**FORSCHUNGSZENTRUM JÜLICH GmbH**
**Zentralinstitut für Angewandte Mathematik**
**D-52425 Jülich, Tel. (02461) 61-6402**

Interner Bericht

# Beiträge zum Wissenschaftlichen Rechnen
# Ergebnisse des
# Gaststudentenprogramms 2006
# des John von Neumann-Instituts
# für Computing

*Rüdiger Esser (Hrsg.)*

FZJ-ZAM-IB-2006-14

November 2006

(letzte Änderung: 1. 11. 2006)

# Vorwort

Die Ausbildung im Wissenschaftlichen Rechnen ist neben der Bereitstellung von Supercomputer-Leistung und der Durchführung eigener Forschung eine der Hauptaufgaben des John von Neumann-Instituts für Computing (NIC) und hiermit des ZAM als wesentlicher Säule des NIC. Um den akademischen Nachwuchs mit verschiedenen Aspekten des Wissenschaftlichen Rechnens vertraut zu machen, führte das ZAM in diesem Jahr zum siebten Mal während der Sommersemesterferien ein Gaststudentenprogramm durch. Entsprechend dem fächerübergreifenden Charakter des Wissenschaftlichen Rechnens waren Studenten der Natur- und Ingenieurwissenschaften, der Mathematik und Informatik angesprochen. Die Bewerber mussten das Vordiplom abgelegt haben und von einem Professor empfohlen sein. Die zehn vom NIC ausgewählten Teilnehmer kamen für zehn Wochen, vom 7. August bis 13. Oktober 2006, ins Forschungszentrum. Sie beteiligten sich hier an den Forschungs- und Entwicklungsarbeiten des ZAM und der NIC Forschergruppe Computer-gestützte Biologie und Biophysik. Sie wurden jeweils einem oder zwei Wissenschaftler zugeordnet, der mit ihnen zusammen eine Aufgabe festlegte und sie bei der Durchführung anleitete.

Die Gaststudenten und ihre Betreuer waren:

| | |
|---|---|
| Mathias Aust | Guido Arnold, Marcus Richter |
| Mohcine Chraibi | Armin Seyfried |
| Falk Eilenberger | Paul Gibbon |
| Sebastian Höfener | Thomas Müller |
| Florian Janoschek | Godehard Sutmann |
| Bernhard Kühnel | Holger Dachsel, Ivo Kabadshow |
| Martin Magiera | Ulrich Hansmann, Thomas Neuhaus |
| Frank Schmidt | Bernhard Steffen, Oliver Bücker |
| Michèle Wandelt | Bernhard Steffen |
| Markus Weigel | Uwe Pietrzyk, IME |

Zu Beginn ihres Aufenthalts erhielten die Gaststudenten eine viertägige Einführung in die Programmierung und Nutzung der Parallelrechner im ZAM. Um den Erfahrungsaustausch untereinander zu fördern, präsentierten die Gaststudenten am Ende ihres Aufenthalts ihre Aufgabenstellung und die erreichten Ergebnisse. Sie verfassten zudem Beiträge mit den Ergebnissen für diesen Internen Bericht des ZAM. Wir danken den Teilnehmern für ihre engagierte Mitarbeit - schließlich haben sie geholfen, einige aktuelle Forschungsarbeiten weiterzubringen - und den Betreuern, die tatkräftige Unterstützung dabei geleistet haben, insbesondere Bernd Mohr und Boris Orth, die den Einführungskurs gehalten haben. Ebenso danken wir allen, die im ZAM und der Verwaltung des Forschungszentrums bei Organisation und Durchführung des diesjährigen Gaststudentenprogramms mitgewirkt haben. Besonders hervorzuheben ist die finanzielle Unterstützung durch den Verein der Freunde und Förderer des FZJ und die Firma IBM. Es ist beabsichtigt, das erfolgreiche Programm künftig fortzusetzen, schließlich ist die Förderung des wissenschaftlichen Nachwuchses dem Forschungszentrum ein besonderes Anliegen. Weitere Informationen über das Gaststudentenprogramm, auch die Ankündigung für das kommende Jahr, findet man unter http://www.fz-juelich.de/zam/gaststudenten.

Jülich, November 2006 Rüdiger Esser

# Inhalt

# Advanced Memory Access Scheme for Quantum Computer Simulation

Mathias Aust

Universität Leipzig
Institut für Theoretische Physik

E-mail: aust@itp.uni-leipzig.de

**Abstract:** Starting from an already parallelized simulation code for an ideal quantum computer, we implement a more flexible communication scheme. We give a brief introduction into the necessary concepts and notations for quantum computers and discuss the problems of quantum computer simulations. The fundamental concept for parallelization of the code is explained. Based on this concept benefits and problems of the old and new communication schemes are discussed. Finally, both schemes are compared in different benchmarks.

## Introduction

Since Peter Shor's discovery of an efficient quantum algorithm for the factorization of huge numbers, quantum computers have raised large interest among physicists, mathematicians and computer scientists. Though NMR-based quantum computers consisting of up to 7 qubits have been realized in laboratories, no scalable quantum computer, that could perform algorithms on many qubits, has been developed yet.

However, we wish to test the existing algorithms and error-correction codes on their robustness against different error models. This makes the simulation of quantum computers on classical ones necessary, though this problem is bound to be inefficient. This has been shown by Feynman in his 1982 paper [1], where he argued that classical systems cannot efficiently simulate the evolution of quantum systems due to the exponential growth of necessary variables with the system size.

In order to master this enormous task of simulating a quantum computer, we will need to optimize our code as much as possible though the gain may be frustratingly small.

## Quantum Computers

This section gives a short introduction to the basic concepts and notations necessary to describe a quantum computer. For an overview over the different aspects of quantum computers and for more details see Los Alamos Science [2].

Note first that any computational model depends on its physical realization. In other words: For any computation, that can be performed in our universe, there must exist a physical device, that is able to perform this calculation. Thus the laws of physics dominating this device determine what is computable.

From this point of view a calculation is formulated as follows:

**Input:** Preparation of the initial state of a physical system

**Algorithm:** Time-evolution of the system

**Output:** Measurement and interpretation of the final state

The most fundamental physical laws are quantum mechanical. We want to include these laws in our computational model.

The central features of quantum mechanics can be summarized as the superposition principle, a deterministic (unitary, reversible) time-evolution and the restriction to projective (probabilistic) measurements. Especially the superposition principle allows us to solve problems on quantum computers, which cannot be tract-ed by classical ones, using massive parallelization in the following form:

1. Produce a superposition of *all* possible inputs.aust/
2. Execute some sequence of transformations on the state.
3. Interference between the different outputs reveals a global feature. $\rightarrow$ Measure!

A prominent example for such an algorithm is the factoring algorithm by Shor, which can decompose a natural number into its two Co prime factors in polynomial time with respect to the number of digits. The best classical algorithm known for this problem take exponential time.

*Qubits*

The smallest piece of classical information is the bit. It can be realized by a physical system with only two possible states $0$ and $1$. According to the postulates of quantum mechanics, a system with different allowed states can also exist in a superposition of these states. The two states of the classical bit become the orthonormal basis in the 1-qubit Hilbert-space $\mathcal{H}_1 = \mathbb{C}^2$:

$$\begin{array}{ccccccc} 0 & \rightarrow & |0\rangle & = & |\uparrow\rangle & = & \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 1 & \rightarrow & |1\rangle & = & |\downarrow\rangle & = & \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{array}$$

A general state is the complex, linear combination of these states:

$$|\psi\rangle = a_0|0\rangle + a_1|1\rangle = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \qquad \text{with } |a_0|^2 + |a_1|^2 = 1, \quad a_0, a_1 \in \mathbb{C} \tag{1}$$

Vectors differing only by a global phase represent the same state.

The elements $a_i$ are called probability amplitudes. During a measurement the state collapses into one of the basis states, and $|a_i|^2$ is the probability of the system to be found in the state $|i\rangle$.

The two restrictions on the vectors representing a physical state can be matched by the following choice of coefficients:

$$\begin{array}{rcl} a_0 & = & \cos(\theta/2) \\ a_1 & = & e^{\imath\phi}\sin(\theta/2) \end{array}$$

This provides the interpretation of the 1-qubit state space as a one-sphere, the so-called Bloch-sphere (see figure 1).

A quantum computer needs several qubits. Due to the postulates of quantum mechanics the Hilbert-space of $N$-qubits is the tensor-product of $N$ single-qubit Hilbert-spaces.

$$\mathcal{H}_N = \underbrace{\mathcal{H}_1 \otimes \cdots \otimes \mathcal{H}_1}_{N} \cong \mathbb{C}^{2^N}$$

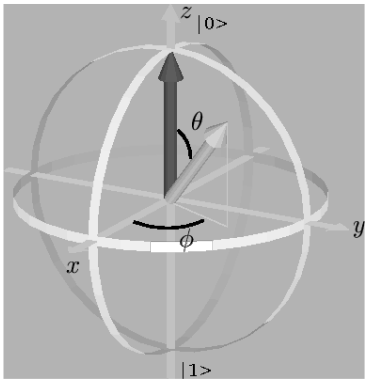



Figure 1: The Bloch-sphere visualizes the 1-qubit Hilbert-space $\mathcal{H}_1$. The poles of the sphere at $\theta = 0$ and $\theta = \pi$ represent the basis states $|0\rangle$ and $|1\rangle$, respectively. All other positions on the sphere correspond to different superpositions of these states.

The $2^N$ basis vectors of $\mathcal{H}_N$ are tensor-products of single-qubit basis vectors of the individual subspaces:

$$\begin{aligned} |0\ldots00\rangle &= |0\rangle \otimes \ldots \otimes |0\rangle \otimes |0\rangle \\ |0\ldots01\rangle &= |0\rangle \otimes \ldots \otimes |0\rangle \otimes |1\rangle \\ |0\ldots10\rangle &= |0\rangle \otimes \ldots \otimes |0\rangle \otimes |0\rangle \\ |0\ldots11\rangle &= |0\rangle \otimes \ldots \otimes |1\rangle \otimes |1\rangle \\ &\vdots \\ |1\ldots11\rangle &= |1\rangle \otimes \ldots \otimes |1\rangle \otimes |1\rangle \end{aligned}$$

Each of these vectors represents a state, in which the values of all qubits are known exactly. Again, a general state is a superposition of these basis states:

$$|\psi\rangle = a_{0\ldots00}|0\ldots00\rangle + a_{0\ldots01}|0\ldots01\rangle + \ldots + a_{1\ldots11}|1\ldots11\rangle = \begin{pmatrix} a_{0\ldots00} \\ a_{0\ldots01} \\ \vdots \\ a_{1\ldots11} \end{pmatrix} \quad [1] \qquad (2)$$

$$\text{with } |a_{0\ldots00}|^2 + |a_{0\ldots01}|^2 + \ldots + |a_{1\ldots11}|^2 = 1, \quad a_{0\ldots00}, a_{0\ldots01}, \ldots, a_{1\ldots11} \in \mathbb{C}$$

Accordingly $|a_{i\ldots jk}|^2$ gives the probability that the whole system is found in the state $|i\ldots jk\rangle$, when measured. The indices refer to the values of the single qubits in that state: The rightmost index gives the value (the state) of qubit $0$, while the leftmost index that of qubit $N-1$.

*Quantum Gates*

As we have seen, the algorithm is executed as the temporal development of the physical system. This time-evolution is described by the Hamiltonian $\mathcal{H}(t)$ via the Schrödinger equation:

$$\begin{aligned} \imath\hbar\frac{\partial}{\partial t}|\psi(t)\rangle &= \mathcal{H}(t)|\psi(t)\rangle \\ \Rightarrow |\psi(t)\rangle &= U(t)|\psi(0)\rangle \quad [2] \end{aligned}$$

[1] Note, that this state does *not* necessarily take a product form such as $|\psi\rangle = (b_0|0\rangle + b_1|1\rangle) \otimes (c_0|0\rangle + c_1|1\rangle)$, where each single qubit is in a pure state, such that the identification with the Bloch-sphere works.

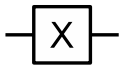
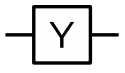
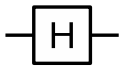
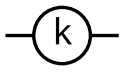
| Gate | Symbol | Matrix representation |
|---|---|---|
| X-Rotation $\frac{\pi}{2}$ |  | $X = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & \imath \\ \imath & 1 \end{pmatrix}$ |
| Y-Rotation $\frac{\pi}{2}$ |  | $Y = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ |
| Hadamard |  | $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ |
| Phase shift |  | $R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{\imath\phi} \end{pmatrix}$ where $\phi = \frac{2\pi}{2^k}$ |

Table 1: Some examples for 1-qubit gates. The matrix notation refers to the 1-qubit state defined in equation (1).

However, describing the whole algorithm by a single unitary time-evolution operator $U(t)$ is extremely complicated. Therefore the algorithm is split up into many simple transformations, so called quantum gates, acting on only one or two qubits.

In the following, we will restrict ourselves to an *ideal* quantum computer, which means, we do not bother about the specific form of the Hamiltonian or the time-evolution of the state vector. Instead, we will consider only special gates acting as instantaneous transformations onto the state vector.

#### 1-*qubit Gates*

The simplest kind of gates act on one qubit only. Table 1 shows some examples in matrix notation on $\mathcal{H}_1$. Using the Bloch-sphere identification of the state vector, all 1-qubit gates can be thought of as rotations on that sphere.

On $\mathcal{H}_N$ a gate $U = \begin{pmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{pmatrix}$ acting on qubit $k$ is obtained by the tensor-product with the identity operator $\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ on all other qubits:

$$\begin{array}{rccccccccc} U_k = & \mathbb{1} & \otimes \ldots \otimes & \mathbb{1} & \otimes & U & \otimes & \mathbb{1} & \otimes \ldots \otimes & \mathbb{1} \\ \text{qubit}: & N-1 & & k+1 & & k & & k-1 & & 0 \end{array}$$

Instead of calculating the whole $2^N \times 2^N$-dimensional matrix we realize that the action of this matrix on the state vector can be formulated in the following form:

$$\begin{array}{c} |\psi'\rangle = U_k|\psi\rangle \\ \begin{pmatrix} a'_{*\ldots*0*\ldots*} \\ a'_{*\ldots*1*\ldots*} \end{pmatrix} = \begin{pmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{pmatrix} \begin{pmatrix} a_{*\ldots*0*\ldots*} \\ a_{*\ldots*1*\ldots*} \end{pmatrix} = \begin{pmatrix} U_{11} \cdot a_{*\ldots*0*\ldots*} + U_{12} \cdot a_{*\ldots*1*\ldots*} \\ U_{21} \cdot a_{*\ldots*0*\ldots*} + U_{22} \cdot a_{*\ldots*1*\ldots*} \end{pmatrix} \end{array} \quad (3)$$

Here the asterisks denote the same indices on both sides of the equation and of course the differing indices are at the $k^{\text{th}}$ position. This means that the $2^N$ elements of the state vector are grouped into pairs on which the simple $2 \times 2$-dimensional gate acts independently.

[2] $U(t) = \exp(-\frac{\imath}{\hbar}\mathcal{H}t)$ if $\mathcal{H}$ is time-independent

*Controlled Gates*

1-qubit gates alone are not enough to perform arbitrary transformations on the state vector. For example, they cannot produce entanglement between different qubits. Therefore we introduce controlled gates, a special class of 2-qubit gates. These gates work similar to the 1-qubit gates on a target qubit, but the gate is only performed on those pairs of amplitudes, in which the control bit index is 1. Table 2 shows two examples in matrix notation.

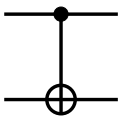
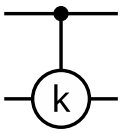
| Gate | Symbol | Matrix representation |
|---|---|---|
| controlled NOT |  | $CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ |
| controlled phase shift |   | $CR_k = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\imath\phi} \end{pmatrix}$ where $\phi = \frac{2\pi}{2^k}$ |

Table 2: Some examples for controlled gates. The matrix notation refers to the 2-qubit state defined in equation (2), with qubit 0 being the control bit and qubit 1 being the target bit.

It has been shown that any unitary transformation on $\mathcal{H}_N$ can be decomposed into 1- and 2-qubit gates. For example the Hadamard-, phase-shift- and controlled NOT-gates form a universal set.

**Simulation**

As we have seen in the previous section, the requirements for a simulation of an ideal quantum computer are the following:

- The state vector consists of $2^N$ complex numbers, which have to be stored in the memory.
- In order to perform a 1-qubit gate we need to act with a $2 \times 2$ matrix onto pairs of the state vector. All amplitudes are worked on.
- For a controlled gate we need to work on half the elements of the state vector.

Thus we are facing a problem that grows exponentially in memory and CPU-time with the number of qubits!

So the question arises: Why do we want to simulate a quantum computer on a classical machine, if the simulation is bound to be inefficient? The reasons are manifold.

- As long as no scalable quantum computer has been developed, we need a simulation to test the algorithms proposed so far and to develop new algorithms.
- We need to test the robustness of the algorithms against different error models (Arnold [4]).
- We want to test and develop error correction codes.
- We want to emulate a quantum computer to simulate some general quantum mechanical system.

*State Distribution*

Since the CPU-time requirement is a soft limit in contrast to the memory requirement of our simulation, the main idea for the parallelization is to distribute the state vector over many processes. This is shown in figure 2.

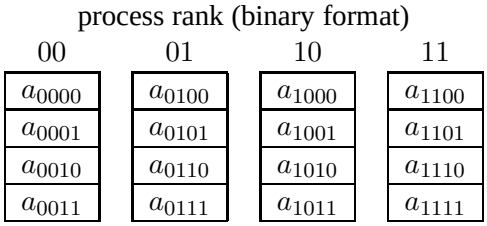



Figure 2: State distribution for an example with $4$ qubits leading to $16$ states distributed over $4$ processes. Note that the two right indices give the local address of the corresponding amplitude in the memory of the process in binary format, while the two left indices represent the rank of the corresponding process in binary format. This observation makes the book keeping very simple. Further note, that we need to double the number of processes if we increase the number of qubits by one, keeping the memory size per process constant.

*Gate Scheme*

As we have seen in equation (3), the action of a 1-qubit gate onto the state vector splits into independent $2 \times 2$-matrices acting on different pairs of amplitudes. This is optimal for parallelization.

Figure 3 shows the scheme, which amplitudes form pairs, depending on the qubit, on which the gate is supposed to act. If both amplitudes of the pairs are located on the same process, we call the corresponding qubit local, otherwise the qubit is said to be nonlocal.

If a gate should act on a nonlocal qubit, communication between the processes involved is needed. Different communication schemes are discussed in the next section.

The 1-qubit gate scheme can easily be generalized to the case of controlled gates: The partner amplitudes are determined by the target qubit of the gate, but the matrix is only applied to those pairs, where the control bit index is 1. Note that the control bit does not need to be local. However, if the control bit is local, we have work balancing, while half the processes do nothing, if the control bit is nonlocal. This may lead to long waiting times before the next communication, when the processes are necessarily synchronized.

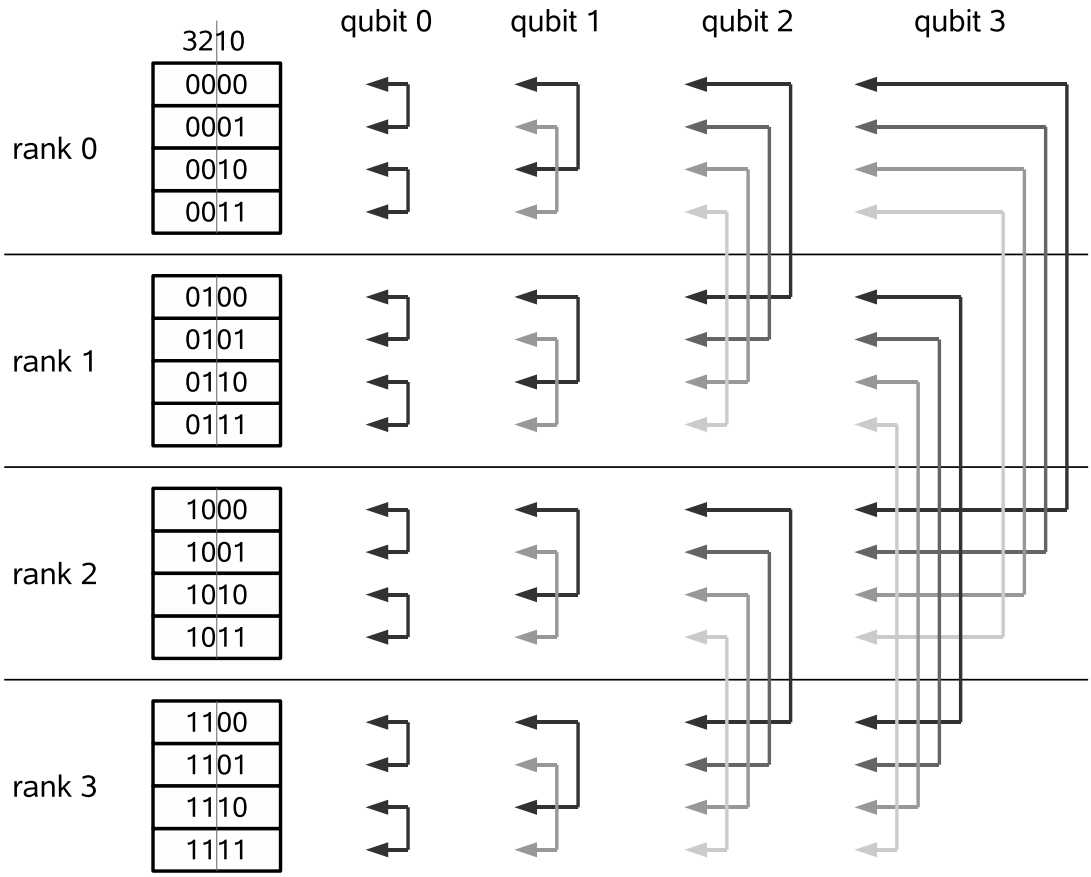
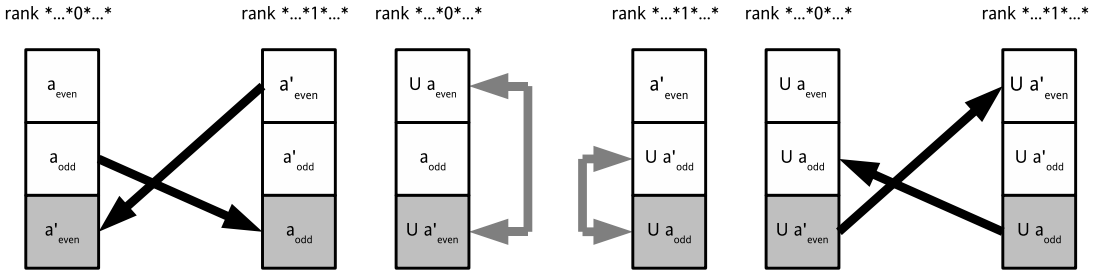



Figure 3: 1-qubit gate scheme for the pairs of amplitudes, depending on the qubit, that the gate acts on. Again $4$ qubits are distributed over $4$ processes. The pairs for a gate acting on qubit $k$ are found as follows: According to equation (3) all qubit indices of the two amplitudes of a pair must be equal except the indices corresponding to qubit $k$. We see that in this case qubit $0$ and qubit $1$ are local, that is, the corresponding partner amplitudes are on the same processes. Qubit $2$ and qubit $3$ are nonlocal and in order to act on those qubits, communication becomes necessary.

*Communication Schemes*

The first communication scheme exploits the even-odd splitting of the amplitudes for most of the gates. It is illustrated and explained in figure 4. It provides optimal cache access and ideal work balancing for nonlocal gates. The major drawback is that it requires *two* massive communications for each gate acting on a nonlocal qubit. In the following we want to develop a communication scheme that reduces the amount of communicated data and in this way provides an even better scaling behavior than this already highly parallelized scheme.




(a) Communication (b) Gate (c) Back communication

Figure 4: Communication using even-odd splitting. The scheme is based on the observation that for all gates, except those acting on qubit $0$, the amplitude pairs consist either of two even or of two odd partners (see figure 3). On each process there is a buffer half the size of the state vector (marked gray). If a gate acts on a nonlocal qubit, each process determines the rank of his partner process, where the missing amplitudes are located. One process sends his even part of the state vector to his partner and receives the odd part from his partners state vector (figure 4(a)). Now both processes have the partner amplitudes they need: One calculates the even part and one the odd part (figure4(b)). When the computation is done, the calculated data in the buffer is sent back (figure 4(c)).

The second communication scheme reduces the number of necessary communications to at most one per nonlocal qubit. This is achieved by exchanging half the amplitudes between corresponding pairs of processes instead of writing them into a special buffer. Now the action of the gate on the previously nonlocal qubit can be performed as if the qubit has taken the position of a local qubit. The information, on which processes the different amplitudes can be found is stored in a permutation matrix $\sigma$, so the back communication can be skipped.

$\sigma$ does not need to be implemented as a matrix. In our case it consists of two $1$-dimensional arrays of length $N$. The first one stores the number of the qubit at a given position, the second one is the inverse and contains the position at which a given qubit is located. Here, the position of a qubit refers to a configuration of amplitudes, such that a gate acting with a pair scheme corresponding to that position actually acts on that qubit. A detailed example is given in figure 5.

Unfortunately, this reduction in communication has a price: We are exchanging the amplitudes instead of writing into a (previously empty) buffer. This requires an extensive intermediate buffering, increasing the nett communication time. Additionally, the cash access may be suboptimal, but this depends on the position of the local qubit, whose place the nonlocal qubit shall take.

However, the real benefits of the permutation method appear if we realize that in this method the communication is not strictly bound to the single gate, for which it makes the qubit local. It will be much more efficient if we perform the communication at carefully chosen times during some complex quantum circuit (quantum algorithm), such that a communication block makes some qubits local on which more than one gates act afterwards.

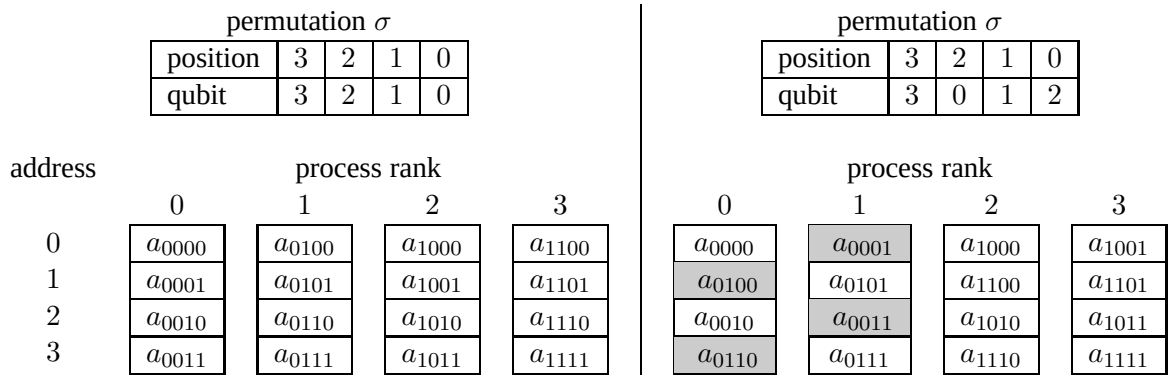



Figure 5: This example illustrates which amplitudes need to be exchanged in order to make the nonlocal qubit 2 local and how the permutation matrix $\sigma$ records the actual permutation. On the left hand side we have the original configuration of the amplitudes. As we have seen before, qubit 0 and 1 are local. Recall that in the pair scheme for a gate acting on qubit 0 the pairs are simply the neighboring amplitudes, that is the addresses 0 and 1 and the addresses 2 and 3. Now we wish to make qubit 2 local and put it into the position of qubit 0. To achieve this, the processes 0 and 1 and the processes 2 and 3 exchange every second amplitude with each other. The corresponding amplitudes on rank 0 and 1 are emphasized with a gray background. The qubit scheme that previously acted on qubit 0 now acts on qubit 2, which is why we say that qubit 2 is now at position 0.

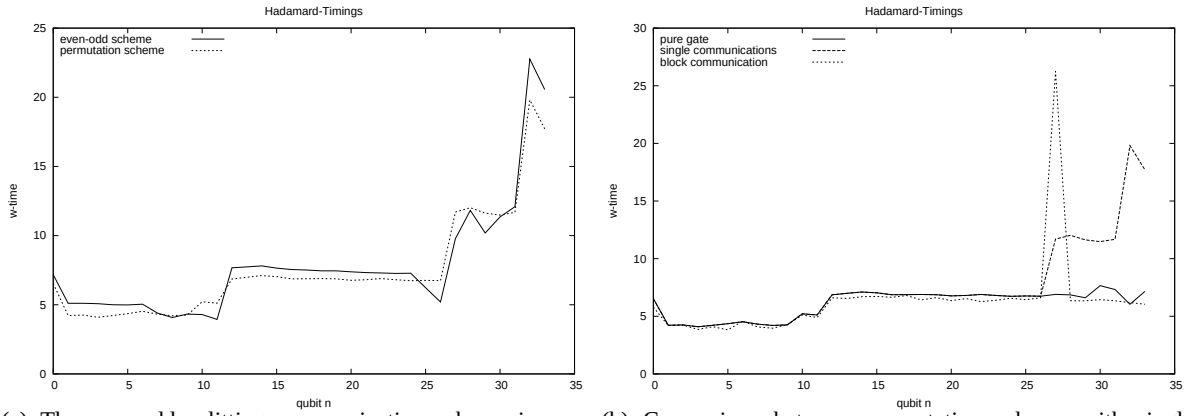
When we wish to make a group of qubits local at once, the communication can be further optimized. This can be seen from the following consideration: Assume we want to make $n$ nonlocal qubits local at a time and on each process there are $N_{\text{loc}}$ amplitudes. We may use the same algorithm as before to make one qubit after the other local. During each call each single process sends half of its state vector. Since the time needed to make all $n$ qubits local is approximately proportional to the amount of data send, the whole operation scales like $T_{\text{single}} \propto (n/2)N_{\text{loc}}$. It is easy to see that during this operation some amplitudes are received by a process during one step and in the next step the same amplitudes are send to a third process. These amplitudes could have been send directly to their destination.

Therefore we implemented a block communication scheme, which first determines the subgroup of $2^n$ processes, which need to communicate with each other and then each single process sends in $2^n - 1$ steps only that part of the state vector to the other processes in the subgroup, that theses processes need. Here, the total amount of data send by a single process and thus the time to execute the operation, scales like $T_{\text{block}} \propto (1 - 1/2^n)N_{\text{loc}}$, since in each single communication, the partner processes exchange $N_{\text{loc}}/2^n$ amplitudes.

*Benchmark Results*

Since the permutation scheme develops its best performance, when cleverly implemented into a complex quantum circuit, there is no straight forward way to compare its performance to the scheme using even-odd splitting of the state vector. Therefore we compare it in different test cases: First, as a worst case scenario for the permutation scheme, the wall-clock time to perform a Hadamard gate on a single qubit; first with the most naive communication method to make each single qubit local individually, and second, with a single block communication. The results are shown and discussed in figure 6.

As another aspect we compare the wall-clock time needed for communication in the permutation scheme to make a group of qubits local in the two ways: once individually and once using block communication. This is shown in figure 7.




(a) The even-odd splitting communication scheme is compared with the permutation scheme, where each nonlocal qubit is made local directly before the gate execution and the gate and communication times are shown together. On the qubits 27-31, where we have only intra-node communication, both schemes give similar results. That means that the gain in reducing the number of communications is absorbed by the loss due to intermediate buffering. But on qubits 32 and 33, where we have slower inter-node communication, the permutation scheme is slightly better.

(b) Comparison between permutation scheme with single qubit communication and block communication. Additionally the time to execute the pure gate without the previous communication is shown. The total communication time in the single qubit case is 47.48s, while the block communication takes only 20.49s.

Figure 6: These plots show the wall clock time to execute a simple Hadamard gate on qubit n of a quantum computer simulation with N=34 qubits on 128 processes. The qubits 0-26 are local, here the different timings are due to different cash access.

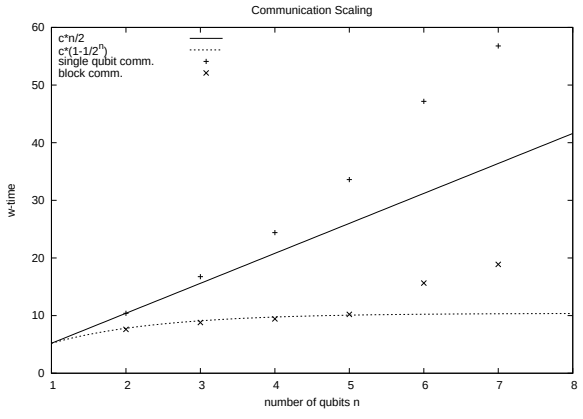



Figure 7: The communication time to make $n = 2$-7 qubits local is shown using the single qubit algorithm $n$ times and the block communication algorithm. The results are compared to the theoretical curves, which are normalized to fit the first data points at $n = 2$ ($c = 10.4s$). While the single qubit algorithm is slightly worse than the theoretical prediction, the block communication fits quite good up to $n = 5$. The increasing time for $n = 6, 7$ is due to the slower inter-node communication.

Finally, we test the scaling behavior of the quantum Fourier transform as an example for a more complex circuit. The circuit is presented in figure 8. Again we compare the scheme based on even-odd splitting with the permutation scheme, which here can develop more of its benefits. It is enough to perform a single block communication during the whole algorithm: Let $m$ denote the number of nonlocal qubits. Since the control bits of the controlled phase gates do not need to be local, the first communication becomes necessary before the Hadamard gate on qubit $N-m$. However, we may choose an earlier point to make all nonlocal qubits local at once. The earliest point would be before the Hadamard gate on qubit $m$, if we make the qubits $0$-$(m-1)$ nonlocal in this turn. Then all qubits needed for the remaining part of the circuit are local and no further communication is necessary.

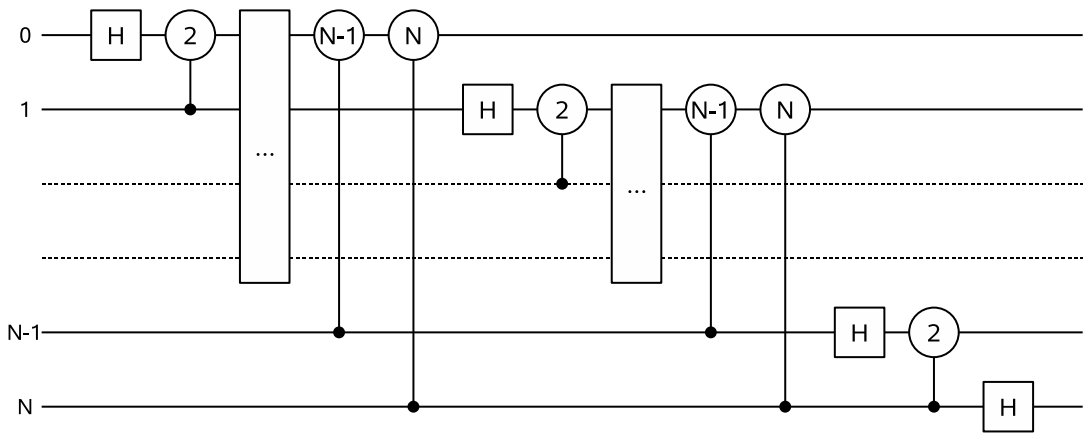



Figure 8: The quantum Fourier transformation is realized by acting on each qubit first with a Hadamard gate followed by a sequence of controlled phase-shifts where the control bits are all following qubits. The QFT can be seen as the quantum kernel of Shor's factoring algorithm and is therefore a reasonable candidate for benchmarking the simulation.

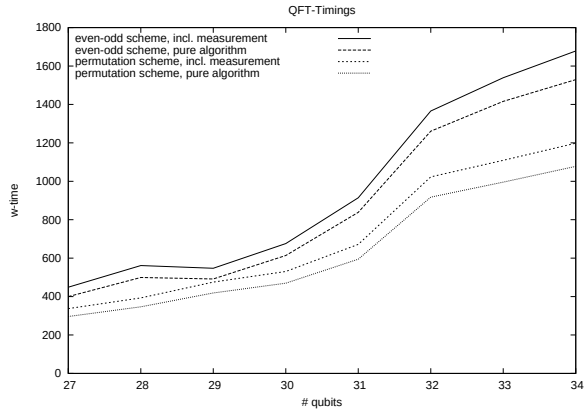



Figure 9: This plot compares the timings of the two different communication schemes to perform a quantum Fourier transformation on 27-34 qubits using $1, 2, 4, \ldots, 128$ processes. Both results are given once as the time only to execute the algorithm and once including the final measurement, which calculates the expectation values of the individual qubits.

## Conclusion and Outlook

We see that the permutation scheme yields further speedup with respect to the previously implemented scheme based on even-odd splitting.

However due to the freedom to choose the point for the communication and the differently efficient cash access for the communication as well as the gates depending on the qubit position, the task of finding the optimal points for the communication is highly non-trivial. This problem may be the subject of further investigation.

## Acknowledgments

I would like to thank my supervisors Markus Richter and Guido Arnold for giving me this very interesting and instructively topic. Also I wish to thank Binh Trieu for helping me with my problems and questions. Especially, I want to thank my fellow guest students for the nice time we had here together.

## References

1. R. P. Feynman, Simulating physics with computers, *Int. J. Theor. Phys.* **21**:467-488, (1982)
2. Necia Grant Cooper, Los Alamos Science 27 (2002), http://library.lanl.gov/cgi-bin/getfile?number27.htm
3. K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, Th. Lippert, H. Watanabe and N. Ito, Massive Parallel Quantum Computer Simulation, *Computer Physics Communication* article in press
4. G. Arnold, M. Richter, B. Trieu, Improving quantum computer simulations, to be published

# Pedestrian Dynamics with Event-Driven Simulation

Mohcine Chraibi

Technical University Hamburg-Harburg

mohcine.chraibi@tu-harburg.de

**Abstract:** For the modelling of pedestrian dynamics we treat persons as self-driven objects moving in a continuous space. On the basis of a modified social force model we qualitatively analyze the influence of various approaches for the driving force of the pedestrians on the resulting velocity-density relation. With an event-driven algorithm we will obtain the following result: if the model increases the required space of a person with increasing and self-adapting current velocity, the reproduction of the typical form of the fundamental diagram is possible.

## Introduction

Microscopic models are state of the art for computer simulation of pedestrian dynamics. The modelling of the individual movement of pedestrians results in a description of macroscopic pedestrian flow and allows e.g. the evaluation of escape routes, the design of pedestrian facilities and the study of more theoretical questions. For a first overview see [1, 2]. The corresponding models can be classified in two categories: the cellular automata models [3, 4, 5, 6, 7] and models in a continuous space [8, 9, 10, 11]. We focus on models continuous in space. They differ substantially with respect to the 'interaction' between the pedestrians and thus to the update algorithms as well. The social force model for example assumes, among other things, a repulsive force with remote action between the pedestrians [8, 12, 13, 14, 15, 16, 17]. Other models treat pedestrians by implementing a minimum inter-person distance, which can be interpreted as the radius of a hard body [10, 11].

One primary test, whether the model is appropriate for a quantitative description of pedestrian flow, is the comparison with the empirical velocity-density relation [18, 19, 21, 22]. As shown in [26] the empirical velocity-density relation for the single-file movement is similar to the relation for the movement in a plane in shape and magnitude. This surprising conformance indicates, that lateral interferences do not influence the fundamental diagram at least up to a density-value of $4.5\,m^{-2}$. This result suggests that it is sufficient to investigate the pedestrian flow of a one-dimensional system without loosing the essential macroscopic characteristics. We systematically modify the social force model to achieve a satisfying agreement with the empirical velocity-density relation (fundamental diagram).

## Modification of the Social Force Model

*Motivation*

The social force model was introduced by [8]. It models the one-dimensional movement of a pedestrian $i$ at position $x_i(t)$ with velocity $v_i(t)$ and mass $m_i$ by the equation of motion

$$\frac{dx_i}{dt} = v_i \qquad m_i \frac{dv_i}{dt} = F_i = \sum_{j \neq i} F_{ij}(x_j, x_i, v_i). \tag{1}$$

The summation over $j$ accounts for the interaction with other pedestrians. We assume that friction at the boundaries and random fluctuations can be neglected and thus the forces are reducible to a driving and a repulsive term $F_i = F_i^{drv} + F_i^{rep}$. According to the social force model [8] we choose the driving term

$$F_i^{drv} = m_i \frac{v_i^0 - v_i}{\tau}, \tag{2}$$

where $v_i^0$ is the intended speed and $\tau$ controls the acceleration. In the original model the introduction of the repulsive force $F_i^{rep}$ between the pedestrians is motivated by the observation that pedestrians stay away from each other by psychological reasons, e.g. to secure the private sphere of each pedestrian [8]. The complete model reproduces many self-organization phenomena like e.g. the formation of lanes in bidirectional streams and the oscillations at bottlenecks [8, 12, 13, 14, 15, 16, 17]. We choose the repulsive force as

$$F_i^{rep}(t) = \begin{cases} 0 & : \quad x_{i+1}(t) - x_i(t) > d_i(t) \\ -\delta(t) v_i(t) & : \quad x_{i+1}(t) - x_i(t) \leq d_i(t) \end{cases} \tag{3}$$

The hard core, $d_i$, reflects the size of the pedestrian $i$. Without other constraints a repulsive force which is symmetric in space can lead to velocities which are in opposite direction to the intended speed. Furthermore, it is possible that the velocity of a pedestrian can exceed the intended speed through the impact of the forces of other pedestrians. In a two-dimensional system this effect can be avoided through the introduction of additional forces like a lateral friction, together with an appropriate choice of the interaction parameters. In a one-dimensional system, where lateral interferences are excluded, a loophole is the direct limitation of the velocities to a certain interval [8, 12].

Another important aspect is the dependency between the current velocity and the space requirement. As suggested by Pauls in the extended ellipse model [28] the area taken up by a pedestrian increases with increasing speed. Thompson also based his model on the assumption, that the velocity is a function of the inter-person distance [10]. Furthermore Schreckenberg and Schadschneider observed in [18, 19], that in cellular automaton model the consideration, that a pedestrian occupies all cells passed in one time-step, has a large impact on the fundamental diagram. Helbing and Molnár note in [8] that the range of the repulsive interaction is related to step-length. Following the above suggestion we specify the relation between required space and velocity for a one-dimensional system. In a one-dimensional system the required space reduces to a required length $d$. In [26] it was shown that for the single-file movement the relation between the required length $d$ for one pedestrian to move with velocity $v$ and $v$ itself is linear at least for velocities $0.1\,m/s < v < 1.0\,m/s$.

$$d = a + b\,v \quad \text{with} \quad a = 0.36\,m \quad \text{and} \quad b = 1.06\,s \tag{4}$$

Hence it is possible to determine one fundamental microscopic parameter, $d$, of the interaction on the basis of empirical results. This allows focusing on the question if the interaction and the equation of motion result in a correct description of the individual movement of pedestrians. Summing up, for the modelling of regular motions of pedestrians we modify the reduced one-dimensional social force model in order to meet the following properties: the movement of a pedestrian is only influenced by the one directly in front; the required length $d$ of a pedestrian that moves with velocity $v$ is $d = a + b\,v$.

For simplicity we set $v_i^0 > 0$, $x_{i+1} > x_i$ and the mass of a pedestrian to $m_i = 1$.

*Event-driven algorithm*

The time stepping algorithm in [27] was very time consuming and not completely correct, then it was not able to determine exactly the collision times. The event-driven algorithm looks for the first collision time and advances each person up to the calculated time. We require a distance function $dist_i(t)$ which depends implicitly on the initial positions, velocities etc.

$$dist_i(t) = (position_{i+1}(t) - radius_{i+1}(t)) - (position_i(t) + radius_i(t)) \tag{5}$$

The function should be zero when the pedestrians $i$ and $i+1$ are in contact, and should change sign as the pedestrian $i$ moves through the contact. We are searching for positive real roots of the equation

$$dist_i(t) = 0 \tag{6}$$

We wish to solve such equation efficiently and reliably: missed collisions could cause problems. For this purpose we use the "pegasus" method for computing the roots of the equation (6). The pedestrians are advanced on a regular step-by-step basis. At the end of each step, the system is examined for the pedestrians, who potentially can collide. For each such pair of pedestrians, the equation (6) is solved retrospectively to find the time at which the collision should have occurred. The flow diagram in Figure 4 shows the processing of the program. The only time consuming part of the algorithm could be the search for the overlapping pedestrians. However, the algorithm become more accurate than the time stepping algorithm in [27], then we can exactly determine the events.

- Update without velocity adaptation

The movement of a pedestrian is only influenced by the driving term. If the required length at a given current velocity is larger than the distance to the pedestrian in front, the pedestrian stops (i. e. the velocity becomes zero). This ensures that the velocity of a pedestrian is restricted to the interval $v_i = [0, v_i^0]$ and that the movement is only influenced by the pedestrian in front. The definition of $d_i$ is such that the required length increases with growing velocity.

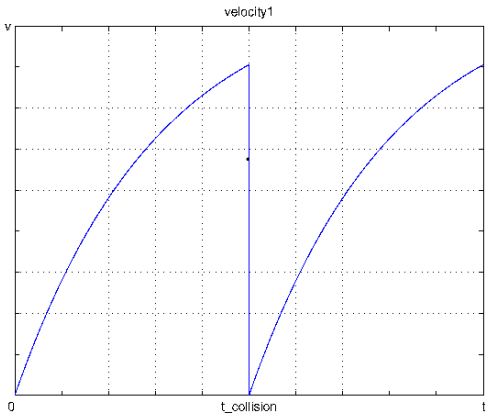



Figure 1: The pedestrian accelerates to $t_{collision}$ where he/she stops.

See the velocity function in Figure 1.

- Update with velocity adaptation

Normally, the pedestrians don't walk ahead until they collide with each other and stop. This observation leads us to adapt our algorithm so that the pedestrians can avoid, as much as they can, collisions.This velocity adaptation was in [20] with the cellular automaton model implemented. The pedestrians shall adjust their actual velocity, in such a way, that they try to keep a minimum distance to the pedestrian in front. Again the force is only influenced by actions in front of the pedestrian. By means of the required length, $d_i$, the range of the interaction is a function of the velocity $v_i$. The movement of the pedestrians is smoother and more conform to the human behaviour.

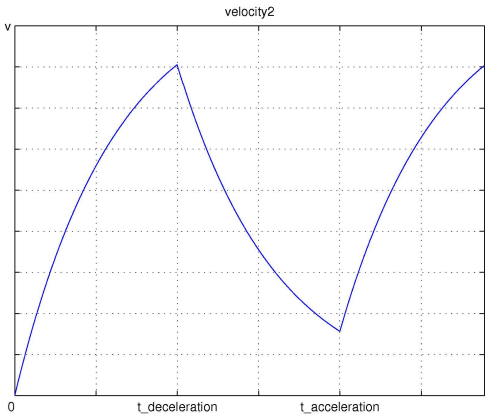



Figure 2: The pedestrian accelerate to $t_{deceleration}$ where he/she starts decelerating.At $t_{acceleration}$ he/she accelerates again.

We will introduce a new distance, called safety distance $\epsilon_i(t)$, which also linearly depends on the velocity:

$$\epsilon_i(t) = \epsilon_a + \epsilon_b \, v_i(t) \quad \text{with} \quad \epsilon_a = 0.125\, m \quad \text{and} \quad \epsilon_b = 0.758\, s \tag{7}$$

And which is defined as the difference between the required length $d_i(t)$ and the step-length $st_i(t)$.

$$st_i(t) = st_a + st_b \, v_i(t) \quad \text{with} \quad st_a = 0.235\, m \quad \text{and} \quad st_b = 0.302\, s \quad \text{according to Ref. [23]} \tag{8}$$

See Figure 3.

We adjust the definitions of $F_i^{drv}$ and $F_i^{rep}$ in (2) and (3), so that the update-velocity changes to:

$$v_i(t) = \begin{cases} v_i^0(t)\,(1 - exp(\frac{-(t-t_{collision})}{\tau})) & : \quad t_{collision} < t < t_{deceleration} \\ v_{dec}\,(exp(\frac{-(t-t_{deceleration})}{\tau})) & : \quad t_{deceleration} < t < t_{acceleration} \\ v_i^0(t)\,(1 - exp(\frac{-(t-t_0)}{\tau})) & : \quad t_{acceleration} < t \end{cases} \tag{9}$$

The constants $v_{dec}$ and $t_0$ are defined, so that the velocity function is continuous. See Figure 2. We solve the equation (6) three times, so that we calculate the following times

- Collision times: Times at which the distance $dist_i(t)$ between the pedestrian $i$ and $i+1$ is null.
- Deceleration times: Times at which the distance $dist_i(t)$ between the pedestrian $i$ and $i+1$ is $\epsilon_i(t)\, f$

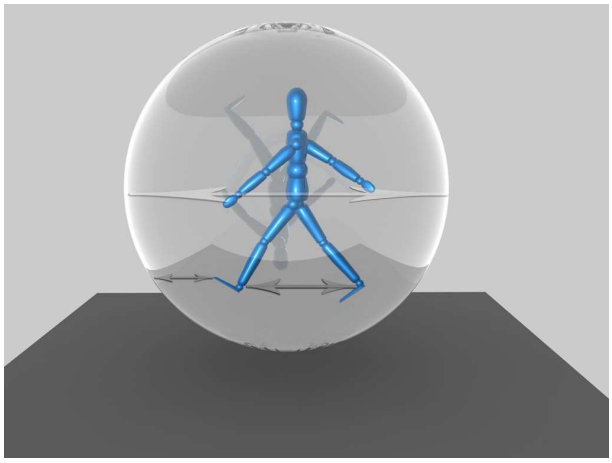


Figure 3:

- Acceleration times: Times at which the distance $dist_i(t)$ between the pedestrian $i$ and $i+1$ is $\epsilon_i(t)$

f is a free parameter with

$$0 < f < 1 \tag{10}$$

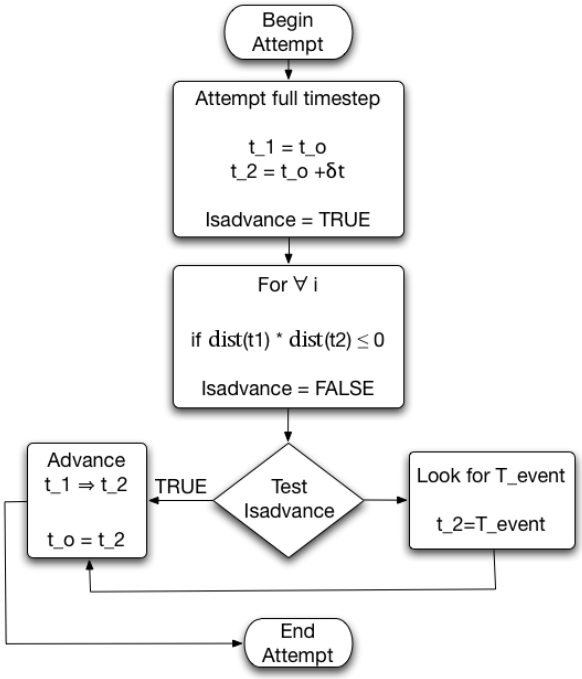



Figure 4:

**Results**

To enable a comparison with the empirical fundamental diagram of the single-file movement [26] we choose a system with periodic boundary conditions and a length of $L = 17.3\,m$. The values for the

intended speed $v_i^0$ are distributed according to a normal-distribution with a mean value of $\mu = 1.24\,m/s$ and $\sigma = 0.05\,m/s$.

