FORSCHUNGSZENTRUM JÜLICH

Jülich Supercomputing Centre

Interner Bericht

Beiträge zum Wissenschaftlichen Rechnen Ergebnisse des Gaststudentenprogramms 2008 des John von Neumann-Instituts für Computing

Matthias Bolten (Hrsg.)

FZJ-JSC-IB-2008-07

FORSCHUNGSZENTRUM JÜLICH GmbH
Jülich Supercomputing Centre
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Beiträge zum Wissenschaftlichen Rechnen Ergebnisse des Gaststudentenprogramms 2008 des John von Neumann-Instituts für Computing

Matthias Bolten (Hrsg.)

FZJ-JSC-IB-2008-07

Dezember 2008

(letzte Änderung: 1. 12. 2008)

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 JSC als wesentlicher Säule des NIC. Um den akademischen Nachwuchs mit verschiedenen Aspekten des Wissenschaftlichen Rechnens vertraut zu machen, führte das JSC in diesem Jahr zum neunten 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 oder sich nach erfolgreichem Bachelor-Abschluss im Masterstudium befinden. Zusätzlich war eine Empfehlung eines Hochschullehrers erforderlich. Die zehn vom NIC ausgewählten Teilnehmer kamen für zehn Wochen, vom 4. August bis 10. Oktober 2008, ins Forschungszentrum. Alle Gaststudenten beteiligten sich hier an den Forschungs- und Entwicklungsarbeiten des JSC und der NIC Forschergruppe Computergestützte Biologie und Biophysik. Sie wurden jeweils einem oder zwei Wissenschaftlern zugeordnet, die mit ihnen zusammen eine Aufgabe festlegten und sie bei der Durchführung anleiteten.

Die Gaststudenten und ihre Betreuer waren:

Niklas Fricke	Walter Nadler
Martin Galgon	Tom Schröder, Helmut Schumacher
Martin Hoffmann	Paul Gibbon, Robert Speck
Christoph Honisch	Jan Meinke
Dorian Domenic Krause	Bernhard Steffen
Stefan Müller	Godehard Sutmann
Markus Peschina	Guido Arnold, Markus Richter, Binh Trieu
Ventsislav Valeriev Petkov	Wolfgang Frings
Lutz Roese-Koerner	Bernhard Steffen
Matthias Voigt	Inge Gutheil

Zu Beginn ihres Aufenthalts erhielten die Gaststudenten eine viertägige Einführung in die Programmierung und Nutzung der Parallelrechner im JSC. 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 JSC. 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. Ein besonderer Dank gilt Wolfgang Frings und Marc-André Hermanns, die den Einführungskurs gehalten haben, Anke Visser, die an der Erstellung dieses Berichtes maßgeblich mitgewirkt hat, und Robert Speck, der mich bei der Organisation in diesem Jahr nach Kräften unterstützt hat. Ebenso danken wir allen, die im JSC 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/jsc/gaststudenten.

Jülich, November 2008 Matthias Bolten

Inhalt

NMR Signal Formation in Capillary Networks

Niklas Fricke

Institut für Theoretische Physik,
Universität Leipzig
Vor dem Hospitaltore 1,04103 Leipzig, Germany

E-mail: niklas.fricke@itp.uni-leipzig.de

Abstract: Nuclear magnetic resonance (NMR) signals are strongly influenced by local field inhomogeneities. This can be exploited to investigate biological tissue on length scales below the resolution of ordinary magnetic resonance imaging methods and may yield information about size and distribution of capillaries in the myocardium allowing to diagnose stenosis of coronary arteries. However, the actual dependence of the signal on physical parameters (such as capillary width), especially under the influence of diffusion, is only poorly understood, and even for very simplified mathematical models, analytical solutions for the relevant quantities are usually not available. Here, a Monte-Carlo approach was chosen to simulate various models for capillary systems. The frequency autocorrelation functions and the NMR-signals obtained are compared to numerical results for the simplest model in order to test its applicability and the validity of the approximations employed in the numerical calculations.

Motivation

In the muscular tissue of the heart, the *myocardium*, capillaries are the dominating blood vessels, accounting for over 90 per cent of all vessel volume. Since a large fraction of the hemoglobin within the capillaries is desoxygenated and since this desoxyhemoglobin has a strong magnetic moment, capillaries invoke local magnetic field inhomogeneities when an external field is applied, thereby affecting NMR-signals.
If the coronary artery supplying the capillaries is stenotic (i.e., if its transport capacities are lowered due to atherosclerosis) the capillaries are wider than usual, counteracting the decreased blood supply. The effect on the NMR-signals of this larger width can be utilized to identify and localize stenosis. It is therefore of paramount importance to understand how NMR-signal decay depends on the capillary width. Unfortunately though, the mathematics involved in describing such systems are rather difficult. In fact only for the simplest capillary model, known as *Krogh's Model* (KM), the frequency correlation function has been calculated numerically [1], while even there the actual signals can be calculated only approximately [2], with an error of uncertain magnitude. Monte-Carlo (MC) Simulations may not yield as much understanding as analytic solutions, but they allow to study more sophisticated and realistic models with comparably small effort and without the introduction of unpredictable errors.

Brief Introduction of the Phenomenon and the Measured Quantities

Nuclear Magnetic Resonance

Nuclei with an odd numbers of nucleons (protons being the most important example in this context) possess a magnetic moment

$$\vec{\mu} = \gamma * \vec{S} \tag{1}$$

where $\vec{S}$ is the spin (magnitude $\hbar/2$) and γ is the gyromagnetic ratio, which is characteristic for the type of nucleus, but also slightly depends on the electronic shell. For the sake of convenience, nuclei, the molecules in which they are bound (in our case H_2O) as well as their magnetic moments shall in the following be referred to simply as “spins”.
When subjected to an external magnetic field $\vec{B}_z$ the spins align parallel or anti parallel to it, the majority preferring the parallel state (the exact quantity depending on the temperature via the Boltzmann factor $e^{-\frac{\Delta E}{k_b T}}$). If a second field $\vec{B}_{xy}(t)$ is applied orthogonal to $\vec{B}_z$ which oscillates with the frequency (known as the *Larmor frequency*)

$$\omega_L = \gamma * |B_z| \tag{2}$$

corresponding to the transition energy from parallel to anti-parallel state, the overpopulation of the parallel state gets lost and the spins can be described effectively as having a zero z-component and rotating in the xy-plane with frequency ω_L. In fact, the correct quantum-mechanical description would be more complicated [3], but this effective, semi-classical picture is sufficient for our purposes. Since the spins precess with identical frequency and phase, one can measure a macroscopic oscillation of the magnetization. From the strength of this resonance as a function of the applied frequency one may learn about quantities of elements and even of chemical compounds present in the observed system. Still more information can be gained by studying the system’s relaxation back to equilibrium after $\vec{B}_{xy}$ has been switched off again. There are two relaxation processes: the *Spin-Lattice Relaxation*, describing the realignment of the spins in direction of $\vec{B}_z$ and the *Transverse Relaxation*, describing the dephasing of the precession of the spins. This latter process is relevant here, since it is invoked by local inhomogeneities in the magnetic field. Indeed, local deviations δB from the mean magnetic field result in slightly different precession frequencies $\delta\omega = \gamma * \delta B$ for spins at different locations and hence cause them to dephase once $\vec{B}_{xy}$ is turned off.

NMR Signals and the Frequency Correlation Function

For the Transverse Relaxation, there are two basic types of signals that can be studied (although there are many more sophisticated techniques). One is the *Gradient-Echo signal* (GE), which is received when the system is allowed to relax undisturbed. The decay of this signal is characterized by its half-life T_2^* (or equivalently by its decay rate R_2^*), although this has to be handled with care, because the decay is usually not simply exponential. The other frequently studied quantity is the *Spin-Echo signal* (SE). Here the relaxing system is subjected to a short magnetic pulse after a certain time Δt, which causes the spins to change the direction of their precession. This reverses the decay; for spins at positions with larger $\delta\omega$, which until the reversing pulse had gained a head start on others, will now catch up on them. If the system were stationary, the total of the initial signal could be recovered. However, if the spins are in motion, each has undergone its individual history of frequency and part of the signal is lost irreversibly. The half-life for the SE signal is denoted by T_2, but here as well, the decay is not just exponential.

Another quantity of interest is the frequency autocorrelation function $K(t)$, defined as the expectation value of the product of a spin's frequencies at different times separated by t:

$$K(t) = \frac{1}{V}\int_V d^3\vec{r_0}\omega(\vec{r_0})\rho(\vec{r_0},t)\omega(\vec{r}(t)) \tag{3}$$

In contrast to the signals, $K(t)$ is usually not directly accessible by experiment (although under certain circumstances it can be measured [4]), but is all the more important for the theoretical description, and among other things it is used to (approximately) calculate the signals.

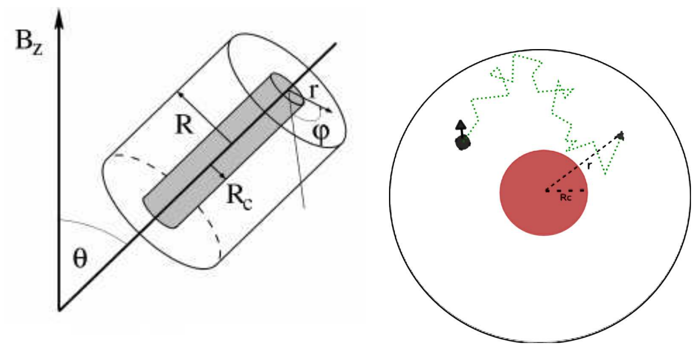
The Analytical Approach

Krogh's Capillary Model

As already mentioned, only a rather crude model for myocardial capillaries has so far been studied theoretically to a larger extent. This model assumes the capillaries to be identical, parallel cylinders (which is fairly justified, see [5]). The frequency shift $\delta\omega$ due to one capillary is then only a function of two parameters and thus allows to consider the model as two-dimensional:

$$\delta\omega(r,\phi) = \delta\omega_0 * R_c^2\frac{cos(2\phi)}{r^2} \tag{4}$$

where R_c is the capillary radius, r the distance to the center of the capillary and ϕ denotes the angle between the projection of $\vec{B_z}$ on the plane orthogonal to the capillaries and the position vector measured from the capillary center. $\delta\omega$ is a constant which collects the factors for the strength of the magnetic field, the magnetic susceptibility of the capillary and the tilting angle between $\vec{B_z}$ and the capillary direction: $\delta\omega_0 = \frac{\Delta\chi}{2}B_0 sin^2(\theta)$.



(a) (b)

Figure 1: (a) Scheme of the geometry modeled by the KM. The inner cylinder is the capillary, the space between the cylinders is the surrounding tissue where the spins can diffuse. (Figure taken from [1]) (b) Scheme of a diffusing spin according to the KM; reflective boundary conditions at the edges.

Now the critical simplification comes into play: instead of considering some kind of capillary lattice, only one capillary (represented as a circle) with a surrounding circular tissue is modeled. The spins may diffuse freely within the surrounding tissue, while the capillary wall, as well as the outer edge represent reflective boundaries. Although the outer reflective boundary conditions are essentially periodic ones, thanks to the symmetry $\delta\omega(r,\phi) = \delta\omega(r,\phi+\pi)$, it is not quite clear how well this model can mimic an actual lattice structure.

Describing the Diffusion

For obtaining the correlation function, the (still challenging) problem consist then essentially in solving the diffusion equation:

$$\frac{\partial \rho(\vec{x},t)}{\partial t} = D\Delta\rho(\vec{x},t) \tag{5}$$

with the described boundary conditions. This has been achieved via expansion into eigenfunctions [1], which yields numerically exact results (see Fig. 3).

To obtain the signals and decay half-lives, one would have to solve the diffusion equation for the local magnetization, known as the Bloch-Torrey Equation:

$$\frac{\partial m(\vec{r},t)}{\partial t} = [D\Delta + i\omega(\vec{r})]m(\vec{r},t) \tag{6}$$

$m(\vec{r},t)$ is a complex function, its time dependent phase reflects the precession in the xy-plane, which is the case for the signals as well. The GE signal is then simply spatial integral over $m(\vec{r},t)$:

$$M(t) = \frac{1}{V}\int_V d^3\vec{r}m(\vec{r},t) \tag{7}$$

and starting from this the SE signal could be calculated directly as well. Unfortunately, owing to its literal complexity, Eq. 6 has so far only been solved approximately [2], using a so called "strong collision" approach. This consists in substituting the diffusion operator $D\Delta$ by the "strong collision operator" $\boldsymbol{D}$ $= \frac{1}{\tau_K}(\boldsymbol{\Pi} - \mathbf{1})$, where $\boldsymbol{\Pi}$ is the projector on the equilibrium distribution and $\tau_K = \int_0^\infty dtK(t)$ is the mean correlation time of the frequency autocorrelation function. This approximation is justified, if the time scales on which diffusion and dephasing occur are very distinct, i.e. if $\delta\omega \ll 1/\tau$ or $\delta\omega \gg 1/\tau$ and is becomes even exact in the limiting cases where $D = 0$ or $D \to \infty$. If, on the contrary, the time scales are close, the method is rather dubious, and in any case the mathematical effort needed in carrying out the calculations is intimidating. Results for signals obtained in that way by Ziener et al. [2] can be seen later (Fig. 6a).

The Monte-Carlo Approach

In order to evaluate the strong collision approximation and Krogh's Model itself (also to go beyond it) MC methods are used here to simulate such systems.
The basic idea of MC is to use random numbers for sampling the phase-space of a thermal system. Usually Markov-Chains are used, with transition probabilities obeying probabilistic laws that lead to the correct statistical distribution. During this sampling, the quantities of interest can be measured; the mean values of those measurements will converge against the desired expectation values.
One great advantage of MC simulations with respect to other computational methods is, that parallelizing them is trivial, because they rely on a large number of *independent* calculations.

Simulating Krogh's Model

In our case, the Markov-Chains are discretized paths of diffusing spins; the transition probabilities are locally given by the Green's Function that solves the unrestricted diffusion equation (Eq.5) in two dimensions, which is a 2d-Gaussian:

$$P(\vec{x}_n \to \vec{x}_{n+1}, \Delta t) = G(\vec{x}_{n+1} - \vec{x}_n, \Delta t) = \frac{1}{4\pi D \Delta t} \exp\left[-\frac{(\vec{x}_{n+1} - \vec{x}_n)^2}{4D\Delta t}\right] \tag{8}$$

At the boundaries, one just has to arrange for the transition probabilities to be “mirrored” correctly, in order to account for the reflective bc’s.
In principle, the algorithm used to determine the frequency correlation function for Krogh’s Model was the following:

1. **Do M times** ($M =$ number of trajectories to be measured; for increasing M, the statistical error will reduce with $\frac{1}{\sqrt{M}}$)
   {
   (a) **Randomly chose a starting position (x_0, y_0) inside the area between the two circles** (This reflects the the fact that the equilibrium density distribution is uniform) **and measure the frequency shift** $\delta\omega$ (Eq.4)

   (b) **Do N times** ($N =$ Number of MC steps in one Trajectory)
   {
      i. **Draw** Δx**,** Δy**, from a Gaussian Distribution** (according to Eq.8), **move to new position $\vec{x}_{n+1} = \vec{x}_n + \vec{\Delta x}$**
         - **If new position is out of the boundaries, reflect it at the boundary.**

      ii. **Measure $\delta\omega$**

   (c) **Calculate K(t):**

$$K_m(n\Delta t) = \sum_{i=0}^{(N+n)/k} \omega[\vec{r}(k * i\Delta t)]\omega[\vec{r}((k * i + n)\Delta t)] \tag{9}$$

   (where k is a natural number between one and N, which affects only the performance speed)

2. **Calculate the mean value of the measured correlation functions:**

$$\langle K(n\Delta t)\rangle = \frac{1}{M}\sum_{m=1}^{M} K_m(n\Delta t) \tag{10}$$

The variance chosen for the Gaussian distribution determines the ratio of time resolution for the discretization of the path (Δt) and diffusion rate (D). It also directly influences the performance speed of the algorithm. This discretization is in fact the only point were a systematic error is introduced, but it can easily be made small enough so not to have any practical relevance, without losing too much performance.
One has some freedom in choosing how many steps are to be executed in one trajectory (provided of course, that the number of steps times Δt is not smaller than the time interval over which one wants to know K) and how many measured values $\delta\omega[\vec{x}(t)] * \delta\omega[\vec{x}(t + \Delta t)]$ are evaluated from one trajectory, which is determined by the choice of k (evaluating simply *all* is not efficient, because they are highly correlated). It is not obvious which choices are best, so for this work, they have been optimized empirically.

The NMR-signals can be obtained in a similar way. Here it makes sense not to chose the time for one trajectory longer than the time interval over which one wants to know the signals. Apart from this, the diffusing spins are simulated in exactly the same manner as before. Again, the frequency shift $\delta\omega$ is measured after each step, but is now integrated over time in order to obtain the accumulated phase $\Omega(t)$:

$$\Omega(n\Delta t) = \sum_{i=0}^{N} \delta\omega(\vec{x}(i\Delta t))\Delta t \tag{11}$$

The estimator for the GE signal can then be calculated by averaging:

$$M_{GE} = \langle \exp(i\Omega(t)) \rangle \tag{12}$$

In case of the Spin-Echo, the relevant phase is different:

$$\Omega_{SE}(n\Delta t) = 2\Omega(n\Delta t/2) - \Omega(n\Delta t) \tag{13}$$

but the signal is obtained analogously:

$$M_{SE} = \langle \exp(i * \Omega_{SE}(t)) \rangle \tag{14}$$

It has to be remarked that in contrast to the case of the correlation function for the signals another parameter becomes relevant, namely the ratio of the diffusion constant and the frequency shift constant $\delta\omega_0$. For $K(t)$, D only changes the time scaling, while $\delta\omega_0$ is just a constant prefactor, which cancels out, if one considers the normalized quantity $K(t)/K(0)$.

Other Models

The capillary model that comes to mind first is an ordered, quadratic lattice of parallel cylinders. Here again, it is sufficient to consider a two-dimensional slice. Furthermore, the movement within the lattice will be represented correctly via just one quadratic unit cell with periodic boundary conditions. But in contrast to the previous situation, the tilting angle α between the projection of $\vec{B}_z$ on the plane orthogonal to the capillaries and the capillary layers now plays a relevant role, as will be discussed later. Simulations for this model have been carried out for different values of α. The situation including the contribution of next nearest capillaries to the field has been studied as well.

Next, a triangular lattice structure of capillaries was simulated via a hexagonal unit cell. Different angles and additional fields from neighboring capillaries have been considered.

Finally, the influence of randomness (which in nature is always present to some degree) has been studied, first via the rather extreme case of a completely randomized lattice of parallel capillaries. Here we do not have a unit cell, so a larger number of randomly located capillaries (with the only restriction that they could not overlap) within a square with periodic bc's was simulated. It turned out that 200 capillaries were sufficient for the results to be self-averaging. As for the ordered models, usually only the fields of a smaller number of near capillaries were taken into account, this was effectuated by the introduction of a finite cut-off radius.
Simulations of this model with different cut-off radii have been realized.

For all those models, the measurements of the correlation function and signals could be done in much the same way as for the KM, the only basic changes being the different geometries on which the random walks took place.

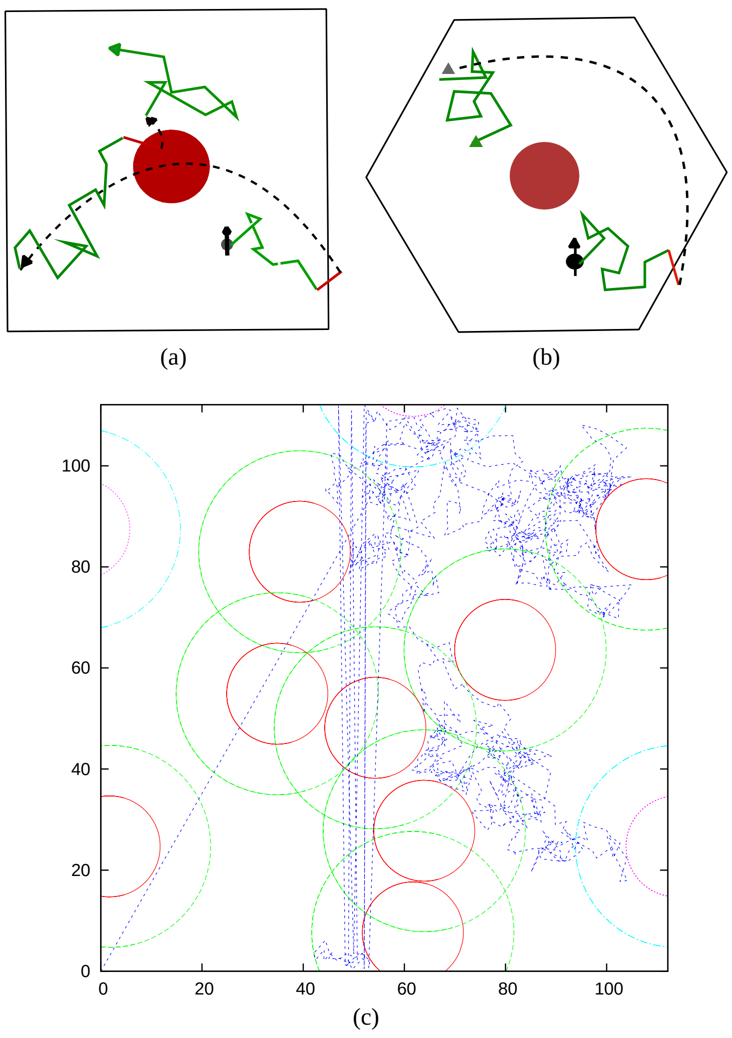


Figure 2: Illustrations of the different models: (a) displays the quadratic unit cell of the square lattice model; (b) the hexagonal unit cell of the triangular lattice model; (c) illustrates the trajectory of one spin in the random-lattice model. The straight vertical lines are the jumps when a boundary is crossed (the diagonal line is a plotting mistake). The pink circles are outside the area in which the spins can move. They do not represent individual capillaries, but are “mirror images”, accounting for the fields of others acting across the boundaries.

Results - Presentation, Comparison, and Discussion

The Frequency Autocorrelation Function

First, $K(t)$ obtained from the simulation of the KM shall be compared to results from numeric calculations done by Ziener et al. [1]. Since those are exact, this is mainly a check for the correctness of the simulations.

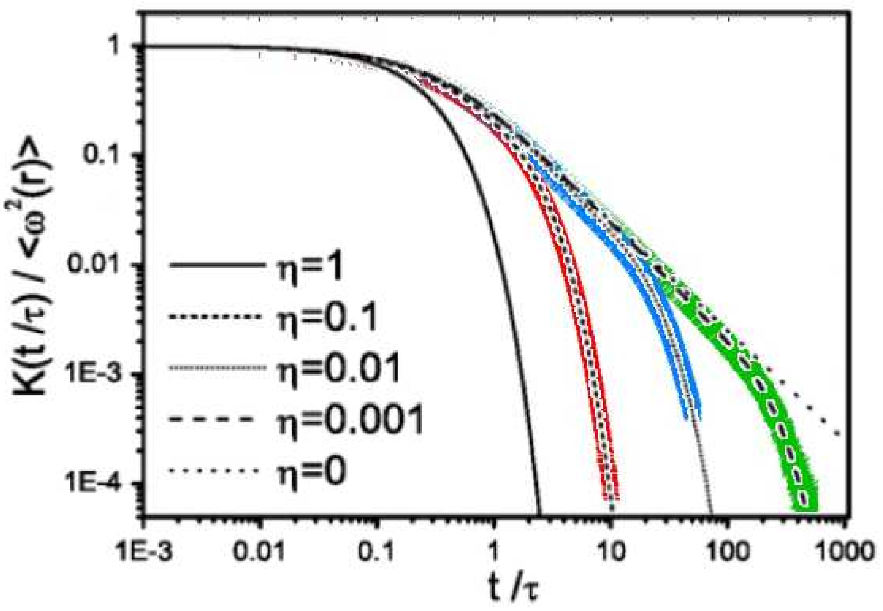


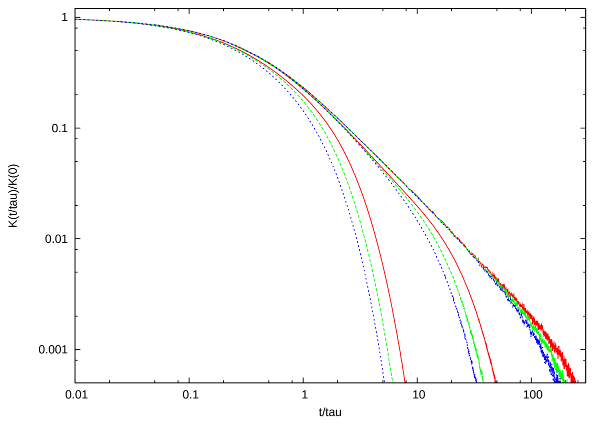
Figure 3: Comparison of the numerical (black) and simulational (colored) normalized frequency correlation function for Krogh's Model with different capillary volumes η. The data points from the simulation are larger than the statistical error bars, so to be visible. $\tau = R_c^2/D$ is the correlation time for the unrestricted diffusion. The numerical curves were taken from [1].

In fact, by graphical means (the data from the numerical calculations not having been at hand), it was not possible to detect any deviations beyond the (small) statistical errors.
This result strongly suggests the correctness of the method employed.

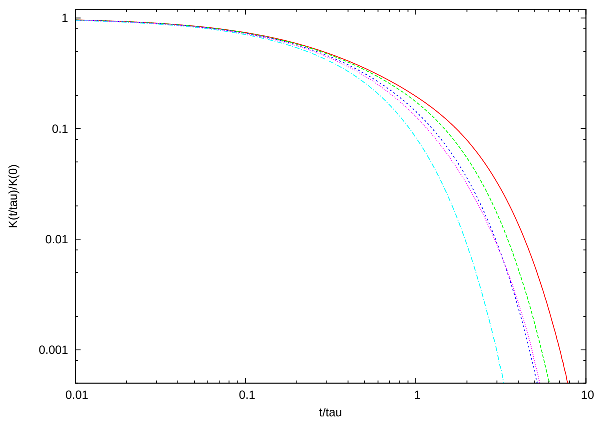
For the periodic lattice structures, $K(t)$ turned out to behave qualitatively similar, but to drop off decidedly faster. In Fig. 4, the tilting angle α was zero and no neighboring capillaries were considered. The effect of neighboring capillaries was in fact found to be very small for the periodic lattices and can therefore be neglected at this point.

The qualitative explanation for the faster drop-off in the case of the square (compared to the KM) might be the following: The effective η is smaller in the case of the square, because angular regions which are "cut off" (compared to a circle) are weighed stronger by the angular factor ($cos(2\phi)$) of $\delta\omega$, while in the corner regions this factor is small. Indeed, if η is chosen larger for the square (i.e., so that the side length of the square matches the outer radius of the KM) the curves are much closer (not shown here).

This, however, can not explain why $K(t)$ drops off even faster in the triangular model (see Fig. 3). In fact, one would expect the contrary, the hexagonal being closer to the circle than the square.
Here, a different explanation seems likely: This even faster drop-off might be due to the fact, that $\delta\omega(\vec{x})$ has stronger gradients near the boundaries in the hexagonal model. More precisely, if a spin crosses a boundary of the square, or is reflected at the outer edge of KM, the $\delta\omega$-field of the closest capillary re-



(a)



(b)

Figure 4: (a): $K(t/\tau)/K(0)$ for the KM (red curve); the square-lattice model (green/dashed) and the triangular lattice model (blue/dotted), with $\eta = 0.1, 0.01, 0.001$ (from right to left). (b): $K(t/\tau)/K(0)$ for the KM (red), the square model with B_z not tilted ($\alpha = 0$) (green/dashed), the triangular model with $\alpha = 0$ (blue/dotted), the square model with $\alpha = \pi/4$ (turquoise/dashed-dotted) and with averaged α (pink/small dots). η is 0.1 for all.

mains the same, while for the hexagon it changes, sometimes even its sign (see Fig. 5). Although the jump in $\delta\omega$ when a boundary is crossed is an artifact that vanishes when more neighboring capillaries are considered, the gradients persist, and it is rather obvious that this tends to reduce the correlations.

To test this hypothesis, one may consider the case of a quadratic lattice with a $\vec{B}_z$ being tilted by $\alpha = \pi/4$. Here those gradients are even stronger (the sign *always* changes when a boundary is crossed!), and thus $K(t)$ should drop very fast, which indeed it does (Fig. 4b, turquoise curve).
Surely, this dependence on α is something artificial that would not occur in patterns that are not strictly periodic (which one cannot expect of a biological tissue). Therefore simulations of the periodic lattices have been carried out, in which α was chosen at random for each trajectory, averaging out its effect (and also the effect of the "effectively smaller" η for the square). For the triangular lattice, the result remained practically unchanged, while for the square the curve lies between the to extremes ($\alpha = 0$ and $\alpha = \pi/4$), as one would have expected (Fig. 4b, pink curve). Besides it happens to be very close to the curve for the triangular lattice.

The results for the totally unordered lattice are very different. Most noticeably, one can observe a strong

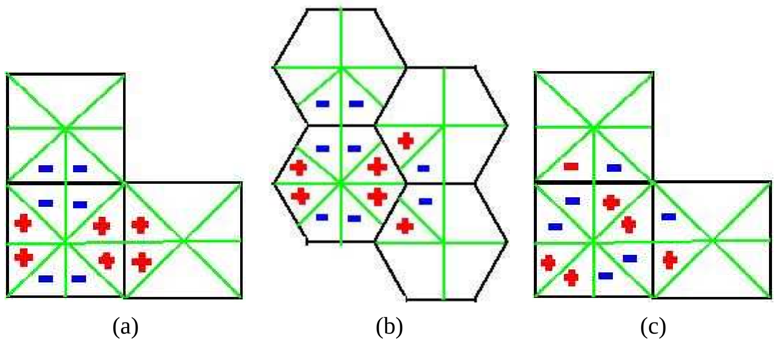


Figure 5: Schematic illustration of the change in $\delta\omega$ at the boundaries (a) square lattice with $\alpha = 0$: no change (b) triangular lattice: moderate change, sometimes change of sign (c) square lattice with $\alpha = \pi/4$: drastic change, always change of sign.

dependence on the cut-off radius, where for the ordered models, the corresponding factor (the number of considered neighboring capillaries) had almost no influence. On the other hand, there seems to be little direct influence of the volume fraction η (not shown here).

The NMR-Signals

As already mentioned, not even the KM allows for the signals to be calculated exactly, except for the limiting cases. For $D = 0$, the curve obtained by Ziener et al. [2] and the curve from the simulation are in convincing accordance, which is another endorsement for the correctness of our algorithm. However, for $D > 0$ the simulation and numerical results deviate considerably, matching more or less only for the first 5ms, see Fig. 6.

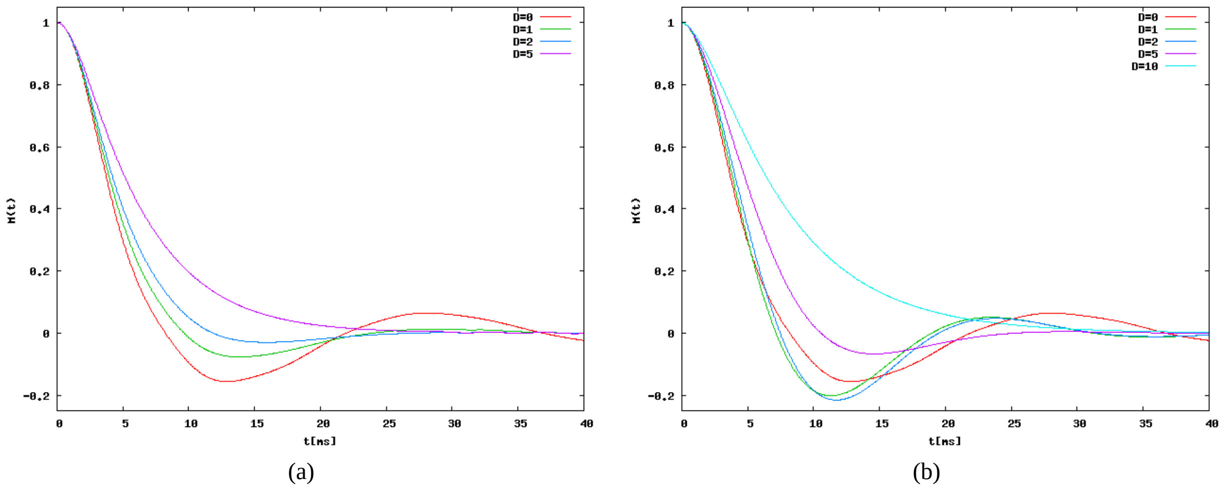


Figure 6: Comparison of the GE signals of KM for varying diffusion constants. Red curve: D=0, green: D=1, blue: D=2, violet: D=5 turquoise: D=10 ($\mu m^2 ms^{-1}$); $\delta\omega0 = 1 ms^{-1}$ (a) Approximate numeric results obtained by Ziener et al. (b) The results from the simulation.

The oscillations one observes for the stationary case ($D = 0$), are predicted to become flattened for $D > 0$ according to the numerical results, while the simulation predicts for them to reduce their period (while the amplitude of the first valley/peak increases) for D=1,2 and to vanish only for larger D's.
For the SE signal (Fig. 7), we also observe a severe discrepancy between the numerical and the simulational results.

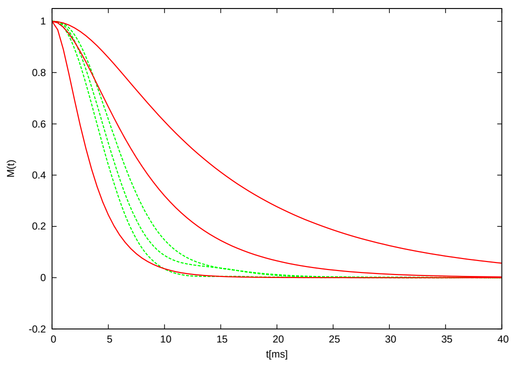


Figure 7: SE signals for the KM. Red lines are numeric results, green (dashed) lines are simulation results. $D = 1, 2, 5[\mu m^2 ms^{-1}]$ (from top to bottom).

The signals of the other models have been studied as well, some of the results are displayed in Fig. 8. As was the case for the correlation functions, the signals of the regular lattices drop of faster than those of the KM, but at least in a qualitatively similar way, while the signals for the randomized lattice seem to be much more persistent. But in contrast to the correlation function, the signals of the random-model turned out to remain almost unchanged when the cut-off radius was enlarged.
In addition to the two dimensional models mentioned here, several three dimensional models have been studied. Here, the motivation was mainly to analyze the influence of contrast agents within vessels or tissue. The results are interesting, but do not yet allow clear conclusions and shall therefore not be presented in this context.

Conclusions and Outlook

It has become clear, that the ability of the Krough's model to mimic a more realistic distribution of capillaries is limited, its main shortcomings being the assumption that periodic boundary conditions can be replaced by reflective ones (which only holds in special cases), and its disregarding of any randomness in the capillary distribution. Furthermore we must conclude that the strong collision approach leads to very inaccurate results for realistic values of D and $\delta\omega_0$.
One thing that still remains to be explained is the strong dependence of the correlation function on the cut-off radius for the random lattice.
The Monte-Carlo method was proved to be an adequate tool for studying myocardial capillary systems. It can easily be adjusted to different geometries, does not require to much computational effort in order to produce precise results, and is well suited to be carried out on parallel architectures. The next step would be to investigate models as close to reality [5] as possible, in order to obtain predictive results which could then be compared to experimental data.

Acknowledgments

I would like to express my gratitude toward the JSC for enabling this work. Special thanks go to Robert Speck and Matthias Bolten for the great organization of the guest student program, to Walter Nadler for his patience and support and to my fellow students for all the useful advice and the nice atmosphere.

References

1. C.H. Ziener, T. Kampf, V. Herold, P.M. Jakob, W.R. Bauer, W. Nadler, J. Chem. Phys. 129 (2008) 014507

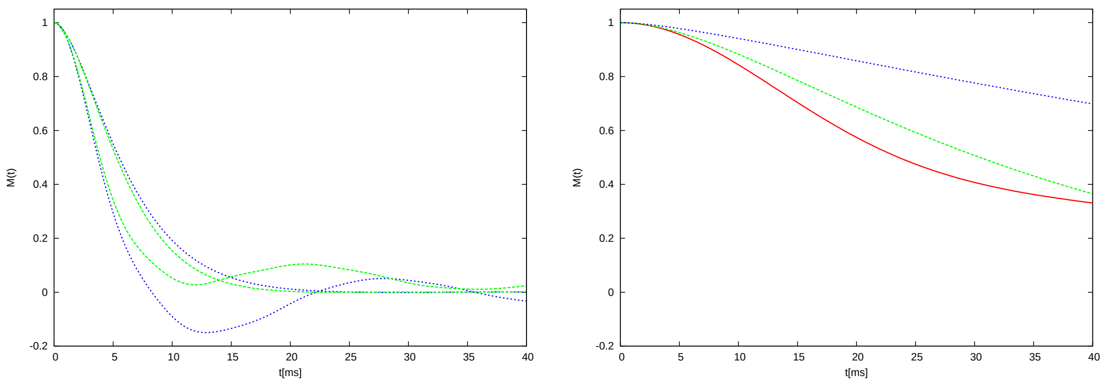


Figure 8: GE signal for different models. (a) square model (green/dashed) and hexagon (blue/dotted) with $D = 0$ (oscillating) and $D = 5$, respectively (b) randomized lattice $D = 0, 1, 5$ (from bottom to top).

2. C.H. Ziener, T. Kampf, G. Melkus, V. Herold, T. Weber, G. Reents, P.M. Jakob, W.R. Bauer, Phys. Rev. E 76 (2007) 031915
3. A. Abragan, *Principles of Nuclear Magnetism*, 1st ed. (Oxford University Press, New York, 1961)
4. J.H. Jensen, R. Chandra, R.A. Ramani, H. Lu, G. Johnson, S.P. Lee, K. Kaczynski, J.A. Helpern, Magn. Res. Med. 55 (2006) 1350
5. K. Rakusan, N. Cicutti, S. Kazda, T. Turek, Hypertension 24 (1994) 205-211

Visualizing Soil-Plant-Interactions Using VTK and Java

Martin Galgon

Bergische Universität Wuppertal, Fachbereich E - Elektrotechnik & Informationstechnik
Campus Freudenberg, Rainer-Gruenter-Straße 21, 42119 Wuppertal

E-mail: galgon@uni-wuppertal.de

Abstract: With R-SWMS being a complex 3D simulation model for water flow, solute transport and root growth, the need to present the generated amount of data in a clear, easy and understandable way is high. As most simple solutions fail at creating the complex images quickly and without a quality that satisfies scientific needs, a new, more powerful concept is needed. We present RSWMSViz, based on the extensive OpenGL accelerated VTK system. RSWMSViz is a Java program for both, high performance and high quality R-SWMS data presentation, coupled with user interaction and file export possibilities.

Introduction

Knowing the mutual influences between a plant and its surrounding soil means knowing the basis to be able to efficiently distribute seedlings on agricultural land, taking optimal use of the available space and minimal rivalry among plants into account. Analyzing and predicting these interactions - water flow and solute transport in particular - between a plant and the surrounding soil will help understanding irrigation and water scarcity problems. Water flow in the soil is determined by dynamic and non-linear parameters, i.e. topography, soil properties, vegetation and boundary conditions. Latter are given by external in- and outflow events, such as rain, irrigation, evaporation, deep drainage or runoff at the surface. The soil also looses water to the plant, which takes up water during daytime to compensate the water loss caused by transpiration at the leaves surfaces. This transpiration is a consequence of the photosynthetic process. Water uptake starts in upper, more humid regions and shifts to lower layers as these regions get dry. At night, water is relocated from deeper humid layers to upper dry layers.

A simulation system for these complex mechanisms, a fully coupled 3D soil-root-model, is R-SWMS [3]. R-SWMS uses two separate systems for soil and root which are coupled by a sink term to predict root water uptake (RWU) based on water potential differences between soil and root. RWU is largely affected by the spatial discretization of the soil around roots. R-SWMS provides several approaches to deal with local soil-root interactions without loosing crucial information by coarser spatial resolution [4].

To understand the huge amount of data produced by simulations, a concept to extract and process this data and finally present it in an eupeptic way has to be found. Due to human evolution, visual representations of the produced data are easy to understand and work with. Converting the received information to on-screen images is a complex task mainly because of two reasons. On the one hand all important information have to be drawn correctly and have to be intuitively understandable. For this purpose the data have to be presented in sufficient quality. On the other hand the image creation process has to be fast enough, not to simply show a single image but to present animations and allow interactions with the 3D data representations.

More accurate measurement techniques and root model development lead to large soil and root systems. Drawing complex 3D root structures and dense soil grids using software rendering only is unbearably slow. Software packages like MATLAB as interim solution are highly inappropriate for this task. Irregular grids, obtained by grid refinement techniques [6], tend to complicate things additionally. To create a platform to supply both, quality and speed, hardware acceleration is inevitable.
With this in mind, we need to find a concept for fast visualization of the prior described interactions and geometry, which can supply a framework for high performance graphics and adequate quality regarding scientific expectations.

In this report, we will combine the R-SWMS model as data source, Java for an easy to use graphical user interface and platform independency, and VTK [1, 2] for fast and high quality visualization to create a program which satisfies all these requirements: RSWMSViz.

R-SWMS: a 3D detailed model for water flow in soil and roots

R-SWMS[1] combines two models, basis of which is a 3D soil water flow and transport model [7], accompanied by a root water flow model [8]. The soil model already includes root growth but RWU is based on empirical relationships. The root model is used for finding the water potential and RWU distribution in the xylem network. Hereto boundary conditions in form of transpiration at the root collar and soil water potential distribution at the root surface are needed. Both systems are coupled by the sink term. The sink term is defined as a weighted sum of radial soil-root fluxes per volume of soil. To find a solution to the connected root-soil problem, iterative coupling between the systems is needed.

The geometry of the soil is given by a 3D cube grid, dividing the soil into voxels (Fig. 1). Each corner of the cube grid represents a soil node for which the Richards Equation is solved.

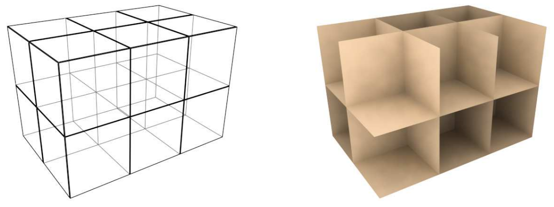


Figure 1: Initial soil grid structure

The Richards Equation is given by

$$\frac{\partial \theta}{\partial t} = C\frac{\partial h}{\partial t} = \nabla \cdot \left(K\left(h\right)\nabla\left(h+z\right)\right) - S\left(h\right) \tag{1}$$

where θ is the water content in $\left[cm^3\ cm^{-3}\right]$, h the water potential (pressure head) in $[cm]$, $C\left(h\right)$ the volumetric water capacity in $\left[cm^{-1}\right]$ defined by $C = \frac{\partial \theta}{\partial h}$, K the non-linear hydraulic soil conductivity in $\left[cm\ d^{-1}\right]$ and S the sink term in $\left[d^{-1}\right]$.

The geometry of the root is represented by a tree-like structure consisting of connected segments of which each juncture defines a root node (Fig. 2, left). To calculate the xylem water potential (h_x) for a node, a linear system of equations is solved:

[1]Root - Simulating Water Flow and Solute Transport in Three-Dimensional Variably-Saturated Media

$$J_r = L_r\left(h_s - h_x\right)$$
$$J_x = -K_x \frac{dh_x}{dl} \tag{2}$$

where J is a flux density in $\left[cm\ d^{-1}\right]$, h_s the water potential at the root surface, l the root segment length in $[cm]$, K_x the xylem conductance in $\left[cm^3\ d^{-1}\right]$ and L_r the radial conductivity in $\left[d^{-1}\right]$.

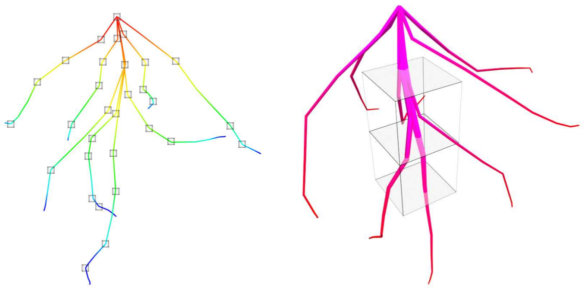


Figure 2: Left: root architecture, right: independent root and soil structures

Root and soil use independent systems (Fig. 2, right), meaning their coordinates not necessarily use the same grid. This leads to a problem regarding the water potential calculation at the root surface, which is a boundary condition for the root system. Overall, four methods can be used to calculate water potential [4, 5] (Fig. 3).

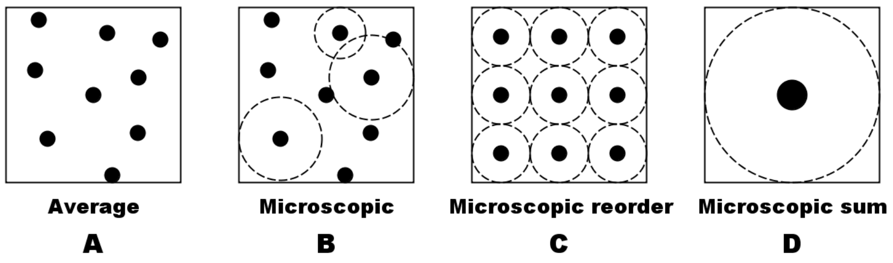


Figure 3: Incorporating local conductivity drop

A first simple method (A) obtains the water potential at the root surface by calculating a weighted average from all soil nodes surrounding a root node. A more accurate microscopic model (B, C, D; analytic approach) uses a variation of the Richards equation. Model B preserves the original root node distribution. Model C redistributes the nodes for more heterogeneity and model D sums up all root nodes in the soil voxel.

By averaging the water potentials (method A) at the soil corner nodes, crucial information is neglected (if the spatial distribution is not fine enough) as the non-linear hydraulic conductivity causes a large drop near the root, especially in dry soil regions. If this local drop is considered, significant changes in the prediction of the xylem water potential are noticed, but the soil water potential gradient is more or less unaffected. The local drop should be considered. To get an accurate gradient throughout the soil and near the root, grid refinement is needed.

Irregular grids reduce computational costs and still preserve high spatial resolution. Refinement is done by bisecting all edges of a specific soil voxel and slicing the volume along the corresponding planes (Fig. 4). It is based on root information, the grid is dense near roots and coarser farther away in the soil [6]. Refinement over time may be considered, based on a posterior error estimate.

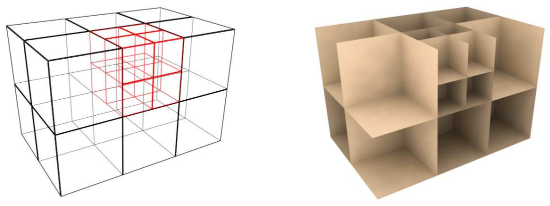


Figure 4: Refined grid

VTK

VTK is an open source library for scientific visualization and image processing. It is based on the rendering library OpenGL and provides a higher abstraction level for easy application development. Written in C++, it consists of over 700 classes supplying a huge framework containing lots of precast data structures, algorithms, modeling techniques and rendering equipment, empowering the developer to visualize his data in high quality and with high performance. VTK provides wrappers for Tcl, Python and Java.

Graphics and visualization model

VTK uses so called data sets to represent different types of three dimensional data. All available VTK data sets consist of a set of points, representing the raw geometry, and a list of cells, each to which a list of points is assigned. The cells represent the topology of the data and allows e.g. interpolation between points. These data sets are created, manipulated and finally rendered by the VTK pipeline where data sets are passed from object to object and constantly are transformed, extracted or enhanced. The Pipeline can be divided into two sections, the data creation and manipulation phase and the image creation phase (Fig. 5). For the creation phase VTK provides multiple classes to create primitive geometry, read in VTK file formats or import different image files and 3D formats. The resulting data sets then can be modified in many ways by using so-called filters or directly be sent to a mapper for rendering. Many filters produce multiple output and/or accept multiple input to perform their task. After modification, the data set has to be passed to a mapper, which creates OpenGL compatible triangular 3D objects from the data sets. The mapper again is assigned to an actor, which represents the object in the scene and handles color, position or size. A renderer then is responsible for producing an image which is drawn into a render window viewport. Finally an interactor can be used to manipulate objects in the scene or the camera of the viewport.

Pipeline execution

Executing the pipeline follows a distinct concept in VTK. When talking about “Lazy Execution” we describe a demand driven model where only the out-of-date parts of the pipeline execute if data has been requested. To determine which parts of the pipeline will re-execute to keep the output data up to date, VTK employs a timestamp system. A method call, which does not require re-execution of the specific pipeline element, may return wrong results if the object was modified right before the requesting call, but no update request was made. In almost every case a manual update request is not necessary as a render initiates the request for data, sends an update request up the pipeline and provokes re-execution of out-of-date elements. The updated data then travels down the pipeline to be rendered (Fig. 6). This execution concept may produce unexpected results if we try to manipulate output data sets somewhere in the pipeline manually. All changes will be overwritten by the next update request made to the pipeline element we requested the output from. To bypass this effect, we simply can create a real copy of the dataset and manipulate the copy, but this solution is quite ineffective regarding memory usage. Another

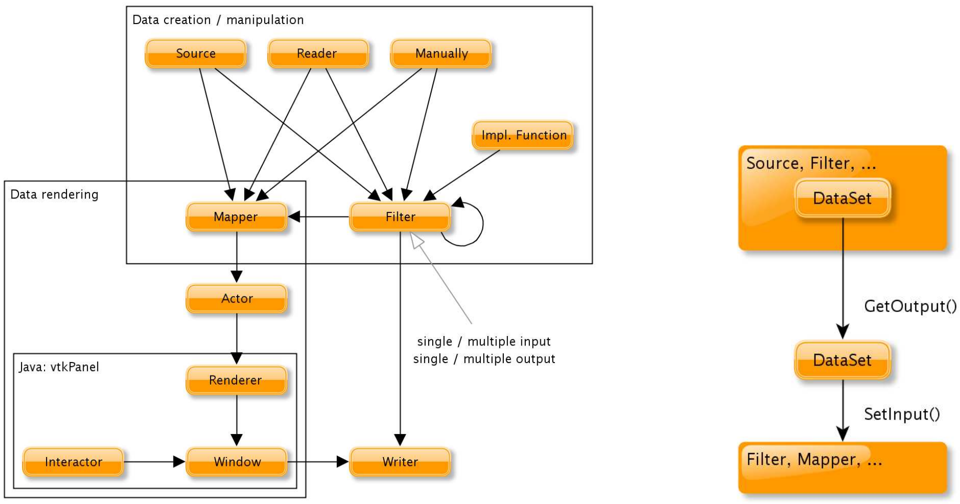


Figure 5: Basic pipeline concept

way is to temporarily establish a pipeline to the point the manual modification should be done. Then pipeline execution is forced and the pipeline is destroyed shortly after. Of course this only works out if the pipeline is not needed anymore after first execution.

