cbx       $78,$82,$92
135 0                                             3       nop       127
    1                                              4567   shufb     $38,$2,$29,$28
137 0                                               56    ai        $20,$79,13
    1                                                6789 cbx       $79,$82,$93
139 0                                                 78  ai        $24,$80,13
    1  01                                              89 cbx       $80,$82,$94
141 0  0                                                9 a $5,$83,$109
    1  012345                                             stqx      $38,$82,$86
143 1   123456                                            lqx       $35,$83,$113
    1    234567                                           lqx       $32,$82,$113
145 0     34                                              ai        $5,$5,13
    0      45                                             a $23,$83,$110
147 0       56                                            a $25,$83,$99
    0d       67                                           ai        $23,$23,13
149 1d       -7890                                        rotqby    $3,$35,$9
    0          89                                         ai        $9,$48,13
151 0           90                                        ori       $48,$41,0
    0d           01                                       ai        $25,$25,13
153 1d           -1234                                    shufb     $33,$3,$32,$31
    0              23                                     a $26,$83,$100
155 0               34                                    a $27,$83,$101
    0d               45                                   ai        $26,$26,13
157 1d               -567890                              stqx      $33,$82,$113
    1                  678901                             lqx       $2,$83,$114
159 1                   789012                            lqx       $35,$82,$114
    0                    89                               ai        $27,$27,13
161 0                     90                              a $28,$83,$103
    0                      01                             a $29,$83,$104
163 0                       12                            ai        $28,$28,13
    1                        2345                         rotqby    $38,$2,$37
165 0                         34                          ai        $29,$29,13
    0                          45                         a $30,$83,$105
167 0                           56                        a $7,$83,$106
    1                            6789                     shufb     $3,$38,$35,$34
169 0                             78                      ai        $30,$30,13
    0                              89                     ai        $7,$7,13
171 0                               90                    a $31,$83,$98
    1                                012345               stqx      $3,$82,$114
173 1                                 123456              lqx       $37,$83,$115
    1                                  234567             lqx       $38,$82,$115
175 0                                   34                ai        $31,$31,13
    0                                    45               a $32,$83,$107
177 0                                     56              a $33,$83,$108
    0d                                     67             ai        $32,$32,13
179 1d                                     -7890          rotqby    $2,$37,$36
    0                                        89           ai        $33,$33,13
181 0                                         90          a $34,$83,$87
    0d                                         01         a $10,$83,$88
183 1d                                         -1234      shufb     $3,$2,$38,$39
    0                                            23       ai        $34,$34,13
185 0                                             34      ai        $10,$10,13
    0d                                             45     a $35,$83,$89
187 1d 0                                           -56789 stqx      $3,$82,$115
    1  01                                            6789 lqx       $39,$83,$85
189 1  012                                            789 lqx       $3,$82,$85
    0                                                  89 ai        $35,$35,13
191 0  0                                                9 a $36,$83,$90
```

```
    0  01                                                 a $37,$83,$91
193 0   12                                                ai        $36,$36,13
    1    2345                                             rotqby    $2,$39,$11
195 0     34                                              ai        $37,$37,13
    0      45                                             a $38,$83,$92
197 0       56                                            a $11,$83,$93
    1        6789                                         shufb     $3,$2,$3,$40
199 0         78                                          ai        $38,$38,13
    0          89                                         ai        $11,$11,13
201 0           90                                        a $39,$83,$94
    1            012345                                   stqx      $3,$82,$85
203 1             123456                                  lqx       $2,$83,$116
    1              234567                                 lqx       $3,$82,$116
205 0               34                                    ai        $39,$39,13
    0                45                                   a $40,$83,$95
207 0                 -67                                 ai        $40,$40,13
    1                   7890                              rotqby    $2,$2,$4
209 1                    ---1234                          shufb     $4,$2,$3,$42
    1                        ---567890                    stqx      $4,$82,$116
211 1                            678901                   lqx       $2,$83,$117
    1                             789012                  lqx       $42,$82,$117
213 1                              ----2345               rotqby    $3,$2,$14
    1                                   ---6789           shufb     $4,$3,$42,$43
215 1                                       ---012345     stqx      $4,$82,$117
    1                                           123456    lqx       $2,$83,$118
217 1                                            234567   lqx       $14,$82,$118
    1  0                                          ----789 rotqby    $3,$2,$12
219 1  -1234                                          --  shufb     $42,$3,$14,$44
    1    ---567890                                        stqx      $42,$82,$118
221 1        678901                                       lqx       $44,$83,$119
    1         789012                                      lqx       $12,$82,$119
223 1          ----2345                                   rotqby    $43,$44,$13
    1               ---6789                               shufb     $13,$43,$12,$45
225 1                   ---012345                         stqx      $13,$82,$119
    1                       123456                        lqx       $4,$83,$120
227 1                        234567                       lqx       $3,$82,$120
    1                         ----7890                    rotqby    $2,$4,$15
229 1                              ---1234                shufb     $14,$2,$3,$46
    1                                  ---567890          stqx      $14,$82,$120
231 1                                      678901         lqx       $46,$83,$121
    1                                       789012        lqx       $45,$82,$121
233 1                                        ----2345     rotqby    $42,$46,$6
    1                                             ---6789 shufb     $44,$42,$45,$47
235 1  012345                                         --  stqx      $44,$82,$121
    1   123456                                            lqx       $43,$83,$122
237 1    234567                                           lqx       $13,$82,$122
    1     ----7890                                        rotqby    $12,$43,$17
239 1          ---1234                                    shufb     $17,$12,$13,$49
    1              ---567890                              stqx      $17,$82,$122
241 1                  678901                             lqd       $4,16($83)
    1                   789012                            lqd       $3,16($82)
243 1                    ----2345                         rotqby    $15,$4,$41
    1                         ---6789                     shufb     $6,$15,$3,$50
245 1                             ---012345               stqd      $6,16($82)
    1                                 123456              lqx       $2,$83,$123
247 1                                  234567             lqx       $50,$82,$123
    1                                   ----7890          rotqby    $14,$2,$16
249 1                                        ---1234      shufb     $49,$14,$50,$51
    0d                                           23       il        $50,23
251 1d 0                                         ---56789 stqx      $49,$82,$123
    1  01                                            6789 lqx       $47,$83,$124
253 1  012                                            789 lqx       $42,$82,$124
    1  --2345                                         --  rotqby    $46,$47,$19
255 1     ---6789                                         shufb     $45,$46,$42,$52
    1         ---012345                                   stqx      $45,$82,$124
```

```
257 1            123456                                  lqx     $44,$83,$125
    1             234567890123456                        hbrr    .L192,.L53
259 1              345678                                lqx     $43,$82,$125
    1               ---7890                              rotqby  $41,$44,$18
261 1                   ---1234                          shufb   $12,$41,$43,$53
    1                       ---567890                    stqx    $12,$82,$125
263 1                           678901                   lqx     $13,$83,$126
    1                            789012                  lqx     $18,$82,$126
265 1                             ----2345               rotqby  $19,$13,$21
    1                                  ---6789           shufb   $17,$19,$18,$54
267 0                                      78            il      $54,23
    1                                       --012345     stqx    $17,$82,$126
269 1                                          123456    lqx     $4,$83,$127
    1                                           234567   lqx     $3,$82,$127
271 1 0                                          ----789 rotqby  $15,$4,$8
    1 -1234                                           -- shufb   $6,$15,$3,$55
273 1   ---567890                                        stqx    $6,$82,$127
    1       678901                                       lqx     $2,$83,$81
275 1        789012                                      lqx     $14,$82,$81
    1         ----2345                                   rotqby  $16,$2,$22
277 1              ---6789                               shufb   $55,$16,$14,$56
    1                  ---012345                         stqx    $55,$82,$81
279 1                      123456                        lqx     $53,$83,$54
    1                       234567                       lqx     $51,$82,$54
281 1                        ----7890                    rotqby  $52,$53,$20
    1                             ---1234                shufb   $49,$52,$51,$57
283 1                                 ---567890          stqx    $49,$82,$50
    1                                     678901         lqx     $47,$83,$97
285 1                                      789012        lqx     $42,$82,$97
    1                                       ----2345     rotqby  $46,$47,$24
287 1                                            ---6789 shufb   $45,$46,$42,$58
    1 012345                                         --- stqx    $45,$82,$97
289 1  123456                                            lqx     $44,$83,$109
    1   234567                                           lqx     $43,$82,$109
291 1    ----7890                                        rotqby  $41,$44,$5
    1         ---1234                                    shufb   $24,$41,$43,$59
293 1             ---567890                              stqx    $24,$82,$109
    1                 678901                             lqx     $22,$83,$110
295 1                  789012                            lqx     $8,$82,$110
    1                   ----2345                         rotqby  $21,$22,$23
297 1                        ---6789                     shufb   $12,$21,$8,$60
    1                            ---012345               stqx    $12,$82,$110
299 1                                123456              lqx     $13,$83,$99
    1                                 234567             lqx     $19,$82,$99
301 1                                  ----7890          rotqby  $20,$13,$25
    1                                       ---1234      shufb   $18,$20,$19,$61
303 1 0                                         ---56789 stqx    $18,$82,$99
    1 01                                            6789 lqx     $17,$83,$100
305 1 012                                            789 lqx     $15,$82,$100
    1 --2345                                          -- rotqby  $4,$17,$26
307 1    ---6789                                         shufb   $3,$4,$15,$62
    1        ---012345                                   stqx    $3,$82,$100