In reality the parameters $\tau, a, b, \epsilon_a, \epsilon_b, st_a, st_b$ and $f$ are different for every pedestrian $i$ and correlated with the individual intended speed. But we know from experiment [26] that the movement of pedestrians is influenced by phenomena like marching in step and in reality the action of a pedestrian depends on the entire situation in front and not only on the distance to the next person. Therefore it's no point to attempt to give fully accurate values of those parameters and we may choose identical values for all pedestrians. According to [17], $\tau = 0.61\,s$ is a reliable value.

For every run at $t = 0$ all velocities are set to zero and the pedestrians are randomly distributed with a minimal distance of $a$ in the system. After $3 \times 10^5$ relaxation-steps we perform $3 \times 10^5$ measurements-steps. At every step we determine the mean value of the velocity over all particles and calculate the mean value over time. The following figures present the dependency between mean velocity and density for different approaches to the velocity function introduced in section "Event-driven algorithm". To demonstrate the influence of the velocity dependence of the required length we choose different values for the parameter $b$. With $b = 0$ one get simple hard bodies.

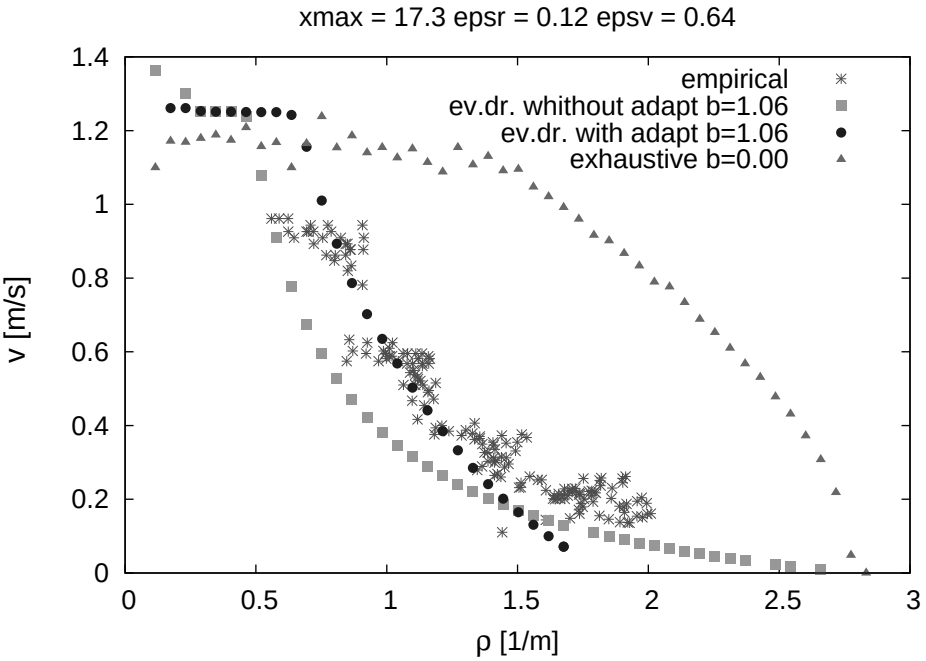



Figure 5: Velocity-density relation for hard bodies with $a = 0.36\,m$ and without a remote action in comparison with empirical data from [26]. The filled triangles result from simple hard bodies. The introduction of a required length with $\epsilon_a = 0.125,$ and $\epsilon_b = 0.758$ leads to a good agreement with the empirical data. We see the difference between the event-driven algorithm with velocity adaptation and without velocity adaptation.

Figure 5 shows the relation between the mean values of walking speed and density for hard bodies with $a = 0.36\,m$ and without remote action, according to the interaction introduced in Equation (2), (3) and (9). If the required length is independent of the velocity, one gets a negative curvature of the function $v = v(\rho)$. With $b = 1.06\,s$ we found a difference between the velocity-density relation predicted by the model and the empirical fundamental diagram. The reason for this discrepancy is that the interaction do not describe the individual movement of pedestrian correctly. But the event-driven algorithm with velocity adaptation produces better results. However they are still inaccurate. The explanation of this

discrepancy is that the arbitrarily chosen value of $f$ at (10), was not correct.

## Discussion and summary

For the investigation of the influence of the required space and the adaptation of the velocity of the pedestrians on the fundamental diagram we have introduced a modified one-dimensional social force model. The modifications warrant that in the direction of intended speed negative velocities do not occur and that the motion of the pedestrians is influenced by objects and actions directly in front only. We assume that pedestrians have a step-length and a security distance. That is the minimum distance that one pedestrian tries to keep from the pedestrian in front of. Thus we showed that the modified model is able to reproduce the typical shape of the empirical fundamental diagram of pedestrian movement for a one-dimensional system, if it considers the velocity-dependence of the required length. From the same empirical fundamental diagram one determines $b = 1.06\,s$, see [26]. We do not have to determine the parameter $\epsilon_i(t)$, then the length requirement $d_i(t)$ and the step-length $st_i(t)$ are empirically known, see [27] and [23]. The velocity adaptation and the empirical known parameters lead to a good agreement with the fundamental diagram. However we still have to find a better value for the only free parameter $f$. The above considerations refer to the simplest system in equilibrium and with periodic boundary conditions. In a real life scenario like a building evacuation, where one is interested in estimates of the time needed for the clearance of a building and the development of the densities in front of bottlenecks, one is confronted with open boundaries and conditions far from equilibrium. We assume that a consistency on a microscopic level needs to be achieved before one can accurately describe real life scenarios. The investigation provides a basis for a careful extension of the modified social force model and an upgrade to two dimensions including further interactions.

## Acknowledgements

We thank Bernhard Steffen for discussion and Thomas Düssel for his multimedia help.

## References

1. M. Schreckenberg and S.D. Sharma (Ed.), *Pedestrian and Evacuation Dynamics* (Springer 2001)
2. E. R. Galea (Ed.), *Pedestrian and Evacuation Dynamics* (CMS Press 2003)
3. M. Muramatsu, T. Irie and T. Nagatani, Physica **A267**, 487 (1999)
4. V. J. Blue and J. L. Adler, Journal of the Transportation Research Board **1678**, 135 (2000)
5. K. Takimoto and T. Nagatani, Physica **A 320**, 611 (2003)
6. C. Burstedde, K. Klauck, A. Schadschneider and J. Zittartz, Physica **A 295**, 507 (2001)
7. A. Keßel, H. Klüpfel, J. Wahle and M. Schreckenberg, in [1], p. 193
8. D. Helbing and P. Molnár, Phys. Rev. **E51**, 4282 (1995)
9. S. P. Hoogendoorn and P. H. L. Bovy, in *Proceedings of the 15th International Symposium on Transportation and Traffic Theory, Adelaide, 2002*, edited by M. Taylor (University of South Australia, Adelaide, 2002), p. 219
10. P. Thompson and E. Marchant, Fire Safety Journal **24**, 131 (1995)
11. V. Schneider and R. Könnecke, in [1], p. 303
12. P. Molnár, *Modellierung und Simulation der Dynamik von Fußgängerströmen*, Dissertation, (Shaker, Aachen, 1996)
13. D. Helbing, I. Farkas and T. Vicsek, Phys. Rev. Lett. **84**, 1240 (2000)
14. D. Helbing, I. Farkas, and T. Vicsek, Nature **407**, 487 (2000)
15. D. Helbing, Rev. Mod. Phys. **73**, 1067 (2001)
16. D. Helbing, I. Farkas, P. Molnar and T. Vicsek, in [1], p. 21
17. T. Werner and D. Helbing, in [2], p. 17
18. T. Meyer-König, H. Klüpfel and M. Schreckenberg, in [1], p. 297
19. A. Kirchner, H. Klüpfel, K. Nishinari, A. Schadschneider and M. Schreckenberg, J. Stat. Mech. **P10011** (2004)
20. R. Barlovic, T. Huisinga, A. Schadschneider and M. Schreckenberg: *Open boundaries in cellular automaton model for traffic flow with metastable states, Physica E* **66**, 046113 (2002)
21. S. P. Hoogendoorn, P. H. L. Bovy and W. Daamen, in [1], p. 123
22. `www.rimea.de`

23. U. Weidmann, *Transporttechnik der Fußgänger*, Schriftenreihe des IVT Nr. 90, zweite ergänzte Auflage, ETH Zürich (1993)
24. S. P. Hoogendoorn, W. Daamen, Transportation Science **39/2**, 0147 (2005)
25. P. D. Navin and R. J. Wheeler, Traffic Engineering **39**, 31 (1969)
26. A. Seyfried, B. Steffen, W. Klingsch and M. Boltes, available for download at `http://arxiv.org`, physics/0506170 (2005)
27. A. Seyfried, B. Steffen, T. Lippert: *Basics of Modelling the Pedestrian Flow, Physica A* **368**, 232-238 (2006)
28. J. L. Pauls, in *Proceedings of the 3rd International Symposium on Human Behaviour in Fire, Belfast, 2004* (Intersience Communications, London, 2004)

# Implementation of a Tabulated Barnes-Hut-Ewald Algorithm for Periodic Boundaries in PEPC

Falk Eilenberger

Universität Jena
Zentrum für Innovationskompetenz Ultraoptik
Institute for Applied Physics

falk.eilenberger@uni-jena.de

**Abstract:** The parallel Barnes-Hut program PEPC was augmented to support periodic boundary conditions. In consequence PEPC now supports a tabulated Barnes-Hut-Ewald mode, allowing periodic boundary simulations for up to and beyond $10^6$ particles. An analysis of errors with respect to different parameter sets was carried out. It could be shown that the additional error introduced by the periodic boundary conditions can be made small or comparable to the intrinsic Barnes-Hut error with reasonable effort. A new interpolation scheme has been developed that achieves higher accuracy and is key for the success of low error simulations in the range of $\vartheta \leq 0.3$. It could also be shown that the code has to do less communication per time, which will improve its scalability to higher numbers of CPUs. A strategy for reducing expensive load operations arising from the Ewald tabulation is also proposed.

## Introduction

*The N-body problem*

The so-called N-body problem with long-ranged interactions still represents one of the great challenges of computational physics[3]. Solutions to this problem promise insight into systems of various fields of science such as Newtonian stellar dynamics[1], physics of non-degenerate plasmas, molecular dynamics of ionic constituents[8] and the interaction of matter with radiation from high-power lasers[6].

The central question of the N-body problem is the solution to the equations of motion for a large set of particles that are subject to internal and external forces and for which initial conditions are known:

$$m_i \ddot{\mathbf{x}}_i = \mathbf{F}_i(\mathbf{x}_1, \ldots, \mathbf{x}_N, t) \qquad \forall i, j \in [1, N] \tag{1}$$

Usually internal interactions are pairwise, isotropic, and conservative:

$$\mathbf{F}_i(\mathbf{x}_1, \ldots, \mathbf{x}_N, t) = \sum_{j \neq i} \nabla_i \phi_{ij}(r_{ij}) + \mathbf{F}_{ext}(\mathbf{x}_i, t) \qquad r_{ij} = |\mathbf{x}_j - \mathbf{x}_i| \tag{2}$$

Conservation of momentum, angular momentum and energy are intrinsic to this set of equations if $\mathbf{F}_{ext} = 0$. Thus one has to solve the quivalent of $6N - 7$ coupled, non-linear differential first order equations. Even for very simple potentials such as $\phi \sim 1/r$ and for only a few particles ($N \geq 3$) there is no closed form analytic solution to this problem. Worse, the only known convergent series expansion does so very slowly[9].

Numerical solutions on the other hand are by no means straightforward. If one wishes to achieve accurate trajectories for individual particles one has to deal with positive Ljapunow exponents and resulting exponential amplification of numerical errors. If one is just interested in global properties the latter problem may well be ignored. But as each of the $N$ particles interacts with $N-1$ partners, the calculation of the forces at each timestep of the numeric integration scheme has a computational complexity that scales as $\mathcal{O}(N^2)$, which is prohibitively expensive for large numbers of particles even in the era of multi-teraflop supercomputing.

*Multipole expansion*

A common approach to overcome this problem is based on the fact that the interaction potential and force decay with increasing distance. It is thus possible to group distant particles and compute their influence as a compound *pseudoparticle*.

Therefore one usually uses a multipole expansion of the force-sum in (2). Consider for example the contribution of a set of particles $\mathcal{J}$ that are close to each other, but far away from particle $i$ to the potential of that particle $\phi_i^{\mathcal{J}}$ for a Coulomb-style interaction:

$$\begin{aligned} \phi_i^{\mathcal{J}}(\mathbf{x}_i, \mathbf{x}_{\mathcal{J}}) &= \sum_{j\in\mathcal{J}} \frac{q_j}{|\mathbf{x}_i - \mathbf{x}_j|} \\ &= \sum_{j\in\mathcal{J}} \left( \frac{q_j}{|\mathbf{x}_i - \mathbf{x}_{\mathcal{J}}|} + \frac{q_j \mathbf{x}_i}{|\mathbf{x}_i - \mathbf{x}_{\mathcal{J}}|^3} \cdot (\mathbf{x}_j - \mathbf{x}_{\mathcal{J}}) + \dots \right) \end{aligned} \tag{3}$$

where $\mathbf{x}_{\mathcal{J}} = (\sum |q_j| \mathbf{x}_j) \cdot (\sum |q_j|)^{-1}$ is the center of charge of pseudoparticle $\mathcal{J}$. The sum is expanded into a Taylor series around $\mathbf{x}_{\mathcal{J}}$ with respect to $\mathbf{x}_j$.

All terms that depend on $i$ can be pulled out of the sum since they do not depend on $j$. Physically $\mathcal{J}$ now acts as a single pseudoparticle with compound properties, so-called multipole moments, that are located at $\mathbf{x}_{\mathcal{J}}$. The multipole moments of the $n^{th}$ order are connected with a potential that decays as $r^{-(n+1)}$. Since the distance to particle $i$ is large, higher order moments can be neglected. Usually terms up to the $2^{nd}$ order, so called quadrupole terms, are taken into account. Equation (3) thus takes the form:

$$\phi_i^{\mathcal{J}}(\mathbf{x}_i, \mathbf{x}_{\mathcal{J}}) \approx q_0^{\mathcal{J}} \cdot \phi_0(\mathbf{x}_i - \mathbf{x}_{\mathcal{J}}) + q_1^{\mathcal{J}} \cdot \phi_1(\mathbf{x}_i - \mathbf{x}_{\mathcal{J}}) + \frac{1}{2} q_2^{\mathcal{J}} \cdot \phi_2(\mathbf{x}_i - \mathbf{x}_{\mathcal{J}}) \tag{4}$$

where the quantities $q_n^{\mathcal{J}}$ and $\phi_n$ denote tensors of rank $n$ and the "$\cdot$" denotes summation over all common indices. The quantities in question are defined as:

$$\begin{aligned} q_0^{\mathcal{J}} &= \sum_{j\in\mathcal{J}} q_j \\ (q_1^{\mathcal{J}})_i &= \sum_{j\in\mathcal{J}} q_j (x_j^{(i)} - x_{\mathcal{J}}) \\ (q_2^{\mathcal{J}})_{i_1 i_2} &= \sum_{j\in\mathcal{J}} q_j (x_j^{(i_1)} - x_{\mathcal{J}})(x_j^{(i_2)} - x_{\mathcal{J}}) \\ \phi_0(\mathbf{x}) &= \frac{1}{|\mathbf{x}|} \\ (\phi_1(\mathbf{x}))_i = \frac{\partial \phi_0(\mathbf{x})}{\partial x^{(i)}} &= \frac{x^{(i)}}{|\mathbf{x}|^3} \\ (\phi_2(\mathbf{x}))_{i_1 i_2} = \frac{\partial (\phi_1(\mathbf{x}))_{i_1}}{\partial x^{(i_2)}} &= \frac{3 x^{(i_1)} x^{(i_2)} - \delta_{i_1 i_2} |\mathbf{x}|^2}{|\mathbf{x}|^5} \end{aligned} \tag{5}$$

and $x_j^{(i)}$ means the $i^{th}$ cartesian component of the position of the $j^{th}$ particle. Forces can be obtained by differentiating equation (4) with respect to $\mathbf{x}_i$. Formulas for higher order terms can be obtained easily by adding more cartesian components to the multipole moments and consecutive application of derivatives to the fields in equation (5).

*The Barnes-Hut algorithm*

Now that we can treat groups of distant particles as compound pseudoparticles we need to find a fast way to group the particles according to their position in space, compute the group's multipole moments in an efficient manner and find out which particles interacts with which other particles or pseudoparticles.

*Building the tree*

A very simple and efficient algorithm of grouping particles is by division over each spatial dimension. This process is repeated in a recursive manner until no segments contains more than one particle. During the process a tree of the segments is build where each leaf represents a particle and each twig represents a segment with more than one particle. Empty boxes are ignored. A sketch of the way the algorithm works is supplied in fig. 1.

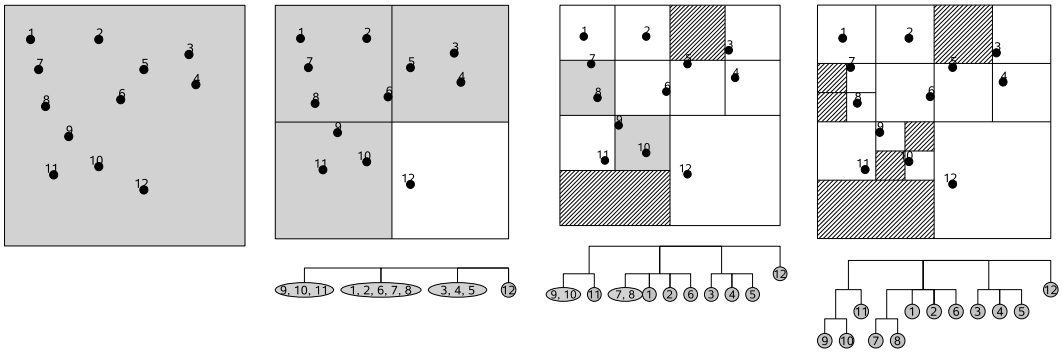



Figure 1: Successive states of the tree-building algorithm. Blue denotes unfinished segments with multiple particles and corresponds to tree twigs. White segments are finished and contain a single particle and correspond to leafs. Striped segments contain no particle.

*Filling the tree*

The tree now does not act as a mere way of managing the particles but leaves us with a natural hierarchy. Each twig of the tree is associated with a pseudoparticle, consisting of all daughter-particles, which are either other pseudoparticles or eventually the particles themselves. The multipole moments are calculated in a recursive transversal of the tree starting at its leaves. Multipole moments of pseudoparticles that only have particles as daughters can be calculated by means of (5) whereas the multipole moments of daughter-pseudoparticles can be combined to give the moments of their parent (see [5]), as skecthed in fig. 2.

*Interaction list building*

The tree itself also contains information on the distance between particles. If a particle $i$ has a certain distance $d$ to a pseudoparticle $\mathcal{J}$ and this pseudoparticles box-length is $s$ then $i$'s distance to the constituents of $\mathcal{J}$ will not vary much, provided $s$ is small in comparison to $d$. It is thus feasible not to resolve the pseudoparticle $\mathcal{J}$. If $s$ is large in comparison to $d$, the multipole expansion will produce errors because

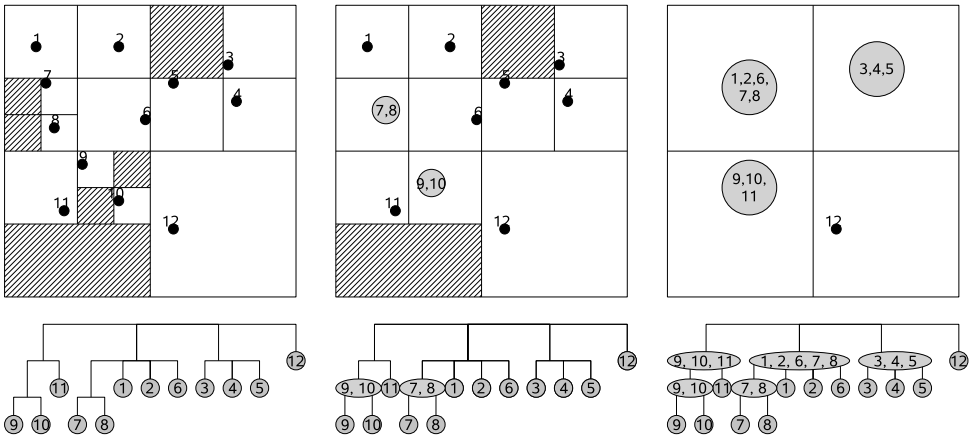



Figure 2: Successive states of the pseudoparticle generating algorithm. Colors are the same as in fig. 1, pseudoparticles are marked purple. See [5] for efficient calculation of the pseudoparticles' moments.

the convergence of (4) relies on the condition that the diameter of the space occupied by $\mathcal{J}$ is small compared to its distance to $i$.

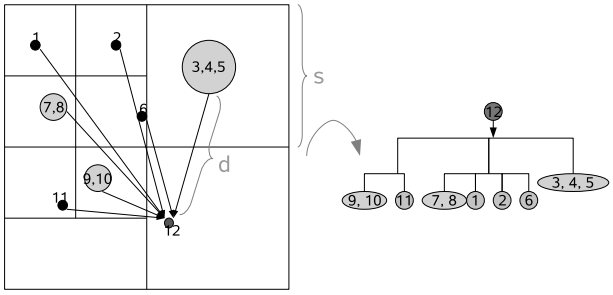



Figure 3: Example for the layout of an interaction list for particle number 12

This immediately gives us a recipe for an algorithm that determines a set of interaction partners for a given particle $i$. Starting at the root pseudoparticle of the tree one determines the distance between the particle and the pseudoparticle $d = |\mathbf{x}_i - \mathbf{x}_{\mathcal{J}}|$ and the box-length $s$ of the pseudoparticle which is related to the size of the total simulation box $L$ and the tree-level of the pseudoparticle $l$ by $s = 2^{-l}L$. If the equation

$$s < d\vartheta \tag{6}$$

is valid then the pseudoparticle does not need to be resolved further. If it is not satisfied the pseudoparticle is resolved into its daughters for which the process is repeated. If a pseudoparticle is a leaf then it is of course not resolved for it represents a particle.

Equation (6) is called a *Multipole-Acceptance-Criterion* and was proposed by [1]. Other MACs with different behavior in terms of speed and accuracy are possible without much change to the algorithm. A thorough analysis and comparison of a multitude of MACs for open boundary conditions can be found in [7].

Note that the behavior of the tree-code can be influenced by choosing the MAC-parameter $\vartheta$ appropriately. A smaller choice of $\vartheta$ will lead to a better resolution of the tree and thus less errors. A larger choice will sacrifice accuracy for speed. All finite choices of $\vartheta$ lead to an asymptotic scaling of $\mathcal{O}(N \log N)$. Useful values are in the range from $0.3$ to $1.0$ providing a good trade-off between speed and accuracy.

*Integration of the equations of motion*

Once the interaction list is known formula (4) can be added up for all interaction partners and gives a total force acting on particle $i$. This is used to integrate the equations of motion (1) numerically. A discussion of possible choices of integration schemes can be found in [10]. PEPC either uses a simple leap-frog scheme or a relativistic Boris rotation scheme.

*Summarisation of a single time-step of the BH-algorithm*

1. build tree by consecutive division of space
2. fill tree twigs with pseudoparticle properties (multipole moments)
3. determine interaction list for each particle
4. calculate Coulomb-forces for partners in interaction list
5. solve equations of motion

*PEPC*

The program package *PEPC* [2] is a parallel implementation of the BH-tree algorithm. Designed as a **P**retty **E**fficient **P**arallel **C**oulomb Solver it has grown somewhat beyond that original scope. It can be used to solve problems involving Newtonian gravitation, complex short range interactions such as Lennard-Jones forces and the interaction with external fields such as intense laser radiation. It also includes features for visualisation and computational steering (see [3]).

The parallelisation procedure is described in [2], some ideas are borrowed from [11], and contains two important steps. Particles are sorted not by an arbitrary particle index but by a Morton-z-ordering space-filling curve. This curve is cut in, more or less, equal segments (depending on the load balance) and the particle informations belonging to these segments are distributed among the CPUs involved in the computation. The last 63 bits of value of the Morton-curve are used to create a particle index and represent the position of the particle in memory. Transversal towards the root of the tree can be done simply by right-shifting the Morton index by 3 bits.

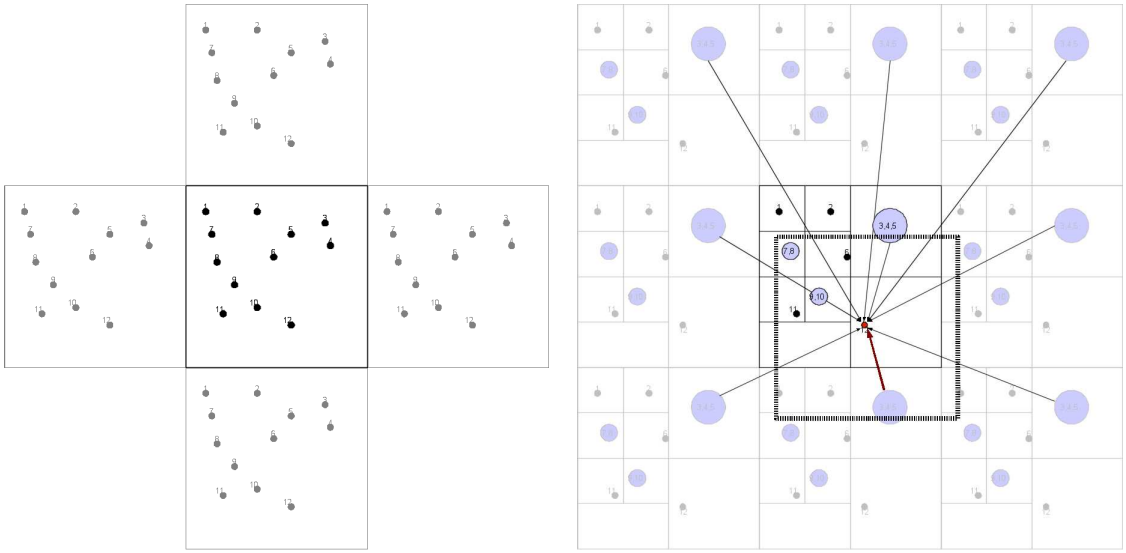
The pseudoparticle information of the most top-level cells belonging to a single CPU (so-called branch-nodes) is shared among all CPUs. If the building of an interaction list needs to resolve cells that are nonlocal to a CPU the cell's data is requested from and sent by the CPU owning it.

**Periodic boundaries**

Even though the combination of multi-teraflop computing and efficient algorithms that reduce the computational effort of the N-body problem with pair-interactions to a tolerable complexity of $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ has opened the door to multi-million particle simulations the simulation of macroscopic bodies with ab-initio methods is sill many orders of magnitude away.

Consider for example a state-of-the-art (by today's standard) ensemble of $N = 10^9$ particles. A typical spacing for particles in condensed matter is well below $1nm$, but assume this is $1nm$ for the sake of simplicity. The ensemble of $N = 10^9$ still only fills a cube with a length of $1\mu m$. This tiny volume does not reflect the properties of a a bulk material one is interested in. It is to a great extend influenced by its boundaries. Interaction with visible light will for example be dominated by the geometric structure of the volume, not by its average dielectric and magnetic properties.

*Reformulation of the force sum*



(a) (b)

Figure 4: (a)Periodic boundary conditions. The central box is the actual simulation region which is surrounded by an infinite number of identical copies. (b)Nearest neighbor search for a certain interaction. Note that the nearest neighbor is not necessarily the particle in the original Simulation box, but may be an image. The dark box marks the particles' minimum image cell, the region the N.N. can be found in.

The usual approach to this type of problem is the introduction of periodic boundary conditions. The simulation box that contains the system is surrounded by an infinite number of identical copies of itself. This is of course not done for real but the dynamics of the ensemble is modified as if it was surrounded by identical copies. The force sum (2) thus takes the form:

$$\mathbf{F}_i(\mathbf{x}_1,\ldots,\mathbf{x}_N,t) = \sum_{\mathbf{n}\in\mathbb{Z}^3} \sum_{\substack{j\neq i \\ \text{if } \mathbf{n}=\mathbf{0}}} \nabla_i \phi_{ij}(r_{ij\mathbf{n}}) + \mathbf{F}_{ext}(\mathbf{x}_i,t) \qquad r_{ij\mathbf{n}} = |\mathbf{x}_i - \mathbf{x}_i + \mathbf{n}L| \tag{7}$$

One has in general to distinguish between short-range and long-range interaction. A criterion is given by [10] which states that potential is short-ranged for 3D space if it asymptotically decays faster than $r^{-3}$. Short ranges potentials (such as the Lennard-Jones potential) can be treated by introducing a cutoff radius for $|\mathbf{n}|$. It is often sufficient to include a single interaction for the nearest neighbor (N.N). The situation is much worse for the Coulomb potential which is very long-ranged. The sum (7) is only conditionally convergent and would need to process $10^{10^3}$ terms to achieve summands in the order of $10^{-10}$ which is totally unacceptable.

*Ewald's method*

Ewald could show that the periodic potential sum (7) can be rewritten with improved convergence, when split in a real space and a reciprocal space sum:

$$\phi_{Ew}(|\mathbf{x}|) = \sum_{\mathbf{n}\in\mathbb{Z}^3} \frac{1}{r_{\mathbf{n}}} = -\sqrt{\pi} + \sum_{\mathbf{n}\in\mathbb{Z}^3} \frac{\operatorname{erfc}(\alpha r_{\mathbf{n}})}{r_{\mathbf{n}}} + \frac{1}{2L} \sum_{\mathbf{h}\in\mathbb{Z}^{\mathbf{3}}\neq\mathbf{0}} \cos(\frac{2\pi}{L}\mathbf{h}\cdot\mathbf{r}) \exp(-\frac{\pi^2 h^2}{L^2\alpha^2}) \tag{8}$$

The terms of both sums decay as fast as $e^{-n^2}$. The parameter $\alpha$ is arbitrary and can be used to tune the convergence of the two sums. A superficial analysis shows that $\alpha L = \sqrt{\pi} \approx 2$ is a good choice that

balances the convergence of both series and leads to the least computational effort. Further remarks can be found in [4]. [10] proposes a method that speeds up the calculation of the second term and would make a larger choice for $\alpha$ seem more appropriate, because it slows the convergence of the reciprocal space term and speeds up the real space sum.

To determine the cutoff radius for the sums we use the following approximations: $\mathrm{erfc}(r) \approx \exp(-r^2)$, $r^{-1} \approx 1$, $L^{-1}cos(r) \approx 1$. For the proposed choice of $\alpha L = \sqrt{\pi}$ and a cutoff radius of $n_{cut} = h_{cut} = 3$ the summands have decayed to $\exp(-9\pi) \approx 10^{-12}$ which is in the region of machine precision and certainly below the threshold of the error introduced by the BH algorithm.

*The Barnes-Hut-Ewald algorithm*

Now we have to apply Ewald's method to the Barnes-Hut algorithm. Recall that the basic idea of that algorithm is to use compound multipoles instead of single particle-particle interactions. So we have to expand (8) into multipole fields. This is done by consecutive application of the gradient to (8) in very much the same way as in (5). The definition of the multipole moments do not change. Since the results are rather long and give no further insight they are omitted and can be found in [5], though with some typos.

Now we need only change the BH-algorithm in a few points and it now looks like this:

1. build tree by consecutive division of space
2. fill tree twigs with pseudoparticle properties (multipole moments)
3. determine interaction list for each particle *considering nearest neighbor image*
4. *calculate Ewald-forces for partners in interaction list*
5. solve equations of motion
6. *particles that leave the box have to reenter from the opposite side*

*The tabulated Barnes-Hut-Ewald algorithm*

In principle we now have found a feasible way to include periodic boundary conditions into the Barnes-Hut algorithm. But this is indeed not quick enough. The functions involved in Ewald's formula (8) are very expensive to calculate and a cutoff radius of $n_{cut} = h_{cut} = 3$ still leaves us with more than 130 terms to sum up for each part of the sum.

A method already used by [8] splits the Ewald potential in two parts:

$$\phi^{Ew} = \phi^{Ew} - \phi^{Coul} + \phi^{Coul} = \phi^{EC} + \phi^{Coul} \tag{9}$$

The second part is the well-known Coulomb-potential of the NN, which is calculated in the classical way. The second part is the contribution from all other particles. This contribution varies slowly and can be calculated on a spatial grid spanning the cube $[-L/2, L/2]^3$ in advance. An interpolation scheme will be used to give values for arbitrary points in the cube.

The above list has thus to be completed by the calculation of the Ewald-correction at the beginning of the simulation run. Two forces are now taken into account - the standard Coulomb-force as well as the interpolated Ewald-correction.

**Implementation of the tBHE algorithm**

*Calculation of the Ewald-correction*

Calculation of the Ewald-correction table is the only conceptual novel part for the inclusion of periodic boundary conditions into PEPC. Since twenty moments have to be stored for each point of the computational grid the computational cost and memory requirements for a table of a typical size of $128^3$ grid points is considerable. The required 335 MB of memory per CPU can not be easily supplied by low memory machines such as the IBM BlueGene/L architecture. An order of $10^5 \ldots 10^6$ flops/grid-point would easily pay the effort of parallelization.

Those costs can be greatly reduced if one exploits at least some of the symmetries inherent in the Ewald quantities. The most obvious is the invariance under the change of sign of one or more cartesian components of the argument:

$$\phi^{EC}(\pm x^{(1)}, \pm x^{(3)}, \pm x^{(3)}) = \phi^{EC}(x^{(1)}, x^{(3)}, x^{(3)}) \qquad \frac{\partial}{\partial(-x^{(i)})} = -\frac{\partial}{\partial x^{(i)}} \tag{10}$$

Derivatives of the Ewald potential, that correspond to the fields of charges and multipoles thus transform like their indices, ie:

$$(\phi_2^{EC}(\pm_1 x^{(1)}, \pm_2 x^{(3)}, \pm_3 x^{(3)}))_{i_1 i_2} = \pm_{i_1} \pm_{i_2} (\phi_2^{EC}(x^{(1)}, x^{(3)}, x^{(3)}))_{i_1 i_2} \tag{11}$$

This reduces the cost for computation and memory by a factor of 8. The table has to be calculated and stored only on the octant $[0, L/2]^3$. When the algorithm requests values for the Ewald-correction all signs of the arguments components are set to "+". After interpolation the signs of the results are corrected to the actual octant using rules like (11).

The second symmetry that is exploited is produced by the invariance of $\phi_{Ew}$ under the permutation of arguments, ie.: $\phi_{Ew}(P\mathbf{x}) = \phi_{Ew}(\mathbf{x})$. Once all values of the Ewald-correction for a certain triplet of cartesian coordinates $(x^{(1)}, x^{(2)}, x^{(3)})$ is known the values of the correction at the points with permuted coordinates $(x^{(\pi_1)}, x^{(\pi_1)}, x^{(\pi_1)})$ can be obtained by a certain permutation. For example:

$$\begin{aligned} (\phi_1^{EC}(x,y,z))_1 &= (\phi_1^{EC}(x,z,y))_1 = (\phi_1^{EC}(y,x,z))_2 \\ &= (\phi_1^{EC}(z,x,y))_2 = (\phi_1^{EC}(z,y,x))_3 = (\phi_1^{EC}(y,z,x))_3 \end{aligned} \tag{12}$$

This symmetry is only used to speed up computation. Results are stored for all triples $(x^{(1)}, x^{(2)}, x^{(3)})$. It can only be exploited if all cartesian components have different values. In practice we achieve a speedup of factor $5.6$, which is pretty close to the theoretical value of factor 6.

The symmetry under arbitary rotations of the Coulomb potential is lost by introduction of cubic image cells. Further symmetries remain uninvestigated. It is i.e. known that the Coulomb-multipole fields of $n^{th}$ order have no more then $2l + 1$ linearly independet entries. Instead of 20 fields only 16 fields would remain. But this symmetry is closely related to the symmetry under arbitary rotations and might be lost, too.

The exploitation of permutation symmetry is closely related to parallelization issues. If it was not exploited one could simplify go through the Ewald-grid and assign every $P^{th}$ point to each of the $P$ CPUs. But permutation symmetry requires one CPU to handle all permutation of each triplet that it is originally designed to. Thus all triplets that do not fulfill the relation $x^{(1)} \leq x^{(2)} \leq x^{(3)}$ are rejected.

A problem arises if the number of grid points per cartesian axis $n_{EC}$ and the number of CPUs $P$ are somehow correlated (ie. they are equal). Some CPUs might end up rejecting all points while others have to carry their load. A successful suppression of this effect was achieved by a simple scheme: within each group of $P$ grid points the $p^{th}$ CPU does not get the same point for every group, but it is mapped onto the group by a pseudo-random series of numbers related to the Fibonacci numbers. Thus any point that a certain CPU calculates is unrelated to the next one, except that it is no more that $2P$ and no less than $1$ grid points away.

Once all CPUs have calculated their share of corrections the results are broadcast so that each CPU has a copy of the complete Ewald-correction grid. Tests show that a distribution scheme seems unnecessary, no more than $64^3$ points are needed for good accuracy, which needs about 40 MB of memory.

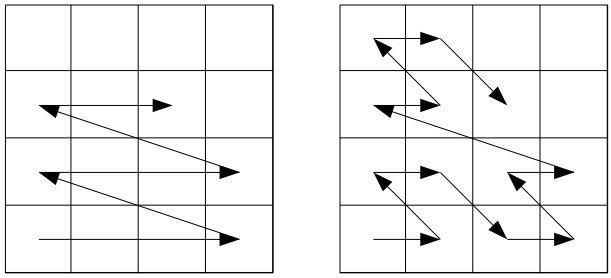


Figure 5: Different schemes for the memory layout of a multidimensional array

Apart from the usual storage scheme for multidimensional arrays ($x$-index is fastest, all others are slow) z-ordered storage was also examined (see fig. 5). The mapping between the z-ordering index $\zeta$ and the cartesian index triplet $(i, j, k)$ is fairly simple, if those indices are given in binary form:

$$\zeta_{3n}\zeta_{3n-1}\zeta_{3n-2}\ldots\zeta_3\zeta_2\zeta_1 = k_n j_n i_n \ldots k_1 j_1 i_1 \tag{13}$$

this can be calculated by simple ISHFT and IOR operations and takes about 30 CPU cycles in each direction, compared to less than 5 CPU cycles for the conventional layout. When the Ewald grid is used for interpolation one always has to load the values of 8 adjacent points that form a fundamental cube. If one uses z-ordered memory layout the probability for a cache miss is lower because adjacent points have a high probability to be close in memory, which is only true for the $x$-direction for conventional layout. A cache miss on JUMP should cost about 100 CPU cycles. A reduction in cache misses might therefore outperform conventional memory layout techniques even though the mapping from cartesian space to memory will be slower.

Tests show that this hope cannot be fulfilled an a system like the IBM/P690 JUMP. The documentation for its Power4 CPU suggests that the length for its cache lines are 128 bytes, whereas we store $20 \cdot 8 = 160$ bytes of data per Ewald grid point. We thus produce cache misses no matter how close adjacent grid point are in memory. A second reason is the fact that even for the conventional memory layout $50\%$ of the data is in neighbouring memory cells, which is no worse than z-ordered layout. Even a reduction of the data per grid point by a factor of 2 (no quadrupole corrections) would not help much because z-ordering will only outperform conventional storage if the cache is at least about 4 times the size of the data per grid point.

Still this feature remains implemented as future machines with larger cache lines with eventually profit from this memory scheme. It can be turned on by setting a certain switch in the simulation's configuration file.

*Interpolation issues*

An crucial issue for the performance, memory usage and accuracy for the tBHE algorithm is the interpolation of the Ewald-correction. Later tests will show that a large portion of the simulation time is spent for fetching data from memory and computing interpolation.

The most straightforward algorithm is a volume-weighted mean of all values of the Ewald-correction of the 8 adjacent grid points, as used e.g. by [6]. This is rather fast and accurate to the first order of the grid-spacing $\Delta$. We diverged slightly from this scheme and implemented and tested two possible interpolation schemes:

1. a modified first-order volume weighted scheme with smaller errors
2. a second-order algorithm taking into account the values of the derivatives (which are known for all Ewald-corrections down to dipole forces/quadrupole potentials)

*Modified first-order scheme*

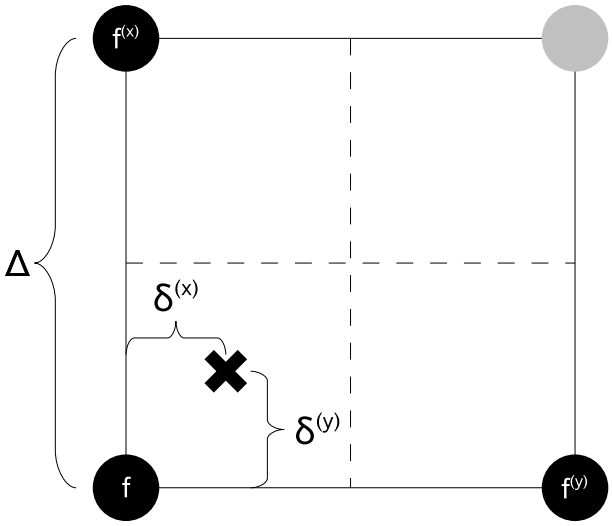



Figure 6: A 2 dimensional version of the process described above. The functional values on the dark grid points are used to approximate the value of the function on the spot that is marked by the cross. The superscripted functional values belong to those corners of the cube that share all coordinates with the top of the tetrahedron except the superscripted one. The second order interpolation scheme also uses the values of the gradient for the function on the dark points.

Instead of using the values of the function on all 8 adjacent points of the surrounding grid, we use values on the isosceles tetrahedron with its top on the grid point closest to the point in question (see fig. 6).

The resulting formula has the following form:

$$\begin{aligned} f(\vec{\delta}) &= f + \frac{(f - f^{(x)})x + (f - f^{(y)})y + (f - f^{(z)})z}{\Delta} + \mathcal{O}\left(\frac{|\vec{\delta}|}{\Delta}\right)^2 \\ x &= \begin{cases} \delta^{(x)} & \delta^{(\mathtt{x})} \leq \Delta/2 \\ \Delta - \delta^{(x)} & \delta^{(\mathtt{x})} > \Delta/2 \end{cases} \end{aligned} \tag{14}$$

Compared to the usual volume averaging method this method has a numerically smaller error. This can be understood because this algorithm uses more local information and does not rely on the more remote influence of the farther corners of the cube.

*Second-order scheme*

Usually higher order interpolation schemes such as splines rely on the values of the function on even more remote points. This strategy has the disadvantage that we would have to fetch much more data from memory, which would make this scheme overly expensive. On the other hand we have the advantage that we know the function's gradient for the monopole potential, the monopole force and the dipole force. Thus for the first 10 fields we can use them as well. The influence of the periodic images of quadrupole forces is small anyway ($\sim r^{-4}$) so the first order interpolation scheme can be applied for this moment without much of an error.

Our scheme uses the same functional values as described above but also uses the components of the gradient on the closest grid point and the sum of the values of the components of the gradient on the other corners of the tetrahedron. The resulting formula is rather formidable. It can be obtained by solving the following system of linear equations for $f(\vec{\delta})$:

$$\begin{pmatrix} f \\ f^{(x)} \\ f^{(y)} \\ f^{(z)} \\ f_x \\ f_y \\ f_z \\ f^{(x)}_{x+y+z} \\ f^{(y)}_{x+y+z} \\ f^{(z)}_{x+y+z} \end{pmatrix} = \begin{pmatrix} 1 & x_1 & y_1 & z_1 & \frac{1}{2}x_1^2 & \frac{1}{2}y_1^2 & \frac{1}{2}z_1^2 & x_1y_1 & y_1z_1 & z_1x_1 \\ 1 & x_2 & y_1 & z_1 & \frac{1}{2}x_2^2 & \frac{1}{2}y_1^2 & \frac{1}{2}z_1^2 & x_2y_1 & y_1z_1 & z_1x_2 \\ 1 & x_1 & y_2 & z_1 & \frac{1}{2}x_1^2 & \frac{1}{2}y_2^2 & \frac{1}{2}z_1^2 & x_1y_2 & y_2z_1 & z_1x_1 \\ 1 & x_1 & y_1 & z_2 & \frac{1}{2}x_1^2 & \frac{1}{2}y_1^2 & \frac{1}{2}z_2^2 & x_1y_1 & y_1z_2 & z_2x_1 \\ 0 & 1 & 0 & 0 & x_1 & 0 & 0 & y_1 & 0 & z_1 \\ 0 & 0 & 1 & 0 & 0 & y_1 & 0 & x_1 & z_1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & z_1 & 0 & y_1 & x_1 \\ 0 & 1 & 1 & 1 & x_2 & y_1 & z_1 & y_1+x_2 & y_1+z_1 & z_1+x_2 \\ 0 & 1 & 1 & 1 & x_1 & y_2 & z_1 & x_1+y_2 & z_1+y_2 & x_1+z_1 \\ 0 & 1 & 1 & 1 & x_1 & y_1 & z_2 & x_1+y_1 & y_1+z_2 & x_1+z_2 \end{pmatrix} \cdot \begin{pmatrix} f(\vec{\delta}) \\ f(\vec{\delta})_x \\ f(\vec{\delta})_y \\ f(\vec{\delta})_z \\ f(\vec{\delta})_{xx} \\ f(\vec{\delta})_{yy} \\ f(\vec{\delta})_{zz} \\ f(\vec{\delta})_{xy} \\ f(\vec{\delta})_{yz} \\ f(\vec{\delta})_{zx} \end{pmatrix} \tag{15}$$

where $x_1$ is defined as $x$ in (14) and $x_2$ just the other way around, $f^{(x)}_{x+y+z}$ is shorthand for $f^{(x)}_x + f^{(x)}_y + f^{(x)}_z$.

For a discussion on timing issues and accuracy please refer to the following section.

*A word about performance and scaling*

An analysis of parallel scaling and performance is not a pressing issue, for this algorithm. We have not altered the way PEPC works in principle, so the $\mathcal{O}(N \log N)$ scaling remains unchanged. The scaling to a larger number of CPUs should actually *improve* because the most expensive step is the interpolation which is a purely local step. As the relative impact of communication (interaction list building and load balancing) gets smaller, scalability increases. A quantitative analysis of this scaling has not been carried out because some performance improvements for the interpolation scheme are planned (see last section) which would algorithmically change the way data is fetched from memory and would render an analysis obsolete.

A profiler run reveals that most time ($70\%$) is spent for interpolation. Of that first oder interpolation needs about $10\%$ (procedure is inlined), second order interpolation about $40\%$ (cannot be inlined) and collection of data from memory about $50\%$. There seems plenty of space for improvements here, as will be discussed in the last section.

Another test (see fig. 7) with a non-inlined first-order interpolation procedure (just about doubles the cost for that routine) and a $25\%$ slower second-order interpolation procedure showed that for all values of $\vartheta$

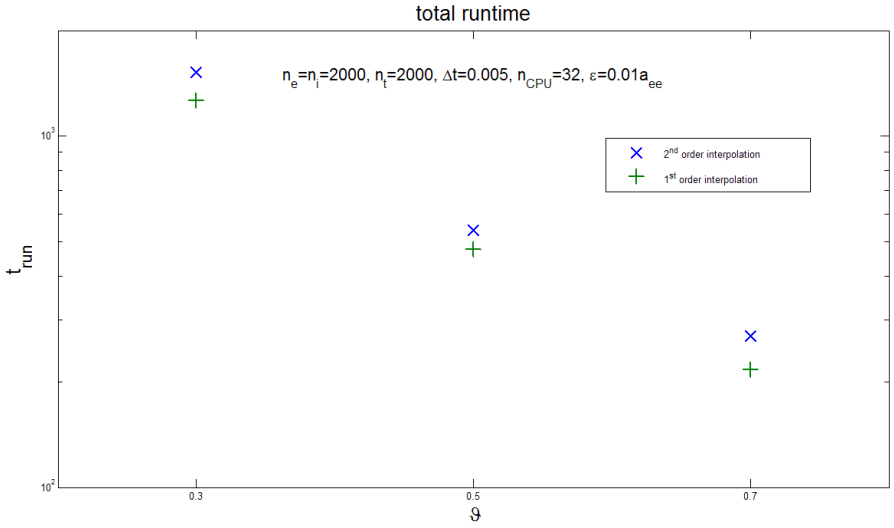



Figure 7: Wall-clock-times of the simulation for different values of $\vartheta$ and first/second order interpolation for 32 CPUs, an Ewald-grid of $32^3$ points, 4000 particles and 2000 timesteps

.

the additional cost for second order interpolation is roughly $20\%$ in comparison to first order interpolation.

**Analysis of errors**

The tBHE algorithm introduces three major errors into the solution of the equations of motion (1). They are:

1. The error due to the multipole grouping of the BH algorithm $\varepsilon_{BH}$
2. Interpolation errors for the Ewald-correction $\varepsilon_{EC}$
3. Integrator errors $\varepsilon_{int}$

A discussion on integrator errors can be found in [10] and is beyond the scope of this report.

*Single step force errors*

The first two errors influence the simulation by introducing incorrect forces. We analyze their influence for different parameters by comparing them to forces obtained by a direct pp-Ewald sum. A popular criterion is the mean squared error, given by:

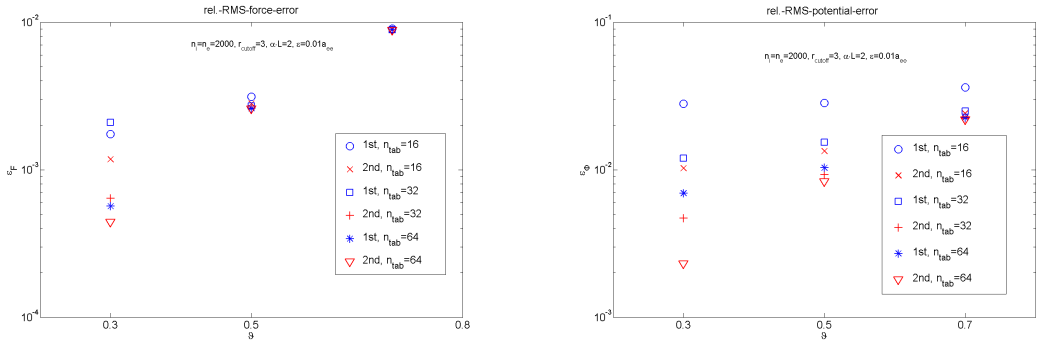
$$\langle \varepsilon \rangle^2 = \frac{\sum_i (f_i^{(\mathtt{tBHE})} - f_i^{(\mathtt{ppE})})^2}{\sum_i (-f_i^{(\mathtt{ppE})})^2} \tag{16}$$

where the sum runs over all particles. It is well suited for a “quick-and-dirty” analysis of force-errors. A more detailed insight can be obtained by the analysis of the empirical distribution function of the relative error:

$$\Phi(\varepsilon) = \frac{1}{N}\left(\sharp i : \log\left|\frac{f_i^{(\mathtt{tBHE})} - f^{(\mathtt{ppE})}}{f_i^{(\mathtt{ppE})}}\right| \leq \varepsilon\right) \tag{17}$$

where "$\sharp i : \ldots$" means "the number of $i$ that satisfy…".

Some results of formula (16) are plotted in fig. 8.




