This work discusses the application of Feynman diagram sampling in quantum field theories. The method uses a computer simulation to sample the diagrammatic space obtained in a series expansion. For running large physical simulations powerful computers are obligatory, effectively splitting the thesis in two parts.

The first part deals with the method of Feynman diagram sampling. Here the theoretical background of the method itself is discussed. Additionally, important statistical concepts and the theory of the strong force, quantum chromodynamics, are introduced. This sets the context of the simulations. We create and evaluate a variety of models to estimate the applicability of diagrammatic methods. The method is then applied to sample the perturbative expansion of the vertex correction. In the end we obtain the value for the anomalous magnetic moment of the electron.

The second part looks at the QPACE 2 supercomputer. This includes a short introduction to supercomputers in general, as well as a closer look at the architecture and the cooling system of QPACE 2. Guiding benchmarks of the InfiniBand network are presented. At the core of this part, a collection of best practices and useful programming concepts are outlined, which enables the development of efficient, yet easily portable, applications for the QPACE 2 system.

Universitätsverlag Regensburg

ISBN 978-3-86845-133-7

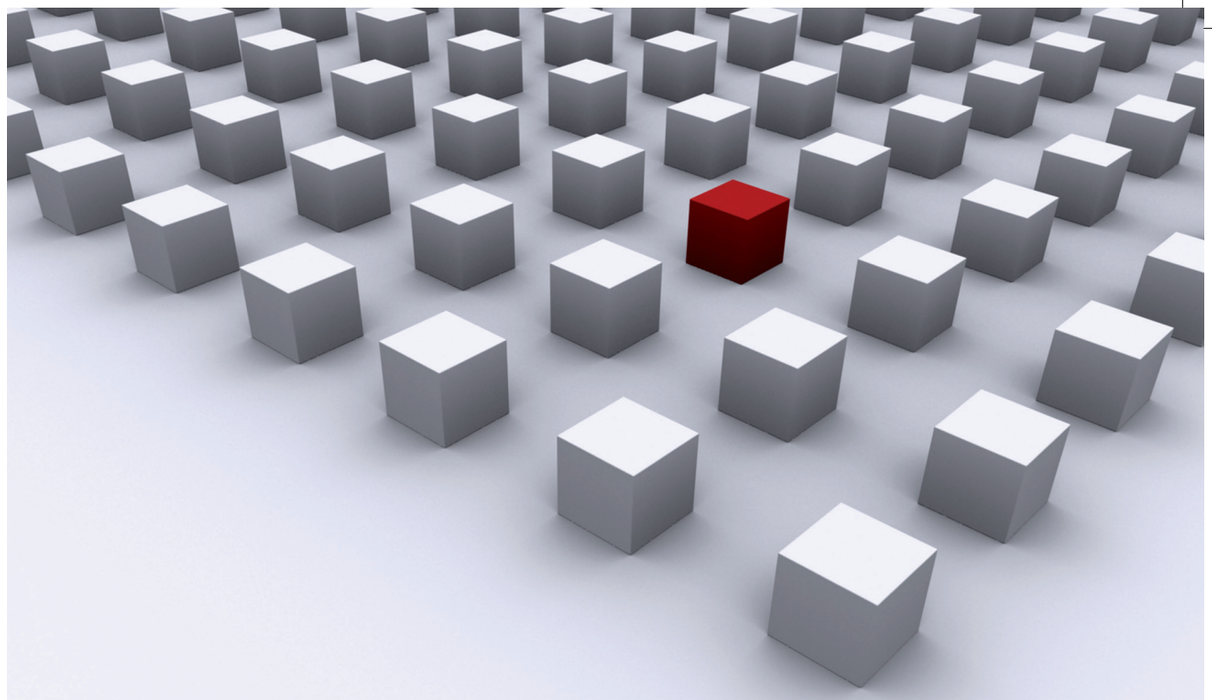
gefördert von:

Alumni der physikalischen Fakultät der Universität Regensburg e.V.

Universität Regensburg

Florian Rappl

Dissertationsreihe Physik - Band 49



Florian Rappl

Feynman Diagram Sampling for Quantum Field Theories on the QPACE 2 Supercomputer

Universitätsverlag Regensburg

Florian Rappl

49

Dissertationsreihe Physik

Florian Rappl

Feynman Diagram Sampling for Quantum Field Theories on the QPACE 2 Supercomputer

Feynman Diagram Sampling for Quantum Field Theories on the QPACE 2 Supercomputer

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

Florian Rappl

aus Regensburg
im Jahr 2015

Die Arbeit wurde von Prof. Dr. T. Wettig angeleitet.
Das Promotionsgesuch wurde am 17.12.2015 eingereicht.
Das Kolloquium fand am 05.02.2016 statt.

Prüfungsausschuss:	Vorsitzender:	Prof. Dr. C. Schüller
	1. Gutachter:	Prof. Dr. T. Wettig
	2. Gutachter:	Prof. Dr. G. Bali
	weiterer Prüfer:	Prof. Dr. J. Fabian

Dissertationsreihe der Fakultät für Physik der Universität Regensburg, Band 49

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

Florian Rappl

Feynman Diagram Sampling for Quantum Field Theories on the QPACE 2 Supercomputer

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 2016

Leibnizstraße 13, 93055 Regensburg

Konzeption: Thomas Geiger
Umschlagentwurf: Franz Stadler, Designcooperative Nittenau eG
Layout: Florian Rappl
Druck: Docupoint, Magdeburg
ISBN: 978-3-86845-133-7

Weitere Informationen zum Verlagsprogramm erhalten Sie unter:
www.univerlag-regensburg.de

Feynman Diagram Sampling for Quantum Field Theories on the QPACE 2 Supercomputer

Abstract

This thesis covers the topic of applying the method of Feynman diagram sampling to quantum field theories. The method uses a computer simulation to sample the diagrammatic space obtained in a series expansion. For running large physical simulations powerful computers are required. Therefore, two independent parts are supplied.

The first part deals with the method of Feynman diagram sampling. The theoretical background of the method itself is discussed. Additionally, important statistical concepts and the theory of the strong force are introduced. A variety of models to study the applicability of diagrammatic methods are evaluated. The method is then applied to sample the perturbative expansion of the vertex correction. In the end we estimate the value of the anomalous magnetic moment of the electron.

The second part describes the QPACE 2 supercomputer by looking at its the architecture and cooling system. Guiding benchmarks of the InfiniBand network are presented. At the core of this part, a collection of best practices and useful programming concepts are outlined, which enables the development of efficient, yet easily portable applications for the QPACE 2 system.

Contents

Dedicated to my aunt Monika (1951 – 2013).

Part I

Feynman Diagram Sampling for QFT

1

Introduction to Feynman Diagram Sampling

The quest for knowledge is deeply connected with the elementary goals of particle physics. In particle physics we are curious to find out more about the basic building blocks of matter. All the recent discoveries at the Large Hadron Collider (LHC) gained tremendous media coverage. Among others the list contains the observation of the Higgs boson [1], the recent discovery of a pentaquark state [2], and new baryon resonances [3].

Even though most people are not directly influenced by achievements of the LHC, they are most certainly indirectly affected by the developments in particle physics. The most striking example is the world wide web, which forms the basis for the information age [16]. There are many other advancements in engineering, computer science, and data analysis, which have been direct or indirect products of research in particle physics.

Particle physics is a discipline that relies heavily on the progress in the IT industry. This is true not only on the theoretical side, where we need excessive computing power to run simulations [51], but also on the experimental side, which has to deal with huge amounts of data. Thus communication with the field of computational physics is definitely relevant. Computational physics is dedicated to apply novel methods found in numerical mathematics and computer science to solve problems in physics.

If we would draw a diagram to illustrate the relations between computational physics, theoretical physics, experimental physics, computer science, algorithms, and electronics, we would probably place computational physics in the center. Not only does computational physics touch many areas, it connects previously disconnected or only weakly connected fields. This enables a much richer exchange and opens the door for new collaborations.

The rich exchange between computational physics and the classical disciplines in physics strengthens their relationship even more. Theoretical or experimental physics have been in good contact beforehand, however, by directly discussing ideas and implementations of simulations these fields are overlapping even more. Usually, the overlap goes way beyond specific fields of study, which is only possible by providing a common notation and language.

An example for a fruitful transfer of ideas is the progress of Markov chain Monte Carlo [68] methods, which are introduced in Chapter 2. Initially developed to solve problems in solid state physics, these methods have been adopted very successfully to simulate particle physics using a discretization known as the lattice [150]. The adjustments and improvements, such as including molecular dynamics [7], from lattice improvements [44] were then again applied to solid state physics. Consequently, the exchange and ongoing search for better methods is beneficial for both fields [58].

The growing toolbox of deterministic, non-deterministic, and mixed methods makes efficient computation of many different models possible. One of the new challenges is to identify the ideal method for a given model. Furthermore, some methods may be completely unsuited for a specific class of problems. We require careful evaluations to determine properties of the available methods. The idea is to construct a generalized mapping for models to find the right methods of simulation.

In this thesis we investigate how diagrammatic methods can be utilized for studying quantum field theories. A little over a decade ago diagrammatic methods have appeared [147]. Initially, they have been limited to only a few problems, but recent developments justify our hope to make them usable beyond their original purposes in solid state physics. The essential method is discussed in Chapter 3.

Our goal in this work is to derive recommendations and constraints when a diagrammatic method can be used. Diagrammatic methods try to sample a possibly infinite series of integrals by identifying suitable weights for state transitions. We are especially interested in the case of applying Diagrammatic Monte Carlo or a specialization of it to solve problems within the theory of QCD.

2 Theory

This chapter introduces important concepts that are used extensively in the succeeding chapters. We start with a brief description of Monte Carlo methods (Section 2.1). In Section 2.2 we walk through an introduction to the most important statistical concepts. This will be our reference for calculating estimates and their respective errors later on.

In the next section we look at all the physics which is relevant for this work. We start by introducing our notation for using special relativity in Section 2.3. Special relativity is used by the physical theory of the strong force, which is introduced in Section 2.4.

Finally, in Section 2.5 we investigate how to discretize the theory for running simulations, which allow the computation of observables. This section also motivates the research that is presented and described in the subsequent chapters of this thesis.

2.1 Monte Carlo Methods

Monte Carlo methods describe a class of algorithms that utilize the computational power of modern computers by relying on repeated random sampling. In most algorithms we repeat a set of steps in a simulations to obtain a an ensemble of samples of a probability distribution, which can be used to estimate properties of the distribution. In our context Monte Carlo methods represent a class of

methods that provides efficient algorithms for simulating physical models and finding solutions to problems that cannot be solved analytically.

The reference to the Monte Carlo Casino of Monaco in the name originated from the resemblance of gambling. Since gambling is a statistical activity, the name hints that algorithms from this class of methods require many runs to satisfy statistical conditions. The most important condition to satisfy is the law of large numbers. A large number of samples is useful, if and only if we scan over the whole data space. Here we use random numbers.

We start our discussion of Monte Carlo methods by introducing the concept of random numbers and random number generators (RNGs). Then we introduce methods to handle distributions and the concept of Markov chains. Finally, generic Monte Carlo methods are discussed.

2.1.1 Random Numbers and Random Number Generators

A sequence of numbers is said to be statistically random when it contains no patterns or regularities. This makes the sequence at any point unpredictable. A number in this sequence is then called a random number. Examples of such sequences can be found in transcendental numbers such as π. They can be constructed, e.g., by noting the results of an ideal dice roll. The statistical randomness is not *true* randomness, but sufficient for uses in statistical applications such as Monte Carlo methods.

Random numbers arise in nature just by employing a macroscopically large number of variables and quantum mechanics. In Monte Carlo methods we require some source of random numbers. This source is then used for the construction of probability distributions. Nature itself would be a great source for such random numbers, however, most processes do not generate random numbers fast enough for our needs. Additionally, most computer systems do not have any access to naturally occurring random numbers.

Nevertheless, it is possible to generate artificial random numbers. Here a long sequence of numbers is created, which does not follow any pattern. The numbers appear to be absolutely random *for all practical purposes*. A random number generator depends on its generating sequence by definition. A very simple implementation of such a sequence just considers the previous state. Here the i-th random number r_i would be generated by

$$r_i = f(r_{i-1}). \tag{2.1}$$

One problem with Equation (2.1) is that we need to supply a proper r_0 as initial condition. Otherwise, no argument for the generating function f is given. We call

r_0 the seed. Ideally, the seed is randomly determined. We might fake an ideal seed by taking a frequently changing, arbitrary value. An example for such a value is the current system time in, e.g., 1000-cycles.

Another problem is that since f is not random, having two equal values $r_i = r_j$, with $i \neq j$, will result in $r_{i+1} = r_{j+1}$. Since computer memory is limited we will hit this point eventually. Thus, we have a repeated sequence after a certain period of generated numbers. Ideally, this period should be as large as possible, however, by just using the previous value we obtain a period that is at best 2^N, where N is the number of bits used to represent the value.

One of the oldest and best-known pseudo random number generator (PRNG) algorithms is the linear congruential generator (LCG). Here we have

$$f(x) \equiv (ax + c) \bmod m, \tag{2.2}$$

where we call a the multiplier, c the increment and m the modulus. In case of $c = 0$ the LCG is often called Lehmer RNG [86]. In practice the LCG is too limited and therefore should only be used with great care. The period of a general LCG is at most the modulus m with typical implementations using $m = 2^{32}$.

A much better choice is given by the multiply-with-carry (MWC) [93] algorithm. It features a very fast generation of random numbers with immense periods, ranging from 2^{60} to $2^{2000000}$. The reason for the improved period length lies in using two equations instead of one. The second equation is used to describe a dynamic increment. We have

$$r_i = f(r_{i-1}, c_{i-1}), \tag{2.3}$$

$$c_i = g(r_{i-1}, c_{i-1}). \tag{2.4}$$

The function f is defined as with the LCG, however, instead of a constant c we use the index-dependent value c_{i-1}. The function g is given by

$$g(x, c) \equiv \lfloor (ax + c)/m \rfloor , \tag{2.5}$$

i.e., we require initial conditions c_i and r_i for both sequences. Of course we could couple the sequences by defining the seed of the second sequence from the first sequence, however, this will certainly lower the period.

Alternatively, we might skip any PRNG and base our simulation directly on a device that uses either classical chaos, e.g., atmospheric noise, or quantum mechanics. Currently there are only external commercial products available. In the near futures central processing units (CPUs) will be equipped by default with a hardware RNG [96] that can be accessed via special machine instructions.

We assert that our RNG is sufficiently fast, while providing a long period and being insensitive to seeds. In the ideal case the generated sequence should be non-correlated. The latter cannot be achieved by PRNG, but we can get very close. In our simulations we either use an implementation of the MWC or the Mersenne Twister [94] PRNG, which has a period length of $2^{19937} - 1$.

2.1.2 Distributions

The algorithms discussed in the last section only generate uniformly distributed random numbers. If we need to use another distribution, we have to look at the desired probabilities first. We start with a uniform random number $u \in [0, 1)$. For obtaining a random number x distributed with $p(x)$ in the interval $[a, b)$ we have

$$P(a \leq x < b) = \int_a^b dx\, p(x) \equiv F(b). \tag{2.6}$$

Now we need to find the inverse of $F(b)$ to get a random number x following the given distribution: $x = F^{-1}(u)$. However, this method is only feasible if the integral can be inverted easily. Therefore, it works only for some distributions, e.g., the exponential distribution, which is given by $p(x) = \lambda \exp(-\lambda x)$ for $x \geq 0$, where $\lambda \in \mathbb{R}_+$ is called the decay rate. In this case we are able to invert the integral to obtain

$$x = -\lambda^{-1} \log(1 - u). \tag{2.7}$$

An example that does not work as smoothly is the normal distribution. The normal distribution is defined to be

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{2.8}$$

where σ is the standard deviation and μ is the expectation value. We cannot find an analytic inverse in terms of elementary functions in the one-dimensional case, but we are able to perform the inversion in two dimensions. We get

$$n_1 = \sqrt{-2\log(1-u_1)}\sin(u_2), \tag{2.9}$$

$$n_2 = \sqrt{-2\log(1-u_1)}\cos(u_2). \tag{2.10}$$

We should implement some buffering to use the already computed result n_2 later. Refusing to store the other solution wastes computing power by throwing away useful numbers.

In general, we can always transform uniformly distributed random values by the accept or reject method. We start by looking for a simple distribution $h(x)$ that

acts as a boundary for $p(x)$, i.e., $p(x) \leq \lambda h(x)$ for some $\lambda > 1$.

The accept or reject method then works by drawing a random number x according to the distribution given in $h(x)$. The value as sample of $p(x)$ is accepted if a second uniformly distributed random number t, with $0 \leq t \leq \lambda h(x)$ satisfies $t \leq p(x)$. If the proposed change is rejected, the procedure has to be repeated until a random number is accepted. This requires a good guess of $h(x)$ to be efficient. Otherwise, a numerical inversion of the integral of $p(x)$ might be faster.

We only looked at continuous distributions so far. The way to generate random numbers from discrete distributions, such as the discrete uniform distribution, the Bernoulli distribution, which is a special case of the Binomial distribution, or the Poisson distribution, is very similar to the continuous case and does not require additional knowledge.

2.1.3 Markov Chains

A Markov chain describes the transition from one state to another in a mathematical system. It is specified by a state space S and by a transition matrix P_{xy}, with $x, y \in S$ such that

$$P_{xy} \geq 0, \qquad \sum_{y \in S} P_{xy} = 1. \tag{2.11}$$

The state space is not restricted to a discrete space and the sum can be understood as an integral. This construction yields two basic features:

- The probability to be in y at time $t+1$ depends only on the position at time t. Hence we do not require to know the past, only the present.
- The probabilities are time-independent. Again, as we do not need to know about the past, the probability is independent of any previous change and therefore time-independent in general.

Among all Markov chains we only consider processes which satisfy the conditions of *ergodicity* and *aperiodicity*. The condition of ergodicity states that we can basically go from any state to any state. There are no restrictions. Therefore, for any x and y there exists a $n > 0$ such that the transition matrix obeys $P^n_{xy} > 0$. Ergodic Markov chains are called irreducible.

Aperiodicity is closely related to the periodic length of randomness. It is fulfilled if the greatest common divisor of the set of integers n satisfying $P^n_{xx} > 0$ is 1. This means that there is no recognizable pattern in the Markov chain.

If P is irreducible and aperiodic we can prove that

$$\lim_{n \to \infty} P^n_{xy} = \pi_y, \tag{2.12}$$

with $0 \leq \pi_y < 1$. In case of $\pi_x \neq 0$ for any x it actually satisfies

$$\sum_x \pi_x = 1. \tag{2.13}$$

Combining Equation (2.12) and Equation (2.13) results in the so-called stationary condition, expressed as

$$\sum_x \pi_x P_{xy} = \pi_y, \tag{2.14}$$

where we call π_x the equilibrium distribution.

The probability described by the equilibrium distribution is unique. We see that π is determined by P, however, in practical applications this is usually be turned upside down. In practice the equilibrium distribution π is known and can be identified.

Then, we devise a transition matrix P such that π satisfies the stationary condition for P. The uniqueness property guarantees that π is the equilibrium distribution of the process. Finally, we can compute expectation values of observables by calculating averages from the Markov process. This follows directly from the ergodic theorem.

As there is an infinite number of matrices P that satisfy the stationary condition, it is often easier to look for a matrix P which satisfies the even stronger condition

$$\pi_x P_{xy} = \pi_y P_{yx}, \tag{2.15}$$

for any x, y. This condition is called *reversibility condition*, or *detailed balance*. It implies the stationary condition. Hence we will always try to ensure detailed balance, as this is an easy way to satisfy the stationary condition.

Even if all conditions are satisfied we may need further development depending on our model, e.g., successive graphs in a Markov chain cannot be used when we want to obtain independent draws. The realizations are usually correlated [123]. We need to optimize our drawing process to maximize the sampled phase space. Even then autocorrelation, as discussed in Section 2.2.4, is an important issue.

2.1.4 Monte Carlo Integration

Monte Carlo integration is a non-deterministic method of estimating the value of an integral [149]. The method is based on random numbers and averages over all function values at independently selected points. It is very efficient for solving multi-dimensional integrals and sums. We start with a D-dimensional integral over a domain Γ of volume V normalized to 1,

$$I = \int_\Gamma d^D x \, f(\mathbf{x}). \tag{2.16}$$

For reasons of simplicity we assume Γ to be a D-dimensional hypercube with the components of $\mathbf{x}$ chosen to satisfy $0 \leq x_k \leq 1$ for $k \in [1, D]$.

We can now independently generate N random vectors $\mathbf{x}_i$ following the uniform distribution of Γ. Each vector is then used per iteration (called "timestep") to evaluate the integrand $f(\mathbf{x}_i)$. The sum of these evaluations ("measurements") approximates the value of the integral,

$$I \approx I_N \equiv N^{-1} \sum_{i=1}^{N} f(\mathbf{x}_i). \tag{2.17}$$

The estimated error scales like

$$\Delta I_N \propto N^{-1/2}, \tag{2.18}$$

which is independent of D.

If D is small, Monte Carlo integration performs worse than standard methods for numerical integration. However, for large D it turns out to be much better than the standard methods. For example, the trapezoidal quadrature has an estimated error of $\Delta I_N \propto N^{-2/D}$ [111]. Therefore, Monte Carlo integration is a possible way to circumvent the curse of dimensionality [71].

From a practical point of view N becomes too large very quickly for the usual methods of numerical integration. In 10 dimensions we already require 10^{10} points for 10 points in each direction. This is just not feasible.

In general, the domain Γ is of volume $V \neq 1$, i.e., the volume has to contribute to the value of the integral. As the volume is related to the probability of picking a specific point $\mathbf{x}_i$, we can simply transform the original integral to study how the chosen probability distribution contributes to the Monte Carlo integration.

For a probability distribution $p(\mathbf{x})$ we always assume proper normalization, i.e., we have

$$\int_\Gamma d^D x \, p(\mathbf{x}) = 1. \tag{2.19}$$

We now transform our integral by changing

$$I = \int_\Gamma d^D x \, f(\mathbf{x}) = \int_\Gamma d^D x \, f(\mathbf{x}) \, \frac{p(\mathbf{x})}{p(\mathbf{x})} = \int_\Gamma d^D x \, p(\mathbf{x}) \, h(\mathbf{x}). \tag{2.20}$$

We obtained an expression that gets integrated using the distribution $p(\mathbf{x})$. The general expression for computing an integral using a Monte Carlo integration can be expressed as

$$I_N = N^{-1} \sum_{i=1}^{N} h(\mathbf{x}_i), \tag{2.21}$$

with the values $\mathbf{x}_i$ being chosen according to the distribution given by $p(\mathbf{x})$.

Monte Carlo integrations perform better if $p(\mathbf{x})$ is chosen to be close to $f(\mathbf{x}_i)$ for all $\mathbf{x}_i$. In this case $h(\mathbf{x}_i)$ is close to being constant. The ideal choice for the probability distribution is given by $p(\mathbf{x}) \propto |f(\mathbf{x})|$. We call the method of using a better suited distribution for generating the arguments of the integral importance sampling.

Importance sampling plays a crucial role in Monte Carlo methods. In practice most of the benefits from using Monte Carlo methods actually come from using appropriate distributions, which are similar to the functions being evaluated. Importance sampling effectively reduces the error by making our evaluations more accurate within a limited number of timesteps.

2.1.5 Monte Carlo Simulations

Monte Carlo simulations have proven to be very efficient in solving statistical problems. The general algorithm of Monte Carlo methods varies from method to method, but they all follow a particular pattern. First we need to define a domain of possible inputs. Now we can generate inputs randomly from a probability distribution over the domain. Then we perform a deterministic computation using the generated inputs. In a final step we aggregate and interpret the results using statistical methods.

We start by demanding that an integration of a valid D-dimensional positive scalar weight function $w(\mathbf{x})$ over the phase space Γ has to yield unity. We express this condition by

$$\int_\Gamma d^D x\, w(\mathbf{x}) = 1, \qquad w(\mathbf{x}) > 0. \tag{2.22}$$

As previously derived in Monte Carlo integrations we may use a weight function to rewrite an existing integral over a given function $f(\mathbf{x})$. In effect we alter the integrand and adjust the integral to a more suitable distribution.

We start by introducing the positive weight function, $w(\mathbf{x})$, to get

$$I = \int_\Gamma d^D x\, w(\mathbf{x})\, \frac{f(\mathbf{x})}{w(\mathbf{x})}, \tag{2.23}$$

which allows us to regard the weight function as the distribution to apply in importance sampling. At this point we specialize the formulation to a general problem in statistical mechanics. We start with

$$\langle A \rangle = Z^{-1} \int_\Gamma d^D x\, \exp(-\beta H(\mathbf{x}))\, A(\mathbf{x}), \tag{2.24}$$

where Z is the partition function expressed as

$$Z = \int_{\Gamma} d^D x \, \exp(-\beta H(\mathbf{x})). \tag{2.25}$$

The energy function is given by the Hamiltonian $H(\mathbf{x})$. The function we are interested in is the observable $A(\mathbf{x})$. The temperature of the system is proportional to the inverse value of β.

Coming back to the more general problem formulation we see that we can now consider using

$$w(\mathbf{x}) = Z^{-1} \exp(-\beta H(\mathbf{x})) \tag{2.26}$$

as our weight function. This is an efficient choice for importance sampling. Now we can approximate the solution for $\langle A \rangle$. We have

$$\langle A \rangle \approx N^{-1} \sum_{i=1}^{N} A(\mathbf{x}_i). \tag{2.27}$$

In the limit of $N \to \infty$ we find the exact solution as the error goes to zero. We call $\mathbf{x}_i$ a configuration. How do we generate a Markov chain for the required N configurations efficiently?

The easiest way to generate a Markov chain is to use the Metropolis-Hastings methods [68], which is a generalization to the method proposed by Metropolis et al. [98]. The algorithm defines the transition from one element to another. A new configuration $\mathbf{x}_{i+1}$ is proposed using a selection probability in conjunction with the current configuration $\mathbf{x}_i$. The selection probability is denoted with

$$P_{\text{prop}}(\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i). \tag{2.28}$$

We accept the new configuration with probability

$$P_{\text{acc}} = \min\left(1, \frac{P_{\text{prop}}(\mathbf{x}_i \leftarrow \mathbf{x}_{i+1})}{P_{\text{prop}}(\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i)} \frac{\Omega(\mathbf{x}_{i+1})}{\Omega(\mathbf{x}_i)}\right), \tag{2.29}$$

where $\Omega(\mathbf{x}_i)$ represents the distribution for the values of the i-th configuration.

If we reject the update, we continue with the current configuration, i.e., $\mathbf{x}_{i+1} = \mathbf{x}_i$. The update is accepted if a generated uniform random number $r \in [0, 1]$ satisfies

$$r \leq P_{\text{acc}}. \tag{2.30}$$

In general, the updates in a Monte Carlo simulation should be optimized in several ways. There are many studies on optimal updating schemes for various models [115]. As a rule of thumb we want P_{acc} to be around 1/4. If P_{acc} is too high,

the integration volume is explored too slowly. A smaller value wastes computer time by rejecting too many updates. For simulations with very expensive updates it is economical to aim for a higher value of the acceptance rate P_{acc}.

It can be shown that the Metropolis-Hastings algorithm satisfies detailed balance. Thus, we ensure that the limiting distribution is given by $\Omega(\mathbf{x})$.

2.1.6 Monte Carlo Simulation of He^3-He^4 Mixtures

The basic idea of a Monte Carlo simulation can be illustrated by using an example: We simulate the behavior of the He^3-He^4 mixture [37]. The model in this example is using two parameters, denoted by β and μ. The former represents the temperature of the system, while the latter is called the anisotropy field.

He^3-He^4 mixtures are very important for cooling systems, which have to provide temperatures lower than 1 K. The process where these mixtures are used is called dilution refrigeration. The physical background is that by mixing these isotopes of Helium thermal energy is absorbed by the system to enable a phase transition.

In more detail, energy is required to transport the He^3 atoms from the He^3-rich phase into the He^3-poor phase. If the atoms can be made to continuously cross this boundary, they effectively cool the mixture. Because the He^3-poor phase cannot have less than a few percent He^3 at equilibrium, even at absolute zero, dilution refrigeration can be effective at very low temperatures. The volume in which this takes place is known as the mixing chamber.

For our model a tricritical point has been predicted [20], which is where a line of second-order phase transitions meets a line of first-order phase transitions. A first-order phase transition exhibits a discontinuity at the first derivative of the free energy, while a second-order phase transition is continuous in the first derivative with a discontinuity in the second derivative of the free energy.

Compared to the Hamiltonian described in [20] we are only interested in the $K \to 0$ case. Our choice of units sets $J = 1$, with $\Delta = \mu_3 - \mu_4$. It is convenient to set $\mu_3 = 0$ introducing $\mu \equiv -\Delta$ in units of J.

The Hamiltonian to study is now given by

$$\hat{H} = -\sum_{\langle i,j \rangle} s_i s_j - \mu \sum_i s_i^2. \tag{2.31}$$

where μ is used to vary the concentration. A large value of μ results in a higher concentration of He^3 in the system. In our simulation we will bring the system into equilibrium for any choice of β and μ. Finally, we try to find the values representing the line of the phase transition.

We use s_i for the value of the spin of i-th site. The spin can take the values 0

(He^3) and ± 1 (He^4). The magnetization is computed via

$$M = N^{-1} \sum_{i=1}^{N} \langle s_i \rangle, \tag{2.32}$$

where N denotes the total number of sites.

Since we have two different types of particles in the system we need a way to distinguish them. We can obtain the number of He^3 atoms, N_3, and He^4 atoms, N_4, by calculating

$$N_3 = \sum_{i=1}^{N} (1 - s_i^2), \tag{2.33}$$

$$N_4 = \sum_{i=1}^{N} s_i^2 = N - N_3. \tag{2.34}$$

For choosing the selection probability $P_{\text{prop}}(\phi' \leftarrow \phi)$ we have two options. Either we update the sites randomly or sequentially. The latter is more straightforward and requires less random numbers as we use a deterministic updating procedure.

In each timestep we propose a new value for a single site. This is a very simple approach and may be replaced by cluster algorithms [121] in practice. Especially the latter has been applied in many cases successfully. An example is the evaluation of path integrals [22].

We may want to speed up our simulation by using a predefined lookup-table. We can only use a lookup-table if we perform local updates, i.e., if the selection probability is independent of the starting configuration ϕ. We have

$$P_{\text{prop}}(\phi' \leftarrow \phi) \propto \exp\left(-S(\phi')\right). \tag{2.35}$$

Note that this implies that the ratio of the proposal to its inverse is only dependent on ϕ and ϕ', i.e.,

$$\frac{P_{\text{prop}}(\phi' \leftarrow \phi)}{P_{\text{prop}}(\phi \leftarrow \phi')} = \frac{\exp(-S(\phi'))}{\exp(-S(\phi))}. \tag{2.36}$$

In our case the lookup-table is a 3×9 matrix. We have three possible states for each site with the sum of the energies of the four nearest neighbors of each site yielding discrete results between -4 and 4.

We want to reproduce the critical line (β_c, μ_c), which gives us information about the phase transition of the mixture. We can then compare our numerical result with an approximation for $\mu_c(\beta_c)$, given by

$$\mu_c = -\frac{\ln(\beta_c - 1)}{\beta_c}. \tag{2.37}$$

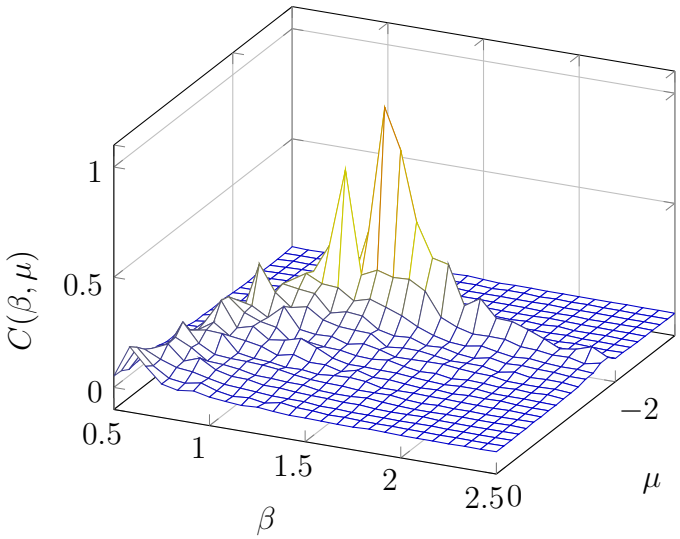


Figure 2.1: Specific heat capacity in the (β, μ)-plane. The phase transition and the tricritical point can be seen.

The solution estimate can only be evaluated for $\beta_c > 1$.

For our numerical evaluation we use L^2 number of sites, with L being set to 64. To obtain enough statistics we record at least 250 configurations during the 2.5×10^6 timesteps per parameter value. Each configuration is obtained in equilibrium state.

The tricritical point of the system (β_c, μ_c) can be found by looking, e.g., at the maximum of the specific heat capacity $C(\beta, \mu)$. The specific heat capacity is defined as the variance of the energy. We can compute it via

$$C(\beta, \mu) = \beta^2 \left(\left\langle H(\beta, \mu)^2 \right\rangle - \langle H(\beta, \mu) \rangle^2 \right). \tag{2.38}$$

With our simulations we get an estimate of

$$(\beta_c, \mu_c) = (1.34(2), -1.89(3)). \tag{2.39}$$

The plot in Figure 2.1 shows the obtained data. The rebinning scheme described in Appendix C.2 has been applied to prepare the data for display. Using the variance of the energy we are able to detect all the interesting regions in the energy landscape of the underlying model.

Depending on our objective we will either use a cold start, where every site is set to the same value, or a hot start, which randomizes the values. Independent of our choice, we have to decide for a number of thermalization updates. Within the thermalization we do not perform measurements. If the measurement is expensive, we should reduce the frequency of measurements. This will reduce the autocorrelation, i.e., exclude configurations, which are too similar and therefore

meaningless.

In the next section we discuss our set of statistical tools, which give us indicators to decide for the number of thermalization updates and the measurement frequency.

2.2 Statistical Tools

Once we leave the world of determinism, we will find our answers purely in statistics. This requires an introduction to statistical methods as well as defining a common language.

Of course, the whole topic serves material for many books. A more general and detailed discussion can be found in various textbooks, e.g., [59]. We only define terms and introduce quantities, which will then be used throughout this thesis. We start by introducing the most basic quantities.

2.2.1 Basic Quantities

One of the most important quantities in statistics is the *mean* of a data sample. Sometimes the mean is called the average of a given dataset. The mean can be easily computed by dividing the aggregate of all data points in the sample by the number of data points N.

If these N data points given by $\mathbf{x} = \{x_i\}$, where $i = 1, \ldots, N$, are distributed with probability density $p(x)$, then the sample mean for a function g is

$$\langle g(\mathbf{x}) \rangle = N^{-1} \sum_{i=1}^{N} g(x_i). \tag{2.40}$$

The mean of the sample itself with $g(x) \equiv x$ is denoted by $\langle x \rangle$.

In the statistical limit of $N \to \infty$ this converges to the so-called *expected value*, which can be computed by evaluating the integral over all weighted values in the domain Ω,

$$\langle g(\mathbf{x}) \rangle_p = \int_\Omega dx\, g(x)\, p(x). \tag{2.41}$$

Of course the arithmetic mean and the expected value can be equal. This will happen if the probability density is a constant, i.e., $p(x) = \|\Omega\|^{-1}$. We call this special probability function the uniform probability.

The *deviation* is the difference from the data points to their mean, i.e., $g(x_i) - \langle g(\mathbf{x}) \rangle$. This leads to the *variance*, which is the mean of the squared deviation entries, i.e.

$$\mathrm{var}\left(g(\mathbf{x})\right) = N^{-1} \sum_i \left(g(x_i) - \langle g(\mathbf{x}) \rangle\right)^2. \tag{2.42}$$

We can calculate the *standard deviation* by taking the square root of the variance. That way we obtain a relation between the deviation and the standard deviation[1].

A very interesting quantity is the *covariance*. The covariance can be interpreted as a the correlation between two datasets. This helps us to find out whether two generated datasets are independent or follow a similar pattern. We can define the covariance of two sets $\mathbf{x}$ and $\mathbf{y}$ as the scalar product of the deviation vectors,

$$\operatorname{cov}(\mathbf{x}, \mathbf{y}) = N^{-1} \sum_{i=1}^{N} (x_i - \langle \mathbf{x} \rangle)(y_i - \langle \mathbf{y} \rangle). \tag{2.43}$$

The definition is very useful to compute the variance of a single set. The variance is obtained by applying the definition of the covariance on a single set, i.e.,

$$\operatorname{var}(\mathbf{x}) = \operatorname{cov}(\mathbf{x}, \mathbf{x}). \tag{2.44}$$

Any simulation can be considered finished once the set of data points have converged and fluctuations are reduced to a minimum. Therefore, the rate of convergence is an important quantity, since it can be used to estimate the number of required timesteps. We can compute the rate of convergence by calculating the average of the square deviations.

Finally, we should introduce the term *standard error* (of the mean). The standard error is closely related to the variance and is interpreted as the error on the mean. It is root of the squared deviations divided by N,

$$\Delta x = \sqrt{\frac{1}{N^2 - N} \sum_{i=1}^{N} (g(x_i) - \langle g(\mathbf{x}) \rangle)^2}. \tag{2.45}$$

The standard error may be given in absolute or relative terms. The absolute error is given by Equation (2.45). In contrast the relative error is the ratio of the absolute error to the computed average value.

In Monte Carlo simulations the standard error over all possible samples, which are independently drawn using the same size N, scales like $1/\sqrt{N}$. We can see that by calculating the standard error of a single sample $\bar{x}$. We find

$$\Delta \bar{x}^2 = \left\langle (\bar{x} - \langle \bar{x} \rangle)^2 \right\rangle = \Delta x^2 / N. \tag{2.46}$$

This is the basic feature of any Monte Carlo algorithm as discussed in Section 2.1.

Monte Carlo simulations impose some bias on the data. It is required to reach the equilibrium distribution from Equation (2.14) before performing any measure-

[1] The relation is expressed by another common name for the standard deviation, namely the *root mean square deviation*.

ments. The number of timesteps to reach equilibrium is strongly related on the model and can be estimated using the autocorrelation time from Section 2.2.4. Additionally, data resampling techniques, such as *Jackknife* or *Bootstrap* are useful.

2.2.2 Jackknife

Jackknife [144] is a method that reduces the dataset to compute several error estimates. Jackknife is usually applied in conjunction with data binning. This way we can get rid of unwanted fluctuations in the dataset. If it is reasonable to assume that these fluctuations are independent we could just compute means and standard errors. However, this assumption is usually wrong. Every Monte Carlo simulation produces correlated fluctuations, which need to be accounted.

Jackknife itself is a generic and cheap estimator for the standard error without having to worry about the propagation of uncertainty. The N data points of our dataset $\mathbf{x}$ get reduced to $N-1$ resampled values. The ordinary statistical analysis is then applied on this set of resampled values. Doing this N times results in a set of mean values $\mathbf{J} = \{J_1, \ldots, J_N\}$.

Computing the standard error yields

$$\varepsilon_J^2 = \frac{(N-1)}{N} \sum_{i=1}^{N} \left(j_i - \langle \mathbf{x} \rangle\right)^2 , \tag{2.47}$$

where $\langle \mathbf{x} \rangle$ is the mean of the full sample.

The bias b of our data can be estimated by using the mean of all Jackknife samples $\langle \mathbf{J} \rangle$ again. We have

$$b = (N-1)\left(\langle \mathbf{J} \rangle - \langle \mathbf{x} \rangle\right). \tag{2.48}$$

We already mentioned that Jackknife is even better if the source dataset is binned. Binning is a technique, where we block our dataset using a number of bins [152]. We can then compute any observable for each bin and estimate the error by applying the standard error of Equation (2.45) to these values. This alone is mostly sufficient to lower the bias significantly.

In general, we obtain more reliable numbers by using Jackknife than compared to plain averages and variances. A reason for this is that Jackknife sample means are distributed $N-1$ times closer to the mean than the original data points. It can be shown that the following relation is true for any Jackknife sample $\mathbf{J}$,

$$J_i - \langle \mathbf{x} \rangle = (N-1)^{-1} \left(\langle \mathbf{x} \rangle - x_i\right), \tag{2.49}$$

since J_i is the mean of $\mathbf{x} \backslash x_i$.

2.2.3 Boostrap

Another method that is able to reduce the bias and compute more robust means and errors is the Bootstrap [47]. While Jackknife is a deterministic process, Bootstrap relies on random numbers[2]. The idea is similar to Jackknife, however, instead of leaving out the i-th element for computing the i-th value of the mean over the remaining data points we select the these points randomly. We try to compute more accurate samples by reusing our data points.

We start again with N data points $\mathbf{x}$. We sample these N points with replacement. The ordinary statistical analysis is then applied on this new set of resampled values. Doing this K times results in a set of mean values $\mathbf{B} = \{B_1, \ldots, B_K\}$.

Computing the standard error yields

$$\varepsilon_B^2 = K^{-1} \sum_{i=1}^{K} \left(B_i - \langle \mathbf{B} \rangle\right)^2 , \tag{2.50}$$

where B_i itself is $N^{-1} \sum_{i=1}^{N} x_{r(i)}$, with a mapping function r. The mapping function is chosen to follow a discrete uniform distribution.

2.2.4 Autocorrelation

Ideally, our sample $\{x\}$ of consecutive measurements of a single observable in a Markov chain contains data points x_i that are completely uncorrelated. In this case, the correlation between two data points factorizes to two independent quantities, i.e.,

$$\langle x_i \, x_j \rangle = \langle x_i \rangle \, \langle x_j \rangle \, . \tag{2.51}$$

However, in reality we will never measure perfectly uncorrelated values. In order to find out how strongly our data points are correlated we can introduce a function, which measures the strength of correlation between the data points.

The function examines two points and tells us how strong the correlation is between them. We have

$$C(x_i, x_{i+t}) = \langle x_i x_{i+t} \rangle - \langle x_i \rangle \, \langle x_{i+t} \rangle = \langle (x_i - \langle x_i \rangle)(x_{i+t} - \langle x_{i+t} \rangle) \rangle \, . \tag{2.52}$$

By assuming invariance under the offset i, we find a function that only depends

[2] Actually, the process of choosing points may be deterministic, e.g., by always using the same seed for the PRNG.

on the shift t. We call $C(t)$ the autocovariance. We approximate

$$C(t) = \langle (x_0 - \langle x \rangle)(x_t - \langle x \rangle) \rangle \approx N^{-1} \sum_{i=1}^{N} C(x_i, x_{i+t}), \tag{2.53}$$

where we use the sum over all values to obtain a good estimate.

The autocovariance is used to express the autocorrelation function, which is a normalized version of the autocovariance starting at 1. Ideally, the autocorrelation function yields values in $[-1, 1]$, where we interpret -1 as anti-correlated data, 1 as correlated data, and 0 as uncorrelated data.

The autocorrelation function $\Gamma(t)$ is thus given by

$$\Gamma(t) \equiv \frac{C(t)}{C(0)}. \tag{2.54}$$

The function is a sum of many different contributions. However, in the simplest case an approximation using a single exponential decay $\exp(-\lambda t)$ is sufficient. In this case, we can compute the so-called exponential autocorrelation time τ_{exp} by identifying

$$\Gamma(t) \sim \exp\left(-\frac{t}{\tau_{\text{exp}}}\right). \tag{2.55}$$

As this is an approximation that might be insufficient for a particular case, we will only use this quantity in comparison.

The property of translation invariance allows us to reduce the dataset to the first $n \ll N$ points. Even though n is much smaller than the whole dataset, it has to be much larger than the autocorrelation time. A sufficiently good estimate should use $\mathcal{O}\left(10^3 \tau\right)$ data points.

The autocorrelation time has to be taken into account when calculating (non-binned) standard errors from the estimated variance where we have to divide by the number of independent data points. This, however, is in general not the total number of data points.

Adjusting the variance from Equation (2.44) to consider the correlation between the data points, we get

$$\begin{aligned} \text{var}\left(\mathbf{x}\right) &= \frac{1}{N^2} \left\langle \sum_{i,j=1}^{N} (x_i - \langle x \rangle)(x_j - \langle x \rangle) \right\rangle = \frac{1}{N^2} \sum_{i,j=1}^{N} C(|i-j|) \\ &= \frac{1}{N^2} \sum_{i=1}^{N} \sum_{t=-N}^{N} C(|t|) \approx 2 \left(\frac{1}{2} + \sum_{t=1}^{N} \Gamma(t) \right) \frac{C(0)}{N} \equiv 2\tau_{\text{int}} \frac{C(0)}{N}, \end{aligned} \tag{2.56}$$

where we call τ_{int} the integrated autocorrelation time. The number of data points

N is now corrected accordingly. We obtain

$$N_{\text{independent}} = \frac{N}{2\tau_{\text{int}}}. \tag{2.57}$$

Here $N_{\text{independent}}$ us the number of independent data points. This replaces most factors N in error calculations and yields much more accurate results. We will implicitly use $N \equiv N_{\text{independent}}$.

In practice we estimate the integrated autocorrelation time by calculating

$$\tau_{\text{int}} = \frac{1}{2} + \sum_{t=0}^{W} \Gamma(t), \tag{2.58}$$

where we choose W in accordance with [152]. A valid scheme is to take the smallest $W \in \mathbb{N}$ that fulfills

$$\sum_{t=0}^{W+1} \Gamma(t) < \sum_{t=0}^{W} \Gamma(t), \tag{2.59}$$

thus implying $\Gamma(W+1) < 0$.

The autocorrelation has a huge effect on our simulations, especially during phase transitions, e.g., see example in Section 2.1.6. For first-order phase transitions the exponentially large tunneling $\tau \sim \exp\left(L^{d-1}\right)$ [142], with L sites in d dimensions, can be observed. The second-order phase transition shows critical slowing down $\tau \sim L^2$ [129].

2.3 Special Relativity

The theory of special relativity connects time and space by obeying a central axiom that introduces the concept of an observer independent limit velocity. Thus, the speed of light in a vacuum is the same for all observers regardless of the motion relative to the light source. The laws of physics are invariant in all inertial systems.

We define the infinitesimal line element ds as

$$(ds)^2 \equiv c^2(dt)^2 - (d\mathbf{x})^2. \tag{2.60}$$

The definition of the line element in Equation (2.60) sets the metric that has to be used for connecting time and space components. We call the metric $\eta_{\mu\nu}$ the Minkowski metric. It is given by a 4×4 matrix of the form[3]

$$\eta = \text{diag}\,(1, -1, -1, -1)\,, \tag{2.61}$$

[3] There are two conventions for the metric, which are different by a factor -1. We use the convention found in [17].

where we use Greek indices (within $0, \ldots, 3$) to access single elements. The symbol $\eta_{\mu\nu}$ retrieves the element from the μ-th row and ν-th column. A dot product of two four-vectors a and b yields

$$a \cdot b = a_\mu \, b^\mu = \eta_{\mu\nu} \, a^\mu \, b^\nu, \tag{2.62}$$

with indices as subscript being used for covariant vectors. Superscript indices mark contravariant vectors. If we encounter a pair of co- and contravariant vectors that use the same index, we perform an implicit scalar product. This is the Einstein summation convention. In contrast to the ordinary Euclidean metric we need a notation to distinguish between co- and contravariant vectors since η is not the unit matrix.

The group of distance preserving functions in Minkowski space is called the Poincaré group. It is a non-Abelian Lie group with 10 generators that contains symmetries under transformations such as translations, rotations, and boosts.

Translation invariance is responsible for energy and momentum conservation. Rotation invariance leads to angular momentum conservation and invariance under boosts to conservation of the center of mass. These two symmetries can be found in the Lorentz group. If a physical theory is Lorentz invariant, it is thus independent of the orientation and velocity of the observer.

The Minkowski metric has some properties that tell us directly what kind of element we are dealing with. There are three types of vectors. Vectors are either called

- timelike (for $ds^2 > 0$),
- spacelike (for $ds^2 < 0$), or
- lightlike (for $ds^2 = 0$).

Lightlike vectors are sometimes referred to as null vectors.

Any Lorentz transformation preserves the type of the vector. That is, a spacelike vector cannot be transformed into a lightlike or timelike vector and likewise for the other two.

It is possible to use the Euclidean metric $\delta_{\mu\nu}$ in the context of special relativity. By rotating the time component of x in the complex plane we transform $x_0 \to i x_0$. This is known as a Wick rotation. An illustration is sketched in Figure 2.2. This effectively changes the metric to $-\mathrm{diag}\,(1, 1, 1, 1)$, which is denoted by $-\delta_{\mu\nu}$. Finally, this yields the general rules for transforming any equation given

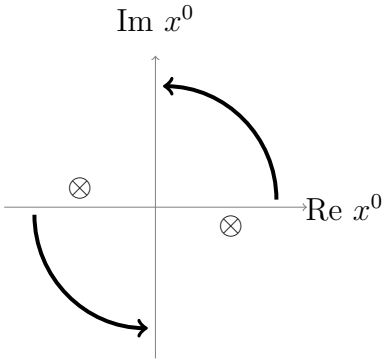


Figure 2.2: Wick rotation in the complex x^0-plane. A pole of the Feynman propagator is indicated by the $\otimes$ symbol.

in Minkowski space to Euclidean space:

$$x_0 \to i x_0, \tag{2.63}$$

$$x_j \to x_j, \tag{2.64}$$

$$d^4x \to i d^4x, \tag{2.65}$$

$$a_\mu \, b^\mu \to -a_\mu \, b_\mu, \tag{2.66}$$

where x, a, and b represent four-vectors. The index j denotes the spatial components of a Minkowski vector. In Euclidean space we cannot distinguish between co- and contravariant vectors. Hence there is no need to express dot products by placing the same index in sub- and superscript on the four-vectors. Instead we always use subscripts and apply the Einstein summation convention, i.e., we sum over repeated indices.

For completeness we already state here the transformation rules for the γ matrices, which is introduced in Section 2.4:

$$\gamma_0 \to \gamma_4, \tag{2.67}$$

$$\gamma_j \to -i\gamma_j, \tag{2.68}$$

$$\gamma_5 \equiv i\gamma_0\gamma_1\gamma_2\gamma_3 \to -\gamma_1\gamma_2\gamma_3\gamma_4. \tag{2.69}$$

Our specific representation of the Euclidean γ matrices are shown in Appendix B.1.

While in Minkowski space $x_\mu x^\mu = 0$ defines the light-cone $x^0 = \pm|\mathbf{x}|$, $x_\mu x_\mu = 0$ in Euclidean space implies $x = 0$. Hence possible singularities on the light-cone like x^{-2} turn into singularities at the point where x is precisely zero. This simplification of the singularity structure is due to the positive definite metric in Euclidean space. It is worth noting that in the Euclidean formulation we cannot distinguish between the three types of vectors.

In this thesis we almost exclusively work in Euclidean space. A prospective

Name	Flavor	Charge	Isospin	Mass
Up	u	$2/3$	$1/2$	$\sim$2 MeV
Down	d	$-1/3$	$-1/2$	$\sim$5 MeV
Charm	c	$2/3$	0	$\sim$1300 MeV
Strange	s	$-1/3$	0	$\sim$100 MeV
Top	t	$2/3$	0	$\sim$173 000 MeV
Bottom	b	$-1/3$	0	$\sim$4200 MeV

Table 2.1: Listing of quark flavors with some of their properties.

problem is a potential inconsistency of Minkowski space-time and Euclidean space-time [95]. However, such problems arise only in situations involving 3 or more reference frames, e.g., in certain particle collisions. Depending on the specific approach, calculations may give deviating results. In the following we avoid such cases.

2.4 Quantum Chromodynamics

QCD is the theory of the strong force, which describes how quarks interact with each other by the exchange of gluons. Quarks are the fundamental building blocks of hadrons, such as baryons. Baryons may appear in forms like protons and neutrons [137]. Quarks carry the so-called color[4] charge and interact via gauge bosons named gluons.

The theory of Quantum Chromodynamics (QCD) is special due to a yet unproven non-perturbative phenomenon called confinement [76]. In practice this means that independent of how much energy is used to separate two quarks from a hadronic bound state in vacuum, instead of freeing a quark from the bound state, the energy creates more quarks to yield a set of separated hadronic bound states. Even though no rigorous mathematical proof for confinement has been accomplished yet, there is enough experimental evidence to support its existence [38].

QCD is an important part of the Standard Model of particle physics. It embodies the complete knowledge about elementary particles, their cross-sections, and a framework for computing properties of particles [15]. It can be considered complete with the recent discovery of the Higgs boson [27], which was theoretically proposed in the mid 1960s [70].

The known quark flavors are shown in Table 2.1. These flavors can be grouped into three generations. The first generation contains the up and down quark, the

[4] The Greek word chroma means color.

second charm and strange, and the third generation the top and bottom quark. All quarks have spin $J = 1/2$ and baryon number of $B = 1/3$. Three quarks form a baryon, such as the proton or neutron. The isospin value is mostly interesting from an historical perspective, where it has been used as a new quantum number to explain symmetries of proton and neutron.

We start with a short derivation of the QCD action, as it can be found in standard text books such as [112, 134, 137].

2.4.1 Derivation

We construct the theory similar to Quantum Electrodynamics (QED), which is based on the assumption of local gauge symmetry $U(1)$. In the case of QCD the special unitary group $SU(3)$ turns out to represent the observed particle spectrum. As a result the interaction takes place via gluons, which carry color charges themselves. The origin of this self-interaction lies in the non-Abelian nature of the $SU(3)$ group.

We start by introducing the fermion fields, described by Dirac spinors

$$\psi_\mu(x), \tag{2.70}$$

where x is the space-time coordinate and μ is the spinor index $(1, \ldots, 4)$. The spinor structure represents one of the two spinor representations of the Lorentz group. We could add another index for the flavor, which would be limited by the number of flavors N_f in the theory. We suppress these indices for most of the discussion.

Similarly, we can represent antifermions. They are noted by

$$\bar{\psi}_\mu(x) \tag{2.71}$$

and lie in the conjugate representation, defined by $\psi^\dagger\gamma_0$.

Using the previously noted considerations we can write down the fermionic part of the QCD action:

$$S_{\text{ferm}} = \int d^4x\, \bar{\psi}_\mu(x)\, \left(D_m(x)\right)_{\mu\nu}\, \psi_\nu(x), \tag{2.72}$$

where $D_m(x)$ is the Dirac operator. It is given by

$$D_m(x) = \gamma_\mu \partial_\mu + m. \tag{2.73}$$

At this point the task is practically reduced to constructing an action, which is

invariant under the following transformations of the fermionic fields:

$$\psi(x) \to \psi'(x) = \Lambda^{-1}(x)\,\psi(x), \tag{2.74}$$

$$\bar{\psi}(x) \to \bar{\psi}'(x) = \bar{\psi}(x)\,\Lambda(x). \tag{2.75}$$

The object $\Lambda \in SU(3)$ is a local unitary Hermitian transformation matrix. To get a scalar action, the fermionic fields ψ and $\bar{\psi}$ obtain a color vector structure. Λ is generated by using real coefficients $\theta_a(x)$ in

$$\Lambda(x) = \exp\left(-i\,\theta_a(x)\,T^a\right). \tag{2.76}$$

In this case we introduce an additional color index $a \in \{1, \ldots, 8\}$. In the adjoint representation we have Hermitian traceless 3×3 matrices with 8 degrees of freedom.

From group theory we know that the generators T^a of the $SU(3)$ Lie algebra have to follow the general commutator

$$[T^a, T^b] = i\,f^{abc}\,T^c, \tag{2.77}$$

where the coefficients f^{abc} are real and totally antisymmetric structure constants. Usually, these generators are normalized to satisfy

$$\mathrm{tr}\left(T^a\,T^b\right) = \delta_{ab}/2. \tag{2.78}$$

The generators T^a are closely related to the Gell-Mann matrices λ^a. The T^a can be represented as $T^a \equiv \lambda^a/2$. They define matrices which act on the color structure of fermion fields $\psi(x)$ and $\bar{\psi}(x)$.

The action as defined by Equation (2.72) and Equation (2.73) is not invariant under the local gauge transformation from Equation (2.74) and Equation (2.75). We can fix that by introducing a new bosonic vector field. We denote the bosonic field to be

$$A_{\mu a}(x). \tag{2.79}$$

where we use the color index a, because the gauge field is in the eight-dimensional adjoint representation of the color group.

To complement the gauge transformation of the fermionic field, we impose a simultaneous transformation of the new field

$$A_\mu(x) \to A'_\mu(x) = \Lambda^{-1}(x)\,A_\mu(x)\,\Lambda(x) + i(\partial_\mu \Lambda^{-1}(x))\,\Lambda(x), \tag{2.80}$$

with $A_\mu(x) \equiv A_{\mu a}(x) T^a$.

To be gauge invariant the partial derivative in the Dirac operator of Equation (2.73) has to be replaced by the covariant derivative,

$$D_m(x) = \gamma_\mu \mathcal{D}_\mu(x) + m, \tag{2.81}$$

where the covariant derivative is defined as

$$\mathcal{D}_\mu(x) = \partial_\mu + igA_\mu(x). \tag{2.82}$$

g is the coupling constant, which determines the strength of interaction in the Lagrangian with respect to the kinetic part.

Taking the knowledge from QED we can derive the action for the gluonic part as a generalization of the QED action. In QED the field strength tensor $F_{\mu\nu}$ containing the photon field $A_\mu(x)$ is given by

$$F_{\mu\nu} = \partial_\mu A_\nu(x) - \partial_\nu A_\mu(x). \tag{2.83}$$

The coupling constant of QED is the elementary charge e. However, to stay consistent with Equation (2.82) we continue to express the coupling with g. Rewriting Equation (2.83) in terms of covariant derivatives we have

$$F_{\mu\nu} = -\frac{i}{g}\left[\mathcal{D}_\mu(x), \mathcal{D}_\nu(x)\right], \tag{2.84}$$

which can now be generalized to QCD to form the gluon field strength tensor $G_{\mu\nu}$. We find

$$G_{\mu\nu}(x) = -\frac{i}{g}\left[\mathcal{D}_\mu(x), \mathcal{D}_\nu(x)\right] = G_{\mu\nu a}(x)\, T^a \tag{2.85}$$

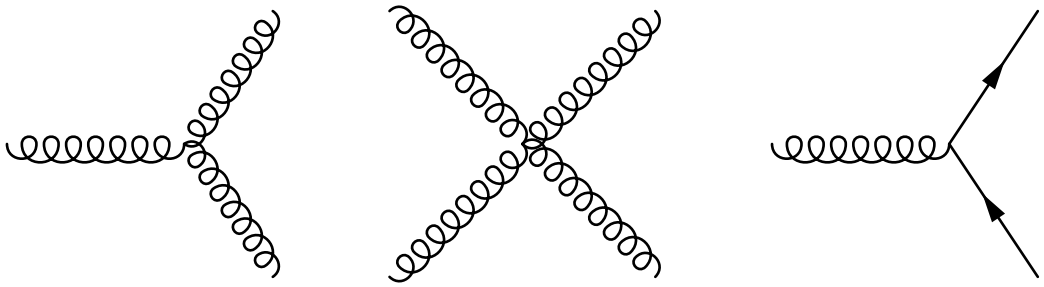
$$= \left(\partial_\mu A_{\nu a}(x) - \partial_\nu A_{\mu a}(x) - f^{abc}\, g\, A_{\mu b}(x)\, A_{\nu c}(x)\right) T^a. \tag{2.86}$$

$G_{\mu\nu}(x)$ transforms under Equation (2.80) exactly as desired, since it has been constructed with the commutator of two covariant derivatives. The transformation yields

$$G_{\mu\nu}(x) \to G'_{\mu\nu}(x) = \Lambda(x)\, G_{\mu\nu}(x)\, \Lambda^{-1}(x). \tag{2.87}$$

Therefore, in analogy to QED the full action of QCD is given by

$$S_{\mathrm{QCD}} = \int d^4x\, \mathcal{L}_{\mathrm{QCD}}, \tag{2.88}$$



(a) Three-gluon vertex (b) Four-gluon vertex (c) Quark-gluon vertex

Figure 2.3: Fundamental QCD vertices for gluon self-interaction and quark interaction.

with the Lagrangian $\mathcal{L}_{\text{QCD}}$ summing over all flavors f,

$$\mathcal{L}_{\text{QCD}} = \sum_f \left(\bar{\psi}^f_\alpha(x) \left[i \left(\gamma_\mu \mathcal{D}_\mu(x) \right)_{\alpha\beta} - m_f \, \delta_{\alpha\beta} \right] \psi^f_\beta(x) - \frac{1}{4} G_{\mu\nu a}(x) \, G_{\mu\nu a}(x) \right). \tag{2.89}$$

With the action we can derive the Feynman rules for calculating QCD processes, such as cross-sections of quark scatterings using multiple vertices and propagators. The asymptotic freedom [63] of the energy dependent coupling $\alpha_s(Q^2) \propto g^2$ allows calculations in the $Q^2 \to \infty$ limit, where $\alpha_s(Q^2) \to 0$. The three fundamental vertices are illustrated in Figure 2.3.

We can compute color factors [65] from these simple one-vertex diagrams. These factors tell us the relative strength of either a quark emitting a gluon ($q \to q+g$), a gluon transforming to a quark-antiquark pair ($g \to q\bar{q}$), or to two gluons ($g \to gg$).

Other applications of QCD can be found when considering finite temperature and density [141]. An example is the QCD phase diagram.

2.4.2 QCD Phase Diagram

One of the aims of studying QCD is to gain more knowledge about the phase diagram of quark matter. This is important for a number of applications, e.g., neutron stars or heavy ion collisions [80]. Unfortunately, the various phases and phase transitions are mostly conjectured and not well known, neither experimentally nor theoretically. The phase diagram takes two parameters, the temperature T and the baryon chemical potential μ to categorize the different states of quark matter.

The chemical potential can be used to portray the imbalance between quarks and anti-quarks in a system. For instance we know that the early universe was very hot and showed perfect quark anti-quark symmetry, i.e., $T \to \infty$ and $\mu \to 0$. Atomic matter as we know it is really a mixed phase, where nuclear matter is surrounded by vacuum. This point is represented by $T \to 0$ and $\mu \approx 310\,\text{MeV}$ in

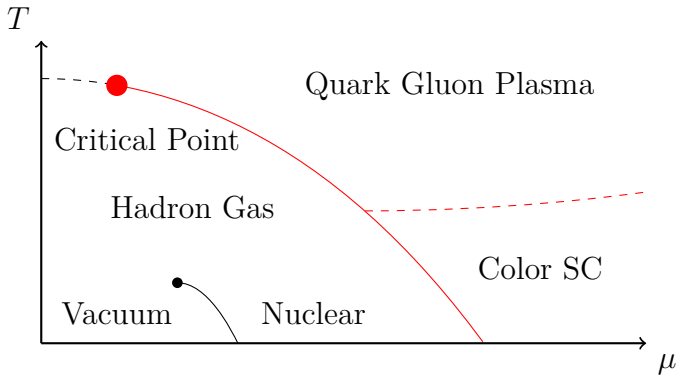


Figure 2.4: Sketch of the QCD phase diagram including the most important phases, the critical point, and the crossover region. The first-order phase transition is indicated by the solid red line.

the phase diagram [8]. Similarly, we can find representative pairs of T and μ for, e.g., neutron stars.

A sketch of the hypothetical QCD phase diagram is shown in Figure 2.4. The end point of the first-order phase transition is a critical point. The dashed line is an analytic crossover region starting at a temperature of 170 MeV at $\mu = 0$. The diagram contains most features, which can be seen in other popular illustrations [56, 126, 130].

The QCD phase diagram is subject to active research. A detailed list of all current activities and upcoming challenges can be found in [23]. We proceed to introduce Lattice QCD, which allows for first principles calculations of equilibrium quantities without chemical potential, i.e., $\mu = 0$. Under special circumstances Lattice QCD may also deliver interesting results up to moderate values of the chemical potential, i.e., $\mu \leq 3T$.

2.5 Lattice QCD

The whole idea of Lattice QCD (LQCD) is to interpret the path integral as a partition function in the sense of statistical mechanics. It was first proposed by Wilson in 1974 [150]. The technique for performing a LQCD simulation is based on statistics and special properties of the underlying procedure.

Non-perturbative phenomena in QCD require numerical simulations. We hereby follow first principles, i.e., instead of assuming some model and doing an analytical calculation according to the assumed model, we only use the action with some experimental input, such as the quark masses and the coupling strength [55]. The experimental input is not required for the simulation to work, but rather to produce realistic output.

In this section we briefly introduce the idea behind computing QCD observables numerically by using LQCD. We outline the necessary steps to transform the continuous theory to a consistent discrete representation. A much more extensive introduction can be found in text books, e.g., in [36, 58].

2.5.1 Discretization

The most straightforward way to discretize the QCD action from Equation (2.88) is to substitute the derivatives in the Lagrangian (Equation (2.89)) with finite-difference approximations. We denote the finite-difference by the lattice spacing a. In this scheme the integration is replaced by a sum over the lattice sites. However, such a naive approach is unfortunately not gauge-invariant for $a \neq 0$. Hence we need to pursue a different approach [64] for discretization.

The discretization of Euclidean space-time in LQCD results in hypercubic lattice. In this lattice the quark fields $\psi(x)$ are placed on the sites. $SU(3)$ valued gauge fields $U_\mu(x)$ represent the links between these sites. The induced spacing a yields a proper regularization, which makes the Quantum Field Theory (QFT) finite. The continuum theory can be recovered by taking the limit $a \to 0$. All the gauge transformations of the continuum theory apply to the lattice formulation as well.

It can be shown that every product of links along a closed loop is a gauge-invariant quantity. We call the product of the smallest possible non-trivial closed loop a plaquette. The plaquette is defined by

$$U_{\mu\nu}(x) \equiv U_\mu(x)\, U_\nu(x+\hat{\mu})\, U_{-\mu}(x+\hat{\mu}+\hat{\nu})\, U_{-\nu}(x+\hat{\nu}). \tag{2.90}$$

The simplest possible gauge action (Wilson gauge action [150]) is given by the product of gauge links around the elementary plaquettes,

$$S_G[U] = \frac{\beta}{3} \sum_x \sum_{\mu<\nu} \Re \operatorname{tr} \left(1 - U_{\mu\nu}(x)\right), \tag{2.91}$$

with the inverse coupling $\beta = 6/g^2$. The fermionic action is discretized to be

$$S_F[\bar{\psi}, \psi, U] = a^4 \sum_{x,x'} \bar{\psi}(x)\, D_m(x,x')\, \psi(x'), \tag{2.92}$$

with the naive choice for the lattice Dirac operator:

$$D_m(x,x') = \left(\sum_\mu \gamma_\mu \frac{U_\mu(x)\, \delta_{x+\hat{\mu},x'} - U_\mu^\dagger(x')\, \delta_{x-\hat{\mu},x'}.}{2a} \right) + m\, \delta_{xx'} \tag{2.93}$$

Unfortunately, the naive choice for the operator results in the so-called doubling

problem. By introducing an artificial term we can essentially obtain the continuum result, however, we loose one of the original properties: chiral symmetry.

