Enabling Large Scale DFT Simulation with GPU Acceleration and Machine Learning

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

Ole Schütt

Dipl.-Phys. Dipl.-Inform., FU Berlin

born on: 09.04.1985

citizen of Germany

accepted on the recommendation of
Prof. Dr. Joost VandeVondele
Prof. Dr. Thomas Schulthess
Prof. Dr. Jürg Hutter

2016
Abstract

The rapid increase in computational power and the development of linear scaling methods now allow for single-energy DFT calculations of systems with 10,000 - 1,000,000 atoms. However, the approach is computationally demanding for routine application, especially if first principles molecular dynamics or relaxation is required. In this work, two complementary approaches were developed to boost the performance of large scale DFT calculations. First, the predominant operation, sparse matrix-matrix multiplication, was ported to GPU accelerators. Afterwards, the performance was further increased employing geometry adapted basis sets obtained with machine learning.

For an increasing number of supercomputers, the majority of computing power is provided by accelerator devices, including CSCS’s current flagship Piz Daint. To take full advantage of the GPU’s high throughput parallelism within the established and highly MPI+OpenMP-parallel CP2K code base required significant algorithmic development. In particular, to accommodate for the GPU’s asynchronous task based programming model, the matrix multiplication code in CP2K’s sparse matrix library DBCSR was redesigned from the ground up. It was transformed from a lock-step scheme into a barrier free algorithm that relies solely on Cuda streams and events, as well as asynchronous MPI calls. This software architecture allows for fully overlapping the double-buffered Cannon’s algorithm with host-to-device copies, CPU book-keeping work, and Cuda kernel execution. Additionally, Cuda kernels for representative matrix sizes were developed together with Nvidia engineers as part of a co-design effort. These kernels served as templates for the auto-tuning of over 2350 kernels for domain specific matrix-sizes. For example, the single-node performance for the important 23 matrix size, comparing GPU+CPU vs CPU-only, was improved by 450%. As a consequence, sparse matrix-matrix multiplication has become an important primitive also for other methods in CP2K.
The second part of this work exploits the observation that the computational cost of a DFT calculation critically depends on the size and condition number of the employed basis set. Here, a small but optimal basis set is obtained by adapting an atom centered basis set to its chemical environment. To avoid a costly and cumbersome optimization for each molecular geometry, a machine learning approach was employed to infer the optimal basis from a limited set of training points. To make this possible, the basis is derived from the molecular geometry in a way that is invariant to rotations of the system and permutations of the atoms. These basis functions are obtained as the lowest energy states of an isolated atom perturbed by a molecular geometry dependent, and machine learned potential. The machine learning maps a fingerprint of the local geometry to optimal parameters for the potential parametrization. A key solved problem was obtaining consistent potentials, thus suitable for learning, which required regularization and effective optimization. The method has been demonstrated using MD simulations of liquid water, where it is shown that minimal basis sets yield structural and dynamical properties that outperform those of semi-empirical methods at similar cost. Compared to the standard approach, the number of flops needed in matrix-matrix multiplications decreased by over four orders of magnitude, leading to an effective 60-fold run-time speedup.
Zusammenfassung

Durch den rapiden Anstieg der verfügbaren Rechenleistung sowie der Entwicklung von linear skalierenden Methoden können Einzelenergien von 10.000 - 1.000.000 Atomen mit DFT berechnet werden. Allerdings ist auch dieser Ansatz für die alltägliche Anwendung sehr rechenintensiv, insbesondere dann, wenn ab-initio Molekularodynamik und Äquilibrierungen benötigt werden. In dieser Arbeit wurden zwei sich ergänzende Ansätze entwickelt, um die Leistungsfähigkeit von grossskaligen DFT-Berechnungen zu verbessern. Zuerst wurde die dominierende Operation, nämlich die Multiplikation von dünnbesetzten Matrizen, auf GPU-Beschleuniger portiert. Anschließend wurde die benötigte Rechenzeit durch einen an die Geometrie angepassten und auf maschinellem Lernen basierenden Basissatz weiter verkürzt.

die Einzelknotenleistung der wichtigen Matrizengröße 23, im Vergleich von einer alleinigen CPU zu GPU+CPU, um 450% verbessert. Als Folge davon ist die Multiplikation von dünnbesetzten Matrizen eine wichtige Primitive auch für andere Methoden in CP2K geworden.

First of all, I would like to express my sincere gratitude to Prof. Joost VandeVondele for his continuous support of my Ph.D research and related projects, for his patience, motivation, and inspiring knowledge.

I would also like to thank Prof. Jürg Hutter for his trust and support for my ideas on CP2K, and for being co-examiner. Furthermore, I would like to thank Prof. Thomas Schulthess for co-examining.

I’m grateful for the numerous insightful discussions with the members of the Nanoscale Simulations group, which I enjoyed very much. And I want to thank in particular Susanne Blatter for her always very kind help and support.

This project would not have been possible without the excellent support on Cuda from Dr. Peter Messmer and the resources from the Swiss National Supercomputing Centre (CSCS) that were provided under the project IDs s441 and ch5.

Finally, I would like to thank my family, friends, and in particular Katharina Mellert for their love and support.
Contents

Abstract i
Zusammenfassung iii
Acknowledgement v

Contents vii

1 Introduction 1
1.1 Why Linear Scaling DFT? ......................... 3
1.2 Why GPUs? .................................. 5
1.3 Why Machine Learning? ......................... 7

2 Linear Scaling DFT 11
2.1 Schrödinger Equation ........................... 11
2.2 Born-Oppenheimer Approximation ............... 12
2.3 Hohenberg-Kohn Theorems ....................... 12
2.4 Kohn-Sham Ansatz ................................ 14
2.5 Finite Basis Sets ................................ 15
2.6 Exploiting Nearsightedness ..................... 17
2.7 Gaussian and Plane Waves Method ............... 19
2.8 McWeeny Purification ........................... 21
2.9 Trace Resetting Purification ..................... 22
2.10 Curvy-Step Method ............................ 23

3 GPU Accelerated Sparse Matrix Matrix Multiplications 25
3.1 DBCSR: A Sparse Matrix Library ............... 25
3.2 Software Architecture for GPU-Acceleration .... 26
  3.2.1 Cannon Layer ................................ 26
  3.2.2 Multrec Layer .............................. 27
  3.2.3 CSR Layer ................................ 28
Contents

3.2.4 Scheduler and Driver Layers ........................................... 28
3.3 Maximizing Asynchronous Progress .................................... 29
  3.3.1 Cuda Streams and Events ........................................... 29
  3.3.2 Double Buffered Cannon on Host and Device ...................... 31
3.4 Libcusmm: GPU Accelerated Small Matrix Multiplications ......... 33
  3.4.1 Small Matrix Multiplication Performance Model .................. 34
  3.4.2 Matrix-Product Algorithm Choice ................................ 36
  3.4.3 GPU Implementation: Generic Algorithm .......................... 36
  3.4.4 Auto-Tuning and Performance ..................................... 38
3.5 Benchmarks and Conclusions ........................................... 38

4 Machine Learning Adaptive Basis Sets ................................. 43
  4.1 Introduction ............................................................ 43
  4.2 Polarized Atomic Orbitals ............................................. 45
    4.2.1 PAO Basis Sets .................................................. 45
    4.2.2 PAO Optimization ............................................... 47
    4.2.3 PAO Gradient .................................................... 49
    4.2.4 Exponential Parameterization ................................. 50
    4.2.5 Potential Parameterization ................................... 51
  4.3 Machine Learning ...................................................... 57
    4.3.1 Forces ............................................................ 58
    4.3.2 Gaussian Processes ............................................ 59
    4.3.3 Descriptor ....................................................... 60
  4.4 Results ................................................................. 61
    4.4.1 Water Dimer ..................................................... 61
    4.4.2 Water Dimer Trajectory ....................................... 64
    4.4.3 Consistency of Energy and Forces ............................. 66
    4.4.4 PAO Basis Sets for Liquid Water ............................ 67
    4.4.5 PAO-ML Energies for Liquid Water .......................... 69
    4.4.6 PAO-ML Molecular Dynamics of Liquid Water ................. 71
    4.4.7 PAO-ML Speedup ................................................ 74
    4.4.8 TiO$_2$ Nanoparticles in Explicit Acetonitrile Solution .... 76
    4.4.9 Computational Setup ......................................... 79
  4.5 Discussion and Conclusions ......................................... 80

5 Conclusion and Outlook .................................................. 83

A Future-Proofing CP2K ................................................... 87

Bibliography ................................................................. 95

viii
Computer simulations play an important role in today’s research and development of novel materials. This is emphasized by the fact that 12 out of the top-100 most cited paper across all scientific fields are related to density functional theory (DFT), including two among the top-10 [1]. Simulation programs, such as the CP2K package, are powerful tools that provide insights to experimental measurements, predict material properties, and aid the rational design of new compounds [2, 3]. In the following, two examples will be presented that are representative for the requirements imposed upon state-of-the-art materials simulations.

**Figure 1.1:** Computational investigation of 2D polymer structure. left: Lateral disorder in 2D polymer after 4 ps of DFTB molecular dynamics (oxygen red, carbon green). middle: Fluctuations in height visualized by displaying only those groups that protrude through a horizontal plane, shaded by the density of nearby atoms. right: Chemical structure of the monomer. Graphics from Payamyar, Schlüter et al. [4].

The first example is from the field of 2D Polymers [5]. Two-dimensional materials are expected to have many possible applications, including electronics, gas storage and separation, catalysis, high performance sensors, support membranes, and inert coatings [6]. Recently, a method for the synthesis of a covalent monolayer sheet at the air/water interface using photochemical polymerization was presented by Payamyar, Schlüter et al. [4]. They mea-
1. Introduction

Measured the mechanical properties of their synthesized sheets using atomic force microscopy. However, elucidating the internal structure of these polymers was more challenging, because the material is too thin for analysis with standard techniques such as X-ray diffraction or transmission electron microscopy. Therefore, a large scale DFT simulation was performed to investigate the conformational stability of the polymer. Since the precise structure was unknown at the time, the simulation was initialized with a large super cell of an idealized geometry. This initial structure was geometry optimized and subsequently equilibrated at ambient conditions using molecular dynamics. It was found that the sheet undergoes conformational changes that lead to a reduced lateral order and a densification by 25%. Figure 1.1 shows the final configuration that was obtained after 4 ps of molecular dynamics. This deformation process was previously unknown and hence the simulations made an important contribution to the structure elucidation of this 2D polymer. The entire simulation was composed of over 25k atoms and required on the order of 8k single point calculations. Since the simulations had no emphasis on electronic properties, the tight binding approximation (DFTB) could be employed. The simulation was performed using CP2K’s linear scaling implementation [7] and took advantage of the novel GPU acceleration that was developed as part of this thesis [8]. The calculations were made feasible through the DFTB approximation and the 2D nature of the system. In the following example, these two advantages are not available.

Figure 1.2: Left: Schematic of a DSSC: A photo-excited dye molecule injects the exited electron into the TiO$_2$ nanoparticles. The remaining electron hole gets regenerated by I$^-$ ions that diffuse within the surrounding acetonitrile solvent. Middle: Scanning electron micrograph of the surface of the mesoporous TiO$_2$ material. Right: Realistic atomistic model of two TiO$_2$ nanoparticles. Left and middle graphics from Grätzel [9].

The second example is from the field of emerging photovoltaics. Novel solar cell designs offer the prospect of scalable roll-to-roll production, unprecedented power conversion efficiencies, and high stability [10]. While there are a number of different cell designs currently under research, they all have in common a disordered, complex structure of soft matter. This makes them very challenging to simulate: The focus on electronic properties requires
1.1. Why Linear Scaling DFT?

With the steady increase in computer power available, larger and larger systems are simulated using electronic structure methods. These large simulations expose the asymptotic scaling of the traditional algorithms used in electronic structure calculations. Many of the traditional algorithms scale cubically or worse with system size, effectively blocking the path to very large scale simulations. The reason lays in the required orthonormality of the Kohn-Sham orbitals, which is needed to fulfill the Pauli exclusion principle. If the orbitals span the entire space, even just verifying the orthonormality of all $N^2$ pairs of orbitals has a cost of $O(N^3)$. On the contrary, it is intuitive that electrons in most systems do not extend over long distances, which was coined as the nearsightedness of electronic matter by Walter Kohn [14].

Formally, nearsightedness corresponds to the decay of the reduced density matrix $P$. This matrix can be interpreted as a one-electron correlation function, i.e. the probability of an electron to be simultaneously present at two points in space:

$$ P(r, r') = N \int \Psi^+(r, r_2 \ldots r_N) \Psi(r', r_2 \ldots r_N) \, dr_2 \ldots dr_N $$
1. Introduction

A very illustrative example for the density matrix decay can be obtained from the one-dimensional Hückel theory applied to a chain of Hydrogen atoms [15]. Starting from a simple Hamiltonian $H$, diagonalization yields the molecular orbitals $\phi_\alpha$. The reduced density matrix $P$ is then constructed from the occupied, i.e. lower half, of the molecular orbitals:

$$P_{ij} = \sum_\alpha \phi_\alpha^*(r_i) \phi_\alpha(r_j)$$

$$H = \begin{pmatrix} 0 & c & c \\ c & 0 & 1 \\ 1 & 0 & c \\ c & 0 & \ddots \end{pmatrix}$$

Metal ($c=1$)
Polyomial decay

Isolator ($c=0.5$)
Exponential decay

Figure 1.3: A chain of Hydrogen atoms described with Hückel theory. Left: Polynomial density matrix decay for metallic system of equidistant atoms. Right: Exponential density matrix decay for isolating system of hydrogen molecules.

Figure 1.3 shows one row $P(0,r)$ of the reduced density matrix for two different values of the coupling constant $c$. Placing the hydrogen atoms equidistant ($c = 1$), leads to a metallic chain and the density matrix decay is polynomial. When instead $H_2$ molecules are formed ($c = 0.5$), the system becomes isolating and the density matrix decays exponentially.

The rapid decay of the density matrix, in particular for isolators, can be exploited to construct methods, which scale linear in system size, beyond a certain threshold-size [16, 17]. Originally, most applications and benchmarks of these methods were restricted to systems with a quasi one-dimensional structure, were the prefactor of linear scaling methods is very favorable. Now, the huge increase in computational power and the refinement of the algorithms, have made it possible to study scientifically relevant two- and three-dimensional systems. Basis sets of good quality and tight numerical thresholds can be employed, essentially allowing for an accuracy that is identical to that of the cubically scaling methods. Figure 1.4 shows a com-
1.2 Why GPUs?

In 1965 Gordon Moore made the observation that the number of transistors in a cost-optimal integrated circuit doubles approximately every two years [18]. It was later coined Moore’s law and became a self-fulfilling prophecy that has driven the semiconductor industry ever since. In 1975 Dennard et al. pointed out that a reduction of transistor dimensions is accompanied by a reduction of the required switching voltage and current [19]. This so called Dennard scaling allowed for an increase of the clock speed along with the miniaturization of the circuits. In combination, Moore’s law and Dennard scaling gave rise to a consistent exponential increase of computer’s single-thread performance for over four decades. However, starting with the 90 nm processor generation, leakage power became a significant constraint and consequently Dennard scaling came to an end around 2004 [20]. Nevertheless,
1. Introduction

The transistor size continued to shrink, still following Moore’s Law. With the clock speed maxed out at around 4 GHz, other means had to be found for translating the increased transistor density into improved application performance.

More transistors allowed for larger caches and novel instruction sets, but eventually the industry had to turn to multi-core processors [21]. The approach taken by traditional CPU manufactures is to replicate their existing serial x86 architecture into multiple cores. The bottleneck of the x86 architecture is its cache coherence semantic, which guarantees that caches remain “invisible” to the programmers. While cache coherence is relatively simple to implement on a single core, in a multi-core system it leads to significant inter-core communication overhead [22]. As a consequence the x86 architecture will be difficult to scale beyond double-digit core counts.

Another important design decision for processors is the ratio between caches and arithmetic logic units (ALUs). ALUs are the circuits that perform the actual computation, e.g. multiplying two floating-point numbers. Before an ALU can perform its computation the operands have to be loaded into its registers. Since loads from global DRAM memory take on the order of hundreds of clock cycles, an ALU can easily idle predominantly while waiting for operands to arrive. For this reason CPUs are equipped with caches and sophisticated pre-fetching logic. Via deep pipe-lining and branch-prediction heuristics the CPU tries to anticipate future load instructions and issues them in advance. Therefore, caches are crucial for the good performance of many algorithms. However, algorithms with a high arithmetic intensity or a predictable memory access pattern do not profit from caches in the same way. A high arithmetic intensity means that load instructions are less frequent compared to floating point instructions. And a predictable memory access pattern means that the responsibility for the pre-fetching can be handed off to the programmer.

![Figure 1.5: Comparison of CPUs with GPUs regarding their share of transistors devoted to cache memory, flow control circuits, and arithmetic logic units. Graphics from Nvidia [23].](image-url)
A prominent class of algorithms that are compute intense and predictable in memory access originate from the field of 3D computer graphics. Graphics processing units (GPUs), whose primary purpose is the acceleration of these 3D computer graphics application, are designed accordingly. GPUs have, compared to CPUs, much less cache memory and it comes without automatic coherence mechanism. It is upon the programmers responsibility to make changes in global memory visible to other GPU cores by issuing explicit memory fence instructions. The amount of control circuits in GPUs has been further reduced through a small instruction set and by taking advantage of the repetitive single-instruction multiple-data (SIMD) nature of computer graphics algorithms. Nvidia GPUs treat 32 threads as a vector unit, called *wrap*, which executes simultaneously the same stream of instructions. All together, these design decisions free up a lot of the chip’s surface for additional ALUs. Figure 1.5 shows a comparison of the amount of transistors devoted to certain functionalities by GPUs and CPUs. As a consequence of their large number of ALUs, GPUs can run thousands of threads simultaneously and reach much higher floating-point performances than CPUs. For example, the Tesla K20X has a theoretical double-precision peak-performance of 1.3TFlop/s, while a SandyBridge CPU (Intel Xeon E5-2670) from the same time reaches only 0.2 TFlop/s. Althoug the K20X GPU requires about twice as much power as the SandyBridge CPU (235W vs 115W), it still delivers a very favorable Flop/Watt performance that makes it attractive for HPC applications. This has led to the appearance of many hybrid CPU+GPU systems in the Top500 and in particular the Green500 list, including Switzerland’s flagship system *Piz Daint* [24].

1.3 Why Machine Learning?

The goal of machine learning is to automatize the process of model development. Models are always inferred from observations. Traditionally, a smart person would first use observations to train her intuition, ie. brain, and subsequently try to convert that intuition into a formalized model. However, thanks to Moore’s Law, computational power has grown much faster than the number of scientists and engineers. It is therefore appealing to leverage that computer power to cut out the human middleman and train a formal model directly.

Modelling has to deal with many forms of uncertainty: Measurement noise, unknown model parameters, and the general structure of the model itself. The mathematical language for uncertainty is probability theory. Its primitive objects are probability distributions, which are governed by the sum
rule and the product rule:

\[ P(x) = \sum_y P(x, y) , \quad P(x, y) = P(x)P(y|x) \]

From these two rules the important Bayes’ rule follows readily:

\[ P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_y P(x, y)} \]

Bayes’ rule can be used to incorporate a new observation into an existing distribution of hypotheses. For example, \( y \) could denote an unobservable model parameter and \( x \) a new experimental measurement. Learning from this new observation means to transform the prior knowledge \( P(y) \) into posterior knowledge \( P'(y) = P(y|x) \). The posterior distribution \( P'(y) \) can then serve as prior for the next learning iteration. Upon convergence, the proper distribution of the model parameter has been inferred from the experimental data.