(a) errors of the forces

(b) errors of the potentials

Figure 8: Mean squared errors for an ensemble of 4000 particles

As the dynamic of the system is determined by the forces we will discuss them at first. The most dominant influence on the errors is produced by $\varepsilon_{BH}$. For large values of $\vartheta > 0.5$, which leads to a very coarse resolution of the particle tree, this error is so large that the choice of the integration scheme and the resolution of the Ewald grid does not influence the error significantly. This regime is suitable for testing or applications with a low accuracy requirement – the algorithm is fast anyway because of the high value of $\vartheta$, which can be further augmented by choosing the quicker interpolation scheme and using only a small correction table, without introducing new errors.

When $\vartheta$ is reduced well below $0.5$ the choice of interpolation scheme and grid resolution becomes more dominant. As this version of the algorithm runs almost an order of magnitude slower than the coarse $\vartheta = 0.7$ approach (see fig. 7) the additional investment of $20\%$ for second order integration and a large Ewald grid seems appropriate.

The situation is similar for the error of the potential. The effect of the interpolation error $\varepsilon_{EC}$ is more influential for all values of $\vartheta$. Only for very high values of $\vartheta > 0.7$ can its influence be neglected against the influence of $\varepsilon_{BH}$.

Smaller average errors can be achieved altogether for the forces, which are typically a factor of $4 \ldots 5$ more accurate than the potential, whose error can not be reduced to below $10^{-3}$ for reasonable choices of the parameters. This is no bad news, since the dynamics of the system is determined by the forces. On the other hand one needs to keep in mind that numbers for the potential energy and thus for the total energy can not be trusted to a precision of more than an $10^{-3}$.

The possible reason for that discrepancy in the achieved accuracy might be found in the slower decay of the potential ($\sim 1/r$ for a p.p. potential compared to $\sim 1/r^2$ the p.p. force). This should lead to a better convergence for the multipole expansion as well as to less variation for the Ewald correction, which both influence the accuracy in a positive way.

A more thorough analysis of the errors can be achieved by taking a look at the empirical distribution functions of the relative errors as shown in fig. 9. The $50\%$ quantile should almost represent the values of fig. 8 for $\vartheta = 0.3$. Although the order of magnitude is reproduced correctly, this is not true for the

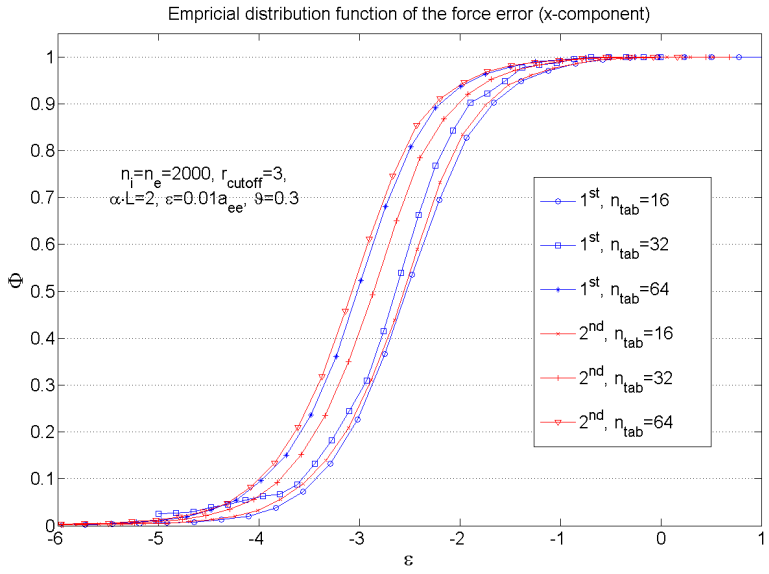



Figure 9: Distribution of the logarithm of the x-components of the relative force error, as defined by (17). This figures scenario is the same as the values of fig. 8 for $\vartheta = 0.3$

order of the data. There are two possible explanations, first: fig. 9 only represents the $x$-component of the forces whereas fig. 8 represents the average of all components; second: (16) includes a square/square root operation which will over-represent large errors and is not present in (17).

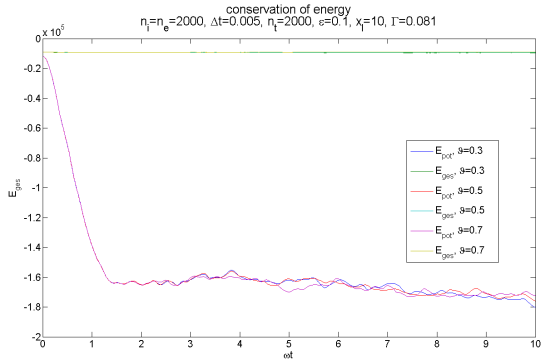
Two more interesting values are the $90\%$ quantiles and the maximum errors. The $90\%$ quantiles which are ordered in the same way as the $50\%$ quantiles, except that they are centered in the range of $10^{-2}$ and are on average one order of magnitude larger than the $50\%$ quantiles. So $90\%$ of the particles are moved by forces with less than $1\%$ error. The second interesting quantity is the largest relative error which is in the range of $100\%$ to $1000\%$ which seems extremely large. A preliminary analysis shows that the forces that have such huge errors are at least two orders of magnitude smaller than the average force, acting on a particle. So the impact on the total dynamics still is very small since the absolute force error can be neglected for those particles.

*Dynamical behavior*

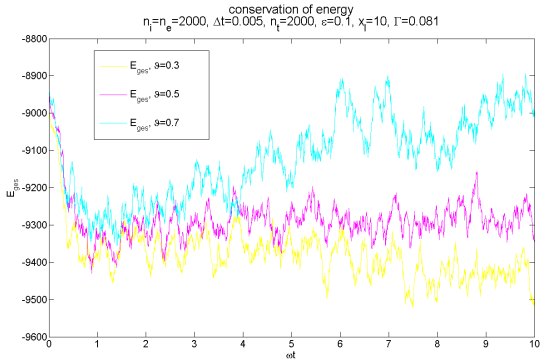
Now that we have gained some insight into the errors that occur in determining the forces it is time to see how the code performs in a more real-life scenario, with multiple timesteps. We test the relaxation of an ensemble of particles to equilibrium. The particles are initially distributed in a purely random fashion over the simulation box. The standard test for such a setup is to check if quantities that are conserved on a macroscopic level are conserved by the code as well or if the code's errors destroy such properties.

In the last subsection we have found that $\vartheta = 0.5$ and $n_{tab} \in 32, 64$ give a good trade-off between accuracy and speed. To determine an appropriate timestep we ran simulations with accurate pp-Ewald forces, thus excluding $\varepsilon_{BH}$ and $\varepsilon_{EC}$. We choose the timestep so that a relative fluctuation of the potential energy of $10^{-3}$ was achieved. By this method we determined a timestep of about $\omega\Delta t = 0.005$. This setup will be used in all following calculations. We will now investigate the influence of the coupling strength $\Gamma$ (see [6]), the BH-parameter $\vartheta$, and to a limited extend the number of particles on the relative fluctuation of the total energy.

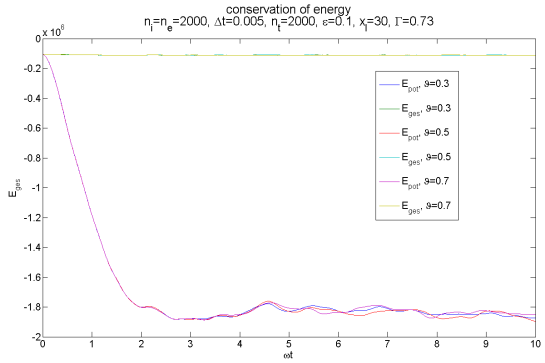
As can be see from fig. 10 and tab. 1 the energy conversion is met expectedly well. It is in the range of the



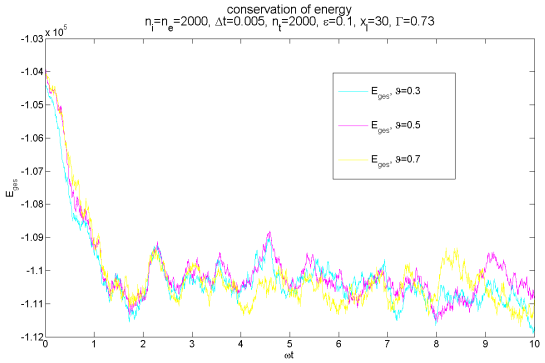

(a) tot. and pot. energy for a weakly coupled plasma



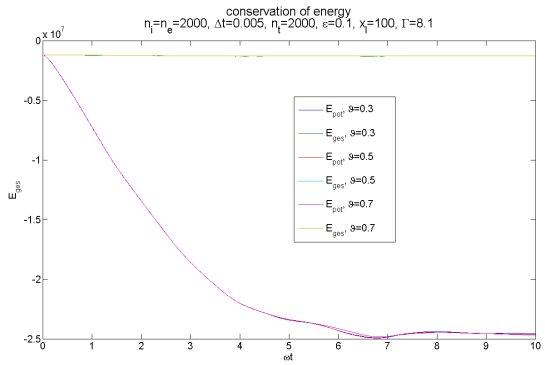

(b) pot. energy for a weakly coupled plasma



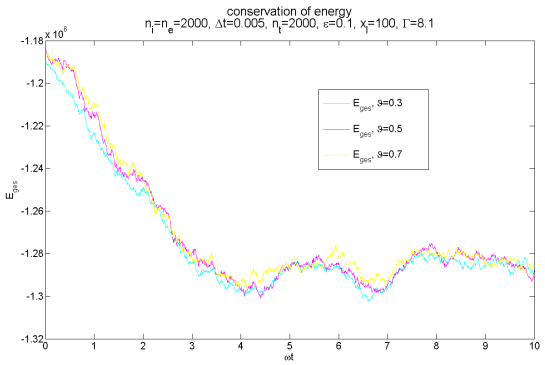

(c) tot. and pot. energy for a semi-coupled plasma




(d) pot. energy for a semi-coupled plasma




(e) tot. and pot. energy for a strongly coupled plasma




(f) pot. energy for a strongly coupled plasma

Figure 10: Energy curves for an initially randomly distributed ensemble of 4000 particles over 2000 timesteps. Summarized results can be found in 1

.

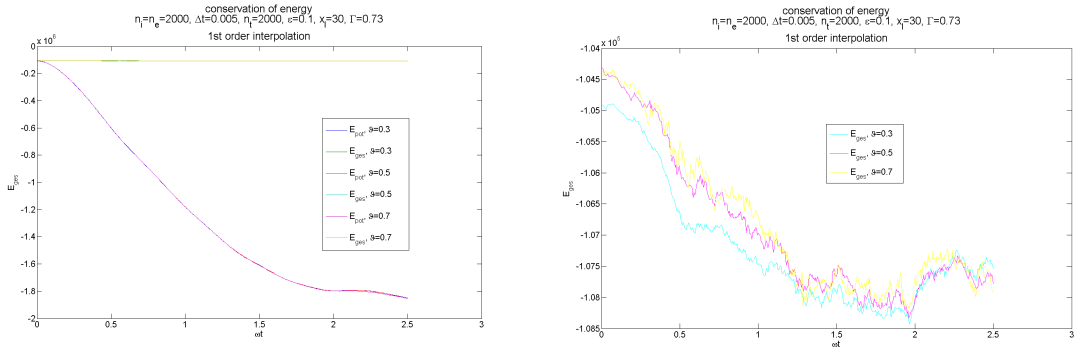
minimum average error for the potential determined in the last section to be no less than $10^{-3}$. Neither coupling strength nor the BH-parameter influence the result of the computation strongly. Only the weakly coupled plasma shows a dependence on $\vartheta$, where the fast version of the algorithm with $\vartheta = 0.7$ produces a slightly diverging behavior for the total energy.

All plots have in common that the initial phase of relaxation into an equilibrium with minimum potential energy is accompanied by a drop in the total energy. This is possibly due to the fact that the extreme non-equilibrium state at the beginning produces huge forces and involves many close encounters for which

| $\varepsilon_{energy} \cdot 10^3$ | $\Gamma = 0.081$ | $\Gamma = 0.73$ | $\Gamma = 8.1$ | $\Gamma = 0.73$ $1^{st}$ O | $\Gamma = 0.73$ $N = 4 \cdot 10^5$ |
|---|---|---|---|---|---|
| $\vartheta = 0.3$ | 3.5 | 4.0 | 4.8 | 2.5 | – |
| $\vartheta = 0.5$ | 3.5 | 4.0 | 4.8 | 2.5 | 4.0 |
| $\vartheta = 0.7$ | 3.9 | 4.0 | 4.8 | 2.5 | – |

Table 1: Relative fluctuation of the total energy for fig. 10, fig. 11, and fig. 12

the potential is not well modelled (see next section).



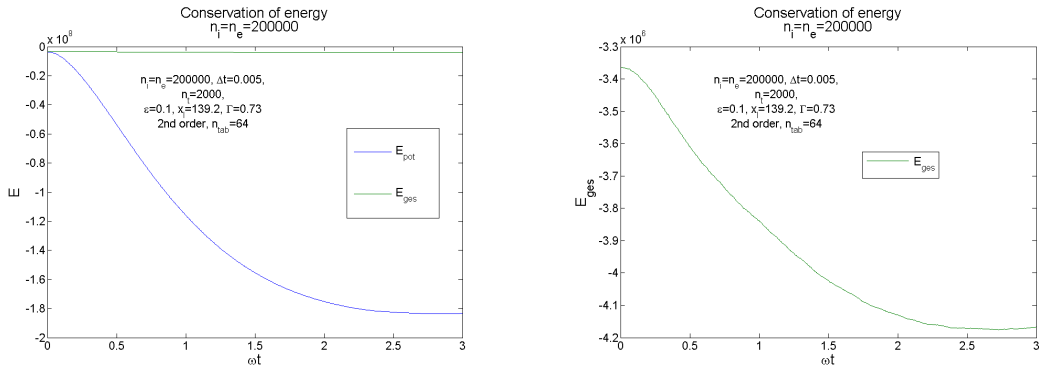

(a) tot. and pot. energy for a semi-coupled plasma

(b) pot. energy for a semi-coupled plasma

Figure 11: Energy curves for an initially randomly distributed ensemble of 4000 particles over 500 timesteps and a purely first order interpolation scheme.

Now we examine the behavior of the energy when a pure first order interpolation scheme is applied. Figure 11 has to be compared to the first quarter of graphs (c) and (d) of fig. 10. All curves look very similar. The initial drop in the total energy is somewhat smaller for the first order interpolation scheme, whereas the difference between the case $\vartheta = 0.3$ and the other cases is more visible. Those results are not unexpected. Only the case $\vartheta = 0.3$ is influenced to a major degree by $\varepsilon_{ec}$, even then the forces errors are rather small for the regime of $n_{tab} = 64$ as used in our case.

Altogether those observations confirm our results of the last section. $\vartheta = 0.5$ is usually a good trade-off between accuracy and speed. A second order interpolation can be avoided for semi-accurate simulation runs with $\vartheta \geq 0.5$, especially when the correction grid is dense ($n_{tab} \geq 64$). The smaller variation of the total energy for the $1^{st}$ order interpolation might be due to the shorter run time of just 500 timesteps.




(a) tot. and pot. energy for a semi-coupled plasma

(b) pot. energy for a semi-coupled plasma

Figure 12: Energy curves for an initially randomly distributed ensemble of 400000 particles over 2000 timesteps

Another interesting case is the behavior for large ensembles of particles. One would expect much smoother behavior for all statistics such as the temperature (kinetic energy) and the potential energy. This can be confirmed by fig. 12 which has a similar global relaxation behavior as smaller ensembles but experiences much less fluctuations.

**An outlook**

*Interpolation speedup*

As described earlier the interpolation procedure is the bottleneck in the current version of the code. It seems especially wasteful to spend a third of the time the code is executing for fetching data from the Ewald correction grid. In hope for a $20 - -25\%$ speedup I propose the following algorithm to reduce the effort for fetching data from memory: instead of calculation the Ewald correction for each particle by simply going through the interaction list entry by entry, one could build a "global-interaction-list" that contains the particle index, the pseudoparticle index, the cartesian indicies of the Ewald-grid cell and the octant. Displacement vetors $\vec{\delta}$ need not be stored, they can be cheaply recomputed. The particle index and the pseudoparticle index need 32 bits each, whereas cartesian indices and the octant need 16 bits each.

For the calculation of the Ewald correction this list is ordered by the cube index and processed in that order. The data of the Ewald grid would then have to be fetched only once per box, not once per interaction. Typical values are $n_{tab}^3 \approx 2.5 \cdot 10^5$ for the number of boxes in the correction grid, $N/P \approx 10^4$ for the number of particles per CPU and $10^3$ for the average number of interactions per particle. One can thus expect to have $10^1 \ldots 10^2$ interactions per grid point which could be processed by a single fetch or even half a fetch if one takes into account that neighbouring boxes share half of their points.

The price to pay would be some additional book-keeping and approximately $2 \cdot 8 \cdot 10^4 \cdot 10^3 = 150\mathtt{MB}$ of memory, as well as additional cache misses while adding the fields, because consecutive interactions no longer belong to the same particle.

*Soft particles*

One further error previously unmentioned is the artificial softening of particles. On the one hand the integrator has great problems to resolve close encounters because of the diverging nature of the Coulomb potential and all related forces. On the other hand we do not have to keep up the Coulomb potential for very close encounters – the nearfield of a particle will eventually be dominated by quantum effects and the actual charge distribution of that particle. It will most certainly not diverge.

Currently we resolve this problem by introducing an artificial softening parameter such as:

$$\frac{1}{r} \to \frac{1}{\sqrt{r^2 + \varepsilon^2}} \tag{18}$$

we apply this substitution for every formula that includes an $r$-term. This is correct for the coulomb interaction but actually destroys the Ewald sum. If one carries through the derivation described above for a charge distribution that gives a potential like (19) one gets a different real space sum and destroys the convergence of the Ewald sum, because the charge distribution is no longer $\delta$-shaped but decays polynomially which cannot be shielded efficiently by the counter-charge.

A more general analysis shows that for a substitution such as

$$\frac{1}{r} \to \frac{1}{\sqrt[n]{r^n + \varepsilon^n}} \tag{19}$$

one would have to choose an $n \geq 4$ and modify the Ewald sum (9) accordingly to achieve fast convergence as defined by [10]. Note that the limit $n \to \infty$ corresponds to a charged spherical shell with radius $\varepsilon$.

As a reasonably physical charge distribution I propose a distribution with exponential decay $\sim \exp(-r/a)$, which would lead to exponential decay in the Ewald sum (which is not much worse then the Gaussian decay for the unmodified sum). The summand of the real part of the sum would then look like:

$$\frac{e^{-\frac{r_{\mathbf{n}}}{a}}}{2} + \frac{-1 + e^{-\frac{r_{\mathbf{n}}}{a}} + \mathrm{Erf}[\frac{r_{\mathbf{n}}}{\alpha}]}{r_{\mathbf{n}}} \tag{20}$$

The interaction itself would also have to be modified because particles are no longer point charges. For longer distances one could use a local multipole expansion of the field at the center of the particle. Cutting off below quadrupole forces one would additionally get a small perturbation to the potential of the particle where the pseudoparticles' charge interacts with the particles' charge-variation by a quadrupole field.

*Dynamic timestepping*

A further simple way to improve the performance of PEPC based on the fact that as the interaction lists are built one gets the minimum distance between two particles for free. This minimum distance dominates the maximum timestep which could be varied accordingly, resulting in better performance for relaxed states and better accuracy for clustered ones.

*Full Maxwell codes*

Nature does not have to face the $\mathcal{O}(N^2)$ of the N-body problem at all. As Maxwell's equation tell us particles act as sources of electric and magnetic fields which propagate in space by themselves. The fields then produce forces once they encounter a particle and thus tell the particles how to move. This is, without further approximations, an intrinsic $\mathcal{O}(N)$ algorithm because there is no non-locality involved in the process. The price to pay is the simulation of the Maxwell fields, which is not all that hard for vacuum (the particles are not treated, as usual, by dielectric functions but on an atomic scale). It could be done by well-developed FDTD methods or even semi-analytical solutions of the vacuum equations in Fourier-space, which would even provide periodic boundary conditions for free.

In contrast to PIC codes one could avoid the hydrodynamic approximation by simulating the particles in continuous space, while using a grid the simulation of the field.

One drawback is that the finite, yet very high, speed of light has to be resolved, which leads to a fairly small timestep. The typical time that light and thus any action needs to propagate form one particle to the next can be approximated by $t_l \approx \frac{\sqrt[3]{n}}{c}$. A typical timescale for the tBHE code is the inverse of the plasma frequency which is defined as $t_p = \omega_p^{-1} = \sqrt{\frac{\varepsilon_0 m}{n}} e^{-1}$. The ration of the two time scales is then

$$\frac{t_p}{t_l} = \frac{1}{e}\sqrt{\frac{m}{\mu_0}} n^{-1/6} \approx 5.3 \cdot 10^6 n^{-1/6} \approx 10^2 \ldots 10^3 \tag{21}$$

suggesting that the increase of resolution in the time domain seems moderate especially for higher densities such as solid bodies.

In consequence this would lead to a totally new class of algorithm, that might be used to study the interaction with light and matter more accurately, as it would naturally include the reaction of the Maxwell fields to the motion of the particles. This might extent the reach of particle simulations into new fields such as nanooptics or nonlinear response to low power laser fields.

## Acknowledgements

No work is ever done in solitude. Among many others I am indebted to the Center for Applied Mathematics, especially Dr. Esser and Mrs. Schmitz for the smooth organisation of the guest student program; the “Verein der Freunde und Förderer des FZ Jülich” especially IBM for the financial support, and Dr. Gibbon for his patience and willingness to share his insight into the awesome labyrinth that is called PEPC and for the inspiring discussions. I would also like to thank my fellow guest students – that might be it; but it sure was a cool time.

## References

1. J. Barnes and P. Hut, *A hierarchical O (N log N) force-calculation algorithm*, Nature **324** (1986), 446–449.
2. P. Gibbon, *PEPC: Pretty Efficient Parallel Coulomb-solver*, Tech. report, Technical Report, Central Institute for Applied Mathematics, FZJ-ZAM-IB-2003, 2003.
3. P. Gibbon, W. Frings, and B. Mohr, *Performance analysis and visualization of the N-body tree code PEPC on massively parallel computers*, Proceedings of Parallel Computing (2005).
4. P. Gibbon and G. Sutmann, *Long-range interactions in many particle simulation*, Quantum simulations of many-body systems: from theory to algorithm. Eds. J. Grotendorst, D. Marx and A. Muramatsu. NIC-series **10** (2002), 467–506.
5. S. Pfalzner and P. Gibbon, *Many-Body Tree Methods in Physics*, Cambridge University Press, 1996.
6. ———, *Direct calculation of inverse-bremsstrahlung absorption in strongly coupled, nonlinearly driven laser plasmas*, Physical Review E **57** (1998), no. 4, 4698–4705.
7. J.K. Salmon and M.S. Warren, *Skeletons from the treecode closet*, Journal of Computational Physics **111** (1994), no. 1, 136–155.
8. MJL Sangster and M. Dixon, *Interionic potentials in alkali halides and their use in simulations of the molten salts*, Advances in Physics **25** (1976), no. 3, 247–342.
9. Karl F. Sundman, *Article Title - Mémoire sur le problème des trois corps*, Acta Mathematica **36** (1912), 105–179.
10. G. Sutmann, *Molecular Dynamics - Vision and Reality*, Computaional Nanoscience: Do It Yourself. Eds. J. Grotendorst, S. Blügel, and A. Muramatsu. NIC-series **31** (2006), 159–187.
11. MS Warren and JK Salmon, *A portable parallel particle program*, Computer Physics Communications **87** (1995), no. 1, 266–290.

# Ein Beitrag zur Entwicklung einer parallelen Version von AOFORCE des Programmpakets TURBOMOLE

Sebastian Höfener

Universität Karlsruhe
Institut für Physikalische Chemie
Lehrstuhl für Theoretische Chemie
E-mail: s.hoefener@chemie.uni-karlsruhe.de

**Zusammenfassung:** Für die Chemie sind Strukturen chemischer Verbindungen und deren Reaktivität von grundlegender Bedeutung. Die Berechnung der molekularen Hesse-Matrix bezüglich Kernverrückungen stellt eine Herangehensweise für beide Problemstellungen dar. Diese erlaubt die Berechnung von Schwingungsspektren sowie die Suche nach Übergangszuständen, die während chemischer Reaktionen auftreten. Eine effiziente Implementierung befindet sich in dem sequentiellen Modul AOFORCE des Quantenchemie-Programmpakets TURBOMOLE, welches als Ausgangspunkt verwendet wird. Nach Analyse der zeitintensiven Programmteile wurde eine Parallelisierung mittels Global Arrays vorgenommen. Der grundlegende Algorithmus wurde dabei nicht verändert. In verschiedenen Programmteilen wurde dynamisches und statisches Loadbalancing implementiert.

## Einleitung

Für die Chemie stehen die Fragen nach der Struktur einer Verbindung und ihrer chemische Reaktivität im Vordergrund. Die Theoretische Chemie beschäftigt sich unter anderem damit, diese Fragestellungen mit Hilfe von physikalischen Modellen zu untersuchen. Standardverfahren zur Bestimmung von Strukturen bestehen darin, Geometrien von Molekülen durch Berechnung der Energie in Abhängigkeit von den Kernkoordinaten zu bestimmen. Ist die Verbindung aber vollkommen unbekannt, läßt dies keinen Schluss zu, ob die richtige Struktur erhalten wurde. Für die Eigenschaften einer Verbindung ist die genaue Konnektivität der Atome entscheidend.
Detailliertere Informationen über eine Verbindung werden erhalten, wenn spektroskopische Untersuchungen durchgeführt werden. Experimentell einfach zugänglich sind in der Regel Schwingungsspektren, d.h. Infrarot- und Raman-Spektren, die als Fingerabdruck einer Verbindung genutzt werden können. Ergibt sich eine Übereinstimmung zwischen gemessenem und berechnetem Spektrum, so kann meist auf eine Übereinstimmung der Geometrie geschlossen werden.
Eine weitere fundamentale Eigenschaft stellt das Reaktionsverhalten von chemischen Verbindungen dar. Chemische Reaktionen können schematisch als Kernverrückungen aufgefasst werden, die zur Energieänderung des Systems führen. Dabei verläuft der Reaktionspfad meist energetisch so tief wie möglich. Das bedeutet, dass der Reaktionspfad zwischen den lokalen Maxima verläuft.

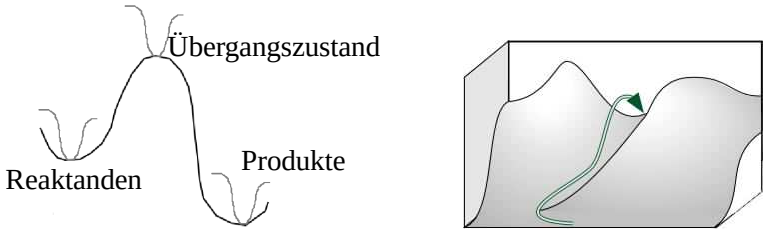



Abbildung 1: Schematische Darstellung eines Reaktionspfades in Abhängigkeit von zwei Freiheitsgraden (rechtes Bild) und dessen Projektion auf die Reaktionskoordinate (linkes Bild).

Ein Maximum auf der Projektion des Reaktionspfades auf die Reaktionskoordinate entspricht einem Punkt, an dem im Verlauf einer Reaktion das System sich in der energetisch ungünstigsten Lage befindet. Dieser Punkt wird Übergangszustand bezeichnet. Die Theorie des Übergangszustandes sagt aus, dass eine Reaktion nur stattfinden kann, wenn auf dem Reaktionsweg mindestens ein Übergangszustand erreicht wird. Mathematisch gesehen, handelt es sich dabei um einen Sattelpunkt n-ter Ordnung, zu dessen Charakterisierung die zweiten Ableitungen berechnet werden müssen.
Die Energien können mit mittlerem Kostenaufwand auf Hartree-Fock- oder DFT-Niveau behandelt werden. Sogar auf diesem unkorrelierten Level ist die Berechnung der zweiten Ableitungen signifikant teurer als die ersten Ableitungen (Gradienten), die für Geometrieoptimierungen standardmäßig berechnet werden. Dabei entsteht der Mehraufwand hauptsächlich dadurch, dass für die zweiten Ableitungen die Antwort des Systems bestimmt werden muss, da die Basisfunktionen (BF) auf den Atomen lokalisiert sind. Dadurch dass die Berechnung der zweiten Ableitungen eine äußerst zeitintensive Methode darstellt, soll dieses Verfahren parallelisiert werden.

## Theoretische Grundlagen

*Normalschwingungen in Molekülen*

Das Problem der Schwingungen in einem Molekül kann von der Rotation abgetrennt werden [1], indem ein Satz von Koordinaten verwendet wird, der sich mit dem Molekül mitbewegt und die Bedingung erfüllt, dass die Verrückungsvektoren keine Translation oder Rotation beschreiben. Die klassische Theorie liefert einen Ansatz über Hook's Gesetz für Harmonische Oszillatoren. Der eindimensionale Fall dient als modellhafte Beschreibung für die Schwingungen eines zweiatomiges Moleküls.

$$E = \frac{1}{2}kx^2 \qquad \omega = \sqrt{\frac{k}{m}}$$

Hierbei steht $E$ für die potentielle Energie, $k$ beschreibt die Kraftkonstante und $x$ die Auslenkung aus der Gleichgewichtslage. $\omega$ bezeichnet die Kreisfrequenz, $m$ die reduzierte Masse. Wird die potentielle Energie in einer Taylor-Reihe entwickelt und nach dem quadratischen Term abgebrochen, ergibt sich folgender Ausdruck, falls die Energie relativ zur Energie in der Gleichgewichtslage ausgedrückt wird.

$$E \approx \frac{1}{2}\frac{\partial^2 E}{\partial x^2}x^2$$

Ein Vergleich mit Hook's Gesetz zeigt den Zusammenhang zwischen den zweiten Ableitungen und den Frequenzen, über die die Berechnung von Schwingungsspektren zugänglich sind. Für den eindimensionalen Fall ergibt dieser:

$$\omega^2 m = k = \frac{\partial^2 E}{\partial R^2} \equiv \mathrm{H}$$

Für Systeme mit mehreren Freiheitsgraden kann der Satz an Gleichungen in Matrixschreibweise zusammengefasst werden:

$$\mathbf{\Omega}^2 = \mathbf{m}^{-\frac{1}{2}}\mathbf{k}\mathbf{m}^{-\frac{1}{2}} = \mathbf{m}^{-\frac{1}{2}}\mathbf{H}\mathbf{m}^{-\frac{1}{2}}$$

Dies entspricht einer Transformation von **H** in massengewichtete Normalkoordinaten mit Normalmoden als Eigenvektoren. Moden werden dann als Normalschwingungen bezeichnet, wenn alle Atome mit derselben Frequenz schwingen. Somit erreichen alle Atome gleichzeitig maximale Auslenkung und die Gleichgewichtslage. Die Amplituden stimmen im allgemeinen nicht überein.
Ein System aus $N$ Atomen besitzt aufgrund von Translation und Rotation $f = 3N - 5$ Schwingungsfreiheitsgrade, falls es linear ist, ansonsten gilt $f = 3N - 6$, so dass fünf bzw. sechs Eigenwerte der Hesse-Matrix null sind. Werden die Eigenwerte eines Übergangszustandes berechnet, liegt mindestens ein negativer Eigenwert vor, da ein Sattelpunkt vorliegt. Dies erlaubt die Identifizierung von Übergangszuständen.

*Hartree-Fock- und Kohn-Sham-Theorie*

In der Theoretischen Chemie werden Eigenschaften von Atomen und Molekülen mit quantenmechanischen Methoden untersucht. Eine Herangehensweise kann über Wellenfunktionen erfolgen. Dabei wird die elektronische Vielteilchenwellenfunktion als Slaterdeterminante angesetzt. Diese verknüpft Molekülorbitale (MO) so, dass die Gesamtwellenfunktion das Pauli-Prinzip nicht verletzt. Minimierung des Energieerwartungswertes mit Hilfe des Variationsprinzips unter der Nebenbedingung orthonormierter Orbitale ergibt die Hartree-Fock Methode [2]. Die MO können als Linearkombination von Basisfunktionen angesetzt werden. Sind diese auf den Atomen zentriert, werden sie als Atomorbitalen (AO), der gesamte Ansatz als LCAO Methode (*linear combination of atomic orbitals*) bezeichnet. Der LCAO Ansatz führt zu den Roothaan-Hall-Gleichungen [3], die in Matrixform notiert werden können:

$$\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon}$$

**C** enthält spaltenweise die Koeffizienten der Basisfunktionen zu einem MO, durch die die Orbitale genähert werden. **S** ist die Überlappungsmatrix, $\boldsymbol{\varepsilon}$ eine Diagonalmatrix mit den Orbitalenergien als Einträgen. Die Fockmatrix **F** hängt selber von den Koeffizienten $C_{\mu i}$ ab, die bestimmt werden müssen. In der Basis der Atomorbitale hat diese folgende Form:

$$F_{\mu\nu} = h_{\mu\nu} + \sum_{\kappa\lambda}\left[(\mu\nu|\kappa\lambda) - \frac{1}{2}(\mu\kappa|\nu\lambda)\right]D_{\kappa\lambda}$$

**h** bezeichnet den Einelektronenbeitrag, $(\mu\nu|\kappa\lambda)$ sind die Zweielektronenvierzentrenintegrale in *charge cloud* Notation über dem Operator $r_{12}^{-1}$. Griechische Buchstaben deuten Größen in der AO Basis an, i,j,... bezeichnen besetzte, a,b,... unbesetzte und p,q,... beliebige Molekülorbitale. **D** ist die Dichtematrix, die für ein geschlossenschaliges System definiert ist als:

$$D_{\mu\nu} = 2\sum_{i} C_{\mu i}C_{\nu i}$$

In der Dichtefunktionaltheorie (DFT) wird das Problem, eine Vielteilchenwellenfunktion zu finden, auf das Problem abgebildet, die Elektronendichte zu bestimmen. Diese hängt nur noch von den drei Raumkoordinaten und nicht mehr von $3N$ Koordinaten ab. Dieser Ansatz führt zu den Kohn-Sham-Gleichungen. Pragmatisch gesehen unterscheiden sich diese nur im zusätzlich auftauchenden Funktional (und je nach Methode im Austausch) von der Hartree-Fock-Methode. Die resultierende Energiebeziehung für geschlossenschalige Systeme lautet:

$$E_{\text{DFT}} = \sum_{\mu\nu} D_{\mu\nu}h_{\mu\nu} + \frac{1}{2}\sum_{\mu\nu\kappa\lambda} D_{\mu\nu}D_{\kappa\lambda}\left[(\mu\nu|\kappa\lambda) - c_x(\mu\kappa|\nu\lambda)\right] + E_{XC} + V_{NN}$$

Der Energieausdruck für die oben angesprochene Hartree-Fock-Methode unterscheidet von diesem in zwei Punkten: Zum einen ist dort kein Austauschkorrelationsbeitrag $E_{XC}$ vorhanden, zum weiteren gilt stets $c_X = 1$.
Von Kernkoordinaten abhängig sind die Operatoren für die Kern-Kern-Abstossung und die Elektron-Kern-Anziehung, sowie alle Atomorbitale. Zweifaches Differenzieren ergibt folgenden Ausdruck für die Einträge der Hesse-Matrix:

$$\begin{aligned} E^{\chi\xi} &= \sum_{\mu\nu}(D_{\mu\nu}h^{\chi\xi}_{\mu\nu} - W_{\mu\nu}S^{\chi\xi}_{\mu\nu}) + V^{\chi\xi}_{NN} + E^{(\chi)(\xi)}_{XC} \\ &+\frac{1}{2}\sum_{\mu\nu\kappa\lambda} D_{\mu\nu}D_{\kappa\lambda}\left[(\mu\nu|\kappa\lambda)^{\chi\xi} - \frac{1}{2}c_x(\mu\kappa|\nu\lambda)^{\chi\xi}\right] \\ &+2\sum_{ij}\left[F^{(\chi)}_{ij}S^{(\xi)}_{ij} + F^{(\xi)}_{ij}S^{(\chi)}_{ij} - 2\epsilon_i S^{(\chi)}_{ij}S^{(\xi)}_{ij} - 2G_{ij}[S^{(\chi)}_{kl}]S^{(\xi)}_{ij}\right] \\ &+4\sum_{ai}\left[F^{(\chi)}_{ai} - \epsilon_i S^{(\chi)}_{ai} - 2G_{ai}[S^{(\chi)}_{jk}]\right]U^{\xi}_{ai} \end{aligned}$$

Die Matrizen $\mathbf{G}[M]$ für beliebige $M$ haben folgende Form:

$$G_{pq}[M^{\chi}_{rs}] = \sum_{rs\mu\nu\kappa\lambda} C_{\mu p}C_{\nu q}C_{\kappa r}C_{\lambda s}\left[(\mu\nu|\kappa\lambda) - \frac{1}{2}c_X(\mu\kappa|\nu\lambda) + f_{XC\mu\nu\kappa\lambda}\right]M^{\chi}_{rs}$$

Für die Größen aus der Dichtefunktionaltheorie $f_{XC\mu\nu\kappa\lambda}$, $V^{(\chi)}_{XC\mu\nu}$ sowie $E^{(\chi)(\xi)}_{XC}$ sind keine Formeln angeben, da sich diese für verschiedene Ansätze (LDA, GGA, Hybridfunktionale) unterscheiden.
Durch Verwendung atomzentrierten Basisfunktionen verändert sich die Wellenfunktion durch die Verrückung der Kerne. Dadurch muss die Antwort des Systems, d.h. die veränderten Koeffizienten berechnet werden. Für diese wird folgender Ansatz verwendet:

$$C^{\chi}_{\mu i} = \sum_{q} C_{\mu q}U^{\chi}_{qi}$$

**U** transformiert die ungestörte Wellenfunktion in die gestörte. Die Vektoren dieser Matrix werden daher als Orbtialrotationen bezeichnet. Allgemein bezeichnen die oberen griechischen Indizes partielle Ableitungen nach den Kernkoordinaten $\chi$ und $\xi$ jeweils in alle drei Raumrichtungen. Enthält die betrachtete Größe keine MO-Koeffizienten, wird keine Klammerung verwendet. Sind MO-Koeffizienten enthalten, deuten Klammern die Verwendung der gestörten Koeffizienten an, keine Klammern bedeuten die Verwendung der ursprünglichen Koeffizienten. Die Rotationen zwischen besetzten und besetzten Orbitalen sind proportional zur Ableitung der Überlappungsmatrix $S^{(\chi)}_{ij}$.

$$U^{\chi}_{ij} = -\frac{1}{2}S^{(\chi)}_{ij}$$

Im Gegensatz dazu werden keine Rotationen innerhalb der virtuellen Orbitale vorgenommen, so dass dieser Block von **U** gleich null ist. Der Block, der zwischen besetzten und unbesetzten Orbitalen transformiert, wird aus der Lösung der *f coupled perturbed Kohn-Sham* Gleichungen erhalten. Diese folgen aus der Notwendigkeit, dass die Energie unter Orbitalrotation konstant sein muss und können wie folgt formuliert werden:

$$(\varepsilon_i - \varepsilon_a)U^{\chi}_{ai} - 4G_{ai}\left[U^{\chi}_{bj}\right] = F^{(\chi)}_{ai} - \varepsilon_i S^{(\chi)}_{ai} - 2G_{ai}\left[S^{(\chi)}_{jk}\right]$$

**Zeitbestimmende Schritte**

*Numerische Integration bei Dichtefunktionalmethoden*

Allgemein stellt bei Dichtefunktionalmethoden die numerische Integration mittels Quadraturalgorithmus einen zeitintensiven Abschnitt dar. Da dies aber kein spezielles Problem bei der Berechnung der zweiten Ableitungen ist, findet im Rahmen dieser Arbeit keine nähere Untersuchung statt. Um letztendlich eine effektive Parallelisierung entwickeln zu können, muss dies berücksichtigt werden. In engem Zusammenhang damit steht die Ableitung des Austauschkorrelationsbeitrages $E_{XC}^{(\chi)(\xi)}$, sowie der ersten Ableitungen in $V_{XC\mu\nu}^{(\chi)}$. Hierbei handelt es sich formal um $\mathcal{O}(N^5)$ Schritte. Da Beiträge nur dann nicht verschwinden, wenn die Gitterpunkte und Basisfunktionen räumlich benachbart sind, beträgt die asymptotische Skalierung für große Moleküle $\mathcal{O}(N)$.

*Berechnung der Ableitungen der Zweielektronenintegrale*

Einen weiteren nicht vernachlässigbaren Beitrag stellen die Berechnung der Beiträge der Zweielektronenintegrale $(\mu\nu|\kappa\lambda)^{\chi}$ und $(\mu\nu|\kappa\lambda)^{\chi\xi}$ dar. Diese formen dünnbesetzte fünf- beziehungsweise sechsfach indizierte Größen, da nicht-verschwindende Beiträge nur dann auftreten, wenn $\chi$ (respektive $\xi$) zu solchen Kernen gehören, die in dem entsprechenden Integral enthalten sind. Dadurch entspricht das Skalierungsverhalten dem der Zweielektronenintegrale. Dieses beträgt formal $\mathcal{O}(N^4)$ und aufgrund von Integralabschätzungen $\mathcal{O}(N^2)$.

*Lösung der CPKS-Gleichungen*

Die zeitintensivsten Schritte stellen die Berechnung der Größen $G_{pq}[M_{rs}^{\chi}]$ dar. Hierbei handelt es sich um $\mathcal{O}(N^5)$-Schritte, da die Matrizen in der Regel nicht dünn besetzt sind. Zur iterativen Lösung der CPKS-Gleichung müssen solche Berechnungen für jede irreduzible Darstellung mehrfach ausgeführt werden. Um den benötigten Speicherplatz zu reduzieren, der in der AO Basis mit $\mathcal{O}(N^4)$ skaliert, werden im sequentiellen AOFORCE die Matrizen in der MO Basis abgespeichert. Die Transformation skaliert mit $\mathcal{O}(N^4)$, aber aufgrund von sehr schnellen Matrixalgebraroutinen bedeutet dies nur einen kleinen Vorfaktor. Durch die ständig wiederholte Berechnung zur Lösung der CPKS-Gleichungen stellen diese den zeitintensivsten Abschnitt bei der Berechnung der zweiten Ableitungen dar.

*Zeitverhältnisse bei verschiedenen Methoden*

Werden Rechnungen mit partieller RI-$J$-Näherung verwendet [5], sinkt die benötigte Rechenzeit für die CPKS-Gleichungen stark ab. Daher ist die Methode, mit der eine Berechnung durchgeführt wird, entscheidend für den Ansatz der Parallelisierung. Werden die zweiten Ableitungen auf HF-Niveau ohne RI-$J$-Näherung berechnet, dann stellt die Parallelisierung von $G_{pq}[M_{rs}^{\chi}]$ eine erhebliche Verbesserung des Skalierungsverhaltens dar. Bei Verwendung anderen Optionen, etwa die Berechnung ausschliesslich der niedrigsten Eigenwerte, ist dies nicht mehr gegeben. Daher müssen langfristig die Berechnung aller beitragenden Terme parallelisiert werden.

**Implementierung**

*Strategie*

TURBOMOLE wurde als Programm für Workstation-Computer entworfen, mit dem Moleküle mit einem breiten Spektrum an schnellen Methoden berechnet werden können. Auf diesen konventionellen

Linux-Plattformen, wie etwa Cluster-Systemen, muss damit gerechnet werden, dass der Hauptspeicher eines Prozessors gegebenenfalls nicht ausreicht, um alle benötigten Daten im Hauptspeicher ablegen zu können. Daher muss dem Programm bei Verwendung bestimmter Methoden die maximale Größe des zur Verfügung stehenden Speichers angegeben werden. Alle Daten, die dieses vorgegebene Limit überschreiten, werden auf der Festplatte abgespeichert. Im Zuge dieser Strategie werden vergleichbar viele Algorithmen in Blöcken abgearbeitet. Dabei wird ein Datenblock von der Festplatte gelesen, bearbeitet und danach wieder auf der Festplatte gespeichert, so dass auch größere Moleküle untersucht werden können. Aufgrund der lokalen Dateistruktur auf Cluster-Systemen stellt dies eine angemessene Lösung dar. Auf Großrechnern, wie z.B. dem IBM p690 JUMP des Forschungszentrums Jülich, die ein nichtlokales Dateisystem haben, beeinträchtigt diese Vorgehensweise die Laufzeit massiv. Daher müssen die zahlreichen Dateizugriffe entfernt und alle Daten, die in den zeitkritischen Schritten bearbeitet werden, komplett im Hauptspeicher vorliegen.
Das Programmpaket TURBOMOLE wird fortlaufend von mehreren Programmierern unabhängig voneinander weiterentwickelt. Dies stellt Anforderungen an die parallele Implementierung: Sie muss gewährleisten, dass der Algorithmus durch Kommunikationsschemata nicht unverständlich wird, so dass der Code die Flexibilität besitzt, erweitert werden zu können. Im Rahmen der ersten Parallelisierung soll der bestehende Algorithmus nicht verändert werden. Dadurch können Fehler leichter gefunden, und ein direkter Geschwindigkeitsvergleich durchgeführt werden.
Für die Parallelisierung wurde das Global Array Toolkit verwendet, welches für Anwendungen in Quantenchemie-Programmen entwickelt wurde.

*Das Global Array Toolkit*

Das Global Array Toolkit wurde am Pacific Northwest National Laboratory (PNNL) in den 1990'er Jahren entwickelt. Es stellt ein effizientes und portables *shared memory* Programmierinterface für Systeme mit verteiltem Speicher zur Verfügung. Dabei kann jeder Prozessor asynchron auf logische Blöcke von Daten (Global Arrays) zugreifen, ohne dass explizit mit anderen Prozessen kommuniziert wird. Dies wird als nullseitigen Kommunikation bezeichnet.
GA können im allgemeinen verwendet werden, falls eine dynamische und unregelmäßige Kommunikationsstruktur vorliegt, die mit dem Message-Passing-Interface (MPI) nur schwer realisiert werden kann. Für das gleichmäßige Auslasten der Prozessoren (Loadbalancing) sind GA dann besonders gut zu verwenden, falls eine dynamische Aufgabenverteilung gewählt wird, d.h. die Aufgabenverteilung wird während der Laufzeit des Programms festgelegt. GA können zusammen mit MPI verwendet werden, so dass dessen Funktionalität erweitert wird.
Wird ein GA erstellt, wird dem Benutzer der Offset der Speicheradresse übergeben. Anhand der Funktion `dbl_mb(wert)` kann gezielt auf einen bestimmten Eintrag zugegriffen werden, wenn es sich bei den Einträgen um Fliesskommazahlen der Genauigkeit `double precision` handelt. So bezeichnet etwa `dbl_mb(wert+1)` die Adresse des zweiten Eintrages einer Matrix, die einer Unterroutine übergeben werden kann.

*Das* TURBOMOLE*-Interface* `zam.parlib`

Im Rahmen der Parallelisierung von TURBOMOLE beschäftigt sich Thomas Müller am Zentrum für Angewandte Mathematik (ZAM) des Forschungszentrums Jülich unter anderem mit der Programmierung eines Interfaces zwischen TURBOMOLE und dem Global Array Toolkit, MPI und ScaLAPACK. Damit soll vermieden werden, dass aufgrund von Änderungen an den parallelen Bibliotheken der TURBOMOLE-Sourcecode umgeschrieben werden muss. Eventuelle Weiterentwicklungen der GA, MPI oder ScaLAPACK müssen dann lediglich innerhalb des Interface-Moduls `zam.parlib` berücksichtigt werden. Die Befehle beginnen ausschliesslich mit dem Präfix '`par_`'. Zusätzlich zu den Befehlen der parallelen

Bibliotheken werden Dienste bereitgestellt, zum Beispiel zur Erzeugung (`par_cntinit`) und Verwaltung von Aufgabenlisten (`par_nextval`).

*Speicherverwaltung und Kommunikation*

Besonders wichtig im Rahmen von parallelen Programmen steht die Abwägung von maximaler Datenlokalität und Kommunkationsaufwand. Je mehr Daten repliziert, d.h. auf jedem Prozessor identisch, gespeichert werden (replicated data), desto weniger Kommunikation ist nötig. Andererseits wird mit dieser Strategie durch die Speicherkapazität pro Prozessor die maximale Größe der Rechnung massiv beeinflusst. Durch Wahl von verteilter Datenspeicherung (distributed data) wird der insgesamt benötigte Speicherplatz reduziert, da große Datenmengen nicht mehr vielfach, sondern nur noch einmal im Speicher gehalten werden. Bei *shared memory*-Systemen wie zum Beispiel dem JUMP ist es mit Hilfe des `Loadlevelers` möglich, eine gewisse Anzahl Prozessoren zu reservieren, von denen nur ein Teil zur Rechnung zugelassen werden. Damit erhöht sich die Größe des verfügbaren Speichers pro Prozessor. In diesem Fall können bei verteilten Daten mehr Prozessoren an der Rechnung mitwirken, als wenn alle Daten repliziert gehalten werden. Aufgrund dessen sollen möglichst viele große Matrizen als GA in den verteilten Speicher gelegt werden. Dies ist zunächst nur innerhalb eines *shared memory*-Systems verwirklicht worden. Auf dem JUMP handelt es sich dabei um sogenannte Knoten mit jeweils 32 Prozessoren. Datenverteilung über mindestens zwei Knoten wurde im Rahmen dieser Arbeit nicht implementiert.

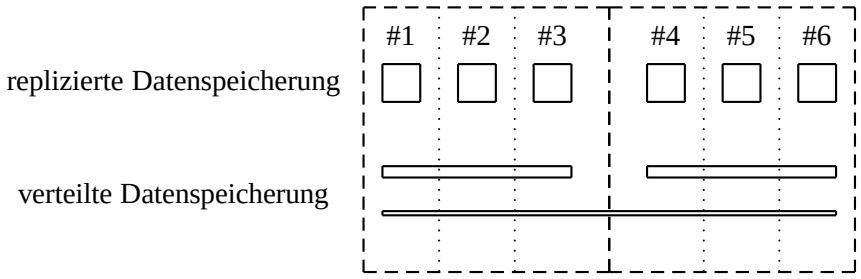



Abbildung 2: Schematische Darstellung von replizierten und verteilen Daten. Gepunktete Linien deuten zusammen mit der Nummerierung unterschiedliche Prozessoren innerhalb eines *shared memory* Systems an, gestrichelte Linien grenzen *shared memory* Systeme voneinander ab. Die Höhe der Kästen deutet symbolisch den benötigten Speicherplatz pro Prozessor an.

Im Rahmen der Speicherung von Daten muss ferner zwischen Kommunikation, Datengröße und Berechnung der Daten abgewogen werden. Je mehr Prozessoren verwendet werden, desto mehr Daten werden redundant, d.h. von jedem Prozessor gleich berechnet. Benötigt die Berechnung bestimmter Daten viel Rechenzeit, so könnte eine mögliche Parallelisierung zunächst darin bestehen, die Daten nicht-redundant parallel zu berechnen und dann im verteilten Hauptspeicher abzulegen. Damit diese Variante eine effektive Lösung darstellt, müssen zwei Aspekte gewährleistet sein: Zum einen darf die Gesamtlaufzeit durch die nötige Kommunikation, um auf die Daten zu zugreifen, nicht ansteigen. Zum anderen dürfen die Daten nicht zu viel Speicherplatz beanspruchen, so dass den Daten, die gespeichert werden müssen, genügend Speicher zur Verfügung steht. Für sehr große Datenmengen, bei denen die einzelnen Berechnungen vergleichsweise kurz sind, wird eine redundante Berechnung im Sinne eines direkten Algorithmus daher immer wichtiger.