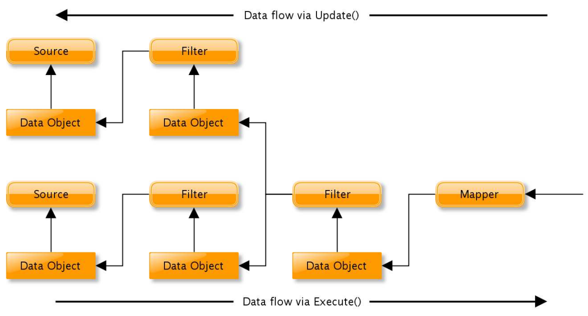


Figure 6: Pipeline data flow

Handling large data sets

VTK by default is configured to handle small amounts of data. To reduce execution time for large data sets, as obtained with R-SWMS, and prevent not modified pipeline elements from re-executing, all filters in the pipeline preserve intermediate results. These results can be freed after execution, but then the whole pipline will have to re-execute on a parameter change. Using this method only will affect huge amounts of data and accelerate computation by reducing overhead.

RSWMSViz

RSWMSViz is the program developed to visualize R-SWMS output files using VTK. RSWMSViz is written in Java and is supposed to run in runtime environments of version 1.6 or higher. It uses a slightly modified VTK 5.0.4 API. For thread safety during file writing processes, vtkPanel 1.18 is used. In the mentioned component, which is part of the Java wrapping system, the VTK internal execution control can be locked explicitly. By doing this, an error that occurs when multiple threads try to access a Linux system's X-Server simultaneously, can be prevented. The situation occurs when using Java to implement a VTK pipeline and trying to write files using a vtkWriter. RSWMSViz can produce output in BMP, JPEG, PNG and Postscript. Additionally, complete time-lines can be rendered into an MPEG2 movie. To gain access to the generated root 3D mesh, the whole scene can be exported as Alias|Wavefront and VRML[2] compatible formats. RSWMSViz is capable of displaying root structures, vertical soil planes and the velocity field. Root and soil planes can be mapped in several different ways, scalar zooming is available for both. The root radius can be adjusted to match the applied mapping (Fig. 17). A wide variety of helper objects can be displayed and manipulated. The velocity field can be clipped in upper and lower vertical direction for better visibility of field details (Fig. 16). Streamlines show the way and speed of an imaginary particle on its way through the vector field (Fig. 15). A loaded time-line can be shifted simply by moving a slider, intersections of the root with the vertical planes can be computed and displayed (Fig. 14). Vertical planes, velocity field and intersections can be viewed in a 2D topview (Fig. 13). These and many other options are directly available at the user interface (Fig. 10). Advanced options, mainly to acquire custom output pictures, can be changed directly inside the source code.

In the next paragraphs we will focus on only a couple of aspects of RSWMSViz. For a full functional overview refer to the user manual.

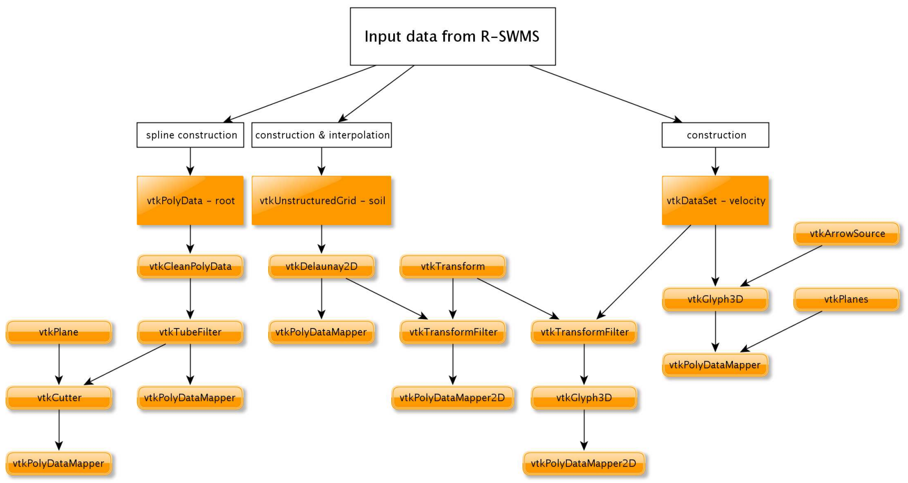


Figure 7: Reduced overview of the most important VTK pipeline segments in RSWMSViz

R-SWMS output files

R-SWMS produces three types of output files which have to be read by RSWMSViz. Additionally, a file containing the initial soil grid is read.

- Root data files (outRoot.#) - contain root data per node

  Geometric data (coordinates, previous node, branch number)

  Axial flow from node to node

  Radial flow from root surface to xylem

  Water potential at xylem and root surface

  Axial and radial root conductivity

- Soil data files (outfem.#) - contain regular or irregular soil data

[2]Virtual Reality Modeling Language, a file format for representing and exchanging 3D interactive vector graphics

Grid coordinates

Volumetric water content

Water potential

Concentration (solute transport)

Sink term (representing RWU)

- Velocity data files (veloci.#) - contain the vector field in x-, y- and z-direction

  Coordinates

  Velocity

RSWMSViz can load a whole set of data files to display changes over time. For this an interval referring to the R-SWMS output file suffix is requested from the user. For each simulation output time a set of the above three files types is loaded. The initial grid file, nodes.in, is loaded only once. It contains the initial unrefined grid used by R-SWMS before any refinement has been performed (if applicable). The internal data structure of RSWMSViz basically follows the structure of the input data.

Delaunay triangulation

A desired feature is the possibility to display the soil grid by means of planes for a specific depth, i.e. z-coordinate. With only an unsorted set of points given, the points have to be ordered by their vertical coordinate. The resulting subsets can then be connected to form a plane. An efficient way to produce triangular meshes from point sets with good results is the Delaunay triangulation[3]. The Delaunay criterion requires no other point of the given point set to be localized inside a surrounding circle of any of the triangular delaunay facets, so a circumcircle of a triangle contains only the three defining points of this very triangle. With this, a Delaunay triangulation maximizes the minimum angle over all triangles inside the grid. The definition can be generalized for higher dimensional simplexes, but for the two dimensional case, the created triangulation is optimal. There are many possible algorithms[4] to implement a delaunay triangulation, the best one runs at $\mathcal{O}(n \log n)$. The Delaunay graph is the dual graph of the Voronoi tessellation[5]. A VTK internal implementation is available for 2D and 3D point sets, where the 2D variation simply ignores all z-coordinates the points may have.

Interpolation

Due to grid refinement, at some layers the planes will be incomplete (Fig. 8, left). To generate a smooth transition while viewing the planes throughout the soil column, the incomplete planes have to be padded by interpolation from the nearest complete planes atop and beneath the incomplete plane. To achieve this the missing points have to be retrieved from an additional grid, which in this case is best chosen as the initial unrefined grid used by R-SWMS as this guarantees the existence of the points needed for interpolation in the adjacent layers. This grid can not be easily reconstructed from the given refined grid and it is much more efficient to take the initial coarse grid (nodes.in). For RSWMSViz it suffices to construct a file which contains each emerging x-, y- and z-coordinate only once because the initial grid is always considered regular and the needed grid can be constructed with this minimal information. Knowing all points of this regular grid, the missing points for a certain plane can be marked during point ordering and are added before the delaunay triangulation constructs the final plane (Fig. 8, right).

[3]Boris Delaunay, 1934

[4]Possible algorithms are: flip, incremental, divide & conquer, sweep, voronoi and convex hull

[5]Georgy Voronoy; a point p in the 2D plane belongs to the interior of a Voronoi-cell defined by a specific point P out of a given point set S, if p is closer to P as to any other point of S.

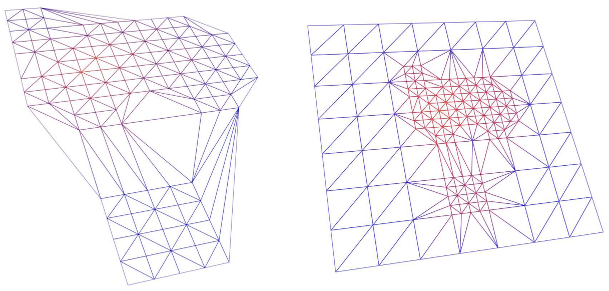


Figure 8: Not interpolated (left) and interpolated (final plane)(right) grid structures after delaunay triangulation

Scalar zooming

When color mapping a 3D object with vertex colors from given per-point scalar values, the overall scalar range determines the difference in color assigned to two adjacent scalar values onto the object. Very small changes in scalar values might not show a visible change in color, not even by using a multicolor gradient. To be able to visualize the scalar behavior in specific regions on the 3D object the color mapping has to be altered. The scalar range, the mapper refers to, is reduced and adjusted to the desired values. To achieve this, the scalar range is shifted using an exponential factor[6], which allows more delicate control at higher zoom levels. Here the shift has to increase more slowly so the user might not miss the desired point or the point is not skipped due to the value stepping of the control component. To empower the user to zoom into any region, the zoom target can be shifted over the whole scalar range. The color will be interpolated along the new zoomed scalar range (Fig. 9). All scalars not in this interval will obtain the corresponding color assigned to the minimum or maximum value.

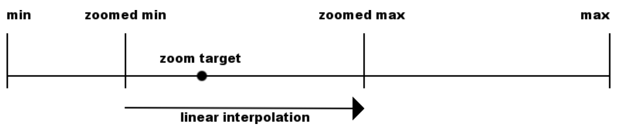


Figure 9: The scalar zooming feature

An example

A large example root structure (Fig. 18) is read, constructed and rendered on a subsecond timescale. Drawing the same root using MATLAB (Fig. 11) took a couple of hours and still was basically just a colored line drawing. RSWMSViz generates a fully three-dimensional mapped root model the user can interact with. Not only a 3D representation is created, RSWMSViz also extracts and computes additional information from the given data. All important parameters can be changed on the fly, additionally computed information like intersection contours can be generated and drawn without delay. Some features are a little slower due to complex calculations (i.e. streamlines, anti-aliasing). Depending on performance of the deployed graphics card, even very large root structures, dense soil grids and complex velocity fields can be rendered and interacted with, simultaneously and in realtime (Fig. 12). Graphical output quality is VTK specific on a high, scientific oriented level.

[6]An empirical derived function with good results is $\frac{1}{e^{x \cdot m}}$, where x is a linear natural factor, given e.g. by an GUI control element and m an multiplier whose optimal value for a certain scalar range is given by $m = \frac{ln(\frac{D}{d})}{X_{max}}$, where D is the overall scalar range, d the minimum occurring difference of two adjacent scalar values and X_{max} the maximum value for the linear scaling factor given by the GUI component. Computing d can only be done quite inefficiently, so a base value of $m = 0.05$ is predefined but can be modified inside the RSWMSViz source code. As an alternative an approximately good value can be found letting d be the minimum difference over all scalar values.

Conclusion and outlook

Alltogether the attempt to develop a platform for visualizing the complex R-SWMS root and soil structures sufficiently fast and in appropriate quality was successful. The user is provided with an extensive amount of visualization options.

Still further improvements can be integrated:

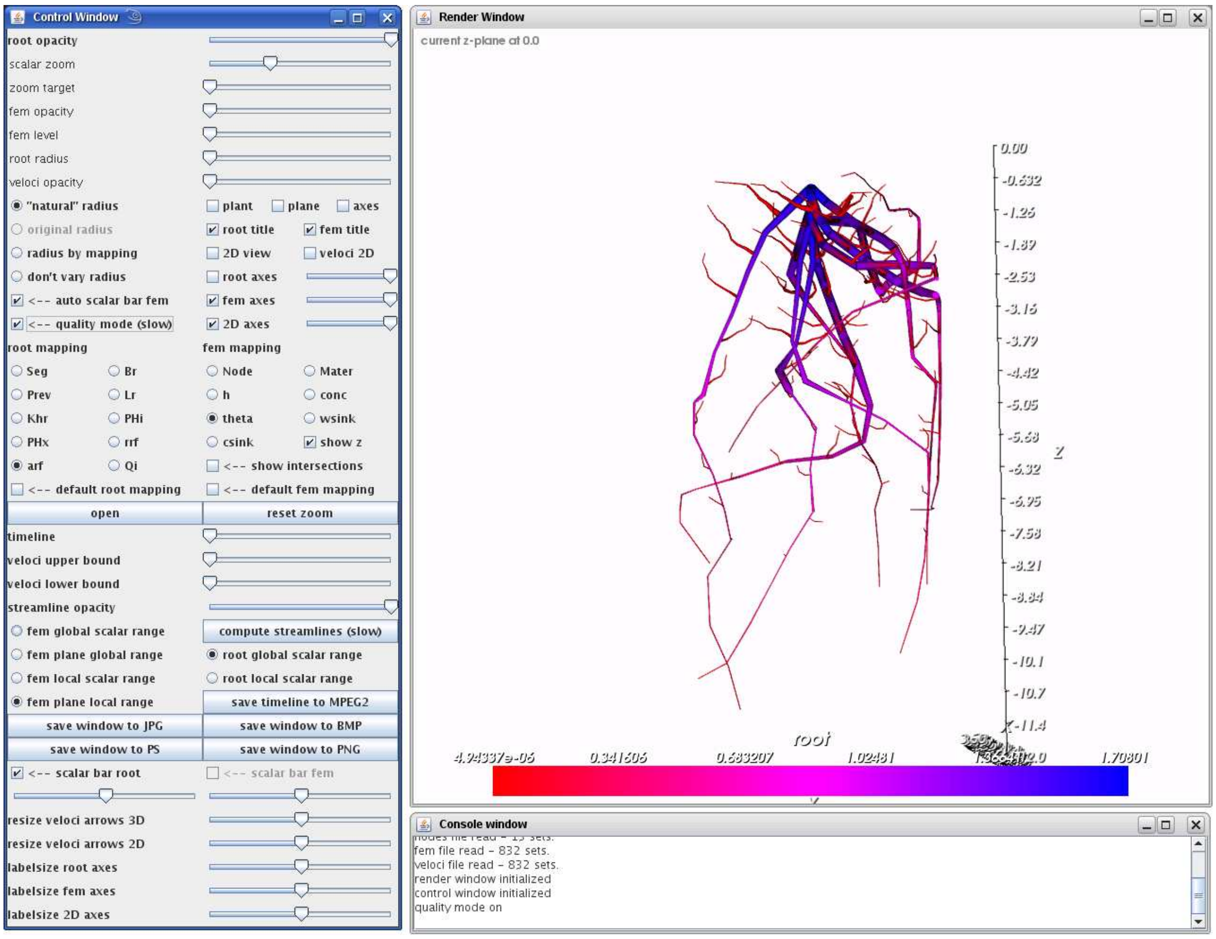
- As R-SWMS will be able to simulate more than one root together in a soil volume, RSWMSViz will have to support multiple roots in the future. It already can handle more roots, if given in a single file with disjunct node numbering. The main disadvantage here would be the joint scalar ranges of the roots, which can approximately be overcome by using scalar zoom.
- A layered view of the soil grid already exists in vertical direction. Expanding in horizontal directions is desired.
- Data management is still not optimal and can be improved by eliminating unnecessary pipeline segments and redundant data copies.
- The rudimentary graphical user interface was implemented for testing purposes only and will have to be improved to be more intuitive and ordered.

Acknowledgment

At this point I'd like to thank my adviser Tom Schröder for always sacrificing his time and supporting me. I also want to thank Dr. Helmut Schumacher for supplying me with VTK knowledge and literature. Furthermore I'd like to express my gratitude towards Matthias Bolten and Robert Speck for once more organizing a successful guest student program. Thanks for these interesting two months. And I am indebted to Prof. Dr. Andreas Frommer at Wuppertal University for his recommendation, which made my attendance possible in the first place. Finally, thanks go to my guest student colleagues who where unexceptional outstanding pleasant companions in these times.

References

1. Avila, L. S., S. Barré, B. Geveci, A. Handerson, W. A. Hoffman, B. King, C. C. Law, K. M. Martin, W. J. Schroeder, 2003, The VTK User's Guide, Kitware, Inc.
2. Schroeder, W., K. Martin, B. Lorensen, 2002, The Visualization Toolkit - An Object-Oriented Approach to 3D Graphics, Kitware, Inc., 3rd Edition
3. Javaux, M., T. Schröder, J. Vanderborght, H. Vereecken, 2008, Potential use of a Detailed Modeling Approach for Predicting Root Water Uptake, Vadose Zone Journal, 7: 1079-1088
4. Schröder, T., M. Javaux, J. Vanderborght, B. Körfgen, H. Vereecken, 2008, Effect of local soil hydraulic conductivity drop using a 3D root water uptake model, Vadose Zone Journal, 7: 1089-1098
5. Schröder, T., M. Javaux, J. Vanderborght, B. Körfgen, H. Vereecken, 2008, Effect of local soil hydraulic conductivity drop on root water uptake at the plant scale, Vadose Zone Journal, submitted
6. Schröder, T., L. Tang, M. Javaux, B. Körfgen, H. Vereecken, 2008, A grid refinement approach for 3D soil-root models, Plant and soil, submitted
7. Somma, F., J. W. Hopmans, V. Clausnitzer, 1998, Transient three-dimensional modeling of soil water and solute transport with simultaneous root growth, root water and nutrient uptake, Plant and Soil, vol. 202, no 2, 281-293
8. Doussan, C., L. Pagès, G. Vercambre, 1998, Modelling of the hydraulic architecture of root systems: An integrated approach to water absorption-model description, Ann. Bot. 81: 213-223



root opacity
scalar zoom
zoom target
fem opacity
fem level
root radius
veloci opacity

[✓ "natural" radius] original radius radius by mapping don't vary radius
plant plane axes [✓ root title] [✓ fem title] 2D view veloci 2D root axes [✓ fem axes] [✓ 2D axes]
[✓ <-- auto scalar bar fem]
[✓ <-- quality mode (slow)]

root mapping: Seg Br Prev Lr Khr PHi PHx rrf [✓ arf] Qi
<-- default root mapping

fem mapping: Node Mater h conc [✓ theta] wsink csink [✓ show z]
<-- show intersections
<-- default fem mapping

timeline
veloci upper bound
veloci lower bound
streamline opacity

fem global scalar range
fem plane global range
fem local scalar range
[✓ fem plane local range]
[✓ root global scalar range]
root local scalar range

[✓ <-- scalar bar root]
<-- scalar bar fem

resize veloci arrows 3D
resize veloci arrows 2D
labelsize root axes
labelsize fem axes
labelsize 2D axes

```
nodes file read - 15 sets.
fem file read - 832 sets.
veloci file read - 832 sets.
render window initialized
control window initialized
quality mode on
```

Figure 10: RSWMSViz graphical user interface (non-final version)

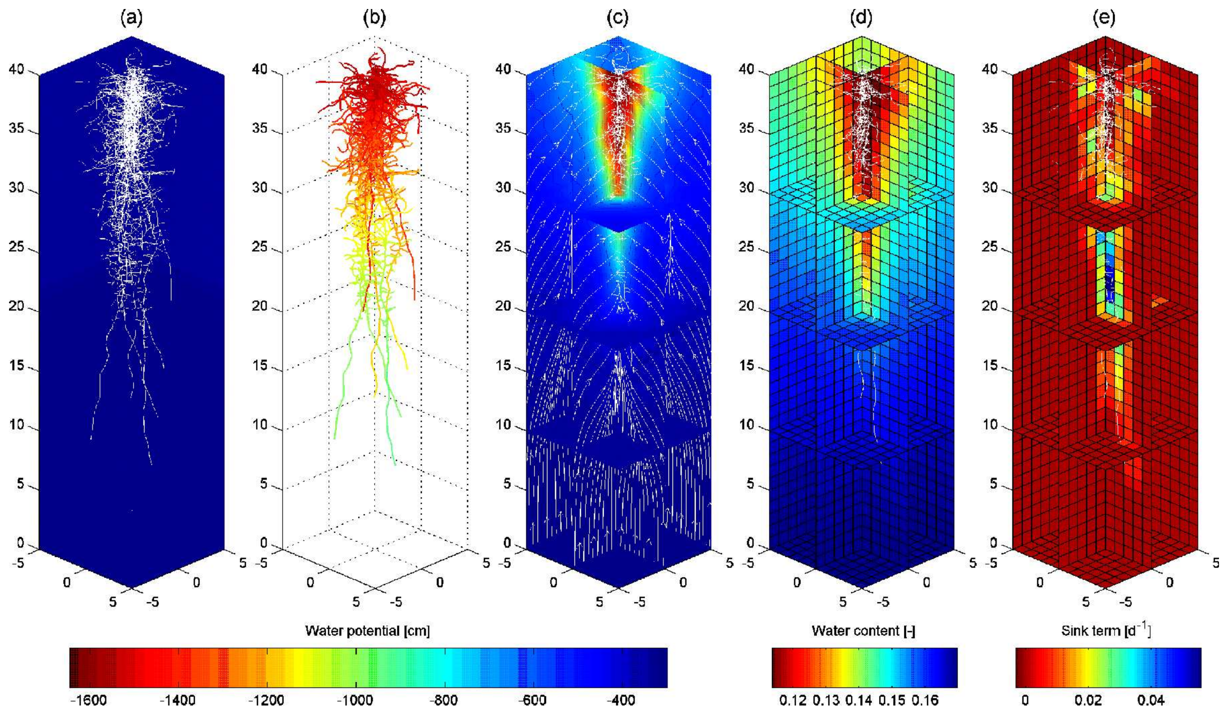


Figure 11: Former visualization result

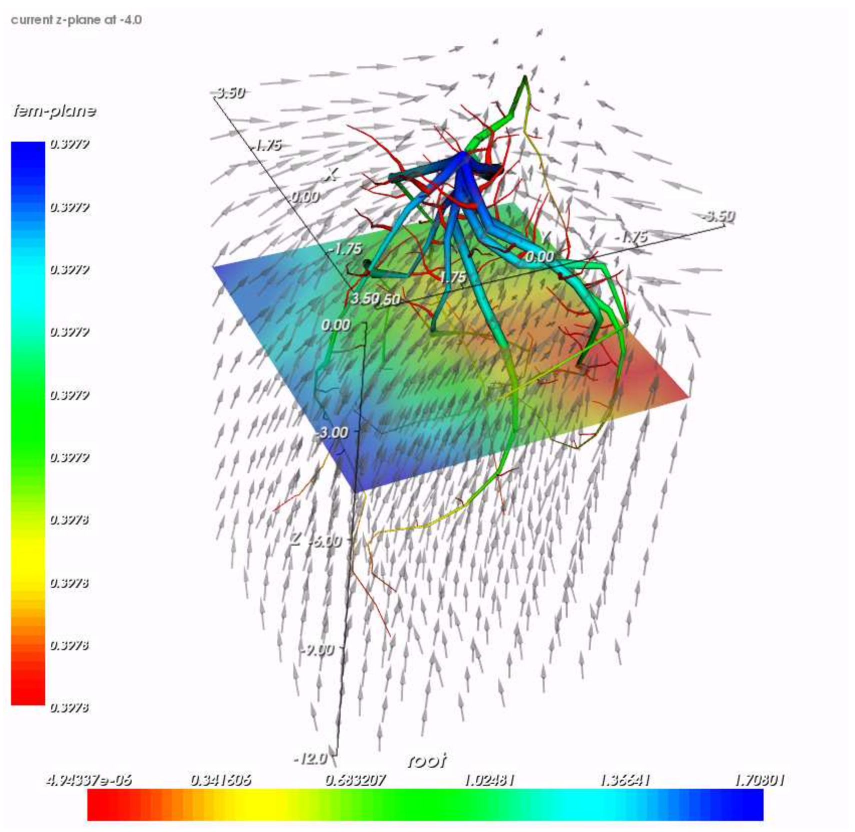


Figure 12: RSWMSViz graphical output showing root structure, vertical soil plane and velocity field

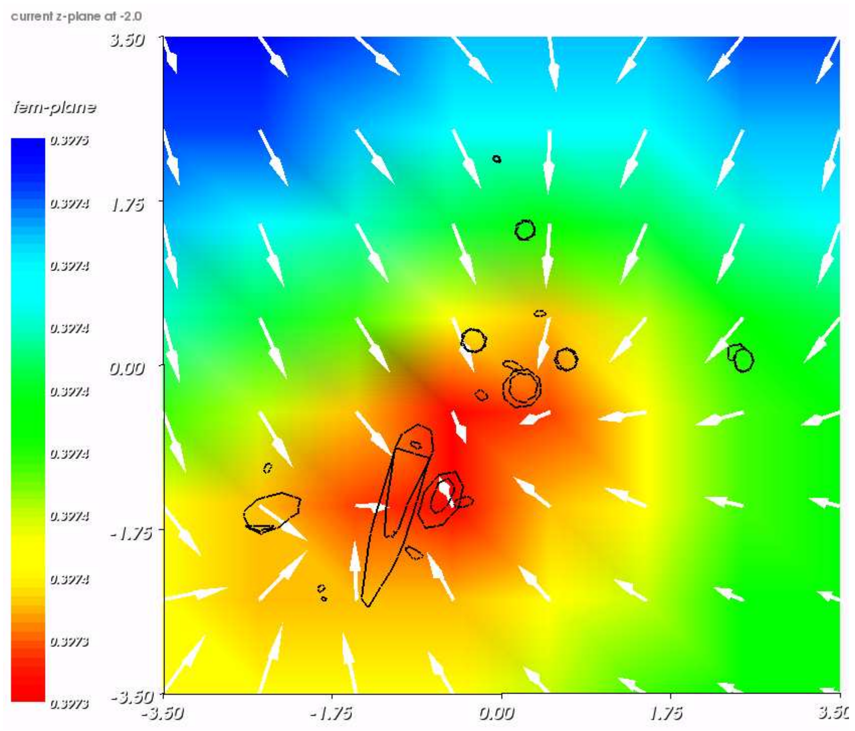


Figure 13: RSWMSViz graphical 2D output showing a vertical soil plane with intersections and velocity field

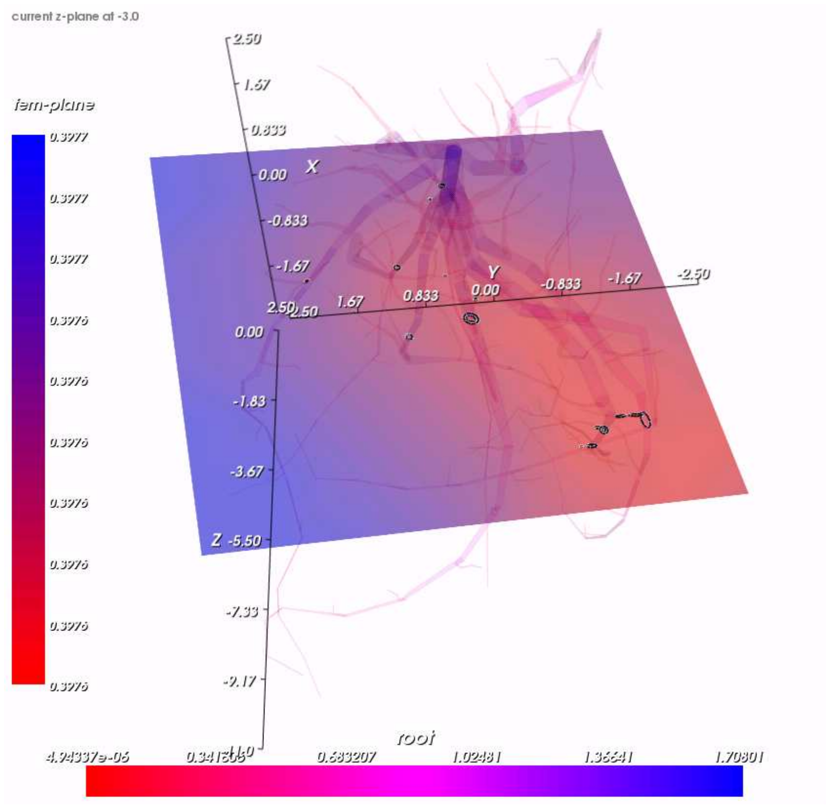


Figure 14: RSWMSViz graphical output showing a transparent root with soil plane intersections

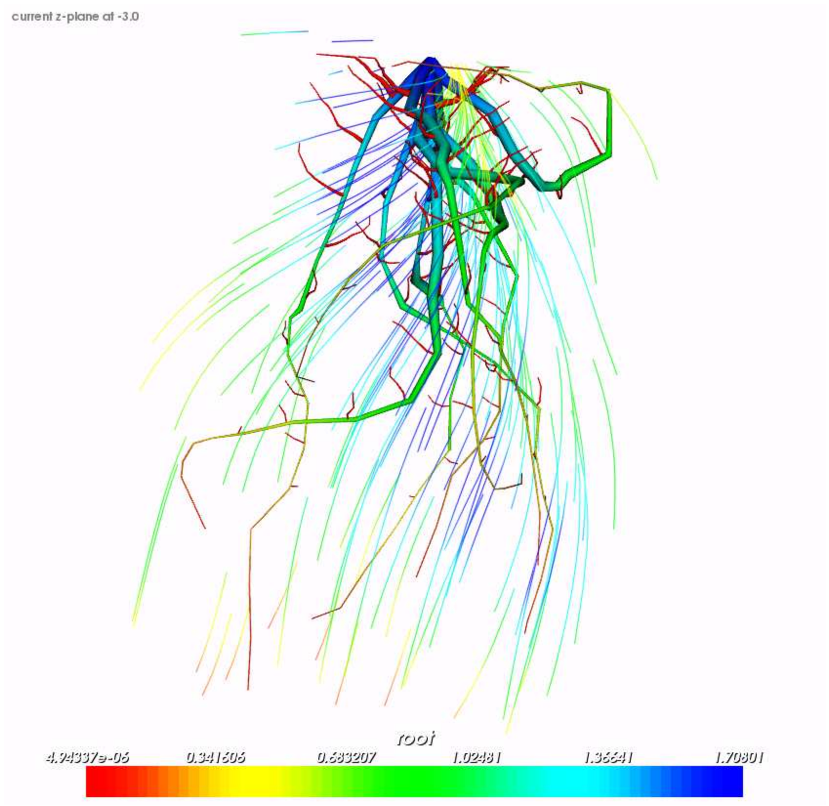


Figure 15: RSWMSViz graphical output showing a root with velocity streamlines

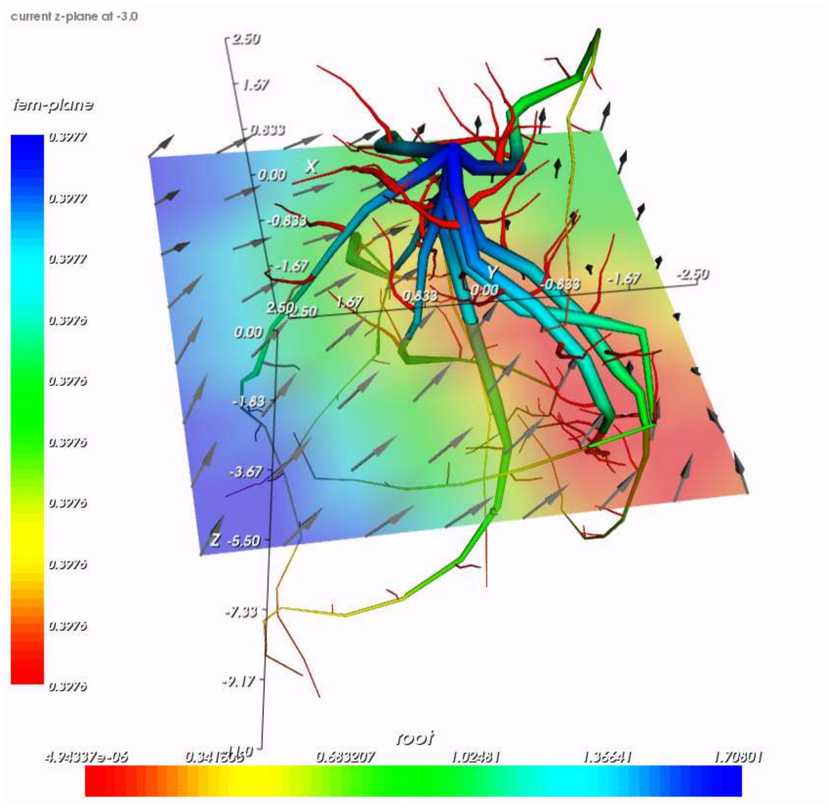


Figure 16: RSWMSViz graphical output showing a root with soil plane and clipped velocity field

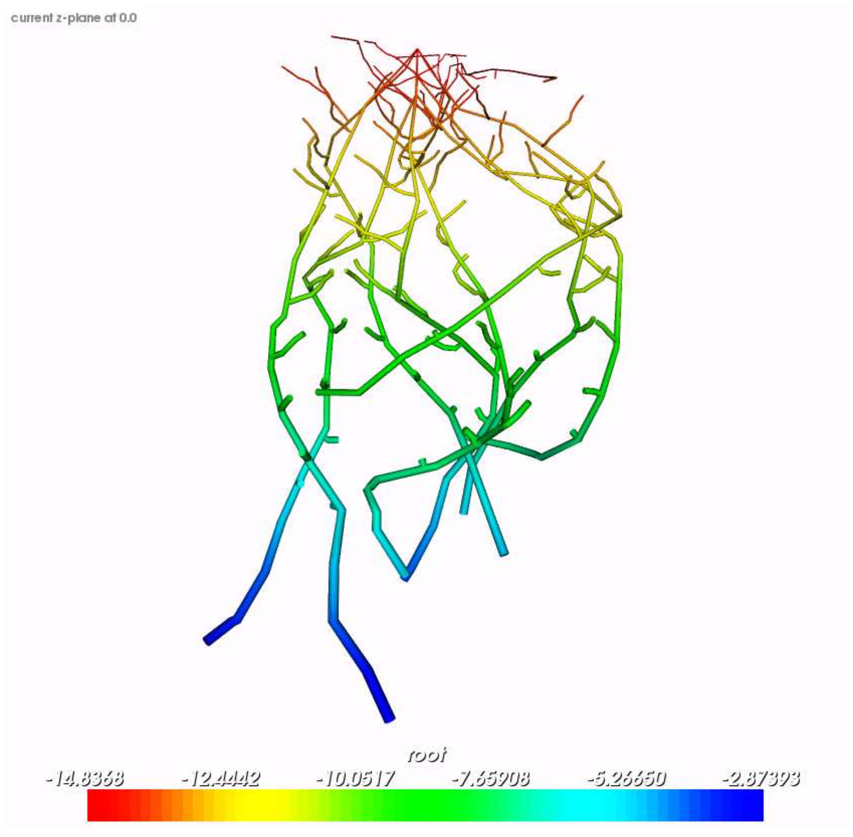


Figure 17: RSWMSViz graphical output showing a root with radius depending on mapping

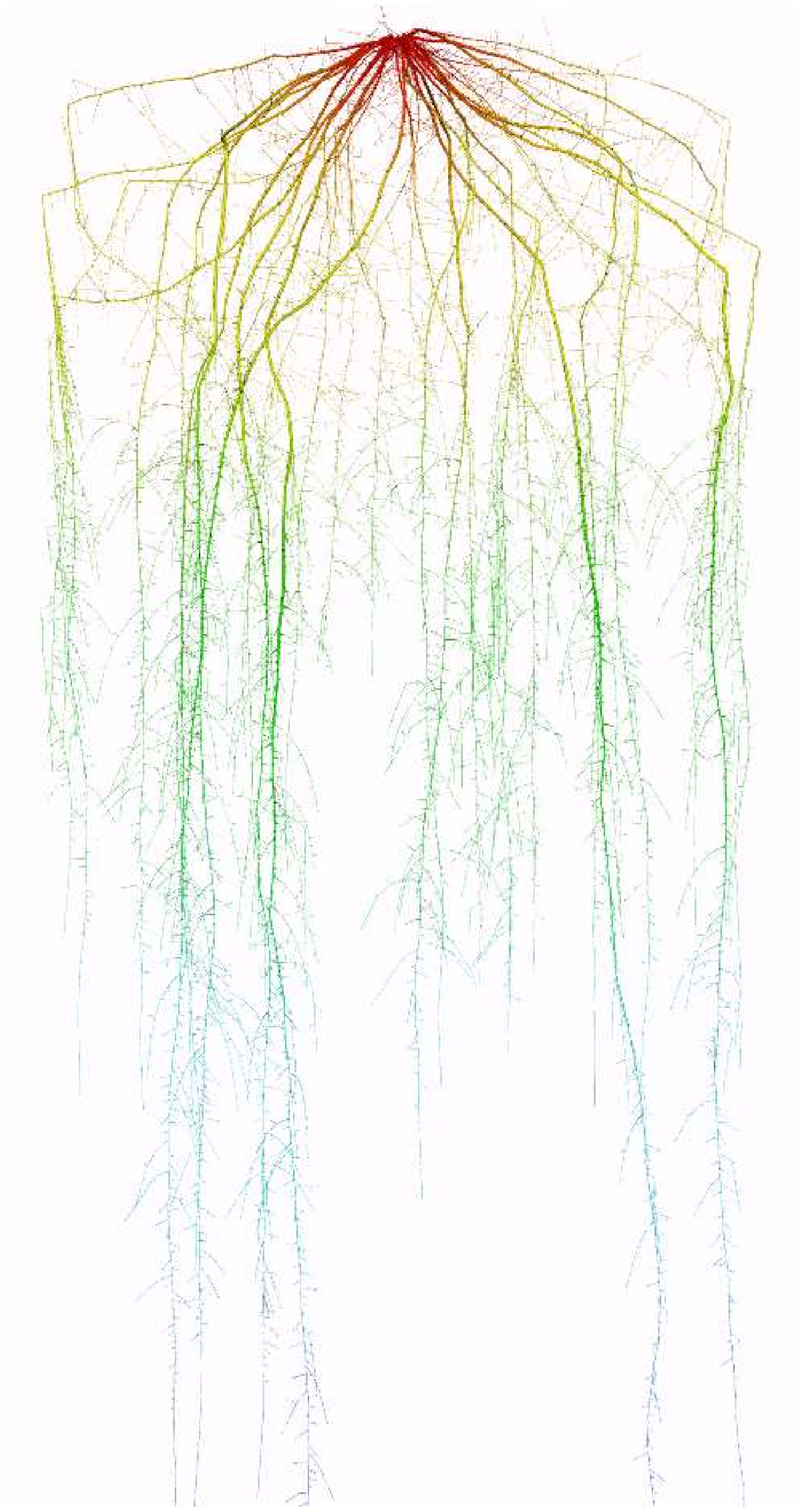


Figure 18: A complex maize root structure of 24510 nodes in a soil column of 12 by 12 by 150 cm, drawn by RSWMSViz with a proposed natural looking branch radius

Structure Analysis of Communication and Memory Behavior in the Parallel Tree Code PEPC

Martin Hoffmann

Martin-Luther-Universität Halle-Wittenberg
Institut für Physik
Von-Seckendorff-Platz 1
06120 Halle

E-mail: mart.hoffmann@gmx.de

Abstract: PEPC – (Pretty Efficient Parallel Coulomb-solver)+ – is an efficient, portable implementation of a parallel tree code based on the ideas of Barnes and Hut [1]. This code for rapid computation of long-range $(1/r)$ Coulomb forces is presented for use as a 'black-box' library for molecular dynamics applications. Previous experience with the code shows that for some particle configurations some processes have a lot more point-to-point communication than others, ultimately leading to storage imbalance. The causes of this problem are analysed with the help of additional statistical diagnostics and real-time visualisation of the tree structure.

Introduction

Two different algorithms for rapid summation of $1/r$ potential developed in the mid-1980s – the hierarchical Tree Code and the Fast Multipole Method (FMM) [2], with respective scalings of $\mathcal{O}(N \log N)$ and $\mathcal{O}(N)$ – have become the standard 'mesh-free' tools for long-range N-body simulation across a broad range of fields [3]. These methods reduce the number of direct particle-particle interactions through the systematic use of multipole expansions, making it possible to perform simulations with millions of particles.

PEPC was initially designed for mesh-free modelling of nonlinear, complex plasma systems [4], but recently extended to other application areas in molecular dynamics in the form of a transparent library. For the PEPC kernel, the Warren-Salmon 'hashed oct-tree' scheme based on a space-filling 'Morton' curve derived from 64-bit particle-coordinate keys has been adopted. The discontinuities inherent in this curve, potentially leading to disjointed domains and additional communication overhead [5], is found to be a relatively minor issue compared to load-balancing and geometrical factors.
The Hashed Oct Tree algorithm developed by Salmon and Warren is well documented [6]: features particular to the code PEPC are also described elsewhere [7, 8].
The structure of this report is as follows: First some fundamentals of oct-trees are introduced,describing the construction of particle keys and domain decomposition and give some definitions. We introduce the Plummer model and some statistics and analysis routines for their later usage. In the following sections we address two particular issues:

1. comparison between fetched particle keys in `tree_stats` and used particle keys in the interaction lists

2. analysis of the problematic processes with a large number of particles per process for a non uniform particle configuration

Fundamentals

Morton-Z-order and particles keys

We used binary coordinate keys to map the 3-dimensional spatial structure onto a one-dimensional space-filling curve. The keys are constructed from the binary interleave operation:

$$\text{key} = \text{placebit} + \sum_{j=0}^{nbits-1} 8^j (4\ \text{bit}(i_z, j) + 2\ \text{bit}(i_y, j) + \text{bit}(i_x, j)) \tag{1}$$

The function `bit()` selects the jth bit of the integer coordinate components (i_x, i_y, i_z), which are computed from:

$$\begin{aligned} i_x = x/s\,,\ i_y &= y/s\,,\ i_z = z/s \\ s &= L/2^{\texttt{nlevels}} \end{aligned}$$

(simulation box length: L, maximum refinement level: `nlevels`)

For a 64-bit machine we can provide 21 bits per coordinate (`nlevels`=20) plus a place-holder bit:

$$\texttt{placebit} = 2^{63}$$

This procedure yields a space-filling curve following the so-called Morton- or Z-ordering.

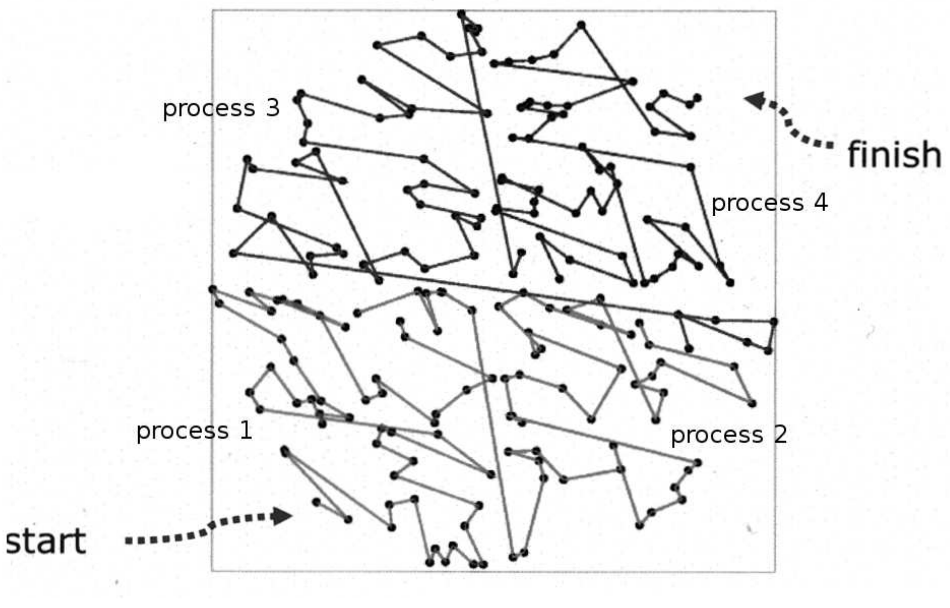


Figure 1: space-filling curve in two dimensions for four processes

In Fig. 1 we give an example of a space-filling curve for some processes. At the transitions from one process to another large discontinuities are observable.

Plummer model

Plummer's model is a standard model for a comparison of simulation methods. Particles are distributed randomly in a sphere with changing density. In the center of the sphere the density is nearly one and falls off with radius. In [9] the way to get initial coordinates is described in detail. In PEPC there is a lower limit for the separation of two particles so the Plummer model has be truncated in the center by removing some particles.

Definitions

In the following context we will use some parameters from the code and some phrases which are often repeated. Here we explain the used abbreviations:

$work(1:npp)$	- length of the interaction lists of all local particles, after `sum_force` it equals $local\_work$
$total\_work$	- summation of $work$ over all previous time steps
$fetches$	- number of all fetches keys in the actual time step
$ships$	- number of all shiped keys in the actual time step
npp	- local number of particles on process x

In the given example we analyse 'problematic processes', also called 'maxima'. In the context of this report it means processes with a larger npp than others.

Routines

`pbalsort`:

PEPC-B is equipped with a load-balancing function for domain decomposition. This function is adapted from [10]. After every time step $work$ is actualized. The value of $work$ is a kind of scaling factor. Processes with long interaction lists get fewer particles in the next time step and vice–versa. So it is possible to increase the value of $work$ with a additional factor greater than one or decrease with a factor less than one. At the beginning of the simulation or after restart the $work$ array is initialized with one.

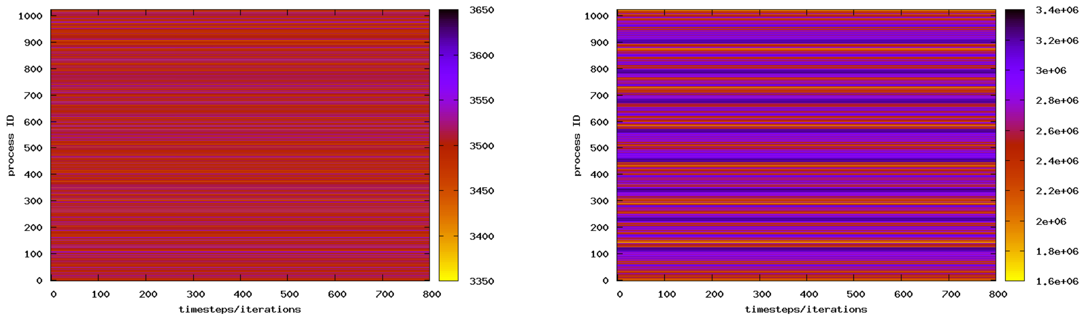
`tree_stats`:

This routine is a simple serial output routine. Rank zero collects all required information from the last time step and writes it to a single file. In addition to this simple task it looks for local maxima. In the actual code the searching takes place on the basis of npp but it is very easy to modify the code to use another values. In a `do-loop` it compares the value from every process with five neighbor processes on both sides. If the actual one has the highest npp `tree_stats` will select this process as a local maximum. The user supplies the number of accepted maxima and can specify:

- $barrier$: empirical value to accelerate the searching
  (mostly 5000 particles per process; if there are no processes found $barrier$ will decreased until zero)
- start and end number for processes for selective visualization, the span in which maxima are searched
- $neighbors$: number of processes next to a local maximum, which are taken, too (left & right)

Uniform example

PEPC-E is used for the analysis of a pure uniform arrangement. All particles are randomly distributed in a cube so that the density of the simulation area is nearly constant. Fig. 2(a) shows the same values like the first example (Fig. 4). But it is much more homogeneous during all time steps. Also the span between highest and lowest npp is not so high. The same result can be seen in the $total\_work$ map.



(a) particles (b) total work

Figure 2: per process

This map shows that PEPC works well for a uniform arrangement. Every process has nearly the same amount of particles and a relatively balanced total work. But the question is: In which way is the communication balanced?

Communication efficiency

This refers to for the quality of the point-to-point communication. If there are just a few unused keys nearly all performed point-to point communications are necessary and we can not spare any time at `tree_aswalk`. To analyse the usage of keys we store all particles keys from every timestep in files. All fetched keys can be found in `tree_aswalk` while all used keys are located in the interaction lists. It is enough when a key is used once in the interaction list so we have to compare both key lists. If a fetched key appears the first time on the interaction list a counter variable is increased. For different passes in `tree_aswalk` we use various counters to reveal distinctions in every pass.

Results

The first setup for this routine was $\Theta = 0.0$ (Table 1). This means that every particle interacts with each other so we should find every particle in the interaction list. In PEPC there are two routines where non local keys are fetched from other processes: `tree_aswalk` and `tree_branches`. The keys from the branch nodes are well known and there is no redundant point-to-point communication. The routine `tree_aswalk` may request much keys which are not used in `sum_force`. It seems that many parent nodes have to be requested to get few children nodes. We try two highly contrasting distributions of particles. The first one is the same random arrangement which is used above and the second one is the aforementioned Plummer model.

The routine can be controlled by summation of the found keys, the local particles and the keys which are fetched in `tree_branches` for $\Theta = 0$. This value equals the number of all particles. That means that the routine works correctly. Tab. 1 shows that `tree_stats` has to fetch appr. 30 per cent more particle keys than it use for the force summation if all particles interacted with each other.

cube; random			Plummer model		
average found keys in the interaction lists for pass 1:					
7709.31			7661.84		
average	max value	min value	average	max value	min value
difference between all found keys and all fetches					
3341.22	3358	3327	3259.84	3275	3243
sum of all fetches					
11050.53	11071	11036	10921.69	10938	10903
difference in per cent					
30.24	30.33	30.15	29.85	29.94	29.74

Table 1: for 32 cpus , 8192 particles, $\Theta = 0.0$

The values above are interesting but they are mostly uncommon for a real simulation with $\mathcal{O}(N \log N)$. A more typical example is the use of $\Theta = 0.6$ (Tab. 2). It seems that PEPC uses more fetched keys for

cube; random			Plummer model		
average found keys in the interaction lists for pass 1:					
946.69			1057.53		
average	max value	min value	average	max value	min value
difference between all found keys and all fetches					
0.13	1	0	0.25	3	0
sum of all fetches					
946.81	1370	525	1057.78	1755	429
difference in per cent					
0.012	0.124	0.00	0.029	0.281	0.00

Table 2: for 32 cpus , 8192 particles, $\Theta = 0.6$

`sum_force` than expected. For this example PEPC used nearly all keys. Also for the Plummer model (Tab. 2) which is a very non uniform distribution the difference is not much higher. One reason could be the small number of particles which we used. To verify this hypothesis we also examined a larger example (see Tab. 3). In this table we see a bigger difference than before but a average variation of 13 per cent is not so bad. Also we can see small variations between different passes. It is remarkable that the variation from time step two to three is tiny. But from pass 1 to pass 2 there a large variation of the

time step 2			time step 3		
average found keys in the interaction lists for pass 1:					
15977.38			15976.99		
for pass 2:					
3208.67			3209.20		
average	max value	min value	average	max value	min value
difference between all found keys and all fetches					
3359.17	9183	23	3358.92	9182	23
sum of all fetches					
22545.23	36614	7391	22545.12	36615	7391
difference in per cent					
13.48	27.12	0.31	13.48	27.12	0.31

Table 3: for 512 cpus, 524288 particles, $\Theta = 0.6$

used keys. In pass 2 only 20 per cent of keys are found in comparison to pass 1. In the first step the most particle keys are fetched and they are not needed in the next step. It indicates that all essential keys could be fetched in single pass.