Unlike most regularization techniques from continuum QCD, the lattice regularization does not rely on the perturbative expansion. Therefore, we can study non-perturbative effects by numerical evaluation of the path integral that defines the theory. Additionally, the LQCD formulation preserves gauge invariance, which makes gauge fixing as in perturbative calculations obsolete.

2.5.2 Simulation

Running LQCD simulations is a compute-intensive task that demands very well-tuned algorithms and specialized machines, such as the one presented in Part II. It should be no surprise that improved algorithms for running Monte Carlo simulations of LQCD have been and are still actively researched [128]. An important milestone has been the introduction of Hybrid Monte Carlo [44]. The idea is to minimize the rejection of configurations while making successive configurations rather independent by performing an integration of the classical time evolution of some Hamiltonian.

Extrapolating to the continuum limit requires simulations with different lattice spacings and fixed physical volume with different lattice sizes. With growing lattices our computational efforts also increase. In the end we have a trade-off between statistics and accuracy that needs to be overcome. Hence, LQCD results come with both statistic and systematic errors. The former arises from the use of Monte Carlo integration, while the latter originates, e.g., from the use of non-zero values of a.

For a single quark flavor with an operator $O[\bar{\psi}, \psi, U]$ the expectation value reads

$$\langle O \rangle = \frac{1}{Z} \int [d\psi]\, [d\bar{\psi}]\, [dU]\, O[\bar{\psi}, \psi, U]\, \exp\left(-S_F[\bar{\psi}, \psi, U] - S_G[U]\right), \tag{2.94}$$

where the integration measure is defined as

$$[d\psi] = \prod_{\alpha,a,x} d\psi_{\alpha a}(x), \tag{2.95}$$

$$[d\bar{\psi}] = \prod_{\alpha,a,x} d\bar{\psi}_{\alpha a}(x), \tag{2.96}$$

$$[dU] = \prod_{\alpha,a,x} dU_{\alpha a}(x). \tag{2.97}$$

We call $[d\psi]$, $[\bar{\psi}]$ the Grassmann measure over the quark and antiquark fields and $[dU]$ the Haar measure over the gauge-field links.

The partition function is given by

$$Z = \int [d\psi]\,[d\bar{\psi}]\,[dU]\,\exp\left(-S_F[\bar{\psi},\psi,U] - S_G[U]\right), \tag{2.98}$$

which considers both, the bosonic and the fermionic part of the action.

The fermion fields in the path integral cannot be represented as ordinary complex numbers. In order to follow the Fermi-Dirac distribution we need to introduce Grassmann numbers, which are anticommuting complex numbers. The issue with Grassmann numbers is their implementation for numerical analysis. Instead, we manipulate the integral analytically using basic Grassmann calculus. We obtain

$$\int [d\bar{\psi}]\,[d\psi]\,\exp\left(-S_F[\bar{\psi},\psi,U]\right) = \int [d\bar{\psi}]\,[d\psi]\,\exp\left(-\bar{\psi}\,D_m\,\psi\right) = \det D_m, \tag{2.99}$$

the so-called fermion determinant.

The partition function can therefore be written as

$$Z = \int [d\bar{\psi}]\,[d\psi]\,[dU]\,\exp\left(-S[\bar{\psi},\psi,U]\right) = \int [dU]\,\exp\left(-S_G[U]\right)\det D_m. \tag{2.100}$$

A more elegant version uses the matrix identity $\log(\det M) = \mathrm{tr}\,(\log M)$ to simplify Equation (2.100) to

$$Z = \int [dU]\,\exp\left(-S_{\text{eff}}\right), \qquad S_{\text{eff}} = S_G[U] - \mathrm{tr}\,(\log D_m)\,. \tag{2.101}$$

We can now run simulations using, e.g., the basics that have been outlined in Section 2.1.5. For the acceptance weight we rely on the Euclidean action.

The Euclidean formulation gives us a few benefits. Since the path integral contains exponential weights, which are real and positive, it is well-suited for numerical simulations. As desired, the classic configuration gets the largest weight. Furthermore, we have a strong connection to statistical physics by $\beta H \leftrightarrow S$. We can use our existing knowledge very easily.

However, the Euclidean formulation comes with severe drawbacks. One of the disadvantages is the inability of computing nuclear reaction cross sections [108]. Hence, LQCD cannot be used to predict likelihoods of scattering events. We have already mentioned that some symmetries are irrevocably lost, such as the chiral symmetry. Naturally, the Poincaré invariance is broken by the hypercubic group.

2.5.3 Finite Temperature and Density

In Section 2.4.2 we already mentioned the QCD phase diagram. The basics of running LQCD simulations at finite temperature and density are sufficiently well understood [90]. One of the remaining issues that prevents us from using LQCD

as the tool for obtaining the QCD phase diagram at high density is the sign problem [136]. Understanding the sign problem and its severity [18] is ongoing research.

A non-zero chemical potential μ breaks the γ_5-hermiticity of the Dirac operator. As a result the determinant of the latter may become complex, which cannot be interpreted as a probability distribution. LQCD therefore does not allow investigation of the interesting color-superconducting phase structure expected at high density and low temperature [132].

Some approximations, such as quenched QCD can even be applied with finite density [81], but have their own limitations. In contrast introducing a non-zero temperature T is possible by compactification of the time dimension. This works in complete analogy to the continuum. The temperature is then obtained via

$$T = \frac{1}{a\, N_t}, \tag{2.102}$$

where N_t is the number of sites in the temporal direction. For simulations at $T \to 0$ we can use lattices with a large N_t. For $T \to \infty$ we require a smaller time dimension.

Besides the approach of Feynman diagram sampling that we is examined in this thesis, there are several other methods that may potentially cure the sign problem [4, 39].

3

Diagrammatic Monte Carlo

This chapter is dedicated to explain the method of Diagrammatic Monte Carlo (DiagMC) [120]. We start with the elementary theory behind the method, which is then specialized quickly to compute series expressed in terms of Feynman diagrams. In Section 3.1 we find the mathematical background and the vocabulary that is used throughout every model we investigate.

One of the most intuitive examples for illustrating the idea behind DiagMC is estimating the value of a single integral. We examine an example of such an application in Section 3.2. Besides introducing the basic concepts of the method we are mainly focused on obtaining information about its ability to handle a mild sign problem.

Finally, we introduce a real-world application that utilizes DiagMC to compute physical quantities. The polaron model described in Section 3.3 is still subject of active research. We will see that DiagMC can be a suitable tool for its analysis. In our study we use a simplified model that already contains many interesting aspects.

3.1 Diagrammatic Monte Carlo

DiagMC is useful for calculating quantities represented in a diagrammatic expansion [147]. The diagrammatic expansion is given in terms of (similar or related) integrals with a variable number of integration variables. The method uses

Metropolis-Hastings [68] type Markov chain sampling as described in Section 2.1.5. It can be applied to arbitrary Feynman diagrams in the thermodynamic limit, i.e., where we have $N \to \infty$ and $V \to \infty$ with $N/V = \text{const}$.

We start with a generalized series, which is dependent on a set of external parameters t and a varying number of internal parameters x, expressed as

$$G(t) = \sum_{n=0}^{L} \sum_{\xi_n} F_{\xi_n}(t), \tag{3.1}$$

where we call $L \leq \infty$ the highest-order. The sum over all ξ_n covers the different diagrams for the nth-order. By using ξ_n for a fixed n we label the different diagram *topologies* at order n. Different topologies share the same set of parameters, but use a different functional representation.

The elements $F_{\xi_n}(t)$ are defined as integrals of the *diagram* $D_{\xi_n}(x_1, \ldots, x_n, t)$ over the internal parameters $\{x_1, \ldots, x_n\}$. We have

$$F_{\xi_n} \equiv \int dx_1 \ldots dx_n \, D_{\xi_n}(x_1, \ldots, x_n, t). \tag{3.2}$$

The *set of diagrams* $\mathcal{F}$ contains all diagrams, i.e.,

$$\mathcal{F} = \{D_{\xi_n} : n \in [0, L], \ \forall \xi_n\}. \tag{3.3}$$

It contains all integrands of the series, independent of the order or topology. If we refer to a diagram in a functional context, we always mean the integrand of functional representation of the particular diagram.

Structures similar to Equation (3.1) appear in many perturbative expansions, e.g., in QFTs, where we have a bijection between the integrals $F_{\xi_n}(t)$ and their representation in form of Feynman diagrams. In addition to DiagMC specific terminology we use the same terminology as known from Feynman diagrams. If, e.g., we talk about an arc in a diagram, we refer to a particular propagator connected to two vertices. Additionally, in Feynman diagrams a vertex represents a point of interaction with propagators standing for Green's functions of particles.

In general the series $G(t)$ does not require a finite value for L, i.e., the highest order could be $L = \infty$. Every order comes with its own set of integration variables. A prerequisite is that the integration variables of the previous order always form a subset of the integration variables of the current order.

In Figure 3.1 we see a sketch of a diagrammatic Monte Carlo simulation. We sample the phase space consisting of the set of diagrams, i.e., diagrams of all orders and all topologies, together with all possible parameter values. The sampling is governed by the update procedures that we impose. Examples and specific rules

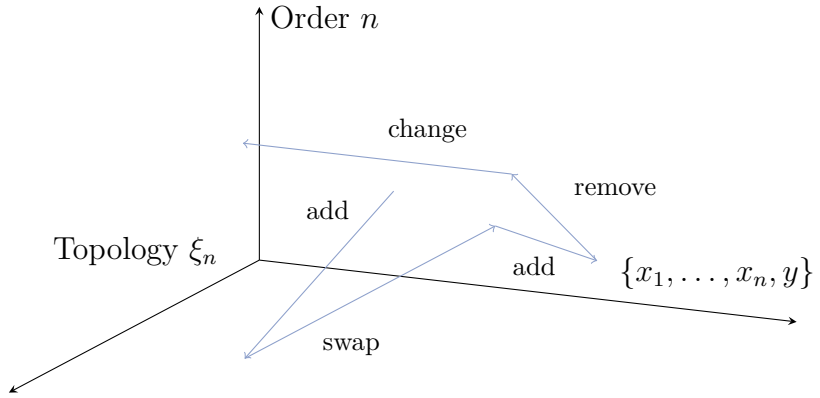


Figure 3.1: Sketch of a few consecutive Monte Carlo timesteps in the phase space of DiagMC. Only accepted updates are shown.

for these procedures are introduced later, e.g., see Equation (3.9).

Basically, we are free to choose any combination, as long as they maintain ergodicity, thus sampling the whole phase space. Finally, we demand that the set of update procedures is performed with respect to the distribution from the series in Equation (3.1) itself.

We now start to use the assumption that there is only a single diagram per diagram order, i.e., $|\xi_n| = 1$. We use

$$D_n(x_1, \ldots, x_n, t) \equiv D_{\xi_n}(x_1, \ldots, x_n, t). \tag{3.4}$$

We can use this assumption to illustrate the basic idea. We will see that the simplification can be easily resolved by introducing one additional update procedure.

The function $G(t)$ is interpreted as a distribution of the external parameters t. A possible solution to obtain a value for $G(t)$ of Equation (3.1) is to use a Markov chain process for sampling the diagrams stochastically. In order to deal with a possibly sign-alternating series, we can rewrite the diagrams $D_n(x_1, \ldots, x_n, t)$ as

$$D_n(x_1, \ldots, x_n, t) = s_n(x_1, \ldots, x_n, t)\ \left|D_n(x_1, \ldots, x_n, t)\right|, \tag{3.5}$$

where $s_n(x_1, \ldots, x_n, t) = \pm 1$ for a particular configuration $\{n, x_1, \ldots, x_n, t\}$.

At this stage we can write

$$G(t) = F_0(t)\left(1 + \frac{F_1(t)}{F_0(t)}\left(1 + \frac{F_2(t)}{F_1(t)}\left(1 + \ldots\right)\right)\right). \tag{3.6}$$

For convergence we require $F_n(t) \to 0$ for $n \to \infty$. In general this is not satisfied. Therefore, we have to deal with series that may be asymptotic or even divergent. For such series DiagMC relies on resummation techniques, such as the Borel summation [24].

There are several arguments [147] why DiagMC is still a suitable method to sample series suffering from the sign-problem. While the complexity C of an extensive configuration space scales exponentially with its cluster volume, the intensive configuration space used by DiagMC is much smaller in practice. We have

$$C \propto (L!)^m\,, \qquad m \sim 1,\ L \leq 10. \tag{3.7}$$

Since DiagMC works in the thermodynamic limit from the beginning, we do not have to worry about the exponential scaling of the computational complexity with the cluster volume. Instead, we only need to sample (a subset of) the diagrams in $\mathcal{F}$. Ideally, the significance of higher-order diagrams should be exponentially suppressed. Otherwise, the formerly mentioned resummation techniques may be helpful.

For sampling the sequence we use Markov chain Monte Carlo as introduced in Section 2.1.5. The acceptance probability to perform an update from state A to state B using the ratio of the transition probabilities $R_{A\to B}$ is using the Metropolis-Hastings criterion from Equation (2.29). It is given by

$$P_{\text{acc}} = \min\left(1, R_{A\to B}\right). \tag{3.8}$$

The ratio of the transition probabilities is the ratio of the diagrams in the provided states and the inverse ratio of the distributions of the generated values that are used in those diagrams.

To make the transition probability positive we separate the function $D_n(x_1, \ldots, x_n, t)$ representing the diagram into two parts as noted in Equation (3.5). We only use the absolute value in the transition probability, leaving the sign to be sampled. Sampling only the sign has some advantages and disadvantages. Most importantly, it avoids possible round-off errors coming from individual samples.

We distinguish between two kinds of updates: Updates that change the shape of the diagram and updates that change the values of the existing parameters. The latter occurs if we, e.g., change the value of the i-th variable in nth- order diagram. Using the distribution $\Omega_i(x)$ for the values x_i we have

$$R_{x_i\to x_i'} = \left|\frac{D_n(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_n, t)}{D_n(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n, t)}\right| \frac{\Omega_i(x_i)}{\Omega_i(x_i')}. \tag{3.9}$$

This class of updates is straightforward. It is basically identical to common methods of simulating a given continuous distribution.

Updates that change the shape of a diagram occur far more often. If we, e.g., go from the nth-order to the mth-order diagram, with $m > n$, we can take the ratios as before. This time, however, we do not need to take the ratios of the

distributions. It is sufficient to divide by the distributions of the additional m parameters.

The transition probability is given by

$$R_{n\to m} = \left| \frac{D_m(x_1, \ldots, x_m, t)}{D_n(x_1, \ldots, x_n, t)} \right| \frac{1}{\Omega_{n+1}(x_{n+1}) \ldots \Omega_m(x_m)}, \tag{3.10}$$

where we omit potential factors from our update procedure.

It is important that updates, which change the shape of a diagram, are always formulated with two complementary updates. Thus when we introduce an add update we need to provide a complementary remove update. The ratio of the transition probabilities for removing $m - n$ diagrams, thus going from the mth-order to the nth-order, is just the inverse of Equation (3.10). We find

$$R_{m\to n} = R^{-1}_{n\to m}. \tag{3.11}$$

There can be an arbitrary amount of potential updates. Besides satisfying all requirements of a Markov chain (see Section 2.1.3) we have to form an ergodic simulation, where all possible diagrams can be generated according to their weight in Equation (3.1).

The update procedures defines our set of updates. These are the possible actions that can be taken during a single Monte Carlo timestep in a DiagMC run. The bare minimum consists of two updates: add and remove. Here we try to add another arc to the current diagram or remove an existing arc from the current diagram. Updates that alter the current diagram without changing its order or set of parameters can be necessary, especially if we have, e.g., $|\xi_n| \neq 1$ for the nth-order.

In the case of multiple topologies, we can introduce an update to change the current topology. Depending on the problem this could be achieved by exchanging propagator targets. Such an update is commonly called a swap operation.

The ratio of the transition probabilities for a swap update can be as simple as

$$R_{\xi_n\to\xi'_n} = \left| \frac{f_{\xi'_n}(x_1, \ldots, x_n, t)}{f_{\xi_n}(x_1, \ldots, x_n, t)} \right|, \tag{3.12}$$

which does not require any change of variables and can be calculated without knowing the values of the involved distributions. Potentially, we need to include some factor that yields the ratio of probabilities of choosing the particular diagrams.

Changing the external parameters is an update procedure that could be restricted to the zeroth-order only. We ensure to keep detailed balance by either

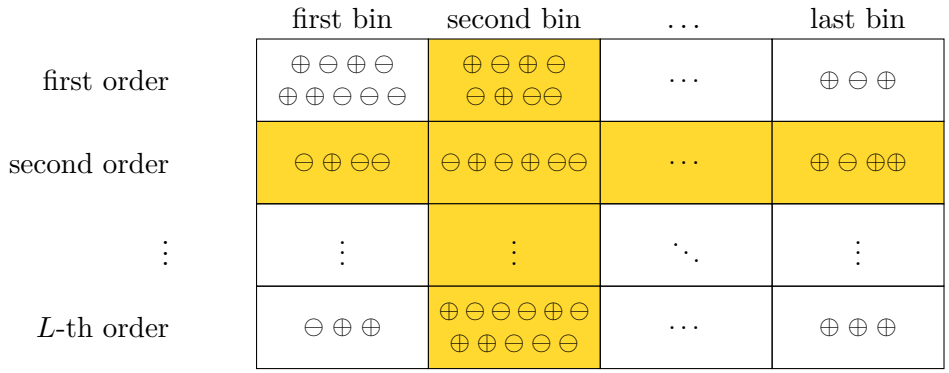


Figure 3.2: Classification of the obtained samples according to diagram order and bin index of the external parameter. The colored row represents Z_2, the colored column H_2. The symbols represent different measurements.

introducing order-dependent coefficients to our transition probabilities or by rejecting proposed changes of the external parameters for diagrams with internal variables.

Finally, we can use the measurements of $s_n(x_1, \ldots, x_n, t)$ to perform a stochastic summation [31] of the series $G(t)$. It is useful to define a set of quantities to keep track of these measurements. We use

$$H_i = \sum_n s_n(x_1, \ldots, x_n, t_i) \tag{3.13}$$

as the aggregate of the measurements in the i-th bin corresponding to the discretized t_i for all sampled diagrams. Obviously, this measurement is sensitive to the external parameters, but insensitive to the diagram order.

Additionally, we can introduce another quantity to be sensitive to the diagram order independent of the external parameters. We define

$$Z_n = \sum_i s_n(x_1, \ldots, x_n, t_i), \tag{3.14}$$

where we sum results independent of their corresponding bins. A derived quantity is the full measurement of all sampled diagrams, i.e.,

$$Z_{\mathrm{MC}} \equiv \sum_n Z_n = \sum_i H_i. \tag{3.15}$$

The previous two quantities are illustrated in Figure 3.2, where a row represents Z_n and a column can be identified with H_i. The cells represent measurements in the given order and bin index. Each sign is a single measurement.

With these quantities at hand we can perform the stochastic summation, which allows us to estimate the value of $G(t)$. In the statistical limit $N \to \infty$ the ratio

of the measured values approaches the integral value, i.e.,

$$\frac{Z_0}{Z_{\mathrm{MC}}} \to \frac{I_0}{I}, \tag{3.16}$$

where we use $I_0 \equiv \int dt\, F_0(t)$ and $I \equiv \int dt\, G(t)$.

According to Equation (3.15) Z_{MC} is the sum over all bins. Similarly, I can be rewritten as

$$I = \sum_i \int_{\mathrm{bin}_i} dt\, G(t). \tag{3.17}$$

We estimate the value of the $G(t)$ in the i-th bin via

$$\int_{\mathrm{bin}_i} dt\, G(t) \leftarrow H_i \frac{I_0}{Z_0}. \tag{3.18}$$

We note that the result and error depend crucially on the fluctuations of Z_0. Hence if $F_0(t)$ (i.e., $D_0(t)$) is small, Z_0 is consequently small as well and we have to deal with a large error.

The interesting part is that the scheme allows us to estimate the value of the n-th element of the series. A combination of Equation (3.16) and Equation (3.18) yields

$$I_n \equiv \int dt\, F_n(t) \leftarrow Z_n \frac{I_0}{Z_0}. \tag{3.19}$$

The correctness of the estimate depends highly on the zeroth-order diagram and the sampling mechanism. Ideally, the diagrams are sampled identical to their real weight in the series.

DiagMC can be used in a variety of scenarios, e.g., see [82, 26]. In the following we look at two examples that illustrate and study different aspects of the method.

3.2 Integration using DiagMC

A very popular usage of Monte Carlo methods is the evaluation of difficult integrals. Evaluating a multi-dimensional integral with Monte Carlo methods is very efficient, since the error does not scale with the number of dimensions. In this example we use DiagMC with a reduced set of diagrams $\mathcal{F}$. We have a single diagram, denoted by $D_1(x)$, where x is chosen to be a one-dimensional parameter. In principle the problem could be easily extended to an arbitrary number of dimensions.

3.2.1 Model and Simulation

Evaluating a single diagram directly is not possible. We have seen that we always need to know at least two diagrams with one that is already evaluated as comparison. Thus, we introduce an artificial diagram D_0, which is just a constant. Our simulation can now run to evaluate

$$I = D_0 + \int dx\, D_1(x). \tag{3.20}$$

By only sampling contributions from $D_1(x)$ we are able to ignore D_0 in the end. Our goal is to find the value of $\int dx D_1(x)$.

We use the following two updates:

add We perform the transition from D_0 to $D_1(x)$ by generating a new value for the parameter x according to some arbitrary distribution $\Omega(x)$.

remove We perform the transition from $D_1(x)$ to D_0. This is the inverse to the previous update procedure.

The add procedure can only be invoked if we are in the state $n = 0$ (ground state), while the remove procedure is only accessible for configurations with $n = 1$. In this simple setup detailed balance is preserved trivially.

This example is already quite useful for understanding the applicability of the algorithm regarding problems with sign fluctuations, potentially resulting in a sign problem. We choose the toy model

$$D_1(x) \equiv f_\alpha(x) = \cos(x)\, \exp(-\alpha\, x)\, \Theta(x), \tag{3.21}$$

where α can be tuned in such a way that finding the correct value of the integral becomes harder due to increased oscillations. $\Theta(x)$ denotes the Heaviside step function, which limits the range of values to $[0, \infty]$. The analytic solution of the

integral $f_\alpha(x)$ is

$$I_\alpha = \int_0^\infty dx\, f_\alpha(x) = \frac{\alpha}{1+\alpha^2}. \tag{3.22}$$

The "hardness" of the sign problem can be computed by considering the ratio of the integrals with and without the absolute value of the integrand. We obtain

$$H(\alpha) = \frac{\int_0^\infty dx\, f_\alpha(x)}{\int_0^\infty dx\, |f_\alpha(x)|} = \frac{\alpha}{\alpha + \operatorname{csch}(\alpha)}. \tag{3.23}$$

In the limit of $\alpha \to 0$ we are able to check that $H(\alpha)$ behaves like

$$\lim_{\alpha\to 0} H(\alpha) \approx \frac{\alpha}{\alpha + \alpha^{-1}} \approx \alpha^2. \tag{3.24}$$

This characterizes the hardness of this sign problem in the region for small α. Apparently, the integral becomes quadratically harder with α approaching zero. This is not close to the sign problem that appears in QCD with finite chemical potential, which scales exponentially, however, it already provides everything necessary for a first study.

3.2.2 Update Procedures

In general we want to use the technique of importance sampling. In case of our toy model the variable x could be generated efficiently by an exponential distribution. This has the advantage that we are already quite close the real integral, thus improving the acceptance rate.

The normalized distribution reads

$$\Omega(x) = \alpha \exp\left(-\alpha\, x\right), \tag{3.25}$$

which cancels the exponential term in the transition probability. Overall the ratios of the transition probabilities between the two states D_0 and $D_1(x)$ are

$$R_{0\to 1} = \left|\frac{f_\alpha(x)}{D_0}\right| \frac{1}{\Omega(x)} = \frac{|\cos(x)|}{\alpha D_0}, \tag{3.26}$$

$$R_{1\to 0} = \left|\frac{D_0}{f_\alpha(x)}\right| \Omega(x) = \frac{\alpha D_0}{|\cos(x)|}, \tag{3.27}$$

where α and D_0 must be elements of $\mathbb{R}_+$.

In our sampling process we will not perform any measurements of the artificial diagram. The measured quantity is the sign of the current value of $D_1(x)$. This means that we are effectively sampling $\operatorname{sgn}(\cos(x))$ by using importance sampling with a distribution that is proportional to $\exp(-\alpha x)$. For the Metropolis-Hastings algorithm we use the given ratios as weights for the acceptance criterion.

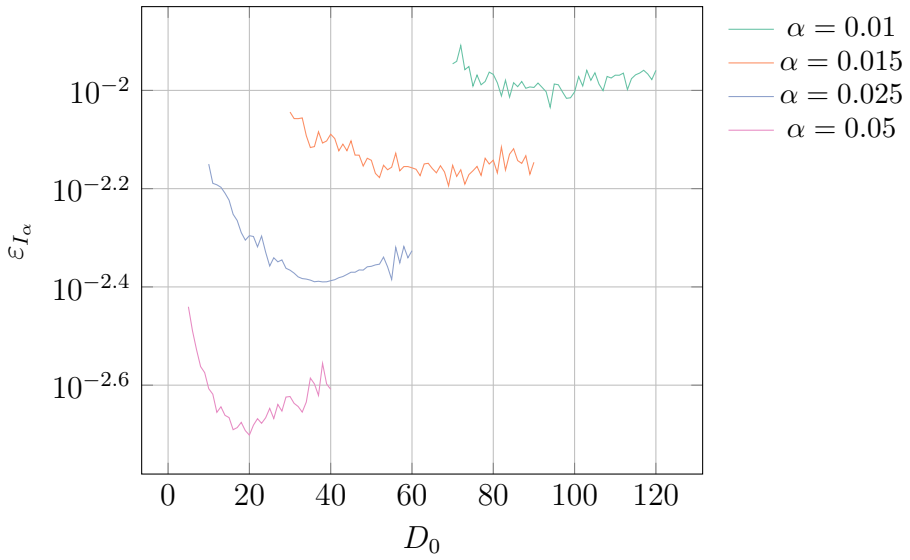


Figure 3.3: The optimal choice of D_0 is close to α^{-1}. All simulations are performed with $N = 10^8$ configurations.

There are some questions remaining. The most important question is: How to get a suitable value for D_0? We are free to choose any value in $\mathbb{R}_+$, but we still have to find out what values are optimal for the simulation, if any.

Naively, we could pick an arbitrary non-zero value for D_0, e.g., 1 would be suitable candidate. This would certainly work, however, it is not very efficient in general, since the acceptance weight depends on α. A solution that works quite well is to choose D_0 such that the weights do not depend on α any more.

By choosing $D_0 = \alpha^{-1}$ we obtain

$$R_{0\to 1} = |\cos(x)|, \tag{3.28}$$

$$R_{1\to 0} = |\cos(x)|^{-1}. \tag{3.29}$$

In Figure 3.3 some simulations with various values for α are shown. Looking only at the computed error for the integral value, we are able to see which choice for D_0 yields the most efficient simulation. In the end, the D_0 with the lowest error was always quite close (or identical) to α^{-1}. Simulations with lower values of α suffered from an increased error, since the number of configurations was fixed at $N = 10^8$. The choice of $D_0 = \alpha^{-1}$ is thus confirmed by numerical evidence.

We this problem we only use a single bin H containing a sequence of samples H_i. Each sample represents the sign of the cosine with the currently generated value for x. The total value of all samples is

$$H = \sum_i H_i, \qquad H_i = \operatorname{sgn}(\cos(x_i)) = \pm 1, \tag{3.30}$$

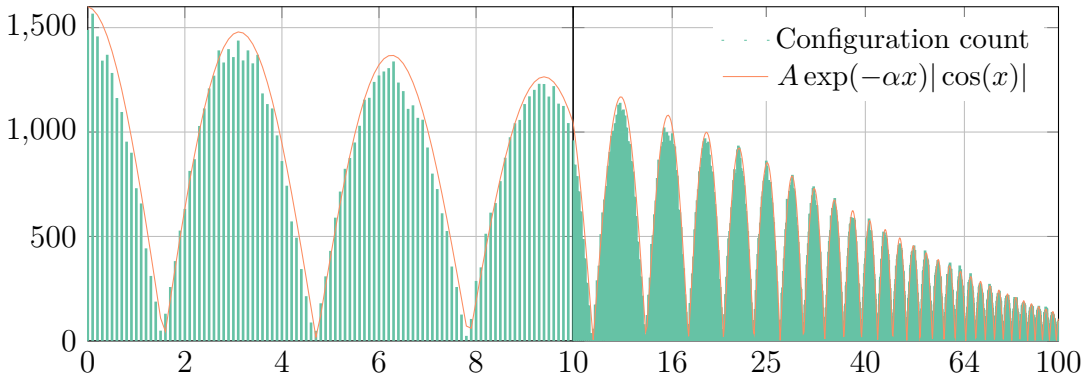


Figure 3.4: Distribution of x values for $N = 10^6$ configurations with $\alpha = 0.025$. The x values have been placed in bins with $\Delta x = 0.1$. The fit to the transition probability uses $A = 1600$.

where x_i is the i-th sampled configuration. It is worth remembering that the artificial diagram D_0 is not sampled and thus the picture looks slightly different than Figure 3.2.

Using H, as well as the constant D_0 and the weight of the artificial diagram, Z_0, we can estimate the value of the integral. In the statistical limit we have

$$I = D_0 \frac{H}{Z_0}. \tag{3.31}$$

For the edge cases we can verify that this equation represents the right behavior:

- In case of D_0 being significantly larger than the integral we expect the simulation to rarely reach the diagram represented by $D_1(x)$. Therefore, H will be close to zero, especially if divided by Z_0, which is much larger than H.
- In case of D_0 being much smaller than the integral we expect the opposite. Here Z_0 is close to 1 as we always start in the artificial diagram. Now the only question is what kind of value H is. H should be close to the value of the integral, relative to D_0.

One thing we can look at is the distribution of x values among the sampled configurations. In Figure 3.4 we see that the transition probability is reflected completely by the chosen values of x. In the given plot we use a binning width of $\Delta x = 0.1$. This is remarkable, as we only use the exponential distribution to generate x. However, since the transition implied using the absolute value of $\cos(x)$, we accept these values with a probability proportional to $|\cos(x)|$.

It is important to realize that the decision for a specific distribution for generating the parameters has no influence on the distribution of the accepted values of the parameters. The distribution will always result in a histogram reflecting the used transition probabilities.

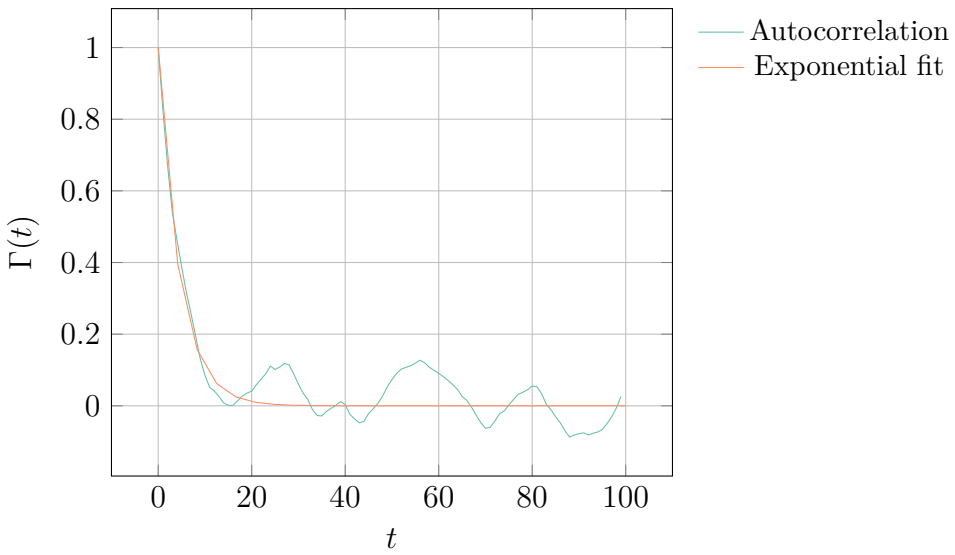


Figure 3.5: The first 100 values for the autocorrelation of a simulation with $\alpha = 0.025$ and $D_0 = 40$. The exponential fit is showing $\exp(-t/\tau)$ with $\tau = 4.5$.

3.2.3 Bias

Before we can consider using any estimation provided by this method, we should investigate if autocorrelation is a problem or not. A good way to study this is to use a sufficient number of configurations (e.g., $t_{\max} = 10^3$) and calculate the integrated autocorrelation time τ_{int} using Equation (2.58). We can improve the accuracy, remove oscillations, and label the error on the calculated autocorrelation. Here we need to consider different initial times t_i in the range $0 \leq t_i \leq t_{\max} - t_i$.

The cutoff parameter W of Equation (2.58) is chosen dynamically by truncating the summation once we encounter a negative value for the autocorrelation. The autocorrelation for a simulation with our toy model is shown in Figure 3.5. We see that the autocorrelation time is quite small. From the exponential fit we can immediately infer $\tau_{\text{int}} \approx 4.5$. Indeed the obtained value is

$$\tau_{\text{int}} = 4.827. \tag{3.32}$$

It seems to be sufficient to set the number of runs to reach equilibrium to $\mathcal{O}(10^2)$.

3.2.4 Evaluation and Results

Running various simulations at different values of α gives us an informative comparison between the numerical estimate and the analytic result of the integral. In Figure 3.6 we use $D_0 = 4$ and $N = 10^8$ for all runs. We can see that the error of the integral is a function of α, D_0, and N. Even with $D_0 = \alpha^{-1}$ we still have the dependency on α and N. In general we obtain a scaling that is similar to the hardness of the sign problem. For smaller α we need much more configurations.

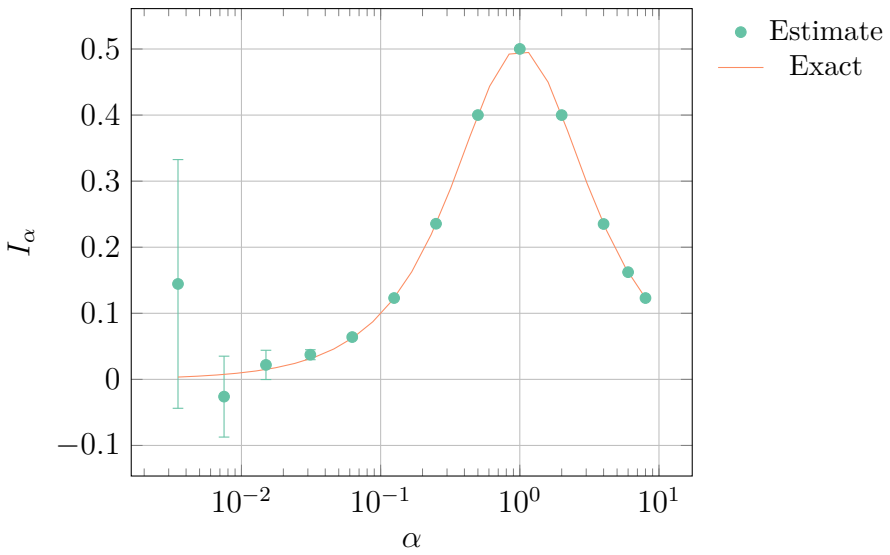


Figure 3.6: The numerically estimated value compared to the analytic result. All simulations are performed with $N = 10^8$ configurations. The estimate was obtained for $D_0 = 4$.

The previous statements can be justified with another simulation. This time our goal is to obtain roughly the same absolute error for different α. We could vary the values of D_0 and N, however, using the previously gained knowledge we can immediately set $D_0 = \alpha^{-1}$. This approximation works well enough to reduce the problem to finding the value of N, giving us roughly the same absolute error as previously.

Our procedure is outlined in Algorithm 1. We do not specify a limit for the α_i to evaluate. The choice of D_0 is implicit, as noted before. The initial values of α and N can be chosen freely.

Algorithm 1 Error Normalization

$\alpha_0 \leftarrow \alpha$
$\varepsilon_0 \leftarrow$ error of run with $N_0 \equiv N$ configurations
for $i \leftarrow 1$ **to** some limit **do**
    $\alpha_i \leftarrow \alpha_{i-1}/2$
    $N_i \leftarrow N_{i-1}$
    repeat
        $\varepsilon_i \leftarrow$ error of run with N_i configurations
        $N_i \leftarrow 2N_i$
    until $\varepsilon_i \leq \varepsilon_0$
end for

Following this procedure we end up with Figure 3.7. We can observe that the absolute error is indeed roughly constant justifying our method. The relative error still grows as $\alpha \to 0$. We are able to verify that the number of configurations has to scale quadratically with smaller values of α to keep the absolute error constant,

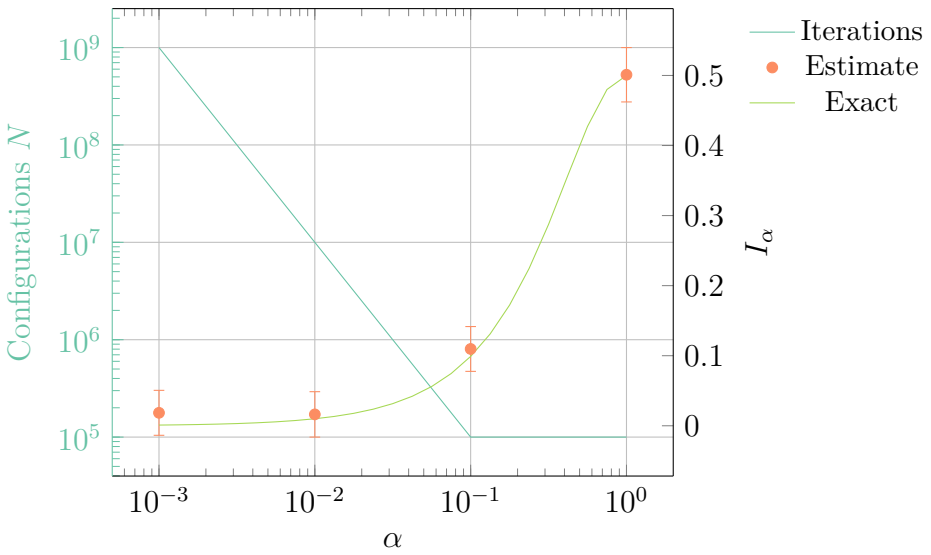


Figure 3.7: The number of configurations to obtain the same absolute error for different choices of α. All simulations use $D_0 = \alpha^{-1}$.

which is something that can only be attacked by computational power and, of course, computing time if we do not change the algorithm fundamentally.

We could make this method much more efficient for the illustrated scenario, e.g., by adding another update procedure, which tries to change the value of the involved variables. Instead of optimizing this method further, we continue to apply DiagMC to another problem.

3.3 Polaron Model

In solid state physics the polaron model is used to describe interactions between electrons and atoms. The polaron itself is a quasiparticle, which represents the union of a conduction electron (or hole) with its self-induced polarization. Usually, this polarization is happening in a polar semiconductor or an ionic crystal. The polaron itself is characterized by its self-energy, its effective mass, and its mobility. It has different properties than non-polarized band-electrons.

The polaron model itself can be used in many situations. These situations are not limited to solid state physics. It is still subject of active research [113]. The problem is quite interesting. The core question is what happens to particles coupled to an environment. This raises questions about properties of the resulting object. In this section we will restrict ourselves to a special variant of the polaron model, the so-called *Fröhlich polaron* [101].

In the following we will mainly follow the original publication of this work [120]. The update scheme and the transition probabilities are derived from our formulation of the problem. We are using natural units $\hbar = c = 1$.

3.3.1 Model and Simulation

The Hamiltonian describing the interaction in the Fröhlich model is given by

$$H = \frac{\mathbf{p}^2}{2m} + \sum_{\mathbf{k}} \omega \, a_{\mathbf{k}}^{\dagger} a_{\mathbf{k}} + \sum_{\mathbf{k}} \left(V_k \, a_{\mathbf{k}} \exp(i\mathbf{k} \cdot \mathbf{r}) + \text{h.c.} \right), \tag{3.33}$$

where $\mathbf{r}$ is the position operator of the electron with band mass m. The momentum operator is represented by $\mathbf{p}$. The operators $a_{\mathbf{k}}^{\dagger}$ and $a_{\mathbf{k}}$ are the creation and annihilation operators for longitudinal optical phonons of the wave vector $\mathbf{k}$. These phonons have the energy ω.

For our simulation we consider the following set of Feynman rules outlined in [102]. We are only interested in a single incoming and a single outgoing electron. The tree diagram is the free electron propagator.

The Green's function that represents the propagator of a free electron reads

$$G^{(0)}(\mathbf{p}, \tau) = \exp\left(- \left(\frac{\mathbf{p}^2}{2m} - \mu \right) \tau \right), \tag{3.34}$$

where μ is the chemical potential.

We also have the propagator for the phonons, which are considered dispersionless in this simplified model. The propagator is given by

$$D(\tau) = \exp\left(-\omega \, \tau \right). \tag{3.35}$$

In order to connect the two propagators we need some kind of interaction. The interaction is taking place in vertices, thus coupling electrons and phonons.

Each vertex connects an incoming electron with an outgoing electron and a phonon. The momentum needs to be conserved in every vertex, which determines the momentum of, e.g., the outgoing electron by knowing the momenta of the incoming electron and the phonon.

For each vertex we get an additional factor of

$$V(\mathbf{k}) = i \sqrt{2\sqrt{2} \, \alpha \, \pi} \frac{1}{k}, \tag{3.36}$$

which is dependent on the incoming momentum $\mathbf{k}$. Since every phonon will always connect two new vertices, we can write one expression that contains both, the phonon propagator and the two vertices. We denote this propagator by $\tilde{D}$. We have

$$\tilde{D}(\mathbf{k}, \tau) = |V(\mathbf{k})|^2 \; D(\tau). \tag{3.37}$$

For our simulation we use the following updates:

add We insert two new vertices with a phonon propagator connecting the two vertices.

remove We remove two consecutive vertices that are connected by a phonon propagator.

swap We permute the connections of two phonon propagators to obtain a new topology.

extend We change the time variable, which tries to substitute the time $t_{\max}$ of the last vertex with a new value.

The conditions are not as simple as in the previous section. In order to preserve detailed balance we need to be careful with the overall construction. For instance, since we cannot perform the add procedure in the highest-order we need to adjust the probabilities for the other updates. There are several ways to exclude such actions while still obeying detailed balance.

A strategy that is not efficient, however, less error-prone and trivial to implement is to allow every procedure in all orders. In this strategy we simply reject updates if we encounter an invalid action, such as adding another arc to the highest-order diagram. This generic procedure works with any set of updates.

3.3.2 Add Update

First we select a random index from the list of available vertices. We just use a uniform distribution. The vertex associated with the selected index is called current vertex. Then we compute the time difference between the current vertex and the next vertex. If there are no vertices then we just use $t_{\max}$, otherwise we calculate $\Delta t = t_{i+1} - t_i$, where i is the chosen vertex index.

At this point we can compute the temporal positions of the proposed pair of vertices, labeled τ_1 and τ_2, with $\tau_2 > \tau_1$. The distributions are given by

$$\Omega_{\tau_1}(\tau) = \frac{1}{\Delta t}, \tag{3.38}$$

$$\Omega_{\tau_2}(\tau, \tau_1) = \omega \exp\left(-\omega(\tau - \tau_1)\right). \tag{3.39}$$

We need to generate the momentum of the phonon propagator. The distribution for the momentum is given by the probability

$$\Omega_{\mathbf{k}}(\mathbf{k}, \tau_1, \tau_2) = \sqrt{\frac{\tau_2 - \tau_1}{(2\pi)^3\, m}} \exp\left(\frac{k^2}{2m}(\tau_2 - \tau_1)\right), \tag{3.40}$$

which cancels the contribution of the electron propagator in our weight. The way

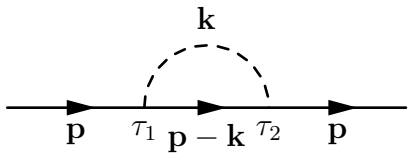


Figure 3.8: The first-order Feynman diagram. The two vertices are labeled τ_1 and τ_2. The dashed line represents the phonon propagator with momentum $\mathbf{k}$. The solid line is the electron with momentum $\mathbf{p}$.

we form our algorithm has to be carried over to the ratio that needs to be used in the acceptance criterion.

The trivial way is to ignore the whole algorithm by just computing the value of the proposed new state. Then we can divide this by the used distribution weights and the value of the current state. The main issue is, that this will not only be inefficient, but also numerically deficient. Performing many unnecessary computations that involve floating point numbers introduces numerical inaccuracies.

Additionally, one advantage of choosing the distributions $\Omega_{\tau_1}(\tau)$, $\Omega_{\tau_2}(\tau, \tau_1)$, and $\Omega_{\mathbf{k}}(\mathbf{k}, \tau_1, \tau_2)$ will be omitted once we go over to compute the full expression. It is much better to do some analytic work beforehand. This should compress the full expression to an optimized version. In the end we save computation time and avoid unnecessary numerical complications.

We start with

$$R_{i-1\to i} = \frac{\tilde{D}(\mathbf{k}, \tau_2 - \tau_1)\, G^{(0)}(\mathbf{p}, \tau_1 - t_i)\, G^{(0)}(\mathbf{p} - \mathbf{k}, \tau_2 - \tau_1)\, G^{(0)}(\mathbf{p}, t_{i+1} - \tau_2)}{G^{(0)}(\mathbf{p}, t_{i+1} - t_i)\, (n_a/n_v)\, \Omega_{\tau_1}(\tau)\, \Omega_{\tau_2}(\tau, \tau_1)\, \Omega_{\mathbf{k}}(\mathbf{k}, \tau_1, \tau_2)}, \tag{3.41}$$

where n_a/n_v is the number of non-crossing arcs (after the proposed change) against the number of vertices (after adding these). The number of non-crossing arcs is determined by counting the number of consecutive phonon connections. The incoming momentum is denoted by $\mathbf{p}$. The generated momentum is called $\mathbf{k}$.

Simplifying Equation (3.41) yields

$$R_{i-1\to i} = \left(\frac{m}{2\pi\Delta\tau}\right)^{3/2} |V|^2\, n_v\, \Delta t \exp\left(\Delta\tau \frac{\mathbf{p}\cdot\mathbf{k}}{m}\right) \left(n_a\, \omega\, k^2\right)^{-1}. \tag{3.42}$$

The add procedure starts with the current diagram, e.g., the first-order diagram shown in Figure 3.8. In this case we have $n_v = 3$ options for choosing the first vertex. n_v is equal to the number of vertices after adding a single vertex. According to this choice the value of Δt is calculated.

Finally, the position of the first vertex τ_3 is generated by considering Equation (3.38). The second vertex τ_4 is created with the exponential distribution

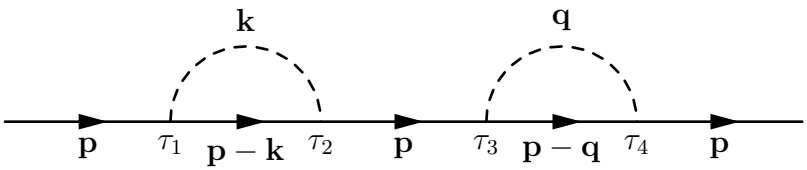


Figure 3.9: A second-order Feynman diagram. Here we have four vertices $\tau_1, \ldots, \tau_4$. The second phonon propagator carries the momentum $\mathbf{q}$.

from Equation (3.39).

3.3.3 Remove Update

The weight requires the number of non-crossing arcs after adding the new pair of vertices to preserve detailed balance. The algorithm for the removal of an arc is different from the add procedure. We only need to know the number of non-crossing arcs, which may be zero. In that case we have to reject the proposed update immediately. Otherwise, if we accept the change, we remove the arc, i.e., two vertices and a propagator. In Figure 3.9 we see a second-order Feynman diagram with two non-crossing arcs.

Detailed balance is preserved between the add and the remove updates, as the remove update transition probability ratio is the inverse of the add update ratio. We only need to re-compute the Δt and $\Delta\tau$, as they have not been fixed and might be different due to intermediate changes in the diagram, e.g., by a set of corresponding add and remove procedures or a number of swap operations.

Overall the ratio for removing an arc is given by

$$\begin{aligned} R_{i\to i-1} &= R_{i-1\to i}^{-1} \\ &= \left(\frac{m}{2\pi\,\Delta\tau}\right)^{-3/2} \omega\, k^2\, n_a \exp\left(-\Delta\tau \frac{\mathbf{p}\cdot\mathbf{k}}{m}\right) \left((n_v - 1)\,|V|^2\,\Delta t\right)^{-1}, \end{aligned} \tag{3.43}$$

where $\mathbf{p}$ is the incoming momentum of the first vertex. The momentum $\mathbf{k}$ represents the phonon momentum. The number n_v represents the number of vertices. We need to exclude one vertex to satisfy detailed balance.

3.3.4 Swap Update

One scenario for rejecting a proposed removal is shown in Figure 3.10. The illustrated second-order diagram cannot be created by the previously introduced updates. However, since it is a valid and probably significant second-order diagram, we need to consider it in our sampling process. We can include such a diagram by providing the ability to swap two vertices. The algorithm for the swap procedure is as follows.

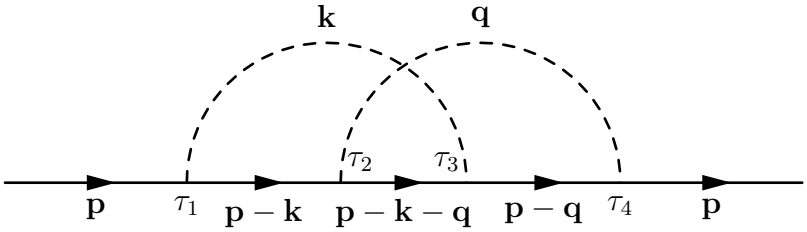


Figure 3.10: Every second-order Feynman diagram can be updated with a swap. Here the vertices τ_2 and τ_3 are swapped, resulting in another second-order diagram.

If the order of the diagram is less than two, then we immediately reject the update. In case of a zeroth-order diagram there is no vertex to select. The first-order diagram would remain unchanged, as the phonon propagator does not depend on $\mathbf{k}$, but only on k. Otherwise we choose two consecutive vertices by using a uniform distribution.

We now need to take the ratio of the new (swapped) diagram against the old (current) diagram. The resulting expression can be simplified and reduced to a ratio that only consists of electron and phonon propagators, which use the previous momentum $\mathbf{p}$ or new momentum $\mathbf{p}'$. Overall we have

$$R_{i,\xi\to\xi'} = \frac{G^{(0)}(\mathbf{p}', \Delta\tau)\, D(\delta t')}{G^{(0)}(\mathbf{p}, \Delta\tau)\, D(\delta t)}, \tag{3.44}$$

where $\Delta\tau$ is $\tau_i - \tau_{i-1}$ with i being the second vertex to swap. δt is sum of temporal position differences between the two arcs before the swap. The value $\delta t'$ is the same sum after the change, i.e., if the indices $1, \ldots, 4$ represent the four involved vertices, where $1 \to 2$ and $3 \to 4$ are connected beforehand, we use

$$\delta t = |\tau_1 - \tau_2| + |\tau_3 - \tau_4|, \tag{3.45}$$

$$\delta t' = |\tau_1 - \tau_3| + |\tau_2 - \tau_4|. \tag{3.46}$$

Reaching the expression in Equation (3.44) is only possible by considering the full equation for a swap and replacing the propagators with their functions given in Equation (3.34) and Equation (3.37). Finally, we can reduce such equations to propagators with different arguments.

By applying this procedure we can generate every possible diagram independent of the order. Naturally, diagrams that are more common appear more often, thus conveying possible symmetry factors. This is ensured by stochastic elements. For instance, a nested diagram as illustrated in Figure 3.11 could be created in two consecutive swap updates. The second swap update needs to handle two vertices that are shifted ± 1 relative to the first one.

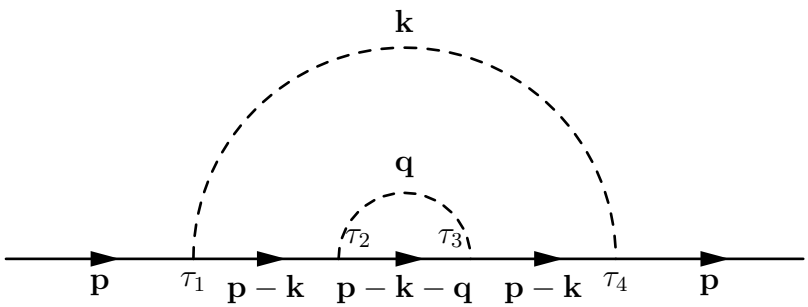


Figure 3.11: Another swap on the second-order Feynman diagram presented in Figure 3.10 forms this structure. The inner arc is directly connected, i.e., it is possible to remove it without further swaps.

3.3.5 Extend Update

The last update procedure is changing the time $t_{\max}$. This is required as we still have one external parameter. The function of the tree level diagram $D_0(t_{\max})$ is given by the electron propagator,

$$D_0(t_{\max}) = G^{(0)}(\mathbf{p_0}, t_{\max}). \tag{3.47}$$

Integration of the tree level diagram yields

$$N = \int_0^{t_\infty} dt\, G^{(0)}(\mathbf{p_0}, t) = \frac{1 - \exp(-E(\mathbf{p_0}, \mu)\, t_\infty)}{E(\mathbf{p_0}, \mu)}, \tag{3.48}$$

where $E(\mathbf{p_0}, \mu) = \mathbf{p_0}^2/2m - \mu$ is the energy at initial momentum $\mathbf{p_0}$. The value of the time t_∞ is supposed to be as large as possible. In practice a value of 100 is more than sufficient. This parameter is chosen once and will not change during the simulation.

Another major difference from this application to the previous example in Section 3.2 is given by the measured values. This simulation does not face any sign changes as the measured values are all positive. There are several ways to confirm this. One way is to look at the matrix elements, which turn out to be part of a symmetric positive definite matrix. Alternatively, we might realize that all variables are either squared or real arguments of the exponential function. This aspect is special for the Fröhlich polaron. In general the polaron problem is a sign alternating series.

3.3.6 Evaluation and Results

We will now look at some of the evaluations that are possible with the data obtained by running such a simulation. Most importantly we need to find a way

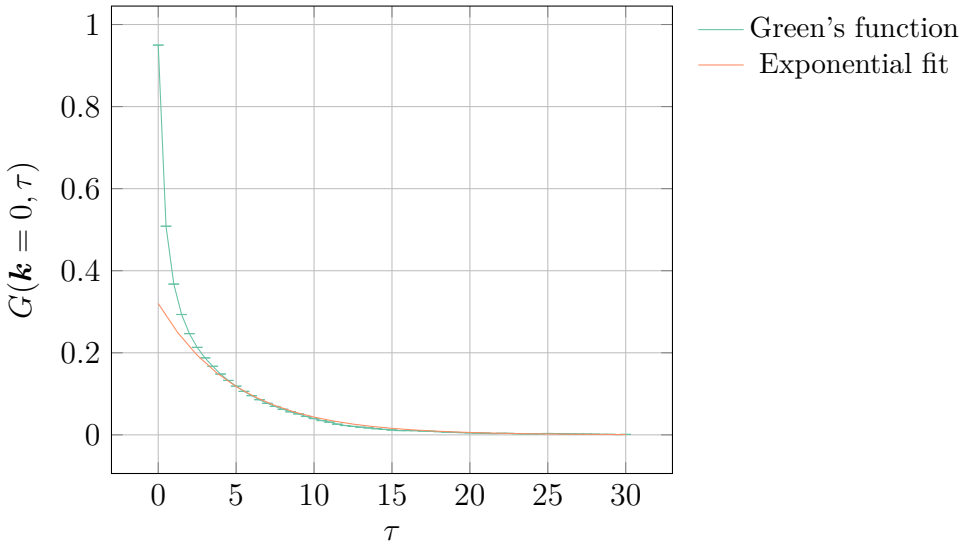


Figure 3.12: The Green's function of the polaron with $\mu = -2.2$ and $\alpha = 2$ for $N = 10^{10}$. The exponential fit is given by $Z_k \exp(\epsilon(\mathbf{k}, \mu)t)$ with $Z_k \approx 0.32$ and $\epsilon \approx -0.2$. The error bars are smaller than the symbols.

to extract the polaron's energy from the given data. One possibility is to use an exponential ansatz, where we determine the parameters with a suitable fitting scheme.

The exponential fit for Figure 3.12 is given by

$$y = Z_k \exp\left(-(E(\mathbf{k}) - \mu)\, t\right), \tag{3.49}$$

where y are the measured data points, i.e., the results of the various bins. The quantity Z_k shows the fraction of the bare electron state in the true eigenstate of the polaron. This is an important physical characteristic of the polaron, usually expressed as in terms of the free particle state F_k and the polaron P_k,

$$Z_k = |\langle F_k | P_k \rangle|^2. \tag{3.50}$$

Now we can apply a linear fit to the values $w_i \equiv \ln(y_i)$. In the end we identify coefficients a and b for the function $w(t) = at + b$. Coming back to our original problem we may calculate

$$Z_k = \exp(b), \tag{3.51}$$

$$E(\mathbf{k}) = \mu - a. \tag{3.52}$$

This gives us an exponential fit, which allows us to extract the energy of the polaron for the given setup. As an example the fit for the data given in Figure 3.12 yields an energy $E(\mathbf{k} = 0) \approx -2.0$.

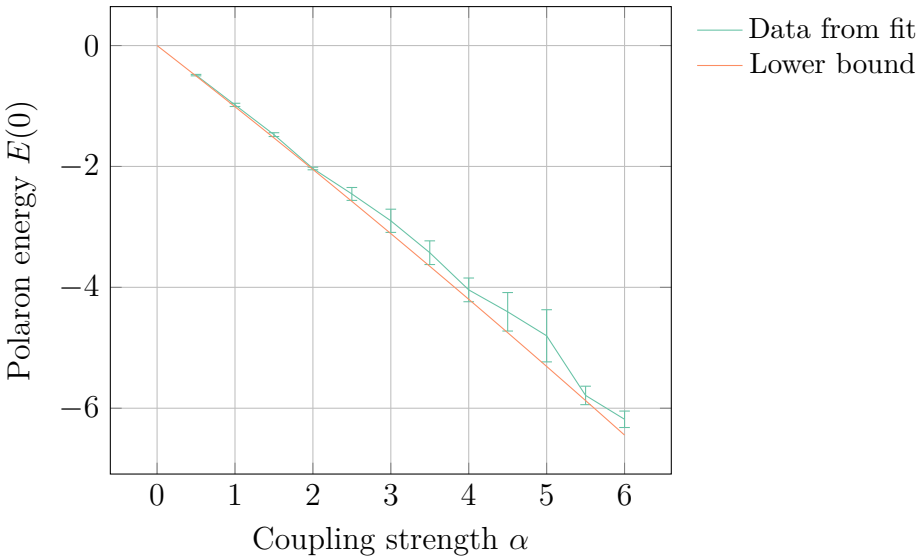


Figure 3.13: Fit of the polaron energy $E(0)$ using the data from simulations with $N = 10^{10}$. The coupling strength α has been varied with a chemical potential μ, that is fine-tuned to be close to $E(\mathbf{k})$. The lower bound is determined by taking the small α approximation.

Using the exponential fit we can evaluate the coupling dependency of the energy $E_0 \equiv E(\mathbf{k} = 0)$. It is advantageous to fine-tune the chemical potential μ to be close to the energy. In the end the algorithm converges much faster if $\exp(-(E_0 - \mu)\,\tau) \approx 1$, i.e., when $E_0 - \mu$ is close to zero. However, it is necessary to constrain $\mu \leq E_0$, otherwise no convergence may be achieved.

The plot in Figure 3.13 shows the dependence of E_0 on the coupling strength α. The lower bound has been determined by using the small α approximation as calculated by Feynman [53]. It can be shown that

$$E_0 \leq -\alpha - \frac{\alpha^2}{81} + \mathcal{O}\left(\alpha^3\right). \tag{3.53}$$

Even though we are have only used a small number of configurations N and a rough energy estimate, the results are already within an acceptable error range. Usually, we would be required to increase N with higher values of α. In this case we did every run with the same coupling constant α, resulting in a larger error for larger couplings.

Another quantity that might be interesting is $E(\mathbf{k})$. We are again fine-tuning the chemical potential μ, not only to support convergence, but to have reliable statistics with an acceptable number of configurations N.

The plot illustrated in Figure 3.14 features the dispersion relation of the polaron. Dispersion relations describe the effect of dispersion in a medium on the properties of a wave traveling within the medium by connecting its wavelength k to its

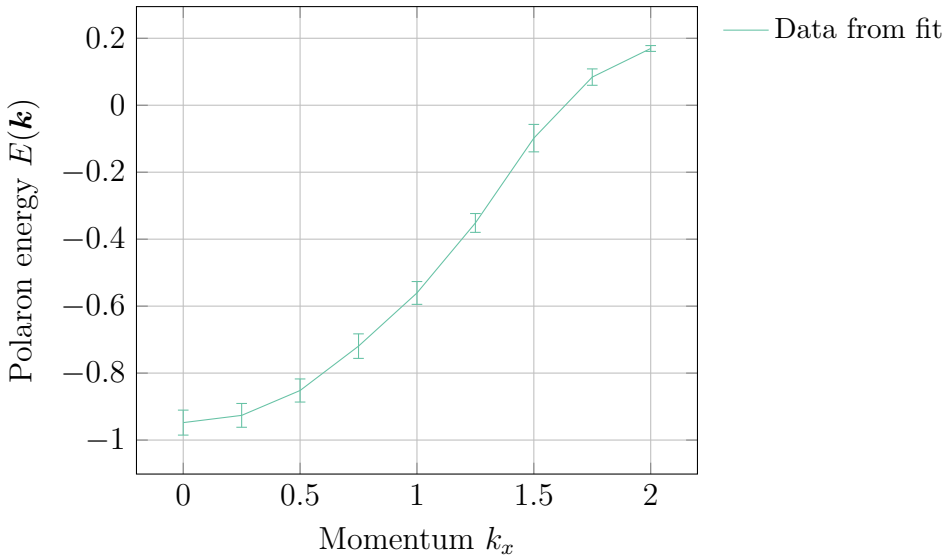


Figure 3.14: Polaron energy $E(\mathbf{k})$ fit to data from simulations with $N = 10^{10}$ at $\alpha = 1$. The chemical potential μ is fine-tuned to be close to the energy.

frequency ω. The momentum $\mathbf{k}$ is proportional to the wavelength k.

The dispersion relation has been obtained for $\alpha = 1$. Again we are using an exponential fit to extract the energy of the polaron. Our results match the research presented by others in experimental [131], numerical [119], or analytical [53] form. We see the characteristic slope of the curve. The polaron problem has been studied with extensions to DiagMC, such as the flat histogram method [40].

4

Anomalous Magnetic Moment

The anomalous magnetic moment of a particle, e.g., an electron, is a contribution that originates from quantum corrections in the contexts of the theory of Quantum Electrodynamics [50]. The correction of the particle's classical magnetic moment can be computed by calculating higher-order Feynman diagrams. This makes computing the anomalous magnetic moment a suitable candidate for studying the diagrammatic Monte Carlo method in the context of QFT.

The anomalous magnetic moment is a great success story for QFT in general. High precision tests with electrons or muons have resulted in an agreement between theory and experiments that remains unmatched until today [74]. Over the years the theoretical, as well as the experimental value have been determined to very high precision.

A recent measurement [66] of the anomalous magnetic moment of the electron reads

$$\frac{g-2}{2} = 0.00115965218073(28), \tag{4.1}$$

where g is the gyromagnetic ratio. It is related to the magnetic moment μ by the equation,

$$\mu = g\,\frac{e\,\hbar}{2m\,c}\,\mathbf{s}, \tag{4.2}$$

with $\mathbf{s}$ denoting the spin. m is the mass of the electron.

The obtained result is slightly more accurate than previous experiments, e.g., [107, 146]. The accuracy went up from 3.7×10^{-9} in 1987 to 0.66×10^{-9} in 2006. Cur-

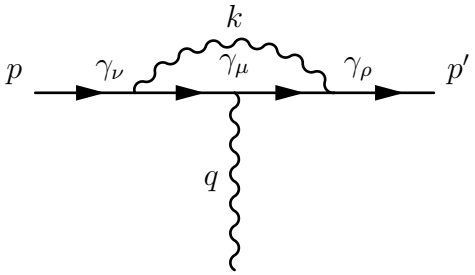


Figure 4.1: The diagrammatic representation of the one-loop correction to the vertex function with fermion momenta p and p'. The loop momentum is given by k, the photon momentum is denoted by q.

rently, the error margin is 0.24×10^{-9}. It is the equivalent of measuring the distance from the Earth to the Moon within the diameter of a single human hair.

Our aim in this field is not to improve current results. Since QED is a well-known and well-tested theory, there is hardly room for improvement. Also numerical $g-2$ calculations have been performed with years of knowledge and experience providing accurate results up to tenth-order [11]. Additionally, well-performing general computations are available [62]. Instead, we aim for a proof of concept to illustrate that DiagMC can be used for such calculations.

This chapter is organized as follows. In Section 4.1 we start by briefly discussing the analytic derivation, as it can be found in textbooks [112, 134]. Then we go into details of our numerical evaluation in Section 4.2. This prepares the presentation of the simulation details in Section 4.3. As we generate all possibly required diagrams beforehand, we need to have a solid diagram generation scheme. We explain our method in Section 4.4. Finally, we discuss results that have been obtained using DiagMC simulations in Section 4.5.

4.1 Analytical Foundation

The anomalous magnetic moment can be calculated from the vertex function. The vertex function describes the scattering of an electron by an external electromagnetic field. The most significant correction comes from the single one-loop, i.e., second-order, diagram. Precise calculations also include higher loop contributions.

For the one-loop diagram we start by simplifying

$$\Gamma^{(1)}_\mu(p',p) = i\int \frac{d^4k}{(2\pi)^4} D_{\rho\nu}(k)(-ie\gamma_\rho)S_F(p'+k)\gamma_\mu S_F(p+k)\,(-ie\gamma_\nu)\,, \tag{4.3}$$

which is the mathematical expression represented by the Feynman diagram shown in Figure 4.1. Here $\Gamma^{(1)}_\mu(p',p)$ is the first correction to the vertex function. Our choice for the (Euclidean) gamma matrices γ_μ is specified in Appendix B.1.