*Angaben zu Änderungen im gesamten Programm*

Um im ersten Schritt eine Programmversion zu erzeugen, die prinzipiell von mehreren Prozessoren parallel ausgeführt werden kann, mussten die Namen aller während des Programmablaufs angelegten Dateien um die Prozessornummer erweitert werden, damit die Datenkonsistenz gewährleistet blieb. Weiterhin

wurden die neu erstellten Global Arrays in sämtliche relevante Übergabelisten eingefügt, da die Abfrage dieser Informationen zu viel Zeit in Anspruch nehmen würde.

*Erste und zweite Ableitungen der Einelektronenbeiträge*

Die Parallelisierung wurde im wesentlichen in den Routinen `stvfd` und `dstvf` vorgenommen. Die Einelektronenbeiträge der ersten Ableitungen beeinträchtigen aufgrund ihres Skalierungsverhaltens die Gesamtlaufzeit nur in vernachlässigbarer Größenordnung. Hauptgründe für die Parallelisierung dieser Routinen stellt der Dateizugriff auf die Datei `'dh'` an dieser Stelle und in der Routine, die die abgeleiteten Fockmatizen konstruiert, dar. Die Modifikationen bestanden zum Großteil im Entfernen aller Dateizugriffe und der Ablage der Daten im verteilen Speicher. Die Berechnung erfolgt nur durch Prozessor Null, der die Ergebnisse auf das GA schreibt. Dies geschieht in der Routine `dstvi` der Integralbibliothek (`intlib`). Für größere Systeme muss auch für die ersten Ableitungen eine echte Parallelisierung entwickelt werden. Durch die Vorarbeit ist dies nun direkt zugänglich, so dass an dieser Stelle nur die eigentliche Aufgabenverteilung implementiert werden muss.

*Berechnung Zweielektronenbeiträge der ersten Ableitungen*

Die erste Ableitung der Fockmatrix setzt sich aus folgenden Beiträgen zusammen:

$$F_{\mu\nu}^{(\xi)} = D_{\mu\nu} h_{\mu\nu}^{\xi} + \frac{1}{2} \sum_{\kappa\lambda} D_{\mu\nu} D_{\kappa\lambda} \left[ (\mu\nu|\kappa\lambda)^{\xi} - (\mu\kappa|\nu\lambda)^{\xi} \right] + V_{XC\mu\nu}^{\xi} - W_{\mu\nu} S_{\mu\nu}^{\xi}$$

Die zeitintensiven Schritte dieser Beiträge stellen die abgeleiteten Zweielektronenintegrale $(\mu\nu|\kappa\lambda)^{\xi}$ dar, so dass die Berechnung dieser Beiträge in der Routine `mkgxi_d` parallelisiert wurde. Die Speicherung der abgeleiteten Fockmatrizen in der Datei `dF` wurde durch ein Ablegen in GA ersetzt. Diese Größe hat einen Speicherbedarf von $3N_{\text{Atome}} \cdot N_{\text{BF}}^2$.
Die Implementierung wurde so gestaltet, dass die Datenstruktur und der Algorithmus erhalten bleibt. Die Aufgabenverteilung erfolgt im Rahmen der ersten Parallelisierung statisch und mindestens atomweise, so dass pro Prozessor $3m$ Ableitungsmatrizen berechnet werden, wobei $m$ die Anzahl der Atome darstellt. Liegen mehr Atome als Prozessoren vor, erhält jeder Prozessor eine Aufgabe. Liegen weniger Atome als Prozessoren vor, erhalten nur die ersten Prozessoren bis zur Anzahl der Atome eine Aufgabe. Dies macht deutlich, dass diese Art der Parallelisierung für größere Atomverbände mit kleineren Basissätzen eine deutlich bessere Skalierbarkeit aufweist. Im Rahmen einer Weiterentwicklung der Parallelisierung muss hier angesetzt werden. In einem nächsten Schritt kann etwa die atomweise Berechnung auf einzelne Fock-Matrizen verkleinert werden. Für sehr kleine Moleküle mit großen Basissätzen kann aber auch mit dieser Strategie keine effektive Skalierung erreicht werden. In diesem Fall muss die Aufgabenverteilung noch weiter verkleinert werden, so dass jeder Prozessor nur Teile von Ableitungsmatrizen berechnet.
Die abgeleiteten Fockmatrizen werden in der Unterroutine zunächst in der AO Basis auf dem GA gespeichert, dann in der Routine `nfo2nca` in die CAO Basis transformiert und erneut auf dem GA abgelegt. Die Dimensionen der Fockmatrizen für die beiden Basen unterscheiden sich, da in der CAO-Basis im Gegensatz zur AO Basis lineare Abhängigkeiten auftreten. Um die Daten möglichst leicht transformieren zu können, wurde ein eindimensionales Array als Speicherformat gewählt.

Erzeugung der $k$ Aufgaben, kleinste Einheit: ein Atom $\mathbf{F}^{(A_x)}$, $\mathbf{F}^{(A_y)}$, $\mathbf{F}^{(A_z)}$
Falls `par_me` $\leq k - 1$
    Berechnung von $(\mu\nu|\kappa\lambda)^{\xi}$ in `dshloop`
    Transformation in die CAO-Basis
    Berechnung der DFT-Beiträge
    Speichern auf GA

Synchronisieren
Blockweises Konstruieren der Beiträge für CPKS-Gleichungen und Hesse-Matrix

Abbildung 3: Dargestellt ist der schematische Programmverlauf in der Unterroutine `mkgxi_d`.

Beim Einlesen des Moleküls erfolgt eine automatische Sortierung der Basisfunktionen nach dem Bahndrehimpuls. Aufgrund dessen erfolgt die Berechnung der Integrale innerhalb von `dshloop` ebenfalls nach l-Quantenzahl. Dies wurde dahingehend geändert, dass die Ordnung nach Atomen erfolgt.
Die blockweise Verarbeitung der Daten zu Fockmatrixbeiträgen zu den CPKS-Gleichungen und der Hesse-Matrix stammt aus der Originalversion und muss langfristig ebenfalls parallelisiert werden.

*Lösung der CPKS-Gleichungen*

Die Organisation der Lösung der CPKS-Gleichungen wird von der Routine `solve_cphf` übernommen. Der erste Schritt bestand erneut darin, alle Dateizugriffe zu entfernen und durch Datenspeicherung in Global Arrays zu ersetzen. Hierbei handelte es sich um die Dateien '`vfile_IRREP`' sowie '`wfile_IRREP`'. Das Suffix `IRREP` bezeichnet dabei die irreduziblen Darstellungen des Moleküls.
Die Routine `solve_cphs` ruft die Routine `respon` in der Bibliothek `escflib` auf. Diese berechnet iterativ eine angegebene Zahl niedrigster Eigenpaare einer symmetrischen Matrix nach dem blockweisen Davidson-Algorithmus [8], mit dem auch das volle Eigenwertproblem gelöst werden kann. Dieser Zyklus benötigt die Berechnung eines Matrix-Vektorproduktes, welches in der Routine `mvproduct` berechnet wird. Innerhalb dieser Routine geschehen die Transformationen und die Berechnung der Matrizen $\mathbf{G}[M]$.

Erzeugung von $k$ Aufgaben
Bis keine Augabe mehr vorhanden
    Transformation MO→CAO (`tramocao`)
    Berechnung der Zweielektronenintegrale (`shloop`)
    Transformation CAO→MO (`tracaomo`)
Synchronisieren
Blockweise Konstruktion der Beiträge für CPKS-Gleichungen und Hesse-Matrix

Abbildung 4: Schematische Darstellung des Programmverlaufs mit dynamischer Aufgabenverteilung in der Unterroutine `mvproduct`.

Die Berechnung der **G**-Matrizen wurde über dynamisches Loadbalancing parallelisiert. Die Aufgabenverteilung erfolgt wie oben erläutert in der Weise, dass mehr Aufgaben erzeugt werden, als Prozessoren vorhanden sind und die Informationen über den Bearbeitungsstatus global gespeichert werden. Jeder Prozessor erhält so lange Aufgaben, bis alle verteilt wurden. Zu einer Aufgabe gehören neben der Berechnung der Zweielektronenintegralen die beiden Transformationsschritte und weitere Schritte des Davidson-Algorithmus. Die Tatsache, dass an dieser Stelle dynamisches, bei der Berechnung der ersten Ableitungen der Zweielektronenintegrale hingegen statisches Loadbalancing verwendet wurde, geschah lediglich zum Kennenlernen der verschiedenen Methoden. Allgemein stellt ein statisches Loadbalancing die bessere Methode dar, da somit direkt feststeht, welcher Prozessor mit welchen Daten arbeiten muss, so dass sich der Kommunikationsaufwand verringert.
Die Erzeugung der Aufgabenliste ist derzeit noch nicht an die parallele Programmversion adaptiert. Aufgrund des Algorithmus muss eine Mindestaufgabengröße bearbeitet werden, so dass die Skalierbarkeit eingeschränkt wird. Gerade in dem Programmteil, der die meiste Rechenzeit benötigt, muss eine nahezu ideale Skalierung erreicht werden. Dazu müssen entsprechend viele Aufgaben erzeugt werden können.
Wie weiter oben erläutert, befinden sich in TURBOMOLE vergleichbar viele Prozeduren, die Daten blockweise verarbeiten. Da im Rahmen von *shared memory*-Systemen mit nicht-lokaler Dateistruktur die

blockweise Abarbeitung von Daten die Geschwindigkeit deutlich senkt, wurden teilweise Routinen neu geschrieben. Ein Beispiel für eine Routine, die komplett neu geschrieben wurde, stellt die Orthonormierung von Vektoren nach dem Modifizierten Gram-Schmidt-Verfahren dar.

*Modifiziertes Gram-Schmidt-Verfahren unter Verwendung von GA*

Die Unterroutine `orth`, die eine blockweise Orthonormierung durchführt, wurde durch die Routine `p_mga` ersetzt. Hierbei wird pro Prozessor ein bestimmter Abschnitt aller Vektoren aus dem GA lokal gespeichert und auf diesem die Orthonormierung durchgeführt. Kommunikation ist nur für die Normierung des Vektors nötig.

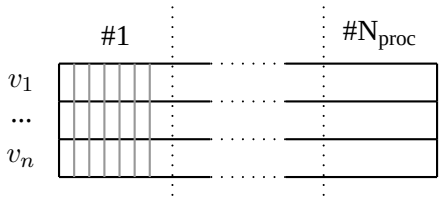



Abbildung 5: Die durchgezogenen horizontalen Linien deuten die schnellen Indizes über die gleichen Einträge in den verschiedenen Vektoren an.

Die Vektoren werden transponiert abgespeichert, so dass der schnelle Index jeweils über den i-ten Eintrag aller Vektorabschnitte läuft. Der langsame Index stellt die verschiedenen Vektoren dar. Lineare abhängige Vektoren werden während der Orthonormalisierung entfernt. Hierzu werden alle folgenden Vektoren um eine Spalte nach vorne kopiert.
Diese Methode ist nur rentabel, wenn auf jedem Prozessor die Vektorabschnitte eine gewisse Mindestgröße besitzen. Im Extremfall könnte pro Prozessor nur ein Eintrag vorliegen. Da die Kommunikation deutlich langsamer ist als die Verarbeitung der Daten auf einem Prozessor, würde die Effizient stark sinken. Dies hat zur Folge, dass im Rahmen der Orthonormierung empirisch eine Idealgröße gefunden werden muss. Alle restlichen Prozessoren bleiben in Warteposition.

*Auftretende Probleme*

Neben Fehlern, die bei jeder Programmierarbeit auftreten können, kann es bei der Parallelisierung von Programmen zu Fehlern kommen, die für sequentielle Programme nicht auftreten können. Ein Beispiel hierfür stellt die Eigenvektorberechnung von Matrizen dar. Die Eigenwerte einer Matrix sind zum Beispiel über das charakteristische Polynom eindeutig festgelegt. Zu jedem Eigenwert gehört aber ein ganzer Eigenraum an Eigenvektoren. Somit kann es passieren, dass aufgrund von numerischen Instabilitäten zu einer Matrix, bei der die Einträge bis auf zehn Nachkommastellen identisch sind, verschiedene Prozessoren zu einem Eigenwert mathematisch korrekt verschiedene Eigenvektoren berechnen. Dies kann im weiteren Verlauf des Programms dazu führen, dass bei iterativen Verfahren eine unterschiedliche Anzahl von Zyklen benötigt wird. Befindet sich innerhalb dieser Zyklen mindestens ein Synchronisationsschritt, so warten manche Prozessoren aufgrund des zusätzlichen Zyklus an einer Stelle im Programm auf Prozessoren, welche die Iteration schon verlassen haben. Somit ist es notwendig, die Eigenvektoren nur von einem Prozess berechnen zu lassen, und die erhaltene Lösung mittels eines sogenannten *broadcasts* an alle anderen Prozessoren zu verteilen.

**Diskussion und Ausblick**

Um die erste Version der Parallelisierung abzuschliessen, ist es notwendig, zunächst die Beiträge der zweiten Ableitungen der Zweielektronenvierzentrenintegrale $(\mu\nu|\kappa\lambda)^{\xi\chi}$ parallel berechnen zu lassen.

Dies konnte aufgrund von Zeitmangel noch nicht implementiert werden. Ist dies geschehen, muss anhand von Beispielen die Skalierung überprüft werden. Hierbei ist es absolut notwendig, alle möglichen Kombinationen der Programmoptionen miteinander zu testen. Dabei sind zu nennen als grundlegende Theorien Hartree-Fock und DFT, beide mit und ohne RI-$J$-Näherung. Ferner müssen kleine Moleküle mit großen Basissätzen mit großen Molekülen mit relativ kleinen Basissätzen verglichen werden. Ferner muss verglichen werden, inwiefern sich die Skalierung unterscheidet, falls die Hesse-Matrix komplett oder nur die untersten Eigenwerte (lowest eigenvalue search, LES) bestimmter irreduzibler Darstellungen berechnet werden. Erst diese Messergebnisse geben Aufschluss darüber, in welcher Reihenfolge die Parallelisierung fortgeführt werden sollte.
Im Rahmen dieser Arbeit wurde die Grundlage gelegt für die langfristig durchzuführende Parallelisierung. Dennoch bleiben viele Stellen unberührt, von denen bekannt ist, dass sie verändert werden müssen, um eine vernünftige Skalierung für große Moleküle unter Verwendung von bis zu etwa 128 Prozessoren zu erreichen.
Langfristig muss tiefer in das Programm eingegriffen werden, als dies im Rahmen einer ersten Parallelisierung durchgeführt wird. Zum einen muss, um eine echte Skalierung zu erhalten, nahezu jeder Beitrag zu den zweiten Ableitungen parallelisiert werden. Dies geschieht jedoch stets unter dem Leitgedanken, den Algorithmus nicht zu verändern. In einer weiteren Stufe muss entschieden werden, welche Aufgaben die parallele Version schwerpunktmäßig erfüllen soll. Stehen sehr große Moleküle im Vordergrund, bei der etwa die MO/AO Transformationen der **G**-Matrizen entscheidend an Zeit benötigen, so muss der Algorithmus selber dahingehend verändert werden, dass Transformationen möglichst umgangen und nur an unvermeidbaren Stellen durchgeführt werden.

## Danksagung

Ich möchte Prof. Dr. W. Klopper der Universität Karlsruhe danken, der mich auf dieses Gaststudentenprogramm aufmerksam gemacht und die Befürwortung geschrieben hat. Ich danke Dr. T. Müller des Zentrums für Angewandte Mathematik (ZAM) für die gute Betreuung vor Ort und die vielen Diskussionen. Weiterhin erkenne ich die großzügige Vergabe von Rechenzeit auf dem JUMP des Forschungszentrum Jülichs durch das ZAM und des John von Neumann-Institus für Computing (NIC) an. Darüberhinaus bin ich dem Förderverein des Forschungszentrum Jülich dankbar, der einen Großteil der Miete finanziert hat.

## Literatur

1. E. B. Wilson, P. C. Cross, J. C. Decius,
   Molecular Vibrations, Dover, 1980
2. D. R. Hartree,
   Repts. Progr. Phs. 11 (1948) 113
3. C. C. J. Roothaan,
   Revs. Mod. Phys. 23 (1951) 69
4. P. Degelmann, F. Furche, R. Ahlrichs,
   Chem. Phys. Lett. 362 (2002) 511-512.
5. P. Degelmann, K. May, F. Furche, R. Ahlrichs,
   Chem. Phys. Lett. 384 (2004) 103-107.
6. T. Müller
   Interface to GA, persönliche Mitteilung
7. Global Arrays Toolkit
   http://www.emsl.pnl.gov/docs/global/
8. E. R. Davidson
   J. Comp. Phys. 17 (1975) 87-94.

# Load Balance and Scalability of Force Decomposition Methods in Parallel Molecular Dynamics Simulations

Florian Janoschek

Universität Stuttgart
Institute for Computational Physics

E-mail: florian@icp.uni-stuttgart.de

**Abstract:** (under construction)

## Introduction

About 50 years have passed since the first molecular dynamics method developed by B. J. Alder and T. E. Wainright was used to investigate the behavior of hard spheres. Since then, molecular dynamics has established itself as a technique applicable to the simulation of many-particle systems of various kind.

Usually the most expensive part of a molecular dynamics simulation is the computation of interactions between the particles. This is because in principle for $N$ particles, there are $N(N-1)$ interactions. If interactions are limited to a short range, this number decreases to $NN_c$, where $N_c$ is the average number of neighbors of a particle within the cutoff radius. However, since the evaluation of interaction potentials typically requires at least one expensive floating point operation like division, and also because the overhead that is necessary to keep track of each particle's neighbors is quite large, execution time is still dominated by the interactions.

Running molecular dynamics simulations in parallel allows to increase the number of particles simulated. There are three major approaches to parallelization: The most simple is *atom decomposition*. This means that every particle resides on "its" *processing element* (PE), which performs integration and calculation of forces related to it. This decomposition scheme is quite easy to implement, but involves a big disadvantage: Because in many simulated systems the particles are free to move through the whole simulation space, the search for interaction partners requires global communication of particle positions spread across all PEs.

For short-range interactions, this problem can be overcome by the second method referred to as *spatial* or *domain decomposition*. As the name implies, the simulation space is divided into one, typically cubic, cell for each PE. All necessary computations for the particles within a cell are performed by the corresponding PE. If the dimensions of a cell are larger than the cutoff radius, only particles in neighboring cells can interact. This way communication becomes local. In three dimensions, every PE has to send position information to only $3^3 - 1 = 26$ other PEs independently from the total number of PEs (and the system size).

Nevertheless, spatial decomposition methods become inefficient in two cases: If—e. g. to calculate time-correlation functions—during the simulation additional data is accumulated for every particle, all this information has to be sent to the target PE, whenever a particle crosses a cell boundary. So even if communication is local it can still be crucial because of the volume transferred.

The second case in which spatial decomposition methods encounter difficulties are inhomogenous sys-

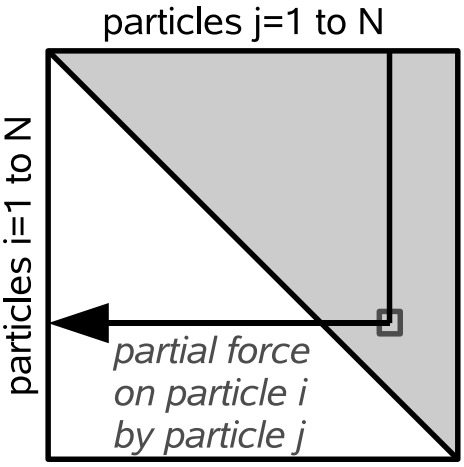



Figure 1: Interaction matrix $\mathrm{F}_{ij}$. Only upper triangular elements must be computed.

tems. When simulating systems with inhomogenous particle densities some cells can easily have a multiple of the work of most of the other cells, which obviously restricts parallel speedup to relatively small values. Different from atom decomposition methods, where balancing the amount of computations performed on every PE is as simple as adjusting the number of particles assigned to it, it is far from trivial to adapt the cell structure to local inhomogeneities and still benefit from local communication.

The subject of the following sections will be the third approach, called *force decomposition*. In this method one comes from the interactions between the simulated particles, that can be visualized as *interaction matrix* (Fig. 1). This matrix consists of the partial forces $\mathrm{F}_{ij}$ acting on particle $i$ due to interaction with particle $j$. Because of Newton's 3rd law

$$\mathrm{F}_{ij} = -\mathrm{F}_{ji} \tag{1}$$

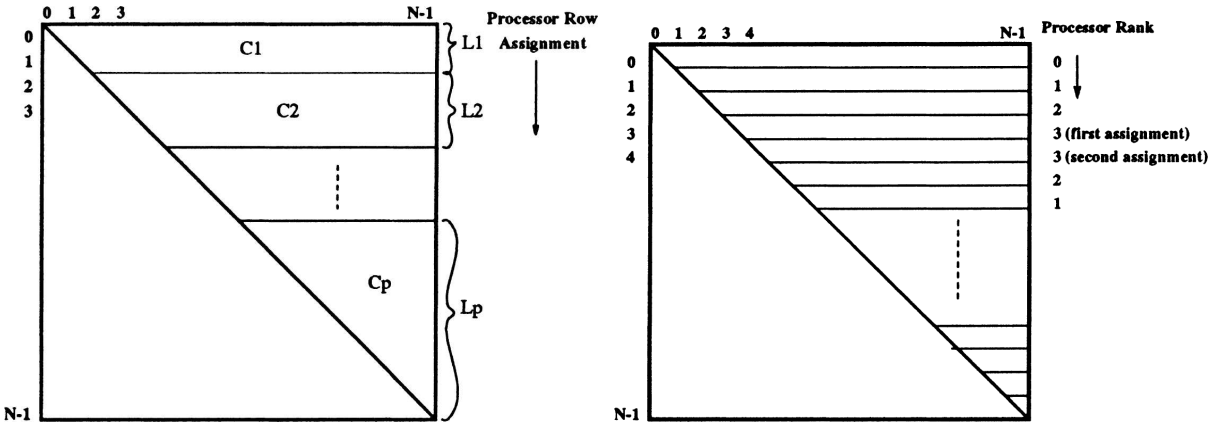
it is sufficient to determine just those interactions with $i < j$. For short-range interactions, $\mathrm{F}_{ij}$ is just sparsely populated, which allows some optimizations concerning the communication, as I will show later.

Now the idea of force decomposition is to split the interaction matrix into $P$ sections, so that every PE computes only a part of all partial forces. In the following section three different ways of distributing the force computations will be shown. Because integration takes by far less time than the computation of interactions, for not too high degrees of parallelization it does not matter whether this work is performed on all PEs redundantly or distributed between all or some PEs.

In general, there are two points in force decomposition methods, where communication can occur:

1. Before the force computation, the position of a particle that was updated in the integration step just before must be sent from the integrating PE(s) to all PEs computing interactions related to that particle.
2. After the force computation, all partial forces on a particle have to be summed up and sent to the PE(s) performing the integration for that particle.

As interaction does usually not depend on particle velocities, they do not need to be sent around. Depending on the force decomposition scheme and the strategy regarding integration, global or local com-




(a) force-stripped row method (b) force row-interleaving method

Figure 2: Both Force decomposition methods proposed by Murty et al. in [2] (figures taken from there)

munication may be required at both points. If integration is done redundantly, positions do not need to be transferred at all.

Like atom decomposition, force decomposition methods should be able to deal with problems on which spatial decomposition fails, namely inhomogenous systems, because in principle it should be possible to adjust the size of the sections of $\mathrm{F}_{ij}$ assigned to a PE like one would adjust the number of particles assigned to a PE in atom decomposition in order to gain load balance. Also, computation of correlation functions means no additional communication since the PE responsible for the integration for a special particle won't change during the simulation.

Compared to atom decomposition, force decomposition algorithms should be very competitive. Both share similar communication structures. In fact, simple force decomposition schemes like the force row-interleaving method shown below could be understood as a kind of atom decomposition. The big advantage of concentrating on the interactions instead of the particles is that load balance can be achieved in a quite intuitional way. Additionally, one could think of massively parallel force decomposition algorithms capable of running on a number of PEs that is higher than the number of particles in the simulated system, which would make no sense in atom decomposition.

**Implemented force decomposition schemes**

In this section I will introduce the three force decomposition methods that were implemented and tested. All methods were fit into the molecular dynamics software `dmmd` developed by G. Sutmann. Due to the modular design, only minor adaptions to the original software were necessary.

*Force-stripped row method*

This method was proposed by R. Murty et al. in [2]. The main idea is to calculate boundaries that divide the upper right interaction matrix into $P$ parts of equal area as shown in Fig 2(a). This has to be done only once during program initialization. Integration is done redundantly on all PEs for all particles, so only the calculation of the total force acting on each particle requires communication, which is done by a global reduction operation.

As long as $\mathrm{F}_{ij}$ is homogeneously populated, and as long as one neglects different loop overhead as well as cache effects due to the length of the matrix rows, which is continously decreasing as the processor rank increases, this scheme should achieve quite good load balance.

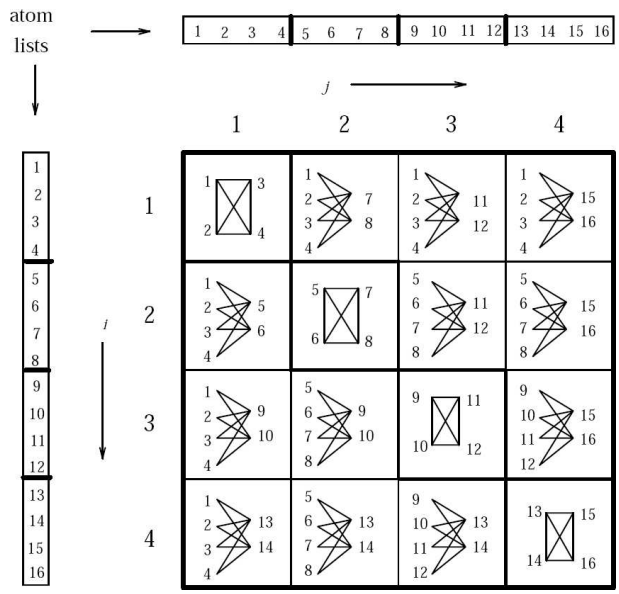



Figure 3: Low communication method (taken from [3]), example for 16 PEs and 16 particles. Diagonal PEs compute the forces between all particles of their $i$-index range, the others compute those exerted by the particles listed right on the ones listed left in each block.

*Force row-interleaving method*

The force row-interleaving method also originates from [2]. It is very similar to force-stripped row method, but the way of obtaining an even distribution of the area of $\mathrm{F}_{ij}$ to the PEs is slightly different: Instead of using calculated boundaries, matrix rows are assigned to the PEs consecutively. After every $P$ rows, the order of assignment is reversed, so that after every $2P$ rows all PEs have the same area assigned. This is shown in Fig. 2(b). Because usually $P \ll N$, the imbalance occurring if $N$ is not a multiple of $2P$ is negligible small.

Taking into account that inhomogeneities in $\mathrm{F}_{ij}$ are likely to cover more than just one row, one expects this method to gain reasonable load balance even for inhomogenous systems, since every PE gets assigned rows that are spread across the whole interaction matrix. However, the computation time per interaction may suffer from worse cache performance due to the fact, that the positions of $i$-particles now must be fetched from a significantly wider range of memory locations compared to force-stripped row method.

*Low communication method*

V. E. Taylor et al. describe in [3] a more sophisticated force decomposition method to reduce the time spent with communication. The single global communication (between $P$ PEs) is replaced by $\sqrt{P}$ parallel communication operations within communicators of size $\sqrt{P}$. This becomes possible thanks to the tricky arrangement of the PEs shown in Fig. 3. Contrary to the methods shown before, not only the $i$- but also the $j$-index range is spread across several PEs. Because integration is distributed across the $\sqrt{P}$ "diagonal" PEs, forces and positions have to be transferred only row- or column-wise from or to the diagonal and to the transposed PE.

The exact procedure done in each cycle of the simulation is described in the following:

1. The diagonal PEs broadcast updated particle positions within their rows and columns.
2. All PEs compute interactions from their assigned $i$- and $j$-range.
3. The lower triangular PEs send their forces to the PEs at their transposed position, where they are added to the forces computed there and vice versa.
4. Within every row all partial forces on the particles assigned to the contained diagonal PE are summed up and sent to that PE.
5. Integration is performed on the diagonal PEs.

The price one must pay for the reduction of communication this method offers is the relatively complicated implementation and an enlarged susceptibility to load imbalances due to local inhomogeneities of $\mathrm{F}_{ij}$ that comes from the additional distribution of the $j$-index range to several PEs.

**Improving load balance**

As force row-interleaving method promises load balance intrinsically (albeit at the cost of poor cash performance) and achieving good load balance for low communication method used on systems with inhomogenous interaction matrixes seems to be very difficult due to the twofold distribution of particle indexes to the PEs, I concentrated on improving force-stripped row method in respect of load balance.

The division of the interaction matrix into parts containing the same area suggested by Murty et al. acts on two assumptions:

- The interaction matrix is homogeneously set everywhere, so in every two sections with the same area also the number of interactions is the same.
- The number of interactions in a row is a measure for the execution time a PE spends with work related to that row.

*Balancing the number of interactions per PE*

*Balancing the consumed time per PE*

**Optimizing communication**

(under construction)

**Results**

*Benchmark systems*

(under construction)

*Performance in homogenous systems*

(under construction)

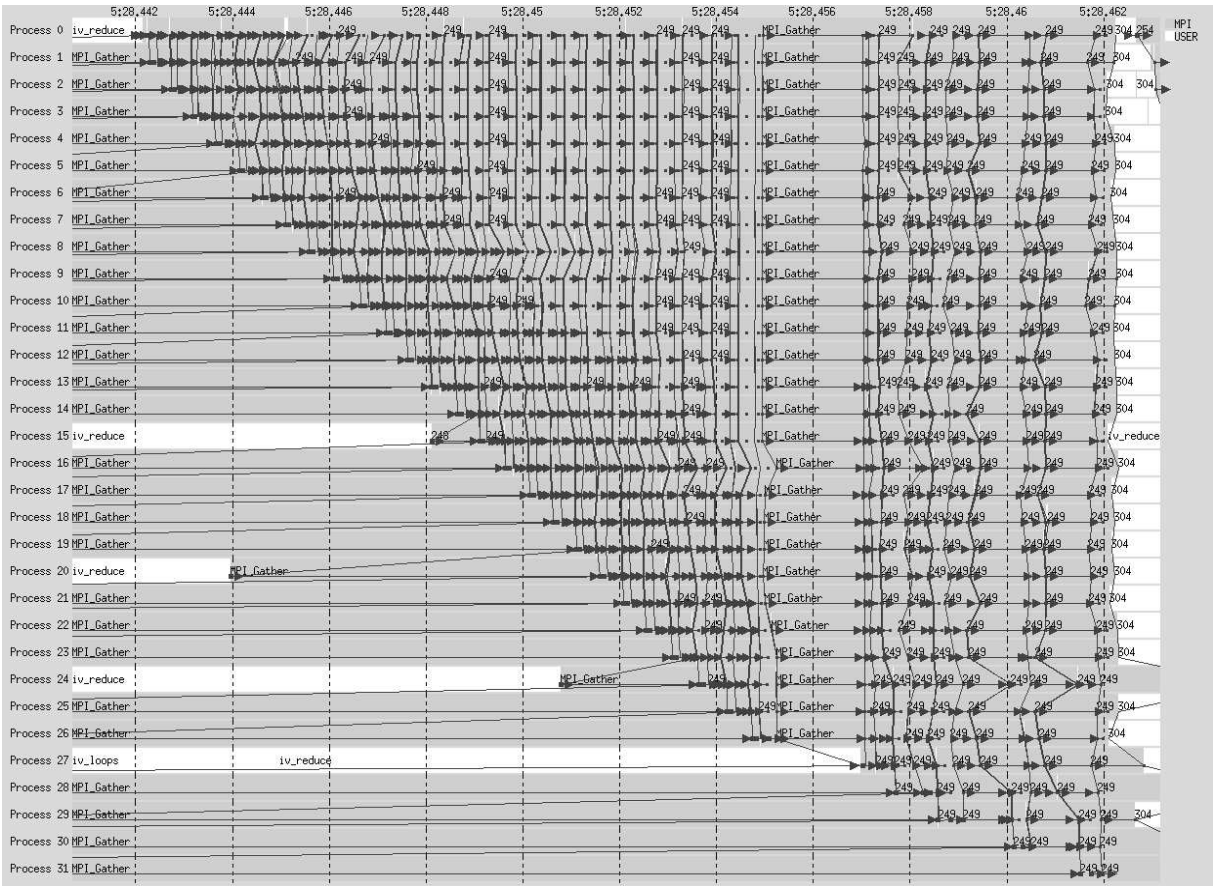



Figure 4: Timeline of the improved force communication visualized by `vampir`

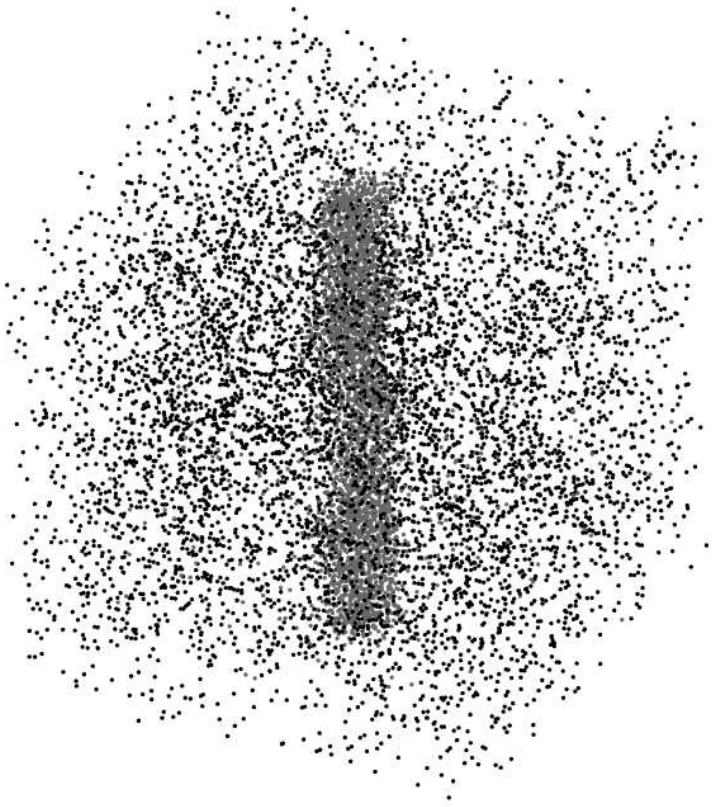
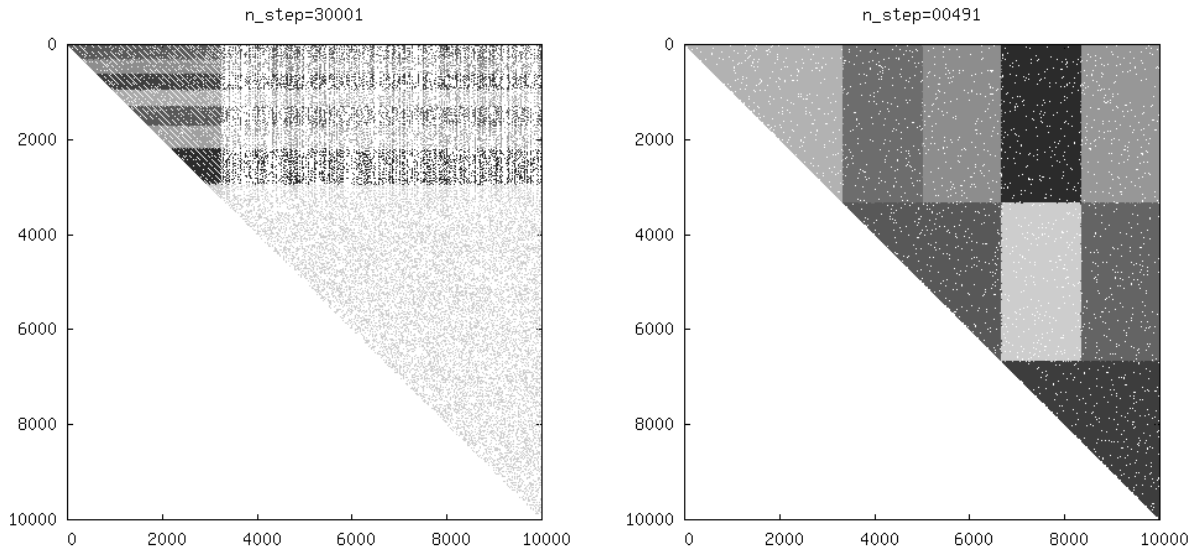


Figure 5: Inhomogenous benchmark system. One can clearly see the “wire” of “solid” running through the periodically continued simulation space.




(a) Force-stripped row method, inhomogenous system, 8 PEs (b) low communication method, homogenous system, 9 PEs

Figure 6: Interaction matrixes for two different systems and force decomposition methods. Interactions computed by different PEs are plotted in different shades of gray.

*Performance in inhomogenous systems*

(under construction)

### Further optimizations

(under construction)

### Acknowledgments

(under construction)

### References

1. G. Sutmann: *Classical Molecular Dynamics*. In Lecture Notes on Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms, eds. J. Grotendorst, D. Marx, A. Muramatsu, NIC Series **10**, 211–254 (2002)
2. R. Murty, D. Okunbor: *Efficient parallel algorithms for molecular dynamics simulations*. Parallel Computing **25**, 217–230 (1999)
3. V. E. Taylor, R. L. Stevens, K. E. Arnold: *Parallel Molecular Dynamics: Communication Requirements for Massively Parallel Machines*. In Proc. of the fifth Symposium on the Frontiers of Massively Parallel Computation, 156–163 (1994)

# Latency Optimized Parallelization of Near-Field Interactions in the Fast Multipole Method

Bernhard Kühnel

Chemnitz University of Technology

bernhard.kuehnel@s2002.tu-chemnitz.de

**Abstract:** This report presents an approach to implement the Fast Multipole Method (FMM) on parallel computers. After introducing the FMM, a fast summation algorithm that reduces the complexity of the Coulomb problem from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$, an implementation of the algorithm and parallelization by dynamic load balancing will be discussed. Finally, a new approach using static load balancing and communication overlapping is introduced. This new scheme is based on Global Arrays and intends to reduce the overall number of communication steps to a minimum, since the restricting part of the parallelization is introduced by the latency.

## Introduction to the Fast Multipole Method

### *Motivation*

Simulations of particle systems usually involve a pairwise potential that decays rapidly for greater distances. The problem is that for $N$ particles, one normally has to compute $\mathcal{O}(N^2)$ pairwise interactions. However, considering the small contribution of faraway particles, an approximation of the long-distance interactions may lead to feasible computational complexities and error estimates.

The Fast Multipole Method (FMM) has been developed for the case of a potential that is proportional to the linear inverse of the distance, in an aperiodic system. It can handle problems in molecular dynamics or cosmology; for example, Coulomb's law states that

$$E_c = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{q_i \cdot q_j}{d_{ij}}. \tag{1}$$

The main advantage of the FMM is its optimal theoretical complexity (computation and memory) of $\mathcal{O}(N)$. The complexity and error estimates are shown in [1] and will not be discussed in this report.

### *Basic outline of the FMM*

Like other fast summation algorithms, the FMM first builds up a hierarchical subdivision of the problem space. All particles are enclosed in a cubic bounding box at the so called "top level", which is then subsequently halved along each dimension, creating a tree of lower levels with smaller child boxes (see Fig. 1 for a 2D example). To guarantee the $\mathcal{O}(N)$ scaling, the tree depth is chosen so that at lowest level, the number of particles enclosed in a certain box is independent of the total number of particles.

The computation begins by calculating the far-field interactions: distant particles that are close to each other can be approximated as one pseudo-particle at the center of the particles' parent box, as shown in Fig. 3. The farther away such particles are, the more of them are pooled together, which greatly saves computation time and is needed for the linear scaling.

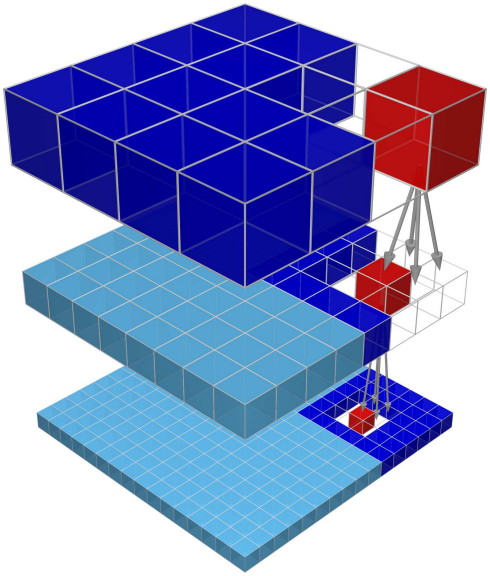


Figure 1: Subdivision hierarchy in the 2D case: each slice represents one level

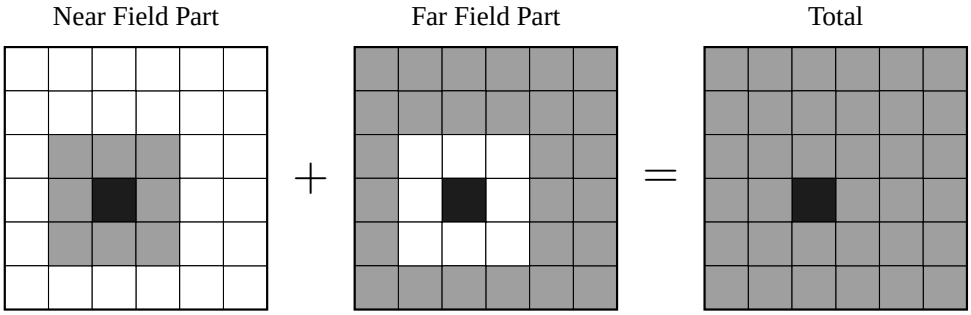
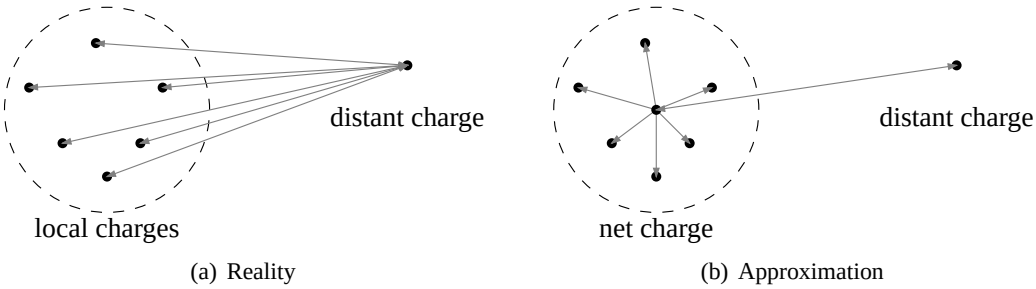



Figure 2: Distinction between near-field and far-field interactions




(a) Reality (b) Approximation

Figure 3: Principle of far-field approximation

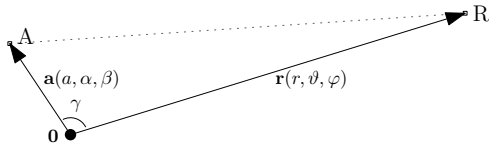



Figure 4: Spherical coordinates of a particle $A$ and the observer $R$

In a second step, the near-field interactions are computed directly, i.e. every box interacts with its near neighbors. Though the threshold between near-field and far-field can be varied, considering only the direct neighbors of each box has proven itself to be sufficiently accurate while showing a very good computational speed.

*Expansion of charge-charge interactions*

Before dealing with the details of the FMM, one must understand the principle of the underlying multipole expansions. First, let us switch to spherical coordinates. The distance $d$ between two particles $A(a,\alpha,\beta)$ and $R(r,\vartheta,\varphi)$ can then be expressed as

$$d^2 = |\mathbf{r}-\mathbf{a}|^2 = r^2 + a^2 - 2ra\cos\gamma, \tag{2}$$

with $\gamma$ being the angle between $\mathbf{r} = \overrightarrow{OR}$ and $\mathbf{a} = \overrightarrow{OA}$ (see Fig. 4).

Now, factorizing Eqn. (2) and defining

$$\begin{aligned} \varepsilon &:= \frac{a}{r}, \\ x &:= \cos\gamma \end{aligned}$$

yields

$$\begin{aligned} d^2 &= r^2\left(1+\varepsilon^2-2\varepsilon x\right), \\ \Rightarrow \frac{1}{d} &= \frac{1}{r}\left(1-2\varepsilon x+\varepsilon^2\right)^{-\frac{1}{2}}. \end{aligned} \tag{3}$$

The last factor of Eqn. (3) is the well-known generating function for Legendre Polynomials, so we can write

$$\frac{1}{d} = \frac{1}{r}\sum_{l=0}^{\infty}\varepsilon^l P_l(x) = \sum_{l=0}^{\infty}\frac{a^l}{r^{l+1}}P_l(\cos\gamma). \tag{4}$$

Finally, applying the spherical harmonic addition theorem [3]

$$\begin{aligned} P_l(\cos(\gamma)) &= \sum_{m=-l}^{l} Y_l^{-m}(\alpha,\beta)\cdot Y_l^m(\vartheta,\varphi), \text{ with:} \\ Y_l^{-m}(\alpha,\beta) &= \sqrt{\frac{(l-m)!}{(l+m)!}}P_{lm}(\cos\alpha)e^{im\beta} \text{ and} \\ Y_l^m(\vartheta,\varphi) &= \sqrt{\frac{(l-m)!}{(l+m)!}}P_{lm}(\cos\vartheta)e^{im\varphi} \end{aligned}$$

leads to

$$\frac{1}{d} = \sum_{l=0}^{\infty}\sum_{m=-l}^{l}\frac{a^l}{r^{l+1}}\frac{(l-m)!}{(l+m)!}P_{lm}(\cos\alpha)e^{im\beta}P_{lm}(\cos\vartheta)e^{im\varphi}. \tag{5}$$

To clarify the factorization of the potential into a multipole expansion $\omega_{lm} = qO_{lm}$ corresponding to the local part, and a Taylor-like expansion $\mu_{lm} = qM_{lm}$ for the distant part, we define

$$\begin{aligned} O_{lm}(a,\alpha,\beta) &= a^l \frac{1}{(l+m)!} P_{lm}(\cos\alpha) e^{im\beta}\,, \\ M_{lm}(r,\vartheta,\varphi) &= \frac{1}{r^{l+1}} (l-m)! P_{lm}(\cos\vartheta) e^{im\varphi}\,. \end{aligned}$$

This yields the final representation for the inverse distance,

$$\frac{1}{d} = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} O_{lm}(\mathbf{a}) M_{lm}(\mathbf{r})\,. \tag{6}$$

*Operators*

To facilitate the notation in the following sections, three operators on multipole and Taylor-like moments will be introduced.

*Operator A: Translate multipole expansions*

In order to go up one level in the tree, a multipole expansion around a child box's center **a** needs to be shifted to the center $\mathbf{a}+\mathbf{b}$ of the parent box. Let us define an operator $A$ that performs this operation:

$$\omega_{lm}(\mathbf{a}+\mathbf{b}) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} A_{jk}^{lm}(\mathbf{b}) \omega_{jk}(\mathbf{a})\,. \tag{7}$$

It can be shown that

$$A_{jk}^{lm} = O_{l-j,m-k}\,. \tag{8}$$

*Operator B: Transform multipole into Taylor-like expansions*

Operator B transforms a multipole expansion around **a** into a Taylor-like expansion around $(\mathbf{b}-\mathbf{a})$.

$$\mu_{lm}(\mathbf{b}-\mathbf{a}) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} B_{jk}^{lm}(\mathbf{b}) \omega_{jk}(\mathbf{a})\,, \tag{9}$$

resulting in

$$B_{jk}^{lm} = M_{j+l,k+m}\,. \tag{10}$$

*Operator C: Translate Taylor-like expansions*

Finally, operator C shifts a Taylor-like expansion around **r** to a new center $(\mathbf{r}-\mathbf{b})$. It is used to carry Taylor-like moments down the tree.

$$\mu_{lm}(\mathbf{r}-\mathbf{b}) = \sum_{j=l}^{\infty} \sum_{k=-j}^{j} C_{jk}^{lm}(\mathbf{b}) \mu_{jk}(\mathbf{r}) \tag{11}$$

with

$$C_{jk}^{lm} = O_{j-l,k-m}\,. \tag{12}$$

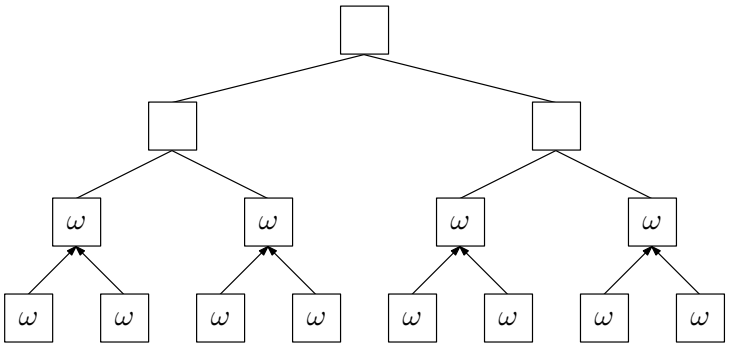



Figure 5: Pass 1: Calculation and shifting of multipole moments

*The FMM in detail*

*Pass 1: Calculate and shift multipoles*

Before creating the box hierarchy mentioned above, particle coordinates are scaled to the range $[0, 1]^3$, both for convenience and numerical stability. To descent one level, each box is split up into 8 child boxes (for 3-dimensional systems) and the particles are sorted into the new boxes. The maximum depth is reached when the number of particles per box becomes smaller than a given constant $K$.

In a second step, multipole expansions for all (non-empty) lowest-level boxes are calculated. Using operator $A$ from Eqn. (8), they are repeatedly shifted up to the third (for $ws = 1$) level of the tree (see Fig. 5), the highest level where well-separated boxes exist.

To be more precise about the well-separatedness criterion, we introduce a parameter $ws$ that defines how many boxes must at least lie between two boxes for them to be able to interact via multipoles. From here on, we will assume that $ws = 1$, meaning that only nearest neighbors are considered near-field. By increasing $ws$, the FMM will become more accurate, but at high performance costs.

*Pass 2: Transform distant multipoles into local Taylor-like expansions*

Pass 2 comprises the shifting of multipole expansions from well-separated boxes to the local box center, thus allowing computation of the far-field interactions. To achieve linear scaling, only those boxes are considered at each level which are themselves well-separated, but whose parent boxes are not. This means that summation is always done at the highest possible tree level. The distant multipoles are transformed using operator $B$ and then summed up, allowing to interact every local multipole expansion with all considered far-field particles at once.

*Pass 3: Transfer Taylor-like expansions to the lowest tree level*

To prepare the computation of the far-field potential in each box, the Taylor-like expansions calculated in pass 2 are consecutively shifted down the tree using the operator $C$ and added up with the existing expansions on each level, until all moments are present at the leaves of the tree.