Non-uniform example

This example use the front end PEPC-B with the load-balancing function. It is designed to simulate laser-plasma interactions.

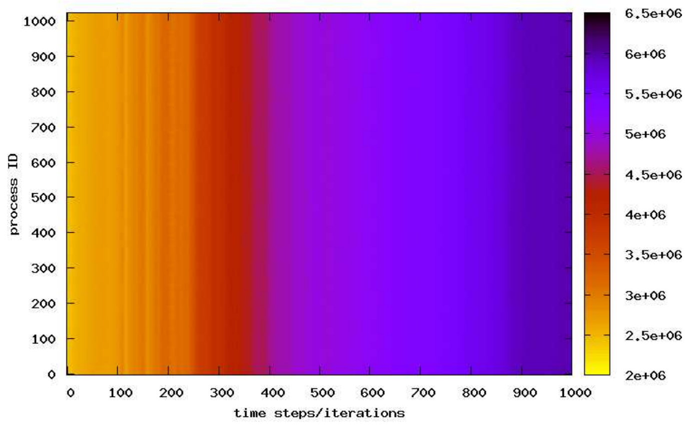


Figure 3: $total\_work$ per process

Problem

This example starts with a thin layer of paarticles which are heated by a laser. After the first time step particles spread out from the layer and move in different directions. This results in different densities in the simulation area.
From the work of Zoltán Szebenyi [11] it is known that some processes cause high point-to-point communications. The values of point-to-point-communication are measured with SCALASCA. It is also known from this work that this fact is related to a high number of particles on certain processes. With the routine `tree_stats` it is possible to illustrate the same attributes after every time step. One can see the same pattern in a map of ships per process for all 1000 time steps and 1024 processes. In contrast a map with $total\_work$ per process is very homogeneous (see Fig. 3). As a conclusion in this context the load-balancing function seems to work very well.

Imbalanced number of particles

This fact leads to some questions:

1. Where are these processes located?
2. Why are there such high numbers of particles?
3. Why do they have some related patterns?
   for example: two maxima move apparently together, some maxima appear or have a bend in the same time step

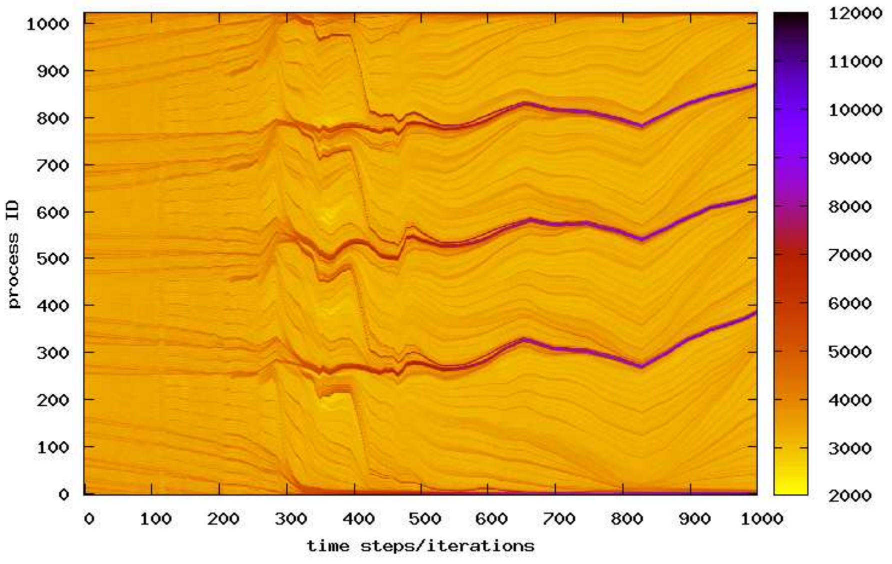


Figure 4: particle per process

In this concrete example there are 5 maxima. Fig. 4 shows that the maxima are not defined by only one process. There is always a group of processes with too many particles. Also the first and the last process provide a maximum.
To answer these questions one have to track the moving of the problematic processes. This problem is solved in `tree_stats` (see section). For the report mostly we set this value to 8 (in `run.h` – $num\_bad\_process$) because we cannot expect to find always exact the 5 maxima of Fig. 4.

The result is shown in Fig. 5. The only problem is that there is only one process for a group of processes with high npp's. But this will corrected by the value of $neighbors$.

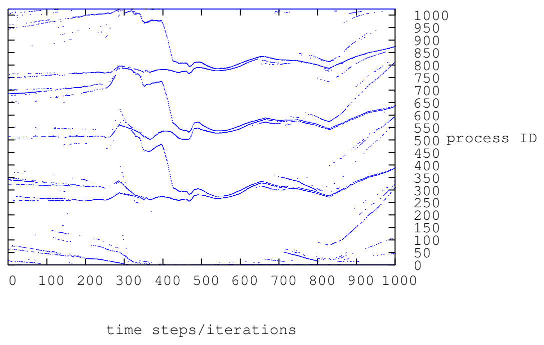
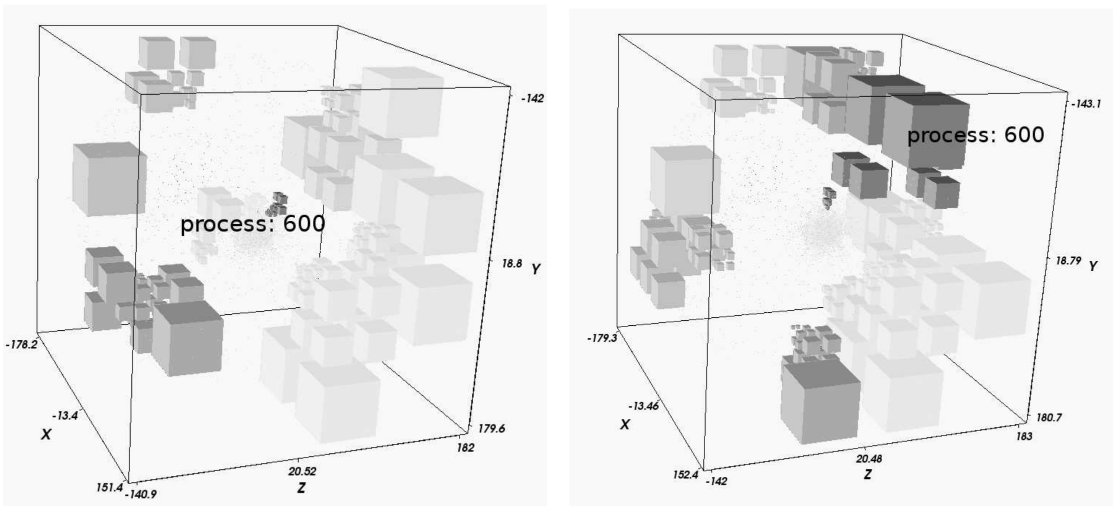


Figure 5: show all tracked processes

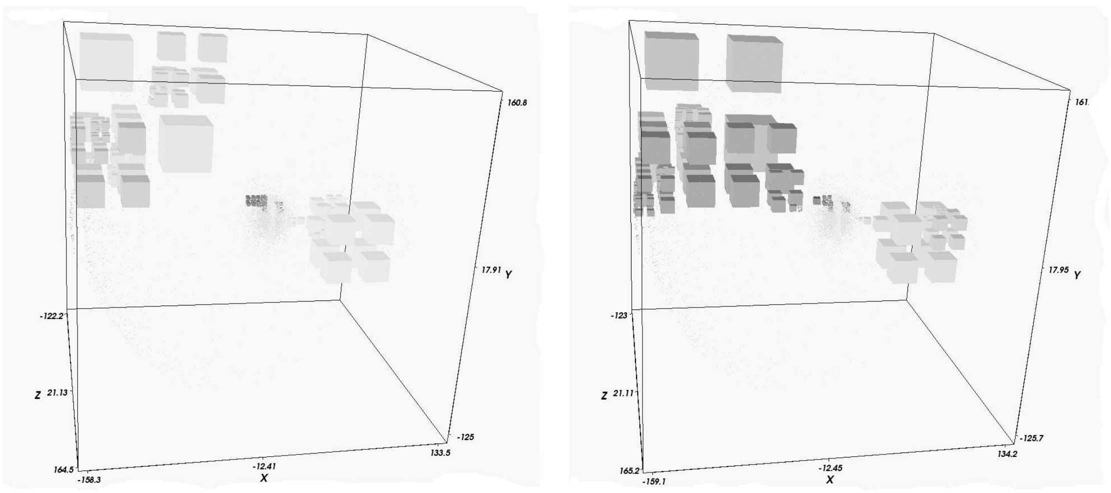
Visualization – 1. question On Jugene one can use netcdf to show the location of the problematic processes. After a complete pass of PEPC-B (here mostly 1000 time steps) the netcdf file is sent to XNbody via `ncreader`. Here visualization mode 3 is used: All branches for a given list of processes are sent to XNbody as colored boxes. This shows the location of a interesting process. The user can switch the color code over to three different characteristics of the selected process. Fig. 6 shows the branch boxes of some processes, in particular is the location of process 600. As Fig. 4 indicates process 600 gets between time step 900 and 1000 a part of the maxima.



(a) (b)

Figure 6: tracking process 600 (the darkest color)moves to the peripheral area and get a lot of particles from (a) to (b), see Fig. 4 time step 900-1000

Process 600 moves from the area of high density to the peripheral area. Other shown processes are a maxima, too, but the selected color code was not chosen well. There is no possibility for a comparison of the two pictures for these processes. Although one can see that all problematic processes are located in the outer border region with low particle density.



(a) (b)

Figure 7: tracking one maximum with 4 neighbors to show the movement in every step, example at bend from time step 900 to 1000 and from process 200 to 400, the dark colors are high cpu-id's

In Fig. 7 it becomes apparent that some processes completely change their locations between different time steps. This can be explained with the characteristics of the Z-order while domain decomposition and the sorting routine. These two parts are the only one which affect the dispersal of particles per process.

Large number of particles equals peripheral area – 2. question The question of the location of the problematic processes leads us directly to an answer of the second question. All processes with high number of particles seem to be located at the border of the simulation area. As mentioned above this area has a low density of particles. This should result in short interaction lists of these processes. The $work$ arrays of these processes have a lower value than of the processes in the main area with high density. Processes at the border of the simulation area will therefore get more particles.
This fact is not important at the beginning of the simulation. The particle distribution density is nearly the same in every part of the simulation area. After some time the particles start to move away and the density is not uniform any more. Processes with high npp have to ship around much more particle informations than others (cf. paragraph problem on page 32). The same pattern can also be found in the value of fetches per process but these processes are minima there.

Possible explanation – 3. question The first and last process are always a problematic because of the fixed start and end position of the Z-order (always at the border). It seems that the other patterns depends on the spreading of the particles during every time step.

First solutions

The length of the interaction list is a vital parameter for the scaling ability of PEPC. It affects the memory at each process for local and non local nodes. But a significant high amount of point-to-point communication is not acceptable for the whole algorithm and it will slow down the calculating.
Maybe it is possible to smooth these maxima with few changes of the code. In the next section we describe and test some ideas .

Naive approach – reset $work$

The simple reset of this array has no effect. After 200 time steps $work$ gets the value of one for all processes. But in the next time step the same pattern appears again and the same distribution in the values of npp and $total\_work$ as without reset.

Additional scaling

For a good smoothing $work$ is reduced for some processes so that these get fewer particles.

$$work_p(1:npp_p) = work_p(1:npp_p) \cdot \frac{npp_p}{avg\_npp} \tag{2}$$

(process p, average particle/process avg_npp)

This scaling was used with different amounts of processes. The idea of taking one process per peak coming from `tree_stats` was ineffective. The surface of the heat map for npp is smoother but the high values remain. For Fig. 8 more than one process is scaled. The maxima are displaced to the upper and lower processes.

The whole map seems much more uniform than before but it is conspicuous that the scale of the right side has a bigger span.

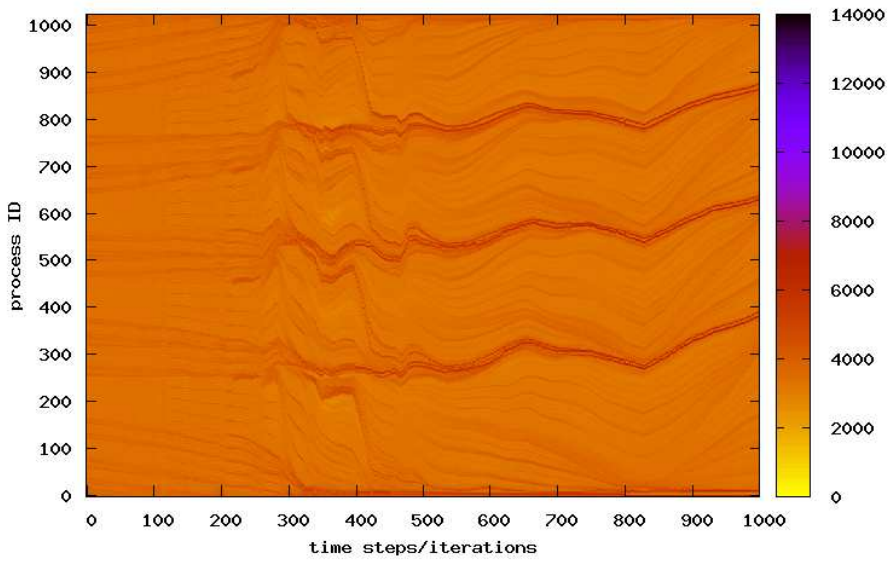


Figure 8: take ± 2 processes of the problematic one

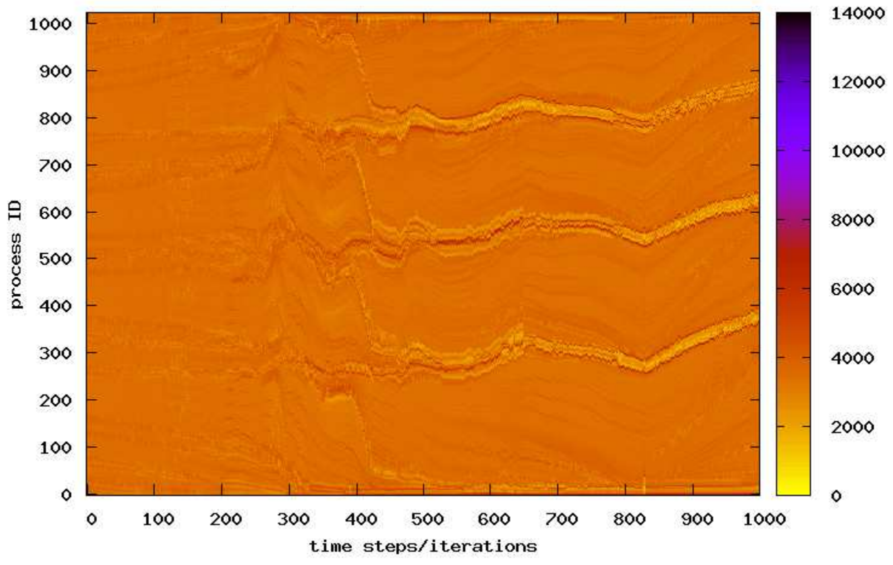


Figure 9: take ± 5 processes of the problematic one and scale with his npp

This effect is not sufficient for the neighbors so one can use the highest npp for all neighbors:

$$work_p(1 : npp_p) = work_p(1 : npp_p) \cdot \frac{npp_{\widehat{p}}}{avg\_npp} \quad (3)$$

$$\forall\, p > \widehat{p} - 5 \cap p < \widehat{p} + 5 \quad (4)$$

(process from `tree_stats` $\widehat{p}$, actual process p, average particle/process avg_npp)

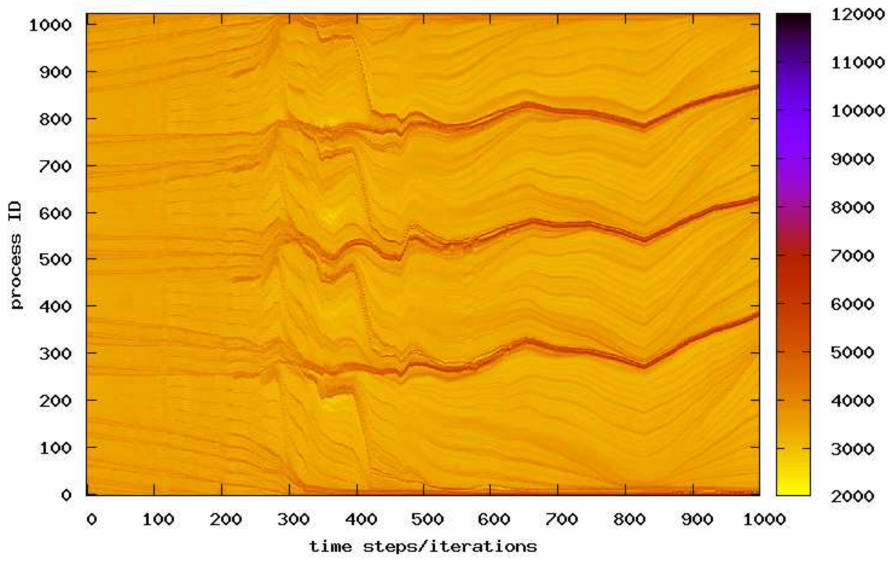


Figure 10: take ±5 processes of the problematic one and scale with $\widehat{npp}$

Fig. 11 shows that the above methods are not efficient enough to smooth the high Peaks well. If we take a average value for npp then large numbers of particles will influence their neighbors too:

$$\widehat{npp}_p = \frac{1}{Q}\sum_q^Q npp_q \;\forall\, q > p - (Q/2 - 1) \cap q < p + (Q/2 - 1) \quad (5)$$

$$\text{here we use } Q = 11 \quad (6)$$

$$work_p(1 : npp_p) = work_p(1 : npp_p) \cdot \frac{\widehat{npp}_p}{avg\_npp} \quad (7)$$

The result in Fig. 10 looks like the maps before but a significant difference is another scale span on the right. There are no larger maxima as before (see Fig. 4). The seen displacement of the maxima has also nearly vanished although the area of smoothing was much bigger than before. Per Peak we get 11 processes from `tree_stats` and the average was taken from 11 processes, too. This results in a span of 21 processes per peak. The map shows that there is a possibility to affect the behavior of `pbalsort` positively from 'outside' the routine.

Comparison

To get a overview of all previous ideas Fig. 11 shows the ratio between the maximum of the actual time step and the average number of particles per process.

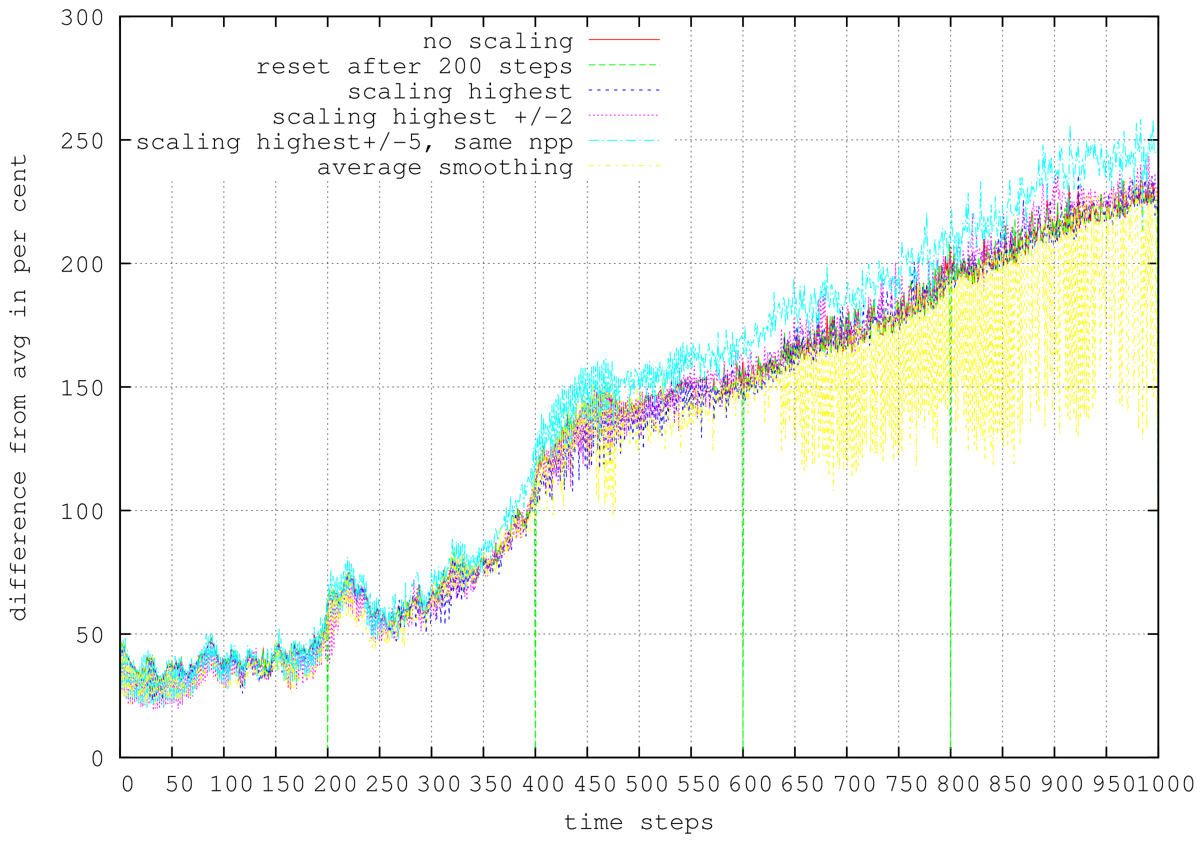


Figure 11: every time step; particles per process: $(\frac{\text{max}}{\text{avg}} - 1) * 100$

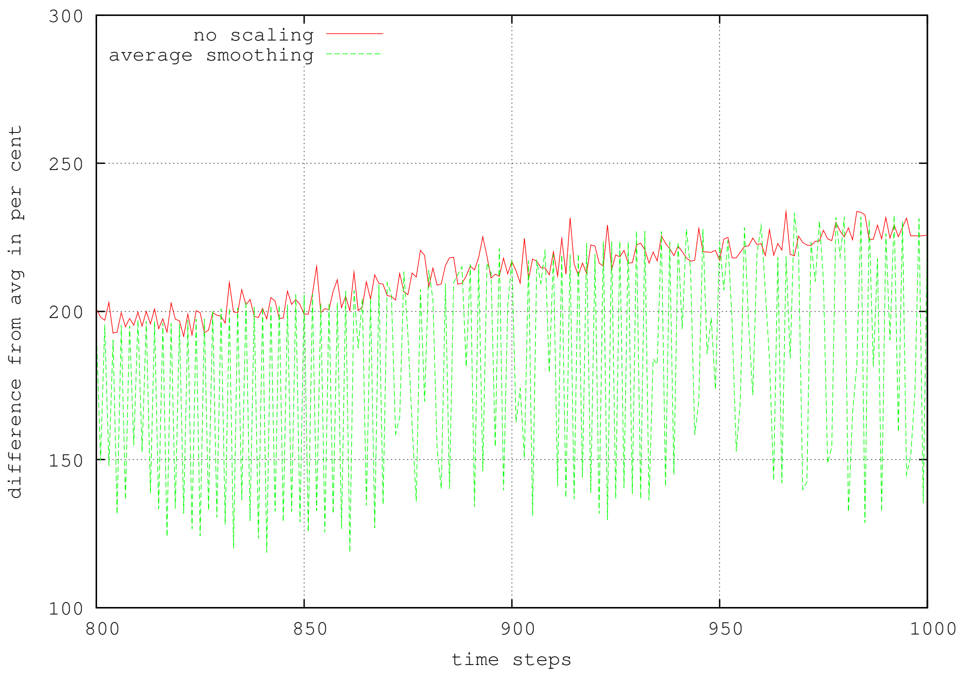


Figure 12: time step from 800 to 1000; particles per process: $(\frac{\text{max}}{\text{avg}} - 1) * 100$, more details

In this case we would like to have a small factor between the average and the maximal values. The only idea which provide a smaller ratio than the no-scaling curve is the last one. However, it seems that the number of particles increases after the smoothing every second timestep (Fig. 12). This problem could easily be fixed by an array which contains all scaled processes from few previous time steps or maybe only the previous one.

Conclusions

With the remarks from above we see that PEPC still have problems with different densities in the simulation area. The uniform cubic distribution shows only small variations in the vital parameters of particles, total work, fetches and ships per process. Most of the fetched keys are needed for force summation even for the Plummer model. This is an unexpected result because of the large density difference for this particle distribution. Such differences determine in the second example large problems. However, the usage of fetched particle keys seems to do well. But for all that it could be correlated to the small number of used processes and particles.
The second part of this report concerned a laser-plasma interaction scenario, results in a very inhomogeneous particle distribution. We showed the behaviour of some processes which moved at the border of the simulation area, receiving much more particles than the average npp. It seems that all processes with a large number of point-to-point communications are located on the periphery. To smooth these maxima it is not enough to choose the simple way of an additional scaling with the local npp. A better correction is the usage of an average npp over some neighbor processes. We were able to reduce high values at the particle per process map without a displacement of the peaks.
The location of the maxima and their behaviour over some timesteps seems to be determined by the progress of the Z-curve and the propagation of the particles away from the center. A solution of this problem could be another space-filling curve and seperate handles of variable density. The resolution of local trees at processes at the border could be reduced for processes in the center. But it is not known in which way this action would interference with particles in an intermediate area. Another possibility could be a group of predefined processes which manage the whole tree information and perform the main point-to-point communication for other processes.

Acknowledgements

I would like to thank my supervisors Dr. Paul Gibbon and Robert Speck for their contributions to this project. I would also like to thank the organisers Matthias Bolten and Robert Speck for enabling me to participate in the program and all staff of JSC for excellent working conditions.

References

1. P. Hut, J. Barnes. A hierarchical o(nlogn) force-calculation algorithm. *Nature*, 1986.
2. V. Rokhlin, L. Greengard. A fast algorithm for particle simulations. *J. Comp.Phys.*, 1987.
3. S. Pfalzer, P. Gibbon. *Many-Body Tree Methods in physics*. 1996.
4. R. G. Evans, E. L. Clark, M. Zepf, P. Gibbon, F. N. Beg. Tree code simulations of proton acceleration from laser-irradiated wire targets. *Phys. Plasmas*, 2004.
5. A. Sameh, A. Grama, V. Kumar. Scalable parallel formulations of the barneshut method for n-body simulations. *Parallel Comp.*, 1998.
6. J. K. Salmon, M. S. Warren. A portable parallel particle program. *Comp. Phys.Commun.*
7. P. Gibbon. Pretty efficient parallel coulomb-solver. Technical report, Forschungszentrum Jülich GmbH, 2003.
8. S. Dominiczak, B. Mohr, P. Gibbon, W. Frings. Performance analysis and visualization of the n-body tree code pepc on massively parallel computers. In *G. Joubert, et al. (eds.), Proc. Parallel Computing , Malaga*.
9. R. Wielen, S. J. Aarseth, M. Hénon. A comparison of numerical methods for the study of star cluster dynamics. *Astron. & Astrophys.*, 1974.
10. X. Li, P. Lu, J. Schaeffer, J. Shillington, P. S. Wong, H. Shi. On the versatility of parallel sorting by regular sampling. *Parallel Computing*, 1993.
11. F. Wolf, Z. Szebenyi, B. J. N. Wylie. Scalasca parallel performance analyses of pepc. *EuroPa 2008, workshop proper 2008*.

Protein Simulations Under Constraints

Christoph Honisch

Westfälische Wilhelms-Universität Münster
Institut für Theoretische Physik
Wilhelm-Klemm-Str. 9
48149 Münster

E-mail: c.honisch@uni-muenster.de

Abstract: Structure prediction of proteins is an important task in biology. There are two principal ways – among others – in which computational methods can contribute to tackle this problem. The first approach consists of computer simulation using molecular dynamics or Monte Carlo based on physical force fields. The second approach uses statistical predictions derived from protein databases of known conformations. In this article a method is proposed to combine the physical and the knowledge based ways of structure prediction by using statistical predictions as constraints in Monte Carlo simulations.

Introduction

Proteins are macromolecules that consist of a chain of amino acids. Each chain folds itself into a unique macroscopic 3d structure, the so called *tertiary structure*, which is completely determined by its sequence of amino acids (*primary structure*). The tertiary structure determines the functionality of the protein. So, understanding of the functionality of a protein requires knowledge about its tertiary structure. Determining those structures experimentally is a very time-consuming and expensive process, so there is a strong demand for computer-based structure prediction.

This problem can be tackled from two directions, physical computer simulations and knowledge-based statistics. Common methods for physical computer simulations are molecular dynamics and Monte Carlo, which both have advantages and disadvantages. The latter method was used in the work of the present article. The knowledge-based approach uses databases of protein conformations collected by experimentalists. Given a specific target protein one can search those databases for proteins with local or global sequence similarities and use statistical methods to derive a prediction of the configuration of the target.

Both ways of structure prediction can be combined by using the knowledge-based predictions as constraints in Monte Carlo simulations. The implementation and testing of such constraints in the MC simulation program SMMP ([1], [2] and [3]) was my task during the guest student program of the NIC.

The next section of this article will give a short introduction into proteins and the MC algorithms that were used. Afterwards the implementation of the different constraint types is explained followed by two sections about examples in which the impact of the constraints was tested on sequences whose structure is known.

Proteins

Proteins belong to the wider class of peptides, i.e. they are macromolecules consisting of a chain of amino acids. Typical sizes of proteins range from 20 to 3000 amino acids, in this context also called *residues*.

There are 20 different types of residues most proteins are built of. The characteristic attribute that distinguishes proteins from other peptides is that each protein assumes a unique macroscopic folding shape, only determined by its sequence of residues. This was demonstrated in the famous *Anfinsen experiment* [4], named after the chemist and winner of the Nobel prize Christian B. Anfinsen.

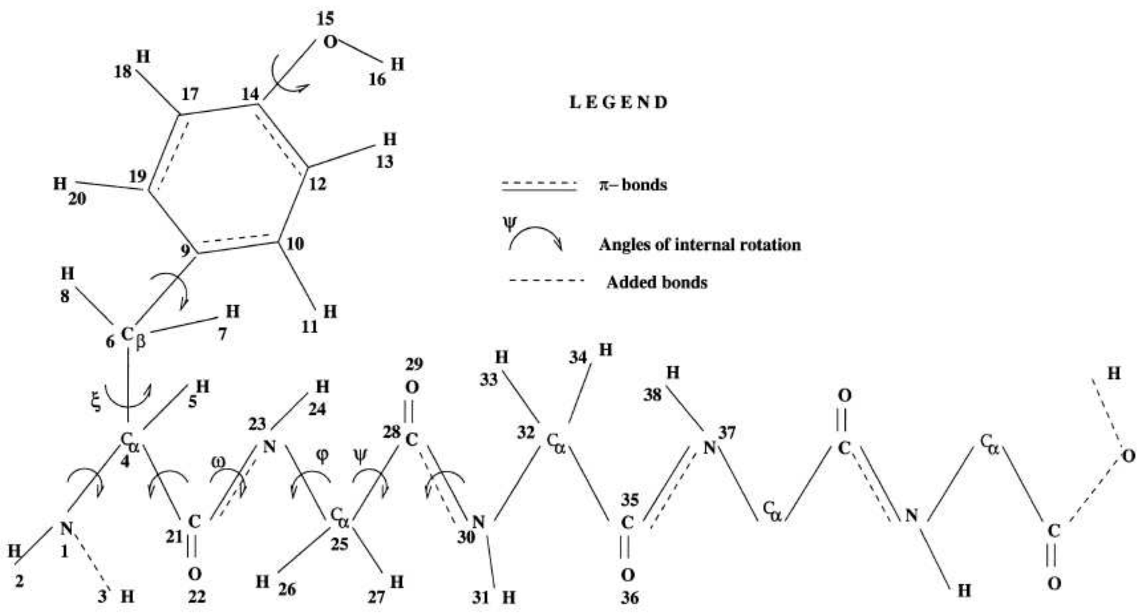


Figure 1: The small peptide Met-enkephalin. The image is taken from [1].

The buildup of a protein is of the form

$$\ldots \underbrace{NH - C_\alpha R - CO}_{\text{Residue } i} - \underbrace{NH - C_\alpha R - CO}_{\text{Residue } i+1} \ldots$$

The repeating atoms N, C_α and C build the backbone of the protein. One residue specific side chain R is attached to each C_α-atom. As an example Fig. 1 shows the buildup of the small peptide Met-enkephalin, consisting of five residues.

Describing the structure of proteins one distinguishes between the primary, secondary and tertiary structure. Primary structure just means the sequence of residues the protein is built of. The secondary structure is defined by hydrogen bonds between nearby residues that lead to specific local conformations. The most common ones are α-helices and β-sheets as shown in Fig. 2. By tertiary structure one means the macroscopic folding scheme of the chain.

The way proteins achieve their stable folded conformation is not yet sufficiently understood. At this point the *Levinthal paradox* [5] should be mentioned. This is a small sample calculation that makes clear that the folded conformation cannot be found just by random fluctuations. In that case the folding process of an average sized protein would take something in the region of 10^{24} years. In reality it happens at time scales between milliseconds and minutes. It is therefore assumed that there are mechanisms that help proteins to find their tertiary structure.

Proteins fulfill a variety of essential tasks in animate systems. There are, e.g., catalytic proteins, transport proteins, or signaling proteins, just to name a few. The functionality is closely related to the tertiary structure. It can happen that proteins do not fold into their native conformations. Some diseases can be traced back to so called *misfolded* proteins. BSE is a famous example. Until now it is not understood why and under which circumstances proteins misfold.

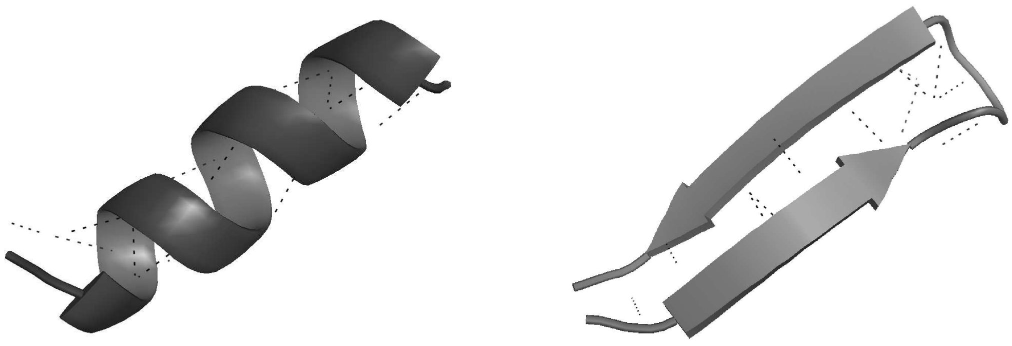

(a) α-Helix **(b)** β-Sheet

Figure 2: The two most common secondary structure elements. The dotted lines denote hydrogen bonds.

Protein Simulations

There are two main problems one tries to tackle with protein simulations. At first one wants to obtain a better understanding of the folding process itself. The second task is structure prediction. This article only deals with the latter.

For Monte Carlo simulation the protein is viewed as a thermodynamic system with fixed number of particles exposed to a heat bath (the solvent), which keeps the temperature fixed. Such a system is called a canonical ensemble and satisfies the Boltzmann distribution

$$p_i = \frac{1}{Z} \mathrm{e}^{-\beta E_i} \quad , \quad \beta = \frac{1}{k_B T} \tag{1}$$

that defines the probability to find a state i with energy E_i in a system at temperature T. Z is a normalization constant being called the partition function.

As we know from statistical mechanics such systems always equilibrate to a configuration with a minimum in the free energy. So the aim of structure prediction is to find that minimum. Two problems are connected to that aim.

The first problem is that the configuration space is huge. To restrict the degrees of freedom to a minimum all bond lengths and bond angles are set to fixed values and only allows the molecule to change some torsion angles that are displayed in Fig. 1. This is the standard geometry model that is also used by the force fields implemented in SMMP. In this geometry model the protein has an average of about five to six degrees of freedom per residue. This is still a large configuration space to sample.

The second problem is that the correct potential energy is not known and until now there is no sufficient approximation in which the energy minima always coincide with the native conformations. So even if you had sampled a big part of the configuration space and are sure to have found the global energy minimum, you still cannot be sure that you also have found the native structure.

The Metropolis Monte Carlo Algorithm

The key idea of MMC is to construct a Markov chain that underlies Eq. (1). This is done by an *acceptance rejection* method which leads to the correct transition probability. The algorithm consists of the following steps:

1. Start from an initial configuration with the energy E_{old}

2. Change one or more degrees of freedom in a random way (move).
3. Calculate the new energy E_{new}.
4. Accept the move with the probability $p = \min\{1, \mathrm{e}^{-\beta(E_{\text{new}} - E_{\text{old}})}\}$.
   - If the move was accepted set $E_{\text{old}} = E_{\text{new}}$.
   - If the move was rejected reverse it.
5. Continue with step 2.

Performing an infinite number of steps it is assured to find the global energy minimum. To find that minimum in finite time is connected to the problem that the energy landscape is very rough. There are many local minima separated by large barriers. If one chooses a low temperature that implies low probabilities to accept moves leading to higher energy states one gets easily stuck in local potential wells for long times. For high temperatures the probability is high to jump out of potential wells before reaching their minima. An ansatz to overcome that problem is *parallel tempering*.

Parallel Tempering

In this method one creates N replicas of the target protein and starts an independent MC run for each replica at a different temperature T_i. After a certain number of iterations replica with “adjacent” temperatures T_i and T_j exchange their conformations with the probability

$$p = \min\{1, \exp\left(-\beta_i E_j - \beta_j E_i + \beta_i E_i + \beta_j E_j\right)\}. \tag{2}$$

This has the effect that low energy conformations are handed over to low temperatures.

Constraints

As already mentioned in the introduction another way of performing structure prediction besides physical computer simulations is to make statistical predictions based on the knowledge of conformations one has about other proteins. Both methods can be combined by using the knowledge-based predictions as constraints in MC simulations. The idea behind this is to focus the search around more probable configurations. The two types of constraints that were implemented are distance and dihedral constraints.

Distance Constraints

Distance constraints were implemented as an additional energy term that depends on the distance r between the C_α atoms of two different residues and has a minimum at the expected distance r_c.

A variety of analytic forms of such a potential is possible. Three different forms were tested:

$$E_1(r) = C_1(r - r_c)^2 \tag{3}$$

$$E_2(r) = C_2\left(\frac{r - r_c}{r_c}\right)^2 \exp\left[\left(\frac{r - r_c}{0.2r_c}\right)^2\right] \tag{4}$$

$$E_3(r) = C_3 \exp\left[\left(\frac{r - r_c}{0.2r_c}\right)^2\right] - 1 \tag{5}$$

Another parameter besides the analytic form of the potential is its strength, i.e., the factors C_i. C_2 and C_3 were set to one for an initial test. For Eq. (3) an algorithm was implemented that fits the parameter

C_1 during the simulation. It works in a way that large deviations from r_c lead to an increase of C_1 while small deviations lead to a decrease. The idea behind this is to make the constraint energy only as high as necessary so that the loss of impact of the physical energy is as low as possible. The aspired setpoint is that the relative deviations $|r - r_c|/r_c$ are in 80% of the time less than 0.2 .

Dihedral Constraints

The dihedral constraints were implemented in form of an additional move in the metropolis algorithm that prefers the expected dihedral angles. This move selects a new value of the current dihedral angle from the von Mises distribution

$$P(\phi) = \frac{\exp\left[\kappa \cos(\phi - \phi_c)\right]}{2\pi I_0(\kappa)} . \tag{6}$$

This distribution function can be viewed as a periodic analogon to the Gaussian distribution where phi_c is analog to mean value and κ^{-1} is analog to the standard deviation. $I_0(x)$ is the modified Bessel function of order 0.

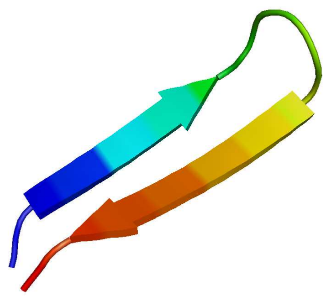
In this case there is no change in the evaluation of conformations – as in the case of the distance constraints – but in the sampling of conformations.

Technical Aspects

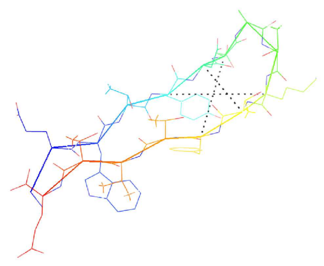
The constraints were implemented in the software package SMMP ([1] - [3]). This package written in standard Fortran is freely available on the internet. It contains several modern Monte Carlo algorithms, minimization routines and other useful tools for the simulation of proteins. Three different energy functions are implemented. In the examples presented in this article only the Lund force field was used. See [3] for a description of this force field.

Example 2: The Hairpin

In this section an example is presented where the different constraint types are tested and compared. The target is a small sequence that was cut out from protein G. Its native structure is shown in Fig. 3a.



(a) Cartoon. In this illustration the secondary structure is pointed out.



(b) Technical. Each line represents a chemical bond. The black dotted lines connect the pairs of atoms under the influence of distance constraints.

Figure 3: The hairpin in two different representations created by the *PyMOL Viewer* (http://pymol.org/).

Six different runs were started:

- Native dihedral constraints
- Native distance constraints with three different potentials
- Statistically predicted dihedral constraints
- No constraints

To test the effect of correct constraints four runs with native constraints, i.e. dihedral angles and distances taken from the native structure, were started. One using dihedral constraints and one for each distance constraint potential. Those constraints only influenced the turn of the molecule.

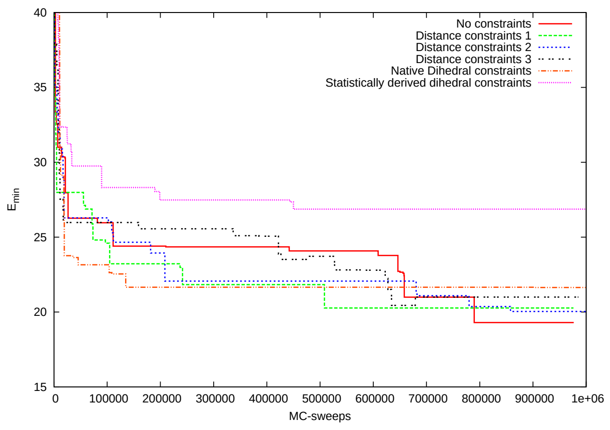
The dihedral constraints acted on the φ and ψ angles, i.e. the backbone dihedral angles, (see Fig. 1) for the residues within the turn of the hairpin. The atom pairs subject to distance constraints are highlighted in Fig. 3b by the black dotted lines. Another run was started using dihedral constraints that were statistically predicted. In this case the φ and ψ angles of all residues were used. Additionally one run without any constraints was performed for comparison.

In each of the six cases a parallel tempering run using 16 temperatures ranging from 274 K to 374 K was made. Starting from a random conformation one million Monte Carlo sweeps were performed, while a Monte Carlo sweep consists of n Metropolis steps, as described above, where n is the number of degrees of freedom of the molecule. Each run was done twice with two different seeds of the random number generator.

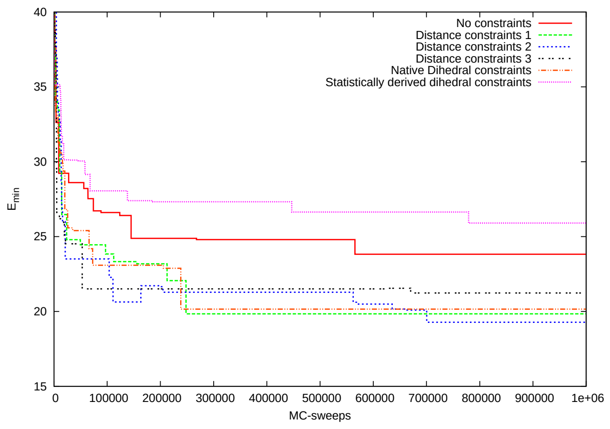
To evaluate the results a couple of measurements are compared. Doing this we always examine the values of the current replica at the lowest temperature. At first we take a look at the minimal physical energies that are plotted in Fig. 4. By comparing the two plots it becomes obvious that the minimal energy is not a really significant evaluation criterion, because there are serious discrepancies between the results of the two test series, especially for the run without constraints. So a significant statement based on this evaluation criterion would require a larger ensemble of test series.

Much more fruitful is an analysis of the average physical energies as well as the average RMSDs [1] plotted in Fig. 5 respectively 6. Looking at the curves of the run using the statistically predicted dihedral constraints one first realizes that the fluctuations are much smaller compared to the other curves. The reason is of course that in this case constraints were used for all residues contrary to the other runs. Compared to the run without constraints there is a slight improvement concerning the average energies and a more significant improvement concerning the average RMSDs. But the average level of the RMSD is quiet high and outliers to low values are rare. This is of course the negative influence of some wrong predictions that were used as constraints.

[1]RMSD = Root Mean Square Deviation. This quantity measures the similarity of the current conformation compared to the native one.



(a) Test series 1.



(b) Test series 2.

Figure 4: Minimal physical energies. The energy terms connected to the distance constraints are subtracted from the total energy. Nevertheless the energies are minimal with respect to the total energy including the distance constraint potentials. This is the reason why some curves are not monotonically decreasing.

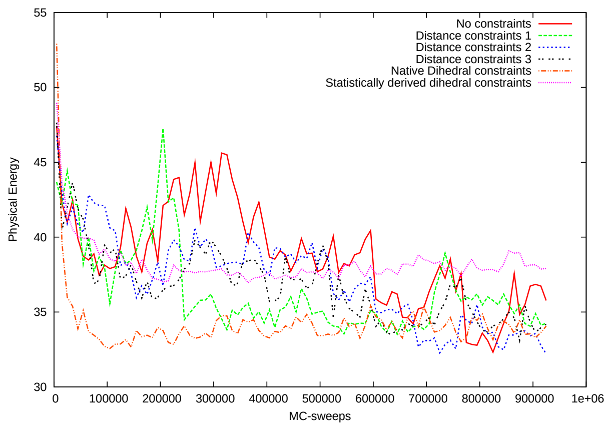
A more convincing improvement was reached in the distance constraints runs. In most cases there are lower average values concerning both the RMSD and the energy. The diagrams do not allow for a clear comparison between the different types of potentials. Looking at test series 2 one would clearly favor potential type three before two and one, but the picture of test series 1 does not look that clear. A larger number of test series is necessary for a significant comparison and evaluation.

The most promising improvement was made in the runs under the native dihedral constraints. In both test series the RMSD reaches mean values below 3 Å in less than 50 K Monte Carlo sweeps and stays at this level for the whole run. Here the aim of a restriction of the search space was reached very well.

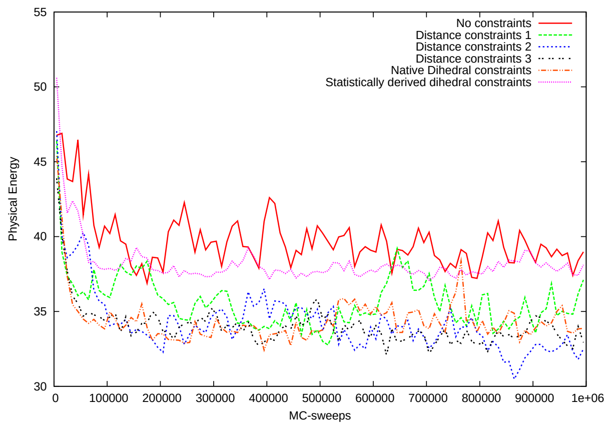
As the last evaluation criterion we examine configurations of the minimal energy for each run that are plotted in Fig. 10 and 11 in the appendix. The best results were reached in the runs with the native dihedral constraints and those with the third distance constraints potential. This is compatible with the evaluation of the mean RMSDs.

Technical Aspects

The simulation runs were performed to one half on the JUMP cluster of the JSC and to the other half on the Nicole cluster of the NIC. Each run lasted between seven and eleven hours using 16 CPUs.



(a) Test series 1.



(b) Test series 2.

Figure 5: Physical energies averaged over 10^4 Monte Carlo sweeps. Again the energy terms connected to the distance constraints are subtracted from the total energy.

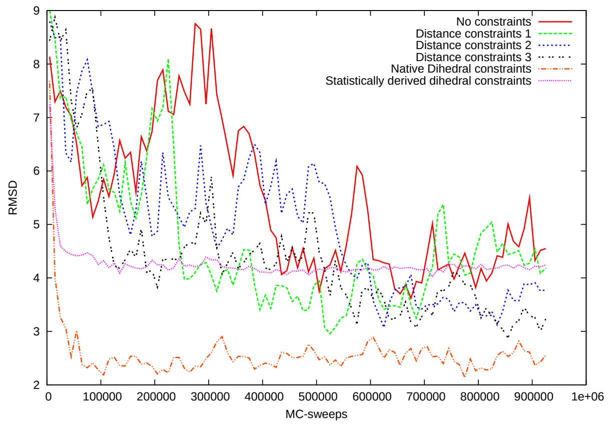
Example 2: 1G9O

Often one is in the lucky situation that a known structure with a very similar sequence to the one under consideration, a so called *template*, exists. In this case one does not start the simulation from a random or stretched configuration but from one that is as close as possible to the template.

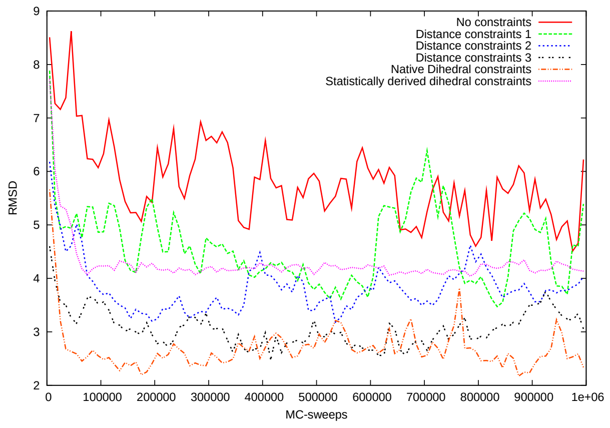
The second example discussed in this article deals with this situation. The target protein is the *FIRST PDZ DOMAIN OF THE HUMAN NA+/H+ EXCHANGER REGULATORY FACTOR*, which can be found in the RCSB protein data bank under the abbreviation 1G9O. It consists of 91 residues. Its native conformation is depicted in Fig. 7a. A good template that was found is *The crystal structure of the first PDZ domain of human NHERF-2 (SLC9A3R2)* – 2OCS in RCSB protein data bank – with a sequence similarity of more than 70 %. Its native structure is shown in Fig. 7b.