The principle of Bayesian inference can also be applied to entire functions, which led to the development of the Gaussian process (GP) [25]. A GP infers a distribution of functions \( P(f) \) from a set of training points \( \{x, f(x)\} \). The inference is based on assumptions about the function’s typical length scales and the noise within the training data. The training of a GP is illustrated in figure 1.6. The left side shows a uniform prior distribution before any training points were added. The three plotted functions are samples drawn from the prior distribution. The shaded area represents the standard deviation around the mean function. Graphics from Rasmussen and Williams [25].

Figure 1.6: Comparison of Gaussian process before and after training with five points. Plotted functions are samples drawn from the process’ distribution. The shaded area represents the standard deviation around the mean function. Graphics from Rasmussen and Williams [25].

The principle of Bayesian inference can also be applied to entire functions, which led to the development of the Gaussian process (GP) [25]. A GP infers a distribution of functions \( P(f) \) from a set of training points \( \{x, f(x)\} \). The inference is based on assumptions about the function’s typical length scales and the noise within the training data. The training of a GP is illustrated in figure 1.6. The left side shows a uniform prior distribution before any training points were added. The three plotted functions are samples drawn from the prior distribution. The shaded area represents the standard deviation around the mean of the GP. The right side of figure 1.6 shows the GP after five training points were added (marked by plus-signs). Notice that
1.3. Why Machine Learning?

the observations were assumed to be noise-free, which is why the inferred functions pass exactly through the training points.

Gaussian processes belong to the class of non-parametric models, which means that their computational costs grow with the amount of incorporated training data. This can become a problem for very large training sets. Many recent breakthrough applications such as speech recognition, computer vision, and Google’s AlphaGo use therefore deep neural networks with a very large, but fixed, number of model parameters [26, 27]. However, it can be shown that in the limit of infinitely many hidden layers a neural network becomes equivalent to a Gaussian processes [28]. For a recent review of probabilistic modelling methods see Ghahramani [29].

The application of machine learning techniques to quantum chemistry is a rather young and very active field. For a recent review see Ramakrishnan and von Lilienfeld [30]. Its aim is to mitigate the high computational cost associated with quantum calculations. Initially, the research focused mostly on predicting observable properties directly from atomic positions [31]. For example, very successful recent applications include the derivation of force fields using neural network descriptions [32, 33, 34]. However, such end-to-end predictions pose a very challenging learning problem. As a consequence they require large amounts of training data with increasing system size, and the learning must be repeated for each property. Fortunately, the past decades of research have provided a wealth of quantum chemical insights. One can therefore build onto established approximations, such as DFT, and apply machine learning only to small, but expensive, sub-parts of the algorithms. Examples are schemes for learning the kinetic energy functional to perform orbital free DFT [35], or learning the electronic density of states at the Fermi energy [36]. In this work, machine learning is used to predict adaptive basis functions for a given chemical environment. With these adaptive basis sets, properties can be computed at much reduced cost.
Density functional theory (DFT) is a very successful electronic structure method. Following [37, 8], this chapter will give a short introduction to DFT with a particular focus on the methods employed by CP2K [2, 7] to achieve computational costs that scale linearly with system size.

2.1 Schrödinger Equation

One important goal in the field of electronic structure is to solve the Schrödinger equation:

\[ i\hbar \frac{\partial}{\partial t} |\psi(t)\rangle = \hat{H} |\psi(t)\rangle, \]

for a system consisting of electrons and nuclei, described by the following Hamiltonian \( \hat{H} \). The electrons are denoted by lower case subscripts and the nuclei, with charge \( Z_I \) and mass \( M_I \), are denoted by upper case subscripts. For the remainder of this chapter atomic units will be used.

\[
\hat{H} = -\frac{1}{2}\sum_i \nabla_i^2 + \frac{1}{2}\sum_{i\neq j} \frac{1}{|\hat{r}_i - \hat{r}_j|} - \sum_{i,l} \frac{Z_l}{|\hat{r}_i - \hat{R}_l|} \\
-\frac{1}{2M_I}\sum_I \nabla_I^2 + \frac{1}{2}\sum_{I\neq J} \frac{Z_I Z_J}{|\hat{R}_I - \hat{R}_J|} \tag{2.1}
\]

Spin is not included in the Schrödinger equation, therefore the required symmetry of the resulting wave functions has to be ensured by narrowing the solution space. For simplicity of the following derivation, it will be assumed that the system possess an even number of electrons and is not spin-polarized. This is not a true restriction of DFT, as it can also be extended to describe spin-polarized systems as well.
Exact solutions of the Schrödinger equation with the Hamiltonian given in (2.1) are only available for very simple systems like the hydrogen atom. In general, solutions are only achievable through a variety of approximations, of which the most important ones will be covered in the following sections.

2.2 Born-Oppenheimer Approximation

The Born-Oppenheimer approximation is typically applied first to the Hamiltonian in (2.1). It separates the dynamics of the electrons and nuclei, which is justifiable due to their mass-difference \( m_e \ll M_I \). This leads to a simplified electronic Hamiltonian where \( N \) denotes the number of electrons:

\[
\hat{H}_{\text{el}} = -\frac{1}{2} \sum_{i=1}^{N} \nabla_i^2 + \sum_{i<j} \frac{1}{|\vec{r}_i - \vec{r}_j|} + \sum_{i=1}^{N} V_{\text{ext}}(\vec{r}_i).
\]

The nuclei are afterwards treated solely as classical point particles, following a Newtonian equation of motion:

\[
M_I \ddot{\vec{R}}_I = -\nabla_I \langle \psi | \hat{H}_{\text{el}} | \psi \rangle.
\]

The nuclei-positions \( \vec{R}_I \) enter the electronic Hamiltonian \( \hat{H}_{\text{el}} \) only as parameters for the external potential, which describes the nuclei-electron interaction via:

\[
V_{\text{ext}}(\vec{r}) = \sum_{I} \frac{Z_I}{|\vec{r} - \vec{R}_I|}.
\]

The interaction energy \( E_{\Pi} \) between the nuclei themselves is not relevant for the electronic dynamic, it can later be added to regain the total energy of the system.

2.3 Hohenberg-Kohn Theorems

Although the Born-Oppenheimer approximation and the resulting \( \hat{H}_{\text{el}} \) poses a great simplification, it does not tackle the central issue, which lays in the Coulomb interaction \( \hat{V}_{\text{int}} \). Since each electron interacts with all other electrons, all degrees of freedom are coupled, which renders any kind of separation-approach unfeasible. This means a solution would require to actually calculate the entire wave function as a whole, a function which depends on \( 3N \) coordinates for a system of \( N \) electrons. A straightforward solution of the high-dimensional Schrödinger equation is therefore computationally not feasible. This phenomenon is not unique to electronic structure
2.3. Hohenberg-Kohn Theorems

theory, but poses the fundamental problem in many other fields of science as well. It was fittingly coined *curse of dimensionality* by Richard Bellman [38]. In short, only a reduction of dimensionality can facilitate a solution of the electronic Schrödinger equation. This is why it was a breakthrough, when in 1964 Hohenberg and Kohn [39] proved their two famous theorems:

**Hohenberg-Kohn Theorem I:** For any system of interacting particles in an external potential $V_{\text{ext}}(\vec{r})$, the potential $V_{\text{ext}}(\vec{r})$ is determined uniquely, except for a constant, by the ground state particle density $\rho_0(\vec{r})$.

**Hohenberg-Kohn Theorem II:** A *universal functional* for the energy $E[\rho]$ in terms of the density $\rho(\vec{r})$ can be defined, valid for any external potential $V_{\text{ext}}(\vec{r})$. For any particular $V_{\text{ext}}(\vec{r})$, the exact ground state energy of the system is the global minimum value of this functional, and the density $\rho(\vec{r})$ that minimizes the functional is the exact ground state density $\rho_0(\vec{r})$.

The proof of these theorems consists of a simple *reductio ad absurdum* and can be found in many textbooks [40]. The original proof by Hohenberg and Kohn is restricted to so-called *V-representable* densities, which are ground state densities of electronic Hamiltonians with some external potential $V_{\text{ext}}(\vec{r})$. But this restriction was later lifted by a more general formulation by Levy and Lieb [41]. The theorems are generally applicable, because electrons are usually in their ground state.

The Hohenberg-Kohn theorems show that for the ground state it is possible to replace the high-dimensional many particle wave function $\psi_0(\vec{r}_1, \ldots, \vec{r}_N)$ with the only three dimensional density $\rho_0(\vec{r})$. Furthermore, they state that an *universal energy functional* $E[\rho]$ must exist, which could then be used to find this ground state density $\rho_0(\vec{r})$ via a minimization approach. However, Hohenberg and Kohn gave no recipe on how to actually find this energy functional.

Since the energy functional $E[\rho]$ is strongly related to the Hamiltonian, it seems like a logical starting point for the quest for a suitable expression of the energy functional:

$$\frac{E[\rho]}{\text{unknown}} = \langle \hat{H}_{\text{at}} \rangle = \langle \hat{\mathcal{T}} \rangle + \langle \hat{\mathcal{V}}_{\text{int}} \rangle + \sum_{i=1}^{N} \langle V_{\text{ext}}(\vec{r}_i) \rangle .$$

In the following each term of $\langle \hat{H}_{\text{at}} \rangle$ will be discussed. The external potential can be rewritten in terms of the density without any further approximations:

$$\sum_{i=1}^{N} \langle V_{\text{ext}}(\vec{r}_i) \rangle = \int V_{\text{ext}}(\vec{r}) \rho(\vec{r}) \, d\vec{r} .$$
The term $\hat{V}_{\text{int}}$, which describes the Coulomb interaction among the electrons, can be approximated with its classical correspondent, which is commonly called Hartree term after D. R. Hartree:

$$\langle \hat{V}_{\text{int}} \rangle \approx E_{\text{Ha}}[\rho] = \frac{1}{2} \int \frac{\rho(\vec{r}) \rho(\vec{r}')}{|\vec{r} - \vec{r}'|} d\vec{r} d\vec{r}' .$$  \hspace{1cm} (2.2)

The kinetic energy term $\hat{T}$ is problematic. Up to this day, no accurate kinetic energy functional could be found. But Kohn and Sham devised an alternative approach.

### 2.4 Kohn-Sham Ansatz

The ansatz of Kohn and Sham [42] avoids the need to find a functional for the kinetic energy. Instead they return to the many particle wave function $\psi$ and approximate it as a single Slater determinant of non-interacting particles $\phi_i$, the so-called Kohn-Sham orbitals:

$$\psi(\vec{r}_1, \ldots, \vec{r}_N) \approx \frac{1}{\sqrt{N!}} \det \{ \phi_i(\vec{r}_i) \} .$$

The determinant ensures the full anti-symmetry of the solution, which is required for the fermionic electrons. This ansatz wave function allows to calculate the kinetic energy in the common way, using the $\nabla$-operator:

$$\langle \hat{T} \rangle = -\frac{1}{2} \int |\nabla \psi|^2 d\vec{r}_1 \ldots d\vec{r}_N$$

$$\approx -\frac{1}{2} \int |\nabla \frac{1}{\sqrt{N!}} \det \{ \phi_i \} |^2 d\vec{r}_1 \ldots d\vec{r}_N$$

$$\equiv T_s$$ \hspace{1cm} (2.3)

However, a determinant contains in general $N!$ terms, because it is defined as a sum over all possible permutations of its arguments. Therefore, the expression for $T_s$ in equation 2.3 is still very expensive to evaluate. Only by assuming that the orbitals are orthonormal, $\langle \phi_i | \phi_j \rangle = \delta_{ij}$, the Slater-Condon rules [43, 44] can be applied to obtain a simple expression for the kinetic energy:

$$T_s = -\frac{1}{2} \sum_{i=1}^{N} \int |\nabla \phi_i(\vec{r})|^2 d\vec{r} \quad \text{with} \quad \langle \phi_i | \phi_j \rangle = \delta_{ij}.$$ \hspace{1cm} (2.4)

Since $T_s$ and $E_{\text{Ha}}$ are only approximations for the expectation values $\langle \hat{T} \rangle$ and $\langle \hat{V}_{\text{int}} \rangle$ of the true solution, Kohn and Sham furthermore devised a correction term, the exchange correlation functional:

$$E_{\text{xc}} = \langle \hat{T} \rangle - T_s + \langle \hat{V}_{\text{int}} \rangle - E_{\text{Ha}}[\rho].$$
An expression for $E_{xc}$ cannot be derived in a strict fashion, but there are a large number of approximations described in the literature. When combining all these terms, they make up the Kohn-Sham functional:

$$E_{KS} = T_s + E_{Ha}[\rho] + E_{xc}[\rho] + \int V_{ext}(\vec{r}) \rho(\vec{r}) d\vec{r}.$$ 

Following the Kohn-Sham theorems, the ground state can be found by minimizing the energy functional. Within the chosen ansatz, this translates into finding a set of Kohn-Sham orbitals $\{\phi_i\}$, which minimize $E_{KS}$. Formally, this leads to a variational problem, with the additional orthonormality constraint from equation (2.4) that had to be introduced for the kinetic energy expression:

$$\delta E_{KS}[\{\phi_i\}] = 0 \quad \text{with constraint:} \quad \int \phi_i^*(\vec{r}) \phi_j(\vec{r}) d\vec{r} = \delta_{ij}.$$ 

The solution of this variational problem yields a set of the $N$ so-called Kohn-Sham equations. They contain new variables $\epsilon_i$, which originate from Lagrange multipliers that were introduced to enforce the orthonormality constraints. The form of the Kohn-Sham equations resembles that of a single particle Schrödinger equation:

$$\epsilon_i \phi_i(\vec{r}) = \left[ -\frac{1}{2} \nabla^2 + \int \frac{\rho(\vec{r}')}{|\vec{r} - \vec{r}'|} d\vec{r}' + \frac{\delta E_{xc}[\rho(\vec{r})]}{\delta \rho(\vec{r})} + V_{ext}(\vec{r}) \right] \phi_i(\vec{r}) \equiv \left[ -\frac{1}{2} \nabla^2 + V_{KS}(\vec{r}) \right] \phi_i(\vec{r}). \quad (2.5)$$

These $N$ Kohn-Sham equations are not independent from each other, as they are coupled by the density:

$$\rho(\vec{r}) = \sum_{i=1}^{N} |\phi_i(\vec{r})|^2.$$ 

Therefore, the Kohn-Sham equations can not be solved directly. Instead, an iterative approach is typically applied, in which the electrostatics converge towards a self consistent field (SCF).

### 2.5 Finite Basis Sets

For a practical implementation of the Kohn-Sham equations (2.5), they have to be expressed in terms of a finite basis of size $M$ that is spanned by suitable functions $\{q_a\}$. Typical choices include plane waves and Gaussian type orbitals (GTO), of which CP2K uses the later. The Kohn-Sham orbitals are
expanded into the finite basis using coefficients $C_i^a$, which are called *molecular orbital coefficients*:

$$
\phi_i(\vec{r}) = \sum_{a}^M C_i^a \varphi_a(\vec{r}) .
$$

The right-hand-side of the Kohn-Sham equations (2.5) translates into the Kohn-Sham matrix $H$:

$$
H_{\alpha\beta} = \int \varphi_\alpha(\vec{r}) \left[ -\frac{1}{2} \nabla^2 + V_{KS}(\vec{r}) \right] \varphi_\beta(\vec{r}) \, d\vec{r} .
$$

(2.6)

Since the basis functions $\varphi_a$ have usually a simple analytic form, the $\nabla^2$-operator can be readily evaluated.

In the case of GTO basis sets, the basis functions are not orthonormal and an overlap matrix $S$ has to be introduced:

$$
S_{\alpha\beta} = \int \varphi_\alpha(\vec{r}) \varphi_\beta(\vec{r}) \, d\vec{r} .
$$

The orthonormality constraint on the Kohn-Sham orbitals translates into a condition on the molecular orbital coefficients:

$$
C_i^TSC_j = \delta_{ij} .
$$

(2.7)

The Kohn-Sham equations expressed in the finite basis take the form of a generalized matrix eigenvalue problem:

$$
HC_i = SC_i\epsilon_i .
$$

(2.8)

After solving the eigenvalue problem, the electronic density can be calculated from the molecular orbital coefficients:

$$
\rho(\vec{r}) = \sum_{i}^M f(\epsilon_i) \sum_{a\beta}^M \varphi_\alpha(\vec{r}) C_i^a C_i^\beta \varphi_\beta(\vec{r}) .
$$

The function $f(\epsilon)$ denotes the occupation number that is determined by the Fermi distribution. For the ground state the zero temperature limit has to be taken, which yields a step-function:

$$
f(\epsilon) = \lim_{T\to0} \left[ 1 + \exp\left( \frac{\epsilon - \mu}{kT} \right) \right]^{-1} = \frac{1}{2} \left[ 1 - \text{sign}(\epsilon - \mu) \right]
$$

(2.9)

The zero temperature approximation (2.9) holds only for systems which exhibit a HOMO-LUMO gap that is larger than $kT$. This is the case for all
2.6. Exploiting Nearsightedness

systems considered in this thesis. The chemical potential $\mu$ has to be chosen such that the correct number of electrons is obtained:

$$N = \int \rho(\vec{r}) \, d\vec{r} = \sum_{i} f(\epsilon_i).$$

As a consequence, the finite basis has to contain at least $N$ basis functions. Typically, a minimal basis set has to contain a few additional basis functions to allow for the correct symmetry of the orbitals. For closed shell systems each orbital can be occupied by two electrons, which allows for a halving of the basis set size.

A further reduction of the basis set size can be achieved by exploiting the fact that only valence electrons participate in chemical bonds. Therefore, the electrons in the lower closed shells are often not simulated explicitly and instead represented by a static pseudo-potential. In CP2K the pseudo-potentials developed by Goedecker, Teter, and Hutter are used [45].

The downside of a minimal basis set is that it oftentimes gives very poor results. Therefore, basis sets are typically chosen much larger than $N$ in order to achieve sufficient accuracy. When solving equation (2.8) this leads to $(M - N)$ additional eigenvectors. They are assigned an occupation of zero and referred to as virtual orbitals. In chapter 4 the novel PAO-ML method will be presented, which allows to obtain good results even when a minimal basis set is employed.

2.6 Exploiting Nearsightedness

Traditional DFT implementations that use the formulation as eigenvalue problem from equation (2.8) exhibit a cubic scaling with respect to system size. The reason lays in the required orthonormality of the Kohn-Sham orbitals. Since the orbitals can potentially span the entire space, even just verifying the orthogonality of all $N^2$ pairs of orbitals has cost of $O(N^3)$. On the contrary, it is intuitive that electrons in most systems do not extend over long distances, which was coined as the nearsightedness of electronic matter by Walter Kohn [14].

Formally, nearsightedness corresponds to the decay of the reduced density matrix $P(\vec{r}, \vec{r}')$. This matrix can be interpreted as a one-electron correlation function, i.e. the probability of an electron to be simultaneously present at two points in space. For isolating systems, as discussed here, $P(\vec{r}, \vec{r}')$ decays exponentially as the distance $|\vec{r} - \vec{r}'|$ increases. In the Kohn-Sham ansatz the
matrix $P$ can be expressed as a simple sum over the orbitals:

$$P(\vec{r}, \vec{r}') = N \int \Psi^{*}(\vec{r}, \vec{r}_2 \ldots \vec{r}_N) \Psi (\vec{r}', \vec{r}_2 \ldots \vec{r}_N) \, d\vec{r}_2 \ldots d\vec{r}_N$$

$$= \sum_{i=1}^{N} \phi_i^{*}(\vec{r}) \phi_i (\vec{r}') .$$

In the finite basis representation the density matrix corresponds to the projector onto the subspace occupied by the orbitals:

$$P_{\alpha\beta} = \sum_{i} f(\epsilon_i) C_{\alpha}^{\dagger} C_{\beta}^{\dagger}$$

$$= \frac{1}{2} \sum_{i} \left[ 1 - \text{sign}(\epsilon_i - \mu) \right] C_{\alpha}^{\dagger} C_{\beta}^{\dagger}$$

(2.10)

The decay of the density matrix can be exploited by introducing a threshold value $\epsilon_{\text{filter}}$ below which matrix elements are treated as zeros. Once the system exceeds a certain size, the matrix $P$ becomes sparse. A precondition for the appearance of sparsity is that the employed basis functions are also localized, like e.g. the GTO basis functions used by CP2K. Localized basis functions will also allow for the matrices $H$ and $S$ to become sparse.