309 1            123456                                  lqx     $6,$83,$101
    1             234567                                 lqx     $5,$82,$101
311 1              ----7890                              rotqby  $2,$6,$27
    1                   ---1234                          shufb   $16,$2,$5,$63
313 1                       ---567890                    stqx    $16,$82,$101
    1                           678901                   lqx     $63,$83,$102
315 1                            789012                  lqx     $61,$82,$102
    1                             ----2345               rotqby  $62,$63,$9
317 1                                  ---6789           shufb   $60,$62,$61,$64
    1                                      ---012345     stqx    $60,$82,$102
319 1                                          123456    lqx     $59,$83,$103
    1                                           234567   lqx     $14,$82,$103
321 1 0                                          ----789 rotqby  $58,$59,$28
```

```
    1 -1234                                           -- shufb   $57,$58,$14,$65
323 1   ---567890                                        stqx    $57,$82,$103
    1       678901                                       lqd     $56,32($83)
325 1        789012                                      lqd     $54,32($82)
    1         ----2345                                   rotqby  $55,$56,$48
327 1              ---6789                               shufb   $53,$55,$54,$66
    1                  ---012345                         stqd    $53,32($82)
329 1                      123456                        lqx     $52,$83,$104
    1                       234567                       lqx     $50,$82,$104
331 1                        ----7890                    rotqby  $51,$52,$29
    1                             ---1234                shufb   $49,$51,$50,$67
333 1                                 ---567890          stqx    $49,$82,$104
    1                                     678901         lqx     $48,$83,$105
335 1                                      789012        lqx     $46,$82,$105
    1                                       ----2345     rotqby  $47,$48,$30
337 1                                            ---6789 shufb   $42,$47,$46,$68
    1 012345                                         --- stqx    $42,$82,$105
339 1  123456                                            lqx     $45,$83,$106
    1   234567                                           lqx     $41,$82,$106
341 1    ----7890                                        rotqby  $44,$45,$7
    1         ---1234                                    shufb   $43,$44,$41,$69
343 1             ---567890                              stqx    $43,$82,$106
    1                 678901                             lqx     $30,$83,$98
345 1                  789012                            lqx     $27,$82,$98
    1                   ----2345                         rotqby  $29,$30,$31
347 1                        ---6789                     shufb   $28,$29,$27,$70
    1                            ---012345               stqx    $28,$82,$98
349 1                                123456              lqx     $26,$83,$107
    1                                 234567             lqx     $24,$82,$107
351 1                                  ----7890          rotqby  $25,$26,$32
    1                                       ---1234      shufb   $23,$25,$24,$71
353 1 0                                         ---56789 stqx    $23,$82,$107
    1 01                                            6789 lqx     $7,$83,$108
355 1 012                                            789 lqx     $21,$82,$108
    1 --2345                                          -- rotqby  $22,$7,$33
357 1    ---6789                                         shufb   $8,$22,$21,$72
    1        ---012345                                   stqx    $8,$82,$108
359 1            123456                                  lqx     $12,$83,$87
    1             234567                                 lqx     $20,$82,$87
361 1              ----7890                              rotqby  $13,$12,$34
    1                   ---1234                          shufb   $19,$13,$20,$73
363 1                       ---567890                    stqx    $19,$82,$87
    1                           678901                   lqx     $18,$83,$88
365 1                            789012                  lqx     $4,$82,$88
    1                             ----2345               rotqby  $17,$18,$10
367 1                                  ---6789           shufb   $15,$17,$4,$74
    1                                      ---012345     stqx    $15,$82,$88
369 1                                          123456    lqx     $3,$83,$89
    1                                           234567   lqx     $9,$82,$89
371 1 0                                          ----789 rotqby  $6,$3,$35
    1 -1234                                           -- shufb   $10,$6,$9,$75
373 1   ---567890                                        stqx    $10,$82,$89
    1       678901                                       lqx     $2,$83,$90
375 1        789012                                      lqx     $16,$82,$90
    1         ----2345                                   rotqby  $5,$2,$36
377 1              ---6789                               shufb   $74,$5,$16,$76
    1                  ---012345                         stqx    $74,$82,$90
379 1                      123456                        lqx     $73,$83,$91
    1                       234567                       lqx     $71,$82,$91
381 1                        ----7890                    rotqby  $72,$73,$37
    1                             ---1234                shufb   $70,$72,$71,$77
383 1                                 ---567890          stqx    $70,$82,$91
    1                                     678901         lqx     $69,$83,$92
385 1                                      789012        lqx     $67,$82,$92
    1                                       ----2345     rotqby  $68,$69,$38
```

```
1                                             ---6789 shufb     $66,$68,$67,$78
1  012345                                          -- stqx      $66,$82,$92
1   123456                                            lqx       $65,$83,$93
1    234567                                           lqx       $63,$82,$93
1     ----7890                                        rotqby    $64,$65,$11
1          ---1234                                    shufb     $62,$64,$63,$79
1              ---567890                              stqx      $62,$82,$93
1                  678901                             lqx       $61,$83,$94
1                   789012                            lqx       $59,$82,$94
1                    ----2345                         rotqby    $60,$61,$39
1                         ---6789                     shufb     $58,$60,$59,$80
1                             ---012345               stqx      $58,$82,$94
0                                 1                   nop       127
1                                  234567             lqx       $57,$83,$95
0                                   34                ai        $83,$83,48
1                                    456789           lqx       $54,$82,$95
1                                     567890          lqd       $56,1392($sp)
1                                      678901         lqd       $14,1616($sp)
1                                       -8901         rotqby    $55,$57,$40
0D                                        ---23       ceq       $52,$83,$14
1D                                           2345     shufb     $53,$55,$54,$56
1  01                                         ---6789 stqx      $53,$82,$95
0                                                 78  ai        $82,$82,48
                                                      L192:
1  01                                              89 brz       $52,.L53
1  01234567890123                                   9 hbrr      .L191,.L54
```

G.2 Matrix Multiplication

G.2.1 With Template Specialisation

The program part for the multiplication of two colour matrices with template specialisation does not contain any loop. It contains 27 complex multiplications each of which involves shuffle operations. Optimisations beyond the level of loop unrolling are not carried out. This program part executes in roughly 300 machine cycles.

```
1                         345678                      lqx       $21,$15,$23
1                          456789                     lqd       $75,16($17)
1                           567890                    lqx       $10,$13,$23
1                            678901                   lqd       $18,48($14)
1                             789012                  lqd       $7,32($17)
1                              890123                 lqd       $4,96($14)
1                               9012                  shufb     $2,$21,$21,$24
1                                0123                 shufb     $16,$75,$75,$24
0                                 123456789           dfm       $76,$21,$10
0                                  234567890          dfm       $3,$75,$18
0                                   345678901         dfm       $19,$10,$2
0                                    456789012        dfm       $6,$18,$16
0D                                    567890123       dfm       $78,$7,$4
1D                                    5678            shufb     $74,$7,$7,$24
1                                      ----0123       shufb     $8,$76,$76,$24
1                                           1234      shufb     $83,$3,$3,$24
0D 0                                         23456789 dfm       $20,$4,$74
1D                                           2345     shufb     $80,$19,$19,$24
```

```
    0  012                                         -456789 dfs       $75,$76,$8
20  0  0123                                          56789 dfs       $16,$3,$83
    1                                                 6789 shufb     $11,$6,$6,$24
22  0  012345                                          789 dfa       $5,$19,$80
    1  01                                               89 shufb     $79,$78,$78,$24
24  1  -1234                                             - shufb     $9,$20,$20,$24
    0    234567890                                         dfa       $10,$6,$11
26  0     345678901                                        dfs       $2,$78,$79
    0d     -567890123                                      dfa       $18,$20,$9
28  1d      -6789                                          shufb     $74,$75,$5,$25
    1         ----1234                                     shufb     $7,$16,$10,$25
30  1              --4567                                  shufb     $8,$2,$18,$25
    0                 567890123                            dfa       $21,$74,$7
32  0                  ---------456789012                  dfa       $74,$21,$8
    1                           ---------345678            stqd      $74,1168($sp)
34  1                                   456789             lqx       $76,$15,$23
    1                                    567890            lqd       $83,16($17)
36  1                                     678901           lqd       $19,16($14)
    1                                      789012          lqd       $6,64($14)
38  1                                       890123         lqd       $11,32($17)
    1                                        901234        lqd       $4,112($14)
40  1                                         0123         shufb     $80,$76,$76,$24
    1                                          1234        shufb     $79,$83,$83,$24
42  0  0                                          23456789 dfm       $20,$76,$19
    0  01                                          3456789 dfm       $3,$83,$6
44  0  012                                          456789 dfm       $5,$19,$80
    0  0123                                          56789 dfm       $18,$6,$79
46  0  01234                                          6789 dfm       $2,$11,$4
    1  0                                               789 shufb     $78,$11,$11,$24
48  1  -1234                                            -- shufb     $9,$20,$20,$24
    1    2345                                              shufb     $75,$3,$3,$24
50  0     345678901                                        dfm       $76,$4,$78
    1      4567                                            shufb     $10,$5,$5,$24
52  0       567890123                                      dfs       $80,$20,$9
    0D       678901234                                     dfs       $79,$3,$75
54  1D       6789                                          shufb     $16,$18,$18,$24
    0D        -890123456                                   dfa       $83,$5,$10
56  1D         8901                                        shufb     $7,$2,$2,$24
    1           ---2345                                    shufb     $8,$76,$76,$24
58  0               345678901                              dfa       $19,$18,$16
    0                456789012                             dfs       $78,$2,$7
60  0                 -678901234                           dfa       $11,$76,$8
    1                   7890                               shufb     $75,$80,$83,$25
62  1                    ----2345                          shufb     $9,$79,$19,$25
    1                         --5678                       shufb     $10,$78,$11,$25
64  0                            678901234                 dfa       $20,$75,$9
    0d                            ---------567890123       dfa       $75,$20,$10
