Technische Universität München
Lehrstuhl für Theoretische Chemie

ParaGauss and ParaTools – Transition State Search and Efficient Parallelization for Density Functional Calculations

Astrid Nikodem

Vollständiger Abdruck der von der Fakultät für Chemie der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 2.4.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Chemie am 22.4.2013 angenommen.

Danksagung

Zunächst möchte ich ganz besonders meinem Betreuer Herrn Professor Dr. Dr. h.c. Rösch danken. Ihm verdanke ich das interessante Thema und eine hervorragende Betreuung. Er hatte immer Interesse an meiner Arbeit und deren Entwicklung. Weiterhin hatte er eine Menge guter Ratschläge parat.

Des weiteren möchte ich Dr. Alexei Matveev danken, der mir viele Tricks zum gelungenen Programmieren zeigte und immer bereit war, sich auf eine Diskussion über unklare Aspekte oder neue Ideen bezüglich der Arbeit einzulassen.

Mein besonderer Dank gilt auch Dr. Sven Krüger, insbesondere für die vielen wertvollen Tipps zur Gestaltung einer Dissertationsschrift.

Ich möchte auch meinen anderen Kollegen aus der Arbeitsgruppe Rösch für eine interessante und produktive Zusammenarbeit danken, insbesondere meiner Zimmernachbarin Duygu Başaran, Cheng-chau Chiu und Dr. Zhao Zhi-Jian für ihre Unterstützung bei der Suche geeigneter Testbeispiele für die Übergangszustandssuchprogramme. Auch für weitere gute Hinweise bezüglich dieser Programme und ihrer Nutzerinterfaces von Dr. Alena Kremleva, Dr. Ion Chiorescu und Dr. Konstantina Damianos bin ich sehr dankbar. Ich danke Thomas Soini für die gute Zusammenarbeit an der Schnittstelle zwischen der dynamischen Lastenverteilung und ihrer bisher wichtigsten Anwendung, den 4-Zentrenintegralen. Eine fundierte Unterweisung in den Aufbau und die Wartung eines Computersystems verdanke ich Dr. Wilhelm Eger. Ein großes Dankeschön geht an alle Mitglieder der Arbeitsgruppe für das Gemeinschaftsgefühl, das sie verbreiteten. Für das Arbeitsklima und anregende Diskussionen danke ich Dr. Alexander Genest, Bo Li, Yin Wu, Dr. Raghunathan Ramakrishnan, Dr. Juan Santana, Dr. Remi Marchal, Dr. Benjami Martorell und Dr. Virve Karttunen.

Mein herzlicher Dank geht auch an die Gruppe von Dr. Vladimir Naslusov am Institute of Chemistry and Chemical Technology in Krasnoyarsk. Die Herzlichkeit und Freundlichkeit mit der ich dort bei meinem dreimonatigen Aufenthalt aufgenommen worden bin machten diese Zeit zu einem unvergesslichen Erlebnis. Deshalb geht mein Dank an Dr. Valdimir Naslusov selbst, an seine Mitarbeiter Dr. Aleksey Shor, Dr. Elena Ivanova-Shor, Dr. Valdimir Rivanenkov und insbesondere Dr. Swetlana Laletina.

Zu Dank verpflichtet bin ich den MAC-Projekten B4 und B6, vor allem für neue Denkansätze, die aus der Zusammenarbeit entstanden sind.

Schließlich möchte ich auch noch meiner Familie, meinen Eltern Henny und Dr. Manfred Nikodem und meinem Bruder Thomas Nikodem, für ihre Unterstützung danken und dafür, dass sie immer an mich geglaubt haben.

List of Abbreviations

BFGS	Broyden-Fletcher-Goldfarb-Shanno
CI-NEB	climbing image nudged elastic band (method)
CI-string	climbing image string (method)
CPU	central processing unit
DIIS	direct inversion in the iterative subspace (method)
DIIS-cc	“direct inversion in the iterative subspace” (method) on the charge coefficients
DIIS-KS	“direct inversion in the iterative subspace” (method) on the Kohn–Sham matrix
DLB	dynamic load balancing
FIRE	fast inertial relaxation engine
HOMO	highest occupied molecular orbital
LUMO	lowest unoccupied molecular orbital
mDL	modified Dimer Lanczos (method)
MPI	message passing interface
NEB	nudged elastic band (method)
RMA	remote memory access
SCF	self-consistent field (method)
SR1	symmetric rank one (matrix update method)
W/ET	work per elapsed time
XC	exchange-correlation (functional)

Introduction

Computer developments reduce the limitations on the size of the chemical systems that can be treated. However, to take advantage of the achievements in computer technology, it is required to adapt the electronic structure codes of computational chemistry. Furthermore it is necessary to reduce the computational cost as much as possible, for example by reducing the number of iterations of internal loops (e.g., in a self-consistent field calculation) as well as for external calls of the program (if special geometries are searched so that the electronic structure have to be calculated several times).

Computer architecture develops steadily. The most popular description of the trend is Moore's law, which predicts that the number of transistors in an integrated circuit will double every two years.[1] This exponential trend seems to be correct for the past. Increase can also be found in memory bandwidth. However what can be seen in recent years and will be even more pronounced in the future of the computer architecture is a change towards parallel processing. Parallel programs could ideally reach speedups linear with the number of applied cores. In reality perfect parallelization is hardly possible, due to communication between the processes, which might include transfer of a large amount of data, and inefficient parallelizable parts in a program. Recall Amdahl's law.[1] Time requirements on one or several cores provide the scaling factor of the program; the term "granularity" is used in parallel processing to define the ratio of the amount of computation to the amount of communication. The scaling factor and the granularity are important quantities for characterizing the quality of the parallel properties of a program.

The scaling factor of a quantum chemistry program depends on the size of the system. For an efficient program, for which the communication overhead increases slower than the computation effort due to the increasing system size, a large palladium cluster, for example, will run more efficiently for a larger number of cores than a small palladium cluster. Quantum chemistry programs, like ParaGauss,[2-6] were often initially developed several years ago. They were adapted to the computer architecture available at that time and the parallelization strategies designed for them. They were extended steadily. If the size of the particle, which provides the surface on which the reaction takes place, should be extended from 75 atoms to more realistic sizes of several hundred atoms, the computational requirements will significantly increase. To prevent the real time required for the calculations to increase likewise, the code should scale well up to several hundred cores, even thousand cores and more. This requires that the structure of large parts of the code ParaGauss has to be changed as it was initially not created for using such large numbers of cores.

Quantum chemistry codes provide approximate solutions to the electronic Schrödinger or the relativistic Dirac equation of molecular systems.[6-8] These equations determine the energy of a system and its many-electron wave function (or electron density). The wave function of the atomic nuclei is usually neglected by treating the nuclei in the Born–Oppenheimer approximation.[7] The electronic wave function of a molecular structure is in most cases searched in an iterative process, called the self-consistent field (SCF) cycle.

The current thesis includes improvements on the parallelization of quantum chemistry calculations by developing a general parallelization library. This library describes a method for dynamically distributing similar tasks, fit for the expected growth in core numbers, and is a significant part of the current thesis. The library is used for example for the task distribution of the integral calculations, both in the pre-SCF and the post-SCF part. In the pre-SCF part these are integrals over combinations of orbital basis functions, which are used to build the Hamiltonian matrices in the SCF cycle. They can be calculated beforehand and dominate the computation time of the pre-SCF part. In the post-SCF part the first- and second-order derivatives of the energy with respect to the positions of the nuclei have to be calculated. This involves integrals like in the pre-SCF part, only with different sizes and values. There are two kinds of these integrals, one kind is calculated numerically over a grid, the other is calculated analytically. The numerically solved integrals generate several tasks of nearly equal computational effort. The analytically solved integrals are calculated in tasks of varying computational effort, requiring therefore a dynamical distribution of the tasks for an effective parallel calculation. Speeding up the evaluation of the calculation of one geometry structure is one contribution of the current thesis. For the SCF cycle which gets harder to converge with increasing system size, several convergence accelerators are available (in ParaGauss). The current thesis describes the implementation of a new accelerator, the second contribution for increasing the speed up of a single-point calculation.

The most interesting (molecular) structures on the potential energy surface required for describing a chemical reaction are the (local) minima and first-order saddle points. The minima represent initial, intermediate, and final states of a reaction. The saddle points connecting two of these minima, often called transition states, are important for determining reaction pathways. The energy difference of a transition state to the preceding local minimum is used to calculate the reaction probability and to estimate the velocity of a chemical transformation. Thus it is crucial to determine the corresponding transition states of a reaction.

These structures are harder to find than those of minima. The special requirements on the structure and the fact that often a special transition state out of many possible ones is searched, belonging to a given reaction path, makes this a challenging task for specialized programs. A toolbox of routines, called ParaTools, was created for that task. It consists of several routines which are useful for searching transition states.

This set of tools can not only be used in combination with ParaGauss but also with other quantum chemistry codes that deliver energy and gradients of the energy. Some routines require the calculation of the energy and its nuclear displacement gradients for various structures, known at the same time. These calculations are processed in parallel. As "embarrassing parallel" tasks, such a procedure scales better than using only the parallelization within the quantum chemistry programs. The implementation and development of routines for ParaTools, also for parallel execution, were done in the context of the current thesis.

One example illustrating the use of the improvements of the current thesis is taken from the field of catalysis. Heterogeneous catalysts are often found in the shape of nano-sized active particles on a mostly inert support. Many theoretical studies of catalytic reactions neglect the support material and concentrate on the active particles.[9] Even with these simplifications it is not possible, due to the high complexity of the systems, to describe exactly these particles on which the reactions should occur. Calculations on computers are limited in the size of the systems treated not only by the approximations used in the theory, but furthermore in calculation time and hardware restrictions regarding the size of the molecules. For example the theoretical study of Yudanov et al.[9] was carried out for a model particle of 79 atoms instead of 300 to 3000 as proposed by the experimentalists.[10] The reduction in particle size and the additional symmetry reduction applied,[9] allow extensive studies with accurate results at the cost of using a model that is simpler than the system treated by the experimentalists. It is another goal of the current thesis that the size of test systems can be increased to large values while accurate results in extensive studies shall become feasible.

The thesis is structured as follows. The first part of this thesis describes the algorithm of the general parallelization library, which replaces the previous one for parallelizing the integrals in the pre- and post-SCF parts. Considering different computer architectures there are two variants available. This part also includes a study on the performance of the methods for different computer architectures. The performance of the integral parts in the pre- and post-SCF is also explored. Additionally this part includes tests on a newly implemented integral part, the so called four-center integrals.[11]

The second part is about accelerating the convergence of the SCF cycle with the "direct inversion in iterative subspace" (DIIS) method.[12] This well established method was implemented as an addition and compared to the methods already existing in ParaGauss.

The third part is about extensions to the ParaTools framework, which is a toolbox of routines intended, among other things, mainly for transition state search. ParaTools included already several methods for searching for a reaction path, the results of which can be used to locate approximately a transition state. These methods were improved and extended. However the main focus of the extension of ParaTools was in adding local transition state search methods. Together with existing reaction path methods, these new methods can be combined to very efficient two-step approaches. Furthermore, this part of the thesis includes a large test of various combinations of the two-step approach on systems inspired by recent investigations that were carried out by the group of Prof. Rösch.[13]

Part I: Parallel Processing

1. Parallelization in the Density Functional Method

1.1 Basics of the Density Functional Method

The current section describes some basics of quantum chemistry programs, especially those using the density functional method. The main focus is on tasks of these programs which can be easily parallelized.

For quantum chemistry programs the Schrödinger equation, or when considering relativistic effects, the Dirac equation, are the fundamental equations. These equations describe the energetics and the distribution of the electrons (and in principle also of the nuclei) of atoms and molecules.[7] The programs provide the energy as a result, and via its first-order derivative, the gradients of the energy, which is used to optimize the positions of atomic nuclei. Even the Schrödinger equation, as the simpler one of them, cannot be solved analytically for other than the simplest cases. Therefore there are several approximations involved in the methods used by quantum chemistry programs. Among the most common methods are the Hartree–Fock method[14] and the density functional theory method.[7] The program ParaGauss,[2-6] the focus of the investigations of this part of the thesis, exclusively uses density functional theory. It contains also the possibility for including relativistic effects by the Douglas–Kroll–Hess method.[8, 15-16] These contributions do not change the principle structure of the program. Thus even though the following description restricts itself to the non-relativistic case the effects of the parallelization are also present in relativistic calculations.

Researchers often are interested in the stationary states of a system which leads to time-independent equations. A common approximation, the Born–Oppenheimer approximation,[7] is that the structure of a molecule, given by the positions of the atomic nuclei, is fixed and only the electronic wave function is to be calculated. The resulting electronic Schrödinger equation is transformed into an eigenvalue equation:

$$\hat{H}\Psi = E\Psi \tag{1.1}$$

with the differential operator, $\hat{H}$, the Hamiltonian which acts on the many-electron wave function Ψ of the system. Using the symbol Z_A for the charge of the nucleus A and atomic units, the standard Hamiltonian is[7]

$$\hat{H} = -\frac{1}{2}\sum_i \nabla_i^2 - \sum_{i,A} \frac{Z_A}{|\mathbf{r}_i - \mathbf{r}_A|} + \frac{1}{2}\sum_{\substack{i,j \\ i\neq j}} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}. \tag{1.2}$$

Here, $\nabla = (\partial / \partial x, \partial / \partial y, \partial / \partial z)$; the vector $\mathbf{r}_\alpha$ designates the location of particle α where α refers to electron *j* or a nucleus *A*. The many-electron wave function Ψ belonging to the lowest energy *E* represents the ground state of the system. In Hartree-Fock and Kohn–Sham theory one approximates the many-electron wave function Ψ of a system of *N* electrons as Slater determinant of *N* ("occupied") one-electron wave functions (orbitals) φ_k. Density functional theory introduces an energy expression that depends only on the electronic density function $\rho(\mathbf{r})$ which, in the Kohn–Sham approach, is generated from *N* occupied orbitals:

$$\rho(\mathbf{r}) = \sum_k |\varphi_k(\mathbf{r})|^2 \tag{1.3}$$

The orbitals φ_k. are generated by solving the Kohn–Sham equation which has the form of a one-electron Schrödinger equation, analogous to Eq. (1.1), except with the one-electron Hamilton operator:

$$\hat{h}_{\mathrm{KS}} = -\frac{1}{2}\nabla^2 - \sum_A \frac{Z_A}{|\mathbf{r}-\mathbf{r}_A|} + \int d\mathbf{r}' \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} + v_{\mathrm{xc}}(\mathbf{r}) \tag{1.4}$$

The exchange-correlation (xc) potential $v_{\mathrm{xc}}(\mathbf{r})$ cannot be calculated exactly because the underlying energy contribution E_{xc} is known only approximately. Only for this reason the Kohn–Sham equation

$$\mathbf{h}_{\mathrm{HS}}\varphi_i = \epsilon_i \varphi_i \tag{1.5}$$

is not an exact equation.

In a well-established strategy for solving the Kohn–Sham problem, one represents orbitals φ_k. as linear combinations of ansatz functions χ_i, so-called "basis functions":[14]

$$\varphi_k = \sum_i \chi_i c_{ik}$$

A popular choice relies on Gaussian-type basis function, that comprise a Gaussian radial part.[5] This choice of functions admits efficient (analytical) procedures for evaluating most of the resulting so-called "matrix elements", i.e. scalar products of pertinent parts of the Hamiltonian sandwiched between two arbitrary basis functions. For instance, the matrix elements of the kinetic energy operator are:

$$t_{ij} = \langle \chi_i | -\tfrac{1}{2}\nabla^2 | \chi_j \rangle = \int d\mathbf{r} \chi_i(\mathbf{r})(-\tfrac{1}{2}\nabla^2)\chi_j(\mathbf{r})$$

Eventually, these matrix elements are collected in a square matrix **t**. This is an example of a 2-center integral because the basis functions involved may be located on (at most) two atomic centers. Four-center integrals arise when one treats the Coulomb interaction of between the electrons:

$$\langle ij|kl\rangle = \int\int \chi_i(\mathbf{r})\chi_j(\mathbf{r}')\chi_k(\mathbf{r})\chi_l(\mathbf{r}')/|\mathbf{r}-\mathbf{r}'|\,\mathrm{d}\mathbf{r}\mathrm{d}\mathbf{r}' \tag{1.6}$$

In an approximate treatment of the electron-electron Coulomb interaction, three-center integrals occur.[17] Only integrals of the exchange-correlation potential and have to be evaluated in numerical fashion. A strategy of adequate accuracy employs a superposition of atom-centered grids.[18-19] Analogous to the matrix **t** of the kinetic energy operator one constructs a Kohn–Sham matrix $\mathbf{h}_{\mathrm{KS}}$ with all terms of the operator in Eq. (1.4).

In the analytical evaluation of integrals one treats so-called "batches", where all matrix elements involving the basis functions of a so-called "shell" are evaluated in one procedural task. A shell index i represents all functions χ_i that share a single Gaussian exponent (radial part) and the same angular momentum (angular part), but differ in the projection of the angular momentum.

This strategy of representing the Kohn–Sham orbitals as linear combination of basis functions turns the solution of the Kohn–Sham equation into a generalized algebraic eigenvalue problem of a symmetric matrix:

$$\mathbf{h}_{\mathrm{KS}}\mathbf{C} = \mathbf{S}\mathbf{C}\boldsymbol{\varepsilon} \tag{1.7}$$

The coefficients c_{ik} of orbital φ_k are collected in column k of the square matrix **C**, the corresponding eigenvalues ε_k in the diagonal matrix **ε**, and the "metric" matrix **S** contains the overlap matrix elements:

$$S_{ij} = \langle \chi_i | \chi_j \rangle = \int d\mathbf{r}\chi_i(\mathbf{r})\chi_j(\mathbf{r})$$

Eq. (1.7) only superficially has the form of a linear eigenvalue problem. In fact this is a nonlinear equation because the Kohn–Sham Hamiltonian (matrix) $\mathbf{h}_{\mathrm{KS}}$ depends on the solution vectors $\mathbf{c}_k$ as the Coulomb potential and the exchange-correlation potential depend on the electron density $\rho(\mathbf{r})$. Therefore, Eq. (1.7) has to be solved iteratively in the SCF cycle, compare Chapter 8. Still, it is possible to store several types of integrals to reduce the computational effort.

Depending on whether the spin is considered explicitly, orbitals can be occupied with one, two, or zero electrons each. The orbitals are divided in occupied and unoccupied ones, by selecting in which orbitals the electrons should be put. It is possible to choose different occupations of the orbitals, given by the occupation numbers n. The ground state is defined as the combination with minimal total energy. The occupied orbitals are further used to build the density matrix $\mathbf{D} = \mathbf{C}n\mathbf{C}^T$. (Here, n is a diagonal matrix that holds the corresponding occupation numbers.) Matrix **D** represents the electron density in terms of the orbital basis.

To avoid calculating four-center integrals (using the density created from the orbitals directly) one often employs an approximate charge density $\tilde{\rho} = \sum h_i u_i$ represented by its own basis vectors u_i. The affected integrals are reduced to three-center integrals, decreasing the computational effort required for generating the Kohn–Sham matrix. The charge fit coefficients h_i which minimize the Coulomb norm of difference between the density of electrons and the approximated density expansion in the auxiliary basis are used to efficiently construct the Coulomb contribution to the KS-matrix.[5, 17]

The simplest expression for calculating the charge fit coefficients is derived from minimizing the Coulomb norm of the density difference $\rho - \tilde{\rho}$:

$$\sum_j G_{ij} h_j = \int d\mathbf{r}_1 d\mathbf{r}_2 u_i(\mathbf{r}_1)[\sum_{k,j} \chi_i(\mathbf{r}_2)\chi_j(\mathbf{r}_2) c_{ik} c_{kj}] / (\mathbf{r}_1 - \mathbf{r}_2) \tag{1.8}$$

This equation uses the matrix **G**, with the following elements:

$$G_{ij} = [u_i \mid u_j] = \int d\mathbf{r}_1 d\mathbf{r}_2 u_i(\mathbf{r}_1) u_j(\mathbf{r}_2) / (\mathbf{r}_1 - \mathbf{r}_2) \tag{1.9}$$

Additional terms might be added to the right hand side to ensure that both ρ and $\tilde{\rho}$ integrate to the same number of electrons.[20] Furthermore it might be preferable to determine only the change in the charge fit coefficients δh_i to the preceding iteration h_i^{old}.[21]

1.2 Parallelizable Tasks in a Quantum Chemistry Program

This section discusses some tasks of a quantum chemistry program which can be easily and efficiently parallelized. The integral parts described in Section 1.1 are well suited for being processed in parallel. There are usually many integrals. Individual analytical integrals can be calculated fast, thus it does not make too much sense to handle them separately by a distribution procedure over the processes. Therefore their evaluation is combined in "batches" which contain several of them. The batches correspond to tasks for the processes, taking various amounts of time.

The integrals, which are calculated numerically, also comprise several tasks. A certain function has to be evaluated on each grid point, and the resulting values eventually have to be summed up to obtain a complete integral. This procedure is also organized in "batches" which comprise the evaluation of the function and the summation of the resulting values on a block of grid points. Blocks are chosen of equal length, except for the last, smaller block.

The number of tasks in the various integral parts depends on the chosen accuracy of the calculation (especially for the numerical grid integration) and the size of the system (for the analytical integrals the k-center integrals scale with N^k for N atoms). The time of computing these tasks can be a significant part of the overall computation time, depending on the chosen basis set, the approximation and, especially, the system size. For Pt_{38} in Subsection 6.6 the four-center integrals require already about 98% of the time for the SCF cycle.

Every task of the analytical or numerical integral parts is independent of the others and can be calculated on any of the cores. ParaGauss calculates them already in parallel, the numeric integration with a method, distributing them evenly to the processes, and the two- and three-center integrals with a master-slave concept, see Section 2.2.[22] During the work of this thesis, these original parallelization concepts of the various integral parts have been replaced by a newly implemented dynamic load balancing strategy, using a work stealing algorithm, see Section 2.3. The four-center integrals, which recently were introduced in ParaGauss,[11] have been parallelized with this strategy from the beginning.

The next section (Section 1.3) gives an overview of how quantum chemistry programs handle parallelization, especially for the integral parts. In Chapter 2 various parallelization strategies are introduced. Chapters 3 and 4 describe the newly implemented strategy in detail, starting with an overview of important facts for the realizations of the different variants in Chapter 3 and describing details of the realization in Chapter 4.

1.3 Parallel Processing in Quantum Chemistry Programs

The trend of quantum chemistry calculations is directed towards increasing system sizes. Additionally, more complex and larger basis sets promise results which are more accurate.

To calculate large systems with quantum chemistry codes using an increasing number of cores is essential. Although large algorithmic differences exist, quantum chemistry codes follow in principle similar schemes. The Hartree–Fock method and the density functional theory face the same challenges related to parallelizable tasks. Both methods contain an SCF cycle and require the calculation of multi-center integrals. These integrals are quite demanding with respect to computer time and therefore parallelization for them promises a good speedup.

Quantum chemistry codes also contain several parallelizable parts related to the SCF cycle, for which different strategies have been tried.[23] Several communication strategies are used for quantum chemistry codes, like parallel virtual machine (PVM)[24] or OpenMP.[25] But most methods nowadays usually rely on MPI which provides a widely available standard.[26]

Serial codes[5, 27] have been replaced by new codes that explicitly consider the challenges of parallelization.[4, 28] For other programs parallel implementations and their performance also have been reported.[29-33] Some codes were developed to run on a specific computer architecture, like low-power 64-bit accelerator technology of ClearSpeed running in parallel with the host CPU[34] or when graphic processing units (GPUs) are used to calculate the Coulomb potential.[35] Ramdas et al.[36] even suggested developing a special-purpose computer for Hartree–

Fock programs, considering the special challenges of these programs. Taylor and Bauschlicher[37] discussed already in 1987 about how quantum chemistry codes may efficiently work in parallel on CRAY computers of that time.

Standardized interfaces[38] might help to exchange easily both, the method how to execute the tasks as well as the way the tasks are distributed. However these interfaces are not that useful for existing codes, as they would require large restructuring. Many quantum chemistry codes distribute the tasks statically[39-41] or with a master-slave concept.[42-44]

Regarding the need of growing numbers of cores and limits of the methods mentioned (see Sections 2.1 and 2.2), it makes sense to take a general look at methods for distributing tasks. Load balancing, thus distribution the load due to the task resource requirements on the processes, can be achieved in a number of ways.[45-46]

Another point is related to the question of storing or recalculating the integrals, required for the SCF loop.[47] Which strategy is more efficient depends on the computer architecture. In our case it might be required that at least some of the integrals will be recalculated in every cycle. This is particularly true for four-center integrals because they (formally) scale for N atoms with N^4.

2. Parallelization Strategies

The parallelization strategies are used to distribute the tasks on the processes if the calculation is done in parallel. There are different kinds of tasks. The expected time requirement of each task might be unknown. However for the parallelization strategies it is usually advantageous if they are at least known to a certain degree. In this regard it is most interesting how the time requirements of different tasks are related to each other. The easiest approach is when the tasks require all the same time but parallelization strategies can also benefit from an uneven expected time requirement. The numerical integrals, already mentioned in Chapter 1, can be handled by tasks which all are expected to take (essentially) the same time. The analytical integrals of Chapter 1 have different time requirements per task and times can only very approximated be estimated. Often, only the information is available which of two tasks is expected to take longer.

Parallelization is a broad field which cannot be covered completely in the present discussion. This work discusses parallelization strategies which can be used for quantum chemistry programs.[1] Parallelization strategies which will be further explored should be general enough to be used for different parts of ParaGauss. The two-electron integrals, which are some of the most demanding tasks for parallelization of ParaGauss, are the first application of the new strategies. The newly implemented algorithm is a work stealing algorithm, see Section 2.3, which tries to be only as complex as required to execute this kind of tasks, but otherwise remains as simple as possible.

Several existing libraries for parallelization, like Scioto,[48] include already code for this purpose. They usually provide additional complexity to our requirements, like the possibility to create new tasks at runtime. The disadvantage of these libraries is that in order to run correctly they require large restructuring of existing code. Rather it is desirable that the library can be called by the code upon request to provide identifiers for the tasks. However, in fact the library demands that the tasks are called as functions by their own code. For example, the tasks must not involve any global variables. As the algorithm is supposed to be used for an existing huge code, ParaGauss,[2, 4] integrating such kind of work stealing libraries is rather difficult. Furthermore using such an existing library introduces many restrictions on further extensions of ParaGauss, which require work stealing, like the already mentioned ban on global variables. Therefore it was decided to create a new work stealing library especially for ParaGauss, which features a simple interface and can be easily integrated into existing code.

Parallelization was included in ParaGauss from the beginning but the various parts were parallelized by separate strategies. This makes it possible to consider only selected parts at a time, having to deal with a bunch of independent tasks instead of several parts with dependencies allowing one to concentrate on one part at a time. There are some further simplifications related to the tasks of the new library. The number of tasks is known before processing starts and stays fixed during the whole calculation. All tasks are of the same kind; the data they require are available for every process, thus there is no preference for some tasks to special processes. There is no order required, in which the tasks have to be performed; in fact, there is no dependency at all between them. The results of all tasks have to be present only at the end of the corresponding program part, the so-called overall task. Therefore the processes have to wait for each other, justifying to look at the overall task separately without bothering what should happen afterwards. The parallelization library does not exchange the results of the tasks; instead the tasks themselves include distributing the results, which are collected after all the tasks have been finished. These restrictions are common to quantum chemistry programs but do not apply to the programs examined by computer scientist.

All parallelization strategies, which will be discussed, share the following general interface. The software allows executing each task by every process, as every process has the start information for all tasks. In the following a “task ID”, an integer number, is enough to identify a specific task. These identifiers are also the only information required to be interchanged between the processes (except some data related to the termination of the parallelization algorithm). To simplify the notation the task ID will not be mentioned explicitly. Thus "stealing a task" means to steal the ID of that task, while executing a task always implies that a process has to select the task belonging to a given task ID. In the following the processes will be called worker. A worker might share some resources with other workers. It might further consist of several parts, run on several cores or comprise several threads.

After considering parallelization strategies in general the following section describes the basics for various relevant parallelization strategies.

2.1 Static Distribution

The simplest way to distribute the tasks for parallel calculation is to distribute them in advance and let every worker run its own share. With every worker knowing the number of tasks and workers as well as its worker ID (the process ID provided by MPI) it is possible to proceed with using the communication exclusively for sharing the results. This completely removes the communication overhead due to distributing the tasks, independent of the number of workers the tasks are performed on. The essential overhead of the part containing these tasks is due to the imbalance of the time spent for working on the tasks by the different workers. This strategy cannot be beaten if the time requirements (relative to each other) for every task are known and used to create a distribution where all process have as much as possible the same time requirement for their tasks. For example if all tasks are supposed to be of the same size an ideal distribution would be to have the same number of tasks on every worker. In cases where the time requirement per task is not known this strategy cannot be used efficiently and for approximated time requirements but of different size the quality of the approximation is important. But even for ideal tasks the computers on which they are running might prevent that the expected time requirements represent the real required time. This is for example the case for heterogeneous clusters, where machines with different performance are used. The machine itself might take some of its CPU time for its own processes resulting in different load for the processes, causing imbalance. If not all machines or CPUs are affected in the same way, this results in further imbalance. In all these cases a static distribution is not able to react and is forced to run with idle time on the faster workers. Therefore it is desired to have some kind of regulation of the task distribution, which considers the current status of the machines as well as the imbalance between the time requirements of the various tasks.

With static distributions, one has to take care that the load on all workers is as similar as possible. The tasks can be handled by a round robin method, which distributes the tasks in turns.[49] For an implementation of the "single-determinant" (or "exact") exchange of a density functional theory hybrid method the static distribution was first balanced using a cost approximation for the expected task time requirement and starting with the second SCF loop by using real costs of the previous iterations. This way Guidon et al. achieved good performance also for batches of different computational cost.[50] Also the three-center electron repulsion integrals, which have different sizes, might be redistributed after every loop, as done by Calaminici et al..[51] Even if the distributed tasks are well balanced, either by the mentioned strategy or by choosing a method were all tasks are of the same size, like hierarchical cubature for the exchange-correlation matrix[52] it cannot be prevented that idle time is left, as the distributions cannot be adapted to external circumstances such as varying computational resources.[39]

The static distribution cannot be used efficiently if the time requirements per task are uneven or unknown. The strategies of Guidon et al.[50] and Calaminici et al.[51] are only useful if the tasks are recalculated in every iteration. In ParaGauss the two- and three-center integrals are only calculated once for the SCF-cycle in a pre-SCF part. As the expected time requirements are known only very roughly the static distribution cannot be used for them.

2.2 Master-Slave Strategy

In the master-slave concept next to the workers (often also called "slaves") which execute the tasks, a master (e.g. a process or a thread) is used to distribute the tasks of the part among the worker. The master can be one of the workers, which does the master task as some extra work, or it can do exclusively the distribution. In the simplest realization every worker who runs idle sends a message to the master requesting new work. The master returns the ID of some tasks which have not been computed yet. This method has been used for several parts in the original implementation of ParaGauss, for example the distribution of the two- and three-center integrals.[3-4, 6]

The advantage of this strategy is that the workers know all the time where to get new work. It is also very easy to determine when all the work is completed, which is the case when the master cannot provide further tasks. The disadvantage, on the other hand, is that the master can easily become a bottleneck for large core numbers or small time requirements for single tasks. This might even be the case, when he does not work himself, but only distributes the tasks for the others. In this special case there is the additional disadvantage that one of the workers is effectively removed. It is possible to compensate parts of the drawback, for example by sending several tasks at once, thus reducing the granularity, which will unburden the master a bit. It is also possible to distribute a part of the tasks statically before the balancing starts, thus removing the large bottleneck at the beginning, when all workers need tasks at the same time. Still this cannot completely prevent the bottleneck, which will still appear if the number of cores is large enough. Thus for only a small number of workers the master-slave concept might be a valid approach but if one aims for several hundred workers simultaneously a different strategy should be applied. Other parallelization strategies follow a more decentralized approach, which should remove these limitations.

The performance of master-slave variants for distributing tasks is influenced by a large number of factors, like the choice of the order in which the loops over the indices of the multicenter integrals are calculated in parallel.[53] The master can also be represented by a global counter,[54-55] which might be used only for the two-electron integrals.[56] Master-slave variants can easily reach a communication overhead. For example Ferrighi et al. found that their Hartree–Fock code DALTON usually scales well up to 64 cores but for some examples generates already notable communication overhead.[42]

2.3 Work Stealing

In a work stealing algorithm all processes are allowed to steal tasks from other workers to get tasks to work on. In our case they are allowed to steal from all other processes. There is no more a central instance responsible for distributing the tasks.

A work stealing algorithm starts with a static distribution of all tasks to be treated. Only when one of the workers runs idle, it tries to get new tasks. Therefore the algorithm is supposed to be stable,[57] i.e. the sum of the load of all cores is bounded from above for all core numbers. In the master-slave method it was obvious where the worker has to look for new tasks, but for the work stealing process a different algorithm has to be used. A simple approach is that a worker randomly selects one of the other workers. It is a disadvantage that nothing prevents many or even all workers trying to steal at the same time from the same source. This should be seen in relation to the small probability that this happens and to the fact that no further communication is required for the workers to arrange who tries to steal from whom.

Kumar et al. compared this strategy to several other stealing approaches by Kumar in theory and experiments.[58] They found that the strategy of random choice of other workers works well on various machines. There were two other strategies, which also were performing well. One of the methods considered the specific computer architecture, on which the test were done, by stealing only among its nearest neighbors. The third strategy (called GRR-M)[58] was performing similarly well, requiring a more complicated algorithm which reduces the chance that several cores want to steal at the same time from the same core by using a global counter, for deciding from which process to steal next. This was combined with a message combining strategy for the requests, considering the specific hardware, it should run on. As the intended implementation should be as simple as possible and be usable on different computer architectures, the strategy of randomly selecting the other worker was used in the present case to find the source for stealing.

The main challenge for the work stealing algorithm is to detect when all tasks are finished. Even if none other worker was found to be able to provide new tasks, this does not mean that there is no other worker with pending tasks somewhere. Also, it is required that a worker, who found out that there are no tasks left, is still present to handle requests from other workers who have not yet found out that all tasks have been finished.

3. General Considerations of Implementation of Work Stealing

The main issue is that workers (as source of a stealing request) should be able to steal pending tasks from other workers (as targets of stealing requests) while these workers are working. In principle it is possible to create an interrupt routine for the targets which checks whether some sources want to steal something and then returns to their own work. However this would complicate the interface. Creating a work stealing with an interface which does not require such an interrupt routine is nice for inserting it into any existing code and makes it highly reusable. Yet, it demands a better way than by interrupt routine to detect whether some source worker is trying to steal from a target as checking this only when new tasks are taken for the target makes the stealing very unpractical.

Two solutions of this issue were considered: remote memory access (RMA) objects (Section 3.2) and multi-threading (Section 3.3). The first one can, at least in case of a suitable computer architecture, avoid the interrupt of the target of the stealing, while the second one moves the duty of finding an appropriate time for interrupt to the thread scheduler. Both methods have advantages and disadvantages, depending on the computer architecture, the MPI implementation, and the software environment. The methods are easily exchangeable at compilation time. The library, which contains these routines, will in the following often be referred to as dynamic load balancing (DLB) library. Dynamic load balancing is an umbrella term, which implies work stealing.

The work stealing algorithm requires a lot of communication between the processes. Due to the wide availability of implementations, which follow the standard of the Message Passing Interface (MPI), the implemented code builds on features of this standard, more precisely on features of the MPI standard Version 2.2.[26]

3.1 General Interface

The main routines of the work stealing library are dealing with a part containing several tasks. This overall part is finished after all the tasks contained have been finished. The tasks are assigned a consecutive number, the task ID. These task IDs are used for the interface with the work stealing library. They can be "loaded" into the library — for this the library requires only the number of available tasks — and are provided on request to the workers ready to work on the related tasks. The worker can request only one task or *n* tasks at once. In the latter case it will get always up to *n* tasks, the number can be smaller than *n* if there are less task available on the current worker. Zero tasks will be returned by the DLB library when that overall part has been completed. The "stealing" of tasks is done internally in the DLB library. To simplify things, task IDs are always processed sequentially. Every worker has at most one interval of consecutively numbered tasks available. The worker is only allowed to start stealing when its storage is empty.

```
call dlb_init (MPI_COMM_WORLD)          ! initialization

call dlb_setup (N)                      ! start a run of DLB
                                        ! with N tasks
  do while [dlb_give_more (n, tasks)]   ! request for up to
                                        !    n new tasks
    do i = tasks(1) + 1, tasks(2)
      call work(i)
    enddo
  enddo

call dlb_finalize ()                    ! termination
```

Scheme 3.1: Example code of a calculation with DLB. The four functions of the DLB interface are shown: dlb_init(), dlb_setup(), dlb_give_more() and dlb_finalize(). The actual work on task *i* is depicted as work(*i*).

The program, for example ParaGauss, using the DLB routines needs to use only four functions, cf. Scheme 3.1. Initialization, dlb_init(), and termination, dlb_finalize(), of the DLB code ensure that the resources are only available when they are really required. Several initializations and terminations are possible during execution of one program, but it is also possible to call several times a task distribution of DLB between initialization and termination of DLB. Task distribution of DLB is started with the setup function, dlb_setup(), which passes the number of tasks to the DLB library. DLB then distributes the tasks statically and prepares everything to allow stealing. Another function, dlb_give_more(), is used to request one or more tasks for a given worker. The worker provides the tasks IDs (required for identifying the tasks) as an interval (a,b), representing all task IDs k with $a < k \leq b$. In case there are no further tasks to provide the worker waits until it was able to steal some or to detect global termination. The return value of dlb_give_more() indicates termination.

Generally, the tasks are evenly distributed over the workers at the beginning. That does not require any communication as every worker knows the total number of tasks and workers available. This allows a static distribution of all tasks at setup, without any communication. For test purpose, a different distribution of tasks at the beginning might be used, but this is not part of the production version. The tasks with the smaller task IDs are returned first. This is the case for the workers taking tasks they will process and for the stealing from another worker. The order in which the tasks are executed has an effect when the tasks are of different size and ordered by the supposed size. The workers steal always about half of the other workers tasks.

3.2 Using Remote Memory Objects

The principle idea of using remote memory objects for the work stealing algorithm is straight forward. Each worker assigns a globally accessible remote memory access (RMA) object and stores all its task IDs into this object. When a worker starts to work on a task, it removes the corresponding task ID from the RMA object. The workers can get tasks from any of the RMA objects, thus “stealing” some from another worker.

MPI-2 supports remote memory access routines. However, the realization and the quality of implementation of these routines varies between MPI implementations. One potential difficulty is an efficient implementation of the MPI lock for the remote memory area, thus the feature which allows other processes to read and write exclusively on this area. The MPI standard allows that MPI implementations use the following restriction to simplify the implementation: the owner of the RMA-area may need to perform some MPI routines at the time, when another worker gets the lock of the RMA object.[26] Using this restriction is especially tempting for single-threaded implementations, like the OpenMPI implementation.[59] This would be hardly better than waiting for the other worker to enter the dlb_give_more() routine, unless the actual task processing also involves MPI communication.

A challenge for the implementation is due to the way the routines are realized in the MPI library. It is impossible to implement a read-modify-write operation using the MPI routines because the RMA area of one worker can be locked by any of the workers. While a worker holds the lock it has exclusive access to the area by using the routines MPI_PUT(), MPI_GET() and MPI_ACCUMULATE(). However it is illegal to access the same part of the

RMA area with different routines during one access epoch, for example between the calls of MPI_WIN_LOCK() and MPI_WIN_UNLOCK().[26] This prevents the read-modify-write() routine and makes it impossible to create a straight forward implementation of the RMA variant.

Despite these complications, the RMA variant is conceptually still rather simple. Some computer architectures provide hardware support for the RMA, which is easy if they have a central memory. One of those machines was the HLRB-II of the LRZ computing center.[60] The system was recently replaced, but tests on HLRB-II indicated that the RMA variant is very effective on such a type of machine, see Subsection 5.1.1 and Chapter 6.

3.3 Multi-Thread Variant

For the multi-thread variant one helper thread is created which handles the stealing and termination part of the algorithm, while the main thread does the actual work. The other thread solely cares about the work distribution, sending on request from other workers' helper threads messages with stolen tasks. The communication regarding the distribution of tasks can be better steered when one uses several threads per worker. The thread management is done automatically by the operating system. The main thread is automatically frequently switched to the helper thread, thus allowing work stealing to interrupt the work. No interrupt routines have to be placed into the task processing code. The threads are realized by POSIX threads which are widely available and standardized.

As the threads sometimes need to use the same resources, e.g., the storage area for the tasks belonging to the current worker, one has to consider thread safety. Thread safety means, for example, that it has to be ensured that at a given time only one of the threads accesses the shared memory and that the threads do not work with data which has been changed, unnoticed by the other threads. This is also required for the MPI routines, especially as working on a task usually includes distribution of the results with MPI routines. Thus even if the routines of the work stealing process would ensure thread safety for their own routines, one still has to consider that the main thread might include arbitrary MPI communication. The MPI-2 standard introduces four thread safety levels, depending on which threads carry out MPI communication (e. g., at the simplest thread level only the main thread of each process is using MPI communication while at the highest thread level all threads can use MPI communication at the same time). However, MPI implementations are not required to provide all thread safety levels. If it is requested to initialize MPI with an unavailable thread level, it may provide the routines of another, higher or lower, thread safety level. Therefore having an MPI implementation which follows the MPI-2 standard does not guarantee thread safety under all circumstances. Some implementations provide the thread safety for hybrid programming in separate libraries to link the program with,[61-62] or by separate packages.[63] These separate routines are not always as well tested as the single threaded ones, they might also not yet be optimized for good performance.[64] Thus the thread-based DLB is not automatically superior to the RMA variant, Section 3.2. It is not even assured to work with arbitrary MPI-2 implementation, while the RMA variant should always be able to work correctly, even if not always effectively.

3.4 Termination Algorithm

The termination detection can be implemented in the same way for both the RMA and the multi-threaded variant.

Termination is detected by comparing the amount of finished tasks against the complete number of tasks. Besides the number of tasks a worker has still to do, it keeps the number of tasks it has already done but not yet reported. The parts corresponding to completed work are reported when a worker gets passive by having finished the last of its current task list. The number of finished tasks has to be collected somewhere.[65] One of the workers, called termination master in the following, is selected for this additional task. But if all finished tasks are collected at this worker there will be a large amount of additional work for it, which might cause imbalance. Therefore finished tasks are collected in a two-step approach. As the algorithm starts with the tasks already distributed, each of the workers can collect the reported tasks for its share of tasks. Therefore, finished tasks will be first reported to the worker for which they were originally assigned. When a worker detects termination of its share, it reports this in the second step to the termination master. The global termination is reached after each worker has reported termination of its share. Subsequently all workers will be informed by the termination master that the end of the overall task has been reached.

The algorithm is a modification of the so-called credit distribution and recovery algorithm, which was designed for a more general case.[65] The original description mentions a challenge which does not appear for the current case as it is related to the fact that tasks might create new tasks during run time.

4. Details of the Implementation

4.1 Remote Memory Variant

In the RMA implementation of the work stealing algorithm the principal idea, as described in Section 3.2, is that a globally accessible RMA object contains an interval with the task ID range, which have not yet been selected for calculation. The tasks are put into the storage in the dlb_setup() routine, cf. Section 3.1. The termination detection and the stealing are done in the routine dlb_give_more(), which has the main duty to provide valid tasks to the calling process. Scheme 4.1 describes how this routine is implemented. The process first tries to get some tasks from its (own) local storage. When there are no more tasks in storage it starts stealing from others. To detect that it can start to steal, it is required to use an "unsafe" read. If the number of stolen tasks is larger than the requested number, the remaining tasks are written back into its own storage (by an "unsafe" write). Notice that a try_read_modify_write() function has to be used for getting tasks, as a read-modify-write cannot be reasonable implemented, see Section 3.2. The unsafe functions are required for the same reason. The safe and unsafe functions, and especially the user lock required for their implementation, are described in detail in Subsection 4.1.1. That subsection contains also the explanations for the special requirements for these routines.

```
procedure dlb_give_more (n, tasks)
   do while (own storage is not empty (found out by unsafe
                           read))
    try_read_modify_write (own storage, up to n tasks
                           to work directly on)
    if trial succeeded
      exit loop
    endif
  enddo
  if no valid tasks provided
    do while (no termination detected)
      select a target to steal from
      try_read_modify_write( target, tasks)
      if stealing had succeeded
        exit loop
      endif
    enddo
    if there are valid tasks
      abstract up to n tasks to work directly on
      put remaining new tasks in own storage (by unsafe
                           write)
    endif
  endif
  return whether there are new tasks and the tasks to
                           work directly on
```

Scheme 4.1: The RMA variant of dlb_give_more(). It requests up to *n* tasks (provided as input) at once.

4.1.1 The User Lock

In the RMA implementation of the work stealing algorithm, there are three routines which handle the access to the ID list in the RMA area. Two of them are so-called “unsafe” routines where the worker reads or changes the content without having to respect the user lock. The third routine corresponds to trying to take some tasks out of the storage, either through stealing or through local access (by accessing own memory) to start calculating them. This routine corresponds to a try-read-modify-write routine. These three routines are required for an efficient implementation of the dlb_give_more() function, see Scheme 4.1.

A so-called user lock on the storage area is used to ensure that always only one worker holds this lock and is therefore allowed to change the contents of the locked area. It is implemented by expanding the RMA area of the task storage with a user lock area. This area contains an integer number for every worker differentiating between “on” and “off”. It is required to have storage for every worker as the worker will have to write into its own storage and read the storage of all other workers at the same time. This is only possible if storage sections do not overlap, compare Subsection 3.2.

The user lock works as follows. A worker that wants to access the memory, has to get first an MPI lock. While it holds the lock it reads out the memory (except its own user lock area) and sets his user lock area to “on”. If no other worker has its user lock set to “on” the current worker got the lock and is allowed to use the content of the locked area. Having finished the current worker has to get once more the MPI lock and release the lock by setting the complete user lock area to “off”. During this MPI lock it might also write back the tasks for the storage if it has changed the content. For the try-read-modify-write() routine the worker would divide the interval of tasks it got from the storage and write back the interval of the tasks which it has not stolen. When a worker finds the user lock “on”, it retreats and tries anew. If it was for its own RMA area, it continues trying to get the same lock, otherwise it selects a new target to steal from. A similar lock has been described for file read and write access realized with MPI.[66]

The unsafe functions, unsafe write and unsafe read, access the RMA area without respecting the user lock. While any worker can do an unsafe read on the RMA area of a source, only the source itself is allowed to do an unsafe write.

4.1.2 Deadlocks and Ensuring Finalization

This subsection will discuss some details of the implementation, which are necessary to ensure that the RMA variant performs well and ultimately completes. The principal termination detection has already been described in Section 3.4. The number of finished tasks (reported by the worker who finished them) is compared against the number of initially distributed tasks. However there are some details required for the RMA variant to work correctly.

A drawback of the approach via a user lock is that the user lock does not provide fairness. A fair lock would guarantee that a worker will eventually get the user lock. This cannot even be ensured if a worker is allowed to retry as often as it wants. One might expect that every requesting worker will get the lock from the MPI routines sometimes, but this is independent of the requirements for getting the user lock. Thus, even with expecting the MPI lock from the MPI routines to be fair about deciding who gets the MPI lock, the user lock is not fair. Because MPI does not know about the user lock and every worker can only try to get the user lock, the user lock cannot be made fair.

Not being able to guarantee that one special worker will get the user lock on a special source worker is the major drawback of the implementation. It could not even be ensured for the workers to get the lock on their own task storages. The workers would be able to perform their action on their own storage within an access period of MPI to the storage but they are still forced to respect the user lock. Therefore, without the usage of the unsafe read and unsafe write functions, a competitive situation might occur which prevents the algorithm to finish at all.

With the unsafe read the workers find out if a storage is empty. As a failed remote worker will always try another target, this is only performed by the source itself, which is not allowed to try to get its tasks from other workers until its own task storage is empty. This is especially required as a worker will only report the completed tasks if the storage is empty. Then the worker attempts to steal new tasks. This is required to ensure that the overall task will terminate.

The unsafe write is related to the fact that a worker would steal more than its current required share. Then it has to allow other workers to eventually steal from its new tasks. It therefore requires access to its own storage area to store the tasks. Thus it is again possible to get a competitive situation. Other workers, wanting to steal from the source, could get all the time the user lock and the source would not be able to continue calculating the stolen tasks, as it could not write back the additional ones. Then these tasks will never be carried out. This is prevented by restricting the try-read-modify-write routine. A worker, finding a task storage empty, is not allowed to change the storage, but will only reset the user lock. Only the owner of the area is allowed to add new tasks, thus it can insert tasks without having to get the user lock (only by having the lock from MPI). It is important that the owner alone, with an unsafe write, will overwrite the task storage. The worker currently having the user lock has to reset the remaining part of the RMA area. Then the next worker, wanting to steal after the lock was released, will find the new tasks of the owner of the storage.

A worker finding only one task in a storage, of which it was going to steal from, will take this task. A worker who has stolen some tasks removes the tasks it can execute immediately before writing the other tasks into its own storage. This is required to prevent that a task is stolen by another worker without ever being executed. Thus if a worker got some tasks, irrespective of the origin, it has to process at least one of them immediately.

4.2 Multi-Thread Variant

In the simplest model of the multithreaded variant a single further thread is used. This helper thread will handle all communication associated with work stealing. Thus the main thread, the actual worker, has only a limited relation to the work stealing process, namely it has to extract new tasks from the storage object, as in a static variant. However, it is only allowed to return without any tasks when the helper thread signals global termination. Then again it has to wait, if it found an empty storage, till the other thread releases it. The helper thread handles work requests from other workers and sends out its own. In addition it sends and receives messages regarding the termination algorithm, thus about the number of tasks others have done for it and whether its own initial number of tasks are finished.

4.2.1 Thread Safe Objects and Scheduling

Several objects have to be used by both threads of a worker. The most prominent of them is the storage object that contains the IDs of the remaining tasks. A second object is the number of tasks finished since the last report. This is used by the main thread to communicate the number of tasks it has completed, information that required by the helper thread when sending the correct number of finished tasks to its owner. Both objects can be secured with the same lock to avoid deadlocks, which could happen in cases where several locks are used in different order.[1] The lock secures two additional flags which contain information about the current status of one thread required by the other thread. One is used by the helper thread to communicate the termination to the main thread. The other tells the helper thread whether the main thread is active or just waiting. This is required for the management of the time distribution for the two threads. Both flags are only set by one of the threads and read by the other, while the other objects are read and written by both threads. It is crucial that these objects are in the lock area. For example, when the main thread finds no new tasks in the storage, it checks for termination. Then an object of POSIX, called a condition, is used to send this thread in the background until the helper thread signals that the condition is met. The helper thread releases the main thread from waiting when it found new tasks or after it set the termination flag. If the helper thread would be allowed to access the termination flag without lock the two mentioned actions might just happen after the main thread checked the flag but before started waiting. Then the main thread would wait forever, as the helper thread has terminated afterwards.

It is not obvious for the thread scheduler, handling the switching between the threads, whether the threads need to work on an equal footing or one of them should be favored. In fact the main thread should be favored in cases where there is still work left while the helper thread should be favored after the worker runs out of tasks. The latter can be achieved with a condition, provided by the POSIX threads. The worker thread can wait till the helper thread signals that the condition has been met either by having found new tasks for the main thread or by having learned about global termination.

```
procedure dlb_give_more (n, tasks)
  lock mutex
    do while (no valid tasks and no termination)
      try to get up to n tasks from storage
      if got no tasks
         set flag to tell helper thread that worker thread
              is waiting
         condition function
           do atomically
             unlock mutex
             start waiting for call of helper thread
           end atomically
           receive call of helper thread
           lock mutex
       endif
     enddo
   unlock mutex
   return whether there are new tasks and the tasks
    to work directly on
```

Scheme 4.2: Thread variant of the procedure dlb_give_more(). The lock of the POSIX thread library, which can be used for the threads access to some variables, is called a mutex. The procedure requests *n* tasks at once.

The test if there are messages to be received is done with a non-blocking MPI routine, thus the tasks of the helper thread in one of its cycles are easily achieved. The helper thread then uses a sleep command to wait for an appropriate time before he starts all over again in cases when the worker thread did not signal (by an object secured by the lock), that it has no work left. In this case the helper thread would remain active and increases the time to check for messages about new tasks. The best duration the helper thread has to wait may depend on the sizes of the tasks and the computer architecture; it may have to be adjusted for a given computing platform. For the machines, used for the performance tests in the Chapters 5 and 6 a time of 0.001 s was providing reasonable results. The dlb_give_more() function is sketched in pseudo-code in Scheme 4.2.

The variant can be further modified to work with blocking MPI routines. However, as tests in Subsection 5.1.1 will demonstrate, the variant with non-blocking MPI routines is the superior one of the two. Therefore, unless stated otherwise in the following, the thread variant will refer to the variant just described.

4.2.2 Deadlocks and Ensuring Finalization

As there is only one lock used in the thread-based implementation of the algorithm, in order to prevent deadlocks of the lock, worker and helper threads only have to ensure that they release the lock before they go into the background.

A second implementation detail, required for avoiding deadlocks and communication loops, is to prevent that the last task in storage is not stolen. Different from the RMA variant, the thread variant is allowed to do this as POSIX ensures that the main thread gets the lock to the thread save area at some time. This prevents the situation where a task is stolen mutually without ever being executed. As helper threads and main threads compete for the lock of the storage, it would be possible, in principle, that always the helper threads get the lock when the last task is in the storage and the main threads get the lock after the helper threads have allowed other helper threads to steal the task. Reserving the last task might not be ideal as the worker to whom the task currently belongs might just have started a long task. However other solutions to this challenge, like ensuring that a task might be stolen only a given number of times, would be much more complicated. As the helper thread works independently of the main thread, it is also not possible without complicating the algorithm to ensure that the worker will work at least on the first of the stolen tasks, as it was done in the case of the RMA implementation, see Subsection 4.1.2.

4.2.3 Ghost Messages

After the termination has been detected, there is still some communication ongoing, primarily due to the workers that are still trying to steal from each other. It is not possible just to abandon pending communications in a reliable fashion, as some of the messages might be just "on the way" and therefore cannot be canceled. These messages might be received later on, without the sender expecting them. These messages are called "ghost messages". All the affected messages belong to the stealing algorithm; all other messages send by DLB were part of the termination procedure and therefore have already arrived. This challenge did not occur for the RMA variant as the one-sided stealing communication was finished automatically after termination was detected.

When the message signaling global termination arrives, the receiving helper thread may still have one ongoing request for work. To avoid ghost messages one has to make sure that every message will be received. This requires a modification in the message sending and especially for the termination algorithm. Messages, which request new tasks, go along with an identifier (an integer number). After the global termination was announced to a worker, it responds with a message to the global environment that contains identifier and target of its last request for work. The global environment collects all these messages and provides the other workers in a global communication algorithm of MPI with this information. Afterwards every worker is able to determine which other workers have sent it their last request. Having stored the identifier of the last message from every worker, a worker can determine whether the message has already been received or not. Therefore it is able to determine how many messages regarding pending work requests it has to wait for. After there are no more messages in transit, that belong to the of the current DLB epoch, the worker terminates completely.

5. Performance of Work Stealing

To determine the performance of the work stealing algorithm, several test examples were considered. The evaluation of the method will concentrate on a single DLB epoch on the example of the software ParaGauss. The times were measured as differences of time stamps, which use the function MPI_WTIME(), providing elapsed wall clock time.[26] Integral parts, already mentioned in Chapter 1 as application area, of three example cases were chosen.

The speedup of a parallel calculation is given by the time requirements on one core divided by the time requirements of the calculation on *p* cores t_1 / t_p. Dividing this ratio by the number of cores one gets the efficiency ϵ_p. The efficiency ideally is 1 (or 100%). This happens when the total CPU time is the same on 1 core and *p* cores and thus no loss of time occurred. This is equivalent to an ideal speedup by the number of cores used.

5.1 Plain Test with Simple Task Set

Using a simple task set is a test of the DLB routines alone, removing any effect which might be specific to a real-world calculation. However this allows one to focus on the effects of DLB and provides a better control over the tested features. To spend CPU, a task is invoked where a simple operation is carried out repeatedly many times. The operations are supposed to keep the CPUs busy, which should, for example, prevent the helper threads from using the time reserved for working. On the other hand, it is not that easy to define the time requirements for such a task. Therefore the average time requirement is different on various machines. Both DLB variants were tested on various machines for comparison, to examine different computer architecture.

With small changes to the stealing process, the thread variant can be used to mimic a master-slave variant, where the helper thread of one worker is doing the master's work. In fact, at the beginning the complete task interval is assigned to one worker and all other workers steal exclusively from this worker. A smaller fraction than usual is stolen: instead of sending half of the available tasks, the master only sends a fraction of 1/*p* of the available tasks. Instead of the usual termination algorithm the helper thread, which is the master, sends the global termination message if it cannot provide further tasks. This method is not the same as an optimized master-slave method and does not correspond to the master-slave method applied in ParaGauss, but it should be able to provide a trend for such an approach.

For testing purposes it is also possible to start with the worst possible initial distribution of the tasks. For this, all tasks were put on a single worker instead of evenly distributing them. In the current example there are no MPI operations other than the ones of the work stealing algorithm.

5.1.1 Tests on HLRB-II

The supercomputer HLRB-II is an SGI Altix 4700 machine.[60] It has 512 cores per compute partition. The connection was realized via a NUMAlink 4 with a bandwidth of 6.4 GByte/s. The MPI version, given as default and used for the current tests, is the MPI implementation of ParaStation.[63] On average a task required 0.15 seconds.

The two DLB variants were tested with both, the best (which is also the default) and the worst distribution of the tasks. Additionally a second variant of the thread variant, using blocking MPI routines (these MPI routines will be only finished when a message has been arrived) was tested on this machine. For this method the MPI routine has to decide when and for how long the helper thread has to wait for a message to arrive. A second helper thread is used for sending the task requests which is steered completely by waiting for conditions. Unfortunately even for a multi-threaded implementation, the blocking routines can be aggressive in a given MPI implementation. Thus, they will check as often and as long as possible whether they find anything to receive. In this way, they will find new messages as soon as possible, but the price is that the thread with the MPI call spends a lot of process time. MPI routines optionally can be less aggressive. For example they can perform a system call when no message was received, this can allow another process to use the CPU.[67] As default this option, making the polling passive, was used for the thread method. The thread method with blocking MPI routines can also mimic a master-slave variant.

The elapsed times on one processor for all calculations are summarized in Table 5.1. The standard deviation of the elapsed time over 5 runs was for all process numbers below 2 seconds. However, for very short overall timings this uncertainty was larger than the value itself. Therefore the values have to be considered as information on a trend. Only for the calculations with 1024 tasks the standard deviation was nearly always below 6%. The elapsed times for up to 64 cores were used to calculate the efficiencies, depictured in Table 5.1. The RMA variant shows nearly almost perfect scaling (Table 5.1). There is hardly any difference between the cases with worst and best starting distribution. Thus, this variant redistributes the tasks efficiently on this kind of computer architecture. Of the variants with threads the one with non-blocking MPI routines surprisingly is more efficient. While the variant with blocking MPI routines shows differences between the performance of the scenario with best and worst starting structure, these effects are hardly visible for the variant with non-blocking MPI routines. Hence, MPI, if set to passive polling on blocking operations, is not very efficient in finding the messages. It is more efficient to have a loop, which is set by the user to passive with help of some wait functions as it is done in the thread variant with non-blocking MPI routines.