In order to take advantage of the sparsity, the density matrix has to be calculated without the cubically scaling diagonalization of the Kohn-Sham matrix $H$. To this end, the step-functions $f(\epsilon)$ in equation (2.10) can be rewritten from a scalar function that acts on the eigenvalues of $H$ into a matrix function that acts on $H$ directly:

$$P = f(S^{-1}H)S^{-1}$$

$$= \frac{1}{2} \left[ 1 - \text{sign} \left( S^{-1}H - \mu 1 \right) \right] S^{-1} .$$

(2.11)

The matrix sign function and the inverse of the overlap matrix $S^{-1}$ in equation (2.11) can be calculated in a number of ways, for example as a power series. Efficient algorithms for the evaluation of these series will be presented in the sections 2.8 and 2.9. The evaluation of a matrix power series requires only two operations, namely matrix-matrix multiplications and additions. These operations scale linearly with system-size, because all involved matrices become sparse beyond a certain threshold-size.

The electron number, which is needed to determine the chemical potential $\mu$, can be calculated from the density matrix via $N = \text{tr}(PS)$. And also the Kohn-Sham energy functional $E_{\text{KS}}[\rho]$ can be expressed in terms of the density matrix, because the electronic density $\rho$ is obtainable directly from
the density matrix:

\[ \rho(\vec{r}) = \sum_{a\beta}^{M} \varphi_a(\vec{r}) P^{a\beta} \varphi_\beta(\vec{r}). \]  \hspace{1cm} (2.12)

In this new formulation the Pauli principle is now reflected by an idempotency condition. It requires that \( PS \) is a projector, which implies that the orbitals have either 0 or 1 as occupation:

\[ PSPS = PS. \]

The eigenvector property of the orbitals is reflected by a commutativity condition. It implies that \( S^{+1/2} P S^{+1/2} \) and \( S^{-1/2} H S^{-1/2} \) share the same eigenspaces:

\[ HPS = SPH. \]

The Kohn-Sham matrix \( H \) still depends on the density matrix \( P \). Therefore, equation (2.11) has to be evaluated repeatedly within an SCF optimization. Figure 2.1 shows in a flow-graph how the linear-scaling scheme works compared to traditional DFT implementations.

2.7 Gaussian and Plane Waves Method

A linear-scaling DFT implementation requires a linear-scaling method for calculating the Kohn-Sham matrix \( H \) in equation (2.6). If a localized basis set is employed, most terms in \( H \) are straightforward to calculate in \( \mathcal{O}(N) \), because the involved operators are local as well.

The Hartree term (2.2), however, is non-local due to the long range of the Coulomb interaction. Formally, the Hartree term can be rewritten into a local term by introducing the electrostatic potential \( V_E \):

\[ E_{Ha}[\rho] = \frac{1}{2} \int \rho(\vec{r}) V_E(\vec{r}) d\vec{r} \]

\[ V_E(\vec{r}) = \int \frac{\rho(\vec{r}')}{|\vec{r} - \vec{r}'|} d\vec{r}'. \]

The electrostatic potential \( V_E \) can be obtained by solving Poisson’s equation:

\[ \frac{\partial^2}{\partial \vec{r}^2} V_E(\vec{r}) = \rho(\vec{r}) \]

Under periodic boundary condition the Poisson’s equation can be solved efficiently by using the Fourier transformed of the density:

\[ \rho(\vec{k}) = \mathcal{F} [\rho(\vec{r})] \]
2. Linear Scaling DFT

In the reciprocal representation the real-space derivatives of Poisson’s equation can be readily calculated:

$$\rho(\vec{k}) = \frac{\partial^2}{\partial \vec{r}^2} V_E(\vec{k}) = \frac{1}{\vec{k}^2} V_E(\vec{k}) ,$$

which makes solving for the electrostatic potential trivial:

$$V_E(\vec{k}) = \frac{\rho(\vec{k})}{\vec{k}^2} .$$

The solution has to be transformed back, using the inverse Fourier transformation, to obtain the real-space representation of the electrostatic potential:

$$V_E(\vec{r}) = \mathcal{F}^{-1} \left[ \frac{\rho(\vec{k})}{\vec{k}^2} \right].$$

For functions that are discretized on a regular grid, the Fourier transformation can be efficiently calculated using a Fast Fourier Transformation (FFT) algorithm, which scales $\mathcal{O}(N \log(N))$ with the grid-size.
In the case of CP2K, the primary basis consists of GTO functions. To nevertheless take advantage of the efficient FFT algorithm, the density has to be transferred from the GTO representation onto a regular grid, which is referred to as *collocation*. After solving Poisson’s equation the obtained electrostatic potential has to be transferred back into the GTO basis, which is referred to as *integration*. This “opportunistic” switching between a Gaussian and a plane wave representation is the core idea of the *Gaussian and Plane Waves* (GPW) method [46], which is illustrated in Figure 2.2.

![Flow-graph of the GPW method](image)

**Figure 2.2:** Flow-graph of the GPW method, which provides a linear-scaling method to calculate the Kohn-Sham matrix.

The GPW method is implemented in CP2K [47], where it is the key for calculating the Kohn-Sham matrix $H$ in $\mathcal{O}(N)$.

### 2.8 McWeeny Purification

The process of calculating the density matrix $P$ from the Kohn-Sham matrix $H$ in equation (2.11) is often referred to as *purification*, because the eigenvalues of $H$ are mapped to 0 and 1. While several purification algorithm exist, one of the simplest is based on the matrix sign function that can be defined as:

$$
\text{sign}(A) = A(A^2)^{-\frac{1}{2}}.
$$

For diagonalizable $A$, eigenvectors of $A$ are eigenvectors of $\text{sign}(A)$, with the eigenvalues of $\text{sign}(A)$ being $-1$ or $1$ for negative or positive eigenvalues of $A$ respectively. Various simple iterative algorithms are available to compute the matrix sign function [48] and these approaches have found early application [49, 50]. These algorithms converge super-linearly and are numerically stable. A simple form, which is also commonly known as McWeeny purification [51], requires only two matrix multiplies per iteration and is given by:

$$
X_0 = \frac{A}{||A||}, \quad X_{n+1} = \frac{1}{2}X_n(3I - X_n^2), \quad \lim_{n \to \infty} X_n = \text{sign}(A).
$$
The convergence criterion employed terminates the iteration at $X_{n+1}$ if

\[ || \mathbb{1} - X_n^2 ||_F < \sqrt{\epsilon_{\text{filter}}} || X_n^2 ||_F , \]

where $||.||_F$ denotes the Frobenius norm. Since the algorithm is quadratically convergent, near convergence, each iteration will approximately double the number of correct digits in the solution. Linear-scaling results from the fact that all matrix operations are performed on sparse matrices, which have a number of non-zero entries per row that is independent of system size. In order to retain sparsity during the iterations a threshold $\epsilon_{\text{filter}}$ is employed to set small entries to zero after multiplication, thereby reducing the data volume and speeding up the following multiplies.

The inverse of the overlap matrix $S^{-1}$ can be computed conveniently using $S^{-1} = S^{-\frac{1}{2}}S^{-\frac{1}{2}}$ where the square root and inverse square root can be obtained from:

\[
\text{sign}\left(\begin{bmatrix} 0 & A \\ 1 & 0 \end{bmatrix}\right) = \left(\begin{bmatrix} 0 & A^{\frac{1}{2}} \\ A^{-\frac{1}{2}} & 0 \end{bmatrix}\right).
\]

A stationary solution of the self-consistent equations can be obtained by a simple mixing approach:

\[
P_{n+1} = \frac{1}{2}(1 - \text{sign}(S^{-1}\hat{H}_n - \mu_n\mathbb{1}))S^{-1}
\]

\[
\hat{H}_{n+1} = (1 - \alpha)\hat{H}_n + \alpha H_{n+1} ,
\]

where $\alpha$ is a mixing parameter between zero and one, and $H_n$ an auxiliary matrix. The fixed point implies that $\hat{H}_n = H_n$ and thus $SP_nH_n - H_nP_nS = 0$.

For each iteration, the total electronic energy $E_n$ and Hamiltonian matrix $H_n$ are computed from the density matrix $P_n$. The value of the chemical potential $\mu_n$ for a given $N$ is determined by bisecting a suitable interval until the following condition is fulfilled:

\[ |\text{tr}(P_nS) - N| < \frac{1}{2} . \]

Note that the $\text{tr}(P_nS)$ is integer-valued unless finite accuracy is employed in the calculation of the sign function. For a given SCF threshold $\epsilon_{\text{SCF}}$, the convergence criterion employed is:

\[ E_n - E_{n-1} < \epsilon_{\text{SCF}} N . \]

## 2.9 Trace Resetting Purification

More advanced algorithms that still use fix point iterations exist. They include the optimization of the chemical potential [52] as part of the density
2.10 Curvy-Step Method

matrix computation, and achieve faster convergence by relaxing the absolute trace conservation [53]. These methods represent a significant advantage over the sign matrix iteration, if the chemical potential is not known in advance, as the cumbersome bisection can be omitted. Also trace resetting (TRS) purification starts from a normalized Hamiltonian matrix \( X_0 \), where the eigenvalues are in the interval \([0, 1]\). At each iteration the algorithm calculates the following quantity:

\[
\gamma_n = \frac{N - \text{tr}[\mathcal{F}(X_n)]}{\text{tr}[\mathcal{G}(X_n)]}.
\]

The actual update step is then chosen based on the calculated \( \gamma \) value:

\[
X_{n+1} = \begin{cases} 
X_n^2, & \gamma_n < \gamma_{\text{min}} \\
\mathcal{F}(X_n) + \gamma_n \mathcal{G}(X_n), & \gamma_n \in [\gamma_{\text{min}}, \gamma_{\text{max}}] \\
2X_n - X_n^2, & \gamma_n > \gamma_{\text{max}}
\end{cases}
\]

(2.13)

The choice of the polynomial functions \( \mathcal{F} \) and \( \mathcal{G} \) is not unique, but an efficient algorithm (TRS4) is achieved by using:

\[
\mathcal{F}(x) = x^2(4x - 3x^2) \\
\mathcal{G}(x) = x^2(1 - x)^2.
\]

For this choice of polynomials the appropriate interval limits for equation (2.13) are \( \gamma_{\text{min}} = 0 \) and \( \gamma_{\text{max}} = 6 \).

2.10 Curvy-Step Method

Another class of algorithms aims at a direct minimization of the energy functional, avoiding the self-consistent mixing, and thus adding robustness. To achieve this, the constraints on the density matrix have to be included into the algorithm. In the work of Li, Nunes, and Vanderbilt [54] this was achieved by using an extended energy functional. Helgaker et al. [55] proposed a parameterization of the density matrix, that conserves idempotency. Within this curvy-step method [56, 57], starting from an idempotent \( P_0 \), as obtained for example from the TRS method, one performs updates of the form

\[
P_{n+1} = e^{-\Delta S} P_n e^{\Delta S},
\]

where \( \Delta \) is an anti-Hermitian matrix, \( \Delta^\dagger = -\Delta \). This unitary transformation is evaluated using the Baker-Campbell-Hausdorff expansion

\[
P_{n+1} = P_n + [P_n, \Delta]_S + \frac{1}{2} [[P_n, \Delta]_S, \Delta]_S + \frac{1}{6} [[[P_n, \Delta]_S, \Delta]_S, \Delta]_S + \cdots,
\]
where the commutator within a nonorthogonal basis is

\[ [X, \Delta]_S = XS\Delta - \Delta SX. \]

In the minimization, the matrix elements of the curvy-step matrix \( \Delta \) are the free variables and are calculated from the energy gradient

\[ \frac{\partial E}{\partial \Delta} = [H, P_n]_S \]

using for example a steepest descent, conjugate gradient, or a Newton-Raphson method.

All of the above algorithms have in common that matrix multiplication is the dominant operation. The performance of the underlying sparse matrix multiplication routines is of paramount importance for the overall computational efficiency.
Chapter 3

GPU Accelerated Sparse Matrix Matrix Multiplications

The linear scaling SCF implementation in CP2K is centered around sparse matrix matrix multiplication [2, 7]. This choice is motivated by the fact that matrix multiplication is a basic primitive that is suitable for a high performance parallel implementation. Furthermore, this operation can be used to compute matrix functionals, such as for example inv, sqrt, sign, and exponential.

Surprisingly, no established software library is available that performs a parallel sparse matrix matrix multiplication. Such a library should, in the context of quantum chemistry, exploit the concept of sub-matrix blocks, rather than individual elements for a description of the sparsity pattern. These sub-matrix blocks, also named atomic blocks as they correspond to basis functions of an atom, are small (typical numbers are 5, 13, 23), and exploiting them is key to achieve good performance. Furthermore, as most calculations are currently performed near the cross-over regime between dense and sparse, the library must be highly efficient for relatively high occupations (for example 50% non-zero elements), and 1000s of non-zeros per row, while optimal performance for very sparse matrices (less than 1-5% non-zero elements) will become more important in the future.

3.1 DBCSR: A Sparse Matrix Library

In order to address the above needs, a general purpose sparse matrix library has been developed [58]. This library is distributed as part of the CP2K package and available as an independent tool [59]. The name of the library is DBCSR, which is an abbreviation for Distributed Blocked Compressed Sparse Row, or Distributed Blocked Cannon Sparse Recursive. The full names emphasize the storage format or the multiplication algorithm respec-
3. GPU Accelerated Sparse Matrix Matrix Multiplications

3.1 Acceleration of Sparse Matrix Multiplications

Data is stored distributed over all processes, using a blocked variant of the compressed sparse row storage format. The parallel algorithm to perform the matrix multiplication is based on the Cannon scheme [60], which is optimal in the dense case, if memory is a limited resource. In particular, it guarantees that communication per process decreases with increasing process count, and is free from all-to-all communication. These properties guarantee strong scaling of the algorithm, and good performance in the dense limit. Nevertheless, sparse matrix multiplication has $O(N)$ flops and $O(N^2)$ data, and reaching peak performance is thus difficult. The ratio of flops to data does not depend on system size, but rather on the number of non-zeros per row. The later depends typically on the accuracy of the calculation, such as tighter filtering thresholds and larger basis sets in our quantum chemical applications. We refer to Ref. [58] for an in-depth discussion. In the following, we focus on those aspects that are important in the context of GPU-acceleration, and on the recent developments that have enabled a significant increase in accelerated performance.

3.2 Software Architecture for GPU-Acceleration

In this section, we outline the various layers of the DBCSR matrix multiplication architecture. It has been designed to decouple the various steps of the calculation, and is schematically shown in Fig. 3.1. As we go down the layers, the granularity of the data becomes smaller and the computational workload increases. However, the higher level layers manage data transfers, optimize data access, and enable asynchronous progress. These steps are essential to fully benefit from the high performance that modern CPUs and GPUs offer.

3.2.1 Cannon Layer

The top-most layer deals with the parallelization of the matrix multiplication over the nodes of a cluster, and enables good parallel performance by managing the message passing between MPI processes. One MPI process can consist of several CPU threads, based on OpenMP, and can off-load to a dedicated or shared GPU. For the MPI-parallelization, the sparse matrices are divided into large sparse sub-matrices named panels. These panels are regular in shape, and by a suitable row and column permutation, the sparsity pattern has been homogenized so that all panels contain approximately the same amount of data, which is favorable for load-balancing the calculations. These panels are distributed over a regular 2D grid of processes and Cannon's algorithm[60] is used to communicate these panels between processes in a regular and ordered fashion, according to ‘ticks’ in the Cannon metronome. In each tick of Cannon’s algorithm, each process sends and receives two panels, multiplies the two panels that are available, and accu-
3.2. Software Architecture for GPU-Acceleration

Figure 3.1: Shown is a schematic representation of the software architecture employed in the GPU accelerated DBCSR library. The various layers correspond to key steps in the matrix multiplication algorithm. While the Cannon layer is essential for the parallelism between processes or on the cluster level, the lower layers deal with parallelism and acceleration on the node level.

mulate the results locally. As discussed below, messages can be processed asynchronously, i.e. be in transit over the network, while computation takes place. The panel data are also uploaded to the GPU in this layer. Our approach to enable the asynchronous message passing and uploads will be discussed in more detail below. The following, lower layers deal with the process or node local multiplication of panels.

3.2.2 Multrec Layer

The *multrec* layer, is a high level process-local layer that aims at optimizing memory access, and in particular at exploiting the deep cache hierarchy of modern processors. Indeed, even for a standard dense matrix multiplication, optimal data reuse is essential to reach good performance. Usually, detailed knowledge of the architecture can be combined with the well known data-
access pattern of a dense matrix multiplication to optimally block matrices and to guarantee best cache reuse. However, due to the unknown sparsity pattern and relatively complicated data structures, this approach is not general enough in the sparse case. An alternative technique, also derived in the context of dense matrix multiplication is therefore employed, which instead uses a recursive approach to matrix multiplication [61]. Matrices are multiplied by recursively dividing the longest dimension of the matrix in two, until sufficiently small matrix dimensions have been obtained and all the data fits fully into a low-level cache. This cache-oblivious algorithm results in a near-optimal data access pattern for dense matrices, without explicit knowledge of the cache hierarchy, and is easily adapted to the sparse case.

3.2.3 CSR Layer

The compressed sparse row layer, or CSR layer, determines from the CSR data which blocks have to be multiplied. It is important to emphasize that the sparsity pattern of the result matrix is not fixed or known a priori, so that this processes is driven by the right-hand-side of the equation \( C = AB \). In DBCSR, a two step approach, well suitable for GPUs, has been adopted. It separates performing the actual floating point operations from the indexing and book-keeping. The CSR layer performs the latter, on the host, deferring flops to lower layers. During the indexing, lists of needed block-multiplications, named ‘stacks’ are generated, and passed on to the lower scheduling layers, i.e. are flushed, as soon as the limited space of a stack is exhausted, or the end of a Cannon tick is reached. In order to allow for efficient processing, so-called homogeneous stacks are employed for the most common block sizes, these contain entries that have all the same block-dimensions, while a default stack contains the remaining cases. An important optimization has been introduced in this layer namely on-the-fly filtering. This optimization employs precomputed matrix block norms to decide if a given block product contributes to the final result significantly in comparison to the sparsity threshold, and skips negligible multiplications.

In actual applications, even for matrices that are dense in data, this optimization can reduce the number of needed flops by a factor two to four. The relative computational cost of the indexing operations depends strongly on the size of the basic blocks employed in the application calling the DBCSR library. It is significant if blocks are as small as 5x5, while it is clearly negligible if blocks are of size 23x23 or larger.

3.2.4 Scheduler and Driver Layers

The scheduler layer receives filled stacks and arranges for their processing by handing them off to one of the drivers. The host-driver is employed for CPU-processing and the Cuda driver for GPU-processing. Both the host and
device drivers are built on top of libraries that efficiently perform small matrix multiplications, libsmm and libcusmm for host and device respectively. libsmm has been described in Ref. [58], while libcusmm is described in detail in a following section. Both libraries are significantly more efficient than standard matrix multiplication libraries for the small matrix sizes that are relevant for quantum chemical applications. The scheduler decides where the stack will be processed, and is currently based on a very simple scheme. The GPU is queried using the event based mechanism described below, and if buffer space is available on the GPU the stack will be handed over to the Cuda driver, otherwise the host-driver processes the stack. The amount of buffer space made available on the device is thus a mechanism to tune the host-device load balancing. Following this, the Cuda driver will check if highly tuned kernels are available in the libcusmm library for the particular matrix sizes in the homogeneous stack. If so, the stack is shipped to the GPU for processing, and otherwise is send to the host driver. The latter can deal with small matrix multiplications of all sizes, ultimately falling back to an optimized BLAS library calling DGEMM.