66  1d                                    ----------456789 stqd      $75,1184($sp)
    1  0                                             56789 lqx       $4,$15,$23
68  1  01                                             6789 lqd       $5,16($17)
    1  012                                             789 lqd       $6,32($14)
70  1  0123                                             89 lqd       $76,80($14)
    1  01234                                             9 lqd       $83,32($17)
72  1  012345                                              lqd       $80,128($14)
    1   1234                                               shufb     $16,$4,$4,$24
74  1    2345                                              shufb     $18,$5,$5,$24
    0     345678901                                        dfm       $19,$4,$6
76  0      456789012                                       dfm       $3,$5,$76
    0       567890123                                      dfm       $79,$6,$16
78  0        678901234                                     dfm       $78,$76,$18
    0D        789012345                                    dfm       $15,$83,$80
80  1D        7890                                         shufb     $7,$83,$83,$24
    1          ----2345                                    shufb     $8,$19,$19,$24
82  1               3456                                   shufb     $2,$3,$3,$24
    0D               456789012                             dfm       $4,$80,$7
```

```
 84 1D               4567                                shufb  $12,$79,$79,$24
    0                 -678901234                         dfs    $16,$19,$8
 86 0                  789012345                         dfs    $18,$3,$2
    1                   8901                             shufb  $11,$78,$78,$24
 88 0                    901234567                       dfa    $5,$79,$12
    1                     0123                           shufb  $9,$15,$15,$24
 90 1                      --3456                        shufb  $10,$4,$4,$24
    0                         456789012                  dfa    $6,$78,$11
 92 0                          567890123                 dfs    $83,$15,$9
    0d                        -789012345                 dfa    $76,$4,$10
 94 1d                         -8901                     shufb  $80,$16,$5,$25
    1                            ----3456                shufb  $7,$18,$6,$25
 96 1                                 --6789             shufb  $8,$83,$76,$25
    0                                      789012345     dfa    $19,$80,$7
 98 0  01234                                 --------6789 dfa    $80,$19,$8
    1  -----567890                                    --- stqd   $80,1200($sp)
100 1        678901                                      lqd    $2,48($17)
    1         789012                                     lqd    $3,64($17)
102 1          890123                                    lqx    $79,$13,$23
    1           901234                                   lqd    $15,48($14)
104 1            012345                                  lqd    $16,80($17)
    1             123456                                 lqd    $4,96($14)
106 1              2345                                  shufb  $12,$2,$2,$24
    1               3456                                 shufb  $78,$3,$3,$24
108 0                456789012                           dfm    $18,$2,$79
    0                 567890123                          dfm    $76,$3,$15
110 0                  678901234                         dfm    $5,$79,$12
    0                   789012345                        dfm    $6,$15,$78
112 0                    890123456                       dfm    $2,$16,$4
    1                     9012                           shufb  $9,$16,$16,$24
114 1                      ---3456                       shufb  $10,$18,$18,$24
    1                          4567                      shufb  $11,$76,$76,$24
116 0                           567890123                dfm    $79,$4,$9
    1                            6789                    shufb  $83,$5,$5,$24
118 0                             789012345              dfs    $15,$18,$10
    0D                             890123456             dfs    $3,$76,$11
120 1D                             8901                  shufb  $7,$6,$6,$24
    0D                              -012345678           dfa    $78,$5,$83
122 1D                               0123                shufb  $8,$2,$2,$24
    1                                 ---4567            shufb  $12,$79,$79,$24
124 0                                     567890123      dfa    $16,$6,$7
    0                                      678901234     dfs    $10,$2,$8
126 0                                      -890123456    dfa    $9,$79,$12
    1                                        9012        shufb  $76,$15,$78,$25
128 1                                         ----4567   shufb  $11,$3,$16,$25
    1  0                                            --789 shufb  $83,$10,$9,$25
130 0  0123456                                          89 dfa    $18,$76,$11
    0d -------789012345                                  - dfa    $76,$18,$83
132 1d        ---------678901                            stqd   $76,1216($sp)
    1                   789012                           lqd    $6,48($17)
134 1                    890123                          lqd    $7,64($17)
    1                     901234                         lqd    $5,16($14)
136 1                      012345                        lqd    $4,64($14)
    1                       123456                       lqd    $12,80($17)
138 1                        234567                      lqd    $78,112($14)
    1                         3456                       shufb  $2,$6,$6,$24
140 1                          4567                      shufb  $8,$7,$7,$24
    0                           567890123                dfm    $16,$6,$5
142 0                            678901234               dfm    $3,$7,$4
    0                             789012345              dfm    $11,$5,$2
144 0                              890123456             dfm    $6,$4,$8
    0D                              901234567            dfm    $2,$12,$78
146 1D                              9012                 shufb  $79,$12,$12,$24
    1                                ----4567            shufb  $15,$16,$16,$24
148 1                                    5678            shufb  $9,$3,$3,$24
```

```
0D                                    678901234      dfm       $4,$78,$79
1D                                    6789           shufb     $10,$11,$11,$24
0                                      -890123456    dfs       $12,$16,$15
0                                       901234567    dfs       $78,$3,$9
1                                        0123        shufb     $83,$6,$6,$24
0                                         123456789  dfa       $5,$11,$10
1                                          2345      shufb     $7,$2,$2,$24
1                                           --5678   shufb     $8,$4,$4,$24
0  01234                                       6789  dfa       $79,$6,$83
0  012345                                       789  dfs       $9,$2,$7
0d 01234567                                      -9  dfa       $15,$4,$8
1d 0123                                           -  shufb     $10,$12,$5,$25
1   ----5678                                         shufb     $3,$78,$79,$25
1        --8901                                      shufb     $11,$9,$15,$25
0           901234567                                dfa       $16,$10,$3
0            ---------890123456                      dfa       $16,$16,$11
1                     ---------789012                stqd      $16,1232($sp)
1                             890123                 lqd       $83,48($17)
1                              901234                lqd       $6,64($17)
1                               012345               lqd       $5,32($14)
1                                123456              lqd       $7,80($14)
1                                 234567             lqd       $4,80($17)
1                                  345678            lqd       $79,128($14)
1                                   4567             shufb     $2,$83,$83,$24
1                                    5678            shufb     $8,$6,$6,$24
0                                     678901234      dfm       $15,$83,$5
0                                      789012345     dfm       $3,$6,$7
0                                       890123456    dfm       $11,$5,$2
0                                        901234567   dfm       $6,$7,$8
0                                         012345678  dfm       $2,$4,$79
1                                          1234      shufb     $12,$4,$4,$24
1  0123456                                  23456789 hbrr      .L195,.L103
1                                            --5678  shufb     $78,$15,$15,$24
1                                               6789 shufb     $9,$3,$3,$24
0D 012345                                        789 dfm       $4,$79,$12
1D 0                                             789 shufb     $10,$11,$11,$24
0  01234567                                       -9 dfs       $12,$15,$78
0  012345678                                         dfs       $78,$3,$9
1   1234                                             shufb     $83,$6,$6,$24
0    234567890                                       dfa       $5,$11,$10
1     3456                                           shufb     $7,$2,$2,$24
1      --6789                                        shufb     $8,$4,$4,$24
0          789012345                                 dfa       $79,$6,$83
0           890123456                                dfs       $9,$2,$7
0d           -012345678                              dfa       $11,$4,$8
1d            -1234                                  shufb     $83,$12,$5,$25
1              ----6789                              shufb     $3,$78,$79,$25
1                   --9012                           shufb     $10,$9,$11,$25
0                     012345678                      dfa       $15,$83,$3
0                      ---------901234567            dfa       $83,$15,$10
1                               ---------890123      stqd      $83,1248($sp)
1                                       901234       lqd       $12,96($17)
1                                        012345      lqd       $6,112($17)
1                                         123456     lqx       $5,$13,$23
1                                          234567    lqd       $7,48($14)
1                                           345678   lqd       $79,128($17)
1                                            456789  lqd       $4,96($14)
1                                             5678   shufb     $2,$12,$12,$24
1                                              6789  shufb     $8,$6,$6,$24
0  012345                                        789 dfm       $11,$12,$5
0  0123456                                        89 dfm       $3,$6,$7
0  01234567                                        9 dfm       $12,$5,$2
0  012345678                                         dfm       $6,$7,$8
0   123456789                                        dfm       $2,$79,$4
1    2345                                            shufb     $78,$79,$79,$24
```

```
214 1     ---6789                                         shufb  $13,$11,$11,$24
    1         7890                                        shufb  $9,$3,$3,$24
216 0          890123456                                  dfm    $79,$4,$78
    1           9012                                      shufb  $10,$12,$12,$24
218 0             012345678                               dfs    $78,$11,$13
    0D             123456789                              dfs    $13,$3,$9
220 1D             1234                                   shufb  $9,$6,$6,$24
    0D              -345678901                            dfa    $5,$12,$10
222 1D               3456                                 shufb  $7,$2,$2,$24
    1                 ---7890                             shufb  $8,$79,$79,$24
224 0                     890123456                       dfa    $6,$6,$9
    0                      901234567                      dfs    $12,$2,$7
226 0                       -123456789                    dfa    $4,$79,$8
    1                         2345                        shufb  $79,$78,$5,$25
228 1                        ----7890                     shufb  $9,$13,$6,$25
    1                             --0123                  shufb  $10,$12,$4,$25
230 0                                123456789            dfa    $11,$79,$9
    0d                               --------012345678    dfa    $79,$11,$10
232 1d  01234                                   ---------9 stqd   $79,1264($sp)
    1   012345                                            lqd    $3,96($17)
234 1    123456                                           lqd    $5,112($17)
    1     234567                                          lqd    $78,16($14)
236 1      345678                                         lqd    $6,64($14)
    1       456789                                        lqd    $13,128($17)
238 1        567890                                       lqd    $4,112($14)
    1         6789                                        shufb  $2,$3,$3,$24
240 1          7890                                       shufb  $7,$5,$5,$24
    0           890123456                                 dfm    $9,$3,$78
242 0            901234567                                dfm    $3,$5,$6