The first thing one has to do is to exchange the residues in which the two sequences differ and guess their conformations. The molecule that emerges after this does of course not fit into the geometry with fixed bond angles and lengths used by the program. The simulation program includes a routine called *regularization* that creates a conformation that fits into this geometry and is as close as possible to the initial one while avoiding very high energies. A description of the routine can be found in [2]. The resulting structure is shown in Fig. 8. This was used as a start configuration for several runs.

At first a usual canonical Monte Carlo run was performed at a very low temperature to simply minimize the energy. 5000 sweeps were made at a temperature of 10 K. The final structure is depicted in Fig. 9.



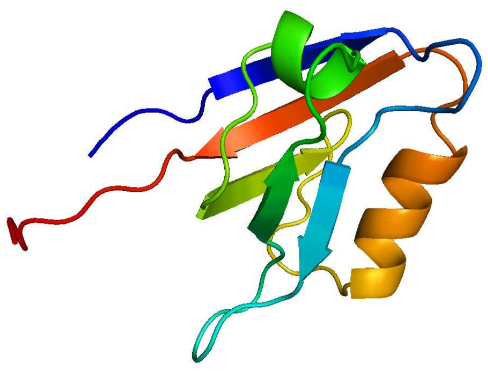
(a) Test series 1.



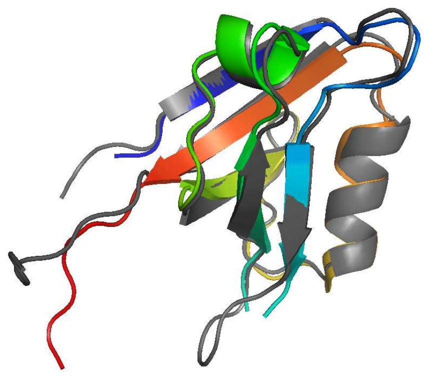
(b) Test series 2.

Figure 6: Root mean square deviation in Å, averaged over 10^4 Monte Carlo sweeps.

Obviously the Monte Carlo run has slightly worsened the result even though the energy was decreased. There are two possible reasons for this. At first it might be due to the approximated force field. At second reason could lie in the way the structure is determined experimentally. In this procedure one has to produce a crystal consisting of many proteins at close quarters that influence of each other. Contrary to that the Monte Carlo run simulates a single protein in a solvent. For that reason the RMSDs are calculated



(a) The target 1G9O.



(b) The template 2OCS. The target 1G9O is depicted in grey for comparison.

Figure 7: The target and its template

excluding the long red tail.

Additionally three parallel tempering runs were started: One without constraints, one with a set of distance constraints and one with a set of dihedral constraints. The constraints were derived partially from the template and partially from statistics. For each of the three runs 50 K sweeps were performed at 32 temperatures ranging from 274 K to 374 K. The minimal energy configurations can be viewed in Fig. 12 in the appendix. Obviously none of the runs improved the configuration that emerged after the regularization, but at least the runs under constraints yielded better results than the one without constraints.

Technical aspects

The parallel tempering runs were done on the *Nicole* cluster of the NIC. Each run took about twelve hours using 32 CPUs.

Conclusion

It could be shown that the use of constraints has great potential to improve protein simulations. But there are many things to do to exploit this potential.

Concerning the distance constraints one has to spend more effort in evaluating how strong the potentials should be and which analytic form of the potential is suited best. Those analytic forms tested in this work all diverge for great deviations with respect to the desired distance. This bears the danger that they reach too high values so that the physical potential looses its influence. Especially when using wrongly predicted constraints the impact could be disastrous. So another analytic form I would propose is

$$E(\{r_i\}) = -\frac{C}{N}\sum_{i=1}^{N} \exp\left[-\left(\frac{r_i - r_{c_i}}{\sigma \cdot r_{c_i}}\right)^2\right], \tag{7}$$

where N is the number of atom pairs under constraints. This has the advantage of a well defined maximum the potential converges to for large deviations. The weight of the constraint potential compared to the physical potential can than easily be controlled by changing the parameter C. The second parameter σ provides the opportunity to alter the range of movement of the constrained atom pairs. I would propose a parameter study for C and σ measuring the deviations of the constraints with respect to their desired values in addition to the other evaluation criteria analyzed in the present article.

The dihedral constraints ansatz introduced here seems to work quite well in the case of correct constraints. So in my opinion the most important step to do next is to develop techniques to identify wrong

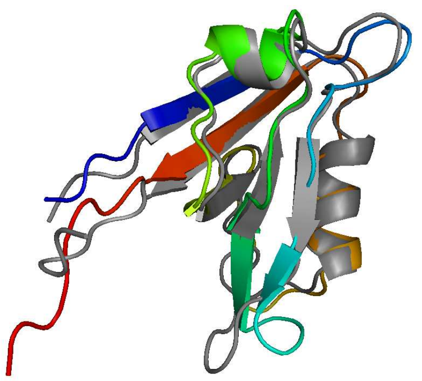


Figure 8: Conformation after the regularization. RMSD = 2.49 Å

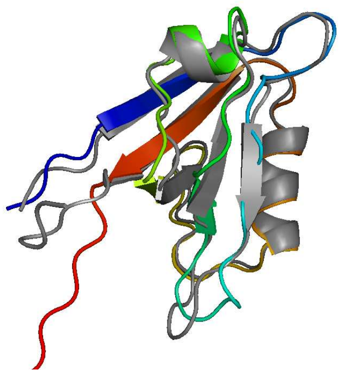


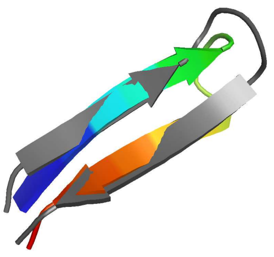
Figure 9: Conformation after canonical Monte Carlo run at 10 K. RMSD = 2.62 Å

constraints. A first ansatz would be to test if there is a significant correlation between the acceptance rate of a constraint move for a specific angle and the deviation between its predicted value and its native value.

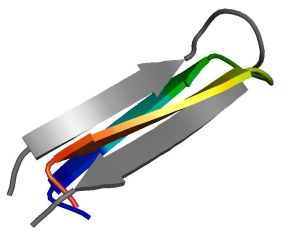
Other questions to be answered are:

- How many constraints are helpful?
- When should they be used? During the whole run or only during parts of it?
- How does a combination of distance and dihedral constraints work?

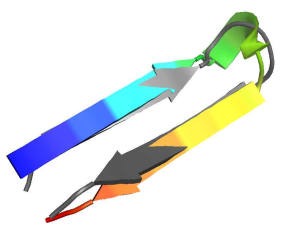
Appendix



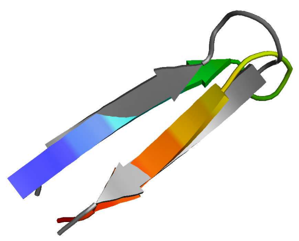
(a) No constraints, series 1. RMSD = 3.42 Å



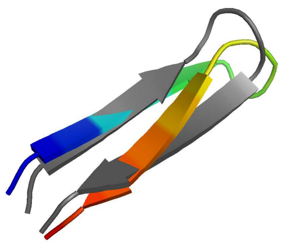
(b) No constraints, series 2. RMSD = 5.71 Å



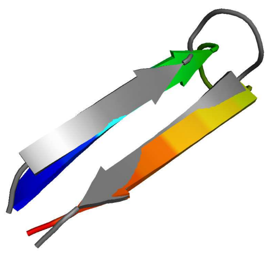
(c) Distance constraints 1, series 1. RMSD = 2.53 Å



(d) Distance constraints 1, series 2. RMSD = 3.89 Å

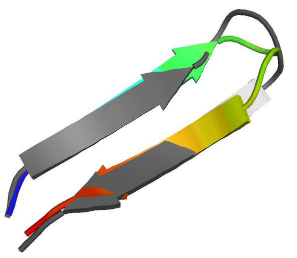


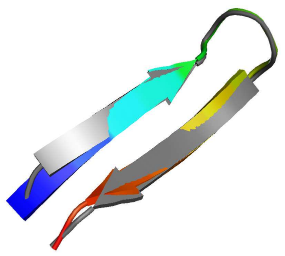
(e) Distance constraints 2, series 1. RMSD = 3.86 Å

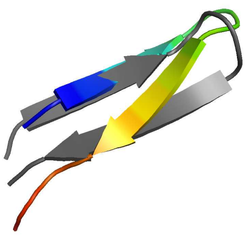


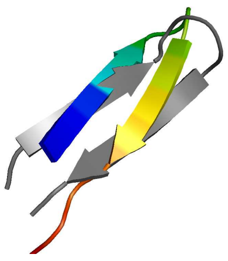
(f) Distance constraints 2, series 2. RMSD = 3.08 Å

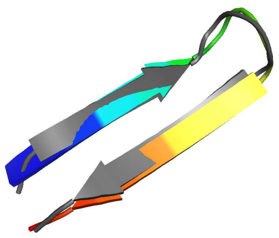
Figure 10: Lowest energy conformations 1. The native conformation is plotted in grey.


(a) Distance constraints 3, series 1. RMSD = 2.93 Å


(b) Distance constraints 3, series 2. RMSD = 1.59 Å


(c) Predicted dihedral constraints, series 1. RMSD = 4.42 Å


(d) Predicted dihedral constraints, series 2. RMSD = 4.52 Å


(e) Native dihedral constraints, series 1. RMSD = 1.98 Å

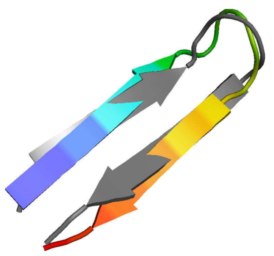
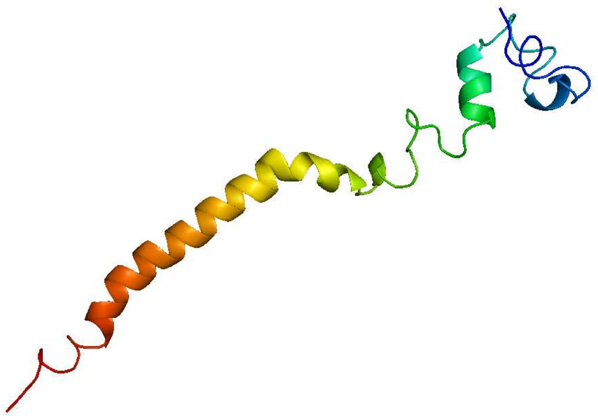
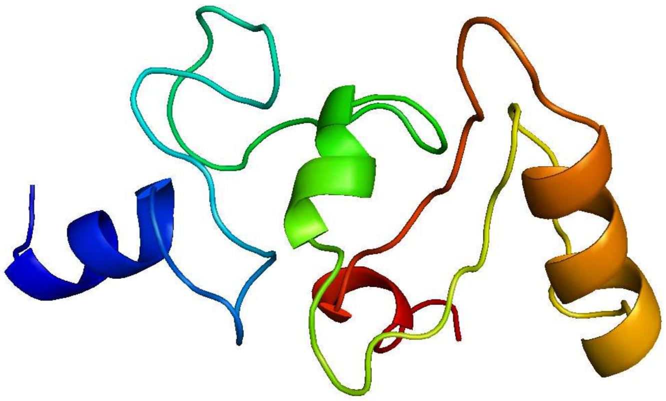

(f) Native dihedral constraints, series 2. RMSD = 2.61 Å

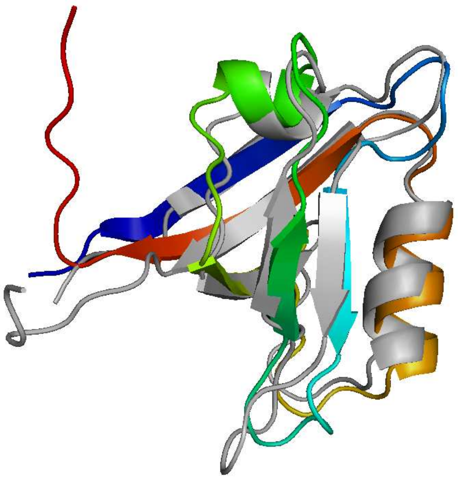
Figure 11: Lowest energy conformations 2. The native conformation is plotted in grey.



(a) No constraints. RMSD = 25.45 Å



(b) Distance constraints. RMSD = 14.75 Å



(c) Dihedral constraints. RMSD = 3.13 Å

Figure 12: Lowest energy conformations resulting from the parallel tempering runs.

Acknowledgment

I would like to thank Lisa Drögekamp for encouraging me to participate in the Guest Students Program and for supporting me writing the application. Furthermore I thank Robert Speck and Matthias Bolten for an impeccable organization of the program and my supervising tutors Dr. Jan Meinke, Dr. Olav Zimmermann and Dr. Sandipan Mohanty for a lot of time and dedication and for entrusting me with an interesting and important task of their field of work. My special thanks go to all other guest students. I really enjoyed the ten weeks together with you!

References

1. Eisenmenger, F., Hansmann, U. H. E., Hayryan, S. and Hu, C.-K., *[SMMP] A modern package for simulation of proteins*, Computer Physics Communications 138, (2001) 192-212
2. Eisenmenger, F., Hansmann, U. H. E., Hayryan, S. and Hu, C.-K., *An enhanced version of SMMP – open-source software package for simulation of proteins*,Computer Physics Communications 174, (2006) 422-429
3. Meinke, J. H., Mohanty, S., Eisenmenger, F. and Hansmann, U. H. E., *SMMP v. 3.0 – Simulating proteins and protein interactions in Python and Fortran*,Computer Physics Communications 178 (2008) 459-470
4. Anfinsen, C. E., *Principles that govern the folding of protein chains*, Science 181 (1973), pp. 223-230
5. Levinthal, C., *Proceedings of a Meeting held at Allerton House, Monticello*, Mossbauer Spectroscopy in Biological Systems (1969)

Parallel Scaletransfer in Multiscale Simulations

Dorian Krause

Institute for Numerical Simulation,
Wegelerstrasse 6, 53115 Bonn

E-mail: dokrause@ins.uni-bonn.de

Abstract: We discuss a recently developed method for coupling molecular dynamics and continuum methods, which is based on a function space approach. The method modifies the bridging domain method by introducing new constraints. We particular focus on the parallel implementation of the method.

Introduction

Molecular dynamic simulations can be used for accurate simulation of critical material behavior, e.g. fractures (see Fig. 1). However the current (massively parallel) computer systems limit the number of atoms in a MD simulation. Unfortunately, the reduction of the system size might lead to spurious “finite size“ artifacts (e.g. due to reflection of sound waves at an artificial surface). Multiscale methods tackle this “finite size problem“ by combining atomistic simulation models like molecular dynamics with macroscopic theories like elasticity. Since e.g. a finite element discretization of a smooth displacement field is possible with a relatively large mesh size, such methods provide ideal “boundary conditions“ for a MD simulation. Still, a large portion of the simulation domain must be simulated by pure molecular dynamics (e.g. around the crack tip). Additionally quantum mechanical computations (e.g. first principle molecular dynamics or density functional theories) might be used near the crack tip (see [1]). Here we present new ideas regarding the parallelization of the weak transfer method ([9, 11]). For this purpose, the underlying theory as well as the parallelization of molecular dynamics and finite elements is reviewed.

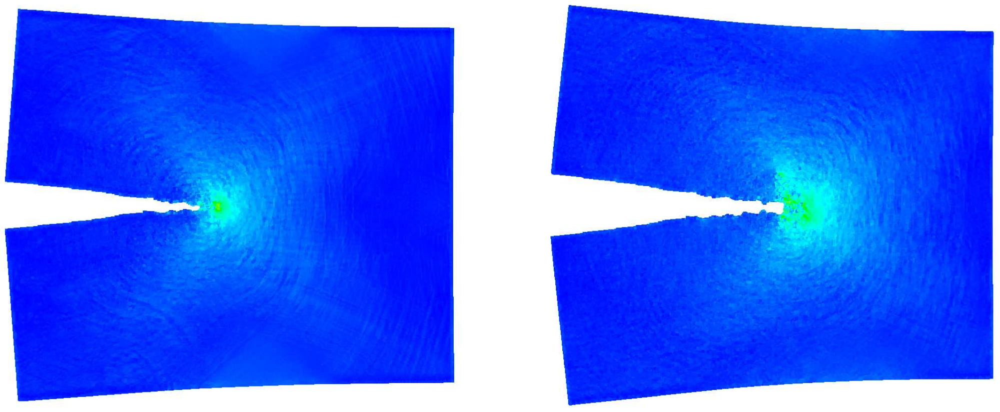


Figure 1: Brittle to ductile transition in a thin 3 dimensional Lennard Jones solid.

Multiscale modeling

Molecular dynamics with short range forces

Let us consider an ideal solid, filling up the volume $\mathcal{B} \subset \mathbb{R}^d$, which undergoes some deformation, due to internal and external forces. On the microscopic level we can model the solid assuming an underlying simple lattice

$$\mathbb{L} = \Big(\sum_{j=1}^{d} \mathbb{Z}\mathbf{a}_j\Big) \cap \mathcal{B}\,.$$

In the reference configuration at each lattice site an atom is located. We will denote the set of atoms by A and use the notation $\mathbf{X}_\alpha$ for the location of the atom $\alpha \in \mathsf{A}$ in the reference configuration. The interaction between atoms is modeled by a potential function $V \in \mathcal{C}^1\big((\mathbb{R}^d)^{\sharp\mathsf{A}};\mathbb{R}\big)$ which we assume to be bounded from below. Such a potential admits a series expansion of the form

$$V\big((\mathbf{x}_\alpha)_{\alpha\in\mathsf{A}}\big) = \sum_{\alpha\in\mathsf{A}} V_1(\mathbf{x}_\alpha) + \sum_{\alpha,\beta\in\mathsf{A}} V_2(\mathbf{x}_\alpha,\mathbf{x}_\beta) + \sum_{\alpha,\beta,\gamma\in\mathsf{A}} V_3(\mathbf{x}_\alpha,\mathbf{x}_\beta,\mathbf{x}_\gamma) + \dots \qquad (1)$$

The 1-body terms V_1 is mostly ignored in the literature since external 1-body forces are treated separately. Following this principle we will assume $V_1 = 0$, i.e. there is no internal 1-body potential contribution. In the simplest case only the 2-body terms V_2 are taken into account. E.g., the Lennard-Jones potential

$$V_{\mathrm{LJ}}(\mathbf{x}_\alpha,\mathbf{x}_\beta) = 4\cdot\big(|\mathbf{x}_\alpha-\mathbf{x}_\beta|^{-12} - |\mathbf{x}_\alpha-\mathbf{x}_\beta|^{-6}\big)$$

has been proposed for modeling an Argon liquid. There exists a great number of (semi-)empirical potential like Finnis-Sinclair, EAM or MGPT potentials, which can be used for modeling “real“ solids (see [12]). All of these potentials require the evaluation of multi-body terms in the expansion (1), increasing the complexity of simulations significantly.

The mass matrix of the atomic system is defined by $M_{\mathsf{A}} = \mathrm{diag}(m_\alpha)_{\alpha\in\mathsf{A}}$, m_α being the mass of atom α. Arranging the positions and momentums of the individual atoms in a column vector, we can write down the Hamiltonian for the particle system

$$\mathcal{H}\big(\mathbf{x},\mathbf{p}\big) = \frac{1}{2}\mathbf{p}\cdot M_{\mathsf{A}}^{-1}\mathbf{p} + V(\mathbf{x}) + V_{\mathrm{ext}}(\mathbf{x})$$

The resulting Hamiltonian equations

$$\dot{\mathbf{z}} = J\nabla_{\mathbf{z}}\mathcal{H}(\mathbf{z}), \quad \mathbf{z} = (\mathbf{x},\mathbf{p})^{\mathrm{T}},\ J = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

yield Newton’s low of motion $M_{\mathsf{A}}\ddot{\mathbf{x}} = \mathbf{F} + \mathbf{F}^{\mathrm{ext}}$ with $\mathbf{F} = -\nabla_{\mathbf{x}}V(\mathbf{x})$.
Molecular dynamics is naturally formulated in Lagrangian coordinates. When coupling molecular dynamics and elasticity theories (which are naturally formulated in Eulerian coordinates) it is necessary to choose a common description. For this purpose we introduce the MD displacement $\mathbf{u}_\alpha = \mathbf{x}_\alpha - \mathbf{X}_\alpha$. The space of all possible displacements is $X := \big\{\mathsf{A} \to \mathbb{R}^d\big\}$.

Example simulation

We describe the simulation of a mode I crack in a brittle two dimensional solid using the Lennard Jones potential. This is an interesting example since the anisotropy of the underlying hexagonal lattice yields an anisotropic stress-strain relation for large strains [2]. The path of the crack always turns in the strong direction, that is the direction yielding the highest surface energy. Fig. 2 shows the time evolution of a

crack initially directed in the weak and the strong direction, resp. As can be seen, the crack does not propagate in the weak direction but rather branches. The newly created crack tips propagate in the strong direction, the angle between the old and the new direction being approximately 30 degree.
The experiments have been conducted using the parallel version of `Tremolo` [12].

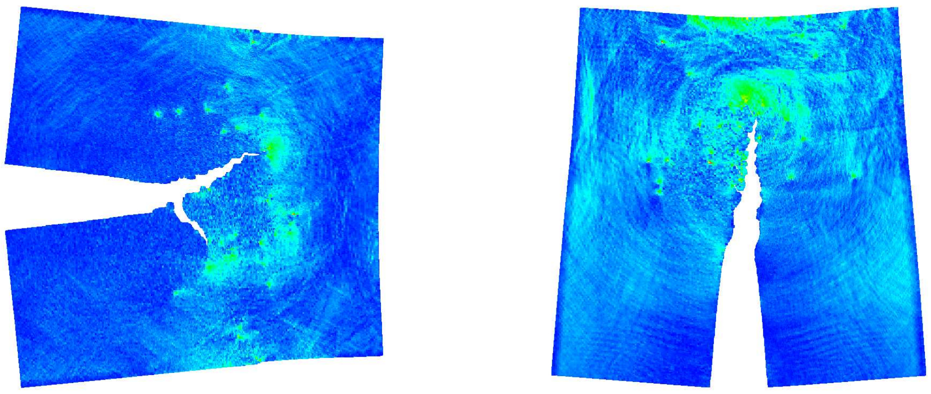


Figure 2: A mode I crack in a 2 dimensional Lennard Jones solid.

The simulation domain contained 200000 atoms. Nevertheless, as can be seen in Fig. 3, the introduction of artificial boundaries results in the reflection of sound waves at the boundary. Moreover, the emission, reflection and returning of these waves takes place on a time scale which is relevant for the crack initiation and propagation. This is a typical example of a finite size problem (see also [17] for another example) and must be taken into account when evaluating the results. Multiscale methods are designed to tackle this problem.

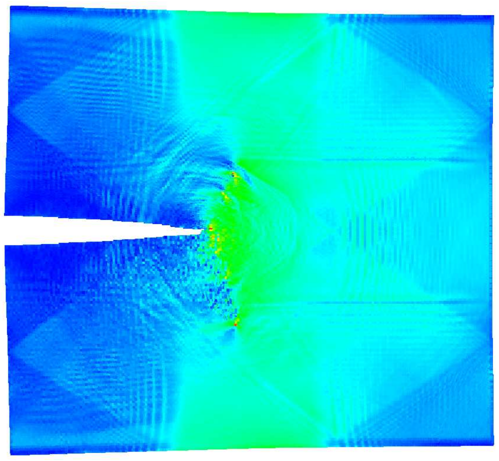


Figure 3: Sound waves propagating through a 2 dimensional Lennard Jones solid.

Elasticity theory

Assuming the mean distance of neighboring atoms in $\mathbb{L}$ to be a few angstrom, we see that the simulation of a cubic-shaped body with diameter 1 mm $\times$ 1 mm $\times$ 1 mm requires the simulation of approximately $N = 10^{21}$ particles. The minimal storage requirement for a simulation with N atoms is ≈ 1 ZB (1 billion gigabyte). Such simulations are currently out of scope. From a macroscopic point of view however, 1 mm^3 still remains small.

Fortunately, many important mechanical properties of solids can be modeled utilizing continuum theories, i.e. neglecting the underlying lattice structure of the solid. From this point of view, $\mathcal{B}$ is a continuum with Lebesgue measure $m(\mathcal{B}) > 0$. The underlying lattice implicitly determines the density function $\varrho : \mathcal{B} \to (0, \infty)$.
The deformation of the body is described by a volume preserving injection $\boldsymbol{\varphi}(t) \in \mathcal{C}^1(\mathcal{B}; \mathbb{R}^d)$ mapping $\mathcal{B}$ at each time t to its deformed configuration $\boldsymbol{\varphi}(\mathcal{B})$. The difference $\mathbf{u} = \boldsymbol{\varphi} - \mathrm{id}$ is called the displacement field.
The deformation of the body introduces internal stresses in $\mathcal{B}$. The law of motion can be obtained from momentum equilibrity in the deformed configuration. Here, we will describe a different approach which works for the large class of homogeneous hyperelastic materials:
Similar to the microscopic model, we introduce a potential function $V : \mathcal{C}^1(\mathcal{B}; \mathbb{R}^d) \to \mathbb{R}$ measuring the stored energy in the deformed configuration of $\mathcal{B}$. In applications the potential is mostly constructed from a density function $w : \mathbb{M}_d \to \mathbb{R}$ by

$$V(\mathbf{u}) = \int_{\mathcal{B}} w(\mathbf{F})\, d\mathbf{x}$$

Here, $\mathbf{F} = \mathbf{I}_d + \nabla \mathbf{u} \in \mathbb{M}_d$ denotes the deformation gradient. The Hamiltonian of the continuum theory is

$$\mathcal{H}(\mathbf{u}, \mathbf{p}) = T(\mathbf{p}) + V(\mathbf{u}) = \int_{\mathcal{B}} \frac{1}{2} \mathbf{p} \cdot \varrho \mathbf{p}\, d\mathbf{x} + \int_{\mathcal{B}} w(\mathbf{F})\, d\mathbf{x} + \int_{\mathcal{B}} \mathbf{b} \cdot \mathbf{u}\, d\mathbf{x}$$

The vector field $\mathbf{b}$ models volume forces. The Hamiltonian is a functional on the (infinite dimensional) phase space and the Hamiltonian equations needs to be interpreted in a weak sense, i.e. by differentiating the functional in the direction of a test function $\mathbf{v}$ in an appropriate subspace of $H^1(\mathcal{B})^d$. The resulting law of motion reads

$$\int_{\mathcal{B}} \varrho \ddot{\mathbf{u}} \cdot \mathbf{v}\, d\mathbf{x} = \int_{\mathcal{B}} \varrho \mathbf{b} \cdot \mathbf{v}\, d\mathbf{x} + \int_{\mathcal{B}} \mathbf{P} : \nabla \mathbf{v}\, d\mathbf{x} \tag{2}$$

or $\varrho \ddot{\mathbf{u}} = \varrho \mathbf{b} - \operatorname{div} \mathbf{P}$ with the nominal stress $\mathbf{P} = \dfrac{\partial w}{\partial \mathbf{A}}$. The tensor $\mathbf{P}$ (which is not be confused with the pressure) can be seen to be the (unsymmetric) Piola transformation of "true" Cauchy-stress $\boldsymbol{\sigma}$ from the deformed configuration to $\mathcal{B}$.

The question remains, how the density w should be chosen. A careful analysis (see [7]) shows that for small deformations all materials behave similar to a Saint-Vernant material with material dependent Lamè constants. For large deformations however complex (possibly non-convex) stored energy functions w yield significantly different properties.
Since we want to couple molecular dynamics with elasticity and since the behavior of the atomistic system is dictated by the choice of the interaction potential, it is natural to ask for way to construct a stored energy function from an atomistic potential. Such homogenization procedures are a current research interest.
With some restricts, the Cauchy-Born rule allows for the construction of a stored energy function w_{CB} from an atomistic potential V. It is assumed that the underlying lattice of the solid locally follows the deformation gradient, i.e. locally the distance $\mathbf{r}_{\alpha,\beta} = \mathbf{x}_\alpha - \mathbf{x}_\beta = \mathbf{F}\mathbf{R}_{\alpha,\beta}$, $\mathbf{R}_{\alpha,\beta} = \mathbf{X}_\alpha - \mathbf{X}_\beta$. In this case the value of w_{CB} at $\mathbf{A} \in \mathbb{M}_d$ is obtained from the energy of the underlying lattice after stretching it according to the transformation $\mathbf{A}$, see [8]. For a 2-body potential the stored energy function is

$$w_{\mathrm{CB}}(\mathbf{A}) = \frac{1}{2 \cdot \vartheta_0} \sum_{\mathbf{s} \in \mathbb{L}} V_2(0, \mathbf{A}\mathbf{s}), \quad \vartheta_0 = \text{volume of principal unit cell of } \mathbb{L}$$

In order to approximate the continuous displacement field $\mathbf{u}$ we employ a $\mathbb{P}1$ or $\mathbb{Q}1$ finite element discretization. Let $\mathcal{T}$ be a shape regular mesh such that $\bigcup_{T \in \mathcal{T}} \overline{T} = \mathcal{B}$ (assume a polygonal shaped $\mathcal{B}$ for simplicity). We denote the node set of $\mathcal{T}$ by $\mathcal{N}$. The finite element space $S \subset H^1(\mathcal{B})^d$ is

$$S = \left\{ \mathbf{u} \in \mathcal{C}(\mathcal{B}, \mathbb{R}^d) \text{ s.t. } \mathbf{u}|_T \in (\mathbb{P}_1)^d \text{ or } \mathbf{u}|_T \in (\mathbb{Q}_1)^d \text{ f.a. } T \in \mathcal{T} \right\}$$

The so called Lagrangian basis functions $\{\theta_p\}_{p\in\mathcal{N}}$ of S are uniquely characterized by the Kronecker-delta property $\theta_p(q) = \delta_{p,q}$ for $q \in \mathcal{N}$. Obviously this implies compactness of $\operatorname{supp}\theta_p$ in $\mathcal{B}$.

Concurrent multiscale coupling

Every concurrent multiscale coupling scheme faces two fundamental problems:

- Molecular dynamics is formulated as an nonlinear ordinary differential equation (ODE) on the finite dimensional phase space while continuum theories are formulated on an infinite dimensional function space, yielding a partial differential equation (PDE) as law of motion.
- For efficient usage of the finite elements as an enveloping boundary for a MD domain it is necessary to work with a FE mesh size which is typically an order of magnitude above the lattice constant. Therefor the MD displacement field contains (high frequency) components which are not representable on the (coarse) FE scale. These components need to be removed from the molecular domain without being reflected at the MD boundary.

The methods present in the literature can roughly been divided in two classes. Methods in the first class (such as the concurrent coupling of length scales method [1]) use an adaptively refined mesh in the coupling zone which allows for pointwise identification of atoms and mesh nodes. Methods in the second class (such as the bridging scale method [18] and the bridging domain method [19]) use ideas originating from domain decomposition methods for the scale transfer. All of these methods work in a finite dimensional space.

The weak coupling concept [9] is inspired by research in the field of nonconforming domain decomposition methods, especially Mortar methods. These methods use L^2 averaging at a coupling interface between to FE meshes instead of pointwise identification to restore optimal convergence of the method. The same ideas are applicable in our multiscale context once we are able to interpret the molecular displacement $(\mathbf{u}_\alpha = \mathbf{x}_\alpha - \mathbf{X}_\alpha)_{\alpha\in\mathsf{A}}$ as an L^2 function. To accomplish the transfer from the discrete space $X = \{\mathsf{A} \to \mathbb{R}^d\}$ to the infinite dimensional space $L^2(\mathcal{B}, \varrho d\mathbf{x})^d$ the authors use techniques from scattered data approximation to be explained shortly:

The Partition of Unity method (PUM) is a meshless approximation method allowing for efficient approximation of the (discrete) MD displacement. The PUM constructs a basis $\varphi_\alpha \in L^2$ of the space $X_{\mathsf{PU}} = \bigoplus_{\alpha\in\mathsf{A}} \mathbb{R}^d \varphi_\alpha$ such that $\mathbf{u}_\beta \approx \sum_{\alpha\in\mathsf{A}} \mathbf{u}_\alpha \varphi_\alpha(\mathbf{X}_\alpha)$. This means that the coefficients of the approximation $\mathbf{u}$ of the MD displacement with respect to the generating system $\{\varphi_\alpha\}_{\alpha\in\mathsf{A}}$ is simply the column vector of the displacement field. The inclusion of the discrete field in the function space therefor comes essentially for free.
The construction of the basis functions proceeds in two steps:

1. To each atom an open patch $\omega_\alpha \subset \mathcal{B}$ is attached so that the union of all patches covers the whole body. The patches should be overlapping to guarantee a good approximation property of the space X_{PU}.
2. Following Shepard's approach [16] we choose a weight function b_α such that $\operatorname{supp} b_\alpha = \omega_\alpha$. Using $\varphi_\alpha = b_\alpha$ surely is a bad choice since (in general) not even constant functions are exactly interpolated. The latter property is equivalent to the partition of unity property $\sum_{\alpha\in\mathsf{A}} \varphi_\alpha \equiv 1$. To construct a partition of unity $\{\varphi_\alpha\}$ we normalize the weighting functions b_α, i.e.

$$\varphi_\alpha(\mathbf{x}) = \frac{b_\alpha(\mathbf{x})}{\sum_{\beta\in\mathsf{A}} b_\beta(\mathbf{x})}$$

The evaluation of the denominator only requires the evaluation of neighboring weight functions due to the local support of b_α.

Exploiting the crystalline structure of the atom starting positions, tree based construction methods can easily been applied for the construction of X_{PU} (see [10]).

In [9] the weak coupling concept is applied to the bridging scale method. For this purpose the least squares projection P, used in the BSM to obtain a representation of the MD displacement on the FE mesh, is replaced by an L^2 projection $P : X_{\text{PU}} \to S$. As shown in [9] this method can be interpreted as an higher order bridging scale method.

The weak bridging domain method [11] combines the bridging domain method together and the weak coupling idea. The method assumes an overlapping domain decomposition of $\mathcal{B}$ in a MD domain $\Omega_D \subset \mathcal{B}$ and a (coarse) FE domain $\Omega_C \subset \mathcal{B}$. The intersection $\Omega_B = \Omega_D \cap \Omega_C$ is called the bridging domain. The geometric setup is also shown in Fig. 4.

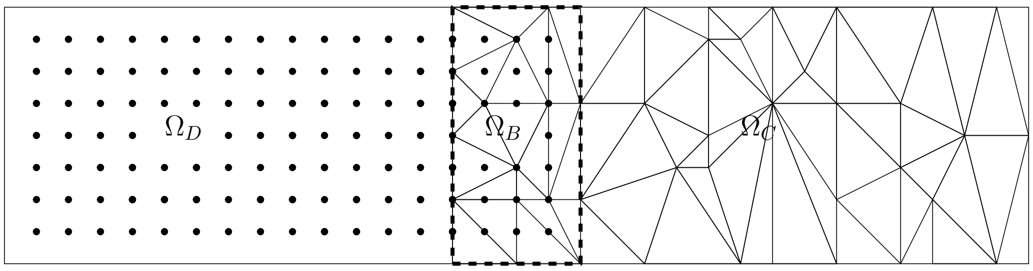


Figure 4: Geometry of a bridging domain simulation

The law of motion of the coupled system is derived from a combined Hamiltonian

$$\mathcal{H}(\mathbf{u}_D, \mathbf{p}_D, \mathbf{u}_C, \mathbf{p}_C; \boldsymbol{\lambda}) = \varpi \cdot \mathcal{H}_{\text{MD}}(\mathbf{u}_D, \mathbf{p}_D) + (1 - \varpi) \cdot \mathcal{H}_{\text{FE}}(\mathbf{u}_C, \mathbf{p}_C) + g(\mathbf{u}_D, \mathbf{u}_C, \boldsymbol{\lambda}) \qquad (3)$$

Here, $(\mathbf{u}_D, \mathbf{p}_D)$ and $(\mathbf{u}_C, \mathbf{p}_C)$ are points of the MD and FE phase space, respectively. The real-valued function g is used to constrain both scales to a common behavior in the bridging domain (see below). The function $\varpi : \mathcal{B} \to [0, 1]$ is a weighting function which prevents from double counting the energy in the coupling zone. To obtain a consistent energy we furthermore require $\varpi|_{\Omega_D \setminus \Omega_B} = 1$ and $\varpi|_{\Omega_C \setminus \Omega_B} = 0$.

Regarding equation (3) two important remarks are necessary:

1. The momentums $\mathbf{p}_D$ and $\mathbf{p}_C$ are **not** the canonically conjugated momentums but rather the kinematic momentums of the MD and FE system respectively. The underlying structure really is a weighted Lagrangian (see [11]).

2. The notation used in (3) is actually an abuse of notation since the multiplication of e.g. the MD Hamiltonian by a real-valued function is not senseful. For a two-body potential V and an homogeneous, hyperelastic material the precise Hamiltonian is

$$\mathcal{H} = \sum_{\alpha \in \mathsf{A}} \varpi(\mathbf{X}_\alpha) \frac{(\mathbf{p}_D)_\alpha^2}{2 \cdot m_\alpha} + \sum_{\alpha \in \mathsf{A}} \varpi(\mathbf{X}_\alpha) \frac{1}{2} \sum_{\beta \in \mathsf{A} \setminus \{\alpha\}} V_2 \left(\mathbf{X}_\alpha + (\mathbf{u}_D)_\alpha, \mathbf{X}_\beta + (\mathbf{u}_D)_\beta \right)$$
$$+ \int_{\Omega_C} \frac{1}{2} (1 - \varpi(\mathbf{x})) \, \mathbf{p}_C \cdot \varrho(\mathbf{x}) \mathbf{p}_C \, d\mathbf{x} + \int_{\Omega_C} (1 - \varpi(\mathbf{x})) \cdot w(\mathbf{I}_d + \nabla \mathbf{u}_C) \, d\mathbf{x}$$

The classical bridging domain method [19] utilizes pointwise constraints in the bridging domain. Let $N : S \to X$ be the evaluation (or interpolation) map, i.e. $(N\mathbf{u}_C)_\alpha = \mathbf{u}_C(\mathbf{X}_\alpha)$. Belytschko and Xiao use the constraints

$$g = \sum_{\alpha \in \mathsf{A}} \boldsymbol{\lambda}_\alpha \cdot ((\mathbf{u}_D)_\alpha - \mathbf{u}_C(\mathbf{X}_\alpha)) = \boldsymbol{\lambda}^{\mathrm{T}}\big(\mathbf{u}_D - N\mathbf{u}_C\big)$$

The Lagrange multiplier $\boldsymbol{\lambda}$ is chosen such that the induced Lagrange forces $\nabla_{\mathbf{u}_D} g$ and $\nabla_{\mathbf{u}_C} g$ yield a corrected trajectory on the manifold $\{\mathbf{u}_D = N\mathbf{u}_C\}$.
Fine fluctuations of the MD displacement field which cannot be represented on the FE mesh (due to the reduced number of dofs) are implicitly removed by the constraints. However, since these fluctuations cannot propagate into the bridging domain, high frequency waves are reflected at the interface $\Omega_D \cap \partial\Omega_B$. The authors propose the use of a lumped multiplier matrix to reduce this effect (see also [4]).

In the weak bridging domain method the pointwise constraints are replaced by equality in an averaging sense. More precisely, using the approximated $\mathbf{u}_D \in X_{\mathrm{PU}} \subset L^2$ the constraints are

$$g = (\boldsymbol{\lambda}, \mathbf{u}_C - \mathbf{u}_D)_{L^2} = (\boldsymbol{\lambda}, \mathbf{u}_C - P\mathbf{u}_D)_{L^2}\,, \quad \boldsymbol{\lambda} \in S$$

where $P : X_{\mathrm{PU}} \to S$ denotes the L^2 projection from the MD space X_{PU} to S. The constrain manifold therefor is $\{\mathbf{u}_C = P\mathbf{u}_D\}$.
Inserting the basis $\{\theta_p\}$ of S and $\{\varphi_\alpha\}$ of X_{PU} we obtain the basis representation of $g = \boldsymbol{\lambda} \cdot (M_C\mathbf{u}_C - R\mathbf{u}_D)$ with

$$(M_C)_{p,q} = \int_{\mathcal{B}} \theta_p\theta_q \, d\mathbf{x} \quad \text{and} \quad R_{q,\alpha} = \int_{\mathcal{B}} \theta_q\varphi_\alpha \, d\mathbf{x}$$

The weak BDM decouples the information transfer between the scales and the damping of the fine fluctuations, which are exactly $\mathbf{u}'_D = Q\mathbf{u}_D = \mathbf{u}_D - NP\mathbf{u}_D$. Furthermore the number of Lagrange multipliers now is $\sharp\,(\mathcal{N} \cap \Omega_B)$ instead of $\sharp\,\big(\bigcup_{\alpha\in\mathsf{A}}\{\mathbf{X}_\alpha\} \cap \Omega_B\big)$.

Since the fine fluctuations in the MD displacement are not affected by the constraints they are allowed to propagate into the bridging domain without disturbance. Therefor it is necessary to apply an additional damping technique to remove these fluctuations.
An example of such a technique, which is particular suited for the weak bridging domain method, is the discrete perfectly matched boundary layer (PML) method ([13]). The PML method introduces two additional force terms to the MD system.

$$\mathbf{F}_{\mathrm{PML}} = -D^2Q\mathbf{u} - 2DQ\dot{\mathbf{u}} \tag{4}$$

The first term alters the stiffness of the MD system in Ω_B while the second one is a frictional damping term. The scaling $D : X_{\mathrm{PU}} \to X_{\mathrm{PU}}$ controls the strength of the additional force terms. It is chosen such that $D_\alpha \ll 1$ near the interface $\Omega_C \cap \partial\Omega_B$, so that interface reflections are minimized. In [11] D_α is chosen as a quadratic function of the distance $\mathrm{dist}(\mathbf{X}_\alpha, \Omega_C \cap \partial\Omega_B)$.

Parallel implementation

Parallelizations based on the message passing paradigm have been successfully applied to both, molecular dynamics as well as finite elements. Multiscale methods allow for simulation of large systems with a significantly reduce number of degrees of freedom compared to classical molecular dynamics, tackling the so called finite size problem. However no methods are known to tackle the “finite time problem“, meaning the limited simulation time due to the very small time steps. These are necessary to resolve the small timescales of the atomic system. This requires a parallel multiscale method with an excellent strong scaling behavior.

Parallel molecular dynamic algorithms

In 1995 Plimpton classified parallel molecular algorithms for short range forces depending on the data/-work decomposition [14]. This classification remains valid up to now, even though clever hybrid methods have been published (see e.g. [5, 17]). Methods utilizing the atom-decomposition of the workload assign disjoint atom sets A_P to the individual processor elements (PEs). The partition $\mathsf{A} = \bigsqcup_P \mathsf{A}_P$ is only change if the load is unbalanced. Force-decomposition methods use a decomposition of the sparse force matrix $(\mathbf{F}_{\alpha,\beta})_{\alpha,\beta\in\mathsf{A}}$ similar to parallel linear algebra algorithms.
In the following we will concentrate on the third class, the so called spatial decomposition methods which assign a subdomain $\Omega_P \subseteq \Omega$ to each PE. In most cases the decomposition is based on a Cartesian splitting, as in Fig. 5. This allows for efficient neighbor communication with only 4 or 6 send/receive operations in 2 or 3 dimensions (“Plimpton scheme“, see [14]).

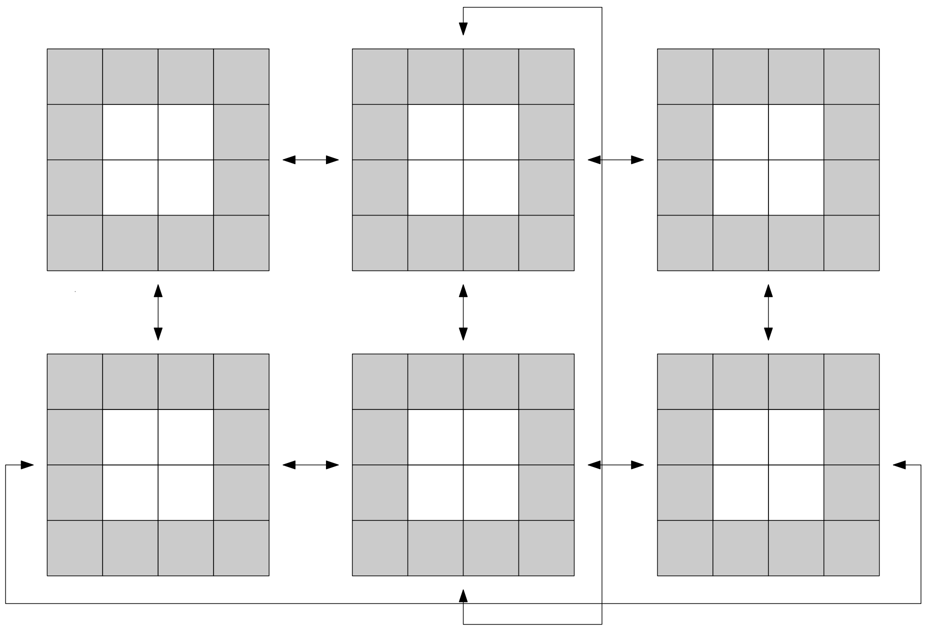


Figure 5: two dimensional decomposition on a 3×2 PE grid with periodic boundaries

For rapidly decaying potentials only interactions of particles α, β with $|\mathbf{x}_\alpha - \mathbf{x}_\beta| \leq r_{\text{cut}}$ are considered relevant. This reduces the number of particles stored on remote PEs, that are needed for the local force evaluation.
In each timestep the coordinates of particles near $\partial\Omega_P$ are communicated and stored in the halo cell of the remote PEs. The forces on the local particles can then be evaluated in parallel. If communication is cheaper than force evaluation (e.g. for MGPT) Newton’s third law can be used. This requires an additional force reduction step. Since particles change their position, it is possible that atom α leaves domain Ω_P. In this case the particle changes its owner process. The migration of particles is handled in a second communication step. New methods (e.g. the midpoint method) reduce the communication volume for the force evaluation by assigning particle pairs to the PEs (see e.g. [6]).
Let us note, that the ability to balance load between the PEs is limited due to the Cartesian geometry of the decomposition. This is no problem for the simulation of e.g. biomolecules in a solvent where the density is highly uniform. However, in solids, cracks and voids can yield a nonuniform density. For this reason, the `ddcMD` code [17] is based on an atom decomposition technique. Future work, will concentrate on such methods.

The spatial decomposition allows for easy construction of the PU basis $\{\varphi_\alpha\}_{\alpha\in\mathsf{A}}$ since the same communication schemes can be applied. First, on each processor a local quad/octree is constructed. Then,

boundary bordures are exchanged using the Plimpton communication scheme. Inserting the remote atoms in a local copy of the remote tree, allows for a parallel construction of the generating system with quasi-optimal complexity (see also [15]).
Let us assume rectangular or cuboid shaped patches ω_α, i.e. $\omega_\alpha = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{X}_\alpha\|_\infty < h_\alpha\}$. In this case all particles in the bordure $\{\operatorname{dist}(\mathbf{x}, \partial\Omega_P) < 2 \cdot h_\alpha\}$ must be communicated. For $h_\alpha \approx r_{\text{cut}}$ we see that the communication volume is doubled compared to MD. However the scalability of the algorithm remains (The construction of $\{\varphi_\alpha\}_{\alpha\in\mathsf{A}}$ needs to be done only once in the initialization phase).
There exists a second, extremely simple, parallel partition of unity method. Since φ_α only depends on the neighboring atoms $\{\beta \mid m(\omega_\alpha \cap \omega_\beta) > 0\}$ one might try to change patches of atoms near the processor boundary $\partial\Omega_P$ by $\omega_\alpha \to \omega_\alpha \cap \Omega_P$. This allows for the construction of φ_α without any communication. Therefor we term the method the “embarrassingly parallel partition of unity“ (EPPU) method. Unfortunately, the approximation quality near the boundary is reduced since by construction $\varphi_\alpha \equiv 1$ near $\partial\Omega_P$ (compare Fig. 6). The same effect occurs near non-periodic boundaries.

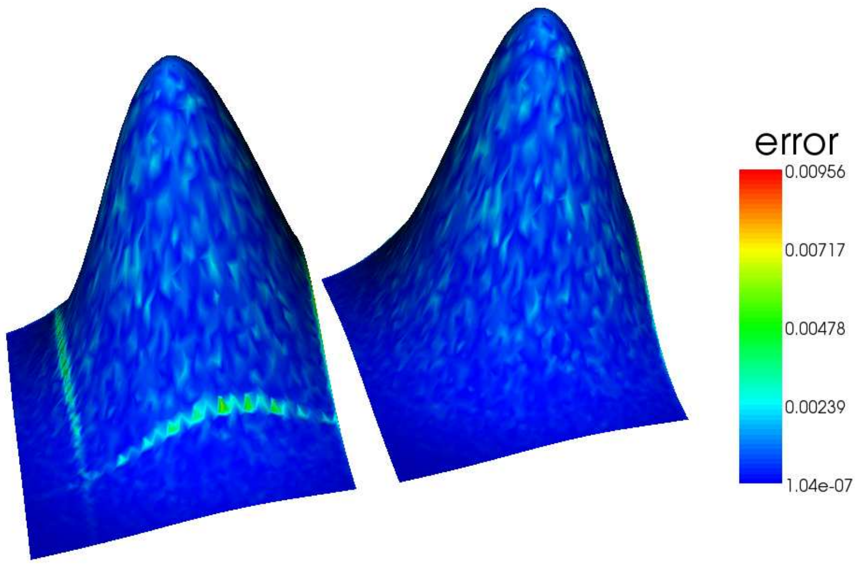


Figure 6: Approximation of Franke’s function with 50000 “atoms“ using EPPU and PPU on 4 PEs. The resulting graph has been triangulated.

Parallel finite element algorithms

Parallel finite element codes usually employ a domain decomposition of $\Omega = \bigsqcup_P \Omega_P$ of the simulation domain Ω. Each subdomain Ω_P is assigned to an PE. Additionally, each PE stores halo cells. The width of the additional bordure depends on various factors, e.g. the support of the basis functions $\{\theta_p\}$.

Let load_P denote the load of a PE P. An easy performance model assumes $\operatorname{load}_P \propto \sharp\mathcal{T}_P$ where $\mathcal{T}_P$ is the submesh of $\mathcal{T}$ stored on the PE. If the mesh is held fix during the simulation, the load (as well as the communication volume) remains fix. In the simple model, the communication volume can be assumed to be proportional to the number of edges between remotely stored elements.