3.3 Maximizing Asynchronous Progress

Part of the challenge in writing efficient GPU-accelerated code, is to exploit the asynchronous task based programming model. Whereas on a homogeneous system typically all processors execute the same program on different parts of the data in a lock-step fashion, on a hybrid system the CPU and GPU complement each other, and are partially independent. In order to fully utilize such a system, different programs need to be executed on the CPU and the GPU. Typically, the host-CPU drives the GPU-device by handing over tasks, and while the GPU is executing these tasks, the CPU can perform other tasks on its own. In the following subsections, our approach to enable this asynchronous processing is explained.

3.3.1 Cuda Streams and Events

Once the host has submitted a task to the device, the CPU loses control over it and the GPU has significant freedom to schedule the task execution. However, dependencies between tasks might be present. For example, a task processing some data might depend on the completion of a prior task that copies this data from the host to the device. These dependencies have to be made explicit by the programmer. The Cuda programming environment provides two powerful mechanisms to enable further concurrency and to enforce dependencies: streams and events. Streams are a simple mechanism to establish dependencies and to enable concurrency. Tasks submitted to a given stream are processed in the order in which they are submitted, while tasks from different streams can be processed in any order or con-
3. GPU Accelerated Sparse Matrix Matrix Multiplications

![Diagram of concurrency and dependencies in DBCSR](image)

**Figure 3.2:** Enabling concurrency and enforcing dependencies in DBCSR. Multiple streams are used to transfer data from the host to the device, and to process independent stacks. Dependencies between the streams, for example a panel upload and stack calculations, and between host and device, for example device buffer reuse, are enforced using events.

Currently, using multiple streams is essential to overlap computation with host-to-device or device-to-host transfers, and to enable concurrent task execution. Events can be used to express more general dependencies. Just like a task, an event can be created and submitted to a stream, and is processed after the previous task submitted to the same stream is completed. However, tasks can be submitted that wait for the completion of events in other streams. These “waiting-tasks” will block a stream until the referred event has occurred, and by submitting waiting-tasks prior to an actual task on the same stream, multiple cross-stream dependencies can be enforced.

In Fig. 3.2, the scheme that is employed in the DBCSR library is illustrated. Stack buffers are transferred and processed in a number of independent streams, so that the stack buffer transfers can overlap with computations in other streams, and that concurrent stack processing is possible. The GPU can only process stacks if the panel and stack data is present, so that for each kernel dependencies on the completed transfer of the A, B, and C panels, taking place in different streams, and completion of the stack buffer transfer, in the same stream, must be present. Retaining the A, B, and C panels on the device while stack buffer processing is in progress is enforced with additional events. Notice that explicit synchronization between host and device.
3.3. Maximizing Asynchronous Progress

device is rarely needed, the host can query events to make sure that, for example, stack buffers have been uploaded before they are overwritten with new data. The host only has to wait for the device when the previous panel is still in use, and at the very end when the final results are downloaded from the device.

The Cuda API allows for an unlimited number of stream, but these are mapped to a limited number of hardware queues. Both the number or hardware queues and the mapping scheme are likely to change depending on the hardware and Cuda version. Unfortunately, mapping of otherwise independent streams to the same hardware queue can lead to unwanted serialization. Therefore, only a limited number of streams is created in DBCSR, specifically, two streams are exclusively for host-to-device transfers, one for odd Cannon ticks and one for even ticks, while a configurable but small (typically 2-4) number of streams is used for stack transfers and kernel launches. Finally, ‘priority streams’ are a recent Cuda feature that introduces some way for the programmer to influence scheduling of kernels. In DBCSR this feature is used to load balance between host threads. In addition to generating stacks, occasionally a host threads will also process a stack. This happens when a host thread has no more free stack buffers available, i.e. when the device is busy. In order to avoid that the device works on buffers of a thread that has finished its work already, and a busy thread looses time processing stacks, stack buffers come in two flavors: priority buffers and posterior buffers. A limited number of priority buffers is assigned to each thread, and mapped to a stream with high priority, while the posterior buffers are mapped to streams with lower priority. The effect of this is that the device will focus on doing the work for those threads that are actively generating stacks, i.e. writing them to the priority buffers, while the posterior buffers are handled later. These buffers, as discussed below, are useful to overlap computation and communication during message passing or host to device transfers. Good performance requires that the number of priority buffers is tuned such that device never idles if all threads are active and using priority buffers only.

3.3.2 Double Buffered Cannon on Host and Device

In a sparse matrix multiplication algorithm, both data movement and floating point operations can contribute significantly to the total runtime. Maximum performance can only be achieved when the corresponding resources are utilized in parallel. To accomplish this, a double buffered scheme has been employed for both host and device. As shown in Fig. 3.3, these two panel buffers are used in a complementary fashion, while one buffer is used for the computation, the other buffer is overwritten as part of a data transfer operation. Data transfer happens between MPI processes and between
Steps of the algorithm depend on the hardware and on the science problem at hand.

Figure 3.3: Schematic representation of the double buffered Cannon algorithm, which illustrates how the use of two host and two device buffers for the A and B panels enables overlapping of message passing, host to device memory copies and computations. The ratio of the time needed for the important steps of the algorithm depends on the hardware and on the science problem at hand.
3.4 Libcusmm: GPU Accelerated Small Matrix Multiplications

Libcusmm is designed to accelerate small matrix multiplications on GPUs. The core computational kernel involves the computation of stacks of small matrix multiplications. The result block matrix $C^u,v$ is computed as

$$C^u,v = A^u \cdot B^v$$

where $A^u$ and $B^v$ are small matrices. The host and device communication is managed through double buffering to ensure smooth data exchange.

Once a panel has been received on the host, it is copied to the corresponding device buffer using an asynchronous host-to-device copy operation. At the same time, the CPU threads start to generate and fill stacks buffers. Stack buffers are transferred and processed by the device as soon as the host-to-device panel copy has finished. Typically, the CPU threads can generate the stacks faster than the GPU can process them, allowing for a large number of stack buffers to be outstanding. These outstanding stack buffers can be processed by the device while the MPI transfer and the host to device copy of the next panel to the second buffer is taking place. Good performance requires that the number of posterior buffers is tuned such that the device never idles during these transfers. A too large number of posterior buffers might lead to host threads waiting for the previous device panel buffer to finish. In the last Cannon tick, posterior buffers are not employed, as threads and device should finish roughly at the same time.

Fast device-to-host transfers require host-pinned memory. Since allocating host-pinned memory and Cuda device memory are slow operations, and memory usage is hard to predict in the case of varying sparsity patterns, memory-pools have been introduced that are persistent across sparse multiplications and only allowed to grow. In our application, the gain in performance outweighs the additional complexity and the fact that less memory is available for the rest of the application in between matrix multiplications.

Finally, whereas the MPI standard specifies non-blocking versions of send and receive (isent / irecv), actual implementations often perform the complete transfer in the corresponding wait statements. We have found that this is in particular the case for multi-megabyte messages, as is required for the panel transfers. To nevertheless overlap computation and communication, a ‘communication thread’ has been introduced in the OpenMP parallel version of the DBCSR library. The master thread, which is responsible for all communication, is underloaded compared to the other threads, and will, given the barrier free nature of the implementation, enter early in a polling loop based on test._any to progress outstanding MPI communication. Tuning the load of the master thread, message passing can be effectively overlapped with computation performed by the other threads and the device.
the product of the block matrices $A_{u,v}^w$ and $B_{w,v}^{w,p}$ according to

$$C_{i,j}^{u,v} = C_{i,j}^{u,v} + \sum_{w,k} A_{i,k}^{u,v} B_{k,j}^{w,v}$$  \hspace{1cm} (3.1)$$

using superscripts to indicate the matrix block indices and subscripts to denote the matrix elements in each of the block matrices. The sum over $w$ takes the sparsity pattern of $A$ and $B$ into account, i.e. the product will be omitted whenever either $A_{u,v}^w$ or $B_{w,v}^{w,p}$ is absent, or their norms are small. Furthermore, $w$ can only refer to those parts of $A$ and $B$ that are part of the panels of $A$ and $B$ that are local to the node for a given tick of Cannon’s metronome. In a single stack, anywhere between one and a few tens of products will be present for a given block $C_{u,v}^{u,v}$. Note that this operation resembles the batched DGEMM operation in CUBLAS, but that this library expects all $C$ matrices in a single batch to be different, and can thus not be used. In the following, the steps necessary to optimize these products on GPUs are described.

### 3.4.1 Small Matrix Multiplication Performance Model

At first sight, matrix multiplication seems dominated by floating point operations, while memory transfer is less important. This certainly is the case for large matrices, but not quite for the small matrices required in the current context. It is therefore useful to look at the arithmetic intensity, which...
3.4. Libcusmm: GPU Accelerated Small Matrix Multiplications

![Diagram of matrix multiplication](image)

**Figure 3.5:** Inner-product (left) and outer-product (right) form of matrix multiplication. The yellow areas indicate elements that can be computed independently by accessing the highlighted areas of $A$ and $B$.

We define $\text{arithmetic intensity}$ to be the ratio of number of floating point operations vs. number of bytes transferred between memory and processing units. In order to perform the matrix multiplication, $A$ and $B$ will need to be loaded from the device memory to the streaming multiprocessor (SM), while $C$ might be assumed present on the SM (favorable limit of a large number of contributions from the summation over $w$), or might need to be loaded and stored as well. For the multiplication of an $m \times k$ by a $k \times n$ matrices, the intensity is thus between $\frac{2mnk}{8(\text{mk}+\text{kn})}$ and $\frac{2mnk}{8(\text{mk}+\text{kn}+2\text{mn})}$. In order to reach the favorable limit, the DBCSR library might sort the stacks, such that $C$ matrix access occurs in order, prior to handing them to the GPU. Given a K20X GPU with 1.3TFlops peak double precision performance and 250GB/s peak bandwidth, an arithmetic intensity of at least 5.2 is needed to achieve peak performance. With ECC turned on, a bandwidth of 180 GB/s is more realistically achievable for a kernel of this complexity, so an arithmetic intensity of at least 7.2 is needed to reach peak performance. Multiplications of matrices smaller than 60x60 are thus necessarily limited by the memory bandwidth, and this remains an important factor, even for significantly larger matrices. This clearly implies that the optimization should focus on reaching optimal memory bandwidth usage. For selected sizes of the small matrices encountered in CP2K applications, the arithmetic intensity, the reachable flop rate, and the actually achieved performance are shown in Fig. 3.4.
3. GPU Accelerated Sparse Matrix Matrix Multiplications

3.4.2 Matrix-Product Algorithm Choice

The first step in implementing the small matrix products is to pick the most appropriate algorithm. Fig. 3.5 shows two possible algorithms for computing the matrix product \((C = C + AB)\). In the canonical form, the result elements in \(C\) are computed using the inner product of rows of \(A\) and columns of \(B\), while an alternative algorithm is based on an outer product of columns of \(A\) and rows of \(B\). These two algorithms result in the same number of floating point operations, but the latter option exhibits significantly more parallelism in that it allows for computing an update for all elements of \(C\) using a single column of \(A\) and a single row of \(B\). An additional benefit of using outer products is data locality, the outer product algorithm touches elements of \(A\) and \(B\) only once, while for the inner products, when computing one row of \(C\), one row of \(A\) and the entire matrix \(B\) needs to be accessed. Based on the model developed in the previous section it is known that the kernel’s performance for problem sizes of interest to CP2K will be limited by memory bandwidth. The outer products algorithm is therefore preferred.

3.4.3 GPU Implementation: Generic Algorithm

The next step in the design of a kernel for small matrix products is to consider data locality. Initially, the \(A\), \(B\), and \(C\) matrices, as well as the product descriptors, the so-called stacks, are all located in global memory on the GPU. Each entry in the stack describes one matrix-matrix product, thus con-
3.4. Libcusmm: GPU Accelerated Small Matrix Multiplications

taining three pointer to the blocks in the A, B, and C panels. After the kernel has read a stack entry, it fetches the matrices A and B, and updates the C matrix with the product of A and B. The matrix sizes of interest correspond to typically 10 – 1000 elements per result matrix C, limiting the degree of parallelism to a similar order. An appropriate choice is therefore to process a matrix product using a single thread block. While this allows for efficient synchronization between the threads processing one product, it requires appropriate safe-guards to avoid data races between multiple updates of the same C matrix block. Multiple consecutive products updating the same result matrix C can be processed by the same thread block, requiring fewer reads and writes of C from global memory. In addition to reducing the number transfers of C between global memory and the SM, this also reduces the probability of collisions that happen when multiple thread blocks update the same C matrix block at the same time. On Kepler-Generation GPUs, atomic memory operations are efficient enough and are hence used to prevent data races. In this context, the overhead of using atomics instead of regular memory updates is on the order of 5%.

How a single thread block deals with the data is illustrated in Fig. 3.6 and explained in the following. First, given that the elements of the result matrix C do not need to be shared between threads, the ideal location to store C is registers. In order to increase instruction level parallelism per thread, and given the large number of registers available per thread, a small tile (T) rather than a single result matrix element is processed per thread. The optimal choice of tile dimensions (MxN) is determined via auto-tuning as described later. Next, the elements of matrix A and B need to be accessed by multiple threads, thus making them ideal candidates to be stored in shared memory. In order to avoid that shared memory utilization limits the number of concurrent thread-blocks (occupancy), a maximum of 3kB per thread-block can be used. For larger blocks A and B, it is thus desirable to read only parts into shared memory, we name these parts slabs. Using the outer product algorithm, these matrix slabs only need to be read once per product. The optimal width (w) of the slabs is also determined by auto-tuning. Given that matrices are stored in column major format, reading a slab of A still leads to perfectly coalesced memory access. However, reading only a slab of matrix B can lead to both significant memory access penalties and complex address computations. In order to avoid these penalties it is therefore desirable to compute the product \( C = A(B^T)^T \) instead, resulting in perfectly coalesced memory accesses with simple address computations for both A and B. Given that typically each B matrix is used many times per Cannon tick, the cost of transposing the full panel of B’s once after the upload in a separate kernel is negligible compared to the time savings due to more efficient memory access and simplified address calculation. Finally, once the entire product is computed, and only when the next stack entry refers
to a different C block, the results are added to the corresponding block in global memory. In order to ensure coalesced writes, an intermediate step is employed, in which slabs of C (of width v) are first put in shared memory, and only then added using an atomic compare-and-swap operation.

3.4.4 Auto-Tuning and Performance

The generic algorithm outlined above requires several parameters (M, N, w, and v), and finding an optimal set of values is not always intuitive. Fetching larger panels of A and B tends to improve performance, but at the same time will also increase the shared memory footprint and limit occupancy, thus potentially limiting the amount of latency hiding. A similar effect occurs for the number of result elements processed per thread: increasing this parameter improves the instruction level parallelism, but at the same time this limits the number of thread blocks resident on each SM. Additionally, some matrix sizes allow for significantly simplified versions of the general kernel and separate implementations were developed. In order to hide details of the tool chain, such as register allocation, that are unknown or subject to change, an autotuning framework based on a small standalone benchmark application is used to find optimal parameters and implementations for each given set of block dimensions m,n,k. It has been verified that the kernel performance in the small standalone benchmark application is very similar to the one observed in full CP2K simulations.

Fig. 3.4 shows the performance obtained in the mini-app for relevant blocks sizes and optimal parameters. The performance is close to that estimated from the model based solely on memory bandwidth considerations. For very small matrices, the measured performance starts to deviate from the theoretically expected performance. We currently attribute this to the warp granularity of the execution on the SM, but have not further optimized for these sizes as we expect that small matrix sizes can just be handled on the CPU side if needed. Finally, for comparison, batched DGEMM in cuBLAS (version 5.0) for a 23x23x23 problem runs at 132 GFLOPS on a K20X, while the current implementation in libcusmm achieves about 322 GFLOPS. For most of the small matrix sizes of interest, a speedup in the range of 2-4x has been measured. This demonstrates the quality of the generated kernels, and the appropriate choice of optimization techniques.

3.5 Benchmarks and Conclusions

In this final section, we illustrate the performance of the linear scaling GPU based implementation. In doing so, we attempt to cover synthetic benchmarks, current application style simulations, as well as very large scale simulations. Given the computational demands of these simulations, the focus
is on parallel application of CP2K. The latter calculations have been performed on a recent hybrid architecture, a Cray XC30, which was installed in the fall of 2013 at CSCS, Switzerland. This machine is named Piz Daint, and is currently the leading European computer in the Top500 list. It features 5272 hybrid compute nodes, with one Intel Xeon E5-2670 processor (8 core, Sandy Bridge), and one NVIDIA K20X per node. The nodes are connected with an aries network based on a dragonfly topology.

As a first demonstration of performance, we focus in two synthetic benchmarks on the matrix-matrix multiplication of nearly dense matrices (occupied 50% or more), with favorable block sizes (23x23). A single node CPU-GPU comparison is shown in Fig. 3.7. In this case, the potential of the GPU is demonstrated, as it outperforms 12 Sandy Bridge cores by a significant factor. Furthermore, increasing the number of CPU cores, the hybrid implementation displays improving performance, showing that host-device sharing is effective. The CPU-only curve demonstrates good parallel efficiency of the OpenMP code. Taking this benchmark to the scale of 5184 hybrid nodes, a matrix of size 536544 x 536544, with 50% occupation, and 23x23 subblocks, can be multiplied in approx 36 seconds with a sustained machine performance in excess of 2 Pflops (nearly 400 Gflops per node). Thus, exploiting the fact that the matrices are 50% occupied, already brings a speedup over a dense matrix multiplication. Indeed, assuming a dense parallel matrix multiplication to run at 6.2 Pflops (the Linpack number for Piz Daint), such a calculation would require 50s. This performance illustrates the quality of the parallel implementation of the sparse matrix code.
3. GPU Accelerated Sparse Matrix Matrix Multiplications

<table>
<thead>
<tr>
<th></th>
<th>Amorph</th>
<th>H₂O</th>
<th>TiO₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of atoms</td>
<td>13846</td>
<td>20736</td>
<td>9786</td>
</tr>
<tr>
<td>number of basis functions</td>
<td>133214</td>
<td>158976</td>
<td>169624</td>
</tr>
<tr>
<td>block sizes</td>
<td>5, 13</td>
<td>23, 13</td>
<td>26, 13</td>
</tr>
<tr>
<td>number of SCF steps in benchmark</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>filtering threshold</td>
<td>10⁻⁶</td>
<td>10⁻⁶</td>
<td>10⁻⁵</td>
</tr>
<tr>
<td>typical matrix occupation %</td>
<td>16</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>run time on 169 x 2 SB [s]</td>
<td>372</td>
<td>275</td>
<td>446</td>
</tr>
<tr>
<td>run time on 169 x 1 SB + 1 K20X [s]</td>
<td>272</td>
<td>187</td>
<td>263</td>
</tr>
<tr>
<td>performance ratio on 169 nodes</td>
<td>1.4</td>
<td>1.5</td>
<td>1.7</td>
</tr>
<tr>
<td>GPU flop %</td>
<td>92</td>
<td>99</td>
<td>88</td>
</tr>
</tbody>
</table>

Table 3.1: Key quantities of three linear scaling benchmarks that are distributed with CP2K. The run time is provided for two setups, one in which 2 Sandy Bridge (SB) CPUs are present per node, and a hybrid architecture in which 1 SB and 1 K20X GPU is present per node. Performance ratio compares the run time between these setups. GPU flop % gives the percentage of flops that is executed on the GPU in the hybrid setup.