    0             012345678                               dfm    $5,$78,$2
244 0              123456789                              dfm    $78,$6,$7
    0D              234567890                             dfm    $2,$13,$4
246 1D              2345                                  shufb  $12,$13,$13,$24
    1                ----7890                             shufb  $8,$9,$9,$24
248 1                     8901                            shufb  $10,$3,$3,$24
    0d                     901234567                      dfm    $4,$4,$12
250 1d                     -0123                          shufb  $13,$78,$78,$24
    0                        123456789                    dfs    $9,$9,$8
252 0                         234567890                   dfs    $3,$3,$10
    1                          3456                       shufb  $7,$5,$5,$24
254 0                           456789012                 dfa    $6,$78,$13
    1                            5678                     shufb  $12,$2,$2,$24
256 1                             --8901                  shufb  $8,$4,$4,$24
    0                                901234567            dfa    $5,$5,$7
258 0                                 012345678           dfs    $10,$2,$12
    0d                                 -234567890         dfa    $7,$4,$8
260 1d                                  -3456             shufb  $13,$3,$6,$25
    1                                     ----8901        shufb  $78,$9,$5,$25
262 1                                          --1234     shufb  $12,$10,$7,$25
    0   0                                         23456789 dfa    $9,$78,$13
264 0   -123456789                                 ------- dfa    $78,$9,$12
    1     --------012345                                  stqd   $78,1280($sp)
266 1              123456                                 lqd    $2,96($17)
    1               234567                                lqd    $4,112($17)
268 1                345678                               lqd    $6,32($14)
    1                 456789                              lqd    $7,80($14)
270 1                  567890                             lqd    $5,128($14)
    1                   678901                            lqd    $8,128($17)
272 1                    7890                             shufb  $3,$2,$2,$24
    1                     8901                            shufb  $14,$4,$4,$24
274 0                      901234567                      dfm    $17,$2,$6
    0                       012345678                     dfm    $4,$4,$7
276 0                        123456789                    dfm    $6,$6,$3
    0                         234567890                   dfm    $7,$7,$14
278 0                          345678901                  dfm    $3,$8,$5
```

```
1                             4567                      shufb     $10,$8,$8,$24
1                              ---8901                  shufb     $13,$17,$17,$24
1                                  9012                 shufb     $12,$4,$4,$24
0                                   012345678           dfm       $5,$5,$10
1                                    1234               shufb     $14,$6,$6,$24
0                                     234567890         dfs       $2,$17,$13
0D                                     345678901        dfs       $17,$4,$12
1D                                     3456             shufb     $8,$7,$7,$24
0D                                      -567890123      dfa       $12,$6,$14
1D                                       5678           shufb     $10,$3,$3,$24
1                                         ---9012       shufb     $13,$5,$5,$24
0                                             012345678 dfa       $7,$7,$8
0                                              123456789 dfs      $3,$3,$10
0   01                                          -3456789 dfa      $5,$5,$13
1                                                 4567  shufb     $2,$2,$12,$25
1   012                                          ----9  shufb     $4,$17,$7,$25
1   --2345                                              shufb     $14,$3,$5,$25
0      345678901                                        dfa       $6,$2,$4
0d      ---------234567890                              dfa       $17,$6,$14
1d              ----------123456                        stqd      $17,1296($sp)
```

G.2.2 Without Template Specialisation

Without template specialisation the C++ compiler produces 2 nested loops to carry out the matrix multiplication. The inner-most loop body contains roughly 70 machine cycles. In order to realise the complex multiplication the real and imaginary parts need to be accessed, i.e. sub-quad word access is necessary. This results in rotate and shuffle operations. In order to multiply the two colour matrices the inner loop body is executed 9 times. For better visibility we emphasised the rotate operations, most of which (not the first two appearing in the code) result from accessing the real and imaginary parts in complex multiplication.

```
0                                   34                  a $13,$81,$34
1                                    456789             lqx       $16,$81,$34
0                                     56                ori       $21,$83,0
0                                      67               il        $22,0
0                                       7               nop       127
1                                        --0123         rotqby    $8,$16,$13
0                                           ---45       a $7,$8,$34
1   0                                          56789    lqx       $28,$8,$34
0                                               6       nop       127
1   -1234                                       ----    rotqby    $27,$28,$7
0     ---56                                             a $19,$27,$33
                                                        L63:
1         678901234567890                               hbrr      .L214,.L64
0          7890                                         shli      $23,$22,4
0           89                                          ori       $18,$26,0
0            90                                         ai        $25,$19,8
0             01                                        ai        $24,$19,24
0              12                                       a $28,$37,$23
0               23                                      ai        $23,$19,40
0                34                                     ai        $17,$28,8
1                 4                                     lnop
                                                        L64:
0                  56                                   ai        $12,$18,8
```

```
1                 678901                              lqd       $9,0($18)
0                  78                                 ai        $28,$17,-8
1                   890123                            lqd       $13,0($25)
0                    90                               ai        $11,$18,56
1                     012345                          lqd       $16,0($12)
1                      123456                         lqd       $14,0($19)
1                       234567                        lqd       $30,0($17)
1                        3456                         cdd       $31,0($17)
1                         4567                        rotqby    $5,$9,$18
1                          5678                       rotqby    $27,$13,$25
1                           6789                      rotqby    $2,$16,$12
0                            7                        nop       127
1                             8901                    rotqby    $15,$14,$19
0                              90                     ori       $14,$31,0
1                               0123                  cdd       $10,0($28)
0                                123456789            dfm       $9,$5,$27
0                                 234567890           dfm       $7,$27,$2
0                                  34                 ori       $27,$31,0
0                                   45                ori       $13,$10,0
0                                    56               ori       $16,$10,0
0                                     67              ai        $12,$18,104
0                                      789012345      dfma      $9,$15,$2
0d                                      890123456     dfms      $7,$15,$5
1d                                       --------6789 shufb     $29,$9,$30,$31
1  012345                                         ---- stqd      $29,0($17)
1   123456                                            lqd       $8,0($28)
1    -----7890                                        shufb     $2,$7,$8,$10
1          ---123456                                  stqd      $2,0($28)
1              234567                                 lqd       $6,0($11)
1               345678                                lqd       $3,0($24)
1                456789                               lqd       $4,48($18)
1                 567890                              lqd       $5,16($19)
1                  --8901                             rotqby    $30,$6,$11
1                     9012                            rotqby    $15,$3,$24
1                      0123                           rotqby    $31,$4,$18
1                       1234                          rotqby    $29,$5,$19
0                        -345678901                   dfm       $10,$15,$30
0                          456789012                  dfm       $3,$31,$15
0                           567890123                 dfms      $10,$29,$31
0                            678901234                dfma      $3,$29,$30
0                             --------456789012       dfa       $7,$7,$10
0                                     567890123       dfa       $9,$9,$3
1                                      --------3456   shufb     $4,$7,$2,$13
1  012                                          ---789 stqd      $4,0($28)
1  0123                                             89 lqd       $11,0($17)
1  ----4567                                          - shufb     $2,$9,$11,$14
1       ---890123                                     stqd      $2,0($17)
1           901234                                    lqd       $6,0($12)
1            012345                                   lqd       $5,0($23)
1             123456                                  lqd       $15,96($18)
1              234567                                 lqd       $31,32($19)
1               345678                                lqd       $11,0($28)
1                -5678                                rotqby    $10,$6,$12
1                  6789                               rotqby    $30,$5,$23
1                   7890                              rotqby    $29,$15,$18
0                    89                               ai        $18,$18,16
1                     9012                            rotqby    $13,$31,$19
0                      012345678                      dfm       $8,$30,$10
0                       123456789                     dfm       $3,$29,$30
0                        -345678901                   dfms      $8,$13,$29
0                          456789012                  dfma      $3,$13,$10
0                           --------234567890         dfa       $4,$7,$8
0d                                  345678901         dfa       $2,$9,$3
1d                                  ---------1234     shufb     $14,$4,$11,$16
1  0                                          ---56789 stqd      $14,0($28)
```

```
89 1  01                               6789 lqd      $5,0($17)
   1  --2345                           ---- shufb    $12,$2,$5,$27
91 1     ---678901                          stqd     $12,0($17)
   0         78                             ai       $17,$17,16
93 0           -90                          ceq      $6,$21,$17
                                            L214:
95 1             -1234                      brz      $6,.L64
   0               23                       ai       $22,$22,3
97 0                34                      ai       $19,$19,48
   0                 45                     ceqi     $17,$22,9
99 0d                 56                    ai       $21,$21,48
   1d                 -6789                 brz      $17,.L63
```

Bibliography

[1] D. J. Gross, F. Wilczek, Asymptotically Free Gauge Theories. I, Phys. Rev. D 8 (10) (1973) 3633–3652. doi:10.1103/PhysRevD.8.3633.

[2] D. J. Gross, F. Wilczek, Asymptotically free gauge theories. II, Phys. Rev. D 9 (4) (1974) 980–993. doi:10.1103/PhysRevD.9.980.

[3] Accessed 12/2010 [link].
URL http://en.wikipedia.org/wiki/Quark_model

[4] T. van Ritbergen, J. A. M. Vermaseren, S. A. Larin, The Four-Loop Beta Function in Quantum Chromodynamics, Phys. Lett. B400 (1997) 379–384. arXiv:hep-ph/9701390, doi:10.1016/S0370-2693(97)00370-5.

[5] R. Horsley, Habilitation Schrift, Berlin 2001: The Hadronic Structure of Matter – a lattice approach.

[6] W.-M. e. a. Yao, Review of Particle Physics, Journal of Physics G: Nuclear and Particle Physics 33 (1) (2006) 1.

[7] S. L. Glashow, Partial-symmetries of weak interactions, Nuclear Physics 22 (4) (1961) 579 – 588. doi:10.1016/0029-5582(61)90469-2.

[8] S. Weinberg, A Model of Leptons, Phys. Rev. Lett. 19 (21) (1967) 1264–1266. doi:10.1103/PhysRevLett.19.1264.

[9] A. Salam, Nobel Lecture, 1968.

[10] C. Amsler, et al., Review of Particle Physics, Phys. Lett. B667 (2008) 1. doi:10.1016/j.physletb.2008.07.018.

[11] S. L. Adler, Calculation of the Axial-Vector Coupling Constant Renormalization in β-decay, Phys. Rev. Lett. 14 (25) (1965) 1051–1055. doi:10.1103/PhysRevLett.14.1051.