Figure 5.1 shows the speedup of the RMA variant and the thread variant with non-blocking MPI on HLRB-II. The RMA variant shows essentially ideal speed-up. An exception occurs for a single result at 16 cores of the calculation with best start distribution of 128 tasks, where the speedup, 10.9, is unexpectedly small. The exception may have been due to an unfortunate behavior of the machine during the test. The thread variant with non-blocking MPI routines is performing worse. The differences between the two start distributions are visible starting from 8 processes (Figure 5.1 right panel). Only for large numbers of tasks it stays near the ideal line. This confirms that the RMA variant of DLB is the right choice on HLRB-II. The reason for the difference is most certainly the hardware support for the RMA objects on HLRB-II.

In case of the equal start distribution (best) there was hardly any stealing, thus a static variant would also produce results of the same quality than the RMA and the thread variants. A static variant might be even superior as it does not produce any communication overhead.

However if the distribution is not ideal, the thread variant and especially the RMA variant are superior as seen by the successful redistribution for the worst start distributions.

Table 5.1: Elapsed real time (on one core) in seconds and efficiency for the RMA, the thread variant (2 Th.) and the thread variant with two helper threads (3 Th.) as well as Master-slave variants to both thread variants. All timings are averages over 5 runs. The initial distribution of the tasks was either the best possible (b), with tasks distributed as even as possible, or the worst possible (w) with all tasks on one core. Calculations were carried out for up to 64 cores.

Method	Number	Time	efficiency for given number of cores					
	of tasks	in s	2	4	8	16	32	64
RMA w	1024	157.0	1.00	1.00	1.00	1.00	1.00	0.99
RMA b	1024	157.2	1.00	1.00	1.00	1.00	0.96	0.99
3 Th. w	1024	157.1	0.98	0.90	0.76	0.56	0.37	0.22
3 Th. b	1024	157.1	0.99	1.00	0.99	0.96	0.96	0.93
3 Mast.	1024	157.2	0.99	0.91	0.84	0.70	0.43	0.18
2 Th. w	1024	157.2	1.00	1.00	0.98	0.94	0.89	0.71
2 Th. b	1024	157.2	1.00	1.00	1.00	0.99	0.99	0.86
2 Mast.	1024	157.2	1.00	0.99	0.95	0.82	0.52	0.20
RMA w	512	78.5	1.00	1.00	1.00	1.00	0.99	0.97
RMA b	512	78.6	1.00	1.00	0.99	1.00	1.00	0.98
3 Th. w	512	78.6	0.96	0.84	0.63	0.43	0.26	0.14
3 Th. b	512	78.6	1.00	0.99	0.98	0.95	0.64	0.87
3 Mast.	512	78.6	0.97	0.84	0.73	0.56	0.29	0.10
2 Th. w	512	78.6	1.00	0.99	0.98	0.87	0.72	0.52
2 Th. b	512	78.6	1.00	1.00	1.00	0.99	0.97	0.94
2 Mast.	512	78.6	1.00	0.98	0.91	0.69	0.35	0.11
RMA w	128	19.8	1.01	1.01	1.01	1.00	0.99	0.91
RMA b	128	19.6	1.00	1.00	1.00	0.68	0.98	0.91
3 Th. w	128	19.8	0.88	0.57	0.36	0.21	0.10	0.06
3 Th. b	128	19.8	0.99	0.96	0.94	0.62	0.75	0.10
3 Mast.	128	19.8	0.93	0.62	0.48	0.27	0.10	0.03
2 Th. w	128	19.7	0.99	0.98	0.79	0.56	0.59	0.24
2 Th. b	128	19.7	1.00	1.00	0.99	1.00	0.99	0.95
2 Mast.	128	19.7	0.99	0.94	0.73	0.36	0.12	0.03
RMA w	32	4.9	1.00	1.00	0.94	0.97	0.92	0.31
RMA b	32	4.9	1.00	1.00	0.99	0.97	0.92	0.42
3 Th. w	32	5.1	0.62	0.36	0.17	0.09	0.04	0.02
3 Th. b	32	5.1	0.95	0.91	0.47	0.13	0.05	0.02
3 Mast.	32	5.1	0.63	0.36	0.29	0.10	0.03	0.01
2 Th. w	32	4.9	0.96	0.88	0.70	0.35	0.22	0.09
2 Th. b	32	4.9	1.00	0.99	1.00	0.99	0.38	0.10
2 Mast.	32	4.9	0.98	0.80	0.40	0.12	0.03	0.02

Both master-slave variants start to get problems with more than 16 cores and show no further scaling. Even with 1024 tasks the time requirement was growing for 64 cores compared to 32 cores for these variants. One has to consider that the present incarnation of the master-slave variant is only an adaption of the work stealing variant, which has not been much optimized. It is possible to improve it further, e.g. by removing the worker thread of the core which carries out the duties of the master. However the limitations are not completely removable. The bottleneck might be shifted to higher core numbers by these improvements but likely it will not disappear.

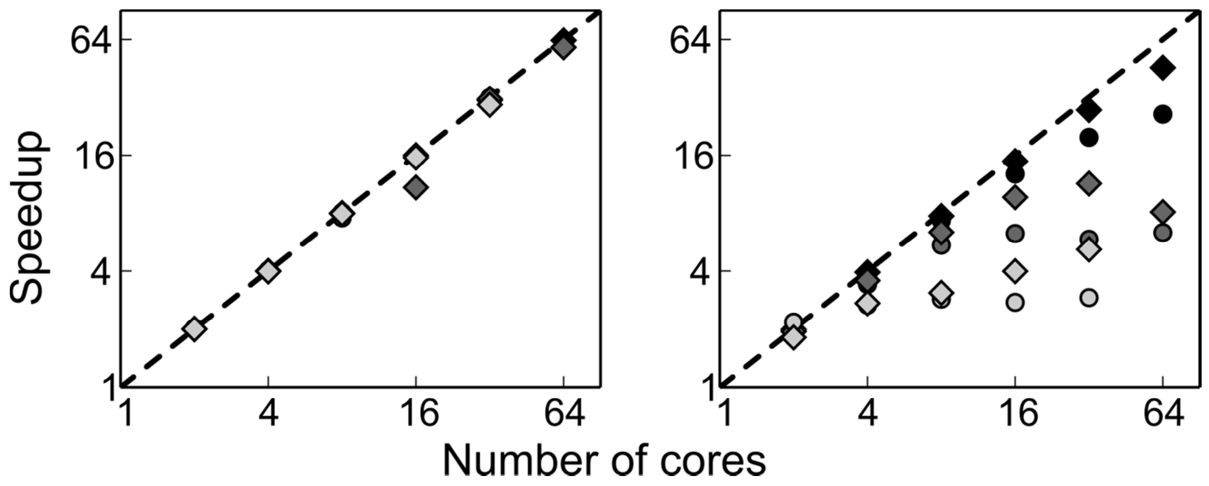

Figure 5.1: Speedup of several of the examples from Table 5.1, which were run on HLRB-II. For 1024, 128 and 32 tasks (color going from dark to light) the start distributions were taken as worst case (circles) or best case (diamonds). Left panel: work stealing variant with RMA; right panel: thread variant with non-blocking MPI routines. For orientation the ideal speedup is given as dashed line.

The difference between active and passive MPI scheduling strategies for the thread variant with blocking MPI routines can be seen in Table 5.2. For the standard setting of MPI on HLRB-II the blocking MPI routines wait for events by spinning in a loop.[67] With standard settings the thread variant with blocking MPI routines requires roughly twice the run time, except for worst-case examples on large numbers of cores. This holds especially for the case where the tasks have been initially distributed in the best way. For the worst case the limitations for higher core numbers become visible although this was not the case for the method with non-blocking MPI routines.

Table 5.2: Elapsed times (in s on one core) of the DLB thread variant with blocking MPI on HLRB-II for 1024 tasks and efficiency on various numbers of cores. The MPI was either taken as it was (standard) or the polling was made passive.

MPI	Distr.	Time	2	4	8	16	32	64
Standard	worst	314.6	0.99	0.98	0.91	0.8	0.62	0.41
Standard	best	315.0	0.99	0.99	0.96	0.93	0.86	0.71
Passive	worst	157.1	0.98	0.90	0.76	0.56	0.37	0.22
Passive	best	157.1	0.99	1.00	0.99	0.96	0.96	0.93

Of the two thread variants the one with non-blocking MPI routines can be seen to perform better. Even when the MPI routines were allowed to check for messages in an aggressive way, the speedup of the variant with three threads was still worse. The computation times with passive MPI routines are even better. The computer architecture should have an effect in the performance difference between the RMA and the thread variants. However this effect is expected to be similar for the various thread variants. Only the different scheduling of the MPI threads should affect the performance, but this effect seems always in favor of the variant with non-blocking MPI routines. The speedup of the method with blocking MPI routines should be best if the threads with MPI functions poll aggressively. However the speedup is still not comparable to the variant with non-blocking MPI, (Table 5.2). Thus, as it is possible to work efficiently with a method, where the helper thread is steered explicitly, namely the method with non-blocking MPI. Further studies will be restricted on this method.

On HLRB-II the thread variant is nearly comparable with the RMA approach. However, especially for a small number of tasks and large core numbers, the differences become notable. This illustrates the fact that HLRB-II is especially well suited for the RMA variant, having hardware support for the RMA objects.

5.1.2 Tests on a Nehalem Linux Cluster

The Nehalem cluster contains two 4-core Nehalem Xeon E5540 processors per node. The nodes were connected by a 1 Gbit Ethernet. OpenMPI version 1.4.2 was used, compiled with MPI threads enabled (to allow thread safety levels up to MPI_THREAD_MULTIPLE, the highest possible thread level). Tasks required on average 0.33 seconds.

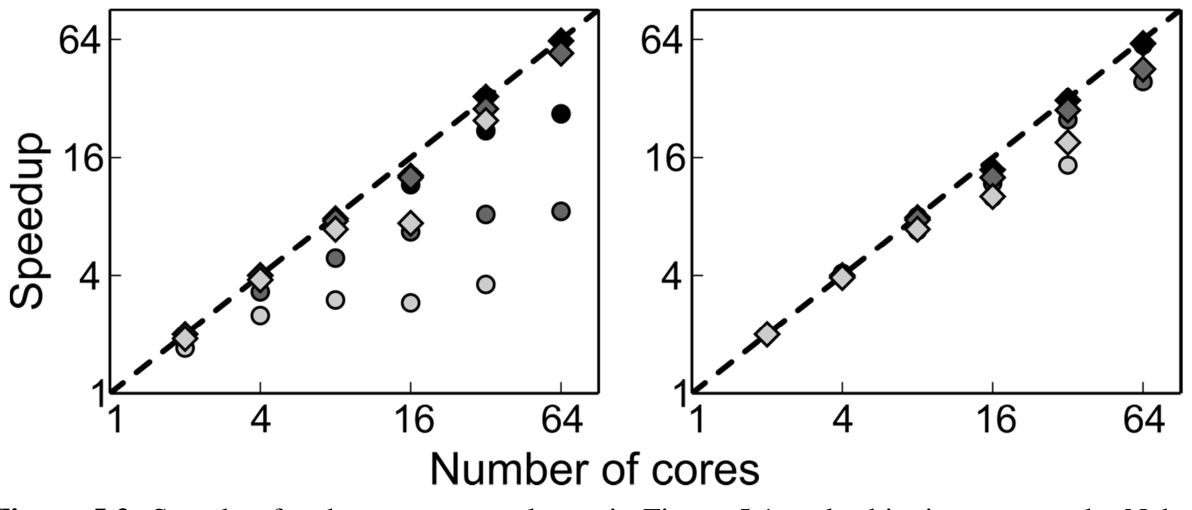

Figure 5.2: Speedup for the same examples as in Figure 5.1, only this time run on the Nehalem cluster.

The elapsed times for the calculations on one core are summarized in Table 5.3 which compares the RMA variant, the thread variant, and a master-slave modification of the thread variant. Additionally the efficiencies of 2 to 64 cores for the variants are also provided. The standard derivations of the elapsed times, used for calculation this measurement, were for all but the case with 32 tasks smaller or equal to 10%, for the two cases with highest numbers of tasks, 1024 and 512, at most ~5%.

Table 5.3: Elapsed real time (on one core) for the RMA and the thread variant as well as a Master-slave variant. All timings are averages over three calculations. The starting distribution of the tasks was either as even as possible, the best start distribution (b), or all tasks on one core, the worst start distribution (w). Times (in seconds) were calculated for 1 to 64 cores.

Method	Number	Time	efficiency for given number of cores					
	of tasks	in s	2	4	8	16	32	64
RMA w	1024	327.1	0.99	0.97	0.91	0.72	0.69	0.42
RMA b	1024	331.3	1.01	0.99	0.98	0.81	1.02	0.98
Th. w	1024	316.2	1.01	1.02	1.00	0.84	0.98	0.94
Th. b	1024	315.5	0.99	0.99	0.99	0.87	0.98	0.95
Mast	1024	313.9	1.00	0.98	0.97	0.74	0.52	0.21
RMA w	512	164.3	0.98	0.96	0.83	0.64	0.54	0.37
RMA b	512	163.6	0.99	0.99	0.96	0.79	1.00	0.94
Th. w	512	158.6	1.01	1.02	1.00	0.83	0.95	0.88
Th. b	512	159.9	1.01	1.01	1.00	0.84	0.98	0.93
Mast	512	159.4	1.00	1.01	0.93	0.65	0.37	0.12
RMA w	128	40.4	0.95	0.84	0.61	0.42	0.26	0.13
RMA b	128	42.0	1.00	0.99	0.95	0.79	0.88	0.85
Th. w	128	39.3	0.99	0.99	0.95	0.74	0.78	0.61
Th. b	128	39.6	0.99	1.00	0.96	0.79	0.87	0.71
Mast	128	39.5	1.02	0.95	0.75	0.33	0.12	0.03
RMA w	32	10.2	0.87	0.63	0.37	0.18	0.11	0.06
RMA b	32	10.1	0.96	0.94	0.86	0.46	0.77	0.06
Th. w	32	10.0	1.00	0.97	0.85	0.62	0.46	0.22
Th. b	32	9.8	0.99	0.96	0.86	0.63	0.60	0.21
Mast	32	9.6	0.97	0.81	0.39	0.12	0.03	0.01

The Nehalem cluster does not provide any hardware support for RMA implementations. Thus it was expected that the thread variant is performing better than the RMA variant, as is confirmed by the results in Table 5.3. Notable is especially the large difference between the RMA calculations with the best and with the worst start distribution of tasks. On the other hand, especially for large numbers of tasks there is hardly any difference visible between the results of the best and the worst starting distribution of the tasks for the thread approach (Table 5.3). This indicates that the RMA implementation cannot steal as fast as the thread variant on the computer architecture under discussion. The performance of the master-slave variant is comparable to the other methods for small core numbers. If more than 8 cores are used, the efficiency of the master-slave variant drops significantly. The performance of all methods drops for more than 8 because for up to 8 cores everything takes place on one node, whereas for larger numbers of cores several nodes (up to 8 for the case with 64 cores) have to be used. This result reflects the fact that intra-node communication is much faster than communication between different nodes. Still, the thread variant succeeds in providing efficiencies of around 90% on 64 cores for 512 tasks.

The speedup of the RMA and the thread variant for some selected calculations as shown in Figure 5.2 emphasizes again that the thread variant is performing much better than the RMA variant on the Nehalem cluster and is even performing well for small numbers of tasks.

5.1.3 Tests on LRZ Linux clusters and SuperMIG

Scaling tests were also carried out on the test clusters ICE[68] and MPP[69] and the migration system SuperMIG,[70] for the successor SuperMUC of the HLRB-II machine.[60] Results of the averages for the elapsed times can be found in Appendix A.

The ICE cluster[68] contains the same processors as the Nehalem cluster but uses an Infiniband interconnect. For the tests the MPT implementation[71] of MPI was used. In contrast to the previous tests, the size of the tasks was varied by choosing randomly how often one of the loops has to be performed, compare Section 5.1. The tasks lasted between 0.1 and 0.4 s, the average was around 0.24 seconds. Tests were done again for different numbers of tasks, large ones with 1024 and 512 tasks and small ones with 32 and 128 tasks. Tests were carried out on 1 to 16 cores. All methods tested, RMA and thread variant, as well as a static distribution, scaled very nicely. There are hardly any differences between the worst and the best starting distribution visible, thus the task redistribution in both cases is rather efficient. This is most certainly related to the MPI implementation as for the Nehalem Linux Cluster with the same cores larger differences were already visible for 8 cores and a larger task size (Subsection 5.1.2). The thread variant always uses a bit more time than the RMA variant, however it scales better. As there is also an overhead in the case of a single core this can be attributed most certainly to the time sharing with the helper threads, while the RMA variant is not burdened by this kind of overhead. As this effect is not visible on other machines, it might be also related to differences in the implementations of MPI thread levels. The speedup between 8 and 16 cores is less than ideal. Here the factor plays a role that for 16 cores the calculations have to be performed distributed over two nodes.

The MPP cluster is another of the testing clusters of LRZ.[69] It has 16 cores per node and the nodes communicate via Infiniband. The MPI implementation provided by Intel was used; it is provided as an alternative to the default MPI version on many of the LRZ’s computers.[72]

The same tests as for the ICE cluster were undertaken, the time per task was 0.4 s on average for the DLB calculations (but only 0.3 s for the static variant).

The results for this cluster show a preference of the thread variant over the RMA variant, see Appendix A. Even for 1024 tasks (the calculations with the longest run time) the difference between the best and the worst starting distribution for the RMA variants is visible for larger core numbers. The speedup is comparable for the static and the thread variant. The thread variant with the best starting distribution is always a bit better than the static variant, the same holds for the thread variant with worst starting distribution for the larger numbers of tasks. The timings are not directly comparable as the overall working times for the static variant are significantly smaller. This can be related to the fact that this variant was not using any communication of MPI. None of the other machines used showed this behavior so far. This is an artifact of the simple example as calculations on real systems would also require MPI.

The migration system SuperMIG[70] for the new system SuperMUC of LRZ provides an even more interesting example. SuperMIG is composed of Westmere-EX Intel Xeon E7-4870 10C processors with 40 cores per node and Infiniband interconnect. Calculations with 1 to 80 cores were performed. All calculations with up to 32 cores used one node only, but also on two nodes the various methods, RMA, thread and static variant, were all rather efficient for larger numbers of tasks, see Appendix A. With roughly taking 5 seconds for 80 cores the speedup was still around 60 in most cases. The variance of the task length was small enough to yield for the static task distribution variant an equally good scaling as for work stealing. The changes between the performances of the worst and the best starting distribution lead to the conclusion that the thread variant is more efficient in redistributing the tasks than the RMA variant. Therefore the variant with threads will be used for further tests and application calculations on this machine.

5.2 Numerical Integrals

Figure 5.3: Octahedral structure of a palladium cluster with 19 atoms.

As first computational chemistry application of the DLB algorithm the numerical integrals of the exchange-correlation (XC) functionals in ParaGauss (working on a multicentered grid) are taken.[18-19] This part has to be called once in every SCF iteration of the Kohn–Sham method. It has to be called once again in the post-SCF part if also the gradients of the energy with respect to atomic coordinates are of interest. This special case requires more computational effort.

For a test of the numerical XC integral parts an octahedral palladium cluster with 19 atoms in D_{4h} symmetry was selected (Figure 5.3) ParaGauss uses symmetry: the number of symmetry inequivalent atoms is only 5. The Slater exchange[73] was used (with the prefactor of 2/3)[74-75] for the exchange part of the XC integrals, the correlation part was expressed by the Vosko-Wilk-Nusair (VWN) functional.[76] The basis sets are provided in Appendix H.

The computational effort of the numerical integrals depends on the number of grid points to be treated for each symmetric inequivalent atom. Using a tight mesh, there are 78470 such points to be dealt with. ParaGauss provides some vector parallelization in these calculations. In our example a worker choses to process up to 128 points at once as a single task. For the post-SCF calculation all these points have to belong to the same atom, therefore it might happen several times that a worker has to process fewer grid points in the last task for a specific atom, but this example requires more than 600 individual tasks. Additionally it is possible for DLB to divide tasks at the end to increase balancing.

A new version of ParaGauss using DLB for the scheduling of the parallel calculation of the numerical integrals was compared against two older production versions of the ParaGauss version V3.1.5. Relevant for the tests was the fact that the two production versions, V3.1.5 and V3.1.5 R, differed by a restructured grid integral part in the newer version; also the percentage of parallel context —the percentage of code which was shared by all worker—was increased. The only code difference between the more recent production version, V3.1.5 R, and the new version with DLB (V3.1.5 G-DLB) was in the part of the numerical integrals. Performance tests were executed on the Nehalem cluster (Subsection 5.1.2). Two different variants of DLB were applied. Both, the static backend implementation and the production version, V3.1.5 R, use essentially the same algorithm. The only difference is whether the static distribution is done by the DLB library or by ParaGauss itself. The DLB variant with work stealing uses the thread variant, as this variant was shown to perform better than the RMA approach on the Nehalem cluster (Section 5.1).

Good scaling is obtained for all methods for the instance of the numerical XC integral part, for calculating the XC energies in the SCF part. The results from the first iteration in which the integrals are calculated are considered here. There is no large difference between the timings for the production version V3.1.5 R and the two variants of DLB, see Table 5.4.

This shows that, on the one hand, the necessary change in the code to allow the usage of DLB is properly done without loss of efficiency and that, on the other hand, the dynamic variant can be also used efficiently for this part. The good scaling of the static variants is due to the homogenous computer architecture and the very homogenous size of the tasks. Thus, it is enough to see that the dynamic variant is always competitive. The overhead due to communication, mainly related to the termination algorithm, is invisibly small.

Table 5.4: Elapsed real timings in seconds and efficiency for executing the numerical integration of the XC energy contributions in the SCF part of ParaGauss for the cluster Pd_{19} in O_h symmetry (Figure 5.3). Compared are two versions of ParaGauss, V3.1.5 and V3.1.5 R, against the version V3.1.5 G-DLB with both static and thread variant of DLB employed.

Method	Time	2	4	8	16
PG: V3.1.5	8.2	0.99	0.98	0.98	0.95
PG: V3.1.5 R	8.2	0.97	0.99	0.96	0.95
STATIC	8.2	0.99	0.99	0.96	0.96
Thread	8.2	1.00	1.01	0.95	0.86

Time requirements in this part are of a factor of 8 smaller than in the case of the gradients of the XC energy with respect to the nuclear positions. For the calculation of the gradients of the XC energy on the numerical grid the corresponding time requirements are summarized in Table 5.5. For this example the overhead due to communication of the thread variant is also invisible. The differences between the two variants are of the size of the standard deviation. Thus, DLB can also be favorably applied to parallel calculations of tasks of known size which are homogenously distributed and where dynamic load balancing is not necessary. This feature of DLB allows rather wide application and thus allows one to simplify complex parallel programs where thus far different parallel algorithms had been used.

Table 5.5: Elapsed real timings in seconds and efficiency for the XC numerical gradient integral part in the post-SCF part of ParaGauss. Compared are two versions of ParaGauss, V3.1.5 and V3.1.5 R, against the version V3.1.5 G-DLB with both static and thread variant of DLB employed, see also Table 5.4.

Method	Time	2	4	8	16
PG: V3.1.5	66.8	0.50	0.50	0.50	0.49
PG: V3.1.5 R	66.6	0.98	0.95	0.95	0.93
STATIC	66.9	0.99	0.95	0.95	0.96
Thread	67.2	0.96	0.97	0.96	0.91

Noticeable is also the difference between the two production versions V3.1.5 and V3.1.5 R in the post-SCF main part, see Table 5.5, (but not in the SCF XC part). The difference between the versions is some simplifications and a restructuration of the code, as preparation for the change to use the DLB interface. This shows another advantage of the DLB interface: the complexity of the code is reduced (and so are chances for remaining errors in the code).

5.3 Three-Center Integrals

The three-center integral part of ParaGauss[16, 69-71] present more challenge for the dynamic load balancing than the previous test case, as here the individual tasks have different sizes. One reason for this is that the indices of the three-center integrals according to which they are bundled in tasks belong to a shell, which may contain different numbers of Gaussian basis functions (Section 1.1). It is possible to estimate the computational time required to calculate such a task. For example for three- (and two-) center integrals an estimate is available using the number of symmetry-equivalent connections between the atoms, the number of different exponents in the primitive basis functions and the angular momentum of the chosen orbitals.[77] These estimates are used to rearrange the tasks in decreasing order of computational effort; they are the same for all invocations of the overall task.[22, 77-79]

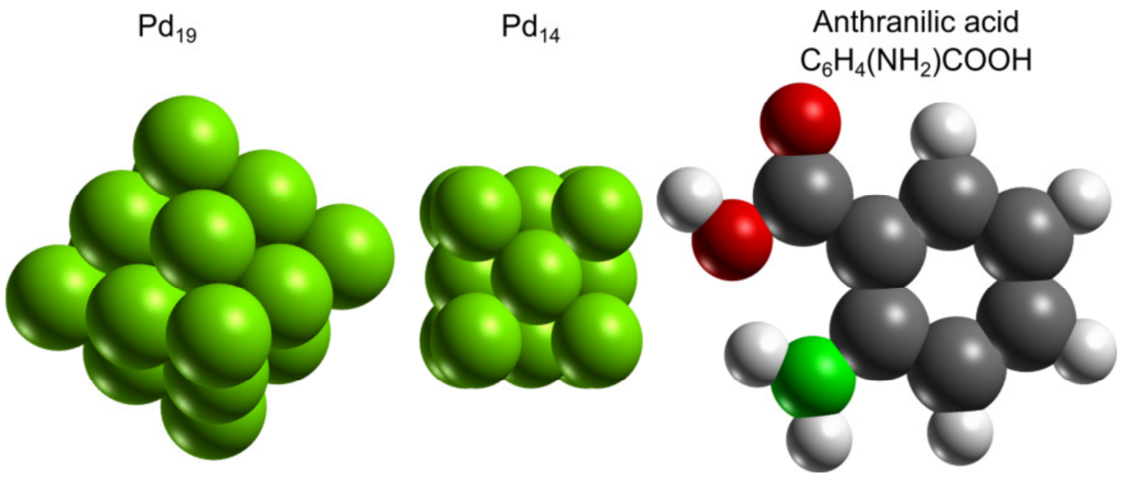

Figure 5.4: Structures of test systems for the calculation of three-center integrals.

There are three different kinds of three-center integrals. The first appears when calculating the Coulomb energy with the fitting function approach (Section 1.1); it is called before the SCF cycle. The second kind is related to the first-order derivatives of the Coulomb potential with respect to the atomic positions, which is part of the gradients of the energy, and is calculated after the SCF cycle, thus post-SCF. If second-order derivatives of the potential energy are to be calculated analytically, the three-center integral routines are even called a third time for the corresponding integrals, again in the post-SCF part. The calls in the post-SCF part for the first- and second-order energy derivatives are done separately and also the results are collected right after the DLB loop finished. Thus they can be examined separately. There are large differences in the time requirements for the tasks between the different three-center integral parts, but as used for the time estimate, the order of size should not be changed much.

In the following, three test examples will be discussed (Figure 5.4). Calculations were carried on the local Nehalem cluster; see Subsection 5.1.2. The basis sets for the atoms used in the examples are provided in Appendix H.

5.3.1 Palladium Cluster with 19 Atoms

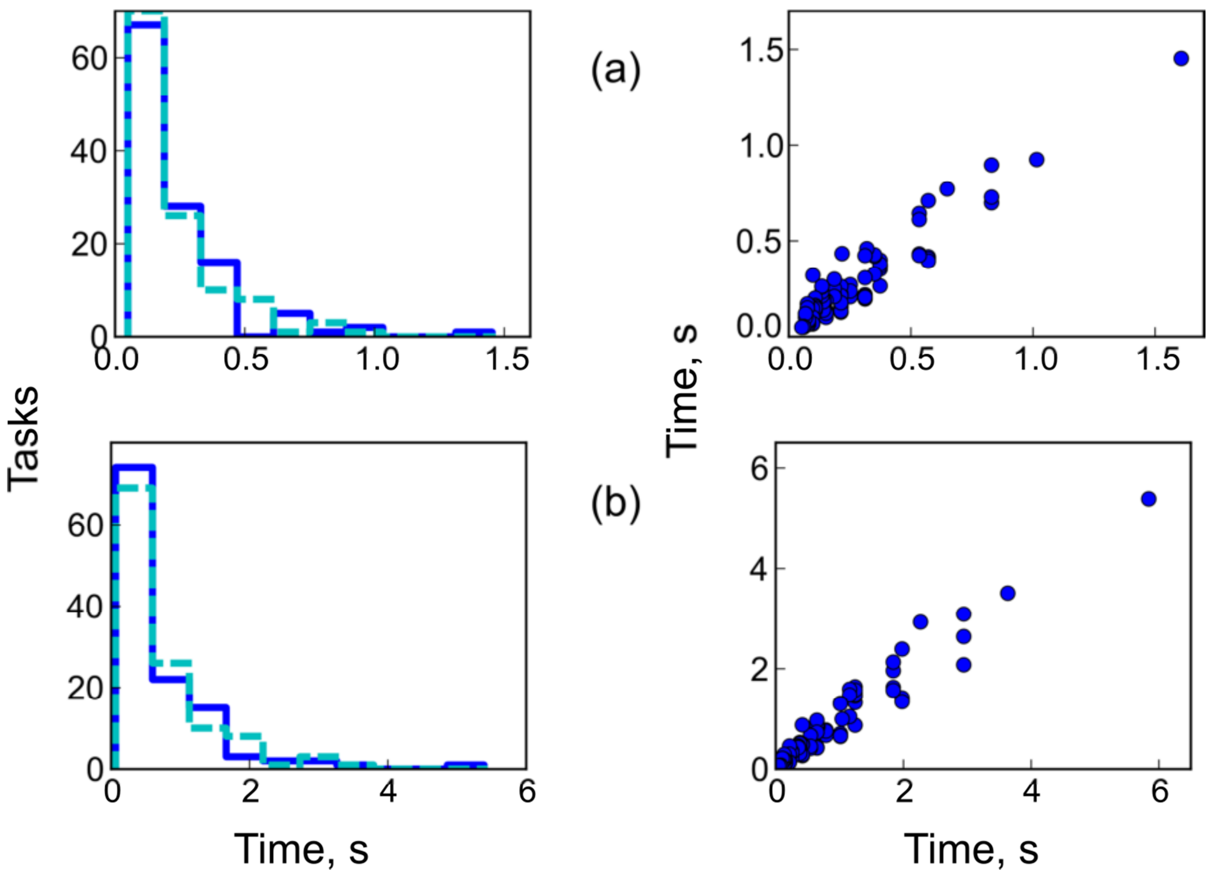

Figure 5.5: Comparison of real and estimated times for the various tasks of the parallel calculation of three-center integrals for (a) the energy and (b) the gradients of the energy calculation of Pd_{19}. Left panel: occurrence of tasks in time interval for real measurements (solid) and for the cost function (dashed). The results of the cost function were scaled with help of linear regression on the interval of the times. Right panel: real time against the estimated time (after scaling) for every task.

Figure 5.5 shows the distribution and how well the estimated times fit the real timings for the Coulomb energy related call of the three-center integrals of Pd_{19} which has already been used as example in Section 5.2; see also Figure 5.4. Altogether there are 120 tasks, of which many are small and only few take longer than half a second to be computed. The cost function (which creates the estimates) provides the right trend, see Figure 5.5, but there is some scattering around the real values visible as the function is not exact. The true values vary between 1.5 and < 0.05 seconds; the estimated values have been scaled to fall into the same interval.

Both distribution and fitting of the values is similar for the other call of the three-center integral module, for the gradients of the energy evaluation (Figure 5.5). The main difference is that the real times are between 5.4 and 0.07 seconds, but both trend and scattering are preserved.

Table 5.6: Elapsed times in seconds and the efficiencies for different methods, the production version V3.1.6 C (with master-slave method) and three version of the ParaGauss version V3.1.6 A-DLB, with the three different DLB versions, static, thread and RMA. Calculations of Pd_{19} used 2 to 8 cores for efficiency calculations of the three-center integral part for the energy and gradient calculation. Averages were taken over three runs.

Method	Time	2	4	8
Energy				
V3.1.6 C	25.9	0.87	0.73	0.67
Static	25.8	0.88	0.66	0.58
Thread	26.2	0.96	0.89	0.84
RMA	25.2	0.94	0.86	0.67
Gradients				
V3.1.6 C	85.2	0.86	0.78	0.75
Static	84.2	0.95	0.77	0.74
Thread	85.4	0.99	0.97	0.94
RMA	83.6	1.00	0.94	0.82

Table 5.6 compares the elapsed times and efficiencies of three-center integral parts when calculating the energy and the gradients of the energy. Four methods were compared: an old production version of ParaGauss, V3.1.6 C, and the developer version with DLB (V3.1.6 A-DLB)), where for DLB the static, the RMA and the thread variant were employed. The efficiency for up to 8 cores was not that good for all methods and the best value for the 8 cores was 0.94. But one has to consider that the three-center integral part contains more work than the dynamic load balancing loop, like preparations, finalizing and the results of the calculation have to be collected after the DLB loop has finished. This is different from for the simple test case of Section 5.1. On the other hand, the differences in timings are mainly due to the differences in the efficiency of the dynamic load balancing. The old production version, V3.1.6 C, still distributes with a master-slave variant.

For all methods the tasks were ordered according to their approximated time requirement, starting from the largest. Thus the balancing at the end should be done with the smaller tasks.

For the smaller example, the pre-SCF three-center integrals, the production version with the master-slave algorithm is superior to the static variant for 4 and 8 cores, Table 5.6. However, both the thread and the RMA variant show a better performance. The more cores are used, the larger is the gap between the old and the new variants. Although the old variant was still useful and superior to a static distribution, the work stealing approaches are the best choice for the current calculations. Increasing the number of cores should further enlarge the gap between the two solutions. The thread variant is getting better than the RMA approach for increasing core numbers, confirming the result of the simple DLB tests that it is the best method on the Nehalem cluster (Subsection 5.1.2). The RMA variant only profits from a smaller overhead due to its implementation, thus starting with smaller elapsed time requirements. The speedup for the thread variant is always better than for the RMA variant. This gets even more noticeable for the second instance where three-center integrals are evaluated, at the post-SCF gradient of the energy calculation, see Table 5.6. The larger task size there benefits the thread variant which needs already for four cores less time than the RMA variant. The efficiency is also always much better for the thread than the RMA approach and is reaching even 0.94 instead 0.82, respectively, for 8 cores. The production version is hardly better than the static variant, in case of 4 cores they are both requiring 5 seconds more than the other two variants (Table 5.6).

The performance is supposed to be strongly influenced by the molecule used as an example. For the numerical integrals this effect might only concern the number of tasks and the time per task, but for the three-center integrals it can also affect the spectrum of computational times needed for the various tasks. In the following subsections, Subsections 5.3.2 and 5.3.2, two other examples are used to explore these effects.

5.3.2 Palladium Cluster with 14 Atoms without Symmetry

The scaling behavior of three-center analytic integral calculations was also examined for a palladium cluster of 14 atoms but without any symmetry constraints (Figure 5.4). This cluster is smaller than the cluster Pd_{19} just discussed. However, removing the symmetry constraints makes the size of this example larger, as there are now 903 instead of 120 tasks to consider. On the other hand the size of all tasks becomes nearly the same, see Figure 5.6. The time requirement per task is decreasing. No task with a time requirement of more than a second occurs in the energy calculations, and only 21 tasks are to be carried out in the part of three-center gradients of the energy calculations. For the energy integrals there are even only 11 tasks requiring over 0.5 seconds. 432 tasks were requiring below 0.1 seconds, nearly half of them, while for the gradient related integrals it was with 192 tasks already less. The small time requirements per task might also be the reason for the large differences between the distributions of real time and estimated time (Figure 5.6).

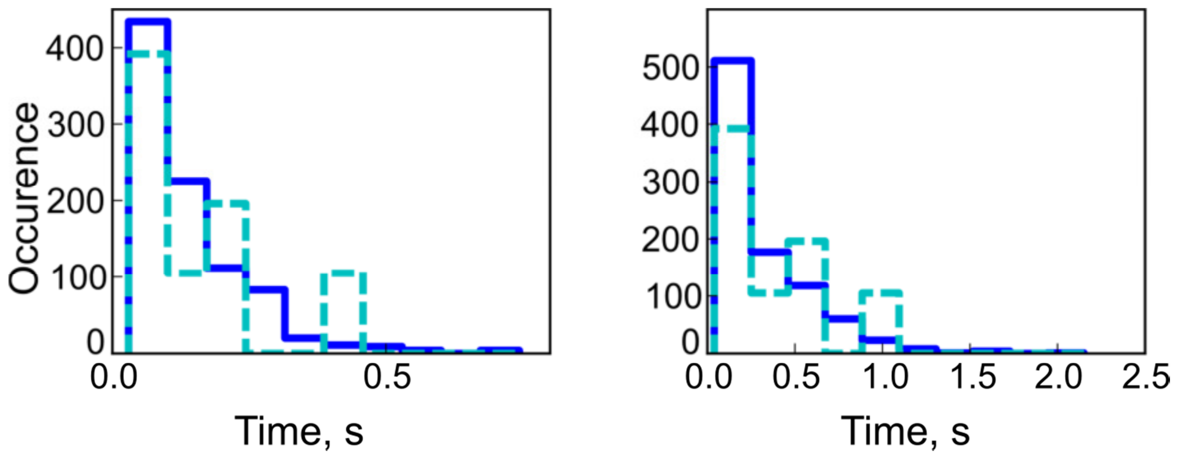

Figure 5.6: Distribution of the tasks for the three-center integrals of the Pd_{14} cluster in C_1. The solid line refers to the real time distribution the dashed line to the approximated time requirements, scaled on the same region than the real distribution. Left panel: the integrals for the energy calculation; right panel: the integrals for the force calculation.

The dynamic load balancing algorithm is also requiring some time, the RMA variant for the remote memory access and the thread variant to handle the calls to the routine mutex regarding the locks on the storage area. This influences the scaling. The elapsed times and efficiencies are shown in Table 5.7.

Table 5.7: Elapsed times (seconds) and efficiency for the three-center integral calculations for the energy and the gradient of the energy of a Pd_{14} cluster calculation on 1 – 8 cores. Shown are the production version V3.1.6 C (with master-slave method) and three version of the ParaGauss V3.1.6 A-DLB version, with the three different DLB versions, static, thread and RMA.

Num. procs.	Time	2	4	8
Energy				
V3.1.6 C	103.0	0.58	0.51	0.38
Static	109.1	0.81	0.66	0.45
Thread	104.0	0.90	0.78	0.54
RMA	103.0	0.86	0.70	0.53
Gradients				
V3.1.6 C	245.5	0.86	0.78	0.68
Static	253.5	1.00	0.99	0.54
Thread	246.5	0.98	0.96	0.90
RMA	244.7	1.01	1.00	0.89

For the ParaGauss reference and the thread variant, the results of this example do not scale as well as for the Pd_{19} cluster in D_{4h} symmetry (Subsection 5.3.1). For example the efficiency on 8 cores for Pd_{14} and Pd_{19} and the energy related calculation by the thread variant was 0.54 and 0.84, respectively. The static variant does not suffer that much. Quite the contrary, there are two facts which improve the performance of the static task distribution relative to the Pd_{19} cluster: the computational times between the tasks differ less, resulting in a better static balance. The average time per task is decreasing, thus the time required by the other variants to decide which task to calculate next is getting more important. The best performance results when one has to take only the subsequent value from an array, without any locks, as done in the static variant. The RMA variant is doing quite well too, even better than the thread variant.

5.3.3 Anthranilic Acid

All examples examined so far were Pd clusters with only one kind of atoms. A completely different example is the organic molecule anthranilic acid, see Figure 5.4. It contains four different types of atoms: carbon, oxygen, nitrogen and hydrogen This molecule leads to 1485 tasks of three-center integrals.

Table 5.8: Elapsed times (in s) and efficiency for various three-center integral parts with different methods on 1-8 cores. Shown are three versions of the ParaGauss version V3.1.6 A-DLB, with the three different DLB versions, static, thread and RMA.

Num. procs.	Time	2	4	8
Energy				
Static	58.9	0.92	0.81	0.49
Thread	62.8	0.96	0.91	0.63
RMA	59.9	0.91	0.80	0.57
Gradients				
Static	344.3	0.99	0.95	0.71
Thread	347.0	0.99	0.96	0.88
RMA	348.1	1.00	0.96	0.84
2. deriv.				
Static	2671.9	1.00	0.98	0.58
Thread	2714.6	0.99	0.99	0.91
RMA	2671.9	1.01	1.01	0.86

The time requirements of them are quite different for the three parts where integrals of this type are calculated. For the energy part the tasks needed 0.04 s on average, lying between 0.006 and 0.23 s. The gradient of the energy integrals had times between 0.04 and 1.73 s, with an average of 0.24 s. The three-center integrals of the second-order derivatives of the energy needed between 0.1 and 18.7 s each, with an average of 1.94 s. The distribution in all cases included a large amount of tasks, which were requiring only very short times, see Figure 5.7. However the estimates of the cost were not fitting as well as in case of the palladium cluster, see Figure 5.4: there are large differences between the real and the approximated values visible. With the use of different atom types the estimate function has to deal with more parameter resulting in more possibilities for deviations to the real times.

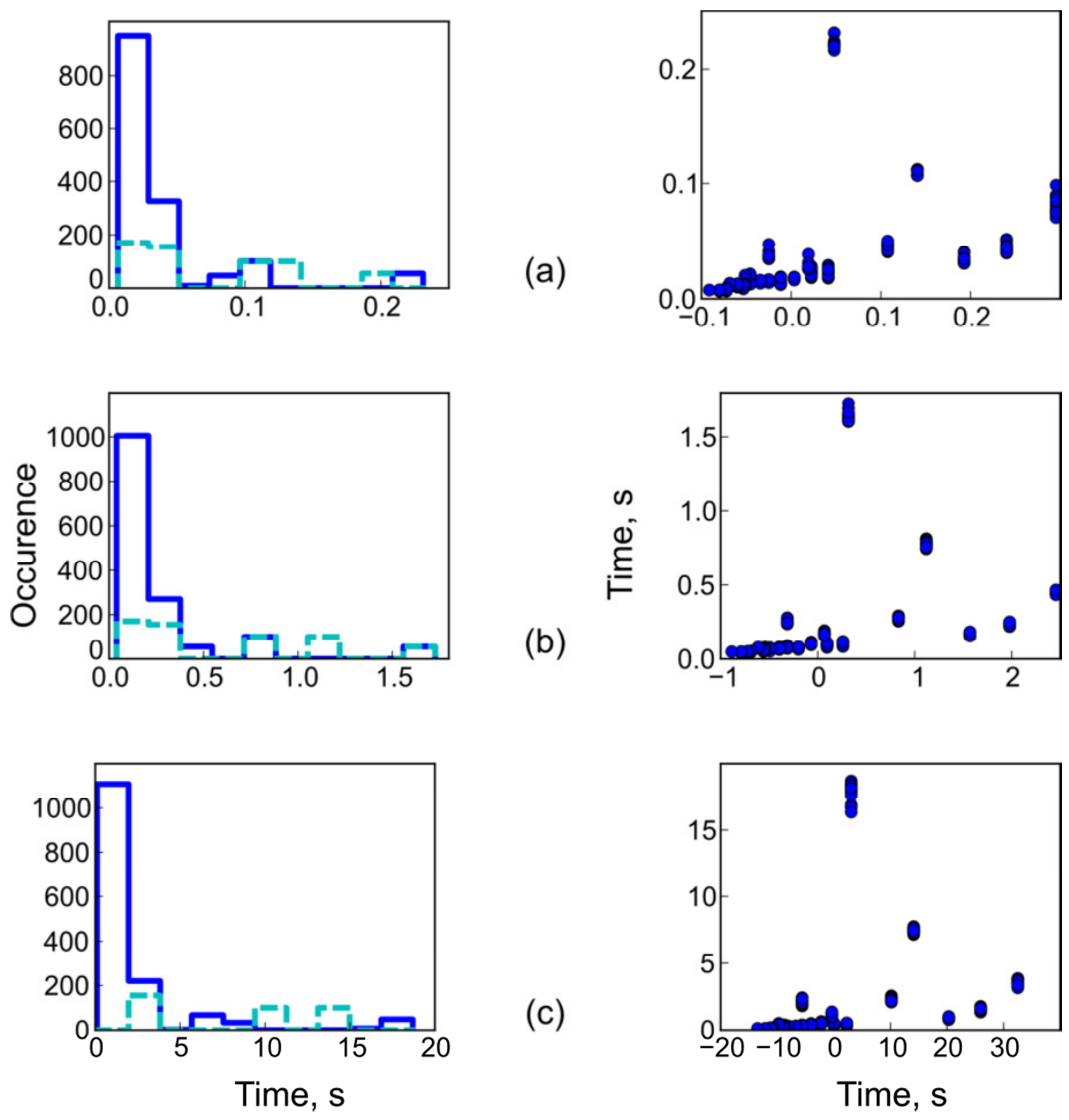

Figure 5.7: Distribution and plots of the three-center integrals of the anthranilic acid. Left panel: occurrence of tasks in time interval for real measurements (solid) and for the cost function (dashed). The results of the cost function were scaled with help of linear regression on the interval of the times. Right panel: the real time against the approximated time (after scaling) for every task. (a) for the energy, (b) for the gradients of the energy and (c) for the analytical second-order derivative of the energy.

The results show that the two dynamic variants are comparable to each other, see Table 5.8. For few cores the efficiency of the static variant is also of the same order.

6. Large-Scale Testing on Four-Center Integrals

6.1 Systems and Method

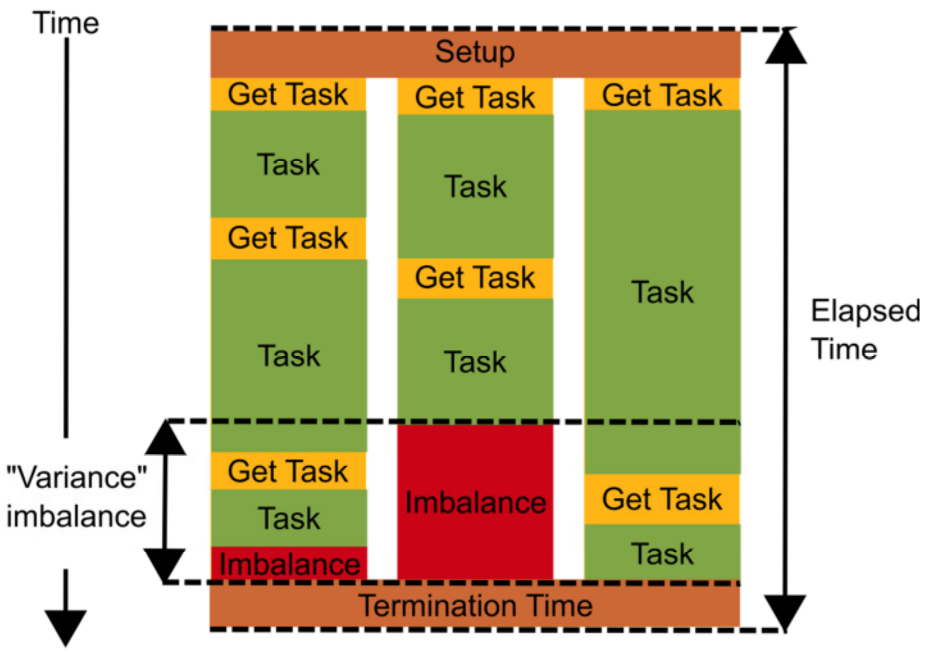

Figure 6.1: A sketch of the time measurements of the four-center integral part on an example of seven tasks and three cores. The time progresses from top to bottom, every column indicates a core. Setup and termination is done by all cores. In between every core calculates its tasks independently. There are two types of “lost” time: due to determining the next task and the imbalance at the end. The times for determining the next task include the complete time till the next task can be performed, therefore also the mapping of the task ID to the real task. The sketch displaces the distribution on the example of the thread DLB method.

The following sections describe the procedure and the results of large-scale testing of the recently constructed four-center integral part of ParaGauss.[11] Calculations for the following tests were carried out on the HLRB-II cluster,[60] which allows the use of large core numbers. As already mentioned in Section 5.1 HLRB-II has 512 cores per node, however only up to 510 of them can be used for the performance tests. The DLB method used for the tests was always the method with remote memory access, which was found to be the superior of the two work stealing algorithm implementations for this hardware (Section 5.1). A static variant was used for comparison.

The four-center integral part is the part with the largest size, among all the integral parts, regarding both the number of tasks and the computation time per task. Furthermore the code can be processed separately from ParaGauss. Not having to run always the complete electronic structure program reduces the required time for testing this part. This makes this part well suited for large real time tests of the work stealing algorithm. The analytical four-center integrals are calculated using the MD2 algorithm,[80-81] following the direct SCF approach.[82] In the test runs only prescreening of the integrals was used to reduce the number of integrals to be calculated.[83] Further improvements of the method to reduce the number of explicitly calculated integrals were not included.[11] Some of the tasks, the ones where the first and the third as well as the second and the fourth indices where the same, were separated and calculated beforehand, as they are used for prescreening. The remaining tasks were calculated in a second loop, which is larger. The following performance tests concentrate on this second loop as overall task. After the first loop it was required to provide the results of it for all workers, allowing them for every task to prescreen it. This was done by a global exchange, which separates anyhow the two loops.

Therefore putting a barrier before this block of the second loop for the tests should not change much and keep the results relevant for further production runs without the barrier. However it allows one to concentrate properly on only one application of the work stealing algorithm, as already done in the previous examples. The time between this barrier and the end of the work stealing loop can be divided into several types of timings, see the example sketch for three cores in Figure 6.1. During the setup phase the tasks are distributed and locally stored. This phase is part of the overhead of the work stealing algorithm. As every core can do this on its own, independent of the distribution method applied, this effort should be negligible. The true working times are the times spent in the routines to calculate the integrals. To optimize them is not part of the current investigation, thus there is no further division of timings needed. The gaps between these working times are part of the overhead, which should be avoided. They refer to the time when a new integral task is requested from the worker till the time when it starts the working routine again. The gaps include the time the work stealing algorithm requires to get a new task, which can usually not be neglected, as it involves reading a locked area.

Further contributions to this time are given by the fact that the DLB routine provides only a task ID, which has to be transformed into values or intervals of the four indices of the integrals, defining the computational task. This can take a varying amount of time depending on which method is used. Two methods are available. The integrals can just be distributed in the order of their running indices. A slower algorithm tries to consider the size of each task, which can be estimated by a cost function, similar as for the three-center integrals (Section 1.1). The cost sorting is explained in Section 6.2 in detail. Already for moderately sized systems there are many more tasks than for the cases of three-center integrals. Therefore it is not feasible to store the task indices in a sorted list. The IDs of the four-center integrals calculated first in the four-center integral module are still nominally present in the list. As some tasks are removed by prescreening, the worker, which got their task ID, skips calculating them and switches to the next task ID. The times required for transforming the task ID in the ID of the indices are also included in the statistics. For both, work times and overhead times, the sum for each core is calculated. Between the calculation of the last task and the end of the overall task the workers spend additional overhead time, due to imbalance and the termination algorithm. These times are lumped together. The core finishing last is supposed not to create any overhead time due to imbalance. Therefore, the smallest time measured formally as overhead due to imbalance is assigned to be the time of the termination overhead. The imbalance overhead time of the other workers is reduced by the amount of time which has been identified in this way as termination time. All these times $T_i(p)$ are summed over all cores *p* up to the number *n* of cores used for the following observations.

$$T_i^{(n)} = \sum_{p=1}^{n} T_i(p) \tag{6.1}$$

Except the time spent on the actual calculation, the times for all other tasks are classified as overhead. The detailed timings described above allow defining a different kind of efficiency besides to the usual one based on scaling. This efficiency compares the time $T_{\text{work}}^{(n)}$ spent in the working routines with the total elapsed time $T_{\text{elapsed}}^{(n)}$ between the start and the end of the four-center integral module for the number of cores *n*.

$$e_{\text{W/ET}} = \frac{T_{\text{work}}^{(n)}}{T_{\text{elapsed}}^{(n)}} \tag{6.2}$$

In the following, this value will be referred to as work/elapsed time (W/ET) efficiency. For the examples where calculations with one core were also performed, it is also possible to provide besides the W/ET values the standard efficiency e_{n}, which uses the elapsed times for different core numbers.

$$e_n = \frac{T_{\text{elapsed}}^{(1)}}{T_{\text{elapsed}}^{(n)}} \tag{6.3}$$

Thus it is possible to compare the different efficiency results and to verify that the W/ET efficiency $e_{\text{W/ET}}$ is able to provide correct information about how efficient a specific calculation was.

As in the other tests, see Chapter 5, the times were measured by MPI routines as elapsed wall clock time.

6.2 Task Sorting

To sort the tasks according to the (expected) calculation time, an approach was used that differs from that employed for the three-center integrals to be used because maintaining a sorted list of tasks is too expensive for four-center integrals. Instead, a mapping of integral numbers onto the task was established.[84]

Simple approaches for estimating the cost of a task use, among other things, the angular momentum l_i for every shell i.[83] In the current context the contribution to the cost approximation related to the angular momentum was inspired by the solid harmonic tensor formalism.[85-86] This gives rise to a term $2l_i + 1$ in the cost estimation of shell i. Another factor per shell i is the number of basis functions n_i (without considering contractions). As cost estimate C of a task the following expression was used:

$$C = \prod_{i=1}^{4} n_i^{\nu} (2l_i + 1)^{\mu} \tag{6.4}$$

or

$$\log C = \sum_{i=1}^{4} \nu \cdot \log(n_i) + \mu \cdot \log(2l_i + 1) \tag{6.5}$$

.

The two parameters ν and μ are taking care of the fact that the two contributions related to the angular momentum and the number of basis functions influence the time requirement for a task in different ways. The parameters were adjusted for the test cases H_2O and UF_6, the latter in two different basis sets. The values of the parameter were $\nu \approx 0.8$ and $\mu \approx 1.5$. The cost of a task is given as the product of the multiplied cost contributions of all its four indices. This rather rough approach to estimate the cost is sufficient in the current context as the sorting is done only approximately and only the trend of the task sizes is of interest.

In the cost sorting algorithm the interval of the costs (logarithmic scale) is divided into cost windows of equal width to which the tasks are assigned. To find the task belonging to number n in the sorted sequence, the original task list is scanned skipping the tasks that fall outside of the current cost window. If the end of the list is reached, the next cost window is selected and the search is started again from the first element of the original list.

The decision to which window a task belongs, that lies exactly at a border, was changed shortly after the tests. During the tests a task a with $b_l < a \le b_r$ belongs into the cost window of the interval (b_l, b_r), while for the current implementation it is required to fulfill $b_l \le a < b_r$. The different choices are depictured in Figure 6.2. This should not have a large effect on the current calculations. A realistic distribution of the tasks according to estimated times (together with the real times) can be seen in Section 6.4 in the right panel of Figure 6.4 where also the borders for the cost windows are shown. This shows also that the time estimate by the cost function corresponds reasonably well to the real time to justify sorting according to the estimated costs. Note that the copper clusters of Section 6.4 were not among the examples for adjusting the parameter.

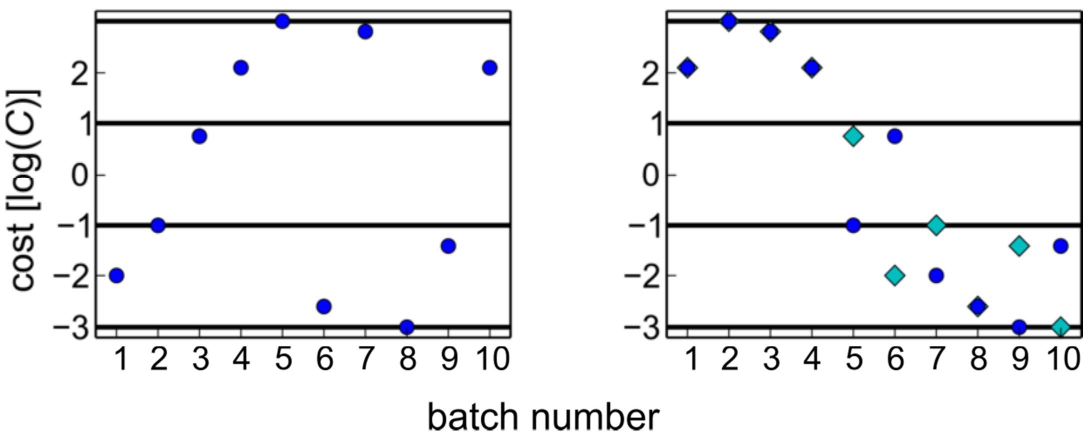

Figure 6.2: Example of cost sorting of tasks to task number. The example uses three cost windows. Left panel: unsorted tasks. Right panel: tasks sorted according to cost windows for performance tests (diamonds) and after the recent cost sort version (dot). The differences are due to changes in assigning the tasks, which are exactly on the border of a window.

The algorithm needs some time to identify the next task to process. The average time per task for this identification increases especially if the sorted tasks are accessed in random order. The cost sorting algorithm used for work stealing should encounter this regime only occasionally, when the work on a newly stolen set of tasks is started. Still this algorithm produces more overhead compared to a static variant.

In order to justify the overhead due to the cost sorting algorithm it is required that for the static variant every worker possibly gets about the same amount of tasks from every cost window. This is realized by assigning the task numbers to the cores in turn. For the work stealing variant it is in addition very favorable if all workers start with large tasks.

6.3 The Test Systems

Seven copper clusters Cu_n (*n* from 4 to 20) were chosen as test systems. The atoms were arranged in regular structures, where the nearest-neighbor inter-atomic distances are ~2.5 Å. (The value of bulk Cu is 2.554 Å.[87]) The structures were inspired by studies on small copper clusters.[88-89] The geometries of the copper clusters chosen are shown in Figure 6.3.

As before the number of calculated tasks is reduced by prescreening of the four-center integrals. The task of the first loop and the screened tasks are not considered in the efficiency measurement. For Cu_{10} and Cu_{20} some tasks were screened. Table 6.1 provides the total calculation times, obtained as the sum of the real times of all workers (elapsed times measured only while the tasks are being calculated). They are averaged over all calculations carried out with the system, thus over the various core numbers and methods, as the real working time should not be affected by the task distribution. This number is rather stable; the standard deviation for all examples is below 1%. The real working time increases strongly with increasing size of the system. Each task required approximately 1 second.