More important is application level performance for realistic simulation setups. In order to assess this, we employ three benchmarks that are also part of the CP2K distribution, named amorph, H₂O and TiO₂. These describe an amorphous organic hole conducting material, bulk liquid water and Titanium dioxide nanoparticles respectively. Geometries are realistic, disordered, three dimensional, and with periodic boundary conditions. Basis sets are of double zeta quality (DZVP-MOLOPT-SR-GTH) and include diffuse primitives, contraction based on molecular optimization makes them at the same time accurate and suitable for linear scaling calculations in the condensed phase [62]. Since these benchmarks are designed to run quickly on a relatively small number of compute nodes they have reduced SCF counts. Key quantities and results are provided in Table 3.1. First, for the given basis sets and thresholds, it shows that systems of approximately 10000 atoms can be computed in minutes using 169 nodes (only 4% of the national supercomputer). This opens the way for scientific applications based on models of this size, including geometry optimization and molecular dynamics based relaxation. Second, this comparison at 169 nodes shows a speedup of 1.37–1.70 going from a traditional homogeneous node to a hybrid node. In these cases, the GPU is processing most of the flops. More detailed analysis shows that for the amorph benchmark the small block sizes limit the speedup, while the H₂O testcase is already limited by MPI communication.

The largest system computed so far on the hybrid system Piz Daint is shown in Fig. 3.8. It consists of aggregated nanoparticles of TiO₂ in an explicit acetonitrile solvent, as found in dye sensitized solar cells, and requires 77538 atoms and 772868 basis functions. For a filtering threshold of 10⁻⁶ a matrix occupation of 4% is found. Running on 5184 nodes, a single SCF step takes
3.5. Benchmarks and Conclusions

Figure 3.8: Aggregated nanoparticles in explicit solution (77538 atoms) can be run on the Piz Daint computer (5272 hybrid compute nodes) at approx. 122s per SCF step.

approximately 122s. Performance is roughly 30 Gflops per node, as the calculation is strongly dominated by MPI communication. The GPUs perform 99.4% of the flops.

To conclude, we have shown that linear scaling SCF calculations with good quality on large three dimensional systems have become possible with good time to solution. As such, linear scaling approaches on large models have become one of the many tools that atomistic simulation offers to investigate an ever increasing range of systems. The progress can be attributed to an evolution and interplay between hardware, algorithms and implementations. The GPU work presented here is a prime example.
Chapter 4

Machine Learning Adaptive Basis Sets

4.1 Introduction

The rapid increase in computational power and the development of linear scaling methods [16, 17] now allows for easy single-point DFT energy calculations of systems with 10,000 - 1,000,000 atoms [7, 63]. However, the approach is computationally demanding for routine application, especially if first principles molecular dynamics or relaxation is required. The computational cost of a DFT calculation depends critically on the size and condition number of the employed basis set. Traditional linear scaling DFT implementations employ basis sets which are atom centered, static, and isotropic. Since molecular systems are never isotropic, it is apparent that isotropic basis sets are sub-optimal. Therefore, in this work a scheme is presented for small adaptive basis sets, which adjust themselves to the local chemical environment. These chemical environments are subject to change, e.g. during the aforementioned relaxations or samplings. In order to map chemical environments to basis functions in a predictable fashion, a machine learning (ML) approach is used. The analytic nature of a ML framework allows for the calculation of exact analytic forces, as they are required for dynamic simulations.

The idea of representing a chemical environment with adapted atomic or quasi-atomic basis functions dates back several decades. It underlays e.g. many early tools used for the investigation of bonding order [64, 65, 66, 67, 68, 69]. Also more recent methods for extracting atomic orbitals from molecular orbitals build on this idea [70, 71, 72, 73, 74, 75]. Besides using adaptive basis sets for analytic tasks, they can also be used to speed up SCF algorithms, which was pioneered by Adams [76, 77, 78]. The approach was later refined by Lee and Head-Gordon [79, 80] and subsequently applied to linear scaling DFT by Berghold et al. [81]. Many linear scaling DFT
packages have also developed their own adaptive basis set scheme: The CONQUEST program [63] uses local support functions, derived either from plane waves [82] or pseudo-atomic orbitals [83]. The ONETEP program [84] uses non-orthogonal generalized Wannier functions (NGWFs) [85]. The BigDFT program [86] uses a minimal set of on-the-fly optimized contracted basis functions [87]. Other related methods include numeric atomic orbital [88, 89, 90, 91] and localized filter diagonalization [92, 93, 94, 95, 96]. Recently Mao et al. used perturbation theory to correct for the error introduced into a DFT calculation by a minimal adaptive basis [97].

Here, we focus on polarized atomic orbitals (PAOs) and build on the work of Berghold et al. [81]. PAOs are linear combinations of atomic orbitals (AOs) on a single atomic center, called primary basis in the following, that minimize the total energy when used as a basis. As a result, small PAO basis sets are usually of good quality. In fact, minimal PAO basis sets have been studied in most detail. The variational aspect is advantageous when computing properties, such as e.g. nuclear gradients. PAO basis sets can be increased in quality in two ways. First, as the flexibility of the primary basis set is increased, the resulting PAO basis sets improve. In fact, if the primary basis on each atom were complete, a minimal basis PAO would yield the complete basis set result. In that case, PAOs could be made similar to the maximally localized Wannier functions of the system, which are a small, complete and localized basis. In practice, typical primary basis sets are far from complete, and this limit is not reached. Additionally, optimizing the PAO basis becomes more challenging as the primary basis increases in size. A second, more convenient, approach is to increase the size of the PAO basis. Indeed, there is no reason to keep this basis minimal, and as we will show below, a modest increase in size can significantly improve the quality. Of course, as the PAO basis approaches the dimensions of the primary basis, so will its quality. It is noteworthy that for small molecules PAO basis sets of small size can exactly reproduce the energy obtained with the primary basis set, as long as the number of PAO functions per atom equals the number of occupied orbitals. For example, a PAO basis with four functions on Oxygen and Hydrogen will exactly reproduce the energy of a water molecule, if we consider valence electrons only. However, despite these qualities, the use of PAOs in simulation has been very limited, which we attribute to the difficulty of optimizing these PAOs for each molecular geometry. As we will show, especially for non-minimal PAOs or with flexible primary basis sets, the optimization procedure converges relatively slowly. Our aim is to exploit the adaptivity of the PAO basis, but to avoid this tedious optimization step by a machine learning approach.
4.2 Polarized Atomic Orbitals

In the following sections, a detailed description of polarized atomic orbitals (PAO) basis sets will be provided, as well as various schemes to parameterize and optimize them. A PAO basis is derived from a primary basis via a linear transformation. This linear transformation is performed for each atom independently and can be seen as a unitary transformation (rotation) of the primary basis set, followed by a truncation to a smaller subset of basis functions. Various parameterizations of this unitary transformation can be employed, each with advantages depending on the specific purpose. Here, two different parameterizations will be presented. The first is a traditional parameterization using a matrix exponential. This parameterization is complete and minimal, and is suitable to test the PAO method in general. However, the values of the optimal parameters are not invariant under rotation of the molecule being studied, or more precisely, depends on the Cartesian directions in space that are employed in the definition of the primary basis. This makes this parameterization inconvenient for machine learning. The second parameterization is derived in this work, and is aimed at providing a rotational invariance. In this so called potential parameterization the relative positions of neighboring atom define the directions in space, making the parameters independent of the orientation of the molecule. The parameterization derives the unitary transformation from the eigenvectors of an auxiliary atomic Hamiltonian for the central atom that is perturbed by a potential originating from the neighboring atoms. The parameters of the potential are the variables that can be optimized and later machine learned. The challenge of this parameterization is related to the fact that it is neither guaranteed to be complete nor minimal. One practical scheme, which consists of a linear expansion and various regularization terms, is presented and tested in the following sections.

4.2.1 PAO Basis Sets

The polarized atomic orbital basis is derived from a larger primary basis through linear combinations among functions centered on the same atom. In the following, the notation from Berghold et al. [81] has been adopted. Variables with a tilde denote objects in the smaller PAO basis, while undecorated variables refer to objects in the primary basis. Formally, a PAO basis function \( \tilde{\varphi}_\mu \) can be written as a weighted sum of primary basis functions \( \varphi_\nu \), where \( \mu \) and \( \nu \) belong to the same atom:

\[
\tilde{\varphi}_\mu = \sum_\nu B_{\mu\nu} \varphi_\nu
\]  

(4.1)
As a consequence of the atom-wise contractions, the transformation matrix $B$ assumes a rectangular block-diagonal structure:

$$B = \begin{pmatrix} B_1 & 0 \\ \vdots & \ddots \\ 0 & B_{N_{\text{atom}}} \end{pmatrix}$$

Since the primary basis is non-orthogonal, the tensor property of the involved matrices has to be taken into account [98]. Covariant matrices like the Kohn-Sham matrix $H$ and the overlap matrix $S$ transform differently than the contravariant density matrix $P$. Hence, two transformations matrices $A$ and $B$ are introduced.

$$PAO \leftarrow \text{Primary} \quad \text{Primary} \leftarrow PAO$$

$$\tilde{P} = A^T PA \quad P = B \tilde{P} B^T \quad \text{Contravariant}$$

$$\tilde{H} = B^T HB \quad H = A \tilde{H} A^T \quad \text{Covariant}$$

$$\tilde{S} = B^T SB \quad S = A \tilde{S} A^T \quad \text{Covariant} \quad (4.2)$$

Notice that, $A^T B = B^T A = I$ gives the identity matrix in the PAO basis, while $AB^T = BA^T$ is the projector onto the subspace spanned by the PAO basis within the primary basis. In order to treat the matrices $A$ and $B$ in a simple and unified fashion, they are rewritten as a product of three matrices:

$$B = NUY \quad A = N^{-1} UY \quad (4.3)$$

Due to the atom-wise contractions, the matrices $N$, $U$, and $Y$ are block-diagonal as well. The matrices $N^{\pm 1}$ transform into the orthonormal basis, in which co- and contra-variance coincides and the distinction can be dropped. The unitary matrix $U$ rotates the orthonormalized primary basis functions of each atom such that the desired PAO basis functions become the first $m_I$ components. The selector matrix $Y$ is a rectangular matrix, which selects for each atom the first $m_I$ components. Each atomic block $Y_I$ of the selector matrix is a rectangular identity matrix of dimension $n_I \cdot m_I$, where $n_I$ denotes the size of the primary basis and $m_I$ the PAO basis size for the given atom $I$:

$$Y_I^T = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 1 & 0 & \ldots & 0 \end{pmatrix}$$

In the formulation from equation (4.3) the PAO basis is now solely determined by the unitary diagonal blocks of matrix $U$, without any loss of generality.

The matrix $N^{-1}$ ($N$) is responsible for orthonormalizing the basis functions belonging to the same atom. Hence, it contains the (inverse) square root
of the diagonal blocks of the overlap-matrix $S$ in the primary basis. None of the matrix multiplications required in the transformation is expensive to compute, since the matrices are either diagonal, or expressed in the small PAO basis.

### 4.2.2 PAO Optimization

In order to find the optimal PAO basis, an intricate minimization problem needs to be solved. In particular, the total energy must be minimal with respect to the rotation matrix $U$, and the density matrix $P$. During this minimization, the electronic density, which influences the Kohn-Sham matrix $H$, changes, making the optimization non-linear, or self-consistent. Additionally, the optimization must yield an idempotent density matrix ($PSPS = PS$), which is commonly enforced using constrained optimization.

Li, Nunes, and Vanderbilt (LNV) have developed an unconstrained formulation of the electronic-structure problem, which includes the idempotency constraint directly in the energy functional [54]:

$$
\Omega(P) = 3 \text{tr}[PSPH] - 2 \text{tr}[PSPSPH]
$$

(4.4)

The LNV functional $\Omega$ does not yield the Kohn-Sham total energy, but it has the same variational solution $P$ as the original Kohn-Sham functional. Given its unconstrained nature, it is a convenient formulation to optimize the PAO basis.

In order to employ the LNV functional within the PAO context, $P$ is expressed as a function of $\tilde{P}$, using equation (4.2), since it is $\tilde{P}$ that is computed in the PAO basis. After permuting the arguments of the traces in the functional, this is equivalent to computing the same LNV functional with quantities in the PAO basis:

$$
\Omega = 3 \text{tr}[PSPH] - 2 \text{tr}[PSPSPH]
$$

(4.5)

This functional must be variationally optimized with respect to $U$, which is the determining factor in the transformation matrix $B$ and corresponds to a change in PAO basis, and with respect to $\tilde{P}$, which corresponds to an optimization of the electron density.

Various optimization schemes are conceivable and we discuss two earlier approaches, as well as a new scheme. In their original publication Lee and Head-Gordon [79] chose to optimize $U$ and $\tilde{P}$ simultaneously by performing joined optimization steps in both variables along $\partial \Omega / \partial U$ and $\partial \Omega / \partial \tilde{P}$. After
each step they recalculated the Kohn-Sham matrix $H$. In a well preconditioned setup, the combined optimization of $U$ and $\tilde{P}$ might require relatively few optimization steps. One disadvantage of this scheme are possible instabilities if a good starting point for the PAO basis or $\tilde{P}$ is not available. Indeed, the LNV functional gives rise to run-away solutions for density matrices with eigenvalues $\lambda_p$ outside of the convergence interval $[-0.5, 1.5]$. Additionally, the recomputation of $H$ at each minimization step can be relatively expensive.

Berghold et al. [81] decided to replace the LNV-based optimization of $\tilde{P}$ with a more efficient method. In their scheme, a single optimization step along $\partial\Omega/\partial U$ is followed by a full optimization of $\tilde{P}$ while keeping $U$ fixed. By separating the optimization of $\tilde{P}$ and $U$, they could conveniently reuse existing implementations for self consistent field (SCF) optimization. During an SCF optimization the Kohn-Sham matrix $H$ is recalculated at each optimization step of $\tilde{P}$. The drawback of the scheme presented by Berghold et al. [81] is the high number of Kohn-Sham matrix constructions, as each update of $U$ triggers a full SCF.

Here, an optimization scheme is proposed which relaxes the self-consistency by keeping the Kohn-Sham matrix $H$ fixed during the optimization of $\tilde{P}$ and $U$. This allows for a significant speedup, in particular for calculations on smaller training systems that are dominated by the Kohn-Sham matrix updates. In this novel scheme at every optimization step along $\partial\Omega/\partial U$ the fixed Kohn-Sham matrix $H$ is transformed into the current PAO basis, yielding $\tilde{H}$, which serves as input for the TRS4 algorithm to calculate the corresponding optimal $\tilde{P}$. Only after the optimization of $U$ and $\tilde{P}$ has converged, an SCF optimization is performed to update the Kohn-Sham matrix $H$. The optimization of $U$ and $\tilde{P}$ is alternated with the optimization of $\tilde{P}$ and $H$ until overall convergence is achieved. The entire optimization scheme is illustrated in figure 4.1. Before launching into the first PAO optimization cycle an initial guess for $H$ has to be generated. Naturally, a very good initial guess can be obtained from an SCF optimization in the primary basis. This is exactly the operation that should be avoid later on. However, for the generation of training points much smaller systems can be used, which makes this step affordable and an effective choice.

For each step of the PAO optimization, equation (4.2) is used to transfer the fixed Kohn-Sham matrix $H$ from the primary into the current PAO basis, where it is denoted by $\tilde{H}$. The corresponding density matrix $\tilde{P}$ is obtained through purification of $\tilde{H}$, using the TRS4 algorithm [53]. The advantage of the TRS4 algorithm is its reliable retention of the electron number, which is very import when the initial PAO basis is unphysical. The implementation of TRS4 in CP2K requires an orthonormal basis. Therefore, the overlap matrix $S$ is also transformed into the PAO basis and $S^{\pm1/2}$ are calculated
4.2. Polarized Atomic Orbitals

Initial guess for Kohn-Sham matrix $H$

PAO optimize $\tilde{P}$ and $U$
using fixed $H$

Mixing of PAO basis with previous iteration

Converged?
Variational result reached

Yes

No

SCF optimize $\tilde{P}$ and $H$
using fixed $U$

Figure 4.1: Optimization scheme used for finding variational PAO basis

using Newton-Schulz iterations [99]. The PAO basis is then optimized by minimizing the LNV energy functional $\Omega$ with respect to the unitary matrix $U$, as detailed in the next section.

The convergence of the PAO optimization is determined based on the norm of the gradient $\frac{\partial \Omega}{\partial U}$. The overall optimization is terminated when the PAO optimization converges within two steps. The alternating optimization of basis and density matrix might give rise to an oscillatory, slow convergence behavior. These oscillations are dampened with a mixing of the PAO basis in between optimization cycles. A further speedup is achievable by terminating the PAO optimization after a preset number of steps, typically a few hundred. This effectively reduces the accuracy of the intermediate PAO basis, which is tolerable during the early phase of the optimization. Towards the end of the overall optimization, the PAO optimization part converges within a few steps anyways.

4.2.3 PAO Gradient

An efficient optimization of the PAO basis requires the derivative of the total energy with respect to the rotation matrix $U$, which defines the PAO basis. Since the LNV functional and the total energy share the same variational minima, it can be used to calculate the gradient. To this end, the chain rule has to be used extensively: First the LNV functional $\Omega$ from equation (4.5) has to be derived with respect to the PAO matrices $\tilde{P}$, $\tilde{H}$, and $\tilde{S}$. Then the
derivative of the PAO matrices with respect to the transformation matrices \( A \) and \( B \) can be calculated by using equations (4.2) to express them in the primary basis. Finally, the relation of the transformation matrices to the rotation matrix \( U \) is provided by their definition in equation (4.3). Hence, the derivative of the LNV functional with respect to the rotation matrix \( U \) follows as:

\[
M_{ij} = \frac{d\Omega}{dU_{ij}} = \sum_{ab} \frac{\partial \Omega}{\partial \tilde{P}_{ab}} \frac{\partial \tilde{P}_{ab}}{\partial U_{ij}} + \sum_{ab} \frac{\partial \Omega}{\partial \tilde{H}_{ab}} \frac{\partial \tilde{H}_{ab}}{\partial U_{ij}} + \sum_{ab} \frac{\partial \Omega}{\partial \tilde{S}_{ab}} \frac{\partial \tilde{S}_{ab}}{\partial U_{ij}} = \sum_{ab} M_{1,ab} \frac{\partial \tilde{P}_{ab}}{\partial U_{ij}} + \sum_{ab} M_{2,ab} \frac{\partial \tilde{H}_{ab}}{\partial U_{ij}} + \sum_{ab} M_{3,ab} \frac{\partial \tilde{S}_{ab}}{\partial U_{ij}}
\]

\[
M_1 = \frac{\partial \Omega}{\partial \tilde{P}} = 3\tilde{H}\tilde{P}\tilde{S} + 3\tilde{S}\tilde{P}\tilde{H} - 2\tilde{H}\tilde{P}\tilde{S}\tilde{P} - 2\tilde{S}\tilde{P}\tilde{S}\tilde{H}
\]

\[
M_2 = \frac{\partial \Omega}{\partial \tilde{H}} = 3\tilde{P}\tilde{S}\tilde{P} - 2\tilde{P}\tilde{S}\tilde{P}\tilde{P}
\]

\[
M_3 = \frac{\partial \Omega}{\partial \tilde{S}} = 3\tilde{P}\tilde{H}\tilde{P} - 2\tilde{P}\tilde{H}\tilde{P}\tilde{P} - 2\tilde{P}\tilde{S}\tilde{P}\tilde{H}
\]

\[
M = \frac{d\Omega}{dU} = 2N^{-1T}PAM_1Y^T + 2N^THBM_2Y^T + 2N^TSBM_3Y^T
\]

In the above derivation some intermediate quantities, such as \( \partial \tilde{P}_{ab}/\partial U_{ij} \), carry four indices. One can avoid to deal with four indexes explicitly by performing the calculation in “reverse” and successively applying the chain rule to immediately sum up two of the four indexes.

Note that the gradient in (4.6) does not require any multiplications between two matrices from the primary basis, which makes its computationally efficient. Furthermore, using sparse linear algebra, this step scales linear with systems size. This is important, because this gradient will later also be used to calculate forces, e.g. for molecular dynamics simulations of large systems.