*Pass 4: Calculate the far-field potential*

In each lowest-level box, the Taylor-like expansion for the entire far-field is now present along with the multipole expansions for the local particles. As shown in Eqn. (5), the far-field potential for every

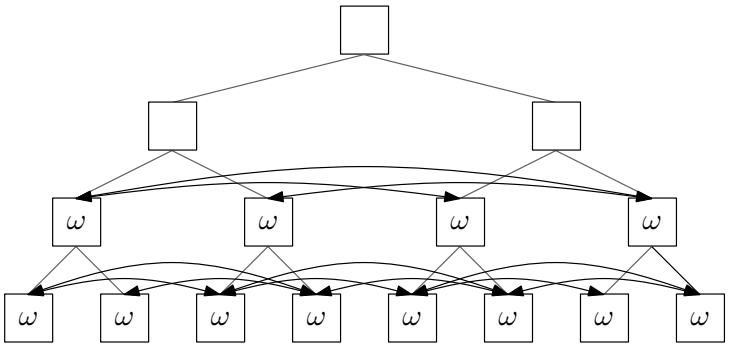



Figure 6: Pass 2: Transforming multipole moments

local particle can now be computed by evaluating the Taylor-like expansion for each particle multipole moment.

*Pass 5: Compute the near-field interactions*

The near-field part of the potential is computed via direct summation over every pair of particles in the current and all adjacent boxes (i.e. not well-separated ones), at the lowest tree level. For nomenclature, we will distinguish between in-box and box-box interactions, depending on whether two particles lie in the same, or in different boxes. This distinction is necessary for proper handling of data structures in our algorithm.

**Problem description**

Although an efficient sequential implementation of the fast multipole method already exists, it is unsuitable for very large problems ($10^7$ particles and beyond) due to memory and time limitations. The parallelization of the algorithm should concentrate on the passes 2 and 5 of the FMM which consume $\sim 90\%$ of the computational time altogether. Additionally, the fundamental differences between both passes' data structures (multipoles at box centers vs. particles inside boxes) suggest different approaches in parallelizing these two cases. In this report, only the nearfield part will be modelled and examined.

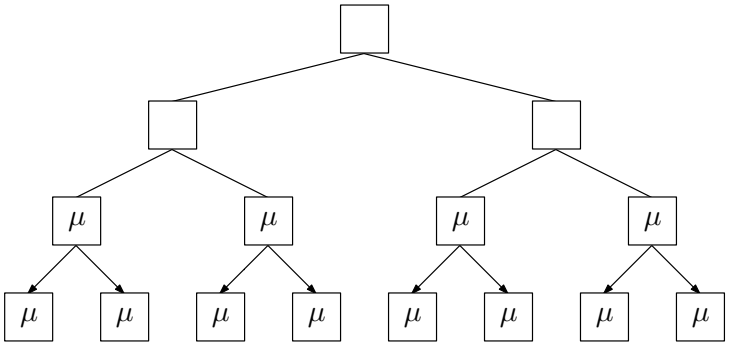



Figure 7: Pass 3: Shifting Taylor-like moments

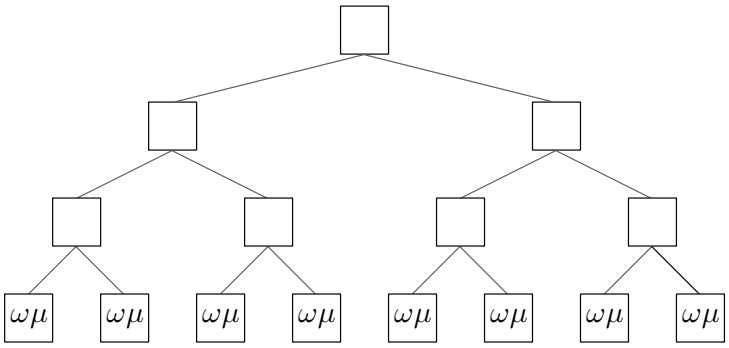



Figure 8: Pass 4: Evaluating the far-field contributions

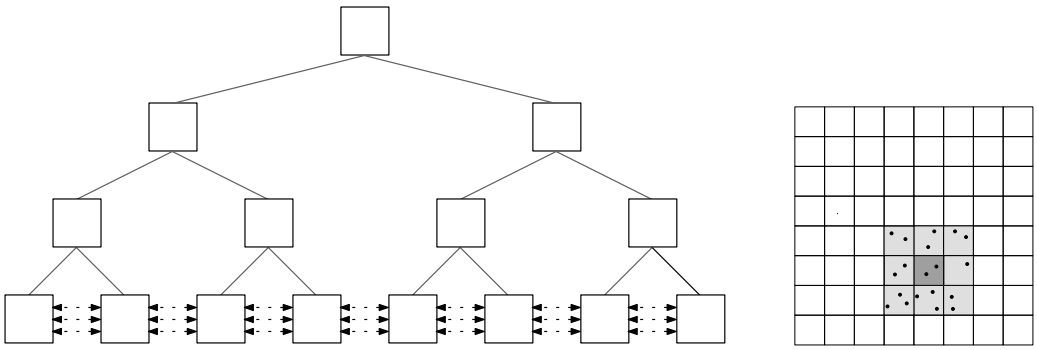
*Existing code*

At first, the sequential code, on which the parallel implementation is based, will be introduced. The simplified call tree for pass 5 computation can be visualized as follows[1]:

1. input data: particle charges `kuehnel/q`, particle coordinates `xyz`
2. `pass1...pass4`
3. `sortcharges(q,xyz,ibox)`
4. `pass5(q,xyz,ibox,enearfield)`
   (a) `skipeevector(ibox)`
   (b) `pass5inbox` (loop over lowest-level boxes)
      - `coulinbox` (calculate interactions between particles inside one box)
   (c) `pass5bibj` (loop over lowest-level boxes and their nearest neighbors [w.r.t. $ws$])
      - `coulinbox` (calculate interactions between particles inside two boxes)
5. output results: total `energy`, potential `fmmpot`, gradient (`fmmgrad`)

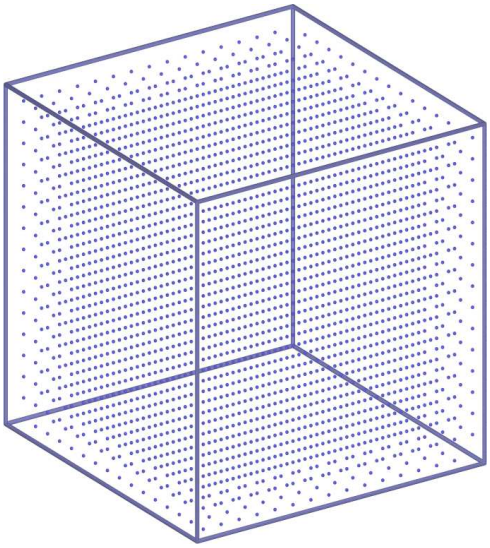
The main data structures are `q(1:N)` (particle charges, input), `kuehnel/xyz(1:3,1:N)` (particle coordinates, input), `kuehnel/ibox(1:N)` (associate particles with boxes, integer helper structure),

[1]Note that all names printed in `monospaced font` represent actual entities in the code, mostly subroutines or data structures.

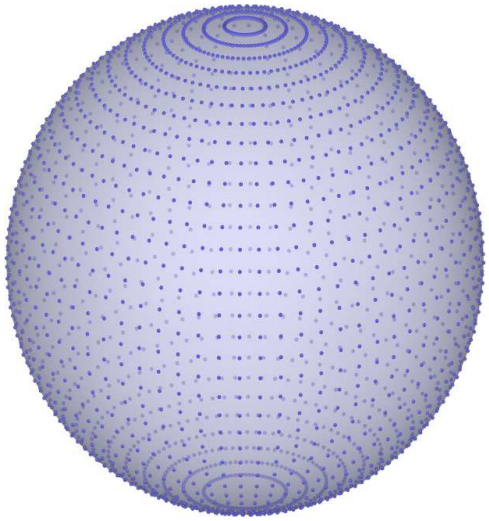


(a) Tree view (b) 2D visualization

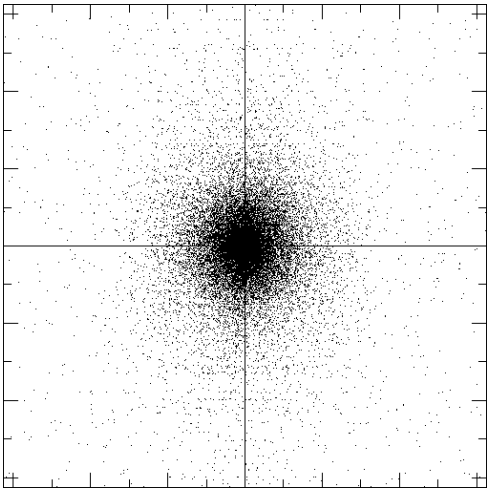
Figure 9: Pass 5: Calculating the near-field part



(a) Homogeneous distribution



(b) Spherical distribution



(c) Cluster

Figure 10: Typical test cases

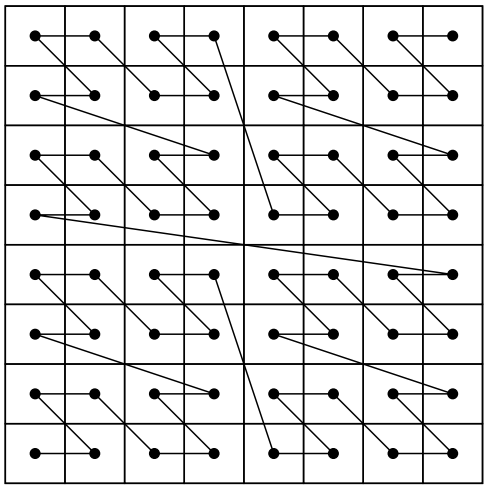


Figure 11: A 2D Z-curve, used to generate box numbers

`energy` (single value, output), `fmmpot(1:N)` and `fmmgrad(1:3,1:N)` (potential and gradient at each particle, output), where $N$ is the number of particles. Every box of the tree is identified by a unique number that is generated by bit-wise interleaving the integer x-, y- and z- coordinates in the box grid at a given level: e.g, a box with coordinates $x_3x_2x_1x_0$, $y_3y_2y_1y_0$, $z_3z_2z_1z_0$ $(x_i,\, y_i,\, z_i \in \{0,1\})$ will get the number with binary representation $z_3y_3x_3z_2y_2x_2z_1y_1x_1z_0y_0x_0$. Geometrically, this method generates a space-filling Morton (or Z-) curve as shown in Fig. 11. The `ibox` vector switches between two representations, depending on the stage of the execution: upon its generation, the "normal" form (Fig. 12(a)) with the actual (positive) box number at every particle index is used. After sorting the particles so that the `ibox` vector has an ascending order, it is transformed to the "skipvector" form (Fig. 12(b)). This representation permits quick traversal of the boxes: the first charge in a box holds the box's number, the other charges get negative `ibox` entries indicating the distance to the next box, and the last charge in a box gets a negative `ibox` entry that points to the beginning of that box.

*Target machines*

It is important for the identification of possible bottlenecks to have some knowledge on the targetted system. The JUMP supercomputer at Research Centre Jülich is an SMP cluster of 41 nodes with 32 IBM Power4+ processors and 128 GB RAM each, resulting in a peak performance of 8.9 Tflops. The nodes are connected through a high performance switch with a peak bandwith of 1.4 GB/s and measured average latencies of $30\mu s$. Shared memory can be accessed inside a node at the cost of $\sim 3\mu s$.

The second target system is JUBL, the IBM BlueGene/L at Jülich. Even though this system has a much better peak performance (45.8 Tflops on 8192 dual-core PowerPC440 processors), the local memory per processor is limited to 512 MB. Therefore, the granularity of the algorithm has to be fine enough to

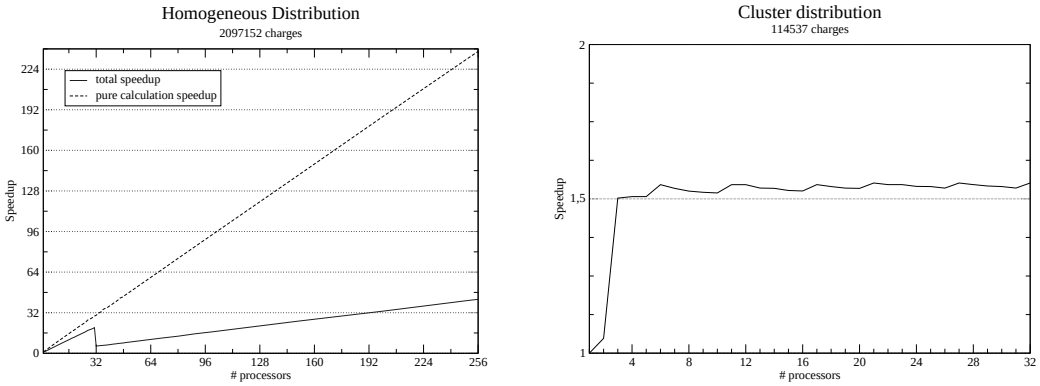
| 3 | 3 | 3 | 4 | 4 | 4 | 4 | 6 | 6 | 6 | 6 | 7 | 7 | 9 | 9 | 9 | 9 | 13 | 13 | 13 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

(a) "normal"

| -3 | -2 | -3 | 4 | -3 | -2 | -3 | 6 | -3 | -2 | -3 | 7 | -1 | 9 | -3 | -2 | -3 | 13 | -4 | -3 | -2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

(b) "skipvector"

Figure 12: Different forms of the `ibox` vector




(a) Latency impact

(b) Theoretical speedup on an inhomogeneous system

Figure 13: Identification of the main bottlenecks in the initial communication model (do not let a box cross a processor border, fetch single boxes as needed with blocking communication). In the simulation, the computation time normally spent in the `coulinbox` and `coulbibj` subroutines was accumulated separately from the latencies generated by every necessary communication.

permit computation of large problems on this machine.

*The Global Arrays package*

In order to maintain the existing data structures as far as possible, the Global Arrays library (maintained by Dr. J. Nieplocha) [4] is used. It offers automatic or manual distribution of arrays, one-sided and non-blocking communication and a convenient shared-memory-like view on almost any system.

*Initial parallelization approach*

As a first approach, the sequential algorithm has been parallelized by I. Kabadshow [2] using GA and dynamic load balancing. While on a single JUMP node the speedup was acceptable (growing linearly up to a value of $\sim 20$ for 32 processors), communicating over the network drastically increased latencies due to communication at box level. Additionally, the performance of the dynamic load balancer on inhomogeneous systems was insufficient.

These results motivated a deeper analysis of the bottlenecks for different cases. Given the measured latencies and the computational speed of the JUMP cluster, the parallel behaviour of the different test cases has been simulated with the model of non-overlapping, box-wise communication and with only complete boxes on each processor. As shown in Fig. 13, communication over the network and load imbalance on inhomogeneous systems are indeed the main bottlenecks in this model.

Following these observations, two main goals can be formulated for the approach presented in this report:

1. Prefetch larger "bunches" of data, instead of communicating at box level, thus saving latencies.

2. Overlap communication with computation to further hide long latencies between nodes.

3. Focus on good scaling of inhomogeneous distributions as well.

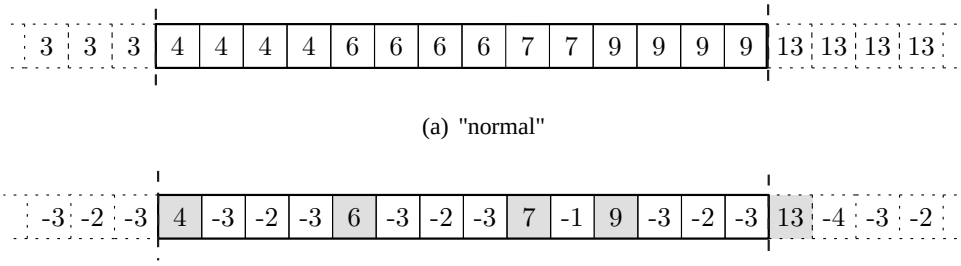



Figure 14: Box alignment along processor borders

**Program design**

In the following section, a parallel algorithm that uses static load balancing and latency optimization will be described along with the theory behind it. For a quick overview, the algorithm can be divided into 6 stages:

1. Box alignment
2. Preparation of communication
3. Initiation of communication
4. Local computation
5. Load balancing of split boxes
6. Iterative communication

*Initial data distribution*

All input data is supposed to be uniformly distributed on the processors. More specifically, processor `kuehnel/i` holds the portion $[(\texttt{i}-1)\cdot\frac{\texttt{ncharges}}{\texttt{nprocs}} : \texttt{i}\cdot\frac{\texttt{ncharges}}{\texttt{nprocs}}]$ of the vectors `g_q`, `g_xyz`, `g_ibox`, `g_fmmgrad` and `g_fmmpot` as global arrays; small rounding deviations pose no problem. The `ibox` vector is expected in the "normal" form (see Fig. 12(a)).

*Box alignment and load balancing*

Since for performance reasons the `coulinbox` and `coulbibj` subroutines only operate locally, every box that is passed to one of the subroutines must reside on a single processor. However, as a compromise between load balancing and overhead, "large" boxes that cross a processor border will be split up in a later step, while "small" ones are simply going to be transferred to a neighboring processor.

*Transferred boxes*

Every "small" box is communicated to the processor which already owns the box's largest part. As a criterion which boxes are small, we will set their maximum allowed size `s_chunk_b`, e.g. by defining the largest tolerated load imbalance `tol_imbalance`.

*Algorithm*

On each processor, do:

1. Compute the sizes and indices of the local left- and rightmost boxes, and store this information into a global array of dimension $\mathcal{O}(\texttt{nprocs})$,
2. Gather that array into a local data structure `boxinfo(1:nprocs)`, containing the lowest and hightest box and charge numbers on each processor,
3. Assign every box that crosses a processor border to the processor that owns its largest part, if the size of the smaller bit is less than `s_chunk_b`,
4. Reshape the global arrays `g_q`,`g_xyz`,`g_ibox`,`g_fmmgrad` and `g_fmmpot` into the resulting irregular form using `fmmga_reshape()` – now, every small box resides on a single processor,
5. Update `boxinfo` locally to reflect the new distribution.

*Split boxes*

"Large" boxes (w.r.t. the tolerance defined above) are split up on processor borders, creating new sub-boxes. In order to keep the loop over the boxes fast, the existence of sub-boxes is not queried in the first stage of the algorithm (for processors with many boxes containing few particles, an `if` statement even in the outer loop would cause a significant slowdown). Instead, a list of the split boxes will be created using the `boxinfo` array (see above), so that the few ($\mathcal{O}(\texttt{nprocs})$ boxes) left out interactions can be computed in a second stage.

*Preparing communication*

*Interaction between regular boxes*

Like in the sequential program, each processor determines the neighbors[2] of the boxes it holds, i.e. the boxes which lie geometrically above, on the right or in front of the designated box. The boxes that are not available locally will be directly sent by their owner (which determines the receiver via `l_procdata`) using a minimal amount of communication requests (worst-case latency: $\mathcal{O}(\texttt{nprocs})$) and stored into a local buffer. If that buffer is (in some very rare cases) too small to hold all neighboring boxes, all concerned interactions will be postponed until the buffer can be reused.

In order to achieve the $\mathcal{O}(\texttt{nprocs})$ latency, foreign boxes are put by the processor that holds them into the local memory of the processor requiring the boxes. This is possible because of the local knowledge of the box distribution and the one-sided communication library used.

[2] For neighboring split boxes, only the first sub-box is considered.

*Algorithm*

1. Determine local boxes that are needed by another processor,
2. Determine the processors they reside on,
3. Compute the total number of "items" (charges, coordinates, `ibox`) that every processor will receive,
4. Check local buffer space against total size,
5. If possible, create a sufficiently large send buffer,
6. Issue a `ga_put()` command to initiate the data transfer.

*Interaction with sub-boxes*

Three types of interactions have not been considered yet:

*Interaction between normal boxes and the sub-boxes (except the first one) of a neighboring split box*

There are three possible calculation sites for the aforementioned interactions: The owner of the regular box, the processors that hold the split box, or another, noninvolved processor. The last possibility will not be considered here (allthough it might result in much better load balancing) because much more programming and coordination effort/overhead would be necessary, and last but not least that processor would need enough memory for three portions of every global array (instead of only two).

The first possibility might at first seem like a reasonable choice (the owners of the split box has already got more load from internal interactions). However, if the split box extends over a few hundred processors, then we would have to compute 100 iterations on 1 processor instead of 1 iteration on 100 CPUs.

Therefore, following the above considerations, the processors holding the second and following sub-boxes of a split box will also compute the interactions with their predecessors.

*Interaction between all sub-boxes of a split box and all but the first sub-box of a neighboring split box*

Two extreme cases can be imagined in this scenario:

1. one split box extends over only two processors, the other split box is very large,
2. both boxes are equal in size and quite big.

The first case is compareable to the interaction with regular boxes, so the computation should be done by the processors owning the larger box. In the second case, this would obviously lead to half of the processors doing nothing. In this scenario, an alternating computation scheme would be best.

In order to encompass both cases, a cyclic distribution that depends on the ratio of the two box sizes will now be introduced. Let $\mathcal{A}$ be the smaller and $\mathcal{B}$ the larger box, $A$ and $B$ the number of processors holding each of them[3]. A given interaction will be computed on either a processor $i$ of group $\mathcal{A}$, or a processor $j$ of group $\mathcal{B}$.

[3]The first sub-box of one of the boxes (dependent on how they lie geometrically) is ignored in this section.

Then, that interaction is computed on processor $i$ if

$$(i+j) \bmod \left\lceil \frac{A+B}{A} \right\rceil = 0\,, \tag{13}$$

or, alternatively,

$$(i+j) \bmod \left\lfloor \frac{A+B}{A} \right\rfloor = 0\,. \tag{14}$$

Otherwise, it is computed on processor $j$.

A decision which of the above computation schemes is used depends on the resulting load imbalance. It can be shown that for Eqn. (13), the upper bound for the number of interactions computed by processor group $\mathcal{A}$ is $\frac{AB}{A+B}$, while the interaction number for group $\mathcal{B}$ can only be bounded above by $\frac{AB}{A+B} + \frac{A}{2}$, with an average count of $\frac{AB}{A+B}$ interactions per processor. Examination of Eqn. (14) results in a similar behavior with $A$ and $B$ switched in the estimates, and practical tests reveal some rare combinations of $A$ and $B$ where these bounds are strict. Therefore, we use a workaround that computes both worst-case estimates for the given values of $A$ and $B$ and uses the computation scheme associated to the minimum of both values (thus corresponding to the minimum load imbalance). Empirical results show a worst-case load imbalance of $\sim 18\%$ with the workaround, while it was at $\sim 50\%$ without. For details, see Fig. 15.

*Interaction between the sub-boxes of the same split box*

The interactions that were handled by `coulinbox` before splitting the box can be visualized as a complete graph, where the processors holding a sub-box are the vertices and interactions are represented by the edges. Now, if we orient the edges so that they point to the higher processor number of a pair $(i, j)$ if $(i+j) \bmod 2 = 0$, then all even processors will have to compute

$$\left\lfloor \frac{\binom{\texttt{nprocs}}{2}}{\texttt{nprocs}} \right\rfloor = \left\lceil \frac{\texttt{nprocs-1}}{2} \right\rceil$$

foreign sub-boxes, and the odd processors' part will be $\left\lfloor \frac{\texttt{nprocs-1}}{2} \right\rfloor$ sub-boxes.

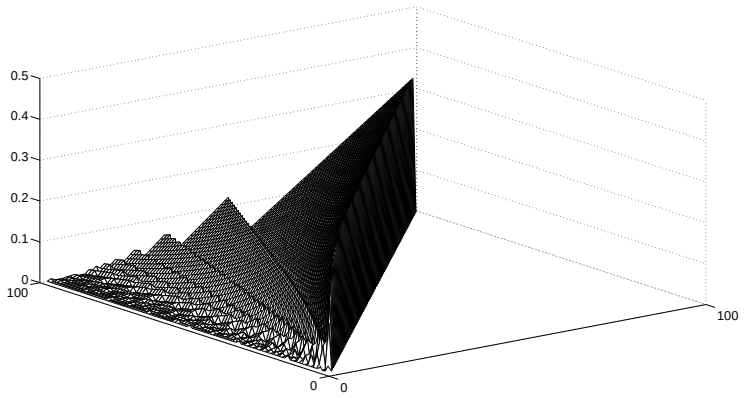
*Communication and computation*

For cache optimization, all local interactions (which boxes lie on the local part of the global array) are computed in multiple loops with pre-computed boundaries, first leaving out the interactions with buffered boxes. The latter are also computed continuously in a following stage.

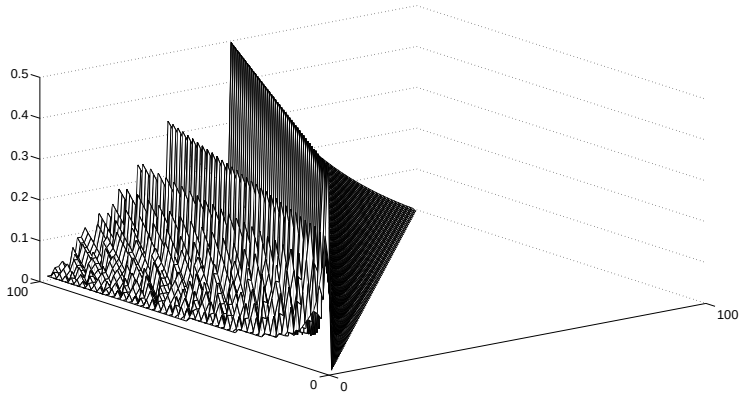
The communication requests to fill the buffer can be issued before the start of the first phase and complete during the calculation. After a call to `ga_sync()`, the buffer may be accessed for the second phase.

**Results**

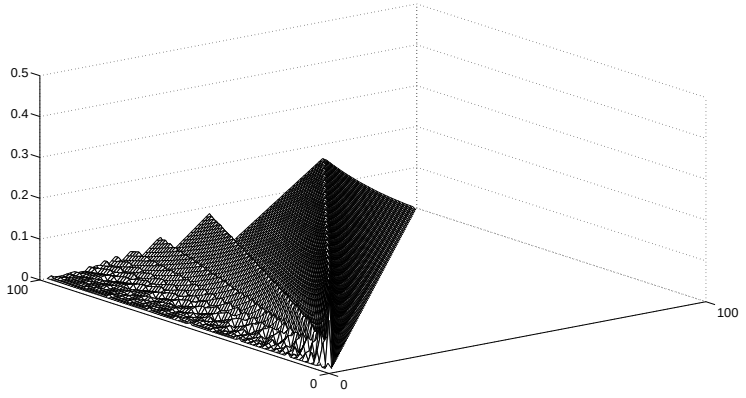
Fig. 16 shows the scaling of a medium-sized problem on up to 8 JUMP nodes. While the first parallel implementation with dynamic load balancing was not practical on more than one node (due to box-wise communication and resulting latencies), the new program can go beyond this barrier and may still be fine-tuned w.r.t. communication overlapping. Additionally, a larger number of particles could result in a better and more realistic measurement: in the above case, the computation time on 256 processors was less than 0.1s. Unfortunately, measurements for these very large systems were not possible due to memory limitations in the remaining sequential code.




(a) using Eqn. (13)




(b) using Eqn. (14)




(c) using the minimum

Figure 15: Verification of the calculated upper bounds of the load imbalance when distributing the interactions between two split boxes. The X and Y axes represent values for $A$ and $B$, the Z axis shows the deviation from the (optimal) average load in percentages.

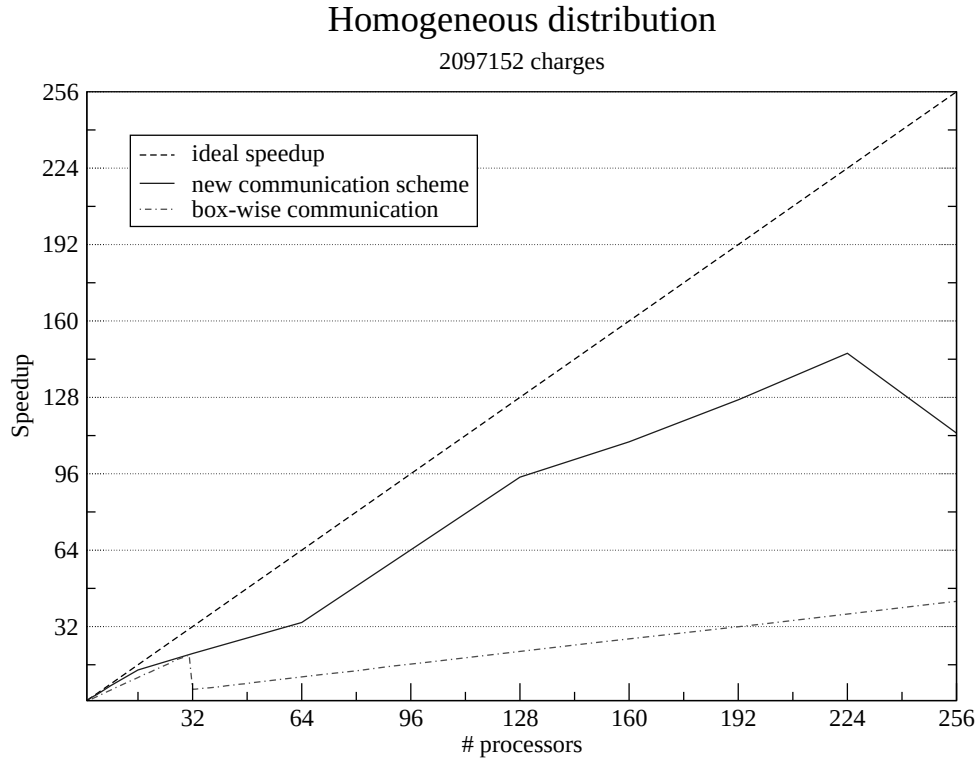



Figure 16: Speedup measurements for a homogeneous distribution of $8^7$ charges on JUMP. The simulation results from Fig. 13 have been included for comparison.

**Outlook**

Due to the guest students program's time limitations, the split box communication scheme was not yet fully implemented, thus no speedup measurements were possible. Beyond this remaining implementation issue, several further enhancements to the code are possible.

For example, the aforementioned fine-tunings can be implemented by slightly modifying the outer loop in `pass5inbox` and the former `pass5bibj`. The duration of every non-blocking communication operation is estimated by given latency and (if necessary) bandwidth values, then, the corresponding number of "`icharge` loops" is calculated (e.g. using given profiler results for the current machine). That way, as much computation as possible is used to overlap communication.

To go even further, one could compute non-local interactions as the foreign chunks arrive instead of waiting for all locally needed data to be available. However, due to the computational or memory usage overhead in trivial implementations, the necessity of this step will still have to be analyzed in detail, with respect to the present network topologies.

The remaining optimizations particularily target load balance. First, notice that until now, the number of particles per processor is balanced. Although, computation time mainly depends on the number of interactions between the particles held on a processor (locally or received by others). While for homogeneous distributions the first method issues an almost optimal computational speedup (see Fig. 13(a)), the number of near-field interactions can be expected to grow with decreasing particle distance. Thus, a good heuristic for each box's number of interactions will have to be developed to optimize data distribution.

A much simpler approach to improve load balance is based on the observation that very concentrated agglomerations of particles usually do not fit on a single processor, resulting in the creation of split boxes.

Therefore, the scaling could be improved up to a certain degree by reducing the maximum number of particles per processor that holds a split box, using an empirically determined factor.

**Acknowlegements**

I would like to thank my supervisors Dr. Holger Dachsel and Ivo Kabadshow for their great and continued support during the last ten weeks. I am also grateful for Dr. Rüdiger Esser, who made this guest students program possible. Furthermore, I greatly appreciate the work of Prof. Dr. Grotendorst and other members of the Central Institute for Applied Mathematics, who helped correcting this report and solving other occurring problems. Finally, I would like to express my gratitude to Prof. Dr. Meyer from TU Chemnitz for recommending me for the guest students program.

**References**

1. C. A. White and M. Head-Gordon. Derivation and efficient implementation of the fast multipole method. *J. Chem. Phys.*, 101:6593-6605, October 1994
2. I. Kabadshow. The Fast Multipole Method - Alternative Gradient Algorithm and Parallelization. Diploma thesis, TU Chemnitz, 2006
3. http://mathworld.wolfram.com/
4. Global Arrays Documentation at http://www.emsl.pnl.gov/docs/global/

# Determination of Ground State Degeneracy of Systems with Phase Transitions of First Order and a Simple Chain Model Using Improved Parallel Monte Carlo Methods

Martin P. Magiera

Universität Duisburg - Essen
Campus Duisburg
Theoretische Physik

E-mail: m.magiera@uni-duisburg.de

**Abstract:**

The determination of ground states of systems with many degrees of freedom and their degeneracy is not only an interesting topic in statistical physics, but also in computational biophysics (protein folding) or financial market research. For such systems energy landscape becomes rough and the minimization of their Hamiltonian nontrivial, so statistical methods become necessary. In this work simple systems with phase transition of second and first order (Ising and Potts model) are studied by Monte Carlo simulations, using parallel tempering and multiple Gaussian Monte Carlo methods to make a random walk through the complete phase space possible. The trivial degeneracy of these systems is calculated, simulations are performed parallel on "*Jülich Multiprocessor*" (*JUMP*). Finally simple two dimensional chain systems are simulated and their ground state degeneracy is determined, where a new phase transition like behavior is found.

## Introduction

The natural description of biological systems (problem of protein folding [1]), complex financial market development or any other complicated problem requires some nontrivial interaction terms in the Hamiltonian, so solution of these systems analytically becomes in most cases impossible. Simulation of these very rough potentials by any Monte Carlo or molecular dynamics simulation furthermore becomes complicated, because the studied system easily gets trapped in a local minimum of the energy landscape. So in that case there is a certain probability, that a computer simulation will never reach a ground state or global minimum, at least not in finite time. Nevertheless knowledge of these ground states or states near the ground energy is very important, because it can solve fundamental questions about protein folding and misfolding in nature, the best behavior in a market situation or whatever the task is about. So improvement of Monte Carlo techniques to exceed these natural energy barriers and find global minima in free energy faster is an actual research topic, and several methods were developed and improved in a short time, as the Multicanonical method [2, 3] or the Wang Landau method [4, 5].
In this work mainly parallel tempering is used, an algorithm which scales excellently on parallel machines and reaches good Monte Carlo results after short time. For systems with rough phase transitions additionally the Boltzmann weight is modified by multiplied Gaussian functions, a new technique of T. Neuhaus [6], to improve the random walk in $\beta$ space. Finally the program is rewritten to simulate short

chain models with discrete Lennard Jones interaction between chain nodes, using the same techniques. Simulating these short chains shows unpredicted but interesting effects and forces a further study in the future. We thought of several expansions of the simple model, which are discussed in the last part and give a great potential for studies in the future.

## Theory

*Statistical Physics of Canonical Ensembles*

A very powerful method to describe physics of systems with a huge number of degrees of freedom is the canonical ensemble. The basic idea of this ensemble is, that the examined system $H(\vec{\eta})$ is connected by a heat bath to environment, which can assume any inverse temperature $\beta = \frac{1}{k_B T}$. Only heat can be exchanged, for energy or particle exchange the system is isolated. Using the Boltzmann distribution, one finds the canonical density function, which describes the probability to find the system at temperature $\beta$ in the state $\vec{\eta}$:

$$\rho_\beta(\vec{\eta}) = \frac{e^{-\beta H(\vec{\eta})}}{Z_c} \tag{1}$$

where $Z_c$ is the canonical partition function, given by

$$Z_c(\beta) = \int_{\vec{\eta}} d\vec{\eta}\, e^{-\beta H(\vec{\eta})}$$

Knowledge of the density function (1) contains the solution of the statistical system $H(\vec{\eta})$. Physical quantities like the energy can be calculated with the density function by integrating this quantity over whole phase space $\vec{\eta}$:

$$\langle E \rangle_\beta = \int_{\vec{\eta}} d\vec{\eta}\, \rho_\beta(\vec{\eta})\; H(\vec{\eta}) \tag{2}$$

However, it is easier and more favorably not to know how system behaves in the higher dimensional phase space, but in one dimensional energy space, especially if one calculates some energy quantities as the ground state degeneracy. Then the energy density of states function, called $n(E)$ here, is interesting and fundamental to solve the system. Using the Laplace transformation one can calculate the canonical partition function with the density of states function:

$$Z_c(\beta) = \int_E dE\; n(E)\; e^{-\beta E} \tag{3}$$

The density of states function also allows to calculate any physical quantity like inner energy, specific heat, etc.

$$\begin{aligned} \rho_\beta(E) &= n(E)\frac{e^{-\beta E}}{Z_c(\beta)} \\ \langle E \rangle_\beta &= \int_E dE\; E\; \rho_\beta(E) = \int_E dE\; n(E)\frac{e^{-\beta E}}{Z} \qquad (4) \\ c_{\,V,\beta} &= \left\langle E^2 \right\rangle_\beta - \left\langle E \right\rangle_\beta^2 \qquad (5) \end{aligned}$$

It is important to realize, that the density of states function $n(E)$ is a global function, which gives the number of states a system can ever reach at any temperature $\beta$, while the density function $\rho_\beta(E)$ only gives the probability to reach a state according to the heat bath temperature.

*Physical Systems*

*Ising Model*

The Ising Model is the most simple system one can describe with the canonical ensemble. It is solved analytically in two dimensions [8], so it is an ideal test bed for the methods used for statistical physics. The Ising model gives the possibility to simulate magnetic quantities by adding an additional term to the Hamilton function. In this work only energetical quantities are in point of interest, so the simple Hamiltonian without magnetic properties is used:

$$H = -\sum_{\langle ij \rangle} J_{ij}\sigma_i\sigma_j \tag{6}$$

The brackets $\langle ij \rangle$ express a sum over all next neighbors $\sigma_i$ and $\sigma_j$ of a spin ensemble, placed on a rectangular 2 dimensional lattice with periodical boundary conditions. A spin pair $\langle ij \rangle$ interacts through the interaction term $J_{ij}$, which is for simplicity taken isotropic throughout the whole system. $\sigma_i$ can occupy two possible states, $|\uparrow\rangle$ and $|\downarrow\rangle$, or $|1\rangle$ and $|-1\rangle$. For $J_{ij} = 1$ a pair of equal spin configurations gives a negative contribution to the total energy, different spins a positive. In the two dimensional Ising Model, one gets a phase transition of second order at $\beta_t = \frac{k_B}{2J}\ln(1+\sqrt{2}) \sim 0.441$. Beyond this temperature a highly ordered system is present, above the order disappears.

*Potts Model*

To make more than two states possible per spin and simulate some systems with phase transitions of first order, the Potts model [7] can be used. Equal to the Ising model spins are located at a n dimensional lattice and interact through $J_{ij}$. Now any integer value between 1 and $q_{max}$ is a possible state for each spin.

$$H = -\sum_{\langle ij \rangle} J_{ij}\delta_{q_i,q_j} \quad q_i \in \{1; 2; \ldots; q_{max}\} \tag{7}$$

$\delta_{q_i,q_j}$ is the Kronecker delta function, which returns 0 for $q_i \neq q_j$ and 1 for $q_i = q_j$. By adding a factor to $J_{ij}$ and adding a constant to $H$, one gets (6) from (7) with $q_{max} := 2$. The Potts model performs a phase transition of first order at $\beta_t = \frac{k_B}{2J}\ln(1+\sqrt{q_{max}})$, if $q_{max} \geq 5$ and of higher order far smaller $q_{max}$ [9].

*Chain Model*

For the chain system some chains of length $L$ are constructed with $L+1$ chain nodes. The distance between two nodes stays always constant, so their angles represent $L$ degrees of freedom. Each node interacts with its neighbors after the next due to the Lennard Jones potential, which is a typical representation for short range interactions.

$$H = \sum_{i,j} 4\epsilon \left( \left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6} \right) \tag{8}$$

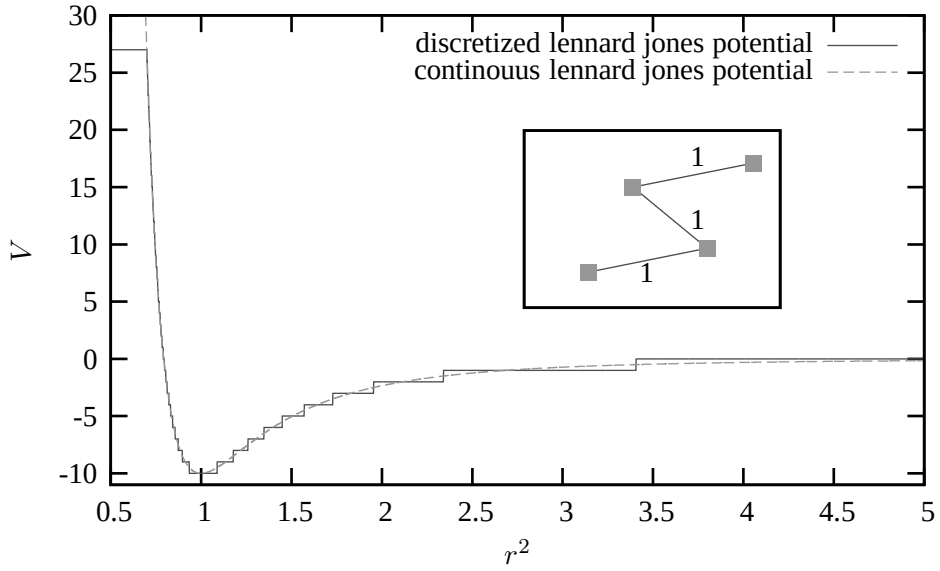



Figure 1: Plot if the discrete and the continuous Lennard Jones potential $V(r^2)$. The discrete version has no specific global minimum, but a 'plateau', This will be significant for later ground state degeneracy calculation. In the inlay an example for simulated chain is plotted. The distances between the nodes are fixed to one, angles are varied through the Monte Carlo algorithm.

The parameters $\sigma$ and $\epsilon$ are chosen in the simulation the way that the potential has its minimum at the distance between two next neighbors, which is normed to 1. The potential minimum is normed to $-10$, so the $\beta$ values simulated here have to be scaled by the factor 10 to be compared with $\beta$ values of usual simulations, where the potential minimum is at $E = -1$. For performing the simulation, the Lennard Jones potential had to be discretized, and the residuum at $r = 0$ had to be eliminated. The potential reaches a maximum at a certain minimal distance, which is chosen carefully. A too high potential maximum needs much memory at bigger chain systems, because the energy range scales with $\mathcal{O}(L^2)$. A too small one lets appear edge effects like overlap of nodes.

**Simulation**

*Monte Carlo*

Because one cannot cross phase space $\vec{\eta}$ continuously (using Potts Model you have $q_{max}^V$ different states, when $V = L^d$ is the number of simulated degrees of freedom) usually Monte Carlo methods are used. That means, that states $\vec{\eta}$ are randomly generated, quantities measured, new random states created and so on. In order to make these simulations more efficiently, Nicolas Metropolis developed an efficient algorithm in the 1950s [10], which weights the random walk through phase space due to the Boltzmann factor $e^{-\beta E}$. Using the so called Metropolis algorithm, only at the specific temperature $\beta$ occupied states are simulated. Finally this weighting has to be recalculated (see page 85) to get a density of states. Using the Metropolis algorithm, a new state is created by p.e. changing one spin or angle. For this new state the energy function (6- 8) is calculated, and depending on the energy difference between the old energy value and the new one the probability to assume the new state is given by Metropolis criterium (9).

$$\begin{aligned} \omega_{\vec{\eta}_i \to \vec{\eta}_j} &= \min(1, e^{-\beta \Delta E}) \\ \Delta E &= E_{\vec{\eta}_j} - E_{\vec{\eta}_i} \end{aligned} \tag{9}$$

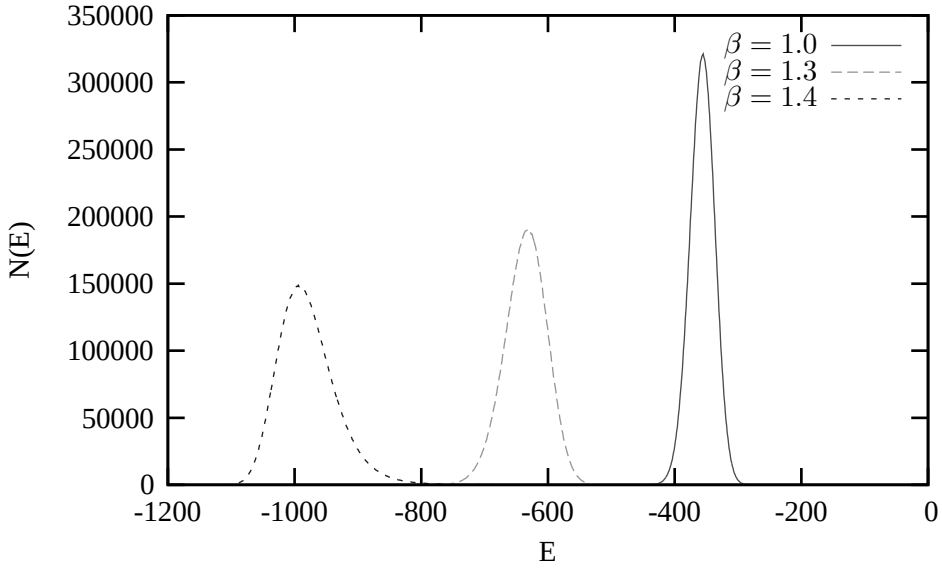



Figure 2: Plot of single histogram functions from several MC - Simulations, each with 15.625 MegaSweeps (App. A). Each simulation crosses only a specific interval of $E$ - space.

Using the Metropolis algorithm the system performs a random walk in a certain $E$ - interval as one can see in fig. 2. To get a simulation of the whole phase several simulations for several $\beta$ - values are needed.

*Parallel Tempering*

To cover the whole phase- or energy space, a random walk through the $\beta$ - space is very useful. That is done by the parallel tempering algorithm [11]. For parallel tempering several simulations of different temperature $\beta_i$, so called replicae, are performed parallel for a specific time of Monte Carlo sweeps (App. A). Then a trial parallel tempering step is done, so two neighbored replicae exchange their temperatures or configurations. The new energy functions are calculated for each replica, and one gets a probability for the exchange:

$$\begin{aligned} p_{(\vec{\eta}_i,\beta_i),(\vec{\eta}_j,\beta_j)\to(\vec{\eta}_i,\beta_j),(\vec{\eta}_j,\beta_i)} &= \frac{e^{-\beta_i E_{\vec{\eta}_j}}\ e^{-\beta_j E_{\vec{\eta}_i}}}{e^{-\beta_i E_{\vec{\eta}_i}}\ e^{-\beta_j E_{\vec{\eta}_j}}} = e^{\Delta\beta\ \Delta E} \\ \Delta E &= E_{\vec{\eta}_j} - E_{\vec{\eta}_i} \\ \Delta\beta &= \beta_j - \beta_i \end{aligned} \tag{10}$$

Combining (10) with the Metropolis criterium, one gets the in the simulations used swop probability:

$$\omega_{(\vec{\eta}_i,\beta_i),(\vec{\eta}_j,\beta_j)\to(\vec{\eta}_i,\beta_j),(\vec{\eta}_j,\beta_i)} = \min(1, e^{\Delta\beta\Delta E}) \tag{11}$$

The trial parallel tempering step only makes sense for small $\Delta E$ and $\Delta\beta$, so only swaps between neighbored replicae are tried in the simulations. In order to keep them small, several well chosen $\beta_i$ - replicae are needed ($\beta_i$ - bin). The performance of the parallel tempering algorithm can directly be measured by defining an ergodicity time: An ergodic system is a system, where every point of phase space is arbitrarily close to the phase space trajectory. The ergodicity or tunneling time $\tau$ is defined here as the mean time the system needs to tunnel from a high temperature phase $\beta_{min}$ to a low temperature phase $\beta_{max}$ and back. An optimal $\beta_i$ - bin will produce very low $\tau$ values, a too large $\Delta E$ will let $\tau$ decrease.

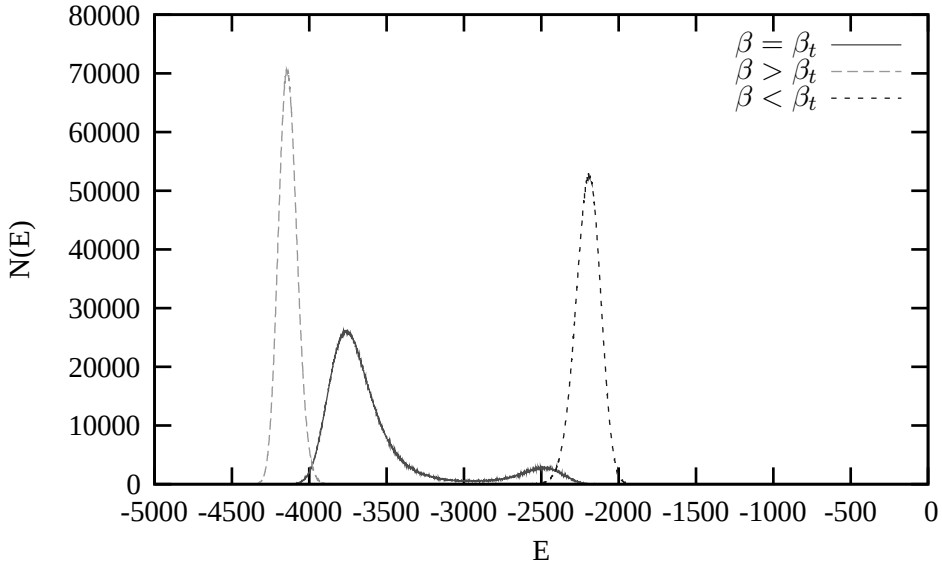



Figure 3: Plot of the single histogram functions for $\beta$ greater, lower and equal to $\beta_c$ for $48^2$ Potts model with $q_{max} = 8$. At E = -3000 appears an energy gap, which blocks random walk through energy space. The high - and low temperature peak follow the relationship $\frac{1}{q_{max}}$, because at low temperature there are $q_{max}$ possible states the system can occupy.

For systems with rough phase transitions of first order like the Potts model with $q_{max} \geq 5$ one gets an energy gap between high and low temperature (fig. 3), so the probability for a parallel tempering swap in that region becomes zero and $\tau$ decreases. In order to avoid this energy gap at the critical temperature, one can modify the Boltzmann factor of the canonical ensemble and create a probability for the system to occupy a state in the energy gap. That is done p.e. by introducing some Gaussian functions of width $\Delta E$ at energies $E_0$ and temperature $\beta_t$ [6].

$$p_i(E, \beta_t) = e^{-\beta_t E}\ e^{-\left(\frac{E-E_{0i}}{\Delta E_{0i}}\right)^2} \tag{12}$$

By carefully selecting $E_{0i}$ (App. C) simulation of systems with phase transition and crossing of the energy gap by parallel tempering becomes possible (see table 1).

| | ergodicity time $\tau$ in supersweeps | |
|---|---|---|
| system size | not modified ensemble | modified ensemble |
| $24^2$ | 5.834.749 | 490.539 |
| $32^2$ | 7.864.691 | 707.200 |
| $48^2$ | inf | 2.677.837 |

Table 1: The mean ergodicity times of three modified and not modified systems, calculated after simulations. $\tau$ is given in units of supersweeps. For the not modified $48^2$, $\beta$ ensemble, there was no tunneling event throughout the whole simulation.