As mentioned in Section 2.3 we deal with this problem in Euclidean space. This explains the additional i in the overall vertex function as compared to common textbook representations, such as [112]. Our choice implies the usage of the Euclidean versions for the photon propagator,

$$D_{\rho\nu}(k) = \frac{i\delta_{\rho\nu}}{k^2}, \tag{4.4}$$

with the Euclidean metric $\delta_{\rho\nu}$, and the fermion propagator,

$$S_F(p) = \frac{-i}{i\not{p} + m} = -\frac{\not{p} + im}{p^2 + m^2}. \tag{4.5}$$

Therefore, the one-loop contribution to the vertex function can be expressed as

$$\Gamma^{(1)}_{\mu}(p', p) = -\frac{ie^2}{(2\pi)^4} \int \frac{d^4k}{D} \gamma_\nu \left(\not{p}' + \not{k} + im\right) \gamma_\mu \left(\not{p} + \not{k} + im\right) \gamma_\nu, \tag{4.6}$$

where we label the denominator as

$$D \equiv k^2 \left[(p + k)^2 + m^2\right] \left[(p' + k)^2 + m^2\right]. \tag{4.7}$$

We can solve Equation (4.6) with Feynman parameters. This technique allows us to rewrite the integral in a simpler form, using identities such as

$$\begin{aligned} A_1^{-m_1} A_2^{-m_2} \ldots A_n^{-m_n} = \\ \int_0^1 dz_1 \ldots dz_n \delta\left(\sum_i z_i - 1\right) \frac{\prod_i z_i^{m_i - 1}}{\left(\sum_i z_i A_i\right)^{\sum_i m_i}} \frac{\Gamma(\sum_i m_i)}{\prod_i \Gamma(m_i)}, \end{aligned} \tag{4.8}$$

where $\delta(x)$ is the Dirac delta function and $\Gamma(x)$ is the gamma function.

Our goal is to identify the charge form factor $F_1(q^2)$ and the magnetic form factor $F_2(q^2)$ from the vertex function. These form factors only depend on q^2 with $q \equiv p' - p$. However, such a separation is only possible if p and p' are on-shell momenta, i.e., they satisfy $p^2 = p'^2 = -m^2$. In this case, we can rewrite Equation (4.6) to look like

$$\Gamma_\mu(p', p) = F_1(q^2)\, \gamma_\mu - F_2(q^2)\, \frac{\sigma_{\mu\nu} q_\nu}{2m}, \tag{4.9}$$

with $\sigma_{\mu\nu} = \frac{i}{2}[\gamma_\mu, \gamma_\nu]$. Note that this separation is valid for all orders.

At the end of the day we arrive at a formula that could be used to compute $F_2(q^2)$, which gives us the one-loop correction to the magnetic moment in the

$q^2 = 0$ case. Generally, the form factors correspond to

$$g = 2\,(F_1(0) + F_2(0)), \tag{4.10}$$

where $a \equiv F_2(0)$ and, by normalization, $F_1(0) = 1$. For calculating the deviation given by the magnetic form factor $F_2(0)$ in the one-loop case we use the Feynman parametrization to change the denominator D of Equation (4.6). We get

$$\frac{1}{D} = \int_0^1 dz_1\, dz_2\, dz_3\, \delta(z_1 + z_2 + z_3 - 1)\, \frac{2}{L^3}, \tag{4.11}$$

which includes a shift in k. With the three Feynman parameters z_1, z_2, and z_3, we introduce $l \equiv k + z_2 p + z_3 p'$ and $\Delta(q^2) \equiv m^2(z_2 + z_3)^2 - z_2 z_3 q^2$. The changed denominator L is given by

$$L = (k + z_2 p + z_3 p')^2 - (z_2 p + z_3 p')^2 = l^2 - \Delta(q^2). \tag{4.12}$$

The integration is then performed over l. For a scattering process we have $q^2 < 0$, resulting in $\Delta(q^2) > 0$. We can interpret $\Delta(q^2)$ as an effective mass.

Now we can separate the original expression for the vertex function to identify the magnetic form factor $F_2(q^2)$. We obtain

$$F_2(q^2) = \frac{\alpha}{2\pi} \int_0^1 dz_1\, dz_2\, dz_3\, \delta(z_1 + z_2 + z_3 - 1) \left(\frac{2m^2\, z_3\, (1 - z_3)}{m^2(1 - z_3)^2 - q^2\, z_1\, z_2} \right) + \mathcal{O}\left(\alpha^2\right), \tag{4.13}$$

which requires the previously noted on-shell condition. Furthermore, we need to use the Gordon identity in Euclidean space, which reads

$$\bar{u}(p')\,(2m\,\gamma_\mu)\,u(p) = \bar{u}(p')\,(\sigma_{\mu\nu} q_\nu - ip' - ip)\,u(p). \tag{4.14}$$

In the end, we find that for $q^2 = 0$ we have $F_2(0) = \alpha/2\pi$, where α denotes the fine-structure constant given by the electric charge e as $e^2/4\pi$.

Following the outlined scheme we can compute the magnetic moment of the electron. The result is

$$g = \begin{cases} 1 & \text{Classical} \\ 2 & \text{Dirac eq. .} \\ 2.002319\ldots & \text{QED} \end{cases} \tag{4.15}$$

In general the corrections from the QED vertex are not the only contributions to the anomalous magnetic moment a, as used in Equation (4.10). The experimental value $a_e^{\text{exp}} = 1\,159\,652\,180.73(28) \times 10^{-12}$ [66] of the electron's magnetic moment

is the sum of many contributions [75], such as

$$a_e = a_e(\text{QED}) + \Delta a_e(\text{QCD}) + \Delta a_e(\text{EW}), \tag{4.16}$$

where $a_e(\text{QED})$ can be split into four different parts,

$$a_e(\text{QED}) = a_e(e) + \Delta a_e(e,\mu) + \Delta a_e(e,\tau) + \Delta a_e(e,\mu,\tau). \tag{4.17}$$

The most significant contribution comes from $a_e(e)$. The other parts are several orders of magnitude smaller. Their individual contributions are calculated to be [75]

$$\Delta a_e(e,\mu) = 2.71(0) \times 10^{-12}, \tag{4.18}$$
$$\Delta a_e(e,\tau) = 0.01(0) \times 10^{-12}, \tag{4.19}$$
$$\Delta a_e(\text{QCD}) = 1.68(2) \times 10^{-12}, \tag{4.20}$$
$$\Delta a_e(\text{EW}) = 0.039(0) \times 10^{-12}. \tag{4.21}$$

The contribution of $\Delta a_e(e,\mu,\tau)$ was omitted as it represents the least significant part with a value of 2.3×10^{-21}. Therefore, it makes sense for our proof of concept study to focus purely on $a_e(e)$.

4.2 Numerical Evaluation

Before we can establish a simulation using DiagMC we do some numerical tests to identify potential problems and come up with a generic ansatz to set up the simulation. In this section we discuss the numerical evaluation of the first- and second-order diagrams. The evaluation is done by integration in conjunction with the trapezoidal rule or the Monte Carlo method.

Numerical integrations of the Feynman integrals for the anomalous magnetic moment are not uncommon. Most of the higher-order integrals are only known from numerical evaluations. Usually, we start by converting momentum space Feynman integrals into Feynman-parametric integrals analytically. Then we would evaluate these integrals using an iterative-adaptive Monte Carlo integration routine such as VEGAS [79].

Other methods to perform calculations of higher-order Feynman diagrams include transformations of the integrals, e.g., using difference equations [85]. This method tries to cluster the integrals in such a way to obtain master integrals, which can be calculated more efficiently.

4.2.1 Form Factor Extraction

We have to start at Equation (4.6). This is mandatory as (automatically generated) higher-order diagrams cannot be simplified analytically as easily as the one-loop diagram. Thus, we do not want to limit our possibilities at this point by assuming special properties of a single diagram for all other diagrams.

Our goal is to compute the magnetic form factor $F_2(q^2)$. Extracting the form factor from the full integral is only possible with a sophisticated projection. We start with the definition of the electromagnetic current J_μ, which is expressed using the previously defined vertex function $\Gamma_\mu(p', p)$ as

$$J_\mu = -ie\,\bar{u}(p')\,\Gamma_\mu(p', p)\,u(p). \tag{4.22}$$

We now define $r \equiv p' + p$ for simplifying expressions that contain sums of the external lepton momenta. This is useful for showing the most general structure of an electromagnetic vertex. We have

$$\begin{aligned}\Gamma_\mu(p', p) = {} & K_1 q_\mu + K_2 r_\mu + K_3 \gamma_\mu + K_4 \not{q} q_\mu \\ & + K_5 \not{r} q_\mu + K_6 \not{q} r_\mu + K_7 \not{r} r_\mu \\ & + K_8 \sigma_{\mu\nu} q_\nu + K_9 \sigma_{\mu\nu} r_\nu + K_{10} \sigma_{\nu\rho} q_\mu q_\nu r_\rho \\ & + K_{11} \sigma_{\nu\rho} r_\mu q_\nu r_\rho + K_{12} \varepsilon_{\mu\nu\rho\delta} \gamma_5 \gamma_\nu q_\rho r_\delta,\end{aligned} \tag{4.23}$$

which contains the full set of information. In the following we need to find the appropriate steps to identify the form factors.

In general the contributions from Feynman diagrams do not have the tensor structure of Equation (4.9). To retrieve this structure and obtain the most general version, as shown above, we have to contract each contribution Γ_μ with the external legs, i.e.,

$$\bar{u}(p')\,\Gamma_\mu\,u(p). \tag{4.24}$$

The contribution can be written in a combinatorial form, expanding all possible contractions of $p' \pm p$, i.e., r and q, with γ matrices. Now we can identify the various factors K_i, which are then required to retrieve the magnetic form factor $F_2(q^2)$. We can recombine these factors to obtain

$$F_1(q^2) = 2im\,K_2 + K_3 - 4m^2\,K_7 - 2m\,q^2\,K_{11} - q^2\,K_{12}, \tag{4.25}$$

$$F_2(q^2) = -2im\,K_2 + 4m^2\,K_7 + 2m\,K_8 + 2m\,q^2\,K_{11} - 4m^2\,K_{12}. \tag{4.26}$$

The decomposition into the various K_i follows directly from applying the Dirac equation on Equation (4.24) with additional usage of the on-shell condition [19]. In Figure 4.2 the various coefficients from Equation (4.26) are illustrated in depen-

dency of the loop momentum k. The exact method of extraction is described for one-loop in Section 4.2.2. In principle we also have a third form factor $q_\mu F_3(q^2)$, but this term vanishes overall.

Only a few factors K_i are actually required for computing $F_2(q^2)$ using Equation (4.26). The projection for the required components reads:

$$K_2 = \frac{1}{4r^2}\mathrm{tr}\left(r_\mu \Gamma_\mu\right), \tag{4.27}$$

$$K_7 = \frac{1}{8r^2}\left(\frac{1}{q^2}\mathrm{tr}\left(q_\mu q_\nu \gamma_\nu \Gamma_\mu\right) + \frac{3}{r^2}\mathrm{tr}\left(r_\mu r_\nu \gamma_\nu \Gamma_\mu\right) - \mathrm{tr}\left(\gamma_\mu \Gamma_\mu\right)\right), \tag{4.28}$$

$$K_8 = \frac{1}{8q^2r^2}\left(r^2\mathrm{tr}\left(\sigma_{\mu\nu} q_\nu \Gamma_\mu\right) + \mathrm{tr}\left(\sigma_{\nu\rho} r_\rho q_\nu r_\mu \Gamma_\mu\right)\right), \tag{4.29}$$

$$K_{11} = \frac{1}{8q^2r^4}\left(r^2\mathrm{tr}\left(\sigma_{\mu\nu} q_\nu \Gamma_\mu\right) + 3\mathrm{tr}\left(\sigma_{\nu\rho} r_\rho q_\nu r_\mu \Gamma_\mu\right)\right), \tag{4.30}$$

$$K_{12} = \frac{-1}{8q^2r^2}\mathrm{tr}\left(\varepsilon_{\mu\nu\rho\delta} r_\delta q_\rho \gamma_5 \gamma_\nu \Gamma_\mu\right). \tag{4.31}$$

Furthermore, we require the following projections for the subtraction scheme presented later in Section 4.2.4. These components can be calculated via

$$K_1 = \frac{1}{4q^2}\mathrm{tr}\left(q_\mu \Gamma_\mu\right), \tag{4.32}$$

$$K_3 = \frac{1}{8}\left(\mathrm{tr}\left(\gamma_\mu \Gamma_\mu\right) - \frac{1}{q^2}\mathrm{tr}\left(q_\mu q_\nu \gamma_\nu \Gamma_\mu\right) - \frac{1}{r^2}\mathrm{tr}\left(r_\mu r_\nu \gamma_\nu \Gamma_\mu\right)\right), \tag{4.33}$$

$$K_4 = \frac{1}{8q^2}\left(\frac{3}{q^2}\mathrm{tr}\left(q_\mu q_\nu \gamma_\nu \Gamma_\mu\right) + \frac{1}{r^2}\mathrm{tr}\left(r_\mu r_\nu \gamma_\nu \Gamma_\mu\right) - \mathrm{tr}\left(\gamma_\mu \Gamma_\mu\right)\right), \tag{4.34}$$

$$K_5 = \frac{1}{4q^2r^2}\mathrm{tr}\left(q_\mu r_\nu \gamma_\nu \Gamma_\mu\right), \tag{4.35}$$

$$K_6 = \frac{1}{4q^2r^2}\mathrm{tr}\left(r_\mu q_\nu \gamma_\nu \Gamma_\mu\right), \tag{4.36}$$

$$K_9 = \frac{1}{8q^2r^2}\left(q^2\mathrm{tr}\left(\sigma_{\mu\nu} r_\nu \Gamma_\mu\right) - \mathrm{tr}\left(\sigma_{\nu\rho} r_\rho q_\nu q_\mu \Gamma_\mu\right)\right). \tag{4.37}$$

Using the antisymmetrization given by γ_5 and the identity

$$\varepsilon_{\gamma\nu\rho\delta}\gamma_\delta\gamma_5 = \gamma_{[\mu}\gamma_\nu\gamma_{\rho]} \tag{4.38}$$

with the fully antisymmetrized product $\gamma_{[\mu}\gamma_\nu\gamma_{\rho]}$, we can compute K_{12} explicitly.

A direct evaluation yields

$$\begin{aligned} K_{12} = \frac{1}{16q^2r^2}\big(&2\mathrm{tr}\left(q_\nu \gamma_\nu r_\mu \Gamma_\mu - r_\nu \gamma_\nu q_\mu \Gamma_\mu\right) + \\ &\mathrm{tr}\left(\left(r_\nu \gamma_\nu q_\mu \gamma_\mu - q_\nu \gamma_\nu r_\mu \gamma_\mu\right)\gamma_\rho \Gamma_\rho\right)\big). \end{aligned} \tag{4.39}$$

The antisymmetrized product includes a combinatorial factor of 1/3!. After ex-

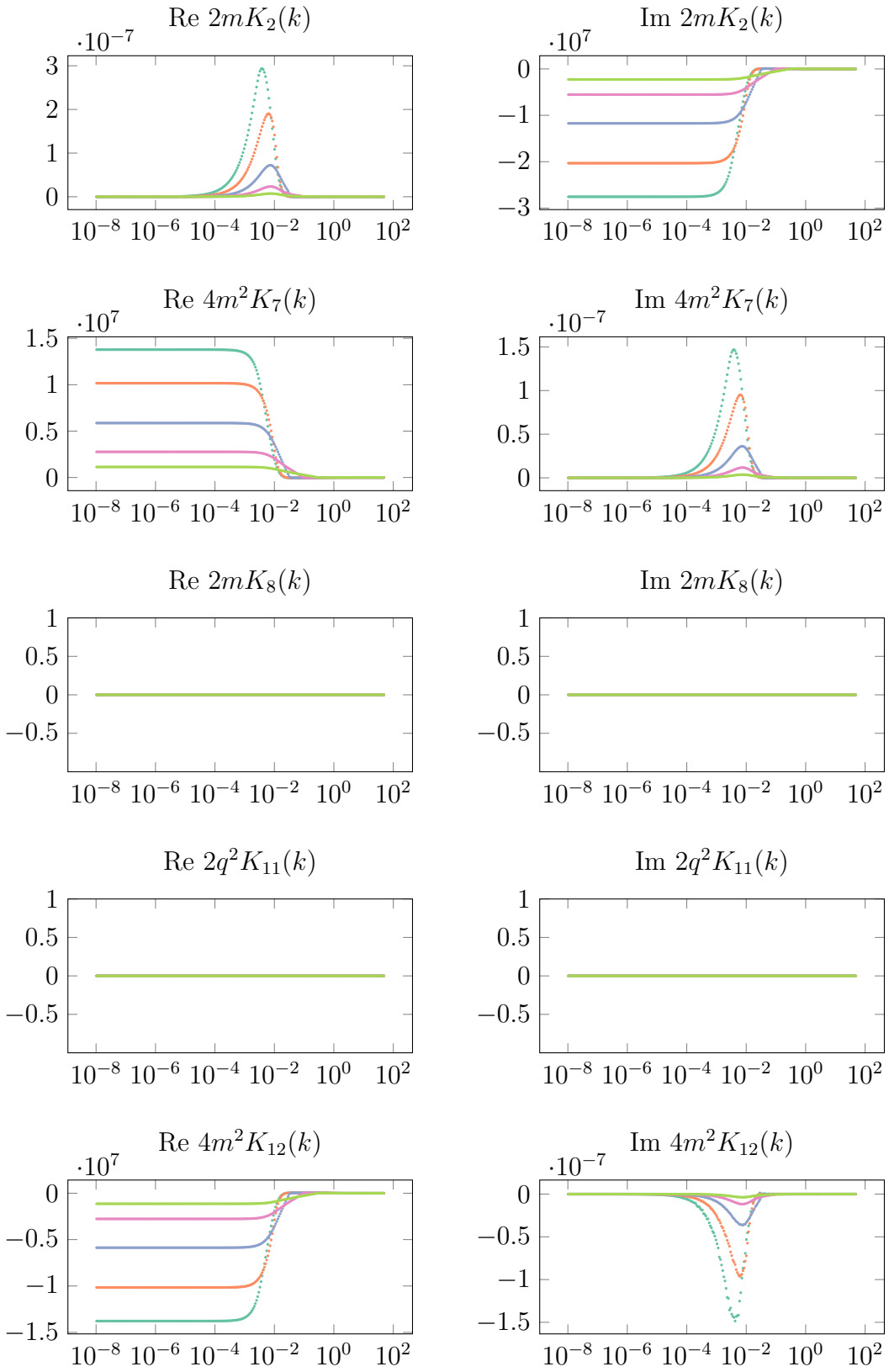


Figure 4.2: The contributing factors K_i in dependence of the loop momentum k for the one-loop diagram. These plots have been created by applying a numerical integration scheme that follows the midpoint rule with $N = 256$. The integrand of the loop momentum contains an additional factor k from a coordinate transformation $k \to \log(k)$. The different lines correspond to $q^2 = -10^{-1}m^2$ (green), $q^2 = -10^{-2}m^2$ (magenta), $q^2 = -10^{-3}m^2$ (blue), $q^2 = -10^{-4}m^2$ (orange), and $q^2 = -10^{-5}m^2$ (teal).

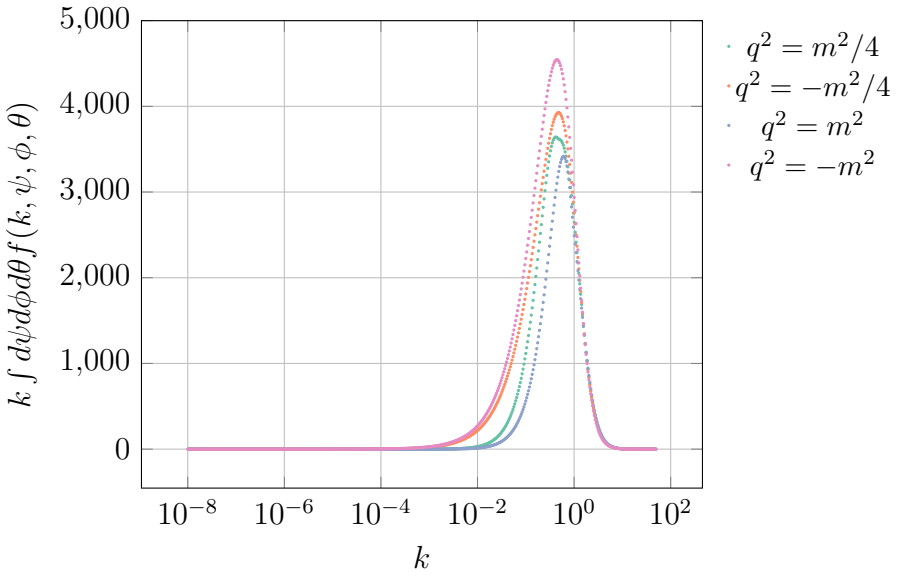


Figure 4.3: Integrand of $F_2^{(1)}(q^2)$ after angular integration using the midpoint rule. We use $N = 512$ for ψ and ϕ. The angle θ has been integrated analytically. The angles ψ and ϕ have been evaluated numerically. The integrand contains an additional factor k coming from the transformation $k \to \log(k)$. The plot shows the values of the integrand in dependence on the integration variable k for some choices of q^2.

panding the antisymmetrized product we obtain an expression that leads directly to the computationally simplified version presented in Equation (4.39).

4.2.2 One-Loop Integration

For understanding how the decomposition works, we look at results for a numerical integration of the one-loop diagram as shown in Figure 4.1. We apply a simple midpoint rule in 4 dimensions. To simplify the problem, we transform the coordinate system to spherical coordinates and directly integrate the third angle. This allows us to evaluate the integrand and the decomposition independently. Additionally, the magnitude k is transformed to a logarithmic scale. This allows us to tune an infrared (IR) cutoff parameter denoted by k_{min}, which shows up as $\log(k_{\text{min}}^2)$ in our evaluation.

From Figure 4.2 we can infer that the IR contributions cancel. Therefore, the one-loop contribution $F_2^{(1)}(q^2)$ to $F_2(q^2)$ is not IR divergent. The imaginary part of the component K_2 is the counter term to the real parts of K_7 and $-K_{12}$ if we consider the right proportionality constants as shown in Equation (4.26). However, due to the used scale we cannot see where the contribution to the real part of K_{12} is arising. In the end, this is a mixture from all three coefficients. The factors K_8 and K_{11} do not contribute at one-loop level.

None of these factors contributes to the imaginary part, which sums up to zero

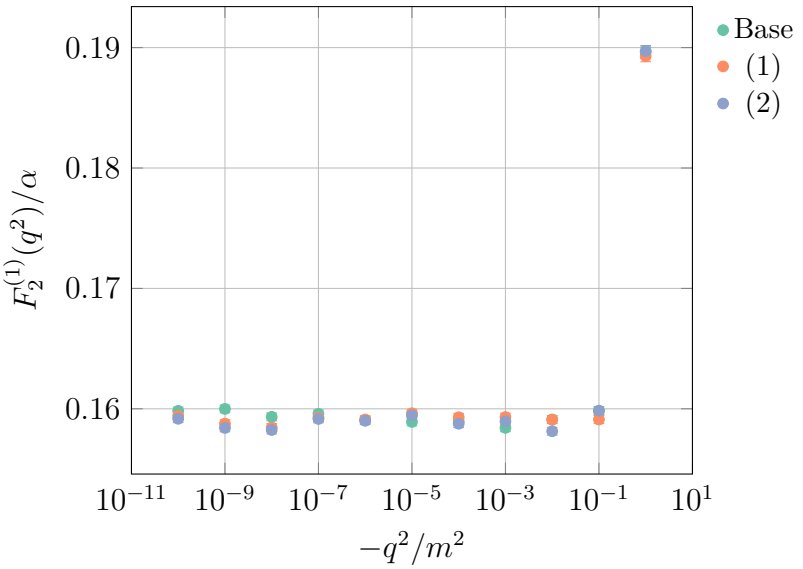


Figure 4.4: Comparison of the integration of the vertex function with the projection to obtain the form factor for different values of q^2/m^2 (named *Base*). (1) This integration uses Equation (4.40). (2) This integration is applying the Equation (4.41). All results have been obtained using Monte Carlo integration with $N = 10^7$ configurations by generating the data points from a spherical distribution.

after performing the integration over the angle coordinates. In the end, we can see a finite, although very small, contribution to the imaginary part coming from the real part of K_2, as well as the imaginary parts of K_7 and K_{12}. These values, however, cancel exactly after multiplying them with the proportionality constants of Equation (4.26). We do not have any contributions from K_{11} and K_{12} to the imaginary part.

If we look at the value of the integrand for different values of q^2, we see a very similar distribution as shown in Figure 4.3. This plot uses a numerical integration with the midpoint rule. The angle coordinates are already integrated, leaving the magnitude of the k vector for integration. The integrand contains an additional factor k due to the logarithmic transformation. Even though all contributions seem to vanish on the scale shown in Figure 4.2, we see a peak arising at $k \sim m^2/4$. After the peak all curves seem to share the same curve.

It makes sense to integrate the vertex function using different levels of analytic transformations for comparison. The result of this comparison is shown in Figure 4.4. We apply the projection described in Section 4.2.1) using the decomposition of the corresponding factors from Equation (4.26). Starting with Equation (4.6) we can use some of the identities from Appendix B.2 to obtain

$$F_2^{(1)}(q^2) = \frac{8m^2e^2}{(2\pi)^4} \int d^4k \frac{1}{D} \left(\frac{k^2}{r^2} - \frac{(q \cdot k)^2}{q^2 r^2} - \frac{3(r \cdot k)^2}{r^4} - \frac{r \cdot k}{r^2} \right), \tag{4.40}$$

where the denominator D remains the one from Equation (4.7). This allows us to directly apply a four-dimensional numerical integration for obtaining the value of the one-loop contribution to the magnetic form factor $F_2(q^2)$.

Another possibility is to transform the integration variables from Cartesian coordinates to spherical coordinates. This allows us to directly integrate over one of the angles. The advantage is that the potential problem in the denominator becomes less dangerous.

The result for calculating the first-order correction of the magnetic form factor $F_2^{(1)}(q^2)$ after simplifying the integral with the transformation to spherical coordinates is

$$F_2^{(1)}(q^2) = \frac{4m^2\alpha}{\pi^2} \int dk\, d\theta\, d\psi\, \frac{k\, l^3\, \sin(\psi)}{D} \left(l\,(1 - 3\cos^2(\psi)) - h \right), \tag{4.41}$$

where we use $l \equiv k\sin(\theta)$ and $h \equiv r\cos(\psi)$.

All shown ways yield the same results for tested values of q^2/m^2. The difference lies in their efficiency. To obtain an accurate result by evaluating the integral of Equation (4.41) requires less effort than by using Equation (4.40). The denominator D depends on both angles. We have

$$D = k^2 r^2 \left(k^2 + l\,h - q\,k\cos(\theta)\right)\left(k^2 + l\,h + q\,k\cos(\theta)\right). \tag{4.42}$$

4.2.3 Two-Loop Integration

At this point it makes sense to look at two-loop, i.e., fourth-order, diagrams. Here it is useful to prefer Monte Carlo integration over deterministic methods as outlined in Section 2.1.4. This time, however, we have more than a single diagram. We should evaluate all of them to find potential problems.

In Figure 4.5 the contributing two-loop diagrams are shown. For the two-loop level we have to deal with 9 different diagrams, where three diagrams have the same topology, but use $m \in \{m_e, m_\mu, m_\tau\}$ for the inner fermion loop. The physical reason is that there is a likelihood of not only creating virtual electrons in the process, but also muons or τ leptons.

We start with a discussion of the integrals representing the nine two-loop diagrams. For the diagrams shown in Figure 4.5a, Figure 4.5b, and Figure 4.5c we have

$$\begin{aligned}\Gamma^{(2,1)}_{\mu,m_e}(p',p) = (-1)\,i^2 \int \frac{d^4k}{(2\pi)^4}\frac{d^4k'}{(2\pi)^4} D_{\nu\delta}(k) D_{\rho\sigma}(k')(-ie\gamma_\nu) S_F(p'+k) \\ \gamma_\mu S_F(p+k)(-ie\gamma_\rho)(-ie\gamma_\delta) S_{F,m_e}(k-k')(-ie\gamma_\sigma) S_{F,m_e}(k).\end{aligned} \tag{4.43}$$

The different masses would be included by using different Feynman propagators

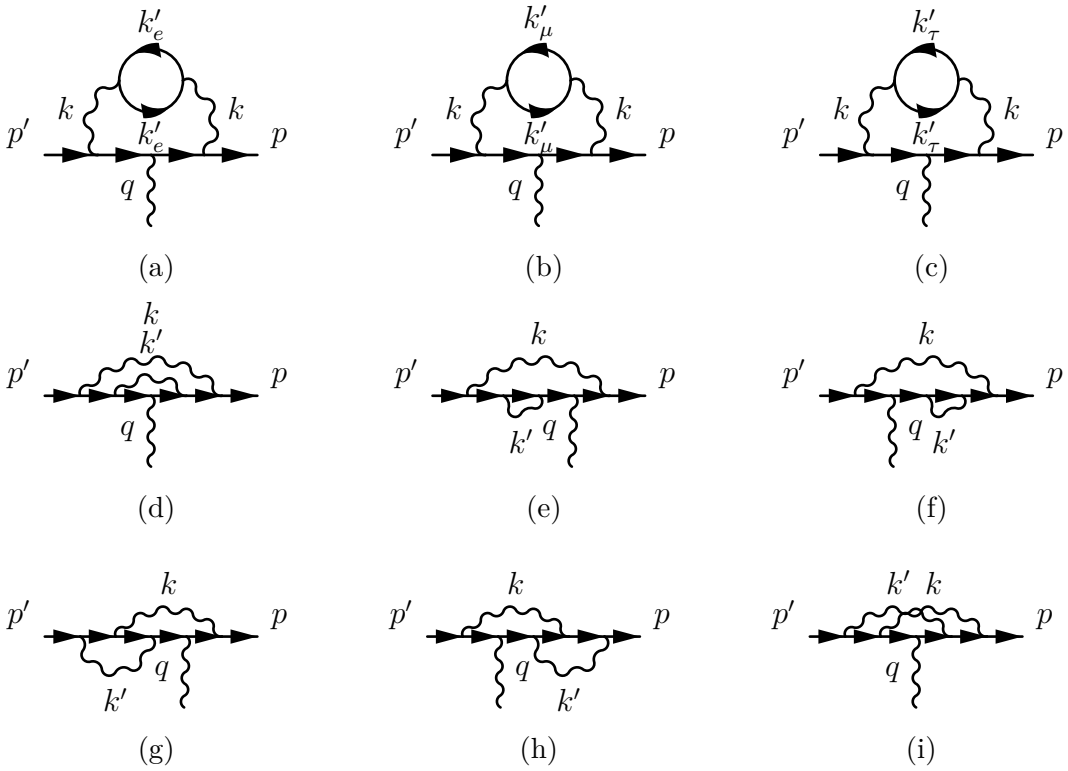


Figure 4.5: The 9 two-loop diagrams for the QED vertex correction.

$S_F(p)$, i.e., propagators that use, e.g., $m = m_\mu$, instead of $m = m_e$. For our purposes we will only use $m = m_e$, thus omitting the two (suppressed[1]) diagrams from the evaluation.

The diagram from Figure 4.5d is represented by

$$\Gamma^{(2,2)}_\mu(p',p) = i^2 \int \frac{d^4k}{(2\pi)^4}\frac{d^4k'}{(2\pi)^4} D_{\rho\nu}(k) D_{\sigma\delta}(k')(-ie\gamma_\nu) S_F(p'+k) \\ (-ie\gamma_\delta) S_F(p'+k+k')\gamma_\mu S_F(p+k+k')(-ie\gamma_\sigma) S_F(p+k)(-ie\gamma_\rho). \tag{4.44}$$

This one is the naive two-loop diagram, which just duplicates the direct photon propagator from the one-loop diagram.

In Figure 4.5e we see another obvious two-loop diagram. By inserting a self-energy correction diagram on the inner electron propagator we obtain

$$\Gamma^{(2,3)}_\mu(p',p) = i^2 \int \frac{d^4k}{(2\pi)^4}\frac{d^4k'}{(2\pi)^4} D_{\rho\nu}(k) D_{\sigma\delta}(k')(-ie\gamma_\nu) S_F(p'+k) \\ \gamma_\mu S_F(p+k)(-ie\gamma_\sigma) S_F(p+k+k')(-ie\gamma_\delta) S_F(p+k)(-ie\gamma_\rho). \tag{4.45}$$

Similarly, we have the same setup for Figure 4.5f. Here the self-energy correction

[1] The two-loop diagrams for m_μ and m_τ have coefficients 5.197×10^{-7} and 1.837×10^{-9} [11]. Their contribution is roughly 1.7×10^{-6} to the overall two-loop correction.

is applied to the (inner) outgoing electron propagator. We have

$$\begin{aligned}\Gamma_{\mu}^{(2,4)}(p',p) = i^2 \int \frac{d^4k}{(2\pi)^4}\frac{d^4k'}{(2\pi)^4} D_{\rho\nu}(k) D_{\sigma\delta}(k')(-ie\gamma_\nu) S_F(p'+k) \\ (-ie\gamma_\delta) S_F(p'+k+k')(-ie\gamma_\sigma) S_F(p'+k')\gamma_\mu S_F(p+k)(-ie\gamma_\rho).\end{aligned} \tag{4.46}$$

Effectively, there is not much difference to the representation for Figure 4.5e as it is time-reversal symmetric.

The vertex correction can be applied by passing an existing vertex, e.g., a vertex that holds a photon propagator connecting the incoming and the outgoing propagators. In Figure 4.5g we see this correction on the incoming propagator. We have

$$\begin{aligned}\Gamma_{\mu}^{(2,5)}(p',p) = i^2 \int \frac{d^4k}{(2\pi)^4}\frac{d^4k'}{(2\pi)^4} D_{\rho\nu}(k) D_{\sigma\delta}(k')(-ie\gamma_\nu) S_F(p'+k) \\ \gamma_\mu S_F(p+k)(-ie\gamma_\sigma) S_F(p+k+k')(-ie\gamma_\rho) S_F(p+k')(-ie\gamma_\delta).\end{aligned} \tag{4.47}$$

Similarly, we see the time-reversal symmetric diagram in Figure 4.5h. In this case, we have the correction on the outgoing side, leading to

$$\begin{aligned}\Gamma_{\mu}^{(2,6)}(p',p) = i^2 \int \frac{d^4k}{(2\pi)^4}\frac{d^4k'}{(2\pi)^4} D_{\rho\nu}(k) D_{\sigma\delta}(k')(-ie\gamma_\delta) S_F(p'+k') \\ (-ie\gamma_\nu) S_F(p'+k+k')(-ie\gamma_\sigma) S_F(p'+k)\gamma_\mu S_F(p+k)(-ie\gamma_\rho).\end{aligned} \tag{4.48}$$

Finally, we may cross the arcs resulting in a diagram similar to Figure 4.5d. In Figure 4.5i we see such a process. We have

$$\begin{aligned}\Gamma_{\mu}^{(2,7)}(p',p) = i^2 \int \frac{d^4k}{(2\pi)^4}\frac{d^4k'}{(2\pi)^4} D_{\nu\delta}(k) D_{\rho\sigma}(k')(-ie\gamma_\sigma) S_F(p'+k') \\ (-ie\gamma_\nu) S_F(p'+k+k')\gamma_\mu S_F(p+k+k')(-ie\gamma_\rho) S_F(p+k)(-ie\gamma_\delta).\end{aligned} \tag{4.49}$$

From the analytical representations we can form a set of potential difficulties and hints that are useful for further investigation.

The plots contained in Figure 4.6 show an integration over the previously defined analytical representation of Figure 4.5d performed with Monte Carlo integration using $N = 10^6$ configurations. For illustration purposes we use the custom $\operatorname{logsgn}(x)$ function, which is defined as

$$\operatorname{logsgn}(x) = \begin{cases} +\log(1+x) & \text{if } x \geq 0, \\ -\log(1-x) & \text{otherwise.} \end{cases} \tag{4.50}$$

This allows us to grasp as much information from the given plots as possible.

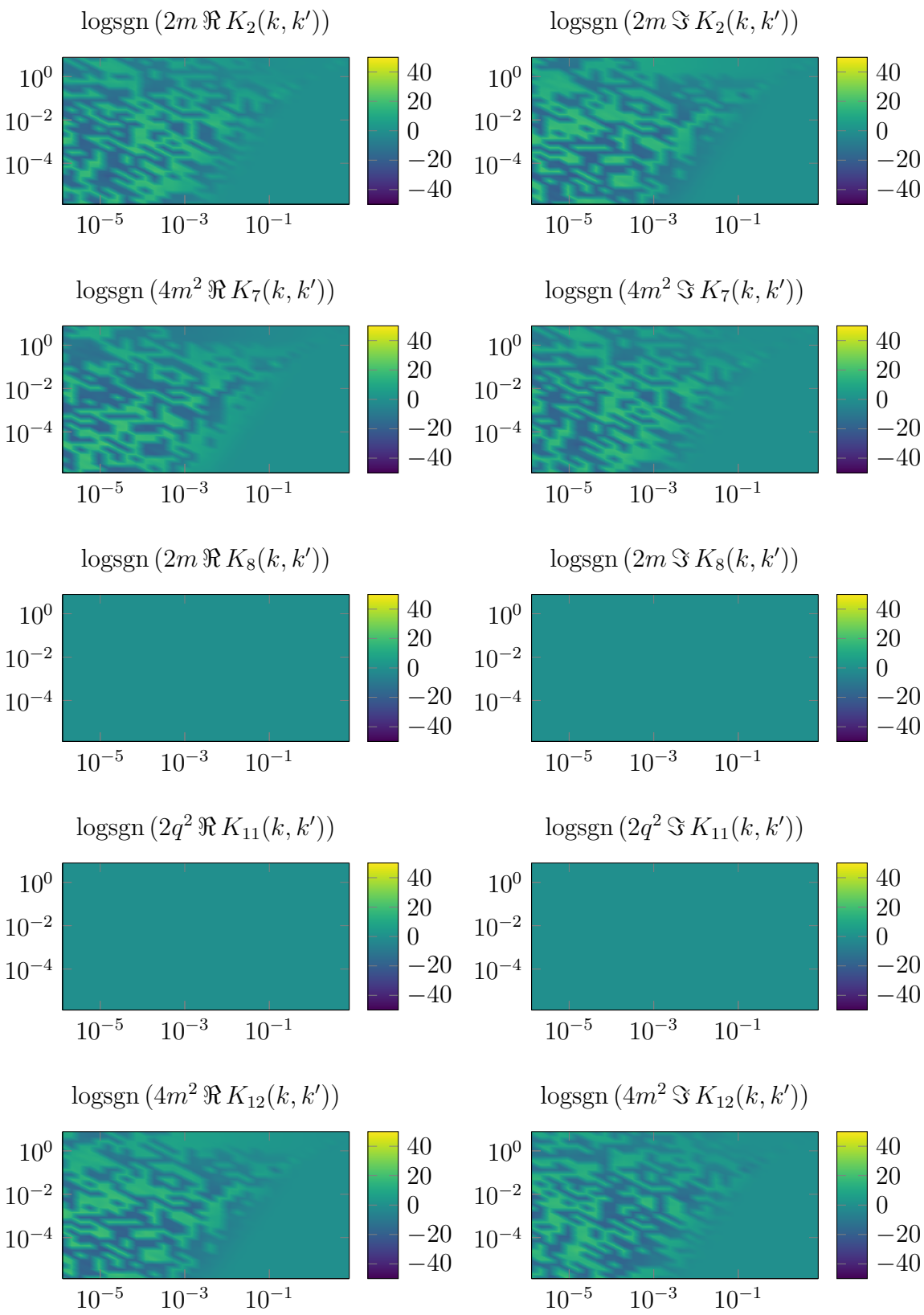


Figure 4.6: Integration over the angles of the two-loop diagram shown in Figure 4.5d for $\Gamma_\mu^{(2,2)}(p',p)$ with $q^2 = -10^{-5}m^2$. The horizontal axis shows the magnitude of k, the vertical axis shows the magnitude of k'. The integrand contains an additional factor kk' coming from the transformations $k \to \log(k)$ and $k' \to \log(k')$. The angles have been integrated using Monte Carlo integration ($N = 10^6$ configurations). The logsgn (x) function is defined in Equation (4.50).

As with the one-loop integration shown in Figure 4.2 we observe that the diagram does not show a contribution of the factors K_8 and K_{11}. We do, however, get some contributions (especially in terms of K_8) from other two-loop diagrams (Figure 4.5g and Figure 4.5i).

The sign changes are mostly noise coming from the integration scheme. The only physical meaning can be associated to the edges, i.e., between zero and non-zero regions. Using a combination of angles and opposite angles we would see that only the imaginary part of K_2, as well as the real parts of K_7 and K_{12} do not vanish. This cancellation occurs in the statistical limit naturally, however, does not take place in our case with only 10^6 samples.

The arising problems can be seen by looking at the bounding box. For $k \to \infty$ and $k' \to \infty$ we require that the function yields zero. Even though there may be some kind of cancellation in total, we cannot deal with the shown divergences in our simulation. We need an efficient treatment to obtain meaningful results in our sampling scheme.

Most diagrams in the two-loop order are problematic by definition. For instance the diagrams in Figure 4.5a, Figure 4.5b, and Figure 4.5c contain closed fermion loops. The diagrams in Figure 4.5e and Figure 4.5f come with direct self-energy corrections. In general, these corrections result in divergences, which would spoil the simulation. Plots for the other diagrams are appended to Appendix D.1.

From the previously shown plots we can only conclude that we require an efficient scheme for numerically calculating the values of higher-loop diagrams. We choose to follow the methods and guidelines given in [10]. In the following we indicate the most important steps for a regularization scheme leading to a renormalization of the divergent diagrams.

4.2.4 Subtraction Procedure for Higher-Order Diagrams

Analytically, the IR and UV divergences can be attacked, e.g., by introducing a Feynman cutoff to the photon propagator, i.e., substituting

$$\frac{1}{k^2} \to \frac{1}{k^2+\lambda^2} - \frac{1}{k^2+\Lambda^2} = \int_{\lambda^2}^{\Lambda^2} \frac{dx}{(k^2+x)^2}, \tag{4.51}$$

where Λ represents the UV cutoff and λ the IR cutoff. This, in combination with other diagrams, produces finite, cutoff-dependent quantities as $\Lambda^2 \to \infty$, $\lambda^2 \to 0$. However, we want to use a numerical scheme that provides point-by-point cancellation [28].

The origin of IR divergence in higher-order diagrams is the vanishing of the denominator of the photon propagator k^{-2} in the limit $k \to 0$ in conjunction with

lepton propagators [87]. The photon propagator alone would give a finite result on the integration over k. When the external momentum p is constrained by the on-shell condition $p^2 = -m^2$, the denominator of the lepton propagator behaves as

$$\frac{1}{(p+k)^2 + m^2} = \frac{1}{2p \cdot k + k^2} \approx \frac{1}{2p \cdot k}, \tag{4.52}$$

in the limit of small k. The logarithmic IR divergence can be observed in a scenario involving a photon propagator and two such propagators. For self-energy subdiagrams we find, however, three propagators. Hence, the divergence is actually linear instead of logarithmic.

In general, we have four types of subdiagrams that are responsible for UV divergences. These are quantified by their external electron (N_e) and photon (N_γ) propagators. The degree of UV divergence of a Feynman diagram G, called $\omega(G)$, can be obtained by

$$\omega(G) = 4 - \left(N_\gamma + \frac{3}{2} N_e\right). \tag{4.53}$$

UV divergences appear if $\omega(G) \geq 0$. This can be verified for all problematic types of diagrams:

- Electron self-energy ($N_e = 2$, $N_\gamma = 0$)
- Photon self-energy ($N_e = 0$, $N_\gamma = 2$)
- Vertex-like ($N_e = 2$, $N_\gamma = 1$)
- Photon-Photon scattering ($N_e = 0$, $N_\gamma = 4$)

The full implementation of a system that generates correctly renormalized integrands is beyond the scope of this work. Nevertheless, the implementation should still be good enough to provide sufficiently accurate values. We use the previously calculated subtractions and values [148] to fine-tune our integrands. In the following paragraphs we illustrate how to apply the scheme by discussing the formerly evaluated two-loop diagram.

The goal of the subtraction method is to define an operator $\hat{R}_G$ that acts on a Feynman diagram G resulting in a new representation without divergences. This means that we can obtain the renormalized contribution of the two-loop diagram $\Gamma_\mu^{(2,2)}$ to the anomalous magnetic moment by using

$$F_{2,\text{ren.}}^{(2,2)}(q^2) = \hat{R}_G \left(\Gamma_\mu^{(2,2)}\right), \tag{4.54}$$

where G denotes the diagram illustrated in Figure 4.5d.

In the scheme from [148] the operator $\hat{R}_G$ is represented by a sum of operators,

$$\hat{R}_G = \sum_{\substack{F=\{g_1,\ldots,g_n\}\in\mathfrak{F}[G],\\ G'\in\mathfrak{J}[G]\cap F}} (-1)^{n-1}\hat{M}_{g_1}^{F,G'}\cdots\hat{M}_{g_n}^{F,G'}. \tag{4.55}$$

The symbol $\mathfrak{F}[G]$ is used to indicate the family of all forests F of the diagram G, where $G \in F$. A forest F is a nonempty subset of the set

$$\{G, G_1, \ldots, G_m\}, \tag{4.56}$$

with UV divergent subdiagrams $G_1, \ldots, G_m \subset G$. While the set from Equation (4.56) contains overlapping subdiagrams, a forest is restricted to subdiagrams that do not share a vertex or propagator.

Furthermore, we define $\mathfrak{J}[G]$ to be the set of UV divergent vertex-like diagrams $G' \subseteq G$ connecting to the external photon leg.

In case of our previously discussed two-loop diagram we find

$$\mathfrak{F}[G] = \{\{G\}, \{G_1, G\}\}, \tag{4.57}$$

$$\mathfrak{J}[G] = \{G_1, G\}, \tag{4.58}$$

where G_1 denotes the inner subdiagram, which is equivalent to the one-loop diagram discussed in Section 4.2.2. In this case the sum of the operators from Equation (4.55) consists of three terms, namely

$$\hat{R}_G = \hat{M}_G^{\{G\},G} - \hat{M}_G^{\{G_1,G\},G_1}\hat{M}_{G_1}^{\{G_1,G\},G_1} - \hat{M}_G^{\{G_1,G\},G}\hat{M}_{G_1}^{\{G_1,G\},G}. \tag{4.59}$$

At this point we need to define the operator $\hat{M}_{G''}^{F,G'}$. In the scheme from Volkov [148] the operator is just a placeholder corresponding to

$$\hat{M}_{G''}^{F,G'} = \begin{cases} \hat{A}_{G''} & \text{if } G' = G'', \\ \hat{U}_{G''} & \text{if } G'' \notin \mathfrak{J}[G] \text{ or } G'' \subset G', \\ \hat{L}_{G''} & \text{if } G'' \in \mathfrak{J}[G] \text{ and } G' \subset G'' \neq G, \\ \hat{L}_{G''} - \hat{U}_{G''} & \text{if } G'' = G \text{ and } G' \neq G. \end{cases} \tag{4.60}$$

These operators are specified to behave as follows:

- $\hat{A}_G$ projects the diagram G to obtain its magnetic form factor, which contributes to a_e, i.e.,

  $$\hat{A}\left(\Gamma_\mu(p', p)\right) = F_2(q^2). \tag{4.61}$$

  The required operations can be read of from Equation (4.26).

- $\hat{U}_G$ depends on the type of diagram. It distinguishes between the following three types of diagrams:

  - Self-energy subdiagrams $\Sigma(p)$, e.g., the subdiagram contained in Figure 4.5e. The operator replaces p^2 by $-m^2$. This requires identifying the diagram, expressing it mathematically using the Feynman rules, and performing the replacement by taking p on-shell. This is all prepared by the code generator described in Section 4.4.
  - Vacuum polarization subdiagrams $\Pi(q)$, e.g., the subdiagram contained in Figure 4.5a. Here we need to perform a Taylor expansion for small q up to order $\omega(G)$. Similarly to the previous case, the subdiagrams need to identified and transformed into expressions by the code generator. For simplicity of the code generation it was decided to compute the derivatives numerically using central finite differences.
  - Diagrams for the vertex correction $\Gamma_\mu(p, 0)$. We have to remove parts that are proportional to p_μ. This boils down to

    $$\hat{U}\left(\Gamma_\mu\right) = a(-m^2) + m\, d(-m^2). \tag{4.62}$$

    The diagram in Figure 4.5d contains such a subdiagram. More details on the coefficients from Equation (4.62) follow later, see Equation (4.66) and Equation (4.69).

- $\hat{L}_G$ performs the on-shell renormalization of vertex-like diagrams $\Gamma_\mu(p, 0)$. We need to replace p^2 with $-m^2$ and remove parts proportional to $\sigma_{\mu\nu}$. We can use

  $$\hat{L}\left(\Gamma_\mu\right) = a(-m^2) + m\, b(-m^2) + m^2\, c(-m^2), \tag{4.63}$$

  with details on these coefficients following later, see Equation (4.66), Equation (4.67), and Equation (4.68).

Using these operators in Equation (4.59) we find that our two-loop diagram G from Figure 4.5d contains a vertex-like subdiagram G_1, which leads to

$$\hat{R}_G = \hat{A}_G - \left(\hat{L}_G - \hat{U}_G\right)\, \hat{A}_{G_1} - \hat{A}_G\, \hat{U}_{G_1}. \tag{4.64}$$

This form gives us a description of the different tasks that we need to take care of to construct an expression representing the diagram without any divergences.

The remaining question is how the coefficients from Equation (4.62) and Equation (4.63) can be retrieved. Going back to the full vertex given in Equation (4.23)

we can see that

$$\begin{aligned}\Gamma_\mu(p,0) = (K_1+K_2)\; p_\mu + K_3\,\gamma_\mu + (K_4+K_5+K_6+K_7)\; \not{p}\, p_\mu \\ + (K_8+K_9)\; \left(\not{p}\,\gamma_\mu - \gamma_\mu\,\not{p}\right),\end{aligned} \tag{4.65}$$

represents the number of contributions that may be projected using the formerly stated coefficients K_i, which can be retrieved using the projectors given in Equation (4.27) to Equation (4.37).

To be consistent with the scheme from Volkov we need to identify

$$a(p^2) \equiv K_3, \tag{4.66}$$

$$b(p^2) \equiv K_1 + K_2, \tag{4.67}$$

$$c(p^2) \equiv K_4 + K_5 + K_6 + K_7, \tag{4.68}$$

$$d(p^2) \equiv K_8 + K_9. \tag{4.69}$$

Hence we already have all the projectors to apply the operator $\hat{M}^{F,G'}_{G''}$ on vertex-like diagrams. Now we can perform the outlined procedure given in Equation (4.62) and Equation (4.63).

The full expression for obtaining the contribution of the herein discussed two-loop diagram may be expressed as

$$\begin{aligned}F^{(2,2)}_{2,\text{ren.}}(q^2) = F^{(2,2)}_2(q^2)\; \left(1 - a_{G_1}(-m^2) + m\, d_{G_1}(-m^2)\right) \\ - \left(m\, b_G(-m^2) + m^2\, c_G(-m^2) - m\, d_G(-m^2)\right)\; F^{(1)}_2(q^2),\end{aligned} \tag{4.70}$$

where the additional index denotes the type of diagram to use for the projection. The subtraction scheme needs to be performed directly on the integrand. Details on the application of this scheme are given in Section 4.3.2. It is required to integrate the scheme in our diagram generation algorithm explained in Section 4.4. In the following we will exclusively refer to $F_{2,\text{ren.}}$ as F_2.

4.3 Simulation Details

As far as the DiagMC simulation is concerned we can apply our knowledge from the examples in Chapter 3. We need to construct transition probabilities that form an ergodic set of updates, such that every diagram and value can in principle be visited.

To support our simulation we have to use distributions for the internal parameters that enhance the convergence of the simulation. In principle, we are free to choose an arbitrary distribution, however, in practice we would suffer from a very inefficient computation.

4.3.1 Parameter Distributions

Our first task is therefore to find a set of useful distributions for generating the internal momentum k. Then we can define the update procedures for our simulation. The trivial choice is to use a four-vector

$$k \equiv (k_1, k_2, k_3, k_4), \tag{4.71}$$

where we generate each component k_i uniformly. However, keeping in mind that additional constraints, such as

$$k_{\min} \leq |k| \leq k_{\max}, \tag{4.72}$$

may be useful or even required, we can estimate that the trivial choice is probably not working well. We use the boundaries $k_{\min}$ and $k_{\max}$ to limit the parameter to the contributing range. In the simplest scenario they can be inferred from plots like the one shown in Figure 4.3.

A suitable choice that fulfills Equation (4.72) without much problems is to choose spherical coordinates. This choice gives us three angles and a magnitude, which could be generated in a way that obeys Equation (4.72) by construction.

When we transform any integral from a tesseract to a 3-sphere with parameters $\{r, \varphi, \theta, \psi\}$ we need to insert the following Jacobian J, given by

$$J = r^3 \, \sin^2(\theta) \, \sin(\psi). \tag{4.73}$$

At this point we can rewrite our original distribution $\Omega_k(k_1, k_2, k_3, k_4)$ to the new parameters. We use

$$\Omega_k(k) \equiv \Omega_k(r, \varphi, \theta, \psi) = \Omega_r(r) \, \Omega_\varphi(\varphi) \, \Omega_\theta(\theta) \, \Omega_\psi(\psi). \tag{4.74}$$

The distribution $\Omega_k(k)$ is composed of the distributions for each parameter.

As an example the distribution of $\Omega_\varphi(\varphi)$ is a simple uniform distribution normalized to the parameter space $\varphi \in [0, 2\pi]$. We have

$$\Omega_\varphi(\varphi) = \frac{1}{2\pi}, \tag{4.75}$$

as we expect from such a uniform distribution.

Naively, we might be tempted to use uniform distributions for all three angles. However, there is a better choice - at least for the distributions of ψ and θ - coming from the Jacobian of Equation (4.73).

The distribution of ψ can then be identified as

$$\Omega_\psi(\psi) = \frac{1}{2}\sin(\psi), \tag{4.76}$$

where the factor $1/2$ has been obtained by normalization. To generate variables according to this distribution we need to follow the techniques mentioned in Section 2.1.2.

First we derive the cumulative distribution function. We obtain

$$F(\psi) = \frac{1}{2}\int_0^\psi d\psi' \, \sin(\psi') = \frac{1}{2}\left(1 - \cos(\psi)\right), \tag{4.77}$$

which allows us to write $x = F(\psi)$. Solving this equation for ψ yields

$$\psi = \arccos(1 - 2x), \qquad x \in [0, 1]. \tag{4.78}$$

The previous example is one of the rare situations where both, the cumulative distribution function and its inverse, exist in an analytical form.

The same scheme can only be applied partially for θ. As previously, we can identify the correct distribution from the Jacobian. We get

$$\Omega_\theta(\theta) = \frac{2}{\pi}\sin^2(\theta), \tag{4.79}$$

which has been normalized by a factor $2/\pi$. Computing the cumulative distribution function is possible as well. For the distribution $\Omega_\theta(\theta)$ we obtain

$$F(\theta) = \frac{2}{\pi}\int_0^\theta d\theta' \, \sin^2(\theta') = \frac{2}{\pi}\left(\theta - \cos(\theta)\,\sin(\theta)\right). \tag{4.80}$$

The only remaining part is to find the inverse of $F(\theta)$. We need to solve

$$x \stackrel{!}{=} F(\theta), \qquad x \in [0, 1]. \tag{4.81}$$

As this cannot be solved analytically, we will use a numerical scheme to generate θ in our simulation. One possibility is to use an accept-reject method. This, however, is quite expensive and wastes too many random values. Another possibility is to define a lookup table $\{x_i, \theta_j\}$ for Equation (4.81).

The main advantage of a lookup table is its speed. We generate N entries and approximate the result linearly from the already computed answers for $x \in [0, 1]$ via

$$\theta = (\theta_{j+1} - \theta j)\,\frac{x - x_j}{x_{j+1} - x_j} + \theta_j, \qquad j = \lfloor xN \rfloor. \tag{4.82}$$

Depending on the choice of N and the function we may need a higher-order ap-

proximation. It turns out that for the given distribution we require at least a cubic interpolation with a lookup table containing $\mathcal{O}(10^3)$ entries. The reason for choosing this approximation lies in the Taylor expansion of $F(\theta)$, as specified in Equation (4.80),

$$F(\theta) = \frac{2}{\pi}\left(\frac{2}{3}\,\theta^3 - \frac{2}{15}\,\theta^5 + \frac{4}{315}\,\theta^7 + \mathcal{O}\left(\theta^9\right)\right). \tag{4.83}$$

Therefore, we need to look for a better way to solve this problem.

In this case we can use Newton's method with starting value of $\theta_0 = \pi x$. The required precision can be reached within 10 iterations. This is sufficient, efficient, and, most importantly, precise enough. Furthermore, we do not waste any memory for the lookup table.

The distribution for the magnitude r is chosen in such a way to give us at least another factor of r in the numerator. Together with the r^3 from the Jacobian we cancel divergences from the photon propagator immediately.

Another aspect that needs to be considered is that the constraints (or cutoffs) from Equation (4.72) are fulfilled. We have

$$\Omega_r(r) = \left(r\,\log(k_{\max}/k_{\min})\right)^{-1}, \tag{4.84}$$

which is basically a r^{-1} distribution within our boundaries.

We can generate variables according to this distribution by using the exponential function, i.e.,

$$r = k_{\min}\,\exp\left(-x\,\log(k_{\min}/k_{\max})\right), \qquad x \in [0,1]. \tag{4.85}$$

This way we have all required distributions to generate values for the k four-vector in spherical coordinates. A good distribution flattens the sampled space and eliminates pairs from the sampling, which would cancel anyway.

For higher-order diagrams it makes sense to use the Feynman-parametric representation of the integrands, see Equation (4.8). Instead of generating four-vectors k we generate a scalar z_i for each internal line. The z_i need to sum to unity, i.e., $\sum_i z_i = 1$. Furthermore, we require $z_i > 0$. A suitable distribution for z is the Dirichlet distribution. For n internal lines we have

$$p(z_1, \ldots, z_n, \alpha_1, \ldots, \alpha_n) = \frac{1}{B(\alpha)} \prod_{i=1}^{n} z_i^{\alpha_i - 1}, \tag{4.86}$$

where the α_i define the so-called concentration. The normalization is given by

$$B(\alpha) = \frac{\prod_{i=1}^{n} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{n} \alpha_i\right)}. \tag{4.87}$$

In our case we choose a symmetric Dirichlet distribution with $\alpha_i \equiv \alpha$ for all i. Hence we obtain

$$\Omega_z(z) = \frac{\Gamma(n\,\alpha)}{(\Gamma(\alpha))^n} \prod_{i=1}^{n} z_i^{\alpha-1}. \tag{4.88}$$

In the simplest scenario we pick the flat distribution using $\alpha = 1$. We can then generate the values z_i by using some exponential distribution $\exp(-y_i)$, which leads us to $z_i = y_i / \sum_i y_i$.

Generally, numbers that obey the Dirichlet distribution can be generated using the gamma distribution

$$p(x, \alpha, \beta) = \frac{x^{\alpha-1}\,\exp(-x/\beta)}{\Gamma(\alpha)\,\beta^{\alpha}}, \qquad \alpha > 0, \quad \beta > 0, \tag{4.89}$$

with β being set to 1. The parameter α corresponds to the concentration parameter of the Dirichlet distribution. Finally, we obtain the values z_i by using $z_i = x_i / \sum_i x_i$ as before.

4.3.2 Update Procedures

For the available update procedures we follow the same routines as outlined in Section 3.3. We implement the following updates, which form a set similar to the one used for the polaron problem discussed in Section 3.3:

add We choose one of the diagrams from the next-loop order.

remove We choose one of the diagrams from the previous-loop order.

swap We choose another diagram from the same-loop order.

mutate We change one of the internal loop momenta.

We will always fix the value of q^2/m^2, such that $q^2/m^2 = \text{const}$, even though it is possible to parameterize the simulation with a flexible value for q^2. This would allow us to gain information about the limit $q^2/m^2 \to 0$ by extrapolating the gathered data. Such an approach may be investigated in future works.

First, we look at the add procedure for going from zero-loop D_0 to the single one-loop diagram $D_1(k)$. The zero-loop is represented by an arbitrary constant, much like in the integration example from Section 3.2. We use

$$R_{0\to1} = \left|\frac{D_1(k)}{D_0}\right| \frac{1}{\Omega_k(k)} = \frac{|D_1(r,\varphi,\theta,\psi)\,r^3\,\sin^2(\theta)\,\sin(\psi)|}{|D_0|\,\Omega_r(r)\,\Omega_\varphi(\varphi)\,\Omega_\theta(\theta)\,\Omega_\psi(\psi)} \tag{4.90}$$

$$= 2\pi^2 \left|\frac{D_1(k)}{D_0}\right| r^4\,\log(k_{\max}/k_{\min}). \tag{4.91}$$

We may want to rewrite $D_1(k)$ to take spherical coordinates as inputs. Otherwise we have to calculate

$$k_1 = r\ \cos(\theta), \tag{4.92}$$
$$k_2 = r\ \sin(\theta)\ \cos(\psi), \tag{4.93}$$
$$k_3 = r\ \sin(\theta)\ \sin(\psi)\ \cos(\varphi), \tag{4.94}$$
$$k_4 = r\ \sin(\theta)\ \sin(\psi)\ \sin(\varphi), \tag{4.95}$$

to obtain the k four-vector to use with $D_1(k)$, where $k \equiv (k_1, k_2, k_3, k_4)$.

The remove procedure for going from the one-loop diagram to zero-loop is just the inverse of the add procedure. We find

$$R_{1\to 0} = \left|\frac{D_0}{D_1(k)}\right| \Omega_k(k) = \left|\frac{D_0}{D_1(k)}\right| \frac{1}{2\pi^2\, r^4\ \log(k_{\max}/k_{\min})}. \tag{4.96}$$

At this point it seems reasonable to define the diagram functions. The n-loop diagram function at topology ξ_n is given by applying the $\hat{R}$ operator on the corresponding diagram using the integrand of the vertex correction $\Gamma_\mu^{(\xi_n)}$, named $f_\mu^{(\xi_n)}(k_1, \ldots, k_n)$. We define

$$D_{\xi_n}(k_1, \ldots, k_n) \equiv \hat{R}_{\xi_n}\left(f_\mu^{(\xi_n)}(k_1, \ldots, k_n)\right), \tag{4.97}$$

with the projection operator $\hat{R}$ from Section 4.2.4. The operator is required for the renormalization of the diagram associated with ξ_n. Integration over the introduced function D_{ξ_n} using all parameters yields the form factor,

$$F_2^{(\xi_n)}(q^2) = \int d^4k_1 \ldots d^4k_n\, \hat{R}_{\xi_n}\left(f_\mu^{(\xi_n)}(k_1, \ldots, k_n)\right). \tag{4.98}$$

In practice, values of the function $D_{\xi_n}(k_1, \ldots, k_n)$ need to be constructed from the K_i shown in Equation (4.23), which have been obtained using the projectors defined in Equation (4.27) to Equation (4.37).

Removing a diagram implies taking the inverse ratio of the transition probabilities for adding a diagram. We have

$$R_{1\to 0} = \left|\frac{D_0}{D_1(k)}\right| \Omega_k(k) = \left|\frac{D_0}{D_1(k)}\right| \frac{\pi^{-2}}{2r^4\ \log(k_{\max}/k_{\min})}. \tag{4.99}$$

For the generalization from the first-order corrections to arbitrary diagrams, we need to include some more distributions and update procedures. For instance, when dealing with more than one set of variables, it makes sense to have an update that just mutates an arbitrary set.

It is possible to change the topology from D_{ξ_n} to $D_{\xi'_n}$ using the ordinary transition probabilities introduced in Section 3.1. The selection probability for a particular diagram D_{ξ_n} has to be adjusted to match the symmetry factor S_{ξ_n} of the respective diagram. The symmetry factors will be deduced in our diagram generation scheme, which is presented in the next Section 4.4.

For the swap update we have

$$R_{\xi_n \to \xi'_n} = \left| \frac{D_{\xi'_n}(k_1, \ldots, k_n)}{D_{\xi_n}(k_1, \ldots, k_n)} \right| \frac{\Omega_{\xi_n}(\xi_n)}{\Omega_{\xi_n}(\xi'_n)}, \tag{4.100}$$

where we use a uniform distribution for the selection probability of a particular diagram. We have

$$\Omega_{\xi_n}(\xi) = \frac{S_\xi}{\sum_{\xi_n} S_{\xi_n}}, \tag{4.101}$$

with the symmetry factor S_ξ for a given topology ξ.

A little bit less delicate is the mutate update procedure. This update is supposed to change the value of any currently available variable. It implies the uniform selection of a variable to change.

The ratio of transition probabilities for the mutate update looks like

$$R_{k_i \to k'_i} = \left| \frac{D_{\xi_n}(k_1, \ldots, k_{i-1}, k'_i, k_{i+1}, \ldots, k_n)}{D_{\xi_n}(k_1, \ldots, k_{i-1}, k_i, k_{i+1}, \ldots, k_n)} \right| \frac{\Omega_k(k_i)}{\Omega_k(k'_i)\Omega_i(i')}, \tag{4.102}$$

where we use the composed distribution from Equation (4.74) and a uniform distribution $\Omega_i(i)$ that is simply $1/n$. We need this to accommodate for our random selection of a variable to change. Since variables are not inserted on random occasion, we only use the distribution in the denominator.

4.4 Diagram Generation

Before we apply the DiagMC method to our problem we need to have a robust scheme for diagram generation. The formulation of QED in terms of diagrams was first described by Feynman [52]. The series to approximate a_e(QED) as specified in Equation (4.16) can be written as

$$a_e(\text{QED}) = c_1 \left(\frac{\alpha}{\pi}\right) + c_2 \left(\frac{\alpha}{\pi}\right)^2 + c_3 \left(\frac{\alpha}{\pi}\right)^3 + c_4 \left(\frac{\alpha}{\pi}\right)^4 + \ldots, \tag{4.103}$$

where the one-loop result c_1 has been derived in Section 4.1 using Equation (4.13). The two-loop result c_2 can be expressed in analytical form as well. Even though c_3 can be calculated in analytical form using difference equations [84], the result is already much too long for printing or human receptivity. The four-loop coefficient

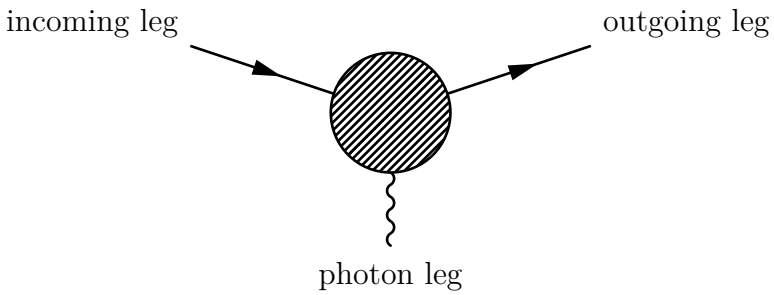


Figure 4.7: The generalized process for the anomalous magnetic moment.

c_4 is only known numerically [11].