### 4.2.4 Exponential Parameterization

The PAO basis is determined by the unitary matrix \( U \). The unitariness imposes a constraint upon the optimization of \( U \). This constraint can be absorbed by a suitable parameterization of \( U \). In the following, two possible parameterizations of unitary matrices are presented. Since the matrix \( U \) has a block diagonal structure, the discussion will focus only on a single block, which constitutes a unitary matrix by itself.

The simplest way to parameterize a unitary matrix is to use the matrix ex-
4.2. Polarized Atomic Orbitals

ponential of a skew-Hermitian matrix:

\[ U = \exp(X'), \quad \text{with } X' = \begin{pmatrix} 0 & X \\ -X^T & 0 \end{pmatrix} \]  

(4.7)

The PAO basis set is only affected by rotations which move a basis function in or out of the first \( m \) selected components. Hence, the diagonal blocks of matrix \( X' \), ranging from 1 to \( m \) and from \( m \) to \( n \) can be fixed to zero. The remaining off-diagonal block is denoted by \( X \) in equation (4.7). This leaves \( m \cdot (n - m) \) free parameters for the optimization, which is the lowest possible number, i.e., the exponential parameterization is minimal.

The gradient \( \partial U / \partial X' \) can be calculated by diagonalizing the matrix \( X' \), yielding the eigenvector-matrix \( R \) and the corresponding eigenvalues \( \lambda_i \). Then, using the outer derivative \( M \) from (4.6), the gradient can be constructed in the following way:

\[ D_{ij} = \begin{cases} e^{\lambda_i}, & i = j \\ \frac{e^{\lambda_i} - e^{\lambda_j}}{\lambda_i - \lambda_j}, & i \neq j \end{cases} \]

\[ F = (R^\dagger M^T R) \otimes D \]

\[ G' = \frac{\partial \Omega}{\partial X'} = RFR^\dagger - R^\dagger F^T R \]  

(4.8)

where \( \otimes \) denotes the Hadamard product. The gradient \( G = \partial \Omega / \partial X \) corresponds to the off-diagonal block of matrix \( G' \).

The exponential parameterization has the nice properties of being simple and minimal. However, it lacks rotationally invariance. A rotationally invariant parameterization yields the same optimal \( X \) regardless of the system’s orientation. The exponential parameterization inherits its orientation from the primary basis set, which itself is based on the underlying Cartesian system. Consequently, the optimization of matrix \( X \) yields different results depending on the orientation of the atoms relative to the simulation cell. The main drawback of this arises in the context of the machine learning approach, where the amount of training data needed would be large because all orientations would have to be covered. Poor learning quality would for example result in the emergence of artificial torque forces. How to establish a rotationally invariant parameterization is discussed in the following section.

4.2.5 Potential Parameterization

In order to achieve rotational invariance, the unitary transformation matrix \( U \) has to be parameterized in a new way. In the potential parameterization
this proceeds in two steps. First, a potential matrix $V$ is introduced, employing all matrix elements as parameters. Second, the matrix elements are expressed as potential terms that depend on the neighboring atoms, which introduces the rotational invariance. Figure 4.2 provides an overview of the PAO optimization with the potential parameterization.

**Parameterization by Potential Matrix Elements**

In order to parameterize the rotation matrix $U$ by a potential matrix $V$, an auxiliary atomic Hamiltonian $H_{aux}$ is constructed and subsequently diagonalized. The eigenvectors of this diagonalization correspond to the rotation matrix $U$, or more chemically, the lowest $m$ states of the auxiliary Hamiltonian are the PAO basis functions:

$$H_{aux} = N(H_0 + V)N$$

$$H_{aux} = U \text{diag}\left(\{\lambda_i\}\right) U^\dagger$$

Here, the atomic Hamiltonian $H_0$ describes the isolated spherical atom, so that in the absence of $V$, the PAO basis will reproduce this system exactly. This form of the potential parameterization is without loss of generality, and can reproduce exactly the solutions found by the exponential parameterization. However, contrary to the exponential parameterization, it is not
4.2. Polarized Atomic Orbitals

unique and several matrices $V$ can lead to the same PAO basis sets. Indeed, the symmetric matrix $V$ has $n(n+1)/2$ free parameters, which are far more than the minimal $m \cdot (n-m)$ parameters. These additional degrees of freedom merely lead to rotations among the selected and unselected states, or shift the eigenvalues of the eigenstates.

We exploit this in the gradient for the potential parameterization, which is written as:

$$D_{ij} = \begin{cases} 0, & (i \leq m) \text{ and } (j \leq m) \\ 0, & (i > m) \text{ and } (j > m) \\ \frac{1}{\lambda_i - \lambda_j}, & \text{otherwise} \end{cases}$$

$$F = (M^T U) \otimes D$$

$$\frac{\partial \Omega}{\partial V} = \frac{1}{2} N (UFU^t + U^t F U) N,$$  \hspace{1cm} (4.9)

where corresponding contributions are removed from $D$, avoiding divergence in case of degenerate eigenvalues in selected or unselected block of states. Nevertheless, the excess degrees of freedom can lead to undesired side-effects of the potential matrix parameterization which are addressed in the following.

**Avoiding a Closing Gap.** The first problem relates to the fact that the subspace spanned by the lowest $m$ eigenstates can change very rapidly (discontinuously) as a function of the potential matrix $V$, when the gap between the highest selected eigenvalue $\lambda_m$ and the lowest unselected eigenvalue $\lambda_{m+1}$ closes. This is reflected in a divergence of the gradient (Eq. 4.9) in that case. To avoid this problem in the optimization of the matrix $V$, a repulsive term is added to the objective function that is short ranged and keeps selected and unselected eigenvalues apart:

$$f_{gap}(\Delta) = \alpha \exp \left[ - \left( \frac{\Delta}{\beta} \right)^2 \right]$$

$$E_{gap} = \sum_{i=1}^{m} \sum_{j=m+1}^{n} f_{gap}(\lambda_i - \lambda_j)$$  \hspace{1cm} (4.10)

Used values for the strength and range of $f_{gap}$ are $\alpha = 0.005 \text{ Ha}$ and $\beta = 0.1 \text{ Ha}$. Even in presence of this additional term, the optimal $V$ will still exactly reproduce the result of the exponential parameterization, given the excess degrees of freedom in $V$. 

53
4. Machine Learning Adaptive Basis Sets

The gradient contribution from $E_{\text{gap}}$ follows as:

$$D_{\text{gap},ij} = \begin{cases} 
\sum_{k=1}^{m} f'_{\text{gap}}(\lambda_i - \lambda_k), & (i = j) \land (m < i) \\
\sum_{k=m+1}^{n} f'_{\text{gap}}(\lambda_i - \lambda_k), & (i = j) \land (i \leq m) \\
0, & i \neq j
\end{cases}$$

$$F = (M^T U) \otimes D_{\text{gap}}$$

$$\frac{\partial E_{\text{gap}}}{\partial V} = \frac{1}{2} N (U F U^T + U^T F^T U) N \quad (4.11)$$

**First Potential Regularization.** The non-uniqueness of the optimal matrix $V$, is problematic in the context of machine learning, as very similar configurations can have very different matrices $V$, which hinders the interpolation between these points. In order to regularize towards a unique solution that varies smoothly between configurations, the following $L_2$ regularization term was added:

$$E_{\text{reg}1} = c \cdot \text{tr}[V^T V] \quad (4.12)$$

This regularization introduces a deviation of $V$ from the optimal solution, and can thus not be selected arbitrarily. The used value for the regularization strength is $c = 0.001 \text{ Ha}$.

The corresponding contribution to the gradient is

$$\frac{\partial E_{\text{reg}1}}{\partial V} = 2 c V \quad (4.13)$$

**Parameterization by Potential Matrix Terms**

The parameterization by potential matrix elements is not rotational invariant by itself. However, it allows for making a second step, namely the expansion of this matrix in terms that follow the rotation of the system. Naturally, this results from representing these terms in the primary basis as matrices $V_i$ obtained from potentials that are located on neighboring atoms, multiplied by new variables $X_i$:

$$V = \sum_i X_i V_i \quad (4.14)$$

This can be seen as a linear expansion using $V_i$ as basis to represent the matrix $V$. With this choice, the representation of the potentials $V_i$ changes as the system is rotated, but the value of the variables $X_i$ remains the same. As such a rotationally invariant parameterization has been obtained.

The gradient in the potential parameterization follows as:

$$G_i = \frac{\partial \Omega}{\partial X_i} = \text{tr} \left[ V_i^T \cdot \frac{\partial \Omega}{\partial V} \right] \quad (4.15)$$
4.2. Polarized Atomic Orbitals

The number of terms in equation (4.14), as well as the precise nature of $V_i$ remain to be specified, and are discussed in the following.

**Second Potential Regularization.** The linear expansion in equation (4.14) must be sufficiently flexible to represent the optimal matrix $V$, as would be obtained with the direct potential parameterization (section 4.2.5). In practice, a relatively large basis is needed, and it is difficult to keep the potential terms $V_i$ linearly independent. In the presence of near linear dependencies, the expansion coefficients $X_i$ become non-unique, and a second regularization term is needed. This regularization is inspired by Tikhonov regularization\[100\], and aims to limit the magnitude of $X_i$ by penalizing expansion on linearly dependent potential terms. It is based on the overlap matrix of the potential terms:

$$V_{ij} = \text{tr} \left[ V_i^T V_j \right]$$ (4.16)

as

$$R = \beta \min \left[ 1, \frac{\alpha}{\sqrt{V}} \right]$$

$$E_{\text{reg2}} = \sum_{ij} X_i R_{ij} X_j$$ (4.17)

where $\min \left[ 1, \frac{\alpha}{\sqrt{V}} \right]$ denotes an inverse were the smallest eigenvalues are capped, such that eigenvalues below $\alpha$ lead to a maximum penalty $\beta$. Used values are $\alpha = 10^{-6}$ and $\beta = 0.001$ Ha. This regularization, in particular $\beta$, affects the quality of the parameterization, the influence of which is explored in section 4.4.4.

The corresponding gradient contribution reads:

$$\frac{\partial E_{\text{reg2}}}{\partial X_i} = 2 \sum_j R_{ij} X_j$$ (4.18)

**Explicit Form of the Potential Terms.** The explicit form of the potential terms $V_i$ must be sufficiently flexible to span the relevant subspace. Yet, they must also depend smoothly on atomic positions, be independent of the atom ordering, and be sufficiently local in nature. While we expect that more advanced forms can be found, the following scheme has been employed:

$$(V_i)_{\mu\nu} = (V_{kuv})_{\mu\nu} = \int P_u \Phi^*_\mu(\vec{r}) \Phi_k(\vec{r}) P_\nu \phi_\nu(\vec{r}) d\vec{r}$$ (4.19)

where

$$\Phi_k(\vec{r}) = \sum_{j \neq l} w_j \cdot (\vec{r} - \vec{R}_j)^{2k} \cdot \exp \left\{ -\beta_j (\vec{r} - \vec{R}_j)^2 \right\}, \quad k = 0, 1, 2, \ldots$$ (4.20)
is a potential that results from spherical Gaussian potentials centered on all neighboring atoms. The weights $w_J$ and exponents $\beta_J$ can depend on the atomic kind of the neighboring atom $J$, e.g. its electron negativity or van der Waals radius, and $k$ introduces additional flexibility, the used value was $k = 2$.

$$P_u \phi_v = \begin{cases} \phi_v, & \phi_v \in \text{shell } u \\ 0, & \text{otherwise} \end{cases}$$

is a projector on shells of basis functions that share a common radial part, and the same angular momentum number $l$, but have different $m$ quantum numbers. Specializing the terms by different $l$ quantum number and radial part, introduces the needed flexibility, while retaining the rotational invariance. Non-local pseudopotentials have some resemblance to this scheme. Finally, additional terms are added that just result from the central atom, these are give by

$$\left(V_{uv}\right)_{\mu\nu} = \begin{cases} P_u P_v \delta_{\mu\nu}, & l_u = l_v \\ 0, & \text{otherwise} \end{cases}$$

Note that trivially degenerate terms with $l_u = l_v = 0$ are included only once.

**Optimization and Preconditioning**

Combining the regularization terms from the previous sections with the original LNV function yields the following full energy functional:

$$\mathcal{L} = \Omega + E_{\text{gap}} + E_{\text{reg}} + E_{\text{reg2}}$$

The variational PAO basis is obtained from optimizing the functional $\mathcal{L}$ with respect to the parameters $\vec{X}$ and the density matrix $\tilde{P}$. This optimization is performed using the Conjugate Gradient (CG) algorithm for $\vec{X}$, and TRS4 for $\tilde{P}$. Figure 4.2 shows an overview of the optimization procedure using the potential parameterization.

The regularizations introduced above ensures a unique $\vec{X}$ of small magnitude. However, they do not remedy the fact that the optimization of these degrees of freedom is still poorly conditioned. Therefore, the overlap matrix of the potential terms $V$ from equation (4.16) is used to construct the preconditioner $V^{-1/2}$. To protect against the small eigenvalues in $V$, the spectrum was capped at $\delta = 0.01$:

$$T = (\max [\delta, |V|])^{-1/2}$$
The preconditioner $T$ is then incorporated into the CG algorithm in the following way:

$$G'_t = TG_t$$

$$\beta_t = \frac{\text{tr}[G'_t G'_t^T] - \text{tr}[G'_t G'_{t-1}^T]}{\text{tr}[G'_{t-1} G'_{t-1}^T]}$$

$$D'_t = \beta_t D'_{t-1} - G'_t$$

$$D_t = TD'_t$$

$$X_{t+1} = X_t + \alpha_t D_t$$

The CG mixing parameter $\beta_t$ is calculated using the Polak–Ribière formula. It is reset whenever $\beta_t < 0$ or the new gradient $G'_t$ has moved too far away from the previous search direction $D'_{t-1}$ such that the cosine of their angle exceeds 0.1. The step-length $\alpha_t$ is determined via a one-dimensional line search. To obtain a robust and yet efficient line search, a three-point interpolation was combined with golden bisection as fallback. While the three-point search is sufficient for the vast majority of cases, the golden bisection guarantees that the line search always finds a lower energy, even e.g. when crossing a saddle point.

### 4.3 Machine Learning

Machine learning, simply put, means to approximate a complex, possibly unknown, function from a given set of training points. After an initial training, the learning machinery can be queried for any point, from within the training realm, and it should return a “reasonably” interpolated function value. The amount of required training data grows with the difficulty of the learning problem. Therefore, one tries to keep the learning problem as small as possible by exploiting a priori knowledge about the function’s domain and codomain.

In the context of PAO, this simplification is achieved through a so-called descriptor and the previously described potential parameterization. The descriptor takes as input all atom positions and returns a low-dimensional feature vector: $\{ \vec{R}_i \} \rightarrow \vec{q}$. The potential parameterization takes as input a PAO parameter vector and returns the unitary matrix that eventually determines the PAO basis: $\vec{X} \rightarrow U$. Hence, the learning machinery only has to perform the mapping of feature-vectors onto PAO parameter vectors: $\vec{q} \rightarrow \vec{X}$. Figure 4.3 provides an overview of how the PAO-ML scheme calculates the transformation matrices from given atomic positions using the potential parameterization.

A number of different learning methods have been proposed, including neural networks [101] and regression [31]. For this work a Gaussian Processes
4. Machine Learning Adaptive Basis Sets

For each atom

- Descriptor: \( \vec{q} = \text{fingerprint}(\{ \vec{R}_I \}) \)
- Machine Learning: \( \vec{X} = \text{prediction}(\vec{q}) \)
- Potential Terms: \( V_{i\alpha,\beta} = \langle \phi_\alpha | \Phi_i(\{ \vec{R}_I \}) | \phi_\beta \rangle \)
- Auxiliary Hamiltonian: \( H_{\text{aux}} = H_0 + V \)
- Perturbation Potential: \( V = \sum_i X_i \cdot V_i \)
- Diagonalization: \( U = \text{eigenvectors}(H_{\text{aux}}) \)
- PAO Transformation: \( B = N \begin{pmatrix} U_1 & \cdots & U_N \end{pmatrix} Y \)
- Atom Positions: \( \{ \vec{R}_I \} \)

Figure 4.3: Overview of the PAO-ML scheme for using the potential parameterization and machine learning to calculate the PAO basis from given atomic positions.

[25] was chosen, because it provides variances alongside its predictions. However, no assumptions were made about the employed algorithm, which means the PAO-ML scheme can also be used in combination with any other machine learning method.

4.3.1 Forces

In order to run molecular dynamics simulations, accurate forces are essential. Forces are the derivative of the total energy with respect to atom positions. While a variationally optimized PAO basis does not contribute any additional force terms, the same does not hold for approximate PAO basis sets. The advantage of using a pre-trained machine learning scheme, as compared to an on-the-fly scheme, is the possibility to calculate accurate forces nevertheless.

The PAO-ML scheme contributes two force terms that have to be added to the common DFT-forces \( \vec{F}_{\text{DFT}} \). One term originates from the potential terms \( V_i \) from equation (4.19), which are anchored on neighboring atoms. The other force term arises from the descriptor, which takes atom positions as input. For the calculation of these additional force terms, the previously derived gradients \( \partial \Omega / \partial V \) and \( \partial \Omega / \partial \vec{X} \) can be reused:

\[
\vec{F} = -\frac{dE}{d\vec{R}} = \vec{F}_{\text{DFT}} - \frac{\partial \Omega}{\partial V} \frac{\partial V}{\partial \vec{R}} - \frac{\partial \Omega}{\partial \vec{X}} \frac{\partial \vec{q}}{\partial \vec{R}} \quad (4.25)
\]

The entire scheme for performing MD simulations with PAO-ML is shown in figure 4.4.
4.3.2 Gaussian Processes

A Gaussian process [25] is a generalization of the Gaussian probability distribution. While a probability distribution describes a random variable, which is a scalar or vector, a stochastic process describes a function. The shape of the function is inferred from a given set of training points by the rules of Bayesian inference. Since a function is an infinite dimensional object, it would be computationally unfeasible to describe its complete shape explicitly. Instead Gaussian processes require as input a set of \textit{test-points}, at which the function values are inferred.

The inference is based on a formal description of the smoothness of the estimated function, the so-called \textit{covariance function} $\text{cov}(\vec{q}, \vec{q}')$. Its values range from zero to one and give for the estimated function the correlation between the function values for two points from its domain. For this work the common assumption was made that the correlation between two function values decays exponentially with the squared distance of the corresponding arguments:

$$\text{cov}(\vec{q}, \vec{q}') = \exp\left[-\frac{1}{2\sigma^2} (\vec{q} - \vec{q}')^2\right] \quad (4.26)$$

To train a Gaussian Process for a given set of $N_{\text{train}}$ training points

$$\{(\vec{q}_1, \vec{X}_1), \ldots, (\vec{q}_{N_{\text{train}}}, \vec{X}_{N_{\text{train}}})\},$$
4. Machine Learning Adaptive Basis Sets

the covariance matrix $K$ has to be constructed, regularized, and inverted:

$$K_{ij} = \text{cov}(\vec{q}_i, \vec{q}_j)$$

$$A = (K + \epsilon I)^{-1} \quad (4.27)$$

The so-called hyper-parameters $\sigma$ and $\epsilon$ determine the characteristic descriptor length-scales and the training data noise. For the noise level $\epsilon = 10^{-4}$ was used, while the length-scale $\sigma$ had to be adopted to the descriptor settings.