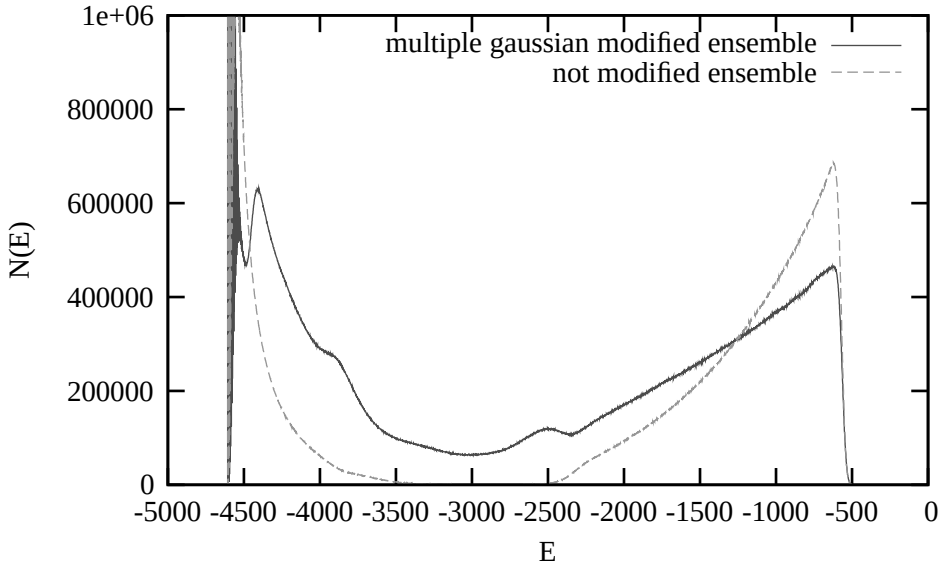



Figure 4: Plot of a multi histogram function of Potts model with $q_{max} = 8$. At the energy gap E = -3000 there now appear histogram entries because of the modification of the Boltzmann factor with multiple Gaussian functions (12).

*Simulation Data*

All the simulations performed for these studies lasted one gigasweep, a common quantity for Monte Carlo simulations. As input a carefully selected $\beta_i$ template (App. C) is created. As an initial system for the spin simulation a high temperature random system was created, for chains all angles were set to zero. In each simulation step in the spin simulation one randomly selected spin was flipped (*magiera/single spin flip algorithm*), analogue in the chain systems one randomly selected angle was changed. The selection range of the new angles $\Delta\phi_i$ is limited, and varies throughout the whole simulation to get a satisfying acceptance rate for Metropolis criterium (9) $\omega \sim 0.5$. All together it stayed constant with small deviations. Before measurement begins, an equilibration period is simulated, which lasts $\frac{1}{10}$ of whole simulation time.
During the simulations all Monte Carlo sweep energy values are saved in a multi histogram $P_{multi}$. The simulations were performed parallel on several processors, so message passing through *MPI* was necessary for parallel tempering swaps and collecting the histograms after the simulations. However, most computing time of the optimized code is spent in calculating random numbers, so the little message passing does not impair the linear scaling behavior with the processor number.

*Reweighting and Normalization*

In order to get the density of states function $n(E)$ of simulation data $P_{multi}$, the Boltzmann factors have to be reweighted and the solution has to be normalized. Here one has to care about the possibly introduced modifications to simulate phase transitions. In the reweighting algorithm by Ferrenberg and Swendsen [12, 13] free energies $F_i$ are iterated for each simulated replica $\beta_i$. After reweighting the high temperature limes ($\beta = 0$) of the resulting distribution function is normed with the known density of states for the respective systems. For Potts model one gets $q_{max}^V$ states, for chains it is straightforward to find $(2\pi)^L$ as the number of states at infinite temperature. For better handling of the huge quantities logarithms are useful.

$$
\begin{aligned}
\log P(E) &= \log P_{multi}(E) + \log N - \sum_{\beta} e^{-\beta E - F_{\beta_i}} \\
F_{\beta_i} &= \log \sum_{E} P(E) e^{-\beta E - \left(\frac{E-E_0}{\Delta E_0}\right)^2} \\
Z_0 &= const \sum_{E} P(E) \\
&= e^{const + F_0}
\end{aligned}
$$

*Expected Degeneracys*

For Ising and Potts models, the ground state degeneracy is straightforward the number of possible spin states $q_{max}$. The chain degeneracies will be more complicated. Depending on the chain length $L$ there will be several possible ground states $N(L)$ given by geometry, lying on a triangular lattice. The minimum plateau, created by discretization of the Lennard Jones potential, makes some fluctuations of the chain nodes possible. Each ground state angle $\phi_i$ is allowed to fluctuate in a certain range $\Delta\phi_i$, depending on the ground state geometry and the value of all $\phi_{j\neq i}$. So because of the extreme coupling there is the non trivial integral (13) to solve in order to calculate a phase space factor for each ground state. The most procuring point is, that geometry of the ground state is not known a priori, and every state might have a complete different one. In the simulations the first node is fixed, so every ground state is degenerated concerning rotation around this first node.

$$
\xi_L = \int d\phi_0 d\phi_1 \dots d\phi_{L-1} = 2\pi \int d\phi_1 \dots d\phi_{L-1} \tag{13}
$$

For simplification one can imagine, that the coupling gets smaller for more finely discretized potentials, and set it constant to a $\Delta\phi = \Delta\phi_1 = \dots = \Delta\phi_{L-1}$. The $\Delta\phi$ value is determined by solving the integral for the most trivial chain of length $L = 2$, where no coupling exists, and one can perform the calculation analytically using the potential discretization. Then one gets a $\xi_L = 2\pi(\Delta\phi)^{L-1}$.

$$
\begin{aligned}
\xi_2 &= 2\pi \int_{\phi_{min}}^{\phi_{max}} d\phi_1 \\
&= 2\pi \left| \arccos\left(1 - \frac{r^2}{2}\right) \right|_{r_{min}}^{r_{max}} \\
\xi_L &= 2\pi \left( \left| \arccos\left(1 - \frac{r^2}{2}\right) \right|_{r_{min}}^{r_{max}} \right)^{L-1}
\end{aligned} \tag{14}
$$

In order to be able to estimate the results the best, for the chain systems ground state configurations are saved. When a in a test run determined ground state energy is reached, by the interaction map of the state it is checked, if this ground state had ever been reached before. If not, the new interaction map is saved. The interaction map is a regular matrix, where the interaction contribution of each pair of nodes is saved. Because of symmetry, always exact two states share one interaction map, which are mirror symmetrically to the axis going through the first two nodes of a chain.

## Results

*Density of States Functions and Degeneracy of Ising and Potts*

The density of states functions of the Ising and Potts models show expected shape. One gets some few states at low temperature energy area, which grows strong up to very large values for the high temperature states. For checking the results of the Potts model simulation some energetic quantities have been calculated (fig. 8) and finite size effects are used to extrapolate $\beta_t$ (fig. 6). Finally, the ground state degeneracies are plotted, and show the expected quantities. The error bars for bigger systems decrease, because here less random walks through phase space are performed. While the number of Monte Carlo sweeps scales with $\mathcal{O}(L^2)$, $\tau$ scales with $\mathcal{O}(L^{2.17})$ (for Ising model [14]), so the error bars decrease for larger systems. This effect is amplified at the systems with phase transition of first order, as one can conclude from fig. 5.

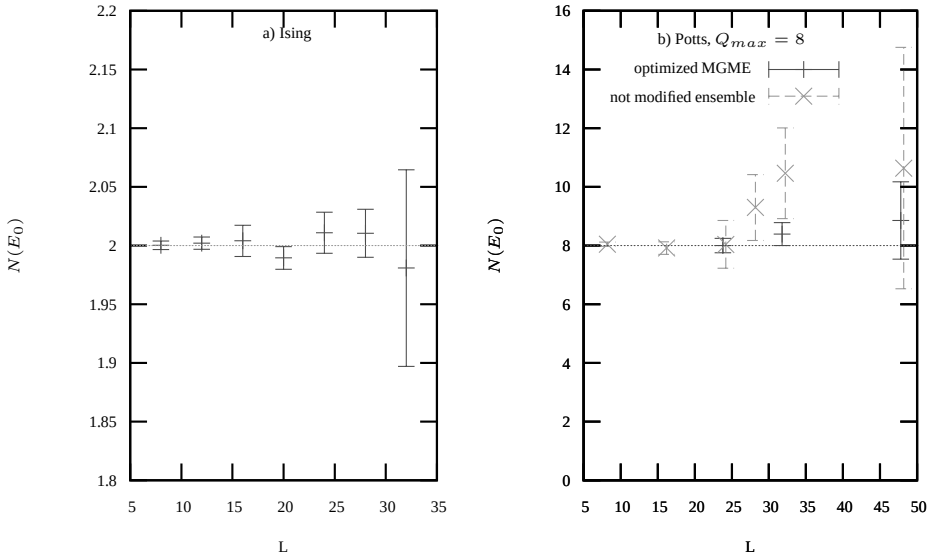



Figure 5: Plot of the ground state degeneracies of Ising (a) and Potts model ($q_{max} = 8$, b). For the Potts model modified and not modified ensembles are simulated calculated.

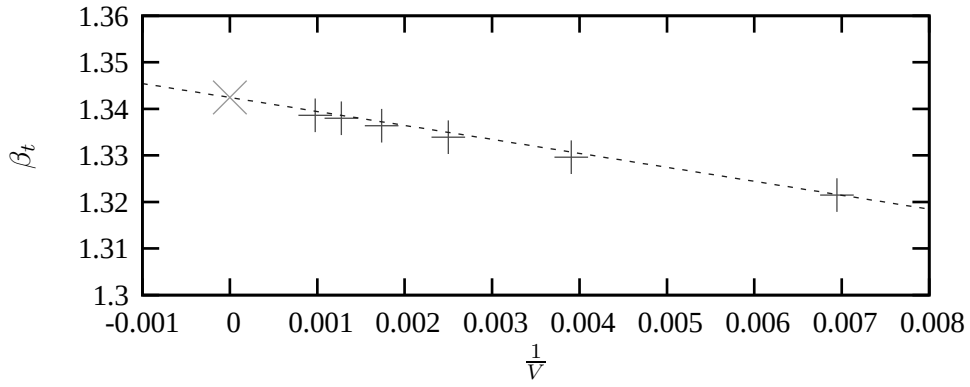



Figure 6: Recalculation of finite size scaling effects, what is done by setting $\frac{1}{V} = 0$. The red crosses are simulated quantities, taken manually from specific heat calculation (fig. 8), while the green cross is the analytic solution for $\beta_t$, taken from [9].

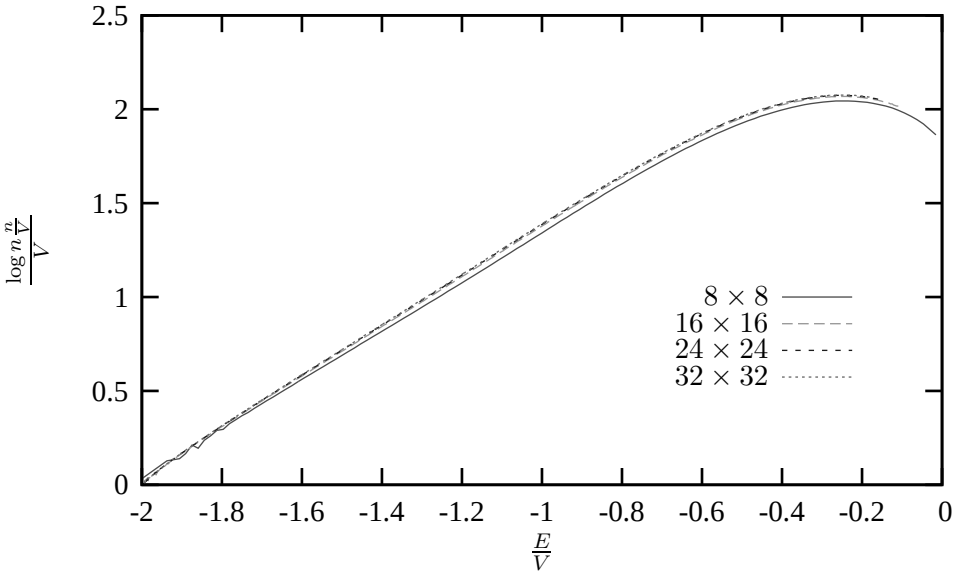



Figure 7: Logarithmic density of states function of Potts model for different system sizes, normed by the system size. With the system size $n(E)$ decreases rapidly.

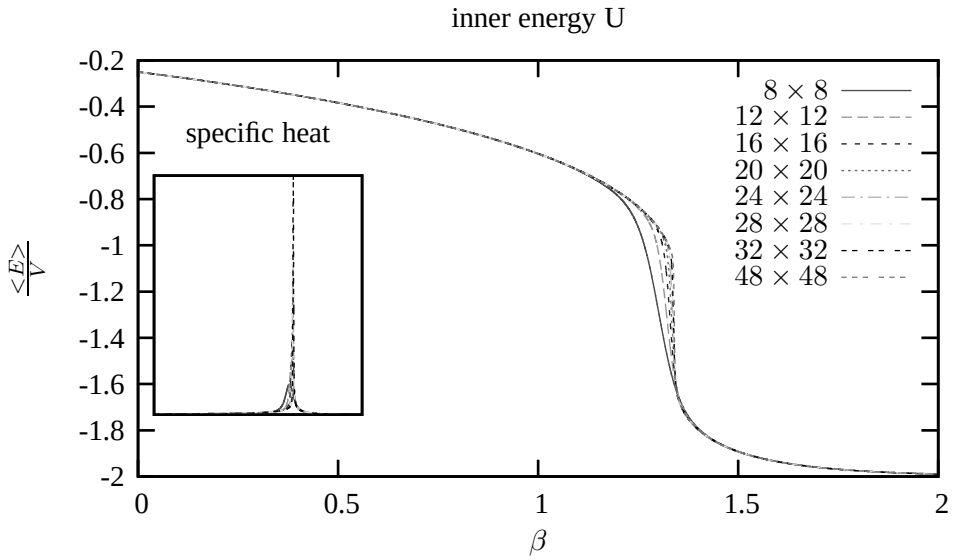



Figure 8: Plot of calculated energies $\frac{<E>}{V}$ (eq. 4) and calculated specific heat (eq. 5) from the density of states functions (fig. 7). The phase transition at $\beta_t \sim 1.342454$ is visible.

*Density of States Function and Degeneracy of the Chain Model*

After the positive results of the simulations of Ising and Potts Models, several simulations for chains of different length were performed. For every chain length a very symmetrical arrangement of the nodes on a triangular lattice appears, so a continuous computation of such arrangements appears meaningful to check the results. For the chain length $L = 6$ one expects a special behavior, because two dimensional chains with this length arranged on a triangular lattice show a high symmetry. This becomes clear, when one looks at the number of calculated states at tab. 2.

| system Length L | $E_0$ | calc. deg. | contacts without nn | cont. in LJ - min. | found gst. | simul. deg. |
|---|---|---|---|---|---|---|
| 2 | -10 | 2 | 1 | 1 | 2 | 2 |
| 3 | -21 | 6 | 3 | 2 | 6 | 6 |
| 4 | -32 | 20 | 6 | 3 | 20 | 19.93 |
| 5 | -44 | 28 | 10 | 4 | 28 | 27.84 |
| 6 | -66 | 20 | 15 | 6 | 20 | 10.61 |
| 7 | -77 | 156 | 21 | 7 | 156 | 3320 |
| 8 | -89 | 604 | 28 | 8 | 604 | 25738 |

Table 2: Some calculated quantities for chains with the boundary condition of a triangular lattice. At $L = 6$ symmetry of the system affects p.e. number of contacts in LJ - minimum and finally the simulated degeneracy. The data for "contacts in LJ minimum" and "found ground states" is taken from saved interaction maps (fig. 13, also special symmetry for $L = 6$ is visible here), "contacts without next neighbors" are calculated by iteration ($N_{L+1} = L + N_L$), simulated degeneracy is calculated by using eq. 14

.

The density of states functions for the chain model show for low chain lengths have a very peaked shape, what is well known from solid bodies. These peaks at certain energy values can be explained by folding mechanisms of the chains. For higher chain lengths this density of states function becomes continuously (fig. 10). Calculation of the degeneracy by the simple reweighting and normalization algorithm gives strange non integer numbers (fig. 11), which go through zero for larger lengths. Performing the calculation of an phase space factor (14) leads to in tab. 2 predicted integer values until chain length $L = 5$. Then the higher symmetry of the hexagon lets the approximation of $\xi_L$ break down. Simulating some larger values ($L = 7, L = 8$) gives no trivial correction for $\xi_L$, so here some further studies are required.

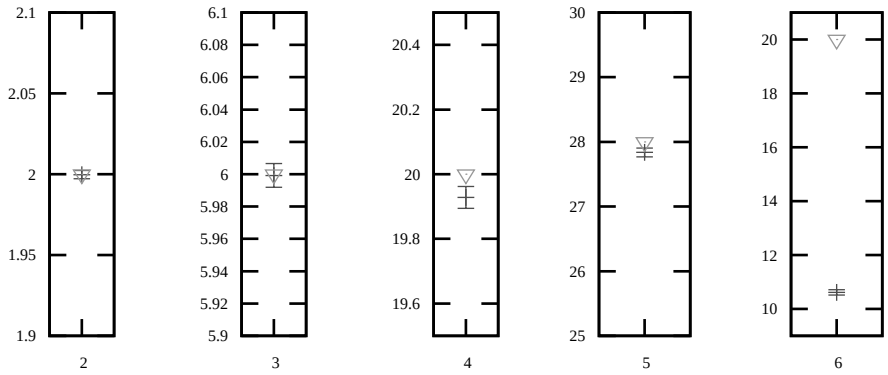



Figure 9: Plot of the ground state degeneracy after devision by $\xi_L$. The green triangles are calculated with triangular lattice boundary conditions. For $L < 6$ the results are correct.

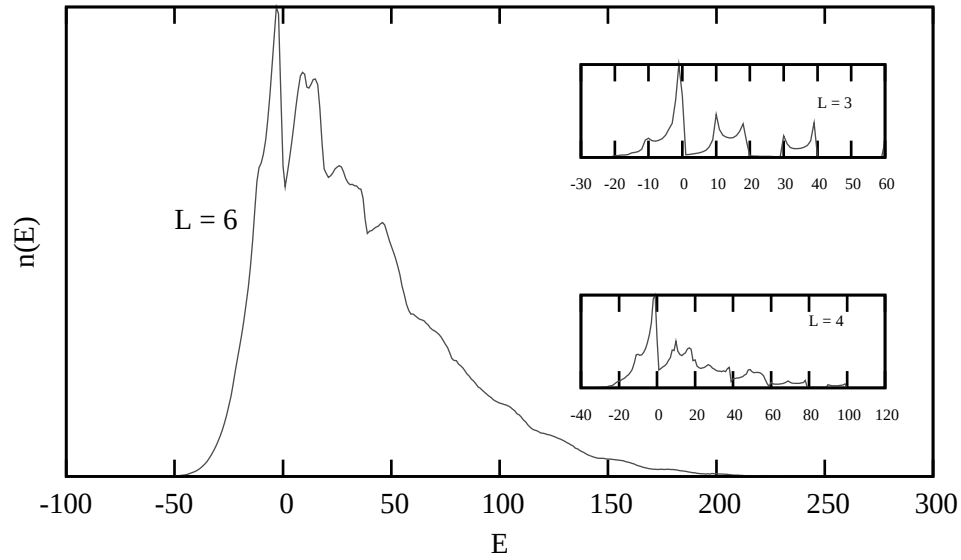



Figure 10: The density of state functions for different chain lengths. For larger length function looses discrete peaks, caused by folding mechanisms.

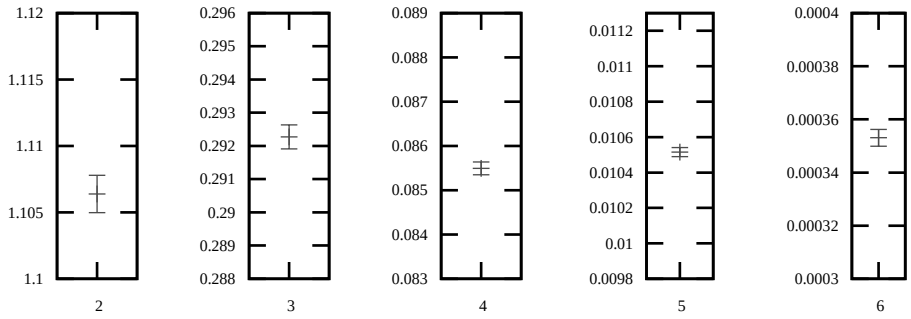



Figure 11: The ground state degeneracy without devision by phase space factor $\xi_L$. Values go through zero for larger length

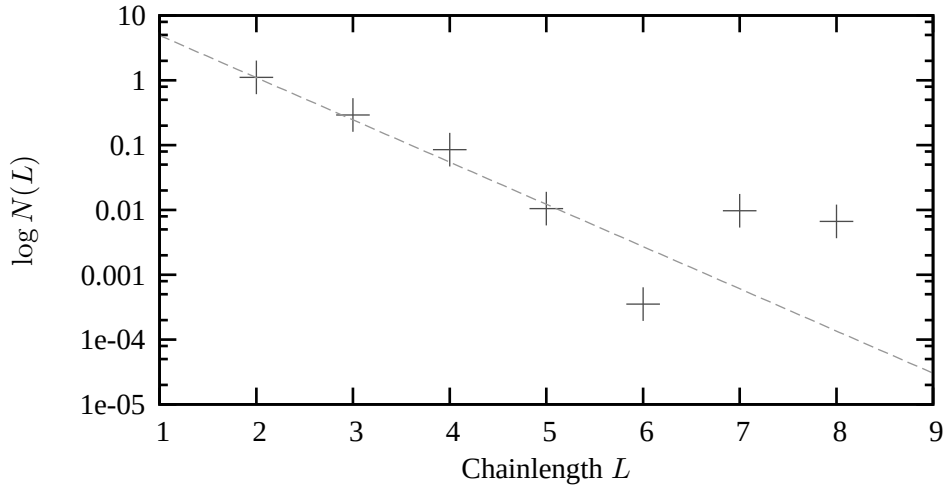
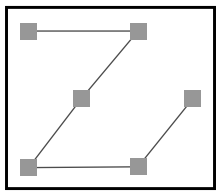



Figure 12: Plot of the ground state degeneracy logarithmic. At $L = 6$ the change in phase space is present.

In fig. 13 several types of ground states are plotted, which have been reached in simulations. From these images one can specify some from proteins known shapes: a, c and d can be classified as *sheets*, while b has a *helical* structure. It seems clear, that creating a sheet out of the helical configuration requires transition of high energy barriers, so here our Monte Carlo improvement has done good work.
Helical structure shows stripes in the interaction maps, which would be repeated for longer simulations. The sheet configuration shows some clusters of minima in the interaction maps. So you can judge about shape of the whole chain, just from studying the interactions maps. One interaction map describes two ground states: the shown one and a mirror symmetric with fixed first node - node connection.

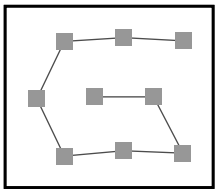
$L = 5, E = -44$



a)

$$\begin{pmatrix} -1 & \mathbf{-10} & 0 & -1 & NN & x \\ 0 & \mathbf{-10} & -1 & NN & x & NN \\ -1 & \mathbf{-10} & NN & x & NN & -1 \\ -10 & NN & x & NN & -1 & 0 \\ NN & x & NN & -10 & -10 & -10 \\ x & NN & -10 & -1 & 0 & -1 \end{pmatrix}$$

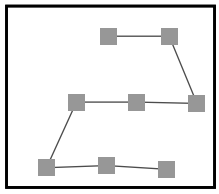
$L = 8, E = -89$



b)

$$\begin{pmatrix} -1 & -10 & -1 & 0 & 0 & 0 & 0 & NN & x \\ \mathbf{-10} & -10 & 0 & -1 & 0 & -1 & NN & x & NN \\ \mathbf{-10} & -1 & 0 & 0 & -1 & NN & x & NN & 0 \\ \mathbf{-10} & 0 & 0 & -1 & NN & x & NN & -1 & 0 \\ \mathbf{-10} & -1 & 0 & NN & x & NN & -1 & 0 & 0 \\ \mathbf{-10} & -10 & NN & x & NN & -1 & 0 & -1 & 0 \\ \mathbf{-10} & NN & x & NN & 0 & 0 & 0 & 0 & -1 \\ NN & x & NN & -10 & -1 & 0 & -1 & -10 & -10 \\ x & NN & -1 & -10 & -10 & -10 & -10 & -10 & -1 \end{pmatrix}$$

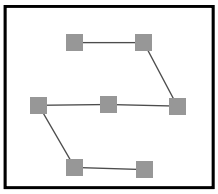
$L = 7, E = -77$



c)

$$\begin{pmatrix} 0 & -1 & \mathbf{-10} & \mathbf{-10} & -1 & 0 & NN & x \\ -1 & 0 & -1 & \mathbf{-10} & -10 & NN & x & NN \\ 0 & 0 & 0 & -1 & NN & x & NN & 0 \\ \mathbf{-10} & -1 & 0 & NN & x & NN & -10 & -1 \\ \mathbf{-10} & \mathbf{-10} & NN & x & NN & -1 & -10 & -10 \\ -1 & NN & x & NN & 0 & 0 & -1 & -10 \\ NN & x & NN & -10 & -1 & 0 & 0 & -1 \\ x & NN & -1 & -10 & -10 & 0 & -1 & 0 \end{pmatrix}$$

$L = 6, E = -66$



d)

$$\begin{pmatrix} 0 & -1 & \mathbf{-10} & \mathbf{-10} & -1 & NN & x \\ -1 & 0 & -1 & \mathbf{-10} & NN & x & NN \\ \mathbf{-10} & -1 & 0 & NN & x & NN & -1 \\ \mathbf{-10} & \mathbf{-10} & NN & x & NN & -10 & -10 \\ -1 & NN & x & NN & 0 & -1 & -10 \\ NN & x & NN & -10 & -1 & 0 & -1 \\ x & NN & -1 & -10 & -10 & -1 & 0 \end{pmatrix}$$

Figure 13: Some ground states for different chain lengths with their specific interaction maps. The triangular lattice is responsible for the geometry of the ground states. Shape specific potential terms are marked.

## Conclusion and Outlook

After the simulation of the Potts and the Ising model, the power of improved Monte Carlo methods like *modifying the Boltzmann weight* or *parallel tempering* becomes clear: At the same simulation time the error bars of the density of states functions are reduced by factor 3, for larger systems with rougher phase transitions even larger savings of time are conceivable. Furthermore the spin systems proofed the reweighting algorithm, which gives correct density of states functions.
The simulation of the chains showed this symmetric alignment on a lattice, what makes it possible to calculate some ground states without Monte Carlo by using these lattice boundary conditions. It is important to say, that this is only possible because the introduced chains have this unique interaction constant. Introduction of a second sort of nodes (dimer system) would break this symmetry.
The results of ground state determination with the approximation are surprisingly exact, until this breakdown at $L = 6$. One could interpret this as a phase transition, where crystallization takes place. It is interesting how the system behaves for larger length, and whether there will be a certain correction factor to multiply to the phase space factor, whenever a new symmetry is reached. Finally it is interesting whether and how this crystallization effects the macroscopic properties of the chain.
In the next step the system should be transformed into three dimensions and one should be able to examine different sorts of nodes (polymer system). This affords a large amount of computing power, so the algorithms to calculate the potential energy should be optimized.

## Acknowledgments

I want to thank Prof. Dr. Entel and Dr. Esser, who made the work and simulations on supercomputers possible for me. Special thanks go to Priv.-Doz. Dr. T. Neuhaus, my supervisor, who guided me through the world of Monte Carlo simulations in long discussions and taught me several tricks. Furthermore I want to thank Prof. Dr. U. H. E. Hansmann and his group, especially Dr. J. Meinke and Dr. S. Mohanty, for the very good integration in the CBB group at the Neumann Institute for Computing. I will always remember this pleasant time.

**Definition of Monte Carlo Quantities**

*Monte Carlo Sweep*

In order to improve efficiency of the algorithm, measurement is only done after each Monte Carlo sweep. During one Monte Carlo sweep every degree of freedom of the system should have been selected to flip once, so one Monte Carlo sweep lasts $V = L^d$ single spin flips.

*Parallel Tempering Swop*

The Monte Carlo steps between two parallel tempering updates are called one parallel tempering swop. In the simulation the number of Monte Carlo sweeps a swop contains is equal to the number of replicae simulated.

*Parallel Tempering Supersweep*

A supersweep contains a swop multiplied with the number of replicae. It is the minimum of simulation steps the system can reach, to pass from the lowest temperature to the greatest, so it is a suitable quantity to measure ergodicity time $\tau$.

**Calculating Errors by Jackknife Binning**

For error estimation with the jackknife method, one divides simulation time into $n$ parts. During the simulation, there will exist $n + 1$ bins of a measured quantity $c$: In bin 0 one calculates the average value of $c$ over the whole simulation. In the other bins $n$ additional average values of $c$ are measured, while in $c_1$ all $c$ - values are considered but these measured in period 1. So in $c_2$ all $c$ - values are considered but these in period two. This binning can be done one the fly. Finally the error can be calculated:

$$\sigma = \sqrt{\sum_{i=1}^{n} (c_i - c)^2}$$

This algorithm is suitable for not to huge bin - numbers $n$. Fortunately, usually small values like $n = 10$, as used in my simulations, give reliable results.

**Generating $\beta_i$ templates**

For generating useful $\beta_i$ templates and suitable Gaussian functions for a perfect simulation, it needs to perform an initial run with some trial templates to calculate an initial density of states function. Using such an initial density of states function one can calculate these single histograms, given in fig. 2, for any $\beta$ and possible modification:

$$\begin{aligned} P_\beta(E) &= n(E)\, e^{-\beta E} \\ P_{\beta_t, E_0}(E) &= n(E)\, e^{-\beta E} e^{-\left(\frac{E-E_0}{\Delta E}\right)^2} \end{aligned}$$

Performing a normalization of these functions by $\int_E dE P_\beta(E) = 1$ makes it possible to calculate an overlap $\psi$ between two functions $P_{\beta_i}$ and $P_{\beta_j}$. This quantity $\psi$ scales similar to the Metropolis quantity

$\omega$ (9), so one can write an algorithm which iterates some $\beta_i$ values to get suitable overlaps $\psi$. In systems with phase transition it is useful to calculate these $\beta_i$ values until the critical temperature $\beta_t$ is reached. Then one calculates the position of the second peak of critical histogram function (fig. 2) and creates few Gaussian functions between these peaks, as it is done in fig. 14. When the critical energy gap is crossed that way, one takes the next temperature without any Gaussian modification.

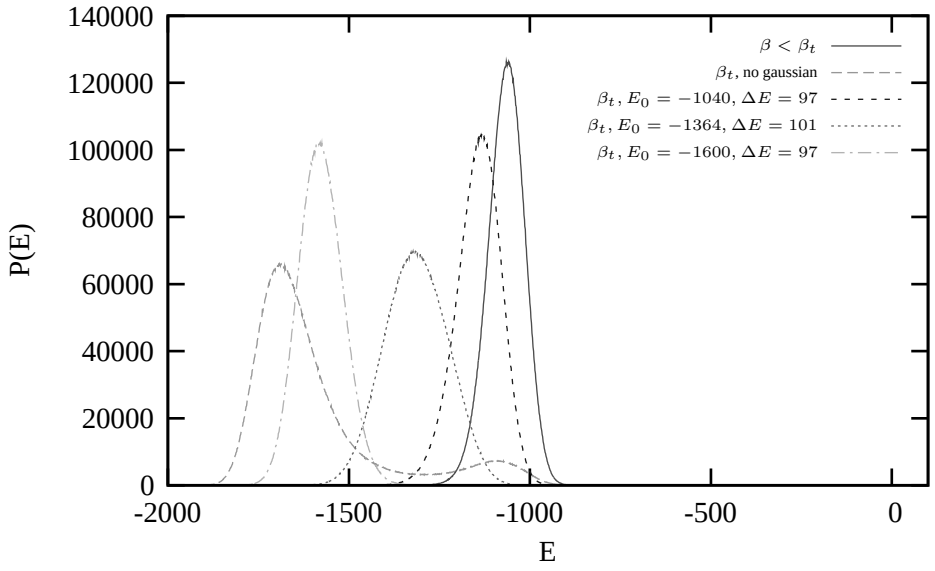



Figure 14: Plot of some histogram functions $P_\beta(E)$ calculated by using a simulated density of states function. Between two peaks of critical temperature $\beta_t$ few Gaussian functions are set so that an overlap criterium $\psi \sim 0.63$ is fulfilled.

## References


1. U.H.E. Hansmann and Y. Okamoto, Curr. Opin. Struc. Biol. 9 (1999)
2. B. A. Berg and T. Neuhaus, Phys. Lett. B **267**, 249 (1991)
3. B. A. Berg and T. Neuhaus, Phys. Rev. Lett. **68**, 9 (1992)
4. F. G. Wang and D. P. Landau, Phys. Rev. E **64**, 056101 (2001)
5. F. G. Wang and D. P. Landau, Phys. Rev. Lett. **86**, 2050 (2001)
6. T. Neuhaus, Phys. Review E **74**, 1 (2006)
7. R. B. Potts, Proc. Camb. Phil. Soc. **48** 106 (1952)
8. Lars Onsager, Phys. Rev. **65**, 117-149 (1944)
9. R. J. Baxter, Solid State Phys. **6** L445-L448 (1973)
10. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, H. Teller, E. Teller, J. Chem. Phys. **21(6)**, 1087 (1953)
11. E. Marinari and G. Parisi, Europhys. Lett. **19**, 451 (1992)
12. A. Ferrenberg, R. Swendsen, Phys. Rev. Lett. **61**, 23 (1988)
13. A. Ferrenberg, R. Swendsen, Phys. Rev. Lett. **63**, 12 (1989)
14. M.E.J. Newman & G. T. Barkema, *Monte Carlo Methods in Statistical Physics*, Oxford University Press (1999)

# Performance-Messungen eines parallelen Jacobi-Davidson Eigenwertlösers

Frank Schmidt

Technische Universität Chemnitz

Email: frank.schmidt@s2003.tu-chemnitz.de

**Zusammenfassung:** Das Jacobi-Davidson Verfahren wurde 1996 vorgestellt. Seither beschäftigt sich die Fachwelt mit Aussagen zur Stabilität und Effizienz des Verfahrens. In dieser Ausarbeitung wurden einige Untersuchungen zu verschiedenen Parametereinstellungen für einen bereits vorhandenen parallelen Jacobi-Davidson Löser durchgeführt und dessen Stabilität getestet.

## Das Jacobi-Davidson Verfahren

Im folgenden wird das Jacobi-Davidson Verfahren, welches von SLEIJPEN und VAN DER VORST (1996) erstmals beschrieben wurde, kurz erläutert. Hierbei beschränke ich mich auf die Bestimmung des größten Eigenwertes einer symmetrischen Matrix $A$. Gegeben ist also das folgende Eigenwertproblem:

$$Au = \lambda u \qquad \text{mit } A \in \mathbb{R}^{n \times n} \text{ symm.} \tag{1}$$

Gesucht ist der größte Eigenwert $\lambda_{max}$ und ein dazugehöriger Eigenvektor $u_{max}$ von $A$.

Die grundlegende Idee des Jacobi-Davidson Verfahren ist es, die Matrix $A$ in einen „kleinen“ Unterraum $\mathcal{V}$ zu projizieren, in diesem das Eigenwertproblem „direkt“ zu lösen und anschließend den Unterraum $\mathcal{V}$ geschickt zu erweitern. Dieser Prozess wird iteriert bis der Fehler klein genug ist. Die einzelnen Schritte werden zunächst detaillierter beschrieben.

### *Projektionsschritt*

Es sei $V = [v_1, \ldots, v_m] \in \mathbb{R}^{n \times m}$ orthonormal (d.h $V^T V = I_m$) und $\mathcal{V} = \operatorname{span}\{v_1, \ldots, v_m\}$. Die Projektion von $A$ „in $\mathcal{V}$“ ist gegeben durch

$$H = V^T A V. \tag{2}$$

Falls $\mathcal{V}$ einen Eigenvektor $u$ von $A$ (zum Eigenwert $\lambda$) enthält, so hat $H$ ebenfalls $\lambda$ als Eigenwert. Falls umgekehrt ($\theta$,$s$) ein Eigenpaar von $H$ ist, so ist ($\theta$,$u$) mit $u = Vs$ ein Ritzpaar von $A$ bzgl. $\mathcal{V}$ (d.h. $Au - \theta u \perp \mathcal{V}$ mit $u \in \mathcal{V}$).
Da die Projektion (bzw. Matrixmultiplikation) stetig ist, sind die Eigenwerte von $H$ Näherungen für die Eigenwerte von $A$, deren Eigenraum „nah an $\mathcal{V}$“ liegt. Ziel ist es also $u_{max}$ möglichst gut durch $\mathcal{V}$ anzunähern.
Der Projektionsschritt ist in verschiedenen Verfahren zur Eigenwertbestimmung zu finden. Er ist keine Neuerung des Jacobi-Davidson Verfahrens. Die eigentliche Innovation des Jacobi-Davidson Verfahrens ist die Wahl des Vektors $v_{m+1}$, um den der Unterraum $\mathcal{V}$ erweitert wird.

*Unterraumerweiterung*

Es seien $\theta$ bzw. $u \in \mathcal{V}$ (normiert) die aktuellen Näherungen für $\lambda_{max}$ bzw. $u_{max}$. Der Vektor $u_{max}$ wird zerlegt in

$$u_{max} = u + u^{\perp} \tag{3}$$

mit $u^{\perp} \in U^{\perp}$ und $U = \text{span}\{u\}$ (wobei $u_{max}$ dazu passend skaliert wird). Das Jacobi-Davidson Verfahren versucht den Vektor $u^{\perp}$ anzunähern und erweitert den Raum $\mathcal{V}$ um diese Näherung. Hierfür wird die Matrix $A$ wie folgt in den Raum $U^{\perp}$ projiziert:

$$B = (I - uu^T)A(I - uu^T). \tag{4}$$

Umstellen und Vereinfachen liefert die folgende Darstellung von $A$

$$A = B + Auu^T + uu^T A - \theta uu^T. \tag{5}$$

Einsetzen von (5) und (3) in (1) und mit Hilfe von $Bu = 0$ liefert

$$(B - \lambda_{max} I)u^{\perp} = -r + (\lambda_{max} - \theta - u^T A u^{\perp})u \tag{6}$$

mit

$$r = Au - \theta u.$$

Der Faktor vor $u$ muss verschwinden, da $(B - \lambda_{max} I)u^{\perp}$ und $r$ senkrecht zu $u$ sind. Damit gilt die folgende Gleichung:

$$(B - \lambda_{max} I)u^{\perp} = -r. \tag{7}$$

Da $\lambda_{max}$ nicht bekannt ist, wäre das Lösen dieser Gleichung ähnlich aufwändig wie das Lösen des Ausgangsproblems und kommt somit nicht in Frage. Allerdings ist $\theta$ „nah“ an $\lambda_{max}$ und damit kann die Gleichung (7) angenähert werden durch:[1]

$$(B - \theta I)u^{\perp} = -r. \tag{8}$$

Einsetzen von (4) in die letzte Gleichung und unter Verwendung von $(I - uu^T)u^{\perp} = u^{\perp}$ liefert die so genannte Korrekturgleichung

$$(I - uu^T)(A - \theta I)(I - uu^T)u^{\perp} = -r. \tag{9}$$

*Der Algorithmus*

In Algorithmus 1 ist das beschriebene Jacobi-Davidson Verfahren zum Bestimmen des größten Eigenwertes einer Matrix $A$ dargestellt. Zur direkten Implementierung ist der Algorithmus nur bedingt geeignet, da einige Matrixmultiplikationen eingespart werden können. Außerdem bleiben Methoden zum Lösen der Korrekturgleichung offen.

*Verallgemeinerungen*

In den angegebenen Literaturhinweisen ist das hier beschriebene Jacobi-Davidson Verfahren allgemeiner dargestellt. Es wird gezeigt, wie die größten bzw. kleinsten $p$ Eigenwerte einer Matrix bestimmt werden können. Mithilfe von harmonischen Ritzwerten ist es auch möglich, innere Eigenwerte mit dem Jacobi-Davidson Verfahren zu ermitteln. Auch die hier getroffene Einschränkung einer symmetrischen Matrix $A$ kann fallen gelassen werden. Es werden außerdem Hinweise zum Lösen der Korrekturgleichung gegeben.

[1] Falls $\theta$ nicht nah an $\lambda_{max}$ ist, so kann es effizienter sein, in Gleichung (8) $\theta$ durch einen festen Wert nahe $\lambda_{max}$ zu ersetzen.

ALGORITHMUS 1: JACOBI-DAVIDSON

1. **Start:** Wähle normierten Startvektor $v$, eine maximale Unterraumgröße $m_{max}$, setze $V := [v]$ und $m := 1$
2. **Iteration:** Solange bis Konvergenz eintritt,
   - berechne Projektion $H := V^T A V$
   - berechne größtes Eigenpaar ($\theta$,$s$) von $H$
   - setze Ritzvektor $u := Vs$ und Fehlervektor $r := Au - \theta u$
   - löse näherungsweise die Korrekturgleichung

     $$(I - uu^T)(A - \theta I)(I - uu^T)t = -r$$

     mit $t \perp u$
   - orthonormalisiere $t$ gegenüber $V$ und füge Ergebnis zu $V$ hinzu
   - $m \leftarrow m + 1$
   - Falls $m > m_{max}$, dann Restart mit $V := [u]$

**Testmatrizen**

*Selbsterzeugte Matrizen*

Mithilfe von Matlab wurden verschiedenartige „kleinere“ Matrizen generiert. Dabei wurden für die unsymmetrischen Matrizen zunächst zufällige reelle Diagonalmatrizen erzeugt und anschließend eine Basistransformation mit ebenfalls zufälligen Matrizen durchgeführt. Durch diese Vorgehensweise wird ein reelles Spektrum gewährleistet. Das Erstellen der symmetrischen Matrizen erfolgte direkt. Hierbei wurden vollbesetzte und strukturlose „dünnbesetzte“ (Dichte $= 0.1$) Matrizen erzeugt. Die Größe dieser Matrizen liegt zwischen $32 \times 32$ und $2048 \times 2048$.
Diese Matrizen spielen aufgrund ihrer kleinen Größe und Strukturlosigkeit in den durchgeführten Tests des Programms nur eine untergeordnete Rolle.

*Die bcsstk und bcsstm Matrizen*

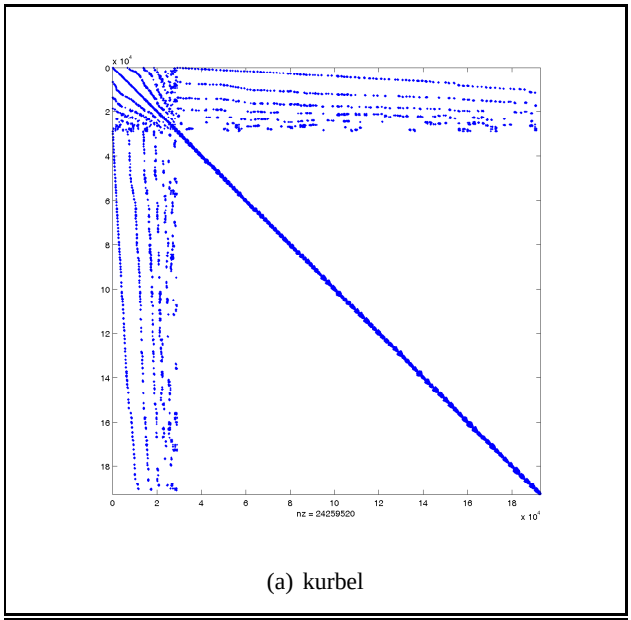
Bei den Matrizen *bcsstk1* bis *bcsstk13* und *bcsstm1* bis *bcsstm13* handelt es sich um Matrizen von Matrix Market [7]. Dies sind strukturierte, dünnbesetzte, reelle, symmetrische sowie (semi-)definite Matrizen mit Praxisbezug von 1982. Ihre Größe liegt zwischen $48 \times 48$ und $2003 \times 2003$. Genauere Informationen zur Matrixstruktur, wie zum Beispiel Strukturplot oder City-Plot, erhält man auf der Webseite von Matrix Market.

*Die kurbel, w124g und w124f Matrix*

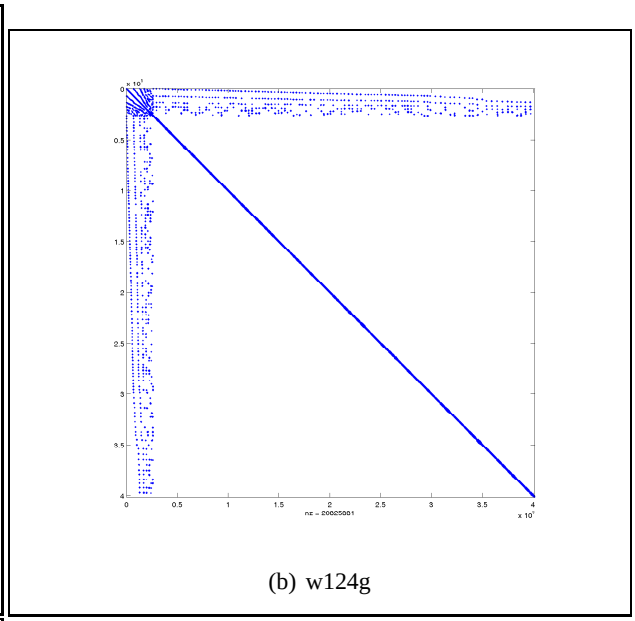
Die drei am ZAM erzeugten Matrizen *kurbel*, *w124g* und *w124f* beschreiben die Steifigkeit von einer Kurbel bzw. Autokarosse. Sie sind dünnbesetzt, reell und symmetrisch. Viele Tests wurden nur an diesen Matrizen durchgeführt, da sie eine große Dimension und Bezug zu praktischen Anwendungen besitzen. In Tabelle 1 ist die Dimension und die Anzahl der Einträge der Matrizen dargestellt. Abbildung 1 zeigt die Struktur der Matrizen.

| Matrix | Dimension | Einträge |
|---|---|---|
| kurbel | 192 858 | 24 259 520 |
| w124g | 401 595 | 20 825 881 |
| w124f | 1 310 622 | 68 828 510 |

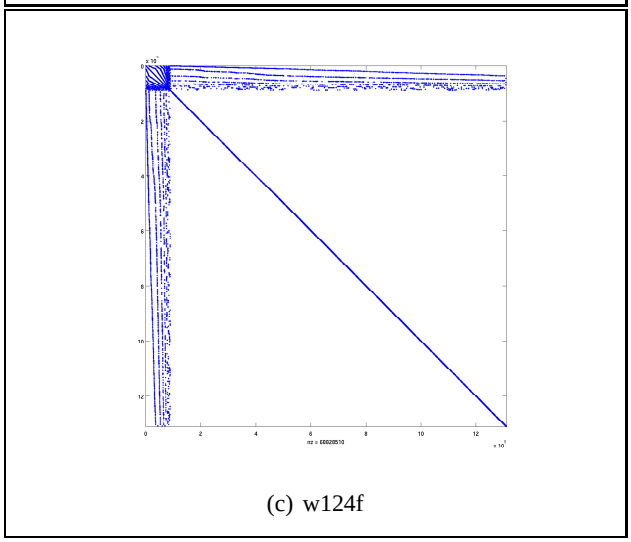
Tabelle 1: Matrixgröße/Matrixeinträge



(a) kurbel



(b) w124g



(c) w124f

Abbildung 1: Strukturplots

**Probleme**

*Die Zeit als Messgröße*

Im Laufe verschiedener Testläufe auf dem Supercomputer JUMP stellte es sich heraus, dass die Zeit kein gutes Maß für zuverlässige Geschwindigkeitsaussagen ist. Es gab Fälle, in denen die Zeit für den Jacobi-Davidson Algorithmus um mehr als Faktor 5 variierte. Hierfür kommen verschiedene Ursachen in Frage. Zum einen kann, bei kleineren Matrizen, das Betriebssystem die Zeitmessung verfälschen. Zum anderen ist bei großen Matrizen die restliche Belegung des entsprechenden Nodes durch andere Programme von

großer Bedeutung. So können sehr kommunikationsintensive Fremdprogramme die Laufzeit des Jacobi-Davidson Programms drastisch erhöhen.
In den durchgeführten Tests wurde deshalb häufig die Anzahl der benötigten Matrix-Vektor Multiplikationen (mit der Ausgangsmatrix $A$) betrachtet. Diese Messgröße ist relativ unabhängig vom Durchlauf und verhält sich bei großen Matrizen ähnlich wie die Zeit. Als Motivation hierfür soll Abbildung 2 dienen.

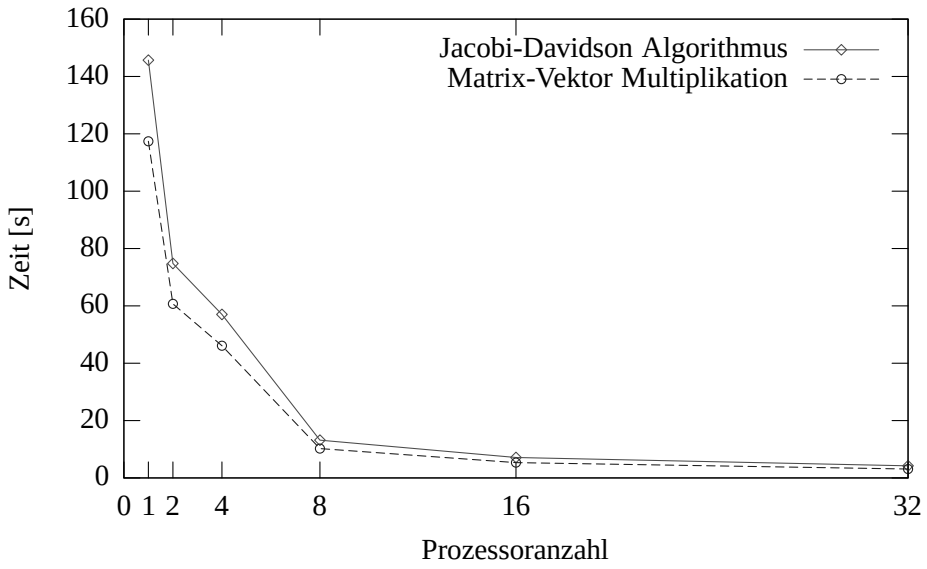



Abbildung 2: Zeitenverteilung für Matrix *kurbel* in Abhängigkeit von der Prozessoranzahl

*Schwankende Iterationszahl*

Beim Testen des Jacobi-Davidson Programms hat sich gezeigt, dass die benötigten Iterationszahlen sehr stark (bis zu Faktor 30) vom jeweiligen Durchlauf abhängen. Unter gewissen Voraussetzungen wurden sogar falsche Ergebnisse generiert. Das verwendete Programm war mit der Compileroption *-O3* (d.h. Optimierungsstufe 3) übersetzt. Es hat sich durch einen Zufall herausgestellt, dass mit Optimierungsstufe 0 die genannten Probleme nicht auftreten. Da auch Optimierungsstufe 2 zu keinen Problemen führte, wurde diese Stufe verwendet. Welche Optimierungsmechanismen des Compilers an dem beschriebenen Verhalten Schuld ist, wurde nicht genauer untersucht.