[12] W. I. Weisberger, Renormalization of the Weak Axial-Vector Coupling Constant, Phys. Rev. Lett. 14 (25) (1965) 1047–1051. doi:10.1103/PhysRevLett.14.1047.

[13] S. Sasaki, K. Orginos, S. Ohta, T. Blum, Nucleon axial charge from quenched lattice QCD with domain wall fermions, Phys. Rev. D68 (2003) 054509. arXiv:hep-lat/0306007, doi:10.1103/PhysRevD.68.054509.

[14] N. Cabibbo, Unitary Symmetry and Leptonic Decays, Phys. Rev. Lett. 10 (12) (1963) 531–533. doi:10.1103/PhysRevLett.10.531.

[15] K. Dannbom, L. Y. Glozman, C. Helminen, D. O. Riska, Baryon magnetic moments and axial coupling constants with relativistic and exchange current effects, Nucl. Phys. A616 (1997) 555–574. arXiv:hep-ph/9610384, doi:10.1016/S0375-9474(97)81119-0.

[16] J. M. Gaillard, G. Sauvage, Hyperon Beta Decays, Ann. Rev. Nucl. Part. Sci. 34 (1984) 351–402.

[17] R. P. Feynman, Very High-Energy Collisions of Hadrons, Phys. Rev. Lett. 23 (24) (1969) 1415–1417. doi:10.1103/PhysRevLett.23.1415.

[18] M. Gell-Mann, Symmetries of Baryons and Mesons, Phys. Rev. 125 (3) (1962) 1067–1084. doi:10.1103/PhysRev.125.1067.

[19] G. Zweig, An SU(3) model for strong interaction symmetry and its breaking (CERN-TH-412) (1964) 80 p.

[20] A. V. Manohar, An introduction to spin dependent deep inelastic scatteringarXiv:hep-ph/9204208.

[21] D. Mueller, D. Robaschik, B. Geyer, F. M. Dittes, J. Horejsi, Wave functions, evolution equations and evolution kernels from light-ray operators of QCD, Fortschr. Phys. 42 (1994) 101. arXiv:hep-ph/9812448.

[22] A. V. Radyushkin, Nonforward Parton Distributions, Phys. Rev. D56 (1997) 5524–5557. arXiv:hep-ph/9704207, doi:10.1103/PhysRevD.56.5524.

[23] X.-D. Ji, Deeply-virtual Compton scattering, Phys. Rev. D55 (1997) 7114–7125. arXiv:hep-ph/9609381, doi:10.1103/PhysRevD.55.7114.

[24] M. Diehl, Generalized Parton Distributions, Phys. Rept. 388 (2003) 41–277. arXiv:hep-ph/0307382, doi:10.1016/j.physrep.2003.08.002.

[25] X.-D. Ji, Off-forward Parton Distributions, J. Phys. G24 (1998) 1181–1205. arXiv:hep-ph/9807358, doi:10.1088/0954-3899/24/7/002.

[26] A. V. Radyushkin, Generalized Parton Distributions arXiv:hep-ph/0101225.

[27] K. Goeke, M. V. Polyakov, M. Vanderhaeghen, Hard Exclusive Reactions and the Structure of Hadrons, Prog. Part. Nucl. Phys. 47 (2001) 401–515. arXiv:hep-ph/0106012, doi:10.1016/S0146-6410(01)00158-2.

[28] L. Kadanoff, Operator Algebra and the Determination of Critical Indices, Phys. Rev. Letters 23 (1969) 1430.

[29] F. David, On the Ambiguity of Composite Operators, I.R. renormalons and the Status of the Operator Product Expansion, Nucl. Phys. B 234 (1984) 237.

[30] V. A. Novikov, M. Shifman, A. Vainshtein, V. Zakharov, K. Wilson, Operator Product Expansion: Can It Fail ? , Nucl. Phys. B 249 (1985) 445.

[31] B. E. White, Factorisation in deeply virtual Compton scattering: Local OPE formalism and structure functions, J. Phys. G28 (2002) 203–222. arXiv:hep-ph/0102121, doi:10.1088/0954-3899/28/2/302.

[32] C. G. Callan, Broken Scale Invariance in Scalar Field Theory, Phys. Rev. D 2 (8) (1970) 1541–1547. doi:10.1103/PhysRevD.2.1541.

[33] K. Wilson, Non-Lagrangian Models of Current Algebra, Phys. Rev. 179 (1969) 1499.

[34] K. Wilson, Renormalization Group and Strong Interactions, Phys. Rev. D3 (1971) 1818.

[35] J. W. Negele, et al., Insight into nucleon structure from lattice calculations of moments of parton and generalized parton distributions, Nucl. Phys. Proc. Suppl. 128 (2004) 170–178. arXiv:hep-lat/0404005, doi:10.1016/S0920-5632(03)02474-5.

[36] LHPC Collaboration, Transverse structure of nucleon parton distributions from lattice QCD, Phys. Rev. Lett. 93 (2004) 112001. arXiv:hep-lat/0312014, doi:10.1103/PhysRevLett.93.112001.

[37] Detmold, W. and Melnitchouk, W. and Thomas, A. W., Moments of isovector quark distributions from lattice qcd, Phys. Rev. D 66 (5) (2002) 054501. doi:10.1103/PhysRevD.66.054501.

[38] P. Hägler, J. W. Negele, D. B. Renner, W. Schroers, T. Lippert, K. Schilling, Moments of nucleon generalized parton distributions in lattice QCD, Phys. Rev. D 68 (3) (2003) 034505. doi:10.1103/PhysRevD.68.034505.

[39] W. Detmold, W. Melnitchouk, J. W. Negele, D. B. Renner, A. W. Thomas, Chiral Extrapolation of Lattice Moments of Proton Quark Distributions, Phys. Rev. Lett. 87 (17) (2001) 172001. doi:10.1103/PhysRevLett.87.172001.

[40] X.-D. Ji, Off-forward parton distributions, J. Phys. G24 (1998) 1181–1205. arXiv:hep-ph/9807358, doi:10.1088/0954-3899/24/7/002.

[41] E. Leader, M. Anselmino, A crisis in the parton model: Where, oh where is the proton's spin ? , Zeitschrift für Physik C Particles and Fields 41 (1988) 239–246. doi:10.1007/BF01566922.

[42] C. Gattringer, C. B. Lang, Quantum chromodynamics on the lattice: an introductory presentation, Lecture Notes in Physics, Springer, Berlin, 2010.

[43] K. G. Wilson, Confinement of Quarks, Phys. Rev. D 10 (8) (1974) 2445–2459. doi:10.1103/PhysRevD.10.2445.

[44] H. B. Nielsen, M. Ninomiya, Absence of Neutrinos on a Lattice : (I). Proof by Homotopy Theory , Nuclear Physics B 185 (1) (1981) 20 – 40. doi:10.1016/0550-3213(81)90361-8.

[45] H. B. Nielsen, M. Ninomiya, Absence of Neutrinos on a Lattice : (II). Intuitive Topological Proof , Nuclear Physics B 193 (1) (1981) 173 – 194. doi:10.1016/0550-3213(81)90524-1.

[46] K. G. Wilson, Quarks and strings on a lattice, New Phenomena in Subnuclear Physics. Part A. Proceedings of the First Half of the 1975 International School of Subnuclear Physics, Erice, Sicily, July 11 - August 1, 1975, ed. A. Zichichi, Plenum Press, New York CLNS-321 (1977) p. 69.

[47] T. Matthews, A. Salam, Nuovo Cim. 12 (1954) 563.

[48] T. Matthews, A. Salam, Nuovo Cim. 2 (1955) 120.

[49] M. Creutz, Overrelaxation and Monte Carlo simulation, Phys. Rev. D 36 (2) (1987) 515–519. doi:10.1103/PhysRevD.36.515.

[50] S. Duane, A. D. Kennedy, B. J. Pendleton, D. Roweth, Hybrid Monte Carlo, Physics Letters B 195 (2) (1987) 216 – 222. doi:10.1016/0370-2693(87)91197-X.

[51] T. Takaishi, P. de Forcrand, Simulation of $N_f = 3$ QCD by Hybrid Monte Carlo, Nuclear Physics B - Proceedings Supplements 94 (1-3) (2001) 818 – 822. doi:10.1016/S0920-5632(01)01013-1.

[52] S. Aoki, R. Burkhalter, M. Fukugita, S. Hashimoto, K.-I. Ishikawa, N. Ishizuka, Y. Iwasaki, K. Kanaya, T. Kaneko, Y. Kuramashi, M. Okawa, T. Onogi, S. Tominaga, N. Tsutsui, A. Ukawa, N. Yamada, T. Yoshié, Polynomial hybrid Monte Carlo algorithm for lattice QCD with an odd number of flavors, Phys. Rev. D 65 (9) (2002) 094507. doi:10.1103/PhysRevD.65.094507.

[53] I. Horvath, A. D. Kennedy, S. Sint, A new exact method for dynamical fermion computations with non-local actions, Nuclear Physics B - Proceedings Supplements 73 (1-3) (1999) 834 – 836. doi:10.1016/S0920-5632(99)85217-7.

[54] M. A. Clark, A. D. Kennedy, Accelerating Dynamical-Fermion Computations Using the Rational Hybrid Monte Carlo Algorithm with Multiple Pseudofermion Fields, Phys. Rev. Lett. 98 (5) (2007) 051601. doi:10.1103/PhysRevLett.98.051601.

[55] K. Symanzik, Continuum limit and improved action in lattice theories. 2. O(N) non-linear sigma model in perturbation theory, Nucl. Phys B 226 (1983) 205.

[56] K. Symanzik, Continuum limit and improved action in lattice theories. 1. principles and ϕ^4 theory , Nucl. Phys B 226 (1983) 187.

[57] S. B., W. R., Improved continuum limit lattice action for QCD with Wilson fermions, Nucl. Phys. B 259 (1985) 572.