A trained Gaussian process can be queried for the most likely function value at a test-point $\vec{q}_*$ via:

$$w_i = \sum_j^{N_{\text{train}}} \text{cov}(\vec{q}_*, \vec{q}_j) A_{ji} \quad (4.28)$$

$$\vec{X}_* = \sum_i^{N_{\text{train}}} w_i \vec{X}_i \quad (4.29)$$

the variance of this prediction is given by:

$$\text{var}[q_*] = \text{cov}(q_*, q_*) - \sum_i^{N_{\text{train}}} w_i \text{cov}(q_i, q_*) \quad (4.30)$$

Notice that, the variance does not depend on the function values, hence it does not truly indicate the quality of the interpolation, but rather the vicinity of training points. Training a Gaussian process has $O(N_{\text{train}}^3)$ complexity due to the required matrix-inversion, while predicting a function value takes only $O(N_{\text{train}})$ operations if $\sum_i A_{ji} \vec{X}_i$ has been precomputed. The prediction scales $O(N_{\text{atom}})$, because the basis of each atom is predicted independently. For efficiency reasons a Cholesky decomposition is used instead of performing full inversions. The average of the training data serves as prior.

The calculation of the force contributions requires the derivative of the total energy with respect to the feature vector. Here, the PAO gradient $\vec{G} = \partial \Omega / \partial \vec{X}$ from equation (4.15) can be reused:

$$\frac{\partial \Omega}{\partial w_i} = \vec{G} \cdot \vec{X}_i$$

$$\frac{\partial \Omega}{\partial q} = \sum_i^{N_{\text{train}}} \frac{\partial \Omega}{\partial w_i} \cdot \vec{q}_i - \vec{q} \frac{\partial \Omega}{\partial \vec{q}} = \sum_i^{N_{\text{train}}} \frac{\partial \Omega}{\partial w_i} \cdot \vec{q}_i - \vec{q} \frac{\partial \Omega}{\partial \vec{q}} \text{cov}(\vec{q}, \vec{q}_i) \quad (4.31)$$

4.3.3 Descriptor

The purpose of a descriptor is to characterise the chemical environment of an atom. For the application to PAO, the descriptor has to be rotationally invariant, analytically derivable, and cheap to compute. The search for
a good general-purpose descriptor is an ongoing research effort [102, 103, 104]. For this work a variant of the descriptor proposed by Sadeghi and Goedecker et al. [105] was chosen. It is based on an overlap matrix which is calculated from atom centered Gaussians. The eigenvalues of this overlap matrix are then used as descriptor. In order to make the descriptor local, it contains a distance screening. Deviating from the original proposal, this screening was implemented via a third Gaussian. Hence, the overlap matrix for the atom $I$ has the following form:

$$Q_{JK} = \sqrt{\frac{\beta_I}{\pi}} \sqrt{\frac{\beta_K}{\pi}} \int e^{-\sigma_l (\vec{r} - \vec{R}_I)^2} \ w_J e^{-\beta_I (\vec{r} - \vec{R}_J)^2} \ w_K e^{-\beta_K (\vec{r} - \vec{R}_K)^2} \ d\vec{r}$$ (4.32)

This three-center integral can be solved analytically. The eigenvalues $\vec{q} = \{q_i\}$ of the matrix $Q$ are then taken as the actual descriptor:

$$Q = R \text{diag}(\vec{q}) R^\dagger$$ (4.33)

The $\sigma_l$ acts as a screening parameter, it determines the cutoff radius $r_c$ of the descriptor. The indexes $J$ and $K$ run over all neighbors within the sphere of radius $r_c$ around $R_I$. To allow for the descriptor to distinguish between different atomic species, the exponents $\beta_I$, $\beta_K$ and weights $w_J$, $w_K$ can take different values depending on the atomic kind.

The PAO force contribution to the $l$’th atom from the machine learning part can be readily calculated using the expression $\partial \Omega / \partial \vec{q}$ from equation (4.31) in the previous section:

$$D = R \text{diag} \left( \frac{\partial \Omega}{\partial \vec{q}} \right) R^\dagger$$

$$F_l = -\frac{\partial \Omega}{\partial \vec{R}_l} = -\sum_{JK} D_{JK} \cdot \frac{\partial Q_{JK}}{\partial \vec{R}_l}$$ (4.34)

4.4 Results

4.4.1 Water Dimer

The implementation was first tested on a water dimer system as shown in figure 4.5. Starting from a relaxed geometry, the Oxygen-Oxygen distance was varied between 2.4Å and 4.0Å in steps of 0.1Å.

The left side of figure 4.6 shows the corresponding reference energy curve, which was obtained with the DZVP-MOLOPT-GTH basis set. It shows that an O-O distance of 2.4Å is more than 11mHa higher in energy than the equilibrium configuration, which is equivalent to a temperature of about 3500K.
Figure 4.5: Water dimer with atom enumerations that was used as test system.

Figure 4.6: Left: Energy dissociation curve of water dimer. Right: Errors introduced by PAO method and potential parameterization.

The right side of figure 4.6 shows the errors introduced by the PAO method and the potential parameterization. For the PAO calculations four PAO basis functions were used for both Hydrogen and Oxygen atoms, ie. a PAO 4-4 scheme. The blue curve denotes the energy difference between the reference DZVP results and the exponential parameterization. It show an error of 1.9mHa at a close O-O distance of 2.4Å, and a monotone decay towards larger distances, at 4.0Å the error has dropped below 0.13mHa. The exponential parameterization is minimal and complete. Therefore, the error originates solely from the reduced number of basis functions. A water molecule has here four molecular orbitals, because the Oxygen’s 1s electrons are included in the pseudo potential. Therefore, a PAO 4-4 scheme can exactly represent the four MOs of a single water molecule by simply contributing one PAO basis function of each atom to each of the MOs. The monotonously decaying error for larger O-O distances reflects the exactness in the limit of two isolated water molecules.

The green curve in the right side of figure 4.6 denotes the error that the potential parameterization adds to the PAO error. It shows an error of 0.5mHa at the close O-O distance of 2.4Å, and a nonuniform decay towards larger distances. Around 3.4Å the error increases shortly to 0.4mHa, at 4.0Å the error has again dropped below 0.12mHa. The error introduced by the potential parameterization originates from the incompleteness of the potential terms.

4. Machine Learning Adaptive Basis Sets
The curve shows that the potential parameterization struggles whenever the chemical situations becomes more complex: At very short O-O distances and at the intermediate distances where the Hydrogen bond forms.

![Figure 4.7: Dissociation curve of PAO parameters for the first water molecule (atoms 1-3). Top: PAO parameters for the exponential parameterization. Bottom: PAO parameters for the potential parameterization.](image)

Figure 4.7 shows the optimized PAO parameters of the first water molecule (atoms 1-3). In order to keep the plots readable, only a selected subset of representative PAO parameters is shown.

The top row in figure 4.7 show the parameters for the exponential parameterization. The parameters of the exponential parameterization correspond to angles of rotations among the primary basis functions. The plots reveal three problems with the exponential parameterization that make it unfit for machine learning: 1) The exponential parameters are not rotationally invariant. Atom 2 and atom 3 are chemically equivalent as these are the Hydrogens pointing away from the water dimer. However, their rotation angles are different because they refer to the underlying Cartesian coordinate system. As a consequence atom 2 and 3 have different parameters despite their symmetry. 2) The sign of the exponential parameters is ambiguous. Changing the sign of all angles simultaneously will result in PAO basis functions with opposite sign, but it will not alter the spanned vector space. Such a random sign flip occurred e.g. at atom 2 between 3.6Å and 3.7Å. 3) The final problem with the exponential parameterization is its sensitivity. The plots shows that the angles can assume large values, reaching up $\pi/2$, while at the same
time they are fairly constant over the course of the dimer dissociation. The reason is that the exponential parameterization also has to account for the isolated atoms. As a consequence, the total energy is very sensitive to the angles, while the angles are not very sensitive to the chemical environment.

The bottom row in figure 4.7 show the parameters for the potential parameterization. The parameters of the potential parameterization correspond to coefficient of the potential terms. The plots show that the potential parameterization is indeed rotationally invariant. The two Hydrogen atoms 2 and 3 share very similar parameters as one would expect from their symmetry. What is not show in the figure is that, for large O-O distance all four Hydrogen atoms converge to the same parameters and the two Oxygens become alike as well. The plots also show that the potential parameters evolve smoothly along the dimer dissociation. This is achieved through the $L_2$ regularization term. Finally, the curves show the high sensitivity to the chemical environment. For example, at atom 1 the parameters change rapidly when the Hydrogen bond is established around 3.4Å and in the high-energy regime below 2.8Å. For the regularization the default values were used and the potential beta and weight were set to 1.0 for Oxygen and Hydrogen.

4.4.2 Water Dimer Trajectory

The machine learning implementation was also first tested on the water dimer system. Using the primary DZVP basis set an MD trajectory with 2000 steps was generated. From this trajectory every 25th frame was selected, yielding a total of 81 frames. Each of the selected frames was PAO optimized using the potential parameterization. The blue curve in figure 4.8 shows the difference between the energies obtained with the PAO basis compared to the DZVP reference. The increased error around step 500 and 2000 correspond to the formation of a Hydrogen bond. This is in agreement with the previous findings from the dimer dissociation curve, were the error also slight increased during the Hydrogen bond formation.

Out of these 81 reference frames, the first 41 frames were used as training-data for the learning machinery. The green curve in figure 4.8 shows the difference between the energies obtained with optimized PAO basis sets and the machine learned PAO basis sets. The curve shows that the additional error introduced by the machine learning remains mostly below 0.4 mHa. Towards the end of the trajectory the error increases, because the system has evolved too far away from the realm of the training data. The curve furthermore shows that the training points are not exactly reproduced by the machine learning. This is a property of Gaussian processes, whose inference is influenced also by nearby training points even if the descriptor matches exactly. In the descriptor the beta was set to 0.27 for O and 0.35 for H, the
4.4. Results

Figure 4.8: PAO errors for a set of 81 configurations that were generated with an MD simulation. The blue curve denotes the error introduced by the PAO method. The green curve denotes the error introduced additionally by the potential parameterization.

screening was 0.13 for O and 0.21 for H, while the descriptor weight was 1.0 for both elements. The length-scale of the Gaussian process was set to 0.71.

Figure 4.9 shows how the PAO parameters evolve over the course of the trajectory. Again only a subset of relevant PAO parameters is show, namely the same as those in the dissociation curve at the bottom of figure 4.7. The top row of figure 4.9 shows the reference data from the optimized PAO basis sets. The high frequent components of the curves correspond to the intra-molecular vibration modes of the two waters. These are small in magnitude, because the chemical environment changes only slightly during an oscillation cycle. The low frequency contributions correspond to changes in the two molecules’ relative position and orientation. These movements alter the chemical environment substantially, in particular for atom 1, which forms occasionally a Hydrogen bond with atom 6. This is reflected by large changes in the corresponding PAO parameters.

The bottom row shows the PAO parameters that were obtained from the learning machinery. Again the first 41 frames served as training data. The curves show that the machine learned parameters reproduce the reference data qualitatively correct. However, many of the detailed features are not
reproduced very accurately. Nevertheless, the resulting error remains small, as previously shown in figure 4.8. Hence, this demonstrates the favorable robustness of the potential parameterization.

### 4.4.3 Consistency of Energy and Forces

In order to validate that the forces provided by the PAO-ML implementation are consistent with its energies, a series of short molecular dynamics simulations with different time steps was performed. The simulations were carried out in the NVE ensemble, using the water dimer system from the previous section. For the integration of Newton’s law of motion the velocity-Verlet algorithm [106] has been employed. Since the NVE ensemble conserves the system’s total energy, any energy fluctuations should be due to the integration error. The velocity-Verlet algorithm has an integration error that is of second order in the time step. This means the standard variation of the energy fluctuations are proportional to the square of the time step, i.e. $\sigma_E \propto \Delta t^2$.

Figure 4.10 shows the fluctuations obtained with time steps of 0.4fs, 0.2fs, and 0.1fs. The standard variations extracted from these fluctuation curves are: 5.00 $\mu$Ha, 1.23 $\mu$Ha, and 0.31 $\mu$Ha. This matches nicely the fourfold decrease expected for a time step halving. It shows that the energy fluctuations are solely caused by the velocity-Verlet’s integration error as they are controllable via the time step. This confirms the consistency of the PAO-ML implementation.
4.4. Results

Figure 4.10: Energy fluctuation during a series of MD simulation of a water dimer using the PAO-ML scheme. The simulations were conducted in the NVE ensemble using different time steps \( \Delta t \) to demonstrate the controllability of the integration error.

4.4.4 PAO Basis Sets for Liquid Water

In this section, the performance of the method for bulk liquid water is explored. This system has a long tradition within the first principles MD community, as it is both important and difficult to describe [107]. From an energetic point, the challenge arises from the delicate balance between directional Hydrogen bonding and non-directional interactions such as van der Waals interactions [108]. The relatively weak interaction can furthermore be influenced by technical aspects, such as basis set quality. Additionally, the liquid is a disordered state, which requires sampling of configurations for a proper description. The disorder makes it also an interesting test case for the ML approach, as the variability of the environment of each molecule can be large.

First, the suitability of the PAO approach for the liquid is investigated for various combinations of primary basis, PAO basis set size, PAO parameterizations, and ML setups. To do this, 71 frames containing 64 water molecules, spaced 100fs apart, are taken from an earlier MP2 MD simulation at ambient temperature and pressure [109]. For these configurations, the energy shift and the root mean square deviation (RMSD) of the energies
Table 4.1: Show the characteristics of the exponential parameterization of PAO as a function of the primary basis and the PAO size, for bulk liquid water configurations containing 64 molecules. ∆E is the average energy difference [a.u.] with respect to the TZV2P reference, RMSD quantities the fluctuations of the energy [a.u.] around the reference after subtraction of the average, cond. refers to the condition number of the overlap matrix, and iter. to the number of iterations needed for the PAO optimization to converge.

<table>
<thead>
<tr>
<th>Basis Set</th>
<th>Iter.</th>
<th>RMSD</th>
<th>Cond.</th>
<th>Iter.</th>
<th>RMSD</th>
<th>Cond.</th>
<th>Iter.</th>
<th>RMSD</th>
<th>Cond.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>564</td>
<td>0.1551</td>
<td>1480</td>
<td>564</td>
<td>0.1551</td>
<td>1480</td>
<td>564</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
<tr>
<td>TZV2P-MOLOPT-GTH</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
<td>615</td>
<td>0.1551</td>
<td>1480</td>
</tr>
</tbody>
</table>
4.4. Results

with respect to a reference potential energy surface as obtained with a large (TZV2PX-MOLOPT-GTH) basis set are reported in table 4.1. These energies help to quantify the accuracy of the obtained potential energy surface (PES). Whereas the energy shift is guaranteed to decrease with increasing size of the PAO basis or primary basis set, the RMSD provides a better measure on how faithful the obtained PES is reproduced. To start with the energy shift ($\Delta E$) with respect to the reference, several observation can be made. First, a minimal (4-1) PAO basis set is 4 to 8 times lower (depending on the primary basis) in total energy than a fixed (SZV) basis, which is the expected result of the optimization. The effect of increasing the primary basis is relatively small, reducing $\Delta E$ from 0.985 a.u. (DZVP) to 0.595 (TZV2PX), emphasizing that converging a minimal PAO basis by using a more flexible primary basis is relatively difficult. Interesting is the fact that the RMSD for the 4-1 PAO increases when the primary basis is increased, which is counter intuitive at first, since it implies that worse PES are obtained if the PAO basis set size and primary basis set size are not sufficiently balanced. As soon as the PAO basis set size is increased from minimal to intermediate sizes (e.g. 4-4, 6-4, 8-4), the performance of the PAO basis set very closely resembles the performance of the used primary basis set, requiring slightly larger PAO basis set sizes for the better larger primary basis sets sizes, e.g. DZVP-4-4, TZV2P-6-4, TZV2PX-8-4. Furthermore, a PAO TZV2P-6-4 is better in quality than a full DZVP, a PAO TZV2PX-8-4 is better in quality than full TZV2P. The latter observation is especially interesting in light of the reported condition numbers. Indeed, PAO basis sets of increasing size gently increase in condition number from 6 for an minimal basis sets (4-1) to 140 (8-4), being even in the latter case one order of magnitude better than the full DZVP and two orders of magnitude better than the full TZV2P, despite the fact that these full basis sets of the MOLOPT type are already much better conditioned than typical split valence basis sets [62]. The condition number of the PAO basis does not significantly depend on the primary basis, but mostly on its size. The very small condition number of the PAO basis sets is an important contribution to their efficiency in linear scaling calculations. Finally, the required number of PAO iterations shows a clear trend in that the optimization becomes computationally more demanding as the size of either the primary or the PAO basis increases. A minimal PAO basis requires a few hundred iterations to converge, an TZVP2P-8-4 basis requires nearly 10000 iterations to reach full convergence. This makes the optimization of PAO basis sets ‘on-the-fly’ (for each MD step) too demanding for practical applications, and is an important motivation for the ML scheme.

4.4.5 PAO-ML Energies for Liquid Water

Based on the results from the exponential parameterization, three PAO basis set have been selected to investigate the performance of the potential param-
4. Machine Learning Adaptive Basis Sets

<table>
<thead>
<tr>
<th>PAO name</th>
<th>primary O size</th>
<th>H size</th>
<th>2nd pot. regul.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAO-4-1</td>
<td>DZVP-MOLOPT-GTH</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>PAO-4-4</td>
<td>DZVP-MOLOPT-GTH</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>PAO-6-4</td>
<td>TZV2P-MOLOPT-GTH</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.2: parameters for the potential and ML parameterization. The potential beta and weight are 1.91, 4.96 for O and 3.04 and 2.76 for H. The descriptor beta is 0.11 for O and 0.16 for H, while for both elements the weight and screening are 1.0 and 0.66 respectively. The length-scale of the Gaussian process is 0.15.

<table>
<thead>
<tr>
<th>PAO name</th>
<th>exp</th>
<th>potential</th>
<th>ML</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ΔE</td>
<td>RMSD</td>
<td>ΔE</td>
</tr>
<tr>
<td>PAO-4-1</td>
<td>0.9851</td>
<td>0.0036</td>
<td>0.9984</td>
</tr>
<tr>
<td>PAO-4-4</td>
<td>0.2468</td>
<td>0.0025</td>
<td>0.2564</td>
</tr>
<tr>
<td>PAO-6-4</td>
<td>0.0945</td>
<td>0.0009</td>
<td>0.1131</td>
</tr>
</tbody>
</table>

Table 4.3: A direct comparison of the exponential, rotationally invariant and machine learning parameterization for selected PAO basis sets. The test system and quantities computed are as shown in table 4.1.

A comparison was made between the parameterization and the machine learning scheme. These basis sets are listed in table 4.2, with their corresponding parameter for the second potential regularization. The machine learning approach employed one out three frames, for a total of 23 frames out of 71, for learning, while all 71 frames have been employed for computing ΔE and the RMSD. The results for these basis sets and parameterization are shown in table 4.3. Consistent with the results obtained for the water dimer, the potential parameterization and the machine learning scheme introduce relatively small - but non-negligible - errors. For ΔE, the additional error introduced is on the order of 20%, and similar in relative magnitude for the smaller PAO-4-1 and larger PAO-6-4. The error results both from the potential parameterization, as well as from the machine learning. For the RMSD, the error is larger and is between 50% and 100%. For the smaller PAO-4-1 it is due to both the potential parameterization and well as the machine learning step, while for the larger PAO-6-4, the largest contribution of the error comes from the potential parameterization. In performing these measurements, the importance of of carefully selected descriptors for the machine learning has become apparent. The values reported in table 4.2 are the result of a derivative free optimization of the RMSD. We observe that the same values work well for the three different basis sets studied, but also expect this choice to be non-universal. For example, the descriptor is relatively local in nature, which is reasonable if we assume that the optimal PAO basis is mostly a result of the local bonding motif. Such a local description allows learning with a relatively small amount of data, but in the presence of a large amount of data, a wider ranging descriptor could perform better.
4.4. Results

4.4.6 PAO-ML Molecular Dynamics of Liquid Water

So far, we have tested the performance of the method based on frames sampled with a traditional approach. More challenging for a ML method is sampling configurations based on predicted energies. In particular, to verify that instabilities and unphysical behavior are absent when the method is given the freedom to explore phase space. To test and verify the performance, molecular dynamics simulations have been performed for 64 molecules of water at experimental density and 300K, producing trajectories between 20 and 40ps depending on the method, employing standard basis sets of triple zeta quality (TZV2P-MOLOPT-GTH), PAO-4-1 and PAO-4-4 basis sets, density functional tight binding (DFTB) [110, 111] as well as traditional minimal (SZV-MOLOPT-GTH) basis sets. To put these results further in perspective, simulations have been performed with two density functionals, namely PBE-D3 and revPBE-D3, and are also compared to experiment.