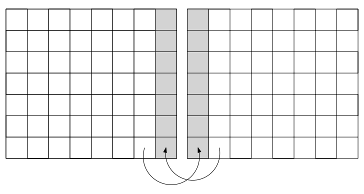


Figure 7: Partition of a structured FE mesh.

The decomposition of a (unstructured) mesh $\mathcal{T}$ into pieces $\mathcal{T}_P$ such that

$$\Big|\,\mathrm{load}_P - \frac{1}{\sharp\mathrm{PEs}} \sum\nolimits_{P'} \mathrm{load}_{P'}\,\Big| \le \mathrm{TOL}$$

is not an easy task at all. Finding a balanced partition which additionally minimizes the communication volume is a NP complete problem. Fortunately efficient heuristics for such graph-partition problems exist (e.g. recursive coordinate bisection, recursive spectral bisection).

Parallel multiscale method

We will describe a parallel algorithm based on a disjoint partition of the available PEs in MD and FE processors, i.e. every PE either stores atom coordinates or a submesh. This allows for parallel force evaluation and for a flexible choice of the ratio $\sharp$(MD PEs)$/\sharp$(FE PEs) (in most cases we will use a lot more MD processors elements than finite element PEs). However, scale transfer always requires message passing. Overlapping schemes will be explored once experience with the disjoint approach has been gained.

The weak bridging domain method allows for the use of structure preserving integrator for the constraint coupled system. We will describe the use of the SHAKE/RATTLE integrator. The following algorithm describes one timestep with stepsize τ.

1. Advance the momentums $\mathbf{p}_D^n$ and $\mathbf{p}_C^n$ to trial values $\mathbf{p}_?^{*,n+1/2} = \mathbf{p}_?^n + \frac{\tau}{2}\mathbf{F}_?^n$, $? \in \{D, C\}$.
2. Compute new trial positions $\mathbf{u}_?^{*,n+1} = \mathbf{u}_?^n + \tau\, M_?^{-1}\mathbf{p}_?^{n+1/2}$. Here $M_?$ denotes the mass matrices of the resp. scales. For the finite element system we use a lumped mass matrix.
3. Compute the residual $\mathbf{r} = M_C\mathbf{u}_C - R\mathbf{u}_D \in S$. Lagrange multipliers are obtained inverting the linear system $\big(\frac{1}{2}\tau^2\Lambda\big)\boldsymbol{\lambda} = \mathbf{r}$ with $\Lambda = M_C^{-1} - RM_D^{-1}R^{\mathrm{T}}$.
4. Correct the trial values $\mathbf{p}_?^{*,n+1/2}$ and $\mathbf{u}_?^{*,n}$, i.e.
$$\mathbf{p}_?^{n+1/2} = \mathbf{p}_?^{*,n+1/2} - \frac{1}{2}\tau\,(\nabla_{\mathbf{u}_?} g)\,\boldsymbol{\lambda} \qquad \mathbf{u}_?^{n+1} = \mathbf{u}_?^{*,n+1/2} - \frac{1}{2}\tau^2\,(\nabla_{\mathbf{u}_?} g)\,\boldsymbol{\lambda}$$
5. Compute forces $\mathbf{F}_?^{n+1}$.
6. The molecular dynamics PEs compute the product $\mathbf{u}_D' = Q\mathbf{u}_D$ and add the PML force terms (4) to $\mathbf{F}_D^{n+1}$.
7. Compute $\mathbf{p}_?^{*,n+1} = \mathbf{p}_?^{n+1/2} + \frac{1}{2}\tau\mathbf{F}_?^{n+1}$
8. Solve the multiplier system $\big(\frac{1}{2}\tau\Lambda\big)\boldsymbol{\mu} = \mathbf{p}_C^{*,n+1} - RM_D^{-1}\mathbf{p}_D^{*,n+1}$

9. Correct the momentums $\mathbf{p}_?^{n+1} = \mathbf{p}_?^{*,n+1} - \frac{1}{2}\tau\left(\nabla_{\mathbf{u}_?} g\right)\boldsymbol{\mu}$

The parallelization of steps 1, 2, 5 and 7 have been described above. Step 4 and 9 are easily executed in parallel once the computed Lagrange forces have been communicated. Regarding the parallel execution of step 3, 6 and 8 we face two problems:

- Due to the Lagrangian point of view in molecular dynamics algorithms, the intercommunicator graph, i.e. the graph describing the communication between FE and MD processor elements, is not time independent. This means that e.g. a finite element PE a priori doesn't know the set $\{P' \mid \text{exists } \alpha \in \mathsf{A} \text{ s.t. } \mathbf{x}_\alpha \in (\Omega_D)_{P'} \text{ and } R_{p,\alpha} \neq 0\}$ for $p \in \mathcal{N}_P$.
- The dimension of Q and Λ is $\sharp\left(\bigcup_{\alpha\in\mathsf{A}}\{\mathbf{X}_\alpha\} \cap \Omega_B\right)$ and $\sharp\left(\mathcal{N} \cap \Omega_B\right)$ respectively. In most cases these matrices are too small to be distributed over all available PEs. Unfortunately this leads to scalability problem since the serial nature of the multiscale SHAKE/RATTLE algorithm doesn't allow those PEs not involved in the computation of $Q \cdot (-)$ or Λ^{-1} to compensate the waiting time. These PEs remain idle.

We will not address the second issue here since it most likely requires a complete redesign of the MD and FE algorithms.
A possible solution to issue 1 is the adaption of the intercommunicator graph in each particle communication step (see [3]). This allows each finite element PE to know its MD neighbors and vice versa. However additional communication is necessary in case of particle migration.
Assuming a static distribution of the matrices Q, R and Λ we propose the use of 1-sided communication to build the intercommunicator graph implicitly. Using an all-to-all communication in the initialization phase every particle knows the set of all PEs that require either the displacement or velocity vector of the particle to accomplish their task. Moreover information about the buffer layout(s) of those PEs are attached to each particle. Using PUT and GET operations each particle is able to either send a message to a remote PE which is waiting for input or receive e.g. the values $(Q\mathbf{u}_D)_\alpha$, $(Q\dot{\mathbf{u}}_D)_\alpha$ so that the PML force term can be computed locally.
The matrix Q and Λ are distributed over all processors P which either fulfill $m\left(\bigcup_{T\in\mathcal{T}_P}\overline{T} \cap (\Omega_B)_P\right) > 0$ (in case of a finite element PE) or $\sharp\left(\bigcup_{\alpha\in\mathsf{A}}\{\mathbf{X}_\alpha\} \cap (\Omega_B)_P\right) > 0$ (in case of MD PEs). Assuming a good balanced number of atoms/nodes a row wise block decomposition is appropriate.

Due to the disjoint partition of the PEs, in either FE or MD processor elements, data needed for the assembling of the matrices R, Q and Λ are not in the memory of a single process. It is necessary to collect data on either one of the both sides (or swap copies of the local memory). Differently from [3], we choose the molecular dynamics PEs for the assembling of the three matrices. Using this approach we can gain profit from the tree-structure used for the construction of $\{\varphi_\alpha\}_{\alpha\in\mathsf{A}} \subset X_{\text{PU}}$. It is then possible to assemble R, Q and Λ with quasi optimal complexity $O(\sharp(\text{BD elements}) \cdot \log\sharp(\text{BD atoms}))$. Furthermore, the number of molecular dynamics PEs is larger therefor yielding a better distribution of the computational expensive cut-detection and quadrature.

Concluding remarks and acknowledgment

The implementation of a multiscale method, as the one presented, requires a large software stack. During my stay at Juelich I was able to implement a large portion of the necessary software. Most of this work found its way only implicitly (through experience gain) into this report.
Additionally, using the excellent infrastructure in Juelich, I was able to conduct many molecular dynamics fracture simulations which greatly improved my knowledge.

I would like to thank my advisor Dr. B. Steffen for the many interesting discussions concerning fracture mechanics with molecular dynamics. I thank my supervisor Prof. R. Krause and my colleagues at the University Bonn, esp. K. Fackeldey, for the permanent support during the work for my diploma thesis on multiscale simulation.
Further, I thank the organizer of the guest student program, Matthias Bolten and Robert Speck, for the courageously work. I also thank my fellow guest students for the pleasant working atmosphere.

References

1. F. F. Abraham, J. Q. Broughton, N. Bernstein, E. Kaxiras. Spanning the continuum to quantum length scales in a dynamic simulation of brittle fracture, Europhys. Lett., 44(6), 1998.
2. F. F. Abraham. Dynamics of Brittle Fracture with Variable Elasticity, Phys. Rev. Lett., vol. 77(5), 1996.
3. G. Anciaux, O. Coulaud, J. Roman. High Performance Multiscale Simulation or Crack Propagation, ICPPW, Proceedings of the 2006 International Conference Workshops on Parallel Processing, 2006, pp 473.
4. G. Anciaux, O. Coulaud, J. Roman, G. Zerah. Ghost force reduction and spectral analysis of the 1D bridging method, Rapport de recherche 6582, INRIA, 2008.
5. K. J. Bowers, R. O. Dror, D. E. Shaw. Zonal methods for the parallel execution of range-limited N-body simulations, Journal of Computational Physics 221, 2007.
6. K. J. Bowers, R. O. Dror, D. E. Shaw. The Midpoint Method for Parallelization of Particle Simulations, Journal of Chemical Physics, vol. 128, 2006.
7. P. G. Ciarlet. Mathematical elasticity, vol. 1, Elsevier Science Publishers, 1988.
8. W. E, P. B. Ming. Cauchy-Born rule and the stability of crystalline solids: Static problems, Arch. Rat. Mech. Anal., vol 183, 2007.
9. K. Fackeldey, R. Krause. Multiscale Coupling in Function Space - Weak Coupling Between Molecular Dynamics and Continuum Mechanics, 2008, submitted to Int. J. Num. Math. ENGRG.
10. K. Fackeldey, D. Krause, R. Krause. Quadrature and Implementation of the Weak Coupling Method, INS Preprint No. 0807.
11. K. Fackeldey, D. Krause, R. Krause, C. Lenzen. INS Preprint, to appear.
12. M. Griebel, S. Knapek, G. Zumbusch. Numerical Simulation in Molecular Dynamics, Springer, Berlin, Heidelberg, 2007.
13. S. Li, X. Liu, A. Agrawal, A. C. To, Perfectly Matched Multiscale Simulations for Discrete Systems: Extension to Multiple Dimensions, Physical Review B, vol. 74, 2007.
14. S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, Journal of Computational Physics, vol. 117, 1995.
15. M. A. Schweitzer, A Parallel Multilevel Partition of Unity Method for Partial Differential Equations, Springer LNCSE vol. 29, 2003.
16. D. Shepard. A two-dimensional function for irregular spaced data. ACM National Conference, 1868.
17. F.-H. Streitz, J. N. Glosli, M. V. Patel, B. Chan, R. K. Yates, B. R. de Supinksi. 100+ TFlop Solidification Simulations on BlueGene/L, Proceedings of the 2008 ACM/IEEE conference on Supercomputing, 2008
18. G. J. Wagner and W. K. Liu. Coupling of atomistic and continuum simulations using a bridging scale decomposition, J. Comp. Phy. 2003.
19. S. P. Xiao, T. Belytschko. A brigding domain method for coupling continua with molecular dynamics, Comp. Meth. Appl. Mech. Engrg., 2004.

Fast Computation of the Ewald Sum

Stefan Müller

University of Technology Chemnitz

E-mail: stefan.mueller@s2005.tu-chemnitz.de

Abstract: In this report a fast method for computing the Ewald sum for many-particle systems based on the recently developed Fastsum algorithm [1] is proposed. The algorithm is implemented in C and details of the implementation of this algorithm are described. The influence of some parameters on the accuracy of this method is evaluated.

Introduction

We want to evaluate a central potential

$$V(\boldsymbol{r}_j) = \sum_{k=1,\, k\neq j}^{N} \frac{q_j}{\|\boldsymbol{r}_j - \boldsymbol{r}_k\|},$$

$j = 1, \ldots, N$, where q_j denotes the charge and $\boldsymbol{r}_j \in \mathbb{R}^3$ the position vector of a particle j in a three-dimensional periodic system of N particles. Let $\|\cdot\|$ denote the Euclidean norm. This is motivated by the need to calculate trajectories $\boldsymbol{r}_j(t)$ of particles interacting along this potential by the equation of motion

$$m_j \frac{\mathrm{d}^2 \boldsymbol{r}_j}{\mathrm{d}t^2} = -q_j \nabla_j V,$$

where m_j denotes the mass of the particle j.

Because of the particles being part of a periodic system, or a system with periodic boundary conditions, i.e. we have N particles inside a center box of dimension three and box length equal to one and this box periodically expanded in every direction in the three-dimensional space, we have to evaluate

$$V_s(\boldsymbol{r}_j) = \sum_{\boldsymbol{n}\in\mathbb{Z}^3}{}' \sum_{k=1}^{N} \frac{q_j}{\|\boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{n}\|} \tag{1}$$

instead of $V(\boldsymbol{r}_j)$, where the prime denotes that for $\boldsymbol{n} = \boldsymbol{0}$ the term with $k = j$ is dropped. This can be rewritten as

$$V_E(\boldsymbol{r}_j) = \underbrace{\sum_{\boldsymbol{n}\in\mathbb{Z}^3}{}' \sum_{k=1}^{N} q_k \frac{\operatorname{erfc}(\alpha\|\boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{n}\|)}{\|\boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{n}\|}}_{=:S_1(\boldsymbol{r}_j)} + \underbrace{\sum_{\boldsymbol{l}\in\mathbb{Z}^3\setminus\{\boldsymbol{0}\}} \sum_{k=1}^{N} q_k \frac{\mathrm{e}^{-\frac{\pi^2\|\boldsymbol{l}\|^2}{\alpha^2}}}{\pi\|\boldsymbol{l}\|^2} \mathrm{e}^{2\pi\mathrm{i}\boldsymbol{l}\cdot(\boldsymbol{r}_k - \boldsymbol{r}_j)}}_{=:S_2(\boldsymbol{r}_j)} - \frac{2\alpha}{\sqrt{\pi}} q_j, \tag{2}$$

where $S_1(\boldsymbol{r}_j)$ is called the real space sum and $S_2(\boldsymbol{r}_j)$ the reciprocal space sum. Details on how this can be done can be found in [2]. We call (2) the Ewald sum. On the following pages we want to propose an algorithm to compute an approximation of this formula.

The Fastsum Algorithm

We want to compute the sums

$$f(\boldsymbol{r}_j) := \sum_{k=1}^{N} q_k K(\boldsymbol{r}_j - \boldsymbol{r}_k), \tag{3}$$

$j = 1, \dots, N$, where $K(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^3$ is a certain kernel function. In our case we are interested in

$$K(\boldsymbol{x}) = \frac{\operatorname{erfc}(c\|\boldsymbol{x}\|)}{\|\boldsymbol{x}\|}.$$

For the Fastsum algorithm we have to require that

$$\begin{aligned} \|\boldsymbol{r}_j\|_\infty &< \frac{1}{4} - \frac{\varepsilon_B}{2}, \\ \|\boldsymbol{r}_k\|_\infty &< \frac{1}{4} - \frac{\varepsilon_B}{2}, \\ \Rightarrow \|\boldsymbol{r}_j - \boldsymbol{r}_k\|_\infty &< \frac{1}{2} - \varepsilon_B. \end{aligned}$$

We can compute $f(\boldsymbol{r}_j)$, $j = 1, \dots, N$ in $O(N \log N)$ using the Fastsum algorithm proposed in [1].

The first step in this algorithm is to regularize the kernel $K(\boldsymbol{x})$ near the singularity at $\mathbf{0}$, i.e. inside $U_I := \{\boldsymbol{x} \in \mathbb{R}^3 : \|\boldsymbol{x}\| < \varepsilon_I\}$, and near the boundary of $\Pi^3 := [-\frac{1}{2}, \frac{1}{2})^3$, i.e. in $U_B := \{\boldsymbol{x} \in \mathbb{R}^3 : \frac{1}{2} - \varepsilon_B < \|\boldsymbol{x}\|\}$. Here ε_I denotes the inner boundary and ε_B the outer boundary. Obviously we have to state that $\varepsilon_I + \varepsilon_B < 0.5$. We call this regularized kernel $K_R(\boldsymbol{x})$. The degree of smoothness of $K_R(\boldsymbol{x})$ is called p. This regularization is done to get a smooth function, which can be continued one-periodically.

The regularized kernel $K_R(\boldsymbol{x})$ can now be approximated by its Fourier series $K_{RF}(\boldsymbol{x})$:

$$K_R(\boldsymbol{x}) \approx K_{RF}(\boldsymbol{x}) = \sum_{\boldsymbol{l} \in \Lambda} b_{\boldsymbol{l}}\, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{x}}. \tag{4}$$

where $\Lambda := [-\frac{n}{2}, \frac{n}{2} - 1]^3 \cap \mathbb{Z}^3$. The regularization of $K(\boldsymbol{x})$ introduces an error $K_{NE}(\boldsymbol{x}) := K(\boldsymbol{x}) - K_R(\boldsymbol{x})$ due to the inner regularization. Now we can approximate $K(\boldsymbol{x})$ by

$$K(\boldsymbol{x}) \approx K_{NE}(\boldsymbol{x}) + K_{RF}(\boldsymbol{x}).$$

We call $K_{NE}(\boldsymbol{x})$ the nearfield.

The nearfield sum $f_{NE}(\boldsymbol{r}_j)$, $j = 1, \dots, N$, is calculated directly by evaluating the sum

$$f_{NE}(\boldsymbol{r}_j) = \sum_{k \in I_{\varepsilon_I}^{NE}(j)} q_k K_{NE}(\boldsymbol{r}_j - \boldsymbol{r}_k), \tag{5}$$

where $I_{\varepsilon_I}^{NE}(j) := \{k \in 1, \dots, N : \|\boldsymbol{r}_j - \boldsymbol{r}_k\| < \varepsilon_I\}$.

The Fourier sum $f_{RF}(\boldsymbol{r}_j)$, $j = 1, \dots, N$, can be calculated by

$$\begin{aligned} f_{RF}(\boldsymbol{r}_j) &= \sum_{k=1}^{N} q_k K_{RF}(\boldsymbol{r}_j - \boldsymbol{r}_k) \overset{\text{by (4)}}{=} \sum_{k=1}^{N} q_k \sum_{\boldsymbol{l} \in \Lambda} b_{\boldsymbol{l}}\, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \cdot (\boldsymbol{r}_j - \boldsymbol{r}_k)} \\ &= \sum_{\boldsymbol{l} \in \Lambda} b_{\boldsymbol{l}} \underbrace{\left(\sum_{k=1}^{N} q_k \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_k} \right)}_{=: a_{\boldsymbol{l}}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_j}. \end{aligned} \tag{6}$$

So, applying the NFFT^T on the inner sum yields

$$f_{RF}(\boldsymbol{r}_j) \approx \sum_{\boldsymbol{l}\in\Lambda} b_{\boldsymbol{l}} a_{\boldsymbol{l}} \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{l}\cdot\boldsymbol{r}_j},$$

which, after multiplication of the coefficients $b_{\boldsymbol{l}}$ and $a_{\boldsymbol{l}}$, can be calculated by the NFFT.

Applying the Fastsum Algorithm on the Ewald Sum

The above described algorithm was designed for open systems of particles. Therefore we have to adjust it to work on systems with periodical boundary conditions as well. The first step to achieve this is to choose the parameter α such that the sum over $\boldsymbol{n}$ in $S_1(\boldsymbol{r}_j)$ in (2) can be neglected due to the fast decrease of the kernel function, i.e. only the term with $\boldsymbol{n} = \boldsymbol{0}$ is considered. By the choice of α we want to assure that the function value of the kernel function $K(\boldsymbol{x})$ has already fallen below e_α for all $\boldsymbol{x}$ with $\|\boldsymbol{x}\| > \frac{1}{2} - \varepsilon_B$. We call $R_C := \frac{1}{2} - \varepsilon_B$ the cut-off radius of the real space sum and

$$e_\alpha := \frac{\mathrm{erfc}(\alpha R_C)}{R_C} \tag{7}$$

the truncation error introduced by neglecting the terms with $\boldsymbol{n} \neq \boldsymbol{0}$. In figure 1 we see that the truncation error decreases very fast for increasing values of α, e.g., if we want to get $e_\alpha = 10^{-9}$, we would choose $\alpha \approx 8.8$.

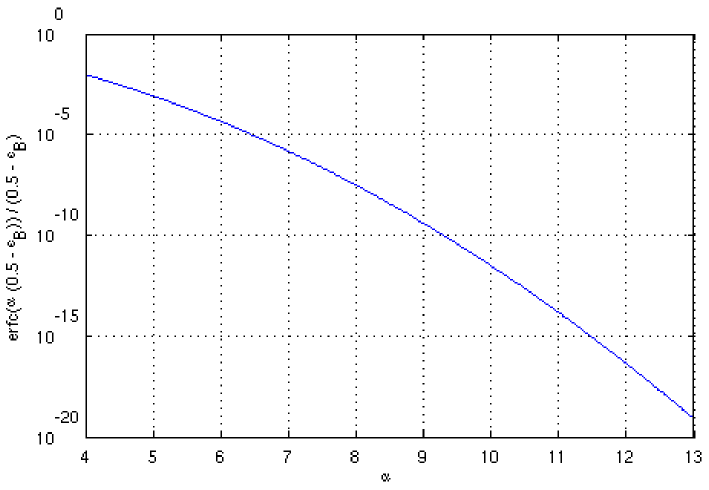


Figure 1: The function value of $e_\alpha = K(\boldsymbol{x}) = \frac{\mathrm{erfc}(\alpha\|\boldsymbol{x}\|)}{\|\boldsymbol{x}\|}$ for $\boldsymbol{x}$ with $\|\boldsymbol{x}\| = \frac{1}{2} - \varepsilon_B$ depending on α.

By restricting the calculation of the formula given in (2) to $\boldsymbol{n} = \boldsymbol{0}$, we assure that for each particle j, $j = 1, \ldots, N$, we only have to consider $N - 1$ particles from the setup of our periodic system as described at the beginning as interacting partners. Unfortunatly, we cannot just choose one fixed box with length one and all N particles inside and ignore all periodic image particles. If we would do so, the potential calculated for particles near the boundary of that fixed box would be very erroneous, because close neighbors of these particles with high influence on the result would be neglected.

Instead, for each particle j at position $\boldsymbol{r}_j$, $j = 1, \ldots, N$, we have to consider an individual box with length one and this particle at its center. So, for calculating interactions between particle j and the other $N-1$ particles some particles in the center box have to been shifted by a vector in $\mathbb{Z}^3\backslash\{\boldsymbol{0}\}$. These shifted particles are called image particles. Because the Fourier sum $f_{RF}(\boldsymbol{r}_j)$ in (6) is already one-periodic, we do not have to adjust anything in the computation of this term.

We only have to change the computation of the nearfield sum $f_{NE}(\boldsymbol{r}_j)$ in (5). Here we are going to compute

$$f_{NE}(\boldsymbol{r}_j) = \sum_{k \in \tilde{I}^{NE}_{\varepsilon_I}(j)} q_k K_{NE}(\boldsymbol{r}_{jk}), \tag{8}$$

where $\tilde{I}^{NE}_{\varepsilon_I}(j) := \{k \in 1, \dots, N : \|\boldsymbol{r}_{jk}\| < \varepsilon_I\}$ and $\boldsymbol{r}_{jk} = \boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{n}$ with $\boldsymbol{n}$ such, that

$$\|\boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{n}\| = \min_{\boldsymbol{y} \in \mathbb{Z}^3} \|\boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{y}\|.$$

This means that $\|\boldsymbol{r}_{jk}\|$ is the minimum distance from particle j to particle k or any of its images. In our algorithm we achieve this behaviour by calculating $\boldsymbol{r}_j - \boldsymbol{r}_k$ and componentwise shifting of this vector by ± 1 in such a way that we get a vector $\tilde{\boldsymbol{r}}_{jk}$ for which

$$\|\tilde{\boldsymbol{r}}_{jk}\|_\infty \leq \tfrac{1}{2}$$

holds. Then it also holds true that $\tilde{\boldsymbol{r}}_{jk} = \boldsymbol{r}_{jk}$.

This enables us to calculate

$$\begin{aligned} S_1(\boldsymbol{r}_j) &= \sum_{\boldsymbol{n} \in \mathbb{Z}^3} {}' \sum_{k=1}^{N} q_k \frac{\operatorname{erfc}(\alpha \|\boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{n}\|)}{\|\boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{n}\|} \\ &\approx \sum_{k \in \tilde{I}^{NE}_{\varepsilon_I}(j)} q_k K_{NE}(\boldsymbol{r}_{jk}) + \sum_{\boldsymbol{l} \in \Lambda} b_{\boldsymbol{l}} \left(\sum_{k=1}^{N} \alpha_k \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_k} \right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_j}, \end{aligned}$$

$j = 1, \dots, N$, where $\Lambda = [-\frac{n}{2}, \frac{n}{2} - 1]^3 \cap \mathbb{Z}^3$, in (2) using the Fastsum algorithm.

We can write $S_2(\boldsymbol{r}_j)$, $j = 1, \dots, N$, as

$$S_2(\boldsymbol{r}_j) = \sum_{\boldsymbol{l} \in \mathbb{Z}^3 \setminus \{\boldsymbol{0}\}} \sum_{k=1}^{N} q_k \underbrace{\frac{\mathrm{e}^{-\frac{\pi^2 \|\boldsymbol{l}\|^2}{\alpha^2}}}{\pi \|\boldsymbol{l}\|^2}}_{=: d_{\boldsymbol{l}}} \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \cdot (\boldsymbol{r}_k - \boldsymbol{r}_j)} = \sum_{\boldsymbol{l} \in \mathbb{Z}^3 \setminus \{\boldsymbol{0}\}} d_{\boldsymbol{l}} \left(\sum_{k=1}^{N} q_k \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_k} \right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_j} \tag{9}$$

and reconsidering (2) leaves us with

$$\begin{aligned} V_E(\boldsymbol{r}_j) \approx & \sum_{k \in \tilde{I}^{NE}_{\varepsilon_I}(j)} q_k K_{NE}(\boldsymbol{r}_{jk}) + \sum_{\boldsymbol{l} \in \Lambda} b_{\boldsymbol{l}} \left(\sum_{k=1}^{N} \alpha_k \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_k} \right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_j} \\ & + \sum_{\boldsymbol{l} \in \mathbb{Z}^3 \setminus \{\boldsymbol{0}\}} d_{\boldsymbol{l}} \left(\sum_{k=1}^{N} q_k \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_k} \right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_j} + \frac{2\alpha}{\sqrt{\pi}} q_j. \end{aligned} \tag{10}$$

We introduce a cut-off parameter $l_{\max} \leq \frac{n}{2} - 1$ such that we only have to pay attention to $d_{\boldsymbol{l}}$ with $\|\boldsymbol{l}\|_\infty \leq l_{\max}$ and can assume $d_{\boldsymbol{l}} = 0$ for $\|\boldsymbol{l}\|_\infty > l_{\max}$, see figure 2. E.g. if we want to neglect only $d_{\boldsymbol{l}}$ with $d_{\boldsymbol{l}} < 10^{-9}$ for $\alpha = 8.8$ we would choose $l_{\max} = 11$.

We define

$$\tilde{b}_{\boldsymbol{l}} := \begin{cases} b_{\boldsymbol{l}} + d_{\boldsymbol{l}} & \text{if } \|\boldsymbol{l}\|_\infty \leq l_{\max},\ \boldsymbol{l} \neq \boldsymbol{0}, \\ b_{\boldsymbol{l}} & \text{otherwise.} \end{cases} \tag{11}$$

Now we rewrite (10) as

$$V_E(\boldsymbol{r}_j) = \sum_{k \in \tilde{I}^{NE}_{\varepsilon_I}(j)} q_k K_{NE}(\boldsymbol{r}_{jk}) + \sum_{\boldsymbol{l} \in \Lambda} \tilde{b}_{\boldsymbol{l}} \left(\sum_{k=1}^{N} \alpha_k \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_k} \right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_j} + \frac{2\alpha}{\sqrt{\pi}} q_j. \tag{12}$$

Thus we obtain the following fast algorithm, which we call FAST.

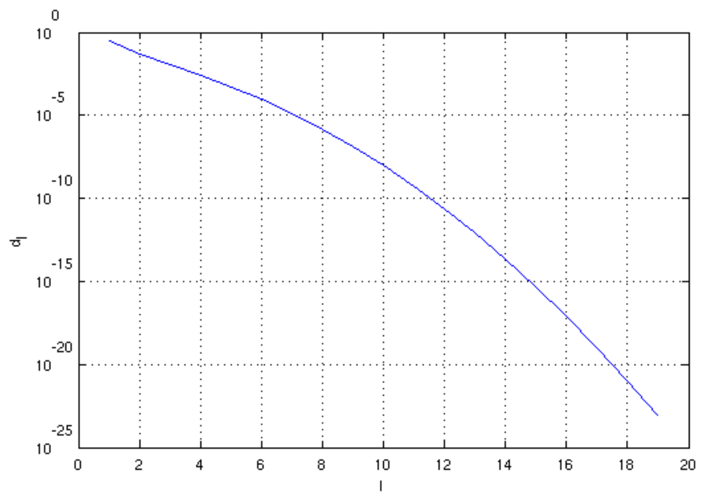


Figure 2: Upper bound for the values of the neglected coefficients $d_{\boldsymbol{l}}$ depending on $l_{\max}$ for $\alpha = 8.8$.

Precomputation:

i. Computation of the Fourier coefficients $b_{\boldsymbol{l}}$ for $\boldsymbol{l} \in \Lambda$ of the regularized kernel $K_R(\boldsymbol{x})$ as described in [1].
ii. Computation of the coefficients $d_{\boldsymbol{l}}$ for $\boldsymbol{l} \in \bar{\Lambda} = [-l_{\max}, l_{\max}]^3 \cap \mathbb{Z}^3$ by (9).
iii. Computation of the coefficients $\tilde{b}_{\boldsymbol{l}}$ for $\boldsymbol{l} \in \Lambda$ by (11).
iv. Computation of $K_{NE}(\boldsymbol{r}_{jk})$ for $j = 1, \ldots, N$ and $k \in \tilde{I}_{\varepsilon_I}^{NE}(j)$.

1. For $\boldsymbol{l} \in \Lambda$ compute by NFFT^T
$$a_{\boldsymbol{l}} := \sum_{k=1}^{N} q_k \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{x}_k}.$$
2. For $\boldsymbol{l} \in \Lambda$ compute
$$c_{\boldsymbol{l}} := a_{\boldsymbol{l}} \tilde{b}_{\boldsymbol{l}}.$$
3. For $j = 1, \ldots, N$ compute by NFFT
$$f_{RF}(\boldsymbol{r}_j) := \sum_{\boldsymbol{l} \in \Lambda} c_{\boldsymbol{l}} \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{r}_j}.$$
4. For $j = 1, \ldots, N$ compute the nearfield corrections
$$f_{NE}(\boldsymbol{r}_j) := \sum_{k \in \tilde{I}_{\varepsilon_I}^{NE}(\boldsymbol{r}_j)} q_k K_{NE}(\boldsymbol{r}_{jk}).$$
5. For $j = 1, \ldots, N$ compute
$$\tilde{f}(\boldsymbol{r}_j) := f_{NE}(\boldsymbol{r}_j) + f_{RF}(\boldsymbol{r}_j).$$

Direct Algorithms for Computing the Ewald Sum

For testing purposes two direct algorithms computing (2) were implemented. Both compute $S_2(\boldsymbol{r}_j)$, $j = 1, \ldots, N$, by

$$S_2(\boldsymbol{r}_j) = \sum_{\substack{\boldsymbol{l} \in \mathbb{Z}^3 \setminus \{\boldsymbol{0}\} \\ \|\boldsymbol{l}\|_\infty \le l_{\max}}} d_{\boldsymbol{l}} \left(\sum_{k=1}^{N} q_k \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_k} \right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \cdot \boldsymbol{r}_j}$$

using a user-specified value for $l_{\max}$.

The first algorithm, called PER, computes

$$S_1(\boldsymbol{r}_j) = \sum_{\boldsymbol{n} \in \mathbb{Z}^3} {}' \sum_{k=1}^{N} q_k \frac{\operatorname{erfc}(\alpha \|\boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{n}\|)}{\|\boldsymbol{r}_j - \boldsymbol{r}_k + \boldsymbol{n}\|}$$

by evaluating the double sum for a user-specified range of $\boldsymbol{n}$. The purpose of this algorithm was mainly to validate the correctness of the second algorithm, called DIR. For DIR it is presumed that the cut-off radius R_C is 0.5. I.e., each particle j, $j = 1, \ldots, N$, of our central box interacts with $N - 1$ particles and for each interaction $\boldsymbol{r}_{jk}$ instead of $\boldsymbol{r}_j - \boldsymbol{r}_k$ is used, as described in (8), i.e. DIR computes

$$S_1(\boldsymbol{r}_j) \approx \sum_{k=1}^{N} q_k \frac{\operatorname{erfc}(\alpha \|\boldsymbol{r}_{jk}\|)}{\|\boldsymbol{r}_{jk}\|}.$$

For sufficently great values of α the latter algorithm can be considered exact and is therefore used for error calculations in relation to our fast algorithm.

Parameter Studies

The following parameters can be changed by the user to alter the behaviour of the fast algorithm outlined above.

N	Number of particles in the grid per dimension, so there will be N^3 particles total in the central box;
n	Expansion degree, number of Fourier coefficients for the regularized kernel function $K_R(\boldsymbol{x})$ per dimension;
$l_{\max}$	Range of the coefficents $d_{\boldsymbol{l}}$;
m	Parameter for the truncation interval of the window functions in the NFFT algorithm, see [1];
p	Degree of smoothness of the regularized kernel function;
ε_I	Boundary of the inner regularization of the kernel function;
ε_B	Boundary of the outer regularization of the kernel function;
α	Parameter of the kernel function $K(\boldsymbol{x})$.

Regular Grid

We set up our particles in a three-dimensional regular grid with alternating charges, i.e.

$$\boldsymbol{r}_j = \frac{(j_1, j_2, j_3)^T + \frac{1}{2}}{N} - \frac{1}{2}, \qquad q_j = (-1)^{j_1 + j_2 + j_3},$$

where $j = 1, \ldots, N^3$ and $j_1, j_2, j_3 \in \{0, \ldots, N - 1\}$ are chosen such that $j = 1 + j_1 + j_2 N + j_3 N^2$. We choose $N = 12$ and get $N^3 = 1728$ particles in total.

Furthermore, we set the truncation parameter $m = 6$, the degree of smoothness $p = 6$, the inner boundary $\varepsilon_I = 0.1$ and the outer boundary $\varepsilon_B = 0.1$. Thus we get $R_C = \frac{1}{2} - \varepsilon_B = 0.4$ for the cut-off radius of the real space sum in (2). We choose $\alpha = 7.11$, and get

$$e_\alpha = \frac{\operatorname{erfc}(\alpha \cdot R_C)}{R_C} = \frac{\operatorname{erfc}(7.11 \cdot 0.4)}{0.4} \approx 1.4425 \cdot 10^{-4}$$

for the truncation error, see (7).

As a first step we want to determine the range $l_{\max}$ for the coefficients $d_{\boldsymbol{l}}$ in the reciprocal sum $S_2(\boldsymbol{r}_j)$ such that its calculation introduces no relative error bigger than e_α. In table 1a one can see the values for $S_2(\boldsymbol{r}_j)$ calculated by the direct algorithm DIR for increasing $l_{\max}$. Note that because of the setup of the particles all calculated values are the same for each particle in the system. We consider the computed value for $l_{\max} = 50$ as correct and therefore choose $l_{\max} = 6$.

$l_{\max}$	$S_2(\boldsymbol{r}_j)/q_j$
4	-1.2015561383818074e−16
5	-1.1473093337051994e−16
6	2.8361908863417218e−08
7	2.8361908863523468e−08
8	2.8361908863521638e−08
9	2.8361908863521578e−08
10	2.8361908863521578e−08
50	2.8361908863521578e−08

(a) For $\alpha = 7.11$.

$l_{\max}$	$S_2(\boldsymbol{r}_j)/q_j$
4	1.9180872965151214e−15
5	1.9181073806426144e−15
6	3.6291998815771393e−13
7	3.6291998815771393e−13
8	3.6291998815771383e−13
9	3.6291998815771373e−13
10	3.6291998815771373e−13
25	3.6291998815771373e−13

(b) For $\alpha = 5.74$.

Table 1: Values for S_2, computed by DIR for increasing values of $l_{\max}$ inside the regular grid for different values of α.

In figure 3a we see the maximum relative error

$$E := \max_{j=1,\ldots,N^3} \left| \frac{V_E^{\mathrm{DIR}}(\boldsymbol{r}_j) - V_E^{\mathrm{FAST}}(\boldsymbol{r}_j)}{V_E^{\mathrm{DIR}}(\boldsymbol{r}_j)} \right|, \tag{13}$$

where $V_E^{\mathrm{DIR}}(\boldsymbol{r}_j)$ denotes the result of the direct algorithm with $l_{\max} = 25$ and $V_E^{\mathrm{FAST}}(\boldsymbol{r}_j)$ the result of the fast algorithm, depending on the number of Fourier coefficients for the regularized kernel.

In addition to this we consider $\alpha = 5.74$ as the kernel parameter. Here we get

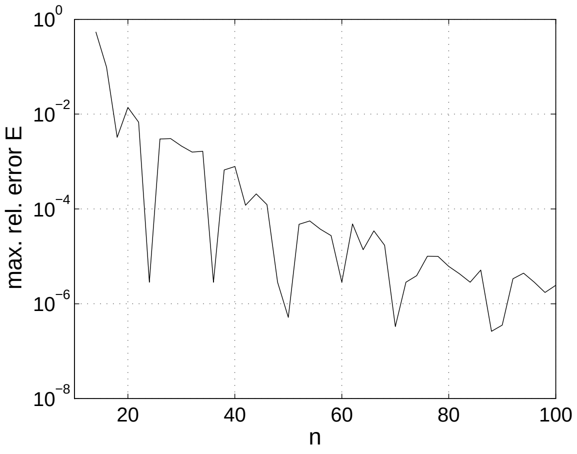
$$\frac{\operatorname{erfc}(5.74 \cdot 0.4)}{0.4} \approx 2.9154 \cdot 10^{-3}$$

as the truncation error. Similar to what we did above, we get table 1b and, after choosing $l_{\max} = 6$, figure 3b, again containing the maximum relative error E depending on the number of Fourier coefficients of the regularized kernel.

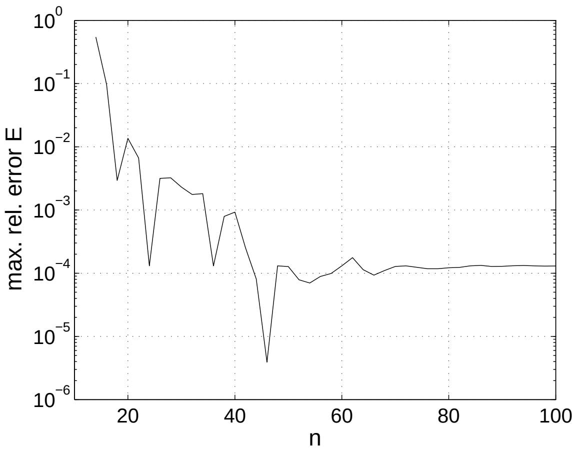
Distorted Grid

We change the setup of our system by moving the 1728 particles slightly away from their positions in the regular grid, i.e. we have

$$\boldsymbol{r}_j = \frac{(j_1, j_2, j_3)^T + \frac{1}{2} + u}{N} - \frac{1}{2}, \qquad q_j = (-1)^{j_1+j_2+j_3},$$



(a) For $\alpha = 7.11$.



(b) For $\alpha = 5.74$.

Figure 3: The maximum relative error E depending on n inside the regular grid for different values of α.

where $j = 1, \ldots, N^3$ and $j_1, j_2, j_3 \in \{0, \ldots, N-1\}$ are chosen such that $j = 1 + j_1 + j_2 N + j_3 N^2$ as above. The offset u is a pseudo random number that is uniformly distributed in $[-\frac{2}{5}, \frac{2}{5}]$. We set $\alpha = 7.11$ and therefore get a truncation error of $1.4425 \cdot 10^{-4}$. For the other parameters we use the same values as before.

In table 2a one can see the average values for $S_2(\boldsymbol{r}_j)$, $j = 1, \ldots, N^3$, depending on increasing values of $l_{\max}$. Here $\operatorname{avg}_{j=1,\ldots,N^3} |S_2(\boldsymbol{r}_j)|$ denotes

$$\frac{1}{N^3} \sum_{j=1}^{N^3} |S_2(\boldsymbol{r}_j)|.$$

We consider the results of DIR for one run with $l_{\max} = 25$ as exact. Relative to this, the third column contains the maximum relative error of DIR, which we define as

$$E_{S_2}^{\text{DIR}} := \max_{j=1,\ldots,N^3} \left| \frac{S_2^{\text{DIR},25}(\boldsymbol{r}_j) - S_2^{\text{DIR}}(\boldsymbol{r}_j)}{S_2^{\text{DIR},25}(\boldsymbol{r}_j)} \right|,$$

where $S_2^{\text{DIR},25}(\boldsymbol{r}_j)$ is the result of DIR with $l_{\max} = 25$ and $S_2^{\text{DIR}}(\boldsymbol{r}_j)$ the one with $l_{\max}$ as outlined in the table. Our choice is $l_{\max} = 8$.

$l_{\max}$	$\operatorname{avg}_{j=1,\dots,N^d} \lvert S_2(\boldsymbol{r}_j)\rvert$	$E_{S_2}^{\text{DIR}}$
5	3.6644536306526931e+00	1.4252744120429841e+00
6	3.6645144045521389e+00	1.2161400630302546e−01
7	3.6645166036632268e+00	3.7538720204219581e−03
8	3.6645166954637993e+00	1.2883961507175747e−04
9	3.6645166950460055e+00	2.4138313578243623e−06
10	3.6645166948716570e+00	2.1087638633564823e−08
11	3.6645166948693109e+00	5.3731907791577379e−10
12	3.6645166948693082e+00	3.3732887006525980e−12
13	3.6645166948693086e+00	5.9865782119568971e−15
14	3.6645166948693086e+00	0.0000000000000000e+00
15	3.6645166948693086e+00	0.0000000000000000e+00
25	3.6645166948693086e+00	0.0000000000000000e+00

(a) For $\alpha = 7.11$.

$l_{\max}$	$\operatorname{avg}_{j=1,\dots,N^d} \lvert S_2(\boldsymbol{r}_j)\rvert$	$E_{S_2}^{\text{DIR}}$
3	2.9721054254298727e+00	5.1318339982418371e+00
4	2.9722702377648695e+00	1.8849066032676384e−01
5	2.9722616806350595e+00	9.0919837455896244e−03
6	2.9722623625160316e+00	6.2611485221720402e−05
7	2.9722623705516575e+00	1.0739409216685170e−06
8	2.9722623705810456e+00	5.3703826720300918e−09
9	2.9722623705812343e+00	2.4512142280307418e−11
10	2.9722623705812272e+00	1.9465702756890063e−14
11	2.9722623705812272e+00	0.0000000000000000e+00
12	2.9722623705812272e+00	0.0000000000000000e+00
25	2.9722623705812272e+00	0.0000000000000000e+00

(b) For $\alpha = 5.74$.

Table 2: Values for S_2, computed by DIR for increasing values of $l_{\max}$ inside the distorted grid for different values of α.

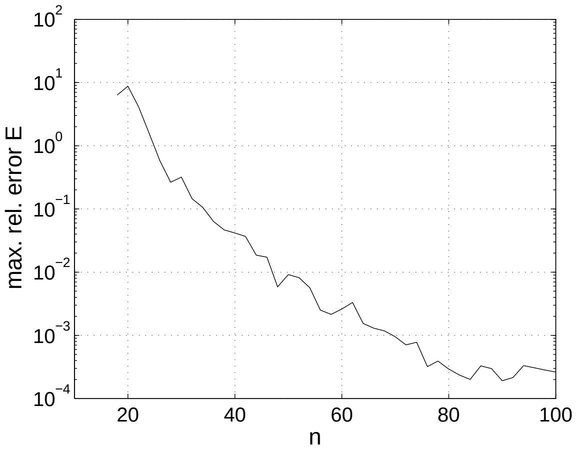
The maximum relative error E as defined in (13) depending on the number of Fourier coefficients n^3 of the regularized kernel is plotted in figure 4a. Here again the results of DIR are regarded as exact values for the error computation. We see that the computational error of our fast algorithm is much smaller when dealing with particles in regular configurations, just as we had to expect.

We repeat the whole procedure for $\alpha = 5.74$. The results are outlined in table 2b and, with the choice of $l_{\max} = 6$, figure 4b.

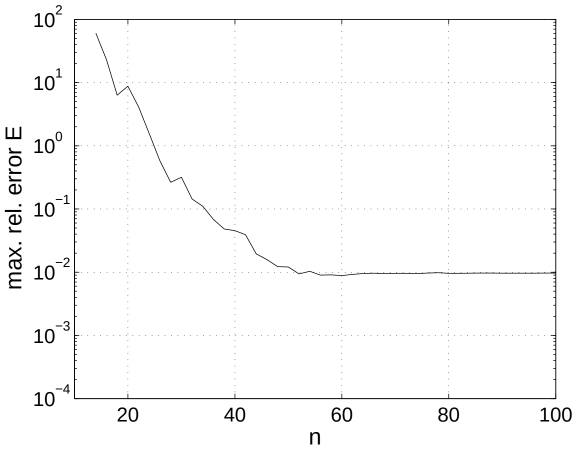
In figure 5 we have changed the size of the inner regularization of the kernel function by setting $\varepsilon_I = 0.15$ and $\varepsilon_I = 0.25$. The kernel parameter is set to $\alpha = 7.11$. The other parameters are as above. By setting $\varepsilon_I = 0.25$ we guarantee that for each particle all 26 direct neighbors are inside the nearfield of the real space sum. This increases the accuracy of the results significantly but will also result in much more computational effort for larger particle systems.

Summary and Outlook

We developed and implemented a fast algorithm that computes an approximate solution for the Ewald sum (2) and requires $O(N \log N)$ arithmetic operations. The implemetation features the computation of



(a) For $\alpha = 7.11$.



(b) For $\alpha = 5.74$.

Figure 4: The maximum relative error E depending on n inside the distorted grid for different values of α.

an highly accurate direct algorithm which can be used for error calculations. All relevant parameters can be set in the program. There are several debug output mechanisms in the program which can be activated by compiler defines.

The algorithm was tested with two different particle configurations. In these tests we tried to find good choices for the algorithm parameters. Two main requirements to succeed in this are not yet met. At first, the algorithm has to be tested with particle systems holding much more particles. Second, we can not evaluate our algorithm until comprehensive time measurements have been done. After this we well be able to compare the proposed algorithm with already existing algorithms with different approaches.

Acknowledgments

I wish to thank my supervisors Dr. G. Sutmann from the Forschungszentrum Jülich and Prof. Dr. D. Potts from the University of Technology Chemnitz for their proficient and immediate support and advise. I also thank M. Bolten and R. Speck for the great organization of the guest student program. Finally I would like to thank the other guest students for their continual assistance whenever problems arose.

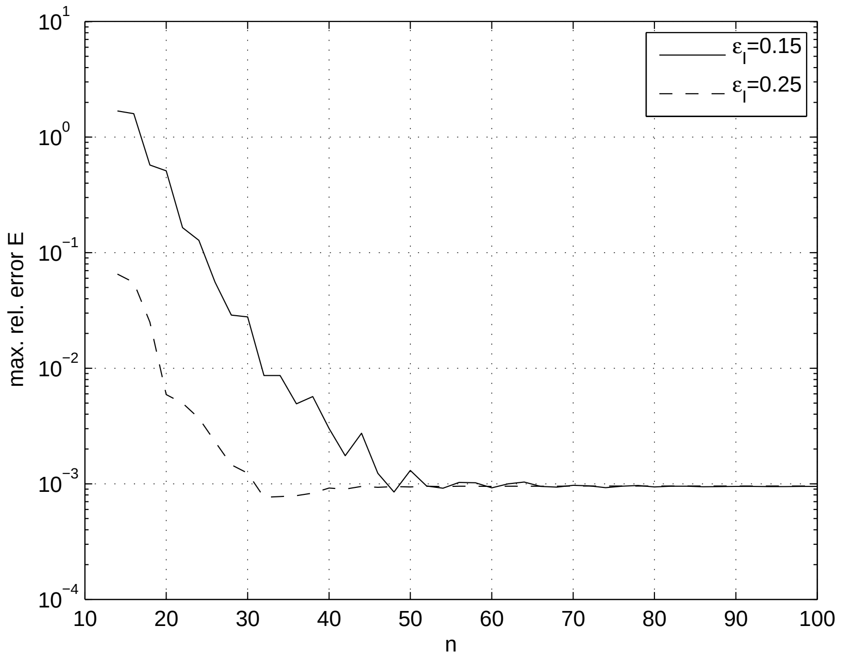


Figure 5: The maximum relative error E depending on n inside the distorted grid for $\alpha = 7.11$ and $\varepsilon_I = 0.15$ (upper line) and $\varepsilon_I = 0.25$.

References

1. D. Potts and G. Steidl. Fast summation at nonequispaced knots by NFFTs. *SIAM J. on Sci. Comput. 24,* pp. 2013-2037. 2003.
2. P. Gibbon and G. Sutmann. Long-Range Interactions in Many-Particle Simulation. *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms, Lecture Notes*, pp. 467-506. 2002.

Correcting Erroneous Quantum Algorithms

Markus Peschina

Johannes Gutenberg University Mainz

E-mail: peschina@students.uni-mainz.de

Abstract: The IQCS (Improved Quantum Computer Simulations) software package is used to simulate Grover's quantum search algorithm. The programming language C, and the machines JUMP and JUGENE are used. The simulation deals with decoherence errors and gate-imperfections and there are four different error stages implemented: ideal, erroneous, ancilla QEC (Quantum Error Correction) and full QEC.

Introduction

A Quantum Computer is able to solve some problems more efficiently than classical computation can do. The quantum mechanical generalization of a bit is called qubit. One can think of it as an orthonormal basis of a spin $\frac{1}{2}$ system. The notation for one general qubit is as follows:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1}$$

$$1 = |\alpha|^2 + |\beta|^2 \quad \alpha, \beta \in \mathbb{C}. \tag{2}$$

In this text qubits sometimes are labeled in computational basis states and sometimes qubit-wise, so $|0\rangle$ means $|0..0\rangle$ or $|4\rangle$ means $|100\rangle$. For more information to the general definition of qubits and quantum mechanics see [2], chapters 1-2.
In 1996 Lov Grover introduced his approach on quantum amplitude amplification [1], this is known as Grover's algorithm or quantum search algorithm ([2], chapter 6). We will recapitulate the very basic ideas of quantum search and shortly repeat the basics of quantum error correction (QEC) and of the Steane Code.