The coefficients have been determined [75] to be

$$c_1 = \frac{1}{2} \qquad \text{(1 diagram, Schwinger (1948))}, \tag{4.104}$$

$$c_2 = -0.328478965579 \qquad \text{(7 diagrams, Sommerfield (1958))}, \tag{4.105}$$

$$c_3 = 1.181241456 \qquad \text{(72 diagrams, Laporta (1996))}, \tag{4.106}$$

$$c_4 = -1.9144(35) \qquad \text{(891 diagrams, Kinoshita (2007))}. \tag{4.107}$$

The number of diagrams gives the number of topological distinct diagrams. It is necessary to evaluate a subset of diagrams with different lepton masses in closed fermion loops to obtain these coefficients.

Our simulation scheme needs to use the Feynman rules derived for QED processes to generate all the possible diagrams for the current loop-order. We have two options for implementing such a requirement. Either we follow a method similar to the one used in Section 3.3, where we connect propagators to vertices during the simulation, or we generate all possible diagrams beforehand.

Since the computational cost is much higher [67] than in the polaron example we choose to generate the diagrams. Software applications to help us solving this task are available [97, 153], however, we have to integrate the custom subtraction scheme, which motivate us to provide our own solution. Furthermore, we want to create a well-performing C++ code that follows our conventions.

The algorithm we describe in this section is based on permutation and determines all possible diagrams for a given loop order. A positive side effect of using permutations is the ability to determine the symmetry factors of the generated diagrams. Obtaining these factors otherwise has been proven difficult [110].

In a permutation scheme we can count the number of occurrences for each diagram. The diagrams itself are compared by using a suitable hash function. Furthermore, we have to ensure that every diagram is valid, i.e., that it represents indeed a one particle irreducible correlation function that can be used in the

context of the anomalous magnetic moment.

The diagram generation for the anomalous magnetic moment starts by introducing a special node, called the *root vertex*. The root vertex has the photon leg and, like any other QED vertex, two lepton legs. In our scheme we will never touch the photon line of the root vertex. Thus, it will never be connected to any other node.

For the n-loop diagrams $\mathcal{G}_n \subset \mathcal{F}$ we need to consider $2n$ additional vertices. This gives us a total of $2n+1$ vertices, $3n$ propagators, and 2 legs that need to be handled. The diagram in Figure 4.7 shows the generalized process with the external legs and the root vertex. The blob represents the variable space for inserting and connecting the other vertices.

Following the conventions from [31] we label the existing vertices by introducing numbers, $v \in \{1, \ldots, 2n+1\}$. Additionally, we have two "end nodes" for the incoming ($v = 0$) and outgoing ($v = 2n+2$) legs. It is important to distinguish between the terms node and vertex here. While every vertex is a node, not every node is a vertex. A vertex refers to a legitimate point of interaction in a QED diagram. A node refers to a point of interaction in a graph. A QED diagram is a graph with special rules.

For the lepton propagators we use a mapping $w(v)$, where v and $w(v)$ denote the two connected nodes. We can define the inverse mapping $\bar{w}(v)$, which returns the corresponding anti-lepton propagator, i.e., the lepton propagator with reversed nodes. We have

$$v \in [0, 2n+1] \mapsto w(v) \in [1, 2n+2], \tag{4.108}$$

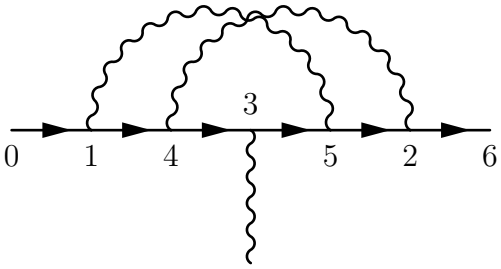
$$v \in [1, 2n+2] \mapsto \bar{w}(v) \in [0, 2n+1]. \tag{4.109}$$

Thus, $\bar{w}$ maps the image of w back to $v \in [0, 2n+1]$. The external legs are represented by 0 and $2n+2$.

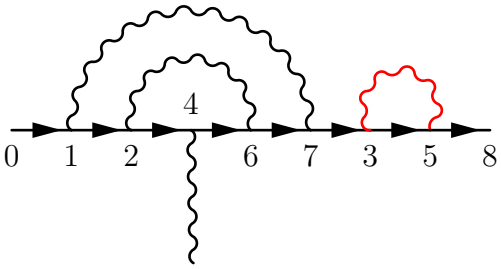
In our scheme we start by connecting the vertices v with $2n+2-v$ via a photon propagator. An example can be seen in Figure 4.8a. Here we connect the vertices $\{1, 5\}$ and $\{2, 4\}$. As the sum of the labels of these vertices is always $2(n+1)$ we have a very quick way to check for photon pairings. The root vertex has $v = n+1$. In Table 4.1 we identify the mapping function and the inverse assignment for the mentioned example.

To obtain all combinations it is sufficient to permute the mapping $v \mapsto w(v)$. This way we have $(2n+2)!$ possibilities, far more than diagrams. We can directly exclude some invalid sequences, e.g., the identity function or any other sequence where $v = w(v)$ for any v.

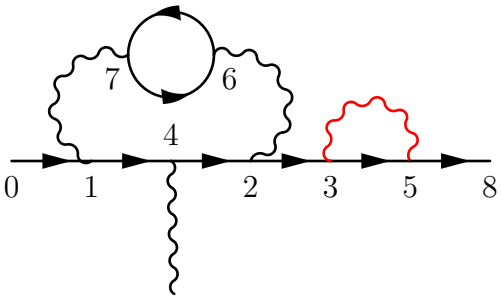
Finally, we impose some restrictions on our selections. We forbid $w(0) = 2n+2$.



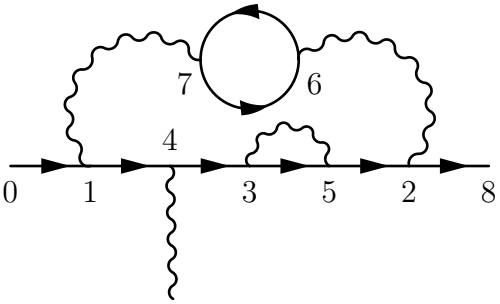
(a) valid (two-loop)



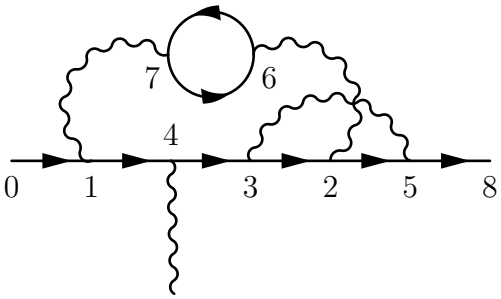
(b) invalid (three-loop)



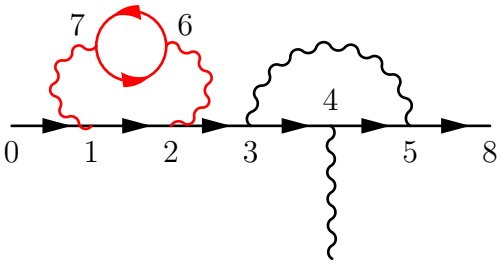
(c) invalid (three-loop)



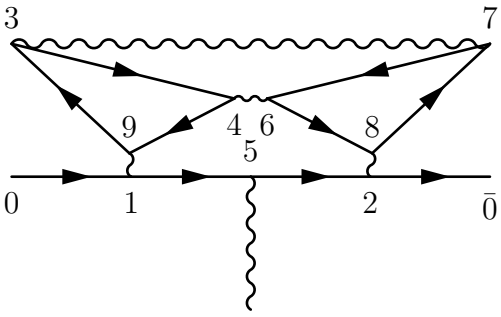
(d) valid (three-loop)



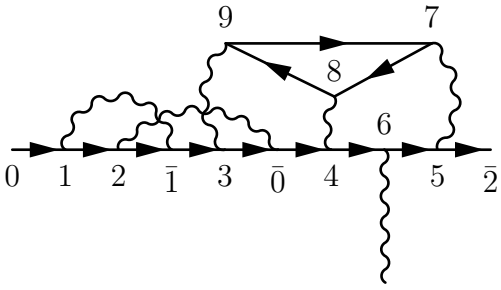
(e) valid (three-loop)



(f) invalid (three-loop)



(g) valid (four-loop)



(h) valid (five-loop)

Figure 4.8: Examples for generated diagrams with the corresponding vertex labels. The propagators marked with the red color render the corresponding diagram invalid for evaluation in the context of the anomalous magnetic moment.

v	$w(v)$	$\bar{w}(v)$
0	1	—
1	4	0
2	6	5
3	5	4
4	3	1
5	2	3
6	—	2

Table 4.1: Mapping of the vertices shown in Figure 4.8a.

Also $w(0) = n + 1$ and $w(n + 1) = 2n + 2$ do not represent legal diagrams for computing the anomalous magnetic moment. Besides these trivial cases we leave the rest to a verification algorithm. The equivalence of two diagrams is determined by applying a suitable hash function. The permutation part is shown in Algorithm 2.

Algorithm 2 Diagram Generation

```
 1: function PERMUTE(vertices W, vertices V)          ▷ initially W = ∅
 2:     if |V| > 0 then
 3:         for each v ∈ V with v ≠ |V| do
 4:             PERMUTE(W ∪ {v}, V \ {v})
 5:         end for
 6:     else                                          ▷ at this point V = ∅
 7:         h ← COMPUTEHASHCODE(W)
 8:         if hashcode h not added yet then
 9:             add h to known hashcodes
10:             if ISVALID(W) then
11:                 add diagram W
12:             end if
13:         end if
14:     end if
15: end function
```

To explain the algorithm in more detail it is useful to introduce the cycle notation for diagram permutations σ. We use

$$\sigma = \begin{pmatrix} 0 & 1 & \dots & 2n+1 \\ w(0) & w(1) & \dots & w(2n+1) \end{pmatrix} \equiv (w(0)\, w(w(0))\, w(w(w(0)))\, \dots). \quad (4.110)$$

We can shorten this notation, by realizing that the node $v = 2n + 2$ has always to be the last one, unless there are fermion loops. If we take, e.g., the diagram with $\sigma = (134625)$ we can express it as $\sigma = (134)(25)$.

If we follow the vertices of Figure 4.8a from the incoming leg, we have $\sigma = (14352)$. The root vertex has the number 3. We can now identify the rules for validating diagrams. Vertices connected by photon propagators need to be found and checked first. In the given case there is no irreducible subdiagram, as every connection passes at least another vertex. Both photon propagators cross the root vertex. As a minimum requirement for all valid $g-2$ diagrams we need at least one photon propagator that crosses the root vertex.

The three-loop diagram shown in Figure 4.8b illustrates an invalid diagram. Here we have $\sigma = (1246735)$. We can immediately identify the subdiagram consisting of the vertices $\{3, 5\}$. The given diagram is thus composed of a two-loop diagram with a simple second-order diagram.

We now consider another three-loop diagram, (14235)(76), as illustrated in Figure 4.8c. Even though we have a fermion loop connected to both sides, we have practically the same scenario as beforehand. Again, we can identify a subdiagram that is independent from the three-loop diagram. As a rule we can identify that once the sum of an even number of vertex indices on the left or right is $2n+2$, the diagram is invalid.

A different case is shown in Figure 4.8d. This diagram is expressed as

$$\sigma = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 4 & 8 & 5 & 3 & 2 & 7 & 6 \end{pmatrix} = (14352)(76). \tag{4.111}$$

We can identify the subdiagram (35), however, it is not placed on the left or right side of the main cycle. The rest can be validated like (142)(76), which is equivalent to the three-loop diagram (132)(54). The diagram shown in Figure 4.8e is valid as well. A validation of the expression (14325)(76) yields that no independent subdiagrams can be found. Another negative example is given in Figure 4.8f. A quick check of (12345)(76) shows an independent subdiagram on the left side.

There are two more cases that are worth mentioning. The first one is shown in Figure 4.8g. Here we see a diagram with two fermion loops denoted by

$$\sigma = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 5 & 10 & 9 & 3 & 2 & 8 & 6 & 7 & 4 \end{pmatrix} = (152)(943)(687). \tag{4.112}$$

This is a legal four-loop diagram for the perturbative expansion of the anomalous magnetic moment. It contains corrections to the original loop itself, splitting it into two loops with three vertices each, which are connected. This diagram is naturally generated just by permuting $w(v)$. The validation scheme is outlined in Algorithm 3.

The five-loop diagram shown in Figure 4.8h can be generated as well. We

Algorithm 3 Diagram Validation

```
1: function IsValid(vertices D)
2:     M ← {v ∈ D : v ∉ loop}                    ▷ exclude v from fermion loops
3:     C ← {v ∈ D : v ∉ M}
4:     σ ← Σ_{v∈C} v
5:     r ← n + 1                                  ▷ n is the loop order
6:     if r ∈ M and σ = 0 ∨ σ ≠ r|C| then
7:         k ← index of r ∈ M
8:         M_L ← {M_1, M_2, ..., M_{k−1}}
9:         M_R ← {M_{|M|}, M_{|M|−1}, ..., M_{k+1}}   ▷ |M| is the cardinality of M
10:        i ←IsConnected(M_L, C)
11:        o ←IsConnected(M_R, C)
12:        if i is true and o is true then
13:            return true
14:        end if
15:    end if
16:    return false
17: end function
```

have 13 vertices with $v = 6$ being the root vertex. We can expand our notation of Equation (4.110) to contain numbers $\bar{n} = 10 + n$ for denoting the diagram as $(12\bar{1}3\bar{0}465)(978)$. The diagram is legal, since the photon propagator $\left\{1, \bar{1}\right\}$ passes the start of the propagator $\left\{2, \bar{0}\right\}$, which passes vertex 3. This vertex is connected to 9, which has a connection to vertex 5 via the photon propagator $\{5, 7\}$ and the fermion loop (978). As they are all connected, we have a legal diagram for calculating five-loop corrections to the anomalous magnetic moment. The connection is verified in the sub-algorithm shown in Algorithm 4.

After we generated all possible diagrams for a certain loop-order n we need to extract the C++ code for our application. Going from right to left we insert algebraic elements for their corresponding diagrammatic representations. We follow the standard Feynman rules for QED in Euclidean space to apply the mapping

$$\text{Vertex} \mapsto ie\gamma_\mu, \tag{4.113}$$

$$\text{Photon} \mapsto i\delta_{\mu\nu}/k^2, \tag{4.114}$$

$$\text{Electron} \mapsto -(\not{p} + im)/(p^2 + m^2), \tag{4.115}$$

$$\text{Integral} \mapsto i(2\pi)^4, \tag{4.116}$$

$$\text{Fermion-loop} \mapsto (-1). \tag{4.117}$$

The code will be optimized further during compilation and uses the data structures and conventions defined in our application. Therefore, even though our algorithm scales worse than exponential we only have to pay the cost of diagram creation

Algorithm 4 Diagram Connectivity

```
1: function IsConnected(vertices M, vertices C)
2:     for each m ∈ M do
3:         M ← M\{m}
4:         p ← 2n + 2 − m                          ▷ n is the loop order
5:         if p ∉ M and p ∉ C then
6:             return true
7:         else if p ∈ M then
8:             k ← index of p ∈ M
9:             M ← {M_1, ..., M_{k−1}}
10:        else                                    ▷ leads to fermion loop
11:            C* ← {v ≠ p ∈ C : connected via ferm. prop. to p}
12:            C ← C(\C* ∪ {p})
13:            M ← C* ∪ M
14:        end if
15:    end for
16:    return false
17: end function
```

once. The generated diagrams are fixed and can be used with full optimizations. Consequently, the runtime performance is much better than without prior generation and compilation of diagram functions. Finally, we can use some trivial analytical tricks and convert higher-order diagrams to their Feynman-parametric representation.

4.5 Results

Before we discuss the results for a full DiagMC simulation using multiple loops, we should have a look at one-loop simulations. In Figure 4.9 we compare the result of different simulations with a varying number of configurations to the analytic result of $F_2^{(1)}(0)$. For q^2/m^2 we choose -1×10^{-4} at $\alpha = 1$. We see that even for a small number of configurations, e.g., 1×10^5, we obtain the exact result within a reasonable small error. The error determination seems to be alright, e.g., for the smallest number of configurations we match the exact result within two standard deviations. For the larger number of configurations the error bars become smaller than the symbols. Hence even for values of $|q^2|$ that do not satisfy $|q^2| \ll m^2$ we achieve the analytic result with quite some accuracy.

The observation from Figure 4.9 is reason enough to be confident that we can successfully estimate the magnetic form factor at $q^2 = 0$ by extrapolating the result of runs with small $|q^2/m^2|$. As a first order approximation, we will quote a very small non-zero value of q^2/m^2 as approximate $q^2 = 0$ result throughout this feasibility study instead of performing an extrapolation to $q^2 = 0$. It was checked,

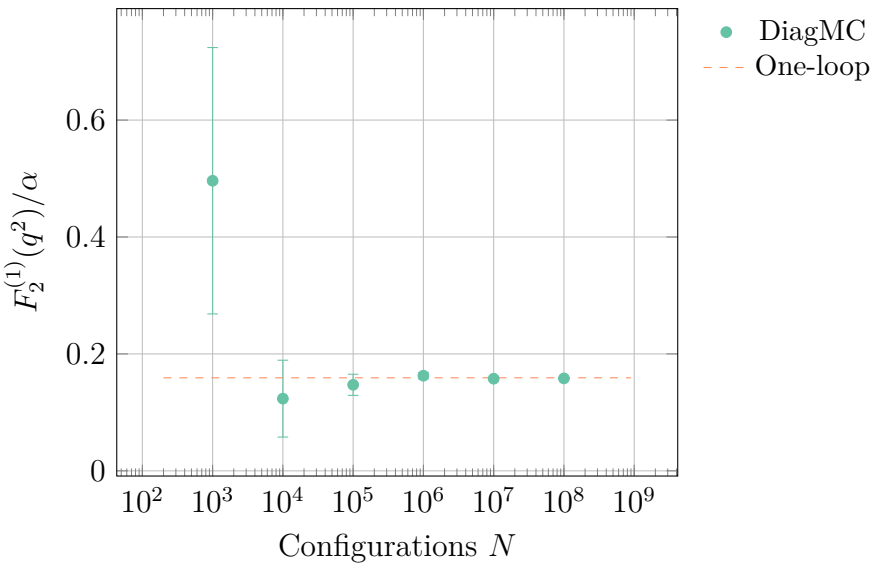


Figure 4.9: Dependency on the number of configurations for a simulation with $\alpha = 1$ at $q^2/m^2 = -0.0001$. The DiagMC runs have been performed up to one-loop order with two kinds of updates (add, remove). We choose $D_0 = 1$. The dashed line is the exact result for the one-loop diagram.

that the dependence of the results on q^2/m^2 is smaller than the statistical error from the finite number of configurations.

For higher-order simulations with $\alpha \approx 1/137$ we expect the autocorrelation time to be quite large, as most updates will be rejected. Indeed for a simulation up to one-loop order we obtain

$$\tau_{\text{int}} = 267(39). \tag{4.118}$$

One way to lower the autocorrelation time is to use a better value for the artificial diagram D_0. At the end of this section we sketch a far better way, which omits the artificial diagram altogether.

There is a debate whether $\alpha(q^2)$ must be considered in our scenario and if we could use some α that fulfills $1/137 \leq \alpha \leq 1$ for the full simulation. While a constant α seems like a good approximation in this scenario[2], the latter is indeed a good question. As α is part of the transition amplitude we would immediately gain a much higher sampling rate. Nevertheless, the higher sampling rate for higher-order diagrams directly results in fewer samples from lower-order diagrams. We already know that the contribution from these lower-order functions is more significant (by a factor of α^{-1} per loop-order). Therefore, it makes sense to use α in the transition amplitude, thus sampling more significant diagrams more often.

The performed one-loop simulation is similar to the simple example discussed in Section 3.2, even though the details of computation are much more complex.

[2] Even if we use $\alpha(q^2)$ the value would be some (other) constant for a given q^2.

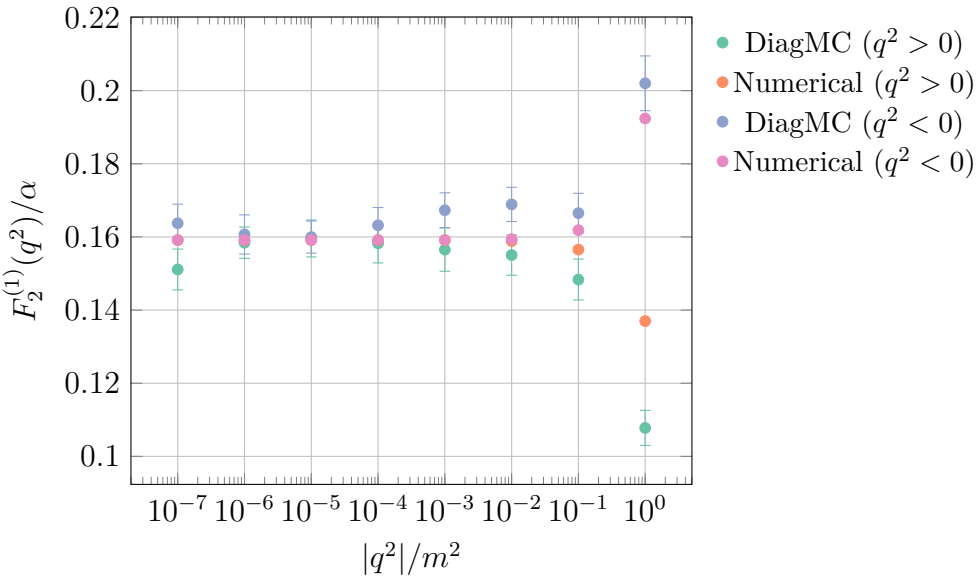


Figure 4.10: Comparison of a simulation with $N = 10^6$ configurations to the numerical evaluation of Equation (4.13) with $\alpha = 1$. Excellent agreement in the valid region, where $|q^2| \ll m^2$. The error bars for the numerical evaluation are smaller than the symbols.

Most importantly, we only use two complementary updates, add and remove. Simulations that include higher orders need to contain other updates, such as swap or a change of variables, as well.

It is interesting to compare our simulation with the analytic approach in more detail. Starting with Equation (4.13) we can integrate over the Feynman parameter z_1 to obtain

$$F_2^{(1)}(q^2) = \frac{\alpha}{2\pi} \int_0^1 dz_3 \int_0^{1-z_3} dz_2 \, \frac{2m^2 \, z_3 \, (1 - z_3)}{m^2 \, (1 - z_3)^2 - q^2 \, (1 - z_2 - z_3) \, z_2}. \tag{4.119}$$

For $q^2/m^2 = 0$ this yields the famous result of $\alpha/2\pi$. However, how does it compare to our calculations for arbitrary $q^2/m^2 \neq 0$? Are we able to produce the same result as with Equation (4.119)?

In Figure 4.10 we observe a deviation of the results for $|q^2| \to m^2$, especially with $q^2 > 0$. For small values of $|q^2|/m^2$ we do not see this discrepancy. Instead, we observe an excellent agreement within our statistical error estimates. The question now is if this disagreement influences our simulation.

Assuming that all the analytic equations for the DiagMC simulation are correct, we can blame some of the conditions that have been mixed into the formation of Equation (4.13). The most probable reason that may spoil the desired equivalence is that q is spacelike, i.e., $q^2 < 0$. In this case the effective mass is definitely positive. For timelike scattering energy transmission the effective mass can be negative, which renders the whole equation ill-defined. As a consequence, we

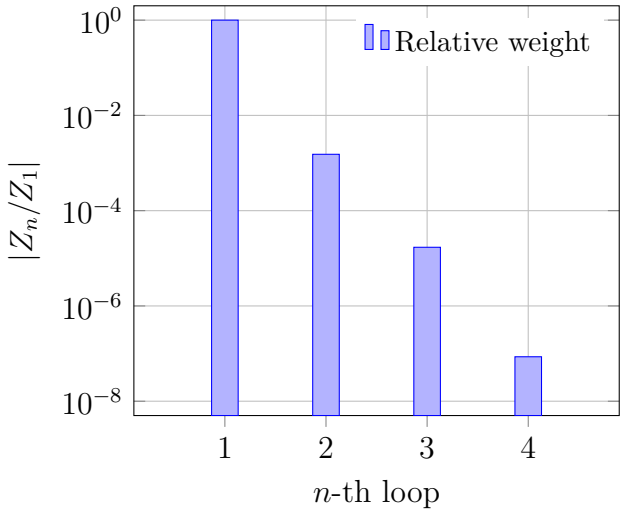


Figure 4.11: Sampled number of the diagrams from different loop orders for $N = 10^{10}$ configurations with $q^2/m^2 = -10^{-6}$. The numbers have been normalized to the one-loop diagram.

should restrict ourselves to evaluations with (small) spacelike momentum q, as results seem to agree in the corresponding range.

Looking at a simulation up to arbitrary[3] loop order we can observe some of the advantages of the stochastic interpretation. In Figure 4.11 we see the frequency of the loop orders normalized to the number of one-loop diagrams. The given plot was created by including the artificial diagram $D_0 \equiv \alpha$. The sampling included twice as many zeroth-order than one-loop diagrams. The acceptance rate was about 15 %. As with the other evaluations in this section the projection from Equation (4.26) was used. The calculation of the weights follows Equation (3.14).

In the simulation we generated $N = 10^{10}$ configurations with $q^2/m^2 = -10^{-6}$. With the obtained data we are able to estimate

$$a = 0.001159681(14). \tag{4.120}$$

This result is close to the currently known value given in Equation (4.1). Most importantly, we can deduce information about the sampled orders. In principle we are free to use the information from the sampling process for estimating the value of any (sampled) order. Of course, we need to have sufficient samples from the order of interest to provide a decent level of accuracy. This is also possible with different topologies.

With our sampling scheme we can obtain information about the coefficients c_n

[3] We only generated code for evaluating diagrams up to five loops as we estimated that we will not be able to sample (enough) diagrams to go to or beyond five-loops.

Order n	Samples N_n	Weight Z_n
0	6.228×10^9	6.228×10^9
1	3.764×10^9	9.913×10^8
2	1.077×10^7	-1.513×10^6
3	2.548×10^5	1.690×10^4
4	2216	-85

Table 4.2: Results for sampling the diagrams from different loop orders for $N = 10^{10}$ configurations with $q^2/m^2 = -10^{-6}$. The artificial diagram has been set to $D_0 \equiv \alpha$.

as outlined in Equation (4.103). We can estimate the coefficient via

$$c_n \leftarrow c_1 \left(\frac{\alpha}{\pi}\right)^{1-n} \frac{Z_n}{Z_1}, \tag{4.121}$$

with the sum over all sampled signs for the n-order Z_n and first-order Z_1 diagrams. For c_1 we use the exact value, $c_1 = 1/2$. If we look at the estimation of the two-loop coefficient, we see that it is in a fairly good agreement with the literature, e.g., [87]. We have

$$c_2 = -0.3285(23). \tag{4.122}$$

Other orders contain less samples and therefore cannot yield estimates with the same agreement. As an example, we can estimate the three-loop coefficient c_3 to be

$$c_3 = 1.158(62). \tag{4.123}$$

This is still quite close to the value of 1.181 [84]. Even with the $\mathcal{O}(10^3)$ samples for the four-loop diagrams from the evaluation shown in Figure 4.11 we may still find

$$c_4 = -3.4(2.4), \tag{4.124}$$

which is within the range of the known value, -1.91 [11]. In Table 4.2 we see the weights of all sampled orders for this simulation. No five-loop diagrams have been sampled in this simulation.

Generating 10^{10} configurations takes about a week on an ordinary computer. Using powerful and dedicated resources, such as the custom-made supercomputer QPACE 2, which is described in Part II, we can obtain 10^4 more statistics in the same time. As the method works in the thermodynamic limit, we are mostly bound by statistics. Future studies may use the created computational resources for more precise estimations.

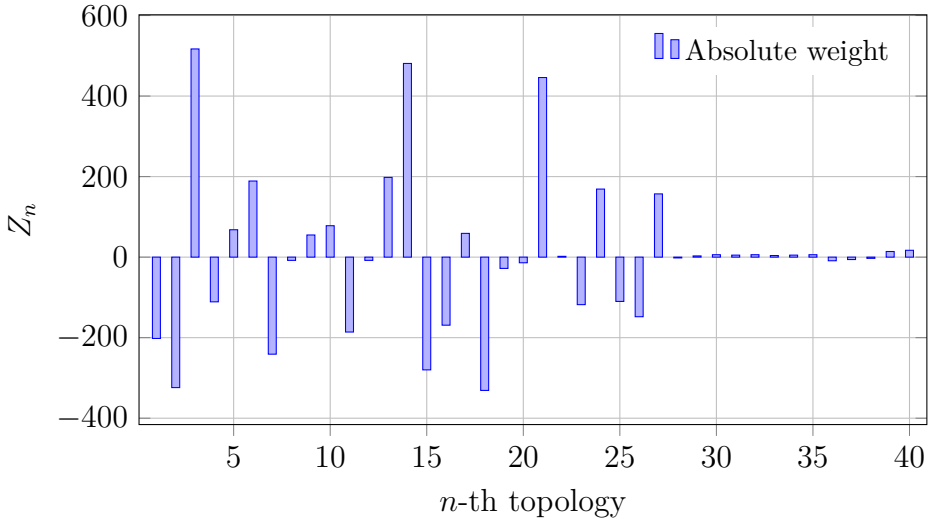


Figure 4.12: Sampled number of the diagrams from different three-loop topologies for $N = 10^{10}$ configurations with $q^2/m^2 = -10^{-6}$.

Alternatively, we can start in an already estimated order to evaluate only the next order, e.g., going from four-loop to five-loop. This way a reliable estimate can be achieved even for higher-orders. Nevertheless, since the number of diagrams is factorial growing, it is questionable if the desired cancellation still occurs in the anticipated manner. For the full series the answer is easy, as higher-order diagrams are weighted much less than lower-order diagrams. Therefore, an exact cancellation is not really required.

Going back to our original study we see that the same evaluation can be applied to diagrams within a given order. This way we obtain information which diagrams are more important than others. The more important diagrams could then be studied more carefully without having to care about the negligible or potentially less important diagrams.

For such a study we change our simulation a bit. We exclude the artificial zeroth-order diagram. Additionally, we do not need the add and remove update procedures. Initially, we generate values for all required parameters and we choose a topology randomly by using the already obtained symmetry factors. During the simulation we can only alter the topology and the set of parameters.

In Figure 4.12 we see the histogram for sampled weights during a simulation involving three-loop diagrams. We use the 40 topologically unique diagrams, which potentially contain their mirror diagrams. The diagrams 1 to 28 do not contain a closed fermion loop. Overall, the whole set of 72 diagrams is involved in this study.

One thing that can be observed is that the diagrams with closed fermion loops are quite suppressed compared to the ones with no closed fermion loop. Another

Topology n	Weight Z_n	Rel. Weight	Comparison
3	514	1.0	1.0
14	479	0.93(6)	0.84
21	444	0.86(6)	0.81
18	−331	−0.64(3)	−0.64
15	−279	−0.54(4)	−0.60
2	−320	−0.62(8)	−0.52
1	−202	−0.39(7)	−0.42
7	−237	−0.46(4)	−0.41
6	188	0.37(5)	0.39
11	−186	−0.36(6)	−0.38

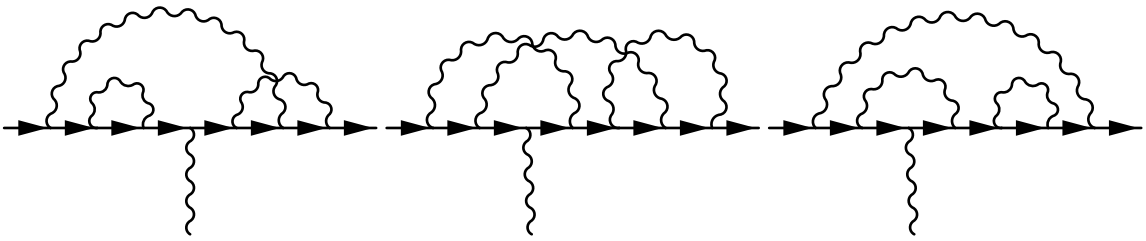
Table 4.3: Results for sampling the diagrams from different three-loop topologies for $N = 10^{10}$ configurations with $q^2/m^2 = -10^{-6}$. The values have been normalized to the largest value (diagram 3). The comparison values have been taken from [87]. For brevity only the ten most significant diagrams are shown. The less significant diagrams tend to have a larger error.

thing to note is that the sign of the involved diagrams is mixed. Of course, this was quite expected, but it illustrates the point, that a variety of different diagrams needs to be sampled to be able to estimate the weight of a diagram order correctly. Similarly, a variety of parameters needs to be sampled to estimate the weight of a diagram topology well enough.

The estimates for the different diagrams can be compared to some reference calculations, e.g., in [87]. Already with a manageable number of configurations and a runtime of far less than a day, we can obtain results that align very well to exact results. Since the diagram space grows factorial, we do not expect to use this method for exact calculation of arbitrary order. The region, where it actually shines is to give us an indicator of the dominant topologies quite fast. This behavior can be observed in all of our studies.

In order to estimate the value of a diagram in the previous study, we need to have at least the exact value of one of the involved diagrams. Otherwise we are not able to use our estimation as outlined in Section 3.1 by using Equation (3.18). The values in Table 4.3 have been obtained by normalization to the largest value (diagram 3). For comparison we also normalized the values found in [87] to their largest value. The errors on the normalized values have been evaluated by applying Jackknife on the binned samples.

In Figure 4.13 we see the diagrammatic representations of the three diagrams that have been estimated to contribute most to the three-loop corrections. The



(a) $n = 3$ (b) $n = 14$ (c) $n = 21$

Figure 4.13: The estimated three most significant diagrams (including mirrors) to the three-loop correction of the anomalous magnetic moment of the electron.

estimation included mirror images. Hence each of the diagrams includes a mirror diagram that is obtained by time-reversal.

Finally, we may want to increase our sampling efficiency by lowering the autocorrelation time. One possibility would be to adjust the value of the artificial diagram D_0 to be closer to $D_1(k)$. However, a constant is a suboptimal approximation to a function such as $D_0(k)$. A much better solution is to avoid the artificial diagram altogether. This is possible, since we know already the value of $D_1(k)$. Furthermore, we have studied many ways to compute partial values of the integral on the fly.

It is possible to use the same ansatz as chosen for simulating the polaron model in Section 3.3. This way we obtain the full integrand up to arbitrary orders. The estimated data can then be used in further evaluations that go beyond the value of the integral. Again, the polaron problem already illustrated the basic steps for such evaluations.

5 Conclusions

Observables in quantum field theories may be expressed as infinite sums of Feynman diagrams. During the last decade it has been realized that stochastic methods are very useful to estimate the value of the entire diagrammatic series. This requires a diagram-generating procedure, which is ergodic, such that all diagrams contributing to the series must be generated. Furthermore, it is necessary that each diagram is generated with a probability proportional to its weight in the series to be studied.

In this work we presented the method of Feynman diagram sampling using DiagMC. We applied the method to several problems in QFT. This presentation included the application to a simple model in condensed matter physics and the calculation of the anomalous magnetic moment of the electron. Along the way we developed efficient techniques for the generation and optimized evaluation of diagrams. We gained general insights relevant for its application for other QFTs like QCD.

From a computational perspective DiagMC can be easily parallelized and does scale very well. This lets us utilize additional processing powers, such as GPUs [133]. The ability to run simulations in the thermodynamic limit right away is definitely appealing. In this thesis we presented a variety of problems that can be solved using Feynman diagram sampling.

The success of the DiagMC method for a range of applications is, to large extent, due to small error bars we have for the sign-alternating sums of higher-

order diagrams. In general, it is expected that the computational complexity of getting small error bars for sign-alternating sums in the limit of large N is exponential (or even factorial) in N since it usually scales with the configuration space volume.

The method alone will not be a replacement for studies based on the lattice. Nevertheless, it can be used as a useful extension or possible replacement for related methods, such as the numerical stochastic perturbation theory [91]. As outlined we may need to consider a combination of analytic diagrammatic tricks, e.g., skeleton diagrams, as well as general resummation techniques to potentially estimate quantities correctly. This introduces a model-dependency that is not comparable to methods based on first principles, such as LQCD. Furthermore, regularization and cancellation need to be investigated and well understood.

In non-Abelian gauge theories Schwinger-Dyson equations can be written in terms of gauge invariant quantities. A formulation of Schwinger-Dyson equations known as Migdal-Makeenko loop equations [92] is using gauge invariant Wilson loops. The set of diagrams $\mathcal{F}$ could then contain closed sequences of links on the lattice. However, one of the main issues with this ansatz is the stochastic interpretation to gain physical insights.

Generally, the diagrammatic methods seem to be good fits and viable companions for certain problems, especially in the area of solid state physics. In particle physics no new insights have been obtained by using them. This work presents a first approach to apply DiagMC to a physically relevant QFT, not a toy model or problem originated from condensed matter physics. We showed that using DiagMC we could retrieve the known contributions to the anomalous magnetic moment of the electron in a very efficient manner.

As of today, only simple theories with known results or simplified models have been studied, e.g., ϕ^4 theory [34] discussed in Appendix E.3. Interestingly, the sign problem of some models, such as the $O(N)$ sigma-model in the large-N limit [25], becomes milder with DiagMC as the continuum limit is approached. This gives us some hope for further applications of DiagMC algorithms to asymptotically free field theories.

In the future, the constructed method could be used to include diagrams besides the ones from QED. Additionally, further approaches, such as mixing a topology dependent selection scheme with a pregenerated code may be investigated. Finally, we have to examine is if we can also obtain these results using Schwinger-Dyson equations.

Part II

The QPACE 2 Supercomputer

6

Introduction to Supercomputing

The term *supercomputer* was coined by the CDC 6600 mainframe computer. The CDC 6000 was introduced in the 1960s. Its outstanding ability was that it outran all other computers at its time by a full magnitude. Since then a supercomputer is a special computing device that is at the front-line of contemporary processing capacity. This includes the speed of calculations, memory and networking capability in terms of latency and bandwidth.

The history of supercomputers has seen quite a few changes in the fundamental design of such machines. In the 1970s it was particularly the instruction set and processor design that defined a supercomputer. Twenty years later the industry had made a dramatic shift to thousands of processors, which would not have been possible without improved networking hardware and much lower prices for CPUs.

Today we are going towards exaflops, i.e., 1×10^{18} flop per second, where a floating point operation (flop) is usually referring to single precision arithmetic. Such a number cannot be achieved by scaling the hardware alone. Fundamental design changes are required again. This has been realized in the previous decade, when the classic version of Moore's law, stating that the chip performance will double every 18 months [77], came to an end.

Nowadays Moore's law is reduced to state that the number of transistors per area will double every year. The frequency of computing devices is not increased anymore. In the end the additional space due to smaller and more efficient production processes is used for additional processor cores. The exponential growth

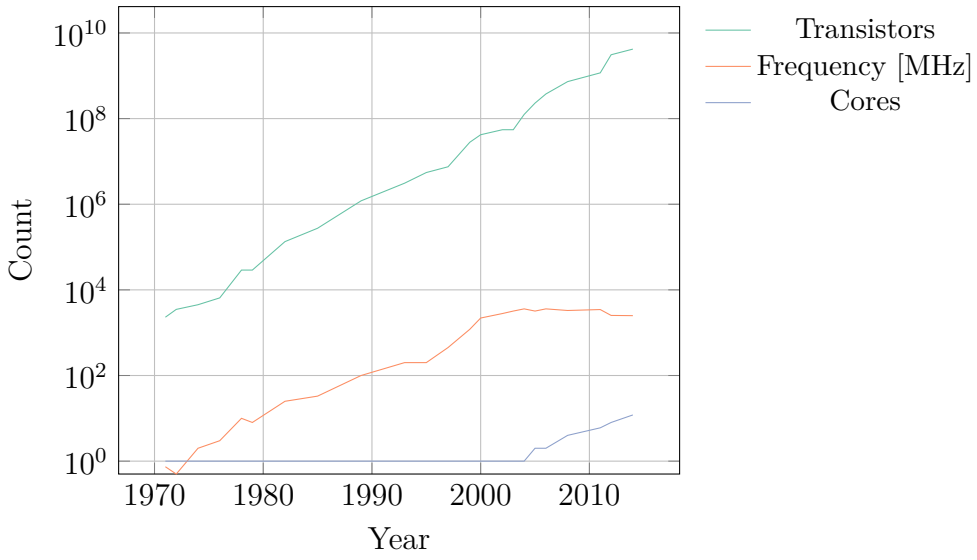


Figure 6.1: Number of transistors, cores, and the standard frequency in a single processor. It starts in the year 1971 with the introduction of the Intel 4004 processor. The data is taken from the microprocessor chronology [30].

of transistors per processor is illustrated in Figure 6.1.

The end of the era of increasing frequencies is confirmed by the data illustrated in Figure 6.1. As a major consequence we see the beginning of a new era: Multi-core processors with highly efficient cores in terms of power consumption. The tendency for more efficient cores, however, brings us back to a development that started in the 1970s. Building more efficient cores is possible by including more advanced instructions, i.e., single instructions that do the work of multiple instructions in a single cycle.

The instrument of measuring the innovation and developments in the supercomputing industry is the Top 500 list [139]. It contains a ranking of the five hundred fastest computer systems in the world. The list has been established in 1993 and is published twice a year. It can be used to observe the growth of supercomputing performance. Additionally, it gives some strong hints and impressions about available computing performance in general. The left plot in Figure 6.2 reveals that the exponential growth of the fastest supercomputer is roughly the same as the one of the slowest supercomputer that made it into the Top 500.

high performance computing (HPC) is driven by the goal to come as close as possible to the theoretical peak performance of a system. It is therefore required to specialize the implementations of the underlying algorithms for the target architecture. Nevertheless, even good programmers will not be able to achieve satisfying performance without touching the algorithms. There is already a long list of existing advanced parallel algorithms, that provide a minimum overhead in synchronization.

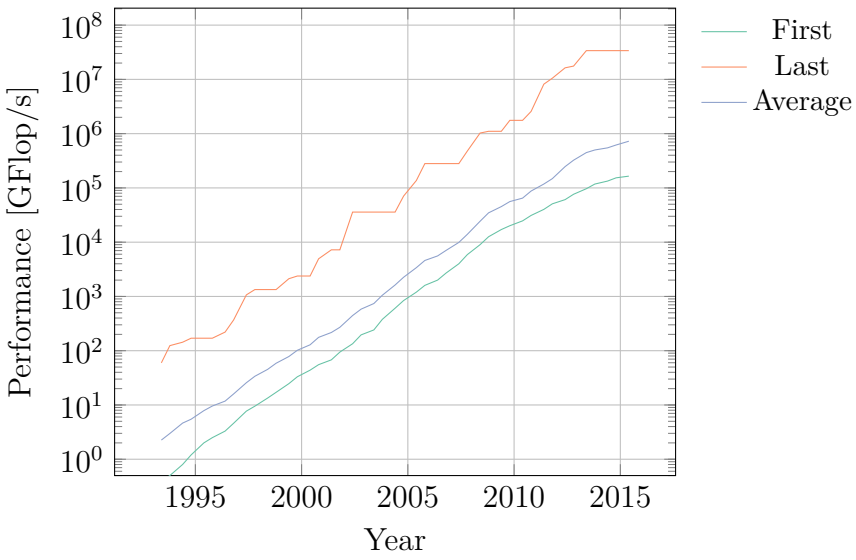


Figure 6.2: Growth of supercomputers performance according to the list of the 500 fastest supercomputers [139].

Recently another important list has been initiated: the Green 500 [49]. This list also contains a ranking of supercomputers. However, while the Top 500 ranks its entrants by their computations per second, the Green 500 ranks computers by their efficiency. The efficiency is measured in flops per Watt, i.e., computations per Joule. One condition to be in the Green 500 is that the machine is sufficiently fast to enter the Top 500. As a result machines may be required to leave the Green 500 despite being efficient enough.

The development towards better efficiency is of course driven by consumer products. Especially portable devices such as smartphones, tablets or laptops benefit from more power efficient CPUs. In the end not only the hardware has to improve for more effective computing, but also the software. The plot on the right in Figure 6.3 shows the development of the Green 500 list. An interesting quantity that can be obtained by performing an exponential fit over the acquired data. Using the performance data from the Top 500 and the shown efficiency from the Green 500 we can calculate the development of the power consumption per supercomputer.

The trend to very efficient supercomputers is crucial for further performance improvements. The exponent of the performance development of an average Top 500 system is still higher than the exponent of an average Green 500 machine. Right now the power demand is already on the boundary to a disaster from an ecological and an economical point of view. The rising demand will not help to alleviate this situation. Therefore we require much more efficient systems. Active research has to be done on more efficient cooling, network infrastructure, and processing architecture.

In the following chapters we describe the QPACE 2 supercomputer. The project

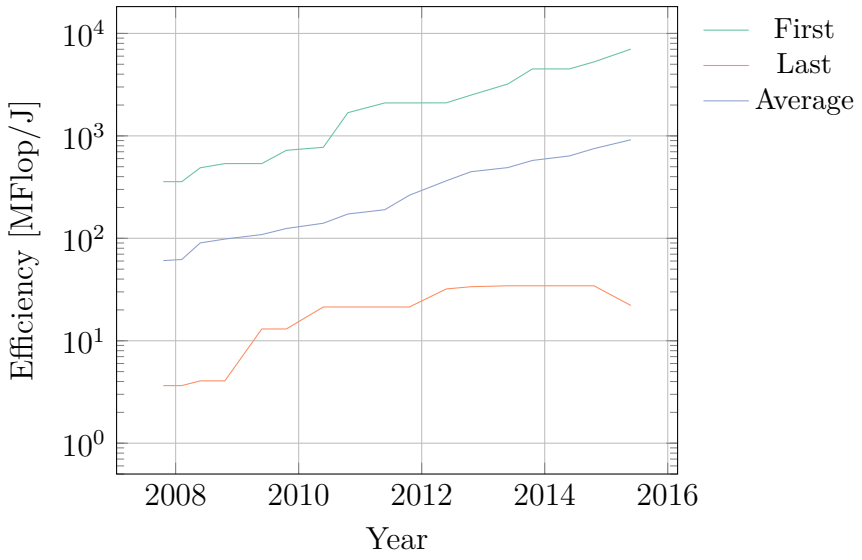


Figure 6.3: Efficiency development of supercomputers according to the list of the 500 most efficient supercomputers [49].

aims to provide a highly efficient computing platform for LQCD. Additionally, general purpose applications may be performed very efficiently. In Chapter 7 we look into the architecture of the system, as well as its computing and network units. A cooling technology that performs well was required to deal with heat transportation with low energy costs. The discussion of our solution takes place in Chapter 8.

While the project is very interesting on the hardware side, there are a lot of different software topics as well, which are discussed in Chapter 9. Most importantly, we go through a list of best practices to use SIMD and other techniques most efficiently on the QPACE 2 system. Finally, we introduce a new pattern, which tries to decouple a necessary scaling mechanism from specific implementations without significant performance loss.

7 The QPACE 2 Supercomputer

7.1 The QPACE 2 Project

Performing LQCD simulations is a compute-intensive task. The computation of sufficiently large lattices requires an enormous amount of calculations [55]. The most cost-effective systems, which are capable of finishing the computation within a reasonable time are custom-built supercomputers. Herein we describe the QPACE 2 supercomputer [13].

Our supercomputers have to be customized to provide all properties to be perfectly suited for the application of LQCD as introduced in Section 2.5. The first QCD Parallel Computing Engine (QPACE) machine is a perfect example for a custom architecture that is suited for QCD applications [14]. Its purpose was to provide a maximum of computing efficiency for LQCD applications, while keeping the required energy at a minimum. The architecture was highly optimized for power efficiency, placing first on the Green 500 ranking [49].

The release cycle for computer hardware is very fast. The rapid development results in the need to replace machines every few years in order to increase the overall performance by a full magnitude. It is therefore necessary to design systems in such a way, that core components can be easily exchanged with improved versions. A modular design tries to minimize costs in both, development and production.

The QPACE 2 project has been established as the natural successor of the orig-

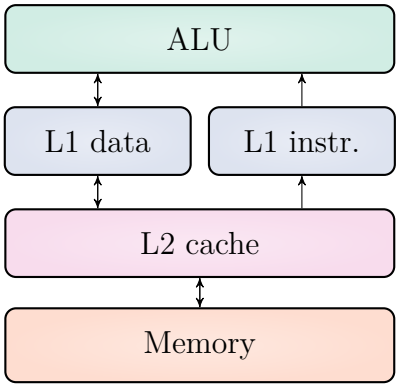


Figure 7.1: The classic architecture for a CPU features a single core that contains a cache hierarchy with access to main memory.

inal QPACE project [61]. Again it aims for high power efficiency. This is an important goal, since the power bill of a supercomputer usually exceeds the price of purchasing the product by far. Additionally, QPACE 2 tries to solve other important problems like providing an environment that is very familiar for most programmers and allowing even larger lattices to be simulated than its predecessor [103].

7.2 Architecture

QPACE 2 uses a modular architecture based on a Von Neumann model [105]. The system features a very modern and power efficient cooling system and the ability to change the computational heart of the system [13]. The system's CPUs are minimalistic and would not be used as the workhorse of a power efficient supercomputer. Instead the QPACE 2 system uses accelerators. The trend towards accelerator based systems also marks the end of the classical single core architecture shown in Figure 7.1.

In the previous years graphics processing units (GPUs) have widely been used as computational accelerators. However, using GPUs for ordinary computational tasks might be problematic. First, programmers might write code, which is not suited for the special architecture of such a GPU. To overcome this programmers need to spend more time on little details. Second, previously written code has to be ported, even though sometimes porting an application may be straightforward. A requirement is a certain similarity between the key features of the used architectures. A shift from CPU to GPU based computation presents a dramatic shift. This makes porting existing applications tedious. Third, there are applications for which the architecture of a GPU might not be the ideal choice. Here even experienced GPU programmers will not be able to outperform the application's

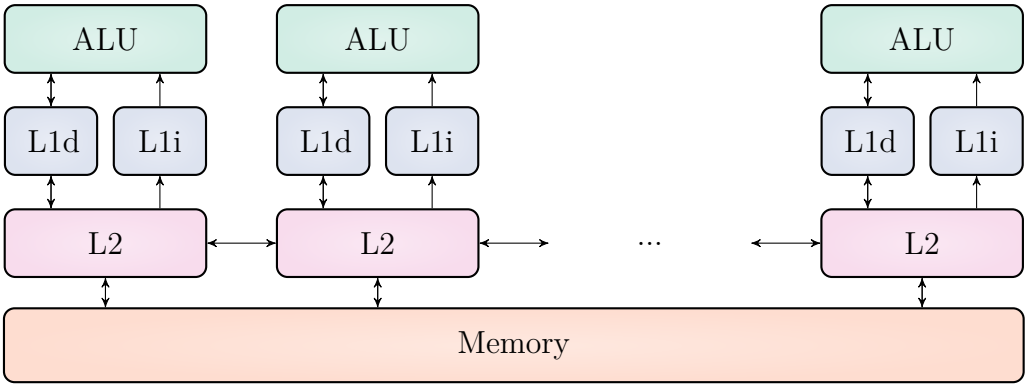


Figure 7.2: The MIC architecture features many weak cores that are connected via a ring bus interface.

performance on CPUs.

The accelerator that we use is the Intel Xeon Phi, which has been codenamed Knights Corner (KNC). The novel feature of this product is to combine the advantages of a GPU with the benefits found in ordinary CPUs. Effectively, the combination means that programmers can still rely on their knowledge for x86 systems, while enjoying the advantages of a Many Integrated Core (MIC) system. The block diagram displayed in Figure 7.2 represents the essential design, which is a series of lightweight CPUs connected via a ring bus.

QPACE 2 uses one CPU and four accelerators per node. The four Xeon Phis communicate over the Peripheral Component Interconnect express (PCIe) bus. This enables a very fast communication with low latency. A more detailed description of the used co-processor is given in Section 7.3.

The communication with other nodes is established over InfiniBand. Here the layout is the topology of a hyper-crossbar. A hyper-crossbar is the ideal solution for enabling closest neighbor communicator despite a limited wiring. The concept is described in greater detail in Section 7.4. The section also contains basic information on the used InfiniBand adapter card and some performance evaluations.

Parts of the architecture of QPACE 2 are inspired by its predecessors QPACE and iDataCool (IDC). While QPACE featured a custom field-programmable gate array (FPGA) network [61] with machines that used the IBM Cell processor, IDC [99] presented a standard architecture that used hot water cooling. The network and processing power has been replaced by an industry solution to guarantee support and reduce the development time.

The diagram in Figure 7.3 shows the conceptual architecture of a single QPACE 2 compute node, called a *brick*. A brick contains a network adapter with a dual port network processor (NWP), the four KNC cards, and a host processor card. The host processor card is named *Juno*. Juno is equipped with the Intel Xeon

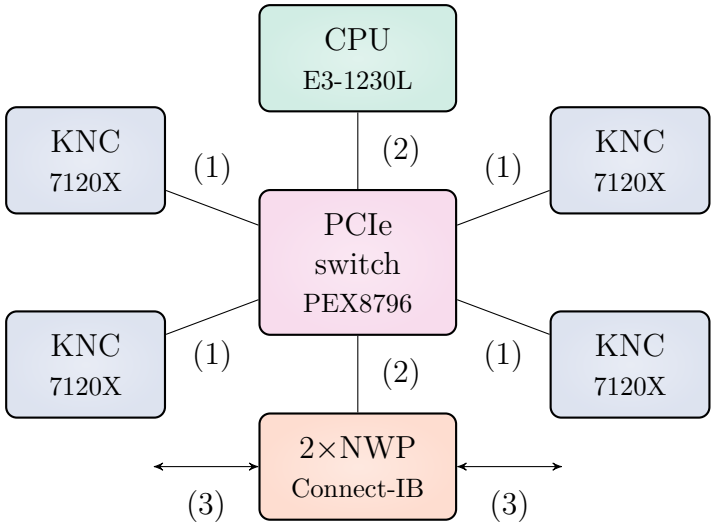


Figure 7.3: The conceptual architecture of a single QPACE 2 brick. The PCIe switch connects the four co-processors with the network processor and the CPU. The connections are given by (1) PCIe Gen. 2 x16 with 8 GB/s, (2) PCIe Gen. 3 x16 with 16 GB/s, and (3) FDR IB with 56 Gb/s.

E3-1230L processor running at 1.8 GHz. All these cards are connected via the PCIe bus provided by the PLX switch that exists on the midplane. The midplane is named *Mars*. The PLX switch has 96 lanes with PCIe 3.0 standard. Each of the six PCIe slots has 16 lanes.

Additionally, every brick has several additional connectors, e.g., for USB, VGA, a single 1 Gb/s Ethernet port to the host processor card and a single 100 Mb/s Ethernet port to the baseboard management controller (BMC). The connectors have been placed on the brick's front panel. The BMC of the host processor card is discussed in Section 7.5. There are connectors for debugging and light-emitting diodes (LEDs) for status information.

The QPACE 2 project has been proposed for a single rack installation. One rack consists of 64 bricks, or 256 KNCs. The rack fits 8 bricks in a row with 4 bricks side by side. The idea to fill the rack from both sides leads to an even higher compute density. Since a single brick has a height of 3 U, we need 24 U for all bricks. The 19 in rack comes with a height of 42 U. It still has some space left that is used for power distribution, cable management, a login node, as well as three Ethernet switches, and four switches for the brick-to-brick communication over InfiniBand.

The single rack of the QPACE 2 system has two power distribution units (PDUs). They are responsible for distributing the available power to the power supply units (PSUs), which are connected to the bricks via conductor rails, which are made of copper. The PSUs consists of five CPR-2021 units build by Com-

puware. Each unit can deliver up to 2000 W. Ultimately, the purpose of this architecture is to provide 12 V to every Mars by a so-called power backplane. The power backplane is directly connected to the conductor rails.

Such a high power consumption results in excessive energy dissipation. An effective cooling solution is therefore a crucial requirement. The cooling solution for QPACE 2 is discussed in Chapter 8.

7.3 The Intel Xeon Phi Co-Processor

The Intel Xeon Phi co-processor is the first x86 compatible accelerator card on the market. It combines some of the advantages of GPUs with ordinary CPUs. One of the features is that programming for the Xeon Phi feels like programming for an ordinary processor, that just provides a full magnitude more cores. Nevertheless, in order to use as much of the available computation power as possible, at least a basic understanding of the Xeon Phi's architecture is required.

If we look at Intel's roadmap we find that production processes in 2015 are able to yield 14 nm nodes with gate lengths smaller than 5 nm. Products created by such processes are part of the Haswell and Skylake microarchitecture generation. Each generation contains at least two product lines, based on different processes. This is known as the *Tick-Tock* model. A "tick" represents a shrinking of the process technology using the currently available microarchitecture. The following "tock" uses the same process, but designates a new microarchitecture. Recently an optional "refresh" cycle has been introduced, which accounts for the expanding development time by releasing a smaller update to the current microarchitecture.

Vector instructions have evolved from 128 bit Streaming SIMD Extensions (SSE) in the Nehalem generation to the 512 bit AVX-512 instruction set in Skylake. The Skylake microarchitecture generation features the first 10 nm process, which is produced under the codename Cannonlake. The Xeon Phi is based on the 22 nm process used for the products with codename Ivy Bridge and Haswell CPUs. Besides the silicon fabrication there are barely any other similarities to standard processors from these architectures. The Xeon Phi is actually based on the Larrabee microarchitecture [106].

There are a number of different models that have been released for the Xeon Phi. In Figure 7.4 a model with active cooling is displayed. All numbers are given for the Intel Xeon Phi 7120X, which is the version that has been chosen for the QPACE 2 project. This version features more memory, a higher frequency, and more cores than other models. Compared to the model 7120P, the 7120X comes without any bracket or passive cooling unit. This is ideal for our purposes, as we want to mount our own cooling solution.

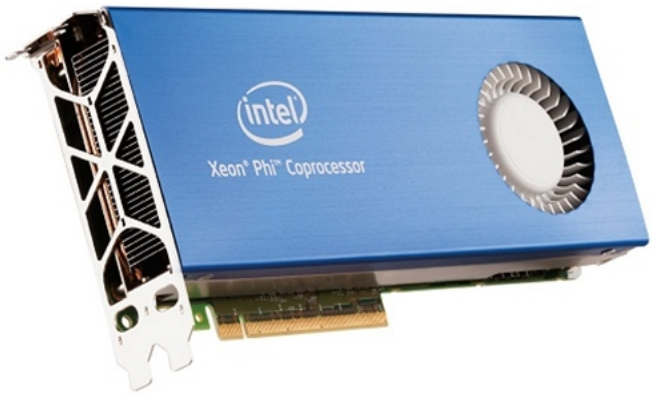


Figure 7.4: An Intel Xeon Phi with active cooling. Picture courtesy of Intel [33].

In the following we call the basic unit of computation a core. Every core has a number of arithmetic logic units (ALUs), which perform operations given in a program. A program is defined by its current state, the stream of instructions, and the data to handle. A core contains fast memory to cache upcoming or reoccurring instructions or data. A core of the Xeon Phi is much less powerful than one of any current Xeon processor. As a rule of thumb we may need at least a factor of three more cores on the Xeon Phi to achieve a similar performance [124].

Typically, a CPU contains more than one core. This allows running several tasks at the same time. In operating system (OS) terms we denote these tasks as threads in general. The OS schedules threads according to a priority list, which might lead to threads having to wait when all cores are currently busy processing other tasks.

The Xeon Phi 7120X has 61 cores, where each core delivers four hardware threads. Nevertheless, execution per thread is only possible every other cycle, which reduces the effective frequency by half. Due to the complicated execution model threads are used most efficiently if it is ensured that two similar operations, i.e., operations which share any part of the same execution pipeline, are not performed at the same time. Since the Xeon Phi runs a customized OS, it makes sense to dedicate a full core to the corresponding operating system. Therefore, we should only use a maximum of 240 threads over 60 cores.

A critical aspect for every multi-core processor is the concept of sharing memory. By considering some memory being shared with another processor, it is possible to alleviate common communication problems. A good example would be basic synchronization. The Xeon Phi has multiple memory layers. It does not use the random access memory (RAM) of the host, but provides its own memory. The memory is following the Graphics Double Data Rate (GDDR) 5 standard, which is comparable to the memory available on GPUs. Our model ships with 16 GB of

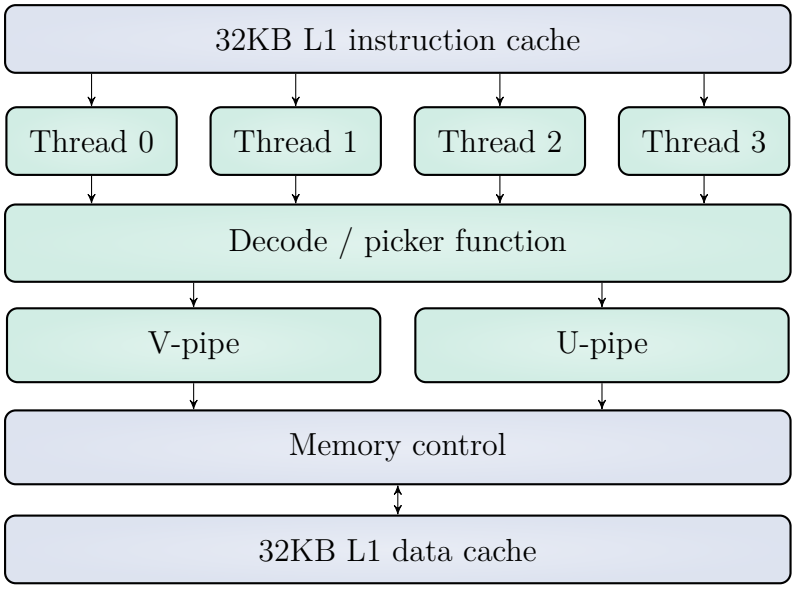


Figure 7.5: Core pipeline of the Xeon Phi architecture. The instruction fetcher and branch prediction unit gets the data into the prefetch buffers of the four contexts. The decoder bundles the data from the buffers, before the picker function determines the execution context at a given clock cycle.

RAM. The L2 cache is shared between the two hardware threads.

To distinguish between components that are directly involved in the instruction pipeline and those, which are only essential for the general performance, the terms core and uncore are being used. The latter describes the functions of a CPU which are essential for performance, without being strictly required. This includes the QuickPath Interconnect (QPI) controllers, the snoop agent pipeline, or the on-die memory controller as well. The core, however, contains the required components such as the ALU, floating point unit (FPU), L1, or L2 cache. Some bus controllers such as PCIe or Serial Peripheral Interface (SPI) are actually part of the chipset and not in the CPU at all.

The superscalar processor core is implemented to provide a 64 bit in-order architecture, which fetches and decodes instructions from four hardware threads. However, it is important to know that instructions are executed only every other cycle, resulting in four threads that use half of the frequency. Additionally, to standard x86 and x64 instructions on 32 bit and 64 bit operands, special Intel MIC instructions are included, which operate on 512 bit operands. However, none of the previous single instruction, multiple data (SIMD) extensions is available. This excludes instruction sets like Multimedia Extensions (MMX), SSE or Advanced Vector Extensions (AVX).

Instead a new instruction set has been included, which makes use of the dedicated 512 bit wide registers and Vector FPU (VPU). This VPU is provided for

Parameter	L1	L2
Coherence	MESI	MESI
Size	32 KB + 32 KB	512 KB
Associativity	8-way	8-way
Line Size	64 B	64 B
Banks	8	8
Access Time	1 cycle	11 cycles
Policy	pseudo LRU	pseudo LRU
Duty Cycle	1 per clock	1 per clock
Ports	Read or Write	Read or Write

Table 7.1: The main properties of the Xeon Phi's L1 and L2 caches. The LRU cache policy discards the least recently used items first.

each core. The special instruction set features efficient support for reciprocal, square root, power, and exponent operations. Special operations, such as scatter, gather, or streaming store capabilities to achieve higher effective memory bandwidth are included. Some special operations, such as fused multiply-add (FMA), are executed in the VPU instead of the standard ALU pipelines. Furthermore, the Xeon Phi introduces novel vector scatter and gather operations [88].

Even though the processor is said to be x86 compatible, it is missing full compatibility. For instance, binary support is excluded. Existing programs need to be recompiled from their sources to work with the Xeon Phi. A handful x86 instructions are missing, including a few fairly common ones. In the end, the added features, such as the new VPU, streaming stores and others, are considered more important than the lack of real x86 compatibility.

There are two separate L1 caches. Each cache can contain 32 KB of information. One is for data and the second one is specialized for instructions. Associativity is 8-way, with a cache line-size of 64 B. On the Xeon Phi two instructions can be issued per cycle, as long as both are split to two different pipelines, which are called U-pipe and V-pipe [122]. The V-pipe can only be used to execute scalar instructions. The V-pipe is responsible for prefetches, loads, and stores. All of the VPU instructions are thus issued from the core through the U-pipe. The load-to-use latency of the L1 cache is 1 cycle, which allows using a requested value in the next cycle. Vector instructions follow a different pattern, which might take longer. The block diagram in Figure 7.5 illustrates the core pipeline of the Xeon Phi architecture.

The L2 cache is a unified cache capable of storing data and instruction informa-

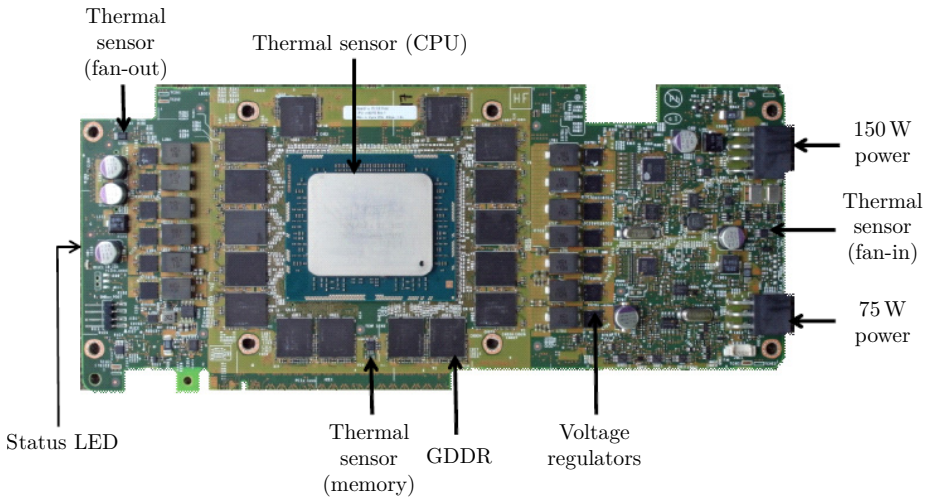


Figure 7.6: Annotated illustration of the front PCB of a Xeon Phi. The key components have been labeled with arrows and a short description.

tion. Each core contributes 512 KB of L2 to the globally shared L2 cache storage. The private L2 cache is kept fully coherent by a distributed tag directory. Associativity is 8-way as with the L1 cache. The cache line-size is 64 B. An important difference is, however, the raw latency is much higher than for the L1 cache. We have at least 11 clock cycles load- to-use latency. The most important properties of the two cache levels are presented in Table 7.1. The least recently used (LRU) policy requires keeping track of what was used when, which is expensive if we want to make sure that the algorithm always discards the least recently used item.