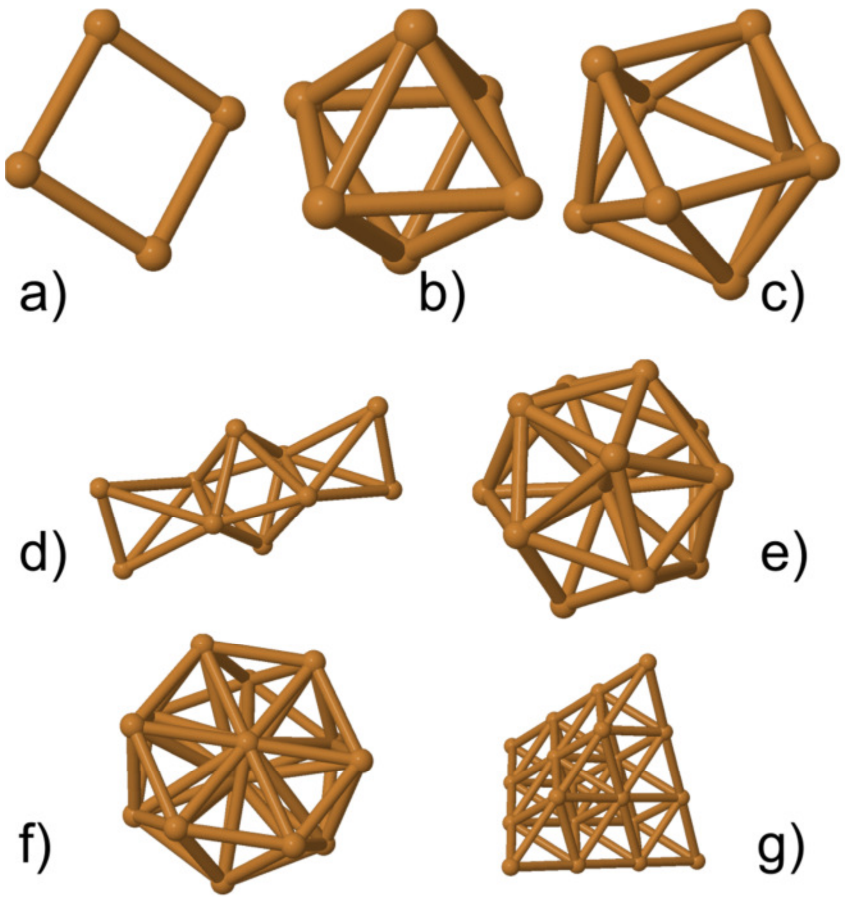

Figure 6.3: Structures of the copper clusters used as test systems. a) Cu_4, b) Cu_6, c) Cu_8, d) Cu_{10}, e) Cu_{12}, f) Cu_{14}, g) Cu_{20}.

The distribution of the time requirements for the tasks as well as the approximate costs can be visualized in the same way as for the three-center integrals (Section 5.3). The calculation time of a single task is not expected to depend on the number of atoms involved, but mostly on the angular momentum of the various shells which are the same for all atoms of the same kind. Thus all copper clusters of Figure 6.3 should show in principle the same cost distribution of the tasks. It will not be exactly the same distribution as not all combinations of the integral indices are appearing in the task numbers because some combinations refer to the same integral. The distribution of the smallest clusters Cu_4 is given in Figure 6.4 showing again the number of small tasks (< 2 sec) overweighting the number of larger ones (time > 2 sec). Only 16 tasks need more than 7 seconds, more than half of them are smaller than one second. In case that the tasks are sorted with the help of cost windows, there are eight windows, as shown in Figure 6.4.

Table 6.1: Statistics about the copper test examples. Shown are for each cluster size the number of tasks in the relevant part and the averaged real working time (as elapsed wall clock) in hours of the tasks over all core numbers and methods (thus all calculation of Section 6.4).

Num. atoms	4	6	8	10	12	14	20
Num. tasks	3003	14535	44850	107720	221445	407253	1672223
Calc. time	1.0	5.0	15.9	37.5	79.3	140.7	600.7

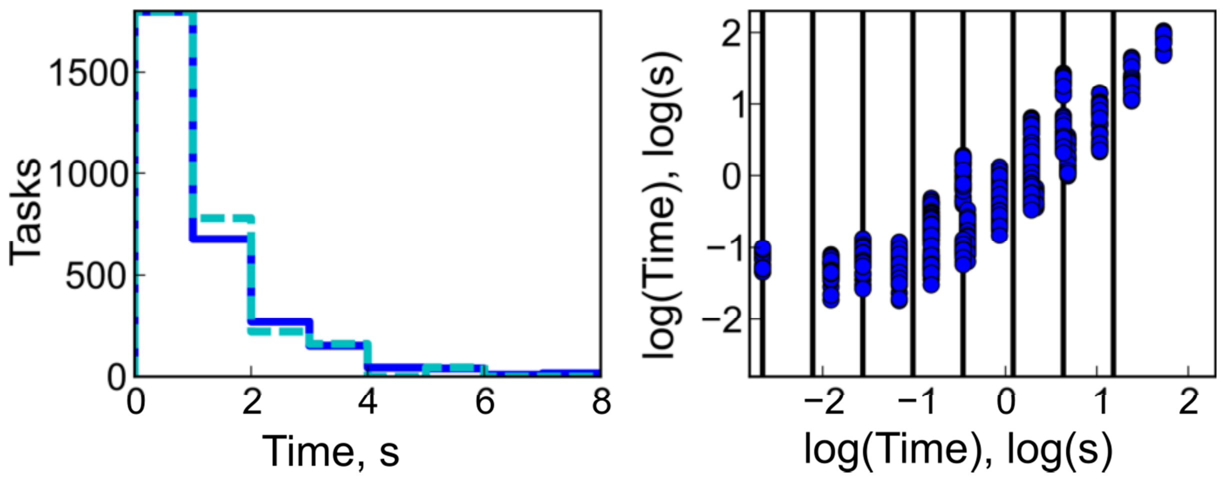

Figure 6.4: Time requirement for tasks calculation for Cu_4. Left panel: how often a task fell into the given interval. Right panel: the mapping of the time requirements onto the time expectations in logarithmic scale. The cost windows used for sorting according to the estimated times are also shown (the lines mark the border).

The decision for the number of cost windows was made after a test with Cu_4 on 128 cores: Cost sorting of tasks should improve the performance of the load balancing methods. This should be especially the case when it is applied together with a dynamic load balancing method. Treating smaller tasks at the end should optimize the balancing. Cost sorting should also improve the performance of a static task distribution, as it should prevent that all large tasks end up on one core. However the effect should be more noticeable for a dynamic variant. Figure 6.5 shows how the W/ET efficiency, Eq. (6.2), changes with the number of cost windows on 128 cores. The W/ET efficiency grows with increasing numbers of cost windows, thus the better the tasks are sorted the more time is saved. On the other hand, the time required for scheduling is also increasing, see Figure 6.5. This increase is roughly linear for the current example with a slope of 0.13 s/cost window. For the current example this is negligible, as it amounts at most to 1.4 s for 10 cost sorting windows, while the complete computation time for Cu_4 is around 3450 s, see Table 6.1. This overhead for sorting is also much smaller than the overhead due to the imbalance which is at least 80 s for the various cost sorting windows.

Other factors may affect the amount of overhead. With increasing number of cores the overhead should increase as every core has to perform the scheduling over the tasks numbers, but the influence of the system size should be even larger. For N being the number of angular momentum shells of all atoms of a system, the number of tasks (before screening) is proportional to the forth power of it, $n_{\text{task}}{\sim}N^4$. This is also the reason why the tasks are not sorted beforehand and a list of them is provided. For the cost sorted calculation one can expect that (on one core) the list has to be calculated $w{*}n_{\text{task}}$ times in case of w windows. Nevertheless the number of cost windows should have a moderate effect only, as the increase in the number of tasks will dominate the overhead. Still it does not make sense to choose w very large. With more than 8 cost windows the gain in the W/ET efficiency is only marginal, see Figure 6.5. Therefore, all calculations with cost sorting, discussed in the following, were carried out with 8 cost windows.

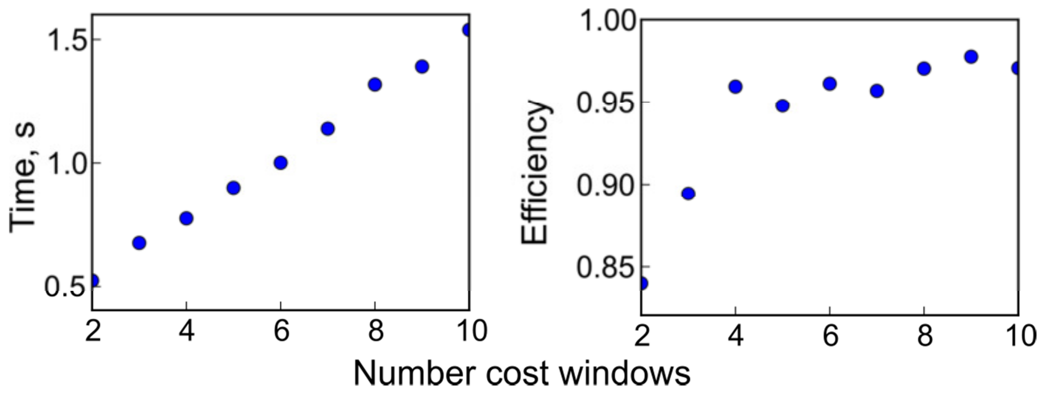

Figure 6.5: Scheduling time and W/ET efficiency for the four-center integrals of Cu_4 on 128 cores for varying numbers of cost sorting windows. Calculations were done with dynamic load balancing including cost sorting.

For a systematic test, three methods were used to distribute the tasks. The dynamic load balancing strategy with cost sorting, as defined in Section 6.3, is one of them. Additionally it is possible to test dynamic load balancing on the original task number assignment, thus with minimal cost requirement to transform the task ID into the indices. The third method is a static variant, which uses also the cost sorting. Calculating the largest tasks first should not have an effect in this case. However the sorting should provide every core with nearly the same amount of tasks of an expected computational afford. It prevents that, for example, all tasks belonging to the largest computational afford, will be scheduled on the same core. This strategy provides the additional benefit of revealing the mere overhead of the cost sorting. For the tests on HLRB-II the RMA variant of DLB was chosen for both dynamic distribution methods, on SuperMIG the thread variant, as these were the respective variants performing best on the given machines for the simple tests, compare Section 5.1.

6.4 The Test Process on HLRB-II

The first part of the test was done with the smallest copper cluster, Cu_4, using varying numbers of cores. The calculations were also done with a single core, therefore it is possible to calculate next to the W/ET efficiency also the real efficiency, see Figure 6.6. The general trend is the same for both efficiency measures and the differences between them are always smaller than 0.02. This supports that the W/ET efficiency can be used to examine the quality of calculations. It can be even used if only a single calculation has been done for the specific system, while the real efficiency requires that there are at least two calculations, of which one should be a serial run for a proper efficiency. Furthermore the real efficiency has the drawback that it does not consider the overhead of a calculation on one core, different from the W/ET efficiency. In the following the efficiency of the results is examined using the W/ET efficiency.

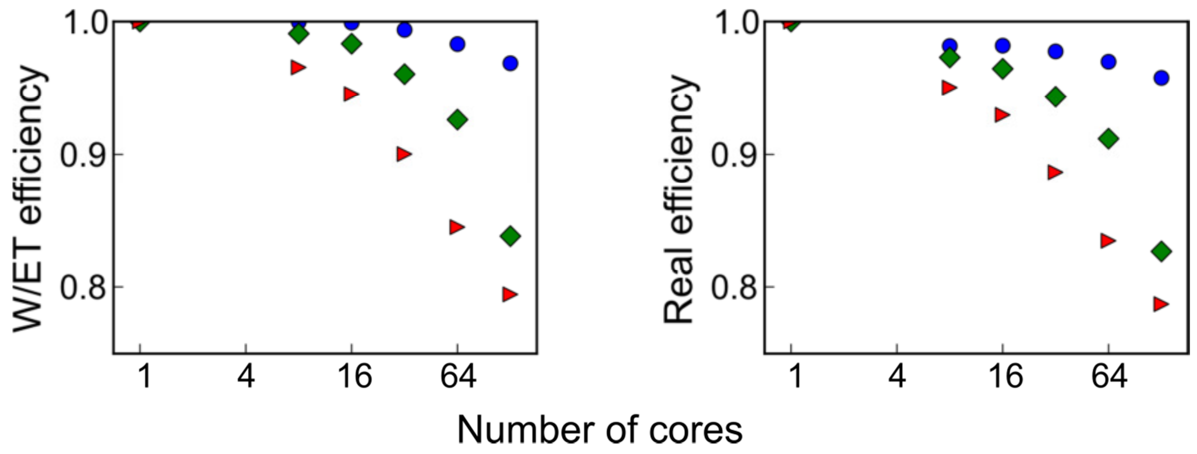

Figure 6.6: W/ET efficiency and real efficiency for the example of Cu_4 for 1 to 128 cores. Dynamic load balancing with cost sorting (dots), dynamic load balancing on unsorted task list (diamonds) and static cost sorted method (triangles). Averages over three runs.

Examining the efficiency the difference between the performance of the various methods becomes evident. Using DLB with cost sorting was doing best for Cu_4, having still a W/ET efficiency of 0.97 for 128 cores. Using DLB with unsorted tasks was doing always worse. Especially between 64 and 128 cores there was a large drop in the W/ET efficiency from 0.93 to 0.84. This indicates that for the method the number of cores is getting too large for this system size. There are only 23 tasks on average per core and the calculation lasted around 30 seconds on every core. For randomly distributed tasks there were too few to archive a balanced load situation. The DLB with cost sorting with its high efficiency of 0.97 illustrates that the sorting is responsible for the improvement. The static variant with cost sorting is the variant with the lowest efficiency, it achieves only 0.79 on 128 cores. Already for 8 cores it is doing worse with an W/ET efficiency of 0.97 than the best variant for 128 cores. It is relying on the fact that on average the cores would statically get about the same time requirement, which is not the case.

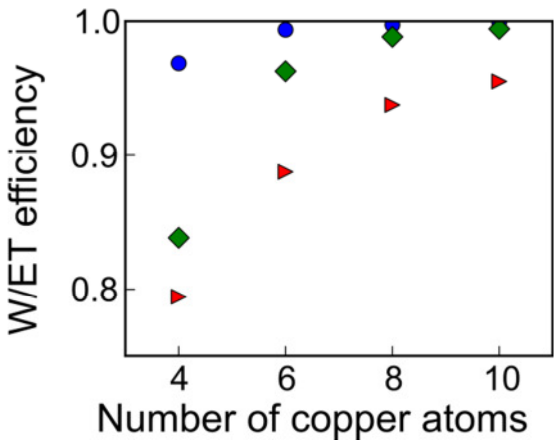

Figure 6.7: W/ET efficiency for varying numbers of copper atoms in the copper clusters on 128 cores. Dynamic load balancing with cost sorting (dots), dynamic load balancing without cost sorting (diamonds) and static task distribution with cost sorting (triangles). Averages over three runs.

When the amount of tasks is increased by increasing the cluster size, see Figure 6.7, all three methods improve when 128 cores are used. The distribution of the tasks is expected to stay more or less the same as in Figure 6.4, but there are more tasks of varying size. Thus comparing the efficiencies for the different system sizes makes sense. The order of the methods with respect to the efficiency stays the same. The cost sorted DLB variant is still the best, showing already for Cu_6 an efficiency better than 0.99. But for Cu_{10} the DLB variant without cost sorting is getting close in efficiency, yielding also a very good values above 0.99 (Figure 6.7).

The overhead related to the termination of the dynamic load balancing depends on the number of cores, but is independent of the number of tasks, thus the system size, and the sorting of the tasks. For one core it is too small to get a reliable measurement while it is about 0.1s for 128 cores. For calculations with up to 510 cores it is still ~1s only. It grows rather slowly. Thus in our case it is negligible. The time for the setup should be also negligible as it includes only the time to set some variables to their start values. There is no global communication required during setup. The setup time is considered in the total elapsed time of the loop, but is not measured separately. The other two types of overhead time are more interesting: the overhead due to load imbalance at the end of the algorithm and the overhead due to the scheduling itself. The time requirements for the scheduling of the tasks are increasing with both core number and system size, see Figure 6.8. On the other hand, the time requirement for scheduling per core decreases with the number of tasks per core (can be calculated by dividing the scheduling overhead and the number of tasks through the number of cores). For Cu_4 the scheduling time is still negligible for 128 cores, it amounts to 1.4 s. The requirements for the DLB with cost sorting method are always higher than for the static variant with cost sorting. The calculations with DLB with unsorted tasks require a nearly constant scheduling overhead, as it depends only on the system size. The scheduling time for the DLB with unsorted tasks is only slightly increasing, thus work stealing overhead alone is not responsible for the difference in the DLB with cost sorting and the static with cost sorting run.

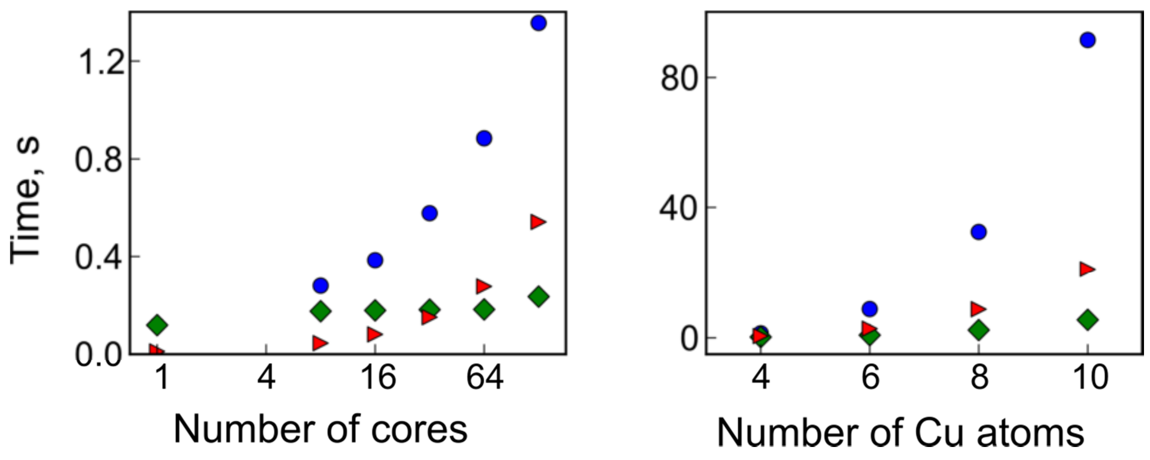

Figure 6.8: Time requirements for the scheduling. Left panel: for Cu_4 on different numbers of cores. Right panel: for various copper cluster sizes on 128 cores. Dynamic load balancing with cost sorting (dots), dynamic load balancing without cost sorting (diamonds) and static task distribution with cost sorting (triangles). Averages over three runs.

It was already indicated in Section 6.2 that the cost sorting is better optimized for a case, where the tasks are kept ordered; this is the case for the static variant with cost-sorting. The DLB variant with cost-sorting gets the task IDs more randomly, because of work stealing. The largest effect for the increase in the scheduling time for the static and the DLB variant with cost sorting is therefore the growing cost for determining which task belongs to the provided task IDs.

The increase of the scheduling time with the number of atoms, see Figure 6.8, is steeper than with the number of cores. One has to consider that the number of tasks increases much faster with the system size than the number of atoms, see Table 6.1. It is (formally) proportional to the power of four of the number of atoms. The increase of the time requirement for scheduling per tasks is nearly linearly, but with the increase of tasks with the number of atoms even for moderately sized molecules this cost is supposed to grow to noticeable size. Therefore the cost sorted DLB variant will not always be superior to the unsorted one.

The tests described above were also done for larger systems and core numbers. The scaling with core numbers was tested with Cu_{10} doing also calculations on 256 and 510 cores. Calculations distributed over several partitions met technical problems. The efficiency results are presented in Figure 6.9. The results look qualitatively the same as the results for Cu_4 (Figure 6.6). Again the DLB variant with cost sorting is close to the ideal results. The efficiency is not going down that much as in the case of Cu_4, thus it would be possible to use efficiently larger numbers of cores here. For 510 cores there are on average more than 200 tasks and 4.5 min computational time to spend on every core.

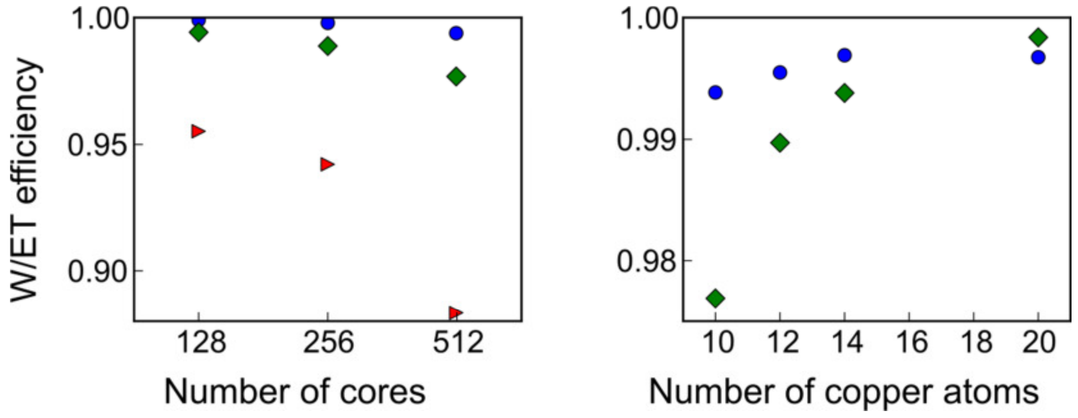

Figure 6.9: W/ET efficiency for Cu_{10} on 128 to 510 cores and W/ET efficiency on 510 cores for different sizes of copper clusters. Methods: dynamic load balancing with cost sorting (dots), dynamic load balancing without cost sorting (diamonds) and (only for Cu_{10}) static distribution of cost sorted tasks (triangles).

In a further test, the system size was increased up to Cu_{20} on 510 cores for DLB with and without cost sorting, see Figure 6.9. The efficiency of the DLB method with unsorted tasks was increasing much faster than for the DLB method with cost sorted tasks. With the latter approach, the efficiency is even slightly smaller for Cu_{20} than for Cu_{16}. The different increases of the efficiency lead to the fact that DLB with unsorted tasks is more efficient for Cu_{20} than the variant with sorted tasks whereas the opposite is true for systems smaller than Cu_{14}. The reason for this observation is the increasing cost for scheduling with the system size, as already been observed for the smaller clusters (Figure 6.8). Cu_{20} belongs to the already mentioned case where the scheduling overhead (through cost sorting) is too large for providing an improvement over the larger imbalance time of the DLB method on the unsorted list. Not only the real time of scheduling increases with system size, as observed for smaller systems, but also the relative contribution to the overall computation time (Figure 6.10). Already for Cu_{14} the scheduling costs more time than the load imbalance at the end. For this case the load imbalance of the unsorted tasks is still larger than the time needed for scheduling, thus for the Cu_{14} calculation the strategy with cost sorting is still more effective. For larger systems Cu_n, $n > 14$, cost sorting reaches its limit of efficiency.

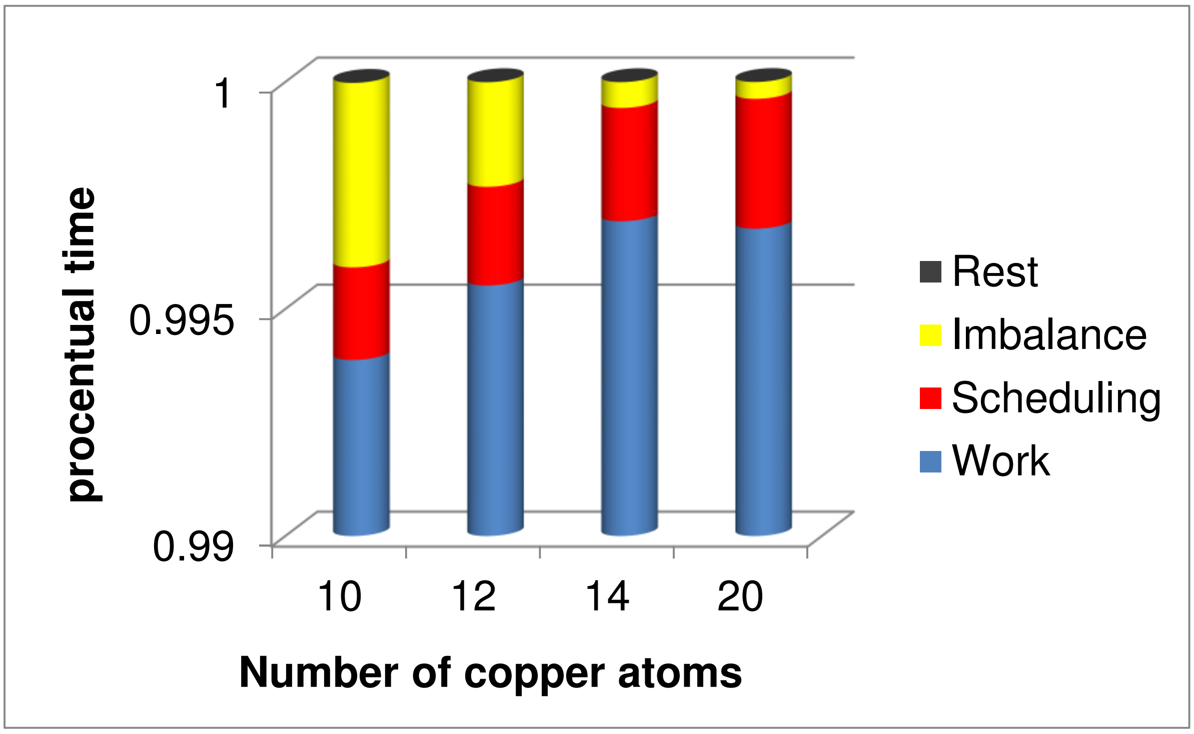

Figure 6.10: Distribution of elapsed time for the scheduling variant of dynamic load balancing with cost sorting for copper cluster with 10 to 20 atoms on 510 cores. The contribution of termination is not visible as its contribution is below 10^{-4} %.

Further optimizations of the calculation, which can lower the time required per task, can further restrict the area for applying DLB with cost sorting. For the 1672223 tasks of Cu_{20} there are already 6230 s spend for overall scheduling on 510 cores. For a static task scheduling with cost sorting, scheduling costs only 1297 s overall, about a factor of 5 less. The time required for scheduling for DLB without sorting the tasks, 170 s, is rather negligible compared to the scheduling effort of other approaches.

The timings (averaged over three runs) for all calculations are provided in Appendix B.

6.5 Cu_4 on SuperMIG

The tests on HLRB-II showed that the DLB variant with RMA can be very efficient on specific machines. A machine, where the DLB variant with threads, was the superior strategy compared to the RMA variant (Section 5.1) was SuperMIG. This machine allows core numbers as large as on HLRB-II. Thus SuperMIG is appropriate for a large test set on real systems using the thread variant.

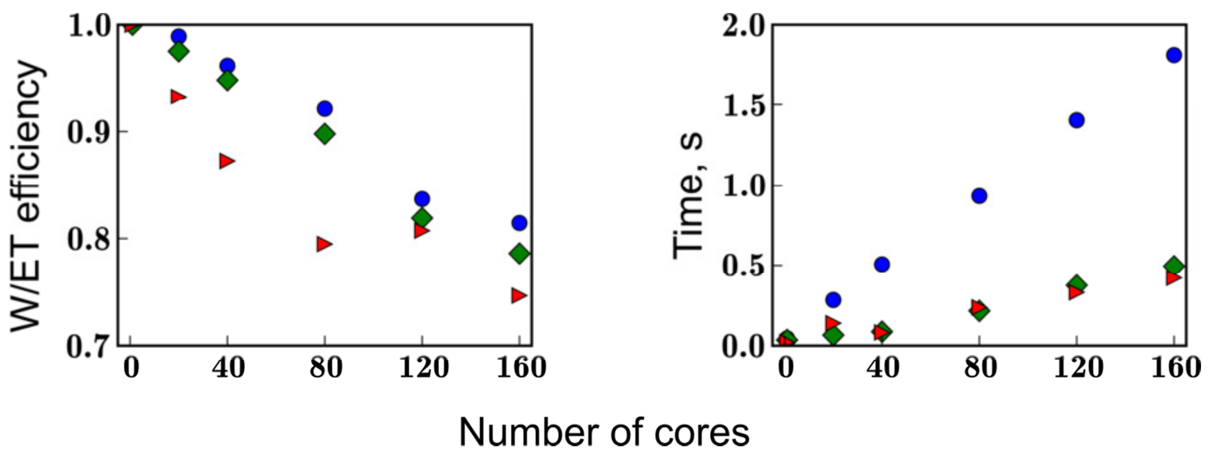

Figure 6.11: W/ET efficiency (left) and time requirement for scheduling (right) for Cu_4 on 1 to 160 cores of SuperMIG. Methods: dynamic load balancing with cost sorting (dots), dynamic load balancing with unsorted tasks (diamonds) and static scheduling with sorted tasks (triangles).

The core numbers in the tests on SuperMIG were increased in steps of 40 cores, which corresponds to one node on this architecture. The three strategies, DLB with cost sorted tasks, DLB with unsorted task list and static distribution of cost sorted tasks, were tested. All timings can be found in Appendix C. Runs with 1 core allow calculating the standard efficiency. However the standard efficiency was not a good measure for the current performance tests. The sum of working times increases significantly, at least 300s, between 1 and higher numbers of cores. As this effect was also observed for the static variant (with cost sorting), it should not be an effect of the DLB scheduler. The reason for this effect might be that the system treats a serial program differently. The effect was also present on HLRB-II, but was with less than 100s difference between the working times of serial and parallel calculations much smaller.

The overall time requirement for the work has decreased due to the computer hardware. Instead of requiring 1.2 s on average on HLRB-II, the time per task was reduced to about 0.8 s on SuperMIG. Thus, the time required for the mere working, which was around one hour on HLRB-II, compare Table 6.1, is decreased for SuperMIG by around 20%. The decrease in overall time together with the working time difference between serial and parallel calculations makes the standard efficiency a bad choice for observing the efficiency, instead the W/ET efficiency is exclusively used.

The W/ET efficiency might not be suitable for observing the thread variant of DLB as the time for working includes the times in which the helper thread of the DLB thread variant is running. However, this effect can be neglected because the real working times for core numbers larger than 1 were stable (thus only slightly and randomly varying) and for some of the calculations using DLB even smaller than for the static variant. This allows to calculate the W/ET efficiency using Eq. (6.2), see Figure 6.11. Comparing the results for SuperMIG with the efficiency of the calculations on HLRB-II, one sees that the relative order of the various methods is the same. As expected for a faster machine, the efficiencies drop faster with the CPU count for the SuperMIG calculations, Figure 6.11, than for the RMA variant on HLRB-II, Figure 6.6.

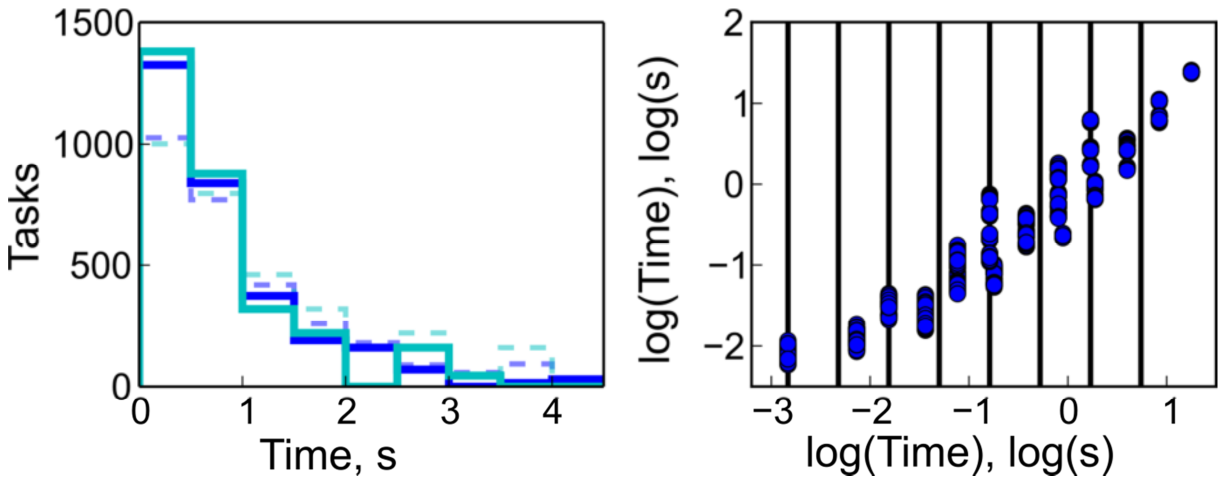

Figure 6.12: Distribution of the time requirements per task. Left panel: number of tasks according to working time. Calculations on SuperMIG (solid lines) and HLRB-II (dashed line). Dark: actual time requirements; light: mapped expectation times. Right panel: correlation of estimated and real working times on SuperMIG. The cost windows for the expected times are also shown (solid lines).

There is one variant, cost-sorted DLB, which is especially performing worse on SuperMIG than on HLRB-II. This is due to the load imbalance at the end. All other overhead timings measured gave comparable results. The reason for the more pronounced load imbalance is not fully understood. A cause for this difference in performance between HLRB-II and SuperMIG can be the varying time requirements for the tasks. Comparison of expected time to the real time requirements per task on SuperMIG, see Figure 6.12, is approximately as good as on HLRB-II. On the other hand, the number of tasks in the window of smallest cost were 1326 (see Figure 6.12) on SuperMIG (where the interval was 0–0.5 s) while there were 1796 on HLRB-II (where the interval was from 0–1 s). Thus the main reason for the drop in efficiency is most certainly only related to an unfavorable distribution. Another reason might be the fact that different variants of DLB were used for SuperMIG and HLRB-II. Thus, for example, the thread variant is not allowed to steal the last task from a core to ensure proper termination, see Subsection 4.2.2. However, on SuperMIG the DLB variant with threads is still preferable over the variant with RMA. This can be seen at the example of 40 cores, where the scheduling overhead (including the time for DLB to fetch the next task number from its storage) was for the thread variant less than 1‰ of the result of the RMA variant (scheduling times in Appendix C). They needed 0.5 s and 684.2 s, respectively.

The scheduling overhead on SuperMIG was quite similar to that on HLRB-II (Figure 6.8). For the cost sorted DLB variant it was nearly the same, see Figure 6.11. Only the calculations of DLB with an unsorted task list shows some increase in the time requirements.

Altogether using DLB for the redistribution of the tasks was as efficient on SuperMIG as it was on HLRB-II. Only the usage of the cost sorting in addition was becoming less efficient for high core numbers. Already before, cost sorting was computationally demanding and on the HLRB-II machine it might have profited immensely from a good balance of the task length. Still, for the small example of Cu_4 it provided in both cases a better performance than without. As the larger systems were shown to scale even better than the smaller systems, see Section 6.4, especially for the method with DLB on an unsorted list, the good scaling on the small example with up to 180 processes indicates that larger systems might scale even better. Thus cost sorting can be useful especially for smaller systems when a fast processing is required, e.g. in dynamic simulations.

6.6 Benchmarking on Four-Center Integrals with Symmetry Screening

A version of the four-center integrals (marked with tag AN Perf Test ERI4C) integrated in ParaGauss (V3.1.8 ALPHA 7) was used for tests on the Nehalem cluster (Subsection 5.1.2). The tests used for dynamic redistribution the thread variant of DLB, which was found to be superior on this cluster, see Section 5.1. All test calculations used results of the first SCF loop, in which the four-center integrals were calculated. The version of the four-center integrals, used in this test, is described in detail by Soini.[11]

The two loops, the first for calculating the tasks required for the prescreening and the second one with the remaining tasks, were merged into one loop. Thus all calculations run now over all tasks. In addition to the simple screening,[83] already applied in the Section 6.4 for the larger clusters, a screening due to the symmetry was introduced.[11] To test how this affects the scaling of the DLB-driven scheduler, a platinum cluster with 38 atoms in O_h symmetry was used as an example. This example had 67,611,006 tasks of which 1,467,266 were actually calculated. Thus due to the screening only about 2 % of the tasks needed to be calculated. On the Nehalem cluster these tasks required about 0.001 s calculation time on average. This is already very small, especially compared to the calculation time per task of about 1s for the previous examples. If all the omitted tasks are also considered, the time requirement per task is reduced to $2 \cdot 10^{-5}$ s. This is an average over all tasks which were scheduled. The scheduler using DLB required about $5 \cdot 10^{-6}$ s for determining a task ID. Thus, the overhead due to the scheduler was about 25 % of the computation time. Even if the load distribution would be perfect, the efficiency can only reach 80%. A static scheduler without any cost sorting, as a comparison, requires about $2 \cdot 10^{-7}$ s for scheduling a task. To use DLB efficiently, the relation of calculation time per task to the scheduling time had to be improved. By concentrating several of the tasks to a DLB scheduled task the average calculation time for a DLB task increases and the scheduling overhead per task decreases.[11] Additionally, for a serial run on one core, the DLB scheduling was removed completely by concentrating all tasks in a single DLB task.[11] The timings for calculations with a static scheduler, a scheduler using DLB for the task distribution and a scheduler using the concentrated tasks for DLB (200 per core) are available in Appendix D. For DLB on concentrated tasks the time for scheduling is less than 1% of the overall computational time. Calculations were performed with up to 24 cores on the Nehalem cluster. The observed load imbalance is smallest for the DLB variant working on all tasks, however the scheduling overhead is much larger than the imbalance overhead for the other methods.

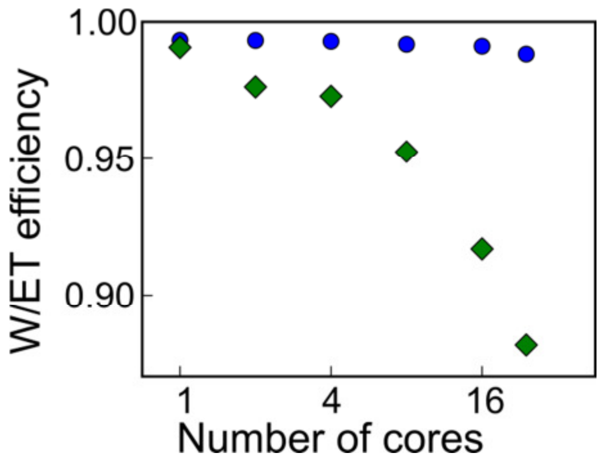

Figure 6.13: W/ET efficiency, see Eq. (6.2), for calculations with different core count for Pt_{38} of the four-center integral part of ParaGauss on a Nehalem cluster. A static variant (diamonds) and a variant using DLB for concentrated tasks (dots) are shown.

The SCF cycle in the calculation with Pt_{38} was not converged but rather aborted after the third iteration, thus including two iterations with four-center integrals and one without. For these iterations the four-center integrals required with about 3000s already 98% of the time of the whole SCF-cycle. In this regard on can already see, that ensuring a good scaling with increasing process numbers for the four-center integrals can easily become a very important aspect of ensuring a good scaling of the complete quantum chemistry program. This is especially the case as the four-center integrals are scaling with N^4 for N atoms, while the rest of the SCF-cycle only scales with up to N^3.

The dynamic redistribution of the tasks with DLB is very useful. Its W/ET efficiency (Figure 6.13) is strongly improved compared to the variant with static distribution. For 24 cores DLB on concentrated tasks still has a performance of 0.99, while the static distribution has 0.88. Thus, one only has to ensure that the ratio of the times, for the actual calculation and the DLB overhead required to assign a new task, is reasonable. The optimum ratio will depend on the computer architecture and the given structure to be calculated, but the example showed that, with the current implementation, it is possible to get the scheduling time below 1% of the overall time without rendering the performance inefficient.

7. Conclusions and Discussion

Work stealing algorithms offer favorable scaling of parallelization for complex tasks.[45-46] These methods avoid the bottleneck of a master process which has to feed tasks to many "slaves".

In order to get a simple interface to this work stealing algorithm, it is necessary to make strong use of methods and functionalities of the MPI-2 standard. Two methods, which are easily exchangeable, were created, building on such different functionalities as remote memory objects (RMA) or running a program with multiple threads. It turned out that these two methods perform differently on various computer architectures, thus offering complementary advantages. The method with remote memory objects works best when specific hardware support is provided. The thread method requires a thread safe MPI. This is not always given or it is not as well maintained as an implementation without threads. However, one may expect that in the future such MPI functionalities will be better maintained. Scalability tests with a simple dummy set of tasks unraveled differences of the various computer architectures. The thread method performed better on most of the tested architectures. Only HLRB-II having RMA hardware support shows a clear superiority for this method.

There are three modules in the quantum chemistry program ParaGauss, where work stealing algorithms are currently in use. For all of them tests demonstrated that dynamic load balancing routines scale well. The case of the numerical integrals illustrated that tasks of equal size are easily parallelized without suffering from an overhead of the scheduling algorithm. This was tested for up to 16 cores on the Linux Nehalem cluster for Pd_{19} in octahedral symmetry. The real benefit of the dynamic load balancing routines shows up if the tasks have different sizes. Sorting them from large to small increases the performance of the module, as it allows a finer load balancing towards the end. This results in a good improvement for the three-center integrals. For the four-center integrals (tested in a standalone version) it was possible to reach an efficiency of 99% on 520 cores on HLRB-II for the largest example tested, Cu_{20}. This test was done with the RMA variant of DLB, the thread variant of DLB reached an efficiency of 82% on 160 cores of SuperMIG of the LRZ. For this test only the significantly smaller cluster Cu_4 was used.

In cases like the four-center integrals, where it is not possible to presort the tasks, an approximate sorting at runtime is only favorable if it does not require too much time. Thus for the tests on the standalone version with copper clusters this was the case for Cu_4 to Cu_{14}, while for Cu_{20} the work stealing variant, which did not use sorting, was preferable. The usability of the sorting might be further reduced if the time requirements for the tasks do not add up nicely. Therefore, for large system sizes and core numbers, which both increase the overhead of the sorting, a method without sorting is much more preferable. This will become more important when the calculation time per tasks is decreased by further improvements of the algorithm and when more tasks are skipped by screening. In this case even for smaller core counts and smaller system sizes, the distribution of unsorted tasks is preferable to sorting them on the fly.

If the time requirement for a single task gets small in comparison to the time required to get a task ID, it is preferable to bundle several tasks into a single work package to be scheduled. This is for example required for a version of the four-center integrals with improved screening. Especially if symmetry reduction results in a large amount of tasks requiring very small time for calculation, such bundling of tasks is mandatory.

Part II: Convergence Acceleration of the Self-Consistent Field Procedure

8. The Self-Consistent Field Procedure

The self-consistent field (SCF) procedure is a central part of density functional codes like ParaGauss. It solves the subproblem of adapting the electron density and the derived energy to the geometry of the system. This is achieved by repeatedly solving the Kohn–Sham eigenvalue problem, compare Section 1.1. Unfortunately Eq. (1.5) is not a closed form expression. Thus instead of a direct calculation it is solved iteratively.

In each SCF iteration i first the Kohn–Sham matrix $\mathbf{h}_{\mathrm{KS}}^{(i)}$ is built (see Section 1.1), for all matrix elements related to the contributions of Eq. (1.4) and the basis functions χ_α of Section 1.1. This matrix requires the electron density matrix $\mathbf{D}_i = \mathbf{C}_i n_i \mathbf{C}_i^T$ for the matrix $\mathbf{C}_{i-1}$ of eigenvectors; n_i is the diagonal matrix of occupation numbers of the various Kohn-Sham levels from the previous iteration i–1. The eigenvalue problem, Eq. (1.7), will provide the energies ε_i and expansion coefficients $\mathbf{C}_i$ of the eigenvectors. To obtain the approximate charge density $\tilde{\rho} = \sum h_i u_i$ for the basis vectors u_i (see Section 1.1), the charge fit coefficients h_i have to be selected accordingly. For this purpose, the Coulomb self-interaction of the density difference $\rho - \tilde{\rho}$ between the exact density, represented by the density matrix $\mathbf{D}_i$, and the approximate density is minimized (see Section 1.1).[5, 17, 20] The resulting charge fit coefficients h_i and the expansion coefficients $\mathbf{C}_i$ are used for the next iteration.

The first iteration starts with a zero density. The SCF cycle is repeated until convergence. The convergence is tested in ParaGauss by three different measures, comparing the value with the result of the last iteration: the energy, the charge fit coefficients and the density matrix **D**. All three measures have to get smaller than the corresponding convergence criteria.

9. Convergence Acceleration

The convergence of the SCF procedure might meet some challenges. In the Kohn–Sham formalism the errors of the electronic self-interaction energy are different for occupied and unoccupied orbitals.[21] The gap between the highest occupied and the lowest unoccupied orbital (the HOMO-LUMO gap) is therefore often too small. In the SCF part, in general, the orbitals with the smallest eigenvalues are occupied. Thus, a too small a gap can lead to a case where the occupation of some of the orbitals around the HOMO-LUMO gap changes frequently during the SCF process. This is especially common for systems involving metal centers or clusters, where the gap is usually very small or even absent. Convergence of the SCF procedure is less stable as a result. Furthermore the simple approach of reaching this convergence as described in Chapter 8 is not expected to converge fast. This leaves room for improvements also in the cases where orbital occupations are more stable.

For this purpose ParaGauss offers several convergence acceleration methods. The theory behind the implementation of (dynamic) damping and the perturbation theory approach are only shortly summarized in this thesis (Sections 9.1 and 9.2). It has been described already in detail elsewhere.[90] The current thesis concentrates on the direct inversion in the iterative subspace (DIIS) method which has been newly implemented. It is described together with implementation details in Chapter 10. This method is reported to be very efficient,[91-92] even if this might be only expected for situations close to convergence, making it essential that other methods are available as a start-up procedure. Another difference to the other method implemented in ParaGauss, which all operate on the charge fit coefficients, is a variant of the DIIS method which operates on the Kohn–Sham matrix instead.

9.1 Damping

The damping algorithm is realized by mixing the current value of an object iterated in the SCF procedure, in a cycle with the value from the last iteration.[90, 93] The mixing scheme can be applied to various objects, e.g. the Kohn–Sham matrix, the density matrix or the orbital coefficients. ParaGauss mixes the coefficients of the approximate charge density.

To carry out the mixing, the SCF cycle i is interrupted at the point after a specific (scalar) object A^i has been calculated. Instead of continuing with the output of this iteration A^i_{out}, a mixed input object A^{i+1}_{in} for the next cycle is generated. In the simplest case it is a fixed mixture of the input and output of the last iteration: $A^{i+1}_{\text{in}} = (1-\beta)A^i_{\text{in}} + \beta A^i_{\text{out}}$ with a mixing parameter $0 < \beta < 1$.

ParaGauss uses this static mixing with a fixed parameter β as a start-up procedure for the dynamic damping.[90, 94] In the latter approach the value β is calculated anew in every cycle using input and output values, A^i_{in} and A^i_{out}, of the last two iterations. The converged value of A should be a fixpoint. Assuming a linear relation between the input and output of the components of A in the last iterations, it is possible to determine parameters $\beta = 1/(1-m)$ and the corresponding fixpoint from the slope $m = \left(A^i_{\text{out}} - A^{i-1}_{\text{out}}\right) / \left(A^i_{\text{in}} - A^{i-1}_{\text{in}}\right)$. For the charge fit coefficients, which are provided as a vector, the average over all component specific first-order derivatives m$_k$ is taken to determine the overall mixing parameter. Ultimately, the values of β are restricted to the range between 0.1 and 1 to ensure progress.

9.2 Perturbation Theory

Convergence acceleration by means of perturbation theory is an approach to reduce the switching of occupied and unoccupied orbitals between SCF iterations.[21, 95] This gain in stability is the only improvement to be expected by this method. The method will have an accelerating effect on the convergence only in situations where such situations occur, for example for large metal clusters. Applying perturbation theory allows and requests that it is used together with other acceleration methods. For example already early on perturbation theory was supposed to be used together with mixing.[21] Unfortunately this combined application does not need to enhance the convergence in all cases, but it might disturb the converging of other acceleration methods used instead of mixing.

The method uses a perturbation theory approach for energy levels of the orbitals, especially those which are near to the Fermi level. Instead of adapting the energies themselves the approximate charge density is created with a correction term $\Delta\tilde{\rho} = \sum \Delta h_i u_i$ (using the basis vectors u_i of the approximate charge density), by a corresponding change Δh_i to the charge fit coefficients ($h_i^* = h_i + \Delta h_i$). This gives an additional term to the Coulomb component of the Kohn–Sham operator, $\tilde{V}(\mathbf{r}) = \int d^3\mathbf{r}' \Delta\tilde{\rho}(\mathbf{r}')/|\mathbf{r}-\mathbf{r}'|$, and with this to the energy, with the side effect that the coefficients for the orbital functions φ_k have to be adapted. If the change in the energy is considered up to second-order perturbation theory, it results in a quadratic term which penalizes large changes in the approximate charge density:

$$\Delta E^{(2)} = \sum_{k,l}\sum_{i,j} \Delta h_i \Delta h_j (n_k - n_l) \frac{[u_i \,|\, \varphi_k\varphi_l][u_j \,|\, \varphi_k\varphi_l]}{\epsilon_k - \epsilon_l} \tag{9.1}$$

Here the notation $[u_i \,|\, \varphi_k\varphi_l] = \int d\mathbf{r}_1 d\mathbf{r}_2 u_i(\mathbf{r}_1)\varphi_k(\mathbf{r}_2)\varphi_l(\mathbf{r}_2)/(\mathbf{r}_1 - \mathbf{r}_2)$ was used. The last part of the expression on the right hand side of Eq. (9.1) looks similar to the expression for the elements of the matrix **G**, see Eq. (1.9) :

$$G'_{ij} := -\sum_{k,l} (n_k - n_l) \frac{[u_i \,|\, \varphi_k\varphi_l][u_j \,|\, \varphi_k\varphi_l]}{\epsilon_k - \epsilon_l}. \tag{9.2}$$

The perturbation theory approach is used exclusively for the determination of the charge fit coefficients. Other implementations do not seem to be that successful.[21] Introducing the perturbation theory term due to the second-energy correction, Eq. (9.1), effectively leads to replacing the matrix **G**, in the calculation of the approximate charge density expansion coefficients in Eq. (1.8) by a sum $\mathbf{G} + \mathbf{G}'$.[21] In ParaGauss the orbitals φ_l and φ_k used in perturbation theory are restricted to only a few in a small energy range around the HOMO-LUMO gap.

10. Direct Inversion in the Iterative Subspace

10.1 General Method

The direct inversion in the iterative subspace (DIIS) method is an extrapolation method to accelerate iterative searches of fixpoints with respect to some parameter.[12, 91-92] In the current context the minimal energy with respect to the electron density is sought for. The general concept of the method is to extrapolate one of the quantities which is iterated in the fixpoint search in a subspace obtained before by an iterative procedure.

The DIIS method can be applied in the SCF approach to several variables P, such as the Kohn–Sham matrix or the expansion coefficients of the approximate charge density. The method is supposed to work well in the quadratic region around the stationary point.[91] Far from this region it is often not very effective.[96] Therefore it is often applied only together with another convergence acceleration method which switches to DIIS in the last phase of convergence.

In the DIIS method, one replaces the current value of a parameter $\mathbf{P}_{i+1}$ by an extrapolation determined from values of previous iterations similar as the mixing approach (Section 9.1). However, instead of using only the input parameter $\mathbf{P}_i$ from the previous iteration, DIIS uses values from m preceding iterations of $\mathbf{P}_j$ to build a linear combination. To find the best DIIS linear combination

$$\mathbf{P}_{i+1} = \sum_{j=1}^{m} a_j \mathbf{P}_{i-m+j} \tag{10.1}$$

with coefficients a_j an error vector $\mathbf{e}_j = \mathbf{e}(\mathbf{P}_{i-m+j})$ associated with each $\mathbf{P}_{i-m+j}$ is minimized using the square of the L$_2$ norm, $\frac{d}{da_k}\left|\sum_{j=1}^{m} a_j \mathbf{e}_j\right|^2 = 0$. A practical choice for the error vector is to use the residuum vector which should vanish at convergence:

$$\Delta\mathbf{P}_i = \tilde{\mathbf{P}}_{i+1} - \mathbf{P}_i \tag{10.2}$$

where the parameter $\mathbf{P}_i$ as input results in the value $\tilde{\mathbf{P}}_{i+1}$ of the quantity **P** in the next SCF iteration (before extrapolation by DIIS). For keeping the parameter $\mathbf{P}_{i+1}$ meaningful the DIIS parameters a_j have to be normalized:

$$\sum_{j=1}^{m} a_j = 1 \tag{10.3}$$

To minimize the error vector, including the constraint of Eq. (10.3) as a corresponding Lagrange multiplier λ, one derives a system of linear equations by the Lagrange method with a Gramian matrix $B_{ij} = \mathbf{e}_i \cdot \mathbf{e}_j$ augmented by a row and a column:

$$\begin{pmatrix} B_{11} & B_{12} & \dots & -1 \\ B_{21} & \dots & \dots & -1 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & B_{mm} & -1 \\ -1 & -1 & -1 & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_m \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ -1 \end{pmatrix} \tag{10.4}$$

The error vector given a as residuum vector, as described in Eq. (10.2), is a suitable choice for the charge fit coefficients, the Coulomb matrix or the density matrix. This residuum vector vanishes at the convergence of the SCF cycle.

For the Kohn–Sham matrix $\mathbf{h}_{\mathrm{KS}}$ as recurrence variable **P**, however, another definition of the error vector is proposed which was suggested to be superior (the new error vector can be used more efficiently and the computational cost is lower):[12]

$$\mathbf{e}(\mathbf{h}_{\mathrm{KS}}) = \mathbf{h}_{\mathrm{KS}}\mathbf{DS} - \mathbf{SDh}_{\mathrm{KS}} \tag{10.5}$$

This definition exploits the fact that the self consistent Kohn–Sham matrix $\mathbf{h}_{\mathrm{KS}}$ commutes with the density matrix **D** at convergence. The commutator relation can be derived from the definition of the density matrix and Eq. (1.7) considering that at convergence the iteration number is unimportant. To formulate this vector independent of the normalization convention for basis functions, it is transformed into an orthogonal basis using the overlap matrix **S**:[12]

$$\mathbf{e}'(\mathbf{h}_{\mathrm{KS}}) = \mathbf{S}^{-1/2}\mathbf{e}(\mathbf{h}_{\mathrm{KS}})\mathbf{S}^{-1/2} \tag{10.6}$$

A technique to increase the performance of DIIS is to scale the diagonal elements of the error matrix **B** by a factor (1 + *d*), where *d* is a small positive number.[92]

10.2 Implementation in ParaGauss

ParaGauss contains two implementations of the DIIS algorithm. One implementation operates on the Kohn–Sham matrix using the error vector of Eqs. (10.5) and (10.6). The second implementation, which has proven to be less useful, employs the residuum vector as error vector and operates on the charge fit coefficients, see Eq. (10.2). In the following DIIS-KS will always refer to the implementation, that operates on the Kohn–Sham matrix, while the other implementation operating on the charge fit coefficients will be called DIIS-cc.

The size of the Gramian matrix **B**, is limited by the maximal number of stored error vectors and variables of previous iterations. If the SCF algorithm proceeds to more iterations the oldest of the stored entries is replaced. The methods start only after several iterations with the default convergence acceleration strategy, the dynamic damping method, see Section 9.1. The starting iteration for DIIS is controlled by a threshold for the maximum component of the current error vector, indicating that a regime where DIIS should be able to work is reached. As this threshold might be chosen too large for a specific system and large fluctuations might occur, it is possible to return to the original convergence scheme. It is not desirable to switch between dynamic damping and DIIS too often. Thus one returns to dynamic damping only after a significant increase of the norm of the error. The threshold for returning to the default acceleration strategy therefore is set to 10 times the threshold for initializing DIIS.

The DIIS methods start first with accumulating a specific number of pairs ($\mathbf{e}_i$, $\mathbf{P}_i$) before an extrapolation is done. DIIS-KS has an option to step-in every *m*th iteration, only collecting error vectors $\mathbf{e}_i$ and corresponding parameters $\mathbf{P}_i$ in between. The number of stored pairs, the initialization threshold, the number of steps to collect data before the mixing starts and the maximal number *m* are parameters of the current implementation. These parameter are shared by DIIS-KS and DIIS-cc.

The two methods, DIIS-KS and DIIS-cc, can be alternated or used in parallel. Alternation is controlled by a threshold for the maximal component of the error vector. It is possible to stop DIIS-KS and either to return to the original method or to continue with DIIS-cc, for the benefit of getting the charge fit coefficients better converged.

11. Performance Tests and Results

Test examples demonstrating the applicability of DIIS-KS and DIIS-cc were chosen from two groups of applications: The first one are actinide complexes. These systems are represented by an actinide ion and the water molecules of the complex treated by the all-electron quantum chemistry. The first member of this group was neptunyl, $[NpO_2(H_2O)_5]^+$, see Figure 11.1. Applying D_{5H} symmetry constraints, this corresponds comprises 4 unique atoms. A second example of this group is a charged americium complex, aquated Am^{3+} without symmetry constraints (C_1 symmetry). The complex included eight water molecules. The second group of examples are palladium clusters in O_h symmetry with different numbers of atoms, see Figure 11.1. The basis sets for all test systems are provided in Appendix H.

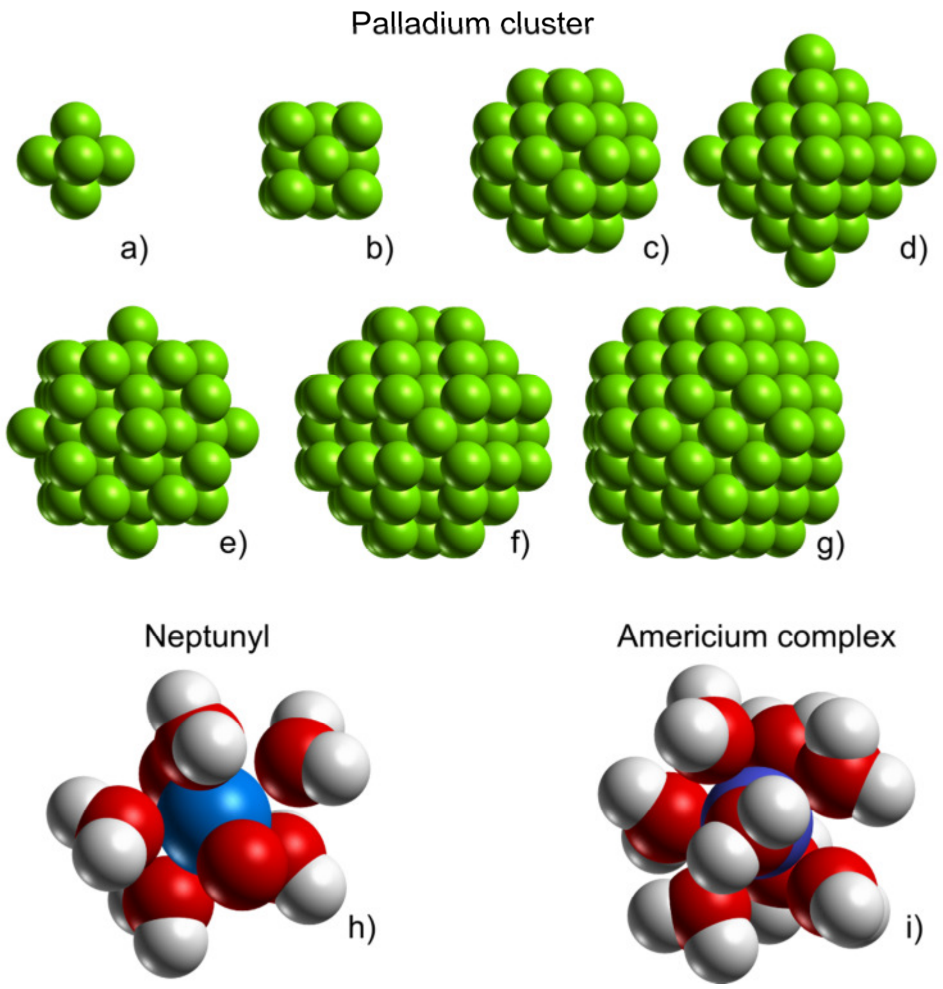

Figure 11.1: Sketches of the geometries of evaluated test examples: a) Pd_6, b) Pd_{14}, c) Pd_{38}, d) Pd_{44}, e) Pd_{68}, f) Pd_{92}, g) Pd_{116}, h) $[NpO_2(H_2O)_5]^+$, i) $[Am(H_2O)_8]^{3+}$.