*Erratisches Verhalten*

Bei umfangreichen Untersuchungen zur optimalen Unterraumgröße der *kurbel* Matrix hatte der daraus entstandene Plot keine schöne „glatte/konvexe“ Form, wie es zu erwarten gewesen wäre. Er hatte an einigen Stellen unerwartet hohe Spitzen. In einem zweiten Testlauf mit den selben Parametern traten Spitzen an anderen Stellen auf. Es handelte sich also um einen ähnlich Effekt wie der im vorherigen Abschnitt.

Im Programm gibt es zwei Stellen, welche sich je nach Durchlauf unterschiedlich verhalten.

- Zum Programmstart wird der Parameter $\xi$ (siehe [3]) anhand einer Heuristik bestimmt. Die eingesetzte Heuristik arbeitet mit Zeitmessungen, welche vom Lauf abhängen. Der Parameter $\xi$ steuert die Verteilung der Matrix auf die Prozessoren. Da der TFQMR lokale Schätzer verwendet, sind auch dessen Ergebnisse vom Lauf abhängig.

- Das Programm benutzt an vielen Stellen den *MPI_Allreduce* Befehl. Im MPI Standard ist nicht festgelegt, in welcher Reihenfolge die jeweilige Operation ausgeführt wird. So kann es passieren, dass sich zum Beispiel die Summationsreihenfolge ändert und damit auch das Ergebnis kleine Unterschiede aufweist.

Der erste Punkt kann als Ursache ausgeschlossen werden, da in diesen Testläufen der Parameter $\xi$ fest eingestellt wurde und der QMR verwendet wurde. Die zufälligen Startvektoren sind nur „pseudo-Zufallsvektoren“. Sie werden mit keinem „Seek“ initialisiert und sind daher bei jedem Lauf gleich. Es sind also wahrscheinlich die unterschiedlichen Ergebnisse des *MPI_Allreduce* Befehls und ein anschließendes Aufschaukeln verantwortlich.

Ein häufiger Test war die Bestimmung der vier größten Eigenwerte der *kurbel* Matrix mit einer maximalen Unterraumgröße von $144$. Es wurden Zufallsstartvektoren und der QMR mit variabler Iterationszahländerung in Abhängigkeit vom vorherigen Residuum eingesetzt. Der Test lief mit einer Wallclock-Time von $30$ Minuten auf vier Prozessoren. Die Ergebnisse einiger Testläufe sind in Tabelle 2 dargestellt.

| **Nr.** | **benötigte Matrix-Vektor Multiplikationen** | **benötigte Zeit** |
|---|---|---|
| 1 | $473$ | 0,5min. |
| 2 | $935$ | 1min. |
| 3 | $22\,470$ | 22min. |
| 4 | keine Konvergenz in $30$min. | keine Konvergenz in $30$min. |

Tabelle 2: Ergebnisse des Testlaufs

Mit kleineren Abweichungen in den unterschiedlichen Läufen wurde aufgrund der parallelen Ausführung gerechnet. Jedoch sind derart große Schwankungen außergewöhnlich und unerwünscht. Um das Verhalten des Programms näher zu untersuchen, wurden zunächst die $16$ größten Eigenwerte bestimmt. Diese sind in Tabelle 3 dargestellt.

| **Nr.** | **Eigenwert** | **Norm des Residuums geteilt durch die Norm des Startresiduums** |
|---|---|---|
| 1 | 78337143.6792446673 | 0.389667356888467043E-12 |
| 2 | 68714018.2899170369 | 0.120569193223112707E-11 |
| 3 | 65997054.7506054416 | 0.306784070663745878E-11 |
| 4 | 65995584.0387703851 | 0.426916064756758607E-11 |
| 5 | 61429290.5423521921 | 0.915676303448734842E-11 |
| 6 | 60869221.3540580496 | 0.108509989335896413E-10 |
| 7 | 60863056.7173618898 | 0.958767339047622252E-11 |
| 8 | 59788202.1677016392 | 0.108326601231570041E-10 |
| 9 | 56391896.0051525608 | 0.123979869618634244E-11 |
| 10 | 55962628.6925787479 | 0.970153191182112792E-12 |
| 11 | 55608592.8164603859 | 0.302635171016304726E-11 |
| 12 | 53353903.1005171686 | 0.396838564987453000E-11 |
| 13 | 53353869.8730062991 | 0.800286891984117259E-11 |
| 14 | 53234587.7390047088 | 0.275365437006186683E-11 |
| 15 | 52191759.1899077818 | 0.327431405562694047E-11 |
| 16 | 52109179.7526207864 | 0.776088649112433149E-11 |

Tabelle 3: Die größten Eigenwerte der *kurbel*-Matrix

Auffällig ist, dass in den fehlgeschlagenen Läufen die Eigenwerte der projizierten Matrix[2] deutlich größer waren als die der *kurbel* Matrix. Falls die Projektionsmatrix[3] orthonormal ist, ist dies im „analytischen Sinn“ nicht möglich. Es liegt der Verdacht nahe, dass die Orthonormalisierung mittels Gram-Schmidt mögliche Entartungen nicht gut genug abfängt.

Die eingesetzte Gram-Schmidt Unterroutine wurde vor zehn Jahren entwickelt. Sie bricht kontrolliert ab, falls die Norm des zu orthonormalisierenden Vektors im Absoluten oder relativ zu seiner Ausgangsnorm zu klein wird. Es wird ein Restart durchgeführt, falls einer der beiden Fälle eintritt. Diese Gram-Schmidt Routine wurde für eine andere Maschine mit einer anderen Arithmetik entwickelt. Möglicherweise sind die eingestellten Konstanten zum Abfangen von Entartungen für die aktuelle Maschine *JUMP* ungeeignet. Eventuell kann man durch ein erneutes Ausrichten dieser Konstanten die Stabilität des Programms erhöhen. Womöglich werden dadurch die Probleme aber nur verschoben. Fraglich ist auch, ob es sinnvoll ist, abzubrechen wenn der Vektor relativ zu seiner Ausgangsnorm klein wird. Dies kann in anderen Anwendungen sinnvoll sein, schränkt hier aber eventuell ein.

Offensichtlich liegt der dritte und vierte Eigenwert der *kurbel*-Matrix nah beieinander. Ob dies zu den Problemen beiträgt ist unklar. Da auch der sechste und siebte Eigenwert nah beieinander liegen, wurde derselbe Test für die Suche nach den sieben größten Eigenwerten durchgeführt. In diesem Fall traten allerdings keine derartigen Schwankungen auf.

Bei der Suche nach den kleinsten und größten Eigenwerten ist die interne Sortierung der Eigenwerte die selbe (d.h. die kleinen Eigenwerte stehen am Beginn und die größeren hinten in der Sortierung). Das heißt aber auch, dass bei der Suche nach den kleinsten Eigenwerten die „guten“ am Anfang stehen, während die „guten“ bei der Suche nach den größten Eigenwerten am Ende sind. Nachdem die Korrekturgleichung für alle Eigenwertnäherungen gelöst wurde, werden die Lösungen in der entsprechenden Reihenfolge gegenüber dem alten Raum orthonormalisiert. Bei der Suche nach den größten Eigenwerten ist die Reihenfolge, in der die Vektoren orthonormalisiert werden, gerade anders herum als bei der Suche nach den kleinsten Eigenwerten. Wird beim Orthonormalisieren festgestellt, dass der entstehende Raum entartet ist, wird ein Restart durchgeführt, ohne dabei die noch verbleibenden Vektoren zu betrachten. Es hat sich als günstiger erwiesen, die Sortierung für die Suche nach den größten Eigenwerten umzudrehen. In diesem Fall trat der beschriebene Effekt nicht mehr auf. Wahrscheinlich wurde damit auch die Stabilität des Programms erhöht.

*Konvergenz bei inneren Eigenwerten*

Es gab einen Test zur Suche nach den sechs inneren (um $0$) Eigenwerten der *w124f* Matrix. Hierfür wurden zufällige Startvektoren, eine maximale Unterraumgröße von $30$, eine Genauigkeit von $10^{-4}$ sowie der QMR verwendet. Das Programm ist nach vier Stunden auf acht Prozessoren nicht terminiert. In Tabelle 4 sind die Eigenwertnäherungen der letzten Iterationen dargestellt, wobei die unterste Zeile der letzten Iteration entspricht.

| 1. Eigenwert | 2. Eigenwert | 3. Eigenwert | 4. Eigenwert | 5. Eigenwert | 6. Eigenwert |
|---|---|---|---|---|---|
| 0.884264527254549182 | 1.07083063998100614 | 2.39388793378719944 | 1.14063531914638094 | 19.0802816343215582 | 16.0234371920489060 |
| 0.884264527254549182 | 43.0181528726322497 | 2.39388793378719944 | 51.4488476600509301 | 58.1165816859718447 | 62.6144877367067068 |
| 0.884264527254549182 | 1.21804506749024366 | 2.39388793378719944 | 1.14259929730281362 | 19.1008916250217702 | 16.0755172113749225 |
| 0.884264527254549182 | 42.9096322875273515 | 2.39388793378719944 | 51.4007781376761343 | 58.0763761969845831 | 62.5662056114556648 |
| 0.884264527254549182 | 1.24096955945044507 | 2.39388793378719944 | 1.14978541731688577 | 19.1614087567927704 | 16.0326540927974861 |
| 0.884264527254549182 | 42.8691018400509662 | 2.39388793378719944 | 51.3345184934093695 | 58.0430838196949637 | 62.5360893554243660 |
| 0.884264527254549182 | 1.24974617370067032 | 2.39388793378719944 | 1.21731147804381035 | 19.8100549738468104 | 15.9886791273221451 |

Tabelle 4: Die letzten Iterationen bei der Suche nach den inneren Eigenwerten der *w124f*-Matrix

[2] In der Notation von oben ist dies die Matrix $H$.
[3] oben mit $V$ bezeichnet

Merkwürdig ist ein „alternieren“ beim zweiten, vierten, fünften und sechsten Eigenwert. Die Eigenwerte werden vom Programm der Größe nach sortiert. Ein anschließendes Berechnen des Rayleigh Quotienten bringt diese Sortierung durcheinander. Wahrscheinlich ist der Rayleigh Quotient an dieser Stelle nicht immer eine gute Wahl. Wann er eingesetzt werden sollte, wurde nicht untersucht.

Da die Matrix positiv definit ist, sollte die Suche nach den kleinsten Eigenwerten und die Suche nach den inneren Eigenwerten um $0$ die selben Ergebnisse liefern. Allerdings werden bei inneren Eigenwerten, im Gegensatz zu den kleinsten Eigenwerten, harmonische Ritzwerte verwendet. Die Suche nach den vier kleinsten Eigenwerten mit den selben Einstellungen wie oben lieferte das in Tabelle 5 dargestellte Ergebnis. Es ist zu beachten, dass die geringe Genauigkeit nur wenig signifikante Stellen liefert. Die Werte können daher nur der Orientierung dienen.

| Nr. | Eigenwert | Norm des Residuums geteilt durch die Norm des Startresiduums |
|---|---|---|
| 1 | 0.993083821401112 | 0.3955126062630147E-04 |
| 2 | 1.112595643284176 | 0.5015081663031635E-04 |
| 3 | 2.311282205472778 | 0.6491474776747729E-04 |
| 4 | 7.783607261108338 | 0.8848028197518182E-04 |

Tabelle 5: Die kleinsten Eigenwerte der *w124f*-Matrix

**Die Tests**

Das zu testende Programm *Jada15* verfügt über die Möglichkeit nach „beliebig vielen“ kleinsten, inneren oder größten Eigenwerten zu suchen. Voraussetzung hierfür ist, dass das Spektrum der Matrix reell ist. In der Datei *jada15.in* können eine Reihe von Eingabeparameter gesetzt werden. Es können u.a. die maximale Unterraumgröße, das Matrixformat, Methoden für Startvektoren sowie verschiedene Strategien zur Lösung der Korrekturgleichung eingestellt werden. Eine ausführliche Beschreibung des Programms und seiner Parameter ist in [2] und [3] zu finden.

*Wahl der Startvektoren*

Das Programm hält einen Parameter zur Wahl der Startvektoren bereit.[4] Es kann zwischen den Einheitsvektoren sowie Vektoren, welche Diagonalinformation oder orthogonale Teile der Diagonalen von $A$ ausnutzen, und pseudo-Zufallsvektoren gewählt werden. Um herauszufinden, welche Methode am geeignetesten ist, wurden für die *bcsstk* Matrizen die unterschiedlichen Methoden untersucht. Es wurde jeweils auf vier Prozessoren nach den vier größten Eigenwerten mit einer maximalen Unterraumgröße von $30$ und einer Wallclock-Time von $3$ Minuten gesucht. Als Löser wurde der QMR verwendet, wobei die Iterationszahl in Abhängigkeit vom vorherigen Residuum erhöht wird. Das Ergebnis dieses Tests ist in Abbildung $3$ dargestellt.

Die dritte Methode war in keinem der Versuche erfolgreich. Die Ursache ist vermutlich ein Fehler in der Implementierung, welcher nicht genauer untersucht wurde. An den Stellen, wo keine Markierungen für die Einheitsvektoren zu finden sind, ist das Verfahren nicht rechtzeitig terminiert. Die Methode der Einheitsvektoren kann derzeit nur verwendet werden, wenn die Anzahl der gesuchten Eigenwerte kleiner-gleich der Anzahl der eingesetzten Prozessoren ist.
Anhand der eben beschriebenen Nachteile und der Abbildung 3 empfiehlt es sich also die Diagonalinformation der Matrix auszunutzen oder mit Zufallsvektoren zu arbeiten.

[4] In der Notation von oben entspricht dies einer Startbasis von $\mathcal{V}$.

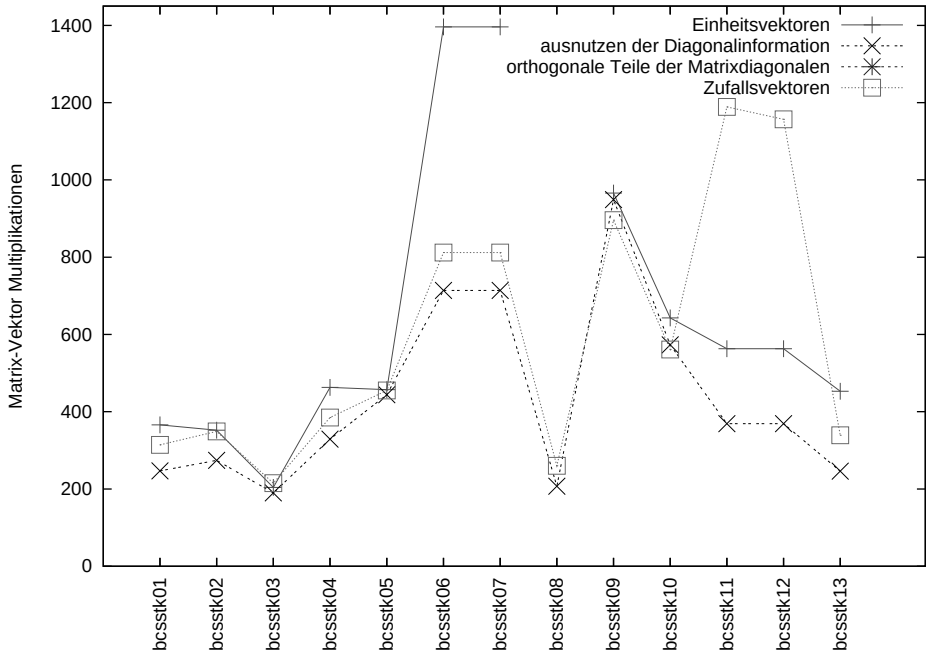



Abbildung 3: Anzahl der benötigten Matrix-Vektor Multiplikationen, abhängig von den Startvektoren

*Abhängigkeit von der Dimension*

In diesem Test wurde die Laufzeit (bzw. die benötigten Matrix-Vektor Multiplikationen) in Abhängigkeit von der Matrixdimension untersucht. Hierfür wurden die selbsterzeugten Matrizen verwendet, da sich diese Matrizen hauptsächlich in der Größe und nicht in ihrer Struktur unterscheiden. Es wurden jeweils die vier größten Eigenwerte auf einem Prozessor mit einer maximalen Unterraumgröße von 30 und einer Wallclock-Time von 30 Minuten gesucht. Als Löser wurde wieder der QMR verwendet. Das Ergebnis dieses Tests ist in Abbildung 4 zu sehen.

Verwendet man für diesen Test die *bcsstk* bzw. *bcsstm* Matrizen, so tritt ein „unstetiger“ Verlauf der Kurve auf. Es gibt nahezu gleichgroße Matrizen, welche in der Anzahl der benötigten Matrix-Vektor Multiplikationen sehr variierten. Ein Beispiel hierfür ist in Tabelle 6 dargestellt. Ursache hierfür, ist die

| **Matrix** | **Dimension** | **benötigte Matrix-Vektor Multiplikationen** |
|---|---|---|
| bcsstk08 | 1 074 | 260 |
| bcsstk09 | 1 083 | 913 |
| bcsstk10 | 1 086 | 561 |

Tabelle 6: Die benötigten Matrix-Vektor Multiplikationen für bcsstk08/09/10

unterschiedliche Struktur der Matrizen und die unterschiedliche Verteilung der Eigenwerte. Die Struktur der Matrizen ist in Abbildung 5 dargestellt.

Für unsymmetrische Matrizen konvergiert das Verfahren deutlich langsamer. In Tabelle 7 ist dies für die einzelnen Matrizen dargelegt.
Ein Grund ist, dass die Orthogonalprojektion schlechter geeignet ist, da sich die „Rechts-Eigenvektoren“ von den „Links-Eigenvektoren“ unterscheiden. Auch die von ihnen aufgespannten Räume sind nicht die selben. Dies reduziert die Konvergenzordnung.
Ein weiterer Grund könnte in der Matrizenerzeugung liegen. Beim Erstellen wurde eine Diagonalmatrix mit der Inversen einer Matrix multipliziert. Die Inverse war nicht exakt, dadurch können die Eigenwerte

der unsymmetrischen Testmatrizen einen kleinen imaginären Anteil besitzen. Der imaginäre Anteil eines Eigenwertes wird im Programm einfach ignoriert. Es muss den dadurch entstehenden Fehler durch eine höhere Genauigkeit im Realteil ausgleichen, wodurch die Konvergenz verlangsamt wird.

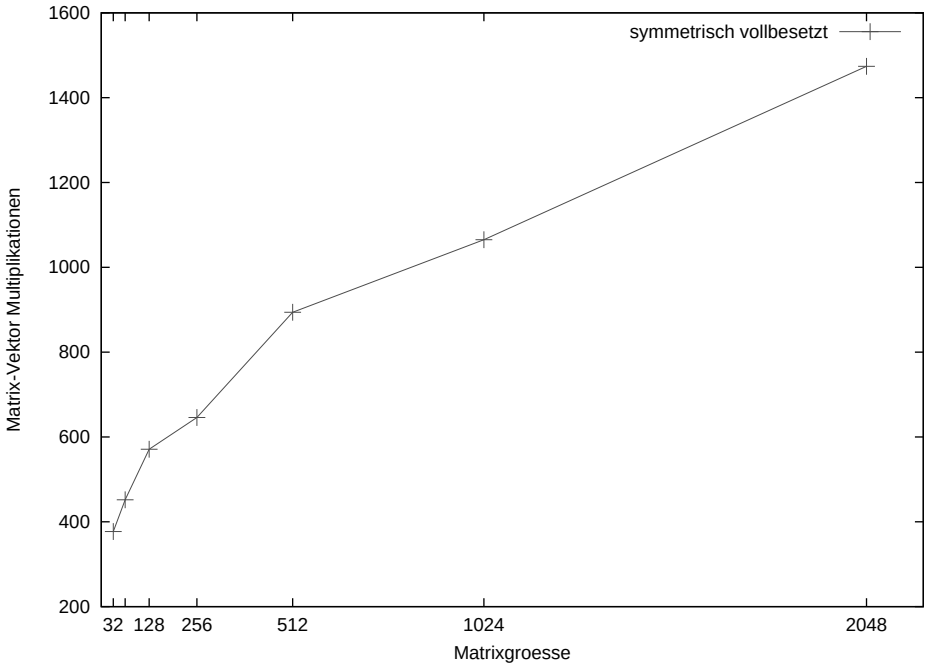
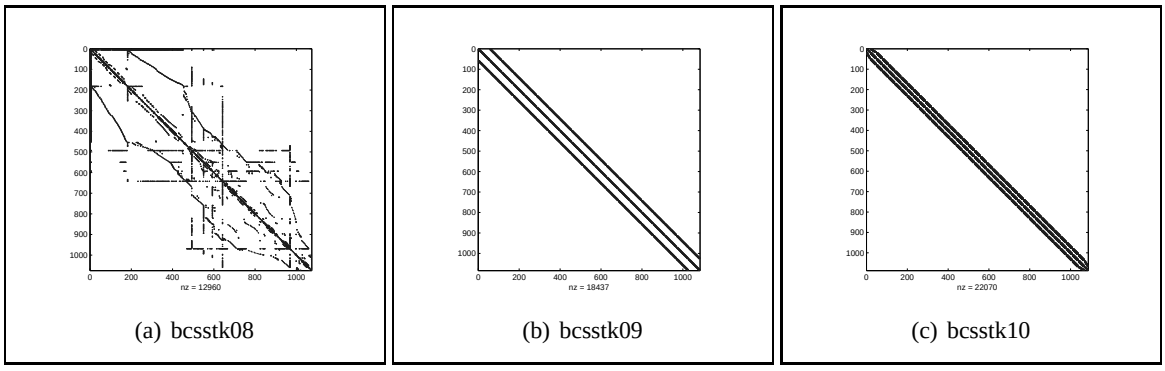



Abbildung 4: Anzahl der benötigten Matrix-Vektor Multiplikationen in Abhängigkeit von der Matrixdimension für symmetrisch vollbesetzte Matrizen



(a) bcsstk08 (b) bcsstk09 (c) bcsstk10

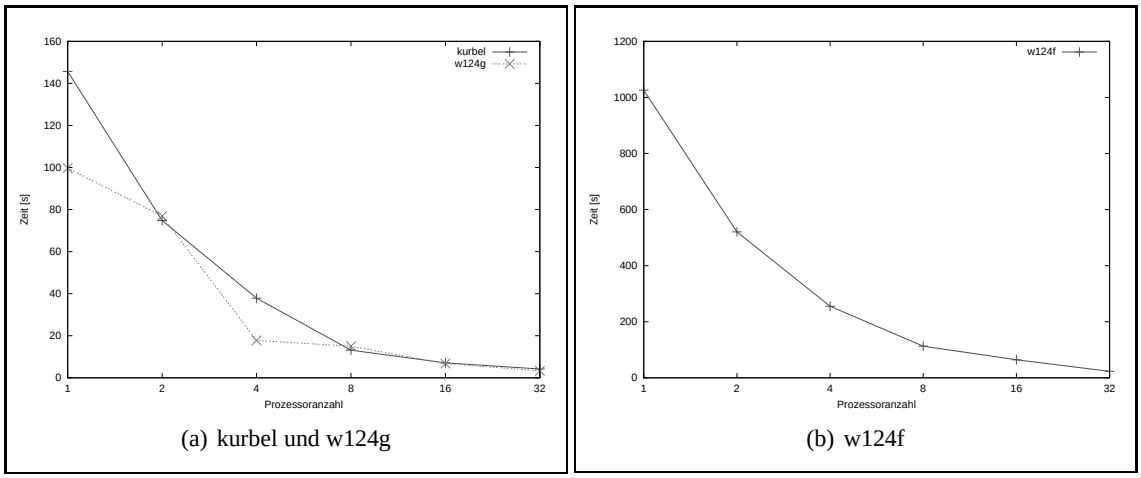
Abbildung 5: Strukturplots

| Matrix | Dimension | benötigte Matrix-Vektor Multiplikationen |
|---|---|---|
| unsym_1 | 32 | 540 |
| unsym_2 | 64 | 673 |
| unsym_3 | 128 | 30 902 |
| unsym_4 | 256 | keine Konvergenz in 30 min. |
| unsym_5 | 512 | keine Konvergenz in 30 min. |
| unsym_6 | 1 024 | 137 962 |

Tabelle 7: benötigte Matrix-Vektor Multiplikationen für unsymmetrische Matrizen

*Skalierbarkeit*

Um die Skalierbarkeit des Algorithmus zu testen, wurde für die drei großen Matrizen die Zeit der Jacobi-Davidson Unterroutine auf mehreren Prozessoren gemessen. In diesem Test musste auf die Zeitmessung zurückgegriffen werden, da die Anzahl der Matrix-Vektor Multiplikationen (fast) unabhängig von der Prozessoranzahl ist. Gesucht wurde wieder nach den vier größten Eigenwerten mit einer maximalen Unterraumgröße von 30. Als Löser wurde der QMR mit sechs Iterationen verwendet. Das Resultat dieses Testlaufs ist in Abbildung 6 dargestellt.




(a) kurbel und w124g (b) w124f

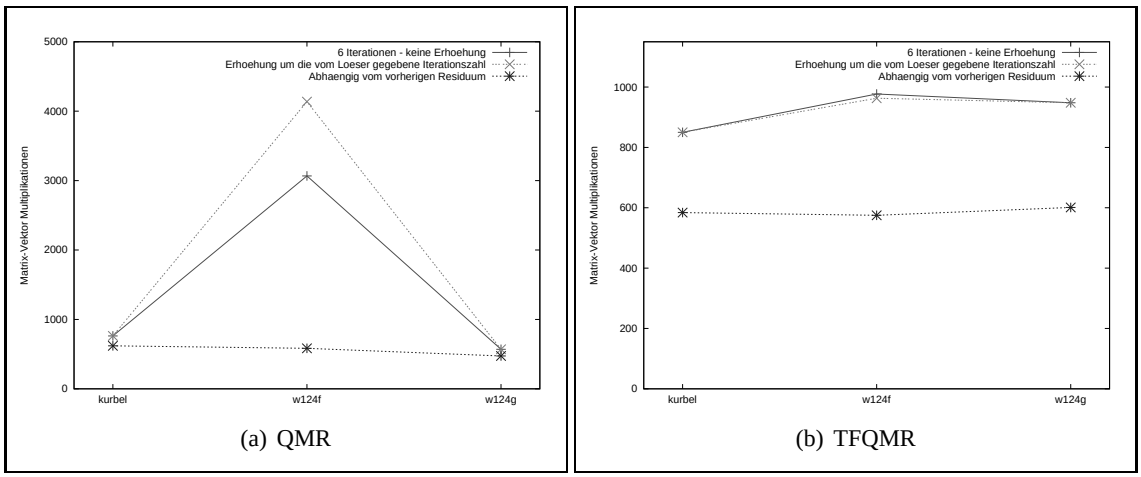
Abbildung 6: Skalierbarkeit

Die Skalierbarkeit des Programms hängt sehr stark von der Skalierbarkeit der Matrix-Vektor Multiplikation ab und diese wiederum von der Matrixstruktur sowie dem Matrixformat. Daher lassen sich hierzu nur schwer allgemeingültige Aussagen treffen.

*QMR / TFQMR*

Im Programm sind die zwei iterativen Lösungsverfahren QMR und TFQMR implementiert, wobei der QMR nur für symmetrischen Matrizen funktioniert. Es kann eingestellt werden, welcher Löser mit wievielen Iterationen verwendet werden soll. Hierbei ist es möglich die Iterationszahl fest einzustellen oder im Programmverlauf dynamisch regulieren zu lassen. Die Regulierung kann anhand einer vom Löser gegebenen Iterationszahl oder anhand des vorherigen Residuums erfolgen. Welche Methode am geeignetesten ist, wurde zunächst für die großen Matrizen *kurbel*, *w124g* und *w124f* untersucht. Es wurden auf je vier Prozessoren mit einer maximalen Unterraumgröße von 30 die vier größten Eigenwerte bestimmt. Die Ergebnisse sind in Abbildung 7 zu finden.

Wie aus Abbildung 7 hervorgeht, empfiehlt sich die Erhöhung der Iterationszahl in Abhängigkeit vom vorherigen Residuum. Ein direkter Vergleich zwischen QMR und TFQMR mit dieser Einstellung ist in Abbildung 8 dargestellt. Mit dieser Einstellung liegen der QMR und TFQMR etwa gleich auf.




(a) QMR (b) TFQMR

Abbildung 7: QMR bzw. TFQMR mit Iterationszahländerung

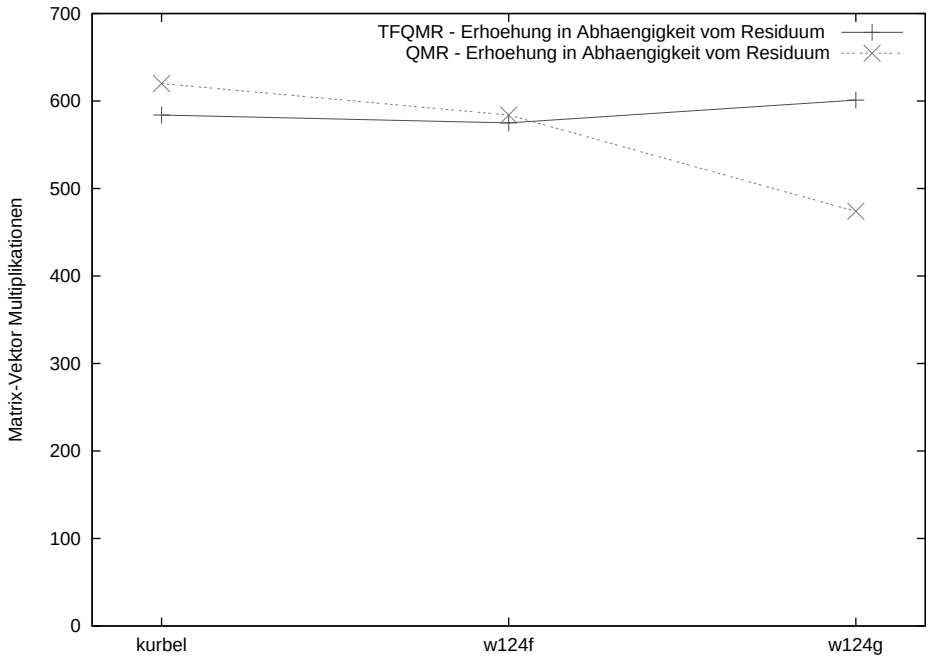



Abbildung 8: Anzahl der benötigten Matrix-Vektor Multiplikationen in Abhängigkeit vom Löser

In einem weiteren Test wurde untersucht ob sich die Erhöhung der Iterationszahl in Abhängigkeit vom Residuum auch bei den *bcsstk* bzw. *bcsstm* Matrizen rentiert. Das Ergebnis ist in Abbildung 9 wiedergegeben.

Es werden weniger Matrix-Vektor Multiplikationen benötigt als mit den anderen beiden Verfahren, jedoch sind die Unterschiede hier nicht so groß.

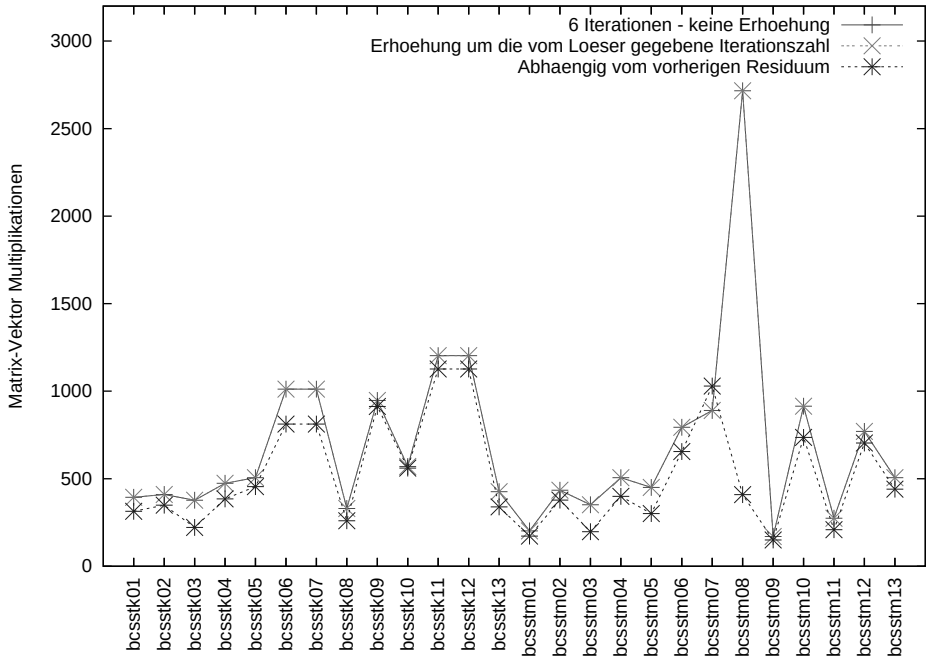



Abbildung 9: QMR mit Iterationszahländerung

### Schlusswort

Die durchgeführten Untersuchungen fanden alle an speziellen Matrizen statt. Allgemeine Aussagen sind daher kaum möglich. Es gab nur wenige Tests für unsymmetrische Matrizen, da diese meist kein reelles Spektrum haben.
Außerdem wurden nur wenige Tests für die Suche nach den kleinsten bzw. inneren Eigenwerten durchgeführt. Die ausgeführten Tests für die Suche nach den kleinsten Eigenwerten schienen sich jedoch ähnlich zu verhalten, wie die bei der Suche nach den größten Eigenwerten. Es sieht so aus, dass die Suche nach inneren Eigenwerten nicht problemlos funktioniert. Für genauere Untersuchungen war leider keine Zeit vorhanden.

### Literatur

1. G. L. G. SLEIJPEN AND H. A. VAN DER VORST, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401-425.
2. A. BASERMANN, *Iterative Verfahren für dünnbesetzte Matrizen zur Lösung technischer Probleme auf massiv-parallelen Systemen*, Jül-3015, Forschungszentrum Jülich GmbH (1995)
3. R. PUTTIN, *Modulare und parallele Implementierung des Jacobi-Davidson-Verfahrens*, Interner Bericht FZJ-ZAM-IB-2005-17, Forschungszentrum Jülich GmbH (2005)
4. B. JANSSEN, *Eine Entwicklungsumgebung für iterative Eigenwertverfahren in MATLAB*, Interner Bericht FZJ-ZAM-IB-2005-18, Forschungszentrum Jülich GmbH (2005)
5. PETER ARBENZ AND MICHIEL E. HOCHSTENBACH, *A Jacobi-Davidson Method for Solving Complex-Symmetric Eigenvalue Problems*, Preprint 1255, Department of Mathematics, Utrecht University, 2002
6. DAVID L. HARRAR II, *On the Davidson and Jacobi-Davidson Mathods for Large-Scale Eigenvalue Problems*, Centre for Mathematics and its Applications School of Mathematical Sciences Australian National University Canberra, ACT 0200
7. NATIONAL INSITUTE OF STANDARDS AND TECHNOLOGY, *Matrix Market*, http://math.nist.gov/MatrixMarket

# Parallele Performance der Matrix-Vektor-Multiplikation unter Einsatz verschiedener Speicherformate

Michèle Wandelt

Bergische Universität Wuppertal

E-mail: michelewandelt@web.de

**Zusammenfassung:** In diesem Artikel wird die parallele Performance der Matrix-Vektor-Multiplikation für dünnbesetzte Matrizen in verschiedene Speicherformaten untersucht. Dabei wurde so vorgegangen, dass die Matrix-Vektor-Multiplikation einem im ZAM entwickelten Programmpaket entnommen, angepasst und erweitert wurde. Dieses Programmpaket wurde im Rahmen einer Dissertation [1] in FORTRAN implementiert und im Verlaufe einer Diplomarbeit [2] erweitert.

## Einleitung

In vielen naturwissenschaftlichen Anwendungen treten Matrix-Vektor-Multiplikationen $A \cdot x$ mit dünnbesetzten Matrizen $A$ auf. Da dünnbesetzte Matrizen in verschiedenen Formaten gespeichert werden können und die Operation $A \cdot x$ üblicherweise einen Großteil der Rechenzeit beansprucht, wurden zwei verschiedene Speicherformate - CRS und JDS - hinsichtlich ihres Skalierungsverhaltens und ihres Speedups auf dem Parallelrechner JUMP (JUelich Multi Processor) untersucht.
In diesem Artikel werden zuerst die beiden Speicherformate beschrieben und die Algorithmen für die Matrix-Vektor-Multiplikation vorgestellt. Anschließend wird die Struktur der untersuchten Matrizen dargestellt und gezeigt, wie die Matrix und der Vektor bei parallelen Anwendungen auf die Prozessoren verteilt wird. Dann wird der benutzte parallele Algorithmus erläutert und die Ergebnisse der Messungen präsentiert. Die beiden Formate wurden zum einen ausgewählt, weil das CRS-Format bereits implementiert war und das JDS-Format in der Literatur für die untersuchten Strukturen als besonders gut beschrieben wird. Zum anderen sind es die Standardformate der Pakete 'SPARSEKIT' [7] und 'ITPACK' [8].

## Speicherformate

Für dünnbesetzte Matrizen stehen verschiedene Speicherformate zur Verfügung, die abhängig von der Struktur der Matrix unterschiedlich gut für die Matrix-Vektor-Multiplikation geeignet sind. Es wurden zwei Speicherformate untersucht, das zeilenweise gepackte Format und das verschobene Diagonalformat.

### *Zeilenweise gepacktes Format (CRS)*

Das zeilenweise gepackte Format CRS (Column Row Storage) speichert die Matrix zeilenweise ab. Dafür werde drei Felder benötigt: Ein Feld mit Werten, eines mit Spaltenindizes und eines mit Zeigern auf die Zeilenanfänge.

Die Werte der Matrix sowie die zugehörigen Spaltenindizes werden zeilenweise in den Feldern 'value' und 'column_index' abgelegt, wobei die Reihenfolge der Zeilenelemente nicht beibehalten werden muss. Besteht die Matrix aus $N$ Nicht-Null-Elementen, so haben diese beiden Felder die Länge $N$. Das Feld 'row_pointer' enthält im $i$-ten Eintrag einen Zeiger auf die Stelle in den Feldern 'value' und 'column_index', an denen das erste Element der $i$-ten Zeile abgelegt wurde. Es beginnt mit dem Wert 1 und enthält im letzten Eintrag die Anzahl der Nichtnullelemente plus Eins.
Sei $A \in \mathbb{R}^{n \times m}$ eine dünnbesetzte Matrix mit $N$ Nichtnullelementen. Dann wird sie im CRS-Format folgendermaßen in drei Felder gespeichert:

- INTEGER-Array 'row_pointer' der Länge $n + 1$
- INTEGER-Array 'column_index' der Länge $N$
- REAL-Array 'value' der Länge $N$

Es werden also insgesamt $2 \times N + n + 1$ Elemente gespeichert. Der Vorteil dieses Formates ist die Unabhängigkeit von der Struktur der Matrix; es ist das Basisformat für das in `FORTRAN` geschriebenen Programmpaket '`SPARSEKIT`', das für die Konvertierung dünnbesetzter Matrizen zu Austauschzwecken benutzt werden kann und elementare Funktionen zur Matrix-Manipulationen beinhaltet.

*Beispiel*

Das CRS-Speicherformat wird im folgenden Beispiel einer Matrix mit sechs Zeilen, sieben Spalten und 17 Nichtnullelementen dargestellt.

$$\begin{pmatrix} \mathbf{0.1} & \mathbf{0.2} & 0.0 & 0.0 & 0.0 & 0.0 & \mathbf{0.3} \\ 0.0 & \mathbf{0.4} & 0.0 & 0.0 & \mathbf{0.5} & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & \mathbf{0.6} & 0.0 & \mathbf{0.7} & 0.0 \\ 0.0 & \mathbf{0.8} & 0.0 & 0.0 & \mathbf{0.9} & 0.0 & \mathbf{1.0} \\ 0.0 & \mathbf{1.1} & 0.0 & \mathbf{1.2} & \mathbf{1.3} & \mathbf{1.4} & 0.0 \\ \mathbf{1.5} & 0.0 & 0.0 & \mathbf{1.6} & 0.0 & 0.0 & \mathbf{1.7} \end{pmatrix}$$

Abbildung 1: Beispiel einer dünnbesetzten Matrix

Die drei Nichtnullelemente der ersten Zeile der Matrix aus Abbildung 1 werden in dem Feld 'value' abgelegt, die zugehörigen Spaltenindizes in dem Feld 'column_index' (siehe Abbildung 2); der Zeiger 'row_pointer' verweist auf den Anfang der ersten Zeile wird auf den Wert '1' gesetzt. Da nun schon drei Elemente gespeichert sind, bekommt der nächste Eintrag des Feldes 'row_pointer' den Wert '4'. Die zwei Werte und Spaltenindizes der zweiten Zeile der Matrix werden in den nächsten beiden Spalten der Felder 'value' und 'column_index' abgelegt.
Sind alle 17 Elemente der Matrix gespeichert, so wird der Wert '$N + 1$' im $(n + 1)$-ten und letzten Eintrag des Feldes 'row_pointer' abgelegt.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| row pointer | 1 | 4 | 6 | 8 | 11 | 15 | 18 | | | | | | | | | | |
| value | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 |
| column index | 1 | 2 | 7 | 2 | 5 | 4 | 6 | 2 | 5 | 7 | 2 | 4 | 5 | 6 | 1 | 4 | 7 |

Abbildung 2: Speicherung der Matrix aus Abbildung 1 im CRS-Format

*Verschobenes Diagonalformat (JDS)*

Im verschobene Diagonalformat JDS (Jagged Diagonal Storage) werden die Nichtnulleinträge der Matrix wieder zeilenweise gespeichert. Dabei wird die maximale Anzahl '$r_{max}$' der Nichnullelemente pro Zeile ermittelt und sowohl die Werte als auch die Spaltenindizes in je einem zweidimensionalen Feld abgespeichert.
Sei $A \in \mathbb{R}^{n \times m}$ eine dünnbesetzte Matrix mit $N$ Nichtnullelementen und maximal $r_{max}$ Einträgen pro Zeile. Dann hat man im JDS-Format also zwei Felder der Größe $n \times r_{max}$:

- INTEGER-Array 'column_index'
- REAL-Array 'value'.

Dieses Format ist im Wesentlichen eine Erweiterung des CRS-Formats und besonders für Bandmatrizen mit größerer Bandbreite und einem nicht allzu hohen Besetztheitsgrad innerhalb der Bänder geeignet. Die Anzahl der Elemente pro Zeile sollte annähernd homogen verteilt sein, damit nicht allzu viele Nullen mit abgespeichert werden.
Bei der Konvertierung einer vollen Matrix in das JDS-Format werden die Zeilenelemente nach links verschoben. Dabei bleibt die Zeileninformation erhalten, die Spalteninformation geht allerdings verloren und muss extra abgelegt werden (siehe Abbildung 3). Im nächsten Beispiel wird die Speicherung der Matrix aus Abbildung 1 dargestellt.

Werte:

$$\begin{pmatrix} \mathbf{0.1} & \mathbf{0.2} & \mathbf{0.3} & 0.0 \\ \mathbf{0.4} & \mathbf{0.5} & 0.0 & 0.0 \\ \mathbf{0.6} & \mathbf{0.7} & 0.0 & 0.0 \\ \mathbf{0.8} & \mathbf{0.9} & \mathbf{1.0} & 0.0 \\ \mathbf{1.1} & \mathbf{1.2} & \mathbf{1.3} & \mathbf{1.4} \\ \mathbf{1.5} & \mathbf{1.6} & \mathbf{1.7} & 0.0 \end{pmatrix}$$

Spaltenindizes:

$$\begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{7} & 0 \\ \mathbf{2} & \mathbf{5} & 0 & 0 \\ \mathbf{4} & \mathbf{6} & 0 & 0 \\ \mathbf{2} & \mathbf{5} & \mathbf{7} & 0 \\ \mathbf{2} & \mathbf{4} & \mathbf{6} & \mathbf{7} \\ \mathbf{1} & \mathbf{4} & \mathbf{7} & 0 \end{pmatrix}$$

Abbildung 3: Speicherung der Matrix aus Abbildung 1 im JDS-Format

Das JDS-Format wird als Basisformat für das Paket ITPACK, einer in FORTRAN geschriebenen Bibliothek zur iterativen Lösung großer dünnbesetzter linearer Systeme, verwendet.

**Serielle Matrix-Vektor-Multiplikation**

Sei $A \in \mathbb{R}^{n \times m}$ eine Matrix mit $n$ Zeilen und $m$ Spalten und $x \in \mathbb{R}^m$ ein Vektor der Länge $m$. Für vollbesetzte Matrizen hat die Matrix-Vektor-Multiplikation $y = A \cdot x$ den Aufwand $\mathcal{O}(n \cdot m)$. Für dünnbesetzte Matrizen lässt sich dieser Aufwand durch Verwendung eines geeigneten Speicherformats und des passenden Algorithmus zur Matrix-Vektor-Multiplikation drastisch reduzieren.
Sei die Matrix $A \in \mathbb{R}^{n \times m}$ nun dünnbesetzt mit $N$ Nichtnull-Elementen und maximal $r_{max}$ Einträgen pro Zeile.

*CRS-Format*

Bei der Matrix-Vektor-Multiplikation im CRS-Format wird nicht die ganze Matrix mit dem Vektor multipliziert, sondern nur die Nicht-Null-Einträge. Aus diesem Grund verringert sich der Aufwand von $\mathcal{O}(n \cdot m)$ auf $\mathcal{O}(N)$, wobei für dünnbesetzte Matrizen $N << n \cdot m$ gilt. Algorithmus 1 zeigt die veränderte Matrix-Vektor-Multiplikation.

**Algorithm 1** Matrix-Vektor-Multiplikation bei Speicherung der Matrix im CRS-Format

**for** $i = 1, \ldots, n$ **do**
  y(i)=0
  **for** $j = row\_ptr(i), \ldots, row\_ptr(i+1) - 1$ **do**
    $y(i) = y(i) + value(j) * x(column\_index(j))$
  **end for**
**end for**

*JDS-Format*

Im JDS-Format werden abhängig von der Struktur der Matrix und der gespeicherten Nullen im Vergleich zur Matrix-Vektor-Multiplikation für volle Matrizen nur wenige Multiplikationen ausgeführt. Da Felder in Fortran spaltenweise im Speicher angeordnet werden, wird die Matrix hier im JDS-Format transponiert abgespeichert und die spaltenorientierte Matrix-Vektor-Multiplikation mit Aufwand $\mathcal{O}(n \cdot r_{max})$ benutzt. Für $r_{max} << m$ bedeutet dies einen erheblichen Zeitgewinn. In Algorithmus 2 ist die Multiplikation dargestellt.

**Algorithm 2** Spaltenorientierte Matrix-Vektor-Multiplikation bei Speicherung der Matrix im JDS-Format

**for** $i = 1, \ldots, n$ **do**
  y(i)=0
  **for** $j = 1, \ldots, r_{max}$ **do**
    $y(i) = y(i) + value(j, i) * x(column\_index(j, i))$
  **end for**
**end for**

**Parallele Matrix-Vektor-Multiplikation**

Bei den Tests wurde alleine die parallele Perfomance der Matrix-Vektor-Multiplikation untersucht. Dabei wird davon ausgegangen, dass die dünnbesetzte Matrix $A$ und der Vektor $x$ bereits so auf die Prozessoren verteilt sind, dass auf den einzelnen Prozessoren ungefähr gleich viele Nichtnullelemente $N$ vorliegen. Außerdem soll das Kommunikationsschema bereits berechnet sein. Sei $p$ die Anzahl der benutzten Prozessoren. Dann wird die Matrix zeilenweise auf die einzelnen Prozessoren verteilt, und zwar so, dass die ersten Prozessoren mindestens $\left\lceil \frac{N}{p} \right\rceil$ Elemente bekommen. Die Zeilen des Vektors werden nach dem selben Schema verteilt wie die der Matrix.

*Beispiel*

$$\left(\begin{array}{ccc|ccc|c} \mathbf{0.1} & \mathbf{0.2} & 0.0 & 0.0 & 0.0 & 0.0 & 0.3 \\ 0.0 & \mathbf{0.4} & 0.0 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.6 & 0.0 & 0.7 & 0.0 \\ \hline 0.0 & 0.8 & 0.0 & 0.0 & \mathbf{0.9} & 0.0 & 1.0 \\ 0.0 & 1.1 & 0.0 & \mathbf{1.2} & \mathbf{1.3} & 1.4 & 0.0 \\ \hline 1.5 & 0.0 & 0.0 & 1.6 & 0.0 & 0.0 & \mathbf{1.7} \\ 0.0 & 0.0 & 1.8 & 0.0 & 0.0 & 0.0 & 0.0 \end{array}\right) \left(\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ \hline x_4 \\ x_5 \\ \hline x_6 \\ x_7 \end{array}\right)$$

Abbildung 4: Beispiel Matrix-Vektor-Multiplikation

Angenommen, man möchte die in Abbildung 4 dargestellte Multiplikation auf drei Prozessoren parallel

ausführen lassen. Dann wird die Matrix so auf die einzelnen Prozessoren verteilt, dass auf den ersten beiden mindestens

$$\left\lceil \frac{N}{p} \right\rceil = \left\lceil \frac{18}{3} \right\rceil = 6$$

Elemente vorhanden sind und die restlichen auf dem letzten.
Die erste Zeile enthält drei Nichtnullelemente, die zweite zwei und die dritte ebenfalls zwei. Also bekommt der erste Prozessor die ersten drei Zeilen der Matrix und des Vektors zugewiesen und muß die Matrix-Vektor-Multiplikation für $3 + 2 + 2 = 7$ Elemente berechnen.
Einen Teil der Matrix-Vektor-Multiplikation kann direkt auf dem Prozessor berechnet werden, auf dem die Matrix-Elemente gespeichert sind, da der passende Teil des Vektors lokal vorliegt, für den restlichen Teil muß erst mit den anderen Prozessoren kommuniziert werden. Zur Verdeutlichung des lokalen Anteils der Matrix bei dieser Verteilung ist er in Abbildung 4 fett gedruckt.