Grover's Algorithm

The algorithm needs one "working" register which encodes the database in a binary form and in addition some ancilla qubit to process calculations. The state may be written as $|\phi\rangle|y\rangle$, where $|\phi\rangle$ denotes (n-1) qubits and $|y\rangle$ denotes the ancilla, so the state holds n qubits. As other quantum algorithms the quantum search starts in the state

$$|\phi\rangle|y\rangle = |0..0\rangle|0\rangle = |0\rangle^{\otimes(n-1)}|0\rangle. \tag{3}$$

In order to prepare a database state which contains every possible entry up to $N = 2^n$, a Hadamard transformation on every qubit of the working register is performed (perfect superposition of every possible state)

$$\hat{H}_\phi^{\otimes n}|\phi\rangle|y\rangle = \frac{1}{\sqrt{N/2}} \sum_{x=0}^{N/2} |x\rangle\,|y\rangle. \tag{4}$$

In order to get the oracle function work properly, the ancilla should be prepared in $|y\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, this can be achieved by

$$\hat{H}_{\phi}^{\otimes n}\hat{H}_y\hat{X}_y|\phi\rangle|y\rangle = \frac{1}{\sqrt{N/2}}\sum_{x=0}^{N/2}|x\rangle\,\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{5}$$

In the further document $|\psi\rangle$ denotes the perfect superposition of (n-1) qubits $|\psi\rangle = \frac{1}{\sqrt{N/2}}\sum\limits_{x=0}^{N/2}|x\rangle$. The oracle is defined as an unitary operator $\hat{O}_{x0}$, so that

$$\hat{O}_{x0}|x\rangle\,|y\rangle = |x\rangle\,|y \oplus f(x)\rangle, \tag{6}$$

$$f(x): \qquad [0,1]^n \longmapsto [0,1]\,, \qquad \text{with } f(x) = \begin{cases} 1 & \text{if } x = x_0, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

With the previous definition of the ancilla $|y\rangle$ the effect of $\hat{O}_{x0}$ on the prepared state is a conditional phase shift of the searched computational basis state

$$\hat{O}_{x0}|\psi\rangle\,|y\rangle = (-1)^{f(x)}\,|\psi\rangle|y\rangle. \tag{8}$$

Definition 1. *Grover Operator* $\qquad \hat{G} = (2|\psi\rangle\langle\psi| - \mathbb{I})\,\hat{O}_{x0}.$

By taking account of the geometric interpretation (in [2], chapter 6.1.3) the ideal number of Grover iterations R can be determined. Let Cl(x) be the closest integer to the given real number x, then

$$R = Cl\left(\frac{\pi}{4\,arcsin(\frac{1}{\sqrt{N}})} - 0.5\right). \tag{9}$$

Quantum Error Correction

This section is a very short repetition of QEC, it is based on ([2], chapter 10) and [3], which I advise to read for further information. The basic idea of QEC is to add redundant information before erroneous calculations take place and afterwards to find which error has occurred. The simplest way to do this is the 3-qubit-code, where

$$\begin{aligned} |0\rangle \quad &\Longrightarrow \quad |0\rangle|0\rangle|0\rangle, \\ |1\rangle \quad &\Longrightarrow \quad |1\rangle|1\rangle|1\rangle. \end{aligned}$$

It is crucial to get the fact straight that there is no cloning of states at this point. If *one* error occurs inside the block to the next correction step, the position of this error can easily be determined; two errors destroy our state. In addition to classical bitflip errors, phaseflips can also appear; together with the product of both, a bitflip *and* a phaseflip, they form an orthogonal error basis. Steane's code is able to correct one bitflip and one phaseflip error in one block, whereas only 7 qubits per secured qubit are needed. It's another convenience of Steane's 7-qubit-code that CNOT, Hadamard and the Pauli gates can be applied transversally (for example, the X-gate cannot applied transversally on the 3-qubit-code encoded state). As a penalty the transversal T-gate is not possible on steane codes. Because this fact has some impacts on the implementation of Grover's algorithm, I will refer to this point. The logical zero is encoded in the equally weighted superposition of all even weight codewords of the Hamming code (even numbers of 1's, see [3]):

$$\begin{aligned} |0\rangle_{\text{Steane}} \quad \Longrightarrow \quad & \frac{1}{\sqrt{8}}\Big(|0000000\rangle + |0001111\rangle + |0110011\rangle + |0111100\rangle \\ & + |1010101\rangle + |1011010\rangle + |1100110\rangle + |1101001\rangle\Big), \end{aligned} \tag{10}$$

$$|1\rangle_{\text{Steane}} \quad \Longrightarrow \quad \frac{1}{\sqrt{8}}\Big(|1111111\rangle + |1110000\rangle + |1001100\rangle + |1000011\rangle$$
$$+|0101010\rangle + |0100101\rangle + |0011001\rangle + |0010110\rangle\Big). \tag{11}$$

In this basis we are able to find and correct bitflip errors. In order to find phaseflips we have to rotate the basis by applying one Hadamard rotation to each of the seven qubits (transversal H operator):

$$\hat{H} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}. \tag{12}$$

In the rotated basis bitflips are phaseflips and vice versa, so that we can correct phaseflips (which are now bitflips). The correction step needs 5 additional ancilla qubits for the purpose of fault tolerant processing. The fundamental fact at this point is that in order to find errors, the error information of the unknown pure state $|\psi\rangle$ is "copied" onto the ancillas, but the state of the ancillas has been carefully chosen (Shor state) to ensure that only information about the error can be read by measurement.

If we store one qubit in an unknown pure state $|\psi\rangle$ on a storage device with imperfections and the recovered state is denoted by ρ_{out}, we are going to detect a loss of fidelity:

$$F = \langle\psi|\rho_{\text{out}}|\psi\rangle = 1 - \varepsilon. \tag{13}$$

If we use Steane's code to encode the information in an 7-qubit-block then the recovered state can maintain an improved fidelity, even if the encoding, decoding and error-recovery processes will suffer from the same error:

$$F = 1 - O(\varepsilon^2). \tag{14}$$

IQCS

Improving Quantum Computer Simulations is a software package to simulate quantum computer algorithms. The basic concept is to store the complex amplitudes of the whole state vector of a pure state $|\psi\rangle$, constituted by n qubits. One qubit gates (which are $2^n \times 2^n$ matrices) have a clear inner structure so they are applied to the state vector on the fly, instead of storing the whole data. The same is true for two qubit gates. Among others the IQCS package implements CNOT, H, $\pi/8$- gates, which form a universal set of gates.

Implementations

There are four different implementations: In the first one all errors are equal zero, in the second one operational and decoherence errors are enabled (error model is discussed later on), the third implementation tries to stabilize the algorithm by encoding and correcting only the ancilla qubit and the last is the fully corrected algorithm.

Implementation of ideal Grover algorithm

If we take advantage of the following identities

$$\hat{H}\hat{H} = \mathbb{I} \qquad \text{and} \qquad \hat{H}^{\otimes n}|0\rangle = \sum_{x=0}^{N}|x\rangle \tag{15}$$

the Grover operator as defined above (Definition 1) $\hat{G} = (2|\psi\rangle\langle\psi| - \mathbb{I})\,\hat{O}_{x0}$ can be rewritten as

$$\hat{G} = \hat{H}^{\otimes n}(2\,|0\rangle\langle 0| - \mathbb{I})\hat{H}^{\otimes n}\,\hat{O}_{x0}. \tag{16}$$

The matrix representation of an additional phase shift to state $|0\rangle$ equals the matrix representation of $\hat{O}_0$ (oracle searching the state $|0\rangle$). So we rewrite the Grover operator according to

$$\hat{G} = \hat{H}^{\otimes n}\,\hat{O}_0\,\hat{H}^{\otimes n}\,\hat{O}_{x0}. \tag{17}$$

As $\hat{H}$ is already implemented in previous versions of IQCS, the only need is to implement the oracle function. It is crucial to have in mind that the way to implement the Grover operator is not unique. Besides it is possible to implement the additional phase shift without using the ancilla. As an impact of the implementation which is chosen here, the ancilla becomes more important to the stability of the algorithm, because it is accessed twice per grover iteration instead of once.

By convention the oracle is of kind $C^n\,NOT$-Gate, whereas the C^n functionality is implemented as black box. The ideal implementation of the black-box-oracle is called fCNOT-function. First thing to do is finding the rank of the processor and finding the index of the searched state. The variable NSTATES is defined as the number of states stored per processor and x is given as computational basis state (int).

```
int rank = x / NSTATES;
int i = (x % NSTATES);
```

After rank and index are determined a *local* NOT operation is performed. The operation is always local on one processor, because the ancilla qubit is at position 0.

```
Start in $|0\rangle^{\otimes(n-1)}\,|0\rangle$
    1. $\mathbb{I}^{\otimes n-1} \otimes \hat{X}_{ancillar}$
    2. $\hat{H}^{\otimes n}$
    3. perform R (see eqn. 9) grover iterations:
        $\rightarrow \hat{G} = \hat{H}^{\otimes n}\,\hat{O}_0\,\hat{H}^{\otimes n}\,\hat{O}_{x0}$
        $\rightarrow$ save data (amplitudes, etc.)
    4. write out data
```

The algorithm is performed as described in the left box, whereas one grover iteration denotes one sequential use of the grover operator of eqn. (17). The main load of the algorithm are the repetitive application of Hadamard-gates to the working qubits. There is no need for any statistic iteration, since every run provides exact the same result.

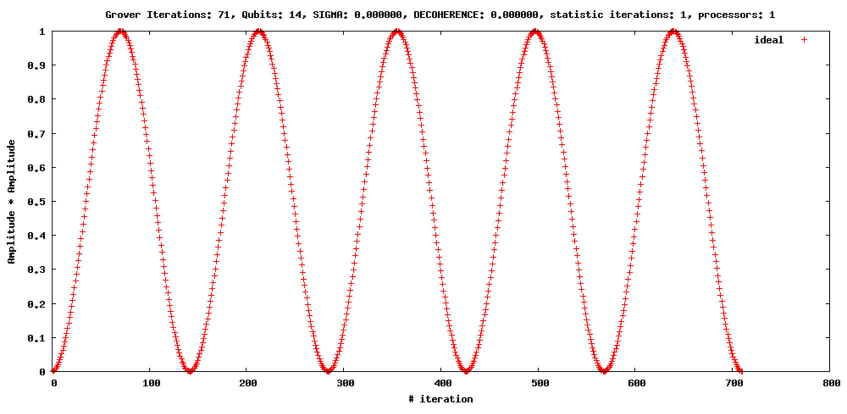


Figure 1: Ideal implementation, no statistic

The result for this first implementation is shown in fig. (1). Because of the graphical interpretation that was mentioned before, a squared sinus function is estimated for this graph. The simulation ran with 14 qubits, whereof 1 is the ancilla and the others are labeled “working qubits” (encoding the database). The x-axis is assigned to the number of sequential grover applications and the y-axis to the probability of measuring the searched state. The ideal number of grover operator applications is printed in the title and labeled “Grover Iterations”. The amplitude of the searched state is found inside the program using the same method as in the C^n black box. The value is summed over both possibilities of the ancilla qubit since the measurement is performed only on the working qubits.

Implementation of erroneous Grover algorithm

The error model of the IQCS software package consists of operational and decoherence errors which are labeled “sigma” and “deco” at this point. Decoherence is modeled using the deco error to represent the environment-qubit interaction. Every timestep each qubit suffers from decoherence, whereas one timestep is simply the duration of one quantum operation (one must have in mind that it is possible to do operations in parallel at this point). With probability $(1 - q)$ nothing happens to the qubit, whereas with probability $q/3$ each either bitflip, phaseflip or bitphaseflip is applied to the qubit. Gate-imperfections are modeled using the sigma error. For this purpose every one- and two-qubit gate is fragmented into planar rotations R_ϵ or phase shifts P_ϵ:

$$R_\epsilon(\theta) = \begin{pmatrix} cos(\theta+\epsilon) & -sin(\theta+\epsilon) \\ sin(\theta+\epsilon) & cos(\theta+\epsilon) \end{pmatrix}, \qquad P_\epsilon(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i(\phi+\epsilon)} \end{pmatrix}. \tag{18}$$

The parameter sigma denotes the standard deviation σ of a Gaussian distribution $f(\epsilon) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\epsilon}{\sigma})^2}$ for the angle error ϵ. For example the Hadamard operation:

$$\hat{H}_\epsilon = R_\epsilon(\frac{\pi}{4})\, P_\epsilon(\pi). \tag{19}$$

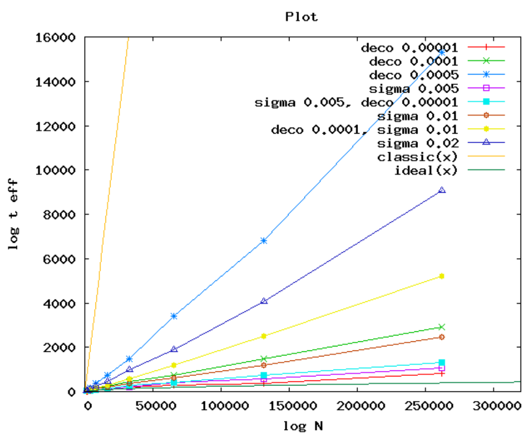
The way to implement the fCNOT-function is straight forward: C^n part is the same as before (because the memory layout is the same), but the NOT-operation is composed out of (18). In order to perform a *local* operation, only those ranks storing information of the searched state x_0 join in the fCNOT-function, the others are idle. For convention the preparation of the database and the ancilla qubit is free of any error, only the grover operator is erroneous. There are done several runs with different single and combined errors. In general the maximum amplitude of the searched state is smaller for higher error rates. The more interesting fact is that its position is reached with less applications of the grover operator (as described in [4]). The data is taken and processed in the same way as fig. (1). The area around the maximum amplitude is fitted using a parabolic function; its parameters are used to get the abstract position of maximum amplitude and the probability of x_0. Using different values of n, sigma and deco leads to the plot shown in fig. (2a). But in order to compare the probabilistic quantum algorithm with the classic, non probabilistic search of an unstructured database, a variable t_{eff} is introduced:

$$P \overset{!}{=} 1 - (1-p)^k, \tag{20}$$

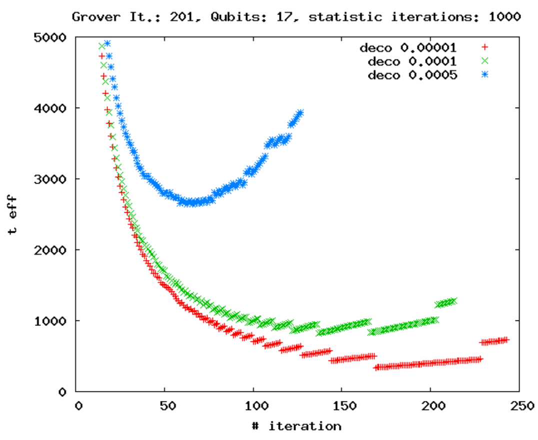
$$t_{eff} = k \cdot N_{\text{Grover}}. \tag{21}$$

Whereas p is the probability to measure the searched state x_0 and P is set to $P = 0.99$. The idea is to repeat quantum search until the probability to fetch x_0 is at least P. The “time” needed for this repetition is t_{eff}.

Errors provide less computational effort to reach the maximum possible amplitude. But fig. (2b) shows that it is not possible to save computational power due to errors. If it was possible, curves with different error rates would cross each other. Furthermore the minima of more erroneous curves would occur at



(a) Comparison plot - The classic(x) and ideal(x) curves are functions, so they do not suffer from integer divisions. This is the reason why the other curves seem to have discontinuities.



(b) Is it possible to gain something by errors? The granularity of this plot is because of the definition of t_{eff}: The type of the parameter k is integer, hence there are discontinuities.

Figure 2: Plots

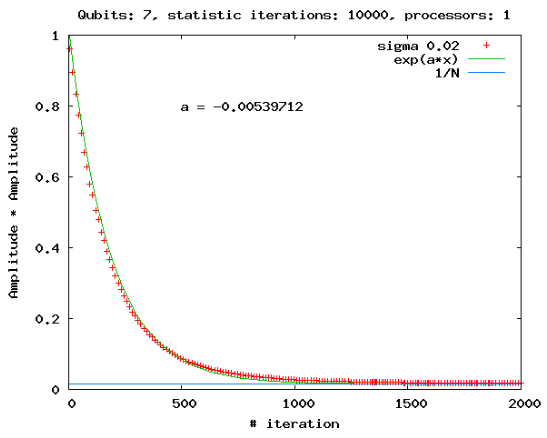
lower iteration numbers as the minima of less erroneous curves. Within the scope of $n = [9; 20]$ more plots like fig. (2b) are gathered to foreclose special cases.

There are two different approaches to analyze the current error model. As previously discovered in [4] the amplitude of x_0 decreases by fulfilling an exponential law $A_{x_0}(n_{\text{iteration}}) = e^{\alpha n_{\text{iteration}}}$. Due to conservation of probability in long running times the amplitudes of all states should (in the average case) equal $\frac{1}{N}$. To be able to confirm this result (even though in [4] is used a slightly different error model) there are made some different runs with much more applications of Grover's operator and different values of sigma/deco (fig. 3a). The results are shown in fig. (3c), (3d). The dependency of the exponent of decrease α on the deco parameter shows a nearly linear behavior, in contrast to the more or less quadratic dependence on the sigma parameter. This behavior is because the sigma parameter refers to a deviation and the deco parameter is a probability. As second approach to analyze the error model, a couple of runs are taken with different sigma and deco. The result is an array of curves: One error is left variable while the other one is used as parameter for the set of curves and vice versa. The plots (see fig. 3b) should look like the inverse of the so called "error-norm"-plots in [5], and indeed they are.

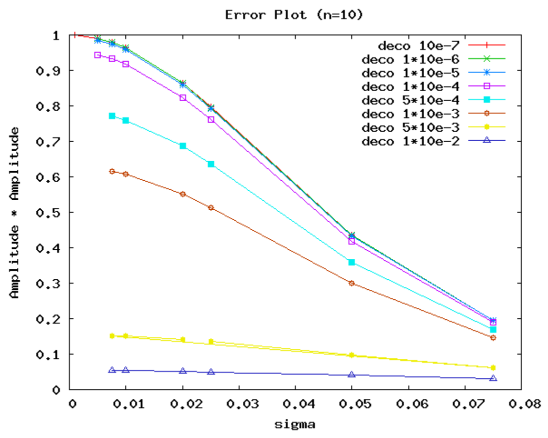
The data is gathered by building the arithmetic average over each amplitude value of many independent runs each starting from the same errorfree initial state. There are some statistical artifacts which are not related to the quantum search algorithm. Taking the geometrical interpretation of amplitude amplifying into account, the grover operator can be described as 2 dimensional rotation which rotates only by a fixed angle β. A first approach to transfer the sigma/deco error model into this simplified interpretation is that β is no more a fixed angle, but varies with deviation δ_k for each rotation k. In a sequence of rotations it is clear that $\delta_k < \delta_{k+1}$ since errors propagate quadratically. In this model there is only a flux of probability from one state to another by courtesy of rotations (which is a simplification), so every amplitude-vector $\hat{G}^k|\psi\rangle$ lies on a circle around the origin for every k. Whereas

$$\hat{G} \in SO(2) \qquad \text{and} \qquad |\psi\rangle \in \mathbb{R}^2. \tag{22}$$

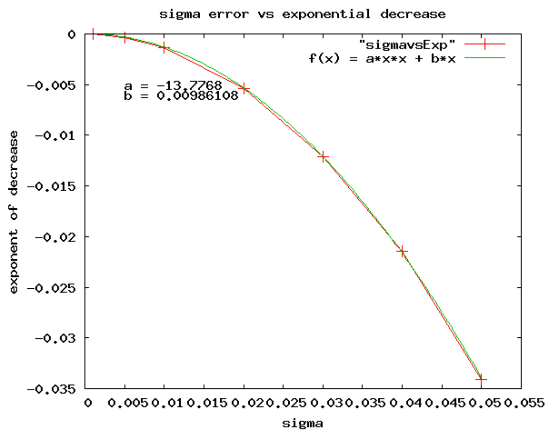
By taking the average over all statistical iterations points laying on a segment of a circle of length corresponding to δ are summed up. Due to this summation the mean norm of the resulting vector $|\psi\rangle_k$ after k rotations becomes smaller (see fig. 4a). To relate this to the whole algorithm: Even if there are no probability transferring errors, the over-all probability will decrease because of taking the mean value. This effect has another impact onto the minima, by taking the mean value for every probability, sometimes



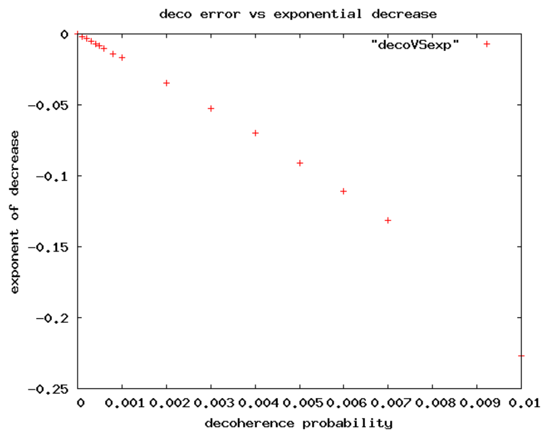
(a) An exponential law is used to describe the decay of the amplitude of the searched database entry. The constant part is because of conservation of probability during the algorithm.



(b) The behavior of quantum search while considering different errors.



(c) Dependence of the exponent of decrease on the sigma parameter. The quadratic part of an polynomial interpolating function is highly exaggerated.



(d) Dependence of the exponent of decrease on sigma error. The behavior is nearly linear.

Figure 3: Plots analyzing the current error model.

points laying on the flank of the minimum are summed up into the minimum (compare fig. 4e). As a result the minimum at position k has a higher probability $p_{\mathrm{min},k}$ as in a single run and also:

$$\delta_k < \delta_{k+1} \quad \Rightarrow \quad p_{\mathrm{min},k} < p_{\mathrm{min},k+1}. \tag{23}$$

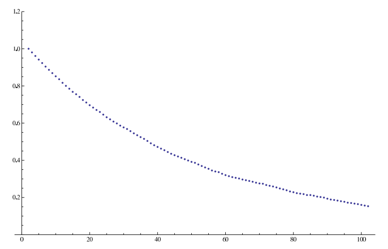
But both the whole curve and the minima obey a law of exponential decrease, so there is a maximum $p_{\mathrm{min},j}$

$$\exists j, \forall k : p_{\mathrm{min},j} > p_{\mathrm{min},k}. \tag{24}$$

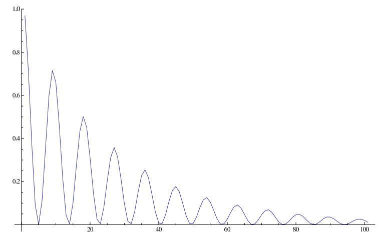
To verify that this effect is due to statistical averaging, a vector-matrix multiplication with independent erroneous rotation angles has been implemented using *Mathematica*, the results are shown in fig. (4).

Stabilized ancilla qubit

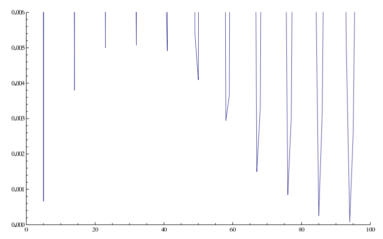
The main load of the algorithm is to calculate Hadamard operations. If they work faultily only a full QEC scheme helps to correct them. The only other operation is the oracle operator, or fCNOT-function. It only works properly if the ancilla qubit is well prepared in the state $|y\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. So the first basic



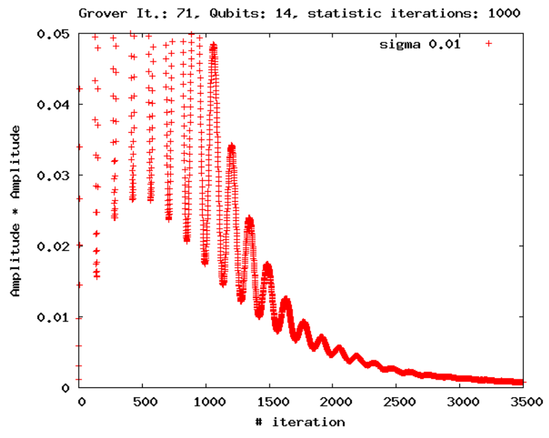
(a) The average norm of a 2 dimensional vector over 10.000 statistical iterations. The vector is rotated 100 times each iteration, using imperfect rotation angles. The initial state is similar to the Grover algorithm initial state $|\psi\rangle$, the rotation angle is β and the system refers to $n = 7$ qubits. Because of the summation to get the mean value, the norm decreases.



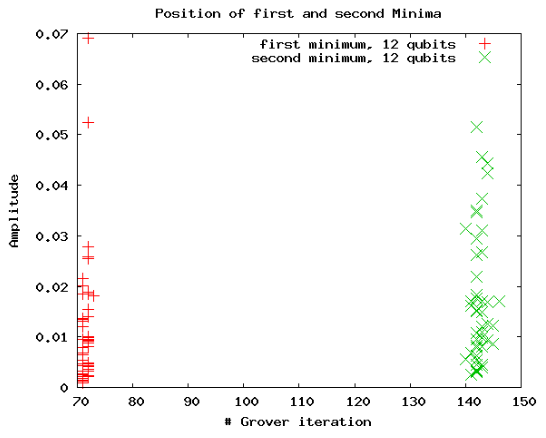
(b) Simplified 2 dimensional model to reproduce the output of grover simulation. Only using imperfect rotations and adjacent building of the mean value, the resulting envelope seems to decrease exponentially.



(c) The same as picture the left one, but magnified to identify the behavior of the minim-as.



(d) Simulation of Grover's algorithm with IQCS. Magnified y-axis to see the behavior of the minim-as.



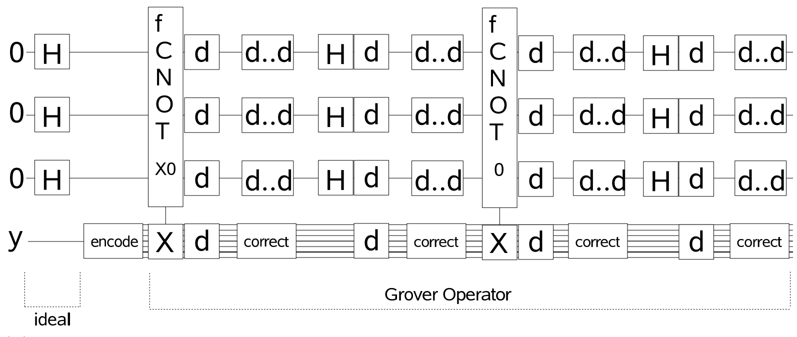
(e) The position of the first and second minimum differ in each statistical iteration, but to get the average they all are summed up into one value respectively.

Figure 4: Plots analyzing statistical artifacts.

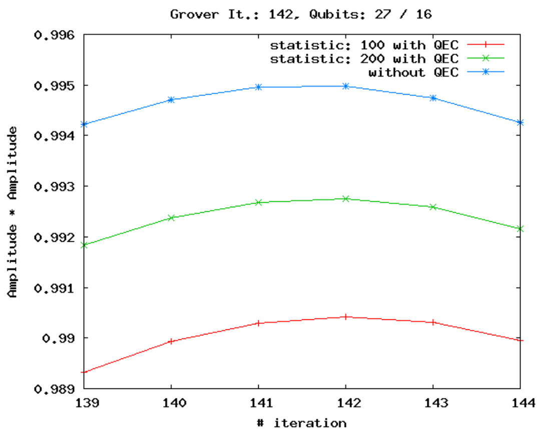
approach to stabilize the algorithm is to stabilize the ancilla qubit. For this purpose a 7-qubit Steane code is used. The encoding and decoding functions in IQCS are implemented in a fault-tolerant way, so the Steane code needs 5 additional qubits, so called "measurement" qubits. The whole memory layout has to be adjusted:

$$\underbrace{n_w}_{\text{working qubits}} + \underbrace{7}_{\text{ancilla qubit}} + \underbrace{5}_{\text{measurement qubits}} . \tag{25}$$

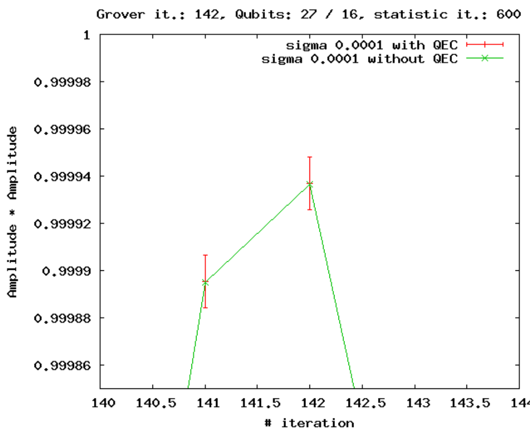
The algorithm is implemented as shown in fig. (5a). The main task is the implementation of the fCNOT-function: The NOT-part is realized transversally (a huge advantage of Steane codes), but the C^n-part differs a little bit from previous implementations. At the final measurement the state of the 7 ancilla qubits and the 5 measurement qubits does not matter and is therefore not measured. Therefore all amplitudes of the 2^{7+5} different lower order bit states can be summed giving the amplitude of the working qubits basis state respectively. As the only interesting amplitude is the one corresponding to x_0, only the sum of these 2^{7+5} possible states contributing to x_0 are calculated and written out each grover iteration. In the same manner the C^n-part of the fCNOT-function has to find the right 2^{7+5} block and then perform the transversal NOT operation. In general the different ranks have to communicate to do this, so the variables rankStart, rankStop, indexStart, indexStop are calculated in order to find the right ranks, which are contiguous, and the right index where the data is stored. Afterwards a new split-communicator is created in MPI and the NOT operation is performed within the local index range of indexStart and indexStop.



(a) circuit diagram - only the preparation is ideal. The "y" denotes the well prepared ancilla-state $|y\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.



(b) Simulated using sigma=0.001. Only the first maximum in probability of x_0 is illustrated. The statistical iterations are very insufficient, but the probability seem to become higher.



(c) The same plot as to the left, but with sigma=0.0001. The statistical iterations are insufficient too, which is illustrated by the standard deviation of the measurement.

Figure 5: Analyzing stabilization of ancilla qubit.

The results of the implementation to stabilize the ancilla qubit are unfortunately still unfinished. Problems are the insufficient statistical iterations and therefore too little computational power. The achieved outcomes of the simulation are not precise enough to decide whether the approach of stabilizing the ancilla is stabilizing the whole algorithm. There are done measurements with 2, 16, 22, 27 and 28 qubits; in case of 2 qubits the results are discussed later on, together with full QEC schemes. Because much more statistical iterations are needed to provide proper results in the presence of decoherence, this error is switched off and only the sigma error is simulated. The task is to find a maximum error threshold the current correction scheme provides an improvement in the maximum probability of x_0. In general the former discussed error threshold is only true in case of full QEC. So the searched threshold could depend on the size of the system. The simulation is very expensive, because to find a threshold, the standard deviation of the mean amplitudes must be disjoint. Therefore the investigation is focused on the n=2 and n=27 qubits case to get more precise results. For the case n=27, one run denotes 100 statistical iterations and take time of $\approx 10hs.$. In fig. (5c) one run is drawn and the result of merging two runs. The problem is, that one run includes only one event that cannot be corrected by the current QEC scheme (2 bitflip or 2 phaseflip errors in one block). In order to take the right mean value it is crucial that at least 100 or more "events" take place, so there is a need of at least a factor of 100 more statistics to get acceptable results using this method. The approach to find the threshold in case of smaller operational errors is displayed in fig. (5c). The smaller error rates worsen the problem as events of uncorrectable errors appear even more rarely. Our preliminary result is that it is not possible to correct the whole algorithm by only stabilizing the ancilla qubit.

Stabilized algorithm

The last implementation that is done concerns the fully encoding of all "working" qubits and the ancilla qubit. The new memory layout is

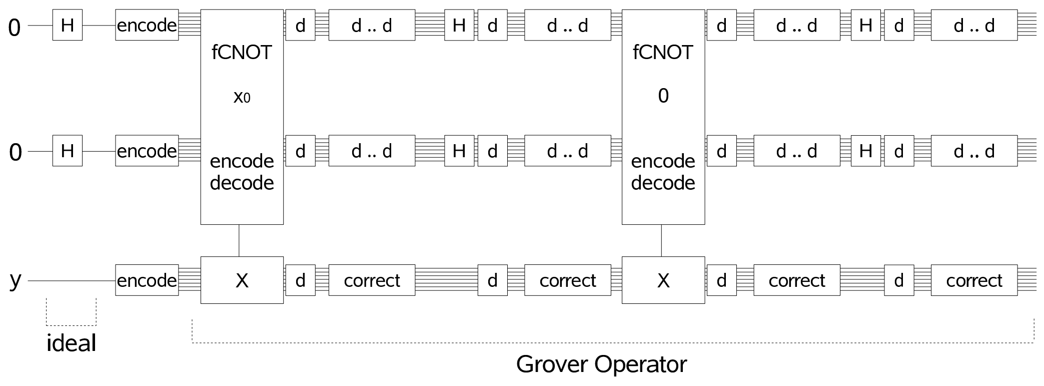
$$\underbrace{7 \cdot n_w}_{\text{working qubits}} + \underbrace{7}_{\text{ancilla qubit}} + \underbrace{5}_{\text{measurement qubits}} . \tag{26}$$

Because the complexity of this scheme is much higher than in previous implementations, our simulation is limited to $n_w = 2$, which uses a total number of $n = 26$ qubits. The total state vector needs $1.6gb$ of memory (which is doubled for every additional qubit), the next possibility $n_w = 3$ needs $206gb$. Furthermore by using eq. (9) with $n = 2$ the value R denoting the grover iterations becomes $R = 1$ and the possibility to measure x_0 after 1 grover iteration is $p = 1$, so every value except 1 must occur due to errors. The implementation of the c^n-part of the fCNOT-function is now more artificial as in previous implementations. The states of the working qubits are now distributed over the whole state vector. There are three possibilities of how the C^n can be implemented. The first one **(a)** is to use unitary operator decompositions, but for this case a $\frac{\pi}{8}$-gate is necessary and this can not be implemented transversally on Steane codes. The second one **(b)** is to perform ideal decoding before each fCNOT, then read out the amplitude as before and afterwards to perform ideal encoding. The disadvantage is that there is a passive error correction included, since the whole possibilities of 2^7 states are projected into one state. This passive correction has no physical reason and is treated as artifact of the method. The third one **(c)** discussed here is to calculate the exact position and blocksize of all 8^{n_w} Steane code parts which form one computational basis state and perform a local NOT operation on this states. This third method is also very artificial, because the other states, beside the Steane code states, could have nonzero amplitudes in case of errors, but these amplitudes are completely ignored. The implementation here follows the second **(b)** possibility.

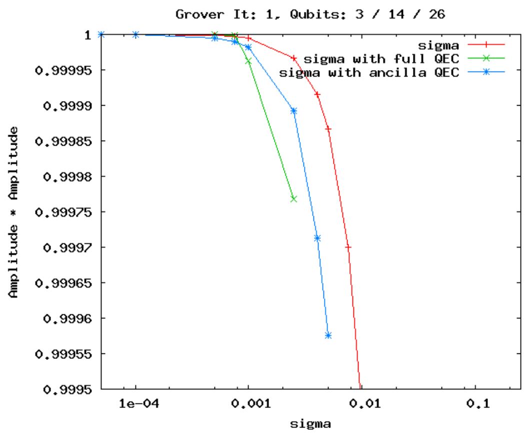
The first approach is to encode all qubits, but introduce correction steps only onto the ancilla qubit, as it is displayed in fig. (6a). The results of several runs with different sigma errors (the decoherence is again turned off to reduce computation time) are shown in fig. (6b). Stabilizing only the ancilla qubit in the case of $n = 2$ again turns out to be unsuitable to stabilize the whole algorithm. Although the statistics is insufficient, the trend of the curve points to worse values. Up to $\sigma = 0.0005$ full QEC improves the probability to find x_0 compared to uncorrected Grover. As expected, the full QEC scheme gets worse for higher sigma values (threshold lays between $[0.001; 0.0005]$), this is because of the enhanced length of the algorithm due to error correction. This approach of a fully corrected Grover algorithm works because of the passive correction that takes place during the fCNOT-function. The next improvement is to introduce correction steps to each of the encoded working qubit, at the same place where the correction steps of the ancilla qubit are performed. The difference of this "really" full QEC scheme to the previous one is marginal (see fig. 6c) (this fortifies the prediction that the passive correction has a huge influence to the algorithm).

Scaling

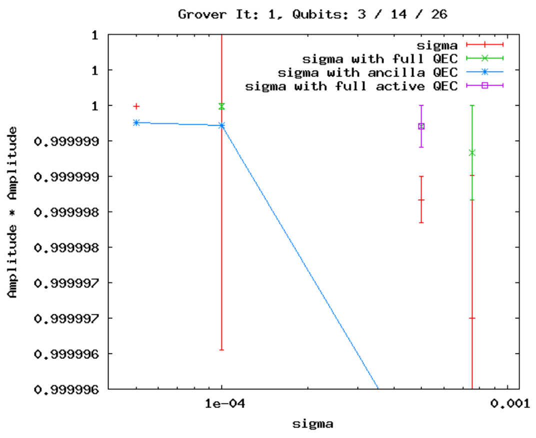
The scaling measurements are done for every different implementation. The time to perform the quantum search is measured using the MPI command "Wtime()" and the data is displayed as speed-up plots. The ideal case would be a straight line, as running the task with doubled number of processors should be done in half the time. The best performance until system size of $n \approx 19$ qubits can be achieved in running



(a) circuit diagram - only the preparation is ideal. The “y” denotes the well prepared ancilla-state $|y\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.



(b) Plot comparing three QEC schemes in case of n=2. Statistics: without QEC 100,000,000, ancilla QEC 3,200,000 and full QEC 30,000 iterations. The worst method is to apply correction steps to the ancilla. For higher errors the full encoding scheme becomes worst.



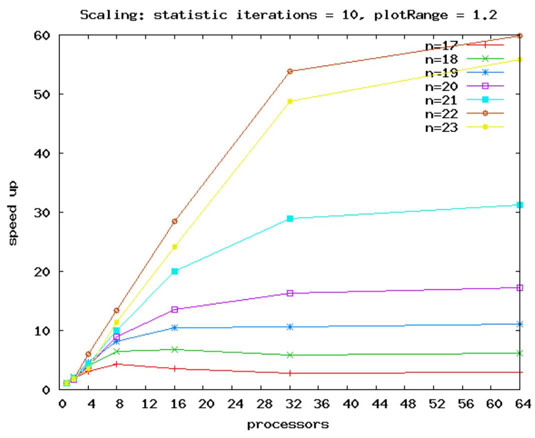
(c) The same plot as the left one, but with magnified y-axis. There is one interesting point: At sigma=0.0005 full QEC is better than no QEC. This point is simulated using full QEC and full, active QEC (with correction steps on all qubits).

Figure 6: Analyzing stabilization of all qubits.

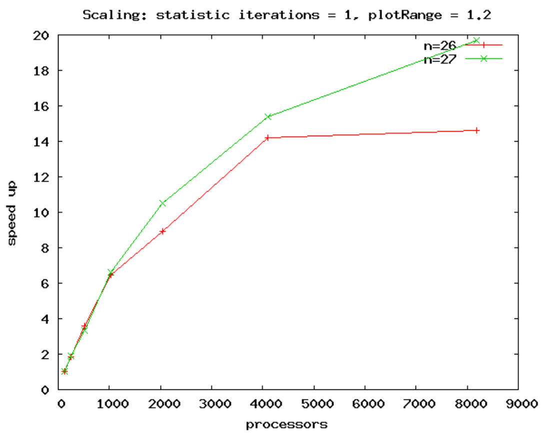
serial jobs (see fig. 7a). The interesting thing is that in case of n=[20;23] the simulation behaves in a super-scalar manner. There are two opposed effects: The time needed to calculate the operations on the local processor and the time needed to send the data through the network. If the local data is too small, the communication will delay the calculation, on the other hand if the data is too big, the local performed calculation will delay the whole program. The optimum system size seems to be n=22, as the super-scalar manner reaches its maximum. This can be understood as cache effect, as the cache-size on the JUMP system is 4mb and the local memory needed to store the state vector is in the same scale. But, this effect is not analyzed deeply during this work, as the simulation time in the region where the effect occurs is too long to get results in an equitable timescale.

Conclusion and Outlook

Grover’s algorithm was implemented using the IQCS software package and the programming language C. There were four different implementations: ideal, erroneous, ancilla QEC, full QEC and the algorithm could suffer from decoherence errors and imperfections in gate applications. The error model was analyzed and the results are suitable to further investigations. We could quantify the dependence of the probability to find the searched database entry on decoherence and operational errors. We analyzed the



(a) Machine: JUMP



(b) Machine: Jugene.

Figure 7: Scaling plots

deviation of the searchtime ideally scaling as $\sim \sqrt{N}$ in the presence of these errors. There are different statistical artifacts: Displacement of minima, decrease of the amplitude in dependence of the number of grover applications (additional to the "real" decrease, which can be seen in single runs). As a preliminary result (due to low statistics) stabilizing only the ancilla qubit does not result in a higher amplitude of the searched element. Within the black box implemented fCNOT-function the correctness of QEC could be verified. For our investigated system we could quantify that up to $\sigma_{\text{thres}} \approx 5\cdot 10^{-4}$ full QEC does improve the grover algorithm compared to non QEC. As forecast the next logical step is the implementation of a transversal $\frac{\pi}{8}$-gate as shown in [6], to make the fCNOT-function a unitary operator.

Acknowledgment

I would like to thank Dr. Mathias Bolten and Robert Speck for the great organization of the guest student program. Special thanks goes to my supervisors Guido Arnold, Marcus Richter and Binh Trieu who have always answered my questions and have made this work to be done. Further I want to thank Prof. Dr. Schömer who advised me to this program. And finally I have to thank the other guest students for the productive and humorous atmosphere.

References

1. L. K. Grover, A fast quantum mechanical algorithm for database searching, Proc. 28th Annual ACM Symposium on Theory of Computing (STOC), 1996, 212-219.
2. M. A. Nielsen, Isaac L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press (2000), ISBN 0-521-63235-8
3. J. Preskill, "Reliable quantum computers", Proc. Roy. Soc Lond. A 454, 385-410 (1998), quant-ph/9705031v3
4. P. J. Salas, Noise Effect On Grover Algorithm, European Physical Journal D, vol 46, number 2, year 2008
5. G. Arnold, M. Richter, B. Trieu, and Th. Lippert, Improving Quantum Computer Simulations, PARS Conf. Proc. 2007, PARS-Mitteilungen Nr.24, ISSN 0177-0454, 166-175, (2008).
6. A. G. Fowler, Constructing arbitrary single-qubit fault-tolerant gates, quant-ph/0411206v1, 30. Nov. 2004

SIONlib
Scalable I/O library for Native Parallel Access to Binary Files

Ventsislav Petkov

Computational Science and Engineering
Technische Universität München
Boltzmannstraße 3, D-85748 Garching
E-mail: v_petkov@mytum.de

Abstract: High Performance Computing is playing a major role in the research and the development process nowadays. The increasing performance of the computers and the development of the data storage devices open the door for new research fields and discoveries. However, the increasing number of processors used in the supercomputers today poses a new problem to the developers: the scalability of the applications.
One of the major factors for this is the data access and storage. How should the data be written in parallel? Is there an optimized way for doing this? How are the files maintained?
SIONlib is an attempt to answer these questions by providing an easy and efficient interface for working with binary files in parallel.

Introduction

SIONlib, the **S**calable **I**/**O** library for **N**ative parallel access to binary files, started as an output library for *Scalasca*[1], a performance analysis toolset, but later on *SIONlib* was separated from measurement toolkit. It is a small library for writing and reading data in parallel from thousands of processors and it is especially designed to work with binary files. This file access pattern is the most commonly used in many different applications, e.g.,

- Trace files for performance analysis tools
- Scratch/Temporary files
- Application dependent checkpointing

The library provides a simplified file handling for parallel programs that formerly had to read or write binary data in parallel from/to separate files. Handling of thousands of files is no more needed as there is only one or a few big files containing all the data.
Reading/writing can be done using the standard C-I/O interface so minor changes are needed to convert an existing code to work with *SIONlib*. Just the standard *fopen()* and *fclose()* should be replaced with a customized version from *SIONlib*. Later on, the normal *fread()* and *fwrite()* can be used for reading or writing binary data using that pointer.
Moreover, when creating a file, *SIONlib* aligns the data blocks of that file with the file system blocks.

Idea and Motivation

The increasing computational power of the computers and the decreasing price of the data storage devices provide new research opportunities. Simulation programs are becoming more and more complex and they are dealing with huge amount of data. Storing or retrieving this data might take a major part of the program's runtime and has a great influence on the scalability of the applications. There are different problems and solutions when it comes to parallel I/O and some of them are presented in the following sections.

File locking

Distributed filesystems are mainly used for the high performance computing to provide storage space and parallel access to files. They exchange information about the blocks that compose a file but not about the files themselves. This approach has the advantage of packed placement of the data and increased performance.

However, the integrity of the filesystem should always be maintained. This is done by protecting the file system meta-data from concurrent access using a mechanism called *Block Locking*. This is done using low level locks and has the effect of serializing parallel accesses to the internal data structures of the filesystem.

The same procedure is also used to maintain data consistency of the files. No two processes are allowed to write to the same block at the same time. The internal structure of the filesystem is mainly based on *inodes*. Each one of them contains all the information about a file except the data stored in it (filename, size, access rights, number of links to this file, etc). It is vital to protect this information from concurrent modification in order to maintain the integrity of the filesystem.

Parallel file access strategies

There are different approaches when working with files in parallel. They solve some of the problems one faces when using files in parallel but also introduce different problems.

The two most common ones are:

- Parallel I/O using separate files
- Parallel I/O using native direct access

The first strategy is the easiest to implement and it is the most common one. It uses a separate file for each process so the number of files depends directly on the number of processors. Due to the increasing number of CPUs nowadays, this approach has problems with scalability and maintenability. The main problems are:

- Serialization on the output folder:
  Directories are a special case of a file in Unix. As such they also have *inodes*. This inode contains information about the files inside the folder and it is updated when a file is either added to or removed from the directory. This information should be kept consistent during parallel accesses so the same mechanism of locking is used on the folder's inode.

Having to create thousands of files at once will inevitably lead to folder locking and the process will be serialized. This has a impact on the performance and rises major scalability problems.

- Difficult to handle and backup the files:
  Execution on more than 10k cores will create a huge amount of files. Migrating and archiving so many files and organizing the data are a complicated and time consuming tasks. This situation may also lead to fragmentation of the filesystem.

The second common approach is using native direct access for working with the files. It solves the problem of maintenability by storing all the data in one big file. Unfortunatelly, it brings other problems:

- The filesystem locks the access to the data block-wise in order to maintain data consistency.
  Tasks writing to the same data block need to wait and the process is again serialized. This leads again to low performance and poor scalability.

- No information about the file structure such as start of the data for each task, size, structure, etc, is available.
  The data structure inside the file should be hardcoded in the specific program and it will not be portable. Creation of generic tools for working with the files will not be possible.

- Chunksize should be specified in advance.
  The precise size of the data that will be written to the file should be known in advance. This size cannot change after the file is created.

MPI I/O is also part of this parallel file access strategy. Its performance and features strongly depend on the specific implementation. However, it is designed to work with derived datatypes and not directly with binary data. This approach provides flexibility but makes the I/O interface more complex - special care should be taken when working files. What is more, it does not use a standard C-I/O interface making it difficult to integrate in existing software code. [4]

Features of SIONlib

In an attempt to solve these problems and provide an easy and efficient way of working with files in parallel, *SIONlib* was designed. It solves many of the problems that the other two approaches are facing and at the same time staying as simple to use as possible. *SIONlib* can use one or a few files to store the data from all processes. The main advantages are:

- Data aligned in chunks each with the size of the filesystem blocks *(Fig. 1)*:
  There is no need of locking because each process accesses its own block. As a result there is no more serialization on the access to the file providing a better scalability and performance.

- Meta information about the data structure in the file:
  Generic tools exist that can operate on any file created with *SIONlib*. These include splitting a big file in small pieces containing only the data from each task, rearranging and defragmenting the data in the file, information about the data stored, etc.

- Chunksizes should also be specified in advanced.
  However, *SIONlib* will automatically flush the file and assign more space if it is needed. This way the size of the data written by each task can increase dynamically.

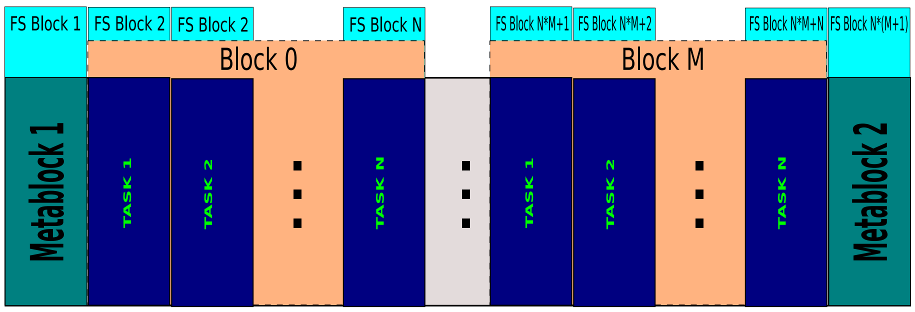


Figure 1: SIONlib File Format

Fortran API

High performance computing deals mainly with scientific simulations and a lot of the software is written in Fortran. Being written completely in C, there was no support for these applications. As a result, the library was extended and the Fortran API was included.
Due to the differences in the way Fortran and C handle files, all the I/O is done in pure C. This way, the created files are portable and they can be used from programs written in both languages. Wrapper functions are also provided for writing/reading binary data to/from the files: *fsion_fwrite()* and *fsion_fread()*.

Integration of SIONlib in PEPC

As a direct application of the this new feature, *SIONlib* was integrated in a development version of PEPC. The name is an abbreviation for Pretty Efficient Parallel Coulomb-solver and it is a parallel tree-code for rapid computation of long-range Coulomb forces in N-body particle systems. The public version of PEPC is divided into kernel routines and 'front-end' applications. A front-end called PEPC-B, code for simulating laser- or beam-plasma interactions, was selected and *SIONlib* was integrated in it. [2]

Initially, it was using one file per task and time step. All the files were seperated in folders according to the processes they belong to - each task having its own directory. This was done as a workaround for the common problem of folder's inode locking. However, this approach does not scale good because of the need to create thousands of folders and later on hundreds of files in each one of them. All the folders are created in the preprocessing phase and as a result it long delay till the simulation is up and running.
What is more, one can easily hit the maximum number of files limit. This will lead to an I/O error and the program behaviour will be unexpected.