[58] M. Lüscher, S. Sint, R. Sommer, P. Weisz, Chiral symmetry and O(a) improvement in lattice QCD , Nucl. Phys. B 478 (1996) 365.

[59] M. Lüscher, Probing the Standard Model of Particle Interactions , Proceedings of the Les Houches Summer School, edited by R. Gupta, A. Morel, E. DeRafael, and F. David Elsevier (1999) Amsterdam.

[60] G. Heatlie, C. T. Sachrajda, G. Martinelli, C. Pittori, G. C. Rossi, The improvement of hadronic matrix elements in lattice QCD, Nuclear Physics B 352 (1) (1991) 266 – 288. doi:10.1016/0550-3213(91)90137-M.

[61] T. A. Degrand, P. Rossi, Conditioning techniques for dynamical fermions, Computer Physics Communications 60 (2) (1990) 211 – 214. doi:10.1016/0010-4655(90)90006-M.

[62] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. V. der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition, SIAM, Philadelphia, PA, 1994.

[63] W. Press, S. Teukolsky, W. Vetterling, B. Flannery, Numerical Recipes: The Art of Scientific Computing, New York: Cambridge University Press, 2007.

[64] W. Wilcox, T. Draper, K.-F. Liu, Chiral Limit of Nucleon Lattice Electromagnetic Form Factors, Phys.Rev.D 46 (1992) 1109–1122. arXiv:hep-lat/9205015.

[65] N. Cundy, et al., Non-perturbative improvement of stout-smeared three flavour clover fermions, Phys.Rev. D79 (2009) 094507. arXiv:0901.3302, doi:10.1103/PhysRevD.79.094507.

[66] S. Aoki, et al., Bulk first-order phase transition in three-flavor lattice QCD with $\mathcal{P}(a)$-improved Wilson fermion action at zero temperature, Phys. Rev. D72 (2005) 054510. arXiv:hep-lat/0409016, doi:10.1103/PhysRevD.72.054510.

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

[68] W. Bietenholz, et al., Tuning the strange quark mass in lattice simulations, Phys.Lett. B690 (2010) 436–441. arXiv:1003.1114, doi:10.1016/j.physletb.2010.05.067.

[69] M. Göckeler, et al., Determination of light and strange quark masses from full lattice QCD, Phys. Lett. B639 (2006) 307–311. arXiv:hep-ph/0409312, doi:10.1016/j.physletb.2006.06.036.

[70] A. D. Sokal, Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms, 1996.

[71] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes in C: The Art of Scientific Computing. Second Edition (1992).

[72] B. Efron, The jackknife, the bootstrap and other resampling plans (1994).

[73] A. Ali Khan, et al., Axial coupling constant of the nucleon for two flavours of dynamical quarks in finite and infinite volume, Phys.Rev. D74 (2006) 094508. arXiv:hep-lat/0603028, doi:10.1103/PhysRevD.74.094508.

[74] D. Pleiter, et al., Lattice 2010, PoS (2010) 153.

[75] T. R. Hemmert, B. R. Holstein, J. Kambor, Chiral Lagrangians and Delta(1232) interactions: Formalism, J. Phys. G24 (1998) 1831–1859. arXiv:hep-ph/9712496, doi:10.1088/0954-3899/24/10/003.

[76] M. Lüscher, Commun. Math. Phys. 104 (1986) 177.

[77] G. Colangelo, S. Durr, C. Haefeli, Finite volume effects for meson masses and decay constants, Nucl. Phys. B721 (2005) 136–174. arXiv:hep-lat/0503014, doi:10.1016/j.nuclphysb.2005.05.015.

[78] M. Göckeler, et al., Meson decay constants from $N(f) = 2$ clover fermions, PoS LAT2005 (2006) 063. arXiv:hep-lat/0509196.

[79] L. A. Ahrens, et al., Measurement of Neutrino - Proton and anti-neutrino - Proton Elastic Scattering, Phys. Rev. D35 (1987) 785. doi:10.1103/PhysRevD.35.785.

[80] V. Bernard, N. Kaiser, U.-G. Meissner, Chiral dynamics in nucleons and nuclei, Int. J. Mod. Phys. E4 (1995) 193–346. arXiv:hep-ph/9501384, doi:10.1142/S0218301395000092.

[81] S. Choi, et al., Axial and pseudoscalar nucleon form-factors from low- energy pion electroproduction, Phys. Rev. Lett. 71 (1993) 3927–3930. doi:10.1103/PhysRevLett.71.3927.

[82] H. Abele, The neutron. Its properties and basic interactions, Prog. Part. Nucl. Phys. 60 (2008) 1–81. doi:10.1016/j.ppnp.2007.05.002.

[83] Y. Aoki, et al., Lattice QCD with two dynamical flavors of domain wall fermions, Phys. Rev. D72 (2005) 114505. arXiv:hep-lat/0411006, doi:10.1103/PhysRevD.72.114505.

[84] H.-W. Lin, T. Blum, S. Ohta, S. Sasaki, T. Yamazaki, Nucleon structure with two flavors of dynamical domain-wall fermions , Phys. Rev. D78 (2008) 014505. arXiv:0802.0863, doi:10.1103/PhysRevD.78.014505.

[85] T. Yamazaki, et al., Nucleon form factors with 2+1 flavor dynamical domain-wall fermions, Phys. Rev. D79 (2009) 114505. arXiv:0904.2039, doi:10.1103/PhysRevD.79.114505.

[86] C. Gattringer, et al., Hadron Spectroscopy with Dynamical Chirally Improved Fermions, Phys. Rev. D79 (2009) 054501. arXiv:0812.1681, doi:10.1103/PhysRevD.79.054501.

[87] C. Alexandrou, et al., Axial Nucleon form factors from lattice QCD arXiv:1012.0857.

[88] J. D. Bratt, et al., Nucleon structure from mixed action calculations using 2+1 flavors of asqtad sea and domain wall valence fermions, Phys. Rev. D82 (2010) 094502. arXiv:1001.3620, doi:10.1103/PhysRevD.82.094502.

[89] C. Alexandrou, Hadron Structure and Form Factors, PoS LATTICE2010 (2010) 001. arXiv:1011.3660.

[90] J. J. de Swart, The Octet model and its Clebsch-Gordan coefficients, Rev. Mod. Phys. 35 (1963) 916–939. doi:10.1103/RevModPhys.35.916.

[91] M. J. Savage, J. Walden, SU(3) breaking in neutral current axial matrix elements and the spin-content of the nucleon, Phys. Rev. D55 (1997) 5376–5384. arXiv:hep-ph/9611210, doi:10.1103/PhysRevD.55.5376.

[92] J. Dai, R. F. Dashen, E. E. Jenkins, A. V. Manohar, Flavor Symmetry Breaking in the 1/N Expansion, Phys. Rev. D53 (1996) 273–282. arXiv:hep-ph/9506273, doi:10.1103/PhysRevD.53.273.

[93] R. Flores-Mendieta, E. E. Jenkins, A. V. Manohar, SU(3) symmetry breaking in hyperon semileptonic decays, Phys. Rev. D58 (1998) 094028. arXiv:hep-ph/9805416, doi:10.1103/PhysRevD.58.094028.

[94] K.-S. Choi, W. Plessas, R. Wagenbrunn, Axial charges of octet and decuplet baryons, Phys.Rev. D82 (2010) 014007. arXiv:1005.0337, doi:10.1103/PhysRevD.82.014007.

[95] H.-W. Lin, K. Orginos, First Calculation of Hyperon Axial Couplings from Lattice QCD , Phys.Rev. D79 (2009) 034507. arXiv:0712.1214, doi:10.1103/PhysRevD.79.034507.

[96] K. Orginos, D. Toussaint, Testing improved actions for dynamical Kogut-Susskind quarks, Phys. Rev. D59 (1999) 014501. arXiv:hep-lat/9805009, doi:10.1103/PhysRevD.59.014501.

[97] J. B. Kogut, L. Susskind, Hamiltonian Formulation of Wilson's Lattice Gauge Theories , Phys. Rev. D11 (1975) 395. doi:10.1103/PhysRevD.11.395.

[98] K. Orginos, D. Toussaint, R. L. Sugar, Variants of fattening and flavor symmetry restoration, Phys. Rev. D60 (1999) 054503. arXiv:hep-lat/9903032, doi:10.1103/PhysRevD.60.054503.

[99] D. B. Kaplan, A Method for simulating chiral fermions on the lattice, Phys. Lett. B288 (1992) 342–347. arXiv:hep-lat/9206013, doi:10.1016/0370-2693(92)91112-M.

[100] D. B. Kaplan, Chiral fermions on the lattice, Nucl. Phys. Proc. Suppl. 30 (1993) 597–600. doi:10.1016/0920-5632(93)90282-B.

[101] Y. Shamir, Chiral fermions from lattice boundaries, Nucl. Phys. B406 (1993) 90–106. arXiv:hep-lat/9303005, doi:10.1016/0550-3213(93)90162-I.

[102] V. Furman, Y. Shamir, Axial symmetries in lattice QCD with Kaplan fermions, Nucl. Phys. B439 (1995) 54–78. arXiv:hep-lat/9405004, doi:10.1016/0550-3213(95)00031-M.

[103] G. Erkol, M. Oka, T. T. Takahashi, Axial Charges of Octet Baryons in Two-flavor Lattice QCD, Phys.Lett. B686 (2010) 36–40. arXiv:0911.2447, doi:10.1016/j.physletb.2010.02.016.

[104] S. Alekhin, J. Blümlein, S. Klein, S. Moch, The 3-, 4-, and 5-flavor NNLO Parton from Deep-Inelastic- Scattering Data and at Hadron Colliders, Phys. Rev. D81 (2010) 014032. arXiv:0908.2766, doi:10.1103/PhysRevD.81.014032.

[105] J. Blümlein, H. Böttcher, A. Guffanti, Non-singlet QCD analysis of deep inelastic world data at $O(\alpha_S^3)$, Nucl. Phys. B774 (2007) 182–207. arXiv:hep-ph/0607200, doi:10.1016/j.nuclphysb.2007.03.035.