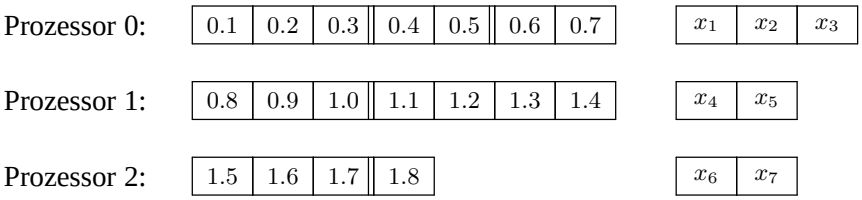



Abbildung 5: Verteilung der Elemente der Matrix-Vektor-Multiplikation der Abbildung 4

*Idee des verwendeten Algorithmus*

Da neben der Matrix auch der Vektor $x$ auf die einzelnen Prozessoren aufgeteilt ist, stehen nicht alle zur Multiplikation benötigten Daten des Vektors auf dem jeweiligen Prozessor direkt zur Verfügung. Im weiteren Verlauf wir die Multiplikation der Matrix-Elemente mit den auf dem Prozessor vorhandenen Vektordaten als "lokale Matrix-Vektor-Multiplikation" bezeichnet und die Multiplikation der Matrix-Elemente mit den auf anderen Prozessoren liegenden Daten als "nicht-lokale Matrix-Vektor-Multiplikation". Dabei geht man davon aus, dass der Großteil der zu berechnenden Daten lokal auf den einzelnen Prozessoren gespeichert ist und ein kleiner Anteil an Vektordaten zwischen den Prozessoren ausgetauscht werden muß.
Die Idee des Algorithmus ist nun, auf das Kommunikationsschema zurückzugreifen und zuerst die Kommunikation zwischen den Prozessoren zum Austausch der für die nicht-lokale Matrix-Vektor-Multiplikation benötigten Daten nichtblockierend anzustoßen. Währenddessen wird der lokale Anteil des Matrix-Vektor-Produktes berechnet. Danach wird über eine Schleife durch Benutzung des MPI-Kommandos `MPI_WAITSOME` der inzwischen übertragene Teil des Vektors zu dem nicht-lokalen Anteil des Matrix-Vektor-Produktes hinzugerechnet. Am Ende wird mit dem MPI-Befehl `MPI_WAITALL` die Kommunikation abgeschlossen und anschließend die lokal und nicht-lokal berechneten Ergebnisse zusammengefügt.
Unter der Annahme, daß der größte Teil der benötigten Vektordaten lokal vorhanden ist, geht bei der zeitaufwändigen Kommunikation keine Zeit verloren, was der große Vorteil dieses Algorithmus 3 ist.

**Algorithm 3** Idee zur parallelen Matrix-Vektor-Multiplikation

Sende die lokalen Daten nichtblockierend
Empfange die nichtlokalen Daten nichtblockierend
Berechne das lokale Matrix-Vektor-Produkt
Benutze die MPI-Routinen `MPI_WAITSOME` und `MPI_WAITALL`
    zur Erkennung der bereits empfangenen Daten
Berechne das nichtlokale Matrix-Vektor-Produkt
Füge die Werte zusammen

**Untersuchte Strukturen**

Es wurden zwei Arten von Matrizen untersucht. Zum einen wurden Tests mit bandstrukturierten Matrizen (wobei das Band wieder dünnbesetzt ist) und zum anderen mit Matrizen generellerer Struktur durchgeführt.

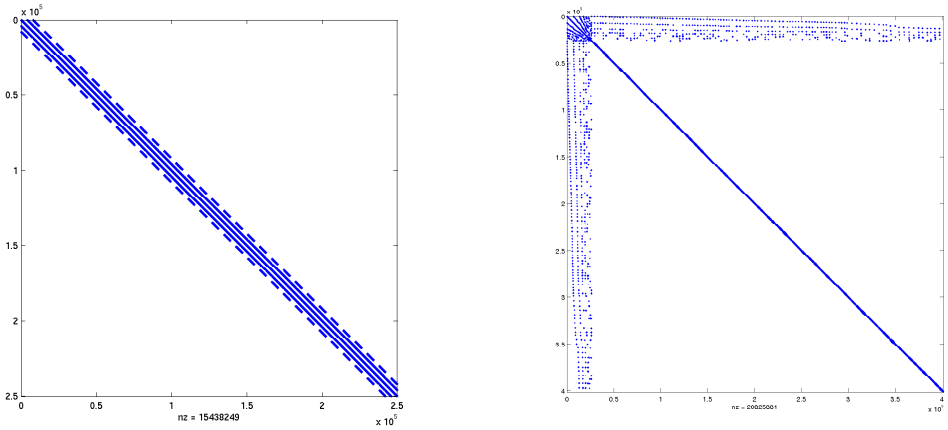



Abbildung 6: Struktur der untersuchten Matrizen

Dabei wurden die Matrizen, die sowohl einen bandstrukturierten als auch einen generellen Teil besitzen, noch einmal aufgeteilt und für die verschiedenen Strukturen getrennt untersucht.

*Bandstruktur*

Die bandstrukturierte Matrix ist aus einer FEM Diskretisierung eines Würfels mit 27 Elementen (HEX27) entstanden, wobei 99 Knoten in jeder Dimension vorlagen. Die Kenngrößen dieser Matrix sind in Tabelle 1 beschrieben.

| | |
|---|---|
| Größe im Speicher | 1 GB |
| Zeilenanzahl | 970299 |
| Nichtnullelemente | 60698457 |
| maximal Anzahl an Einträgen pro Zeile | 125 |
| Besetztheitsgrad | $\frac{60698457}{99^3 \cdot 99^3} \approx 0.006\%$ |

Tabelle 1: Kenngrößen der untersuchten FEM Matrix

*Matrizen mit unterschiedlichen Strukturanteilen*

Die untersuchten Matrizen mit verschiedenen Strukturanteilen beschreiben Steifigkeiten einer Autokarosserie beziehungsweise einer Kurbel. Die Kenngrößen dieser Matrizen sind in Tabelle 2 beschrieben. Dabei werden die Matrizen, die die Steifigkeit einer Autokarosserie beschreiben, mit `W124G` und `W124F` bezeichnet, wobei `G` für ein gröberes, `F` für ein feineres Gitter steht.

| Name | Kurbel | W124G | W124F |
|---|---|---|---|
| Größe im Speicher | 278 MB | 240 MB | 793 MB |
| Zeilenanzahl | 192858 | 401595 | 1310622 |
| Nichtnullelemente | 24259520 | 20825881 | 68828510 |
| maximal Anzahl an Einträgen pro Zeile | 459 | 132 | 132 |
| Besetztheitsgrad | $\approx 0.065\%$ | $\approx 0.013\%$ | $\approx 0.004\%$ |

Tabelle 2: Kenngrößen der untersuchten Steifigkeits-Matrizen

*Anteil der lokalen Matrix-Vektor-Multiplikation*

Bei der oben beschriebenen Verteilung der Daten auf die einzelnen Prozessoren ist der prozentuale Anteil der lokalen Elemente bei den untersuchten Matrizen je nach Prozessoranzahl ungleich groß. Bei der Matrix-Vektor-Multiplikation der bandstrukturierten Matrix FEM 99 werden die Daten so auf die Prozessoren verteilt, dass der lokale Anteil zunächst hoch bleibt und ab ungefähr 24 Prozessoren schnell sinkt (siehe Abbildung 7).

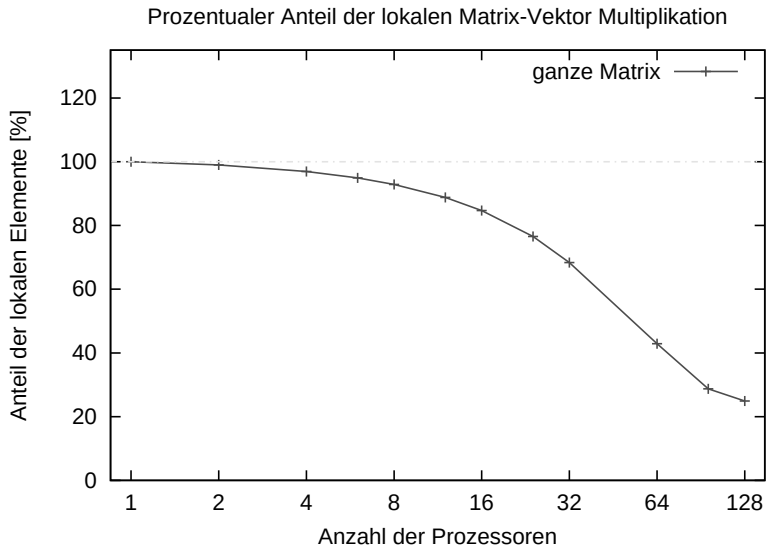



Abbildung 7: Prozentualer Anteil der lokalen Elemente der Matrix FEM 99

Der lokale Anteil der Matrizen mit verschiedenen Strukturanteilen sinkt schneller ab als der der bandstrukturierten, bleibt dafür aber dann stabil (siehe Abbildung 8). Diese Matrizen wurden unter anderem aufgeteilt in einen band- und einen unstrukturierten Teil und die benötigten Zeiten für die Matrix-Vektor-Multiplikation separat gemessen. Da die betrachteten Matrizen symmetrisch sind, wurde zur Aufteilung die obere Dreiecksmatrix betrachtet. Nachfolgend wird ein Kriterium zur Aufteilung beschrieben, das sich für die Aufteilung als praktikabel erwiesen hat. Als Kriterium für die Auswahl der bandstrukturierten Matrix wurde die Differenz der Spaltenindizes des ersten und letzten Elements einer Zeile gewählt. Zuerst wird dieser Wert auf die Anzahl der Zeilen gesetzt und in einer Variablen $k$ gespeichert. Verringert sich die Differenz der Spaltenindizes in einem Schleifendurchlauf über alle Zeilen im Vergleich zu dem Wert $k$ um den Faktor $0.5$, so wird der Startindex der bandstrukturierten Matrix neu auf die aktuelle Zeile gesetzt, $k$ bekommt den neuen (kleineren) Wert.

**Algorithm 4** Algorithmus zur Aufteilung der Matrizen in einen band- und unstrukturierten Anteil

Sei $A \in \mathbb{R}^{n \times n}$ eine symmetrische Matrix mit unterschiedlichen Strukturanteilen. Wähle nun die obere Dreiecksmatrix und speichere sie im CRS-Format.

```
k = n; wahl = 1
for i = 1, ..., n do
  hilf = column_index(row_pointer(i+1)-1) - column_index(row_pointer(i)-1
  if hilf ≤ 0.5· max_diff then
    k = hilf
    wahl = i
  end if
end for
```

**Messungen**

Die benutzten Programme wurden in FORTRAN geschrieben und unter Verwendung der Bibliothek MPI (Message Passing Interface) parallelisiert.
Es wurden einige Testläufe zur Messung der Dauer und des Speedups der Matrix-Vektor-Multiplikation durchgeführt. Der Speedup ist als Quotient aus der Ausführungszeit des gegebenen Algorithmus für einen Prozessor und der für $p$ Prozessoren definiert. Dabei wurde die nicht-lokale Matrix-Vektor-Multiplikation stets im CRS-Format berechnet, wärend für die lokale Matrix-Vektor-Multiplikation die Speicherformate CRS und JDS benutzt wurden. Außerdem wurde die Matrix-Vektor-Multiplikation für jedes Format einmal ohne Optimierung und einmal mit der Compileroption `-O3` getestet.
Unter Verwendung der MPI-Routine `MPI_WTIME` wurde die Zeit gemessen, die die reine Matrix-Vektor-Multiplikation auf JUMP (Juelich Multi Processor) benötigt. Dabei wurden zehn einzelne Matrix-Vektor-Multiplikation pro Messung durchgeführt und das Minimum als Referenzwert genommen, da die Messwerte geschwankt haben (siehe Abbildung 9 und 11). Nachfolgend sind die gemessenen Zeiten für verschiedene Prozessoranzahlen, Compileroptionen und Speicherformate dargestellt.

**Ergebnisse**

*Bandstrukturierte Matrix*

Bei den durchgeführten Messungen ohne Compileroptimierung skalieren das CRS-Format und das JDS-Format ungefähr gleich gut, wobei die Performance des CRS-Formats leicht besser abschneidet. Bei Verwendung von 128 Prozessoren fällt dies Format aus ungeklärten Gründen stark ab, so dass die hier benötigte Zeit mit der bei Benutzung von 64 Prozessoren vergleichbar ist. Der Speedup beider Formate ist sehr gut und liegt teilweise sogar über der Ideallinie, was darauf hindeutet, dass Probleme, die wahrscheinlich mit dem Zugriff auf den Cache zu tun haben, bei der seriellen Matrix-Vektor-Multiplikation auftreten.

Schaltet man die Compiler-Optimierung `-O3` ein, so skaliert das CRS-Format deutlich besser als das JDS-Format. Zudem fallen keine Probleme auf, wenn man die Matrix-Vektor-Multiplikation auf 128 Prozessoren berechnet. Das Speedup-Verhalten ist auch hier sehr gut; es müssen wieder Probleme bei der seriellen Matrix-Vektor-Multiplikation auftreten, da der Speedup auch hier zum Teil über der Ideallinie liegt.
Da die Matrix-Vektor-Multiplikation mit Compiler-Optimierung `-O3` um einen Faktor, der zwischen zwei und drei liegt, schneller ist, ist diese Option zu bevorzugen.

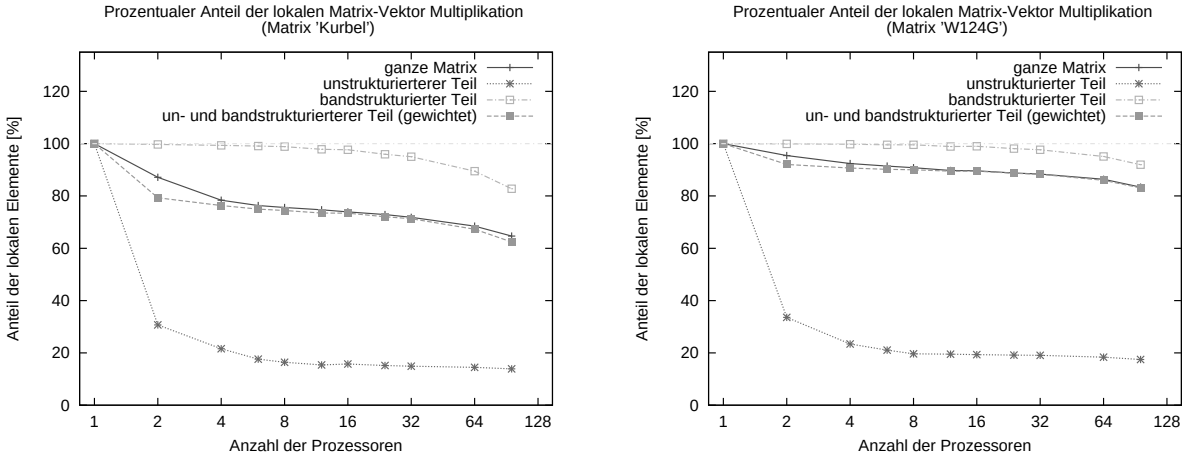



Abbildung 8: Prozentualer Anteil der lokalen Matrix-Vektor-Multiplikation bei den Matrizen zur Beschreibung der Steifigkeiten einer Kurbel (links) und einer Autokarosserie (rechts)

*Matrizen mit verschiedenen Strukturanteilen*

Bei der Matrix, die die Steifigkeit einer Kurbel beschreibt, sieht das Skalierungsverhalten ähnlich wie das der bandstrukturierten Matrizen aus. Die Aufteilung der Matrizen bringt gegenüber dem JDS-Format einen kleinen Zeitvorteil bei Einsatz weniger (bis zu acht) Prozessoren, das CRS-Format ist stets besser. Mit eingeschalteter Compiler-Optimierung `-O3` benötigt die Berechnung des Matrix-Vektor-Produktes der aufgeteilten Matrix bei mehr als 32 Prozessoren deutlich mehr Zeit als die nicht aufgeteilte. Außerdem skaliert auch hier das CRS-Format mit Compiler-Optimierung `-O3` viel besser als das JDS-Format. Insgesamt ist die Performance der Matrix-Vektor-Multiplikation auch hier stets gut (siehe Abbildung 11), der Speedup liegt wieder um die Ideal-Linie (siehe Abbildung 12). Das Skalierungsverhalten der Steifigkeits-Matrizen der Autokarosserie entspricht dem der Steifigkeits-Matrix der Kurbel.

## Diskussion und Ausblick

Nach den durchgeführten Tests lassen sich Matrix-Vektor-Multiplikationen für dünnbesetzte Matrizen sehr gut parallelisieren. Allerdings sinkt der Anteil der lokalen Matrix-Vektor-Multiplikation ab einer bestimmten Prozessoranzahl stark ab, so dass ein größerer Anteil der zeitaufwändigen Kommunikation notwendig wird.
Bemerkenswert ist, dass die Matrix-Vektor-Multiplikation bei Speicherung der Matrix im CRS-Format stets eine bessere Perfomance als bei Verwendung des JDS-Formats liefert, was sich allerdings mit den Ergebnissen von Kajiyama et al. (siehe [4]) deckt.
Um weitere Aussagen über die passendsten Speicherformate dünnbesetzter Matrizen mit verschiedenen Strukturen treffen zu können, müßte man mehr Tests mit anderen Matrizen und Speicherformaten durchführen.

## Danksagung

Ich möchte mich bei Dr. Bernhard Steffen für die gute Betreuung während des Praktikums bedanken. Außerdem danke ich Dr. Rüdiger Esser für die perfekte Organisation des Gaststudentenprogramms, Prof. Dr. Andreas Frommer für das Schreiben der Befürwortung, Oliver Bücker für die stete Hilfsbereitschaft und nicht zuletzt den anderen Teilnehmern des Gaststudentenprogramms für den regen Erfahrungsaustausch und die schöne Zeit.

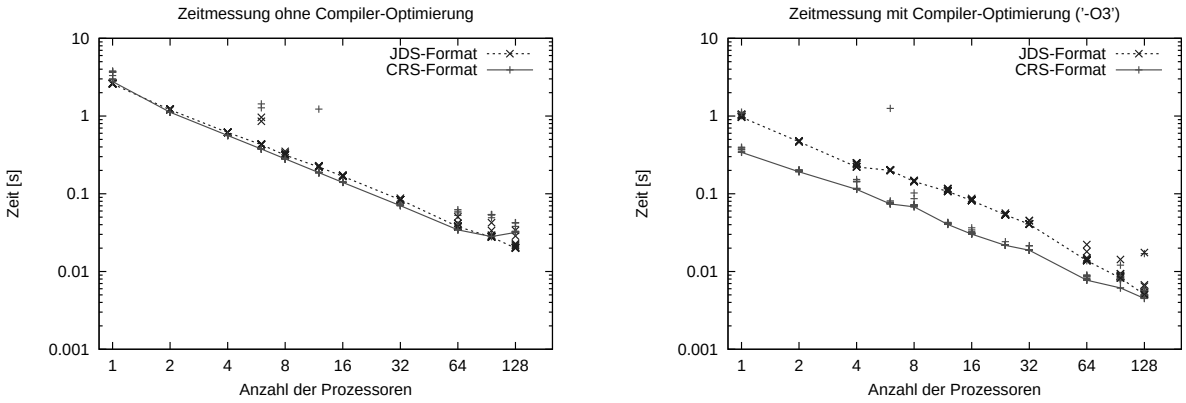



Abbildung 9: Skalierungsverhalten der getesteten Matrix FEM 99

## Literatur


1. A. Basermann,
   Iterative Verfahren für dünnbesetzte Matrizen zur Lösung technischer Probleme auf massiv-parallelen Systemen, Forschungszentrum Jülich, Bericht Jül-3015, Januar 1995, 151 Seiten
2. R. Puttin,
   Modulare und parallele Implementierung des Jacobi-Davidson-Verfahrens (2005), Interner Bericht FZJ-ZAM-IB-2005-17, Dezember 2005, 83 Seiten
3. R. Boisvert, R. Pozo and K. Remington,
   The Matrix Market Exchange Formats : Initial Design, National Institute of Standards and Technology Internal Report, NISTIR 5935, December 1996.
4. T. Kajiyama, A. Nukada, R. Suda, H. Hasegawa, and A. Nishida,
   A Performance Evaluation Model for the Matrix Computation Framework SILC, In Proceedings of IFIP International Conference on Network and Parallel Computing (NPC2006), IFIP, in press.
5. A. Frommer,
   Vorlesungsskript zu Algorithmen und Datenstrukturen II - Parallele Algorithmen,
   http://www.math.uni-wuppertal.de/ frommer/manuscripts/manuscriptsindex.html
6. Übersicht über schwachbesetzte lineare Systeme,
   http://www.dorn.org/uni/sls/index.html
7. Youcef Saad,
   Sparsekit: a basic tool kit for sparse matrix computations, Technical Report, Computer Science Department, University of Minnesota, June 1994.
8. D. Young and D. Kincaid.
   “The ITPACK Package for Large Sparse Linear Systems,ïn Elliptic Problem Solvers, (M. Schultz, ed.), Academic Press, New York, 1981, pp. 163-185.

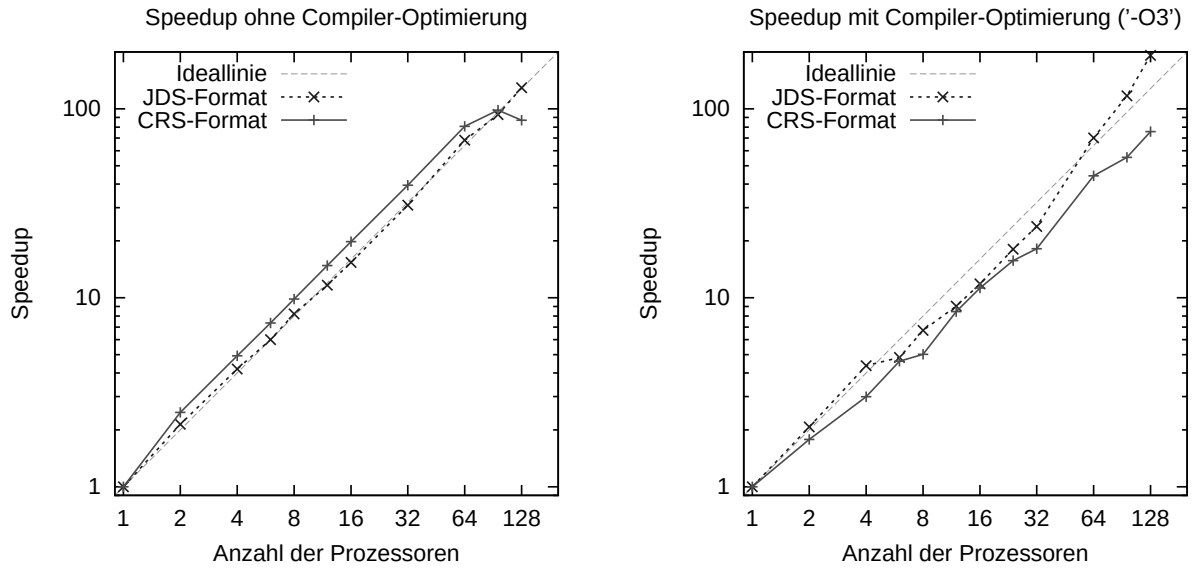



Abbildung 10: Speedup der getesteten Matrix FEM 99

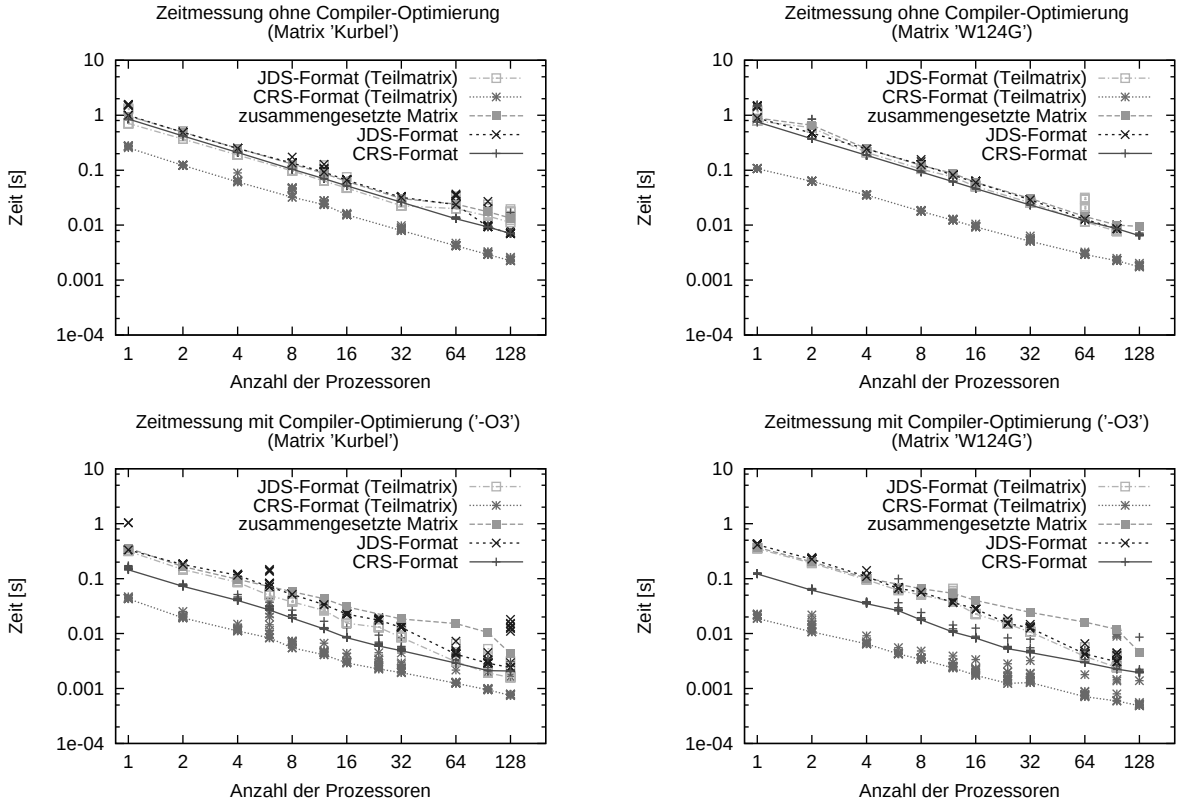



Abbildung 11: Skalierungsverhalten der Steifigkeits-Matrix einer Kurbel (links) und einer Autokarosserie (rechts)

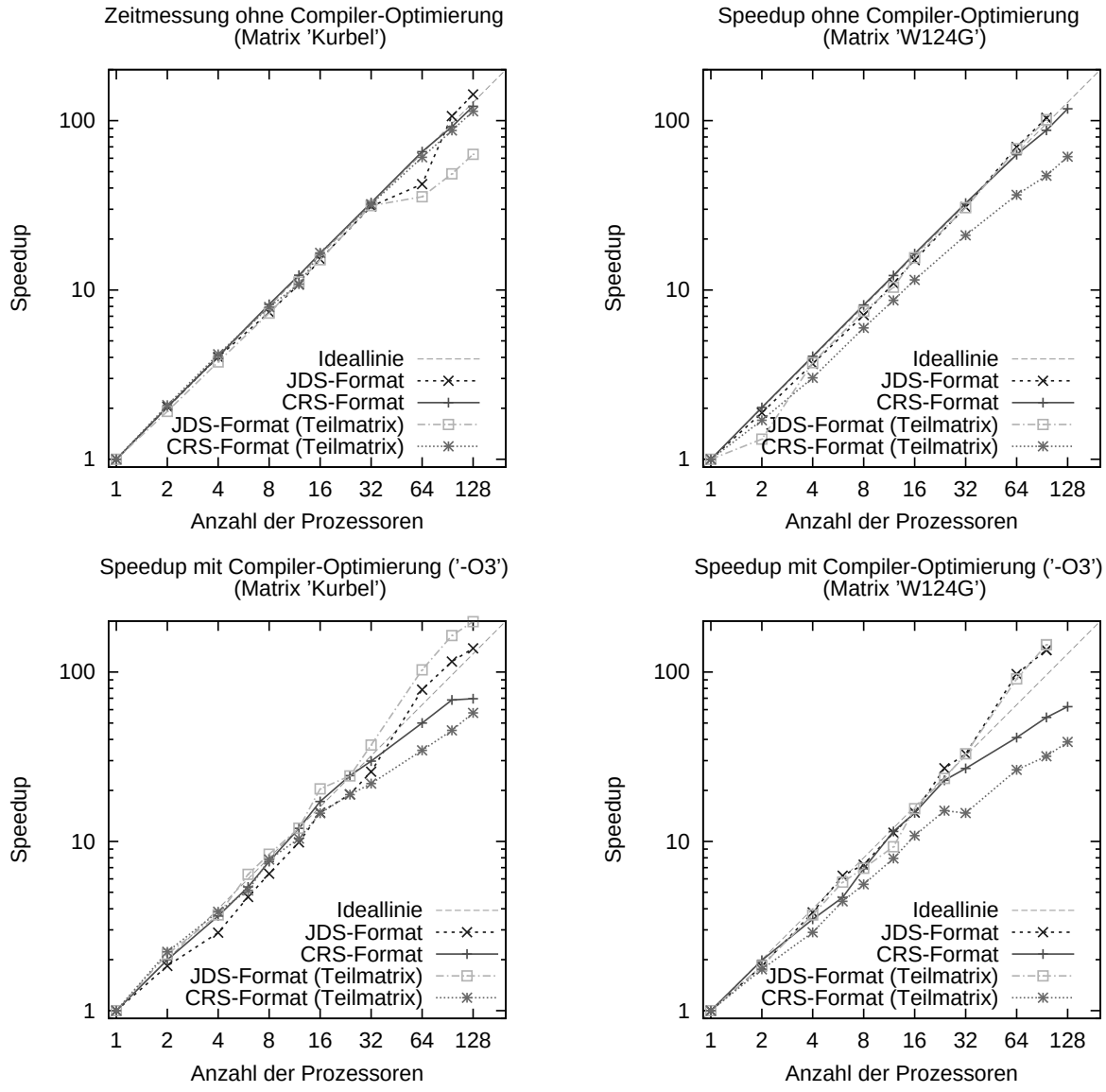



Abbildung 12: Speedup der Steifigkeits-Matrix einer Kurbel (links) und einer Autokarosserie (rechts)

# Rigide und nicht-rigide Bildregistrierung mit dem Insight Toolkit

Markus Weigel

Universität zu Lübeck

E-Mail: markus.weigel@informatik.uni-luebeck.de

**Zusammenfassung:**
Diese Ausarbeitung gibt eine Einführung in die rigide und nicht-rigide Bildregistrierung für medizinische Anwendungen und zeigt, wie diese im Insight Segmentation and Registration Toolkit (ITK) umgesetzt sind. Als spezielles nicht-rigides Verfahren wird der Algorithmus von Thirion vorgestellt und seine Umsetzung in ITK untersucht. Das Fernziel eines verallgemeinerten Registrierungsframeworks wird diskutiert.

## Einleitung

Bildregistrierung ist ein praktisches Problem aus der medizinischen Bildverarbeitung. Zu gegebenen Bilddaten soll eine Abbildung gefunden werden, welche die Bilder „möglichst gut“ zur Deckung bringt.

Der Anlass dafür kann vielfältig sein. So kann z. B. mit zwei verschiedenen bildgebenden Verfahren wie der Magnet-Resonanz-Tomographie (MRT) und der Positronen-Emissions-Tomographie (PET) je eine Aufnahme des Gehirns eines Patienten gemacht worden sein. Nun wäre es interessant, eine Bildfusion zu erstellen, um damit die anatomischen Merkmale (sichtbar im MRT-Bild) mit den Stoffwechsel-Informationen des PET-Bildes zusammen zu führen. Da sich der Patient zwischen den Aufnahmen bewegt (evtl. sogar in ein anderes Gerät überführt wird), kann nicht davon ausgegangen werden, dass die zwei Bilder ohne weitere Verarbeitung überblendet werden können. Hier wird Registrierung notwendig.

Ein anderes Beispiel sind histologische Serienschnitte eines Gehirns. Grob vereinfacht wird dabei z. B. einer toten Ratte das Gehirn entnommen und mit einem Spezialgerät in hauchdünne Scheiben zerschnitten. Diese werden eingescannt, und im Rechner soll daraus ein dreidimensionales Bild des Rattenhirns rekonstruiert werden. Das Problem ist, dass die Scheiben während der Verarbeitung verformt werden und dadurch nicht mehr genau aufeinander „passen“. Diese Verzerrung zu korrigieren ist ebenfalls eine Aufgabe der Bildregistrierung.

Weitere Anwendungen der Registrierung beinhalten die Korrektur von Artefakten, die durch Herzschlag und Atmung entstehen.

Bei der Registrierung ist im allgemeinen nicht klar, welche Art der Transformation zulässig ist, um das eine Bild (das *Template*, $T$) auf das andere (*Reference*, $R$) abzubilden. Sind z. B. Knochen auf dem Bild zu sehen, so dürfen diese durch die Transformation nicht verbogen werden. Diese Art der Registrierung nennt man *rigide*.

Sind hingegen auch andere Transformationen erlaubt, spricht man von *nicht-rigider* Registrierung. Beispielsweise wäre es möglich, jedem Pixel eines Bildes individuell eine neue Position zuzuordnen.

In diesem Bericht werden rigide und nicht-rigide Registrierung vorgestellt, und es wird auf ihre Implementation in ITK eingegangen, dem *Insight Segmentation and Registration Toolkit*.

**Mathematische Formulierung des Registrierungsproblems**

*Die allgemeine Problemstellung*

Registrierung ist ein Optimierungsproblem. Um das Registrierungsproblem formulieren zu können, muss zunächst klar sein, was eigentlich im mathematischen Sinne ein Bild ist. Ein Bild sei im folgenden eine Abbildung

$$T : \mathbb{R}^d \to \mathbb{R}$$

die jedem Pixel (Voxel, ...) $\mathbf{x} \in \mathbb{R}^d$ einen Grauwert $T(\mathbf{x})$ zuordnet. Dabei wird zusätzlich gefordert (vgl. [1]), dass $T$ kompakten Träger hat, dass $0 \leq T(x) < \infty$ für alle $x \in \mathbb{R}^d$ gilt, und dass $\int_{\mathbb{R}^d} T(\mathbf{x})^k dx$ endlich ist für $k > 0$.

Um zwei Bilder $R$, *Reference*, und $T$, *Template*, zu registrieren, wird eine zulässige Abbildung $\phi : \mathbb{R}^d \to \mathbb{R}^d$ gesucht, so dass

$$\mathcal{D}[R, T \circ \phi] = min.$$

Dabei ist $\mathcal{D}$ ein noch näher zu definierendes Ähnlichkeitsmaß.

Welche Klassen von Abbildungen $\phi$ zur Registrierung zulässig sind, hängt von der Art der Registrierung ab, die gewünscht ist, und von den Inhalten von $R$ und $T$. Hierin manifestiert sich auch der Unterschied zwischen rigider und nicht-rigider Registrierung.

*Rigide Registrierung*

Rigide Transformationen erlauben nur Verschiebung und Drehung von $T$. Daraus ergibt sich für $\phi$ folgende Form:

$$\phi(p) = Q\, p + \mathbf{b}$$

wobei $Q \in \mathbb{R}^{d \times d}$ orthogonal sei, $\det(Q) = 1$ und $\mathbf{b} \in \mathbb{R}^d$.

Im Fall $d = 2$ sieht $\phi$ also wie folgt aus:

$$\phi_{\theta, \mathbf{b_x}, \mathbf{b_y}} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \mathbf{b_x} \\ \mathbf{b_y} \end{pmatrix}$$

Die Transformation $\phi$ hat in diesem Fall also die drei Parameter $\theta$, $\mathbf{b_x}$ und $\mathbf{b_y}$, die optimiert werden können, um eine optimale Registrierung zu erhalten.

*Nicht-rigide Registrierung mit dem Algorithmus von Thirion*

Nicht-rigide Verfahren zur Bildregistrierung erlauben beliebige Verformungen von $T$. Die Abbildung $\phi$ hat dabei folgendes Aussehen:

$$\phi_{\mathbf{u}}(x) = x - \mathbf{u}(x)$$

Dabei ist $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^d$ ein Verschiebungsfeld (engl. *displacement field*), das für jeden Punkt angibt, wohin er verschoben wird.

Diese Form der Transformation und das sich daraus ergebende Optimierungsproblem ist schlecht gestellt, insbesondere ist die Lösung im Allgemeinen nicht eindeutig (vgl. [1]). Ein weiteres praktisches Problem

sind die sehr vielen Dimensionen des Parameterraums, die ein gewähltes Optimierungsverfahren sehr verlangsamen.

Trotz dieser Schwierigkeiten gibt es mathematische Überlegungen, wie auch dieses Problem als Optimierungsproblem behandelt werden kann (vgl. [1]). Diese sollen hier jedoch nicht vertieft werden.

Ein anderer Ansatz ein Verschiebungsfeld zu berechnen wurde von Thirion vorgeschlagen (vgl. im Folgenden [5, 4]). Die Vorstellung ist dabei, dass die Deformierung von $T$ ein zeitlicher Prozess ist.

$R$ und $T$ liegen eine Zeiteinheit voneinander entfernt, an den Enden des Intervalls $[0, 1]$. Damit kommt man zur Formulierung einer Intensitätsfunktion $i(x, y, t)$, die zu gegebenen Koordinaten $x$ und $y$ und einem Zeitpunkt $t$ den Grauwert eines Pixels liefert. Für $t = 0$ gilt dann $i(x, y, t) = R(x, y)$, für $t = 1$ gilt $i(x, y, t) = T(x, y)$.

Man setzt voraus, dass sich die Intensität eines Bildpunktes bei der Verschiebung zwischen $R$ und $T$ nicht verändert, also:

$$i\left(x(t), y(t), t\right) = \text{const.}$$

bzw.

$$\frac{d}{dt} i\left(x(t), y(t), t\right) = 0$$

Durch Umformungen und eine Approximation (vgl. [1, 5]) kommt man zu der folgenden optischen Flussgleichung:

$$\mathbf{v} \cdot \nabla R = T - R$$

Dabei bezeichnet $\mathbf{v}$ den gesuchten Vektor der Kräfte, die auf $T$ wirken, um es in $R$ zu überführen. Thirion schlägt folgende Lösung der Gleichung vor:

$$\mathbf{v} = \frac{(T - R) \cdot \nabla R}{(\nabla R)^T \nabla R}$$

Der erhaltene Vektor $\mathbf{v}$ lässt sich damit für gegebene Koordinaten einfach berechnen. Thirion schlägt außerdem weitere Varianten dieser Lösung vor, die die Lösung stabilisieren, wenn $\nabla R$ nahe Null ist.

Die Umrechnung des Kraftfeldes $\mathbf{v}$ in ein Verschiebungsfeld $\mathbf{u}$ wird bei Thirion nicht näher erläutert. Als einfachste Lösung erscheint es, $\mathbf{u} = \mathbf{v}$ zu setzen, so ist es auch in ITK umgesetzt.

Es wären aber auch andere Möglichkeiten denkbar, vor z. B. dann, wenn kein dichtes Kraftfeld berechnet wird, sondern die Kräfte z. B. nur an den Konturen in den Bildern $R$ und $T$ wirken. Die Konturen müssten dann allerdings vorher durch einen entsprechenden Algorithmus erkannt werden.

*Wahl des Ähnlichkeitsmaßes*

Für die Bildregistrierung wird wird ein Ähnlichkeitsmaß benötigt, das die Differenz zwischen $R$ und $T \circ \phi$ berechnet. Es ist die Zielgröße der Optimierung und dient auch zur Definition des Abbruchkriteriums.

Die Wahl des Ähnlichkeitsmaßes ist entscheidend für das Gelingen der Bildregistrierung, denn es legt die Richtung der Optimierung fest. Es gibt für die Wahl keine eindeutigen Kriterien, im Allgemeinen können aber für die multimodale Registrierung nicht dieselben Metriken verwandt werden wie für die monomodale.

Bei monomodaler rigider Registrierung wird z. B. der mittlere Quadratische Fehler (engl. *sum of squared differences*) als Ähnlichkeitsmaß verwendet (vgl. [1]):

$$\mathcal{D}^{SSD}[R, T \circ \phi] = \frac{1}{2} \int_{\mathbb{R}^d} (T(\phi(x)) - R(x))^2 dx$$

Bei einer Implementation dieser Metrik kann natürlich nur mit diskreten Pixeln gearbeitet werden. Für eine endliche Anzahl Pixel $N$ verwendet man (z. B. in ITK):

$$SSD\left(R, T \circ \phi\right) = \frac{1}{N} \sum_{i=1}^{N} \left(R_i - \left(T \circ \phi\right)_i\right)^2$$

Dabei bezeichne $R_i$ das $i$-te Pixel von $R$ und analog von $T \circ \phi$.

**Umsetzung in ITK**

Das *Insight Segmentation and Registration Toolkit* (ITK) (vgl. im Folgenden [2]) ist eine Programmierumgebung, die eine umfangreiche open source Sammlung von C++-Klassen für die Bildverarbeitung zur Verfügung stellt. Darin enthalten ist auch ein Framework für die Bildregistrierung, das im Folgenden in seinen Grundzügen vorstellt werden soll.

*Die Verarbeitungspipeline*

Die Bildverarbeitung in ITK ist als Pipeline organisiert. An deren Anfang stehen eine oder mehrere Bildquellen (engl. *sources*), z. B. Dateien auf der Festplatte des Rechners. Am Ende der Pipeline findet durch einen so genannten *mapper* eine Ausgabe statt, z. B. auf dem Bildschirm oder wieder in eine Datei. Dazwischen können das oder die Bilder mit verschiedenen Filtern bearbeitet werden.

In ITK gibt es entsprechende C++-Klassen, die Quellen, Mapper und Filter realisieren. Sie verfügen über die Methoden `SetInput()` und `GetOutput()`, womit sie zu einer Pipeline zusammengefügt werden können.

Der Vorteil dieser Pipeline-Architektur besteht in der Ersparnis von Rechenzeit und Speicherplatz. Werden Parameter an Filtern verändert, wird nur der Teil der Pipeline neu berechnet, welcher stromabwärts bezüglich der Veränderung liegt. Außerdem erlaubt die Pipeline *streaming*, d. h. die Bilder können, falls möglich, in kleine Teile zerteilt werden, welche die Pipeline einzeln durchlaufen und danach wieder zum Ergebnis zusammengesetzt werden. Dadurch kann der benötigte Platz für Zwischenergebnisse reduziert werden.

*Das Registrierungsframework*

Das Registrierungsframework in ITK kann besonders flexibel zur rigiden Bildregistrierung eingesetzt werden. Die Basisklasse dafür ist `itk::ImageRegistrationMethod`, die das Zusammenspiel von Transformation, Metrik, Optimierer, Interpolierer und der beiden Eingabebilder steuert (s. Abbildung 1).

Damit `ImageRegistrationMethod` funktioniert, muss eine Instanz davon erstellt werden, und anschließend müssen Metrik, Optimierer, Transformation, Interpolierer, Reference und Template gesetzt werden.

Die folgenden Codebeispiele zeigen, wie das Registrierungsframework schrittweise aufgebaut werden kann. Dabei wird deutlich werden, dass ITK umfangreich Gebrauch von C++-Templates macht. Zunächst werden die Typen von Template und Reference gesetzt:

```
const unsigned int Dimension = 2;
typedef float PixelType;

typedef itk::Image<PixelType,Dimension> ReferenceImageType;
typedef itk::Image<PixelType,Dimension> TemplateImageType;
```

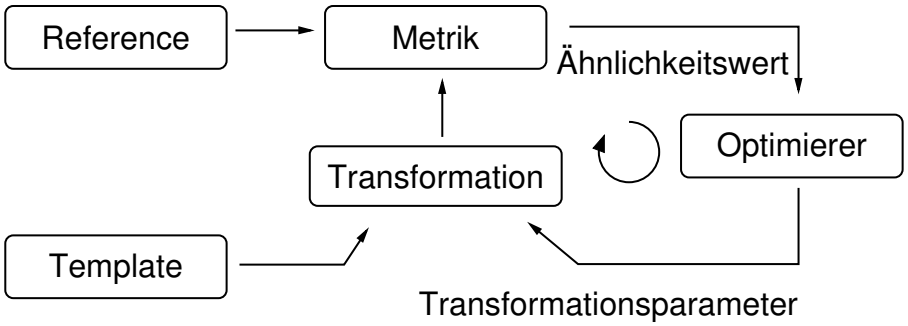



Abbildung 1: Das Registrierungsframework in ITK, wie es durch `itk::ImageRegistrationMethod` umgesetzt wird. Das Template wird durch eine parametrisierte Transformation iterativ verändert und mit Hilfe der Metrik mit dem Referenzbild verglichen. Die Metrik ist die Eingabe für den Optimierer, der die Transformationsparameter optimiert. Nicht dargestellt ist der Interpolierer. (Bild modifiziert nach [2])

Anschließend werden die übrigen Komponenten des Frameworks definiert und initialisiert. Im Falle der Metrik geschieht dies wie folgt (die hier gewählte Metrik ist der oben vorgestellte mittlere quadratische Fehler):

```
typedef itk::MeanSquaresImageToImageMetric<
                                    ReferenceImageType,
                                    TemplateImageType> MetricType;

MetricType::Pointer metric = MetricType::New();
```

Eine Besonderheit von ITK, die hier sichtbar wird, sind die sog. *smart pointer*. Dieses Konzept für die Speicherverwaltung, das auch unter der Bezeichnung *reference counting* bekannt ist, fügt zu jedem Zeiger auf ein (besonders großes) Objekt im Speicher einen Referenzzähler hinzu. Dieser kann benutzt werden, um nicht mehr benötigten Speicherplatz im Heap unverzüglich freizugeben, wenn kein Zeiger mehr darauf zeigt.

Analog zur Metrik werden auch die restlichen Komponenten des Registrierungsframeworks initialisiert. Ist dies geschehen, können alle Komponenten bei einem Objekt vom Typ `itk::ImageRegistrationMethod` registriert werden:

```
typedef itk::ImageRegistrationMethod<
                 ReferenceImageType,MovingImageType> RegistrationType;
RegistrationType::Pointer registration = RegistrationType::New();

registration->SetMetric(metric);
registration->SetOptimizer(optimizer);
registration->SetTransform(transform);
registration->SetInterpolator(interpolator);

registration->SetFixedImage(referenceImageReader->GetOutput());
registration->SetMovingImage(templateImageReader->GetOutput());
```

Die letzten zwei Zeilen zeigen, wie die Ausgaben der zwei Bildquellen für *reference* und *template* als Eingabe des Registrierungsobjekts verwendet werden, entsprechend dem Pipeline-Design von ITK.

Das Registrierungsframework ist so flexibel, weil die Typen der einzelnen Komponenten (in Grenzen) frei gewählt werden können.
Die Klasse der verwendeten Metrik muss bloß von `itk::ImageToImageMetric` erben, die des Optimierers von `itk::SingleValuedNonLinearOptimizer`, die Transformation von `itk::Transform` und der Interpolierer von `itk::InterpolateImageFunction`.

Dieses Design erleichtert es, eine Komponente des Frameworks unabhängig von den übrigen auszutauschen. Außerdem können auch neue, selbst geschriebene Klassen integriert werden, solange Sie von der richtigen Basisklasse erben.

*Implementation von Thirion*

Der Algorithmus von Thirion ist in ITK nicht als Transformation, sondern als Filter implementiert. Die oben vorgestellte Gleichung zur Ermittlung des Verschiebungsvektorfeldes ist dabei in verschiedenen Varianten umgesetzt worden (vgl. [2]).

Eine davon, die aus den Klassen `itk::DemonsRegistrationFilter` und `itk::DemonsRegistrationFunction` besteht, soll hier kurz vorgestellt werden. In Abwandlung zur oben vorgestellten Gleichung beinhaltet der Nenner dabei noch einen Normalisierungsterm:

$$\mathbf{u}(x) = \frac{(T(x) - R(x)) \cdot \nabla R(x)}{(\nabla R(x))^T \nabla R(x) + (T(x) - R(x))^2 / K}$$

Der Filter startet mit einem Verschiebungsfeld $\mathbf{u}_0(x)$ und berechnet darauf aufbauend für jedes Pixel $x$

$$\mathbf{u}_n(x) = \mathbf{u}_{n-1}(x) - \frac{(T(x + \mathbf{u}_{n-1}(x)) - R(x)) \cdot \nabla R(x)}{(\nabla R(x))^T \nabla R(x) + (T(x + \mathbf{u}_{n-1}(x)) - R(x))^2 / K}$$

Ist kein spezieller Startwert $\mathbf{u}_0(x)$ vorgegeben, werden alle Einträge auf Null gesetzt. Die Iterationsgleichung ist hier vereinfacht dargestellt, denn tatsächlich wird $\mathbf{u}_n$ nach jeder Iteration noch mit einem Gauß-Filter geglättet.

Ein Beispiel für die Registrierung mit Thirions Algorithmus, die mit ITK berechnet wurde, zeigt Abbildung 2. Man erkennt, dass die Registrierung zwar teilweise erfolgreich verläuft, aber trotzdem hat das Quadrat auch noch nach 2000 Iterationen abgestumpfte Ecken. Außerdem stellen sich mit zunehmender Iterationszahl vermehrt Fehler am Rand des Bildes ein.

(a) $n = 0$ (b) $n = 100$ (c) $n = 200$ (d) $n = 400$ (e) $n = 2000$

Abbildung 2: Nicht-rigide Registrierung mit Thirions Algorithmus. Als *Template* diente ein Kreis, als *Reference* ein Quadrat, $n$ bezeichnet die Anzahl der Iterationen.

**Diskussion über ein verallgemeinertes Registrierungsframework**

Das vorgestellte Registrierungsframework von ITK bietet vielfältige Möglichkeiten, verschiedene Metriken, Optimierer und Transformationen zu kombinieren. Damit kann die rigide Bildregistrierung sehr komfortabel gehandhabt werden.

Die nicht-rigiden Registrierung erscheint dagegen komplizierter. Auf der einen Seite gibt es nicht-rigide Verfahren, die als Transformation in ITK implementiert sind und sich daher gut in das Registrierungsframework einpassen. Auf der anderen Seite gibt es Fälle wie den Algorithmus von Thirion, der anders implementiert ist und nicht dazu passt.

Die derzeitige Umsetzung des Algorithmus von Thirion besteht aus einer einzigen Iterationsgleichung. Eine Aufteilung in Metrik, Transformation und Optimierer gibt es nicht, sie wäre aber durchaus wünschenswert. Insbesondere ist dieser Algorithmus durch das Fehlen einer separierten Metrik nur für die monomodale Registrierung geeignet.

Eine Abtrennung der Metrik widerspricht aber teilweise der Herleitung von Thirion. Eine wesentliche Voraussetzung war dabei, dass sich die Intensität eines Bildpunkts bei der Deformation nicht ändert. Will man aber multimodal registrieren, so kann die Intensität nicht als konstant angenommen werden. Daher ist ein einfaches „Einsetzen“ einer Metrik wie z. B. der *mutual information*, die sich für multimodale Registrierung eignen würde, nicht ohne weitere Überlegungen möglich. Man könnte z. B. versuchen, die Herleitung so zu modifizieren, dass sie ohne den direkten Vergleich von Grauwerten im Bild auskommt.

Ein Fernziel könnte es sein, ein so verallgemeinertes Registrierungsframework zu implementieren, dass darin die rigide Registrierung als Spezialfall der nicht-rigiden erscheint. Dabei wird das Problem auftreten, dass die Optimierung durch sehr hochdimensionale Parameterräume sehr zeitaufwändig gerät. Abhilfe könnten Ansätze wie in [3] schaffen, wo Informationen über Nullspalten in Jacobi-Matrizen zur Effizienzverbesserung ausgenutzt werden.

## Literatur


1. J. Modersitzki: *Numerical Methods for Image Registration*. Oxford University Press, 2004.
2. L. Ibáñez et al.:*The ITK Software Guide*. Second Edition. Kitware Inc., 2005.
3. M. De Craene et al.: *Incorporating Metric Flows and Sparse Jacobian Transformations in ITK*. Insight Journal, 2006.
4. J.-P. Thirion: *Image Registration as a diffusion process: an analogy with Maxwell's demons.* Medical Image Analysis 2 (1998) 3. S. 243–260.
5. J.-P. Thirion: *Fast Non-Rigid Matching of 3D Medical Images* Medical Robotics and Computer Aided Surgery (MCRAS 1995), Baltimore. S. 47–54.