In an attempt to overcome this problems and improve the scalability of PEPC, *SIONlib* was integrated in a development version. This extension provided the following advantages:

- Only one file per time step needed
  $\Rightarrow$ No more thousands of folders
  $\Rightarrow$ Faster startup & Easy handling of the simulation data

- Homogeneous execution after restart

Multiple Files Support

Another recently added feature of *SIONlib* is the support for working with multiple physical files using only one logical file. The main reasons for this extension are:

- Maximum filesize limit
  Different systems have different restrictions on the filesize. This may lead to problems, for example, with performance analysis tools that generate trace files with the size of more that 2TB. One way to overcome this limit is by using a few smaller files instead of a big one.
  Another example is the limit of the 32-Bit systems. The filesize is restricted to about 2GB when using a standard 32-Bit file pointer to index the number of bytes. However, this problem no longer exists on newer systems due to the extended support for large files (LFS).

- Different optimal strategies on different filesystems
  On one hand, the best performance some filesystems can be achieved using just a few big files. On the other hand, many small files can be the optimal strategy for others.

- Metacomputing

  - Local files for each computing resource
  - Less communication and data trasfer
    ⇒ Better performance and scalability

- Local files on not consistent global filesystems
  There might be a delay after creating a file on a global filesystem until it becomes visible to the rest of the compute nodes. This might lead to unexpected behaviour of the application and different I/O errors. One possible workaround for this problem is to use only files that are local to each compute node. This way, the file will be accessible directly after creation and the workflow will be consistent.

- IO Nodes oriented approach on BG/P
  Having one file per I/O node leads to faster performance on the Blue Gene/P platform.

Current State

The following chapter gives a short look at the current state of *SIONlib* and my work connected with this library during the 10 weeks guest gtudent program at the Jülich Supercomputing Centre.

- C and Fortran Interfaces:
  Initialy, only support for applications written in C existed, but the support was extended also for Fortran code.

- Multiple files can be read/written at a time: In the beginning, *SIONlib* was designed to use only one big file for all tasks. However, this proved to be inefficient in some situations and now the library can work with multiple files at once. There are two different ways of specifying the number files used:

  - Directly specifing the number of files:
    Exactly that many physical files will be created and they will be equally distributed among the processes.

  - Specifing a global and a local communicator:
    The number of physical files will be equal to the number of local communicators inside the global one. Thus, the processes in each local communicator will get a separate file. This approach can be used when the number of local communicators is not known in advance.

  This files are opened using only one function call. This will return a file pointer that will be specific to each process and will point to different physical files. As a result, there will be only one logical file and multiple physical ones *(Fig. 2)*.

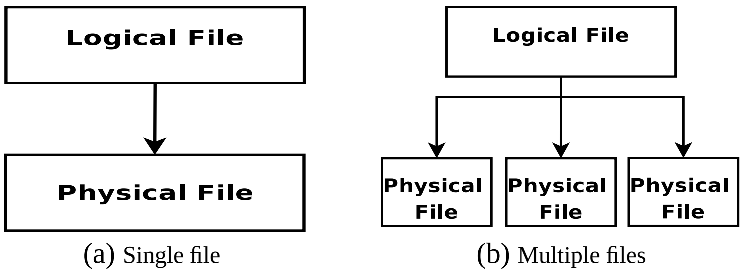


Figure 2: Multiple files support

- Doxygen documentation:
  The whole API and the internal structures are now documented with Doxygen. This provides a cleaner and more detailed information about the library. Help documentation can be generated in many different formats just with a simple command.

- Tested on GPFS, Lustre:
  Extensive tests of the library were completed on the General Parallel File System (GPFS). Basic functionality was also tested on a Lustre file system.

- Integration in a development versions of Scalasca and PEPC-B: As a direct application of the new Fortran API, *SIONlib* was integrated in PEPC-B. There exists a development version of Scalasca working with the this library. However, it is using an old version of *SIONlib* and an update is to be completed soon.

SIONlib Internals

SIONlib File Format - Overview

The file format used by *SIONlib* can be seen on *Fig. 3*. It contains the following three main parts:

- Metablock 1:
  This block contains static file information *(Fig. 4)* and it is written exactly after the file is opened.

- *M* blocks of chunks:
  One chunk is allocated for each process. It contains the data written by that task and has the size requested in the beginning. If a process tries to write more data than specified, a new block of chunks is created and the current file pointer is set to the correct position in the new block. This way, the size of the data can be dynamically increased and is not restricted by the requested value.

- Metablock 2
  This block stores dynamic information depending on the number of tasks and writes *(Fig. 5)*. It is written just before the file is closed.

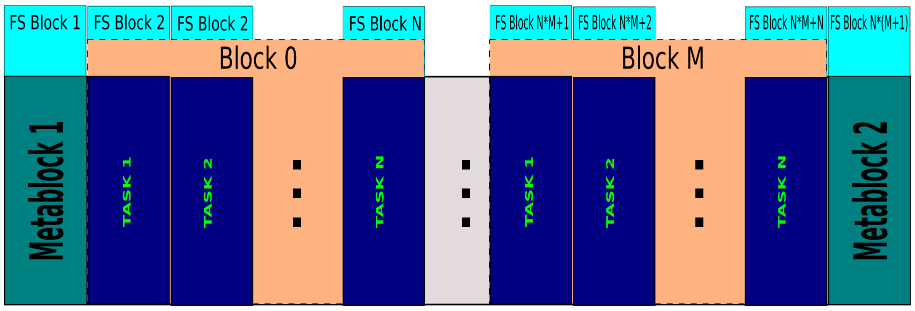
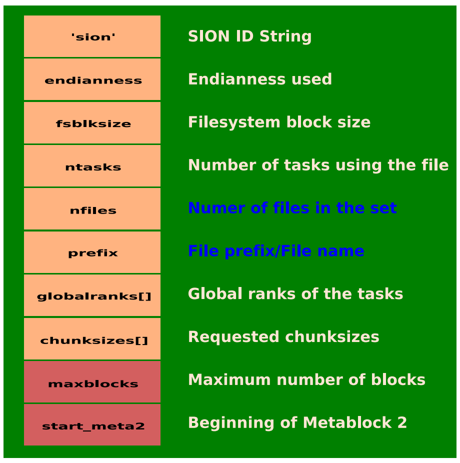


Figure 3: SIONlib File Format

SIONlib File Format - Metadata

As already mentioned, the internal information about the file and the data structure is saved in two blocks. Their structure can be seen on *Fig. 4* and *Fig. 5*.



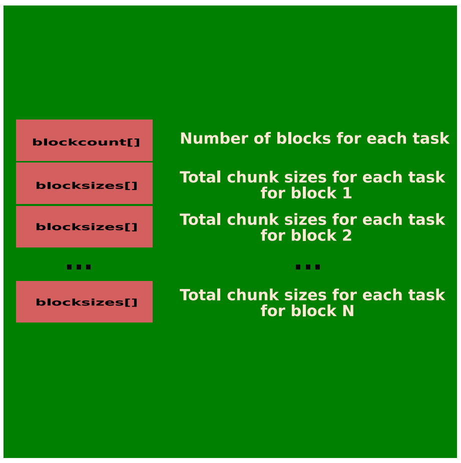

blockcount[]
Number of blocks for each task
blocksizes[]
Total chunk sizes for each task for block 1
blocksizes[]
Total chunk sizes for each task for block 2
...
...
blocksizes[]
Total chunk sizes for each task for block N

Figure 4: SION Metablock 1

Figure 5: SION Metablock 2

SIONlib File Format - chunksize dependence

When the file is created, the chunks of the file belonging to each process are aligned with the filesystem blocks. The resulting data structure depends on the ratio between the chunk size and the filesystem block size. The most common case are:

- When the requested chunk size if less than half of the filesystem block size, the whole block is still allocated and more than two chunks of data can be written to it. This is done in order to eliminate possible block locking by assigning the whole block to one task. As a result, there might be empty space left after the data.

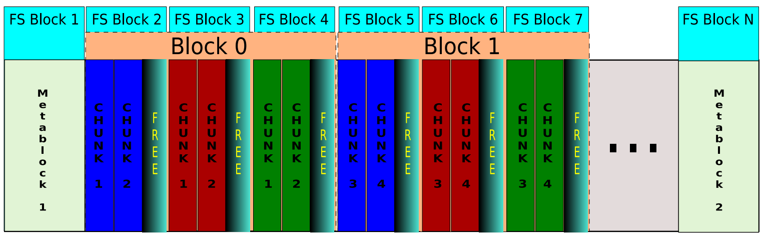


Figure 6: $\frac{Chunksize}{FSBlkSize} < \frac{0.5}{1}$

- Another situation is when two chunks of data perfectly fit into one filesystem block. There is no empty space after the data.

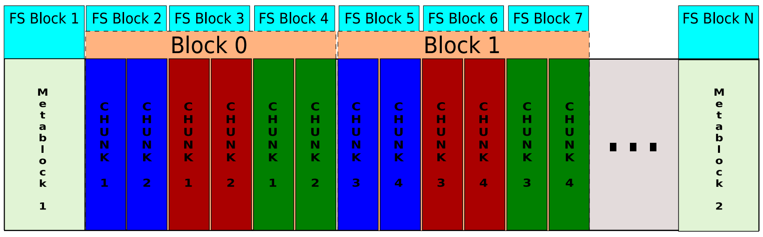


Figure 7: $\frac{Chunksize}{FSBlkSize} = \frac{0.5}{1}$

- Most often, the chunk fits exactly the filesystem block. This results in a nicely structured data in the file with no empty spaces.

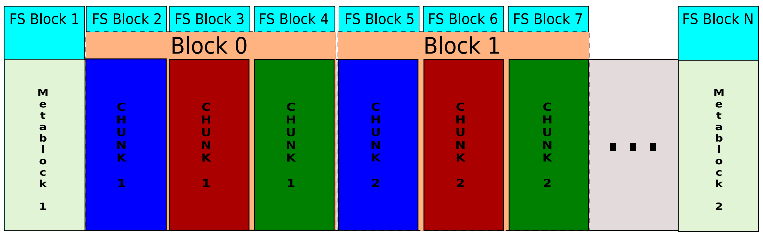


Figure 8: $\frac{Chunksize}{FSBlkSize} = \frac{1}{1}$

SIONlib Fortran API

Writing a Fortran API for a C library is not always a straight-forward procedure. There are some key points that need to taken into account:

- Arguments:
  Fortran passes arguments by *reference* so all C function arguments should be defined as pointers
- Strings:
  For each string argument Fortran passes an additional argument being the length of the string. This is represented in C by a *long int* variable which is passed by value. Thus, additional parameters should be included after the normal arguments in the C functions.

- Procedure name:
  In Fortran the name of a function after compilation depends on the compiler. There are different conventions:
  - All letters uppercase
  - All letters lowercase
  - Trailing underscore(_)
  - Trailing double underscore (__)

  This name should be the same in both C and Fortran object files for the successful linkage of the final executable. To overcome this problem, different macro definitions are used for each case.

- Sharing I/O between C and Fortran:
  Generally, writing a file in C and later on reading it in Fortran is not recommended. Even though Fortran uses internally file pointers just like C, access to them is not provided to the programmer. What is more, Fortran writes the data based on records and inserts special information in the file. A better and more portable strategy is to do all the I/O either with C or Fortran. As *SIONlib* is completely written in C, all the I/O in the Fortran API is also done with the C-I/O interface. Wrapper functions are also provided for working with binary files from Fortran.

  As a final result, *SIONlib* works internally with C file pointers and the files are accessed from Fortran using a special SION identifier (INTEGER)! This approach provides portability of the datafiles, data consistency and resembles the Fortran handles files.

Local and global communicators

As an alternative to the common way of opening a SION file where the number of files should be explicitly specified, there is a version working directly with communicators. Both a global and a local communicators should be specified and *SIONlib* will create one file for each local communicator. No additional information about the splitting of the processes is needed, thus providing easier code integration.

However, there is no MPI function that returns the number of subcommunicators of a communicator. So after the processes are divided in groups, there is no way get further information about the splitting. The following algorithm is used in *SIONlib* for this purpose:

1. Get the global and local ranks of the current process.
2. Use *MPI_Gather* to collect the local ranks to the global task 0.
3. gRank 0: Check the collected local ranks and for each 0 entry increment a counter.
4. gRank 0: Send the current counter value to the task having a local rank 0.
5. lRank 0: After receiving the value, that task broadcasts it in the local communicator.
6. A file suffix is formed using the received value.
7. The final value of the counter is the number of local communicators.

SIONlib Workflow - Create a new file for writing

When a file is opened in write mode, the workflow is as follows:

- sion_paropen_multi_mpi(. . .)
  1. Generate a suffix for the output file(s)
  2. Create a SIONLIB ID
  3. Allocate space for the internal stuctures
  4. Open the file(s)
  5. Write Metablock 1
  6. Set the task specific file position
- sion_ensure_free_space(. . .)
  1. Get the current position in the file
  2. Add to it the required data size(*nbytes*)
  3. Compare it with the size of the allocated chunk in the block
  4. If bigger ⇒ create a new block and flush the file
  5. Set the new file position accordingly
- fwrite(. . .)
- sion_parclose_mpi(. . .)

SIONlib Workflow - Open a file for reading

The following steps are performed when a file is opened for reading:

- sion_paropen_multi_mpi(. . .)
  1. Generate a suffix for the input file(s)
  2. Create a SIONLIB ID
  3. Allocate space for the internal data stuctures
  4. Open the file(s)
  5. Read Metablock 1
  6. Read Metablock 2
  7. Set the task specific file position
- sion_bytes_avail_in_block(. . .)
  1. Check if there the end of the current block is reached
  2. If yes ⇒ set the pointer to the next available block
- fread(. . .)
- sion_parclose_mpi(. . .)

Benchmarks

Benchmark Environment

The following benchmarks were performed on the Jugene system in the Jülich Supercomputing Centre. It is based on the Blue Gene / P platform with 16 racks in total. Specific for this system is the presence of pure compute nodes and others doing in I/O [5]. Every 128 compute nodes access one I/O node. As a result, there are 8 I/O nodes and 1024 compute nodes in 1 rack of the Jugene system.
All the compute nodes connect to the I/O nodes using internal network. The latter are connected with a 10 Gb optical network to the file-server. Thus, the theoretical bandwidth of the I/O operations is up to 10 GB per rack. In practice, however, it was measured to be 6 GB/s using a special I/O benchmark directly on the file server. This is mainly due to the current configuration of the fileserver in the Jülich Supercomputing Centre.

GPFS Benchmarks

SIONlib PARTEST on Jugene

As a small benchmark, a test program was run on 1024 compute nodes in VN mode (4 cpu cores per compute node) with 1 TB of data being first written to and then read from the filesystem. In the VN mode there is one MPI task running on each core. Table 1 shows the results of this test using *SIONlib*.

Files	Size	Open / s		Close / s		BW / MB/s	
		W	R	W	R	W	R
1	1024	3.24	0.17	13.65	0.43	1510	4093
2	1024	1.55	0.58	10.00	1.35	2022	3794
4	1024	0.30	0.33	3.19	0.16	2471	3693
8	1024	0.31	2.15	3.45	1.76	2543	4104
16	1024	1.65	1.79	1.60	6.16	2554	4076

Table 1: 1 Rack PARTEST Benchmark (4096 MPI Tasks)

Table 2 shows the results from a test with the same configuration but on 8 racks having in total 32768 cores.

F	Size	Open / s		Close / s		BW / MB/s	
		W	R	W	R	W	R
1	1024	7.25	0.16	0.15	0.06	3468	5792
2	1024	8.14	1.72	1.41	1.72	2825	5778
4	1024	6.90	2.40	0.57	0.79	2978	5632
8	1024	10.46	2.60	12.06	5.86	4796	5793
16	1024	8.70	0.32	6.85	8.15	5112	5685
64	1024	12.70	15.00	8.94	10.42	5598	5753

Table 2: 8 Racks PARTEST Benchmark (32768 MPI Tasks)

MPI-IO Test on Jugene

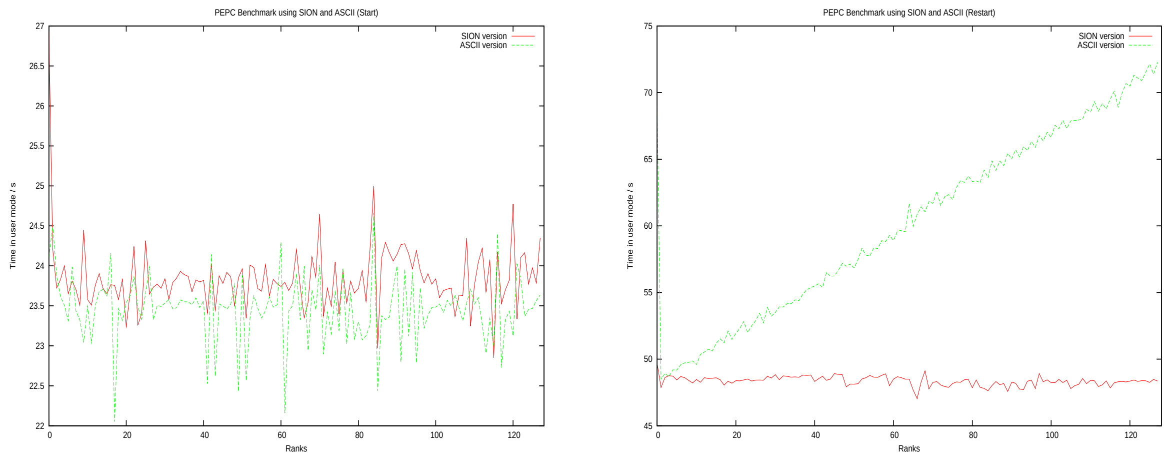
Table 3 shows the results of the same test using a MPI I/O without specifying implementation specific options.

Files	Size	Open / s		Close / s		BW / MB/s	
		W	R	W	R	W	R
1	1024	-	-	-	-	-	-
2	1024	-	-	-	-	-	-
4	1024	0.30	0.24	0.10	0.33	2095	2709
8	1024	46.10	0.30	0.29	0.09	2087	2790
16	1024	156.35	0.31	1.61	0.52	2551	2907

Table 3: 1 Rack MPI-IO Benchmark

PEPC-B IO Benchmarks

Two versions of PEPC-B were compared: the original one dumping the particles as ASCII and the new version using *SIONlib*. The benchmark was done on 128 processors using 3000000 ions. For the sake of the measurement, only two time steps were performed and then the simulation was restarted for 8 additional steps. As it can be seen from *Fig. 9*, the distribution of the user mode time after starting is homogeneous in both versions. However, there is an additional preprocessing phase in the normal PEPC-B where the whole output folder structure is created. As already discussed this approach brings many disadvantages and can take a long time to complete.
Moreover, after the simulation is restarted, the user mode time is not uniformly distributed with the normal ASCII version. This is not the case with the version that includes *SIONlib*.



(a) PEPC-B after Start

(b) PEPC-B after Restart

Figure 9: PEPC-B Benchmark on 128 CPUs

Outlook

Being in development phase, there are still different features to be included in *SIONlib*. The following list presents the most important ones:

- Different Endianess Support
- OpenMP Support
- Calculation of the number of files depending on the total global size and the maximal size per file
- Redundancy information for file reconstruction

Summary

SIONlib provides a simple interface for working in parallel with files. It can be easily integrated in existing software codes with just some minor changes. An example of this is the addition of *SIONlib* support in PEPC-B where only two files were changed and it was done using the new Fortran API.
Another feature of the library is the portability of the files created. All the files contain information about the structure of the stored data and they can be used by different programs written both in C and Fortran. Using this meta-information, one can create generic tools for working with the files. The following programs are already available:

- sionsplit - splits a big file in smaller pieces containing only the data for one process;
- siondefrag - orders the data in the file;
- siondump - prints the meta-information from a file;

Moreover, *SIONlib* is also independent from the underlying communication libraries. This way, the support for different parallel interfaces can be easily extended so that it can work with OpenMP, PThreads, etc.
SIONlib can work with multiple files allowing different optimization strategies and it can overcome limits such as maximum file size, number of files quota, etc.
Last but not least, the library provides fast file access - around 5.6 GB/s out of maximum 6 GB/s in the completed GPFS tests, and can improve the scalability of the applications.

References

1. SCALASCA - Scalable Performance Analysis of Large-Scale Applications
   http://www.scalasca.org
2. PEPC - Pretty Efficient Parallel Coulomb-solver
   http://www.fz-juelich.de/jsc/pepc/
3. SIONlib Presentation, Wolfgang Frings, JSC, 28.07.2008
4. MPI: A Message-Passing Interface Standard, Version 2.1
5. IBM System Blue Gene Solution: Blue Gene/P Application Development, ISBN 0738431648

Preconditioners for the Conjugate Gradient Algorithm

Lutz Roese-Koerner

Universität Bonn
Institut für Geodäsie und Geoinformation
Nussallee 17, 53115 Bonn

E-mail: lutz.roese-koerner@geod.uni-bonn.de

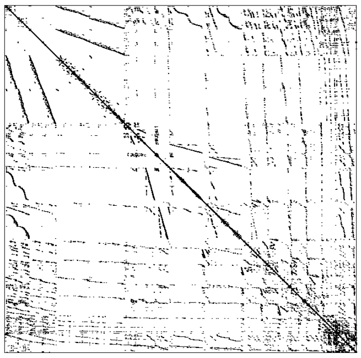
Abstract: In this report the Conjugate Gradient Algorithm is explained and the results of the SPARSKIT implementation of this method are shown. Some properties of Incomplete LU Factorization with Threshold are examined and its capability to be used as preconditioner for CG is analyzed. Finally, some attempts are made to detect a problem occurring when using ILUT and CG for certain matrices.

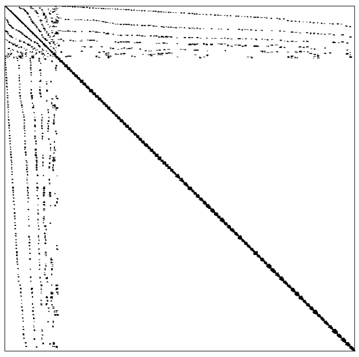
Introduction

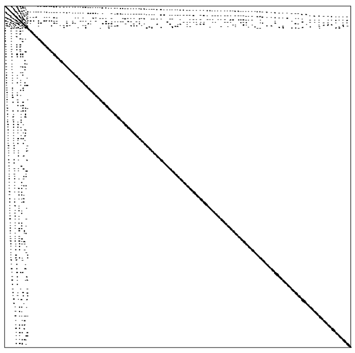
Preconditioners are used in many areas of numerical mathematics, e.g. whenever large sparse linear systems are to be solved for different right-hand-sides or the convergence rate of an iterative solver should be improved. Due to the fact that most preconditioning techniques only work for special tasks or need information about the input data, it is not easy to find a preconditioner that is applicable to most sparse problems. The Incomplete-LU-Factorization-with-Threshold-approach (ILUT) described in this report is an example for this kind of preconditioner. Therefore, many mathematical libraries contain an implementation of ILUT or a similar preconditioner.
ILUT is used at the Jülich Supercomputing Centre (JSC) in a Jacobi-Davidson eigenvalue solver, too. This software requires repeated solutions of a linear system (with different parameters and right-hand-sides) up to 1,000 times by a user supplied method. A combination of ILUT and the Generalized Minimal Residual method (GMRES) is used for this task. Since there are known problems with certain matrices using this eigenvalue software, the aim of this project is to examine the behavior of the ILUT preconditioner implemented in the library "SPARSKIT". Furthermore, the performance boost provided for the Conjugate Gradient Algorithm from the same library is to be analyzed.

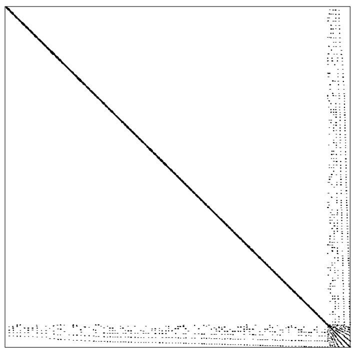
Data

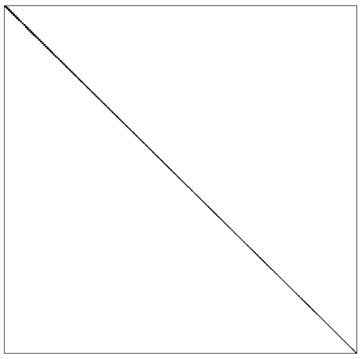
The following section describes the input data and points out some special characteristics. Table 1 shows dimension n and number of nonzero elements nnz of the input matrices. The left half of this table contains three matrices, which are known to cause problems in the eigenvalue software. Two of them (*kurbel* and *w124g*) appear twice, as different numbering schemes are tested (see figure 1). All five matrices originate from the finite elements context and occupy 350 to 650 MB disk space each. The five matrices on the right half of table 1 are normal equation matrices from a geodetic context. They are generated using spherical harmonic expansion up to degree and order 30 respectively 100 (implied by the first number in matrix name). All matrices of degree and order 100 are produced with the same set of observations. If the matrix name contains "oM" (one missing), one arbitrary observation is deleted before computing the normal equations. This destroys the block diagonal structure (see figure 1f). In order to create sparse matrices, every element below some threshold is dropped. As there were many elements


(a) Matrix: K0001


(b) Matrix: kurbel, numb. scheme: 1


(c) Matrix: w124g, numb. scheme: 1


(d) Matrix: w124g, numb. scheme: 2


(e) Matrix: N_100_5e6

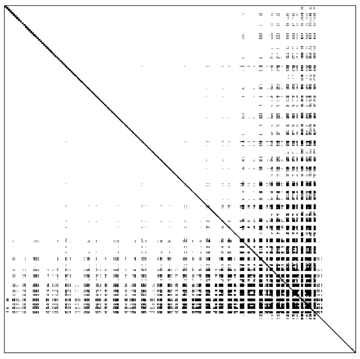

(f) Matrix: N_100_oM_5e6

Figure 1: Structure plots of some of the used input matrices.

with large values the threshold is set to $1e6$ respectively $5e6$. Although these five matrices have a smaller dimension, they are denser than the other matrices.
All matrices are block diagonal dominant, symmetric and positive definite.

Matrix	n	nnz	Matrix	n	nnz
K0001	235.962	12.860.848	N_30_5e6	957	6,032
kurbel_1	192,858	24,259,521	N_100_5e6	10,197	171,049
kurbel_2	192,858	24,259,521	N_100_1e6	10,197	192,331
w124g_1	401,595	20,825,882	N_100_oM_5e6	10,197	296,897
w124g_2	401,595	20,825,882	N_100_oM_1e6	10,197	8,177,599

Table 1: Input matrices with dimension d and number of non-zero elements nnz.

Conjugate Gradient Algorithm (CG)

Description of the algorithm

The Conjugate Gradient Algorithm (CG) is a commonly used iterative solver for linear systems of equations of type

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{1}$$

$\mathbf{A}$ is a symmetric and positive definite input matrix, $\mathbf{b}$ is a vector containing the elements of the right-hand-side of the linear system and $\mathbf{x}$ is the solution, which is to be computed. It is a Krylov Subspace

Method (like Arnoldi's Method or the Generalized Minimal Residual Method described e.g. in [3]). If $\mathbf{A}$ does not have the mentioned characteristics, CG can still work, but this is not guaranteed. Finding the solution is seen as a minimization problem of the quadratic form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x^T A x} - \mathbf{b^T x} + \mathbf{c}. \tag{2}$$

The name Conjugate Gradient is due to the $\mathbf{A}$-orthogonality (conjugacy) of each search direction to all the others. Two vectors $\mathbf{a}$ and $\mathbf{b}$ are $\mathbf{A}$-orthogonal if

$$\mathbf{a^T A b} = 0. \tag{3}$$

Therefore, one only needs to go once in each direction. While CG is not often used as a direct solver (due to roundoff errors), it is more frequently applied to problems as an iterative one, since it approximates the solution well after very few steps.
Using CG for solving a linear system consists of iteratively computing the following terms (see [4]): First, a starting point $\mathbf{x_0}$ is to be determined (e.g. $\mathbf{x_0} = \mathbf{0}$). The residual $\mathbf{r}$ is chosen to be the initial search direction $\mathbf{d_0}$

$$\mathbf{d_0} = \mathbf{r_0} = \mathbf{b} - \mathbf{A x_0}. \tag{4}$$

The step-length α is the distance covered at each step (computed by line search)

$$\alpha_i = \frac{\mathbf{r_i^T r_i}}{\mathbf{d_i^T A d_i}}. \tag{5}$$

The residual $\mathbf{r_{i+1}}$ and the solution $\mathbf{x_{i+1}}$ of the next iteration step can be computed utilizing this information

$$\mathbf{x_{i+1}} = \mathbf{x_i} + \alpha \mathbf{d_i}, \tag{6}$$

$$\mathbf{r_{i+1}} = \mathbf{r_i} - \alpha \mathbf{A d_i}. \tag{7}$$

Afterwards, the Gram-Schmidt-Conjugation is used, computing a factor β for updating the next search direction $\mathbf{d}$

$$\beta_{i+1} = \frac{\mathbf{r_{i+1}^T r_{i+1}}}{\mathbf{r_i^T r_i}}, \tag{8}$$

$$\mathbf{d_{i+1}} = \mathbf{r_{i+1}} + \beta_{\mathbf{i+1}} \mathbf{d_i}. \tag{9}$$

The steps mentioned above are repeated until a maximum number of iterations or a given convergence criterion is reached. As the residual $\mathbf{r}$ is computed in each iteration, the decrease of its norm is often used as convergence criterion.

Methods

Since CG is a well-known iterative solver, it is implemented in many packages of mathematical software. In this project a CG routine from the Fortran77 library "SPARSKIT" by Yousef Saad is used, which is described in detail below (see also [2]). Some additional Fortran90- and C-subprograms were written, whenever needed functions were not provided in the library. It was tried to modify the "SPARSKIT"-library itself as few as possible (most modifications done are only for debugging or the output of some interim values). As the intention was to analyze the efficiency of preconditioners for CG, the first step was to use this iterative solver without a preconditioner in order to have a comparison. Most

routines of “SPARSKIT” (including the preconditioners) work with matrices in CSR-format (see section “SPARSKIT” for details on the different formats). Since the CG module of “SPARSKIT” is designed to be used with a preconditioner, it needs the $\mathbf{A}$-matrix as input in the CSR-format as well as a preconditioner-matrix $\mathbf{M}$ in **MSR-format**.
Hence, a C-function was written, which designs an identity matrix of size n in MSR-format. This subroutine can be called instead of a function provided by “SPARSKIT” in order to create the needed input data for the CG module. Using an identity matrix for preconditioning will lead to the same results as doing no preconditioning (but with additional computation time due to some matrix vector products). Having only the input matrix given, a right-hand-side-vector $\mathbf{b}$ is to be created, for which the system is to be solved. This is done by computing $\mathbf{b} = \mathbf{Ax}$, with $\mathbf{x}$ containing only ones (different values were tried without influencing the behavior). Except for some matrices, for which the use of a special initial value is mentioned, the initial guess was chosen to be a zero vector of length n.

Results

The following results were computed solving the linear system with the CG-algorithm from “SPARSKIT“ on the cluster “JULI” (Jülich Linux Cluster) with the parameters mentioned above. As the library only provides sequential programming, all calculations were done on a single processor. First, the system was solved for the three input matrices *K0001*, *kurbel* and *w124g*, which originate from FEM context (see introduction). While *kurbel* and *w124g* have a similar structure and size, *K0001* does not (see figure 1), which may explain its different behavior shown in figure 2. In this figure, the (slow) decrease of the residual is plotted versus the number of iterations (logarithmic scale). Having computed more than

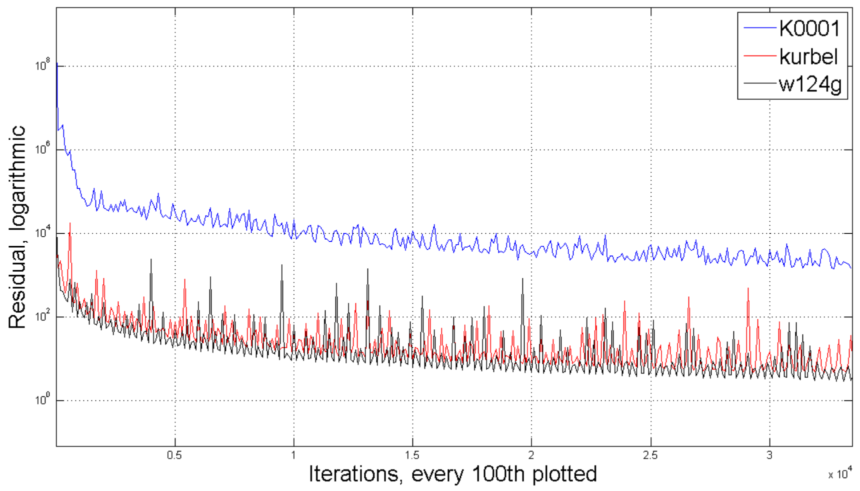


Figure 2: Convergence of CG for three different input matrices (residual norm vs. iterations).

30,000 iterations, the criterion of convergence

$$||\mathbf{r}|| = t_{rel} * ||\mathbf{b}|| + t_{abs} \tag{10}$$

was satisfied for *K0001* with the relative tolerance $t_{rel} = 10^{-10}$ and the absolute tolerance $t_{abs} = 10^{-6}$. Solving the system for the other two matrices was aborted after 72 hours without satisfying the above criterion. In order to maintain comparability, same tolerances were used for all input data. No “weaker” tolerances were chosen, as this would have led to an earlier convergence for matrix *K0001*, whose final residual norm is already high for this choice of the tolerance parameters.
The effect of resorting elements in input matrices is shown in figures 3 and 4. While the overall behav-

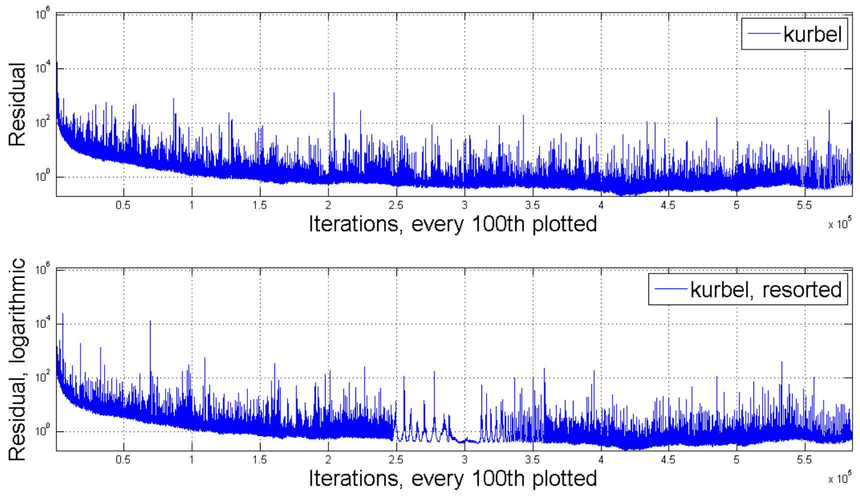


Figure 3: Convergence of CG for input matrix *kurbel* with different numbering schemes.

ior of the residual norm of the two *kurbel* matrices is similar (except one part in the centre), there is a significant “jump“ in the resorted version of *w124g*. Both phenomena are probably effects of roundoff errors that appear at different parts of the computation. This “jump” can be interpreted as the change of the CG algorithm into another search-subspace.

The same issue is plotted for some matrices from geodetic context in figure 5. To guarantee comparability the right-hand-sides of these systems of normal equation matrices are computed the same way as above. All the matrices generated using spherical harmonic expansion up to degree and order 100 behave similarly, unaffected by potential deletions of observations. Solely the matrix containing coefficients up to degree and order 30 shows unexpected behavior: After a first decrease of the residual norm, there is a peak at the 116th iteration, followed by another phase of decrease until convergence is reached. This might be due to the fact that the residual norm $||\mathbf{r}||$ is plotted and not $||\mathbf{A}^{\frac{1}{2}}\mathbf{r}||$ which has a monotone convergence behavior.

One problem yet unsolved is, that for four of the matrices (degree/order: 100) CG only converges if the initial guess is close to the actual solution. Otherwise, it aborts with an error message while trying to compute $\mathbf{d}^{\mathbf{T}}\mathbf{A}\mathbf{d}$, because the values exceed the range provided by `double`. Using `long double` instead of `double` enlarges the range in which the initial guess has to be. But this approach also fails for larger differences to the actual solution. This may be caused by an inappropriate distribution of the huge values in $\mathbf{A}$. As preconditioning will probably change this, no further attempts (e.g. normalizing the system) were made to avoid this.

Preconditioner

Since the convergence rate of iterative solvers is strongly influenced by the condition of the input matrix $\mathbf{A}$, many attempts were made to improve the condition number without distorting the solution. In particular, if one system is to be solved for different right-hand-sides, preconditioning could save a lot of computation time. The common approach is to solve the linear system

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{11}$$

instead of the original problem $\mathbf{A}\mathbf{x} = \mathbf{b}$. $\mathbf{M}$ is the so-called preconditioner matrix. As the identity matrix has the best condition number possible, $\mathbf{M} = \mathbf{A}$ would be a perfect preconditioner. However, as computing the inverse of $\mathbf{A}$ would be as expensive as solving the original problem, several attempts were made

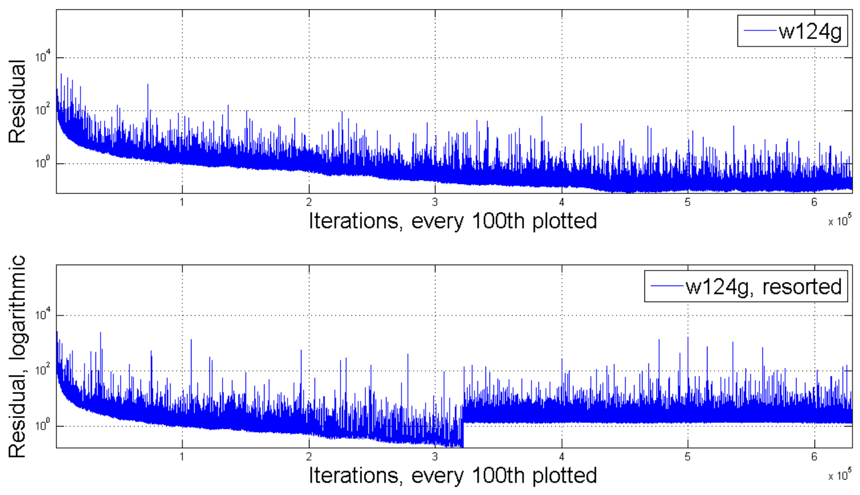


Figure 4: Convergence of CG for input matrix *w124g* with different numbering schemes.

to choose a different preconditioner matrix $\mathbf{M}$. Beside approaches such as the Jacobi preconditioner, the Gauss Seidel preconditioner or successive overrelaxation (all described in [3]), there are some "classic" factorization techniques, which can be modified and used as preconditioners: Incomplete Cholesky and incomplete LU factorization. This report focuses on incomplete LU factorization.

Incomplete LU Factorization (ILU)

The basic principle of the Incomplete LU Factorization (ILU) is the same used for ordinary LU Factorization: The matrix $\mathbf{A}$ is decomposed in a lower triangular matrix $\mathbf{L}$ with ones on the main diagonal and an upper triangular matrix $\mathbf{U}$

$$\mathbf{A} = \mathbf{LU}. \tag{12}$$

Then these two matrices are stored together, neglecting the known diagonal entries of $\mathbf{L}$.

$$\mathbf{M} = \mathbf{LU} \tag{13}$$

is used as preconditioner. One big drawback is that the LU Factorization of a sparse matrix does not have to be sparse itself. Therefore, instead of computing an entire LU Factorization, $\mathbf{A}$ is decomposed into

$$\mathbf{A} = \mathbf{LU} - \mathbf{R}. \tag{14}$$

The residual matrix $\mathbf{R}$ is used to drop specified elements of $\mathbf{A}$ and is often realized in practice by a non-zero pattern P, which contains the indices of allowed non-zero elements. Doing an ILU consists of two major steps (same as in the Gaussian Elimination), which are constantly repeated **but only for elements of the non-zero pattern P**. The first step is to divide each element of the line k by the diagonal entry a_{kk} and to use this as a factor for the elimination process in the following rows

$$a_{ik} := a_{ik}/a_{kk}, \qquad \text{if } (i,k) \in P, \tag{15}$$

$$a_{ij} := a_{ij} - a_{ik}a_{kj}, \quad \text{if } (i,j) \in P. \tag{16}$$

It can be shown, that ILU retains most matrix characteristics (depending on the chosen version of ILU, see below). If $\mathbf{A}$ is a Minkowski-matrix (M-matrix), all properties will be preserved. M-matrices arise e.g. from discretisations of elliptic Partial Differential Equations (PDEs) - e.g. the Laplace-matrix - and have following characteristics [1]:

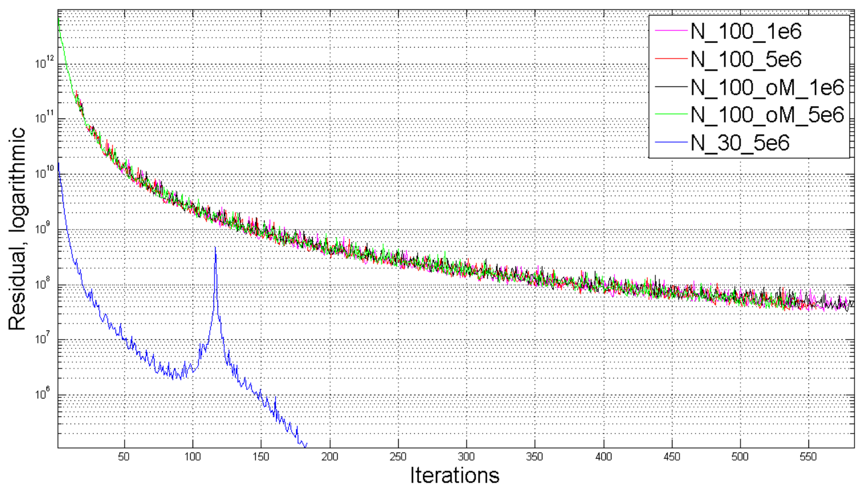


Figure 5: Convergence of CG for input matrices from satellite geodesy, the first number determines degree and order of the spherical harmonics, the second the threshold used for dropping elements."oM" signals that an arbitrary observation was deleted, before computing normal equations.

1. $a_{i,i} > 0$ for $i = 1, ..., n$,
2. $a_{i,j} \leq 0$ for $i \neq j, \qquad j = 1, ..., n$,
3. $\mathbf{A}$ is nonsingular,
4. $\mathbf{A^{-1}} \geq \mathbf{0}$.

An M-matrix is defined by the properties 2 and 4. If these are satisfied, properties 1 and 3 are as well. The last property is equal to the request for weak diagonal dominance.
Different versions of ILU could be distinguished by their way of choosing the non-zero pattern P (and sometimes additional computations):
The "basic" version is called "Zero Fill-in ILU" **(ILU(0))**. Here, P is chosen to be exactly the non-zero pattern of $\mathbf{A}$. As a result, there are only slightly more elements in $\mathbf{A}$ if the product $\mathbf{LU}$ is computed explicitly, than there were in the "original" $\mathbf{A}$.
The next possible step is called "ILU with level of fill" **(ILU(p))** and allows additional entries in $\mathbf{L}$ and $\mathbf{U}$, depending on the level of fill p.
The "Modified ILU" **(MILU)** uses another approach: The sum of all elements dropped of one row is added to the diagonal entry.
Not the location but the magnitude of the elements in $\mathbf{A}$ is important for the decision of dropping an element or not, when "ILU with threshold" **(ILUT)** is used. Sometimes this approach is referred to as "ILUT with Pivoting" (**ILUTP**) when the reordering of matrix elements is also implemented.
Several other versions of ILUs were developed and used, such as **ILUS** or **ILUC**. For more details on the above mentioned methods, see [3]. One big advantage all these ILU-versions have in common is, that no information about the structure of the input matrix is required (preconditioners with that property are rare). In the following, this report will focus on ILUT and its use as preconditioner for the CG algorithm.

ILU with Threshold (ILUT)

One big drawback of ILU(0)/ILU(p) is their blindness to numerical values. This can lead to keeping "uninteresting" close to zero elements in $\mathbf{L}$ or $\mathbf{U}$ while dropping far bigger elements. ILUT tries to avoid

this by dynamically choosing its pattern P with a double threshold strategy: Every element which is below some threshold τ is dropped. Next, only the p largest elements in each row of $\mathbf{L}$ and $\mathbf{U}$ are kept. While τ is used to reduce computation cost, p helps to control memory usage. p is also called level of fill, which is somewhat misleading, because it has a different meaning from the level of fill of ILU(p). ILUT can be computed, using equations 15 and 16 and replacing the pattern P by the two dropping rules depending on τ and p.
It is important that only off-diagonal entries are dropped, as otherwise $\mathbf{M^{-1}A}$ does not necessarily have the same characteristics as $\mathbf{A}$. As the non-zero pattern P is chosen dynamically, **ILUT will destroy symmetry**, which will cause problems if combining it with CG.

SPARSKIT

“SPARSKIT” is an open-source Fortran77-library by Yousef Saad [2], which provides four different ILU preconditioners (including ILUT) and nine iterative solvers (CG, GMRES, etc.). Besides solving linear systems, one aim of this - since 2005 in version 2 available - software package is to enable easy data exchange or conversion between different working groups or institutes. Designed for sparse matrix computation, it internally uses the Compressed Sparse Row format (CSR) for storing matrices.
All computations done for this project are computed using “SPARSKIT”-routines whenever possible. That is not only true for the pure solving-step of the equation system, but also for the reading in of data in different formats, converting these formats, generating test matrices, plotting the results and using “SPARSKIT”-provided “BLAS”-routines.

The test-programme

As there are existing problems with the ILUT from “SPARSKIT” for the shown input matrices, a test-programme was written in order to detect the root of the problem and to determine the performance boost provided by the preconditioner.
The main programme is written in C, calling the “SPARSKIT” Fortran77- and some additional Fortran90- and C-functions. First, the header of the input matrix is read and memory is dynamically allocated. “SPARSKIT” uses three working arrays for computation and control: One integer array, which parameters have to be set (like the kind of convergence criterion, etc.), a double array, which contains some parameters (like tolerances) and the double array $\mathbf{w}$, in which (interim) data can be stored. The parameters are set as mentioned above and the right-hand-side is computed in the described way. Now the matrix entries are read by a “SPARSKIT” provided function, stored in the memory allocated and the Fortran90 routine “testcg” is called, which was developed by adjusting and extending a “SPARSKIT” intern test-programme. Depending on whether a preconditioner should be used or not, either the routine for generating an identity matrix or for the ILUT preconditioner is called. In both cases, the time spent in the subroutine is measured. The last step is to call the CG-routine, measure the time it takes for each iteration and output the results (first elements of the solution and norm of solution).
These steps work well for most test matrices (generated with or without use of “SPARSKIT” and in different sizes), but fail for the matrices described in the introduction. Depending on the level of fill p, different errors occur: If the level of fill is greater than one, the CG-routine aborts with the error message: “while trying to detect a break-down, an abnormal number is detected”. If p equals one, no error message occurs, but CG diverges and if the level of fill equals zero, CG diverges for *K0001* and seems neither to diverge nor to converge for the other matrices. In each of these cases, the error only appears, if ILUT is used for preconditioning, but the ILUT error-flag signals always: “successful return”.

Debugging

Some additional challenges arise when debugging a programme that uses input data of several hundred megabytes. For example, it might no longer be possible to have $\mathbf{A}$ and $\mathbf{M}$ in the memory of the workstation (1 GB) at the same time. Allocation should be done dynamically as static allocation would lead to problems with the stack limit of JUMP (Jülich Multi Processor), which is used for debugging, as there are no debugging tools available on JULI. It might also cause problems to check some matrix properties during runtime (like computing a determinant) as this will take a lot more computation time and use additional memory.
Detecting the origin of the problem, two major directions are to be examined. The first is to determine whether the ILUT generated preconditioner matrix does not assure that $\mathbf{M^{-1}A}$ has the same properties as $\mathbf{A}$. The second possibility is a general malfunction of ILUT under certain conditions.
As "SPARSKIT" does not provide a symmetric version of ILUT, it is assumed that preconditioning will destroy symmetry. This could lead to problems with the CG algorithm, but since they also occur for a level of fill of zero, which means that $\mathbf{M}$ is not only symmetric but also a diagonal-matrix in this case, this can not be the reason for the malfunction. Another possibility is, that if $\mathbf{A}$ is only barely positive definite some numerical problems might appear while preconditioning. In order to avoid this, small values were added to the diagonal entries of the input matrix **before** preconditioning. This distorts the result, but may lead to better diagonal dominance. Unfortunately, it did not prevent CG from "detecting an abnormal number". The next approach was to add small values to the diagonal of $\mathbf{M}$ **after** preconditioning. This would not get the problem by its root, but may help to fix it and get a better understanding of what goes wrong. Unfortunately, the abnormal numbers remain and CG detects "$Not\ a\ Number$"s (NaNs) on the main diagonal of the preconditioner matrix. Since adding an arbitrary value to NaN will be NaN again (at least in Fortran), this ansatz also failed. The next approach made to avoid loss of necessary properties, was the conversion of the main-diagonal-NaNs into (small) positive values. As above, this should also be seen as an effort to fix the problem in order to detect its origin, not to compute accurate results. But as before, that does not lead to any changes in behavior. Due to abnormal numbers in off-diagonal entries CG aborts again. No efforts were made to convert all these NaNs, as this would have only led to more distortion of the results and did not seem to be very promising in order to find the root of the problem.
Since the approaches to obtain the necessary properties by modifying $\mathbf{A}$ or $\mathbf{M}$ failed, the position in the source-code where abnormal numbers appear first is to be localized: The errors seem to occur only while dropping elements of the upper Matrix $\mathbf{U}$. The dropping procedure is realized by computing

$$a_{ij} = a_{ij} - r, \tag{17}$$

with $r = 0$ if a_{ij} is a fill-in element and $r = a_{ij}$ else. The element r, which should be subtracted, contains sometimes NaN. This variable is not only used for dropping purposes but also as interim variable for swapping elements in the working array w, which is used for storing elements of $\mathbf{A}$, $\mathbf{x}$ and $\mathbf{b}$. In order to detect the problem more precisely, this variable was split in two (s and $s\_$). Due to the fact that NaNs still occur in both variables, it is assumed that there are abnormal numbers in w already (because the swapping variable s as well as the dropping variable $s\_$ obtain their values from this working array).
The first appearance of the error depends on the chosen level of fill. The higher p is set, the earlier the error occurs (not strict for small changes in p).