The evolution of the three convergence measures, working on the density matrix, on the charge fit coefficients, and on the energy differences, as the SCF procedure progresses are used for the following tests. The energy measure, the absolute value of the difference of the current energy to the value of the last iteration is given in the unit of energies and therefore well defined. The measure for the charge fit coefficient uses next to the charge fit coefficient matrix *h* the diagonal elements of the overlap matrix of the basis functions of the approximate charge density in Coulomb norm $G_{ij} = [u_i \,|\, u_j] = \int d\mathbf{r}_1 d\mathbf{r}_2 u_i(\mathbf{r}_1) u_j(\mathbf{r}_2) / (\mathbf{r}_1 - \mathbf{r}_2)$. The resulting measure is given as $\Delta h = \max_k |\, \Delta h_k \sqrt{G_{kk}} \,|$. This measure is invariant to changes of the basis functions χ_α of the approximate charge density. The third measure, using the density matrix of the orbital coefficients, is given (in cases without spatial symmetry constraints) as $\Delta D = \max_{i,j} |\, \Delta D_{ij} \,|$. In case of spatial symmetry constraints the measure is created for each irreducible representations and the overall measure is given as the maximum of them. In contrast to the other two measures the values of the density measure depend on the choice of the orbital basis functions χ_α. This results in some arbitrariness regarding the magnitude of the measure. In cases where these orbital functions are normalized, the measure is normalized as well. However in a general case this does not need to be the case as the implementation used for ParaGauss does not rely on normalized orbital functions. Thus the user should be aware that for basis sets chosen with unfortunate norms the convergence criteria for the density measure might provide misleading results.

Furthermore the convergence is usually limited. As DIIS is not intended to tighten the convergence, that can be achieved, but rather the rate of convergence, the focus of the following test cases is on the latter. Thus for observing the behavior of the different convergence accelerators the most important values in the following will be the limits for the convergence measure and the convergence rate required to reach these limits. Reaching a special convergence threshold, as required for production calculations, will not be in the focus of the present discussion.

11.1 Convergence Behavior for Small Examples

The SCF convergence behavior was inspected for two examples with runs including 200 iterations for various convergence acceleration techniques. All these techniques start with a dynamic damping approach. The dynamic damping phase, see Section 9.1, was preceded by 5 cycles with a static mixing with coefficient $\beta = 0.5$. The mixing coefficient could be chosen that large as the test examples are rather simple. After the maximal component of the error vector reached the threshold of 0.1 Hartree the procedure switched to either DIIS-KS or DIIS-cc. For none of the test examples in this section perturbation theory was used. As test examples the neptunyl ion and a small palladium cluster with 14 atoms were taken, see Figure 11.1. Thus one small example of each class was treated.

The evolution of the three convergence measures, the norm of the density matrix, of the charge fit coefficients, and of the energy differences, as the SCF procedure progresses, are given for the neptunyl ion in Figure 11.2. Convergence measures reach a limit for all methods though, for the iterations towards the end all methods seem to fluctuate slightly around that limit. For the density convergence this limit was around 10^{-12} in relative units for all methods, as measured by the average over the last 20 iterations. The value of that limit for the energy is between 10^{-10} and 10^{-11} Hartree. The limit for the charge fit coefficient convergence measure is substantially lower for the DIIS-cc acceleration method. Indeed while the convergence limit for the DIIS-KS or without DIIS was $6 \cdot 10^{-5}$ and $5 \cdot 10^{-6}$ in relative units, the DIIS-cc method reaches the limit at around 10^{-18} to 10^{-20}. Convergence thresholds of 10^{-8} Hartree and 10^{-9} for energy and density measure, respectively, are more than sufficient in the present context. Only for the charge fit convergence measure, where, in practice, criteria between 10^{-4} and 10^{-5} are used, the limits are rather near to the desired convergence.

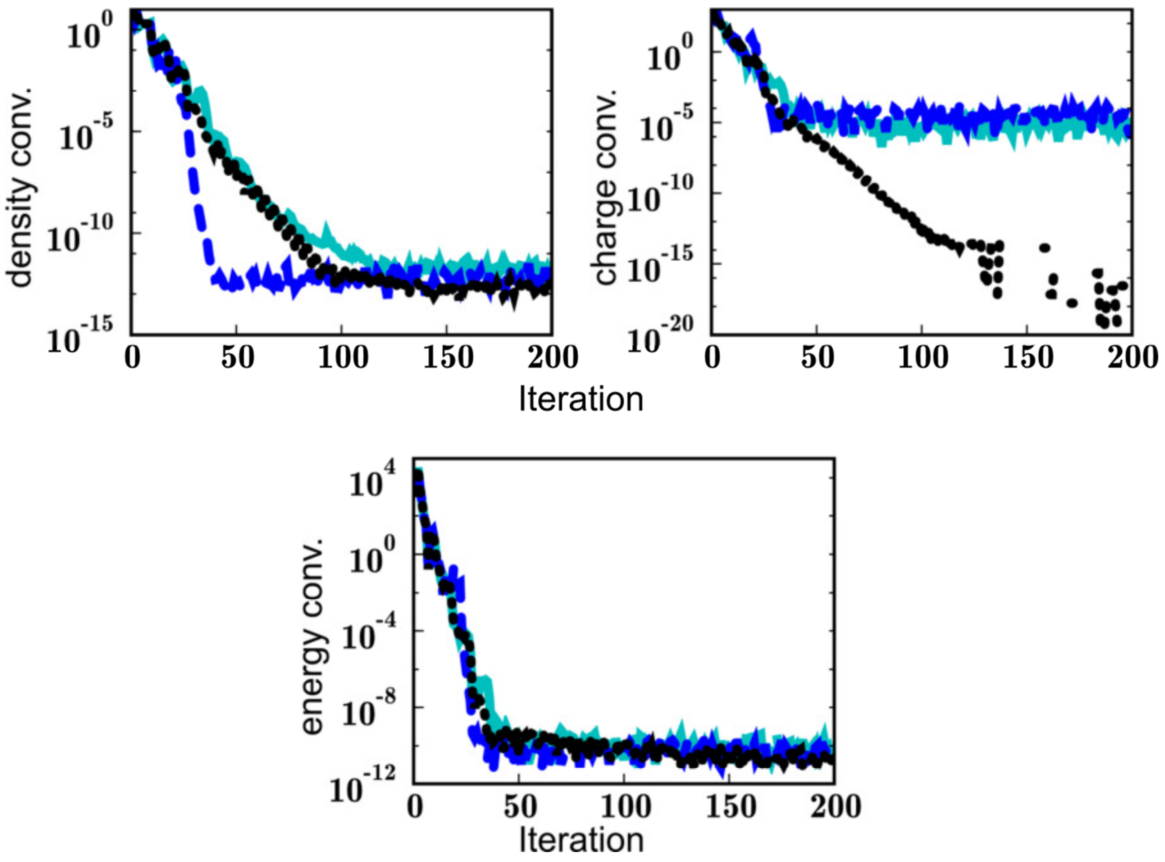

Figure 11.2: Convergence criteria evolution during calculation of $[NpO_2(H_2O)_5]^+$. The density and the charge convergence criteria are in relative units, the energy convergence criterion is in Hartree. Three calculation techniques are compared: with dynamic damping (solid), DIIS-KS (dashed) and the DIIS-cc methods (dotted) after meeting the DIIS initialization threshold.

The same overall behavior of the convergence measures can be seen for the Pd_{14} example in Figure 11.3. All convergence acceleration methods approach a limit with small fluctuations. For the energy criterion all calculations fluctuate about a value below $5 \cdot 10^{-10}$ Hartree. Also the density convergence limit is not much different for the three methods. For the last 20 iterations the average of the density matrix changes is between $3 \cdot 10^{-10}$ for the dynamic damping method against $4 \cdot 10^{-11}$ and 10^{-11} for the DIIS-KS and the DIIS-cc variants, respectively. These values are again in relative units. Again, the behavior of the charge fit coefficients stands out. While DIIS-KS and the dynamic damping methods reach a limit of the order of 10^{-7} and 10^{-6} in the relative units, the limit for the DIIS-cc method is significant lower. For both test examples, Pd_{14} and neptunyl, the clear difference in behavior related to the convergence limit of the DIIS-cc and the DIIS-KS method might be explained by the fact that DIIS-cc operates directly on the charge fit coefficients while the DIIS-KS method affect convergence of the charge fit coefficients only indirectly. The DIIS-cc method is therefore by design explicitly minimizing the convergence measure of the charge fit coefficients.

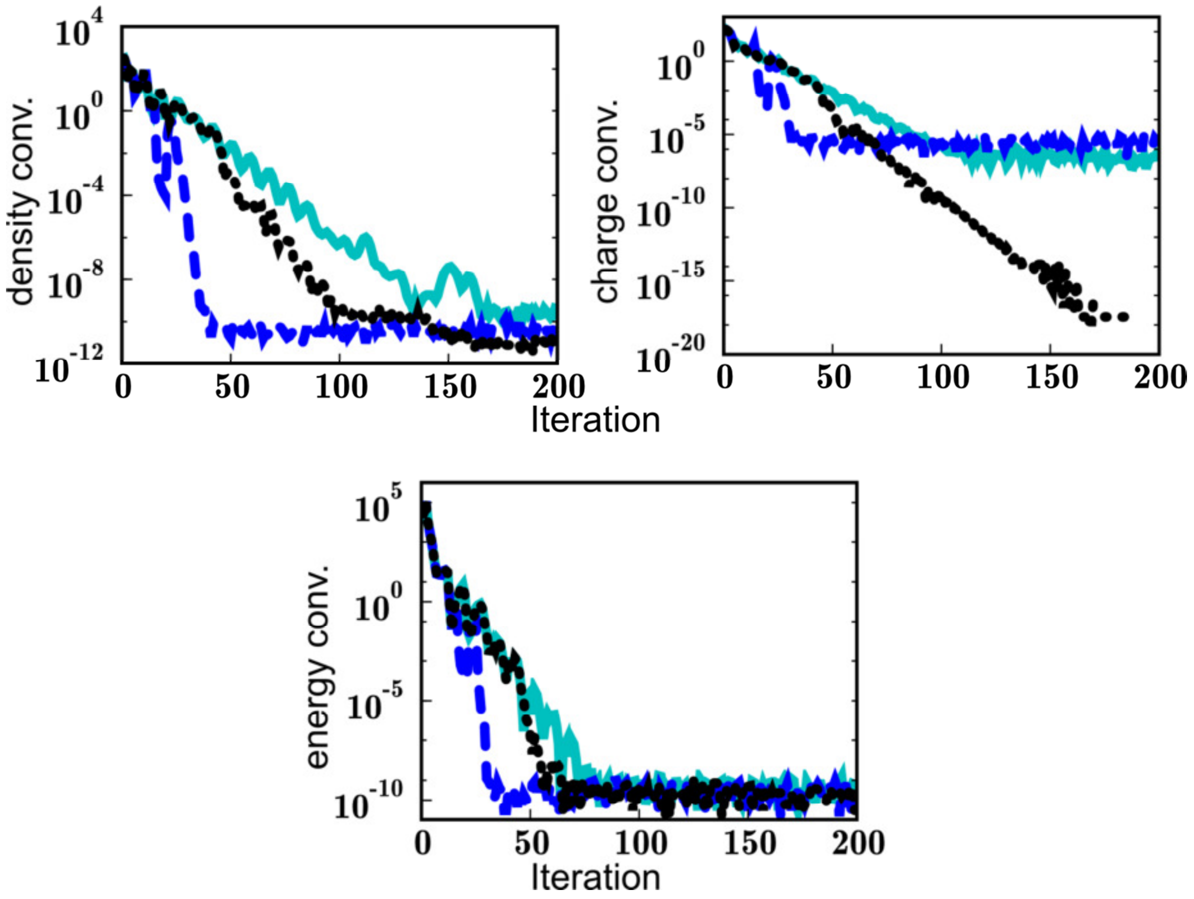

Figure 11.3: Convergence criteria evolution during calculation of Pd_{14}. Shown are convergence criteria for the density (in arbitrary units), the charge fit coefficients (in arbitrary units) and the energy (in Hartree). Three different calculations are compared: with dynamic damping (solid), DIIS-KS (dashed) and the DIIS-cc methods (dotted).

This convergence behavior demonstrates that in case a tight charge fit coefficient convergence is required, it is favorable to use DIIS-cc. Unfortunately DIIS-cc is not the best method for the convergence if considering the other properties or measuring different effects like the convergence rate. Indeed, the DIIS-KS method shows the steepest decrease for all convergence criteria. In fact, the switch over from the dynamic damping acceleration technique to the DIIS-KS method takes place slightly earlier than in the case of DIIS-cc. However, because of the different convergence rates it is not expected that tuning the starting point for the DIIS-cc method could change its overall performance. Another fact should be also mentioned: at convergence the total energies of the calculation with the DIIS-KS method and the one with the traditional dynamic damping technique differ by about 10^{-9} Hartree. The same order of differences appeared for all the other calculations, which are done for further tests with DIIS-KS within this section. However the total energies of the calculation converged by DIIS-cc differ by about 10^{-6} Hartree from these results. This means that DIIS-cc converges to a slightly different solution, a fix point for the charge fit coefficients.

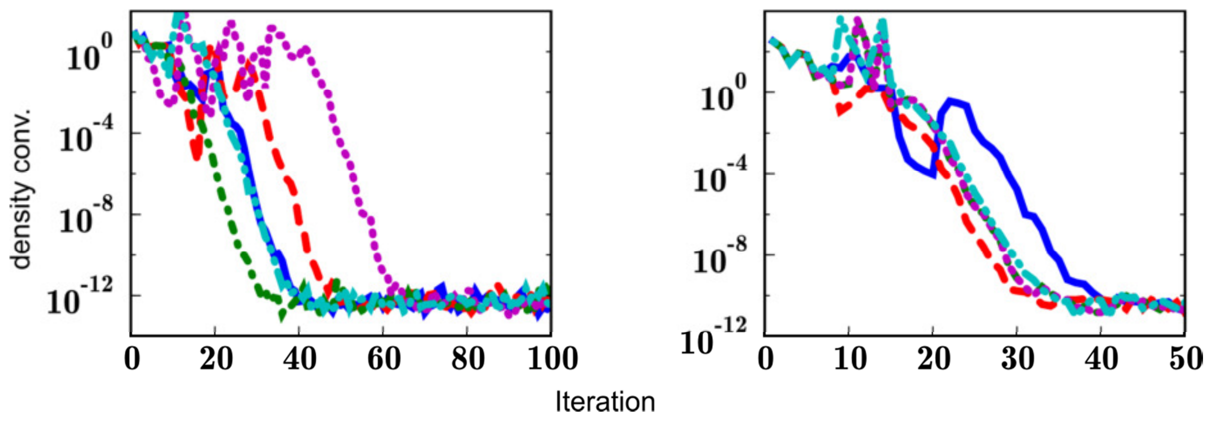

Figure 11.4: Evolution of the relative density convergence. The curves show the behavior for various thresholds at which DIIS-KS starts: 0.1 (solid), 0.3 (dashed), 0.5 (dash dotted), 0.8 (dotted), 1.0 (dash dot dotted). Left panel: $[NpO_2(H_2O)_5]^+$; Right panel: Pd_{14}.

DIIS should only work well for convergence acceleration towards the end of the calculation. Already in its first description it was explained that it cannot be expected to be useful for the initial convergence.[12] Changing the initial convergence threshold, after which the program switches to DIIS, should affect the convergence behavior. Too large a value will force DIIS to work in an unfavorable region, maybe even preventing convergence. Too small a value will reduce the overall speed of convergence. For both test cases, the neptunyl complex and Pd_{14}, different threshold values in the range between 0.001 and 1 Hartree were tested for the DIIS-KS method. Some of these curves are shown in Figure 11.4 for the density convergence. For each test example all calculations ended with the same convergence limit. The convergence rate is also the same after DIIS-KS steps in. The main difference in the evaluation of starting thresholds is the iteration number of the start of convergence.

Especially the calculation with 0.8 Hartree as the initiation threshold of the DIIS-KS method had difficulties in the beginning to converge, showing a fluctuating behavior that seemed to indicate that the threshold was chosen too large.

As all calculations show more or less the same convergence rate in the region where DIIS-KS works efficiently, i.e. after the fluctuations have died off, a good measure for comparing various initiation thresholds is to compare the iteration in which a given value for the convergence criteria was reached. A value of the convergence criteria well below the fluctuations at the start and above the convergence limit at the end should be used. Table 11.1 summarizes the number of iterations required to reach the energy criterion of 10^{-8} Hartree (a common value used for production calculations) in calculations with different initiating thresholds for DIIS-KS for both test examples. Note that the case with exclusively dynamic damping (compare Figure 11.2 and Figure 11.3) is given as threshold 0 Hartree in Table 11.1.

Table 11.1: Number of iterations to converge the energy to 10^{-8} Hartree for two test systems and varying initial thresholds for DIIS-KS.

Threshold	$NpO_2(H_2O)_5^+$		Pd_{14}	
in Hartree	DIIS start	conv.	DIIS start	conv.
0	–	37	–	64
0.001	20	33	46	50
0.01	19	28	37	39
0.025	12	33	22	39
0.05	11	28	15	26
0.06	11	28	15	26
0.07	11	28	15	26
0.08	11	28	15	26
0.09	11	28	14	30
0.1	11	28	14	30
0.15	11	28	13	32
0.2	11	28	13	32
0.3	10	35	8	23
0.4	8	19	8	23
0.5	8	19	7	25
0.6	8	19	7	25
0.7	4	51	7	25
0.8	4	51	7	25
0.9	2	26	7	25
1.0	2	26	6	25

For neptunyl the thresholds of 0.7 and 0.8 Hartree require large number of iterations, 51, to reach convergence. This was already shown by the curve for 0.8 Hartree in Figure 11.4, featuring large fluctuations in a significantly large interval at the beginning. The even larger thresholds were requiring less iterations, 26, see Table 11.1. One reason for this might be that the thresholds starting with 0.7 are too large, so that DIIS-KS starts in an unfavorable region, far from the quadratic region. In this regard the smaller results for the even lager threshold might be an effect of chance. The best results for DIIS-KS obtained with initiation thresholds between 0.4 and 0.6 Hartree, are about half of the iteration count of the dynamic damping approach.

In case of the palladium cluster the number of iteration for the various initiation thresholds are much closer to each other. Only for very small values of the initiation threshold the number of iterations increases. The best value for the threshold to start DIIS-KS in this case is about 0.07 Hartree, which is much lower than in the case of the neptunyl. This also indicates that there is no universal optimal threshold. Still both systems performed well in the explored region. Turning of initiation thresholds provided an improvement for nearly all cases except for the thresholds 0.7 and 0.8 Hartree for neptunyl, where a larger number of iterations than with dynamic damping is obtained (Table 11.1).

Considering convergence criteria of 10^{-8} Hartree for the energy and 10^{-5} and 10^{-9} for density and charge fit coefficient criteria, respectively, the neptunyl test example required 69 iterations with the reference dynamic damping calculation. DIIS-cc and DIIS-KS (with the default threshold of 0.1 Hartree) required 66 and 33 iterations, respectively, thus 4% and 52% less iterations. The reduction was even better for Pd_{14} (with the same convergence criteria). The dynamic mixing calculation required with 132 iterations significantly more than DIIS-cc with 95 iterations or DIIS-KS with even 36 iterations. In this case DIIS-KS required 73% less iterations.

11.2 Pd_n Clusters, n= 6–116, with Perturbation Theory

To get an impression of how DIIS-KS behaves for larger systems than those discussed in the previous subsection a series of palladium cluster was examined, see Figure 11.1. The clusters were all of O_h symmetry, with 6, 14, 38, 44, 68, 92 and 116. For metal clusters featuring a small HOMO-LUMO gap, the convergence acceleration via perturbation theory is usually applied within ParaGauss. Thus perturbation theory is also applied for all of the following calculations, providing equal terms for comparisons. However, when DIIS-KS is started, perturbation theory is turned off. This is also in agreement with earlier considerations about this topic to facilitate convergence.[21] The occupation numbers of the orbitals around the HOMO-LUMO gap are smoothened by using a so-called broadening function of the orbital energies.[5] For the clusters investigated in this section the broadening was done with the Gaussian function with a half-width of 0.1 eV.

The convergence limit of the three criteria, for energy, density and fitted density, varies with the size of the clusters. While the energy convergence limit is below 10^{-10} Hartree for the smallest system, the average over the last 20 iterations for the largest system is only $7 \cdot 10^{-8}$ Hartree. The largest part of the change in the limit is most certainly related to the increase in the global energy. If only relative energy differences are considered the values are rather stable, about 10^{-15} to 10^{-14}. In a similar fashion, the convergence limit for the density on average is between $6 \cdot 10^{-6}$ and $3 \cdot 10^{-12}$ and the charge fit coefficients between $2 \cdot 10^{-4}$ and $2 \cdot 10^{-7}$. To a large extent, this decrease in accuracy may be related to the increase of values for the fitted approximate charge density and the usual density. The global norm of both increases with the number of atoms. While for the charge fit coefficients values considering these changes differ only between $3\cdot10^{-9}$ and $6\cdot10^{-10}$, the effect for the density measure is not that prominent, with values still differing between 10^{-14} and 10^{-9}. This might be partly related to the fact that the density measure is not normalized. The calculation using DIIS-KS with the standard DIIS initiation threshold of 0.1 Hartree on Pd_{116} did not converge (just as the calculations with dynamic damping), but showed large fluctuations from the beginning. Only after the initiation threshold for DIIS-KS was set to a smaller value of 0.05 Hartree the Pd_{116} calculation also converged. For all other systems the convergence limits for the calculation with DIIS-KS and dynamic damping are of the same order. Only for the charge fit coefficients the convergence limits for the calculations with DIIS-KS were always worse.

Table 11.2: Convergence limit of the electronic density and number of iterations to reach a convergence of twice the limit (iter.) for the palladium clusters with n atoms. Two different DIIS initiation thresholds (in Hartree) are compared.

	Without DIIS		DIIS, Thr. = 0.1		DIIS, Thr. = 0.05	
n	Conv.	Iter.	Conv.	Iter.	Conv.	Iter.
6	1E-11	63	4E-12	31	3E-12	31
14	2E-10	56	3E-11	40	3E-11	34
38	1E-07	36	2E-08	34	2E-08	33
44	2E-07	38	4E-08	44	3E-08	40
68	2E-06	39	3E-07	57	2E-07	49
92	3E-06	52	9E-07	50	7E-07	51
116	5E-06	57	–	–	6E-06	84

For the density convergence, the convergence limit and the number of iterations, to reach twice that value are given in Table 11.2 for all cluster sizes. The number of iterations was taken at this value to avoid the fluctuations around the limit while profiting from a faster convergence rate. The same two performance indicators for the charge fit coefficient and the energy convergence are summarized in Appendix E.

The number of iterations to reach the convergence limit was usually not significantly different for DIIS and dynamic damping. Especially for the larger cluster, Pd_{44} to Pd_{116}, the dynamic damping with perturbation theory required fewer iterations for the density and the energy convergence. For the smaller clusters, with 6 to 38 atoms, the strategy using DIIS outperformed dynamic damping with perturbation theory.

This shows that the DIIS method does not perform well together with perturbation theory and Fermi broadening. It seems to be of only marginal use for such calculations.

11.3 Actinide Complexes

Another large test example, the hydrated americium ion, $[Am(H_2O)_8]^{3+}$ in C_1 symmetry, see Figure 11.1, was used with level broadening by a Fermi function, with a width of 0.2 eV, but without perturbation theory convergence acceleration.

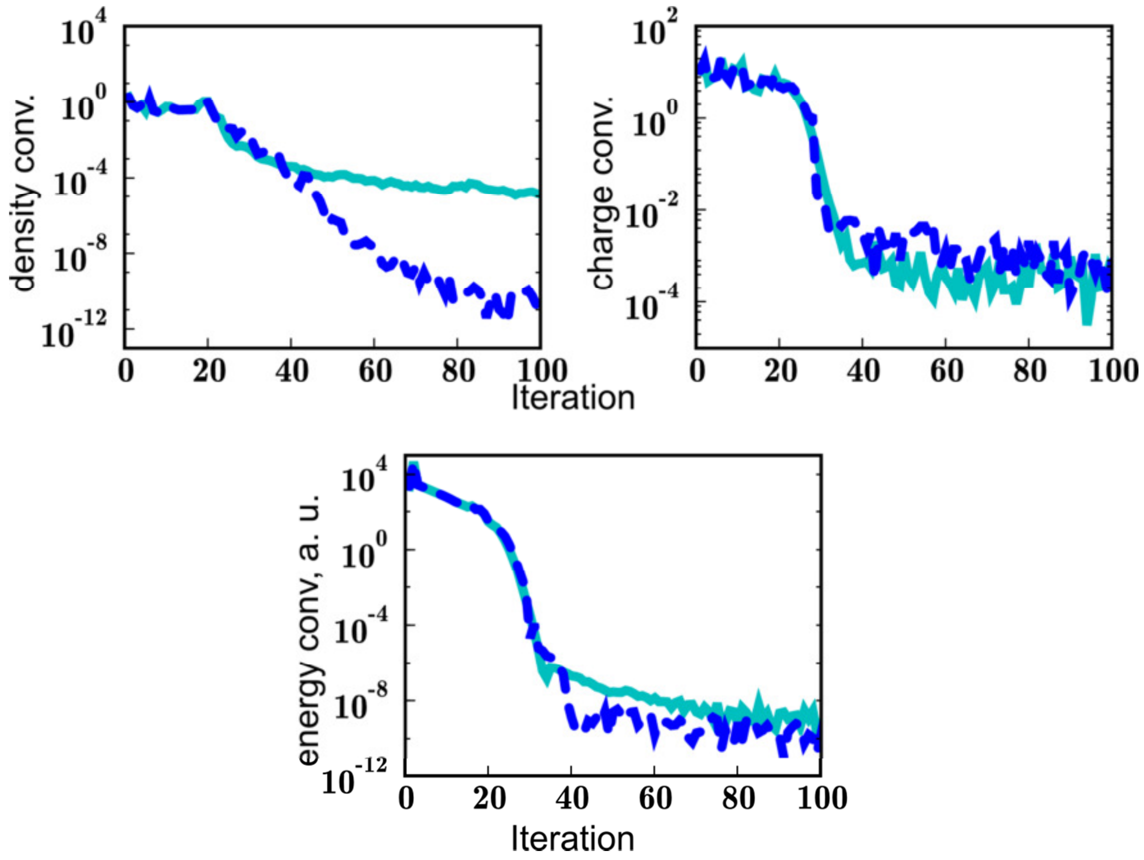

Figure 11.5: Evolution of convergence of energy, density and fitted density for hydrated Am^{3+}. Shown are the evolution of the convergence of the density (in relative units), the charge (in relative units) and the energy (in Hartree). Two calculations are compared: with dynamic damping only (solid) and with DIIS (dotted) after the DIIS initiation threshold of 0.05 Hartree was reached.

A reference calculation was carried out with a combination of static and dynamic damping: the first 15 iterations were done with a fixed mixing coefficient of 0.1 before changing to the dynamic damping. The DIIS-KS method was applied after the threshold of 0.05 Hartree in the convergence criteria was reached. With this initial threshold the calculations switched over to DIIS-KS after 27 iterations. The evolution of the convergence of energy, density and fitted density for the first 100 iterations is shown in Figure 11.5. There is hardly any difference in the behavior of the convergence of the charge fit coefficients between the calculation with and without DIIS-KS. In fact the calculation without DIIS-KS reaches the threshold of two times its convergence limit in iteration 37, while the calculation with DIIS-KS requires 6 iterations more for a slightly larger limit.

For the other two quantities, energy and density, this looks different. The energy convergence has a convergence limit of $8 \cdot 10^{-10}$ Hartree for the calculation without DIIS-KS and the value of $1.6 \cdot 10^{-9}$ Hartree is reached in iteration 74. With DIIS-KS the convergence limit is even $3 \cdot 10^{-10}$ Hartree and $6 \cdot 10^{-10}$ Hartree is reached in iteration 41 almost twice as fast. By design, DIIS-KS is expected to provide the best improvement for the density convergence: this is also true in the current case. While DIIS-KS reaches the accuracy of $5 \cdot 10^{-12}$ in iteration 87, the calculation without DIIS-KS is still far from this convergence region in iteration 199 with an accuracy of about $4 \cdot 10^{-7}$. Using convergence criteria of 10^{-8} Hartree, 10^{-9} and 10^{-3} for the energy, the density and the charge fit convergence, respectively, the calculation with DIIS-KS converges in iteration 68, while the reference calculation using only dynamic mixing did not reach convergence within 200 iterations at all. DIIS-KS is therefore well suited for usage in calculations without perturbation theory convergence acceleration.

12. Conclusion

Two variants of the direct inversion in the iterative subspace (DIIS) method have been implemented in the course of this work. They are not suited for the initial convergence. Thus one initially has to invoke a dynamic damping method and switch to the DIIS methods later on. One DIIS variant (DIIS-KS) works on the Kohn–Sham matrix while the other one (DIIS-cc) extrapolates the charge fit coefficients.

The DIIS methods can be applied efficiently for various systems. However, for systems with small HOMO-LUMO gap the usage of dynamic damping together with the perturbation theory is still recommended. For other systems DIIS, especially DIIS-KS, can be used to improve the convergence. Convergence of the density matrix, the charge fit coefficients and the energy are differently affected by DIIS. DIIS-KS shows usually for all variables the fastest convergence rate; its improvement of the speed of the convergence is the largest for the density. The convergence limits of the density matrix and the energy difference are of the same order for dynamic damping and both DIIS acceleration techniques. The convergence limit for the charge fit coefficients is often not as tight as for the other variables. However DIIS-cc allows in some circumstances to converge the charge fit coefficients to a tighter accuracy than the others quantities. This is the main advantage of this method.

For some small systems, the neptunyl anion and Pd_{14}, DIIS-KS reduced the number of required iterations of about 52% and 73% compared to the reference calculation with dynamic damping for the same choice of convergence criteria, relatively. For a larger test system, hydrated americium ion, the improvement especially for the density convergence measure was even larger.

Thus, in general DIIS method is a useful tool, which can increase the approach to convergence and the accuracy of final results for quite a lot of systems. However to get in all cases a fast and reliable convergence further investigations are required. For example, developing a method with improved convergence in cases with a small HOMO-LUMO gap would be a valuable choice for further efforts.

Part III: ParaTools

13. Methods for Transition State Search on Potential Energy Surfaces

13.1 Exploring Potential Energy Surfaces

A major task for computational chemistry is the search for specific geometries on the potential energy surface $E(\mathbf{r})$, a function of the geometry vector **r**, which comprises the positions r_A of all nuclei A in a three-dimensional space. Stationary points, geometries which are of special interest, have vanishing gradients of the energy with respect to the nuclear positions **r**, $\mathbf{g}(\mathbf{r}) = \nabla E(\mathbf{r}) = 0$. If the second-order energy derivatives, collected in the so-called Hessian matrix $\mathbf{B}$, give rise only to positive eigenvalues, the stationary point is a local minimum. Local minima on the potential energy surface correspond to the most important chemical structures — stable and meta-stable species, including also reaction intermediates. A transition state of a reaction is a stationary point with exactly one negative eigenvalue of the Hessian matrix. Reaction pathways are curves on the hypersurface interconnecting two of the minima typically via a transition state. The most important ones are those with minimal energy barrier, called minimum energy pathways. The path which follows the direction of least curvature of the potential energy surface, starting from a minimum, can end in a blind valley for complicated reactions. An alternative is the "steepest descent path", which always follows the direction of steepest descent, down from a transition state. The direction $\mathbf{t}(s)$ along the path (the tangent) at position **r**, with s being the coordinate which parameterizes the path, of such a steepest descent path is given by

$$\mathbf{t}(s) = -\mathbf{g}(\mathbf{r}(s)) \tag{13.1}$$

Following Fukui,[97] this equation defines the "intrinsic reaction path" which intersects each equipotential contour in orthogonal fashion.

Various methods exist for determining an approximation of the intrinsic reaction path. One example for these methods, which are widely used,[98-104] are chain-of-state methods, where the path approximation is represented by n images (sometimes also called path nodes), the chain of states. Interesting properties of a reaction, like the reaction rate, can usually be calculated without knowledge of the complete path but using only properties of the transition state. However, the transition state estimates from paths with common convergence criteria usually do not reach the desired accuracy.[101, 105] Methods for searching locally for a transition state can be used both independently[106-107] or as a refinement of a chain-of-state method.[108-113] The focus of the current thesis is on a two-step strategy, using a chain-of-state method to generate an approximate transition structure which then is refined by a local transition state method. Several choices for the different methods are considered.

There exist two important classes of chain-of-state methods: nudged elastic band (NEB)[114-115] and string[116-122] methods. Both classes require one to define tangents of the path at every image position. The component of the energy gradients at the images parallel to the path does not need to vanish. Both classes of methods also need to keep the images evenly or otherwise regularly distributed along the path. The main difference between these two classes of methods is that the NEB method modifies the energy gradients to care about all these requirements at once. This is done by adding so-called string forces to mimic an elastic band between the images. The string methods decouple the two requirements, optimization of the path and distribution of the images along the path. Thus, only the gradient component perpendicular to the path is optimized; in a separate step, called “respacing”, one takes care of the distribution of the images. In the respacing step, which may be applied only occasionally, the images are redistributed on the path to fulfill the conditions for spacing.

Besides the standard implementation of such methods it is worth mentioning that there exist some more specialized methods, modifications of the standard methods, which are supposed to provide advanced transition state estimates. For example, the searching string method is a version of the string method which intends to space the images more tightly around the transition state.[118] Another modification is the climbing image (CI) approach where one image is supposed to reach the transition state. This approach was first suggested by Henkelman et al.[123] as a modification of the nudged elastic band method. It was also adapted for usage in combination with string methods.[124]

When searching locally for a transition state, one has to consider that the search is similar to the search for a local minimum as both geometries are stationary points. Nevertheless, the search for a transition state is much more complicated than the search for a local minimum because for a transition state the Hessian matrix needs to have exactly one negative eigenvalue. Getting all eigenvalues positive (as for the minimum) is simpler and can, for example, be done by the quasi-Newton methods, which use first- and approximate second-order derivatives of the energy with respect to the nuclear positions. The positive definiteness of the approximate second-order derivatives of the energy, the Hessian matrix, can be enforced for the minimum search; thus the stationary point to which the method converges has with high probability the right signature. The signature of a transition state cannot be enforced by common update strategies. Thus a direct approach to a transition state is not feasible with a quasi-Newton approach. It is only possible to use a quasi-Newton method which searches for the next stationary point, or to turn to a more complex strategy. When using the simple quasi-Newton method, there is a high probability, especially for starting geometries far from the structure of the transition state, that the search fails to locate the intended transition state and ends instead in any other stationary point. This method is therefore only useful if the geometry is already close to the transition state. For example, the program package Gaussian implements synchronous transit methods which switch automatically to quasi-Newton when getting close enough to a stationary point.[125]

Solutions to make the quasi-Newton method work also for transition state search when starting further off the convergence region usually involve a calculation of all eigenvalues of the Hessian. Poppinger[126] proposed a method where all eigenvalues are calculated and in case of a wrong number of negative ones a step along the mode direction of the unwanted curvature (minimization for the second negative one, maximization if all curvatures are positive) is done in order to reach a region with the desired signature of eigenvalues. When only one negative eigenvalue is obtained, a Newton–Raphson step is carried out. The possibility of using also the other eigenmodes for the determination of the correction step has also been discussed.[127-128] Baker prefers to shift all eigenvalues and then uses an rational function optimization (RFO) approach.[127] In this way the search direction is chosen such that the energy along one mode of the approximate Hessian should be maximized, the energy along all other eigenvectors should be minimized. Baker mentioned that the mode along which the energy should be maximized does not need to be the mode that corresponds to the lowest eigenvalue of the approximate Hessian but could be arbitrarily chosen from the eigenmodes of the approximate Hessian. In this case the mode with the largest overlap to the mode of the last iteration is used further during the progressing optimization.[127]

A different class of local transition state methods is often addressed under the name “mode following”. These procedures define a special mode direction which should represent the reaction coordinate. This direction is treated differently from the others when the geometry is updated. One way of carrying this out is to separate steps along and orthogonal to the mode direction.[129] However it seems to be a superior strategy to adapt also the mode direction itself.[130] A method, which combines the two types of steps by using a modified gradient, does alternating update steps for the geometry and the mode direction. This is the so-called Dimer method, for which several modifications are available.[131-133]

13.2 Routines of the Program Package ParaTools

ParaTools is a collection of routines, written in the programming language Python, which work on a potential energy surface. For these routines the potential energy surfaces can come from various sources, from a simple analytical potential like the Lennard-Jones potential up to a potential energy surface based on a complicated density functional code. ParaTools treats the source as a “black box” algorithm that provides energies and gradients of the energy for selected coordinates. Many of these “black box” routines are python interfaces for quantum chemistry programs, obtained from an open source program package, the atomic simulation environment (ASE).[134]

Most of the ParaTools routines concentrate on the tasks of finding transition states or approximations to a minimum energy path. For these tasks a selection of the routines mentioned in Section 13.1 were implemented. This selection includes both chain-of-state methods and local transition state search methods to allow performing a two-step strategy of transition state search as mentioned in Section 13.1.

The chain-of-state methods implemented in ParaTools are a version of the NEB method, which follows the descriptions of Henkelman et al.,[115] and three string methods, which share the general interface but differ in the distribution of the images. The three string methods are: first, a standard string method, with images equally spaced on the path; second, a growing string variant[122, 135] and at last, a searching string variant.[118] The growing string and searching string variants start with a reduced string and add further images when some specific requirements are met. The differences between these variants are the positions at which the new images are inserted and whether some of the images of the remaining path are frozen. All string methods make use of cubic splines to interpolate the path between the images, allowing one to approximate the path tangents at the images as tangents of the cubic splines. All string methods implement a search for new positions on the path for respacing. All variants of the string methods implemented differ from the ones described by Chaffey-Millar et al.[118] in the algorithm to construct a spline parameterization, see Chapter 17, and the possibility of choosing a metric for the coordinate space, see Chapter 18. Additional standard optimizers were implemented, see Chapter 15. The novel enhancements include also climbing image variants for all path searching methods, see Chapter 16.

Additionally ParaTools was enhanced by some local transition state search methods, see Chapter 14. A simple quasi-Newton method and a standard dimer method[131] were implemented as references. The main enhancement for the local transition state search methods was a new interpretation of the dimer method motive, which was developed and implemented in the context of this thesis.

14. Local Transition State Search Routines

14.1 Quasi-Newton Methods

Newton methods are widely used to locate the geometries that represent minima on potential energy surfaces, using first- and second-order derivatives of the potential energy. The methods propose a step, which on a quadratic potential energy surface leads directly to the minimum. Quasi-Newton methods approximate the matrix of the second-order derivatives of the energy, the Hessian matrix $\boldsymbol{B}$ (or its inverse $\mathbf{H} = \mathbf{B}^{-1}$), using the changes of the gradients of the energy with respect to the nuclear positions during the preceding steps, by one of several Hessian update methods.

There are many adaptions of the Hessian update schemes for transition state search dealing with the fact that the matrix of second-order derivatives of the energy should have exactly one negative eigenvalue. This requirement rules out the BFGS (Broyden–Fletcher–Goldfarb–Shanno) update method,[136] which enforces positive definiteness of the approximate Hessian. On the other hand, the SR1 method (symmetric rank 1) will maintain only the symmetry, allowing for negative eigenvalues. The SR1 update formula for the inverse of the Hessian matrix $\mathbf{H} = \mathbf{B}^{-1}$ is:

$$\mathbf{H}^{k+1} = \mathbf{H}^{k} + \frac{\mathbf{u}\cdot\mathbf{u}^{T}}{\mathbf{u}^{T}\mathbf{y}^{k}}, \quad \mathbf{u} = \Delta\mathbf{x}^{k} - \mathbf{H}^{k}\mathbf{y}^{k} \tag{14.1}$$

with $\mathbf{y}^{k} = \mathbf{g}^{k+1} - \mathbf{g}^{k}$ and $\Delta\mathbf{x}^{k} = \mathbf{x}^{k+1} - \mathbf{x}^{k}$ being the gradient difference and the corresponding step from $\mathbf{x}^{k}$ to the new geometry $\mathbf{x}^{k+1}$, respectively. A quasi-Newton approach with the SR1 update may or may not converge to a transition state. If one of the first geometries, where the Hessian matrix is not yet very accurate, would lead into a region with several or no negative eigenvalues, the method is likely to converge to the stationary point belonging to this region. It simply is not advisable to start a transition state search from far off the solution. The method is properly used only for geometries which are already nearly converged to the intended transition state where the maximum geometry update step size is small. Then quasi-Newton methods are supposed to converge fast and reliably.

14.2 The Standard Dimer Method

The Dimer method was invented by Henkelman and Jonsson[131] to find saddle points on high dimensional potential surfaces. Their method is useful if the second-order derivatives of the energy are not accessible or much too costly to compute. Henkelman and Jonsson[131] used two images at $\mathbf{x}_{1+}$ and $\mathbf{x}_{1-}$ of the system, connected by a vector of fixed length in the direction of the so-called mode vector ***m*** with $\mathbf{x}_{1+} - \mathbf{x}_{1-} = 2l \cdot \mathbf{m}$, $|\mathbf{m}| = 1$. The images should be arranged such that the mode vector is along the direction of lowest curvature *C* of the potential energy surface at the location of the dimer midpoint $x_0 = (\mathbf{x}_{1+} + \mathbf{x}_{1-}) / 2$. To achieve this, the images can be rotated around their midpoint (rotation step). In a translation step the images are moved such that the energy in the direction of the approximated lowest curvature mode is maximized and the energy in the complementary directions is minimized.

These ideas were used and modified by other authors.[132-133, 137] who found it convenient to use the midpoint at $\mathbf{x}_0$ and one of the images (at $\mathbf{x}_1$) for defining the dimer. These authors only approximate the gradients at the other image $\mathbf{x}_2$. With this approach one takes into account that one usually does not have an analytical potential surface and the cost of each gradient evaluation is substantial. Therefore, it is rather helpful to calculate explicitly the point, which is supposed to reach the saddle point. Furthermore the results for the midpoint can be reused in the dimer rotations; in case of maintaining two images one has to calculate the gradients for both images anew in each rotation step. Also, there is a tendency to do several rotation steps till convergence is reached before doing a translation step, instead of doing alternating rotation and translation steps. The implementation in ParaTools follows the method described by Heyden et al.[132]

14.2.1 The Translation Step

The gradients at the dimer midpoint $\mathbf{x}_0$ should vanish at a transition state. Different to, for example, a minimum, this should not be reached by minimizing the potential energy in all directions (for a minimum all eigenvalues of the matrix of second-order derivatives of the energy are positive), but by maximizing the potential energy in the direction of the mode vector (corresponding to a negative eigenvalue). This can be rephrased as following a modified force, where the component of the true force along the dimer direction is inverted. A modified force does no longer correspond to the true potential energy surface, thus the minimization algorithm has to work only on the forces. Henkelman et al.[131] suggested to use the conjugate gradient method for the minimization, which was seconded by Heyden et al.[132] The current implementation uses the Polak–Ribiere[138] approach for the conjugate gradient method with Powell's restart suggestion.[139] The only change to the implementation for minimizing is that the modified forces had to be used instead of the usual ones.

14.2.2 The Rotation Step

Every rotation is carried out in a plane, which is defined by the mode vector and a second direction, $\mathbf{k} \perp \mathbf{m}$, derived from a so-called "rotation" force. This rotation force is defined as the difference between the gradient projections perpendicular to the mode vector at the two dimer endpoints, $\mathbf{F}^{\mathrm{rot}} = -\mathbf{g}_{1+} + \mathbf{g}_{1-} + (\langle \mathbf{g}_{1+} | \mathbf{m} \rangle - \langle \mathbf{g}_{i-} | \mathbf{m} \rangle)\mathbf{m}$, for the gradients $\mathbf{g}_{1+}$ and $\mathbf{g}_{1-}$ at $\mathbf{x}_{1+}$ and $\mathbf{x}_{1-}$, respectively. When using the midpoint, the expression is similar:[132]

$$\mathbf{F}^{\mathrm{rot}} = -2(\mathbf{g}_1 - \mathbf{g}_0) + 2\langle \mathbf{g}_1 - \mathbf{g}_0 | \mathbf{m} \rangle \mathbf{m} \tag{14.2}$$

For the ideal case of a quadratic surface the two definitions are equivalent. It is possible to take the force vector $\mathbf{F}^{\mathrm{rot}}$ as the second direction required to specify the plane although often one applies a conjugate gradient approach to the rotation force to improve direction $\mathbf{k}$ for the plane. This requires changing the usual conjugate gradient approach:[138]

$$\mathbf{k}_i = \mathbf{F}^{\mathrm{rot}}(i) + \gamma_i \mathbf{k}'_{i-1} \tag{14.3}$$

Here the weighting factor γ_i and the modified old direction $\mathbf{k}'_{i-1}$ are used; the latter is obtained from the direction $\mathbf{k}_{i-1}$ of the preceding step as follows. The old direction $\mathbf{k}_{i-1}$ is not orthogonal to the current mode vector **m**. To define the plane by a second direction $\mathbf{k}_i$, the modified old direction $\mathbf{k}'_{i-1}$ should be orthogonal to the current mode vector but should still lie in the old rotation plane:[131]

$$\begin{aligned} \mathbf{k}^{\mathrm{h}} &= \left(\mathbf{k}_{i-1} - \langle \mathbf{k}_{i-1} | \mathbf{m} \rangle \mathbf{m}\right) \\ \mathbf{k}'_{i-1} &= |\mathbf{k}_{i-1}| \cdot \mathbf{k}^{\mathrm{h}} / |\mathbf{k}^{\mathrm{h}}| \end{aligned} \tag{14.4}$$

Using these formulae the vector $\mathbf{k}'_{i-1}$ which replaces the old direction $\mathbf{k}_{i-1}$ of the usual conjugate gradient approach fulfills these requirements. The length of this vector $\mathbf{k}'_{i-1}$ is the same as the length of the old vector $\mathbf{k}_{i-1}$. The weighting factor γ_i is calculated following Polak–Ribiere,[138] only using the rotation forces:

$$\gamma_i = \frac{\langle \mathbf{F}^{\mathrm{rot}}(i) - \mathbf{F}^{\mathrm{rot}}(i) | \mathbf{F}^{\mathrm{rot}}(i) \rangle}{\langle \mathbf{F}^{\mathrm{rot}}(i) | \mathbf{F}^{\mathrm{rot}}(i) \rangle} \tag{14.5}$$

If the weighting factor is negative, $\gamma_i < 0$, the procedure is restarted as suggested by Powell.[139]

Having defined the plane $(\mathbf{m}, \mathbf{k})$ by two orthogonal vectors and fixed the midpoint as well as the dimer length $2l$, there is only one degree of freedom left, which could be used to find the best mode vector in the plane. This degree of freedom is expressed as a rotation angle

ϕ. The geometries of the dimer point can thus be expressed as a function of the rotation angle, $\mathbf{x}(\phi)$, with $\mathbf{x}(0)=\mathbf{x}_1$. The curvature of the potential energy surface at the midpoint in the direction along the rotated mode vector can be approximated as difference of the gradient projections:

$$C(\phi)=\langle \mathbf{g}(\mathbf{x}(\phi))-\mathbf{g}_0 \,|\, \mathbf{x}(\phi)-\mathbf{x}_0 \rangle / l^2 \tag{14.6}$$

One assumes the curvature to be a quadratic function around the midpoint:

$$C(\phi)\simeq a_0/2+a_1\cos 2\phi+b_1\sin 2\phi \tag{14.7}$$

The parameters of this equation can be calculated with Eq. (14.6), using two geometries, $\phi=0$ and $\phi=\phi_1$:

$$\begin{aligned} b_1 &= \frac{1}{2}\frac{\partial C}{\partial \phi}\bigg|_{\phi=0} \\ a_1 &= (C(0)-C(\phi_1)+b_1\sin 2\phi_1)/(1-\cos 2\phi_1) \\ a_0 &= 2(C(\phi_1)-a_1) \end{aligned}$$

[The first-order derivative of the curvature can also be approximated with the help of Eq. (14.6).] In another, simpler, approximation, the rotation angle ϕ^* that corresponds to the minimum curvature of the potential energy surface as found in the plane, can be determined from the gradients of one geometry $\mathbf{g}_1$, allowing to calculate $C(0)$ and $\partial C/\partial\phi$:

$$\phi^* = -\frac{1}{2}\arctan\frac{\partial C/\partial \phi}{2|C(0)|} \tag{14.8}$$

This can be used as the second geometry at ϕ_1, as required for the quadratic approximation. Heyden et al.[132] suggested doing another rotational step ϕ_1 for the interpolation instead. In this way, a more stable numerical procedure is expected when ϕ^* becomes small near convergence. The standard dimer method of ParaTools uses $\phi_1=\pm\pi/4$ with the same sign as the approximate angle ϕ^*. The quadratic approximation, Eq. (14.7), being a periodic function, has two minima that correspond to the two equivalent choices of the mode vector. Thus, any of the two directions can be used as a new mode vector **m**; e.g., see Eq. (14.2) .

To avoid wasting computer time, the number of rotation steps is limited and two convergence criteria are applied. The calculation is stopped if the rotation angle $\phi_{\min}$ to the minimum of Eq. (14.7) is below a threshold. The rotation is also stopped if the approximate rotation angle ϕ^* is smaller than the same threshold. A small angle ϕ^* indicates that the mode vector **m** is an eigenvector of the Hessian matrix of the system, as in this case the gradient difference $\mathbf{g}–\mathbf{g}_0$ is parallel to the mode vector; the two contributions $\partial C / \partial \phi$ and $2|C(0)|$ of the angle can be identified as the parts of the gradient difference perpendicular and parallel to the mode vector. This is one of the suggestions of Heyden et al.[132] for reducing the number of required gradient evaluations. For every completed dimer rotation step two gradient calculations are necessary.

14.3 The Modified Dimer-Lanczos Method

The newly implemented modified Dimer-Lanczos (mDL) method differs from the standard dimer method, Section 14.2, in both update steps, the mode update and the midpoint update. The principle procedure of the dimer method with alternating update procedures is kept. While for the mode vector update step a simplified version of the Lanczos method is used to determine the eigenvector of the smallest eigenvalue,[137] the method for the translation step is a novel approach to combine the strength of the quasi-Newton approach with the strength of the dimer method.

14.3.1 Mode Vector Update by the Lanczos Method

The Lanczos method is one of the methods that allow one to determine a specific eigenvalue and the corresponding eigenvector of a matrix **A**.[140] These methods were originally intended for solving large systems of linear equations $\mathbf{Av} = \mathbf{c}$, making use of the Krylov subspace. For a starting vector **c** the Krylov subspace of order $k \le n$ of an $n \times n$ matrix **A** is given as $\mathcal{K}^{(k)}(\mathbf{c}, \mathbf{A}) = \mathrm{span}\{\mathbf{c}, \mathbf{Ac}, \ldots, \mathbf{A}^{k-1}\mathbf{c}\}$. For the maximum size $k = n$ it represents the complete space. The Arnoldi method, or for symmetric matrices like in our case the Lanczos method, searches for the solution of the linear equation in the subspace $\mathcal{K}^{(k)}(\mathbf{c}, \mathbf{A})$ expressed in an orthogonal set of basis vectors. One can also use the basis vectors for getting approximate eigenvalues of the matrix **A**, which corresponds to the Hessian, in $\mathcal{K}^{(k)}(\mathbf{c}, \mathbf{A})$ and take the eigenmode with the smallest eigenvalue as the dimer mode vector. First a Lanczos approach was used in the so-called “ART nouveau” method[141] to replace the eigenvector decomposition of the Hessian for large systems described by analytical potentials. Then the usability of the Lanczos method was tested by Olsen et al.[137] who found it to be superior to the dimer rotation for tight convergence criteria. For less tight convergence criteria both methods for determining the mode vector were comparable.[137]

The mode vector **m** of the mode update step is used as the starting vector **c**. The matrix **A**, which corresponds here to the Hessian, is not explicitly accessible, instead the vector product $\mathbf{Av}$ for any vector $\mathbf{v}$ in the Krylov space, which represents a step from the midpoint $\mathbf{x}_0$ with unit length, is approximated by

$$\mathbf{Av} = \tilde{\mathbf{g}}(\mathbf{v}) = \frac{\partial E}{\partial \mathbf{x}}(\mathbf{x}_0 + l\mathbf{v}) - \frac{\partial E}{\partial \mathbf{x}}(\mathbf{x}_0) \tag{14.9}$$

for a fixed length l, which corresponds to half the distance between the two images of the dimer; cf. the dimer length in Subsection 14.2.2. The Lanczos basis vectors

$$\mathbf{q}^{(i)} \;=\; n_i \left(\tilde{\mathbf{g}}(\mathbf{q}^{(i-1)}) - \sum_{k<i} b_{ki} \mathbf{q}^{(k)} \right) \tag{14.10}$$

with the normalization constant n_i and the matrix elements

$$b_{ij} \;=\; \left(\langle \mathbf{q}^{(i)} \tilde{\mathbf{g}}(\mathbf{q}^{(j)}) \rangle + \langle \mathbf{q}^{(j)} \tilde{\mathbf{g}}(\mathbf{q}^{(i)}) \rangle \right) / (2l) \tag{14.11}$$

build the approximate subspace Hessian $\mathbf{B}^{(k)}$. This Hessian is a tridiagonal matrix. It is possible to calculate the elements of matrix $\mathbf{B}^{(k)}$ in step k without having to store all vectors $\mathbf{q}^{(i)}$ and $\tilde{\mathbf{g}}(\mathbf{q}^{(i)})$ of the previous iterations, as done both in the original method[140] and in a quantum chemical application.[137] However, as the maximal dimension $k_{\max}$ is expected to stay rather small, this kind of optimization is not useful in the current case. Additionally the vectors $\mathbf{q}^{(i)}$ and $\tilde{\mathbf{g}}(\mathbf{q}^{(i)})$ as determined by the Lanczos algorithm will be also useful for updating the approximate Hessian, see Subsection 14.3.2. Therefore the current implementation calculates the basis vectors $\mathbf{q}^{(i)}$ and the approximate subspace Hessian $\mathbf{B}^{(k)}$ as described in Eqs. (14.10) and (14.11) using only the Lanczos basis.

A few simplifications were applied which might prove to be useful for the current case and which can indirectly be found in the simplifications of the original method:[137, 140]

1. The Hessian matrix should be symmetric. Therefore Eq. (14.11) was taken instead of the standard expression of the matrix elements $b_{ij} \;=\; \langle \mathbf{q}^{(i)} \tilde{\mathbf{g}}(\mathbf{q}^{(j)}) \rangle / l$, which need not to result in a symmetric matrix.
2. Only the elements which are non-zero by the theory of the tridiagonal matrix $\mathbf{B}^{(k)}$ are calculated.

Each iteration of the Lanczos algorithm provides a smallest eigenvalue $e_{\min}^{(k)}$ and the corresponding eigenvector $\mathbf{v}_{\min}^{(k)}$ as an approximation for the mode vector $\mathbf{m}$. The algorithm does not need to do $k_{\max}$ steps every time it is applied but stopping criteria are applied. If the angle between the eigenvectors $\mathbf{v}_{\min}^{(k)}$ of two consecutive iterations is too small (or too close to π), there is hardly any progress and the update procedure is stopped. The second angle to observe is the angle between the current eigenvector $\mathbf{v}_{\min}^{(k)}$ and the expected gradient $\tilde{\mathbf{g}}(\mathbf{v}_{\min}^{(k)}) \approx \sum \alpha_i \tilde{\mathbf{g}}(\mathbf{q}^{(i)})$ with α_i being the components of the eigenvector $\mathbf{v}_{\min}^{(k)} = \sum \alpha_i \mathbf{q}^{(i)}$ in the Lanczos basis. If this angle is small, then the eigenvector $\mathbf{v}_{\min}^{(k)}$ corresponds to a good approximation to an eigenvalue $e_{\min}$ of the complete Hessian matrix $\mathbf{B}^{(n)}$. The criteria were chosen as angles so that they are more easily comparable to the criteria for the dimer rotation.

Neither method for mode vector updating, the standard dimer rotation or the Lanczos algorithm, guarantees to find at convergence the smallest eigenvalue, even of an approximate full Hessian. The only exception is when the Lanczos algorithm stops after exactly *n* iterations. The dimer rotation cannot even guarantee that it finds the smallest eigenvalue from the space in which it searched. Therefore it is always required to have a good starting vector. For moderately large steps the change in the eigenvectors will not be very large, providing the algorithm a fair chance to find the correct mode vector.

The Lanczos method covers the complete space after *n* iterations. Then the algorithm should in principle be converged, fulfilling any convergence criteria. In reality approximations and numerical errors might cause that an error remains. In this case it is not possible to do a next step as there is no direction left to explore. Thus, if the search for the mode vector should be continued after *n* iterations, it is required to do a restart which removes the already tested trial vectors and starts with a Lanczos subspace of dimension 1 in the direction of the last mode vector. The standard dimer method does not suffer from this restriction. However, even after *n* iterations it might still be leaving parts of the space unexamined.