[106] P. Jimenez-Delgado, E. Reya, Dynamical NNLO parton distributions, Phys. Rev. D79 (2009) 074023. arXiv:0810.4274, doi:10.1103/PhysRevD.79.074023.

[107] J. Blümlein, H. Böttcher, A. Guffanti, Non-singlet QCD analysis of the structure function F2 in 3-loops, Nucl. Phys. Proc. Suppl. 135 (2004) 152–155. arXiv:hep-ph/0407089, doi:10.1016/j.nuclphysbps.2004.09.059.

[108] A. D. Martin, W. J. Stirling, R. S. Thorne, G. Watt, Uncertainties on α_S in global PDF analyses and implications for predicted hadronic cross sections, Eur. Phys. J. C64 (2009) 653–680. arXiv:0905.3531, doi:10.1140/epjc/s10052-009-1164-2.

[109] S. Alekhin, K. Melnikov, F. Petriello, Fixed target Drell-Yan data and NNLO QCD fits of parton distribution functions, Phys. Rev. D74 (2006) 054033. arXiv:hep-ph/0606237, doi:10.1103/PhysRevD.74.054033.

[110] C. Allton, et al., 2+1 flavor domain wall QCD on a (2 fm)^3 lattice: light meson spectroscopy with $Ls = 16$, Phys. Rev. D76 (2007) 014504. arXiv:hep-lat/0701013, doi:10.1103/PhysRevD.76.014504.

[111] C. Allton, et al., Physical Results from 2+1 Flavor Domain Wall QCD and SU(2) Chiral Perturbation Theory, Phys. Rev. D78 (2008) 114509. arXiv:0804.0473, doi:10.1103/PhysRevD.78.114509.

[112] P. Hägler, et al., Nucleon Generalized Parton Distributions from Full Lattice QCD, Phys. Rev. D77 (2008) 094502. arXiv:0705.4295, doi:10.1103/PhysRevD.77.094502.

[113] G. A. Miller, A. K. Opper, E. J. Stephenson, Charge symmetry breaking and QCD, Ann. Rev. Nucl. Part. Sci. 56 (2006) 253–292. arXiv:nucl-ex/0602021, doi:10.1146/annurev.nucl.56.080805.140446.

[114] J. T. Londergan, J. C. Peng, A. W. Thomas, Charge Symmetry at the Partonic Level, Rev. Mod. Phys. 82 (2010) 2009–2052. arXiv:0907.2352, doi:10.1103/RevModPhys.82.2009.

[115] J. T. Londergan, A. W. Thomas, The validity of charge symmetry for parton distributions, Prog. Part. Nucl. Phys. 41 (1998) 49–124. arXiv:hep-ph/9806510, doi:10.1016/S0146-6410(98)00055-6.

[116] E. Sather, Isospin violating quark distributions in the nucleon, Phys. Lett. B274 (1992) 433–438. doi:10.1016/0370-2693(92)92011-5.

[117] E. N. Rodionov, A. W. Thomas, J. T. Londergan, Charge asymmetry of parton distributions, Mod. Phys. Lett. A9 (1994) 1799–1806. doi:10.1142/S0217732394001659.

[118] J. T. Londergan, A. W. Thomas, Charge symmetry violation corrections to determination of the Weinberg angle in neutrino reactions, Phys. Rev. D67 (2003) 111901. arXiv:hep-ph/0303155, doi:10.1103/PhysRevD.67.111901.

[119] A. D. Martin, R. G. Roberts, W. J. Stirling, R. S. Thorne, Uncertainties of predictions from parton distributions. I: Theoretical errors, Eur. Phys. J. C35 (2004) 325–348. arXiv:hep-ph/0308087.

[120] G. Martinelli, et al., A General method for nonperturbative renormalization of lattice operators, Nucl. Phys. B445 (1995) 81–108. arXiv:hep-lat/9411010, doi:10.1016/0550-3213(95)00126-D.

[121] M. Göckeler, et al., Nonperturbative renormalisation of composite operators in lattice QCD , Nucl. Phys. B544 (1999) 699–733. arXiv:hep-lat/9807044, doi:10.1016/S0550-3213(99)00036-X.

[122] M. Göckeler, et al., Renormalisation of composite operators in lattice QCD: perturbative versus nonperturbative arXiv:1010.1360.

[123] P. Hägler, Hadron structure from lattice quantum chromodynamics, Phys.Rept. 490 (2010) 49–175. arXiv:0912.5483, doi:10.1016/j.physrep.2009.12.008.

[124] W. Detmold, et al., Chiral extrapolation of lattice moments of proton quark distributions, Phys. Rev. Lett. 87 (2001) 172001. arXiv:hep-lat/0103006, doi:10.1103/PhysRevLett.87.172001.

[125] W. Detmold, W. Melnitchouk, A. W. Thomas, Extraction of parton distributions from lattice QCD, Mod. Phys. Lett. A18 (2003) 2681–2698. arXiv:hep-lat/0310003, doi:10.1142/S0217732304015725.

[126] W. Detmold, C. J. D. Lin, Twist-two matrix elements at finite and infinite volume, Phys. Rev. D71 (2005) 054510. arXiv:hep-lat/0501007, doi:10.1103/PhysRevD.71.054510.

[127] M. Dorati, T. A. Gail, T. R. Hemmert, Chiral Perturbation Theory and the first moments of the Generalized Parton Distriputions in a Nucleon, Nucl. Phys. A798 (2008) 96–131. arXiv:nucl-th/0703073, doi:10.1016/j.nuclphysa.2007.10.012.

[128] Y. Aoki, et al., Nucleon isovector structure functions in (2+1)-flavor QCD with domain wall fermions, Phys.Rev. D82 (2010) 014501. arXiv:1003.3387, doi:10.1103/PhysRevD.82.014501.

[129] D. B. Leinweber, QCD Equalities for Baryon Current Matrix Elements, Phys. Rev. D53 (1996) 5115–5124. arXiv:hep-ph/9512319, doi:10.1103/PhysRevD.53.5115.

[130] H. Leutwyler, The ratios of the light quark masses, Phys. Lett. B378 (1996) 313–318. arXiv:hep-ph/9602366, doi:10.1016/0370-2693(96)00386-3.

[131] J. T. Londergan, A. W. Thomas, Charge Symmetry Violating contributions to neutrino reactions, Phys. Lett. B558 (2003) 132–140. arXiv:hep-ph/0301147, doi:10.1016/S0370-2693(03)00267-3.

[132] M. Glück, E. Reya, A. Vogt, Dynamical parton distributions revisited, Eur.Phys.J. C5 (1998) 461–470. arXiv:hep-ph/9806404, doi:10.1007/s100520050289.

[133] M. Diehl, Generalized Parton Distributions with helicity flip, Eur. Phys. J. C19 (2001) 485–492. arXiv:hep-ph/0101335, doi:10.1007/s100520100635.

[134] S. Aoki, M. Doui, T. Hatsuda, Y. Kuramashi, Tensor charge of the nucleon in lattice QCD, Phys. Rev. D56 (1997) 433–436. arXiv:hep-lat/9608115, doi:10.1103/PhysRevD.56.433.

[135] P. Hägler, Form factor decomposition of generalized parton distributions at leading twist, Phys. Lett. B594 (2004) 164–170. arXiv:hep-ph/0404138, doi:10.1016/j.physletb.2004.05.014.

[136] Z. Chen, X.-d. Ji, Counting and tensorial properties of twist-two helicity-flip nucleon form factors, Phys. Rev. D71 (2005) 016003. arXiv:hep-ph/0404276, doi:10.1103/PhysRevD.71.016003.

[137] M. A. Donnellan, et al., Lattice Results for Vector Meson Couplings and Parton Distribution Amplitudes, PoS LAT2007 (2007) 369. arXiv:0710.0869.

[138] A. A. Khan, et al., Axial and tensor charge of the nucleon with dynamical fermions, Nucl. Phys. Proc. Suppl. 140 (2005) 408–410. arXiv:hep-lat/0409161, doi:10.1016/j.nuclphysbps.2004.11.320.

[139] M. Göckeler, et al., Quark helicity flip generalized parton distributions from two-flavor lattice QCD, Phys. Lett. B627 (2005) 113–123. arXiv:hep-lat/0507001, doi:10.1016/j.physletb.2005.09.002.

[140] K. Jansen, Lattice 2008, PoS (2008) 010.

[141] F. Belletti, S. F. Schifano, R. Tripiccione, F. Bodin, P. Boucaud, J. Micheli, O. Pene, N. Cabibbo, S. de Luca, A. Lonardo, D. Rossetti, P. Vicini, M. Lukyanov, L. Morin, N. Paschedag, H. Simma, V. Morenas, D. Pleiter, F. Rapuano, Computing for LQCD: apeNEXT , Computing in Science and Engineering 8 (1) (2006) 18–29.

[142] P. A. Boyle, D. Chen, N. H. Christ, M. A. Clark, S. D. Cohen, C. Cristian, Z. Dong, A. Gara, B. Joó, C. Jung, C. Kim, L. A. Levkova, X. Liao, G. Liu, R. D. Mawhinney, S. Ohta, K. Petrov, T. Wettig, A. Yamaguchi, Overview of the QCDSP and QCDOC computers, IBM J. Res. Dev. 49 (2) (2005) 351–365. doi:10.1147/rd.492.0351.

[143] H. Baier, et al., QPACE – a QCD parallel computer based on Cell processors arXiv:0911.2174.

[144] Accessed 12/2010 [link].
URL http://www.mcs.anl.gov/research/projects/mpi/

[145] Accessed 12/2010 [link].
URL http://www.top500.org

[146] Accessed 12/2010 [link].
URL http://www.openmp.org

[147] Accessed 12/2010 [link].
URL http://www.green500.org

[148] Accessed 12/2010 [link].