The GDDR memory access is implemented over a ring bus. The bandwidth of the ring bus is 512 bit. It represents a bidirectional ring interconnect, called the core ring interface (CRI). The CRI hosts the tag directories and is responsible for connecting the L2 cache of every core to the memory controllers. The PCIe communication is managed on the CRI. In total eight memory controllers are available to every core for memory access via the ring bus. A memory controller contains the logic necessary to load, store, and refresh dynamic RAM (DRAM). Without constant refreshes, DRAM loses the stored data, since the capacitors leak their charge within some time $t \geq 64$ ms [43].

The Xeon Phi is an ideal system for highly parallelizable applications, e.g., an application that makes use of the DiagMC algorithm for simulating a field theory problem as discussed in the first part. In general, applications that seem to be well-suited for running on GPUs may be good candidates for the running on the Xeon Phi. The reason is that the Xeon Phi combines the advantage of an ordinary x86 CPU with principles that have been seen in the basic architecture of GPUs.

In Figure 7.6 we get a glimpse of the front of the Xeon Phi's printed circuit board (PCB). The most important visually accessible components have been

annotated with a short description. We are mostly interested in the location of the thermal sensors. The information that can be read out from these sensors is valuable for the design of our cooling system. We come back to the sensors shown in the picture and the ones placed on the back of the PCB in Section 8.1.

7.4 InfiniBand Network

A crucial task for any HPC system is the connection between the various nodes. The connection usually follows a certain technical specification and design. For QPACE 2 we choose the InfiniBand (IB) network technology. The topology of the network has been selected to form a hyper-crossbar. In this section we will explain the basic reasons for our choice and outline the expected performance.

We start by introducing the selected hardware, which utilizes the latest generation PCIe standard.

7.4.1 Connect-IB

The reasons for using an IB network in an HPC system are obvious: The very high throughput and the very low latency provide an ideal basis for fast message transport. Originally developed for data centers, it is today the most used interconnect system for HPC systems. It is used by the majority of Top 500 contestants [139]. Over half of the entries choose IB for connecting their computing nodes. Additionally, large computing centers, such as the ones from Google [6], are relying on IB. Here the most important reason to choose IB is its efficiency, especially in terms of power consumption per transfer unit.

Our choice is the Connect-IB adapter. This PCIe card is produced by the company Mellanox. The maximum bandwidth is delivered across the PCIe 3.0 standard with 16 active lanes by using two ports of fourteen data rate (FDR) IB. Here we can take full advantage of FDR. That configuration is able to supply a sufficiently large throughput together with low latency. Furthermore, Connect-IB could be used in conjunction with the PCIe 2.0 x16 standard.

Next to the guaranteed bandwidth and low latency services we additionally like to have direct memory access (DMA) instead of programmed input / output (IO) (PIO). Essentially, this moves the data directly to the IB card instead of having to go over the CPU. This maximizes the CPU efficiency and accelerates parallel and data-intensive performance. Connect-IB offers IO consolidation [57], which may be used to aggregate multiple traffic types into a single switch. The Connect-IB card itself is considered to be power efficient, which makes it an ideal device for the QPACE 2 project.

The ping latency is supposed to be as low as 1 µs, with over 130×10^6 messages per second. The bandwidth over IB is specified to be greater than 100 Gb/s. Most notably DMA allows us to communicate with the card directly from the co-processor. Even though the basic communication is mostly solved via software, the enhanced hardware support prevents otherwise unnecessary copy operations. Here we can directly copy data from the co-processor to the interconnect and vice versa.

Last but not least, the adapter contains two ports. This makes it possible to map every brick to two switches. Our topology requires this particular mapping.

7.4.2 Hyper-Crossbar

We choose a hyper-crossbar topology[1], which has already been used in other systems, most notably in the Computational Physics by Parallel Array Computer System (CP-PACS) project [104] developed by the Center for Computational Sciences at the University of Tsukuba. The topology can also be used efficiently for Ethernet networks [140].

The basic idea is to map every brick in the system to N_S switches, where N_S is the dimension of the topology. Since we decided for the Connect-IB adapter, we can use a two-dimensional topology without having to install further IB cards. Here we have to assign every brick to two switches.

Our IB switches provide 36 ports. That allows us to connect 32 bricks to a single switch while having 4 extra ports that can be connected to a storage system and to a torus of switches. All in all the hyper-crossbar has a higher connectivity than the typical torus solution. We obtain full connectivity in every single dimension with just a single switch hop. The connectivity between each brick is given by performing only a few hops between the switches.

In Figure 7.7 the hyper-crossbar used in QPACE 2 is shown. Each ellipse is representing a single brick, which contains a Connect-IB adapter, providing connection to two switches. Hence every brick is connected to a horizontal and a vertical switch. In total there are four switches for the whole rack, connecting all 64 bricks, or all 256 Xeon Phi co-processors. It is easy to prove that the maximum hopping distance for this scenario is 2. The maximum number of hops is only necessary if the communication is diagonal, e.g., from a brick in the upper left quadrant to a brick in the lower right quadrant. At least one switch has to be used.

The hyper-crossbar topology generalizes easily to higher dimensions. In our specific scenario, however, we apply a mapping from higher dimensional applications

[1] as suggested by Peter Boyle

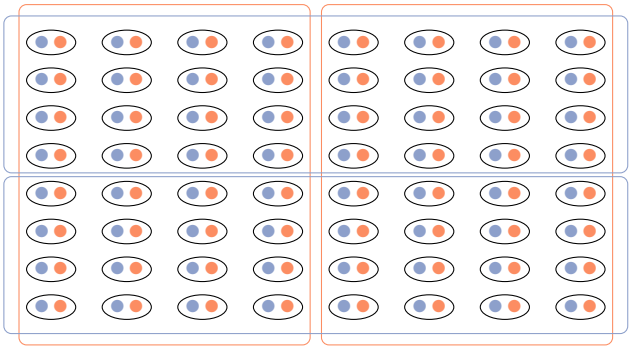


Figure 7.7: The hyper-crossbar topology for 8×8 bricks. Each of the bricks has 2 ports and is connected to one switch in the horizontal (blue) and one switch in the vertical (orange) direction. The switches are indicated by rectangles. They form a 2×2 mesh.

to a lower dimensional hyper-crossbar.

7.4.3 Performance

The performance of a proposed network solution has to be evaluated intensively before any decision should be made. For these evaluations we use a variety of different setups, which cover different cases by emulating the future system in the given scenario as well as possible.

In the following we look at one of these benchmarks. The purpose of the described benchmark is to investigate the performance of the Message Passing Interface (MPI) library against using plain IB verbs within a brick. The latter defines the application programming interface (API) of the IB cards and is considered much more low-level than using a wrapper library, such as MPI.

The setup for this benchmark looks as follows:

- A host CPU attached to the first socket of a sufficiently well equipped host. The host is configured with the distributed OpenFabrics Enterprise Distribution (OFED) stack in version 1.5.
- A single port FDR IB card (Connect-X3) attached to the first socket.
- Two KNC co-processors labeled *mic0* and *mic1*. Both are attached to the first socket of the host.

We are interested in two quantities: The bandwidth B_W and the latency δt. We start by evaluating the performance of communication between two points by using the MPI library in conjunction with the default network protocol of the KNC, which is called Symmetric Communications Interface (SCIF). For the points we choose two unique values from the set containing the host, mic0, and mic1.

As benchmark suite we use the MPI-based OSU benchmarks [89]. Some details about the OSU benchmarks are outlined in Section 9.6.5.

We are not limited to SCIF. An interesting comparison is the possibility of using the Mellanox ConnectX (MLX) adapter. Here we go over the IB network to circumvent SCIF. We can then compare the rows of the plots in Figure 7.8 with each other. While SCIF does not give us a decent performance for small message sizes, we obtain lower latencies for larger messages. Additionally, SCIF reaches the bandwidth limit in any scenario, not just in cases where we measure the host to KNC performance.

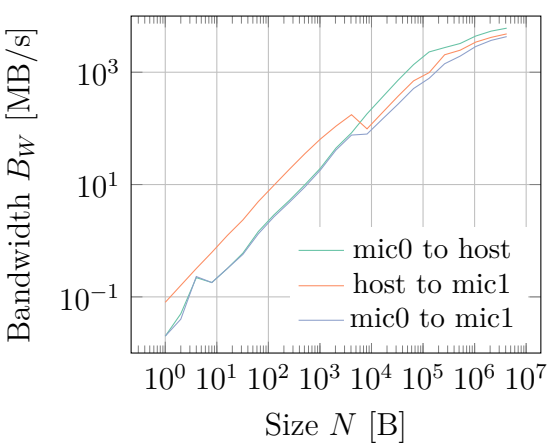
We only show the combinations of sending from host to mic0, mic1 to host, and mic0 to mic1. The direct communication between the co-processors can be reversed to send from mic1 to mic0 without observing any impact on the bandwidth or latency measurements. In general we know that exchanging the MICs will not change the outcome.

Since both ways of using MPI have shown issues in different cases, we need a third option. We use the IB verbs API to directly access the device for managing the communication. One advantage of such an approach is the flexibility. We are free to use specialized commands that go beyond the scope of the MPI specification. The most striking benefit is the reduction of overhead due to software abstraction. Using IB verbs for our communication excludes the previously used OSU benchmarks. Instead we now use a set of standard benchmarks delivered with the IB tools.

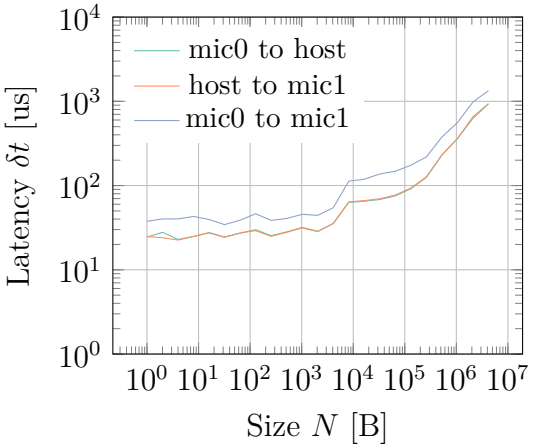
In Figure 7.8e and Figure 7.8f we see the bandwidth and latency for testing the MLX adapter from an application in conjunction with the IB verbs API. We observe that SCIF still seems to utilize the bandwidth better in scenarios where the co-processor is sending the data.

In conclusion we can say that IB verbs should be preferred over ordinary MPI. This statement can be already confirmed for intra-node communication from the provided benchmark. The latency is slightly smaller for small message sizes and the bandwidth indicates a much better behavior. The maximum bandwidth is also reached earlier. Nevertheless, some of the downsides, that have been seen in the comparison of MLX against SCIF, can be observed again. For instance, we can detect a different behavior between sending from the host to the co-processor and sending from the Xeon Phi to the host.

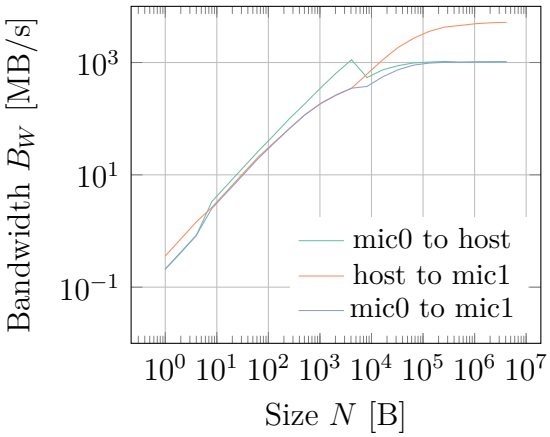
Further benchmarks revealed other interesting aspects. As an example we realized that the write latency is lower than the read latency. Additionally, a write saturates the bandwidth faster than a read. Most of these benchmarks have been made with a single active port using a FDR switch and cables with support for FDR connecting two suitable hosts with a single integrated KNC.



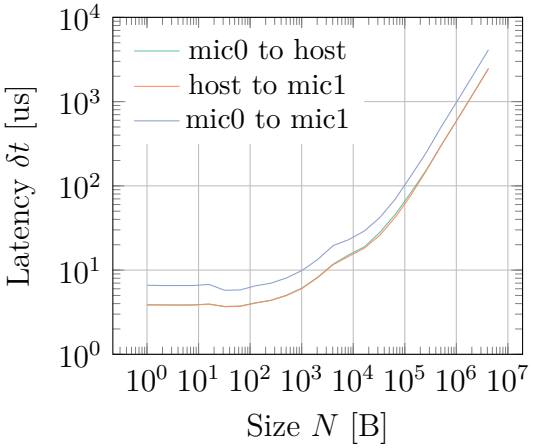
(a) MPI over SCIF bandwidth



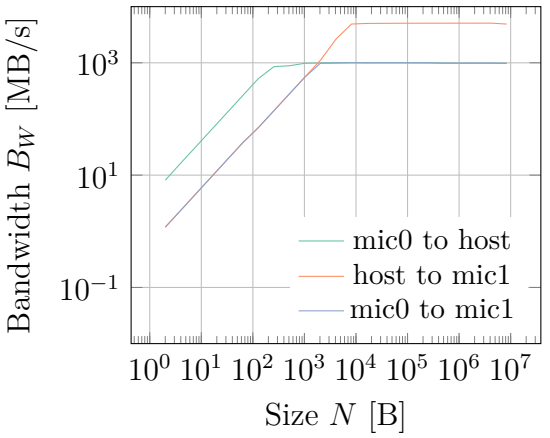
(b) MPI over SCIF latency



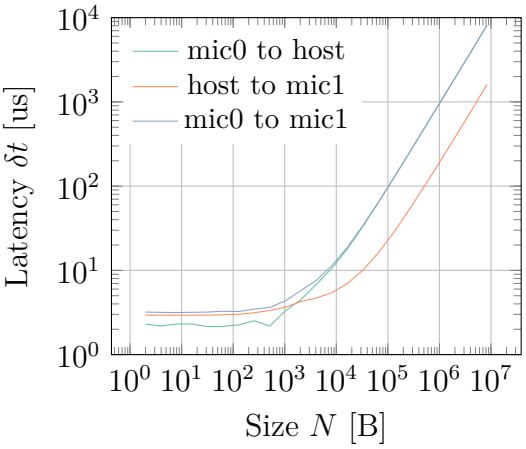
(c) MPI over MLX bandwidth



(d) MPI over MLX latency



(e) IB verbs bandwidth



(f) IB verbs latency

Figure 7.8: Performance of the communication by using MPI over SCIF (Figure 7.8a, Figure 7.8b), MPI over MLX (Figure 7.8c, Figure 7.8d), and IB verbs over MLX (Figure 7.8e, Figure 7.8f). The opposite directions yield the same result and have been omitted.

It has been confirmed that using the dual port effectively doubles the bandwidth for actually sending data. Benchmarks which used the dual ports "simultaneously" have been set up just like the single port benchmarks, however, with the two measurement processes running in the background. Furthermore, experiments with the maximum transmission unit (MTU) show that a large value for the MTU is better. For us the optimum seems to be around 2048.

7.5 Baseboard Management Controller

The QPACE 2 system uses Intelligent Platform Management Interface (IPMI) for resource management and monitoring. IPMI defines a set of interfaces that can be used to manage computer systems and monitor their operations. We use IPMI to control the status of a brick, monitor the brick's activities, and gain remote access to the brick's hardware. An important component in the implementation of the IPMI specification is the BMC.

The BMC is responsible for the intelligence in the IPMI architecture. It is a special microcontroller, which is embedded on the Juno card of a QPACE 2 brick. The BMC manages the interface between the system management software and the CPU card. As an example there are various sensors, such as the temperature sensors on Juno, which are being monitored on the local BMC. If a temperature sensor exceeds a given threshold over a predefined tolerance time, the BMC can autonomously initialize the shutdown sequence. This should prevent possible damage by overheating.

The BMC consists of several software components that make controlled board management possible. The *Das U-Boot* boot-loader starts and loads our primary OS image, which is then automatically configured. On the hardware side the BMC uses an integrated ARMv9 processor with 128 MB RAM and a 16 MB NAND flash drive. The flash drive stores the firmware, which is loaded during the startup. One of the buses that is controlled from the BMC is the Inter-Integrated Circuit (I2C) bus.

For the QPACE 2 system we need to provide a custom version of the firmware. The custom software gives us additional management software, which is adjusted to the capabilities of our brick. As an example we can read the values from the custom temperature and humidity sensors mounted on the brick. The BMC can be controlled remotely via SSH, thus allowing scenarios such as emergency shutdowns, reboots without physical interaction, and system monitoring. The necessary power management is possible by using the integrated hot swap controller over the I2C bus.

8 Hot Water Cooling

8.1 Applications for Cooling Benchmarks

The ability to compare tests across different devices requires a well defined test setup. Next to the basic hardware configuration we also rely on the behavior of our software. The software is used to drive the test by running a set of predefined operations.

Our selection of adequate tests contains the High Performance Linpack (HPL) benchmark, which is one of the standard tests for HPC systems, and the Burn-MIC application. While the former is solving a problem with fixed size and therefore constraint time, the latter guarantees constant usage for an arbitrary amount of time. The implementations are listed in Appendix F.2.

8.1.1 High Performance Linpack

The Linpack benchmark provides a measurement of a system's floating point computing power. The application tries to solve a system of linear equations $Ax = b$ for a dense matrix $A \in \mathbb{R}^{N \times N}$. The operation count that is associated with the benchmark is given by $2N^3/3 + 2N^2$. The size of the problem is determined by the size of the machine. In case of a single co-processor the maximum size is approximately $N = 4 \times 10^4$. For a whole brick N doubles. This is justified by providing four KNCs instead of one. Hence the available memory quadruples.

We use the HPL in the most recent form [42]. The application links to any

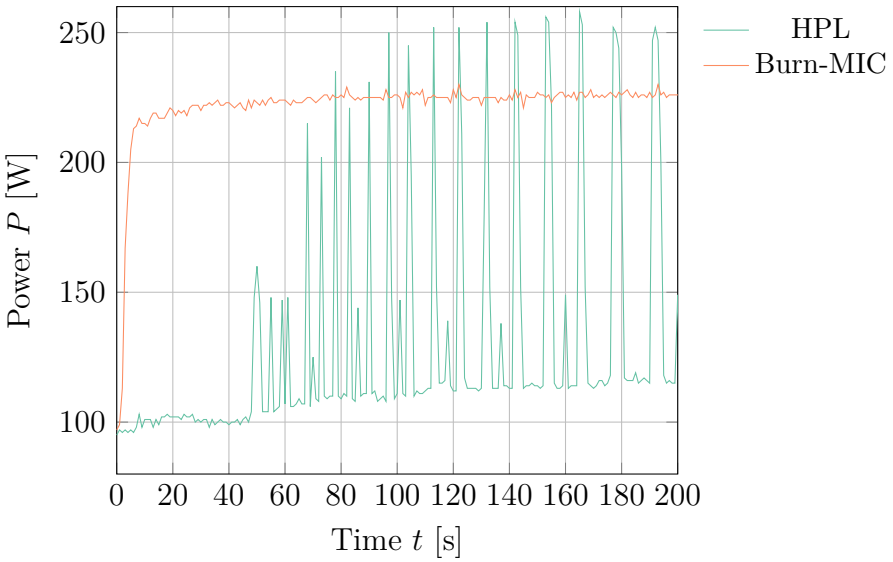


Figure 8.1: Power readings during the beginning of our test applications. There is only one application running on a single KNC simultaneously.

available Basic Linear Algebra Subprograms (BLAS) and MPI implementation, however, we favor the binaries from Intel as they are already optimized for the Xeon Phi. Therefore we use the BLAS implementation is provided by the Math Kernel Library (MKL). The whole HPL application comes in multiple versions, which can be compiled for hybrid execution, offloading, host or KNC only. For the cooling tests we exclusively use a KNC only version, which is run on a single KNC only.

A typical Linpack run does not result in a continuous power or computation demand. Instead we see only a spike in the power consumption during a very short while. Most of the time is spent on the preparation of a run or the cleanup afterwards. We use a Linpack configuration that results in multiple runs to maximize the stress on the co-processor. In these runs we vary the size of the matrix N, going from rather small sizes to larger ones. Each size is run three times to get more statistics about the KNC's actual performance rating.

In Figure 8.1 the characteristic plot for the HPL application running on a single co-processor is drawn. We do not observe a constant power drain, but rather a power consumption that shows the behavior of a rectangular function that appears during the computation with a width that is proportional to the given workload. The resulting temperature plots have to follow a sawtooth function. We have an immediate rise when the computation starts and a slightly slower decline afterwards. Before the actual computation quite a lot of time is spent on preparing the run.

The version of HPL that is used for our cooling benchmarks is not the same version that is deployed for benchmarking the whole system. Here we consider a more

optimized variant [69], which runs over the whole rack with a single configuration. The configuration has been tuned for the desired run.

8.1.2 Burn-MIC

The Burn-MIC application is a custom software that tries to stress the KNC, maximizing its power consumption. The application uses all available threads of a KNC. The threads are pinned to their cores via affinity flags (see Section 9.6.4). The target core is detected by using a unique identifier for each thread. This identifier is determined by the Open Multi-Processing (OMP) framework, which is used for parallelization.

Each of the four threads per core are loaded with work. The first thread will work in loading data from memory, while the second thread is assigned to load the data from the caches. Finally, the third and the fourth thread are jointly doing some work in floating point arithmetic. This covers many of the internal units. The process comes with the promise of utilizing most pipelines permanently.

Every assignment uses intrinsics to fulfill its task. Additionally, intrinsics are required to use the vectorization capabilities (see Section 9.5), which gives us the benefit of enhanced performance and maximized pipeline usage. To have the same runtime for all assignments we need to adjust the size of their working sets properly. While the FPU assignment only deals with a number of cachelines, the cache assignment uses a full L1 cache. The memory assignment has to cope with the L2 cache respectively.

In Figure 8.1 the beginning of a typical Burn-MIC run is shown. In contrast to the HPL application we observe a strong power demand from the start. Furthermore, there are just minimal fluctuations during the run. Overall the application allows us to keep the stress at a very high level, which is the main advantage of Burn-MIC compared to HPL.

Of course, we could practically achieve a similar result by just using a matrix-matrix multiplication that is very well optimized. An example is the DGEMM routine from an optimized BLAS implementation. However, here we would not use the same variety of registers and instructions, which makes Burn-MIC a suitable tool to verify most of the functionality of a Xeon Phi.

8.2 Design of the Interposer and Roll-Bond Plate

Hot water cooling is a technology that has been introduced to HPC in the recent years. One of the first projects has been IDC [99]. The idea propagated into larger projects, such as the 3 petaflops SuperMUC system [45]. QPACE 2 uses hot water

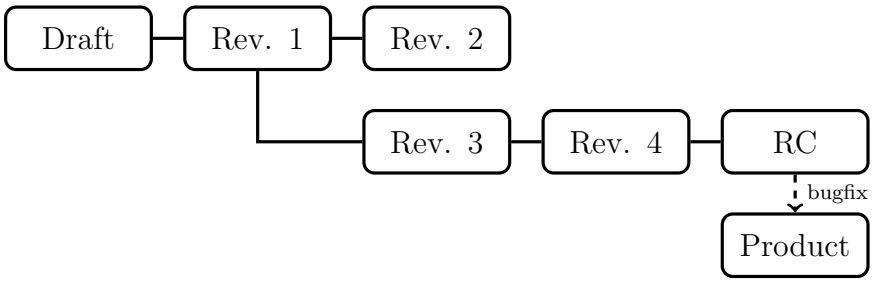


Figure 8.2: Revisions of the KNC interposer with their design ancestors.

cooling in an innovative way that improves the existing concept established by IDC.

The basic principle is similar to ordinary water cooling. However, while ordinary water cooling requires an enormous system composed of chillers, compressors, and more, hot water cooling only relies on a water circuit that can be regulated with slightly cooler water. A heat exchanger makes it possible to have a constant temperature within the cooling circuit. Usually, a temperature around 35 °C is an efficient choice.

The idea originated from the way blood and water move through our body. One of the benefits is the reduction in system size. Systems that rely on hot water cooling may be up to 10 times smaller with a significantly reduced energy consumption. Among the reasons for the popularity of hot water cooling is the cooling efficiency. Overall it is $\mathcal{O}(10^3)$ times more efficient than air cooling. The explanation are two-fold: Water has a higher specific heat capacity than air. Therefore it is able to absorb more energy than air. Additionally, water has a much higher density.

The boards are equipped with a cover that is called a roll-bond plate. The roll-bond plate is not directly attached to the board. Instead it is connected to the board via a so-called interposer. It is crucial that the interposer connects the two parts, board and roll-bond plate, by using the highest thermal conductance. The job of the roll-bond plate is to transport the heat most efficiently to the water. Water flows through a system of tunnels that is incorporated into the roll-bond plate.

8.2.1 Development

The original design for cooling the KNC co-processors has been created by Eurotech. A set of simulations and calculations has been used to verify the availability of desired flow and heat transport properties. However, the production process is not ideal and impurities have the ability to spoil the design. In various revi-

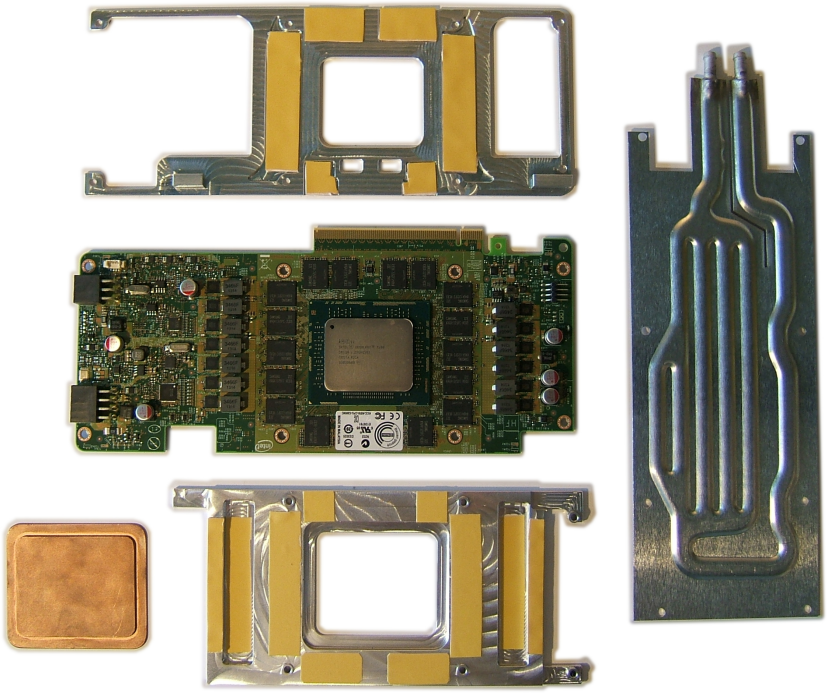


Figure 8.3: The Xeon Phi board (centered) with the interposer components (above and below), the roll-bond plate (left), and the copper block in the bottom left corner. The yellow pads are heat conducting spacers made from TIM.

sions we tried to work around potential flaws. The graph in Figure 8.2 shows the revision version tree.

The initial version contained many features of the final product, but had to be revised nonetheless. Even though the performance has been quite satisfying in retrospect, alterations due to mechanical problems were necessary. Additionally, we had to answer the question if the current design was already good enough, and if small changes could be beneficial.

As an example the main reason for the modifications in the first revision was to reduce the bending stress for the PCB. It has been observed, that the spatial tolerance is not sufficient. Consequently, gaps have been inserted to reduce the mechanical pressure. The gaps should not be too large, otherwise air bubbles are created, which reduce heat conductivity.

We choose to place a block of copper instead of the bare aluminum layer inside. The idea is to use the improved heat conductivity of copper to cover the main heat spot above the die even better. One problem with this approach is the different thermal expansion of copper compared to aluminum. We require some flexible medium to act as a mediator between both components. In our case we use a special high conductive thermal grease. The selection process for this component is discussed in greater detail in Section 8.3.

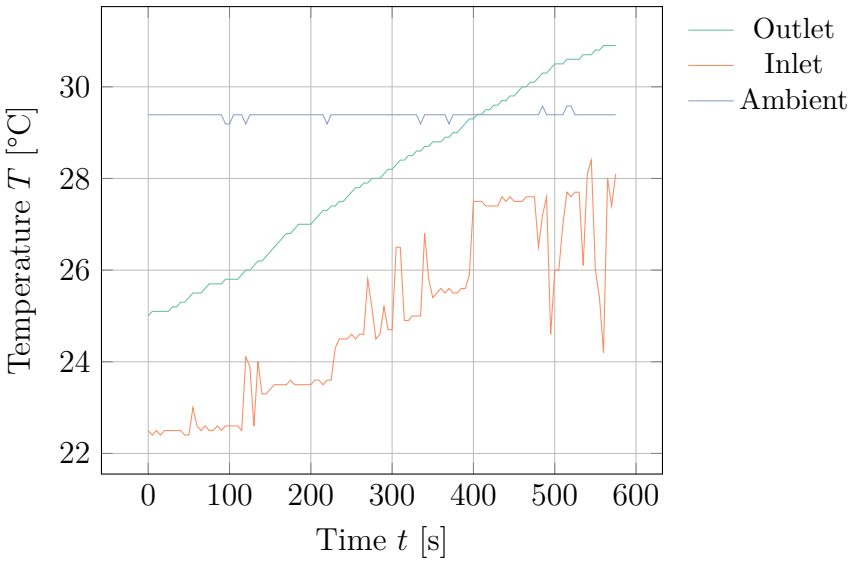


Figure 8.4: Temperature sensor readings in the closed cooling circuit during Burn-MIC with a single MIC. The measurement interval has been set to 5 s.

Finally, some design and process improvements are reviewed and applied after a careful evaluation of the release candidate (RC). The back side contains a 0.5 mm thick layer with thermal interface material (TIM), which insulates the board from the aluminum plate electrically, without disconnecting these components thermally. The edges of the copper inset have been beveled to reduce the block's degrees of freedom.

In Figure 8.3 we see the final layout of the cooling components for the Xeon Phi co-processor cards. The yellow pads in the picture are the heat conducting spacers (TIM). The interposer shown above the Xeon Phi is placed on the back side, which is not directly connected to the roll-bond plate. It is only used to enclose the package. The other interposer is glued to the roll-bond. The connector on the right of the roll-bond plate is used as water inlet, the left connector as water outlet.

8.2.2 Cooling Performance

We use two applications to test the cooling performance. For testing extreme performance in short time periods we use HPL. To gain insights on the cooling performance during continuous high-load, we run the Burn-MIC application. Both applications are described in Section 8.1.

For each application run we measure the temperature and flow of the closed cooling circuit, the temperature on the Xeon Phi co-processor, and its power drain. The plot in Figure 8.4 shows an example of such readings. We see the measurements of the temperature sensors in the cooling circuit during a typical

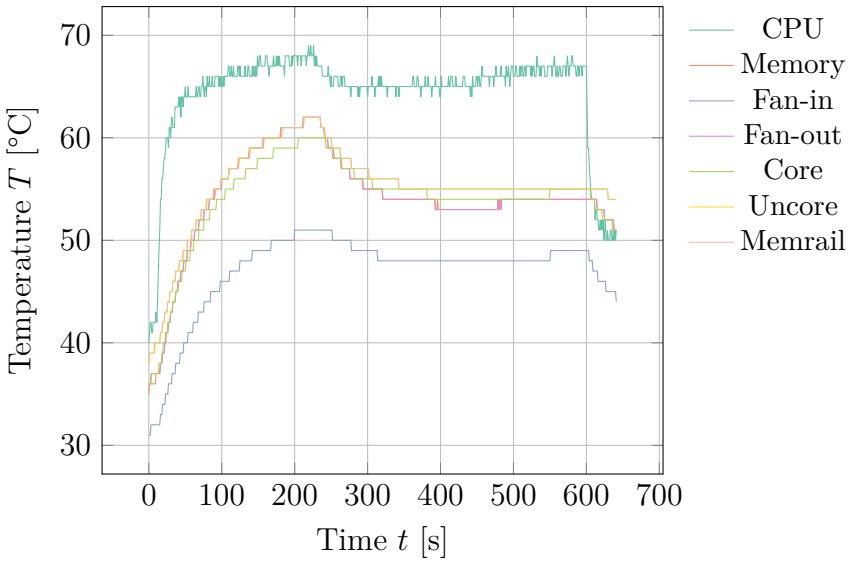


Figure 8.5: Temperature sensor readings in the MIC co-processor during the staged Burn-MIC. The sensor has been read out every second.

run of Burn-MIC.

While the cooling circuit contains three different temperature sensors, that represent two points within the cooling circuit and the environment temperature, the Xeon Phi co-processor has seven. All temperature sensors are placed on the board. Of course, the most significant reading is usually the CPU temperature, which is sitting right next to the die.

Usually, the CPU temperature represents the maximum temperature, as shown in Figure 8.5. The plot illustrates the characteristic curve of a staged Burn-MIC run. We can see two distinct phases in the application execution. The first phase results in a peak temperature that is a higher than the peak of the second phase. In the beginning and the end of the application the temperature readings drop to the idle level of the Xeon Phi.

The picture looks completely different when considering HPL as a benchmark. Here we have a high-load phase alternating with an analysis phase. The latter yields a temperature that is slightly above the idle temperature. The peak of the high-load phase depends on the actual problem size, which directly influences the length of the phase. A longer lasting high-load phase results in a higher temperature peak.

In Figure 8.6 we see the readings for the temperature sensors during a typical HPL benchmark run. We use two Xeon Phi simultaneously. The peak temperature and the progression of the CPU temperatures are very similar. The small deviation between the two co-processors can be explained with side-effects of the test setup, where other devices that radiate heat or block heat transfer may influence the

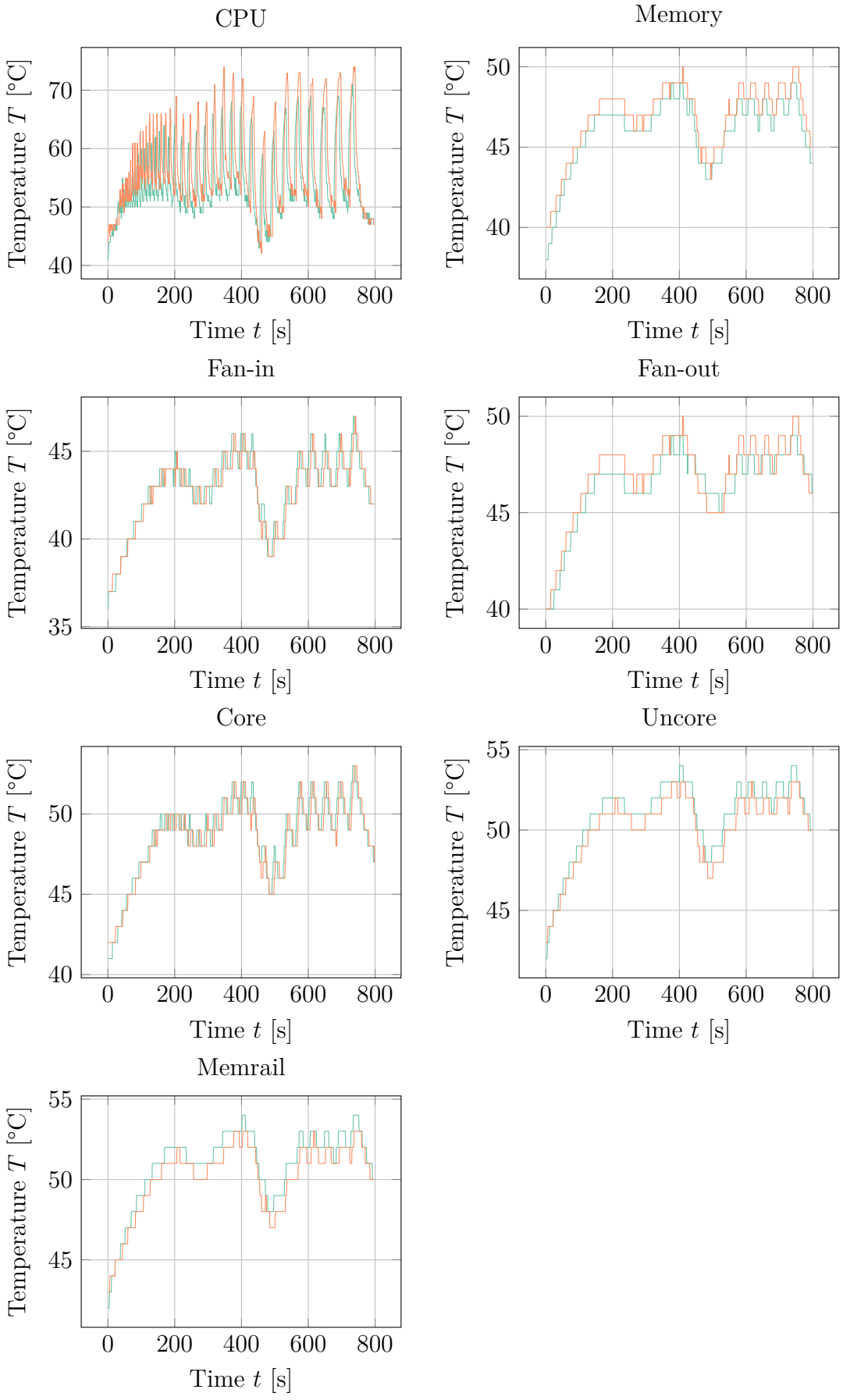


Figure 8.6: Overview over the heat sensors of the two MIC co-processors placed in close proximity to each other during a complete run of HPL.

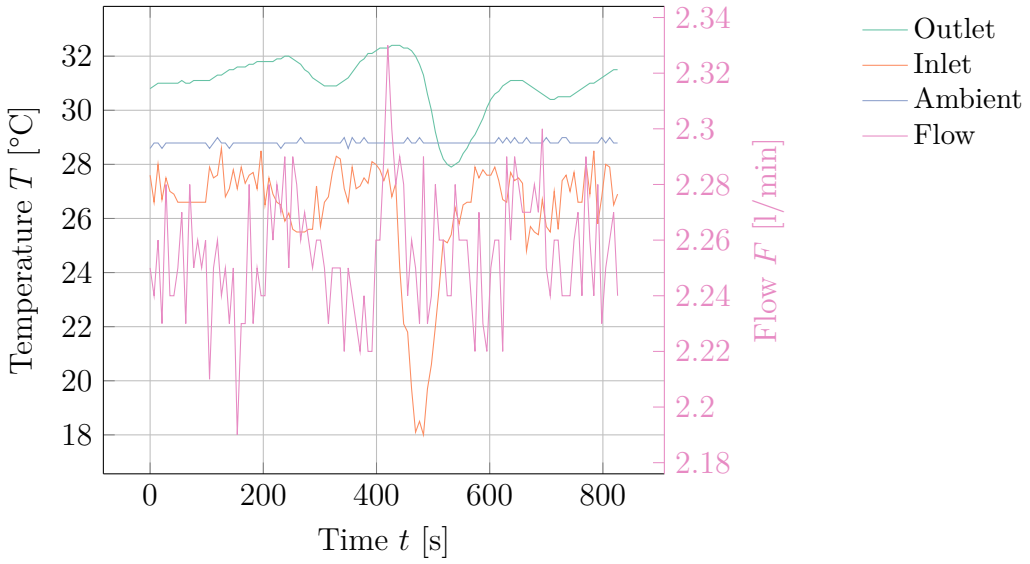


Figure 8.7: The circuit temperature during a run of HPL with two MICs. The measurement interval has been set to 7 s.

temperature. The drop in the middle of the run has been caused by over-cooling.

Another thing we are interested in is the circuit temperature. Additionally, we want to see the flow F during the benchmark. The flow is drawn in the plot of Figure 8.7. Here we observe high variations of the flow. We can infer that the source for the temperature drop around 450 s of Figure 8.6 is actually coming from fresh water that was inserted in the cooling circuit.

Once the inlet temperature falls, cool water is inserted into the circuit. This insertion comes with a higher flow rate. After some delay, the outlet temperature reading starts to drop. One conclusion from the plot is that the heat exchanger regularization has not been optimal during the illustrated run. While flow fluctuations are typical, the variation registered in the inlet temperature sensor should be minimal. Only if the outlet sensor reads approximately constant temperatures, the regularization can be considered optimal. Nevertheless, the over-steering of the regularization is a problem that is addressed separately.

We may look at temperature readings of the other sensors that have been taken during the HPL benchmark. Again we observe that the plots for both MICs show a very similar behavior. Both co-processors seem to be approximately equivalent from the cooling perspective. It is interesting to observe that spikes with high-load do not appear as obvious as the CPU temperature plot. The CPU is certainly one of the best cooled components on the KNC. The larger surface to volume value for the small object encourages the heat transport to the adjacent layers. Therefore, the temperature drops quite fast once the load decreases.

Run	ω [min^1]	F [l/min]	MICs	F_{eff} [l/min]
1	1850	0.6	4	0.15
2	2300	1.1	4	0.25
3	2750	1.5	4	0.40
4	3200	1.9	4	0.50
5	3700	2.2	4	0.55
6	3700	2.2	3	0.70
7	3700	2.2	2	1.10
8	3700	2.2	1	2.20

Table 8.1: Properties associated with each run. The flow is the average value of the measurements during a run of the Burn-MIC application. The effective flow is the flow divided by the number of connected MICs.

8.2.3 Flow Dependency

For measuring the dependency on the flow in the closed cooling circuit we use the Burn-MIC application. We initialize a 180 s run with a single active MIC. Additionally, we vary number of (passive) MICs in the circuit.

In Table 8.1 we see the runs that have measured by varying the frequency of the pump. The pump is driving the closed cooling circuit. The table includes the flow and effective flow of the water within the circuit. The effective flow is the total flow divided by the number of connected MICs (active and passive).

It is worth noting that the run number is ordered in such a way that we start with the lowest pump frequency and end with the highest frequency. Once we reach the pump's limit we start disconnecting (passive) MICs from the cooling circuit until the active MIC is the only one remaining.

During the measurements the cooling circuit is heating up, which is an effect that would be suppressed by an ideal regularization. Since the parameters of the heat exchanger in the test circuit are not optimized for such measurements, we have to take the heating of the cooling circuit into account.

In Figure 8.8 we cannot distinguish between the different levels of flow, just by looking at the different lines. Instead we also have to consider the current circuit temperature. We take the inlet temperature T_i and the outlet temperature T_o of a measurement at a later point in time. Since water flows from the measured point of T_i to T_o, with a heat source between, we have

$$T_o(t) = T_i(t) + \delta T(t), \tag{8.1}$$

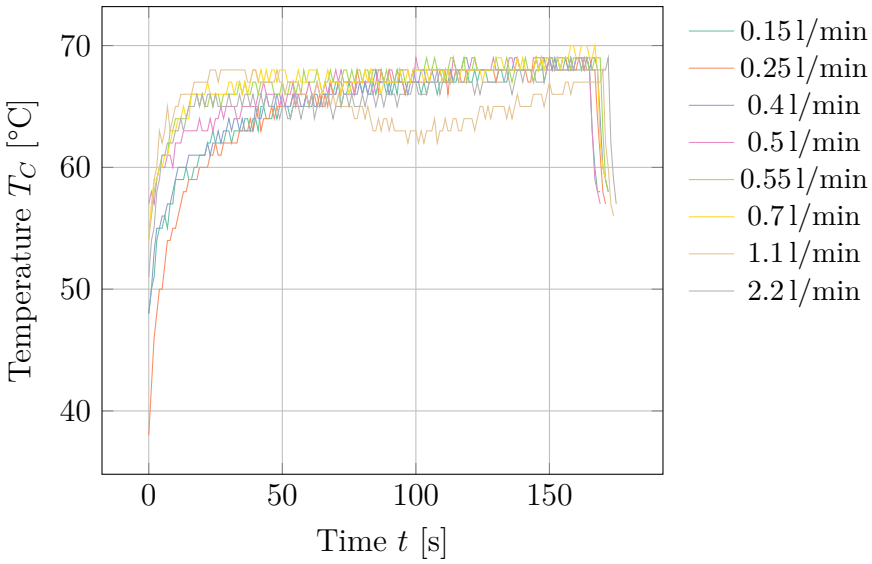


Figure 8.8: The core CPU temperature during the different runs. Run 7 (1.1 l/min) is spoiled due to over-steering of the heat exchanger. Here the maximum temperatures of the runs seem to be degenerate.

where $\delta T \geq 0$ is the rise in temperature between the two points of measurement. The time parameter t indicates the time of measurement.

That picture alone would not be very conclusive. Two measurements at the same time could lead to $\delta T \leq 0$, which is certainly wrong. The issue here is that both, heat propagation and the velocity of the water, are too small for instant changes. Since the heat propagation is effectively smaller than the velocity, a good approximation is to only look the velocity of the water in closed circuit. We can calculate it by taking the area of the hose A and the current flow F, which is the transported volume V over time t. We have

$$v = F/A, \tag{8.2}$$

which can be used, together with the distance d, between the two points of measurement, to compute the time difference Δt. Finally, we can correct Equation (8.1). We find

$$\delta T(t) = T_o(t) - T_i(t - \Delta t), \qquad \Delta t \equiv (d\, A/F), \tag{8.3}$$

where d is the length of the hose between the two points of measurement.

Usually, we are more interested in ΔT, which is the difference of the CPU temperature T_C to the inlet temperature T_i. We only need to adjust the previously

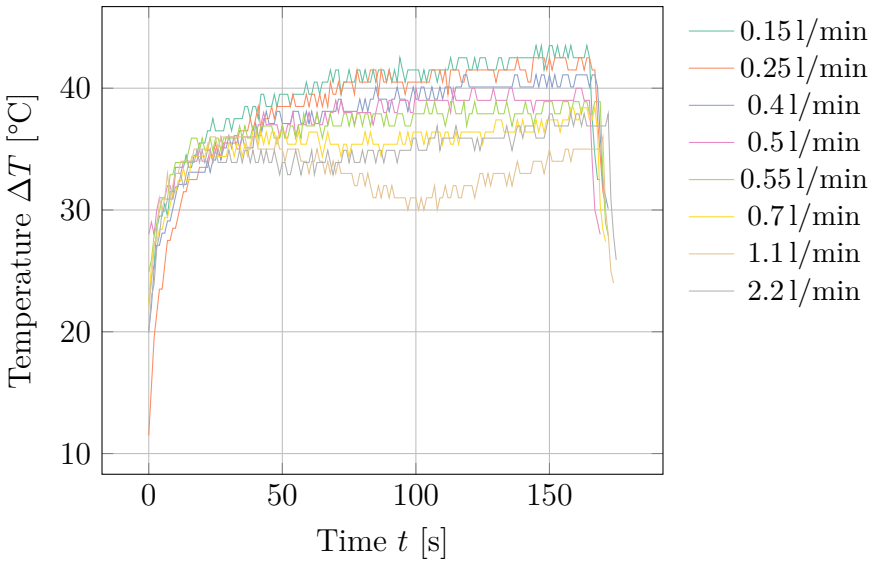


Figure 8.9: The difference between the inlet temperature T_i and the CPU temperature T_C. We can observe the effect of varying the effective flow F_{eff} in the cooling circuit by considering it in the calculating of ΔT.

derived equation Equation (8.3). We obtain

$$\Delta T(t) = T_C(t) - T_i(t - \Delta t). \tag{8.4}$$

Here Δt is calculated as before, with d being adjusted to the length of the hose between the inlet temperature measurement and the CPU measurement. Using ΔT rather than T_C we obtain the plot shown in Figure 8.9. The different flow numbers can be distinguished, which is not possible if we consider T_C alone as shown, e.g., in Figure 8.8. It is necessary to recalculate Δt per run, as it depends on the flow F, which may change between different runs. From the data we can infer a linear dependency on the effective flow, which represents the volume over time that runs through a single MIC co-processor.

Ideally, the circuit would have a constant temperature reading at the inlet sensor, i.e., $T_i(t) \equiv T_i$, for all t. Nevertheless, it is still possible to perform a cross-check using the data. For instance, we can measure how much power is actually converted to heat. Burn-MIC gives us a nice plateau for the CPU temperature, which can be used in conjunction with the current power consumption. We can apply Equation (8.3) to calculate δT, which yields the temperature difference between inlet and outlet for a measurement of T_o.

However, the difference δT is only part of the story. From the plot in Figure 8.10 we can derive that the diffusion equation has to be considered as well. The rapid changes in T_i do not represent the temperature as it would be measured shortly before the MIC co-processor. In a simple approximation we could take the moving

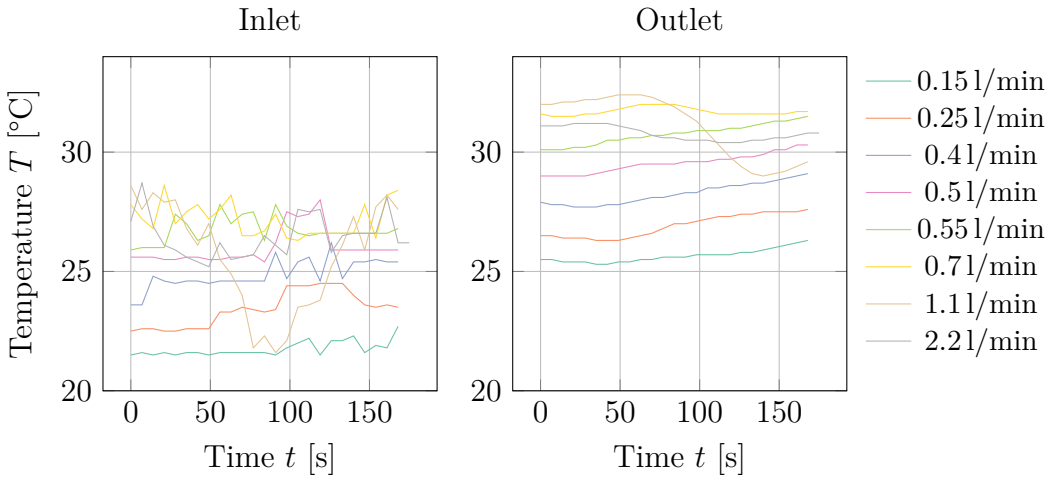


Figure 8.10: Comparison of the circuit inlet temperature (left) and the circuit outlet temperature (right) during the runs. The inlet measurements look rough compared to the outlet measurements. This behavior is due to the regularization by the heat exchanger, which directly influences the inlet temperature.

average in an interval of $\mathcal{O}(10)$ seconds to get a better estimate for the temperature before the MIC. As the outlet temperature readings are quite smooth, no special technique has to be applied here.

The dissipated power can now be calculated by considering the effective flow F_{eff}, the heat capacity of water C_W, the density of liquid water, and the previously derived temperature difference δT. We have

$$\Delta P = F(t)\, \varrho\, C_W\, \delta T(t). \tag{8.5}$$

We find that the dissipated power is approximately equal to the measured power consumption. Thus, nearly all energy is transformed to heat and absorbed by our cooling mechanism.

In the end, however, the efficiency of hot water cooling is dominated by ΔT as outlined. The dependency of ΔT on the (effective) flow F_{eff} is quite important. In general we save energy (and money) by lowering the pump frequency, which reduces the flow in the cooling circuit. The critical question is: What flow is required to reach a satisfying value of ΔT?

The plot in Figure 8.11 shows the measurements of ΔT. The calculation uses Equation (8.4). The flow dependency can be computed with a simple linear fit, however, this strategy is only valid within a certain range, which is why we exclude the data point at 2.2 l/min. The main problem is that we hit a plateau at an effective flow. Nevertheless, since we can vary the pump within specific boundaries, the resulting fit is convincing for the potential linear region.

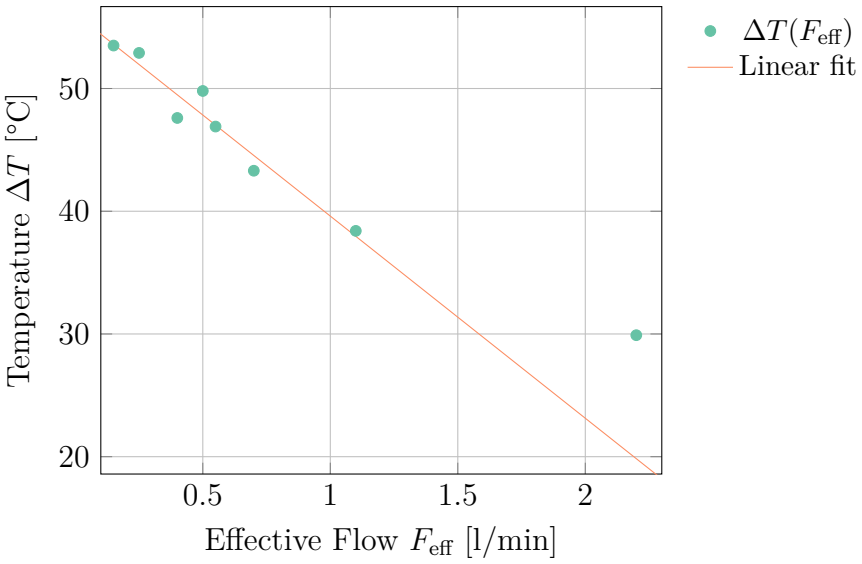


Figure 8.11: The dependency of F_{eff} on ΔT. The linear fit contains an offset 56.074 °C with a slope of 16.470 °C/(l/min). The data point at 2.2 l/min has been excluded from the fit.

The cooling solution seems to work within our boundaries and yields satisfying performance. A related important topic is the optimization of the regulation parameters. Optimizing these parameters to improve the output of the heat exchanger is critical for the overall performance. The CPU temperature of the MIC co-processor T_C is more affected than ΔT. It is desired to provide a nearly constant temperature in the cooling circuit.

The cooling circuit itself has been designed and maintained by Stefan Solbrig. The mechanical design for cooling the components has been created and implemented by Eurotech in Italy. Fine corrections have been applied by the machine shop of the physics department at the University of Regensburg.

8.3 Influence of the Thermal Grease

The die area of the KNC is connected to the interposer via a massive copper block. Choosing the right thermal compound for the copper-aluminum and copper-die connection is crucial. One of the reasons is that the thermal grease has to have the lowest possible electrical conductance, yet very good thermal conductivity. Even if we would choose not to use the same type of grease for the interjacent layer between the die of the co-processor and the copper block, we demand such properties. One reason is the liquid form of the grease in the field, which may result in forming short circuits if electrically conductive.

We choose three different thermal compounds as suitable candidates. Initially

	PK-1	KP98	WLPK 10
Manufacturer	Prolimatech	Keratherm	Fischer
Consistence	hard pasty	soft pasty	pasty
Density (at 20 °C)	$3.2\,\mathrm{g/cm^3}$	$2.2\,\mathrm{g/cm^3}$	$1.4\,\mathrm{g/cm^3}$
Thermal conductivity	$10.2\,\mathrm{W/m\,K}$	$6\,\mathrm{W/m\,K}$	$10\,\mathrm{W/m\,K}$
Dielectric strength	$3\,\mathrm{K\,V/mm}$	$1.5\,\mathrm{K\,V/mm}$	conductive
Price (May 2015)	1 €/g	0.2 €/g	2 €/g

Table 8.2: Comparison of three thermal compounds.

the problem with electric conductivity may not seem as severe as it actually is. Therefore we include WLPK for comparison, even though it is too liquid to ensure no unwanted charge transmission. The main comparison therefore takes place between KP98 and PK-1. Especially the latter has excellent values. The maximum electric field that it can withstand under ideal conditions without experiencing failure of its insulating properties is quite high and it has a higher thermal conductivity than the other candidates.

In Table 8.2 we compare the three candidates directly. For our purposes WLPK is definitely not the ideal choice. It is conductive, its viscosity is too low, and it is the most expensive. Hence there are not only technical, but also economical reasons to use a different thermal paste. Possible alternatives can be found in two very different products. While PK-1 excels in many ways, KP98 has a more suitable consistence combined with an economical advantage.

To get information about the impact of the thermal grease, we can use the different greases in interposers. We can try combinations, such as placing KP98 between the copper block and the interposer, with PK-1 connecting the die of the co-processor to the copper block thermally. We label this combination M198 to reflect the compound.

Measuring the performance of the different thermal pastes demands fully operational units, i.e., fully assembled interposers with roll-bond plates and Xeon Phi co-processors. Every unit is then benchmarked in our test cooling circuit. For the test we perform a 180 s run of Burn-MIC. We measure all temperatures including, but not limited to, the inlet and outlet temperature of the cooling circuit, and the CPU temperature of the KNC. This allows us to compute our quantity of interest ΔT using Equation (8.4), where a small value of ΔT corresponds to good thermal conductance.

We include WLPK and M198 for comparison only. For both the number of samples is statistically insignificant. The focus definitely lies on a decision between

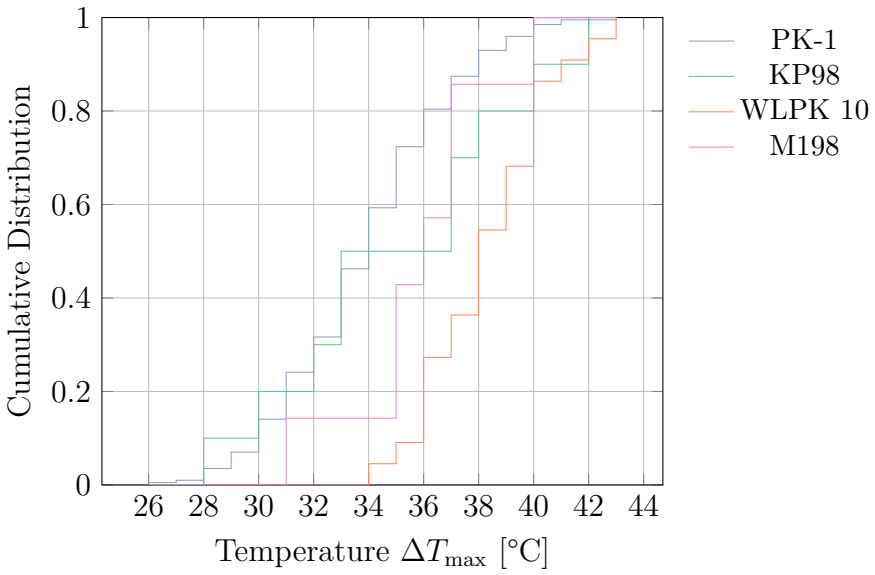


Figure 8.12: Cumulative distributions of the different thermal compounds. We averaged over all samples for the given heat sink pastes. The maximum value for ΔT has been computed for each sample (denoted by $\Delta T_{\max}$).

KP98 and PK-1. For a number of reasons it seems that PK-1 is the ideal choice. The gap to alternative solutions is just too large to reason about different pastes.

The cumulative distribution in Figure 8.12 yields both, the lowest and highest achieved temperatures. The best solution has to show the largest area, i.e., the integration over the cumulative distribution function within our temperature boundaries yields a larger value. We can see that both PK-1 and KP98 seem to be suitable candidates. Nevertheless, while the head of all samples has been sufficiently good for both, the tail of PK-1 got much more stable results than KP98. It seems that using KP98 is a very fragile process. The cumulative distribution for PK-1 reaches 1 earlier than the cumulative distribution for KP98.

Minimizing the risk of assembling parts, which do not perform thermally as expected is important. We conclude that some of the differences we observe can be assigned to production differences of the roll-bond plates. Here we have a lot of potential problems, e.g., the process of gluing the plate to the interposer has to be done manually. This practically makes it hard to have a perfectly flat surface with a uniform thickness. Choosing the optimal paste alleviates the problem.

8.4 Brick Assembly: Tubes and Clamps

One of the most crucial aspects in the design of components that are exposed to water cooling is to prohibit potential leaks. The roll-bond plates are made of aluminum, which has very good corrosion resistance. Nevertheless, we need to

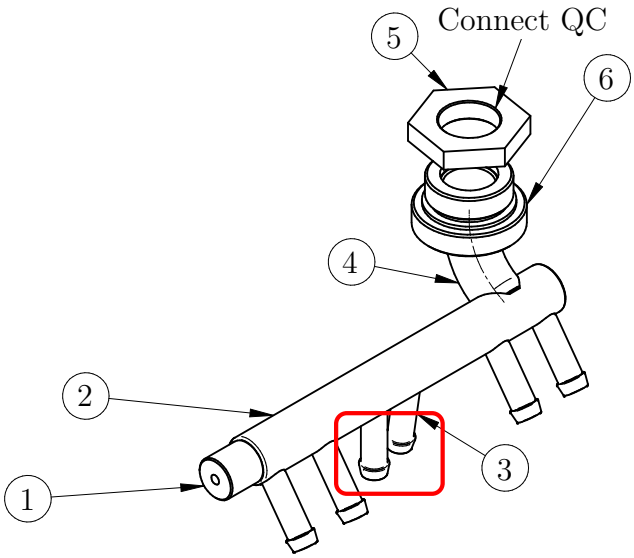


Figure 8.13: Schematics of the lower manifold. The noted points are (1) screw thread for fixing the manifold, (2) distribution pipe, (3) connector with single barb, (4) connection to the quick coupling (QC), (5) sealing ring, and (6) screw thread for sealing. The red rectangle denotes the area where the design of the lower manifold is significantly different compared to the upper manifold.

mix anti-corrosive additives into the cooling water to minimize possible corrosion effects. Additionally to the corrosion inhibitor[1], we need to mix in a biocide substance[2] acting as an antimicrobial enhancer. This is supposed to counter-act the formation of bacteria, which may result in a decrease of water flow through individual components or damages in the piping system. In the worst case leaks could occur.

A brick is connected to the cooling water of the rack via so-called manifolds. There are two manifolds: one for the water inlet (lower) and one for the water outlet (upper). The manifolds are connected to the water circuit of the rack via quick couplings. A schematic of the lower manifold is shown in Figure 8.13. The main task of the lower manifold is to distribute the incoming water to all components. The upper manifold is then aggregating the outgoing water from all components. The two manifolds follow slightly different (mirrored) schematics. The upper manifold contains a much larger angle between the two central connectors (see red rectangle in Figure 8.13).

The roll-bond plates are connected to the manifolds via flexible tubes. This has the advantage of allowing tolerances in the construction, which is desired for a

[1] We can use Kebocor 204N or Clariant Protectogen C Aqua.

[2] We choose Clariant Nipacide BIT 20.

prototype-like machine. We use ordinary EPDM[3] tubes with an inner diameter of 6 mm and an outer diameter of 10.5 mm. The connections are sealed with special ear clamps[4].

The EPDM tube is very flexible and needs to be supported to withstand sudden pressure rises. Normally, two layers of EPDM would be used, however, for a wall thickness of roughly 2 mm this is not possible. Instead, we can use an external textile fabric that covers the tube. In order to fix the fabric onto the tube we need another layer on top. Initially, a thermo-shrinkable tube may seem like a good idea, however, a polyamide film in form of Kapton tape is more suitable.

One of the main issues with thermo-shrinkable tubes is their long-term behavior. In the beginning they are shrunk to completely wrap over the tube, however, while the tube stays soft in the long run, the thermo-shrinkable tube will change its properties over time. Thermo-shrinkable tubes will get more rigid and tend to break if exposed to UV light.

The Kapton tape remains stable across a wide range of temperatures, from −269 °C to 400 °C. It does not require external heat during assembly. Most importantly it seems not to act against the soft tube, but only to support it. This is reason enough to believe that Kapton tape is a good solution as middle layer between the clamp and the textile fabric over the tube.

Testing the long-term behavior is not possible in our case. Instead, we require a test that gives us immediate feedback if a solution should be considered or not. The following procedure represents an extreme test that is useful in the evaluation for determining the most adequate solution.

1. Fill the brick with hot water (close to 100 °C) at 2 bar.
2. Raise the pressure to 10 bar with hot water.
3. Wait 5 minutes.
4. Connect it to the cooling loop with 13 °C operating at 2 bar.
5. Disconnect it and raise the pressure to 10 bar with cool water.

Solutions with no layer over the textile fail in a similar fashion as solutions that use the thermo-shrinkable tube. They all start leaking in the final pressure test, i.e., before the 10 bar are reached. The Kapton tape, however, is able to reach the 10 bar and remains stable for some time. Eventually, even bricks that use the Kapton tape start leaking, but with very small drops. Later on the bricks with

[3] Short for ethylene propylene diene monomer (M-class) rubber.

[4] Our choice is Oetiker PG 167 706R.

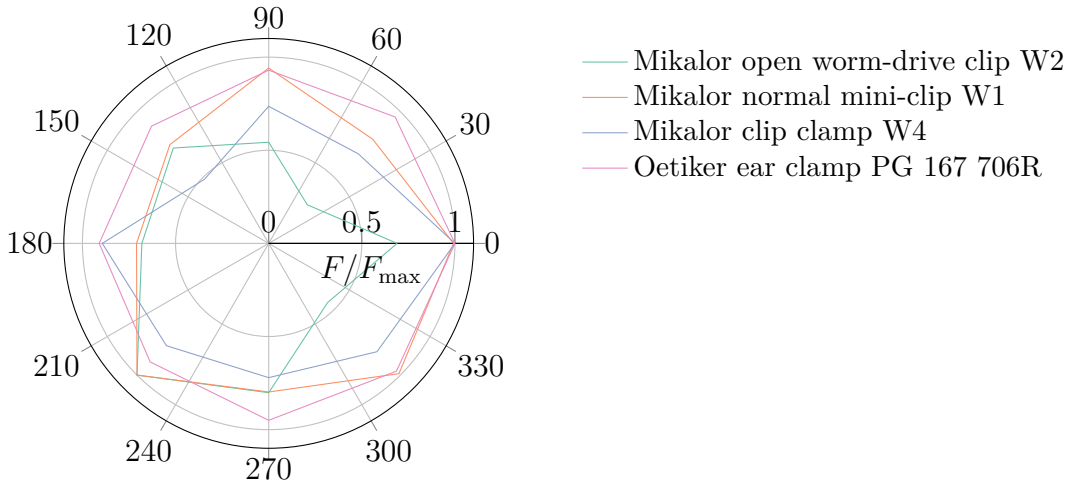


Figure 8.14: Comparison of the effective force on the tube with four different clamps. The force F has been normalized with respect to the maximum force $F_{\max}$ of the respective clamp. Data acquired from [100].

Kapton tape seem to be stabilizing again. The difference is marginal, but reason enough to prefer the Kapton tape over the other solutions.

The greatest influence on the leak probability has the choice of clamps. The origin of this problem lies in the outer diameter of the tube. For instance, by using 12 mm reinforced PVC tubes we observe no leaks with a wide range of possible clamps. However, replacing the 12 mm reinforced PVC tubes with soft PVC tubes featuring smaller outer diameters leads to leaks in conjunction with most clamps. The reason is that most clamps do not convey pressure uniformly enough to tubes with smaller diameters. While thicker walls would solve some of these issues, they would render the assembly procedure hard or even impossible. A thicker wall means a less flexible, i.e., more stiff, tube, however, we require a certain bending radius in our setup.