14.3.2 The Step Update Algorithm

The novel step update algorithm for the modified Dimer-Lanczos method is a quasi-Newton approach with a specially adapted inverse Hessian **H**, which was created using information of the mode update step, and the gradient **g** resulting in a Newton step $\delta\mathbf{x} = \mathbf{Hg}$. One method widely used to minimize the energy is the quasi-Newton method using an update for the approximate Hessian like the BFGS update method;[136] see Section 14.1. However with the approximate Hessian, or even more comfortable for the requirement of the algorithm the approximate inverse Hessian $\mathbf{H}_{\text{BFGS}}$, it is not possible to search for a transition state, as the approximate inverse Hessian is usually positive definite. However as the energy should be minimized in all but one directions, the Hessian can still be used efficiently for a part of the optimization. The direction, in which the energy should be maximized instead of minimized, is approximated by the mode vector **m**. This direction is supposed to correspond to an eigenvector of the ideal inverse Hessian **H**, and has been calculated by the Lanczos method. The Lanczos method provides also the approximate eigenvalue to this vector, which corresponds to a desired negative eigenvalue 1/*C* of the inverse Hessian. To get an approximate Hessian, which includes the eigenvector and eigenvalue as defined by the Lanczos method, an SR1 update step $\Delta\mathbf{H}_{\mathrm{SR1}}$ is used. This update step is carried out with the approximate inverse Hessian $\mathbf{H}_{\mathrm{BFGS}}$ as initial approximation. Thus the general inverse Hessian

$$\mathbf{H} = \mathbf{H}_{\mathrm{BFGS}} + \Delta\mathbf{H}_{\mathrm{SR1}}, \tag{14.12}$$

which should be the best possible approximation with the available information, is used for the quasi-Newton step.

The BFGS part, $\mathbf{H}_{\mathrm{BFGS}}$, is updated in the usual way[136] using steps and gradient differences of the middle point of the dimer. Additionally it is updated with the gradient differences for the various directions of the Lanczos basis from the rotation steps. The BFGS update can use the geometries $\mathbf{q}^{(j)}$ and gradients $\tilde{\mathbf{g}}^{(j)} = \tilde{\mathbf{g}}(\mathbf{q}^{(j)})$ from the Lanczos iterations to get an improved Hessian approximation:

$$\mathbf{H}_{\mathrm{BFGS}}^{(k+1)} = \mathbf{H}_{\mathrm{BFGS}}^{(k)} + \frac{1+\tilde{\mathbf{g}}^{(j)T}\mathbf{z}}{\tilde{\mathbf{g}}^{(j)T}\mathbf{q}^{(j)}}\mathbf{q}^{(j)}\mathbf{q}^{(j)T} - \mathbf{q}^{(j)}\mathbf{z}^{T} - \mathbf{z}\mathbf{q}^{(j)T} \tag{14.13}$$

with $\mathbf{z} = \mathbf{H}_{\mathrm{BFGS}}^{(k)}\tilde{\mathbf{g}}^{(j)} / \left(\tilde{\mathbf{g}}^{(j)T}\mathbf{q}^{(j)}\right)$. The additional update with the Lanczos basis is done before the original update of the algorithm using the translation steps of the geometry center. The BFGS update was originally developed to use the translation steps, which should converge to a stationary point. Thus these steps are more important than those of the Lanczos basis for the algorithm.

The second part of the inverse Hessian, $\Delta\mathbf{H}_{\mathrm{SR1}}$, Eq. (14.14), is an SR1 update employing the mode vector **m** and its eigenvalue *C* of the Lanczos Hessian. Using $\mathbf{y}^{k} = C\mathbf{m}$ and $\mathbf{u}^{k} = \mathbf{m} - \mathbf{H}_{\mathrm{BFGS}}^{k}C\mathbf{m}$, it can be expressed as

$$\Delta\mathbf{H}_{\mathrm{SR1}} = \frac{\mathbf{u}^{k}\mathbf{u}^{kT}}{\mathbf{u}^{kT}\mathbf{y}^{k}} \tag{14.14}$$

Here one takes into account that the Lanczos step length *l* cancels out (the steps are $l\mathbf{q}^{(j)}$). The complete approximate inverse Hessian **H** has at most one negative eigenvalue in the direction of the mode vector **m**; see Appendix F for a proof.

Therefore, the inverse Hessian **H** has the correct signature when the eigenvalue 1/*C* of the eigenvector, which is the mode vector provided by the Lanczos algorithm, (see Subsection 14.3.1) is negative. In the case that $C > 0$, like in the standard dimer method, a special step is done. This special step corresponds to a full relaxation along the gradients perpendicular to the mode vector to reach the reaction path if possible. This special step is always chosen to have the maximal step length; if the step length of the relaxation step is smaller, a step along the mode vector is added in direction opposite to the gradients.

15. Optimizers for the Path Searcher Routines

15.1 General Considerations and Reference Optimizers

The task of an optimizer in the framework of a chain-of-state method is to propose a step for each image of the discrete path representation in every iteration until a fixpoint has been reached. String methods might modify these steps further by respacing the images along the path. All the optimizers applied are based on optimizers used for minimum search and were adapted to deal with a path rather than only a single image. Among the optimizers considered there is none which is aware of the way the path or the tangents are expressed, nor of the constraints which are imposed on the distances between images. There is only one optimizer, called Multiopt, which takes into account that the gradients it works with, are not the gradients of the energy with respect to the nuclear positions but modifications, like only the part perpendicular to the path for the string methods or that they contain the string forces for the NEB method.

There are two conceptual approaches for chain-of-states optimizers. One approach is to treat the n images, corresponding to systems with N atoms each, as a larger system with $n{\cdot}N$ atoms and gradients $\mathbf{G}$, where the component $\mathbf{g}_k$ refers to the atom k, and to modify atomic forces either by eliminating the component collinear with the path tangent for the string methods or by adding spring forces in case of the NEB approach. Then it is possible to use for both methods exactly the same optimizer as for a minimum search on a potential energy surface. However, it is required to use only such methods, which do not use the potential energy itself, but work only with gradients and possibly a Hessian matrix. This is due to the fact that there is no potential energy surface that corresponds to the modified gradients of the chain of states. The second concept for path optimizers is to invoke for every image an instance of a single-image optimizer which independently proposes a step for this image.

Several optimizers of both kinds, operating on a whole chain or on each chain image separately, were compared to each other for CI-NEB and CI-string methods by Sheppard et al.[142] Several optimizers are also included in the Transition State Tools for VASP (Vasp TST Tools)[143] for the NEB method. These optimizers provide also an algorithm following the eigenmode of the lowest eigenvalue of the Hessian, like the dimer method.[131] Thus, no adaption for the chain-of-state methods was done by the developers of Vasp TST Tools.[143] The optimization methods of the Vasp TST Tools are divided into two classes in the Vasp TST Tools, depending on the usability for small accurate gradients or for gradients which are inaccurate or high.

For the current thesis two standard optimizers have been added to ParaTools, one for small and accurate gradients and one for less favorable cases. ParaTools contained already the Multiopt optimizer. The Multiopt optimizer will serve as reference.

The Multiopt optimizer is a special purpose optimizer which has been designed exclusively for the string methods. It follows the scheme of having one optimizer instance for each image, extended by a collective scaling of the step length, and has already been used in an systematic study with the toolbox.[118] It uses the steepest decent direction for every image as a search direction. The step length in the search direction is obtained with an approximate Hessian $\mathbf{B}_j$, which is modified by an SR1 update. A quadratic surface approximation, based on this Hessian, and the gradients and energy of the image at the current position, are used to estimate the minimum in the steepest decent direction. The step length is further restricted by a trust radius.

15.2 The Conjugate Gradient Optimizer

The conjugate gradient optimizer is supposed to be a good solution when accurate gradients of the energy are available. The conjugate gradient optimizer of ParaTools follows in most parts the variant of Sheppard et al.[142] used for NEB and the string methods and can be used for both of them, too. It works on converted vectors of the N_{beads} geometries and gradients. Similar to the description of Sheppard et al.[142] it uses the Polak–Ribiere formula[138] for generating the coefficient $\gamma^{(m)}$ needed for the search direction $\mathbf{Q}_m = -\mathbf{G}_m + \gamma^{(m)}\mathbf{Q}_{m-1}$ in iteration m. The vector of gradients $\mathbf{G}_m$ consists of the components $\mathbf{g}_k^{(m)}$ which were properly modified for string or NEB approaches. To build the new search direction this vector was used to modify the search direction from the previous iteration $\mathbf{Q}_{m-1}$. In ParaTools the optimizer uses this method with the modification of resetting the conjugate gradient approach when $\gamma^{(m)}$ turns negative by using the steepest decent direction instead.[138-139] The coefficient $\gamma^{(m)}$ is calculated as

$$\gamma^{(m)} = \max\left(\frac{(\mathbf{G}_m^T - \mathbf{G}_{m-1}^T)\mathbf{G}_m}{\mathbf{G}_{m-1}^T\mathbf{G}_{m-1}}, 0 \right) \tag{15.1}$$

Normally the conjugate gradient method uses a line search algorithm to define the length of the update step, but the current implementation, as well as the one from Sheppard et al.,[142] does a single Newton step instead. The Newton step has the advantage over a line search that it usually needs much fewer gradient calculations. The required second-order energy derivatives in the conjugate gradient direction are approximated as gradient differences based on a trial step in the search direction. The trial step has a finite length l_t. The step length for the Newton step is further limited to a finite size, in our case to $s_{\max} \cdot N_{\text{beads}}$ with a parameter $s_{\max}$. The step length includes a factor N_{beads} to allow progress also in the case of a large number of images without having to change the step length. The sign of the step vector is not fixed. Thus in special cases it is possible that the step goes into the direction opposite to the gradients, $-\mathbf{Q}_m$. Yet, it is not critical if the first trial step goes into a direction with positive gradient projection. If this Newton step is accepted unconditionally, the gradient projection on the search direction might increase by absolute value, if the quadratic model of the potential energy surface is not accurate enough. Every modified gradient is influenced by the positions of all of the images. The direction perpendicular to the path changes while the image positions change. For the case of string method calculations the positions can even be changed after a Newton step by a respacing procedure. It does not make sense to use much effort for making the gradient projections onto the current step direction vanish because the Newton step procedure cannot ensure that the weakest convergence conditions are fulfilled, hence that the step improves the convergence of the path. However, without having a potential energy surface to judge the process, this would be anyhow complicated. On the other hand, accepting the step in any case can lead the system to a completely wrong structure. There is especially one case, where it does not make sense to accept the step: if one has overstepped. In other words, the projection of the modified gradient has changed its sign and increased in magnitude. It is even useful to demand that the magnitude of the gradient has to drop by a factor, e.g., below 0.9 times its starting value. When such an overstepping occurs for the conjugate gradient optimizer of ParaTools, the erroneously accepted structure and the starting structure of the last iteration are interpolated. This process is called backtracking.[136] Only one backtracking step in sequence is allowed.

15.3 The Fast Inertial Relaxation Engine (FIRE) Optimizer

Like the conjugate gradient optimizer, the FIRE optimizer was originally designed for finding local minima on a potential energy surface.[144] The FIRE optimizer has its grounds in molecular dynamics. The relaxation is considered as a kind of motion, downhill along the potential energy surface in the direction of the so-called velocity which is multiplied by a “time step” to get the length. In each step one interpolates a new direction for the velocity from the previous direction and the steepest descent direction created from the gradient of the energy. The projection of the gradients on the velocity is negative as a rule. The so-called “time step”, which is used as a trust radius, can be increased as long as this condition is met. If the projection of the gradient on the velocity is positive, the proposed step is rejected and the algorithm is restarted, as it is assumed that the algorithm overstepped the minimum. A variety of parameters are available which, for example, define how the time step is modified, how much the velocity vector is influenced by the gradient direction and how this relationship will change over time. In the original proposal of the method it was mentioned that, except for the maximum time step, all parameters have been fixed in their test calculations.[144] Here, the same parameters[144] are also used.

The FIRE optimizer can also be used for path optimization.[142] The energy itself is never used in its algorithm, instead the goal of optimization is to let the gradients vanish. This can also be done for the modified gradient vector of a string or NEB method. FIRE is supposed to be especially suitable in cases of large or inaccurate gradients but to be slower than conjugate gradient in all other cases.[143] This method was implemented into ParaTools in the scope of the current work. It follows the original description[144] using the suggestions of its applications to path optimizations when required[142] and is adapted to work with the objects and path searching methods of ParaTools.

16. The Climbing Image Method

The climbing image (CI) approach combines the search for a path with the search for a transition state by choosing one of the path images as the “climbing image” which is supposed to converge towards the transition state. This image is chosen, after some iterations of preoptimization, as the image of largest energy.

Path searching methods without climbing image can only provide approximations for transition states and cannot deterministically shift an image to the transition state. Thus, the CI approach makes the transition state estimate less ambiguous than the estimate from images of a path approximation.

16.1 Climbing Image Nudged Elastic Band (CI-NEB)

The climbing image approach was first described[123] as a variant of the NEB method. The NEB method was already included in ParaTools. Thus implementation the CI-NEB method, as done in the context of the current work, is straightforward. The NEB approach adds for every image *i* spring forces $\mathbf{f}_i^{\text{spring}}$, which are parallel to the tangent $\mathbf{t}_i$, to the part $\mathbf{f}_i^{\perp}$ perpendicular to the path of the force vector **F**, to generate the modified force vector $\mathbf{F}^*$. The climbing image c should not be affected by the distances to the other images; these distances will be only used for generating the path tangent $\mathbf{t}_c$ at the location of the climbing image. The string forces are therefore not applied to the climbing image. Still it should react different on the forces perpendicular $\mathbf{f}_c^{\perp}$ and parallel $\mathbf{f}_c^{\|} = (\mathbf{f}_c \mathbf{t}_c)\mathbf{t}_c$ to the tangent. At the climbing image both components of the forces should finally vanish. This is correlated with a minimization of the energy in the direction perpendicular to the path and with a maximization of the energy in the direction parallel to the path. As the optimizer tries to minimize the energy in all directions, the parallel forces have to be reversed. The modified force of the climbing image thus is $\mathbf{f}_c^* = \mathbf{f}_c - 2\mathbf{f}_c^{\|}$. The other images are treated as usual, see Chapter 13. They tend to become equally distributed on the two substrings between the minima and the climbing image. This happens because the two substrings do not influence each other.

16.2 Climbing Image String (CI-String) Method

The main aspects of a CI-string method[124] are essentially the same as for the CI-NEB method. The variant implemented in the current work differs in the basic string method, to which the climbing image approach was added. The modified forces, as processed by the optimizer, contain again the inverted parallel part for the climbing image *c*,

$$\mathbf{f}_m^{(j)*} = \begin{cases} \mathbf{f}^{(j)} - 2\mathbf{f}_{\|}^{(j)} & \text{for m} \equiv c \\ \mathbf{f}^{(j)} - \mathbf{f}_{\|}^{(j)} & \text{otherwise} \end{cases}, \tag{16.1}$$

and are projected perpendicular to the path for the other images. Weinan et al.[124] followed a different procedure for the modified forces on the regular images, by using the complete forces for the minimization. In both string methods, the images occasionally need to be respaced.

Let w_j be the desired relative distance, with $0 \le w_j \le 1$, between the first minimum and image *j*, and let l_c be the length of the path from the first image to the climbing image *c* divided by the complete path length. The relative length l_j^*, to be restored in the respacing process, for image *j* is then

$$l_j^* = \begin{cases} l_c w^{(j)} / w^{(c)} & \text{for } j \le c \\ 1 - (1 - l_c)(1 - w^{(j)}) / (1 - w^{(c)}) & \text{for } j > c \end{cases} \tag{16.2}$$

This relative length is calculated for each of the two substrings starting from the minima toward the climbing image. During a respacing step the images are distributed along the path according to the relative length l_j^*. Note that the position of the climbing image at the path is not changed by this procedure for $j = c$.

The CI approach is only started after some steps with the standard string method. It is required to use explicit and variable relative lengths l_j^* for the algorithm as the growing string[66g, 67] and searching string[118] methods do not require that the images should be equidistantly spaced. In the growing string and searching string methods, the string has to be fully grown before a climbing image is selected.

17. Contiguous Path Parameterization for a String Method

The string implementation of ParaTools makes use of cubic splines through N_{beads} images. The paths $\mathbf{p}(s)$ as a function of the path coordinate s generated by having such a spline for every coordinate should approximate the minimal energy path. These paths are used for defining the tangents at the images and for image respacing. Both operations could be done without splines[142] and are independent of each other. In ParaTools splines are used for both operations, which requires that one chooses path parametrization parameters (abscissas) $\{s_i\}$ that generates positions $\{\mathbf{p}(s_i)\}$ which correspond to the image positions $\{\mathbf{x}_i\}$.

17.1 Area of Application for Abscissas

After one has chosen the abscissas, they can be used to generate a path. However, just as there is more than one way to parameterize a path, there is more than one way of defining the abscissas, see Sections 17.2 and 17.3. The abscissas for the image positions $\mathbf{p}(s_i) = \mathbf{x}_i$, the N_{beads}-dimensional vector $\mathbf{S}$ of the abscissas s_i, have to be given as input for building the spline. Together with the geometry vector $\mathbf{X}$, with $X_i = \mathbf{x}_i$ the path $\mathbf{p} = \text{path}(\mathbf{X}, \mathbf{S})$ of cubic splines used in ParaTools string methods is completely defined when the abscissas are selected.

For a given path $\mathbf{p}(s)$ as a function of the path coordinate s calculating tangents at image positions and respacing of image positions is straightforward. The tangents $\mathbf{T}(\mathbf{p}, \mathbf{S})$ are the vector of the first-order derivatives $\mathbf{t}_j = \partial \mathbf{p}(s_j) / \partial s$ at position s_j. A respacing is performed when necessary after the optimizer has proposed a step:

$$\mathbf{X}^{(m+1)*} = \mathbf{X}^{(m)} + \Delta X^{(m)}(\mathbf{T}^{(m)},\ \mathbf{X}^{(m)}, \mathbf{G}^{(m)}, [E^{(m)}]) \qquad (17.1)$$

The step is dependent on the positions $\mathbf{X}^{(m)}$, the gradients of the energy with respect to the nuclear positions $\mathbf{G}^{(m)}$ and the tangents $\mathbf{T}^{(m)}$. The optimizer Multiopt uses also the energies

$E^{(m)}$ for its trust radius. The step can now be either accepted as is, if the path length deviations $D_i^{(m+1)} = \left(l(0,s_j,\mathbf{p})/l(0,1,\mathbf{p})\right) - w_j$ to the desired relative distance w_j are acceptable. The function $l\left(s_a,s_b,\mathbf{p}\right)$ measures the length of the path section between $\mathbf{p}(s_a)$ and $\mathbf{p}(s_b)$ by integrating over the length of the tangent using a norm defined by the metric, see Chapter 18. If the path length deviations are not acceptable new positions $\mathbf{S}^*$ on the path are generated

$$\mathbf{X}_i^{(m+1)} = \begin{cases} \mathbf{p}^*(s^*_i) & \text{for } \max\limits_k(|D_k^{(m+1)}|) > \text{tol} \\ \mathbf{X}_i^{(m+1)*} & \text{otherwise} \end{cases} \tag{17.2}$$

A function, which uses the inverse of the discretized path length function $l\left(s_a,s_b,\mathbf{p}\right)$, yields positions $\mathbf{S}^*$ on a path $\mathbf{p}^* = \text{path}(\mathbf{X}^{(m+1)*},\mathbf{S})$ which are spaced to fulfill the desired distribution $\mathbf{W}$ of the path nodes. Note that the new positions $\mathbf{S}^*$ and the abscissas vector $\mathbf{S}$ will generally not be equal. Thus if a new path with the new positions $\mathbf{S}^*$ and the corresponding geometries $\mathbf{X}^{(m+1)*}$ is generated, it will usually not correspond to the path $\mathbf{p}^* = \text{path}(\mathbf{X}^{(m+1)*},\mathbf{S})$.

17.2 Requirements for Abscissas for Spline Representations

At first thought one would not expect that the abscissas $\mathbf{S}$ have a large effect on the path $\text{path}(\mathbf{X},\mathbf{S})$ as the abscissas are merely auxiliary parameters. For example, scaling the abscissas or adding a constant term to all of them, results in the same tangents and the same geometries after respacing. Without loss of generality one may choose $s_0 = 0$ and $s_n = 1$ for N_{beads} images. The distribution of the remaining images on the abscissa strongly influences the path; consequently the tangents, and the new geometries after respacing, are also affected by the choice.

Figure 17.1 illustrates this for a one-dimensional problem. Three splines generated from the same image data but different abscissas are shown. When the abscissas are chosen equally-spaced, the obtained spline is a straight line while for the other choices curves are generated. Looking for the geometry of a point halfway between the first two images, the path marked by diamonds gets $\mathbf{p}((s_1+s_0)/2) = -0.55$ for this interpolated geometry, which is far off the range of the input values between 0 and 1. The other path marked with triangles is not monotonic either but remains in the domain between 0 and 1. This illustrates that the abscissas have a large effect on the path and the overall calculations. Each of the three examples would provide a different result for respacing or the tangents. The most important aspect about them is that the abscissas for consecutive paths are adapted to each other because otherwise convergence might be prevented. Imagine for example that a calculation would switch between the three abscissa choices of Figure 17.1 during the calculation.

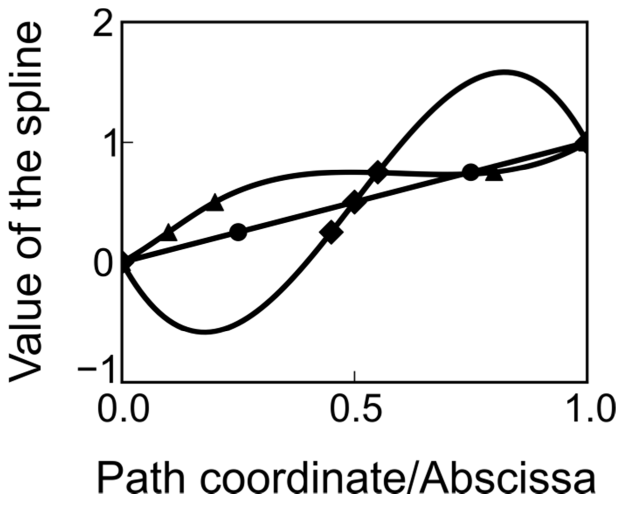

Figure 17.1: Spline shape as function of the distribution of abscissa values. For a one-dimensional problem five values evenly distributed between 0 and 1 are specified (vertical axis). Paths as represented by different markers, depend strongly on the choice of abscissas (horizontal axis). The two terminal images have the same abscissas for all three paths. Circles and diamonds additionally share the (third) node in the middle.

17.3 Abscissa Implementation

It has been already mentioned in Section 17.2 that there exist several possibilities of how the abscissas can be chosen. A simple way would be by keeping a fixed vector of abscissas, $\mathbf{S} = \text{const}$. This approach is very attractive for its simplicity. However, the main drawback is that these abscissas are not taking care about how the images are distributed in the space. Two images far apart from each other might still yield a small distance according to the path length if the difference in abscissa values is small.

The current implementation of the abscissas is quite opposite to this approach and relies on the coordinates of the images $\mathbf{X}$. Instead of storing the values of the abscissas, they are always calculated anew. Besides on the vector of geometries $\mathbf{X}$ for the images, the abscissas depend on the metric (see Chapter 18) which is used for calculating the distances between the geometries $\mathbf{X}$. Whenever a path is required for a vector of geometries $\mathbf{X}$, both for the tasks of getting tangents or for respacing, the abscissas are given as

$$s_j = \sum_{k<j} u_k \Big/ \sum_{k<n} u_k \quad \text{with } u_j = \left|\mathbf{x}_j - \mathbf{x}_{j+1}\right|_{\text{metric}} \tag{17.3}$$

This corresponds approximately to the relative length of the path segments that connects the images. One advantage of this implementation is that the path is only a function of the geometries. Thus also the tangents require only the geometries. Another advantage is that a smooth path with a sufficiently high number of uniformly distributed images will necessarily yield a path length proportional to these abscissas. This approach also decouples the way a path is represented from the question of the quality of the path. If, for example, the simple approach for the abscissa, with a fixed vector $\mathbf{S} = \text{const}$, would be used, the paths would differ for various choices of this vector $\mathbf{S}$ but the same images. A small drawback may be that the optimization process yields different abscissas for each iteration, which may be not that stable as fixed abscissas. Changes in the geometries and changes in the abscissa may accumulate and lead to changes in the tangents or in the geometries through respacing which are larger than it would be necessary for a different choice of abscissas. However, one can expect that near convergence, where the changes between the image geometries are small, the changes in the abscissas will also be small.

18. Metric

18.1 Requirement for Considering Metric

In principle it is possible to treat a metric exactly, but that often is not done because of the entailed numerical effort. The choice of a metric plays a role when reaction path calculations done in different coordinate systems are compared. For comparison of distances, angles between vectors or vector products it is required to have a consistent metric for the coordinate systems. Such problems may be circumvented by using the same coordinate system for all calculations, for example a Cartesian or mass-weighted Cartesian coordinate system. However, internal coordinates, e. g. Z-Matrix or natural internal coordinates, often are advantageous.[97, 145-146] They can reduce the dimension of the system by eliminating global positioning parameters[97] and they can reduce coupling terms in the functions describing the potential energy surface.[146] The current chapter deals with the requirements which appear if internal coordinates should be used as coordinate system for calculations especially with path searching routines.

The current section illustrates one effect when the metric is not considered for the mentioned comparisons. Explanations for the effect are provided in Section 18.3. The current section should be seen only as a motivation for taking some effort in choosing a suitable metric. The effect of various choices of the metric is shown on minimum energy paths, see Section 13.1, to which the chain-of-state methods are supposed to converge for an infinite number of images and at ideal convergence. As an exemplary system a simple analytical potential with two coordinates is chosen: The Müller–Brown potential[147], parameterized by the two variables x_1 and x_2, features three minima and two transition states in the explored rectangular region,

$-1.2 < x_1 < 0.8$ and $-0.2 < x_2 < 1.8$ (Figure 18.1). The variables x_1 and x_2 will be in the following referred to as x coordinate system. For testing purposes a second set of coordinates $y = (y_1, y_2)$ is introduced, which is related to the original set $x = (x_1, x_2)$ by

$$\begin{aligned} x_1 &= y_1 + y_2 / 2 \\ x_2 &= y_2 / 2 \end{aligned} \tag{18.1}$$

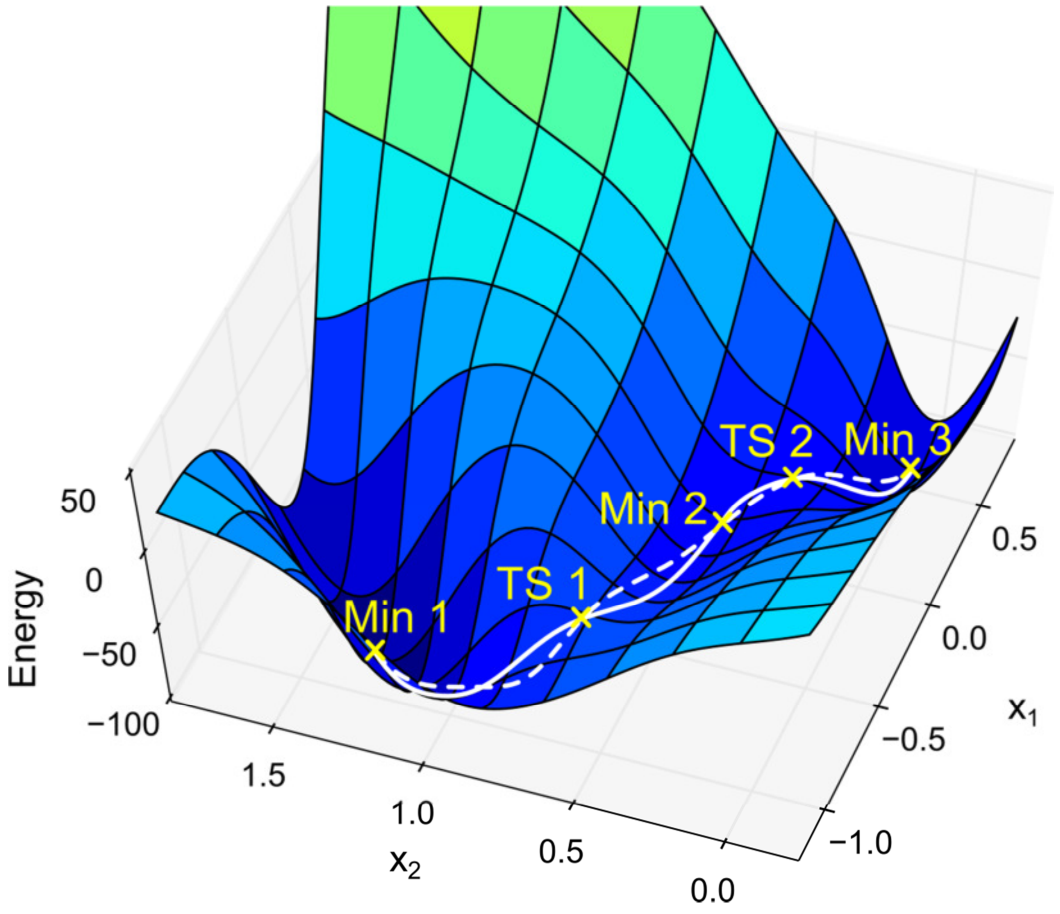

Figure 18.1: Müller–Brown potential with ideal minimal energy path for two parameterizations of the potential energy surface, original (x coordinates, solid) and mapped (y coordinates, dashed).

The simple Euclidean metric, called L_2-metric in the following, corresponds to a norm related to the coordinate choice. This metric is not appropriate for comparing minimal energy paths in various coordinate systems. Thus it is used to demonstrate the errors that appear due various choices of the metric. This is related to the fact that for two coordinate systems both associated with the L_2-metric, one is looking at two different potential energy surfaces. For each of the coordinate systems one might get the ideal path by numerically integration[148] of the differential equation for the minimal energy path, given by the steepest descent equation, Eq. (13.1).

$$\frac{\partial \mathbf{r}}{\partial s} = -\mathbf{g}(\mathbf{r}(s))$$

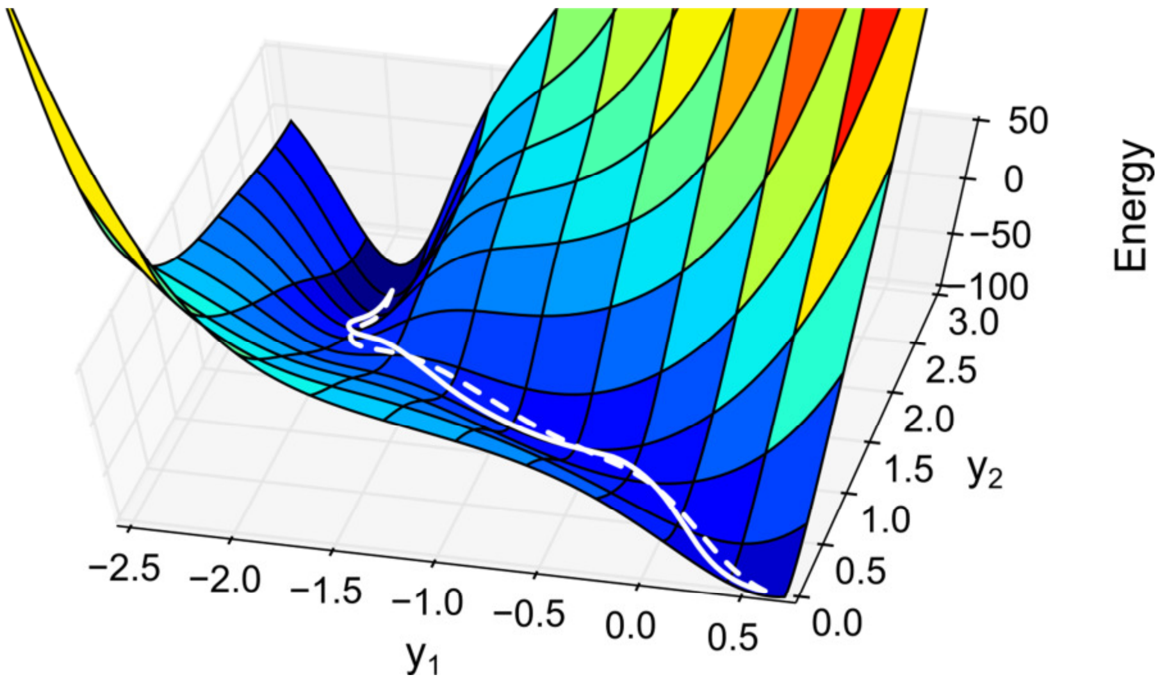

Figure 18.2: Müller–Brown Potential in mapped coordinates with ideal path for two parameterizations of the potential energy surface, using the coordinate systems x (solid) and y (dashed).

The ideal paths are shown in Figure 18.1. These paths for the two coordinate systems are exact to a good accuracy, having only perpendicular gradients left with a square smaller than 0.008 for all discretization modes in their own coordinate system and the L_2-metric. The two ideal paths expose the same stationary points but differ in all other points which is caused by the type of transformation. The original path in x coordinates, the solid curve in Figure 18.1, follows the valley of the Müller–Brown potential surface. The path in y coordinates, dashed curve in Figure 18.1, leaves the valley and has parts where it does not follow the gradients in the x coordinate parameterization. However, this looks quite the opposite when the paths are both observed in y coordinates, see Figure 18.2. In these coordinates the path in y coordinates follows the steepest descent path. This illustrates that it is only possible to say which of two paths corresponds more to the expected minimum energy path after the genuine coordinate system has been chosen.

The energy profile of the two paths is very similar, see Figure 18.3. However, the energy values do not only belong to different pathways on the energy surface, but also to different system configurations, except for the stationary structures.

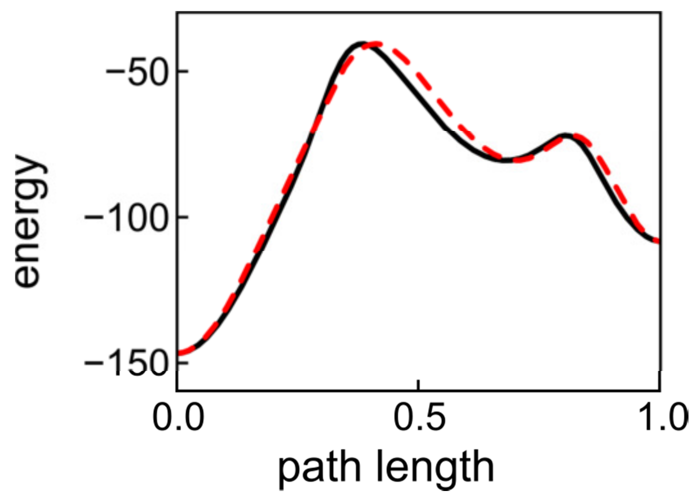

Figure 18.3: Energy profile for the Müller–Brown potential energy surface in x (solid) and y (dashed) coordinates. The path length is measured along the path by the L_2-norm in the respective coordinate systems.

18.2 Metric Formalism

The current section explains the basic concepts of a metric. The description follows closely the one of Brown.[149]

For the total energy $E(x^1, x^2, \ldots, x^n)$ as a smooth continuous function of n coordinates, the total differential dE can be expressed using the incremental changes dx^i in the variables x^i and the partial derivatives $\partial E / \partial x^i$ of E with respect to x^i:

$$dE = \sum_i \frac{\partial E}{\partial x^i} dx^i \tag{18.2}$$

Two vectors can be defined in order to express this total differential as a vector product $dE = \mathbf{g} \cdot \mathbf{d}$

$$\mathbf{g} = \left[\frac{\partial E}{\partial x^1}, \ldots, \frac{\partial E}{\partial x^n} \right]$$
$$\mathbf{d} = \left[dx^1, \ldots, dx^n \right]$$

where **g** is the so-called gradient vector of the energy function E. While **g** is a vector with so-called co-variant components, the differential coordinates **d** form a vector with so-called contra-variant components. The differences between these two kinds of vector representations become visible when a transformation of the coordinates (to a coordinate system with variables y^i) is performed. The differential in the new coordinates can be expressed, using the Jacobean matrix $\mathbf{J}$ with $\partial y^i / \partial x^j$, as

$$dy^i = \sum_j \frac{\partial y^i}{\partial x^j} dx^j \tag{18.3}$$

A similar expression can be derived for the gradients by using the total differential.

$$k_i = \sum_j \frac{\partial x^j}{\partial y^i} g_j \tag{18.4}$$

The main difference between Eqs. (18.3) and (18.4) is in the different coefficients for the transformation. This results in different behavior of the two kinds of coordinate components by coordinate transformations and thus the different names for the representations of these components. For vectors these are the only representations possible.

Gradients and the differential coordinates are as vectors tensors of the first order. The observation can also be extended for tensors of the second order. They are of special interest if the length *ds* of a differential coordinate vector should be obtained. In this case the squared differential distance can be derived from the generalized Pythagorean theorem using the corresponding differential coordinate **d**.

$$(ds)^2 = \sum_\mu \sum_\nu \eta_{\mu\nu} dx^\mu dx^\nu \tag{18.5}$$

According to the Einstein convention, a summation is implied if in a formula the same index is used twice, in an upper and a lower position. This allows to simplify the equation.

$$(ds)^2 = \eta_{ij} dx^i dx^j \tag{18.6}$$

This formula is an important expression for the metric of the space. However, it depends on the coordinate system as the so-called metric tensor $\boldsymbol{\eta}$ will look differently in another coordinate system. However, as the vector product $dx^i dx^j$ can be transformed using Eq. (18.3) one can transform also the metric expression using the assumption that the length *ds* should stay the same. This gives with

$$dx^i dx^j = \frac{\partial x^i}{\partial y^k} \frac{\partial x^j}{\partial y^l} dy^k dy^l \tag{18.7}$$

a chance to express the length *ds* as before by adding the changes to the metric tensor $\boldsymbol{\eta}$.

$$\left(\eta_{kl}\right)^{(y)} = \frac{\partial x^i}{\partial y^k} \frac{\partial x^j}{\partial y^l} \left(\eta_{ij}\right)^{(x)} = J^i{}_k \left(\eta_{ij}\right)^{(x)} J^j{}_l \tag{18.8}$$

The metric tensor can be inverted, which results in the representation with upper indices η^{ij}.

For the example of coordinates of Section 18.1 and the choice of the metric tensor $\boldsymbol{\eta}^{(x)}$ in the original x coordinate system to be the unit matrix ($\eta_{ij} = \delta_{ij}$, using the Kronecker delta symbol which is 1 if the two indices are the same and 0 otherwise), the metric matrix for the y coordinate system is:

$$\boldsymbol{\eta}^{(y)} = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 0.5 \end{pmatrix} \tag{18.9}$$

Every vector has a co- and a contra-variant representation. The gradients **g** and differential coordinates **d** defined above are only in one specific representation fixed by their usage in the total differential; see Eq. (18.2). For orthogonal coordinate systems with coordinates **x** (for which the coordinate axes are mutually perpendicular) the two representations are the same. In this case the coordinates **x** represent a genuine coordinate system. However, also for other coordinate systems it is possible to transform from one representation into another:

$$\delta y^i = \eta^{ij} \delta y_j \tag{18.10}$$

This is required if two vectors should be added to each other; they have to be in the same representation for this operation. Furthermore one is able to write the square of the length of the differential coordinates by using the two representations of a vector.

$$(ds)^2 = dx_i dx^i \tag{18.11}$$

To properly use the metric, without having to deal explicitly with the metric matrix, is to make sure that for an addition vectors are always in the same representation while for a vector product they are in different ones.

18.3 Metric Requirements for Transition State Search

Metric considerations are important for searching the minimal energy path as the metric may distort the path if not properly chosen. This can be seen on the example of different genuine coordinate systems, compare Section 18.1. This effect originates in the definition of the minimal energy path in Eq. (13.1). As mentioned in Section 18.2 the coordinate differences, to which belong the tangents, are provided in a contra-variant vector representation, while the gradients are given with co-variant components. Thus, the fundamental Eq. (13.1) of steps **t** and gradients **g** has to deal with both coordinate representations and should therefore be written as

$$t^j = -\eta^{ji} g_i \tag{18.12}$$

Only in the genuine coordinate system, where the metric tensor η_{ij} is always the unit matrix, one may chose not to write the metric tensor; in any other coordinate system the proper metric has to be used to end up with the same path. If the metric is not chosen suitable and instead the transformation matrix is set in every coordinate system to the unit matrix a situation like in Section 18.1 occurs where the paths are no longer comparable. This procedure is only valid in cases where the main interest is only in stationary points like transition states, which are, in an ideal consideration, not affected by any change of coordinates. This is the case for local transition state searching methods.

For the genuine coordinate system several choices have been considered in the literature,[97, 150-152] mass-weighted coordinates and Cartesian coordinates seem to be the most prominent choices. Fukui describes his intrinsic reaction coordinate[97] exclusively in mass-weighted coordinates. As Quapp and Heidrich pointed out,[153] the physical model of Fukui's IRC path, being a dynamical limit with vanishing kinetic energy is problematic as the steepest decent paths, to which the IRC belongs, seem to use a dynamical description as a motivation and at the same time neglect the dynamics by removing the kinetic energy from the picture. This reduces the path to a mathematically defined curve which follows a decrease in the potential.[153] Minimum energy pathways and also steepest descent paths described in Cartesian coordinates can be quite similar to their counterpart in mass-weighted coordinates,[151] although this is not compulsory. Calculations carried out with different genuine coordinate systems can lead to different results, as already observed by Sana et al.[152]

In principle there is no reason, why one cannot describe minimal energy paths also in other coordinate systems; an example is the first description of Fukui's intrinsic reaction coordinate which uses "$3N-6$ independent internuclear distances"[150] as genuine coordinates on the potential energy surface. All these paths, whether they are based on mass-weighted or Cartesian coordinates, are nevertheless simplifications, as they do not consider, for example, the kinetic energy. It is supposed that the deviation to the real chemical dynamical path, that is the path the reaction would take in reality, of a reaction processes is small for small kinetic energies.[154] Thus, choosing one of the genuine coordinate systems seems to be more a question of taste. On the other hand, employing a correct metric allows one to choose the coordinate system, in which the calculation is performed, freely. The necessary transformation have been described by Fukui[97] and by Quapp and Heidrich.[153]

18.4 Metric in ParaTools

As ParaTools permits the usage of various coordinate systems it is necessary to consider the metric, in order to achieve comparable results of transition state optimization, irrespective of the coordinate choice, see Section 18.3. This is especially important for the path searching routines, which will be the focus of the following evaluation. However, the metric can also be used for local transition state search methods.

The following Sections 18.4.1 and 18.4.2 deal with the implementation of the metric in ParaTools. ParaTools contains three implementations of the metric: the first is a simple Euclidean metric. The other two for internal coordinates are derived from an Euclidean metric together with Cartesian coordinates and the coordinate transformation between internal and Cartesian coordinates. The two variants differ in the way the global positioning, the orientation of the system in space, is handled. The last section of the current chapter, Section 18.5, demonstrates the effect of the choice of the metric on the convergence of paths and on maintaining symmetry properties.

18.4.1 The Simplest Case: The Euclidean Metric

For some of the routines, such as the local transition state search routines, the choice of the metric will have only a small performance effect. The results of these methods do not heavily rely on orthogonality or step length constraints of coordinates, as they are searching solely for a coordinate system independent object. Consequently, these methods can ignore the metric in practice. This is also possible for the path searching routines. In many cases only the transition state estimate from the path is of interest and rough convergence criteria are used or only few images are chosen. In these cases it is reasonable to use a simplified metric, where the contra- and co-variant components of the vectors are identical, thus the transformation matrix is always the unit matrix $\eta_{ij} = \delta_{ij}$. This is the numerical most efficient method. Effectively, this corresponds to an Euclidean metric, irrespective of the type of coordinates used. In the following this case will be referred to as Euclidean metric.

18.4.2 Non-Euclidean Metric

The metric, as used in the following, uses the Cartesian coordinates **x** of the atomic system as genuine configuration space, with contra- and co-variant vector representations being equivalent. This metric is, besides the metric using mass weighted Cartesian coordinates as genuine configuration space, the most ubiquitous metric for path searching routines.[97, 153]

18.4.2.1 Metric without Explicit Consideration for Global Positioning

For describing the relation of contra- and co-variant components of vectors in arbitrary (internal) coordinates **y** it is only required to define the matrix η_{ij}, see Eq. (18.10). With help of the matrix of first-order derivatives from Cartesian to internal coordinates, the Jacobean matrix $\mathbf{J}^i{}_j = \partial x^i / \partial y^j$, this relation looks like:

$$\delta y_i = \left(\mathbf{J}^l{}_i \delta_{lk} \mathbf{J}^k{}_j\right) \delta y^j = \left(\mathbf{J}^T \mathbf{J}\right)_{ij} \delta y^j \qquad (18.13)$$

This relationship can be used to transform vectors with contra-variant components into vectors with co-variant components. For the reverse transformation from co- to contra-variant components the matrix $\boldsymbol{\eta} = \mathbf{J}^T \mathbf{J}$ has to be inverted.

These relations are enough to define a proper metric. However, there is still one thing to consider, as in some cases the metric includes some ambiguity. The following subsection will deal with this case of a "reduced metric". The remaining part of this subsection will provide the reason for requiring two versions of the metric. The dimension of the Cartesian coordinate system of an atomic system is always 3*N* for *N* atoms involved. The dimension for internal coordinates $\mathbf{y}$ can be smaller: $\dim \mathbf{y} \leq 3N$. For example, in case of Z-Matrix coordinates, it would be $3N-6$, with the global positioning coordinates for global translation and global rotation of the whole system being eliminated. These global positioning coordinates are often removed because the energy of a molecular system does not depend on the position and orientation of the system as the whole. However, if a transformation into a 3*N*- dimensional space of, for example, Cartesian coordinates is carried out it is required that a global positioning exists to provide all coordinates for this space and allow an injective transformation. As such a transformation is required for the metric, the global positioning is implicitly present. The six degrees of freedom representing the global positioning can be chosen arbitrarily and are never used directly. They are usually fixed. It is possible to use the metric as defined above for the cases with a smaller dimension of the internal coordinate system, as the matrix $\boldsymbol{\eta} = \mathbf{J}^T \mathbf{J}$ is still invertible. However the Jacobian $\mathbf{J}$ cannot be inverted by itself, leaving some ambiguity for the Cartesian coordinates, which are implicit present in the form $J_{ij}\delta y^j$.

The effect of the resulting ambiguity can be best explained for an example: the internal coordinates can include the implicit global translation by keeping the geometrical center at the origin (if a transforming into Cartesian coordinates is done). If the difference between two geometries in this internal coordinates would be calculated, there will be two contributions: one related to the changes in the internal coordinates and one related to the resulting change in the geometrical center. The second contribution is unwanted.

These effects do not have to appear in all cases in which the internal coordinates do not include the global positioning. In other cases they might be negligible. However, for a correct approach it is required to consider these effects.

When Fukui explained how his intrinsic reaction path could be calculated in Z-Matrix coordinates, he was also aware of having to consider global positioning; he solved the related ambiguity by introducing six more relations to be fulfilled besides the $3N-6$ relations for the transformation of the coordinates.[97] The update procedure he used for his path fulfilled these relations in a natural way. However, for the path searching methods of ParaTools applying this special treatment as restriction on the possible update methods for the paths appeared impractical. Therefore ParaTools includes a different solution by providing the reduced metric, see the following subsection.

18.4.2.2 Reduced Metric Considering Global Positioning

Other than the usual metric, Subsection 18.4.2.1, or the simple metric, Section 18.4.1, the reduced metric discussed here can only be used for a special kind of coordinate systems. These coordinate systems have to be translationally and rotationally invariant, thus the global positioning parameter must not be any of the coordinates. This limits the usability of this metric. However, for coordinate systems without global positioning, it is the correct metric. For these coordinate systems it has even some advantages over the alternative of extending the coordinate systems by a global positioning object and employing the usual metric. The first advantage is that the approach with the reduced metric does not contain as many coordinates as the second approach. The second advantage is that the explicit global positioning parameters, which are added for the second approach, might also induce an effect in the length and other metric related properties. As global positioning should have no such effect, removing these effects should be a superior strategy.

The reduced metric will be derived as standard metric (see Section 18.4.2.1), where the internal coordinates **y** are augmented by six more coordinates **v**, which later will be removed again from the final equation. This results in vectors of the kind $\delta\mathbf{s} = (\delta\mathbf{y}, \delta\mathbf{v})^T$, with transformations between co- and contra-variant representations like as in Subsection 18.4.2.1. Eq. (18.13) can still be used to describe the transformation between the representations. One only has to note that $\mathbf{J} = (\mathbf{J}_y, \mathbf{J}_V)$ contains next to the original Jacobian matrix $\mathbf{J}_y$ also a second submatrix $\mathbf{J}_V$, which represents the first-order derivatives of the Cartesian coordinates with respect to the global positioning parameters. Eq. (18.13) is now extended to the relations (with $i, j = 1,...n$ and $p, q = 1,...6$ for $\dim y = n$):

$$\delta y_i = \left(\mathbf{J}_y{}^T\mathbf{J}_y\right)_{ij} \delta y^j + \left(\mathbf{J}_y{}^T\mathbf{J}_V\right)_{ip} \delta v^p$$

$$\delta v_q = \left(\mathbf{J}_V^T\mathbf{J}_V\right)_{qp} \delta v^p + \left(\mathbf{J}_V^T\mathbf{J}_y\right)_{qj} \delta y^j \tag{18.14}$$

Any displacement on the potential energy surface $E(\mathbf{y})$ is fully characterized by $\delta\mathbf{y}$, but neither δv^p nor δv_p have been defined yet. It is possible to choose six more relations

$\delta v_p \delta v^p = 0$ taking care of the constraints that the global positioning parameters should not contribute to the square norm $\delta s_l \delta s^l = \delta y_i \delta y^i + \delta v_p \delta v^p$. Two solutions for the additional relations that satisfy Eq. (18.14) for arbitrary $\delta \mathbf{y}$ are $\delta v^p = 0$ or $\delta v_p = 0$. The first choice would decouple the co-variant vectors in Eq. (18.14) from each other. This corresponds to the original metric, see Subsection 18.4.2.1, when used for internal coordinates without global positioning parameter. The other choice of $\delta v_p = 0$ leads to the "reduced" metric, which is the subject of this subsection. With this choice Eqs. (18.14) can be simplified, taking into account that δv^i is of no real interest.

$$\delta y_i = \left[\mathbf{J}^T \left(\mathbf{I} - \mathbf{J}_V \left(\mathbf{J}_V^T \mathbf{J}_V \right)^{-1} \mathbf{J}_V^T \right) \mathbf{J} \right]_{ij} \delta y^j \tag{18.15}$$

The derivation of Eq.(18.15) can be found in Appendix G. Although the variables δv^p and δv_p corresponding to global positioning parameters have been eliminated from Eq. (18.15), it is still necessary to define them, as they are required to generate the partial Jacobian matrix $\mathbf{J}_V$. Three degrees, **T**, of the six degrees of freedom of **v** = {**T**, **R**}, correspond to a translation of the system as a whole and the remaining three, **R**, to the rotation of the complete system around a specific point. The rotation center can be chosen as is convenient, e.g., the center of mass or the geometric center of the system. The partial Jacobian $\mathbf{J}_V = \left(\mathbf{J}_T, \mathbf{J}_R \right)$ can be parameterized the same way, with $\mathbf{J}_T$ and $\mathbf{J}_R$ being $3 \times 3N$ matrices containing N 3×3 matrix blocks $\mathbf{J}_T{}^{(k)}$ and $\mathbf{J}_R{}^{(k)}$ related to the atom *k*, respectively. With the definition $\mathbf{R}_k = (R_{k,1},\ R_{k,2},\ R_{k,3})$ for the coordinates of atom *k* with respect to the geometrical center of the system and the translation matrices $\mathbf{J}_T{}^{(k)}$ and the rotation matrices $\mathbf{J}_R{}^{(k)}$ for the atoms *k* are defined as:

$$\begin{aligned} \mathbf{J}_T^{(k)} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ \mathbf{J}_R^{(k)} &= \begin{pmatrix} 0 & R_{k,3} & -R_{k,2} \\ -R_{k,3} & 0 & R_{k,1} \\ R_{k,2} & -R_{k,1} & 0 \end{pmatrix} \end{aligned} \tag{18.16}$$

The transformation between the vector representations can be expressed in a simplified form, using $R_k = (\sum_j R_{k,j}^2)^{1/2}$:

$$\delta y_i = \left[\mathbf{J}_y{}^T \left(\mathbf{I} - \frac{1}{N} \mathbf{J}_T \mathbf{J}_T^T - \mathbf{J}_R \left(\sum_k R_k^2 \mathbf{I} - \sum_k \mathbf{R}_k \mathbf{R}_k^T \right)^{-1} \mathbf{J}_R^T \right) \mathbf{J}_y \right]_{ij} \delta y^j \qquad (18.17)$$

The derivation of this expression can also be found in Appendix G.

The metric in the square brackets of Eq. (18.17) will be referred to as reduced metric throughout this work. Note that the reduced metric and the metric both originate from the Euclidean norm for a parameter space where a given molecular system is represented by the Cartesian coordinates of the nuclei. The difference between the two systems is the treatment of translational and rotational invariance.

18.5 Effect of the Metrics in ParaTools

The metric affects reaction paths in several ways (Section 18.1). Also the rate of the path convergence can be affected by the choice of the metric. For example, a non-Euclidean metric, Subsection 18.4.2.1, and especially a reduced metric, Subsection 18.4.2.2, in combination with internal coordinates may augment chances for numerical errors than a simple Euclidean metric, Section 18.4.1. A third aspect, that will be described in the current subsection, is a manifestation of the choice of metric on differences in the steepest decent path for symmetric molecules and the symmetry properties of a path.

18.5.1 Müller–Brown Potential

As an example to demonstrate the influence of the choice of the metric, the Müller–Brown potential and the two coordinate systems described in Section 18.1 are used to demonstrate the influence of the choice of the metric using the two coordinate systems, *x* and *y* coordinates.

Two choices of the metric will be discussed. The first one is a simple Euclidean metric, an L_2-metric in the respective coordinate system, as defined in Section 18.4.1. For the second case one of the two coordinates is chosen as genuine coordinate system and the other coordinates are derived by a proper basis transformation. As genuine coordinates *x* coordinates have been chosen, thus the metric will be called x^2 metric. For a test several path searching calculations were run with the various metrics chosen.

There are three possibilities to run a string path optimization on the Müller–Brown potential. First one uses the *x* coordinates, for which both metrics describe the same case. The other two possibilities are to use *y* coordinates and either L_2 or x^2 metric. In principle it would be possible to construct a forth case, where the x coordinates are used with y^2 metric. However, it is not expected to provide additional insight as it is comparable to y coordinates with x^2 metric.

Table 18.1: Multiopt (M), conjugate gradient approach (C) and FIRE (F) optimizers for which convergence up to $\Delta g \leq 0.01$ is achieved for various numbers of images in x or y coordinates with L_2 or x^2 metric for the Müller–Brown potential.

Coord./met. Images	x/L_2	y/L_2	y/x^2
3	M, C, F	M, C, F	M, C, F
4	M, C, F	M, C, F	M, C, F
5	M, C, F	M, C, F	M, C, F
6	M, C, F	M, C, F	M, C, F
7	M, F	M, C, F	M, F
8	M, C, F	M, C, F	M, C, F
9	C, F	M, C, F	M, C, F
10	M, C, F	M, C, F	M, C
11	C, F	M, C, F	M, C, F
12	C	M, C, F	M, C, F
13	C,F	M	M, C, F
14	—	M, F	M, F
15	—	M, F	M, F
16	F	M, F	—
17	—	M, F	—
18	—	M, C, F	—
19	—	M, F	—
20	—	M	—
21	—	M, F	—

The results in Table 18.1 show the convergence behavior of these three choices of the test set. It was tested which of the three optimizers, Multiopt, conjugate gradient and FIRE, see Chapter 15, was able to converge the string approach. The convergence was measured with the gradient convergence measure which will be explained in Section 20.2 and for which the convergence threshold was set to $\Delta g = 0.01$. The convergence rate of the path in *y* coordinates with L_2 metric shows that these coordinates in the corresponding potential seem to be more stable as they provide convergence for the Multiopt optimizer for all examples with up to 21 images and for the FIRE optimizer for most of them. The conjugate gradient optimizer for all three test sets, shows convergence problems with increasing number of images. The calculation in x coordinates with L_2 metric did hardly converge at all for more than 13 images. The *y* coordinates with the x^2 metric suggest that the influence of the genuine coordinate system is larger than the effect of the coordinate system the calculations run in, as there is also hardly any convergence for more than 14 images. This is much more similar to the *x*-coordinates than to the *y*-coordinates in L_2 metric. However, there are still differences in the failures of the optimizers between the y coordinates with the x^2 metric and the x coordinates. This demonstrates how the metric may affect the numerical stability of the path searching procedure.

The choice of the coordinate system also affects the shape of the approximate path. Figure 18.4 shows the paths with three images and with eight images, calculated with the Multiopt optimizer. For the case with three images the paths are too simplified to represent the ideal path at all. However, it is already shown that the path optimized in y coordinates and x^2 metric approaches the path in x coordinates and with L_2 metric; the difference in the image positions is at most about 10^{-6}. For the case with 8 images the string calculations approximate the ideal path significantly better. Here the string calculation in x coordinates with L_2 metric and the calculation in y coordinates with x^2 metric are converging to the ideal path in original (x) coordinates. The third string calculation with *y* coordinates and L_2 metric is evidently converging to the ideal path determined in *y* coordinates. Of course, for a finite number of images differences from the ideal paths will remain, even with nearly perfect convergence. Obviously, it is evident that it is the metric, which influences the path to which the string method will ultimately converge in the limit of an infinite number of images and not the choice of the coordinates.

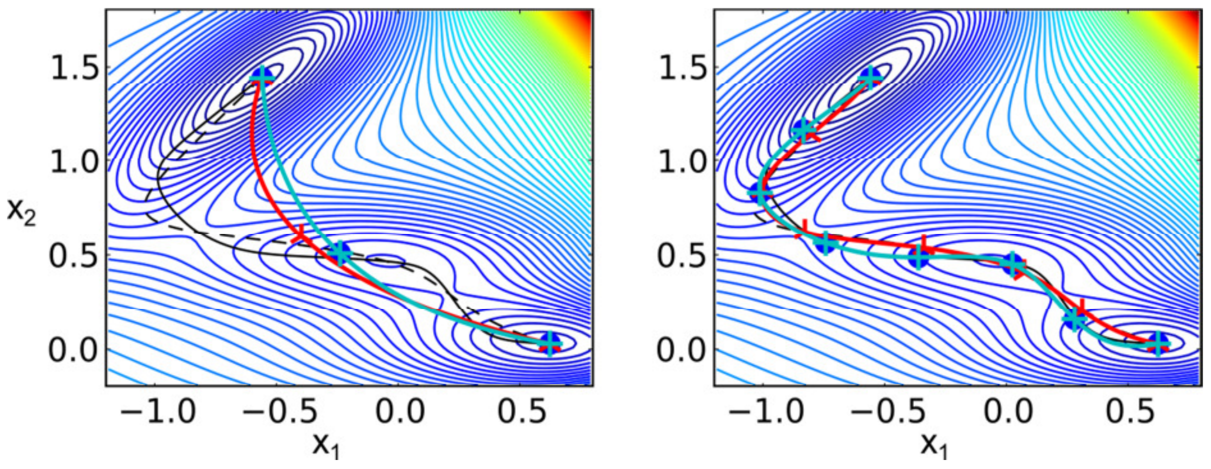

Figure 18.4: Reaction paths on the Müller–Brown potential. Given are the ideal paths for the two coordinate systems, *x* (solid line) and *y* (dashed line) coordinates with the L_2 metric. The string method was used to generate a path, with three (left panel) and eight images (right panel). The position of the images of the strings are indicated. Shown are paths in *x* coordinates and L_2 metric (dots), in *y* coordinates and L_2 metric (tri-up) and in *y* coordinates and x^2 metric (cross).