URL http://www.nvidia.com/cuda

[149] Accessed 12/2010 [link].
URL http://www.khronos.org/opencl/

[150] R. Schreiber, et al., Node Architecture and Power Group Report, Exascale Technology Roadmap Meeting, 2009.

[151] R. Tripiccione, Dedicated computers for LGT , Nuclear Physics B - Proceedings Supplements 17 (1990) 137 – 145. doi:DOI:10.1016/0920-5632(90)90227-L.

[152] T. L. Veldhuizen, D. Gannon, Active Libraries: Rethinking the roles of compilers and libraries , ArXiv Mathematics e-prints arXiv:math/9810022.

[153] S. Haney, J. Crotinger, S. Karmesin, S. Smith, Easy expression templates using PETE, the portable expression template engine, Technical Report LA-UR-99 (1999) 777.

[154] Accessed 12/2010 [link].
URL http://acts.nersc.gov/pooma/

[155] R. G. Edwards, B. Joó, The Chroma software system for lattice QCD, Nucl.Phys.Proc.Suppl. 140 (2005) 832. arXiv:hep-lat/0409003, doi:10.1016/j.nuclphysbps.2004.11.254.

[156] Accessed 12/2010 [link].
URL http://www.lqcd.org/scidac/

[157] J. Chen, W. Watson, W. Mao, Multi-Threading Performance on Commodity Multi-core Processors, in: Proceedings of 9th International Conference on High Performance Computing in Asia Pacific Region, 2007.

[158] P. A. Boyle, The BAGEL assembler generation library , Comp. Phys. Comm. 180 (2009) 2739.

[159] T. Veldhuizen, Expression Templates , C++ Report 7, 1995.

[160] IBM, Cell BE Programming Handbook Including PowerXCell 8i, Version 1.11, 5. Dec. 2008.

[161] Accessed 12/2010 [link].
URL `http://www.perl.org/`

[162] Accessed 12/2010 [link].
URL `http://www.boost.org/`

[163] R. C. Whaley, A. Petitet, J. Dongarra, Automated Empirical Optimization of Software and the ATLAS project, Parallel Computing 2001, 27(1-2):3-35.

[164] M. Frigo, S. G. Johnson, The Design and Implementation of FFTW3 `doi:10.1.1.136.7045`.

[165] A. Nobile, Lattice 2010, PoS (2010) 86.

Acknowledgements

It is an honour for me to dedicate my acknowledgements to individuals, groups, and institutions who supported me in writing this thesis.

Primarily I would like to show my gratitude to Prof. D. Pleiter who advised my research work during the last 3 years. His exceptional expertise in the realm of high energy physics, computer science, and supercomputer design is a great source of inspiration.

This work has been supported in part by the Deutsche Forschungsgemeinschaft (DFG) Sonderforschungsbereich (SFB) - TR55 "Hadron Physics from Lattice QCD" and I would like to show my gratitude to Prof. A. Schäfer who is the head of the DFG SFB TR55 and the host of this thesis. His inspiring personality and endless care for the whole team provides an excellent and enjoyable working environment.

My affiliation within the DFG SFB TR55 was project area C: "Development and deployment of the QPACE parallel supercomputer". The whole development team consisting of physicists and engineers of the Universities of Regensburg, Wuppertal, Ferrara and Milano Bicocca, of DESY Zeuthen and Forschungszentrum Jülich as well as of IBM Böblingen, La Gaude and Rochester deserve sincere respect for the accomplished achievements. I enjoyed working in the project very much and it is a pleasure to thank the whole team. Special thanks go to D. Pleiter for sharing his profound knowledge on parallel computers and IBM developerWorks for helpful discussions.

Additional support was provided by the HPC Europa2 Transnational Access Programme, a pan-European research infrastructure on high performance computing, funded by the European Commission DG Research in the Seventh Framework Programme. HPC Europa2 provided support for 2 research visits (with an aggregate duration of 3 months) to the EPCC, a leading European centre of expertise in advanced research, technology transfer and the provision of supercomputer services to academia and business located at the University of Edinburgh. While there I worked in the Particle Physics Theory (PPT) group with local experts on the investigation of hadron structure and Lattice QCD. I was finding the working environment within in the PPT group to be exceptional and gratefully thank Dr. R. Horsley and Dr. J. M. Zanotti for a fruitful collaboration and unique time.

As a member of the QCDSF Collaboration I would like to thank my collaborators working on hadron structure. Particularly I am again grateful to D. Pleiter, R. Horsley and J. M. Zanotti who were very committed in introducing me to the investigation of hadron structure and never kept me waiting for an advice.

Additionally I would like to send my thanks to Dr. R. Edwards and Dr. B. Joó for helpful discussions on QDP++ during the National Science Foundation (NSF)-Numerical Algorithms and Intelligent Software (NAIS) Workshop 2009 – the interface between algorithms and machines – in Edinburgh.

My place of work was first (24 months) located at the John von Neumann Institute for Computing (NIC) at the Deutsches Elektronensynchrotron (DESY) Zeuthen site and later (12 months) at the particle physics group at the University of Regensburg. I am grateful to the NIC DESY team for providing an exceptional good computing environment and to the particle physics group for the aforementioned enjoyable working environment.

The numerical calculations have been performed on the apeNEXT at NIC/DESY (Zeuthen, Germany), the IBM BlueGeneL at EPCC (Edinburgh, UK), the BlueGeneP (JuGene) and the Nehalem Cluster (JuRoPa) at NIC (Jülich, Germany), and the SGI ICE 8200 at HLRN (Berlin-Hannover, Germany), on HECToR at the EPCC as well as on the IBM/Cell QS-Cluster JUICEnext at NIC. I thank all institutions.

The following publications result from this work:

- F. Winter et al.: "Baryon Axial Charges and Momentum Fractions with $N_f = 2 + 1$ Dynamical Fermions", PoS (Lattice 2010) 163.

- R. Horsley et al.: "Flavour Symmetry Breaking and Tuning the Strange Quark Mass for $N_f = 2 + 1$ Quark Flavours", `arXiv:1012.4371`.

- H. Baier et al.: "QPACE – a QCD Parallel Computer Based on Cell Processors", PoS (Lattice 2009) 001.
- G. Goldrian et al.: "QPACE: Quantum Chromodynamics Parallel Computing on the Cell Broadband Engine", Computing in Science & Engineering (CiSE), November/December 2008 (vol. 10 no. 6) pp. 46-54.
- F. Winter et al.: "QCD Data Parallel for Multicore Acceleration and Semi-Leptonic Octet Hyperon Decays", HPC-Europa2 Annual Book, 2009.
- F. Winter et al.: "Chroma/QDP++ on IBM PowerXCell 8i Processor", HPC-Europa2 Annual Book, 2010.
- R. Horsley et al.: "Isospin Symmetry Breaking in Parton Distribution Functions from Lattice QCD", in preparation.
- F. Winter et al.: "New Design Concepts for an Implementation of QDP++ on Heterogeneous Multicore Acceleration Processors", in preparation.

Abstract

Observables relevant for the understanding of the structure of baryons were determined by means of Monte Carlo simulations of Lattice Quantum Chromodynamics (QCD) using 2+1 dynamical quark flavours. Especial emphasis was placed on how these observables change when flavour symmetry is broken in comparison to choosing equal masses for the two light and the strange quark. The first two moments of unpolarised, longitudinally, and transversely polarised parton distribution functions were calculated for the nucleon and hyperons. The latter are baryons which comprise a strange quark.

Lattice QCD simulations tend to be extremely expensive, reaching the need for petaflop computing and beyond, a regime of computing power we just reach today. Heterogeneous multicore computing is getting increasingly important in high performance scientific computing. The strategy of deploying multiple types of processing elements within a single workflow, and allowing each to perform the tasks to which it is best suited is likely to be part of the roadmap to exascale. In this work new design concepts were developed for an active library (QDP++) harnessing the compute power of a heterogeneous multicore processor (IBM PowerXCell 8i processor). Not only a proof-of-concept is given furthermore it was possible to run a QDP++ based physics application (Chroma) on an IBM BladeCenter QS22.

Zusammenfassung

Die für das Verständnis der Struktur von Baryonen relevanten Observablen wurden mittels Monte-Carlo-Simulationen bestimmt, wobei 2+1 dynamische Quarks verwendet wurden. Dabei wurde insbesondere untersucht, wie sich diese Observablen ändern, wenn die Flavour-Symmetrie gestört wird im Vergleich zum Fall, wo die Massen der beiden leichten Quarks sowie des schweren Quarks gleich gewählt werden. Es wurden die ersten beiden Momente der unpolarisierten, longitudinal sowie transversal polarisierten Parton Distributionsfunktionen von Nukleonen und Hyperonen berechnet. Letztere sind Baryonen mit Strangequark Inhalt.

Gitter QCD Simulationen neigen dazu, extrem teuer zu sein. Dabei reicht die benötigte Rechenleistung bis in den Peta-Flop Bereich und darüber hinaus – ein Leistungsbereich der erst jetzt erschlossen wird. Heterogenes Mehrkernrechnen wird zunehmend bedeutsamer im wissenschaftlichen Umfeld des Hoch- und Höchstleistungsrechnens. Die Strategie, verschiedene Typen von Rechenelementen in einen einzigen Arbeitsfluss zu integrieren bei gleichzeitiger optimaler Zuordnung der einzelnen Arbeitsschritte zu Rechenelementen wird sicherlich Teil des erfolgreichen Wegs zu Exascale sein. In dieser Arbeit wurden neue Designelemente für aktive Bibliotheken (QDP++) unter Ausnutzung der Rechenleistung heterogener Mehrkern-Prozessoren (IBM PowerXCell 8i Prozessor) entwickelt. Nicht nur wurde eine Machbarkeitsstudie unternommen, vielmehr war es möglich eine Physik-Anwendung (Chroma) mit vernünftiger Performance auf einem IBM BladeCenter QS22 auszuführen.