In Figure 8.14 we list some of the different clamps we considered. Besides the Oetiker clamp [78] no other solution seems to be applicable. Next to the most suitable force distribution we are able to assemble the clamp in a fast, yet reliable, manner. This is very important as we need to install 24×64 clamps in total. The open worm-drive clip has to be assembled using a screwdriver. The same is true for the mini-clips, which have not one, but two rails. This leads to a more balanced distribution with to a minimum efficiency of 70 %, compared to the 30 % of the open worm-drive clips. The clip clamp is similar to the Oetiker solution, however, does not feature such a sophisticated system to balance the force. The weakest point gives us 50 % efficiency.

The only disadvantage of the Oetiker clamp is that the clamp will be permanently deformed during the assembly procedure. However, this is only a minor

drawback considering that the manifolds and roll-bond plates contain single barb connectors. In contrast most professional water installations use three barb ridges. The Oetiker clamp is designed to work well on flat surfaces. Overall it seems to be the ideal choice for our purpose.

8.5 Thermal Performance of a Brick

Once the thermal performance of individual components, such as KNC or IB cards, is known, we can perform integration tests. An integration test combines several components. For us the most important integration test benchmarks a complete brick. In addition to the four KNCs, the IB card, as well as Juno and Mars we also have effects coming from the high density due to heat radiation. The components are enclosed in a metal box ("housing"), which yields larger value for ΔT anyway, since heated air is captured inside.

We are mainly interested in three things:

- What is T for single bricks in idle mode and during computation?
- What is the thermal long-term behavior of the QPACE 2 bricks?
- What is the thermal distribution over all bricks under equal load?

The distribution is particularly interesting, since it allows us to find potential problems. Such problems may be related to differences among the bricks or may be an indicator that the flow in the rack is not uniform. The latter has to be examined closely, as the rack is not using the so-called Tichelmann principle [145]. In case of QPACE 2 a quick calculation indicates that the expected effect of not following the Tichelmann principle is small. The additional flow resistance is practically negligible.

Each brick comes with a number of different temperature sensors. The sensors are placed on different components, which are located in different places inside the brick. Some sensors might be more sensitive to the ambient temperature, others might be directly influenced by the cooling circuit's temperature. Hence it is hard to find a single number that represents a brick's temperature in every scenario. Instead, we just pick the temperature from the sensor reading the highest value.

The reasoning behind the assumption that the maximum temperature is a good representative for the brick, is fairly straightforward: If a single brick seems to perform worse than others, even though only a single sensor is reading higher values, the brick still needs to be examined more closely. In the end a device is not cooled sufficiently or the sensor is actually broken. Both scenarios demand

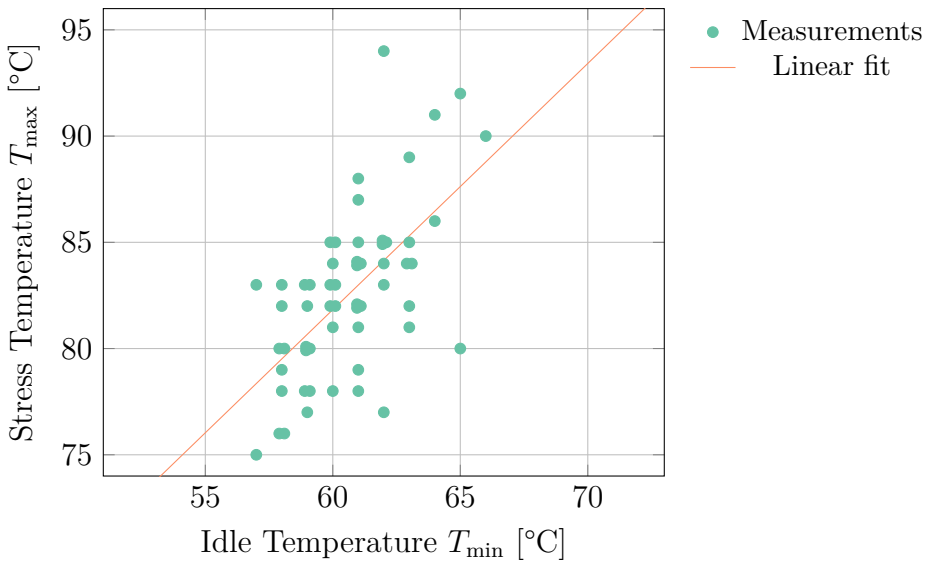


Figure 8.15: Correlation of idle $T_{\min}$ to stress temperatures $T_{\max}$ for 58 bricks. The temperatures of the idle benchmarks have been averaged over 1 h. The stress test consists of a 10 min HPL run. The bricks have been running in idle mode for at least 2 h before any measurement started.

maintenance. We will use the measured absolute temperatures exclusively for a comparison between the bricks.

Unfortunately, a prototype production process contains many steps that may result in decreased quality. As a result we see a fragmentation of (thermal) performance over all bricks. The variation can be visualized by plotting the maximum stress temperature $T_{\max}$ of each brick in correlation to its maximum idle temperature $T_{\min}$. In the ideal case all bricks would be placed at the same point, or at least at the same (horizontal) line. The plot in Figure 8.15 illustrates the fragmentation. We see a variance in both, the idle and the stress temperature.

Even though higher idle temperatures yield higher stress temperatures in general, it does not have to indicate a problem. For instance, one of the bricks with the highest idle temperature does not cross the 80 °C line for its maximum stress temperature. On the other side the highest measured temperature during stress has only been slightly above average in idle mode.

Using the fit shown in Figure 8.15 we can identify the trend of obtaining higher stress temperatures for bricks with higher idle temperatures. Thus we already care about low idle temperatures, even though this is by no means a guarantee for acceptable stress temperatures. In the end we demand that bricks do not exceed a certain threshold temperature during stress.

Measuring the thermal performance during idle mode is straightforward. We simply measure all bricks during operation without performing actual work. In order to measure the bricks with load we alter the measurement procedure. To

minimize radiation effects we only run on 1/4 of the bricks simultaneously. We use a single brick per row per side of the rack. The brick position is the same in every other row and shifts 2 positions in the remaining rows. The sequence for the first run is therefore given by, e.g., $1-3-1-3-\ldots$, where the number denotes the position in the respective row.

Algorithm 5 Stress Measurement

$F \leftarrow (40, 55, 38, 53, 36, 51, 34, 49)$ ▷ initial values
$B \leftarrow (16, 23, 14, 21, 12, 19, 10, 17)$
for $i \leftarrow 1$ **to** 4 **do**
    for $j \leftarrow 1$ **to** 8 **do**
        start job on node F_j ▷ front of rack
        $F_j \leftarrow \begin{cases} F_j > 56, & F_j - 24, \\ F_j \leq 56, & F_j + 8 \end{cases}$
        start job on node B_j ▷ back of rack
        $B_j \leftarrow \begin{cases} B_j > 24, & B_j - 24, \\ B_j \leq 24, & B_j + 8 \end{cases}$
    end for
    wait for all jobs to finish
end for

The algorithm shown in Algorithm 5 maximizes the distance between the bricks, while keeping it constant for each run. The outlined scheme is efficient as we can measure the thermal performance of each brick during stress in just 4 iterations. The initial values of 40 (front) and 16 (back) have been chosen arbitrarily with the constraint that both values have a distance of 24.

Thermally, the ideal solution would have been to only use a single brick at a time. The provided scheme is a good compromise between the required time for all measurements and minimizing ambient heating.

Overall the thermal performance fulfills all of our requirements. Since the thermal performance was measured during summer, we do not expect any problems related to overheating due to design issues during the whole year.

9 Applications on QPACE 2

9.1 Available Applications and Tools

In the previous chapter we introduced some applications for benchmarking the thermal performance. These applications and tools come in very handy during the development process. In general high performance computing requires more than just programming skill. In order to get as close to the theoretical peak performance as possible, a programmer needs tools to inspect possible bottlenecks. The detection of such bottlenecks can be simplified with a tool like Intel vTune. For our purposes there are two major kinds of profilers, statistical profilers and instrumenters.

While statistical profilers work without modifying the original source code, an instrumenter needs to modify the original source to insert instructions, which will be used to identify the function callee and caller. The advantage is that instrumenters can be very fine grained, however, they are more complex and they require the original source. They do not work well with third party libraries. Some instrumenters can work without the original source, but they usually have other side effects.

A statistical profiler is based on sampling. It probes the application's counter at regular intervals using OS interrupts. Such profilers are typically less accurate and specific, but allow the application to run at near full speed depending on the sampling frequency. The resulting data are not exact, but a statistical approxima-

tion. Each sample represents a copy of the application's stack. These copies are stored in a database, which will be evaluated in the end. The stack dump includes information about the whole call tree at the current point in time.

A statistical profiler is able to show some statistics about the distribution of any function calls among all measurements. It is supposed to work, since, e.g., a measurement every ~ 1000 cycles still results in a million measurements per second. The information gain is highly dependent on the chosen sampling frequency. While a high frequency leads to a more accurate result, it will severely decrease the application's performance. Finding a sampling frequency that represents a good compromise between accuracy and performance is therefore crucial.

In practice, sampling profilers can often provide a more accurate picture of the application's execution than other approaches, as they are not as intrusive to the application, and thus do not have as many side effects. Side effects on memory caches or instruction pipelines may spoil the result and should be minimized. In general there is no profiler that does not have any influence on the overall performance.

A good profiler for the Xeon Phi is the software Intel vTune. The application features a project based profiling process that can be evaluated on the command line or in a graphical user interface (GUI). Only the latter makes a detailed analysis possible, since the command line interface (CLI) has some severe limitations. The GUI offers us to investigate with the whole range of gathered data. We can look at memory allocations, function invocations, and cache misses. Most importantly we see, how much time is spent in the different parts of the application.

Another useful set of tools is LIKWID [143]. These helpers are ideal for reading hardware counters and investigating the performance of applications using multithreading. They are especially convenient to investigate the effects of pinning, running micro-benchmarks, or receiving information about the current architecture's thread and cache topology.

9.2 Compilers and Frameworks

Writing a program for the Xeon Phi can be done like on most other x86 platforms. There are only two requirements: We need a text editor (e.g., `vi`) and a compiler for the language of choice (e.g., *C*).

Even though such a setup is working with general purpose software like a compiler from the GNU compiler collection (GCC) (e.g., the program `gcc`), a more specialized tooling is definitely desired. For the Xeon Phi the Intel C compiler (ICC) in conjunction with the `icc` application is considered the optimal choice.

9.2.1 The Intel Compiler

There are many reasons for preferring the icc over the gcc program. From the perspective of HPC it is proven that the Intel compiler is able to perform the best optimizations, especially in combination with Intel hardware. A programmer might argue that the Intel compiler is currently the only way to access the three different execution modes of a Xeon Phi accelerator card.

The three different execution modes are:

Direct Executes the program is directly on the co-processor.

Offload Loads the data from the host, but splits the execution between host and Xeon Phi [41]. This is sometimes called *hybrid* execution.

Native Loads the program from the host to run it on the accelerator. This requires the `micnativeloadex` application, which is included in the Manycore Platform Software Stack (MPSS).

Another feature of the Intel compiler is the possibility to enhance the code with so-called compiler *intrinsics*. These special functions map directly onto mostly a single instruction[1]. The purpose is to avoid mixing in assembly code. Additionally, the usage of intrinsics gives the compiler more information about our code. This should make the application more portable and prevent undetectable errors. The compiler can then insert special assembly instructions or perform further optimizations, which would not have been possible by inserting raw assembly code.

9.2.2 Compilation Flags

In order to compile for the MIC architecture using icc, we need to use the `-mmic` flag. The resulting assembly is binary compatible with the CPU of the Xeon Phi. For efficiency reasons, we should specify additional flags.

In most cases the highest optimization level (`-O3`) is the best, however, in general it is not reliable. Therefore, a comparison of the results of an application compiled with `-O3` with results of the same application compiled with `-O2`, or even `-O0` is strongly recommended. The second optimization level is concerned with speed, i.e., `-O2` introduces auto vectorization, loop optimizations, intrinsic inlines, and intra-file inter-procedural optimizations. The `-O3` level adds some more aggressive loop transformations and collapsing (`if`) statements on top. There is a chance that `-O3` is not only unreliable, but even slower than `-O2`.

There are other interesting flags. We list them briefly:

[1] in some cases they are mapped onto several instructions

`-no-prec-div` Division optimization, however, with loss of accuracy [32].

`-ansi-alias` Big performance improvements possible, if the application adheres to ISO C standard alias ability rules.

`-ipo` The interprocedural optimization works between files to perform optimizations. This definitely increases the required compilation time.

Another possibility that has not been listed previously is the ability for profile-guided optimization (PGO). PGO improves the performance by reducing branch predictions, shrinking code size, and thus eliminating most instruction-cache problems. We start by compiling our application with the `-prof-gen` and the `-prof-dir=`p flags, where p is the path to the profile that will be generated. After running the application we can compile our application again with `-prof-use` instead of `-prof-gen`.

9.2.3 Offloading Work

While the direct and native execution modes rely on the `-mmic` flag, the offloading programming model is used differently. Here we want to produce an executable that is binary compatible with the host system. Therefore, we would specify, e.g., `-mia32`, to generate OS and device specific binaries for the host.

The generated binary already includes special offloading sections, that have been compiled for the MIC. These sections are then packaged into a separate executable, which is contained within the real executable. The communication with and the lifetime of the worker executable is integrated by the compiler. We only need to care about the specific code we want to offload, and the locations of data that are exchanged.

We require two code changes for providing the compiler with sufficient information to create binaries with offload capabilities. We need to give the function(s) we would like to use on the MIC a special attribute. The snippet in Listing 9.1 shows a templated method that is set up with the correct attribute.

```
1 template<typename T, typename... TArgs>
2 static T __attribute__((target(mic))) compute(TArgs... args) {
3   /* ... */
4 }
```

Listing 9.1: Preparing a function for offloading.

In Listing 9.1 we use a variadic template to show a valid function that can be compiled. In general we do not have to use templates or variadic templates for defining functions, which allow offloading.

The first change makes it possible to offload the function on the MIC. It does not prevent us from using it on the host as well, even though the normal use-case would be to execute this part on the co-processor. This is where the second code change is required. We specify that the function call is indeed being offloaded and how input and output should be placed.

The code in Listing 9.2 outlines how an offloaded function call might look like. We can specify variable-specific transportation properties using the `in` and `out` specifiers. These properties tell the compiler to introduce further calls for copying memory from the host to the device or the other way round.

```
1 #pragma offload target(mic) \
2   in(arg: length(N)) out(res: length(N))
3 auto res = compute(arg, ...)
```

Listing 9.2: Offloading a function call.

Sometimes we might want to reuse (or overwrite) a specific chunk of memory. In such scenarios we should include the `inout` specifier, which copies the data from the host to the KNC in the beginning, and the other way round after the function has returned. The specifiers can be thought of like references in C++, where we can declare read, write, and read-write type of parameters.

Alternatively, we can use the offloading model presented within the most recent OMP 4.0 standard. All in all the required code changes follow the same concept. We need to equip the function and the call with more information. Both changes rely on an OMP `pragma` statement. This is very similar to other OMP calls.

9.2.4 Frameworks for Parallelization

No matter what programming model we choose, we have to utilize as many cores as possible to benefit from running code on the Xeon Phi. We need a good strategy that is based on a very efficient threading model. We could either start from scratch, e.g., by using a low-level API such as Pthreads, or we could use one of the many available frameworks. In this section we list and discuss the most interesting options.

We already mentioned the OMP framework, which is delivered as a compiler extension. This has several advantages, like the ability to perform efficient compile-time optimizations. OMP is integrated in the Fortran, C, and C++ compilers from Intel. Every implementation follows the same standard.

Intel gives us two more mature possibilities. One is the Threading Building Blocks (TBB) library. TBB makes heavy use of templates and is therefore only available for C++. Even though code is expanded during compile-time, the com-

piler's ability for code-optimizations is limited compared with OMP. The advantage of TBB is the richness of the library. We will most likely find every useful algorithm to distribute or create tasks that can run in parallel.

The second possibility is called Cilk Plus. This is a very simple language extension, much like OMP. The extension offers three new keywords and an improved array handling. The latter makes Cilk Plus so interesting. Automatic vectorization can be boosted by using the new array notation. It is worth noting that the latest OMP 4.0 standard allows us to write specialized vectorization statements as well. Details on these concepts are discussed in Section 9.5.

A comparison of the efficiency of these frameworks seems to be useful. In Section 9.3 we perform several benchmarks to pick the right framework for our programming efforts. We use the chosen framework to create a proper synchronization barrier. Finally, we gather a range of tips and tricks for obtaining great performance on the KNC.

9.3 Multi-Threading

Two laws are dictating the developments in the HPC industry. The first is Moore's law, which states that the number of transistors grows exponentially over time [77]. We can use this to predict the number of transistors per die in the near future. However, the end of Moore's law is inevitable, as we are approaching the atomic scale, where effects from quantum mechanics will become dominant. The second law has been identified by Amdahl [9]. It yields an estimate about the scaling behavior of applications.

9.3.1 The Need and Cost of Parallelization

A decade ago Moore's law was a synonym for increasing CPU frequency. The main problem with this development was the additional power requirement, mainly for the dynamic power consumption, which is given by

$$P \propto U^2 f, \tag{9.1}$$

where U is the CPU voltage and f is the frequency. The overall power is consumed by the short-circuit and leakage current. While the latter is dependent on the CPU supply voltage, the other two are dependent on the clock frequency. It has been shown that an optimal CPU frequency with minimal energy consumption exists [35]. Further scaling in frequency is therefore unwanted.

Until this energy crisis parallel programming was mostly about using multiple machines connected by some high speed interconnect networks. This is still im-

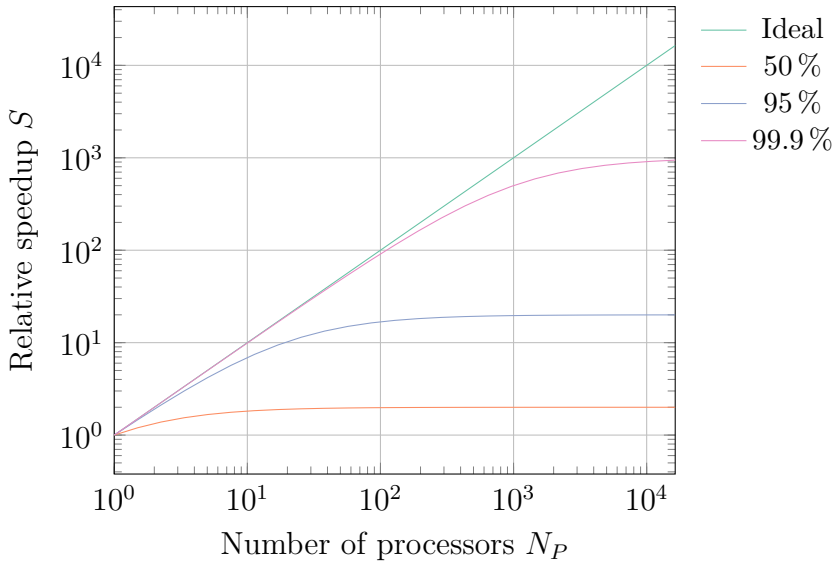


Figure 9.1: Illustration of Amdahl's law for $P \in \{0.5, 0.95, 0.999\}$. The maximum speedup is always limited by the inverse sequential portion, i.e., $(1 - P)^{-1}$. The ideal line $S \sim N_P$ for $P = 1$ is also shown.

portant, but by far not the only source of parallelism. Now programmers need to be aware of at least two levels of parallelism: Multiple machines connected by a network and multiple cores connected by the memory.

In the worst case the cores can only access data placed in the RAM. In the best case two cores share the L1 cache, which is the closest to the CPU's registers. Additionally, some CPUs offer simultaneous multi-threading (SMT), which lets a single core be used like multiple cores. This allows us to run multiple threads on a single core without requiring the scheduler provided by the OS.

The increasing number of cores brings us directly to Amdahl's law. We can compute the speedup S that an application would gain by using parallelization. For this we need to determine the parallel portion P and the number of workers N_P. We calculate

$$S = \frac{1}{(1 - P) + P/N_P}, \qquad \lim_{N_P \to \infty} S = (1 - P)^{-1}. \tag{9.2}$$

The main issue is that P is usually too small. Even though N_P is steadily growing we might end up with limited speedup. For an application that consists of $P = 0.95$, i.e., 95 % parallel portions, we can still only achieve a maximum speedup of 20. The plot in Figure 9.1 shows the speedup according to Equation (9.2) for various parallel portions P.

The equation for Amdahl's law is not very realistic, though. It neglects overhead, which unfortunately cannot be avoided in practice [138]. Even an embarrassingly parallel problem needs to gather sub-results at some point. Therefore

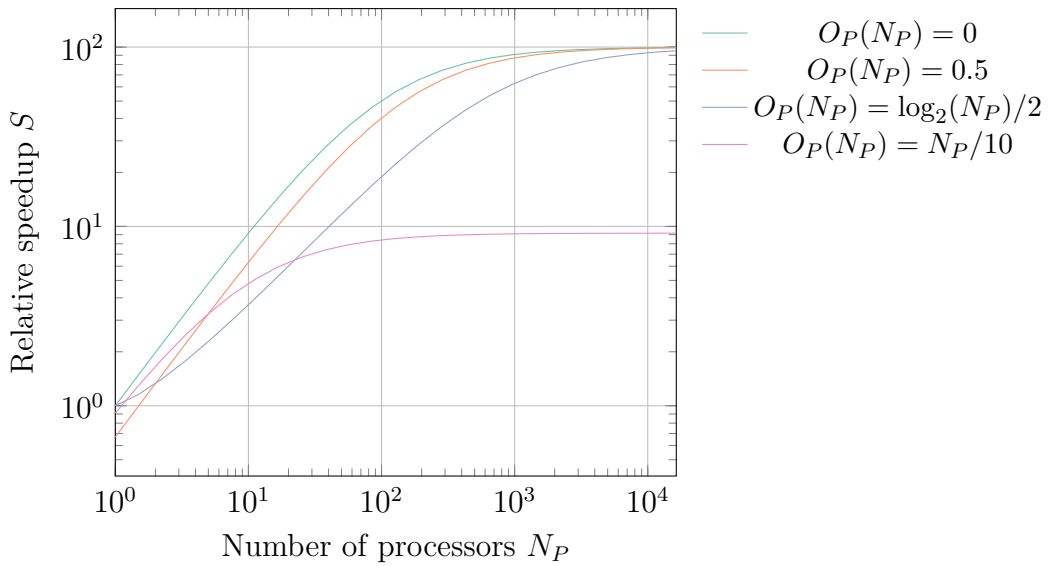


Figure 9.2: Amdahl's law with overhead for $P = 0.99$ with $O_S = 0$ and various choices of the parallel overhead in dependency of the number of workers $O_P(N_P)$.

no application can be fully parallelized, i.e., any practical computation requires synchronization at least once [72].

Given that parallelization is not free we should compute the speedup by using

$$S = \frac{1}{(1 + O_S)(1 - P) + (1 + O_P) P/N_P}, \tag{9.3}$$

where O_S is the overhead in the sequential fraction, e.g., for setting up data structures or computing parameters for parallel computation and O_P is the overhead in the parallel fraction, which is usually dominated by communication cost. In general these two parameters are dependent on the number of processes N_P. Reducing this overhead is one of the primary goals for effective parallel programming.

If we neglect the overhead in the sequential fraction, we can focus on the communication and synchronization overhead instead. In Figure 9.2 we see the dependency of the speedup on the parallel overhead. If the overhead would be independent of the number of workers, which is usually not the case, then we could alleviate the situation by bringing in more workers. Otherwise we see that for constant overheads $O_P = c$ we practically lose only a small fraction in performance. Logarithmically scaling overheads can be overcome quite quickly and usually do not interfere with the scaling behavior much. The real problem emerges with linearly scaling overhead. We need to identify and avoid algorithms with linear overhead or the scaling is significantly decreased.

Amdahl's law is, however, only part of the story. John Gustafson pointed out that the problems solved in parallel are larger than what is attempted on sequential systems. This is known as Gustafson's law. When we increase the number of

workers N_P, the problem size increases as well. An increased problem size usually leads to an increased parallel region P. Therefore, we also improved the scaling. Consequently as problem size increases, scaling improves, as well as relative speedup and efficiency.

Discussions that involve Gustafson's law are usually about strong vs. weak scaling evaluations. The strong scaling behavior always looks worse than the weak scaling behavior. The reason is that a weak scaling evaluation adjusts the problem size to the number of workers. The idea is to keep the amount of work per worker nearly constant. For obtaining practical information about the scaling of a algorithm it makes sense to use weak scaling evaluations.

A good example to illustrate a suitable weak scaling evaluation is a LU-matrix factorization with partial pivoting. As the matrix size N increases, the amount of memory required increases on the order of N^2 but the computation required increases on the order of N^3 and this computation runs very well in parallel. As a consequence LU-matrix factorization has become one of the most common parallel benchmark for supercomputers.

In the KNC case we deal with $\mathcal{O}(100)$ workers on a single processor. Our job is to identify which parallelization framework offers the lowest overhead, such that we can make a reasonable decision what technology to choose for writing our applications.

9.3.2 Evaluations for OMP, TBB, and Cilk Plus

Before we decide which technology [154] to choose, we have to get some experience with the available options. We need to know how these technologies behave under certain circumstances. Especially the scaling with the workload and number of workers is essential. Even though micro-benchmarks are quite often misleading, we are willing to take the risk in order to learn something.

The main goal of these measurements should be to identify potential performance and programming problems. In general, we want to stay as generic as possible, basically only running very simple, rudimentary applications, which are easy to understand.

One of the problems we should not underestimate is the overhead of having a slower thread in our worker ensemble. That might happen if one of the cores is actually shared with the OS. In such a scenario the OS will take resources from our workload, which will effectively slow all workers, since every synchronization is only as fast as the slowest core. Additionally, since synchronization is usually (partially) handled by the OS, we are actually hit twice. Once for using a slower worker, and the second time for never being able to perform the two steps, checking

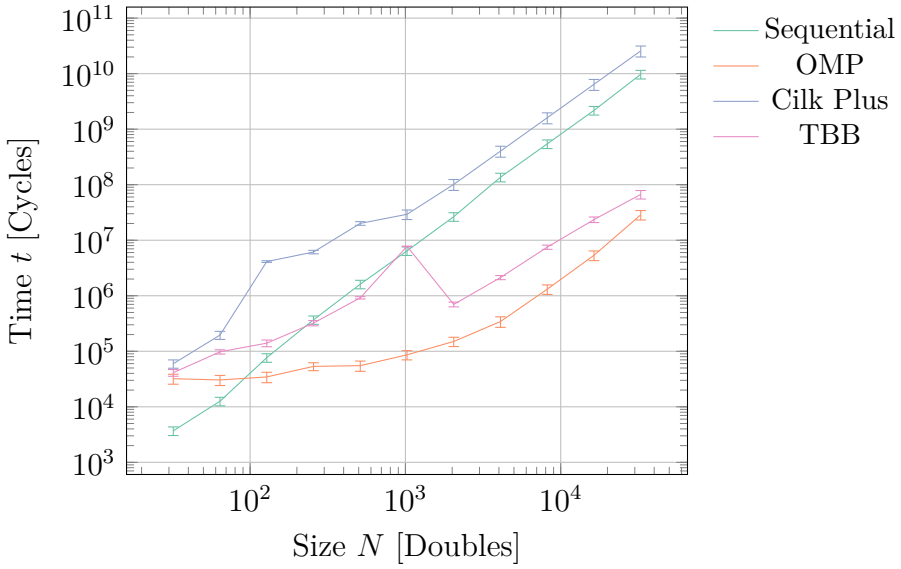


Figure 9.3: Parallelization of a dot product using double precision. Comparison of the sequential execution with available functions from OMP, Cilk Plus, and TBB).

and continuing, in ensuing cycles.

As a valid benchmark we consider a dot product. We can make use of the FMA ability of the KNC. The vectorization capabilities of each framework can be included as well. This compares a truly scalar sequential execution with a fully parallelized one. The plot for this benchmark is shown in Figure 9.3. We can see that the scheduling is probably not very efficient for small data sizes, especially in the case of Cilk Plus.

The upper bound is limited by the streaming capabilities of the Xeon Phi. A bandwidth of roughly 150 GB/s is far too less for providing the dot product with sufficient data. We need around 1.1 TB/s. The result can be explained by knowing that the basic demand is 32 bit/cycle, which has to be multiplied by the frequency f, the number of threads N_P, and the operations per cycle O. We have

$$B_W = O\, N_P\, B_R\, f, \tag{9.4}$$

where the basic demand is denoted by B_R. In our case $O \equiv 2$, $N_P \equiv 120$, and $f \equiv 1.24\,\text{GHz}$. The FMA instruction allows us to increase O from one to two.

For Cilk Plus we do not only pay the price for worker initialization as with the other frameworks, but we do not benefit from multi-threading within the shown workload. The work-stealing mechanism in conjunction with the recursive divide-and-conquer strategy [21] is not well-suited for this benchmark. A possible solution is to change the grain size, which determines the size of the chunks to distribute.

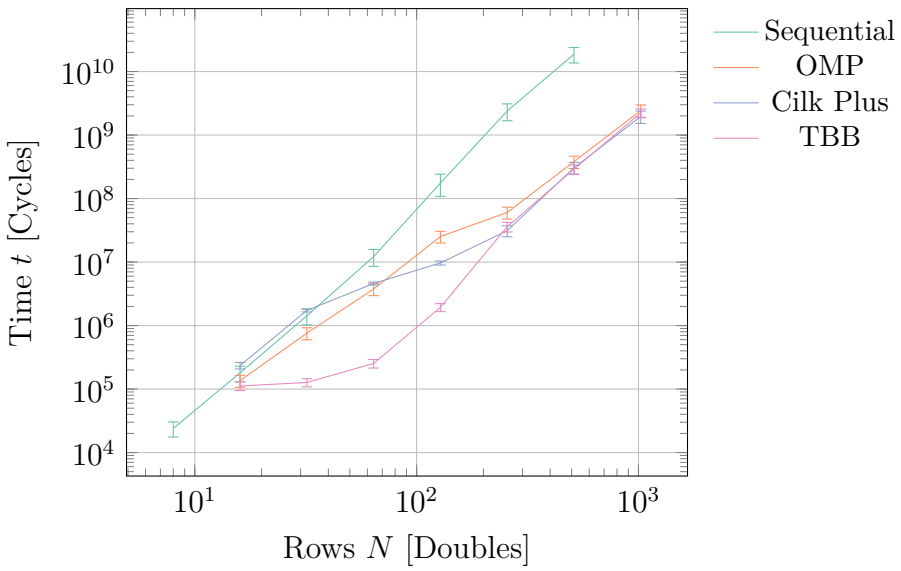


Figure 9.4: Parallelization of a matrix-matrix product using double precision. Comparison of the sequential execution with reduction methods provided by OMP, Cilk Plus, and TBB.

By default the grain size G is

$$G = \min\left(2048, N/(8N_P)\right), \tag{9.5}$$

where N is the size of the problem and N_P is the number of workers.

In general, we are constrained to use a single technology exclusively. The reason is that most threading frameworks come with their own way of managing the lifetime and state of their workers, i.e., threads. If we use multiple frameworks they might interfere with each other. For instance both, OMP and Cilk Plus come with their own runtime. The runtime handles the thread affinity and starts or stops workers according to its algorithms.

Finally, we might be interested in matrix-matrix product. The product of two square matrices with N rows is special, as it implies more computations $\mathcal{O}(N^3)$ than memory consumption $\mathcal{O}(N^2)$. In the end we really have mostly computations instead of a more likely measure of memory bandwidth. A quick check yields a required bandwidth of around 70 GB/s, which is below the sustainable bandwidth. Reusing existing cache-lines makes it possible to lower the demand by a factor 16 than in case of the dot product.

The plot in Figure 9.4 shows the behavior for a matrix-matrix product with double precision floating point numbers. The given size is the number of rows for the square matrices. As with the previously shown plot from Figure 9.3, we do not limit ourselves to threading, but also use vectorization explicitly. This way we can benefit up to a factor of 8, see Section 9.5.

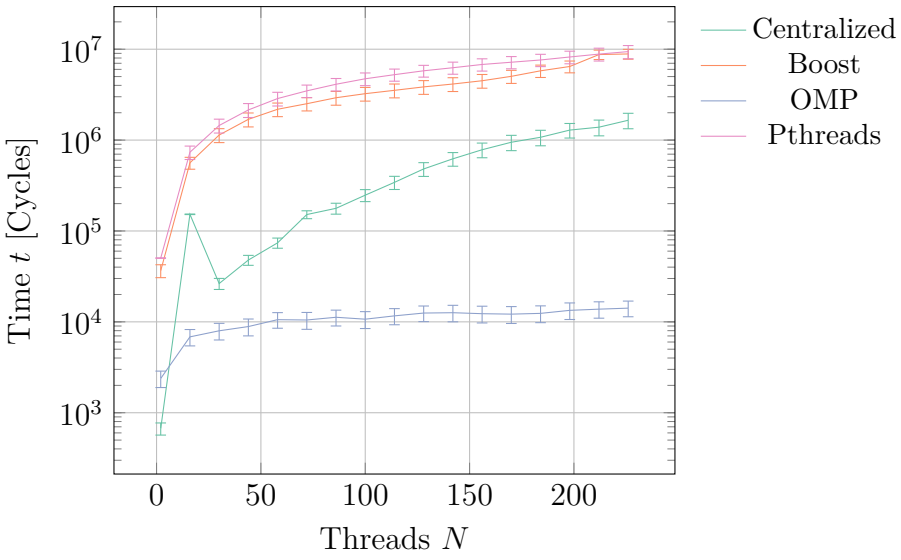


Figure 9.5: Comparison of barriers including a simple centralized barrier, the implementation used by the Boost library, OMP's default mechanism, and the barrier implementation provided by Pthreads.

In this benchmark TBB is able to hide the initialization cost most efficiently. In fact TBB shows great performance in the long run. Besides the efficient scheduler mode we have a loose coupling between the used vectorization technology and the parallelization. While the other two frameworks provide their own vectorization mechanism, for TBB we use auto vectorization from the compiler.

Overall OMP shows very solid performance. For smaller data sizes or workloads we will always have more overhead in a parallel execution than in a sequential one. This is, however, not the regime we are after. Our target data size is possibly more on the other end, where we already hit the memory bandwidth and worker initialization is negligible. Exceeding the limiting memory bandwidth is one of the most common reasons for insufficient parallel performance. Staying within the possible bounds is necessary.

We did not include MPI in these benchmarks. The main reason is that we do not want to use MPI for intra-node communication due to the increased memory consumption. Furthermore, we would like to make efficient use of shared caches and the ring bus. This excluded MPI as a viable option in this discussion.

9.4 Worker Synchronization

Synchronization is one of the most crucial parts in a multi-threaded or distributed application. We need a reliable and efficient mechanism for controlling the program flow. Naturally, every parallelization framework provides a solution that is reliable. For our purposes we need an efficient implementation.

We demand that our synchronization method is reliable and efficient enough to be no real bottleneck. Most importantly the synchronization mechanism should scale well with the number of workers. We require at least $\mathcal{O}(\log N)$ or better, with N being the number of workers.

First we need to evaluate the existing solutions to see if any available implementation already satisfies our requirements. In Figure 9.5 we show the performance of the integrated solutions provided by some popular libraries and frameworks, such as Boost or OMP.

We measure the dependency on the number of workers to get information about the scaling behavior. We can immediately see that a tuned implementation, e.g., the barrier provided by OMP, is worth considering. It uses a better algorithm than just a plain centralized barrier, which is shown for comparison. In fact most barriers use an algorithm that scales logarithmically with the number of workers.

The benchmark uses a scattered worker distribution. Such a distribution implies that the distance between the workers on the cores is maximized. For the Xeon Phi we assign a single worker to each core, before we would start putting another worker on an already occupied core. Theoretically, we could go up to four workers per core. In most scenarios three workers per core performs better.

Testing the performance of a synchronization mechanism in a reliable, reproducible manner is not trivial. We need to find a way that tries to minimize the arrival time of the workers at the synchronization. We need to find the best way to measure the actual synchronization time, which is the overhead introduced by the synchronization function, called a *barrier*.

Algorithm 6 Barrier Measurement

1: **for** each worker w **do**
2: setup test barrier b_T for w ▷ optional
3: call general purpose barrier b_G
4: $t_w^{(i)} \leftarrow$ current time
5: call test barrier b_T
6: $t_w^{(f)} \leftarrow$ current time
7: **end for**
8: $\tau \leftarrow \max_w t_w^{(f)} - \max_w t_w^{(i)}$

Our algorithm to measure the overhead of a barrier τ reliably is shown in Algorithm 6. The general purpose barrier is called to fix potential delays from the setup and to smooth the upcoming synchronization.

This approach can be used to check the synchronization for reliability. We only need to use smallest t_w^f instead of the largest. The reliability requirement is satisfied if $\tau \geq 0$. Of course this is only a quick check and no formal proof.

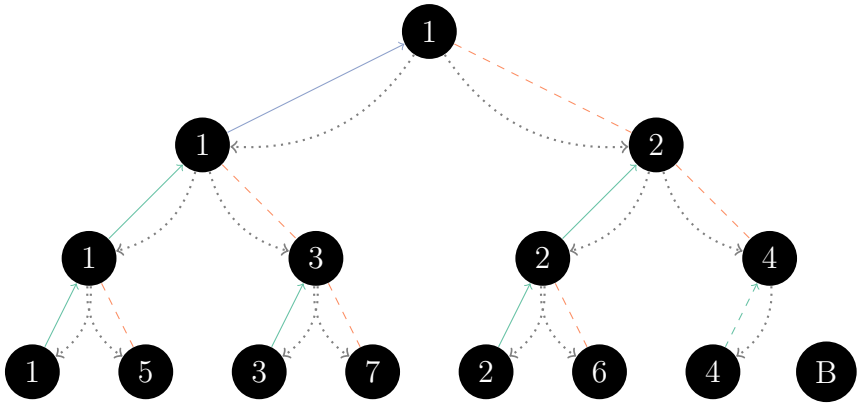


Figure 9.6: Procedure of a tournament barrier consisting of 7 workers. The teal line denotes the winner, the dashed orange line the loser. Byes are not connected. The blue line denotes the tournament winner. This worker has to trigger the wake-up sequence illustrated by the gray edges.

Previously, we concluded that the integrated solutions satisfy our scaling demands, however, not our general efficiency needs. A barrier that costs more than $\mathcal{O}(10^4)$ cycles is too expensive for our applications. Therefore, we need to create an improved barrier. We start by looking for better algorithms and Xeon Phi optimized implementations, which match our demands.

The centralized barrier scales like $\mathcal{O}(N)$, i.e., linear, by design. It could only be interesting for us if the Xeon Phi somehow provides cheaper all-to-all communication than computation. For really small worker counts that might be the case, however, in our case we scale beyond $\mathcal{O}(10^2)$ workers. Here we do not expect any performance advantages using a centralized barrier. Therefore, we use a tournament barrier, which promises much better scaling.

A tournament barrier creates a tree, which scales like $\mathcal{O}(\log N)$, i.e., logarithmically, by design. Each round of the tournament is set up by small centralized barriers, which only involve two workers at a time. Hence we only need point-to-point communication. The algorithm can be improved even further by drawing an efficient grid instead of an arbitrary set of matches. Such a grid prefers neighbor matches to crossing ones.

An effective grid tries to minimize the distance on the ring bus between the workers in each round, which is supposed to minimize potential contention on the ring bus. The basic algorithm is sketched in Figure 9.6. The illustration assumes a total of 7 workers to show the most important concepts. For instance, as there is no eighth worker the match between the fourth worker and the eighth worker ends with a dropout for the fourth worker. The standard algorithm for setting up the tournament grid is illustrated in Algorithm 7. We distinguish between the states champion, winner, loser, and dropout.

Already a naive implementation of the tournament barrier kernel shown in

Algorithm 7 Tournament Barrier Setup

```
1: for each worker w ∈ [0, N − 1] do
2:     c ← 2
3:     p ← 1
4:     for each round r ∈ [1, log2(N)] do
5:         if w|c = 0 and r < N and c < N then
6:             role of w in r is winner
7:             opponent of w in r is w + 1
8:         else if w|c = 0 and r ≥ N then
9:             role of w in r is bye
10:        else if w|c = p then
11:            role of w in r is loser
12:            opponent of w in r is w − 1
13:        else if w = 0 and c ≥ N then
14:            role of w in r is champion
15:            opponent of w in r is w + 1
16:        end if
17:        p ← c
18:        c ← 2c
19:    end for
20: end for
```

Algorithm 8 Tournament Barrier Kernel

```
1: r ← 0
2: while sleeping do                                   ▷ sleep initially
3:     if role in round r is loser then
4:         notify opponent
5:         wait for opponent
6:         stop sleeping
7:     else if role in round r is winner then
8:         wait for opponent
9:     else if role in round r is champion then
10:        wait for opponent
11:        wake up opponent
12:        stop sleeping
13:    else
14:        r ← r + 1
15:    end if
16: end while
17: while r > 0 do                                     ▷ wake up
18:    r ← r − 1
19:    if role in round r is winner then
20:        wake up opponent
21:    else if role in round r is dropout then
22:        r ← 0
23:    end if
24: end while
```

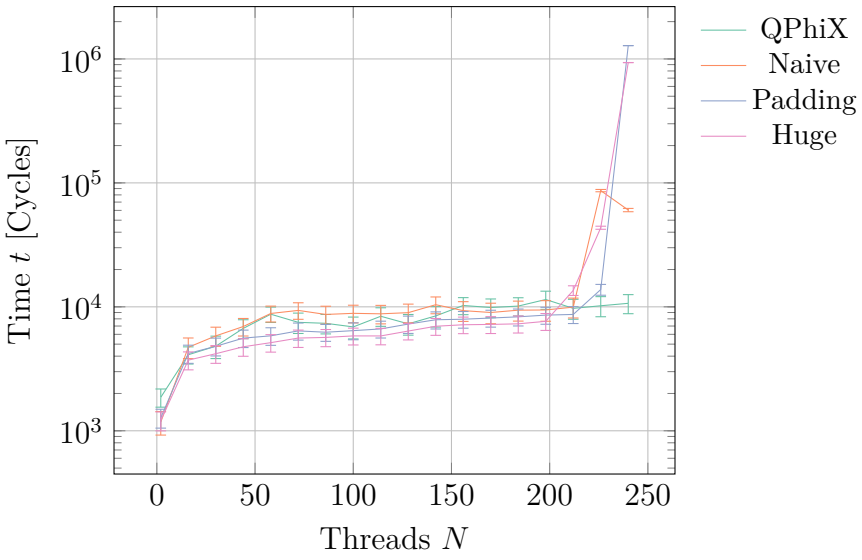


Figure 9.7: Different implementations of a tournament barrier with optimizations for the Intel Xeon Phi.

Algorithm 8 is better than the default implementation provided by the OMP framework. By incorporating a Xeon Phi optimized memory layout we can gain even more. Another potential improvement is the usage of Xeon Phi's streaming stores, which omits loads. Further tricks for programming the co-processor more efficiently can be applied as well.

In Figure 9.7 we compare some of our efforts. In the plot every barrier is definitely faster than the default OMP barrier. All barriers are scaling similarly. They all seem to use a tournament kind of barrier. Most interestingly, the barrier, which uses huge pages in conjunction with a specific padding, shows the best performance until 180 workers. For more threads the QPhiX implementation from Chroma [29] seems to be stable, resulting in the best performance.

Finally, we might be interested in the various locking mechanisms that are provided by all the frameworks we looked into. In general it is possible to use the `lock` prefix for a subset of instructions, e.g., `incl`, and `xchg`, `cmpxchg` for a sequence of instructions. As usual it is beneficial to abstract such hardware-related features by preferring framework or language features.

We can compare standard mutex algorithms against specialized atomic operations or more general reducers. The overhead for three synchronization mechanisms provided in each of the frameworks are shown in the plots of Figure 9.8. The sync fetch-and-add (SFA) mechanism of Cilk Plus represents the fetch-and-add atomic built-in originally coming from the GCC. It also appears in the C++11 standard. In its purest form SFA is used to implement mutual exclusion and many concurrent algorithms in general.

Most importantly, we should avoid or limit the usage of any mutex implemen-

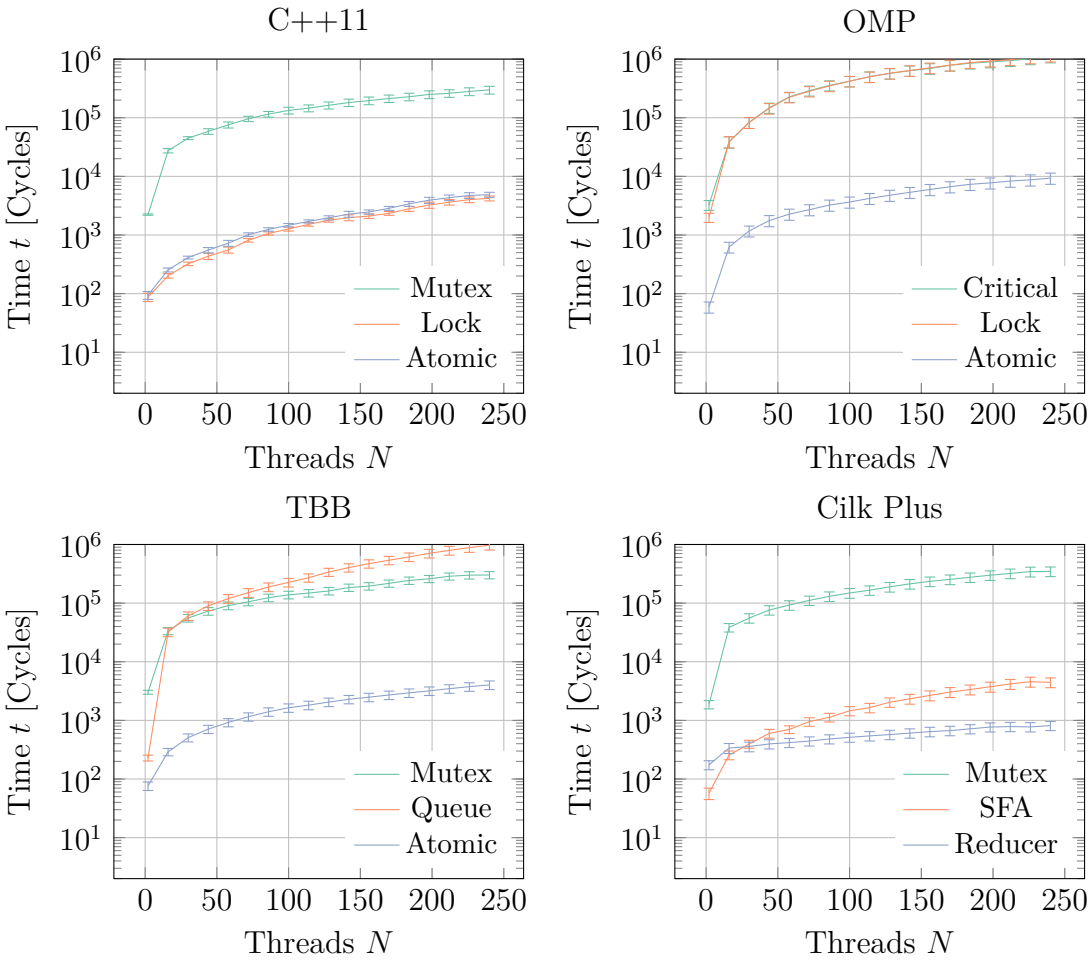


Figure 9.8: Provided synchronization primitives by the different frameworks.

tation. In general we should prefer atomic operations. Interestingly, there are huge differences among the different implementations. By using a more optimized implementation, such as the one provided by the reducers in Cilk Plus, we can save a magnitude in overhead as compared to a more heavyweight version, found in, e.g., OMP.

The reducers in Cilk Plus work very efficiently by limiting the required synchronization. Cilk Plus uses hyper-objects, which is a mechanism that allows different branches of a multi-threaded program to maintain coordinated local views of the same non-local variable [54]. Nevertheless, especially in scenarios where we need to read the value of the atomic variable a lot, OMP is still an excellent alternative.

9.5 Vectorization

Each core of the Xeon Phi is far less powerful than ordinary computer processors. Since massive parallelization also comes with some overhead, it is questionable if using the co-processor may actually result in performance improvements.

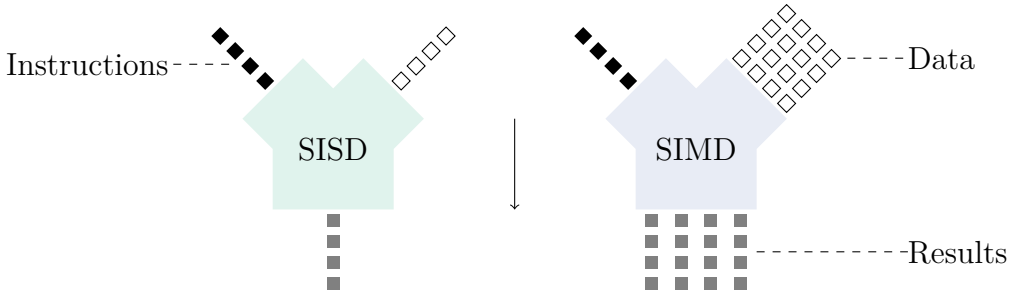


Figure 9.9: SIMD can operate on a vector of data using a single instruction.

In this section we introduce the largest possible performance impact on the Intel MIC platform: vectorization using SIMD instructions. The Xeon Phi contains a 512 bit vector unit that supports integers, as well as single and double precision floating point numbers. It enhances the usage of special math functions.

While classical computer systems only operate with a single instruction per data item, a SIMD system may apply a single instruction on a vector of data items. The maximum length of such a vector on the Xeon Phi is 16 single precision floating point numbers. The basic concept is illustrated in Figure 9.9.

There are several ways to utilize the special vector unit of the Xeon Phi. One is to write assembler code that matches the special operation codes directly. This is neither portable nor robust. More intelligent compilers will not be able to optimize our code even further. The next level would be to use so-called compiler intrinsics. These are special functions that map directly to their respective assembler counterparts. In contrast to writing pure assembler instructions, the compiler knows these functions and may be able to do further optimizations. Finally, it is easier to port from one architecture or compiler to another.

We can even go further and abstract the intrinsics into our own libraries. These libraries may induce best practices and general optimizations, which are then compiler independent. Due to the increased level of abstraction our application may be even more portable. Nevertheless, we also introduced another dependency and we need to be sure that the abstraction is as transparent as possible.

The most portable approach is certainly to let the compiler decide if a certain loop can be vectorized. While the portability is enhanced in this process, the performance is decreased in general. The compiler does not know how to handle any jumps inside the loop, e.g., how to incorporate function calls. Therefore, we lose the ability to make use of so-called elemental functions [60], which can be generated via special vector attributes. An example is illustrated in Listing 9.3.

An elemental function can be used with vector arguments. It can be used with scalar arguments, which makes the function highly flexible. Inside the function we are not allowed to use any jump instruction or non-elemental function call.

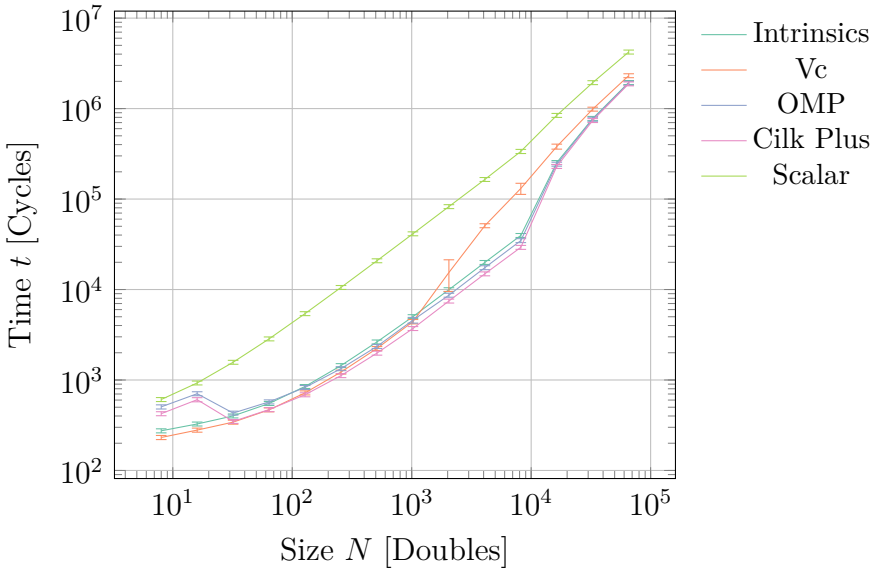


Figure 9.10: Addition of two double precision vectors with the displayed length. We compare a scalar version against a naive implementation using intrinsics, the Vc library, OMP, and Cilk Plus.

Additionally, certain operations are not allowed. Finally, we cannot make use of explicit SIMD inside the function body. There are more restrictions, but it basically boils down to a simple statement: Elemental functions only represent a computation scheme for SIMD, not a full functional unit. They are expanded to a functional unit, either for scalar or for vector usage, by the compiler.

```
1 __attribute__((vector)) double add(double x, double y) {
2   return x + y;
3 }
```

Listing 9.3: Creating an elemental function.

We have written some simple benchmarks to compare the performance of using raw intrinsics with the performance of several libraries. As for the libraries, we have chosen OMP, Cilk Plus, and Vc [83]. OMP seems to be a good choice due to the integration of OMP for parallelization. Cilk Plus is a language extension that is available in Intel's compiler. Being a language extension it is very natural and easy to use. The last library, Vc, is experimental and tries to standardize the vectorization by using transparent abstraction in conjunction with C++ templating features. It is very portable and works with other architectures, e.g., AVX, too.

Among a few other tests, the simple addition of two vectors seems to be the most illustrative. In Figure 9.10 we show the performance of the chosen technologies while adding two vectors of the same length. As data type we have decided to use double precision floating point numbers. This gives us a naive speedup of 8. Indeed we see a speedup close to $\mathcal{O}(10)$. We can observe that writing raw intrinsic code

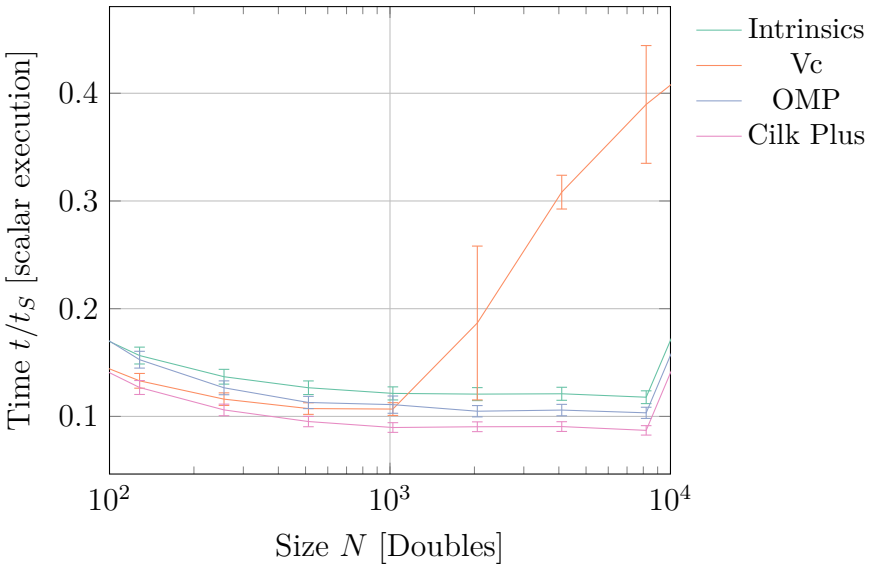


Figure 9.11: A closer look at the performance difference of the addition of two double precision vectors from Figure 9.10. The time is relative to the scalar execution time.

is usually not required for substantial performance gains. The chosen frameworks are all more than competitive.

While Vc uses some interesting startup optimizations, it brings some additional padding to the table. This padding is not well optimized for the Xeon Phi, which consumes too much of the available cache. In effect we see a drop in performance due to requiring data from the L2 cache much earlier than with the other frameworks.

Writing SIMD instructions with intrinsics is a viable solution. Nevertheless, we see that more optimizations can still be done to make the code even faster. The startup cost can be optimized as demonstrated by Vc. The cost for operating on cached data may be reduced further. In the end we observe that programming naively with intrinsics leads to code that runs slightly slower than code that has been annotated with OMP or Cilk Plus `pragma` instructions. Writing faster SIMD code than OMP and Cilk Plus is certainly possible, but an effort that needs to be justified first.

Looking closer at the region with $N \in [10^2, 10^4]$ entries per vector we see some interesting details. In Figure 9.11 we change the time axis to reflect the inverse speedup to the scalar version. We see that the intrinsic version is really bound by our claimed speedup of 8, which gives us a relative time of 0.125 to the scalar version.

Nevertheless, the other options, most importantly Cilk Plus, go even beyond a speedup of 8. The explanation lies in the prefetching instructions, which are automatically inserted by these frameworks. Both, the scalar and the intrinsic

version have been written without using any prefetching instruction. Using some software prefetching is definitely faster than relying on the hardware prefetcher.

From our evaluations it seems that using OMP, or even better Cilk Plus, is the ideal way to use SIMD on the Xeon Phi. The only reason against them is the associated overhead that results in weak performance for small vectors. The advantage of OMP is certainly its portability. While Cilk Plus has great support from ICC, it does not have much support from other compilers even though there is a special branch of GCC and a Clang fork with the Cilk Plus language extensions.

Another reason for OMP is consistency. Doing both, parallelization and vectorization in a single framework is definitely appealing. This way we have a greater dependency on a single framework. This single dependency, however, may be easier to resolve than multiple dependencies. On the other hand, we can argue that a certain distinction is definitely nice to have. Additionally, Cilk Plus offers much finer control than OMP.

If portability really is the pressing issue then Vc does seem to be the ideal option. It offers solid performance with unmatched cross-platform and compiler support. The library contains very useful helpers and allows us to write lower level code. It provides a close-to transparent abstraction with almost no overhead. Once the Xeon Phi support has improved we can expect better performance in the long run.

9.6 Performance and Best Practices

HPC is the art of maximizing the usage of the available computing devices. The rules of the game do not change with the Xeon Phi, however, new rules should be followed to obtain satisfying performance.

Following best practices for the Xeon Phi is important. A good list with very general guidelines and tips has been published by Jeffers and Reinders [73]. An empirical study found in [48] gives us some interesting insights. A study that is oriented towards applications in LQCD can be found in [103].

In the following paragraphs we investigate what kind of best practices should be applied for the Xeon Phi using frameworks like MPI and OMP. The goal is to identify useful improvements and propose them for integration in Chroma [46, 29].

9.6.1 Memory Layout

Implementing parallel algorithms is not a very difficult task. The hard part is optimizing these algorithms in such a way, that they outperform serial implementations and scale well. Therefore, these implementations need to be very flexible

and quite sophisticated. They need to reduce communications and synchronizations.

Usually, it is impossible to transform a serial implementation to a good parallel implementation without changing the data layout. The layout needs to be modified in such a way, that dependencies are minimized or shifted to require a minimum of communication.

An important role in the transformation is the memory structure. The cache coherency protocol and the interaction, e.g., by offloading, with the external CPU have to be considered as well. The KNC uses a ring bus system that connects every core with the main memory (see Section 7.3). In total there are eight memory controllers, with each one being responsible for a part of the memory. The allocation algorithm prefers memory reservations taken by the closest controller.

Memory allocations require synchronization within a system call, which invokes an expensive context switch [135]. It makes sense to reduce allocations or preallocate a larger chunk of memory. The process of memory allocation has to consider the right memory alignment. The alignment describes the specific byte boundaries for a particular memory object.

For the Xeon Phi we need align memory to 64 B. One way to achieve this is by using the special intrinsics provided by the Intel compiler in the header file `immintrin.h`. We see an example for using the intrinsics to allocate (and deallocate) aligned memory in Listing 9.4.

```
1 double* mem = _mm_malloc(sizeof(double) * size, 64);
2 // ...
3 _mm_free(mem);
```

Listing 9.4: Allocating and deallocating aligned memory.

Another alignment issue can be found in the actual usage of memory. At compile-time the address of a pointer is not known. Thus the compiler cannot assume that occurring pointers do actually represent aligned memory or not. We should declare the corresponding data alignment before executing any operation. It is sufficient to specify the alignment once per variable.

The codes in Listing 9.5 and Listing 9.6 illustrate two ways of declaring data alignment information. The first function uses an array-type, which is declared to be 64 B aligned in the beginning of the function. The second function uses arbitrary, maybe overlapping, pointers, which are being declared as aligned before the loop. The access to the memory's content is vectorized with Cilk Plus. The alignment information is important for achieving the best possible performance with vectorization.

```
1 void inc(double p[]) {
2   __assume_aligned(p, 64);
3   for (int i = 0; i < n; ++i) {
4     p[i]++;
5   }
6 }
```

Listing 9.5: Declaring the alignment of variables.

Overlapping may result in inferior performance. It is possible due to complete freedom in pointer arithmetic and cannot be excluded in a programming language like C/C++ when dealing with copy of pointers, such as aliases or function arguments. Usually, such languages already come with a cure. For example, in C we can use the `restrict` keyword to ensure that the compiler assumes non-overlapping memory as shown in Listing 9.6. Naturally, this choice gives us more responsibility and the compiler a lot more freedom for optimizations.

```
1 void mul(double restrict* a, const double restrict* b, const double
      restrict* c, int n) {
2   __assume_aligned(a, 64);
3   __assume_aligned(b, 64);
4   __assume_aligned(c, 64);
5   for (int i = 0; i < n; i += 8) {
6     a[i:8] = b[i:8] * c[i:8];
7   }
8 }
```

Listing 9.6: Alignment with non-overlapping guarantee.

All this is done to increase efficiency of data loads and stores to and from the CPU. A processor moves data more efficiently when that data can be moved to and from memory addresses that are on specific byte boundaries. Additionally, this prevents one potential source for so-called false sharing.

9.6.2 Sharing Data

One more item to consider is false sharing. This happens when two different cores read and write adjacent data in the same cache line. The cache coherency protocol has to make sure that a consistent program flow is ensured. It maintains the consistency between all the caches in a system of distributed shared memory. In this scenario we pay the price for demanding such a consistency.

False sharing consequently results in terrible performance, because the cache lines are continually moving among the caches of different cores as one core after

another tries to write in order to update their data element. Usually this is fixed either by padding to cache line boundaries or by using private variables.

9.6.3 Prefetching

Memory bandwidth and latency are crucial for a well performing system. The programmer has to know how to efficiently use the caches [43]. Most performance is actually wasted by cache misses. A miss is occurring if data should be already in a certain cache level, but in reality is not. This data then has to be requested from a lower level of memory, which might take up to $\mathcal{O}(100)$ cycles. The delay could have been avoided by issuing prefetch instructions. The compiler will then insert such instructions starting with the second optimization level. However, in certain cases we might be required to insert prefetch instructions by hand.

We can use a special compiler `pragma` directive to insert prefetches manually. We could add prefetches with compiler intrinsics. As an example, we consider the code shown in Listing 9.7. The code shows a simple function that takes an array and two integers as input variables. When we loop over the elements of the vector, we can tell the compiler about possible prefetch optimizations. In our case we instruct a `vprefetch1` for the vector with a distance of 16 vectorized iterations ahead. Similarly, we want a `vprefetch0` with a distance of 6 iterations. If we would not have used any `pragma` directive, then the compiler would have picked both distances.

```
1 void foo(int* vec, int m, int n) {
2   for (int i = 0; i < n; ++i) {
3     #pragma prefetch vec:1:16
4     #pragma prefetch vec:0:6
5     for (int j = 0; j < 2 * n; ++j) {
6       vec[i * m + j] = -1;
7     }
8   }
9 }
```

Listing 9.7: Manually prefetching data.

Prefetching is definitely important, but we need to think about implications of our actions. It is possible that we end up with less performance than before, especially when prefetching data that are not actually used, e.g., by blocking bandwidth to the cache. Additionally, cache lines that are used rarely may be prefetched too often. Hence the cache has to throw out cache lines, which are used more actively.

Code runs faster when data are reused while they are still in the processor registers or the processor cache. It is frequently possible to block or tile operations

such that data are reused before they are evicted from cache. The top priority should be assigned to work on limited sets of data. This can be considered an algorithmic optimization, which is always preferable to implementation specific optimizations.

Last but not least, we should always prefer structure of arrays (SoA) over array of structures (AoS). The grouping makes it much easier for the compiler to insert the right prefetching commands. Furthermore, data is usually reused much better. The only major drawback is the strong impact on the memory layout. It is possible that large parts of our applications need to be rewritten to support a SoA data layout.

9.6.4 Thread Affinity and Placement

An interesting parameter to play around is the thread affinity. The thread affinity makes most sense on systems with socket dependent memory latency or bandwidth, i.e., non-uniform memory access (NUMA) systems. Threads in a NUMA system that share some memory should be placed on the same socket. We might want to place IO heavy threads on the socket closer to the IO device.

On the Xeon Phi the ring bus tries to guarantee uniformity. Nevertheless, in practice the NUMA factor is greater than zero. Hence setting thread affinity may be beneficial. There are several (predefined) modes:

- `compact` tries to minimize the distance between the threads.
- `scatter` tries to maximize the distance between the threads.
- `balanced` minimizes the distance between the used cores and maximizes the distance between the active threads.
- `none`, i.e., threads are not bound to any context. This setting should be avoided in general.

Naturally, `balanced` is the affinity to consider. It places threads on single cores first, while still keeping the core's in short distance to each other. In most test cases this is the best possible setting. It is never significantly worse than other options. All these options are set as environment variables, e.g., `KMP_AFFINITY=balanced`.

Even more important than the affinity is the processor binding. If we do not set the processor binding, the host OS is free to shuffle the threads to arbitrary cores. Such a reshuffling may be beneficial for some applications, but is destructive for performance in general. OMP, however, has the ability to suppress shuffling requests. We can use this by setting the environment variable `OMP_PROC_BIND` to `true`.