18.5.2 Lennard-Jones Potential

In the following example the influence of the choice of the metric on symmetry properties of a reaction path is inspected. The example was chosen to be quite simple, both in the geometry and the potential energy surface. However there will be already some effects visible. Various coordinate systems and various metrics are considered in the following. The Lennard-Jones potential[155]

$$U_{\mathrm{LJ}} = 4\epsilon\left[\left(\frac{\sigma}{|r_B - r_A|}\right)^{12} - \left(\frac{\sigma}{|r_B - r_A|}\right)^{6}\right] \tag{18.18}$$

approximates the interaction between neutral atoms by an analytical function. Relevant for the description of the systems are the distances between the atom positions r_A. The two parameters, σ and ε, can be adapted in order to model approximately a pairwise interaction between various types of atoms. In the following example both parameter were 1, thus no specific kind of atoms has been selected. It is always possible to convert the results to a specific kind of atoms by applying the change in the parameter.

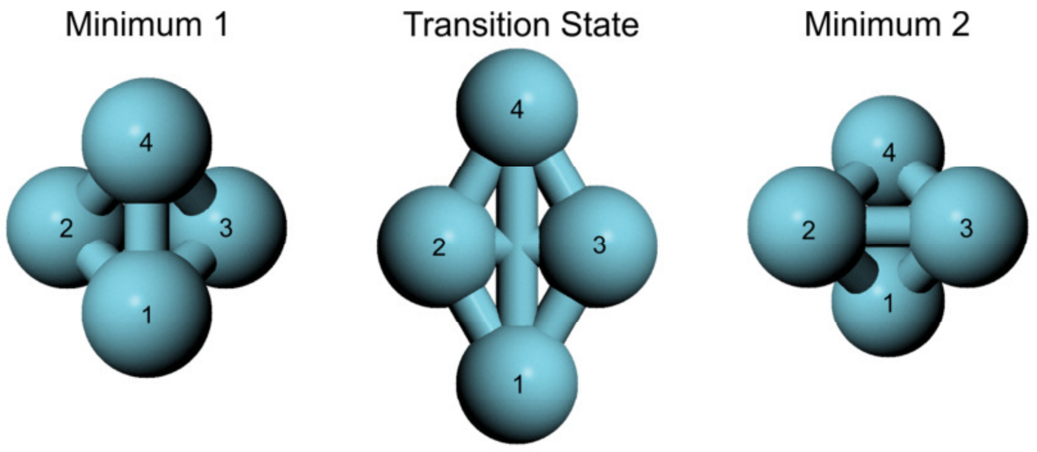

Figure 18.5: Transformation in C_{2V} symmetry between left- and right tetrahedral structures of a four-atomic cluster described by a Lennard-Jones potential. The atoms are labeled for ease of following their movement.

The isomeration reaction of a tetrahedron of four equivalent atoms via a rhombic transition state, Figure 18.5, interacting by a Lennard-Jones potential, follows a path completely in C_{2v} symmetry. This means that the structure is indistinguishable from the original one after a symmetry operation with one of the two orthogonal reflection planes has been performed. Furthermore there exists always a rotation around a specific point which exchanges the atoms 4 and 1 and simultaneously the atoms 2 and 3, see Figure 18.5 for the labels of the atom. The effect of the metric on the symmetry properties is shown for a steepest descent step, see Eq. (18.12), on a geometry which is supposed to converge to the transition state which is a planar structure. As the gradients only depend on the distances between the atoms they should naturally obey the symmetry of the atomic arrangement.

As starting geometry for the transition state search two arbitrary chosen geometries were considered. Geometry 1 is a flat rhombus, with the length for both diagonals being the same as the edge lengths. Geometry 2 is also a flat rhombus but shows diagonals of different length (one is about 1.7 times larger than the other). The length of the force vectors for the two geometries are about 0.3 and 0.2 gradient units.

For both geometries several choices of the coordinate system were examined. For geometry 1 they are the Z-Matrix coordinates, Z-Matrix coordinates with six additional coordinates for the global positioning (GP), Cartesian coordinates, and reduced Cartesian coordinates with six coordinates related to the global positioning being fixed. These fixed coordinates for the reduced Cartesian where chosen to be the same, which were implicitly fixed in the Z-Matrix coordinates. For geometry 2 Cartesian coordinates and reduced Cartesian coordinates with fixed GP were used. In this case the global orientation of the geometry in the Cartesian space was different, hence also the fixed coordinates where chosen differently. Note that therefore the reduced Cartesian coordinates differ for the two geometries.

Table 18.2: Symmetry breaking in the search of a transition state between two structures of a four-atomic cluster. For two geometries corresponding to first estimates to the transition state both fulfilling C_{2v} symmetry the deviation from C_{2v} symmetry was measured after a step along the steepest descent direction in the various coordinate systems. The applied metrics are Euclidean (E), non-Euclidean (N) or reduced (R) metric. Not all combinations are available (n. a.). For others the reached accuracy relative to the step size is provided.

Coord.	Degrees of freedom	E	N	R
Geometry 1				
Z-Matrix	3N-6	$3\cdot10^{-1}$	$8\cdot10^{-1}$	$2\cdot10^{-14}$
Z-Matrix + GPS	3N	$3\cdot10^{-1}$	$2\cdot10^{-14}$	n. a.
Cartesian	3N	$2\cdot10^{-14}$	$2\cdot10^{-14}$	n. a.
Reduced Cartesian	3N-6	$8\cdot10^{-1}$	$8\cdot10^{-1}$	$2\cdot10^{-3}$
Geometry 2				
Cartesian	3N	0	0	n. a.
Reduced Cartesian	3N-6	0	0	0

The six possible distances between the four atoms of the Lennard-Jones cluster can be used to quantify the derivations from C_{2v} symmetry. Two of the distances can be identified with the diagonals of the rhombus structure of the transition state (the distances between atoms 1 and 4, and between 2 and 3); the other four distances correspond to the edges of this structure. The edges have to be of same length due to the C_{2v} symmetry constraint for all geometries. One of the distances is used as reference length. The maximum length difference for one of the other three distances to the reference length is used as a measure of how well a geometry meets the C_{2v} symmetry. The geometries considered are the geometries after a step in the steepest decent direction, see Eq.(18.12). The symmetry inspected after such a step for the two geometries and the various coordinate systems is available in Table 18.2.

The Cartesian coordinates always preserve C_{2v} symmetry, see Table 18.2. For them the metric is the same as the L_2 metric. The Cartesian coordinates cover the complete space. However, having coordinates in $3N$ dimensions is not sufficient for guaranteeing that the symmetry is kept. The Z-Matrix coordinates with global positioning also cover the complete space but if the Euclidean metric is used the symmetry is broken as this is not the correct metric for these coordinates. Only with a non-Euclidean metric the symmetry is kept, see Table 18.2, as the underlying genuine coordinate system (Cartesian coordinates) maintain the symmetry. Thus, it is not enough, if a coordinate system is used, where it is merely ensured that the global positioning parameters are present. For the Z-Matrix coordinates without global positioning using the non-Euclidean metric did not change anything. Only if the reduced metric, especially designed for these cases, is used the symmetry is maintained, see Table 18.2. It is, of course, possible that the symmetry is maintained by the choice of coordinates. This is shown for geometry 2 with reduced Cartesian coordinates where the symmetry is for all metrics maintained, which is not the case for geometry 1. However, this example shows that the maintenance of the symmetry might be affected even by fixing some coordinates. Another challenge can already be identified: even though the reduced metric improves the preservation of the symmetry, rounding errors may still result in larger differences than in the other cases, like for the example Z-Matrix coordinates, as can be seen for the case of reduced Cartesian coordinates for geometry 1 in Table 18.2.

Already the simple example of a small Lennard-Jones cluster shows that the usage of the correct metric for a coordinate system can become important. In our case a specialized coordinate systems allows to avoid symmetry breaking. However, such coordinates are not always available and there might be even more properties, which are affected by the choice of the metric. Thus it makes sense to consider the proper choice of the metric for a given reaction.

19. Parallel Task Processing

While searching for a reaction path, the global transition state searching methods, like NEB and string methods, have to deal with several geometries of the images at once. For generating the next path approximation all of these methods require the gradients of the potential energy with respect to the nuclear positions of every image. The quantum chemistry calculations, which provide the gradients of the potential energy with respect to the nuclear positions, can be performed concurrently. Therefore parallelizing the gradient evaluations of various images appears meaningful. For a mere analytical code, like the ASE version of the Lennard-Jones potential, there is nearly no time spent in calculating the gradients. However for density functional codes, like ParaGauss or VASP, most time will be spent in these calculations making the effect of parallelization notable. The time requirements for the remaining Python code are negligible in comparison to the running time of a typical quantum chemistry calculation. For the path searching methods there are as many parallelizable tasks as there are moving images. There are also many tasks when frequencies of nuclear vibrations are to be calculated in numerical fashion, i.e., when second-order derivatives are estimated as finite differences of first-order derivatives. Using central differences there are 6N gradient evaluations for a system of N atoms, which can be determined independently. This turns frequency calculations into the routine where parallelization will improve the time requirement most; for comparison: path searching routines typically are used with about 5 independent moving images.[13, 118]

The task of processing the geometries in parallel is split into two subtasks. The first is about distributing the cores available on an instance of the quantum chemistry programs. Thus it manages the internal parallelization of the programs as well as the assignment of the cores for the quantum chemistry programs. The second one controls the activation of the various gradient evaluations, thus handles the distribution of the complete set of cores between the instances of the quantum chemistry program.

The first subtask, distributing the subset of cores for one instance of the quantum chemistry program, is the more demanding one. The python interfaces of the quantum chemistry codes have to meet the individual requirements of the code to be called, and are forced to allow using the codes without any change. Every quantum chemistry code has its individual python interface, however those python functions originally were not created for the case of being called in parallel. The codes were either restricted to calculate on one core or were using all available cores in parallel. Some of the codes provide the possibility of restricting the gradient calculation by the python interface to a given set of cores. The number of cores in this set can either be provided by the python interface itself or be fixed. The cores can either be defined by utilities of the operating system, which identify cores with the lowest load, or can be provided directly by the python interface.

Another strategy, which has shown to be successful on a Nehalem cluster (compare Subsection 5.1.2), delegates the scheduling decision to the job scheduling software (here grid engine). In this case ParaTools concurrently submits the jobs to the grid engine and blocks until completion. As an example, the gradients for the 102 geometries of a numerical frequency calculation for anthranilic acid (17 atoms), were calculated using this strategy with ParaGauss on the Nehalem cluster. Even though all gradient evaluations belong to similar geometries (differing by a fixed step in one of the degrees of freedom from the input) and have the same starting condition, the real calculation time per geometry varied by about 10 % (4100 ± 400s). However, the strategy used makes sure that only then computational resources are used when they are really required. Thus, the imbalance in the calculation time does not result in idle time as opposed to a case where a complete core set is exclusively used directly by the ParaTools program. There are two more factors, which affect the inhomogeneity of the running time of a task: some information of a gradient calculation could be reused for further gradient calculations. Additionally, the computer architecture needs not to be homogeneous. The advantage of the scheduling strategy described above is that the resources are only used for the gradient calculation when they are actually needed. On the other hand, this strategy only works efficiently if the queuing time is not too long. This scheduling strategy works for all quantum chemistry programs.

The second subtask of the parallel processing, deciding how to start gradient calculations, is solved in a general way, that works for all quantum chemistry codes. Two groups of functions are available in ParaTools. The simplest member of the first group uses a functionality of Python, provided in the form of Python's pool map function. A pool of workers is created by Python, which autonomously starts all the interfaces for the gradient calculations with the quantum chemistry code. The pool size and, thus, the maximum number of the concurrently executed tasks is limited. All calls to the quantum chemistry code demand the same number of cores.

The second group of functions was created for the case where the number of cores for the quantum chemistry code might be different for various tasks. For this purpose ParaTools contains an advanced function: the cores are assigned statically to the tasks. The tasks have to share a core set. First all tasks get the minimal (user defined) number of cores. However, it is often the case that not all tasks fit in the set of cores. In this case the set of cores has to be assigned several times. For the last assignment of the set of cores it is possible that the number of task for this assignment is smaller than the number of cores in the set of core divided by the minimal number of cores per task. In this case the number of cores per tasks is increased for these tasks. This is done as balanced as possible. As the parallel performance of a quantum chemistry code has its limitations and might become even slower with increasing number of cores after a given threshold, also a maximal number of cores per task is specified.

Workers (threads or processes) are started for every task. They all run on the master machine, thus the machine from which the executables of the quantum chemistry program are started, and are mapped to the assigned cores. A lock, implemented by a special Python Manager construct protects the access to the core reservation states collected in a Python dictionary. The dictionary contains the number of all cores on which currently calculations are running. A worker blocks until all cores assigned to the tasks are free, afterwards has to add them to the reservation list and remove them upon task completion. The order in which the workers are processing is arbitrary as it depends on the order in which they get access to the dictionary. Availability of new cores is signaled by an event to workers waiting for a reservation.

Both groups of functions for distributing cores over various instances of quantum chemistry codes can work together with the strategy of the first subtask of separately scheduling each gradient calculation. For a fixed set of cores assigned to the complete procedure the first group of functions can only be used if the operating system provides a routine to select a subset of empty cores for the current task.

20. Performance Tests

20.1 Goal of the Surface Reactions Study

The systematic study[13] on surface reactions was done to compare the performance of several methods described in the Chapters 13 to 16 for finding a transition state. These methods use a two-step strategy: the first step is a search for an approximate minimal energy path; the second step is a local refinement of a transition state estimate that resulted in the first step.

Simple test potentials as the Lennard-Jones potentials,[155] already mentioned in Subsection 18.5.2, are widely used for testing the performances of transition state searching methods. Systematic studies comparing transition state search methods, like the ones in ParaTools, used only simple analytical potentials[156] or *ab initio* potentials for gas phase reactions.[118, 120, 157] However, the closer the test system to the real life the more relevant are the results. Methods of global and local transition state search are often used by quantum chemists to study surface reactions. For rough transition state estimates chemists apply only path searching methods. The NEB method[98-99, 158] seems to be preferred over string methods.[101] The climbing image variant, especially of NEB, is also popular, especially if the transition state should be known more accurate.[102-104, 159-161] The dimer method is used without previous path optimization[106-107] but also as refinement for the CI-NEB[108-109] and NEB method.[110-113] There are two systematic studies on surface reactions to mention: one by Sheppard et al.[142] and a second one by Klimeš et al.[162]

Sheppard et al. studied diffusion of Pd_4 clusters on a MgO(100) surface and oxygen reactions on the Au(111) surface.[142] In this study Sheppard et al.[142] tested the climbing image variants of NEB and string methods and compared several optimizers for these methods. They used the same definition for the tangents for both methods, which leads to the small differences only between the methods. Sheppard et al.[142] did not see significant differences in performance between the two chain-of-state methods and concluded that the optimization procedure is much more important than the chain-of-state method. In ParaTools, the tangents for the NEB and string methods are determined differently, resulting in larger differences.

The authors of the second study chose diffusion of water and bond breaking of HCl on a NaCl surface as test examples.[162] They compared several methods for finding transition states, including the CI-NEB method and the dimer method. The dimer method required much more gradient evaluations to converge than the CI-NEB method and even failed several times to converge to the correct transition state for the flat potential of water diffusion. For the dissociation of HCl the results of both methods were comparable. From this result the authors concluded that the system and, in particular, the height of the reaction barrier is important for the performance of the methods.

The studies of Sheppard et al.[142] and Klimeš et al.[162] usually used only one type of methods at a time, either path searching methods or local refinement methods. Each type of methods has its advantages and disadvantages. For the user one of the largest advantages of path searching methods is that the methods require only simple input. Usually the two minima of the reaction path are enough. On the other hand, the path searching methods alone might experience convergence problems as the convergence usually is slow. In general only rough convergence is provided.[105] Although image optimization proposes a new path at each step, it does not consider that the images might get respaced. Only the climbing image variants can approach the transition state directly, while the other methods have to interpolate the transition state from the path. The interpolated transition state estimate is typically less accurate than the one achieved by climbing image methods.

Local transition state finding methods, like the dimer methods, have the advantage that they can converge to high accuracy and require less gradient evaluations than the path searching methods. On the other hand, they strongly depend on the quality of the starting geometry. As local methods do not keep any information about the minima on the potential energy surface, the methods can easily approach a transition state (or stationary point of higher order), which does not belong to the reaction of interest. The dimer method is created in a way that it can deal with convex regions of the potential energy surface,[131, 137, 163] allowing a start from a geometry close to a minimum. Especially if there are several transition states close to the minimum, it is possible that a transition state other than the desired one is found. Thus it is often required to generate a starting geometry, which fits the reaction of interest. This in turn needs great chemical intuition from the user.

A two-step strategy, as implemented in the present work, first launches a global, then a local transition state search method. In the first step a minimal energy path search method is used to get a rough estimate of the reaction path. In the second step the approximate transition state suggested from the converged path is refined by a local search method. Such an approach combines the advantages of both types of methods. Also ParaTools was opted for surface reactions involving larger adsorbed species than those from the literature.

20.2 The Test Strategy

A systematic study of the two-step strategy was done with ParaTools.[164] Energies and nuclear displacement gradients of the energy for ParaTools were obtained by the Vienna *ab initio* simulation package, VASP 4.6.[165-166] A density functional approach with plane-wave basis set at the gradient-corrected level was applied.[167] The PAW method was used to describe the core electrons.[168] The energy cut-off of VASP calculations was set to 320 eV. The methods and parameters chosen for the test of the two-step strategy of transition state optimization are described below.

For the path searching step (the first step of the two-step strategy) the study concentrated on the chain-of-state methods NEB, standard string, searching string, and CI-string methods, which were applied with two optimizers, conjugate gradient and Multiopt.[13] Other specifications for this step, like the choice of parameters, were not tested; the default parameters were used instead, see Appendix H.

For the path searching methods, two convergence criteria were applied. The first one is the gradients criterion of the images, the second one uses also the step length in addition to the gradients criterion. The first criterion employs the gradient convergence measure Δg , which is defined as root mean square value of the gradients perpendicular to the tangents in the mixed units of eV/Å or eV/rad if angles were concerned. For convergence this value should be below 0.1.

The second criterion uses a convergence measure for geometry steps in addition to some less tight gradient convergence conditions. The maximal step length of the three images with the highest energies were monitored. If for the three last iterations the maximum of this step length is below the convergence criterion of 0.03 (in units of Å or rad) and additionally the gradient convergence measure is at most $\Delta g = 0.5$, convergence is considered to be reached. This should prevent too long calculation with very small updates of the chain. These small updates hardly change anything but increase the amount of gradient evaluations. If no convergence is reached the algorithm stops after 35 iterations.

In this study, the path was represented by 7 images. All methods studied, except the searching string method, initially construct 5 images in between two given local minima, which are later optimized. The searching string method initially constructs 2 images only and increases the number of images during optimization. It adds a new image after the gradient convergence measure is less than $5 * \Delta g$, which per default case is equal to 0.5.

The CI-string method includes up to 5 initial iterations carried out with a standard string method, for which the gradient convergence criterion of $\Delta g = 0.5$ is applied. For the adding of a new image in the searching string method and for the first 5 iterations in the CI-string method no geometry convergence criteria were used.

To apply the second step of the strategy a transition state is estimated from the optimized path. For this estimate only a subset of all available paths was taken. Only those paths were used, which are chemically reasonable. Paths with molecular bonds broken or built, when it was not expected, or paths with other significant anomalies, were not considered and are omitted in tables. To note, it was not required that the path optimization reaches convergence. Even if the path did not meet the convergence criteria after 35 iterations, it was evaluated. To extract the best transition state estimate from the paths, various strategies were used for the CI-string approach and the other chain-of-state methods. The climbing image of the CI-string approach is expected to converge to the transition state, thus it is used as starting point for the next refinement step. For the other three methods the “spline and polynomial” method[118] was applied to generate a transition state estimate, which uses a cubic spline to interpolate the energy and to find local maxima. It can happen that there several maxima, for example, when the path exhibits loops. In this case the maximum closest to the expected transition state was chosen. This is a situation where chemical intuition is indispensable. The dimer methods, used for the transition state refinement, need a mode vector as input. The path tangent at the position of the transition state estimate was taken as a starting mode vector.

For the refinement (the second) step two different method, the standard dimer (Section 14.2) and the modified Dimer/Lanczos (mDL) methods (Section 14.3) were explored. For the choice of the computational parameters, see Appendix H. A simple quasi-Newton approach (Section 14.1) for the transition state search, which was applied selectively, was not successful. The refinement methods were restricted to at most 150 iterations. As convergence criterion of the refinement step, the absolute component of the gradients were required to 0.02 eV/Å or eV/rad, depending on the parameter considered. With this rough convergence criterion the results are still approximates, which cannot give valid results about the observed reactions, but the results are accurate enough to represent the behavior of this kind of method in real applications.

The same parameters were chosen for both local refinement methods. The distance between the dimer endpoints and its midpoint was set to 0.0025 Å, see Section 14.2. This corresponds to the default distance of the dimer method of Vasp TST tools.[118, 143] For the mDL method the same distance between the new basis trial point and the midpoint was chosen. Both methods were allowed to do at most 10 rotation steps before a translation step was done. The convergence of the mode updating step, as judged by the angles as specified in Sections 14.2 and 14.3, had to be below 0.01°.

20.3 The Test Systems

For the systematic study of transition states of surface reactions eight model systems were used, representing reactions on the Pt(111) surface. The model systems were created in the following way. Various typical reactions of different complexity were taken from recent investigations[110-112] and simplified: the slab model of Pt(111) for these considerations consisted of three layers, two of which were kept fixed. This is a simpler model than the one commonly applied which exhibits five layers in total, three of them fixed.[110-112] Alkyl groups, which are not taking part in the reactions, were replaced by hydrogens, thus also the adsorbed species were simplified. A unit cell of (3 × 3) was found to be sufficient. A vacuum space of at least 10 Å was introduced between the slabs. All systems considered are given in Table 20.1, together with a short description. Simplified in this way, minima and transition states were optimized for all reactions to serve as starting structures and reference states for this study. The structures of the transition states are sketched in Figure 20.1.

Table 20.1: For evaluation of the combinations of transition state search methods eight reactions on Pt(111) were used. Provided is the numbers *N* of atoms moving during transition state search, the barriers ΔE_1 and ΔE_2 (kJ/mol) from the reactant and product sides defined by the reference transition structure, respectively, as well as references to recent computational studies of related reactions. Adapted from Ref. 13.

Nr.	Short description	Reaction	N	ΔE_1	ΔE_2	Ref.
I	C-H bond creation	$CH_2C + H \rightarrow CH_2CH$	14	55	69	110
II	H-shift	$CH_2CH \rightarrow CH_3C$	14	169	218	110
III	C-C bond breaking	$CH_2CO \rightarrow CH_2 + CO$	14	106	141	[169]
IV	O-H bond breaking	$CH_3OH \rightarrow CH_3O + H$	15	57	15	112
V	C-H bond breaking	$CH_3OH \rightarrow CH_2OH + H$	15	30	90	112
VI	Ring opening	$\overparen{CH_2CH_2CH_2CC}CH_3$	24	118	191	111
VII	C-O scission	$CH_2OH + H \rightarrow CH_3 + OH$	15	196	151	[170]
VIII	OH substitution	$CH_3OH + H \rightarrow CH_4 + OH$	16	221	191	[170]

Recent investigations on ethylene transformations[110, 113] inspired the first two reactions (I, II). In System I a carbon-hydrogen bond is created, $CH_2C + H \rightarrow CH_2CH$.[110] A completely different step, which can also take place during ethylene transformation, is a hydrogen shift reaction ($CH_2CH \rightarrow CH_3C$) — system II of the tests.[110] Carbon-carbon bond breaking was taken as system III — the ketene example ($CH_2CO \rightarrow CH_2 + CO$).[169] System IV describes the scission of the O-H bond of methanol ($CH_3OH \rightarrow CH_3O + H$), for which the oxygen is not bound to the surface in the reactant methanol.[112] These four systems cover already a large variety of reactions and were extensively studied.

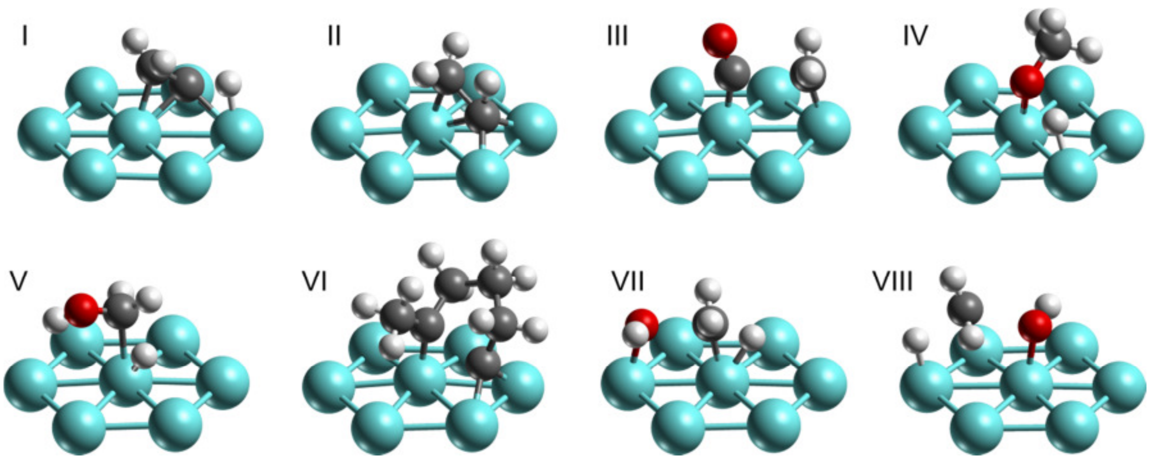

Figure 20.1: Sketches of the transition states of various model reactions. See description in Table 20.1. Adapted from Ref. 13.

Four other surface reactions, V–VIII, were used for further tests. They were only tested for a reduced set of methods. The reaction V, breaking of a C-H bond of methanol ($CH_3OH \rightarrow CH_2OH + H$)[112], should not cause any additional difficulties compared to reaction I. System VI is significantly larger than the other systems, including altogether 33 moving atoms (24 for the adsorbate and 9 for the uppermost surface layer). It corresponds to a ring opening of cyclopentyne.[111] So far only simple reactions have been tested, where only one significant structure change was taking place. To test the methods further, more complex reactions, S_N2, were examined. S_N2 reactions combine two significant structure changes in one step. Both reactions have a C-H bond formation and at the same time a C-O bond breaking. In reaction VII ($CH_2OH + H \rightarrow CH_3 + OH$) reactants and products bind strongly to the surface. In reaction VIII ($CH_3OH + H \rightarrow CH_4 + OH$) methanol is only weakly bound to the surface, while methane is not bound at all.

20.4 The Path Optimization Step

The goal of the study was to find a method, which leads to a reliable transition state at rather low computational cost. When the path was very peculiar, it is not taken for the further evaluation. For example, three paths found were removed from further considerations as they correspond to unphysical behavior of the system. In two cases the hydrogen atom in the dehydrogenation reaction was found to be absorbed by the substrate. It was located between the first and second layers of the substrate. One more path was skipped because in addition to the dehydrogenation reaction C-O bond breaking was observed. However, it is difficult to access the reliability of a path and seven more estimated transition states of the path optimization step finally did not converge to the expected transition state. This will be discussed in detail in Section 20.8. Experience shows that it is not obvious beforehand if the subsequent refinement step would reach the correct transition state or not.[118] The present section is focused on time efficiency and the computational cost.

For all systems a special mixed coordinate set was assigned. In mixed coordinates the adsorbate is described with a Z-Matrix which is connected to the surface by positioning coordinates. These are a global translation vector and a global rotation vector, described by a quaternion.[171] The atoms representing the surface were described in Cartesian coordinates. The advantage of this mixed coordinate system is that the Z-Matrix coordinates describe chemical degrees of freedom, like bonds and angles. At the same time the surface profits from being described in Cartesian coordinates, so one does not have to define the Z-matrix for it, which would be rather intricate. It is also possible to describe the complete system in Cartesian coordinates only, which has been done as alternative.

Table 20.2: Number of gradient evaluations required for the path optimization step. Tested were various chain-of-state methods: NEB, string (S), searching string (SS), and CI-string (CIS). They were combined with the conjugate-gradient (CG) or the Multiopt (MUL) path optimization procedures. The test systems I–IV were treated in either Cartesian (C) or mixed (M) coordinates. Calculations that did not converge within 35 iterations of the optimizer are marked by a star. Omissions indicate cases where no valid path was obtained. Adapted from Ref. 13.

Combination	I-C	I-M	II-C	II-M	III-C	III-M	IV-C	IV-M
NEB	127	–	77	97	87	37	52	82
S-MUL	77	47	117	77	87	52	77	52
S-CG	352*	–	347*	82	87	37	62	87
SS-MUL	100*	94*	70	82	68	17	82	29
SS-CG	133	124	31	183	43	21	–	83
CIS-MUL	117	87	102	67	57	52	67	62
CIS-CG	97	207	87	82	87	37	47	87

The first part of the test explored the reaction path optimization step (the first step) of the two-step strategy of transition state optimization and was restricted to the four reactions I–IV (see Table 20.1).

These two types of coordinate sets were compared in the current test. Also two optimizers were compared: the optimizers Multiopt (Section 15.1) and conjugate gradient approach (Section 15.2) are considered. Note that Multiopt was only designed for string methods and thus the combination NEB with Multiopt is not available. In addition, the four chain-of-state methods — NEB, standard string, searching string, and climbing image string — were compared. All tests were done for the four reactions I–IV, see Table 20.1. By varying the coordinate type, optimizer, and reaction path search method, the optimal approach of the two-step strategy of the transition state search was searched for.

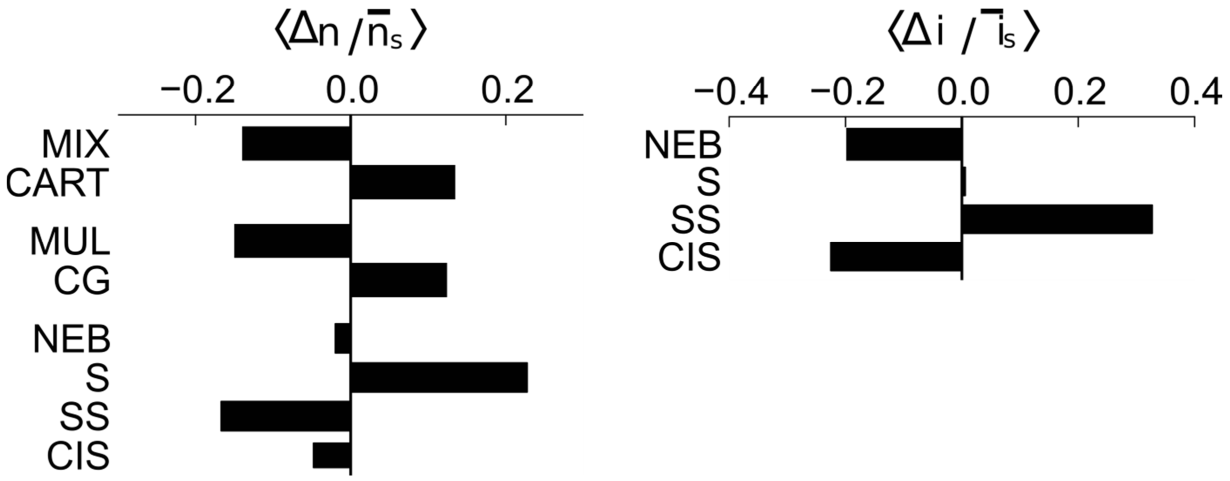

Figure 20.2: The computational efficiency of various path searching methods was compared based on relative differences $\langle \Delta n / \bar{n}_s \rangle$ of the number of gradient calls n with respect to the average $\bar{n}_s$ over systems s ($\Delta n = n - \bar{n}_s$) for systems $s =$ I–IV. Lower is better. Compared are pure Cartesians (CART) and mixed coordinates (MIX), two optimization procedures, Multiopt (MUL) and conjugate gradient (CG) and of four chain-of-state methods: NEB, standard string (S), searching string (SS), and CI-string (CIS). Further shown are the analogous category averages $\langle \Delta i / \bar{i}_s \rangle$ of the relative differences of the number of chains to be calculated from the system specific average $\bar{i}_s$ for the four chain-of-state methods. Adapted from reference 13.

To estimate the efficiency of the path search step the computational cost was evaluated. The most demanding part of the computational cost is the calculation of the gradients by the quantum chemistry code. The remaining cost can be neglected, leaving the number of gradient evaluations as a sound choice for an efficiency criterion. The number of gradient evaluations as the indicator of the method efficiency for all test systems is given in Table 20.2. As one can see these values cover a wide range from 17 to 352. The number depends on the complexity of the system. For example, system I required more steps on average, 130, than system III, 55.

One can actually use the average number of steps to characterize the complexity of the system. However, the average number of gradient calls does not take into account the failures, when no approximate transition state was produced. Thus, instead of using direct averages, it is more appropriate to look at relative differences

$$\Delta n / \overline{n}_s = (n - \overline{n}_s) / \overline{n}_s \qquad (20.1)$$

where $\overline{n}_s$ is the average for systems $s =$ I–IV and n is the number of gradient evaluations for a specific approach. For all methodological combinations the average ratio $\langle \Delta n / \overline{n}_s \rangle$ can be computed and compared. Note that the sum of the average ratios for all methods is not always 0. This is related to the fact that the number of values used for the average ratio can differ for each method. The average ratios are biased as failed calculations are not included, but using the relative differences as indicators should reduce this effect compared to using directly the number of gradient evaluations.

Comparison of Cartesian and mixed coordinates indicates that the average ratios over the relative number of gradients clearly favor the mixed coordinates with $\langle \Delta n / \overline{n}_s \rangle = -0.14$ against the Cartesian coordinates with $\langle \Delta n / \overline{n}_s \rangle = 0.13$ (Figure 20.2). It is also possible to compare directly pairs of calculations, corresponding to the same system, path searching method and optimizer, which differ only in the choice of coordinates. Omitting the pairs with a failed counterpart, there are 18 cases were the mixed coordinates required less gradient evaluations and only 7 cases were the Cartesian coordinates were favorable. Thus, mixed coordinates are clearly favorable for the path searching step.

The optimizer Multiopt also reveals a better performance than its conjugate gradient counterpart. The relative performances show $\langle \Delta n / \overline{n}_s \rangle = -0.15$ against $\langle \Delta n / \overline{n}_s \rangle = 0.12$ (Figure 20.2). For this category it is only possible to compare 22 pairs as there are no direct counterparts for the NEB calculations with conjugate gradient. For 13 of these pairs Multiopt requires less gradient evaluations, while conjugate gradient is more efficient for the remaining 8 ones. The main reason for the preference of Multiopt is that it requires only one gradient evaluation for every image per iteration, while conjugate gradient requires a second one to approximate the curvature in the search direction.

When the chain-of-state methods NEB, standard string, searching string and CI-string are compared to each other via the relative number of gradient calls they yield results of $\langle \Delta n / \overline{n}_s \rangle = -0.02$, 0.23, -0.17 and -0.05, respectively. The searching string method is the best method according to this criterion. However, one has to consider that the bad performance of the string method is mainly due to two calculations which failed to converge within the maximal number of iterations. They contributed more than 300 gradient evaluations each. On the other hand, the searching string method, the best method according to the relative performance criterion, includes also two calculations with failed convergence, but they contribute only around 100 gradient evaluations. The searching string method is favored in this performance measurement by its restriction on maximal three images to be moved during the optimization.[118] This becomes especially clear if another measurement of the performance is taken: instead of counting the number of gradient evaluations one counts how often a new path is created. As all image geometries of the chain are known at the same time, it is possible to parallelize over them. Thus, counting the number of chains *i* is a criterion which might be closer to the real time estimate on a parallel machine. The comparison is again done for a relative measure $\langle \Delta i / \overline{i_s} \rangle$, calculated in the same way as for the gradient evaluations, but this time for the number of chains *i*. This criterion exhibits a different trend for the performance of the methods: the CI-string method wins with $\langle \Delta i / \overline{i_s} \rangle = -0.23$, followed closely by NEB with $\langle \Delta i / \overline{i_s} \rangle = -0.20$. The searching string method, which had strongly profited by its reduced number of gradient evaluations per chain, performs worst with $\langle \Delta i / \overline{i_s} \rangle = 0.33$ and the string method is much better than this with $\langle \Delta i / \overline{i_s} \rangle = 0.01$.

Another aspect to mention is that one of the three failed calculations belongs to the searching string method, while the CI-string method did not yet show any failures (Table 20.2). Thus, the CI-string method is the best method concerning time and stability aspects, while the searching string method is the best concerning the computer resources.

Table 20.3: Gradient convergence measure Δg (see Section 20.2 for the definition) of reaction paths for the method combinations and systems as defined in Table 20.2. Calculations that did not converge within 35 iterations of the optimizer are marked by a star.

Comb.	I-C	I-M	II-C	II-M	III-C	III-M	IV-C	IV-M
NEB	0.10	0.48	0.10	0.11	0.20	0.41	0.18	0.25
S-MUL	0.09	0.12	0.17	0.13	0.28	0.14	0.09	0.10
S-CG	0.12*	0.40	0.69*	0.26	0.26	0.42	0.22	0.17
SS-MUL	0.35*	0.41*	0.42	0.09	0.20	0.18	0.31	0.24
SS-CG	0.07	0.09	0.22	0.24	0.26	0.21	4.74*	0.13
CIS-MUL	0.08	0.10	0.09	0.10	0.21	0.21	0.09	0.08
CIS-CG	0.14	0.21	0.11	0.26	0.26	0.42	0.25	0.22

The transition state estimates obtained by the various methods can be compared against the reference structures. Additionally the convergence of the paths can be compared. This does not allow evaluating the overall quality of the paths but can be used to distinguish different groups within the current results. Furthermore it allows checking whether a path which is not converged might be better omitted from further evaluations. All calculations provided transition state estimates still far off the reference transition states. Several Cartesian components were more than 1 Å away from their values for the reference geometry. Also energy differences between reference system and its estimate are around 30 kJ/mol.

The convergence of the paths was determined with two stopping criteria, one using solely the gradients, which should vanish at a converged path, and one considering the changes between successive paths. The first criterion, using solely the gradient convergence measure Δg (Section 20.2), is a criterion based on the current path, while the second one also depends on the history of the paths. The first criterion is more relevant concerning the quality of the paths, which are the results of optimizations. Table 20.3 shows the value for the convergence criterion Δg of the gradients for the last path. Several of the calculations had stopped because of the second criterion having usually still rather high values of the first measure. The not converged paths, except for the failed one of system IV with the searching string method, showed comparable values, see Table 20.3. This seconds the fact that the not converged calculations need not be a failure. The only reason why they were not converged is that successive paths during the optimization differed too much from each other, so that the procedure did not stop according to the second criteria.

20.5 The Refinement Step

Three methods were explored in the refinement step of the transition state optimization, the second step of the two-step strategy: the standard dimer, the modified Dimer/Lanczos (mDL) and the quasi-Newton method. The simple quasi-Newton method (Section 14.1) failed to find the reference transition state from various start positions while the other two methods were successful. Therefore only two methods were compared in the present section: the standard dimer and the mDL method.

Both refinement methods follow the same scheme, with small differences, see Sections 14.2 and 14.3. Although it would be possible to compare the standard dimer and the mDL methods for each starting structures generated in Section 20.4, it should be sufficient to compare only a subset of all possible structures in order to look for the preferred one of the dimer variants: all path searching methods seem to provide results of comparable quality, see Table 20.3. For this reason only transition state estimates created from the string calculations — except the one which failed for the path of system I — were used as starting positions for the comparison of the two methods. This yields 15 transition state estimates, spread over the four systems I–IV.

The transition state approximation from the path was done as described in Section 20.2. In one case, for system II with conjugate gradient optimizer in Cartesian geometries, the transition state estimate of the "spline and cubic" method with the highest approximate energy belonged to a loop, see Section 20.2. For this case the other transition state estimate of the "spline and cubic" estimation method with smaller approximate energy was taken as starting geometry for the local transition state search.

For 12 cases both methods converged to a structure which could be identified as the reference transition state. The three exceptions are related to system II. For system II there was a stationary point of higher order (two negative eigenvalues) on the potential energy surface geometrically close to the reference transition state. To this particular stationary point the calculations in Cartesian coordinates were converged: for the dimer approach and for mDL method with the Multiopt optimizer. The conjugate gradient optimizer with mixed coordinates produced a starting geometry from which the mDL method converged to a transition state of a different reaction. The two examples where the two methods found different stationary points are excluded from the subsequent discussion. However, the example, where the same undesired transition state by both dimer and mDL methods was found, was considered further. Thus, as a result, there were 13 cases to compare dimer and mDL methods. The methods failed only in two cases to find the reference transition state, thus their overall reliability is comparable.

For 9 of these pairs the mDL method required less gradient evaluations than the standard dimer method. Having a valid pair for every combination, one can directly average the number of the gradient evaluations. This resulted in 191 gradient evaluations on average for the standard dimer method and 150 for the mDL method. Consider that several gradient evaluations per iteration are possible. This was clear from the fact that on average the standard dimer and the mDL method required only 39 and 35 iterations, respectively. This is roughly one iteration per degree of freedom. Note, that the standard dimer method requires two gradient evaluations per translation step, while the mDL method needs only one. Thus, the standard dimer method required in fact 77 gradient evaluations ($2 \cdot 39 - 1$, because after convergence was detected it was not required to calculate a second geometry) on average for dimer translation, while the mDL method needs only 35. This is also the main reason why the mDL method required less gradient evaluations in total (150 vs. 191). For dimer rotations the mDL method required 115 gradient evaluations on average (150–35), while the standard dimer method requires 114 on average (191-77).

In the cases where both methods approached the reference transition state the maximal differences in the geometries to the transition state for the standard dimer method were about 0.06, 0.003, 0.05 and 0.11 Å in the Cartesian L_2 norm, for the systems I to IV, respectively. The total energies were found to deviate at most by 0.09, 0.33, 0.08 and 0.05 kJ/mol, respectively, from the reference energy. The results for the mDL method were comparable, with deviations of 0.04, 0.02, 0.04 and 0.34 Å for the geometries and 0.09, 0.3, 0.06 and 1.3 kJ/mol for the energies, respectively.

Altogether the mDL method performed better than the standard dimer method. The results were of comparable quality and the mDL method required less gradient evaluations. Still it might be possible to improve the mDL method further.

20.6 Suggestions for Improving the mDL Method

The mDL method might perform not that well because all the computational parameters were taken from the standard dimer method and were not specially adapted. As already mentioned in Section 20.5, it required on average nearly the same amount of gradient evaluations for rotation as the standard dimer method. It required less iterations (35), therefore the number of gradient evaluations per iteration was somewhat higher than for the standard dimer method. The hints, which suggest that this difference in performance might be related to the computational parameter as well as the size of the effect will next be examined for an example.

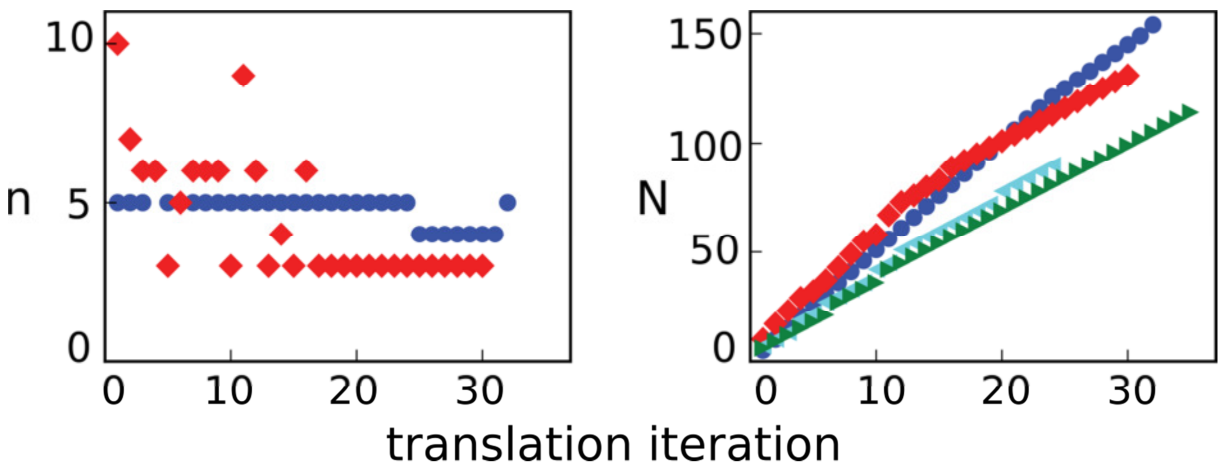

Figure 20.3: Using one of the starting geometries of system III as example the dimer (blue circles) and modified dimer-Lanczos (red diamonds) methods are compared. Left: number of gradient calls n in each iteration. Right: cumulative count N of gradient calls. The cumulative count N is also shown for the modified dimer-Lanczos method with a relaxed mode convergence threshold of 0.02° (cyan triangles, pointing left) and 0.03° (green triangles, pointing right), instead of 0.01°. Adapted from Ref. 13.

For this study, system II in mixed coordinates was chosen, where the string method and the conjugate gradient optimizer have been applied to find the transition state estimate. For this example the standard dimer and the mDL method required a comparable amount of iterations, 32 and 30, respectively. The mDL method needed less gradient evaluations, 132, than the standard dimer method, 155. Figure 20.3 shows how many gradient evaluations were used in which iteration and how the sum of gradient evaluations developed over the iterations for both, the standard dimer and the mDL method. The standard dimer method used nearly uniformly 5 gradient evaluations per iterations and 4 near the end. This corresponds to 2 to 3 gradient evaluations per rotation and thus normally to only one rotation step per rotation phase. On the other hand, the mDL method requires especially at the beginning more gradient evaluations per iteration, up to 10. Only in the second half of the optimization the mDL method becomes more stable and uses only three gradient evaluations, corresponding to a single rotation with 2 gradient evaluations per translation step (Figure 20.3).

The curve of the sum of gradient evaluations N for every iteration of the mDL method starts much steeper (Figure 20.3) than the one corresponding to the standard dimer method. The sum of gradient evaluations N of the mDL method became smaller than that of the standard dimer method only after iteration 20, see Figure 20.3. It is not obvious why the Lanczos algorithm needs more iterations than the standard dimer rotation to determine an appropriate mode vector. One reason might be that the Lanczos algorithm works usually on more degrees of freedom. Thus even for identical values of the convergence criteria for both methods, the Lanczos algorithm may be searching for something which is globally more converged. Hence, the mDL method likely takes too many gradient evaluations on searching for the mode; a somewhat looser convergence threshold might improve the performance of the method.

To explore the possibilities of this suggestion for the test case, see Figure 20.3, two further calculations with the mDL method were carried out, using less accurate convergence thresholds of 0.02° and 0.03° instead of 0.01°. The resulting sum of gradient evaluations can be seen in the right panel of Figure 20.3. The curves belonging to the less accurate convergence criteria are less steep than the original curve. They reflect less gradient evaluations, 91 and 115 versus 132 for the calculation with original convergence criteria. However, the results show also that the least accurate threshold is not always superior, as the calculation with 0.03° used 35 iterations, even slightly more than the standard dimer method. For this specific example calculation the convergence threshold of 0.02° was the best, it needed 91 gradient evaluations for 24 iterations. To determine the best parameter a more extensive study is required, because with the less accurate convergence criteria an inaccurate mode vector can be obtained which prevents convergence towards the desired transition state. Thus, for further test calculations the initial rotation threshold of 0.01° was kept. However, the possibility of decreasing the number of gradient evaluations by about 30% by changing the rotation threshold suggests that the search for improved computational parameter, which are reasonable in all kind of calculations, may further improve the refinement method

20.7 The Global Evaluation

For the global performance analysis of the two-step strategy all transition state estimates of the path optimization step, see Section 20.4, were followed by a refinement method. For the analysis the same criteria for the same set of test systems were used. A similar strategy as in Section 20.4 for the path searching step was employed for the global evaluation. Instead of comparing various refinement methods the better of the two, the mDL method (see Section 20.5) was taken for all calculations. In addition it was now possible to evaluate the failures of the method combination, as the second step should yield the correct transition state.

Among the calculations which failed to provide the correct transition state three types of failures can be observed. The first one was already addressed in Section 20.4 and corresponds to a completely failed path searching step. In Section 20.5 a stationary point of higher order was identified. The third kind of failure, the convergence to a wrong transition state of another reaction, is more severe and there are six examples of this kind. These failures are most certainly due to an unfortunate starting position of the refinement step. The failed cases will be analyzed further in Section 20.8. For the present performance evaluation only successful examples are discussed.

The successful transition states determinations, as obtained with the two-step strategy, did not deviate much from the reference transition states. There is a slight difference between the systems I–III and system IV, the latter being always less accurate. While for systems I–III the total energies deviated only 0.02–0.04 kJ/mol from the reference system, system IV approached them only by 0.47 kJ/mol on average. The maximum absolute deviations in Cartesian coordinates were 0.04–0.09 Å for the systems I–III and 0.6 Å for system IV. The differences for system IV are mostly due to positions of the hydrogens in the CH_3 group. Also the standard deviation from the reference data of the imaginary frequency at the transition state of 483 cm^{-1} was almost twice as large for system IV, 7.2 cm^{-1}, as the deviation for system I, 3.9 cm^{-1}, which happened to be the second largest.

Table 20.4 provides the sums of the gradient evaluations for both steps for all non-failed calculations. The numbers are significantly larger than for the path optimization step before. With an average requirement of 167 gradient evaluations for the mDL method step versus 88 gradient evaluations for the path searching step, the algorithm spends roughly twice as much computational resources in the refinement step than in the path searching step. Thus, it is very important to get a good starting position from the path searching algorithm, even though this might require a larger number of gradient evaluations.

Table 20.4: Total number of gradient calls when applying various two-step procedures to transition state search: a path optimization step followed by a refinement step with the mDL method for tests of systems I to IV. For the designations of the various methods see Table 20.2. Specially marked are failed path optimizations (omissions), higher-order stationary points ("ho") and convergence to an unexpected transition state ("u"). Adapted from Ref. 13.

Comb.	I-C	I-M	II-C	II-M	III-C	III-M	IV-C	IV-M
NEB	319	–	262	225	240	163	249	271
S-MUL	284	202	ho	214	236	166	286	217
S-CG	479	–	536	u	235	169	268	237
SS-MUL	314	287	269	255	159	145	u	247
SS-CG	u	279	237	292	186	139	–	284
CIS-MUL	283	214	289	220	u	152	335	237
CIS-CG	274	375	283	u	u	168	327	227

As in Section 20.4, the relative differences $\langle \Delta n / \bar{n}_s \rangle$ were calculated, instead of direct averages, see Figure 20.4. This time the relative differences were built over the cumulative number of gradient evaluations of the path searching step and the refinement step. Using the relative difference makes even more sense than for the example in Section 20.4. There are 10 cases, instead of three, which have to be omitted from the statistics because of the three different kinds of failures, instead of only one for failed paths. They are widely scattered for the various calculations.

The coordinates, Cartesian and mixed, show the same picture as for the evaluation of the path searching step (Section 20.4) with even the numbers having changed only slightly from $\langle \Delta n / \bar{n}_s \rangle = 0.13$ and $\langle \Delta n / \bar{n}_s \rangle = -0.14$ to $\langle \Delta n / \bar{n}_s \rangle = 0.12$ and $\langle \Delta n / \bar{n}_s \rangle = -0.11$, respectively. This is due to the fact that the mDL method can also use the advantages of the improved mixed coordinates. On the other hand, the mDL method cannot profit from the preferable optimizer for the path searching step. Thus, the performance difference between the various optimizers decreases. While for the path searching step Multiopt gave $\langle \Delta n / \bar{n}_s \rangle = -0.15$ and the conjugate gradient optimizer yielded $\langle \Delta n / \bar{n}_s \rangle = 0.12$, the differences for the complete number of gradient calculations is reduced to $\langle \Delta n / \bar{n}_s \rangle = -0.07$ and to $\langle \Delta n / \bar{n}_s \rangle = 0.06$ for the full optimization, respectively.

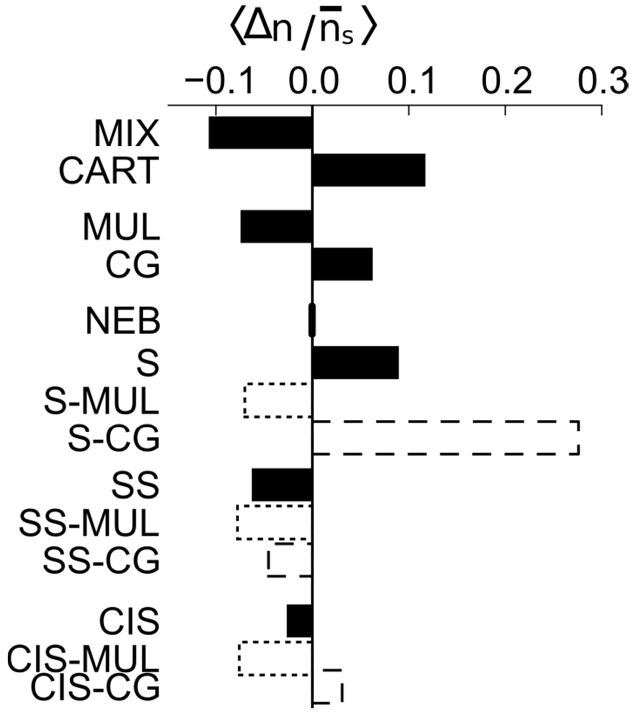

Figure 20.4: Category averages of relative differences $\langle \Delta n / \bar{n}_s \rangle$ for the two-step transition state search, path optimization plus modified dimer-Lanczos refinement. $\Delta n = n - \bar{n}_s$ is the difference of the number n of gradient evaluations for systems s = I–IV and the corresponding average $\bar{n}_s$. Lower is better. Comparison of mixed (MIX) and Cartesian (CART) coordinates, of the two optimization methods, Multiopt (MUL) and conjugate gradient (CG), of four chain-of-state methods NEB, standard string (S), searching string (SS), and CI-string (CIS). For each of the string methods three average values are shown: over all optimizations as well as over optimizations carried out with the path optimizers Multiopt (dotted line) or conjugate-gradient (dashed lines). Adapted from Ref. 13.

Another criterion indicates that the quality of all the transition state estimates is rather similar. For the different chain-of-state methods the performance differences also decreased when the gradient evaluations for the refinement step were added. This might be changed in cases where more effort is used to generate better paths, which can improve the transition state estimate to be refined. However, for the current switch point between path search methods and refinement a decrease is observable. For NEB, standard string, searching string, and CI-string methods the values are $\langle \Delta n / \bar{n}_s \rangle$ = 0.0, 0.09, –0.06, and –0.03, respectively. Especially the difference between searching string and CI-string methods is reduced largely, compare Figure 20.2. These differences are even further reduced if the averages are separated for each optimizer. It is clear that the reason for the relatively bad performance of the string method is due the weakness of the conjugate gradient optimizer (Figure 20.4). The CI-string method exhibits as good a performance as the searching string method, if only the calculations with the Multiopt optimizer are considered; for both methods $\langle \Delta n / \bar{n}_s \rangle = -0.08$ is calculated.

The string method reaches $\langle \Delta n / \bar{n}_s \rangle = -0.07$ if the Multiopt optimizer is used. However, the combination of the conjugate gradient optimizer and the NEB method for the transition state search still performs surprisingly well, compared to combinations of conjugate gradient with the string methods. That leads to the hypothesis that not the conjugate gradient optimizer itself, but the choice of tangents and the respacing procedures contribute to the (relatively) bad performance of this optimizer in the two-step strategy.

Based on this overall analysis it is recommend to use the more reliable optimizer Multiopt for the first step, the path searching step. As it was shown based on the number of gradient calls, the best method combinations are the searching string and CI-string methods together with the Multiopt optimizer in mixed coordinates. In addition the CI-string method has the advantage that more images of the path searching step can be calculated in parallel, see Section 20.4. Thus, the CI-string method (together with the Multiopt optimizer) is recommended as the best choice for the first step. As shown in Section 20.5, the mDL method is the best choice for the refinement step. As a result, the most reasonable choice of procedures for the transition state search with the two-step strategy are mixed coordinates, CI-string method, Multiopt optimizer and the mDL refinement method.

20.8 Cases of Unwanted Structures

There are three types of unwanted structures to consider:

1. The failed path calculations of the path evaluation step, which were already removed in Section 20.4 from the statistics.
2. The calculation stopped at a stationary point of higher order.
3. The calculation found an unwanted transition state.

In case of a failed path one should use another path evaluation method. As these cases are rather rare there is always a different choice possible.

In the case of a stationary point of higher order one may take a single step along the direction of the unwanted eigenmode and restart the refinement. This is explained in detail for the only case, where it appeared: for the stationary point found by the mDL method using the start geometry of the string calculation with the Multiopt optimizer in Cartesian coordinates. With a frequency/eigenmode analysis of the stationary point one finds two imaginary frequencies, 67 cm^{-1} and 1170 cm^{-1}. A step along the eigenvector of the Hessian matrix is taken which belongs to the second smallest eigenvalue (frequency 67i cm^{-1}). The length of this step is set to 0.1 Å. This provides a new geometry. This geometry and the eigenvector belonging to the 1170i cm^{-1} frequency as mode direction are the starting values for a second mDL calculation, which converged after 41 iterations and 153 gradient evaluations to the desired reference transition state. The first calculation had already stopped after 15 iterations and 56 gradient evaluations.

In the third kind of failures of the transition state search, where an unwanted transition state was found, the solution was to improve the path with continued optimization until a better starting geometry for the refinement step was identified. When the unwanted transition state was reached the transition state estimate of the path was not in a region where it converged to the real transition state. The reason might be that it was too far away or in an unfortunate region with the mode vector of the smallest eigenvalue belonging to another reaction. Thus it is desirable to improve further the path, hopefully to reach a better starting geometry.