Conclusions and outlook

The routines ILUT and CG from "SPARSKIT" were tested with several input data and different parameters. While CG works well, problems appear for certain matrices if ILUT is used for preconditioning. As the approaches to modify the input matrix $\mathbf{A}$ and the preconditioner matrix $\mathbf{M}$ in order to prevent CG from aborting failed, it was attempted to detect the root of the problem. Due to the restricted time of the guest-student's programme misbehavior of ILUT was partly localized, but its root is still indistinct. Further possible steps in order to detect the root were not pursued due to the lack of time. For example, one could try to catch the error one step earlier by detecting the NaN-producing computation, with

$$\frac{0}{0}, \qquad \frac{\infty}{\infty}, \qquad \sqrt{-x} \quad \text{or} \quad 0 \cdot \infty.$$

as possible candidates. Using the test matrices as input for other programmes providing ILU / ILUT might also help to gain a better insight into the structure of the problem (libraries like "ILUPACK", "MUMPS" or "hsa" could be used for this purpose). Successful preconditioning with the ILUT-routines of these packages would be another indication for a "SPARSKIT" intern problem.
While "SPARSKIT" is a well-working tool in general, one has to be aware that there could be difficulties occasionally with particular routines that only appear under certain conditions. Especially the examined ILUT-routine seems to be vulnerable to instabilities. So, if other ILUT implementations would also fail for the test matrices used, it might be better to use more stable alternatives such as ILU(p). This would also take in account that "SPARSKIT" does not provide a symmetric version of ILUT, which could lead to further problems with the combination of ILUT and CG.

Acknowledgment

I would like to thank my advisor Dr. Bernhard Steffen for the supervision of my project and Matthias Bolten and Robert Speck for the perfect organization of the guest students' programme. I also want to thank Prof. Wolf-Dieter Schuh from the University of Bonn for recommending me for this programme and the other guest students for a very pleasant working atmosphere.

References

1. K. Fan. Note on M-matrices. *Quaterly Journal of Mathematics, Oxford series*, 11:43–49, 1960.
2. Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations. Technical Report RIACS-90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, 1990.
3. Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelpha, PA, 2. edition, 2003.
4. J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

Analysis of the Influence of the Dual Floating Point Unit "Double Hummer" on the Performance of Specific Library Routines on Blue Gene/P

Matthias Voigt

Technische Universität Chemnitz
Fakultät für Mathematik
Reichenhainer Straße 41
09126 Chemnitz
Germany

E-mail: mattv@hrz.tu-chemnitz.de

Abstract: The Blue Gene/P system of IBM is the newest supercomputer at the "Forschungszentrum Jülich GmbH" in Germany. It is used for applications in natural sciences that are expensive in terms of computational effort. One of Blue Gene's specialties is the dual floating point unit "Double Hummer" which is able to perform two floating point operations in parallel. It enables the user to decrease the runtime of applications significantly. During the usage of the dual FPU various problems appeared. The goal of this work is to show what the user should do and what he should avoid to get an optimal performance while using "Double Hummer". We also show how the performance behaves if we modify the code of calling routines or if we use special compiler options. In this way this paper should be a little help for users to get full benefits of the IBM Blue Gene/P architecture.

Introduction

The IBM Blue Gene/P Architecture

The Blue Gene/P system at the "Forschungszentrum Jülich GmbH" (JUGENE) has 16384 nodes or chip cards, each consisting of 4 IBM Power PC 450 processors with 850 Mhz clock speed. One core is relatively slow but this allows a high power efficiency, a low cooling effort and a very high density of processors in the machine [1]. 32 nodes form one node card and 32 node cards are combined to one rack. The complete JUGENE system consists of 16 racks (= $16 \times 32 \times 32 \times 4$ = 65536 cores). Every node has a theoretical peak performance of 13.6 GFlops/s and so the whole machine reaches a peak performance of 222,8 GFlops/s. The LINPACK benchmark provides a performance of approximately 180 GFlops/s (80% of peak) [2]. Each core has a 32 KB private L1 cache and and a 2 MB private L2 cache which is used as prefetch buffer. There is also an 8 MB shared L3 cache on each node. Additionally every chip card has an amount of 2 GB physical memory with a bandwidth of 13.6 GB/s. 150 I/O nodes manage the data flow between the Blue Gene system and an external file system (JUST) by an external 10 GB/s Ethernet. Several service and login nodes deal e.g. with user logins and job submits. Furthermore the Blue Gene/P system offers five highly efficient application oriented network topologies e.g. the 3-d torus, the global tree or the collective network. The 3-d torus is a 3-d mesh on which nodes on opposite sides are also directly connected. This structure provides a high bandwidth of 5.1 GB/s per node and low worst case latency of 3.2 μs [3].

The Dual Floating Point Unit "Double Hummer"

In addition to the the regular PowerPC floating point instructions (operating on the primary registers), new parallel floating point instructions have been added to operate simultaneously on both the primary and secondary registers. Some of the new dual FPU instructions perform identical operations on each set of registers in parallel. Other instructions allow operands to be copied from one register set to the other, or perform complex cross operations optimized for complex arithmetic. A set of load/store instructions has also been added to perform loads and stores to both sets of FP registers with a single instruction.
Since the PPC450 chip can issue at most one load/store and one FPU operation per cycle, the parallel instructions have the potential to double the floating point performance of the chip. The IBM Mathematical Acceleration Subsystem (MASS) library (and the vector MASSV library), and the IBM Engineering and Scientific Software Library (ESSL) take advantage of the parallel instructions to fully utilize the dual FPU. Hand written code using the parallel instructions can easily access this performance increase. New builtin functions have been added to the IBM XL C and C++ compilers to generate the parallel instructions. Intrinsic functions have been added to the IBM XL Fortran compiler.
The IBM XL compilers will automatically generate parallel FPU instructions, but doubling the floating point performance benefit is not usually achieved for arbitrary floating point code [4].

Introducing Examples

Influence of the Compiler Options on the Data Alignment

Choosing well aligned data is a crucial issue when using Blue Gene/P. Bad aligned data can cause an extensive loss of performance. Double precision data should be 16B-aligned which means that the data should be saved within the 16B borders of the RAM. If a floating point number is not well aligned an alignment exception will be thrown during the runtime. The environment variable `BG_MAXALIGNEXP` can be set to control the number of allowed alignment exceptions before the execution of the program aborts. The default value on JUGENE is `BG_MAXALIGNEXP` = -1. In this case alignment exceptions will not force the program to terminate. One may set `BG_MAXALIGNEXP` = 0 to let the program terminate after the first alignment exception. Thus this can be used to check if the program accesses bad aligned data. To change the value of this environment variable one just has to add an additional option while calling `mpirun` or `llrun`.

```
export BG_MAXALIGNEXP=[value]
[mpirun/llrun] [options] -exp_env BG_MAXALIGNEXP [exec_file]
```

Compiler options have a strong influence on the efficency of the code. High optimization levels are able to detect bad aligned data structures as one will see in the following code example.
Consider a Fortran program that uses two common blocks.

```
      INTEGER             MAXLEN
      PARAMETER           ( MAXLEN = 500000 )
      INTEGER             A, B
      DOUBLE PRECISION    X1( MAXLEN ), X2( MAXLEN ), Y1( MAXLEN ),
     $                    Y2( MAXLEN )
      COMMON              /COMBLK1/ X1, X2, A, /COMBLK2/ Y1, B, Y2
```

The first common block consists of two arrays `X1` and `X2` that each contain 500000 double precision floating point numbers. `X2` directly follows `X1` and so both are contiguously located in the memory at runtime as well. The second common block is in principle the same but the sequence of 8B floating point numbers is interrupted by one integer. Hence `COMBLK2` is a bad aligned data structure because the 16B pattern required by Blue Gene is destroyed by one integer that shifts all entries of `Y2` by 4B in memory. Consider now additionally two subroutines `SUM1` and `SUM2` that perform each a vector addition of `X1` with `X2` and `Y1` with `Y2` respectively using the appropriate common blocks defined in the main program.

```
      SUBROUTINE SUM1( X )
      IMPLICIT NONE
      INTEGER              MAXLEN
      PARAMETER            ( MAXLEN = 500000 )
      INTEGER              A, I
      DOUBLE PRECISION     X( MAXLEN ), X1( MAXLEN ), X2( MAXLEN )
      COMMON               /COMBLK1/ X1, X2, A
      DO 10 I = 1, MAXLEN
         X( I ) = X1( I ) + X2( I )
 10   CONTINUE
      RETURN
      END SUBROUTINE SUM1
```

The code for `SUM2` is similar but it includes the common block `COMBLK2` instead of `COMBLK1`. For compiling the source code the IBM XLF Fortran compiler is used as well as different compiler options to see how these affect the effiency of the code. In the following time measurement we always include the options `-qarch=450 -qtune=450` for compiling without optimization for the dual FPU or `-qarch=450d -qtune=450` for compiling with considering the dual FPU. Table 1 shows how alignment exceptions slow down and how further optimizations from `-O0` (no optimization) up to `-O5` (maximum optimization) speeds up the execution of the program.

Options	SUM1		SUM2	
	-qarch=450	-qarch=450d	-qarch=450	-qarch=450d
-O0	0.019751s	0.019751s	0.883256s	0.847652s
-O1	0.019750s	0.019751s	0.883260s	0.847640s
-O2	0.007943s	0.007943s	0.833583s	0.833555s
-O3	0.005296s	0.004120s	0.833665s	0.834572s
-O3 -qhot	0.005296s	0.003239s	0.864001s	0.833932s
-O4	0.005296s	0.003237s	0.005297s	0.833926s
-O5	0.005296s	0.003237s	0.005297s	0.003250s

Table 1: Execution time of subroutines `SUM1` and `SUM2` with different compiler options

In `SUM1` the performance improves for `-O2` or higher optimization levels. If one uses `-O3` or higher the impact of the dual FPU is detectable. Starting from `-O3 -qhot` "Double Hummer" will reduce the runtime by approximately 40%. The execution of this subroutine takes orders of magnitude more time than that of `SUM1` at lower optimization levels. But it is also observable that the dual FPU improves the performance by a few percent at `-O0` and `-O1`. For a very high optimization level the compiler recognizes the misalignment, corrects it and the runtime is comparable to `SUM1`. With `-qarch=450` this already works for `-O4`, with `-qarch=450d -O5` is needed.

Influence of the Array Declaration on the Performance

Investigation of a C program

In this subsection it is checked how the declaration of arrays in a program affects its performance. Consider a simple handwritten C program that performs the real matrix-matrix multiplication $C := \alpha AB+C$ which is also provided by the LAPACK routine DGEMM.
The static definition of the used arrays looks like the following.

```
#define nmax 1000;
double matA[nmax][nmax];
```

The arrays `matB` and `matC` are defined in the same way. Now the simplest approach to perform the operation is used.

```
for (i=0;i<nmax;i++) {
    for (j=0;j<nmax;j++) {
        for (k=0;k<nmax;k++) {
            matC[i][j]=matC[i][j]+alpha*matA[i][k]*matB[k][j];
        }
    }
}
```

The order of the three loops as well as the parity of `nmax` were changed to look if there occur differences in terms of runtime. The IBM XLC compiler `mpixlc_r` was used with the following flags.

```
CCFLAGS = -O[lvl] -qarch=[val] -qtune=450 [more_opts]
```

Comp.	-O0 -qarch=		-O3 -qarch=		-O5 -qarch=	
	450	450d	450	450d	450	450d
ijk-loop	111.0806s	111.0845s	5.4821s	5.1861s	5.4791s	3.7432s
ikj-loop	54.8142s	54.8005s	5.4820s	5.1860s	5.4791s	3.7413s
jik-loop	111.1539s	111.1581s	5.4821s	5.1861s	5.4791s	3.7835s
jki-loop	106.0454s	106.0522s	5.4821s	5.1860s	5.4791s	3.7413s
kij-loop	51.8134s	51.8105s	5.4821s	5.1860s	5.4792s	3.7412s
kji-loop	105.9441s	105.9516s	5.4821s	5.1861s	5.4791s	3.7465s
essl	0.8587s	0.8564s	0.8578s	0.8576s	0.8562s	0.8576s

Table 2: Time measurement of matrix-matrix multiplications in C with `nmax=999` and static arrays under different compiler options and loop orders, comparison with optimized ESSL routine

Table 2 shows the results of the time measurement for different compiler options and loop orders for `nmax=999` and Table 3 for `nmax=1000`. A measurment of the Blue Gene/P-optimized ESSL library routine DGEMM is also included. It is called by

```
dgemm ("N","N",nmax,nmax,nmax,alpha,matB,nmax,matA,nmax,1.000,matC, \
       nmax);
```

For `-O0` one gets very different results for different loop orders. The reason is that for some loops the memory structure is not well exploited. That means that in advantageous cases (ikj- and kij-loops) the read-write head just has to move to the next entries in the represention of the matrices **B** and **C** within the memory to perform the next cycle in the innermost loop whereas the corresponding element of **A**

Comp.	`-O0 -qarch=`		`-O3 -qarch=`		`-O5 -qarch=`	
	`450`	`450d`	`450`	`450d`	`450`	`450d`
ijk-loop	107.3586s	107.3587s	5.8468s	3.2114s	5.6723s	3.9227s
ikj-loop	54.9437s	54.9437s	5.8469s	3.2115s	5.6724s	3.9255s
jik-loop	111.4669s	111.4679s	5.8468s	3.2115s	5.6723s	3.9255s
jki-loop	106.3641s	106.3641s	5.8469s	3.2114s	5.6724s	3.9254s
kij-loop	51.8563s	51.8553s	5.8469s	3.2115s	5.6724s	3.9255s
kji-loop	106.3183s	106.3194s	5.8469s	3.2114s	5.6724s	3.9255s
essl	0.7513s	0.7513s	0.7525s	0.7512s	0.7525s	0.7512s

Table 3: Time measurement of matrix-matrix multiplications in C with `nmax=1000` and static arrays under different compiler options and loop orders, comparison with optimized ESSL routine

stays untouched. However in the other cases jumps between the rows of some matrices are necessary. With `-O3` and higher optimzation levels the compiler applies loop interchanges to the loop nest and unrolls outer loops twice. Additionally the `-O3`-optimization leads to an awesome performance boost. Moreover the dual FPU is activated and further reduces the runtime by a few percent in the odd case and by approximately 45% in the even case. For `-O5` the results for `-qarch=450` are compareable to those of `-O3` whereas for `-qarch=450d` one gets an additional improvement of 30% in the odd case but a decline of $\approx$ 20% in the even case. For higher optimizations the runtimes in the even case are a bit worse than those of the odd case. But especially the behaviour for `-O3 -qarch=450d` is conspicuous and does not fit to the rest of the results. It can be observed that if matrices of odd size are embedded into arrays of even size the runtime behaves similar to that of the even case (Table 3). The ESSL implementation of DGEMM reacts positively on an even amount of input data but otherwise its runtime is constant under all optimizations of the calling routine.
Consider now the same code but with a dynamic contiguous definition of the arrays.

```
#define nmax 1000;
double **matA = (double**)malloc(nmax*sizeof(double*));
double *matA1d = (double*)malloc(nmax*nmax*sizeof(double));
for (i=0;i<nmax;i++)
    matA[i] = &matA1d[i*nmax];
```

Again, `matB` and `matC` are declared similarly. For calling the ESSL version of DGEMM with dynamically allocated arrays one just has to call the routine with the 1-d versions of the arrays defined above.

```
dgemm ("N","N",nmax,nmax,nmax,alpha,matB1d,nmax,matA1d,nmax,1.000, \
         matC1d,nmax);
```

Table 4 and 5 show the results for the same time measurements as above but with dynamic arrays.

As in the static case without optimization there are completely different results for different loop orders. Again ikj- and kij-loop provide the shortest runtime. But now higher optimization levels do not have any influence on this behaviour. The runtime decreases for higher optimization but the compiler does not perform any loop interchanges or loop unrollings. Note that the relative time saving for the ikj- and the kij-loops are the highest among all loop orders. Furthermore the usage of the dual FPU has no influence on the runtime and there is unlike the static case just a little difference between arrays of odd and even size. Note that the behaviour of the the ESSL routine DGEMM is the same for static and dynamic arrays. As a conclusion it can be said that one should use statically declared arrays if possible. Static arrays provide the best runtime especially for `-O3` or higher optimization levels and the dual FPU takes a positive influence on the runtime. The optimization possibilities of the compiler are higher due to the

Comp.	`-O0 -qarch=`		`-O3 -qarch=`		`-O5 -qarch=`	
	`450`	`450d`	`450`	`450d`	`450`	`450d`
ijk-loop	127.6543s	127.6542s	79.5691s	79.6006s	36.8541s	36.8471s
ikj-loop	64.8427s	64.8427s	24.2691s	24.4611s	10.6261s	10.5008s
jik-loop	128.0674s	128.0673s	79.5345s	79.5929s	36.8968s	36.4679s
jki-loop	130.8488s	130.8487s	96.3188s	97.1742s	79.0192s	79.2330s
kij-loop	62.6452s	62.6453s	25.8485s	26.0783s	11.0350s	10.9230s
kji-loop	130.7086s	130.7083s	96.2278s	97.0431s	78.8980s	79.1170s
essl	0.8564s	0.8563s	0.8578s	0.8586s	0.8578s	0.8575s

Table 4: Time measurement of matrix-matrix multiplications in C with `nmax=999` and dynamic arrays under different compiler options and loop orders, comparison with optimized ESSL routine

Comp.	`-O0 -qarch=`		`-O3 -qarch=`		`-O5 -qarch=`	
	`450`	`450d`	`450`	`450d`	`450`	`450d`
ijk-loop	123.6689s	123.6693s	77.8859s	77.8861s	33.8415s	33.8517s
ikj-loop	64.9763s	64.9766s	23.7375s	23.7376s	11.0541s	11.0542s
jik-loop	127.7432s	127.7426s	78.5880s	78.5883s	34.6879s	34.7536s
jki-loop	128.0071s	128.0075s	95.5732s	95.5734s	78.5005s	78.5004s
kij-loop	62.8064s	62.8065s	25.1701s	25.1701s	12.947s	12.9471s
kji-loop	127.8701s	127.8704s	95.5007s	95.5010s	78.4533s	78.4534s
essl	0.7518s	0.7516s	0.7521s	0.7523s	0.7523s	0.7522s

Table 5: Time measurement of matrix-matrix multiplications in C with `nmax=1000` and dynamic arrays under different compiler options and loop orders, comparison with optimized ESSL routine

better knowledge of array properties at compile time.

Investigation of a Fortran program and comparison to the C results

Consider now a Fortran program that performs exactly the same operations as the C code above. The goal of the following is to analyse the behaviour of the IBM XLF compiler `mpixlf90_r` under the same optimization flags and to compare the results to those of the C version.

Comp.	`-O0 -qarch=`		`-O3 -qarch=`		`-O5 -qarch=`	
	`450`	`450d`	`450`	`450d`	`450`	`450d`
ijk-loop	115.1892s	115.1693s	5.7313s	4.8419s	5.7258s	4.4502s
ikj-loop	111.1044s	111.0705s	5.7313s	4.8419s	5.7258s	4.4830s
jik-loop	115.1409s	115.1328s	5.7313s	4.8419s	5.7258s	4.4916s
jki-loop	59.0778s	59.0725s	5.7313s	4.8420s	5.7258s	4.4310s
kij-loop	110.9408s	110.9410s	5.7313s	4.8419s	5.7258s	4.4798s
kji-loop	56.9662s	56.9558s	5.7313s	4.8419s	5.7258s	4.4484s
essl	0.8576s	0.8588s	0.8571s	0.8578s	0.8571s	0.8563s

Table 6: Time measurement of matrix-matrix multiplications in Fortran with `nmax=999` and static arrays under different compiler options and loop orders, comparison with optimized ESSL routine

Comp.	`-O0 -qarch=`		`-O3 -qarch=`		`-O5 -qarch=`	
	`450`	`450d`	`450`	`450d`	`450`	`450d`
ijk-loop	115.5159s	115.5139s	4.9150s	3.5254s	4.9149s	4.2921s
ikj-loop	111.4825s	111.4829s	4.9150s	3.5255s	4.9149s	4.2812s
jik-loop	111.2701s	111.2612s	4.9150s	3.5254s	4.9149s	4.2812s
jki-loop	59.2390s	59.2137s	4.9150s	3.5254s	4.9149s	4.2847s
kij-loop	111.3162s	111.3167s	4.9150s	3.5254s	4.9149s	4.2812s
kji-loop	56.9461s	56.9453s	4.9150s	3.5254s	4.9149s	4.2815s
essl	0.7517s	0.7513s	0.7512s	0.7506s	0.7512s	0.7525s

Table 7: Time measurement of matrix-matrix multiplications in Fortran with `nmax=1000` and static arrays under different compiler options and loop orders, comparison with optimized ESSL routine

Table 6 and 7 contain the measurement results for static arrays. They are similar to those of the C code but here arrays of even size provide a faster program execution than odd arrays. Note as well that the options `-O3 -qarch=450d` show the same unexpected behaviour than for the C code (here with using evenly allocated arrays). However the performed optimizations are a bit different. For `-O5` the compiler listing just shows a 6 times inner loop unrolling only for ijk- and jik-loops.

Comp.	`-O0 -qarch=`		`-O3 -qarch=`		`-O5 -qarch=`	
	`450`	`450d`	`450`	`450d`	`450`	`450d`
ijk-loop	145.7739s	145.7740s	30.5930s	35.6756s	5.7858s	5.8885s
ikj-loop	140.9074s	140.9088s	37.0187s	37.0199s	5.7858s	5.8885s
jik-loop	145.7471s	145.7469s	30.5807s	35.6015s	5.7858s	5.8885s
jki-loop	89.6512s	89.6311s	10.3027s	10.3720s	5.7858s	5.8885s
kij-loop	140.6853s	140.6850s	36.9467s	36.9463s	5.7858s	5.8885s
kji-loop	87.3443s	87.3542s	12.6150s	12.5981s	5.7858s	5.8885s
essl	0.8577s	0.8581s	0.8578s	0.8567s	0.8577s	0.8574s

Table 8: Time measurement of matrix-matrix multiplications in Fortran with `nmax=999` and dynamic arrays under different compiler options and loop orders, comparison with optimized ESSL routine

Comp.	`-O0 -qarch=`		`-O3 -qarch=`		`-O5 -qarch=`	
	`450`	`450d`	`450`	`450d`	`450`	`450d`
ijk-loop	146.2608s	146.2372s	30.5369s	36.3783s	4.9950s	4.9986s
ikj-loop	141.3701s	141.3645s	37.1243s	37.1242s	4.9951s	4.9994s
jik-loop	142.0382s	142.0389s	25.2470s	35.7083s	4.9950s	4.9996s
jki-loop	89.9853s	89.9098s	10.0569s	10.0568s	4.9951s	4.9994s
kij-loop	141.1833s	141.1698s	37.0656s	37.0658s	4.9950s	4.9986s
kji-loop	87.3795s	87.3141s	12.2902s	12.2902s	4.9950s	4.9995s
essl	0.7513s	0.7513s	0.7520s	0.7511s	0.7527s	0.7517s

Table 9: Time measurement of matrix-matrix multiplications in Fortran with `nmax=1000` and dynamic arrays under different compiler options and loop orders, comparison with optimized ESSL routine

For dynamic arrays the Fortran compiler is able to perform more optimizations than the C compiler. For `-O0` the Fortran version takes about 10 - 50% more time to run, depending on loop order. But using `-O3` decreases that time to 10 - 25% of that one without using optimizations, whereas using a C compiler

only decreases the runtime to approximately 45 - 75% of the original amount. For `-O5` the compiler even performs 4 times inner loop unrollings for the ijk- and jik-loop. Then the runtime can be compared to that for static arrays using the same flags. There is also a difference between oddly and evenly allocated arrays which cannot be observed for the other analysed optimizations. Similarly to the C case there is almost no influence of the dual FPU recognizable. But if one uses `-O3` there are exceptions for the ijk- and jik-loops where the execution with `-qarch=450d` takes about 20 - 40% more time.
Finally it can be said that the XLF compilers can deal better with dynamic arrays than the XLC compilers. But for both one can observe strange behaviour which is explained above. Additionally the optimization possibilities for static arrays are much greater. Thus the user should apply static arrays if possible. In the static case there are small runtime differences between arrays of odd and even size but there is no pattern determinable.

Deeper analysis of the static case with `-O3` and `-O5`

Due to the unexpected behaviour of the compiler on static arrays a more exhaustive analysis of this case is done with matrices of order 1999 and 2000 respectively. The results are shown in Table 10.

Options	`C`		`Fortran`	
	`nmax=1999`	`nmax=2000`	`nmax=1999`	`nmax=2000`
`-O3 -qarch=450`	36.970s	33.731s	42.525s	36.813s
`-O3 -qarch=450d`	50.084s	47.058s	51.462s	47.817s
`-O5 -qarch=450`	36.916s	33.731	42.525s	35.333s
`-O5 -qarch=450d`	30.000s (ijk), 52.937 - 53.248s	21.691 - 21.997s	37.756s (kji), 39.804 - 39.924s	24.876 - 24.983s

Table 10: Runtime of matrix-matrix multiplications in Fortran and C under different compiler flags

Compared to Tables 2, 3, 6 and 7 one gets completely different results. First of all, if `-O3` is activated the results become much worse while using `-qarch=450d`. For `nmax=999` and `nmax=1000` a contrarious behaviour can be observed. Second for both C and Fortran arrays of even size now show a better runtime than those of odd size, as it is expected. Third for `-O5` the dual FPU is exploited in a stronger manner if the arrays have even size. But for the C code the runtime is increased by almost 50% except the ijk-loop was used.
Further tests also show that if the `alpha` in the innermost loop of the computation is supressed the runtime may increase significantly although the computational effort decreases, e.g. for Fortran, `nmax=2000`, `-O5 -qarch=450d` the runtime increases from approximately 24,9s to more than 41s.
All this leads to the assumption that there are still some bugs in the IBM XL compilers.

Library Routine Analysis

Considered Libraries

- **BLAS:** The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations [5].
- **PBLAS:** The PBLAS are the Parallel Basic Linear Algebra Subprograms. The PBLAS are a small core library of linear algebra utilities, which can be highly optimized for various parallel architectures [6].
- **LAPACK:** LAPACK (Linear Algebra PACKage) is written in Fortran77 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of

equations, eigenvalue problems, and singular value problems. The associated matrix factorizations are also provided, as are related computations [7].

- **ESSL:** The ESSL (Engineering and Scientific Subroutine Library) is an IBM product and consists of highly optimized mathematical routines that can be assigned to the following fields: matrix operations (BLAS and more), equation systems, eigenvalue problems, FFT, sorting, interpolation, numerical integration and random number generators [8].
- **BLACS:** The BLACS (Basic Linear Algebra Communication Subprograms) project is an ongoing investigation whose purpose is to create a linear algebra oriented message passing interface that may be implemented efficiently and uniformly across a large range of distributed memory platforms [10].
- **ScaLAPACK:** The ScaLAPACK (or Scalable LAPACK) library includes a subset of LAPACK routines redesigned for distributed memory MIMD parallel computers. It is currently written in a Single-Program-Multiple-Data style using explicit message passing via BLACS for interprocessor communication [11].
- **MUMPS:** MUMPS (MUltifrontal Massively Parallel Solver) is a package for solving linear systems of equations of the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a square sparse matrix that can be either unsymmetric, symmetric positive definite, or general symmetric. MUMPS uses a multifrontal technique which is a direct method based on either the LU or the the LDL^T factorization of the matrix. MUMPS exploits both parallelism arising from sparsity in the matrix $\mathbf{A}$ and from dense factorization kernels [12].

Analysis of the PBLAS routine PDGEMM

Because of the intensive usage of PBLAS subroutines in parallel program packages it is a crucial issue to analyse the behaviour of this library first. In this subsection the parallel matrix-matrix multiplication PDGEMM for the operation $C := \alpha AB + \beta C$ is investigated. A 5×5 process grid is chosen to evaluate the runtime of the operation for square matrices of sizes $N = 3000 \ldots 4000$. The blocksize $NB (= MB)$ of the two-dimensional block cyclic distribution is set to 100.

0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

Figure 1: Two-dimensional block cyclic distribution

In Figure 1 one can see an example of the two-dimensional block cyclic distribution. Consider P processes arranged in a $P_r \times P_c$ rectangular array of processes, indexed in a two-dimensional fashion by (p_r, p_c) with $0 \leq p_r < P_r$ and $0 \leq p_c < P_c$. All the processes (p_r, p_c) with a fixed p_c are referred to as process column p_c. All the processes (p_r, p_c) with a fixed p_r are referred to as process row p_r. The figure fits e.g. for $N = 16$, $P = 4$, $P_r = P_c = 2$ and block sizes $MB = NB = 2$. This layout permits P_c-fold parallelism in any column and calls to the Level 2 BLAS and Level 3 BLAS on local subarrays. Finally this layout also features good scalability properties [13]. The analysis of the considered routine implies again static and dynamic arrays of odd and even size respectively. For compilation

```
mpixlf90_r -qarch=[val] -qtune=450 -O3
```

was used.

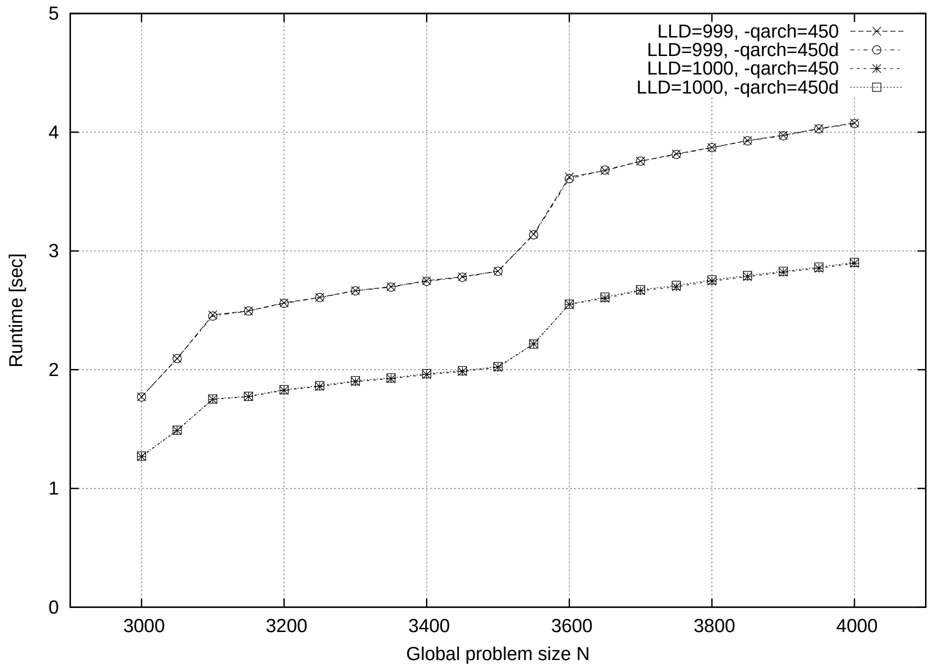


Figure 2: Runtime of PDGEMM with statically allocated arrays on a 5×5 process grid and $NB = 100$

In the static case the local leading dimension (LLD) of the distributed matrices on each process is fixed to 999 or 1000. From Figure 2 it can be easily seen that like in the section above arrays of even size provide a much better performance than those of odd size. The reason is the more advantageous alignment of evenly allocated arrays. Furthermore one notices that there are jumps in each graph for $N = 3000 \ldots 3100$ and $N = 3500 \ldots 3600$. For $N = 500M,\ M \in \mathbb{N}$ every process has the same number of blocks and those are completely full. If one increases the problem size now one process row will have more blocks than the others. For increasing matrix sizes the new entries will be assigned to that blocks and thus the corresponding processes will have more work to do than the other ones. This will of course increase the runtime in a particularly strong manner. Note as well that `-qarch=450d` has no perceptible influence on the performance of the program. PDGEMM is based on the BLAS and since the matrix is distributed on the processors BLAS routines will be called locally. On JUGENE the BLAS routines are included in the ESSL library and as one can read above the ESSL codes are highly optimized for using "Double Hummer".

Consider now the operation with dynamic instead of static arrays. Now every process gets exactly that amount of memory it needs. For the analysis of the odd case matrices of order $N = 2999 + 50M,\ M \in \mathbb{N}$ are investigated. There are almost no differences to the static behaviour except of periodic oscillations for odd N. To explain that in more detail Figure 3 shows a time plot in a higher resolution. Here the time is measured for all $N = 3150 \ldots 3250$ and the results are combined to two graphs for odd or even N respectively. In the odd case a remarkable reduction of the runtime is ascertainable between $N = 3199$ and $N = 3201$. In this example for $N = 3199$ one process row contains matrix blocks that are stored

in an array of local leading dimension 699 whereas the other process rows consist of blocks that are stored in arrays with local leading dimensions of either 600 or 700. If the problem size is now increased to $N = 3201$ all arrays of local size 699 will be completed to size 700 and a corresponding number of arrays of size 600 will be enlarged to size 601. That means that the arrays of odd dimension will be reduced during that step. Due to the dominating influence of arrays with odd size on the runtime this jump in the curve can be explained by the reduced size of odd-sized arrays during that step.
Note that for PDGEMM the block size does not have a major impact on the results as long as the work load distribution is similar on all processors.

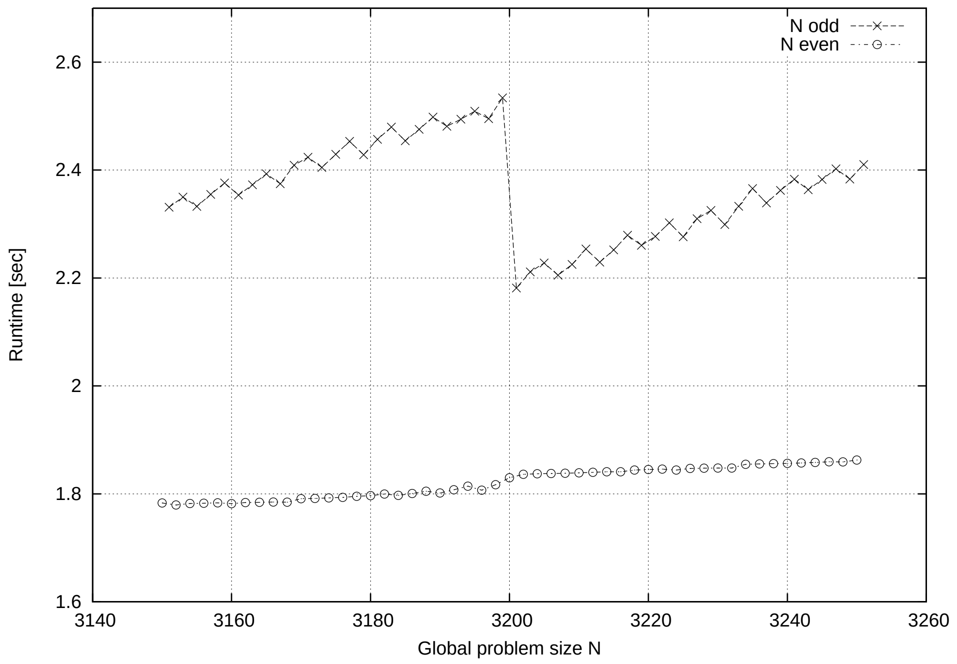


Figure 3: Runtime jumps of PDGEMM with dynamically allocated arrays on a 5×5 process grid and $NB = 100$

Analysis of the ScaLAPACK routine PDSYEVX

PDSYEVX computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix $\mathbf{A}$ by calling the recommended sequence of ScaLAPACK routines. Eigenvalues/vectors can be selected by specifying a range of values or a range of indices for the desired eigenvalues [11]. First the matrix $\mathbf{A}$ is reduced to tridiagonal form using Householder transformation (call PDSYNTRD), second the eigenvalues are computed using bisection (call PDSTEBZ) and third the eigenvectors are computed using inverse iteration and back-transformation (call PDSTEIN and PDORMTR) [14].
Here matrices of order $N = 3000 \dots 3050$ are taken for the computations. They are built with random eigenvalues that are already known before the computations to verify the results. The goal is to compute the largest 10% of the eigenvalues and their corresponding eigenvectors. The measurements take place on a 4×4 process grid. The block size of the two-dimensional block-cyclic distribution is 32. The compilers `mpxlf` for FORTRAN 77 and `mpixlf90_r` for Fortran 90 codes were used with the flags

```
FFLAGS = -O3 -qarch=[val] -qtune=450 [more_opts]
```

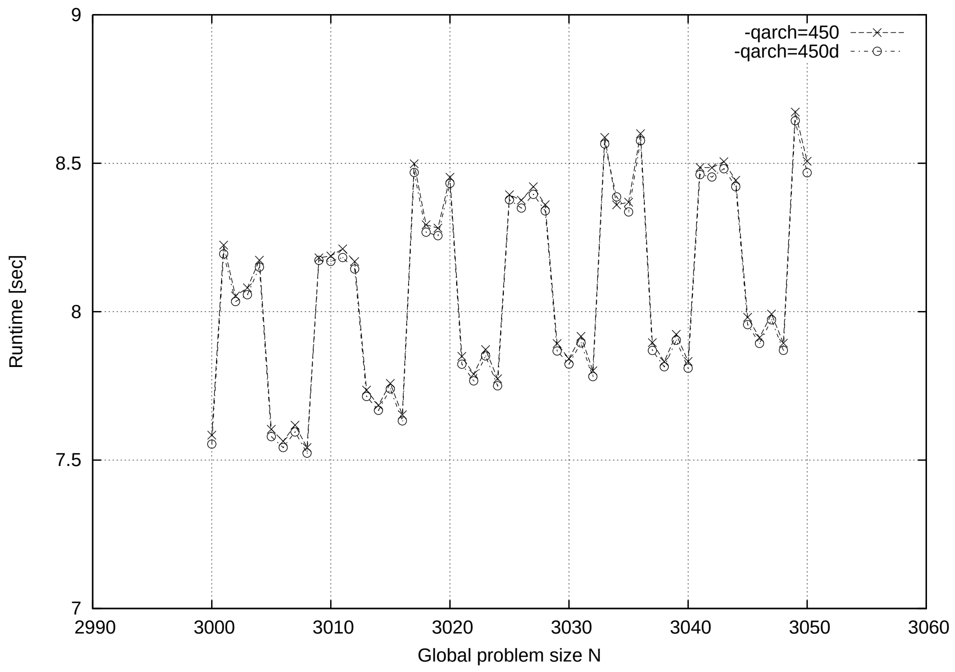


Figure 4: Runtime of PDSYEVX with dynamic arrays on a 4×4 process grid

First the dynamic case is tested. The results can be seen in Figure 4. First one recognizes that the runtimes are arranged in clusters. Unlike in the investigation of PDGEMM local arrays of odd leading dimension do not necessarily lead to a bad performance. Second `-qarch=450d` has again no major influence on the runtime of the routine though there is a very small improvement in general. The reason is again that ScaLAPACK makes intensive use of PBLAS routines which then rely on ESSL calls.
The results of the static case are compareable to those of the dynamic one. Again the values belong to clusters but now the two graphs have different cluster patterns. Like in the dynamic case there is almost no performance improvement by using `-qarch=450d`.

Analysis of the MUMPS library

Last the MUMPS package was analysed. MUMPS uses a self-defined datastructure `DMUMPS_STRUC` whose definition is contained in `dmumps_struc.h` in the include directory. This new type contains all the information needed e.g. input and output data, control variables or error indicators. Here a variable `mumps_par` of type `DMUMPS_STRUC` is used.

```
INCLUDE 'dmumps_struc.h'
TYPE (DMUMPS_STRUC) mumps_par
```

First a real world problem is considered in which a matrix from a finite element discretization of a crank appears. The matrix is square and has an order of 192858. It it sparse due to the fact that it has only 12226189 nonzero entries in the lower triangle. Furthermore it is symmetric positive definite and thus MUMPS is able to exploit this structure (set `mumps_par%SYM = 1`). A master-slave-setup is used (`mumps_par%PAR = 0`) which means that the host is not involved in factorization and solve phase. The host will only hold the initial problem, perform symbolic computations during the initial phase, distribute data, and collect results from other processors [12]. The MUMPS versions 4.7.3 and 4.8.1 are compared both with and without using the dual FPU respectively. Figure 5 shows the results of measurements of the runtime of a linear system having the crank matrix as system matrix for $2, 4, 8, \dots, 512$ processors. With 64 processors the dual mode (`-mode DUAL`) and for 128 and more procssors the virtual node mode (`-mode VN`) were used for the computations. That means that two or four cores on each node are chosen to run the application. This has the advantage of a higher node effiency but the disadvatage of less memory per core because the memory of one node is shared among all its cores. This may lead to program crashes caused by a lack of memory. This happens for example if one tries to solve a crank system under neglection of the positive definity with `mumps_par%SYM = 2` and 64 or more processors in dual or VN mode. From Figure 5 it can be seen first that version 4.7.3 provides a slightly better performance for a lower amount of processors whereas version 4.8.1 shows a better runtime for a high amount of processors. Second it can be seen that MUMPS is not scaling well for many cores. The optimal runtime is reached with 64 processors. For more processors the amount of communication already increases the runtime. Third it can be observed that using `-qarch=450d` does not lead to a noticabel performance improvement. If one takes a look into the code of MUMPS one recognizes that there are a lot of integer computations that basically deal with the indices of nonzero matrix entries to improve its structure. The factorization step itself is using ScaLAPACK and from above one may read that it is already highly optimized due to the intensive use of ESSL calls.

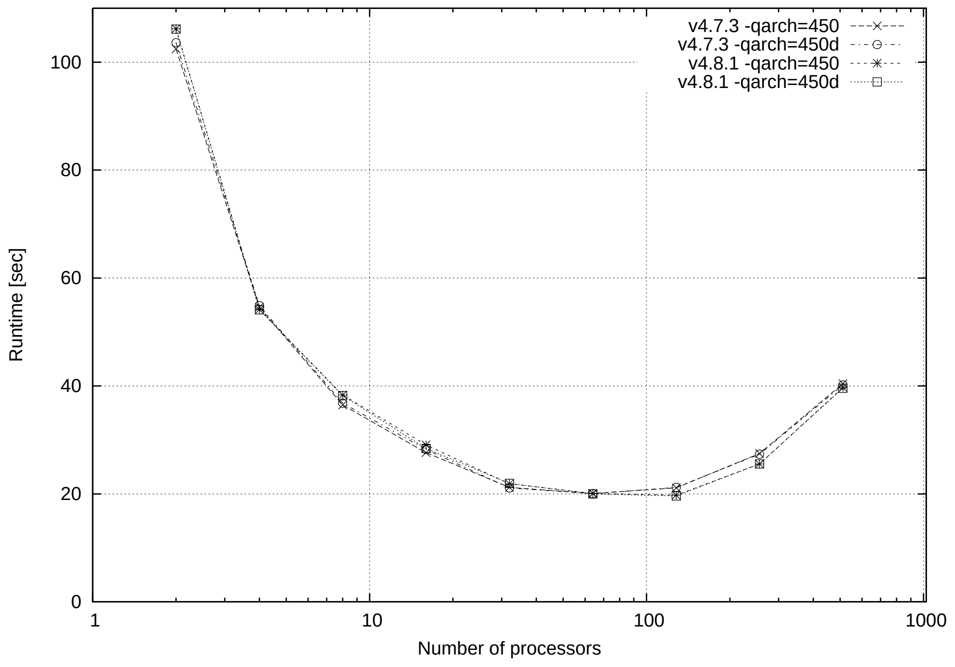


Figure 5: Runtime of MUMPS with the crank matrix and exploiting the structure properties using a master-slave setup

MUMPS was also analysed for systems with unsymmetric system matrix. Here a square matrix of order 4000 with 160000 nonzero elements was chosen to test the program package. The matrix is built in such a way that each row contains exactly 40 nonzero entries. For those systems MUMPS cannot exploit any matrix structure properties and thus the amount of memory which is needed and the execution times rise to a high level. Hence it can happen that a MUMPS call already crashes on Blue Gene/P with matrices of just a few million entries because of memory overflow.

Figure 6 shows the results for the described unsymmetric system. Notice that now version 4.7.3 has a slight runtime advance to version 4.8.1 in most cases. This difference becomes especially visible for 256 and more processors where version 4.7.3 needs about 10% less time than version 4.8.1. The reason is probably the new defined datastructure `DMUMPS_STRUC` which generates a lot of misalignment warnings during the compile process.

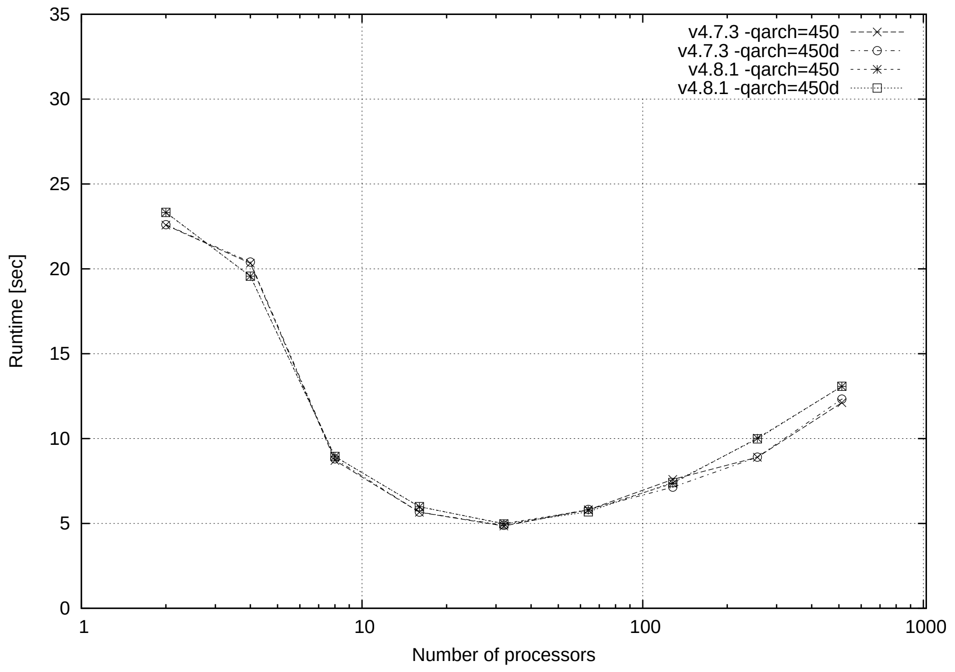


Figure 6: Runtime of MUMPS with an unsymmetric system using a master-slave setup

Last a system without using a master-slave scheme was analysed (set `mumps_par%PAR = 1`). Now the host is involved in the factorization and solve phases. If the initial problem is large and memory is an issue, PAR = 1 is not recommended if the matrix is centralized on processor 0 because this can lead to memory imbalance, with processor 0 having a larger memory load than the other processors. Note that seeting PAR to 1 and using only one processor leads to a sequential code [12].
For an unsymmetric system with a well balanced work load version 4.7.3 always behaves better than version 4.8.1. The runtime benefit is small but anyway about 5 - 10%. Note as well that the sequential version provides a faster program execution than a parallel one with 2 processors.

Conclusion

In the introducing section it could be seen how misalignments can increase the runtime of applications by orders of magnitude. So it is recommended to carefully align the data and regard on corresponding warnings during the compile process. Additionally the behaviour of the IBM XL compilers was analysed under different optimization flags and array declarations. Static arrays could be easily optimized by the C compiler as well as by the Fortran compilers. The test programs showed different preferences for arrays of odd or even size and the dual FPU provided a reasonable performance boost for a suitable optimization level. However for dynamic arrays one could not recognize such a high performance improvement like in the static case while using the dual FPU. Also the Fortran compilers behave much better for those arrays and can perform better optimizations. Though unexpected low runtimes could be observed for some cases that still pose some questions.
In the second section specific routines from different linear algebra libraries and packages were analysed.

During the investigation of the PBLAS routine PDGEMM arrays of even size showed an improved runtime. However for the ScaLAPACK routine PDSYEVX such a behaviour could not be observed although it makes extensive use of the PBLAS routines and thus of the ESSL library. Finally two versions of the MUMPS package for the solution of large sparse equation systems were considered. A worse runtime of the new version 4.8.1 could be observed for a high amount of processors but generally both show similar results.
All the analysed libraries and packages highly rely on the ESSL routines which already provide code that is optimized for using the dual FPU. Thus high performance improvements for libraries that are compiled with `-qarch=450d` are not observeable. The main program may also be compiled with `-qarch=450` while the used libraries are compiled with `-qarch=450d` and vice versa.

Acknoledgement

I would like to thank my supervisor Mrs. Inge Gutheil for the useful help during my whole stay at the research centre. Additionally it would like to appreciate the team of the Jülich Supercomputing Centre for the perfect organization of my stay. Especially Matthias Bolten and Robert Speck are mentionable in this context. Finally I would like to thank the other guest students for the pleasant work atmosphere as well as Prof. Dr. Peter Benner for the support letter that was necessary for the application.

References

1. Brian Smith, Rajiv Bendale, Kirk Jordan, Jerrold Heyman, Bob Walkup, BG/P Software Overview, Training, 2007
   http://www.nccs.gov/wp-content/training/2008_bluegene/03_bgp-comm.pdf
2. Norbert Attig, Marc-André Hermanns, Introduction to JUMP and JUGENE - Programming and Usage, Presentation, August 11, 2008
   http://www.fz-juelich.de/jsc/files/docs/vortraege/ibm-nutzung2008/nic.pdf
3. Evolution of the IBM System Blue Gene Solution, Redpaper, February 2008
   http://www.redbooks.ibm.com/redpapers/pdfs/redp4247.pdf
4. Exploiting the DUAL FPU in Blue Gene, March 2006 (Updated June 2006)
   http://www-03.ibm.com/systems/resources/systems_deepcomputing_pdf_exploitingbluegenedoublefpu.pdf
5. BLAS (Basic Linear Algebra Subprograms)
   http://www.netlib.org/blas
6. PBLAS - The Parallel Basic Linear Algebra Subprograms
   http://people.scs.fsu.edu/ burkardt/f_src/pblas/pblas.html
7. LAPACK – Linear Algebra PACKage
   http://www.netlib.org/lapack
8. Ulrike Begiebing (Editor), Einführung in die Benutzung des zentralen AIX, Benutzerhandbuch, 3. Auflage, 17. Februar 1998
   http://www.fz-juelich.de/jsc/docs/bhb/bhb_html/d0101/
9. MPI: A Message-Passing Interface Standard Version 2.1, Message Passing Interface Forum, June 23, 2008
10. BLACS
    http://www.netlib.org/blacs
11. ScaLAPACK Home Page
    http://www.netlib.org/scalapack/scalapack_home.html
12. MUltifrontal Massively Parallel Solver (MUMPS Version 4.7.3) Users' Guide, May 4, 2007
13. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. C. Whaley, ScaLAPACK Users' Guide, siam, Philadelphia, 1997
14. Robert Speck, Performance of the ScaLAPACK Eigensolver PDSYEVX on IBM Blue Gene/L - First Results. *Rüdiger Esser (Editor), Beiträge zum Wissenschaftlichen Rechnen - Ergebnisse des Gaststudentenprogramms 2005 des John von Neumann-Instituts für Computing, p. 115-128*
    http://www.fz-juelich.de/jsc/gaststudenten/ib-2005-13.pdf