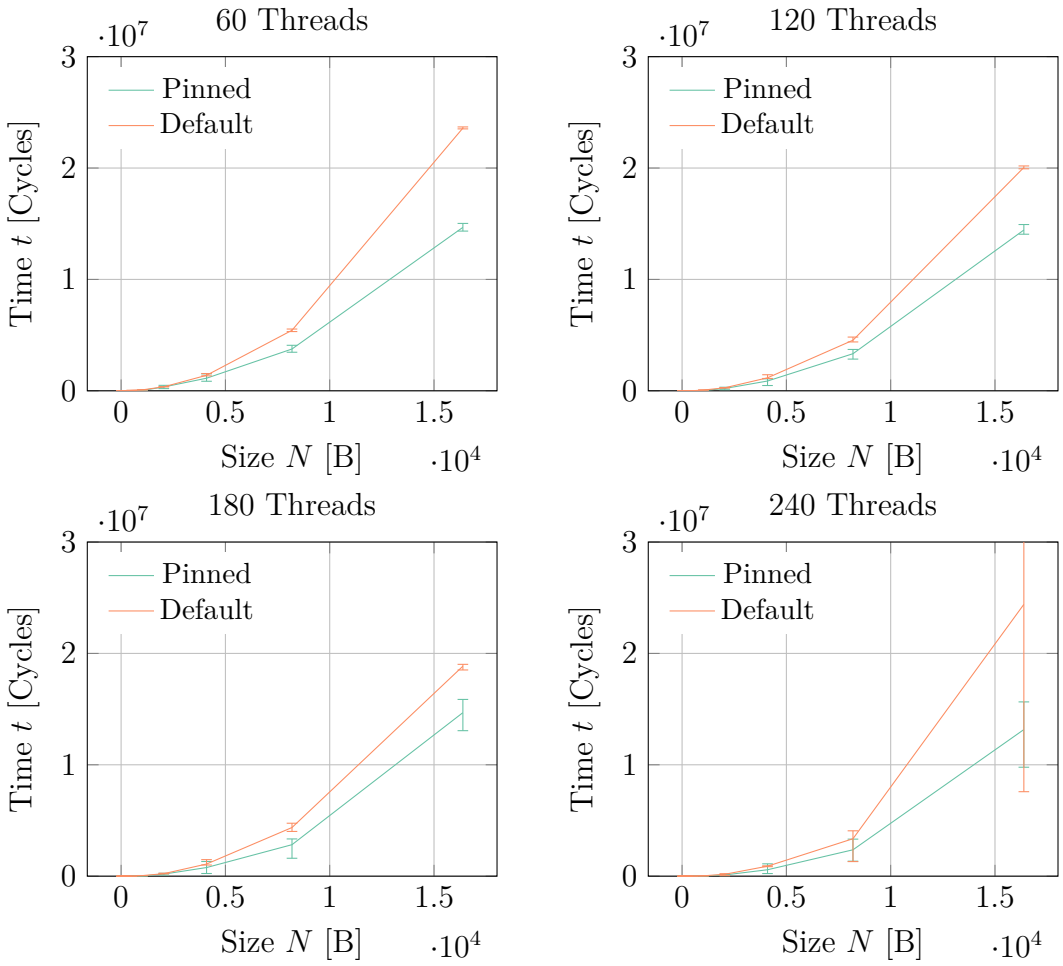


Figure 9.12: The effect of pinning for scaling threaded applications on the Xeon Phi. The benchmark application performed a dot product of two vectors with the given length.

Explicitly pinning the threads to physical cores may not only improve the scaling behavior of our application, but the overall throughput. In Figure 9.12 we observe a better runtime scaling for pinned threads while keeping the number of threads constant. The provided benchmark uses a standard dot product to aggregate data from different places. In general the number of threads seems to be less influential than anticipated.

The optimum solution is to use a pinned program execution model with a balanced thread affinity and a uniformly distributed workload. We neglect possible influences by the host OS, minimize distance between cores on the ring bus, and maximize distance between threads in a core. All in all this approach turns out to be the most efficient. Our findings are backed up by similar evaluations [127].

9.6.5 Node Communication

The inter- and intra-node communication is commonly implemented with the MPI library. We can find many introductory materials to MPI, e.g., the book by Pacheco [109]. For the QPACE 2 project we need to take care of two different levels of MPI communication. From the perspective of MPI a node is defined by a single Xeon Phi.

Therefore, on the one hand we have local MPI nodes, which consist of the Xeon Phi co-processors within the same brick. On the other hand we have the Xeon Phis of other bricks. Additionally, we could distinguish between the nodes within a single-hop reach, and the nodes where communication is required to go over two switches.

A very important benchmark is the HPL application. It is dependent on excellent point-to-point communication performance. This allows us to learn a lot about our communication performance. In the end our goal is to come close to the performance described in [69]. We need to minimize the latency δ and maximize the bandwidth B_W of our network operations.

One way to circumvent performance loss due to unoptimized drivers on the MIC is to utilize the host processor for external communication. A proof of concept has been created by Intel in form of a library called CML proxy. It is a communication layer for the MIC that is supposed to enhance the MIC-MIC data transfer. It works with connections on the same socket and across different sockets. We benchmark the CML proxy in conjunction with the OSU benchmark suite [89]. The test system consists of two nodes, which are connected via quad data rate (QDR) IB.

We are mainly interested in two quantities. Primarily, we have been interested in the bandwidth with the CML proxy library. The OSU benchmarks include both, the ability for a uni-directional and a bi-directional bandwidth measurement. The uni-directional version starts several non-blocking sends via MPI. The receiving side uses matching non-blocking receives. The window size parameter W defines the number of concurrent sends. A single iteration ends when the sending sides gets the receive of all messages acknowledged. The computed bandwidth B_W for sending N bytes k times is given by

$$B_W^{(\mathrm{uni})} = \frac{k\ N\ W}{t_f - t_i}, \tag{9.6}$$

where the first bytes have been sent at t_i and the last acknowledgment has been received at t_f.

The bi-directional version works similar. Here non-blocking sends and receives are started on each node. No acknowledgment is required, since data are received

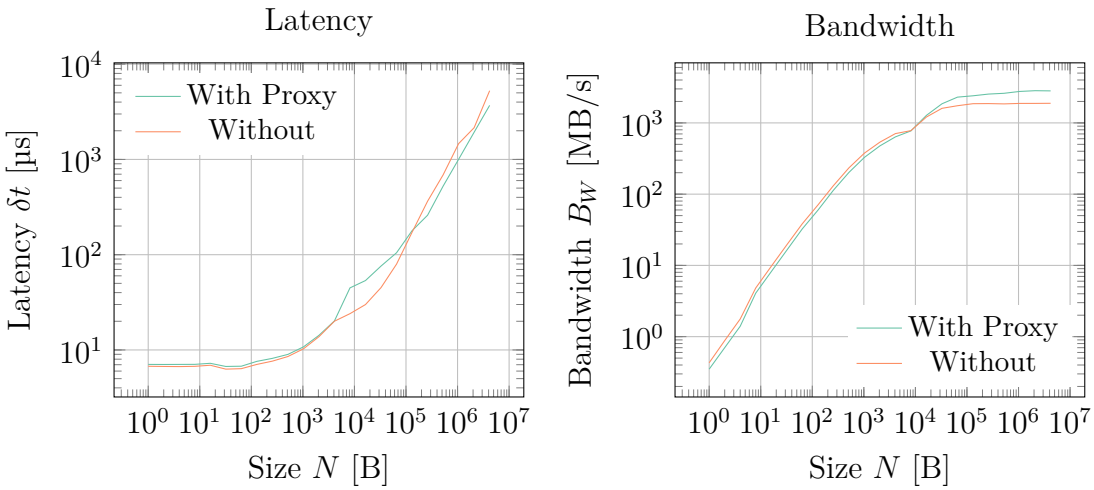


Figure 9.13: Latency and bandwidth of point-to-point communication using MPI alone and in combination with the CML proxy library.

anyway. The bandwidth is computed with an additional factor of 2, which is motivated by measuring on a single system in a symmetric setup. We have

$$B_W^{(\mathrm{bi})} = \frac{2k\ N\ W}{t_f - t_i}. \tag{9.7}$$

Additionally to the bandwidth, we are interested in the latency. The latency is measured with a blocking send from one node to another and back again. This is known as a ping-pong test. It works, because in the regime of small messages, latency is dominating the transfer time. For repeating k ping-pong iterations we have a latency δt of

$$\delta t = \frac{t_f - t_i}{2k}, \tag{9.8}$$

with the same definitions of t_f and t_i as specified for the bandwidth.

The CML proxy relays the message data from the MIC to the host CPUs, which in turn send the data to the destination CPUs. We assign a CPU core to process requests from each local KNC, for extracting the data from local KNC memory to host memory via DMA and for sending the data to the destination CPU. Similarly, at the destination, a CPU core receives the data and uses DMAs to move the data from host memory to the co-processor's memory.

The whole process is performed in a pipelined manner by splitting the application data into several small chunks. The chunk sizes for a given application message are chosen dynamically since smaller chunk sizes can amortize the startup overheads, but at the cost of lower bandwidth. Larger chunk sizes give good bandwidth, but may expose startup overheads. We use a memory mapped request and response queue model for controlling the message handshakes between host and KNC.

In Figure 9.13 we observe a significant performance boost in favor of using the CML proxy layer. We see that latency and bandwidth profit mostly for large message sizes. The scaling of the latency is also better with the CML proxy layer. Most interestingly, the bandwidth seems to be on par with the proxy-less solution until the message exceeds a couple of kilobytes in size. At this point the benchmark with the proxy scales the bandwidth to nearly 3 GB/s, while the benchmark without the proxy stops at the 2 GB/s mark.

Using the CML proxy layer is only a temporary workaround. At some point in the future the MPI solutions will integrate a layer similar to the CML proxy, which is supposed to be more generic and reliable.

9.7 Scaling Architecture

So far we have discussed details of the hardware architecture, parallelization, and vectorization frameworks, as well as best practices with everything being presented in the context of the Xeon Phi. For the final section we look into building a portable, well-performing, and scaling software architecture. As an example we apply our architecture to an implementation of the GMRES algorithm [125].

A "scaling architecture" should be able to handle different levels of parallelization. The lowest level can be found within a single core. Here we want to increase performance by using SIMD instructions. The next level is SMT in the context of a processor. The third level involves communication between processors. Depending on the underlying architecture we could distinguish between processor communication using QPI, PCIe, or a network technology such as IB.

The lowest level can be solved by creating custom data types. Such a data type needs to behave just like its underlying primitive type with additional features such as a length property and integrated SIMD mapping. By using the specialized data type instead of a primitive one, we can easily use the vectorization unit of a particular architecture. If the architecture does not support SIMD we can always fall back to standard loops, which take care of data operations.

We propose an architecture as outlined in the Unified Modeling Language (UML) diagram shown in Figure 9.14. Our proposal makes a lot of demands on the implementing language, e.g., an integrated mechanism to provide zero-cost abstractions. This kind of abstraction is only possible in a compiled language that includes intelligent optimizations.

Our choice for implementing the architecture is the C++ language. C++ provides enough low-level constructs to allow fine grained control. It contains a Turing complete functional meta programming language known as template metaprogramming (TMP). TMP gives us the freedom to write decoupled code, that is

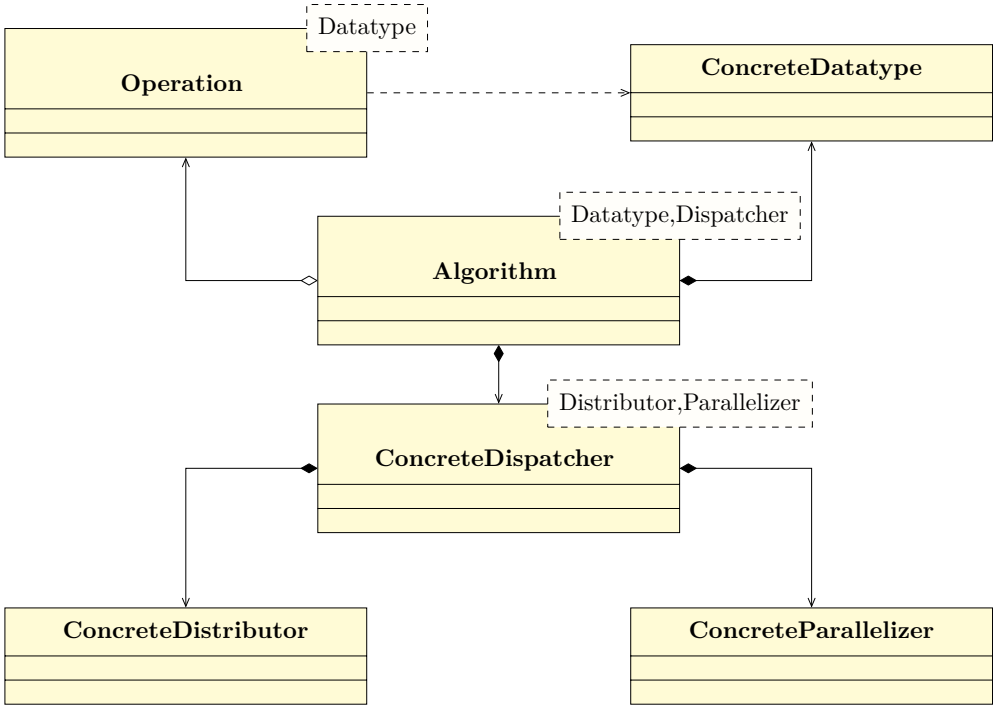


Figure 9.14: UML diagram of the proposed scaling architecture. A language that supports zero-cost abstractions is mandatory for practical implementations.

only bound by functional contracts instead of specific implementations.

The scaling architecture centers around an algorithm. A practically used application may require multiple algorithms, which can depend on different kinds of dispatchers. A dispatcher is an aggregate of one or more distributors and parallelization rules. The implementation uses templates to refine each type and various type traits to use the provided type information. Most of the illustrated architecture actually vanishes during compile-time. At runtime only a very straight chain of instructions remains. Most instructions are tailored to the provided datatype.

The datatype is used to provide the computational base of the algorithm. It can be templated, but this template has to be specialized for the supported data types of a SIMD vector. The datatype exposes elementary operations, e.g., additions, and special math functions, e.g., for computing the sine function. These operations should help to generate the best possible performance.

The algorithm may use smaller functional blocks that appear more often and can be reused. Such blocks should be identified and put into a template for operations. This template should be specialized for particular datatypes. In the end the code is supposed to be easier to reuse and port in the future.

The interplay between parallelization strategies, distribution rules, and algorithms is crucial. We enforce data-oriented programming via the custom datatype. We scale the algorithm even further by using the parallelizer in the algorithm when applicable. We may think about going beyond the single node level, i.e.,

distributed computing. At this level we certainly have the largest communication delays and synchronization problems. We should choose the parts for this scaling area carefully.

As an example implementation of the scaling architecture we implement a simple matrix inversion by using the GMRES algorithm [125]. Inverting a matrix $M \in \mathbb{C}^{N\times N}$ is the same as solving the linear systems $Mx_i = b_i$ for $b_i = (\delta_{i1}, \ldots, \delta_{iN})$ for $i \in [1, N]$. Finally, our solution M^{-1} has N columns, where each column i is given by x_i. We can directly see that the simplest form of parallelization is over the generation of solution columns x_i. These are N independent problems. Since communication is reduced to an absolute minimum we could use this for inter-node scaling, e.g., via MPI. The only things we need to communicate are start conditions such as the matrix M and the source vector b_i, as well as final results in the form of the solution vector x_i. Everything else is independent and allows efficient scaling.

Another form of parallelization can be found in the GMRES algorithm itself. The main iteration of the solver cannot be touched, however, the sub-algorithms can be. We use a QR decomposition and an Arnoldi iteration [12] in the GMRES to construct an upper Hessenberg matrix $H_m \in \mathbb{C}^{m+1\times m}$, where m is the maximum number of Arnoldi iterations. Usually, m is chosen to be $m \leq N$, but this is not required. The main reason this choice is that the algorithm scales like $\mathcal{O}(m^2)$. Constructing H_m is a process that contains matrix-vector and dot products. Thus, we can use multiple threads to parallelize the matrix-vector products if the matrix is much larger than the number of available threads.

Including vectorization is trivially achieved. Any dot product is automatically performed with a SIMD enabled datatype. For really large N we may want to use partitioning in conjunction with parallelization, however, for $N \leq \mathcal{O}(1000)$ there is no real gain, since synchronization costs will dominate in this region.

In our example we scale the algorithm up to four MPI nodes with 120 threads on each compute node. Thus we use a complete brick. Every node makes use of vectorization. In the end we are able to achieve one-third of the theoretical speedup. The theoretical speedup calculation uses a factor of 120 for the number of threads, a factor of 4 for the number of nodes and another factor of 8 for SIMD with a complex double precision type consuming 128 bit per entry.

For smaller problem sizes we perform definitely worse than the scalar version. However, for larger sizes we tend to become competitive. At some point we reach a size that practically hides the intrinsic overhead of the communication scheme. This can be seen in Figure 9.15. Here adding MPI seems to be slightly worse for smaller problems. The origin of this behavior is that even though no information exchange is happening, the MPI communication still has to be established. This,

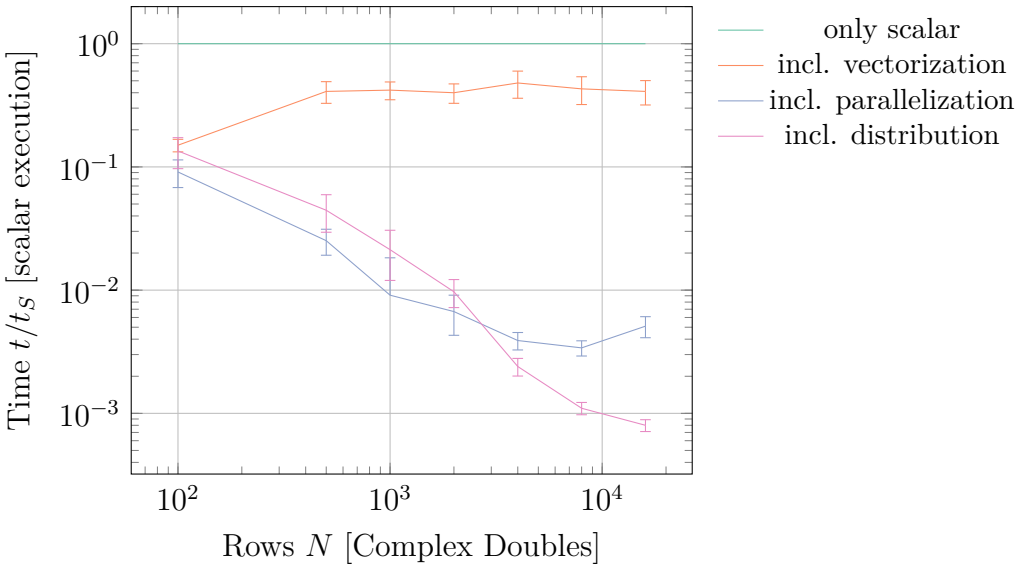


Figure 9.15: Scaling of the sample GMRES implementation from a sequential dispatch to distribution with parallelization and vectorization. For the distribution 4 MPI nodes (1 brick) have been used. The parallelization includes 120 threads. Vectorization implies using the 512 bit vector unit of a Xeon Phi.

of course, could be tuned.

The scaling architecture is suitable for many different algorithms and applications. Its main purpose is to be well performing while decoupling parts of the parallelization from the algorithm. At the end of the day, we gained flexibility and portability.

10 Future Developments

The success of the original QPACE project enabled interesting developments that started with iDataCool and continued with projects such as SuperMUC. QPACE 2 is another important milestone on the road towards exascale computing powered by hot water cooling. Such a cooling is necessary to support exascale machines on an economic basis.

The trend towards heterogeneous architectures cannot be denied, however, the required efforts and necessary application specializations must be taken into account. The chosen architecture for QPACE 2 follows a modular design, which makes maintenance and software design easier. The emphasize on Intel Xeon Phi co-processor cards is a clear statement.

In the future the design could be altered to make use of additional features that might be available in a future Intel MIC based card. It is possible to replace the Xeon Phi of the KNC generation with Knights Landing (KNL) or any other more advanced co-processor card [13]. Hereby additional performance could be gained without requiring a complete redesign of the existing midplane.

Right now there is no way around InfiniBand network cards. Nevertheless, a future design could make use of chip integrated high-speed interconnects. As an example Knights Hill (KNH), a future version of the Xeon Phi produced in a 10 nm process, is supposed to integrate a technology called Omni-Path Architecture. It will support data rates up to 100 Gb/s with higher port density and lower latency than current IB products. It is possible that a future design could feature a novel

PCIe based network solution.

Finally, future developments will continue to explore new cooling processes, which could be more efficient, cheaper, and easier to maintain. Currently a wave of new HPC machines is designed around immersion cooling solutions. The concept is not yet fully developed, but the basic idea is to cool the devices in a dielectric fluid, which is a good thermal conductor. A wide variety of liquids can be used, e.g., electrical cooling oils such as the 3M Fluorinert. These liquids have a low boiling point (typically around 50 °C) and usually low viscosity. Most devices that are cooled by these liquids leave the bath dry. This prevents inefficient maintenance.

The next version will not be a complete redesign, but rather an incremental update that utilizes the existing design with state of the art computing devices. It is easy to predict that the next version of QPACE will surpass QPACE 2 in both, efficiency and computing power.

A

Acronyms

In this work we make heavy use of the following acronyms:

ALU arithmetic logic unit . . . 108

AoS array of structures . . . 164

API application programming interface . . . 114

AVX Advanced Vector Extensions . . . 109

BDMC Bold DiagMC . . . 190

BMC baseboard management controller . . . 106

BLAS Basic Linear Algebra Subprograms . . . 119

CLI command line interface . . . 141

CP-PACS Computational Physics by Parallel Array Computer System . . . 113

CPU central processing unit . . . 6

CRI core ring interface . . . 111

DiagMC Diagrammatic Monte Carlo . . . 34

DMA direct memory access . . . 112

DRAM dynamic RAM . . . 111

FDR fourteen data rate . . . 112

FMA fused multiply-add . . . 110

FPGA field-programmable gate array 105

FPU floating point unit 109

GCC GNU compiler collection 141

GDDR Graphics Double Data Rate 108

GPU graphics processing unit 104

GUI graphical user interface 141

HPC high performance computing 100

HPL High Performance Linpack 118

I2C Inter-Integrated Circuit 117

IB InfiniBand 112

ICC Intel C compiler 141

IDC iDataCool 105

IO input / output 112

IPMI Intelligent Platform Management Interface 117

KNC Knights Corner 105

KNH Knights Hill 172

KNL Knights Landing 172

LCG linear congruential generator 6

LED light-emitting diode 106

LHC Large Hadron Collider 2

LRU least recently used 111

LQCD Lattice QCD 29

MIC Many Integrated Core 105

MKL Math Kernel Library 119

MLX Mellanox ConnectX 115

MMX Multimedia Extensions 109

MPI Message Passing Interface 114

MPSS Manycore Platform Software Stack 142

MTU maximum transmission unit 117

MWC multiply-with-carry 6

NUMA non-uniform memory access 164

NWP network processor . . . 105
OFED OpenFabrics Enterprise Distribution . . . 114
OMP Open Multi-Processing . . . 120
OS operating system . . . 108
PCB printed circuit board . . . 111
PCIe Peripheral Component Interconnect express . . . 105
PIO programmed IO . . . 112
PDU power distribution unit . . . 106
PGO profile-guided optimization . . . 143
PRNG pseudo random number generator . . . 6
PSU power supply unit . . . 106
QCD Quantum Chromodynamics . . . 24
QDR quad data rate . . . 166
QED Quantum Electrodynamics . . . 25
QFT Quantum Field Theory . . . 30
QPACE QCD Parallel Computing Engine . . . 103
QPI QuickPath Interconnect . . . 109
RAM random access memory . . . 108
RNG random number generator . . . 5
SCIF Symmetric Communications Interface . . . 114
SFA sync fetch-and-add . . . 155
SIMD single instruction, multiple data . . . 109
SMT simultaneous multi-threading . . . 146
SoA structure of arrays . . . 164
SPI Serial Peripheral Interface . . . 109
SSE Streaming SIMD Extensions . . . 107
TBB Threading Building Blocks . . . 144
TIM thermal interface material . . . 123
TMP template metaprogramming . . . 168
UML Unified Modeling Language . . . 168
VPU Vector FPU . . . 109

B

Useful Identities

B.1 Gamma Matrices

The following identities have been very useful for simplifying expressions that involve γ matrices. Our matrices are defined to obey

$$\{\gamma_\mu, \gamma_\nu\} = 2\delta_{\mu\nu}\mathbb{I}_4, \tag{B.1}$$

since we work exclusively in Euclidean space. Here $\delta_{\mu\nu}$ is the Kronecker delta. This equation implies $\gamma_\mu\gamma_\mu = \mathbb{I}_4$. We use the convention to sum over repeated indices. In the following we imply the four-dimensional unit matrix.

It is easy to show that

$$\gamma_\nu\gamma_\mu\gamma_\nu = -2\gamma_\mu, \tag{B.2}$$

$$\gamma_\nu\gamma_\mu\gamma_\sigma\gamma_\nu = 4\delta_{\mu\nu}. \tag{B.3}$$

Following these relations we may find that

$$\gamma_\nu\gamma_\rho\gamma_\mu\gamma_\sigma\gamma_\nu = -2\gamma_\sigma\gamma_\mu\gamma_\rho. \tag{B.4}$$

Similarly, we can obtain the relation

$$\gamma_\nu\gamma_\rho\gamma_\mu\gamma_\beta\gamma_\sigma\gamma_\nu = 4\left(\delta_{\rho\mu}\delta_{\beta\sigma} - \delta_{\rho\beta}\delta_{\sigma\mu} + \delta_{\rho\sigma}\delta_{\beta\mu} - \varepsilon_{\rho\mu\beta\sigma}\gamma_5\right), \tag{B.5}$$

where we use the definition of the four-dimensional Levi-Civita symbol,

$$\varepsilon_{\rho\mu\beta\sigma} = \begin{cases} +1 & \text{if } (\rho,\mu,\beta,\sigma) \text{ is an even permutation of } (1,2,3,4) \\ -1 & \text{if } (\rho,\mu,\beta,\sigma) \text{ is an odd permutation of } (1,2,3,4) \\ 0 & \text{otherwise} \end{cases}, \tag{B.6}$$

and $\gamma_5 = -\gamma_1\gamma_2\gamma_3\gamma_4$.

For the multiplication of seven γ matrices we can prove several identities. The following two have been very handy for calculations in this thesis:

$$\gamma_\nu\gamma_\alpha\gamma_\rho\gamma_\mu\gamma_\rho\gamma_\beta\gamma_\nu = -2\gamma_\beta\gamma_\sigma\gamma_\mu\gamma_\rho\gamma_\alpha, \tag{B.7}$$

$$\gamma_\nu\gamma_\rho\gamma_\alpha\gamma_\mu\gamma_\beta\gamma_\rho\gamma_\nu = 4\gamma_\alpha\gamma_\mu\gamma_\beta. \tag{B.8}$$

If we have contractions in certain terms we can try to find an identity, that makes our life easier. An example would be the following equation, which simplifies a product of nine γ matrices. We can prove that

$$\gamma_\nu\gamma_\alpha\gamma_\rho\gamma_\beta\gamma_\mu\gamma_\sigma\gamma_\rho\gamma_\epsilon\gamma_\nu = 4\gamma_\epsilon\gamma_\sigma\gamma_\mu\gamma_\beta\gamma_\alpha. \tag{B.9}$$

There are many possible realizations of the four γ matrices. In this work we have exclusively used the so-called chiral representation. In this representation the chirality operator γ_5 is diagonal.

The explicit representation of the four γ matrices read:

$$\gamma_1 = \begin{bmatrix} 0 & 0 & 0 & i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ -i & 0 & 0 & 0 \end{bmatrix}, \qquad \gamma_2 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix},$$

$$\gamma_3 = \begin{bmatrix} 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \\ -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \end{bmatrix}, \qquad \gamma_4 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{B.10}$$

B.2 Traces

γ matrices will eventually be transformed to scalars by using traces. It is trivial to show that taking the trace of any odd number of γ matrices will result in zero.

This means that, e.g.,

$$\mathrm{tr}\left(\gamma_\mu\right) = 0, \tag{B.11}$$

$$\mathrm{tr}\left(\gamma_\mu\gamma_\nu\gamma_\rho\right) = 0, \tag{B.12}$$

$$\mathrm{tr}\left(\gamma_\mu\gamma_\nu\gamma_\rho\gamma_\alpha\gamma_\beta\right) = 0. \tag{B.13}$$

Additionally, we can show that the trace of γ_5 vanishes. Moreover, we have

$$\mathrm{tr}\left(\gamma_5\right) = 0, \tag{B.14}$$

$$\mathrm{tr}\left(\gamma_5\gamma_\mu\right) = 0, \tag{B.15}$$

$$\mathrm{tr}\left(\gamma_5\gamma_\mu\gamma_\nu\right) = 0, \tag{B.16}$$

$$\mathrm{tr}\left(\gamma_5\gamma_\mu\gamma_\nu\gamma_\rho\right) = 0. \tag{B.17}$$

However, the product of four γ matrices with γ_5 does not vanish under the trace. Here we find

$$\mathrm{tr}\left(\gamma_5\gamma_\mu\gamma_\nu\gamma_\rho\gamma_\sigma\right) = -4\varepsilon_{\mu\nu\rho\sigma}. \tag{B.18}$$

Furthermore, the traces are useful with contracted momenta. For instance, we find

$$\mathrm{tr}\left(\not{a}\not{b}\not{c}\not{d}\right) = 4\left(a\cdot b\, c\cdot d - a\cdot c\, b\cdot d + a\cdot d\, b\cdot c\right). \tag{B.19}$$

There are be many more useful identities, but here we only listed the ones thst have been used throughout this work.

C Evaluations

C.1 Combining Benchmarks

The results in Chapter 9 have been obtained using a custom written application. Even though the application already performs a statistical analysis of all the data gathered, the same run has been repeated on multiple machines using the exact same settings. In the end we have a scenario with n results, all representing averages S_i and errors ΔS_i for $i \in [1, n]$.

Reducing such results to form a single number again requires us to combine multiple averaged data points and their errors. As we've already discussed in Section 2.2, the standard error expressed in Equation (2.45) can be formulated as

$$\varepsilon = \frac{\sigma}{\sqrt{N}}, \tag{C.1}$$

where N is the number of points averaged. σ denotes the standard deviation of the points. The sample standard deviation for a single set of N data points x_i has been defined as

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \langle x \rangle)^2}. \tag{C.2}$$

The mean of this dataset is represented by $\langle x \rangle$.

Before we start with looking into combining n averages, we take care about the special case $n = 2$. We can denote the number of data points with N_a and N_b,

given for the sets a and b. The averages are computed to be

$$\langle a \rangle = \frac{1}{N_a} \sum_{i=1}^{N_a} a_i, \qquad \langle b \rangle = \frac{1}{N_b} \sum_{i=1}^{N_b} b_i. \tag{C.3}$$

We can directly see that $\langle S \rangle$ is calculated from Equation (2.40) to be

$$\langle S \rangle = \frac{1}{N} \sum_{i=1}^{N} S_i = \frac{1}{N_a + N_b} \left(\sum_{i=1}^{N_a} a_i + \sum_{i=1}^{N_b} b_i \right). \tag{C.4}$$

Expressing $\langle S \rangle$ in terms of quantities we know allows us to easily prove that

$$\langle S \rangle = \left(\frac{N_a}{N_a + N_b} \right) \langle a \rangle + \left(\frac{N_b}{N_a + N_b} \right) \langle b \rangle , \tag{C.5}$$

which is just a weighted average of the points. For combining n averages we therefore have

$$\langle S \rangle = \sum_{i=1}^{n} \left(\frac{N_i}{\sum_j^n N_j} \right) \langle x_i \rangle . \tag{C.6}$$

In our special case we always had $N_i = N_j$ for all i and j. As a result we therefore compute the final value to be

$$\langle S \rangle = \frac{1}{n} \sum_{i=1}^{n} \langle x_i \rangle . \tag{C.7}$$

Similarly, we can recombine the error of the datasets ΔS_i. Here our derivation begins again by looking at the special case first. From the general equation for the standard error (Equation (2.45)) we obtain the intermediate result

$$\Delta S^2 = \frac{1}{(N_a + N_b)^2 - (N_a + N_b)} \left(\sum_{i=1}^{N_a} (a_i - \langle S \rangle)^2 + \sum_{i=1}^{N_b} (b_i - \langle S \rangle)^2 \right), \tag{C.8}$$

where we can substitute $M \equiv (N_a + N_b)^2 - (N_a + N_b)$ for readability. Additionally, the variables $M_i = N_i^2 - N_i$ are quite handy for deriving the correct result.

Finally, we obtain

$$\Delta S = \sqrt{\frac{M_a}{M} \Delta a^2 + \frac{M_b}{M} \Delta b^2 + \frac{N_a N_b}{NM} \left(\langle a \rangle - \langle b \rangle \right)^2}. \tag{C.9}$$

Again we can extend the given equation to be used with n data points, but in this case it seems appropriate to make some approximation first. Most importantly we already know that $N_i = N_j$ for all pairs of i and j. As a consequence M_i will be the same for all i.

Another proper assumption is that the dataset is large enough to make $1/N$

negligible. This motivates us to drop the mixed term. At this point we are just left with

$$\Delta S = \frac{1}{2}\sqrt{(\Delta a^2 + \Delta b^2)}, \tag{C.10}$$

which makes the generalization to n points is fairly straight forward. We find that for n such points we obtain

$$\Delta S = \frac{1}{n}\sqrt{\sum_{i=1}^{n} \Delta {S_i}^2}. \tag{C.11}$$

We see that the overall error decreases as compared to all individual measurements. For the case of n measurements with the same result and error, the overall error after combining the intermediate results with be smaller by a factor of $n^{-1/2}$.

C.2 2D Rebinning

The technique of rebinning was used to produce various plots or handle data for evaluations efficiently. Rebinning tries to use already binned data for redistribution to new bins. Usually the number of bins is reduced, but the opposite is also possible. The algorithm that is outlined here can be generalized to N-dimensional rebinning.

We start with data points $a_{ij} \in \mathbb{R}$ with $i \in [1, n]$ and $j \in [1, m]$. Each data point is associated with a parameter value. The parameter values are uniformly distributed, such that x_i and y_j can be calculated via $x_{i+1} = x_1 + i\Delta_x$ and $y_{j+1} = y_1 + j\Delta_y$ respectively, with

$$\Delta_x = \frac{x_n - x_1}{n}, \tag{C.12}$$

$$\Delta_y = \frac{y_m - y_1}{m}. \tag{C.13}$$

In the following we only consider rebinning to another uniform distributed parameter space. In general we could use any other distribution, however, then we need to care more about the assigned parameters. For our purposes we can reduce each parameter to two numbers, e.g., x_1 and Δ_x. This is possible because we know the number of values and the distribution.

Our aim is now to transform the a_{ij} to $b_{kl} \in \mathbb{R}$, where $k \in [1, n']$ and $l \in [1, m']$. As far as the parameters are concerned we see directly that the following

transformation is sufficient:

$$\Delta'_x = \frac{n}{n'}\Delta_x = \frac{x_n - x_1}{n'}, \tag{C.14}$$

$$\Delta'_y = \frac{m}{m'}\Delta_y = \frac{y_m - y_1}{m'}, \tag{C.15}$$

which results in $x'_{k+1} = x_1 + \Delta'_x$ and $y'_{l+1} = y_1 + \Delta'_y$. This covers the parameter transformation in the special case of transforming between uniform distributions.

For the data points the following scheme holds. First we compute the ratios for the transformation and the resulting cell volume change ΔV. We need

$$f_x \equiv \frac{n}{n'}, \tag{C.16}$$

$$f_y \equiv \frac{m}{m'}, \tag{C.17}$$

$$\delta V = f_x \cdot f_y. \tag{C.18}$$

For the mapping of each point a_{ij} to b_{kl} we assign a weight $w_{ijkl} \in [0, 1]$. The transformation now looks as follows,

$$b_{k+1,l+1} = (\delta V)^{-1} \sum_{i=\lfloor k f_x \rfloor}^{\lceil (k+1) f_x \rceil - 1} \sum_{j=\lfloor l f_y \rfloor}^{\lceil (j+1) f_y \rceil - 1} w_{ijkl} a_{ij}. \tag{C.19}$$

We compute the weighted average over all current bins that are (partially) covered by the new bin. As an example we show the weights for a 3×3 to 2×2 reduction. We have

$$\begin{bmatrix} 1 & 1/2 & 1 \\ 1/2 & 1/4 & 1/2 \\ 1 & 1/2 & 1 \end{bmatrix}. \tag{C.20}$$

While the values in the corners can be taken into account fully, the other values are shared and need to be counted only fractionally. Here w_{ijkl} is given by $1/2$ for bins, which are mapped to two bins or $1/4$ for distributing them to 4 bins. Overall each new bin expands to $9/4$ of its original volume, which makes sense since we scale down from 9 to 4 bins while preserving the original volume.

Going in the other direction may be desired as well. Using the previous example again we may now expand from a 2×2 grid to a 3×3 binning. This time the volume per bin shrinks to $4/9$ of its original volume. The coefficient matrix looks like

$$\begin{bmatrix} 4/9 & 2/9 & 4/9 \\ 2/9 & 1/9 & 2/9 \\ 4/9 & 2/9 & 4/9 \end{bmatrix}, \tag{C.21}$$

which works similary to the previously shown one.

How is w_{ijkl} computed? We only need to consider the overlap region from the old bin to the new bin. If we have a more coarse grid we will have $\delta V > 1$. A more fine grid has $\delta V < 1$. However, even transformations that may preserve the local volume can result in information transformation. If $f_x = f_y^{-1}$ we have $m' = nm/n'$ or $m' = f_x m$. Here a reduction in the horizontal direction will still lead to an expansion in the vertical direction.

The weights w_{ijkl} are then calculated via

$$w_{i,j,k,l} = \mathcal{W}_{i,k,n,n'} \mathcal{W}_{j,l,m,m'}, \tag{C.22}$$

where the dimensional weights $\mathcal{W}_{i,k,n,n'}$ are given by

$$\mathcal{W}_{i,k,n,n'} = \min\left(1, \max\left(0, -\left|i - \frac{n(2k+1)}{2n'} + \frac{1}{2}\right| + \frac{n+n'}{2n'}\right)\right). \tag{C.23}$$

The scheme can be generalized to an arbitrary amount of dimensions. It has been sketched in two dimensions to illustrate the generalization without requiring much formality. The closed form of the weights allows us to calculate any of the new patches without knowledge about any of the other patches.

D

Auxiliary Plots

D.1 Two-Loop Integration

The following plots have been obtained while studying the structure and possible divergences of the two-loop diagrams of the anomalous magnetic moment of the electron. They are presented to complete the full set of diagrams and to complement the already shown plot from Section 4.2.

The plots show the integration over the angles of the two-loop diagram shown as indicated in the caption of the plot. All integrations used $q^2 = -10^{-5}$. The horizontal axis of each subplot shows the magnitude of k, the vertical axis shows the magnitude of k'. The integrand contains an additional factor kk' coming from the transformations $k \to \log(k)$ and $k' \to \log(k')$. The angles have been integrated using Monte Carlo integration ($N = 10^6$ configurations). The $\operatorname{logsgn}(x)$ function is defined in Equation (4.50).

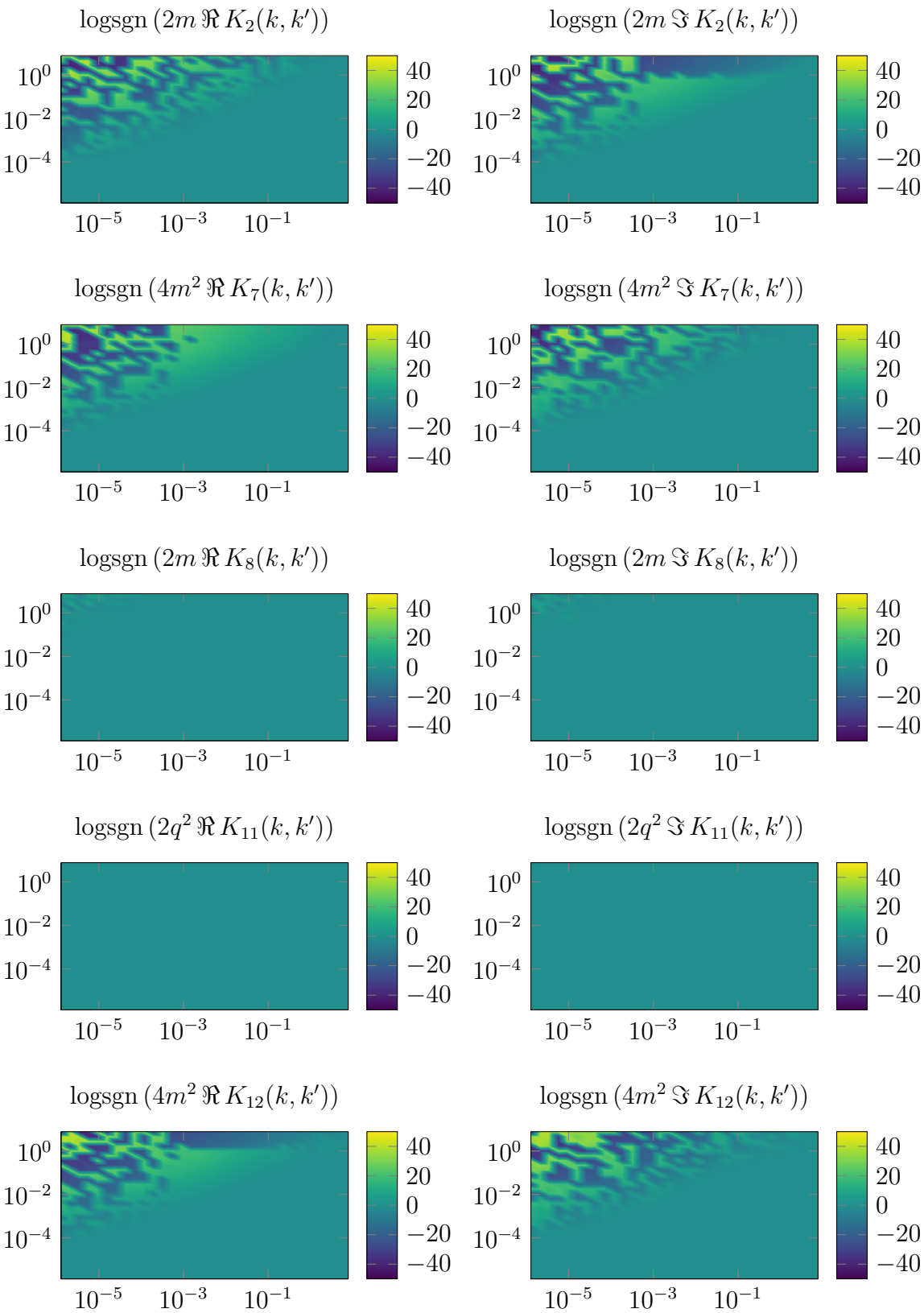


Figure D.1: Evaluation of $\Gamma_\mu^{(2,1)}(p',p)$ from Figure 4.5a.

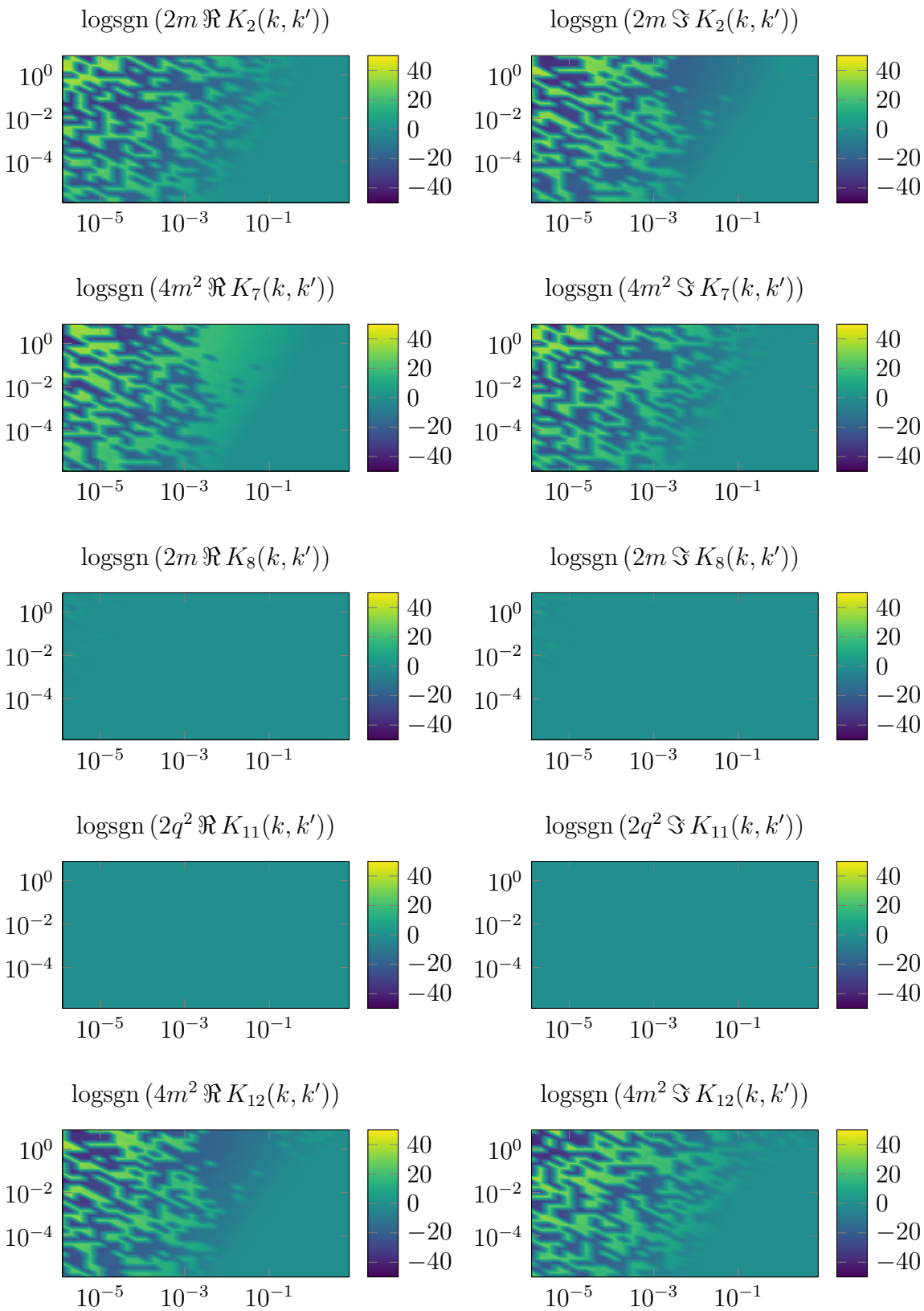


Figure D.2: Evaluation of $\Gamma_\mu^{(2,3)}(p',p)$ from Figure 4.5e.

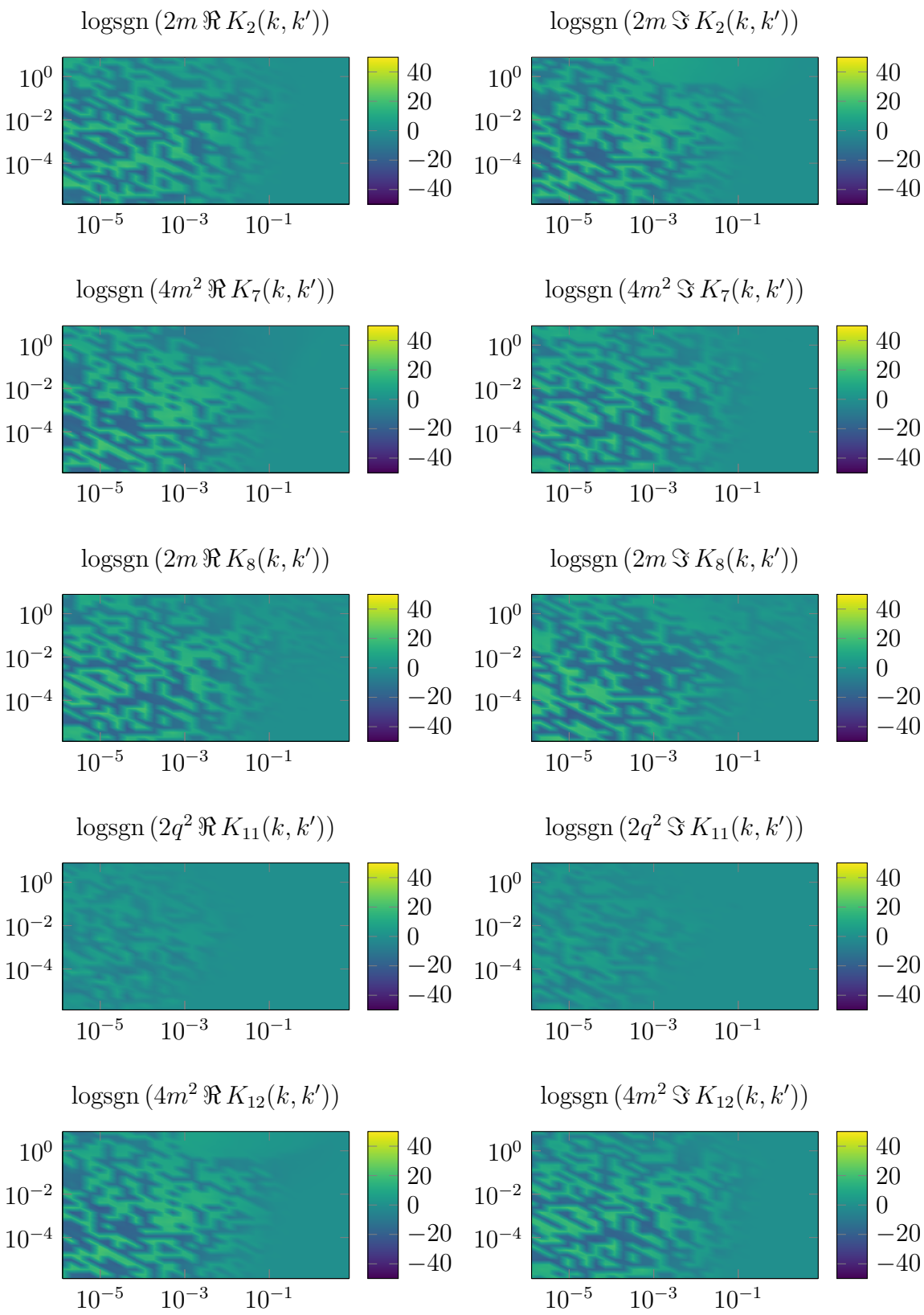


Figure D.3: Evaluation of $\Gamma_\mu^{(2,5)}(p',p)$ from Figure 4.5g.

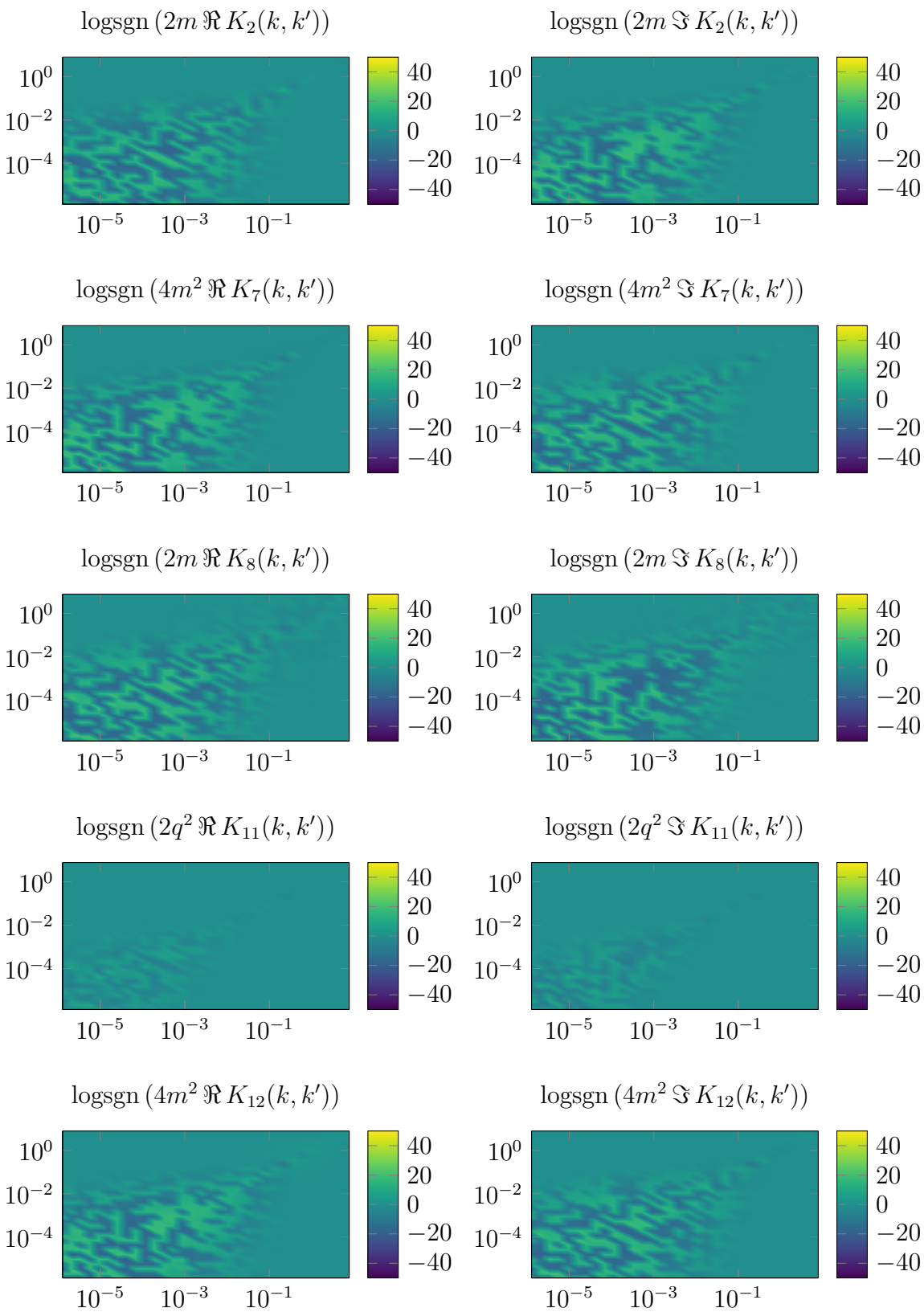


Figure D.4: Evaluation of $\Gamma_\mu^{(2,7)}(p',p)$ from Figure 4.5i.

E

Bold Diagrammatic Monte Carlo

E.1 Bold Diagrammatic Monte Carlo

Sometimes DiagMC is not appropriate, especially in theories where regularization is required or higher-order diagrams make a non-trivial contribution that only affects certain propagators. The technique of Bold DiagMCs (BDMCs) [118] tries to solve these kind of problems. It can be applied, e.g., to solve the Fermi polaron problem [116, 117].

The purpose of BDMC is to reduce the set of diagrams $\mathcal{F}$ sampled by Monte Carlo. This is achieved by considering only skeleton diagrams[1], i.e., diagrams that have no divergences in subdiagrams. As the sign-problem scales exponentially with the number of diagrams, the BDMC technique becomes essential for sign-alternating problems.

One possibility (see [118]) to investigate the technique is to start looking at an equation that defines a vector from the Hilbert space $|f\rangle$ self-consistently, i.e., we have an equation like

$$|f\rangle = |b\rangle + \hat{A}\,|f\rangle\,, \tag{E.1}$$

with some linear operator $\hat{A}$ and an arbitrary source $|b\rangle$. The decomposition into

[1] Any diagram can be expressed as a skeleton diagram by replacing every self-energy part by a (bold) line and every vertex-correction part by a (bare) vertex.

a basis of eigenvector $|\phi_i\rangle$ yields

$$\hat{A}\,|\phi_i\rangle = a_i\,|\phi_i\rangle\,, \tag{E.2}$$

$$|f\rangle = \sum_i f_i\,|\phi_i\rangle\,, \tag{E.3}$$

$$|b\rangle = \sum_i b_i\,|\phi_i\rangle\,. \tag{E.4}$$

Using the decomposition into eigenvectors from above, we can change the initial relation shown in Equation (E.1). We obtain

$$\sum_i f_i\,|\phi_i\rangle = \sum_i (b_i + a_i f_i)\,|\phi_i\rangle\,. \tag{E.5}$$

This allows us to extract an equation that needs to be fulfilled by every coefficient. We identify

$$f_i = b_i + a_i f_i. \tag{E.6}$$

Finding a solution to this problem can be a difficult task. A numerical scheme that could be considered approaches the value by refinement. We iterate to obtain a better version of f_i, which is then used in the next step again. We have

$$\tilde{f}_i^{(n+1)} = b_i + a_i f_i^{(n)}, \tag{E.7}$$

where $f_i^{(n)}$ is a refined value that is computed by considering all previous estimations, denoted by $\tilde{f}_i$.

As an example we have

$$f_i^{(n)} = \frac{\sum_{j=1}^{n} j^\alpha \tilde{f}_i^{(n)}}{\sum_{j=1}^{n} j^\alpha}, \tag{E.8}$$

with the restriction of $\alpha > -1$. We include this restriction to explicitly exclude the divergent harmonic series. At this point we may raise the question how efficient this scheme is.

First we need to derive an estimate for the error. We define the error as the difference to the exact result, i.e.,

$$\delta_i^{(n)} = f_i - f_i^{(n)} = \frac{b_i}{1 - a_i} - f_i^{(n)}. \tag{E.9}$$

Computing the ratio between the error in the $n+1$-th iteration and the n-th iteration gives us an expression that makes estimating the convergence much easier.

By considering the large n limit we obtain

$$\frac{\delta_i^{(n+1)}}{\delta_i^{(n)}} = \frac{\sum_{j=1}^{n} j^\alpha + (n+1)^\alpha a_i}{\sum_{j=1}^{n+1} j^\alpha} = 1 + \frac{(1+\alpha)(a_i - 1)}{n+1}, \tag{E.10}$$

where we used the relation

$$\sum_{j=1}^{n} j^\alpha + (n+1)^\alpha a_i = \sum_{j=1}^{n+1} j^\alpha + (n+1)^\alpha (a_i - 1). \tag{E.11}$$

For $n \to \infty$ we approximate the value of the error as

$$\delta_{i_n} \propto \exp\left((1+\alpha)(a_i - 1)\ln(n)\right). \tag{E.12}$$

Obviously, for convergence the real part of every eigenvalue a_i of $\hat{A}$ needs to be smaller than 1.

It should be noted that negative real parts of a_i are desirable for convergence. Larger absolute values correspond to faster convergence, with the previously specified constraint for positive real values. Our algorithm relies on a parameter α to be specified. We see that the error given in Equation (E.12) scales exponentially with the our choice of α.

The main reason for picking the described scheme is the possibility of large negative values for the eigenvalues. As an example we might come up with a method that uses $f_i^{(j)} \equiv \tilde{f}_i^{(j)}$. We obtain the stronger requirement of $|a_i| < 1$ for all a_i.

It is possible to circumvent problems arising for a_i that do not fulfill our requirements. In this case, we need to change the operator acting on $|f\rangle$ to be hermitian. Luckily, such a transformation is possible. The new formulation of Equation (E.1) reads

$$|f\rangle = \left(1 - \hat{A}^\dagger\right)|b\rangle + \left(\hat{A} + \hat{A}^\dagger - \hat{A}^\dagger\hat{A}\right)|f\rangle. \tag{E.13}$$

The same kind of modification can be done to insert a constant parameter λ into Equation (E.1) if the operator $\hat{A}$ is hermitian. The result looks familiar:

$$|f\rangle = \left(1 - \lambda\hat{A}\right)|b\rangle + \left(1 + \lambda - \lambda\hat{A}\right)\hat{A}\,|f\rangle, \tag{E.14}$$

where we choose $\lambda \in (\lambda_1, \lambda_2)$ according to

$$\lambda_1^{-1} = \min_{a_i > 1} a_i, \qquad \lambda_2^{-1} = \max_{0 \le a_i \le 1} a_i. \tag{E.15}$$

It can be checked that $(1+\lambda)a_i - \lambda a_i^2)$ for λ with former restrictions is always smaller than 1.

Applying the BDMC technique we can sum a series even if it is formally divergent. The method looks very attractive for the sign-indefinite series since it can substantially reduce the number of leading diagrams, and thus alleviate the sign problem. The zero convergence radius argument provided by Dyson does also not represent an unsolvable problem as demonstrated with a zero-dimensional ϕ^4 theory [114].

E.2 S-Wave Scattering

A simple problem that is well suited for illustrating the concept of BDMC is low-energy S-wave scattering. The S-wave scattering problem arises in quantum mechanics, when an incoming beam of particles is scattered by a spherically symmetric potential $V(r)$. We mainly follow the work published in [118].

E.2.1 Model and Simulation

The time-independent Schrödinger equation for a spherical symmetrically potential $V(r)$ has a general solution of the form

$$\psi(r,\theta) = \sum_{l=0}^{\infty} \frac{\chi_l(r)}{r} P_l(\cos(\theta)), \tag{E.16}$$

where $\psi(r,\theta)$ is the desired wave-function. P_l is the Legendre polynomial of order l and χ_l satisfies the differential equation

$$\frac{d^2\chi_l}{dr^2} + \left(k^2 - V(r) - \frac{l\,(l+1)}{r^2}\right)\chi_l = 0. \tag{E.17}$$

with boundary condition $\chi_l(0) = 0$.

If $V(r)$ is proportional to $\Theta(r_0 - r)V_0$, where $\Theta(r_0 - r)$ is the Heaviside step function, we get an asymptotic behavior such that

$$\chi_l(r) \propto \sin\left(k\,r - \frac{\pi}{2}l + \delta_l\right), \tag{E.18}$$

where δ_l is the phase shift of the l-th partial wave. The total scattering cross section σ is then given by

$$\sigma = \frac{4\pi}{k^2}\sum_{l=0}^{\infty}\sin^2\left(\delta_l\right). \tag{E.19}$$

In the model of S-wave scattering we have only one partial wave with $l = 0$ that is contributing, resulting in a single phase shift δ_0. The cross section for S-wave

scattering is given by

$$\sigma = \frac{4\pi}{k^2} \sin^2(\delta_0) \,. \tag{E.20}$$

It is already known that for a finite range potential the cross section must be finite for all energies. This means that the phase shift δ_0 must vanish at least as fast as the momentum k. As a result the expression $\sin(\delta_0)/k$ is finite. An example that requires this condition is the strong interaction potential between nucleons.

In the low-energy regime $k \to 0$ the phase shift behaves as $\delta_0 \to -ka$, where a is some constant with dimensions of length. We call a the scattering length. In this limit we can replace the $\sin(\delta_0)$ with δ_0, which yields

$$\sigma = 4\pi\, a^2, \tag{E.21}$$

the approximate cross section for low energies. This is the maximum cross section. The constant a is called the scattering length and is useful for a description of low-energy scattering.

The scattering amplitude can be computed via the S-matrix approach. For the above equations we obtain the following form in momentum space,

$$f(q) = -u(q) - \frac{1}{\pi} \int_{-1}^{1} d\cos(\theta) \int_0^\infty dq_1\, u(|\mathbf{q} - \mathbf{q}_1|)\, f(q_1), \tag{E.22}$$

where $u(q)$ is the Fourier transform of the potential $V(r)$. The absolute value $|\mathbf{q} - \mathbf{q}_1|$ is expressed in terms of the two magnitudes q and q_1, as well as the angle θ. The argument is therefore computed as follows:

$$|\mathbf{q} - \mathbf{q}_1| \equiv \sqrt{q^2 + q_1^2 - 2q\, q_1 \cos(\theta)}. \tag{E.23}$$

The Fourier transform of the potential $V(r)$ is given by the equation

$$u(q) = \frac{1}{2\pi} \int d^3r\, V(r)\, \exp(-i\, \mathbf{q}\, \mathbf{r}). \tag{E.24}$$

We are mostly interested in the scattering length a. The scattering length a can be computed by considering the low momentum limit of $f(q)$, i.e., $a = -f(0)$.

E.2.2 Update Procedures

Like in the previous sections we will now define orders and weights for our diagrammatic simulation. For this model we only include adding and removing diagrams. There are no other update procedures. First we identify

$$f(q) = f_1(q) + \int_{-1}^{1} d\cos(\theta) \int_0^\infty dq_1\, f_2(q, q_1, \theta), \tag{E.25}$$

which yields

$$f_1(q) = -u(q), \tag{E.26}$$

$$f_2(q, q_1, \theta) = -\frac{1}{\pi} u(\sqrt{q^2 + q_1^2 - 2q\, q_1\, \cos(\theta)}) f(q_1). \tag{E.27}$$

Hence there are at least two diagrams: a first-order and a second-order diagram. The ratios are given by

$$R_{1\to 2} = \frac{\pi^{-1} \left| u\left(\sqrt{q^2 + q_1^2 - 2q\, q_1\, \cos(\theta)}\right) f(q_1) \right|}{|u(q)|} \frac{1}{\Omega_\theta(\theta)\, \Omega_{q_1}(q_1)}, \tag{E.28}$$

and for the inverse operation

$$R_{2\to 1} = \frac{|u(q)|}{\pi^{-1} \left| u\left(\sqrt{q^2 + q_1^2 - 2q\, q_1\, \cos(\theta)}\right) f(q_1) \right|} \frac{\Omega_\theta(\theta)\, \Omega_{q_1}(q_1)}{1}. \tag{E.29}$$

At this point the generation of the value for the momentum q is still missing. We could introduce another update procedure, which would try to create a new value for q. However, simply following the same trick as in Section 3.2 we can introduce an artificial zeroth-order diagram, which is represented by the constant f_0.

Due to the simplified construction of a proper normalization value such an artificial diagram is advantageous for extracting the solution. As a consequence of introducing the constant f_0 to $f(q)$ we need two more ratios, which can be calculated to be

$$R_{0\to 1} = \frac{|u(q)|}{f_0} \Omega_q(q)^{-1}, \tag{E.30}$$

$$R_{1\to 0} = \frac{f_0}{|u(q)|} \Omega_q(q). \tag{E.31}$$

We limit ourselves to $f_0 > 0$. The normalization is determined by f_0 as well.

Bringing everything together an important question is how the recursion on $f(q)$ arising in Equation (E.25) can be resolved. We can see that $f(q)$ not only appears on the left hand side, but also in the part that is identified as $f_2(q, q_1, \theta)$.

We can resolve the recursion by relying on the data that have already been obtained. One could say that for computing $f_i(q)$ we take $f_{i-1}(q)$ as a basis, where the index denotes the Monte Carlo generated version of the value of f at q. This seems like a natural choice, but comes with one important drawback. We are required to supply a start value that is already close enough to the real value of $f(q)$. If we would provide, e.g., $f(q) \equiv 0$, we completely neglect the second-order diagram. Additionally, we do not have any q dependence in our expression.

At this stage we need to introduce an approximation that is used as long as

we do not have sufficient data points. By going back to Equation (E.25) we see that for $f_2(q, q_1, \theta) \equiv 0$ the expression reduces to $f(q) = -u(q)$. Finally, the i-th version of $f(q)$ has the form

$$f_i(q) = \begin{cases} -u(q) & \text{if } i < N_f, \\ h_q \, f_0 \, q_{\max} \, (Z_0 \, \Delta q)^{-1} & \text{otherwise,} \end{cases} \tag{E.32}$$

where h_q is the sum of the measurements in the bin that corresponds to the value of $q \in [0, q_{\max}]$. The value N_f sets the threshold, which marks the point when we gathered enough data. Usually, 10^4 data points in the given bin are more than sufficient. Each bin covers a range of q-values with the width Δq. The normalization Z_0 is the number of measurements of the artificial diagram.