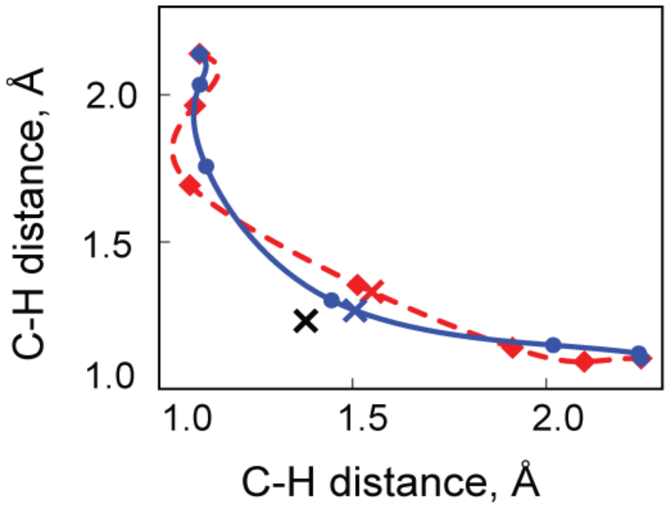

Figure 20.5: Geometry projection of two successive reaction path approximations for the hydrogen shift reaction (system II), converged with $\Delta g = 0.26$ (dashed line) and $\Delta g = 0.10$ (solid line). Structures were located by the modified dimer-Lanczos method when started from the TS estimates of these two paths. TS estimates are indicated by crosses. Estimates on the paths are in the corresponding color. The reference TS is marked in black, off any of the paths. Adapted from Ref. 13.

Using the example of the mixed coordinate calculation of system II, where the transition state estimate was generated with a string calculation using the conjugate gradient optimizer, the strategy is explained in detail. The original string calculation converged after 10 iterations and 82 gradient evaluations to a path with gradient convergence measure $\Delta g = 0.26$. From this path the mDL method converged to a wrong stationary point. The continued string method reached the smaller convergence criteria of $\Delta g = 0.10$ in iteration 18. The original converged path and this new path projected on the relevant coordinates are given in Figure 20.5. The new path has its transition state estimate closer to the reference transition state in the projection shown in Figure 20.5, but the geometry altogether is not that much closer. The geometries in Cartesian coordinates are even a bit further off with 0.26 Å instead of 0.22 Å average differences to the reference transition state. Still the transition state estimate from the improved path converged after 25 iterations and 130 gradient evaluations to the reference transition state.

The three similar cases for the CI-string method are treated the same way. The calculation with the CI-string method, mixed coordinates and the conjugate gradient optimizer for system II reached a value of $\Delta g = 0.26$; it improved in iteration 24 to $\Delta g = 0.11$. The subsequent mDL calculation needed 21 iterations and 110 gradient evaluations. The other two problematic calculations with the CI-string method appeared for system III, calculated in Cartesian coordinates. To resolve them the method, where the Multiopt optimizer had been used, was continued to the iteration 23; the convergence measure improved from $\Delta g = 0.20$ to $\Delta g = 0.09$. For the second calculation the conjugate gradient optimizer took 33 iterations to reach $\Delta g = 0.15$ instead of $\Delta g = 0.26$. The following mDL calculations needed 141 gradient evaluations for 35 iterations and 62 gradient evaluations for 15 iterations for the new starting geometry obtained from calculations with Multiopt and conjugate gradient optimizer, respectively. Both calculations converged to the reference transition state.

When applying the same strategy to the calculation with the searching string method, one has to consider that this method freezes some of its images without further processing. Extending the calculation with the purpose of getting better convergence only to the last moving images might not be enough, as the remaining part of the path is still rather approximate and might prevent the desired improvement. Thus in this case the complete calculation with the searching string method has to be redone with new convergence criteria. This is different from the other calculations where one could reuse the results from the path searching calculation. The gradient evaluations which were required for the refinement step of the transition state search which converged to the wrong stationary point are lost for all calculations and cannot be further used. However, this step was computationally more demanding. Thus the additional loss for the searching string method does not have too much weight. In case of the searching string approach one furthermore has to be careful that the new convergence criterion g_{tol} is not too tight, as it would prevent the chain to add new images because the convergence for getting a new image is set to $5*g_{tol}$ and hence coupled to the gradient convergence criteria g_{tol}, see Subdirectory 20.2. Taking half of the original convergence criteria the calculation with the searching string method with conjugate gradient optimizer in Cartesian coordinates for system I reached $\Delta g = 0.17$ after 18 iterations and 157 gradient evaluations, allowing the mDL method to converge to the reference transition state in 40 iterations and 175 gradient evaluations. The calculation with the searching string method in Cartesian coordinates with Multiopt optimizer for system IV, which had the same challenges, converged with the tighter convergence criteria to $\Delta g = 0.14$ after 13 iterations with 36 gradient evaluations, the mDL method required additionally 64 iterations and 243 gradient evaluations.

20.9 Further Tests

The reactions V and VI from Table 20.1 were only tested with the best methods from Section 20.7, searching string and CI-string methods. Only the superior optimizer Multiopt was used. Calculations were further exclusively performed in mixed coordinates, as the better choice of the coordinate systems found in Section 20.7. The refinement of the transition state estimate was done with the mDL algorithm. Thus two different strategies, one starting with the searching string method the other with the CI-string method and both continued with mDL method, were tested.

20.9.1 System V: Hydrogen Abstraction

For system V, hydrogen abstraction of methanol, see Table 20.1, both strategies converged. For the first step (path searching step) both methods needed 17 iterations to converge.

Relevant coordinates and energies for both strategies are given in Figure 20.6. As can be seen in Figure 20.6, the CI-string method starts with two energy maxima on the first interpolated path. The second maximum is reduced in further calculations, in the last iteration only one maximum remains at a geometry where the hydrogen is not yet very far from the carbon. The searching string method has more problems despite requiring the same amount of iterations. It also starts with two energy maxima on the first path. The resulting path is not as straight as the one from the climbing image method. The last path still exhibits the second energy maximum. Furthermore it is possible to see how a loop near the reactant develops with increasing iterations. Most of the images on the path are applied to represent this loop which provides also the highest energy maximum. As transition state estimate for the searching string method the second largest energy maximum, the one after the loop, is taken. For comparison it is noted that the images of the CI-string method were more or less equally distributed.

Both transition state estimates could be refined towards the reference transition state. With the starting geometry from the CI-string method the mDL method required for these 73 gradient evaluations, together with the 87 from the path search this strategy required 160 gradient evaluations in total. Surprisingly, the strategy with the searching string method required less gradient evaluations, 129, as the sum of 45 evaluations for the path searching step and 84 gradient evaluations for the refinement. This is due to the reduced number of gradient evaluations per chain in the path searching step.

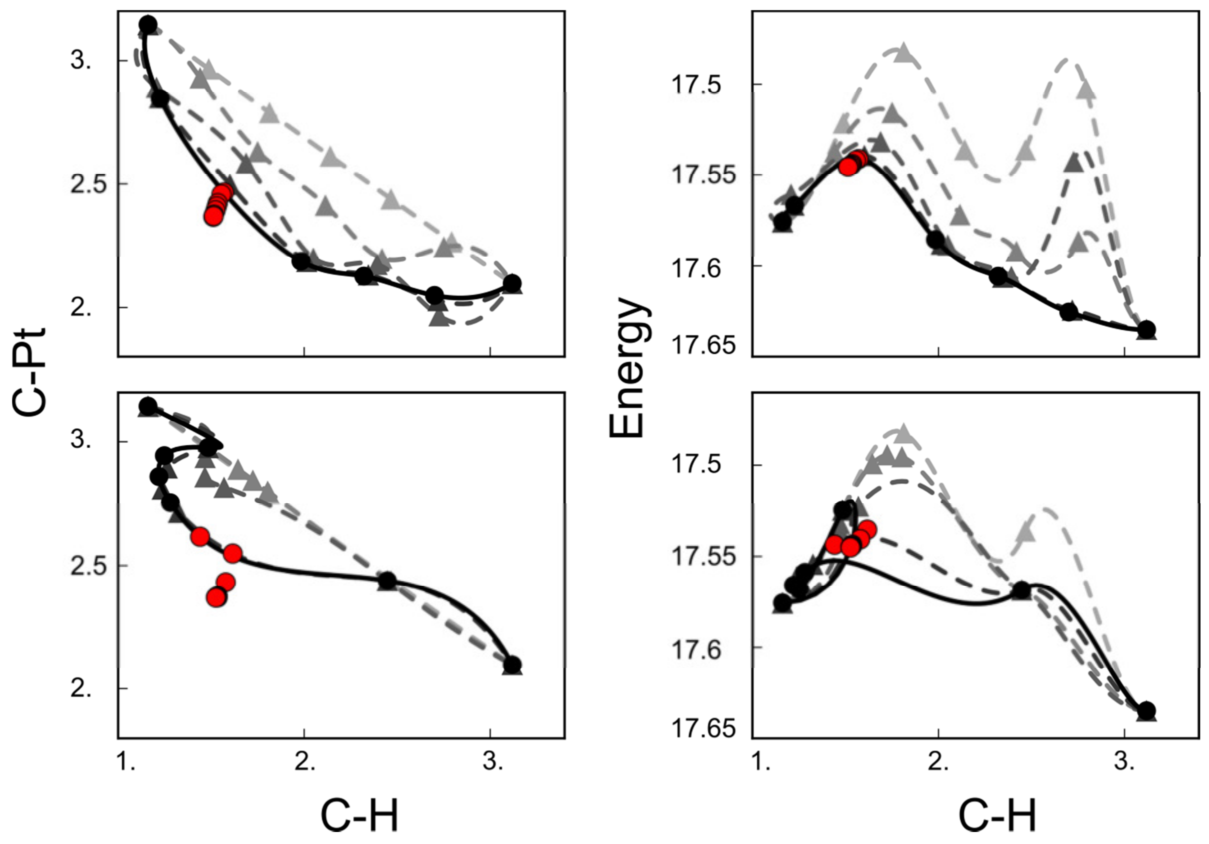

Figure 20.6: Locating the transition state of system V. Relevant coordinates (left) and energy (right). The first, fifth, 10th and 15th path are shown as dashed curves, the higher the number the darker the path. The last (17th) path is shown as solid curve. Every second geometry of the mDL calculation is given as colored circles. Top row: CI-string method as path searching method, bottom row: searching string method as path searching method.

20.9.2 System VI: Ring Opening

The requirements on gradient evaluations for the calculations of system VI, ring opening of cyclopentyne, were of the same order as for the systems tested already in Section 20.7. This is a fortunate result, considering that system VI was the largest system of the test set, having 99 degrees of freedom altogether. The strategy starting with the CI-string method required 193 gradient evaluations and was requiring even less than the strategy with the searching string method which needed 227. The searching string method had problems to converge, running the maximal 35 iterations and stopping with a gradient convergence criteria of $\Delta g = 0.60$. Still the quality of the transition state estimate was sufficient for the subsequent refinement. The refinement took 157 gradient evaluations which is only slightly more than the refinement after the CI-string method with 141 gradient evaluations. As several of the atoms of system VI were not taking part in the reaction, this example indicates that removing the alkyl groups to simplify the systems makes the results still comparable to the larger systems of the recent investigations.[110-112]

20.10 S_N2 Reactions

The last class of reactions to be discussed comprises S_N2 reactions where in one step a bond is broken and simultaneously a bond to a different group is built. This does not mean that the reaction which would in reality take place between the respective reactant and the product needs to be a S_N2 reaction. A so called S_N1 reaction is also possible where the reaction is done in two steps — one where one bond is broken and subsequently another where another bond is formed. However, also a S_N2 mechanism in the following examples. As for the other tests on systems V and VI, Section 20.9, only the searching string and the CI-string methods were used together with the optimizer Multiopt. The calculations were carried out in mixed coordinates.

The fact that there should be two paths between the reactant and product minima, one of a type S_N2, the other one of a type S_N1, can result in challenges for the path searching calculation. As initial path, a linear interpolation between the two minima is used. If a path nearby exists, it is more likely that the path searching method finds this path. One solution to this challenge is to provide a third image for a parabolic starting path to enforce the reaction to take the more complex S_N2 reaction. The creation of these images is described in detail in Appendix J. Alternatively, a path search method, especially adapted to this situation, like the growing string method, may provide a solution. Both variants were tested on systems VII and VIII.

20.10.1 System VII: Hydroxymethyl

In system VII all molecules remain attached to the surface during the reaction. A hydrogen atom approaches the CH_2 group of the hydroxymethyl from which OH is abstracted. Using only reactant and product structure as input to path searching, both the searching string method and the climbing image string method converged. The two resulting paths were near the linear interpolation, see for example the CI-string method in Figure 20.7. The structures of the transition state estimates had the following characteristics: The C-H distance was 1.7 Å for the searching string method and 1.9 Å for the CI-string method. The C-O distances, on the other hand, were even larger, 2.7 Å and 2.6 Å, respectively. These values belong more likely to a S_N1 reaction where first the C-O bond is broken, allowing the OH group to be removed, followed by a step where the C-H bond is formed. For a two-step reaction one would expect a second maximum on the path. However for a path represented by only few images, it is possible that this aspect is missing, especially if this corresponds to a small reaction energy.

If the transition state estimates of the paths are refined with the mDL method, they converged to different structures. The starting geometry from the searching string method converges to a stationary point of second order. The larger negative eigenvalue of the Hessian matrix in Cartesian coordinates (771 cm^{-1}) corresponds to a C-H bond formation, while the smaller second one (29 cm^{-1}) is mainly a rotation of the O-H bond around the oxygen, which was removed with the strategy mentioned in Section 20.8 for the case of a transition state of higher order. However, in this case (other than in Section 20.8) a larger step along the unwanted direction with the length of 0.5 Å is taken. The strategy yielded a S_N1 transition state. The CI-string transition state estimate on the other hand was refined to yet another structure, which was also much lower in energy, but belonged to a reaction step, where both the C-H bond and the O-H bond are already broken and only a rotation of the CH_2 species took place.

The transition state, found with the starting geometry of the searching string method, has an energy of -17519.5 kJ/mol, while the reference transition state of the S_N2 reaction has an energy of -17435.1 kJ/mol. Therefore it is to be expected that the path between reactant and product would usually follow a S_N1 reaction. To have a chance for locating the S_N2 pathway, a third starting point for a parabolic interpolation is generated, which should switch the initial path to a path much more favorable for a S_N2 reaction.

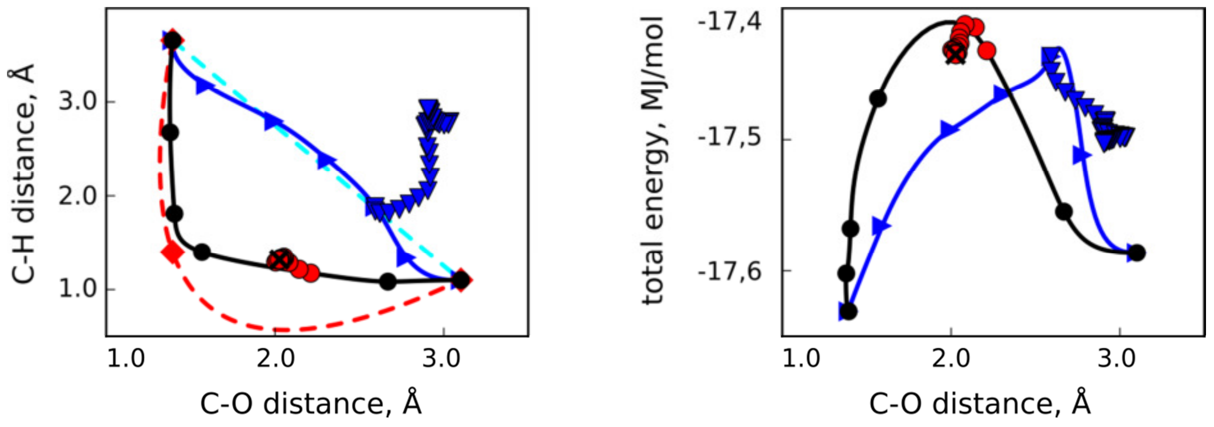

Figure 20.7: Path projections (left) and energy profiles (right-hand) of system VII. The paths were generated with the CI-string method. Shown are the starting path from a linear interpolation (cyan, dashed line) and the resulting converged path (blue, solid line with triangles) as well as the starting path with a parabolic interpolation (red, dashed line with diamonds) and the corresponding converged S_N2 reaction path (black, solid line with circles). The subsequent refinement (red dots or blue triangles) starts from the image of this path with the highest energy. Adapted from Ref. 13.

The parabolic starting path and the path to which it converged with the CI-string method are also shown in Figure 20.7. The parabolic starting path has the disadvantage that for example the C-H bond is getting very small (~ 0.6 Å) for some of the images. However the CI-string calculation managed to converge to a sound path using 97 gradient evaluations, see Figure 20.7. The mDL method required 207 gradient evaluations to reach the transition state of the S_N2 reaction. This transition state belongs to the desired S_N2 reaction as can be deduced from the distances C-H = 1.3 Å and O-H = 2.0 Å; these distances represent existing stretched bonds. The searching string method did not succeed to find a reasonable path; rather, it converged to a path where a water molecule is formed.

For this kind of reactions it makes sense to consider another method for creating a path for the first step (phase) of a two-step strategy; the growing string method was designed for finding a path when the linearly interpolated initial path is not adequate. The growing string method, using the same optimizer and coordinate system as the other two path-searching methods, converged towards the path of the S_N1 reaction. This again can be explained by the fact that the S_N1 reaction is likely to be lower in energy and that the corresponding path exists and is much closer to the starting path than the path of the S_N2 reaction. As the converged path of the growing string method is not likely to lead to the desired transition state of the S_N2 reaction, no refinement was carried out.

20.10.2 System VIII: Methanol

System VIII is even more complicated than system VII, as here the reactant methanol is only slightly bound and the product methane is not attached at all at the surface. The main difference to system VII is an additional hydrogen, which results in a CH_3 group instead of the CH_2 group of system VII. For the current example no strategy was particularly successful. When the correct transition state was found, it always required many gradient evaluations. There is also no path which can be seen to represent the reaction path especially well. Some of the difficulties can be ascribed to the fact that the moieties are only weakly bound to the surface.

For this reaction the path searching methods, started with a linear interpolation between the reactant and product, converged again to a path which is close to the linear path, but not to that of the S_N2 reaction, Figure 20.8. The refinement of the transition state estimate of the searching string method even failed to converge, after having broken all C-H bonds in iteration 150. In contrast, the transition state estimate of the CI-string method converged to the correct transition state of the S_N2 reaction. The refinement with the mDL method needed 286 gradient evaluations, while the CI-string method converged with 62 evaluations. This happened even though the transition state estimate geometry had the relevant distances with the

C-H distance of 1.9 Å and the C-O distance of 3.0 Å even larger than the corresponding ones for system VII. Thus, for system VIII it was not necessary to produce a third starting geometry. However, as the approach with the third starting geometry was successful for the other S_N2 reaction and the calculation starting from a linear path had required many gradient evaluations, it might be worthwhile to compare this approach to the strategy starting with a linearly interpolated path.

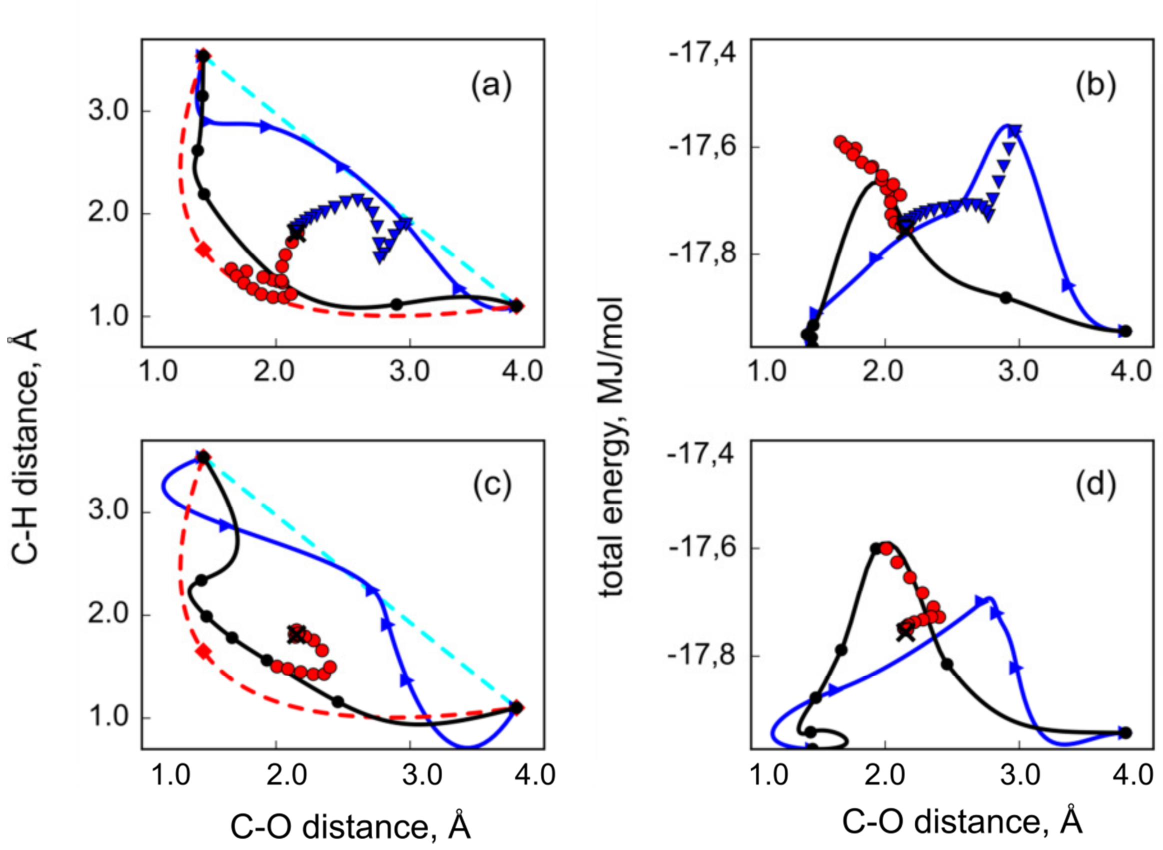

Figure 20.8: Path projections (left-hand column) and energy profiles (right-hand column) for the S_N2 reaction VIII for path search with CI-string method (a, b) and searching string method (c, d). Shown are the starting path from a linear interpolation (cyan, dashed line) and the resulting converged path (blue, solid line with triangles) as well as the starting path with a parabolic interpolation (red, dashed line with diamonds) and the corresponding converged S_N2 reaction path (black, solid line with dots). The subsequent refinement (red dots or blue triangles) was started from the image of this path with the highest energy. Adapted from reference.[13]

Figure 20.8 displays the parabolically interpolated starting path together with the paths, the path searching methods converged to. Here the starting path did not produce such a large loop with C-H distances below 1 Å, as observed in Figure 20.7 for system VII. This difference between the similar systems can be rationalized by the differences in constructing of the third start structure, Appendix J. However, the strategy using the parabolically interpolated starting path does not provide much improvement with respect to the gradient evaluations over the strategy starting with the linearly interpolated starting path.

The searching string method converged with 59 gradient evaluations, the CI-string method required 142 gradient evaluations. Both resulting paths correspond to the same S_N2 reaction. For these reactions the CI-string method met more challenges than the searching string method. This appeared in some movements of the hydrogen on the surface before it returned to a position to react with methanol (these fluctuations are not visible in Figure 20.8). The path with the searching string method was smoother. Both paths provided transition state estimates which correspond to geometries with both small C-H and C-O distances: C-H = 1.3 Å, C-O = 2.0 Å and C-H = 1.5 Å, C-O = 2.0 Å, respectively. The bonds were even shorter than for the reference transition state: C-H = 1.8 Å, C-O = 2.2 Å. The refinement from both paths converged towards the transition state. The mDL method, starting from the transition state estimate of the CI-string method, had problems to find the right direction from the beginning, making something like a loop in the projection of Figure 20.8, before it went straight to the right point. An unfortunate mode direction at the transition state estimate of the CI-string method nearly prevented the convergence, but the mDL method found the correct mode direction after about 30 iterations requiring 278 gradient evaluations for the mDL method altogether. Thus both paths with the CI-string method, irrespective of the starting path, gained by linear interpolation or parabolic interpolation, are not very well fitting the observed reaction. Both transition state estimates are still far away of the actual transition state, having to cover a maximum displacement of 1.8 Å for parabolic interpolation and 1.3 Å for the linear interpolation. The searching string method, started with parabolic interpolation, produced a transition state estimate, which was nearest to the reference transition state; however, there were still 0.9 Å to cover as maximum atomic displacement. The refinement of this structure needed 181 gradient evaluations.

This looks different for the third strategy, which is using the growing string method. The path obtained with the growing string method looks much more promising than all the other paths obtained so far. In the projections, already observed for the strategies starting with CI-string and searching string methods, the path gets much nearer to the reference transition state than any of the other path searching methods, lying somewhat in the middle between the two different paths obtained with the other methods, see Figure 20.9. Still the mDL method did not converge to the reference transition state but converged to a stationary point with two imaginary frequencies, where the methanol molecule is present again. The imaginary frequencies represent to a diffusion of the hydrogen atom and a rotation of the methanol.

The calculation was most successful when the growing string method was continued to the chain iteration no. 55. In this case the refinement reached a transition state. These twenty steps with the growing string method improved the gradient convergence criteria from $\Delta g = 0.31$ to $\Delta g = 0.08$. The path did not change much in the abstracted distances from the one which ended with the 35th chain iteration using standard convergence criteria.

As can be seen in Figure 20.9 the path is much closer to the real transition state in these projections than that of the climbing image string calculations. Also for this path, the refinement step had some difficulties to converge to the correct transition state also for this path. As can be seen in Figure 20.9 the mDL method starting from the improved growing string path first went in these projections in the wrong direction before it turned around. Therefore the refinement required also a relatively large amount of gradient evaluations, 248 for 54 iterations. The mode vector at the starting geometry from the growing string path does not look like it would belong to the S_N2 reaction. This is most probably the cause for the start towards a different transition state at the beginning of the refinement.

For other coordinates than those which are displayed in the projection of Figure 20.9, one can see that there are significant differences between the transition state estimate found with the growing string method and the reference transition state. The three atoms H, C and O were more or less in a line and also the energy of the reference transition state was approached within 5 kJ/mol, which was also the exactness of the converged results of the climbing image and searching string methods. Thus it can be expected that the correct transition state has been found. However, the global orientation of the substrate for the transition state is nearly as undefined as can be observed for gas phase reactions. However this is no surprise as in fact at the transition state only the hydroxyl moiety is bound to the surface.

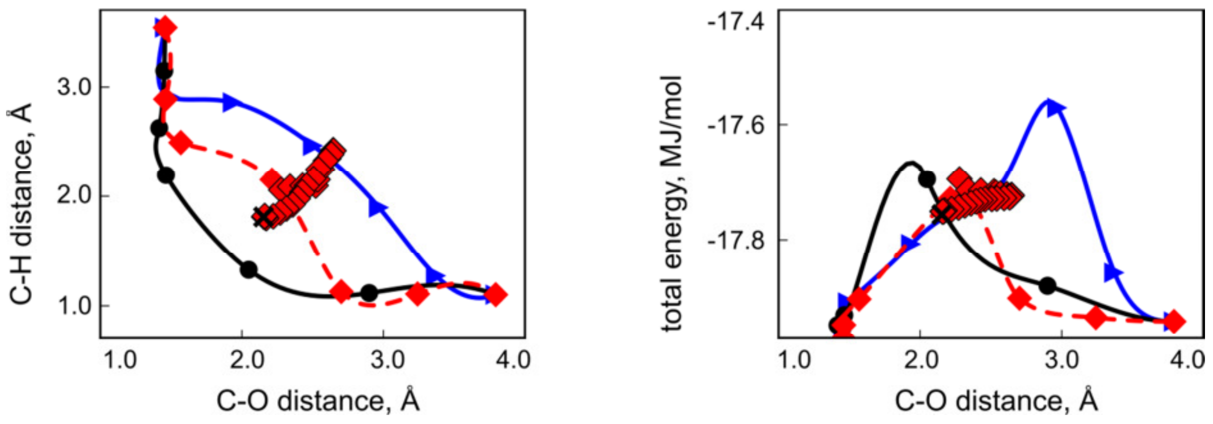

Figure 20.9: Projection in the coordinates, which characterize the abstraction, and energy profile of the growing string calculation (dashed with diamonds) followed by mDL refinement (diamonds). Shown are also the paths with the CI-string method (dots and triangles) from Figure 20.8 and the reference transition state (cross).

Looking at the structures of the growing string path, which had converged with standard convergence criteria, the small angle between the H-C direction and the C-O bond shows that the hydrogen does not lay on a line with the other two moieties, the angle being around 90 degree near the transition state estimate, see Figure 20.10. Further refinement of the growing string path does not improve this aspect. As can be seen in Figure 20.10, the angle improves only slightly over that of the path with 35 iterations. Moreover the refinement first worsened the angle, getting nearly as bad as that of the climbing image path, which had started with two images. Also the mode vector for the starting structure is not going in the right direction. Thus convergence to the correct transition state seems hard to anticipate.

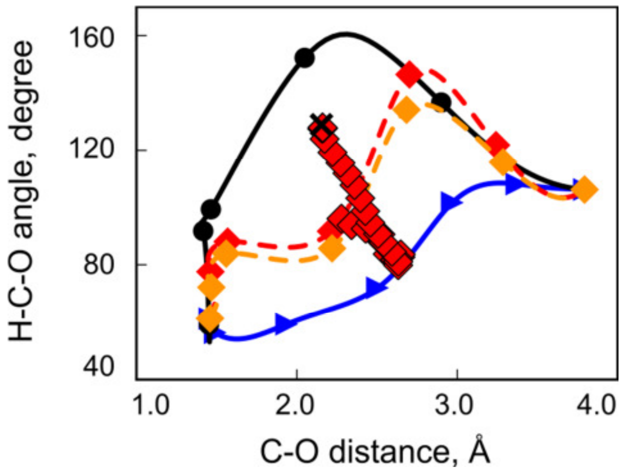

Figure 20.10: The angle between the three moieties involved in the reaction is shown against the abstracted C-O distance. Compared are growing string (dashed line with diamonds, two iterations are shown, the latter in darker color) with CI-string method (triangles starting with two images, dots for starting with three images).

Even as the transition state refinement did not result directly in the expected transition state, the path of the growing string calculation looks more like the real reaction path than the other paths, which do not even get near the transition state. However, it may be misleading to consider only the distances and the energies as in Figure 20.9, as the maximal deviation of one Cartesian coordinate from the transition state estimate to the coordinates of the reference structure is 2.4 Å, which is even larger than for the other paths.

20.10.3 Conclusions from the Tests of S_N2 Reactions

In summary, for paths which are supposed to deviate much from the linearly interpolated path, it might be favorable to start with more images on the start path to get it closer to desired path. However, this strategy does not guarantee that the correct transition state is found or that the refinement method performs better. Another option is to use methods specialized for this type of reactions, like the growing string method. But they might also get problems, especially if the desired path is not the path with lowest energy.

21. Conclusions from the Testing

The success of an automated search for transition states, especially of surface reactions, can still not be guaranteed. The two-step strategy is aiming towards this goal of a successful automated location of a transition state. The advantages of the two steps are combined in this strategy: the path searching methods can start with simple input; the refinement methods, like the standard dimer method or its improved variant, the mDL method, are able to converge to the transition state. The correct moment to switch between the two steps is a sensitive topic, which will require some more investigation. The current tested switch point worked quite well for many of the test examples, but in some cases the path searching methods had to be continued longer, so that the refinement method converges towards the correct transition state.

Of the several path search methods compared it was possible to see that the methods, which were especially optimized for producing a good starting point for the refinement step — searching string and CI-string methods — were performing better than the reference methods, NEB and standard string method, which concentrate on generating the path. These evaluation results seem to make it advisable to design global methods which put more weight on the ability of finding a good transition state estimate than on correctly displaying the path. For the specialized methods one further has to distinguish whether one is more interested in minimizing the computational cost, like gradient evaluations, or the number of non-parallelizable task steps. In the first case the searching string method is superior, as it calculates only on a reduced substring; the overall gradient evaluation requirements are usually less than for other methods. The climbing image (CI) string method is better in the second case, if the computer resources allow one to parallelize a complete chain iteration, of which all images are known at the same time. Although the CI-string method requires more gradient evaluations per chain iteration, it needs less chain iterations. The number of gradient evaluations of the CI-string method are even comparable with the number for the searching string method if the methods are combined with the better of the tested optimizers, the Multiopt. Both methods, and also the reference methods, give transition state estimates of comparable quality as the refinement methods do not favor any of the estimates. Also the cases where additional considerations have to be performed were fairly distributed.

The mDL method, introduced in this thesis, was found to perform better than the standard dimer method. It further shows much room for improvements: parameter and their values had been chosen comparable to the standard dimer method. Other values for the same parameter or even different parameters might give the same results but perform better when combined with the mDL method.

It was also found that the work of generating special coordinates, like Z-Matrix coordinates, even if only for the adsorbate, can pay off regarding gradient evaluation requirements. Therefore, exploration of an automated generation of internal coordinates is an interesting option for further studies.

In summary, it seems advisable to employ a two-step strategy where the climbing image string method with the optimizer Multiopt is followed by a refinement with the mDL method. If possible, both phases of the two-step strategy should be done in better adapted coordinates than the Cartesian ones, for example in mixed coordinates.

In case that already the initial path is far off the wanted transition state, especially if a different path is supposed to be nearby, it might be required that the path search step gets a third image as start structure. This additional image does not have to produce a very good approximation of the exact path, but should be able to guide the approximate pathways roughly to the desired region.

Developments regarding the metric of the methods, which makes only sense for the path searching methods, are not really promising for the current strategy. The metric can make the path more meaningful but as the interest aims more and more toward the transition state estimate, the path is less important. The transition state should be an image of the path in every coordinate system and metric. On the other hand, there is the problem that a chosen metric may deteriorate the convergence of the path. Therefore, it may be useful to maintain properties of the systems, like symmetry of the geometries, but this can also be done by fixing coordinates or using linear combinations of coordinates. As the strategy of fixing or combining coordinates would further reduce the complexity of the system by reducing the number of degree of freedom which have to be considered. Furthermore the strategy improves the convergence and the performance toward convergence. Thus this is a better strategy for maintaining the properties of the system.

Summary

Quantum chemistry programs like ParaGauss[3-6] have been developed to answer computationally complex chemical questions. In view of the large computational effort these codes are usually parallel implementations. They calculate the electronic structure and derive properties of atoms, molecules and other finite aggregates of atoms. These properties can be, for example, a point-wise representation of the potential energy and its nuclear displacement gradients at those points. An important application is the search for specific distinguished points on the potential energy surface, the local minima as educts, intermediates, and products and the first-order saddle points, corresponding to the transition states of reactions that connect these minima. Knowledge of these points allows unraveling the mechanism, the thermodynamics and the kinetics of chemical transformations. This thesis concentrated on three aspects.

- The first effort of this thesis was directed at improving the parallel performance of parts that calculate large amounts of tasks, e.g., the evaluation of so-called integrals.
- The second effort aimed at accelerating the convergence of the self-consistent field procedure.
- As a third aspect of the thesis a toolbox, containing routines for finding transition states, was improved and extended in an essential way and these routines were modified for large scale parallel calculations.

ParaGauss was developed from the beginning as a parallel program. However the master-slave concept, which was the favorite method in the early attempts of parallelization, has its limitations for larger processor numbers. As now supercomputers with many thousands of processors are available, it was desirable to replace this concept by another one that better fits new challenges. A work-stealing approach is a stable strategy[57] that removes the communication bottleneck which appears if the master has to steer too many slaves. A simple interface to the routines for a work-stealing algorithm was designed and implemented to render these routines easily usable and ready for integration at various places. In this way a rather general library for parallel computation of a large set of tasks or of tasks with even unpredictable size was constructed.

A proper work-stealing process requires that a processor can obtain new work from another processor. That other processor should provide facilities for enabling the stealing of tasks. Together this is called a "cooperative stealing process". This was achieved in two ways, via the message passing interface (MPI)[26] and Pthreads.[172] One solution uses remote memory access (RMA) objects which allow for one-sided communication. This is conceptually the simpler of the two approaches addressed in this work. The specification of the RMA routines of MPI require that a user-defined lock is created to allow read and write access to the RMA storage of another process. The second algorithm uses a multi-thread approach. This implementation requires that the MPI routines are thread safe, thus that all threads can access shared resources without disturbing each other.

The reason for having two implementations, which share the same interface and the principle structures, is that MPI implementations differ in the quality of the support of the basic functionalities needed. While some implementations provide only a very poor RMA realization,[59] requesting that the process from which tasks should be stolen is also in a MPI context, the thread safety for others is only provided by a variant which is less well performing and less stable.[64] The newly developed library for work stealing was integrated in ParaGauss for various parts. Test proved that it can be used efficiently for these parts.

For example, a performance test on the HLRB-II computer architecture showed that the RMA variant is able to reach up to 99% efficiency on 510 processors for a calculation of 107720 tasks which lasts approximately 5 minutes on every process. The thread variant on the computer SuperMIG of LRZ still has an efficiency of 82% on 160 processors and an average time requirement of 0.3 minutes for 3003 tasks.

Convergence acceleration using direct inversion in the iterative subspace (DIIS) was implemented in ParaGauss in the course of the current thesis.[91] Tests on actinide complexes revealed that an implementation interpolating the Kohn–Sham matrix, decreases the number of iterations required for reaching the convergence criteria related to energy and density evolution. A second implementation, working instead on the charge fitting coefficients, should improve the convergence on a convergence criteria related to the charge fit coefficients. The two implementations can be used separately or alternately. Tests showed that the DIIS methods are not successful for metal clusters, due to the small HOMO-LUMO gap, for which the perturbation theory method together with dynamical damping remains the best choice for achieving convergence.

The toolbox ParaTools was designed to search for transition states. It can be used together with various quantum chemistry programs, including ParaGauss. Each quantum chemistry program requires its own interface to the toolbox, being threated as black-box machine which generates energies and its nuclear displacement gradients at a given molecular structure. ParaTools contains routines searching for an approximate discrete representation of the reaction path, which were improved and extended in the course of the current thesis. These methods search globally for a transition state.

For local transition state searches, variants of the dimer method were implemented. The implementation consists of a standard dimer method as well as a new variant, referred to modified Dimer/Lanczos (mDL) method. The minimal curvature and the corresponding mode direction, needed for the mDL method, are provided as the smallest eigenvalue and its eigenvector of a Hessian defined in a Lanczos subspace. The new aspect of the method is the combination with a quasi-Newton approach for updating the structure. The approximated Hessian required for this quasi-Newton approach consists of an approximate positive definite Hessian that is updated by a symmetric rank-one update step using the minimal curvature and the corresponding mode direction; curvature and mode direction are provided by the Lanczos iterations. This method should improve the performance over the original dimer method.

A two-step approach for transition state search, which starts with a path searching method and subsequently refines the transition state approximation by a dimer method was extensively tested.[13] This approach allows benefitting from the various advantages for the two methods, like simply available input for the path searching method and faster and tighter convergence of the dimer method.

Applications to surface reaction showed that the newly developed mDL method is superior to the standard dimer method. The mDL method converged in most cases to the transition state belonging to the observed reaction. In few cases a different stationary point was found. For all these cases it was possible to recover the correct transition state by small tuning of the procedure.

For the global evaluation of the two–step approach the time requirements for several path searching methods and options for them were statistically examined. The fewest evaluations of the gradient to the potential energy surface were required when the path search step was done with the searching string method[118] and the specialized optimizer implemented by Chaffey-Millar et. al.[118] The same optimizer together with the climbing image string method required only few gradient evaluations more and less path iterations than the searching string method. Showing generally smoother paths and no path optimization failures, the climbing image string method with the specialized optimizer of Chaffey-Millar et. al.[118] was found to be a good choice for providing an approximation to be improved by the mDL method.

Users of ParaGauss or ParaTools will benefit from the current work in the following way: the improved parallelization will allow using larger numbers of processor and, together with the convergence acceleration of the self-consistent field part, will permit exploring larger and more complex systems, like reactions on small particles.

Further improvements of the parallelization are possible, for example, by explicitly considering the computer architecture for the parallelization algorithm. The two-step approach for the transition state search aims at an automated transition state search which should simplify and accelerate the search for reaction mechanism. Especially finding a more robust path optimization method or creating an intelligent criterion for switching to the local transition state searching method, which considers the goal of reaching the transition state, can further improve the procedure. Further improvements of the mDL method seem possibly, by an adapted choice of the parameter, once more experience from practical applications is available.

Appendix A : Averaged Elapsed Times for Simple Test Cases on LRZ Clusters

The method abbreviations are the same as in Table 5.1. The tables contain the same data than in Table 5.1 but for the three clusters of the LRZ as used in Subsection 5.1.3. ICE and MPP clusters are test clusters of the LRZ, SuperMIG is the migration system for the new petascale system SuperMUC.

Table A1: Elapsed times (in s) and efficiency for calculations on the ICE cluster. The values are averages over three runs each. For abbreviations see Table 5.1.

Method	Number of tasks	Time in s	efficiency for given number of cores 2	4	8	16
Static	1024	250.3	1.07	1.02	1.04	1.03
RMA w	1024	236.7	1.02	1.03	1.03	1.02
RMA b	1024	236.7	1.02	1.03	1.03	1.02
Th. w	1024	246.2	1.01	1.03	1.07	1.06
Th. b	1024	247.0	1.01	1.04	1.07	1.06
Static	512	123.3	1.07	1.02	1.02	0.99
RMA w	512	117.7	1.02	1.03	1.02	1.01
RMA b	512	117.5	1.02	1.03	1.03	1.01
Th. w	512	123.2	1.02	1.04	1.06	1.05
Th. b	512	122.9	1.02	1.05	1.06	1.04
Static	128	29.5	1.05	0.94	0.88	0.81
RMA w	128	29.3	1.00	0.94	0.93	0.77
RMA b	128	29.4	0.98	0.96	0.87	0.78
Th. w	128	30.9	1.00	0.96	0.91	0.74
Th. b	128	31.0	0.98	0.99	0.86	0.71
Static	32	7.5	1.01	0.99	0.97	0.89
RMA w	32	7.1	1.02	1.02	1.01	0.97
RMA b	32	7.0	1.02	1.02	1.00	0.96
Th. w	32	7.4	1.03	1.03	1.02	0.96
Th. b	32	7.4	1.02	1.03	1.03	0.95

Table A2: Elapsed times (in s) and efficiency for calculations on the MPP cluster. The values are averages over three runs each. For abbreviations see Table 5.1.

Method	Number	Time	efficiency for given number of cores			
	of tasks	in s	2	4	8	16
Static	1024	291.9	0.95	1.00	1.00	0.98
RMA b	1024	434.8	1.00	0.99	0.99	0.99
RMA w	1024	434.8	0.99	0.98	0.95	0.87
Th. b	1024	434.8	1.00	1.00	0.99	0.98
Th. w	1024	434.8	1.00	1.00	1.00	0.98
Static	512	146.2	0.95	1.01	1.00	0.96
RMA b	512	216.0	1.00	0.99	0.98	0.95
RMA w	512	216.1	0.99	0.97	0.93	0.76
Th. b	512	216.0	1.00	1.00	0.98	0.97
Th. w	512	216.0	1.00	0.99	0.99	0.96
Static	128	36.0	0.99	0.96	0.93	0.86
RMA b	128	53.9	0.99	0.97	0.93	0.85
RMA w	128	53.9	0.98	0.91	0.72	0.49
Th. b	128	53.9	0.99	0.99	0.95	0.91
Th. w	128	53.9	0.99	0.98	0.94	0.87
Static	32	8.7	0.99	0.89	0.84	0.74
RMA b	32	13.0	0.99	0.90	0.84	0.75
RMA w	32	13.0	0.90	0.72	0.44	0.28
Th. b	32	13.0	0.99	0.90	0.86	0.74
Th. w	32	13.0	0.97	0.94	0.81	0.65

Table A3: Elapsed times (in s) and efficiency for calculations on SuperMIG. The values are averages over three runs each. For abbreviations see Table 5.1.

Method	Number	Time	efficiency for given number of cores						
	of tasks	in s	2	4	8	16	32	64	80
static	1024	327.3	1.00	0.97	0.90	0.96	0.91	0.83	0.79
RMA b	1024	303.9	0.98	0.97	0.88	0.89	0.82	0.81	0.69
RMA w	1024	308.0	1.00	0.95	0.84	0.74	0.58	0.43	0.39
Th. b	1024	302.8	1.00	0.96	0.90	0.86	0.85	0.79	0.77
Th. w	1024	302.1	0.98	0.98	0.91	0.85	0.83	0.79	0.77
static	512	157.0	0.96	0.93	0.86	0.89	0.85	0.71	0.66
RMA b	512	152.4	1.00	0.95	0.89	0.83	0.77	0.69	0.64
RMA w	512	151.1	0.96	0.93	0.75	0.60	0.46	0.30	0.29
Th. b	512	151.0	1.00	0.98	0.90	0.84	0.79	0.73	0.69
Th. w	512	150.3	1.01	0.97	0.91	0.84	0.79	0.71	0.67
static	128	42.0	1.00	0.94	0.87	0.84	0.72	0.59	0.48
RMA b	128	38.7	0.97	0.84	0.81	0.76	0.46	0.57	0.30
RMA w	128	39.3	0.95	0.75	0.59	0.40	0.26	0.16	0.13
Th. b	128	39.1	0.99	0.96	0.90	0.82	0.69	0.49	0.39
Th. w	128	39.3	0.99	0.98	0.91	0.80	0.68	0.49	0.39
static	32	11.1	0.98	0.90	0.79	0.75	0.68	–	–
RMA b	32	10.6	0.95	0.89	0.81	0.71	0.63	–	–
RMA w	32	10.6	0.81	0.60	0.36	0.24	0.15	–	–
Th. b	32	10.5	0.98	0.91	0.84	0.67	0.59	–	–
Th. w	32	10.4	0.94	0.89	0.85	0.62	0.28	–	–

Appendix B : Averaged Computation Times for Four-Center Integral Test Calculations on HLRB-II

Computation times as measured as described in Section 6.3 and shown in Figure 6.1 are given for four-center integral calculations on copper clusters on HLRB-II. The standalone version of the four-center integrals was used; the employed screening divides the calculation in two loops of which only the second (the larger one) is observed. Timings are measured as real time by an MPI routine, see Section 6.3. All values are averaged over three runs. “Rest” contains the setup procedure, rounding errors and everything that is not covered in the other categories. The W/ET efficiency, Eq. (6.2), is given as efficiency. Total times and work times yielded variances below 3%, for variance times and termination times they were with up to 40% larger than 3%. Scheduling times had in general variances below 5%, exceptions are found for Cu_{20} and for Cu_4 with 8 cores and DLB with cost sorted task method.

Table B1: Times in s and efficiency, averaged over three calculations, for four-center calculations of small Cu clusters using DLB with cost sorted tasks.

Cluster	Cu_4						Cu_6	Cu_8
Cores	1	8	16	32	64	128	128	128
Total	3411	3475	3474	3489	3517	3561	18162	57551
Work	3411	3474	3471	3467	3458	3450	18046	57398
Scheduling	0.1	0.3	0.4	0.6	0.9	1.4	8.8	32.5
Imbalance	0.0	0.8	2.6	21.2	58.6	110.1	107.1	120.4
Variance	0.0	2.0	4.9	28.8	74.6	145.1	138.9	144.9
Termination	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1
Rest	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Efficiency	1.000	1.000	0.999	0.994	0.983	0.969	0.994	0.997

Table B2: Times in s and efficiency, averaged over three calculations, for four-center calculations of small Cu cluster using DLB with unsorted tasks.

Cluster	Cu_4						Cu_6	Cu_8
Cores	1	8	16	32	64	128	128	128
Total	3410	3504	3535	3614	3740	4124	18775	58147
Work	3410	3473	3476	3471	3463	3457	18075	57470
Scheduling	0.1	0.2	0.2	0.2	0.2	0.2	0.8	2.4
Imbalance	0.0	31.7	58.8	142.6	276.1	666.3	698.7	676.4
Variance	0.0	44.8	87.2	199.7	381.3	864.8	942.9	922.7
Termination	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1
Rest	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Efficiency	1.000	0.991	0.983	0.961	0.926	0.838	0.963	0.988

Table B3: Times in s and efficiency, averaged over three calculations, for four-center calculations of small Cu cluster using static task distribution with cost sorting.

Cluster	Cu_4						Cu_6	Cu_8
Cores	1	8	16	32	64	128	128	128
Total	3411	3589	3669	3847	4085	4334	20304	61109
Work	3411	3465	3469	3464	3453	3443	18018	57301
Scheduling	0.0	0.0	0.1	0.2	0.3	0.5	2.8	8.7
Imbalance	0.0	123.4	200.0	383.5	632.0	890.3	2282.6	3799.1
Variance	0.0	238.4	362.5	634.4	1558.8	2003.1	4428.6	7648.7
Termination	0.0	0.0	0.0	0.0	0.1	0.2	0.2	0.3
Rest	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Efficiency	1.000	0.966	0.945	0.900	0.845	0.794	0.887	0.938

Table B4: Times in s and efficiency, averaged over three calculations, for four-center calculations of Cu cluster using DLB with cost sorted tasks.

Cluster	Cu_{10}			Cu_{12}	Cu_{14}	Cu_{20}
Cores	128	256	510	510	510	510
Total	135015	135262	135487	286159	507450	2169538
Work	134886	134981	134658	284876	505889	2162507
Scheduling	91.4	151.7	274.9	626.1	1272.8	6230.3
Imbalance	37	130	551	656	287	797
Variance	63	183	681	836	527	1314
Termination	0.1	0.3	2.5	1.0	1.0	1.0
Rest	0.1	0.14	0.2	0.4	0.6	2.7
Efficiency	0.999	0.998	0.994	0.996	0.997	0.997

Table B5: Times in s and efficiency, averaged over three calculations, for four-center calculations of Cu cluster using DLB with unsorted tasks.

Cluster	Cu_{10}			Cu_{12}	Cu_{14}	Cu_{20}
Cores	128	256	510	510	510	510
Total	135840	136460	137849	288868	510103	2168129
Work	135059	134938	134665	285898	506962	2164626
Scheduling	5.5	5.5	7.0	13.2	23.7	170.1
Imbalance	776	1516	3175	2955	3116	3331
Variance	1028	2024	4096	3951	3949	5338
Termination	0.1	0.3	1.0	0.9	1.0	1.0
Rest	0.1	0.1	0.3	0.3	0.4	1.6
Efficiency	0.994	0.989	0.977	0.990	0.994	0.998

Table B6: Times in s and efficiency, averaged over three calculations, for four-center calculations of Cu cluster using static task distribution with cost sorting.

Cluster	Cu_{10}			Cu_{20}
Cores	128	256	510	510
Total	141022	143119	152314	2229856
Work	134703	134841	134543	2160247
Scheduling	21.0	41.7	84.1	1296.7
Imbalance	6297	8234	17684	68292
Variance	11086	17484	33027	136737
Termination	0.6	1.2	3.3	18.5
Rest	0.1	0.15	0.3	1.4
Efficiency	0.955	0.942	0.883	0.969

Appendix C : Averaged Computation Times for Cu_4 Calculations on SuperMIG

Calculation times on the SuperMIG computer of four-center integrals of Cu_4. For details see Appendix B and Section 6.3. For a set of three identical calculations total times had standard deviations smaller than 3%. Working times yield standard deviations smaller than 0.6%. Other contributions show standard deviations significantly larger: scheduling had varied up to 20%, the largest variations were determined for the imbalance and the variance, which were 34% and 39%. The small termination time varied by up to 45%.

Table C1: Times in s and efficiency, averaged over three calculations, for four-center integral calculations of Cu_4. Method: DLB with cost sorted tasks.

Cores	1	20	40	80	120	160
Total	2178	2527	2593	2707	2992	3072
Work	2178	2500	2494	2494	2504	2502
Scheduling	0.0	0.3	0.5	0.9	1.4	1.8
Imbalance	0.0	27.3	98.1	212.7	487.1	568.3
Variance	0.0	47.2	156.2	315.5	661.8	800.4
Termination	0.0	0.0	0.0	0.0	0.0	0.0
Rest	0.0	0.0	0.0	0.0	0.0	0.0
Efficiency	1.00	0.99	0.96	0.92	0.84	0.81

Table C2: Times in s and efficiency, averaged over three calculations, for four-center integral calculations of Cu_4. Method: DLB on unsorted tasks.

Cores	1	20	40	80	120	160
Total	2246	2644	2714	2868	3159	3287
Work	2246	2579	2575	2575	2588	2583
Scheduling	0.0	0.1	0.1	0.2	0.4	0.5
Imbalance	0.0	65.1	139.5	292.6	571.5	702.6
Variance	0.0	100.9	204.7	429.2	782.4	972.2
Termination	0.0	0.0	0.0	0.0	0.0	0.0
Rest	0.0	0.0	0.0	0.0	0.0	0.0
Efficiency	1.00	0.98	0.95	0.90	0.82	0.79

Table C3: Times in s and efficiency, averaged over three calculations, for four-center integral calculations of Cu_4. Method: static with cost sorted tasks.

Cores	1	20	40	80	120	160
Total	2220	2708	2893	3174	3135	3388
Work	2220	2526	2525	2523	2531	2528
Scheduling	0.0	0.1	0.1	0.2	0.3	0.4
Imbalance	0.0	181.6	367.5	650.7	603.7	859.3
Variance	0.0	366.5	730.0	1307.7	1256.5	1504.7
Termination	0.0	0.0	0.0	0.0	0.1	0.0
Rest	0.0	0.0	0.0	0.0	0.0	0.0
Efficiency	1.00	0.93	0.87	0.79	0.81	0.75

Table C4: Times in s and efficiency, averaged over three calculations, for four-center integral calculations of Cu_4 on 40 cores using the RMA variant of DLB with and without cost sorting (CS).

Method	DLB+CS	DLB
Total	3382	3108
Work	2523	2468
Scheduling	684	405
Imbalance	175	236
Variance	390	462
Termination	0.0	0.0
Rest	0.0	0.0
Efficiency	0.75	0.80

Appendix D : Averaged Calculation Times for the Four-Center Integral Implementation of ParaGauss

The timings were extracted from the second SCF loop of the program ParaGauss. The calculations were carried out on a Nehalem Linux cluster with 1 to 24 cores (see Subsection 5.1.2). For definition of subdivisions of timings see Sections 6.3 and 6.6. Standard deviations for total and work times were below 1%; for scheduling times they were below 10%. The DLB methods run always over unsorted task lists.

Table D1: Time in s and efficiency, averaged over three calculations for the four-center integrals of Pd_{38}. Method: DLB over groups of tasks.

Cores	1	2	4	8	16	24
Total	1565	1560	1547	1550	1552	1556
Work	1554	1549	1536	1537	1538	1538
Scheduling	6.6	10.9	8.6	11.0	10.9	13.1
Imbalance	0.0	0.2	1.4	0.9	1.1	3.4
Variance	0.0	0.4	2.9	0.9	3.6	5.2
Termination	3.7	0.4	0.7	0.4	1.5	1.4
Rest	0.2	0.2	0.2	0.3	1.0	1.0
Efficiency	0.9933	0.9932	0.9929	0.9918	0.9911	0.9883

Table D2: Time in s and efficiency, averaged over three calculations for the four-center integrals of Pd_{38}. Method: Method: DLB over all tasks.

Cores	1	2	4	8	16	24
Total	1919	1943	1917	1914	1924	1927
Work	1545	1568	1547	1547	1545	1548
Scheduling	194.4	374.5	370.3	367.0	376.2	377.3
Imbalance	0.0	0.0	0.1	0.1	0.7	0.9
Variance	0.0	0.0	0.1	0.4	1.4	2.2
Termination	179.2	0.1	0.1	0.4	1.2	2.8
Rest	0.2	0.2	0.2	0.2	0.8	0.8
Efficiency	0.8052	0.8071	0.8067	0.8079	0.8033	0.8021

Table D3: Time in s and efficiency, averaged over three calculations for the four-center integrals of Pd_{38}. Method: Method: static task distribution.

Cores	1	2	4	8	16	24
Total	1540	1575	1568	1598	1661	1729
Work	1525	1538	1525	1522	1523	1524
Scheduling	8.6	12.6	19.3	33.7	62.6	90.5
Imbalance	0.0	14.6	6.4	9.9	12.1	21.4
Variance	0.0	29.3	13.6	20.3	22.6	43.2
Termination	5.5	9.8	16.5	32.0	63.1	92.8
Rest	0.2	0.2	0.2	0.2	0.9	0.9
Efficiency	0.9907	0.9765	0.9729	0.9526	0.9170	0.8815

Appendix E : Convergence of the Self Consistent Field Procedure for Palladium Clusters

Convergence of charge fit coefficients and energy as measured by relative values between consecutive iterations are provided for the palladium cluster series, Pd_n with n=6–116, of Section 11.2. The convergence accelerators dynamic damping (without DIIS) and DIIS-KS for two different initiation thresholds are compared.

Table E1: Iteration number (iter.) in which the charge fit coefficient convergence measure in arbitrary units was below twice the value of the convergence limit (conv.) for the palladium clusters with n atoms. The calculations with DIIS started after the provided threshold (Thr.) in Hartree was reached.

	Without DIIS		DIIS, Thr. = 0.1		DIIS, Thr. = 0.05	
N	Conv.	Iter.	Conv.	Iter.	Conv.	Iter.
6	2E-07	30	1E-06	25	9E-07	26
14	5E-07	33	2E-06	29	3E-06	23
38	2E-06	36	2E-05	27	2E-05	26
44	2E-06	36	2E-05	35	2E-05	30
68	4E-06	47	6E-05	40	4E-05	41
92	7E-06	54	1E-04	42	1E-04	43
116	2E-05	58	–	–	2E-04	75

Table E2: Iteration number (iter.) in which the charge fit coefficient convergence measure in arbitrary units was below twice the value of the convergence limit (conv.) for the palladium clusters with n atoms. The calculations with DIIS started after the provided threshold (Thr.) in Hartree was reached.

	Without DIIS		DIIS, Thr. = 0.1		DIIS, Thr. = 0.05	
N_a	Conv.	Iter.	Conv.	Iter.	Conv.	Iter.
6	5E-11	36	9E-11	27	5E-11	27
14	3E-10	36	2E-10	31	3E-10	27
38	4E-09	31	4E-09	31	3E-09	30
44	6E-09	31	4E-09	44	4E-09	33
68	1E-08	40	1E-08	42	1E-08	45
92	4E-08	46	4E-08	39	3E-08	43
116	5E-08	47	–	–	7E-08	69

Appendix F : Number of Negative Eigenvalues of the General Inverse Hessian for the Modified Dimer/Lanczos Method

The combined inverse Hessian $\mathbf{H}^k$, see Eq. (14.12),

$$\mathbf{H}^k = \mathbf{H}^k_{\text{BFGS}} + \Delta\mathbf{H}^k_{\text{SR1}}$$

can have at most one negative eigenvalue, for a positive definite inverse Hessian $\mathbf{H}^k_{\text{BFGS}}$. This can be shown as follows:

An arbitrary eigenvector $\boldsymbol{\theta}$ of the inverse Hessian $\mathbf{H}^k$, which is not equal to **m**, fulfills the following equation:

$$\begin{aligned}\gamma\boldsymbol{\theta} &= \mathbf{H}^k\boldsymbol{\theta} \\ &= \mathbf{H}^k_{\text{BFGS}}\boldsymbol{\theta} + \frac{\mathbf{m} - \mathbf{H}^k_{\text{BFGS}}C\mathbf{m}}{\mathbf{m}^T\mathbf{m} - C\mathbf{m}^T\left(\mathbf{H}^k_{\text{BFGS}}\right)^T\mathbf{m}}\mathbf{m}^T\left(\mathbf{H}^k_{\text{BFGS}}\right)^T\boldsymbol{\theta}\end{aligned}$$

A scalar product with **m** yields:

$$\begin{aligned}\mathbf{m}^T\gamma\boldsymbol{\theta} &= \mathbf{m}^T\mathbf{H}^k\boldsymbol{\theta} \\ &= \mathbf{m}^T\mathbf{H}^k_{\text{BFGS}}\boldsymbol{\theta} + \frac{\mathbf{m}^T\mathbf{m} - \mathbf{m}^T\mathbf{H}^k_{\text{BFGS}}C\mathbf{m}}{\mathbf{m}^T\mathbf{m} - C\mathbf{m}^T\left(\mathbf{H}^k_{\text{BFGS}}\right)^T\mathbf{m}}\mathbf{m}^T\left(\mathbf{H}^k_{\text{BFGS}}\right)^T\boldsymbol{\theta}\end{aligned}$$

This equation can be simplified by using two relations: $\boldsymbol{\theta} \perp \mathbf{m}$ as both $\boldsymbol{\theta}$ and **m** are eigenvectors of the combined inverse Hessian $\mathbf{H}^k$ and $\mathbf{H}^k_{\text{BFGS}} = \left(\mathbf{H}^k_{\text{BFGS}}\right)^T$ as the inverse Hessian H^k_{BFGS} is positive definite.

$$\begin{aligned}0 &= \mathbf{m}^T\mathbf{H}^k_{\text{BFGS}}\boldsymbol{\theta} + \frac{\mathbf{m}^T\mathbf{m} - C\mathbf{m}^T\mathbf{H}^k_{\text{BFGS}}\mathbf{m}}{\mathbf{m}^T\mathbf{m} - C\mathbf{m}^T\mathbf{H}^k_{\text{BFGS}}\mathbf{m}}\mathbf{m}^T\mathbf{H}^k_{\text{BFGS}}\boldsymbol{\theta} \\ 0 &= 2\mathbf{m}^T\mathbf{H}^k_{\text{BFGS}}\boldsymbol{\theta}\end{aligned}$$

Together these relations allow to show that for a eigenvector $\boldsymbol{\theta}$ of the combined inverse Hessian $\mathbf{H}^k$ the component $\mathbf{m}^T H^k_{\text{BFGS}}\boldsymbol{\theta}$ vanishes. This can be used for the scalar product of the eigenvector equation with $\boldsymbol{\theta}$.

$$\begin{aligned}\boldsymbol{\theta}^T\gamma\boldsymbol{\theta} &= \boldsymbol{\theta}^T\mathbf{H}^k_{\text{BFGS}}\boldsymbol{\theta} + \frac{\boldsymbol{\theta}^T\mathbf{m} - C\boldsymbol{\theta}^T\mathbf{H}^k_{\text{BFGS}}\mathbf{m}}{\mathbf{m}^T\mathbf{m} - C\mathbf{m}^T\left(\mathbf{H}^k_{\text{BFGS}}\right)^T\mathbf{m}}\mathbf{m}^T\mathbf{H}^k_{\text{BFGS}}\boldsymbol{\theta} \\ &= \boldsymbol{\theta}^T\mathbf{H}^k_{\text{BFGS}}\boldsymbol{\theta} + 0 \\ \gamma &= \boldsymbol{\theta}^T\mathbf{H}^k_{\text{BFGS}}\boldsymbol{\theta}\end{aligned}$$

All components of $\mathbf{H}^k\boldsymbol{\theta}$ except the one parallel to $\boldsymbol{\theta}$ vanish. Using the relation just found one can see that the eigenvalue γ is only influenced by the positive definite inverse Hessian $\mathbf{H}^k_{\text{BFGS}}$.

This enforces that γ is positive. As $\boldsymbol{\theta}$ has been chosen arbitrary this shows that if the complete inverse Hessian has maximal one negative eigenvalue, which lays in direction of the mode vector.

Appendix G : Derivation of Transformation Formulas for Reduced Metric

The derivation of the equations of Subsection 18.4.2.2, which are essential the transformation formulas of coordinates, are shown in detail in the current appendix.

The observation starts with the simple transformation Eq. (18.13)

$$\delta s_i = \left(\mathbf{J}^l{}_i \delta_{lk} \mathbf{J}^k{}_j\right) \delta s^j = \left(\mathbf{J}^T \mathbf{J}\right)_{ij} \delta s^j$$

which describes how contra- and co-variant representations can be transformed into each other. For the following equations the short hand notation $\left(\mathbf{J}^T\mathbf{J}\right)_{ij}$ will be used for $\left(\mathbf{J}^l{}_i \delta_{lk} \mathbf{J}^k{}_j\right)$. For the reduced metric the coordinates differences are separated into two groups $\delta\mathbf{s} = (\delta\mathbf{y}, \delta\mathbf{v})^T$, one corresponding to the changes in internal coordinates the other to the global displacement. The Jacobean matrix $\mathbf{J}$ can be separated likewise, $\mathbf{J} = \left(\mathbf{J}_y, \ \ \mathbf{J}_V\right)$. Setting the co-variant global displacement to zero, this leads to Eq. (18.14), (with $i, j = 1, \ldots n$ and $p, q = 1, \ldots 6$ for $\dim y = n$)

$$\begin{aligned} \delta y_i &= \left(\mathbf{J}_y{}^T \mathbf{J}_y\right)_{ij} \delta y^j + \left(\mathbf{J}_y{}^T \mathbf{J}_V\right)_{ip} \delta v^p \\ 0 &= \left(\mathbf{J}_V^T \mathbf{J}_V\right)_{qp} \delta v^p + \left(\mathbf{J}_V^T \mathbf{J}_y\right)_{qj} \delta y^j \end{aligned}$$

one can transform the second equation in a relation between δv^p and δy^j and insert it in the first expression:

$$\begin{aligned} \delta v^p &= -\left[\left(\mathbf{J}_V^T \mathbf{J}_V\right)^{-1} \left(\mathbf{J}_V^T \mathbf{J}_y\right)\right]_i^p \delta y^i \\ \delta y_i &= \left(\left(\mathbf{J}_y{}^T \mathbf{J}_y\right) - \left(\mathbf{J}_V^T \mathbf{J}_y\right)^T \left(\mathbf{J}_V^T \mathbf{J}_V\right)^{-1} \left(\mathbf{J}_V^T \mathbf{J}_y\right)\right)_{ij} \delta y^j \end{aligned}$$

The relevant equation for δy_i can be reformulated a bit, getting it more in a form similar to Eq. (18.13), where the identity matrix in the middle is modified to take into account that the global positioning is removed from the metric, see Eq. (18.15).

$$\delta y_i = \left[\mathbf{J}_y{}^T \left(\mathbf{I} - \mathbf{J}_V \left(\mathbf{J}_V^T \mathbf{J}_V\right)^{-1} \mathbf{J}_V^T\right) \mathbf{J}_y\right]_{ij} \delta y^j$$

The choice of the global positioning parameters influences the matrix $\mathbf{J}_V$, while in the relations before no special requirement for them was given. Global translation and global rotation is easily separable in the coordinates $\mathbf{v} = (\mathbf{t}, \mathbf{r})$. A common choice for the translation vector **t** are the three dimensions of an orthogonal (Cartesian) space. For the rotation unit quaternions are used. From the quaternion a rotation matrix which transforms the Cartesian position of every atom can be generated. The quaternions may be expressed as a three component vector **r** for the rotation around a given center. Because the global positioning is not used to change anything for the Cartesian coordinates, it makes sense to set changes in translation and rotation to 0. Now one can write $\mathbf{J}_V$ as $\mathbf{J}_V = (\mathbf{J}_T,\ \mathbf{J}_R)$. $\mathbf{J}_T$ and $\mathbf{J}_R$ are both $3 \times 3N$ matrices and consist of 3×3 blocks $\mathbf{J}_T^{(k)}$ and $\mathbf{J}_R^{(k)}$ for each atom *k* where $\mathbf{R}_k = (R_{k,1},\ R_{k,2},\ R_{k,3})$ are the coordinates of atom *k* with respect to the center of rotation, and $R_k^2 = \sum_i R_{k,i}^2$.

$$\mathbf{J}_T^{(k)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$J_R^{(k)} = \begin{pmatrix} 0 & R_{k,3} & -R_{k,2} \\ -R_{k,3} & 0 & R_{k,1} \\ R_{k,2} & -R_{k,1} & 0 \end{pmatrix}$$

The matrix $\mathbf{J}_V^T \mathbf{J}_V$, of which the inverse is required for the translation, can be expressed as

$$\mathbf{J}_V^T \mathbf{J}_V = \begin{pmatrix} \mathbf{J}_T^T \mathbf{J}_T & \mathbf{J}_T^T \mathbf{J}_R \\ \mathbf{J}_R^T \mathbf{J}_T & \mathbf{J}_R^T \mathbf{J}_R \end{pmatrix} = \begin{pmatrix} \mathbf{1}N & \sum_k R_{k,l}\epsilon_{ijl} \\ \sum_k R_{k,l}\epsilon_{ijl} & \sum_k R_k^2 \delta_{ij} - \sum_k R_{k,i} R_{k,j} \end{pmatrix}$$

If one has the relation $\sum_i \mathbf{R}_i = 0$ there would be no interaction terms between rotation and translation for the matrix $\mathbf{J}_V^T \mathbf{J}_V$. By choosing the geometrical center of the system as rotation center this is achieved: if the geometrical center is chosen as the rotation center, then atomic coordinates fulfill the relation $\sum_i (\mathbf{X}_i - \bar{\mathbf{X}}) = 0$. Therefore by setting the rotation center to the global center the matrix $\mathbf{J}_V^T \mathbf{J}_V$ can be further reduced to

$$\mathbf{J}_V^T \mathbf{J}_V = \begin{pmatrix} \mathbf{1}N & 0 \\ 0 & \sum_k R_k^2 \delta_{ij} - \sum_k R_{k,i} R_{k,j} \end{pmatrix}$$

This allows to simplify the transformation to the final form.

$$\delta y_i = \left[\mathbf{J}_y{}^T \left(\mathbf{I} - \frac{1}{N} \mathbf{J}_T \mathbf{J}_T^T - \mathbf{J}_R \left(\sum_k R_k^2 \mathbf{I} - \sum_k \mathbf{R}_k \mathbf{R}_k^T \right)^{-1} \mathbf{J}_R^T \right) \mathbf{J}_y \right]_{ij} \delta y^j$$

Appendix H : Atomic Basis Sets for ParaGauss Calculations

The basis sets for carrying out calculations with ParaGauss of the Sections 5, 6 and 11 are provided.

Table H1: Number of basis function of the basis sets used in Section 5. The palladium basis was also used in Section 11. The palladium basis is based on a basis set of Huzinaga,[173-174] where 1 s, 2 p and 1 d functions were added and relativistic contractions were used. The basis sets for anthranilic acid were a 6-311++G(2d,2p) basis set[175-176] for hydrogen and 6-311++G(3df,3dp) basis sets for carbon, oxygen and nitrogen.[175-177]

Atom	Basis type	Number of basis functions				
		r^2	s	p	d	f
Pd	Orbitals (uncontracted)		18	13	9	
	Orbitals (contracted)		7	6	4	
	Charge fit	6	17	5	5	
H	Orbitals (uncontracted)		6	2		
	Orbitals (contracted)		4	2		
	Charge fit	2	6	2		
C	Orbitals (uncontracted)		12	6	3	1
	Orbitals (contracted)		5	4	3	1
	Charge fit	6	12	6	3	1
O	Orbitals (uncontracted)		12	6	3	1
	Orbitals (contracted)		5	4	3	1
	Charge fit	6	12	6	3	1
N	Orbitals (uncontracted)		12	6	3	1
	Orbitals (contracted)		5	4	3	1
	Charge fit	6	12	5	5	

Table H2: Number of basis function for the basis sets used in Section 6. For the cupper cluster a well-tempered Gaussian basis set (WTBS),[178-179] was chosen, while for the platinum cluster a def2-TZVP[180-182] basis set is employed.

Atom	Orbital	Number of basis functions			
	Basis	s	p	d	f
Cu	Uncontracted	26	17	13	
	Contracted	4	2	1	
Pt	Uncontracted	8	7	6	1
	Contracted	6	4	3	1

Table H3: Number of basis functions in the basis sets used for the actinide complexes in Section 11. The basis sets for oxygen and hydrogen[183-185] were used for all complexes. The neptunium basis of Minami and Matsuoka was used.[186] For Americium complex a ANO-RCC[187] basis as foundation employing a simplifying contraction was employed.[188]

Atom	Basis type	Number of basis functions				
		r^2	s	p	d	f
O	Orbitals (uncontracted)		9	5	1	
	Orbitals (contracted)		5	4	1	
	Charge fit	5	9	5	5	
H	Orbitals (uncontracted)		6	1		
	Orbitals (contracted)		4	1		
	Charge fit	1	6	5		
Np	Orbitals (uncontracted)		24	19	16	11
	Orbitals (contracted)		10	7	7	4
	Charge fit	9	24	5	5	5
Am	Orbitals (uncontracted)		26	23	17	13
	Orbitals (contracted)		10	8	7	5
	Charge fit	23	26	5	5	5

Appendix I : Parameter Choice for Performance Tests

The performance tests in Chapter 20 have been carried out with ParaTools.[164] The meaning of the parameters of ParaTools is described in the documentation.[164] Therefore, the following tables provide only the values for the parameters. Parameters which have to be set differently for various calculations are not shown, they have to be set according to the requirements of the specific calculations. An exception is the parameter phi_tol for the dimer/mDL calculations as it is only changed at a single occurrence in Section 20.5 and otherwise is set according to Table I3.

Table I1: Parameters for VASP single point calculations.

Parameter	Value
ENCUT	320
ENAUG	650
SIGMA	0.15
EDIFF	1.00E-06
EDIFFG	-2.00E-02
PREC	Normal
GGA	91
VOSKOWN	1
ISYM	0
ISMEAR	1
IALGO	48
NPLANE	1
LMAXPAW	0
NPAR	1
IDIPOL	3
ISIF	2
LCHARG	.FALSE.
LWAVE	.FALSE.
LREAL	.FALSE.

Table I2: Parameter choice for path searching (first step of the two-step strategy). Shown are only the parameters, which are relevant for the calculations and which have not been specified in the text. Be aware that the parameter “spring” is only used for NEB and otherwise ignored.

Parameter	Value
maxstep	0.1
spring	5.0
cpu_architecture	9
maxit	35
output_level	2
output_geo_format	vasp
xtol	0.03
ftol	0.1
pmin	1
max_sep_ratio	0.01
output_path	workplace
beads_count	7
pmax	1

Table I3: Parameter choice for the refinement step of transition state search with dimer or mDL methods. Shown are only the parameter which are relevant for the calculations and which are shared by both methods.

Parameter	Value
phi_tol	0.01
trans_converged	0.02
max_gradients	None
dimer_distance	0.0025
logfile	None
max_step	0.1
trajectory	one_file
cache	None
max_rotations	10
max_translation	150

Appendix J : Start Geometry of Images for a Parabolic Initial Path

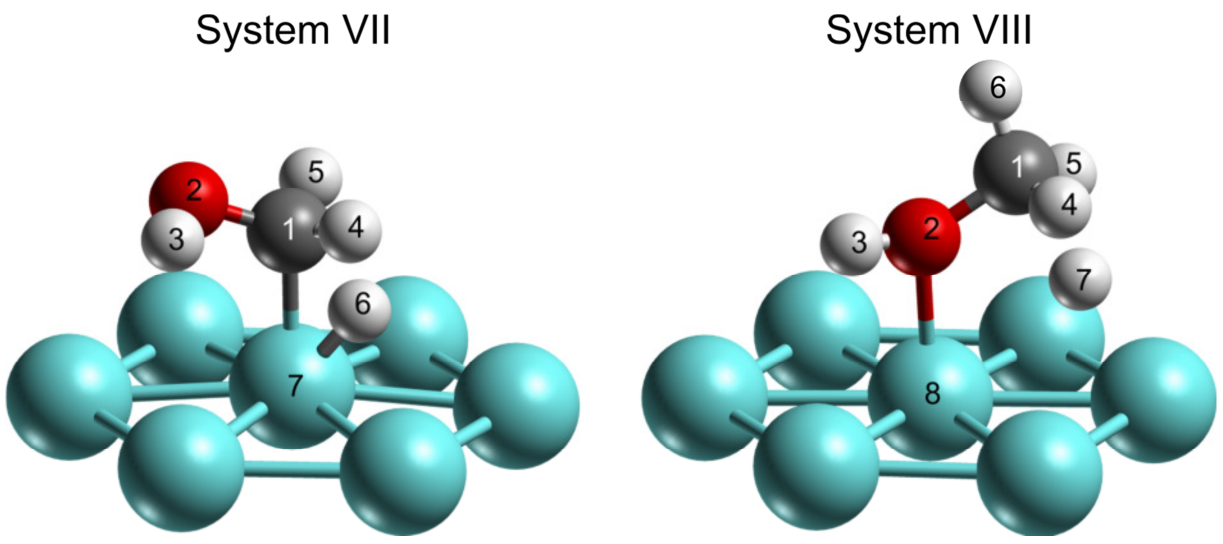

Figure J.1 : Third image for parabolically starting path for the systems VII and VIII.

To create a third image for a parabolic starting path for reaction path optimization, which should be closer to the expected reaction path than a linear starting path, one could use information from previous investigations on the system or from experimental data. However additional information should be required as little as possible. This strategy was tested for the two S_N2 Reactions in Section 20.10. The two structures are displayed in Figure J.1.

System VII (Hydroxymethyl): The reactant geometry was taken as source of the intermediate image. Only the hydrogen center H6 (atom with number 6 in Figure J.1) was repositioned, closer to the CH_2 group (consisting of C1, H4 and H5), with distances of C1-H6 = 1.4 Å and Pt7-H6 = 1.7 Å. The C1-O2 distance was kept at 1.39 Å, as in the reactant.

System VIII (Methanol): As for system VII the reactant geometry was used as source for the third image. The C1-O2 distance was kept fixed during the complete procedure at 1.46 Å. Comparing reactant and product to each other, it was found that the methyl group (atoms C1, H4, H5 and H6 in Figure J.1) is the group moving strongest. Thus, as a first movement the methyl group was rotated around the axis through the oxygen and the platinum surface (thus the axis through O2 and Pt8) in such a way that the carbon reaches the same angle related to the surface as in the product geometry. As next step the hydrogen H7 is moved towards its position at the methyl group, admitting a change in the distance to the surface. Here the distances C1-H7=1.65 Å and Pt8-H7 = 1.89 Å were chosen.

Bibliography

1. Janssen, C. L.; Nielsen, I. M. B., *Parallel Computing in Quantum Chemistry*. CRC Press: Boca Raton, 2008.
2. Belling, T.; Grauschopf, T.; Krüger, S.; Nörtemann, F.; Staufer, M.; Mayer, M.; Nasluzov, V. A.; Birkenheuer, U.; Hu, A.; Matveev, A.; Shor, A. V.; Fuchs-Rohr, M. S. K.; Neyman, K. M.; Ganyushin, D. I.; Kerdcharoen, T.; Woiterski, A.; Gordienko, A. B.; Majumder, S.; Rotllant, M. H.; Ramakrishnan, R.; Dixit, G.; Nikodem, A.; Soini, T.; Roderus, M.; Rösch, N. *ParaGauss*, Version 3.2; Technische Universität München: 2010.
3. Belling, T.; Grauschopf, T.; Krüger, S.; Nörtemann, F.; Staufer, M.; Mayer, M.; Nasluzov, V. A.; Birkenheuer, U.; Hu, A.; Matveev, A.; Shor, A. V.; Fuchs-Rohr, M. S. K.; Neyman, K. M.; Ganyushin, D. I.; Kerdcharoen, T.; Woiterski, A.; Majumder, S.; Rösch, N., ParaGauss, Version 3.1. Technische Universität München: 2006.
4. Belling, T.; Grauschopf, T.; Krüger, S.; Nörtemann, F.; Staufer, M.; Mayer, M.; Nasluzov, V. A.; Birkenheuer, U.; Rösch, N., ParaGauss, Version 1.9. Technische Universität München: 1998.
5. Dunlap, B. I.; Rösch, N., The Gaussian-Type Orbitals Density-Functional Approach to Finite Systems, in: Density Functional Theory of Many-Fermion Systems. *Adv. Quantum Chem.* **1990**, *21*, 317-399.
6. Belling, T.; Grauschopf, T.; Krüger, S.; Mayer, M.; Nörtemann, F.; Staufer, M.; Zenger, C.; Rösch, N. In *Quantum Chemistry on Parallel Computers: Concepts and Results of a Density Functional Method*, Proceedings of the International FORTWIHR Conference on HPSEC, Munich, March 16-18; Bungartz, H.-J.; Durst, F.; Zenger, C., Eds. Springer: Munich, 1998; pp 441-455.
7. Koch, W.; Holthausen, M. C., *A Chemist's Guide to Density Functional Theory*. Wiley-VCH: Weinheim, 2001.
8. Rösch, N.; Matveev, A. V.; Nasluzov, V. A.; Neyman, K. M.; Moskaleva, L. V.; Krüger, S., Quantum Chemistry with the Douglas-Kroll-Hess Approach to Relativistic Density Functional Theory: Efficient Methods for Molecules and Materials. In *Relativistic Electronic Structure Theory - Applications*, Schwerdtfeger, P., Ed. Elsevier: Amsterdam, 2004; Vol. 14, pp 656-722.
9. Yudanov, I. V.; Matveev, A. V.; Neyman, K. M.; Rösch, N., How the C-O Bond Breaks during Methanol Decomposition on Nanocrystallites of Palladium Catalysts. *J. Am. Chem. Soc* **2008**, *130* (29), 9342-9352.
10. Schauermann, S.; Hoffmann, J.; Johánek, V.; Hartmann, J.; Libuda, J.; Freund, H.-J., Catalytic Activity and Poisoning of Specific Sites on Supported Metal Nanoparticles. *Angew. Chem. Int. Ed.* **2002**, *41* (14), 2532-2535.
11. Soini, T. M. Dissertation, in preperation.
12. Pulay, P., Improved SCF Convergence Acceleration. *J. Comp. Chem.* **1982**, *3* (4), 556-560.
13. Nikodem, A.; Matveev, A.; Zheng, B.; Rösch, N., Efficient Two-Step Procedure for Locating Transition States of Surface Reactions. *J. Chem. Theory Comput.* **2013**, *9* (1), 588-599.
14. Szabo, A.; Ostlund, N. S., *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Dover Publications, Inc.: New York, 1989.
15. Douglas, M.; Kroll, N. M., Quantum electrodynamical corrections to the fine structure of helium. *Ann. Phys. (NY)* **1974**, *82* (1), 89-155.
16. Rösch, N.; Krüger, S.; Mayer, M.; Nasluzov, V. A., The Douglas-Kroll-Hess Approach to Relativistic Density Functional Theory: Methodological Aspects and Applications to Metal Complexes and Clusters. In *Recent Developments and Applications of*

Modern Density Functional Theory, Seminario, J. M., Ed. Elsevier: Amsterdam, 1996; pp 497-566.
17. Dunlap, B. I.; Rösch, N.; Trickey, S. B., Variational fitting methods for electronic structure calculations. *Mol. Phys.* **2010**, *108*, 3167-3180.
18. Becke, A. D., A multicenter numerical integration scheme for polyatomic molecules. *J. Chem. Phys* **1988**, *88* (4), 2547-2553.
19. Gill, P. M. W.; Johnson, B. G.; Pople, J. A., A standard grid for density functional calculations. *Chem. Phys. Let.* **1993**, *209* (5-6), 506-512.
20. Dunlap, B. I.; Connolly, J. W. D.; Sabin, J. R., On some approximations in applications of Xα theory. *J. Chem. Phys.* **1979**, *71* (8), 3396-3402.
21. Dunlap, B. I., Rate of self-consistent-field convergence in the X α method. *Phys. Rev. A* **1982**, *25* (5), 2847-2849.
22. Staufer, M. A Parallel Implementation of the Density Functional Method: Analytical Energy Gradients, DF quadrature and Applications to Chemisorption. Dissertation, Technische Universität München, München, 1999.
23. Harrison, R. J.; Shepard, R., Ab Initio Molecular Electronic Structure On Parallel Computers. *Annu. Rev. Phys. Chem.* **1994**, *45*, 623-658.
24. Cooper, M. D.; Burton, N. A.; Hall, R. J.; Hillier, I. H., Combined Hartree-Fock and density functional theory: a distributed memory parallel implementation. *J. Mol. Struct.: THEOCHEM* **1994**, *315*, 97-107.
25. Sosa, C. P.; Scalmani, G.; Gomperts, R.; Frisch, M. J., Ab initio quantum chemistry on a ccNUMA architecture using openMP. III. *Parallel Comput.* **2000**, *26* (7-8), 843-856.
26. Message Passage Interface Forum, MPI: A Message-Passing Interface Standard Version 2.2. 2009.
27. Stanton, J. F.; Gauss, J.; Watts, J. D.; Lauderdale, W. J.; Bartlett, R. J., The ACES II program system. *Int. J. Quantum Chem.* **1992**, *44* (Issue Supplement 26), 879–894.
28. Lotrich, V.; Flocke, N.; Ponton, M.; Yau, A. D.; Perera, A.; Deumens, E.; Bartlett, R. J., Parallel implementation of electronic structure energy, gradient, and Hessian calculations. *J. Chem. Phys* **2008**, *128* (19), 194104.
29. von Arnim, M.; Ahlrichs, R., Performance of Parallel TURBOMOLE for Density Functional Calculations. *J. Comp. Chem.* **1998**, *19* (15), 1746-1757.
30. Li, Y. S.; Wrinn, M. C.; Newsam, J. M.; Sears, M. P., Parallel Implementation of a Mesh-Based Density Functional Electronic Structure Code. *J. Comput. Chem.* **1995**, *16* (2), 226-234.
31. Ernenwein, R.; Rohmer, M.-M.; Benard, M., A Program System For Ab Initio MO Calculations On Vector And Parallel Processing Machines I. Evaluation of integrals. *Comput. Phys. Commun.* **1990**, *58* (3), 305-328.
32. Chasman, D.; Beachy, M. D.; Wang, L.; Friesner, R. A., Parallel pseudospectral electronic structure: I. Hartree-Fock calculations. *J. Comput. Chem.* **1998**, *19* (9), 1017-1029.
33. Brode, S.; Horn, H.; Ehrig, M.; Moldrup, D.; Rice, J. E.; Ahlrichs, R., Parallel Direct SCF and Gradient Program for Workstation Clusters. *J. Comput. Chem.* **1993**, *14* (10), 1142-1148.
34. Brown, P.; Woods, C.; McIntosh-Smith, S.; Manby, F. R., Massively Multicore Parallelization of Kohn-Sham Theory. *J. Chem. Theory Comput.* **2008**, *4* (10), 1620-1626.
35. Yasuda, K., Two-electron integral evaluation on the graphics processor unit. . *J. Comput. Chem.* **2008**, *29* (3), 334–342.
36. Ramdas, T.; Egan, G.; Abramson, D.; Baldridge, K., Towards a special-purpose computer for Hartree–Fock computations. *Theor. Chem. Acc.* **2008**, *120* (1-3), 133-153.
37. Taylor, P. R.; Charles W. Bauschlicher, J., Strategies for obtaining the maximum performance from current supercomputers. *Theor. Chim. Acta* **1987**, *71* (2-3), 105-115.

38. Kenny, J. P.; Janssen, C. L.; Valeev, E. F.; Windus, T. L., Components for integral evaluation in quantum chemistry. *J. Comput. Chem.* **2008**, *29* (4), 562-577.
39. Guerra, C. F.; Visser, O.; Snijders, J. G.; te Velde, G.; Baerends, E. J., Parallelization of the Amsterdam Density Functional Program. In *Methods and Techniques in Computational Chemistry (METECC-95)*, Corongiu, E. C. a. G., Ed. "Université L. Pasteur: Strasbourg, France, 1995; p 305.
40. Colvin, M. E.; Janssen, C. L.; Whiteside, R. A.; Tong, C. H., Parallel direct SCF for large-scale calculations. *Theor. Chim. Acta* **1993**, *84* (4-5), 301-314.
41. Márquez, A. M.; Oviedo, J.; Sanz, J. F.; Dupuis, M., Parallel computation of second derivatives of RHF energy on distributed memory computers. *J. Comput. Chem.* **1997**, *18* (2), 159-168.
42. Ferrighi, L.; Frediani, L.; Fossgaard, E.; Ruud, K., Parallelization of the integral equation formulation of the polarizable continuum model for higher-order response functions. *J. Chem. Phys* **2006**, *125*, 154112.
43. Yoshihiro, T.; Sato, F.; Kashiwagi, H., Distributed parallel processing by using the object-oriented technology in ProteinDF program for all-electron calculations on proteins. *Chem. Phys. Lett.* **2001**, *346*, 313-321.
44. Baker, J.; Pulay, P., An Efficient Parallel Algorithm for the Calculation of Canonical MP2 Energies. *J. Comput. Chem.* **2002**, *23* (12), 1150-1156.
45. Dinan, J.; Olivier, S.; Sabin, G.; Prins, J.; Sadayappan, P.; Tseng, C.-W., A message passing benchmark for unbalanced applications. *Simulat. Model Pract. Theor.* **2008**, *16* (9), 1177-1189.
46. Willebeek-LeMair, M. H.; Reeves, A. P., Strategies for Dynamic Load Balancing on Highly Parallel Computers. *IEEE T. Parall. Distr.* **1993**, *4* (9), 979-993.
47. Mitin, A. V.; Baker, J.; Wolinski, K.; Pulay, P., Parallel stored-integral and semidirect Hartree–Fock and DFT methods with data compression. *J. Comput. Chem.* **2003**, *24* (2), 154-160.
48. Dinan, J.; Krishnamoorthy, S.; Larkins, D. B.; Nieplocha, J.; Sdayappan, P. In *Scioto: A Framework for Global-View Task Parallelism*, 37th international Conference on Parallel Processing, Portland, USA, 9-12 September; Portland, USA, 2008; pp 586-593.
49. Neese, F.; Wennmohs, F.; Hansen, A.; Becker, U., Efficient, approximate and parallel Hartree–Fock and hybrid DFT calculations. A 'chain-of-spheres' algorithm for the Hartree–Fock exchange. *Chem. Phys.* **2009**, *356* (1-3), 98-109.
50. Guidon, M.; Schiffmann, F.; Hutter, J.; VandeVondele, J., Ab initio molecular dynamics using hybrid density functionals. *J. Chem. Phys* **2008**, *128*, 214104.
51. Calaminici, P.; Domíngues-Soria, V. D.; Geudtner, G.; Hernández-Marín, E.; Köster, A. M., Parallelization of three-center electron repulision integrals. *Theor. Chem. Acc.* **2006**, *115* (4), 221-226.
52. Gan, C. K.; Challacombe, M., Linear scaling computation of the Fock matrix. VI. Data parallel computation of the exchange-correlation matrix. *J. Chem. Phys* **2003**, *118* (20), 9128-9135.
53. Takashima, H.; Yamada, S.; Obara, S.; Kitamura, K.; Inabata, S.; Miyakawa, N.; Tanabe, K.; Nagashima, U., A Novel Parallel Algorithm for Large-Scale Fock Matrix Construction with Small Locally Distributed Memory Architectures: RT Parallel Algorithm. *J. Comput. Chem.* **2002**, *23* (14), 1337-1346.
54. Guest, M. F.; Bush, I. J.; van Dam, H. J. J.; Sherwood, P.; Thomas, J. M. H.; van Lenthe, J. H.; Havenith, R. W. A.; Kendrick, J., The GAMESS-UK electronic structure package: algorithms, developments and applications. *Mol. Phys.* **2005**, *103* (6-8), 719-747.
55. Dachsel, H.; Lischka, H.; Shepard, R.; Nieplocha, J.; Harrison, R. J., A massively parallel multireference configuration interaction program: The parallel COLUMBUS program. *J. Comput. Chem.* **1997**, *18* (3), 430–448.

56. Furlani, T. R.; Kong, J.; Gill, P. M. W., Parallelization of SCF calculations within Q-Chem. *Comput. Phys. Commun.* **2000**, *128* (1-2), 170-177.
57. Berenbrink, P.; Friedetzky, T.; Goldberg, L. A. In *The natural work-stealing algorithm is stable*, 42nd IEEE Symposium on Foundations of Computer Science (FOCS), 2001; pp 178-187.
58. Kumar, V.; Grama, A. Y.; Vempaty, N. R., Scalable load balancing techniques for parallel computers. *J. Parallel Distrib. Comput.* **1994**, *22* (1), 60-79.
59. Barrett, B. W.; Shipman, G. M.; Lumsdaine, A. In *Analysis of Implementation Options for MPI-2 One-Sided*, Euro PVM/MPI, Paris, France, Paris, France, 2007.
60. Leibniz-Rechenzentrum *2012*, National Supercomputer HLRB-II: SGI Altix 4700. (accessed 25.3.2013).
61. SGI, Message Passing Toolkit (MPT) User Guide. 2012.
62. Hewlett-Packard Development Company, HP-MPI User's Guide. 11th ed.; 2007.
63. ParTec Cluster Competence Center, ParaStation MPI 5.0. In *Software Product Detailed Description*, 2012.
64. Tatashina, A.; Knoth, A.; Friedley, A.; Angskun, T.; Benton, B.; Bosilca, G.; Bouteiller, A.; Barrett, B.; Casswell, L.; Cot, C.; Bell, C.; D'Amico, B.; Daniel, D.; Dimick, D.; Kerr, D.; Lacher, D.; Gabriel, E.; Mallove, E.; Fagg, G.; Young, G.; Natapov, G.; Shipman, G.; Watson, G.; Herault, T.; Stork, S.; Hoefler, T.; Mason, J.; Hursey, J.; Jan, N.; Squyres, J.; Jurenz, M.; Norteman, K.; Mroz, K.; Knuepfer, A.; Lemarinier, P.; Verkhovsky, L.; Lumsdaine, A.; Matney, K.; Sukalski, M.; Chaarawi, M.; Taylor, M.; Lo, L.-T.; Lui, P.; Shamis, P.; Geoffray, P.; Penoff, B.; Pjesivac-Grbovic, J.; Kambadur, P.; Rasmussen, C.; Brightwell, R.; Castain, R.; Graham, R.; Vandevaart, R.; Awles, R.; Keller, R.; Ayyorgun, S.; Santhanaraman, G.; Melamed, S.; Fan, S.; Sur, S.; Sharma, S.; Dontje, T.; Mattox, T.; Prins, T.; Woodall, T.; Sahay, V.; Yu, W. *openMPI*, 1.4.2; 2010.
65. Mattern, F., Global quiescence detection based on credit distribution and recovery. *Inf. Process. Lett.* **1989**, *30* (4), 195-200.
66. Latham, R.; Ross, R.; Thakur, R., Implementing MPI-IO Atomic Mode and Shared File Pointers Using MPI One-Sided Communication. *Int. J. High Perform. Comput. Appl.* **2007**, *21* (2), 132-143.
67. Leibniz-Rechenzentrum *2012*, Using MPI on SGI Altix Systems. (accessed 25.3.2013).
68. Leibniz-Rechenzentrum *2013*, Intel MPI. (accessed 25.3.2013).
69. Leibniz-Rechenzentrum *2013*, Overview of the cluster configuration. (accessed 25.3.2013).
70. Leibniz-Rechenzentrum *2012*, SuperMUC Petascale System. (accessed 25.3.2013).
71. Leibniz-Rechenzentrum *2012*, SGI Message Passing Toolkit. (accessed 25.3.2013).
72. Leibniz-Rechenzentrum *2010*, Prace prototype systems. (accessed 25.3.2013).
73. Slater, J. C., A Simplification of the Hartree-Fock Method. *Phys. Rev.* **1951**, *81* (3), 385-390.
74. Gáspár, R., Über eine Approximation des Hartree-Fockschen Potentials Durch eine Universelle Potentialfunktion. *Acta Physica Academiae Scientiarum Hungaricae* **1954**, *3* (3-4), 263-286.
75. Kohn, W.; Sham, L. J., Self-Consistent Equations Including Exchange and Correlation Effects. *Phys. Rev.* **1965**, *140* (4A), A1133-A1138.
76. Vosko, S. H.; Wilk, L.; Nusair, M., Accurate spin-dependent electron liquid correlation energies for local spin density calculations: a critical analysis. *Can. J. Phys.* **1980**, *58* (8), 1200-1211.
77. Belling, T. A parallel Implementation of the Density Functional Method. Integral Evaluation and External Potentials. Application to Thiolate Adsorption on Gold Surfaces. Dissertation, Technische Universität München, München, 1998.

78. Görling, A. Zur Verwendung von Dipolmomenten in der LCGTO-LDF-Methode. Dissertation, Technische Universität München, München, 1990.
79. Mayer, M. A Parallel Implementation of the Density Functional Method: Implementation of the Two-Component Douglas-Kroll-Hess Method and Application to Relativistic Effects in Heavy Element Chemistry. Dissertation, Technischen Universität München, München, 1999.
80. McMurchie, L. E.; Davidson, E. R., One- and Two-Electron Integrals over Cartesian Gaussian Functions. *J. Comput. Chem.* **1978**, *26* (2), 218-231.
81. Head-Gordon, M.; Pople, J. A., A method for two-electron Gaussian integral and integral derivative evaluation using recurrence relations. *J. Chem. Phys.* **1988**, *89*, 5777-5786.
82. Almlöf, J.; Faegri, K., Jr.; Korsell, K., Principles for a Direct SCF Approach to LCAO-MO Ab-lnitio Calculations. *J. Comput. Chem.* **1982**, *3* (3), 385-399.
83. Häser, M.; Ahlrichs, R., Improvements on the Direct SCF Method. *J. Comput. Chem.* **1989**, *10* (1), 104-111.
84. Tittle, D.,*personal communication*,Garching 2011
85. Dunlap, B. I., Three-center Gaussian-type-orbital integral evaluation using solid spherical harmonics. *Phys. Rev. A* **1990**, *42*, 1127-1137.
86. Dunlap, B. I., Angular momentum in molecular quantum mechanical integral evaluation. *Comp. Phys. Comm.* **2005**, *165*, 18-36.
87. Ashcroft, N. W.; Mermin, N. D., *Festkörkerphysik*. Oldenbourg Wissenschaftsverlag GmbH: München, 2005.
88. Matulis, V. E.; Ivashkevich, O. A.; Gurin, V. S., DFT study of electronic structure and geometry of anionic copper clusters. *J. Mol. Struct.: Theochem* **2004**, *681* (1-3), 169–176.
89. Itoh, M.; Kumar, V.; Adschiri, T.; Kawazoe, Y., Comprehensive study of sodium, copper, and silver clusters over a wide range of sizes $2 \leq N \leq 75$. *J. Chem. Phys.* **2009**, *131*, 174510.
90. Nörtemann, F. C. A Parallel Implementation of the Density Functional Method. SCF Part, Optimization Package and Application to Gold Clusters. Dissertation, Technische Universität München, München, 1988.
91. Pulay, P., Convergence Acceleration of Iterative Sequences. The Case of Scf Iteration. *Chem. Phys. Let.* **1980**, *73* (2), 393-398.
92. Hamilton, T. P.; Pulay, P., Direct Inversion in the Iterative Subspace (DIIS) Optimization of Open-shell, Excited-state, and Small Multiconfiguration SCF Wave Functions. *J. Chem. Phys* **1986**, *84*, 5728-5734.
93. Ehrenson, S., On Damping in Self-Consistency Cycling Procedures. *Theoret. chim. Acta (Berl.)* **1969**, *14* (2), 136-146.
94. Zerner, M. C.; Hehenberger, M., A Dynamical Damping Scheme for Converging Molecular Scf Calculations. *Chem. Phys. Let.* **1979**, *62* (3), 550-554.
95. Dunlap, B. I., Alternative perspective on density-functional perturbation theory. *Phys. Rev. A* **2007**, *76*, 062512.
96. Kudin, K. N.; Scuseria, G. E.; Cancès, E., A black-box self-consistent field convergence algorithm: One step closer. *J. Chem. Phys.* **2002**, *116* (19), 8255-8261.
97. Fukui, K., The Path of Chemical Reactions - The IRC Approach. *Acc. Chem. Res.* **1981**, *14* (12), 363-368.
98. Chen, M.; Cai, Z.-Z.; Yang, X.-B.; Zhu, M.; Zhao, Y.-J., Theoretical study of hydrogen dissociation and diffusion on Nb and Ni co-doped Mg(0001): A synergistic effect. *Surf. Sci.* **2012**, *606*, L45-L49.
99. Moras, G.; Pastewka, L.; Walter, M.; Schnagl, J.; Gumbsch, P.; Moseler, M., Progressive Shortening of sp-Hybridized Carbon Chains through Oxygen-Induced Cleavage. *J. Phys. Chem. C* **2011**, *115* (50), 24653-24661.

100. de la Hoz, Julibeth M. Martínez; Balbuena, P. B., Geometric and Electronic Confinement Effects on Catalysis. *J. Phys. Chem. C* **2011**, *115*, 21342-21333.
101. Kanai, Y.; Takeuchi, N.; Car, R.; Selloni, A., Role of Molecular Conjugation in the Surface Radical Reaction of Aldehydes with H-Si(111): First Principles Study. *J. Phys. Chem. B* **2005**, *109* (40), 18889-18894.
102. Ozbek, M. O.; Onal, I.; van Santen, R.A., Why silver is the unique catalyst for ethylene epoxidation. *J. Catal.* **2011**, *284* (2), 230-235.
103. Lin, S.; Xie, D.; Guo, H., First-principles study of the methyl formate pathway of methanol steam reforming on PdZn(1 1 1) with comparison to Cu(1 1 1). *J. Mol. Catal. A: Chem.* **2012**, *356*, 165-170.
104. Peng, G.; Sibener, S. J.; Schatz, G. C.; Mavrikakis, M., CO2 hydrogenation to formic acid on Ni(110). *Surf. Sci.* **2012**, *606* (13), 1050-1055.
105. Chin, Y.-H.; Buda, C.; Neurock, M.; Iglesia, E., Reactivity of Chemisorbed Oxygen Atoms and Their Catalytic Consequences during CH4-O2 Catalysis on Supported Pt Clusters. *J. Am. Chem. Soc.* **2011**, *133*, 15958-15978.
106. Fajín, J., L.C.; Cordeiro, M. N. D. S.; Illas, F.; Gomes, J., R.B., Influence of step sites in the molecular mechanism of the water gas shift reaction catalyzed by copper. *J. Catal.* **2009**, *268* (1), 131-141.
107. Cao, X.-M.; Burch, R.; Hardacre, C.; Hu, P., An understanding of chemoselective hydrogenation on crotonaldehyde over Pt(1 1 1) in the free energy landscape: The microkinetics study based on first-principles calculations. *Catal. Today* **2011**, *165* (1), 71-79.
108. Liu, B.; Greeley, J., Decomposition Pathways of Glycerol via C-H, O-H, and C-C Bond Scission on Pt(111): A Density Functional Theory Study. *J. Phys. Chem. C* **2011**, *115*, 19702-19709.
109. Xu, L.; Xu, Y., Effect of Pd surface structure on the activation of methyl acetate. *Catal. Today* **2011**, *165*, 96-105.
110. Zhao, Z.-J.; Moskaleva, L. V.; Aleksandrov, H. A.; Basaran, D.; Rösch, N., Ethylidyne Formation from Ethylene over Pt(111): A Mechanistic Study from First-Principle Calculations. *J. Phys. Chem. C* **2010**, *114*, 12190-12201.
111. Zhao, Z.-J.; Moskaleva, L. V.; Rösch, N., Tuning the selectivity for ring-opening reactions of methylcyclopentane over Pt catalysts: A mechanistic study from first-principles calculations. *J. Catal.* **2012**, *285* (1), 124-133.
112. Basaran, D.; Genest, A.; Rösch, N., Comment on "Towards understanding the bifunctional hydrodeoxygenation and aqueous phase reforming of glycerol"[J. Catal. 269 (2010) 411-420]. *J. Catal.* **2012**, *287*, 210-213.
113. Moskaleva, L. V.; Aleksandrov, H. A.; Basaran, D.; Zhao, Z.-J.; Rösch, N., Ethylidyne Formation from Ethylene over Pd(111): Alternative Routes from a Density Functional Study. *J. Phys. Chem. C* **2009**, *113*, 15373-15379.
114. Jónsson, H.; Mills, G.; Jacobsen, K. W., Classical and Quantum Dynamics in Condensed Phase Simulations. Berne, B. J.; Ciccotti, G.; Coker, D. F., Eds. World Scientific: 1998.
115. Henkelman, G.; Jónsson, H., Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *J. Chem. Phys.* **2000**, *113*, 9978-9985.
116. Weinan, E.; Ren, W.; Vanden-Eijnden, E., String method for the study of rare events. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2002**, *66* (5), 052301.
117. Burger, S. K.; Yang, W., Quadratic string method for determining the minimum-energy path based on multiobjective optimization. *J. Chem. Phys.* **2006**, *124* (5), 054109.
118. Chaffey-Millar, H.; Nikodem, A.; Matveev, A. V.; Krüger, S.; Rösch, N., Improving Upon String Methods for Transition State Discovery. *J. Chem. Theory Comput.* **2012**, *8* (2), 777-786.

119. Quapp, W., A growing string method for the reaction pathway defined by a Newton trajectory. *J. Chem. Phys.* **2005**, *122*, 174106.
120. Peters, B.; Heyden, A.; Bell, A. T.; Chakraborty, A., A growing string method for determining transition states: Comparison to the nudged elastic band and string methods. *J. Chem. Phys.* **2004**, *120*, 7877-7886.
121. Goodrow, A.; Bell, A. T.; Head-Gordon, M., Development and application of a hybrid method involving interpolation and ab initio calculations for the determination of transition states. *J. Chem. Phys.* **2008**, *129*, 174109.
122. Goodrow, A.; Bell, A. T.; Head-Gordon, M., Transition state-finding strategies for use with the growing string method. *J. Chem. Phys.* **2009**, *130*, 244108.
123. Henkelman, G.; Uberuaga, B. P.; Jónsson, H., A climbing image nudged elastic band method for finding saddle points and minimum energy paths. *J. Chem. Phys.* **2000**, *113*, 9901-9904.
124. Weinan, E.; Ren, W.; Vanden-Eijnden, E., Simplified and improved string method for computing the minimum energy paths in barrier-crossing events. *J. Chem. Phys.* **2007**, *126* (16), 164103.
125. Peng, C.; Schlegel, H. B., Combining Synchronous Transit and Quasi-Newton Methods to Find Transition States. *Israel J. Chem.* **1993**, *33*, 449-454.
126. Poppinger, D., On the Calculation of Transition States. *Chem. Phys. Lett.* **1975**, *35* (4), 550-554.
127. Baker, J., An Algorithm for the Location of Transition States. *J. Comput. Chem.* **1986**, *7*, 385-395.
128. Cerjan, C. J.; Miller, W. H., On finding transition states. *J. Chem. Phys.* **1981**, *75*, 2800-2806.
129. Wang, H.-F.; Liu, Z.-P., Comprehensive Mechanism and Structure-Sensitivity of Ethanol Oxidation on Platinum: New Transition-State Searching Method for Resolving the Complex Reaction Network. *J. Am. Chem. Soc.* **2008**, *130* (33), 10996-11004.
130. Shang, C.; Liu, Z.-P., Constrained Broyden Minimization Combined with the Dimer Method for Locating Transition State of Complex Reactions. *J. Chem. Theory Comput.* **2010**, *6* (4), 1136-1144.
131. Henkelman, G.; Jónsson, H., A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives. *J. Chem. Phys.* **1999**, *111* (15), 7010-7022.
132. Heyden, A.; Bell, A. T.; Keil, F. J., Efficient methods for finding transition states in chemical reactions: Comparison of improved dimer method and partitioned rational function optimization method. *J. Chem. Phys.* **2005**, *123* (22), 224101.
133. Kästner, J.; Sherwood, P., Superlinearly converging dimer method for transition state search. *J. Chem. Phys.* **2008**, *128*, 014106.
134. Atomic Simulation Environment Website, https://wiki.fysik.dtu.dk/ase, last viewed: 20 August 2010.
135. Goodrow, A.; Bell, A. T.; Head-Gordon, M., A strategy for obtaining a more accurate transition state estimate using the growing string method. *J. Chem. Phys.* **2010**, *484*, 392-398.
136. Nocedal, J.; Wright, S. J., *Numerical Optimization*. second ed.; Springer: New York, 2006.
137. Olsen, R. A.; Kroes, G. J.; Henkelman, G.; Analdsson, A.; Jónsson, H., Comparison of methods for finding saddle points without knowledge of the final states. *J. Chem. Phys.* **2004**, *121* (20), 9777-9792.
138. Polak, E.; Ribiere, G., Note sur la convergence de methodes de directions conjuguees. *Revue française d'informatique et de recherche opérationnelle, série rouge* **1969**, *3* (1), 35-43.

139. Powell, M. J. D., Convergence Properties of Algorithms for Nonlinear Optimization. *SIAM Review* **1986**, *28* (4), 487-500.
140. Lanczos, C., An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators. *J. Res. Natl. Bur. Stand. (U. S.)* **1950**, *45* (4), 255-282.
141. Malek, R.; Mousseau, N., Dynamics of Lennard-Jones clusters: A characterization of the activation-relaxation technique. *Phys. Rev. E* **2000**, *62*, 7723-7728.
142. Sheppard, D.; Terrell, R.; Henkelman, G., Optimization methods for finding minimum energy paths. *J. Chem. Phys.* **2008**, *128*, 134106.
143. Henkelman, G.; Jóhannesson, G.; Jónsson, H., Methods for Finding Saddle Points and Minimum Energy Paths. In *Progress on Theoretical Chemistry and Physics*, Schwartz, S. D., Ed. Kluwar Academic Publishers: Dordrecht, Netherlands, 2000; pp 269-300.
144. Bitzek, E.; Koskinen, P.; Gähler, F.; Moseler, M.; Gumbsch, P., Structural Relaxation Made Simple. *Phys. Rev. Lett.* **2006**, *97*, 170201.
145. Baker, J., Techniques for Geometry Optimization: A Comparison of Cartesian and Natural Internal Coordinates. *J. Comput. Chem.* **1993**, *14* (9), 1085-1100.
146. Fogarasi, G.; Zhou, X.; Taylor, P. W.; Pulay, P., The Calculation of ab Initio Molecular Geometries: Efficient Optimization by Natural Internal Coordinates and Empirical correction by Offset Forces. *J. Am. Chem. Soc.* **1992**, *114*, 8191-8201.
147. Müller, K.; Brown, L. D., Location of Saddle Points and Minimum Energy Paths by a Constrained Simplex Optimization Procedure. *Theor. Chim. Acta* **1979**, *53*, 75-93.
148. Bollhöfer, M.; Mehrmann, V., *Numerische Mathematik: Eine Projektorientierte Einführung für Ingeniuere, Mathematiker und Naturwissenschaftler* Vieweg + Teubner Verlag: Wiesbaden, 2004.
149. Brown, K., Tensors, Contravariant and Covariant. In *Reflections on Relativity*, Lulu.com Raleigh, 2012.
150. Fukui, K., A Formulation of the Reaction Coordinate. *J. Phys. Chem.* **1970**, *74* (23), 4161-4163.
151. Walsh, T. R.; Wales, D. J., Comparison of reaction pathways calculated by different algorithms
for disilane and water trimer. *Z. Phys. D* **1997**, *40*, 229-235.
152. Sana, M.; Reckinger, G.; Leroy, G., An Internal Coordinate Invariant Reaction Pathway. *Theor. Chim. Acta* **1981**, *58* (2), 145-153.
153. Quapp, W.; Heidrich, D., Analysis of the concept of minimum energy path on the potential energy surface of chemically reacting systems. *Theor. Chim. Acta* **1984**, *66*, 245-260.
154. Mezey, P. G., Reactive Domains of Energy Hypersurfaces and the Stability of Minimum Energy Reaction Paths. *Theor. Chim. Acta* **1980**, *54*, 95-111.
155. Mortimer, R. G., *Physical Chemistry*. third ed.; Elsevier Academic Press: Burlington, USA, 2008.
156. Schlegel, H. B., Exploring Potential Energy Surfaces for Chemical Reactions: An Overview of Some Practical Methods. *J. Comput. Chem.* **2003**, *24* (12), 1514-1527.
157. Hratchian, H. P.; Schlegel, H. B., Finding minima, transition states, and following reaction pathways on ab initio potential energy surfaces. In *Theory and Applications of Computational Chemistry: The First Forty Years*, first ed.; Dykstra, C. E.; Frenking, G.; Kim, K. S.; Scuseria, G. E., Eds. Elsevier: Amsterdam, Netherlands, 2005; pp 195-251.
158. Martínez, J. M., de la Hoz; Balbuena, P. B., Geometric and Electronic Confinement Effects on Catalysis. *J. Phys. Chem. C* **2011**, *115* (43), 21324-21333.
159. Jackson, B.; Nave, S., The dissociative chemisorption of methane on Ni(100): Reaction path. *J. Chem. Phys.* **2011**, *135*, 114701.

160. Zhao, P.; Su, Y.; Zhang, Y.; Li, S.-J.; Chen, G., CO catalytic oxidation on iron-embedded hexagonal boron nitride sheet. *Chem. Phys. Lett.* **2011**, *515*, 159-162.
161. Yang, Z.; Zhang, Y.; Wang, J.; Ma, S., First-principles study on the Ni@Pt12 Ih core--shell nanoparticles: A good catalyst for oxygen reduction reaction. *Phys. Lett. A* **2011**, *375* (35), 3142-3148.
162. Klimeš, J.; Bowler, D. R.; Michaelides, A., A critical assessment of theoretical methods for finding reaction pathways and transition states of surface processes. *J. Phys.: Condens. Matter* **2010**, *22*, 074203.
163. Heyden, A.; Peters, B.; Bell, A. T.; Keil, F. J., Comprehensive DFT Study of Nitrous Oxide Decomposition over Fe-ZSM-5. *J. Phys. Chem. B* **2005**, *109*, 1857-1873.
164. Nikodem, A.; Chaffey-Millar, H.; Matveev, A.; Rösch, N. *ParaTools* ParaTools 2.0; Technische Universität München: München, 2012.
165. Kresse, G.; Hafner, J., Ab initio molecular-dynamics simulation of the liquid-metalamorphous- semiconductor transition in germanium. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1994**, *49* (20), 14251-14269.
166. Kresse, G.; Furthmüller, J., Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Comput. Mater. Sci.* **1996**, *6* (1), 15-50.
167. Perdew, J. P.; Wang, Y., Accurate and simple analytic representation of the electron-gas correlation energy. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1992**, *45*, 13244-13249.
168. Kresse, G.; Joubert, D., From ultrasoft pseudopotentials to the projector augmented-wave method. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1999**, *59* (3), 1758-1775.
169. Alcalá, R., DFT studies for cleavage of C-C and C-O bonds in surface species derived from ethanol on Pt(111). *J. Catal.* **2003**, *218*, 178-190.
170. Jasper, A. W.; Klippenstein, S. J.; Harding, L. B.; Ruscic, B., Kinetics of the Reaction of Methyl Radical with Hydroxyl Radical and Methanol Decomposition. *J. Phys. Chem. A* **2007**, *111*, 3932-3950.
171. Altmann, S. L., *Rotations, Quaternions and Double Groups*. Oxford University Press: New York, USA, 1986.
172. The IEEE and The Open Group, The Open Group Base Specifications Issue 6 2004; Vol. IEEE Std 1003.1, 2004 Edition.
173. Huzinaga, S., GTO basis sets for heavier elements. *J. Chem. Phys.* **1977**, *66*, 4245.
174. Poirier, R.; Kari, R.; Csizmadia, I. G., *Handbook of Gaussian basis sets*. Elsevier: New York, 1985.
175. Krishnan, R.; Binkley, J. S.; Seeger, R.; Pople, J. A., Self-consistent molecular orbital methods. XX. A basis set for correlated wave functions. *J. Chem. Phys.* **1980**, *72* (1), 650-654.
176. Clark, T.; Chandrasekhar, J.; Spitznagel, G. W.; von Ragué Schleyer, P., Efficient Diffuse Function- Augmented Basis Sets for Anion Calculations. III.* The 3-21+G Basis Set for First-Row Elements, Li-F. *J. Comput. Chem.* **1983**, *4* (3), 294-301.
177. Frisch, M. J.; Pople, J. A.; Binkley, J. S., Selfconsistent molecular orbital methods 25. Supplementary functions for Gaussian basis sets. *J. Chem. Phys.* **1984**, *80*, 3265-3269.
178. Huzinaga, S.; Klobukowski, M., Well-tempered Gaussian basis sets for the calculation of matrix Hartree-Fock wavefunctions. *Chem. Phys. Let.* **1993**, *212* (3,4), 260-264.
179. Huzinaga, S.; Miguel, B., A comparison of the geometrical sequence formula and the well-tempered formulas for generating GTO basis orbital exponents. *Chem. Phys. Let.* **1990**, *175* (4), 289-291.
180. Weigend, F.; Ahlrichs, R., Balanced basis sets of split valence, triple zeta valence and quadruple zeta valence quality for H to Rn: Design and ssessment of accuracy. *Phys. Chem. Chem. Phys.* **2005**, *7*, 3297-3305.

181. Eichkorn, K.; Weigend, F.; Treutler, O.; Ahlrichs, R., Auxiliary basis sets for main row atoms and transition metals and their use to approximate Coulomb potentials. *Theor. Chem. Acc.* **1997**, *97*, 119-124.
182. Dolg, M.; Stoll, H.; Flad, H.-J.; Preuss, H., Ab initio pseudopotential study of Yb and YbO. *J. Chem. Phys.* **1992**, *97*, 1162-1173.
183. Van Duijneveldt, F. B. *Gaussian Basis Sets for the Atoms H-Ne for Use in Molecular Calculations*; 1971.
184. Huzinaga, S.; Andzelm, J.; Klobukowski, M.; Radzio-Andzelm, E.; Sakai, Y.; Tatewaki, H., *Gaussian Basis Sets for Molecular Calculations*. Elsevier: Amsterdam, 1984.
185. Frisch, M. J.; Pople, J. A.; Binkley, J. S., Selfconsistent molecular orbital methods 25. Supplementary functions for Gaussian basis sets. *J. Chem. Phys.* **1984**, *80* (7), 3265-3269.
186. Minami, T.; Matsuoka, O., Relativistic Gaussian basis sets for radon through plutonium. *Theor. Chim. Acta* **1995**, *90*, 27-39.
187. Roos, B. O.; Lindh, R.; Malmqvist, P.-Å.; Veryazov, V.; Widmark, P.-O., New relativistic ANO basis sets for actinide atoms. *Chem. Phys. Let.* **2005**, *409*, 295-299.
188. Beridze, G. A relativistic density functional study on americium aqua- and acetate complexes. Master's Thesis, Technische Universität München, München, 2012.