Now the only remaining quantity is the choice of distribution functions $\Omega_q(q)$, $\Omega_{q_1}(q)$, and $\Omega_\theta(\theta)$. The latter is trivial, as we can simply choose a uniform distribution without loss of generality. This is a natural and efficient choice as the dependence on the angle does not favor a specific distribution. However, instead of choosing an angle between 0 and π we can choose a value between ± 1, which already represents the value of the cosine at a given angle.

Choosing $\Omega_q(q)$ follows the same principles as discussed in the previous sections. By identifying a distribution in the weights we are able to extract interesting parts for the importance sampling process. We know that in the end we want a distribution that is proportional to

$$\Omega_q(q) \propto \frac{1}{(q_0 + q)^2}, \tag{E.33}$$

with the constant q_0 to prevent the distribution from diverging. Normalizing this distribution yields a normalization factor of q_0. However, while this is certainly a good choice for q', it is not ideal for q itself. The reason is that q has to be between 0 and $q_{\max}$. With this solution we get $q \in [0, \infty)$.

Therefore we need to limit the integral for computing the normalization factor to $q_{\max}$. We can, however, still use the version with q_0 as normalization factor for $\Omega_{q_1}(q)$. The final distributions for q and q_1 look as follows,

$$\Omega_q(q) = \frac{q_{\max} + q_0}{q_{\max}} \frac{q_0}{(q_0 + q)^2}, \qquad \Omega_{q_1}(q) = \frac{q_0}{(q_0 + q)^2}. \tag{E.34}$$

For computing such a distribution with uniform random numbers as provided by a PRNG we need to make a transformation from the uniform distribution. We just need to use

$$\Omega_q(q) \, dq = u(r) \, dr, \tag{E.35}$$

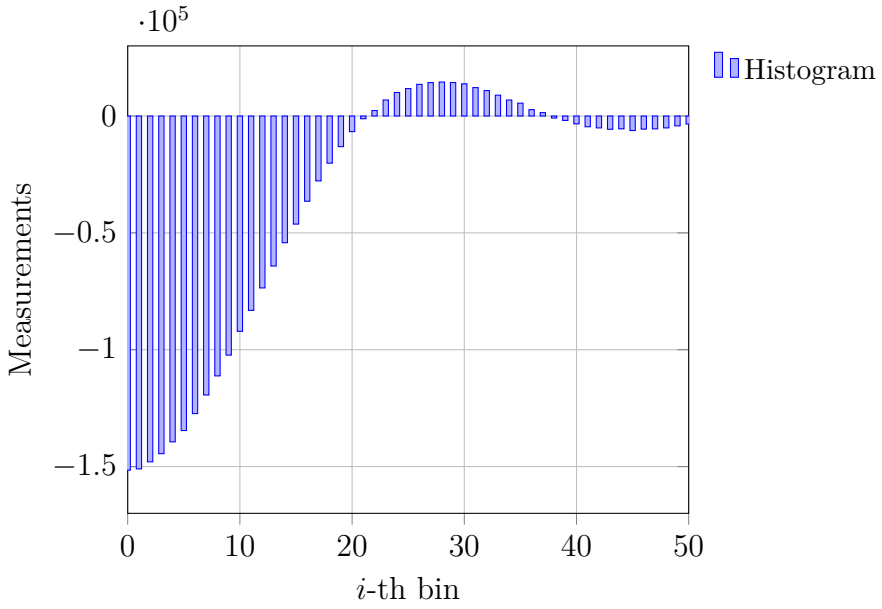


Figure E.1: The histogram to estimate $f(q)$ with $N = 10^7$ iterations. We choose 50 bins for $q \in [0, 10]$. The radius of the spherical potential is fixed at $r_0 = 1$ with $V_0 = 1$.

where $u(r)$ is the uniform distribution. By using $u(r) \equiv 1$ we can solve

$$\int_{w(0)}^{w(r)} dq\, \Omega_q(q) = r. \tag{E.36}$$

The constant $w(0)$ is an additional degree of freedom. We want $w(0)$ to be equal to zero. This simplifies the equation, which reads

$$r = F_{\Omega_q}(w(r)), \qquad w(r) = F_{\Omega_q}^{-1}(r), \tag{E.37}$$

where F_{Ω_q} is the antiderivative of Ω_q. Searching the inverse of this function is then the main task. Actually, only a few examples exist where the inverse can be found analytically. The usual case is to use the simple Metropolis-Hastings algorithm to either accept or reject a numerical solution.

In our case we can find an analytic solution, which yields

$$w(r) = \frac{q_0\, r}{q_0/q_{\max} + 1 - r}, \tag{E.38}$$

for generating values according to the distribution $\Omega_q(q)$. The same distribution can be used for q_1 with $q_{\max} = \infty$.

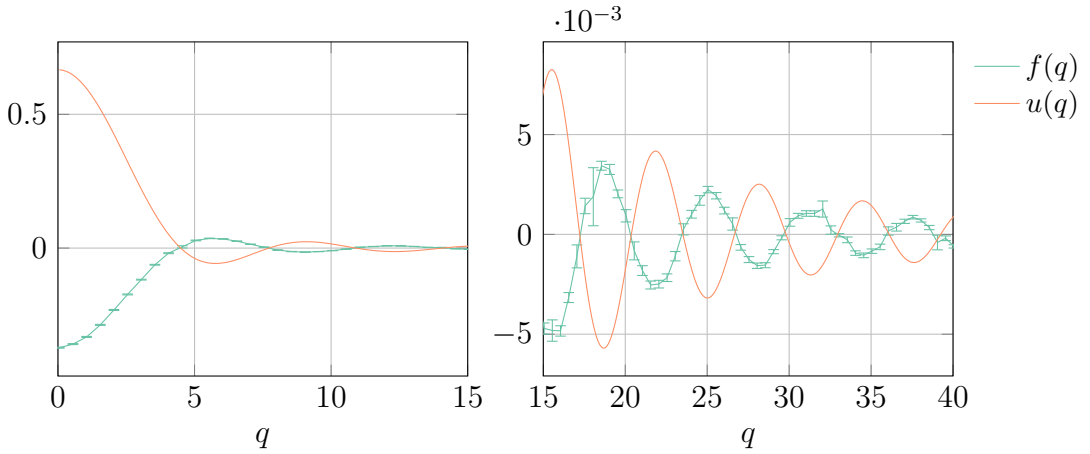


Figure E.2: The numerical estimate of the function $f(q)$ with $N = 10^7$ iterations. The radius of the spherical potential is given by $r_0 = 1$, with $V_0 = 1$.

E.2.3 Evaluation and Results

The interesting quantity is the scattering length a. As an analytical check we can use

$$a = -f(0) = r_0 - \frac{\tanh\left(\sqrt{2V_0}r_0\right)}{\sqrt{2V_0}}, \tag{E.39}$$

which is simply the analytic solution of Equation (E.22) at zero, i.e., for low-energies. In Figure E.1 we see the histogram of a Monte Carlo simulation using BDMC to solve the S-wave scattering problem. The histogram contains all the information needed to estimate $f(q)$.

In general we want a number of bins, as it helps us to be more precise for estimating values $f(q)$, e.g., $\mathcal{O}(10q_{\max})$ for single-digit precision. However, higher bin counts demand more iterations, otherwise while the overall accuracy remains constant the precision per bin is decreasing.

In the end the estimation for the scattering length is possible by looking at the zeroth-bin of the histogram, which is the estimate for $f(q)$ with $q = 0$. In our case the value is estimated to be

$$a = 0.3723(18), \tag{E.40}$$

which is within the error consistent with the analytic value 0.371 817. The simulation used $N = 10^7$ iterations. This takes about a second of computing time on an ordinary machine.

The significance of the histogram shown in Figure E.1 is emphasized by looking at the estimate for the scattering wave function $f(q)$. The plots in Figure E.2 show that the rest is just a scaling factor. Most information is already contained in the histogram. At zero energy with $V_0 = 1$ we obtain the data illustrated in

green. We see that $f(q)$ is qualitatively converging to $-u(q)$ for $q \to \infty$.

We find that for attractive potentials with values $V_0 \leq -10$, good initial conditions, e.g., by using results of previous runs for with slightly smaller $|V_0|$, are important for convergence. Repulsive potentials, i.e., positive values for V_0, converged much faster and provide higher accuracy. No special treatment is required.

With BDMC we can compute integral equations of any kind. A recursive term, as it may appear by Schwinger-Dyson like series, can be handled quite efficiently. We have seen that such models require a little bit more attention only on the analytical side, but can then be handled with the same framework as ordinary DiagMC.

In the next section we look at a more advanced example of simulating a three-dimensional ϕ^4 theory by using its formulation as a Schwinger-Dyson series.

E.3 ϕ^4 Theory

It has been shown that regularization of a diagrammatic series with zero convergence radius is possible [114]. We now look into a more complicated example of applying BDMC. In this section we try to solve a scalar field theory using the ϕ^4 potential [34].

E.3.1 Model and Simulation

We use the Lagrangian

$$\mathcal{L}(\phi) = \frac{1}{2}\left(\partial^\mu \phi(x)\, \partial_\mu \phi(x) - m^2\, \phi^2(x)\right) - \frac{1}{4!}\lambda\, \phi^4(x), \tag{E.41}$$

for a real scalar field $\phi(x)$ with quartic interaction $\lambda\phi^4(x)/4!$.

The bare action for the ϕ^4 theory in three dimensions reads

$$S = \int d^3x \left(\frac{1}{2}\left(-\partial^\mu \phi(x)\, \partial_\mu \phi(x) + m^2\, \phi^2(x)\right) + \frac{1}{4!}\lambda\, \phi^4(x)\right), \tag{E.42}$$

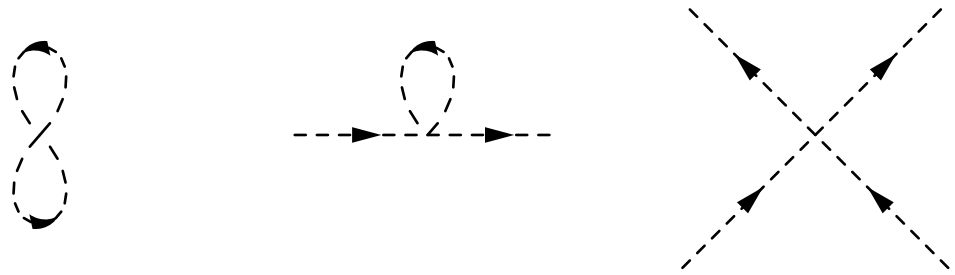
where m is the bare mass and λ is the bare coupling.

In momentum space we define the disconnected field correlators to be

$$G(p_1, \ldots, p_n) = \langle \phi(p_1) \ldots \phi(p_n) \rangle\,, \qquad \phi(p) = \int d^3x\, \exp(i\, p \cdot x)\phi(x), \tag{E.43}$$

with odd correlators evaluating to zero. Due to momentum conservation each correlator carries a factor of $(2\pi)^3 \delta\left(\sum_{i=1}^n p_i\right)$.

The action of Equation (E.42) can be rewritten to express the relations between



(a) Two-loop vertex (b) One-loop vertex (c) No-loop vertex

Figure E.3: Fundamental ϕ^4 vertices for modes of interaction. The disconnected interaction (see Figure E.3a) will be ignored. The tadpole contribution (see Figure E.3b) and the standard vertex (see Figure E.3c) have to be taken into consideration.

the vertices and the propagators explicitly. We find

$$S = \frac{1}{2}\int_{ij} G_{ij}^{-1}\,\phi_i\,\phi_j + \frac{\lambda}{4!}\int_{ijkl} V_{ijkl}\,\phi_i\,\phi_j\,\phi_k\,\phi_l, \tag{E.44}$$

where we shorthand the notation to indicate an integration over the positions of the involved fields. The scalar propagator, represented by the Green's function G_{ij}^{-1}, as well as the potential V_{ijkl} are given by

$$G_{ij}^{-1} \equiv G^{-1}(x_i, x_j) = (-\partial_{x_i}^2 + m^2)\,\delta\,(x_i - x_j)\,, \tag{E.45}$$

$$V_{ijkl} \equiv \delta\,(x_i - x_j)\,\delta\,(x_i - x_k)\,\delta\,(x_i - x_l)\,. \tag{E.46}$$

The indices represent the vertices connected to the legs, e.g., two connected vertices to the two-point function, or four connected vertices to the four-point function.

The basic action can now be used to derive the elementary Feynman rules for this theory. In Figure E.3 we see the elementary diagrams derived from the action. The disconnected contribution is irrelevant for our purposes. The two-point (one-loop) and four-point functions can be used to construct a set of Schwinger-Dyson equations.

In a Schwinger-Dyson equation we express, e.g., the propagator $G(k)$ in terms of a series expansion of its bare propagator $G_0(k)$, using other diagrams representing higher-order corrections. This way we obtain, e.g.,

$$G(k) = G_0 + G_0\,\Sigma\,G_0 + G_0\,\Sigma\,G_0\,\Sigma\,G_0 + \ldots \tag{E.47}$$

$$= \sum_{n=0}^{\infty} G_0\,(\Sigma G_0)^n = \frac{1}{G_0^{-1} - \Sigma}, \tag{E.48}$$

which yields the non-perturbative propagator $G(k)$. The same scheme can be

applied to other quantities as well.

We start by defining the self-energy Σ_{ij} and the one-particle irreducible (1PI) four-point function Γ_{ijkl}. We hereby consider Feynman diagrams as a functional of their elements. For the bare self-energy we find

$$\Sigma_{ij} = -\frac{1}{2}\int_{kl} V_{ijkl} G_{kl} + \frac{1}{6}\int_{klmnop} V_{iklm}\, G_{kn}\, G_{lo}\, G_{mp}\, \Gamma_{nopj}, \tag{E.49}$$

where we denote the free propagator with G_{ij}. Again we abbreviate the integrals and propagators by using an index representing a field's position.

The bare 1PI four-point function can be expressed as

$$\Gamma_{ijkl} = V_{ijkl} + A_{ijkl} + B_{ijkl} + C_{ijkl}. \tag{E.50}$$

The defining equations for A_{ijkl}, B_{ijkl}, and C_{ijkl} represent a recursive scheme that starts to look similar to the previously discussed example of S-wave scattering in Section E.2.

By differentiating the vertex function from Equation (E.50) with respect to the full propagator G_{ij}, we get an expansion in terms of bold correlation functions. For the first derivative we obtain

$$\frac{\partial \Gamma_{ijkl}}{\partial G_{mn}} = \frac{1}{2}\left(V_{ijno}\, G_{op}\, \Gamma_{pmkl} - V_{ijmo}\, G_{op}\, \Gamma_{pnkl}\right) + \mathcal{O}\left(G^2\right). \tag{E.51}$$

It is sufficient to consider only the first-order, which contains a single bold propagator.

The recursive scheme is then built upon the following equations. We have

$$A_{ijkl} = a_{ijkl} + a_{ikjl} + a_{iljk}, \tag{E.52}$$

$$B_{ijkl} = b_{ijkl} + b_{ikjl} + b_{iljk}, \tag{E.53}$$

$$C_{ijkl} = c_{ijkl} + d_{ijkl}, \tag{E.54}$$

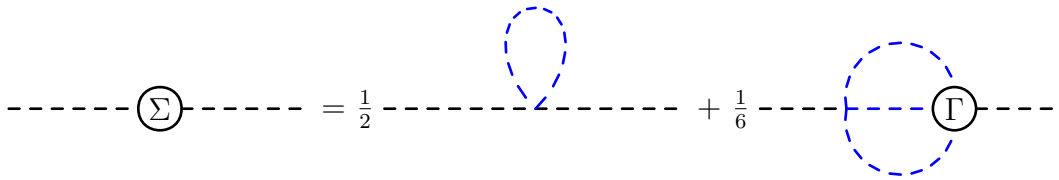
where the first two equations reference diagrams with partial permutations of indices. For a_{ijkl} we have

$$a_{ijkl} = -\frac{1}{2} V_{ijmn}\, G_{mo}\, G_{np}\, \Gamma_{opkl}. \tag{E.55}$$

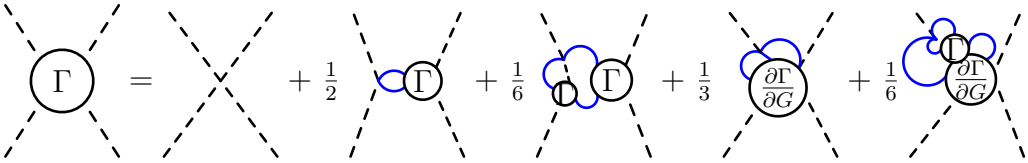
Similarly, for b_{ijkl} we get

$$b_{ijkl} = \frac{1}{6} V_{mino}\, G_{nq}\, G_{or}\, \Gamma_{qrjs}\, G_{st}\, \Gamma_{tklp}\, G_{pm}. \tag{E.56}$$

The terms C_{ijkl} look different than the previous two parts. For these terms we use



(a) Self-energy Σ_{ij}



(b) Four-point function Γ_{ijkl}

Figure E.4: Bold diagrammatic expansions for the self energy Σ_{ij} and the 1PI four-point function Γ_{ijkl} in ϕ^4 theory. The terms on the right hand side are expressed in terms of 1PI exact propagators (indicated by the blue lines).

the bold representation of the derivative term, i.e., we have

$$c_{ijkl} = -\frac{1}{3} V_{imno}\, G_{mp}\, G_{nq}\, G_{or} \frac{\partial \Gamma_{pjkl}}{\partial G_{qr}}, \tag{E.57}$$

$$d_{ijkl} = +\frac{1}{6} V_{imno}\, G_{mp}\, G_{ns}\, G_{ot}\, G_{qu}\, G_{rv}\, \Gamma_{stuv} \frac{\partial \Gamma_{pjkl}}{\partial G_{qr}}, \tag{E.58}$$

which concludes the full functional set required for a BDMC study.

The full set of equations can also be represented diagrammatically. In Figure E.4 the diagrammatic representations of Equation (E.49) (shown in Figure E.4a) and Equation (E.50) (shown in Figure E.4b) are illustrated. The usage of exact propagators has been indicated by blue color.

At this point we can take our knowledge about the series expansion and define the updates for our simulation.

E.3.2 Update Procedures

We have to impose some renormalization conditions by translating the Schwinger-Dyson equations into equations for renormalized correlation functions. We choose

$$G^{-1}(p^2 = 0) = m^2, \tag{E.59}$$

$$\left.\frac{\partial G^{-1}(p^2)}{\partial p^2}\right|_{p^2=0} = 1, \tag{E.60}$$

$$\Gamma(0,0,0,0) = m\,\lambda_R. \tag{E.61}$$

We call λ_R the renormalized coupling. The choice for m is arbitrary. Normally, m is expressed in units of the cutoff Λ, which is set to unity thus implying $m \ll 1$.

As previously we use a suitable distribution for generating the momenta. In this case we need to be close to the propagator $G(p^2)$. The distribution $\Omega_p(p)$ is thus given by

$$\Omega_p(p) = \frac{1}{p^2 + m^2}. \tag{E.62}$$

In the beginning we need to approximate the four-point function. We use the following set of equations. The four-point function itself can be approximated as

$$\Gamma(p_1, p_2, p_3, p_4) \approx \prod_{i=1}^{4} Z_R^{-1}(p_i^2 + m_R^2)\, G(p_1, p_2, p_3, p_4), \tag{E.63}$$

where Z_R is the so-called wave function renormalization constant. In our case we define Z_R over

$$Z_R = \left(1 + \left.\frac{\partial Y(p^2)}{\partial p^2}\right|_{p^2=0}\right)^{-1}, \tag{E.64}$$

with $Y(p^2)$ being given by the difference of the self-energy renormalization to the self-energy, i.e., $Y(p^2) = \Sigma(p^2 = 0) - \Sigma(p^2)$.

The full expression for $Y(p^2)$ is

$$\begin{aligned} Y(p^2) = \frac{\lambda Z_R}{6} \int \frac{d^3k}{(2\pi)^3} \frac{d^3k'}{(2\pi)^3} G_R(k)\, G_R(k') \times \\ \left(G_R(q)\, \Gamma_R(0, k, k', q) - G_R(q)\, \Gamma_R(p, k, k', q)\right). \end{aligned} \tag{E.65}$$

We use q to ensure the conservation of momenta at the vertex, i.e., $q = -\sum_i p_i$, where p_i are all (incoming) momenta for the vertex.

The renormalized Green's function is given by

$$G_R^{-1}(p^2) = Z_R \left(p^2 + Y(p^2)\right) + m^2, \tag{E.66}$$

which satisfies our renormalization conditions from Equation (E.59). The constructed system is finite at any order of perturbation.

We use two kinds of complementary updates. We either propose to create a pair of new momenta, thus inserting a new vertex, or we remove a pair of momenta by merging them. For the latter we will additionally increase the order of expansion. This allows us to go beyond first-order.

For inserting a new pair of momenta we only need to generate a single value p. The second momentum is then chosen to be $-p$ to satisfy the conservation of momentum. The transition probability for inserting a pair of new momenta to the existing n momenta is now given by

$$R_{n \to n+2} = (2\pi)^3 (n+1) \frac{c_{n,l}}{c_{n+2,l}} \frac{Z_R}{\Omega_p(p)}, \tag{E.67}$$

where we define the coefficients $c_{n,l}$ to be

$$c_{n,l} = \Gamma\left(\frac{n+1}{2} + l\right) \alpha^{2-n} \beta^{-l}, \tag{E.68}$$

with the gamma function $\Gamma(x)$.

The parameters α and β are dependent on the renormalization constant. We have

$$\alpha = \left(2\sqrt{Z_R}\right)^{-1}, \qquad \beta = \frac{\pi^3\, m^2}{Z_R}. \tag{E.69}$$

Similarly, the transition probability to merge pairs is given in dependence of these coefficients. By merging pairs we elevate the expansion order, thus increasing l.

The transition probability for merging two momenta p_1 and p_2 is given by

$$R_{n+2\to n} = \frac{1}{(2\pi)^6} \frac{c_{n,l}}{c_{n-2,l+1}} \frac{\Omega_p(p_1)\, \Omega_p(p_2)}{m^2}. \tag{E.70}$$

For simplicity we will leave the expansion coefficients unchanged. That way we do not have to introduce further updates or restarts. Otherwise, the set of updates would not be fully ergodic.

E.3.3 Evaluation and Results

There are a couple of interesting evaluations for simulating the three-dimensional ϕ^4 theory. We will focus on finding agreements with analytical results and estimates from other simulations. Most evaluations deal with the renormalized quantities.

In Figure E.5 we observe the asymptotic behavior of the renormalized coupling. This behavior is in excellent agreement with the existence of an infrared (IR) fixed point. The IR fixed point can be calculated using the ϵ-expansion and RG techniques [151]. The renormalized coupling $\lambda_R(\lambda)$ tends to a fixed value λ_R^*, when the bare coupling λ becomes large, i.e., in the limit $\lambda \to \infty$.

With a constant number of configurations we see an increasing absolute error for larger values of the bare coupling λ. This can be observed in the following plots. This matches our experience from the other examples, that a larger coupling results in more fluctuations and increases the error.

In Figure E.6 we see that our scheme indeed agrees with the analytic result for the lowest orders of perturbation theory. Our estimate for the renormalized mass at small couplings λ yields the same result as the one-loop of the mass renormalization does. This one-loop result is plotted using the dashed line. We use

$$m_R^2(\lambda) = m^2 + \frac{\lambda}{2} \int \frac{d^3p}{(2\pi)^3} \frac{1}{p^2 + m^2}. \tag{E.71}$$

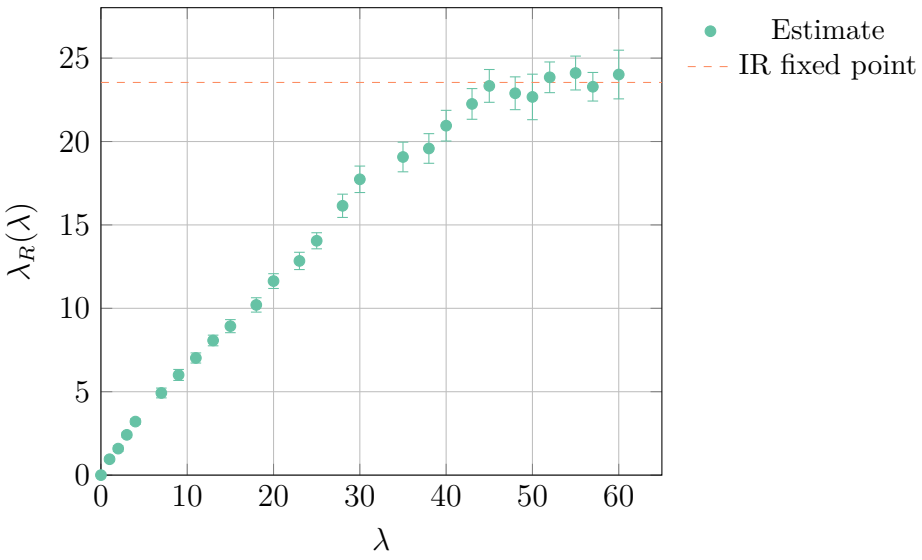


Figure E.5: Estimation for the dimensionless renormalized coupling constant λ_R in dependency of the bare coupling λ. The dashed line indicates the analytically predicted IR fixed point.

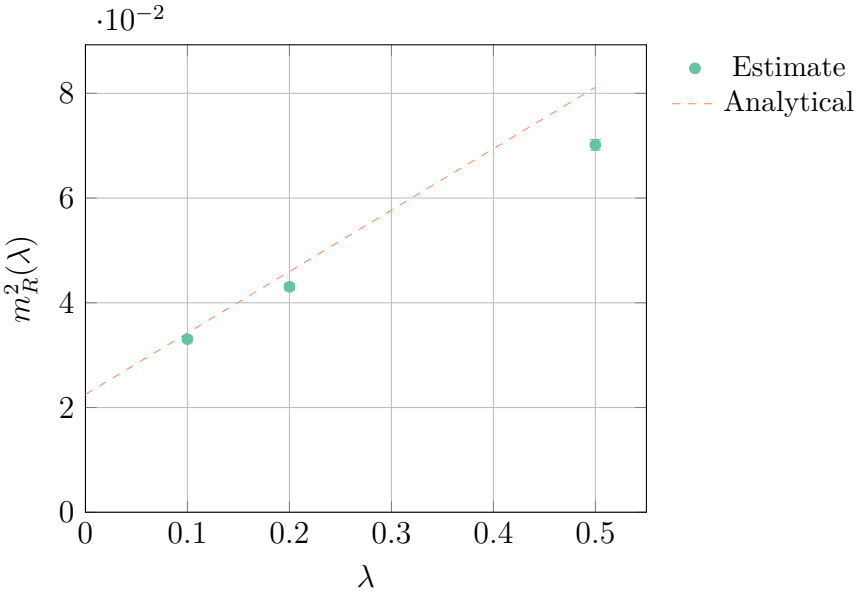


Figure E.6: Estimated renormalized mass m_R as a function of the bare coupling λ for $m = 0.15$. The dashed line corresponds to the one-loop contribution to $m_R^2(\lambda)$.

The divergent contribution can be solved by introducing a cutoff Λ. Using the cutoff-strategy we obtain

$$\int \frac{d^3p}{(2\pi)^3} \frac{1}{p^2 + m^2} \sim \Lambda - m \tan^{-1}\left(\frac{\Lambda}{m}\right), \tag{E.72}$$

where we set Λ to 1 in our simulations. This way we express the bare mass m in terms of Λ.

We can do the same evaluation with the wave function renormalization Z_R. Z_R is close to unity for the chosen values of λ. This is in full agreement with other simulations. The result does not agree with two-loop perturbative calculations of the self-energy diagrams, which indicates $Z_R < 1$ for $\lambda < 1$. The disagreement originates probably from some systematic errors in the field correlator approximations (zero-momentum limit).

In case of ϕ^4 theory for going to higher-orders we need to apply an appropriate resummation technique. It was already found that a Borel summation works for the described model [24].

The problem can be extended to four or more dimensions allowing to study the spectral density function of ϕ^4 [5]. One thing to note is that the result is indeed the non-perturbative spectral density function, as the Schwinger-Dyson equation already results in the full theory, not some truncated series. This makes BDMC in conjunction with resummation techniques quite appealing as a toolset for studying non-perturbative effects.

F

Source Codes

F.1 Diagrammatic Monte Carlo

For testing the constructed methods and algorithms novel applications had to be created. The following applications have directly or indirectly contributed to the results presented in this thesis. Each application is described briefly. More information is available on the project's website, which is given below the description. Permission will be granted upon request.

F.1.1 Feynman Diagram Sampling Application

The main application of the thesis, containing all relevant simulations and many other evaluations. Besides the integration example, the polaron problem, and anomalous magnetic moment, a few examples using BDMC are available. A set of evaluation tools is provided, as well as an implementation of the diagram generator.

- Languages: C++, JavaScript, Bash
- Target: Sequential, OMP, MPI
- Website: https://rqcd.ur.de:8443/FlorianRappl/DiagMC

F.1.2 Monte Carlo Simulation of He^3-He^4 Mixtures

Example simulation using Monte Carlo methods for the He^3-He^4 cooling mixture. The provided application simulates the 2D model for a given set of parameters. It tries to find the critical lines and the tricritical point.

- Language: C++
- Target: Sequential, OMP, MPI
- Website: https://rqcd.ur.de:8443/FlorianRappl/MixingCode

F.1.3 $SU(3)$ Hybrid Monte Carlo Simulation

A simple test code base to run an HMC simulation of a pure $SU(3)$ gauge theory in three dimensions on the lattice. The symmetry, which is spontaneously broken at the deconfinement transition, is the center symmetry. The Polyakov loop is not invariant under the center symmetry transformation.

- Language: C++
- Target: Sequential, OMP
- Website: https://rqcd.ur.de:8443/FlorianRappl/su3hmc

F.1.4 The Harmonic Oscillator on the Lattice

This is very simple HMC sample, which simulates the harmonic oscillator on the lattice. The program follows the influential paper by Creutz and Freedman. The paper discusses the anharmonic oscillator, which is not exactly solvable. The anharmonic term is part of the program and can be controlled by the user.

- Language: C++
- Target: Sequential, OMP
- Website: https://rqcd.ur.de:8443/FlorianRappl/harmosc

F.2 QPACE 2

For the QPACE 2 project most time has been spent on designing, constructing, and testing the hardware. Nevertheless, the full system would not work without software, nor could we run sophisticated tests. The following applications have been created to cover these areas. Each application is described briefly. More information is available on the project's website, which is given below the description.

F.2.1 KNC Benchmarks

The KNC benchmark suite features most of the benchmarks discussed in the previous sections. The whole application was written in such a fashion, that it may be reused or extended for follow-up projects. Most importantly, it produces executables for each of the available threading frameworks.

- Languages: C++, Bash
- Dependencies: MPSS, Vc, TBB
- Website: https://rqcd.ur.de:8443/FlorianRappl/kncbenchmarks

F.2.2 Linpack Benchmark

Entering the Top 500 requires running an optimized version of Linpack. Luckily, we have been in good contact with persons that dedicated much time into implementing such optimizations for the KNC. The given code is our base contains such KNC specific optimizations, combined with improvements tuned for the QPACE 2.

- Language: C
- Dependencies: MPSS, Intel MKL
- Website: https://rqcd.ur.de:8443/qpace2/Linpack

F.2.3 Burn-MIC Utility

In order to have reproducible cooling benchmarks, we required a tool that guarantees a certain load on the device delivered continuously for an arbitrary amount of time. Burn-MIC was designed to be that tool. It was also used in our acceptance tests for QPACE 2.

- Language: C, C++, Bash
- Dependencies: MPSS
- Website: https://rqcd.ur.de:8443/qpace2/BurnMic

F.2.4 Bricktest Runner

The QPACE 2 project was developed jointly with the company Eurotech. It was important to establish some kind of testing procedure that is accepted from all involved parties as an objective tool for verifying the project's goal. The given application runs a number of these acceptance tests.

- Languages: C++, Bash
- Dependencies: MPSS
- Website: https://rqcd.ur.de:8443/qpace2/Bricktest

F.2.5 Hardware Status Website

The physics department sustains quite a large machinery of supercomputers. This website aggregates useful information from all of these computers to show statistics and valuable information to potential users. Besides current load and temperatures, the website contains general (static) information.

- Languages: JavaScript, Jade
- Dependencies: Node.js, NPM
- Website: https://rqcd.ur.de:8443/qpace2/Status

F.2.6 Dual Rail Test

This code represents the sources for a performance test of using multiple rails with Intel MPI. It is mainly based on the OSU micro benchmarks (v4.4.1). It is possible to evaluate the (network) performance of the MICs. In this case the Intel MPI performance benchmarks are used.

- Languages: C++, Bash
- Dependencies: Intel MPI, OFA
- Website: https://rqcd.ur.de:8443/qpace2/DualRailTest

F.2.7 QPACE 2 Info Library

The provided source code exhibits a few helper to retrieve information about the corresponding Intel MICs, the host system, and the cooling circuit. It was designed for the experimental setup found in the LAGER III. The information of the cooling circuit is acquired from a small service.

- Language: C++
- Dependencies: MPSS
- Website: https://rqcd.ur.de:8443/FlorianRappl/qpaceinfo

References

[1] G. Aad et al. “Observation of a New Particle in the Search for the Standard Model Higgs Boson with the ATLAS Detector at the LHC”. In: *Phys. Lett.* B716 (2012), pp. 1–29. arXiv: 1207.7214 [hep-ex].

[2] R. Aaij et al. “Observation of $J/\psi p$ Resonances Consistent with Pentaquark States in $\Lambda_b^0 \to J/\psi K^- p$ Decays”. In: (2015). arXiv: 1507.03414 [hep-ex].

[3] R. Aaij et al. “Observation of Two New Ξ_b^- Baryon Resonances”. In: *Phys. Rev. Lett.* 114 (2015). arXiv: 1411.4849 [hep-ex].

[4] G. Aarts. “Developments in Lattice QCD for Matter at High Temperature and Density”. In: (2013). arXiv: 1312.0968 [hep-lat].

[5] N. Abbasi and A. Davody. “Monte Carlo Computation of Spectral Density Function in Real-Time Scalar Field Theory”. In: (2014). arXiv: 1410.7956 [hep-ph].

[6] D. Abts et al. “Energy Proportional Datacenter Networks”. In: *SIGARCH Comput. Archit. News* 38.3 (2010), pp. 338–347.

[7] B. J. Alder and T. E. Wainwright. “Studies in Molecular Dynamics. I. General Method”. In: *J. Chem. Phys* 31.2 (1959), pp. 459–466.

[8] M. G. Alford et al. “Color Superconductivity in Dense Quark Matter”. In: *Rev. Mod. Phys.* 80 (4 2008), pp. 1455–1515. arXiv: 0709.4635 [hep-ph].

[9] G. M. Amdahl. “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities”. In: *Proceedings, Spring Joint Computer Conference (AFIPS ’67)*. ACM, New York City, 1967, pp. 483–485.

[10] T. Aoyama et al. “Automated Calculation Scheme for α^n Contributions of QED to Lepton g-2: New Treatment of Infrared Divergence for Diagrams without Lepton Loops”. In: *Nucl. Phys.* B796 (2008), pp. 184–210. arXiv: 0709.1568 [hep-ph].

[11] T. Aoyama et al. “Complete Tenth-Order QED Contribution to the Muon g-2”. In: *Phys. Rev. Lett.* 109 (2012). arXiv: 1205.5370 [hep-ph].

[12] W. E. Arnoldi. “The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem”. In: *Q. Appl. Math* 9 (1951), pp. 17–29.

[13] P. Arts et al. “QPACE 2 and Domain Decomposition on the Intel Xeon Phi”. In: *Proceedings, 32nd International Symposium on Lattice Field Theory (Lattice 2014)*. Vol. LATTICE2014. 2015. arXiv: 1502.04025 [cs.DC].

[14] H. Baier et al. “QPACE: A QCD Parallel Computer Based on Cell Processors”. In: *Proceedings, 27th International Symposium on Lattice Field Theory (Lattice 2009)*. Vol. LAT2009. 2009. arXiv: 0911.2174 [hep-lat].

[15] J. Beringer et al. “Review of Particle Physics”. In: *Phys. Rev. D* 86 (1 2012).

[16] T. Berners-Lee, R. Cailliau, and B. Pollermann. “World-Wide Web: The Information Universe”. In: *CACM* 37.1 (1992), pp. 76–82.

[17] J. D. Bjorken and S. D. Drell. *Relativistic Quantum Mechanics*. 1st. McGraw-Hill College, Blacklick, 1965.

[18] J. C. R. Bloch and T. Wettig. “The QCD Sign Problem and Dynamical Simulations of Random Matrices”. In: *JHEP* 1105 (2011). arXiv: 1102.3715 [hep-lat].

[19] J. C. R. Bloch et al. “Nucleon Form Factors and a Nonpointlike Diquark”. In: *Phys. Rev. C* 60 (6 1999). arXiv: nucl-th/9907120.

[20] M. Blume, V. J. Emery, and Robert B. Griffiths. “Ising Model for the λ Transition and Phase Separation in He^3-He^4 Mixtures”. In: *Phys. Rev. A* 4 (3 1971), pp. 1071–1077.

[21] R. D. Blumofe et al. “Cilk: An Efficient Multithreaded Runtime System”. In: *SIGPLAN Not.* 30.8 (1995), pp. 207–216.

[22] M. Boninsegni, N. V. Prokof’ev, and B. V. Svistunov. “Worm Algorithm for Continuous-Space Path Integral Monte Carlo Simulations”. In: *Phys. Rev. Lett.* 96 (7 2006). arXiv: cond-mat/0510214.

[23] N. Brambilla et al. “QCD and Strongly Coupled Gauge Theories: Challenges and Perspectives”. In: *Eur. Phys. J.* C74.10 (2014). arXiv: 1404.3723 [hep-ph].

[24] P. V. Buividovich. “A Method for Resummation of Perturbative Series Based on the Stochastic Solution of Schwinger-Dyson Equations”. In: *Nucl. Phys.* B853 (2011). arXiv: 1104.3459 [hep-lat].

[25] P. V. Buividovich. "Feasibility of Diagrammatic Monte-Carlo based on weak-coupling expansion in asymptotically free theories: case study of $O(N)$ sigma-model in the large-N limit". In: *Proceedings, 33rd International Symposium on Lattice Field Theory (Lattice 2015)*. Vol. LATTICE2015. 2015. arXiv: 1510.06568 [hep-lat].

[26] E. A. Burovski et al. "Diagrammatic Quantum Monte Carlo for Two-Body Problems: Applied to Excitons". In: *Phys. Rev. Lett.* 87 (18 2001).

[27] S. Chatrchyan et al. "Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC". In: *Phys. Lett.* B716 (2012), pp. 30–61. arXiv: 1207.7235 [hep-ex].

[28] J. S. R. Chisholm. "Calculation of S-matrix elements". In: *Proc. Camb. Philos. Soc.* 48 (02 1952), pp. 300–315. ISSN: 1469-8064.

[29] Chroma Collaboration. *The Chroma Library for Lattice Field Theory*. Tech. rep. Accessed: 2015-10-12. JLab, 2011. URL: http://usqcd.jlab.org/usqcd-docs/chroma/.

[30] Wikipedia Contributors. *Microprocessor Chronology*. Online. Accessed: 2015-09-01. 2015. URL: https://en.wikipedia.org/w/index.php?title=Microprocessor_chronology&oldid=665843522.

[31] J. N. Corcoran, U. Schneider, and H.-B. Schüttler. "Perfect Stochastic Summation in High Order Feynman Graph Expansions". In: *Int. J. Mod. Phys. C* 17.11 (2006), pp. 1527–1549.

[32] M. J. Corden and D. Kreitzer. *Consistency of Floating-Point Results using the Intel Compiler*. Tech. rep. Accessed: 2015-11-04. Intel Corp., 2009. URL: https://software.intel.com/en-us/articles/consistency-of-floating-point-results-using-the-intel-compiler.

[33] Intel Corp. *Intel Xeon Phi Product Website*. Online. Accessed: 2015-10-05. 2015. URL: http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html.

[34] A. Davody. "Bold Diagrammatic Monte Carlo Study of ϕ^4 Theory". In: *Phys. Rev.* D88 (2013). arXiv: 1307.7699 [hep-lat].

[35] K. De Vogeleer et al. "The Energy/Frequency Convexity Rule: Modeling and Experimental Validation on Mobile Devices". In: *CoRR* abs/1401.4655 (2014). arXiv: 1401.4655 [cs.OH].

[36] T. DeGrand and C. DeTar. *Lattice Methods For Quantum Chromodynamics*. 1st. World Scientific Publishing, Singapore, 2006.

[37] T. DeGrand and F. Rappl. “Monte Carlo Simulation of Transition and Phase Separation in He^3-He^4 Mixtures”. Report of Independent Studies. 2010.

[38] R. L. Delgado, C. Hidalgo-Duque, and F. J. Llanes-Estrada. “To What Extent is Gluon Confinement an Empirical Fact?” In: *Few Body Syst.* 54 (2013), pp. 1705–1717. arXiv: 1106.2462 [hep-ph].

[39] F. Di Renzo and L. Scorzato. “Numerical Stochastic Perturbation Theory for full QCD”. In: *JHEP* 0410 (2004). arXiv: hep-lat/0410010.

[40] N. G. Diamantis and E. Manousakis. “Flat Histogram Diagrammatic Monte Carlo Method”. In: *Phys. Rev. E* 88 (4 2013). arXiv: 1306.6320 [cond-mat].

[41] J. Dokulil et al. “Efficient Hybrid Execution of C++ Applications using Intel Xeon Phi Coprocessor”. In: *CoRR* abs/1211.5530 (2012). arXiv: 1211.5530 [cs.DC].

[42] J. Dongarra, P. Luszczek, and A. Petitet. “The LINPACK Benchmark: Past, Present, and Future”. In: *Concurr. Comput. Pract. Exper.* 15 (9 2003), pp. 803–820.

[43] U. Drepper. *What Every Programmer Should Know About Memory*. Tech. rep. Accessed: 2015-12-10. Red Hat, Inc., 2007. URL: https://people.freebsd.org/~lstewart/articles/cpumemory.pdf.

[44] S. Duane et al. “Hybrid Monte Carlo”. In: *Phys. Lett. B* 195.2 (1987).

[45] W. Eckhardt et al. “591 TFLOPS Multi-trillion Particles Simulation on SuperMUC”. In: *Supercomputing*. Springer, Berlin, 2013, pp. 1–12.

[46] R. G. Edwards and B. Joó. “The Chroma Software System for Lattice QCD”. In: *Proceedings, 22nd International Symposium on Lattice Field Theory (Lattice 2004)*. Vol. 140. Elsevier, Philadelphia, 2005, pp. 832–834. arXiv: hep-lat/0409003.

[47] B. Efron. “Bootstrap Methods: Another Look at the Jackknife”. In: *Ann. Statist.* 7.1 (1979), pp. 1–26.

[48] J. Fang et al. “An Empirical Study of Intel Xeon Phi”. In: *CoRR* abs/1310.5842 (2013). arXiv: 1310.5842 [cs.DC].

[49] W. Feng and K. W. Cameron. *The Green500 List*. Online. Accessed: 2015-09-30. 2015. URL: http://www.green500.org.

[50] R. P. Feynman. “Mathematical Formulation of the Quantum Theory of Electromagnetic Interaction”. In: *Phys. Rev.* 80 (3 1950), pp. 440–457.

[51] R. P. Feynman. “Simulating Physics with Computers”. In: *Int. J. Theor. Phys.* 21.6 (1982), pp. 467–488.

[52] R. P. Feynman. “Space-Time Approach to Quantum Electrodynamics”. In: *Phys. Rev.* 76 (6 1949), pp. 769–789.

[53] R. P. Feynman. *Statistical Mechanics: A Set Of Lectures*. 2nd. Westview Press, Boulder, 1998.

[54] M. Frigo et al. “Reducers and Other Cilk++ Hyperobjects”. In: *Proceedings, 21st Annual Symposium on Parallelism in Algorithms and Architectures*. ACM, New York City, 2009, pp. 79–90.

[55] A. Frommer et al., eds. *Numerical Challenges in Lattice Quantum Chromodynamics*. Proceedings, Joint Interdisciplinary Workshop. Springer, Berlin, 1999.

[56] K. Fukushima and T. Hatsuda. “The Phase Diagram of Dense QCD”. In: *Rept. Prog. Phys.* 74 (2011), p. 014001. arXiv: 1005.4814 [hep-ph].

[57] S. Gai and C. DeSanti. *I/O Consolidation in the Data Center*. 1st. Cisco Press, Indianapolis, 2009.

[58] C. Gattringer and C. B. Lang. *Quantum Chromodynamics on the Lattice*. 1st. Springer, Berlin, 2009.

[59] J. E. Gentle. *Computational Statistics*. 1st. Springer, Berlin, 2009.

[60] R. Geva. *Elemental Functions: Writing Data Parallel Code in C/C++ using Intel Cilk Plus*. Tech. rep. Accessed: 2015-11-04. Intel Corp., 2011. URL: https://software.intel.com/en-us/articles/elemental-functions-writing-data-parallel-code-in-cc-using-intel-cilk-plus.

[61] G. Goldrian et al. “QPACE: Quantum Chromodynamics Parallel Computing on the Cell Broadband Engine”. In: *CiSE* 10.6 (2008), pp. 46–54.

[62] W. Gong, Z. Nagy, and D. E. Soper. “Direct Numerical Integration of One-Loop Feynman Diagrams for N-Photon Amplitudes”. In: *Phys. Rev.* D79 (2009). arXiv: 0812.3686 [hep-ph].

[63] D. J. Gross and F. Wilczek. “Ultraviolet Behavior of Non-Abelian Gauge Theories”. In: *Phys. Rev. Lett.* 30 (26 1973), pp. 1343–1346.

[64] R. Gupta. “Introduction to Lattice QCD: Course”. In: (1997). arXiv: hep-lat/9807028.

[65] J. Hakkinen and H. Kharraziha. “COLOR: A Computer Program for QCD Color Factor Calculations”. In: *Comput. Phys. Commun.* 100 (1997), pp. 311–321. arXiv: hep-ph/9603229.

[66] D. Hanneke, S. Fogwell Hoogerheide, and G. Gabrielse. “Cavity Control of a Single-Electron Quantum Cyclotron: Measuring the Electron Magnetic Moment”. In: *Phys. Rev. A* 83 (5 2011). arXiv: 1009.4831 [physics.atom-ph].

[67] R. Harlander and M. Steinhauser. “Automatic Computation of Feynman Diagrams”. In: *Prog. Part. Nucl. Phys.* 43 (1999), pp. 167–228. arXiv: hep-ph/9812357.

[68] W. K. Hastings. “Monte Carlo Sampling Methods Using Markov Chains and Their Applications”. In: *Biometrika* 57.1 (1970), pp. 97–109.

[69] A. Heinecke et al. “Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems Based on Intel Xeon Phi Coprocessor”. In: *Proceedings, International Parallel and Distributed Processing Symposium (IPDPS 2013)*. IEEE Computer Society, Los Alamitos, 2013, pp. 126–137.

[70] P. W. Higgs. “Broken Symmetries and the Masses of Gauge Bosons”. In: *Phys. Rev. Lett.* 13 (16 1964), pp. 508–509.

[71] A. Hinrichs et al. “The Curse of Dimensionality for Numerical Integration of Smooth Functions”. In: (2012). arXiv: 1211.0871 [math.NA].

[72] J. Hofmann et al. “Performance Analysis of the Kahan-enhanced Scalar Product on Current Multicore Processors”. In: *CoRR* abs/1505.02586 (2015). arXiv: 1505.02586 [cs.PF].

[73] J. Jeffers and J. Reinders. *Intel Xeon Phi Coprocessor High Performance Programming*. 1st. Morgan Kaufmann, San Francisco, 2013.

[74] F. Jegerlehner. “Essentials of the Muon g-2”. In: *Acta Phys. Polon.* B38 (2007). arXiv: hep-ph/0703125.

[75] F. Jegerlehner and A. Nyffeler. “The Muon g-2”. In: *Phys. Rept.* 477 (2009), pp. 1–110. arXiv: 0902.3360 [hep-ph].

[76] M. Kaku. *Quantum Field Theory: A Modern Introduction*. 1st. Oxford University Press, Oxford, 1993.

[77] M. Kanellos. *Moore’s Law to Roll on for Another Decade*. Tech. rep. Accessed: 2015-11-04. CNET, 2003. URL: http://www.cnet.com/news/moores-law-to-roll-on-for-another-decade/.

[78] E. T. Kenwright. *Ear Clamp*. Google Patents. WO Patent App. PCT / EP2013 / 067, 023. 2015.

[79] T. Kinoshita. “Everyone Makes Mistakes: Including Feynman”. In: *J. Phys.* G29 (2003), pp. 9–22. arXiv: hep-ph/0101197.

[80] T. Klahn, D. Blaschke, and R. Lastowiecki. “Compact Stars, Heavy Ion Collisions, and Possible Lessons For QCD at Finite Densities”. In: *Acta Phys. Polon. Supp.* B5 (2012), pp. 757–772. arXiv: 1111.6889 [nucl-th].

[81] J. B. Kogut, M. P. Lombardo, and D. K. Sinclair. “Quenched QCD at Finite Density”. In: *Phys. Rev.* D51 (1995), pp. 1282–1291. arXiv: hep-lat/9401039.

[82] E. Kozik et al. “Diagrammatic Monte Carlo for Correlated Fermions”. In: *Europhys. Lett.* 90.1 (2010). arXiv: 0907.0863 [cond-mat].

[83] M. Kretz and V. Lindenstruth. “Vc: A C++ library for explicit vectorization”. In: *Softw. Pract. Exper.* 42.11 (2012), pp. 1409–1430.

[84] S. Laporta. “Calculation of Feynman Integrals by Difference Equations”. In: *Acta Phys. Polon.* B34 (2003), pp. 5323–5334. arXiv: hep-ph/0311065.

[85] S. Laporta. “High Precision Calculation of Multiloop Feynman Integrals by Difference Equations”. In: *Int. J. Mod. Phys.* A15 (2000), pp. 5087–5159. arXiv: hep-ph/0102033.

[86] D. H. Lehmer. “Mathematical Methods in Large-Scale Computing Units”. In: *Proceedings, 2nd Symposium on Large-Scale Digital Calculating Machinery*. Harvard University Press, Cambridge, 1951, pp. 141–146.

[87] M. J. Levine and Jon Wright. “Anomalous Magnetic Moment of the Electron”. In: *Phys. Rev. D* 8 (9 1973), pp. 3171–3179.

[88] J. Lin, A. Nukada, and S. Matsuoka. “Modeling Gather and Scatter with Hardware Performance Counters for Xeon Phi”. In: *Proceedings, 15th IEEE International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE Computer Society, Washington, 2015, pp. 713–716.

[89] J. Liu et al. “Microbenchmark Performance Comparison of High-Speed Cluster Interconnects”. In: *Micro, IEEE* 24.1 (2004), pp. 42–51.

[90] M. P. Lombardo. “Lattice QCD at Finite Temperature and Density”. In: *Mod. Phys. Lett.* A22 (2007), pp. 457–472. arXiv: hep-lat/0509180.

[91] M. Lüscher. “Instantaneous Stochastic Perturbation Theory”. In: *JHEP* 04 (2015). arXiv: 1412.5311 [hep-lat].

[92] Y. M. Makeenko and A. A. Migdal. “Exact equation for the loop average in multicolor QCD”. In: *Phys. Lett. B* 88.1 (1979), pp. 135–137.

[93] G. Marsaglia. “Random Number Generators”. In: *J. Mod. App. Stat. Meth.* 2.1 (2003), pp. 2–13.

[94] M. Matsumoto and T. Nishimura. “Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator”. In: *ACM Trans. Model. Comput. Simul.* 8.1 (1998), pp. 3–30.

[95] K. M. Maung et al. “Equivalence of Minkowski and Euclidean Field Theory Solutions”. In: *Proceedings, 5th International Conference on Quark Confinement and the Hadron Spectrum*. World Scientific Publishing, Singapore, 2003, pp. 285–287.

[96] J. Mechalas. *Intel Digital Random Number Generator Software Implementation Guide*. Tech. rep. Accessed: 2015-11-04. Intel Corp., 2014. URL: https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide.

[97] R. Mertig, M. Bohm, and A. Denner. “FEYN CALC: Computer Algebraic Calculation of Feynman Amplitudes”. In: *Comput. Phys. Commun.* 64 (1991), pp. 345–359.

[98] N. Metropolis et al. “Equation of State Calculations by Fast Computing Machines”. In: *J. Chem. Phys* 21.6 (1953), pp. 1087–1092.

[99] N. Meyer et al. “iDataCool: HPC with Hot-Water Cooling and Energy Reuse”. In: *Supercomputing*. Vol. abs/1309.4887. Springer, Berlin, 2013, pp. 383–394. arXiv: 1309.4887 [cs.DC].

[100] Mikalor. *Mikalor Product Information*. Tech. rep. Accessed: 2015-11-05. Mikalor Group, 2015. URL: http://www.mikalor.com.

[101] A. S. Mishchenko et al. “Comprehensive Study of Fröhlich Polaron”. In: *Int. J. Mod. Phys. B* (1999). arXiv: cond-mat/9910025.

[102] A. S. Mishchenko et al. “Diagrammatic Quantum Monte Carlo Study of the Fröhlich Polaron”. In: *Phys. Rev. B* 62 (10 2000), pp. 6317–6336.

[103] Y. Nakamura et al. “Lattice QCD Applications on QPACE”. In: *Proceedings, International Conference on Computational Science (ICCS 2011)*. Elsevier, Philadelphia, 2011, pp. 841–851. arXiv: 1103.1363 [hep-lat].

[104] K. Nakazawa et al. “CP-PACS: A massively parallel processor at the University of Tsukuba”. In: *Parallel Comput.* 25.13–14 (1999), pp. 1635–1661.

[105] J. von Neumann. “First Draft of a Report on the EDVAC”. In: *IEEE Ann. Hist. Comput.* 15.4 (1993), pp. 27–75.

[106] A. Nowak et al. “Does the Intel Xeon Phi Processor Fit HEP Workloads?” In: *J. Phys. Conf. Ser.* 513.5 (2014).

[107] B. Odom et al. “New Measurement of the Electron Magnetic Moment Using a One-Electron Quantum Cyclotron”. In: *Phys. Rev. Lett.* 97 (3 2006).

[108] K. Osterwalder and R. Schrader. “Axioms for Euclidean Green’s functions”. In: *Comm. Math. Phys.* 31.2 (1973), pp. 83–112.

[109] P. S. Pacheco. *Parallel Programming with MPI*. 1st. Morgan Kaufmann, San Francisco, 1996.

[110] C. D. Palmer and M. E. Carrington. “A General Expression for Symmetry Factors of Feynman Diagrams”. In: *Can. J. Phys.* 80 (2002), pp. 847–854. arXiv: hep-th/0108088.

[111] T. Pang. *An Introduction to Computational Physics*. 1st. Cambridge University Press, 2006.

[112] M. E. Peskin and D. V. Schroeder. *An Introduction To Quantum Field Theory*. 1st. Westview Press, Boulder, 1995.

[113] L. Pollet. “Recent Developments in Quantum Monte-Carlo Simulations with Applications for Cold Gases”. In: *Rep. Prog. Phys.* 75 (2012). arXiv: 1206.0781 [cond-mat].

[114] L. Pollet, N. V. Prokof’ev, and B. V. Svistunov. “Regularization of Diagrammatic Series with Zero Convergence Radius”. In: *Phys. Rev. Lett.* 105 (21 2010). arXiv: 1006.4519 [cond-mat].

[115] L. Pollet et al. “Optimal Monte Carlo Updating”. In: *Phys. Rev. E* 70.5 (2004). arXiv: cond-mat/0405150.

[116] N. V. Prokof’ev and B. V. Svistunov. “Bold Diagrammatic Monte Carlo: A Generic Sign-Problem Tolerant Technique for Polaron Models and Possibly Interacting Many-Body Problems”. In: *Phys. Rev. B* 77 (12 2008).

[117] N. V. Prokof’ev and B. V. Svistunov. “Bold Diagrammatic Monte Carlo: A Generic Technique for Polaron (and Many-Body?) Problems”. In: (2008). arXiv: 0801.0911 [cond-mat].

[118] N. V. Prokof’ev and B. V. Svistunov. “Bold Diagrammatic Monte Carlo Technique: When the Sign Problem Is Welcome”. In: *Phys. Rev. Lett.* 99.25 (2007). arXiv: cond-mat/0702555.

[119] N. V. Prokof’ev and B. V. Svistunov. “Fermi-Polaron Problem: Diagrammatic Monte Carlo Method for Divergent Sign-Alternating Series”. In: *Phys. Rev. B* 77 (2 2008). arXiv: 0707.4259 [cond-mat].

[120] N. V. Prokof’ev and B. V. Svistunov. “Polaron Problem by Diagrammatic Quantum Monte Carlo”. In: *Phys. Rev. Lett.* 81 (1998), pp. 2514–2517. arXiv: cond-mat/9804097.

[121] N. V. Prokof’ev and B. V. Svistunov. “Worm Algorithm for Problems of Quantum and Classical Statistics”. In: (2009). arXiv: 0910.1393 [cond-mat].

[122] R. Rahman. *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. 1st. Apress, New York City, 2013.

[123] J. Ray, A. Pinar, and C. Seshadhri. "Are We There Yet? When to Stop a Markov Chain While Generating Random Graphs". In: *CoRR* abs/1202.3473 (2012). arXiv: 1202.3473 [cs.SI].

[124] J. Reinders. *An Overview of Programming for Intel Xeon Processors and Intel Xeon Phi Coprocessors*. Tech. rep. Accessed: 2015-12-10. Intel Corp., 2012. URL: http://software.intel.com/en-us/articles/an-overview-of-programming-for-intel-xeon-processors-and-intel-xeon-phi-coprocessors.

[125] V. Saad and M. H. Schultz. "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems". In: *SIAM J. Sci. Statist. Comput.* 7 (1986), pp. 856–869.

[126] C. Sasaki. "The QCD Phase Diagram from Chiral Approaches". In: *Proceedings, 21st International Conference on Ultra-Relativistic Nucleus-Nucleus Collisions (Quark Matter 2009)*. Vol. A830. Elsevier, Philadelphia, 2009, pp. 649C–656C. arXiv: 0907.4713 [hep-ph].

[127] E. Saule, K. Kaya, and Ü V. Çatalyürek. "Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi". In: *CoRR* abs/1302.1078 (2013). arXiv: 1302.1078 [cs.PF].

[128] R. T. Scalettar, D. J. Scalapino, and R. L. Sugar. "New Algorithm for the Numerical Simulation of Fermions". In: *Phys. Rev. B* 34.11 (1986).

[129] M. Scheffer et al. "Early-Warning Signals for Critical Transitions". In: *Nature* 461.7260 (2009), pp. 53–59. (Visited on 11/04/2015).

[130] T. Schäfer. "Phases of QCD". In: *Proceedings, 20th Annual Hampton University Graduate Studies Program (HUGS 2005)*. World Scientific, Singapore, 2005. arXiv: hep-ph/0509068.

[131] A. Schirotzek et al. "Observation of Fermi Polarons in a Tunable Fermi Liquid of Ultracold Atoms". In: *Phys. Rev. Lett.* 102 (23 2009).

[132] C. Schmidt. "Lattice QCD at Finite Density". In: *Proceedings, 24th International Symposium on Lattice Field Theory (Lattice 2006)*. Vol. LAT2006. 2006, p. 021. arXiv: hep-lat/0610116.

[133] M. Schmitt. "Diagrammatic Determinantal Quantum Monte Carlo Calculations on GPUs". MA thesis. School of Physics and Astronomy, University of Edinburgh, 2013.

[134] W. Siegel. "Fields". In: (1999). arXiv: hep-th/9912205.

[135] L. Soares and M. Stumm. “FlexSC: Flexible System Call Scheduling with Exception-less System Calls”. In: *Proceedings, 9th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, Berkeley, 2010, pp. 1–8.

[136] K. Splittorff and B. Svetitsky. “The Sign Problem via Imaginary Chemical Potential”. In: *Phys. Rev.* D75 (2007). arXiv: hep-lat/0703004.

[137] M. Srednicki. *Quantum Field Theory*. 1st. Cambridge University Press, 2007.

[138] H. Stengel et al. “Quantifying Performance Bottlenecks of Stencil Computations Using the Execution-Cache-Memory Model”. In: *Proceedings, 29th ACM on International Conference on Supercomputing*. ACM, New York City, 2015, pp. 207–216. arXiv: 1410.5010 [cs.PF].

[139] E. Strohmaier et al. *TOP500 Supercomputer Sites*. Online. Accessed: 2015-09-30. 2015. URL: http://www.top500.org.

[140] S. Sumimoto et al. “A Scalable Communication Layer for Multi-dimensional Hyper Crossbar Network Using Multiple Gigabit Ethernet”. In: *Proceedings, 20th Annual International Conference on Supercomputing*. ACM, New York City, 2006, pp. 107–115.

[141] M. H. Thoma. “QCD Perturbation Theory at Finite Temperature / Density and Its Application”. In: *Proceedings, 13th International Conference on Ultra-Relativistic Nucleus-Nucleus Collisions (Quark Matter ’97)*. Vol. A638. Elsevier, Philadelphia, 1998, pp. 317C–328C. arXiv: hep-ph/9801266.

[142] S. Trebst et al. “Ensemble Optimization Techniques for the Simulation of Slowly Equilibrating Systems”. In: *Proceedings, 19th Annual Workshop on Computer Simulation Studies in Condensed Matter Physics*. Springer, Berlin, 2006, pp. 33–47. arXiv: cond-mat/10606005.

[143] J. Treibig, G. Hager, and G. Wellein. “LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments”. In: *CoRR* abs/1004.4431 (2010). arXiv: 1004.4431 [cs.DC].

[144] J. W. Tukey. “Bias and Confidence in Not-Quite Large Samples”. In: *Ann. Math. Statist.* 29.2 (1958), pp. 614–623.

[145] K. W. Usemann. *Entwicklung von Heizungs- und Lüftungstechnik zur Wissenschaft: Hermann Rietschel, Leben und Werk*. 1st. Oldenbourg Verlag, München, 1993.

[146] R. S. Van Dyck, P. B. Schwinberg, and H. G. Dehmelt. "New High-Precision Comparison of Electron and Positron g Factors". In: *Phys. Rev. Lett.* 59 (1 1987), pp. 26–29.

[147] K. Van Houcke et al. "Diagrammatic Monte Carlo". In: *Proceedings, 21st Workshop on Computer Simulations Studies in Condensed Matter Physics*. Vol. 6. Elsevier, Philadelphia, 2010, pp. 95–105. arXiv: 0802.2923 [cond-mat].

[148] S. A. Volkov. "Subtraction Procedure for Calculation of Anomalous Magnetic Moment of Electron in QED and its Application to Numerical Computation at 3-loop Level". In: (2015). arXiv: 1507.06435 [hep-ph].

[149] S. Weinzierl. *Introduction to Monte Carlo Methods*. Tech. rep. NIKHEF, 2000. arXiv: hep-ph/0006269.

[150] K. G. Wilson. "Confinement of Quarks". In: *Phys. Rev. D* 10 (8 1974), pp. 2445–2459.

[151] K. G. Wilson and M. E. Fisher. "Critical Exponents in 3.99 Dimensions". In: *Phys. Rev. Lett.* 28 (4 1972), pp. 240–243.

[152] U. Wolff. "Monte Carlo Errors with Less Errors". In: *Comput. Phys. Commun.* 156 (2004), pp. 143–153. arXiv: hep-lat/0306017.

[153] B. Xiao, H. Wang, and S.-H. Zhu. "A Simple Algorithm for Automatic Feynman Diagram Generation". In: *Comput. Phys. Commun.* 184 (2013), pp. 1966–1972. arXiv: 1209.0949 [hep-ph].

[154] Intel Developer Zone. *Intel Xeon Phi Coprocessor System Software Developers Guide*. Tech. rep. Accessed: 2015-12-10. Intel Corp., 2012. URL: http://software.intel.com/sites/default/files/managed/b5/83/intel-xeon-phi-systemssoftwaredevelopersguide.pdf.

Acknowledgments

No thesis has ever been written in a vacuum. My work does not represent an exception. I was lucky to have talented and motivated people around me, who could help me in many different ways.

First and foremost, I want to express my gratitude to my supervisor Prof. Tilo Wettig. He gave me enough freedom to follow my own research and was always accessible for inspiring discussions. Also Dr. Jacques Bloch played a crucial role in guiding me through the various problems I faced.

Especially in the beginning I received a lot of help from experts in the field of Diagrammatic Monte Carlo. I want to emphasize the contributions of Prof. Lode Pollet from the LMU Munich and Prof. Nikolay Prokof'ev from the University of Massachusetts Amherst. Their hints have been very important.

During my work I looked at different models and experimented with various methods. I want to thank Dr. Pavel Buividovich for sending me information and being open for discussions. Additionally, Dr. Christoph Lehner has to be mentioned for giving me access to the source code of his project PhySyHCAl.

One of my projects during the dissertation dealt with the computation of the λ_3 coefficient from relativistic hydrodynamics. I am very thankful to Prof. Andreas Schäfer that I got the opportunity to work on such an exciting project. During the project I had fruitful discussions with Prof. Laurence Yaffe from the University of Washington and Prof. Matthias Kaminski, who is now working at the University of Alabama.

This is my chance to mention Dr. Simon Mages, who has not only been my research partner for the λ_3 project, but a close friend and very skilled co-worker for some projects, which we completed in our rare spare time. We certainly learned a lot during all these very successful projects. We worked together in the QPACE 2 team. His suggestions for improving this thesis have been very helpful.

As far as the QPACE 2 team is concerned I am sure to forget one or the other important person. Nevertheless, I personally had most contact with Dr. Nils Meyer, Dr. Robert Lohmayer, Dr. Stefan Solbrig, and my former office mate Bernhard Mendl. The administrative support from Peter Georg regarding the su-

percomputing infrastructure has been irreplaceable. I think the QPACE 2 project would have been much worse without these very skilled people. Furthermore, I want to mention Dr. Simon Heybrock and Benjamin Gläßle, who rewrote and tuned many parts of the LQCD software for the Intel Xeon Phi.

Thanks to Dr. Alexander Manashov for providing a productive work environment. Finally, I have to mention Jakob Simeth, as well as my fellow students Rudolf Rödl, Christian Gradl, Markus Schwemmer, and Niki Kilbertus. They motivated me and always had an open ear for my concerns. Thanks for the interesting discussions and guidance.

I want to thank the Studienstiftung des deutschen Volkes for their financial and ideational support. I certainly would not have made so much progress in these years without the scholarship.

Last but not least, I want to thank my family, especially my parents and my wife Simone, for their continuing support.