The Oxygen-Oxygen pair correlation functions of liquid water are shown in figure 4.11. First, these results show that the PAO-4-1 and PAO-4-4 simulations are similar to the reference TZV2P-MOLOPT-GTH results. In particular, PAO-4-1 improves notably over minimal basis SZV-MOLOPT-GTH and DFTB results. The position of the first peak in the O-O pair correlation function matches well the one of the reference functional, which is a significant improvement over the result obtained with a SZV-MOLOPT-GTH basis. The height of the first peak is somewhat overestimated (relative to the TZV2P reference) with the PAO-4-1 basis, while the PAO-4-4 basis slightly underestimates this height. The difference between PAO-4-1 or PAO-4-4 and the reference basis result is not more than the difference between the results obtained with the two density functionals tested. Comparing to the DFTB results, the difference is most significant near the second solvation shell, which is mostly absent or strongly shifted to larger distances with DFTB, whereas the PAO-4-1 reproduces the reference basis results accurately. In the case of revPBE-D3, where least structure is present, the underestimation of the structure with PAO-4-4 leads to a too shallow first minimum, resembling the DFTB result in that region. Generally speaking, already the PAO-4-1 setup leads to a acceptable water model, similar to the reference density functional employed. With the expectation of an excellent match between PAO-6-4 and the reference results, MD simulations with this setup have been attempted. However, it was found that this setup required a very small MD time step for good energy conservation. This is due to small but high frequent features in the potential energy surface that result from rapid changes in the machine learned basis sets. The origin of these rapid changes is the ‘closing gap issue’ as discussed in section 4.2.5, which happens in the machine learned parameterization, even though in the potential parameterization of the training set, this issue is not present. As a satisfactory solution for this problem has not been found, extended simulations with PAO-6-4
have not been performed. This shortcoming should be addressed in future developments.

A question which has not yet been investigated in detail is the amount of reference data that is required to obtain satisfactory results, and the behavior of the model as reference data is reduced. To this end, simulations of 64 waters have been performed with the PAO-4-1 basis set, reducing the amount of reference data with each run from 71 down to just 11 frames,
4.4. Results

Figure 4.12: Shown are static ($g_{OO}(r)$, top panel) and dynamic properties (mean square displacement, bottom panel) of liquid water described using the PAO-4-1 basis set and revPBE-D3, but with various amount of reference data for the machine learning setup. Going from 71 down to just 11 frames (each 64 molecules) of reference data, the properties remain essentially unchanged within their respective statistical uncertainty.

more precisely: 71, 35, 23, 17, 14, 11 frames. The results are shown in figure 4.12 for both the Oxygen-Oxygen pair correlation function, and the mean square displacement (MSD). The latter is a dynamical property, and should be sensitive to the description of the molecules that participate in a diffusive event. This is potentially a rare event, and should thus be more difficult to learn. Interestingly, the obtained pair correlation functions, and the mean square displacement are, within their expected statistical uncertainty, independent of the amount of reference data used in training. This leads to the
conclusion that surprisingly little data might be required to machine learn reasonable PAO basis sets. One possible reason for this interesting property might be that the method is ‘variational’ with respect to the machine learning part. Indeed, a poor prediction leads to a basis set that is worse in quality than the optimized PAO basis, and thus in a total energy that is higher than what would be obtained with a well predicted or optimized PAO basis. As a result, the dynamics will thus spontaneously avoid configurations for which little or no data is available. This is an advantage over ML methods that directly learn the energy, and which might predict energies that are too low for unknown or unphysical configurations, potentially resulting in instability. The bias towards physical and known configurations, and thus the associated stability, of the ML-PAO scheme, might be an important aspect of the method for practical application. Another consequence of the method’s variational nature is that it does not require a separate test set for cross-validation of the training. Instead, a lowering of the predicted energy can be taken as a direct indication of training progress.

4.4.7 PAO-ML Speedup

Finally, benchmark results are shown in table 4.4 to quantify the speedup obtained with this approach in the context of linear scaling calculations. As a test system, a cubic unit cell containing 6912 water molecules is employed, and the total energy is computed using the TRS4 algorithm [53] and a simple mixing scheme. The simulations were run on two different architectures, namely a GPU accelerated Cray XC30, and a dual socket CPU-only Cray XC40. Using instrumentation of the code, the total number of flops needed in the matrix multiplication part of CP2K increases from $4.07 \cdot 10^{12}$ for PAO-4-1, to $4.16 \cdot 10^{15}$ for PAO-4-4, to $61.63 \cdot 10^{15}$ for the reference DZVP-MOLOPT-GTH basis. This demonstrates a 4 orders of magnitude decrease in the number of required flop for the PAO-4-1 method relative to the DZVP-MOLOPT-GTH approach. Furthermore, it also demonstrates the importance of the condition number of the overlap matrix in sparse linear algebra, as the reduction in flops in the dense case would be only 56x (6 vs 23 basis functions per water molecule). We note that all methods have the same primary basis, and thus the same sparsity of the overlap and Kohn-Sham matrix. The reduction in walltime depends on the precise computational setup, and also on which part of the calculation is considered, in particular only the matrix multiplications, or the full calculation. It is noteworthy that the computational overhead of the operations related to the ML are negligible. The largest speedup is observable in the matrix multiplication part, on few nodes, and in the CPU only setup. In this case, the PAO-4-1 scheme yields a 200x reduction, the PAO-4-4 scheme a 10x reduction in walltime. The full calculation, where the construction of the Kohn-Sham matrix is the other major contribution, is faster by 60x in the PAO-4-1 case, and 8x in the
Table 4.4: A comparison of timings obtained with DZVP-MOLOPT-GTH, PAO-4-1, and PAO-4-4 basis sets for the full SCF of a system containing 6912 water molecules in a cubic unit cell. Timings are obtained on two different architectures (XC30 and XC40), using 64-400 nodes. Two timings are reported, namely the total time for this run, and the time spent in sparse matrix-matrix multiplication. OOM stands for ‘out of memory’ on the GPU card. For this system, the total number of flops needed in matrix matrix multiplications (number of multiplications in parenthesis) is $4.07 \cdot 10^{12}$ (152), $4.16 \cdot 10^{15}$ (190), and $61.63 \cdot 10^{15}$ (227) for PAO-4-1, PAO-4-4, DZVP-MOLOPT-GTH respectively.

PAO-4-4 case. As more nodes are employed, this advantage reduces to 50x and 6x respectively. Ultimately, a SCF step takes just 3.3s on 400 nodes, and is dominated by the KS matrix construction. Since the PAO-4-1 setup could be seen as an alternative to semi-empirical methods, providing minimal basis set results without the need of explicit parameterization, a comparison to DFTB results is instructive. A DFTB setup requires for the same single point energy calculation less flops in matrix multiplication ($1.52 \cdot 10^{12}$ versus $4.07 \cdot 10^{12}$), which reflects in the corresponding timings for matrix multiplication (5.791s versus 23.575s on 64 nodes of an XC40). The sparsity of the density matrix is somewhat better in the DFTB case (3.3% versus 5.2% occupation), despite the fact that both methods result in a similar electronic gap and have a similar condition number for $S$. The difference in occupation number is likely due to the fact that the PAO-4-1 basis is longer range. The
total timings are more in favor of DFTB (10.423s versus 87.184s on 64 nodes of an XC40), which is related to the relative expense of building the Kohn-Sham matrix with the underlying DZVP-MOLOPT-GTH basis. With future developments, including the use of a more effective primary basis sets, this ratio could be further reduced, potentially making PAO-4-1 a cost effective alternative to DFTB. The GPU accelerated results illustrate that for the traditional basis sets, GPU acceleration is faster than CPU only calculations, but that sufficiently many nodes need to be available for the calculation to fit in GPU memory. The smaller basis sets allow for running the calculation on at least 10x less nodes without exhausting the GPU memory. The small matrix blocks of the PAO-4-1 case, make it less suitable for GPU acceleration in the current implementation of the DBCSR library[8]. To summarize this section on liquid water, we conclude that the PAO-4-1 approach gives fair structural and dynamical properties of liquid water using a limited set of training data, while reducing the computational cost by 200x and the required flops by 15000x.

4.4.8 TiO$_2$ Nanoparticles in Explicit Acetonitrile Solution

Shown in Fig 4.13 is a challenging test system which is used to demonstrate the potential of the PAO ML scheme. The system consists of two TiO$_2$ nanoparticles that have coalesced, leaving a neck region with several defects. These nanoparticles are in contact with liquid acetonitrile, which interacts relatively strongly at the solid liquid interface[113, 114]. This is a simplified model for the mesoporous oxide / electrolyte interface found in dye sensitized solar cells[115], and could ultimately be used to understand electronic transport through mesoporous oxide materials. From a simulation point of view, the system is not only relatively large (77538 atoms), but is also more challenging because it is non-homogeneous, contains a larger number of chemical elements (Ti, O, C, N, H), and has a relatively small gap, due to defective nature of the oxide. This makes a robust and sufficiently general method required for its simulation. Here, it is shown that the PAO-ML approach allows for more than just energy calculations, and instead makes it possible to use a DFT based setup to perform geometry relaxation and molecular dynamics of such a system. Furthermore, this system has been used as a benchmark for the GPU implementation of the DBCSR library, and reference timings on the full hybrid Piz Daint system (5272 GPU nodes) are available for 50 SCF steps (122s / SCF step)[8].

To obtain the ML reference data for this system, smaller nanoparticles in solution (a total of 4713 atoms, system shown in the top panel of figure 4.13), have been employed. This choice illustrates that the learning system can be quite different from the production system. The PAO basis set sizes used are 1, 4, 4, 4 and 10 for H, C, N, H and Ti, respectively. A full PAO optimiza-
4.4. Results

Figure 4.13: Shown is a test system of aggregated nanoparticles of TiO$_2$ in liquid acetonitrile. The top panel on the left shows the full system of 77538 atoms, while the right shows the smaller training system with 4713 atoms. The middle and bottom panels zoom in on the nanoparticles, and show atoms colored from blue over white to red, depending on the logarithm of the variance (see equation 4.30) for the prediction by the ML setup. Atoms with a large variance are additionally shown as a larger ball. The middle frame shows these results using a single frame of learning data, the lower frame after two frames of learning data.
tion on such a system is computationally demanding, but can be performed in 1-2 days on 128 nodes. Shown in the middle and lower panel of figure 4.13 is a representation of the variance of the prediction (equation 4.30), for each atom. The middle panel, the variance after learning from a single frame, shows clearly that the bulk of the material is well represented and has a very low variance, whereas surface and edges of the nano-crystal have larger variance. We have visually verified that atoms shown with a larger ball are indeed defects, with unusual coordination. Adding a single additional frame of learning data (lower panel), reduces the variance of the surface and edge atoms significantly, while the variance for atoms with unusual coordination does not improve significantly. In such a system with complex structure, this variance is a useful indicator of chemically interesting structure, which is not always easy to obtain visually otherwise. This data could either be used to construct smaller models, in the form of extracted clusters or within an embedding (e.g. QM/MM) description, to generate specific additional reference data, or to describe regions of the system with a larger PAO or even standard basis set. This has not been attempted in this work, but should be explored in future work.

![Figure 4.14: Shown are the potential energy and constant of motion for the solvated nanoparticles (Fig 4.13) during a short MD run, illustrating stability of the MD for these systems.](image)

With this setup, the system has been relaxed using geometry optimization. As in our earlier work on a very large system, namely a virus in solution\[116\] we find it important to use a trust-region based optimizer that limits the step size (to 0.025Å) as initially very large gradients are present. Furthermore, the curvy steps linear scaling algorithm\[117\] has been employed for the optimization of the density matrix, which added robustness
4.4. Results

to the electronic structure part of the optimization procedure. We note that the total energy had to be tightly converged to $10^{-4}$ a.u. to guarantee convergence also for the densities near defects etc. After approximately 100 geometry optimization steps, the RMSD gradient is below $10^{-3}$, and the system was considered sufficiently relaxed for molecular dynamics. To verify this capability, the system was propagated for approximately 180 MD steps (90fs) and, for this short timescale, as shown in figure 4.14, observed to be stable and energy conserving.

Finally, performance aspects of this simulation are worth analysing. First, on 1024 nodes of the XC30, one MD step (with 9 curvy step SCF steps), requires $\approx 360$ seconds. Compared to the reference timings above ($122s$ / SCF on 5272 nodes) this can be considered a 15-fold speedup. The flops per SCF step are reduced by a factor 170. Effectively, the present simulation is performed with tighter threshold in sparse matrix multiplication (filtering based on a threshold of $10^{-7}$), so that the gain is actually somewhat larger. However, the simulation is also possible on fewer nodes and performs well on the XC40 architecture. On 256 nodes of the XC30, one MD step requires $\approx 780$ seconds, while on 256 nodes of the XC40, one MD step requires $\approx 400$ seconds. On 1024 nodes, sparse matrix matrix multiplication accounts for 62% of the total time, while the Kohn-Sham matrix construction accounts for 23% of the total time, and various other operations for the remainder. On 256 nodes, sparse matrix matrix multiplication accounts for 37% and 57% of the total time, while the Kohn-Sham matrix construction accounts for 34% and 25% of the total time, for the XC30 and XC40 architectures respectively. This flatter profile implies that techniques to further improve the performance related with the sparse matrix multiplication can at best speedup the full calculation by a factor of two, and attention must thus also be payed to improved algorithms for the Kohn-Sham matrix construction, in particular since the latter part is not GPU-accelerated. A call graph visualization of the timings from 1024 nodes is shown in the appendix on page 91.

4.4.9 Computational Setup

All the calculations were performed using the CP2K software [3, 47]. CP2K combines a primary contracted Gaussian basis with an auxiliary plane-wave (PW) basis. This Gaussian and Plane Wave (GPW) [46] scheme allows for an efficient linear-scaling calculation of the Kohn-Sham matrix. The auxiliary PW basis is used to calculate the Hartree (Coulomb) energy in linear-scaling time using Fast Fourier Transforms. The transformation between the Gaussian and PW basis can be computed rapidly. The cutoff for the PW basis set was chosen to be at least 400 Ry in all simulations. While the PW basis is efficient for the Hartree energy, the primary Gaussian basis-set is local in nature and allows for a sparse representation of the Kohn-Sham ma-
4. Machine Learning Adaptive Basis Sets

Matrix. For the simulations, the Perdew-Burke-Ernzerhof [118] (PBE) exchange and correlation (XC) functional and Goedecker-Teter-Hutter (GTH) pseudopotentials [45] were used. The linear-scaling calculations were performed with the implementation as described in Ref.[7], which in particular allows for variable sparsity patterns of the matrices. All SCF optimization used the TRS4[53] algorithm, except for the nanoparticles, which employed TRS4 only to obtain the first density matrix, and then used curvy-steps [117] for successive SCF steps. The SCF optimization was converged to a threshold (EPS_SCF) of $10^{-8}$ or tighter, the filtering threshold EPS_FILTER was to $10^{-7}$ or tighter. The default accuracy (EPS_DEFAULT) was set to $10^{-10}$ or tighter. All simulations were run in double precision.

4.5 Discussion and Conclusions

The PAO-ML scheme builds on prior work in which small basis sets of polarized atomic orbitals (PAOs) are obtained by variationally adapting the atom centered basis set to the chemical environment. Optimality of the PAO basis results from a self-consistent optimization of the parameters of the PAO basis and the electronic density with respect to that basis. In this context, a new optimization scheme was proposed that allows for reducing the number of required Kohn-Sham matrix updates, speeding up the optimization for small systems, while retaining the linear scaling nature of the PAO optimization. The optimal basis sets, computed for a small number of configurations, serve as training data for a machine learning (ML) approach to infer optimal basis set in related systems, avoiding costly optimization for each molecular geometry. To make this possible, a suitable parameterization of the PAO basis is needed, as well as a concise representation of the molecular environment of each atom. In our proposed scheme, the parameterization of the basis depends on the molecular geometry in a way that is invariant to rotations of the system and permutations of the atoms, for example leading to numerically equal parameters for symmetry equivalent atoms. To do this, the basis functions are obtained as the lowest energy states of an isolated atom that is perturbed by a parameterized and molecular geometry dependent potential. In order to obtain unique and smoothly varying parameters of the potentials, which make them suitable for learning, carefully designed regularization terms were introduced.

The local molecular geometry of each atom is reduced to a fingerprint using the eigenvalues of a local overlap matrix, making also the fingerprint invariant to rotations and permutations of the atoms. The machine learning step maps this fingerprint of the local geometry to optimal parameters for this perturbation potential. In this work, Gaussian processes are employed as a relatively simple, but well characterized ML procedure. One advantage of this ML technique is that a variance is computed for each fingerprint, in-
The PAO-ML approach has the interesting property that the optimal prediction of the parameters makes the energy minimal with respect to these parameters. During the actual simulation, this implies a certain stability of the simulation, as regions with poorly predicted parameters will be avoided due to their higher energy. For the machine learning step, unlabeled data, i.e. configurations for which no reference data is available, can still be used to judge the quality of the learning procedure, as two different ML setups can be compared using this variational principle as guidance. Ultimately, the whole PAO-ML method provides basis sets that depend in an analytical way on the atomic coordinates. As such, analytic nuclear forces are available, making the method suitable for geometry optimization and energy conserving molecular dynamics simulations.
Conclusion and Outlook

The goal declared in the research plan for this PhD project was to boost the performance of CP2K’s linear scaling DFT implementation to the point where the structure relaxation of an emerging photovoltaic material becomes feasible. This goal was indeed achieved, namely through a combination of GPU acceleration and machine learning techniques.

In the first part of this thesis, the predominant sparse matrix-matrix multiplications were GPU accelerated. To harvest the raw power of GPUs, the high-level algorithm of the DBCSR library had to be carefully rewritten to take the asynchronous nature of the device into account. A set of low-level Cuda kernels for relevant block-sizes were developed together with Nvidia engineers as part of a co-design effort. Based on these kernels an auto-tuning approach was used to create a large library of optimized kernels for other block-sizes. For the targeted block-sizes of up to 32, the measured performances of the Cuda kernels are very close to the achievable roof-line limit. For example, the important 23 block-size runs at 322 GFlop/s on a K20X, which is an improvement of 250% over the batched DGEMM available in cuBLAS.

Due to DBCSR’s good GPU performance, sparse matrix-matrix multiplication has become an important primitive also for other methods in CP2K besides purification. These include e.g. preconditioner [119], real-time propagation [116], random-phase approximation [120], and tensor contractions. This increasing diversity of applications has led to a steadily growing number of relevant block-sizes. Since every Cuda kernel takes up about 10kB of space in the binary, only around 3000 kernels are currently included in CP2K by default. In future work this limitation could be resolved by using the novel runtime compilation library NVRTC, which was introduce with the Cuda 7.0 release. Instead of including instantiated kernels, the CP2K binary could just carry the kernel templates and the database of tuned parameters. The actual kernels would then be generated and compiled just
5. Conclusion and Outlook

in time whenever needed. This approach could be taken a step further by developing a heuristic that can predict suitable kernel parameters for any block-size. For this undertaking the existing auto-tuned parameters could serve as training-data.

In the second part of this thesis the PAO-ML scheme has been developed and tested. PAO-ML employs machine learning techniques to infer geometry adapted atom centered basis sets from training data in a general way. The scheme can serve as an almost drop-in replacement for conventional basis sets to speedup otherwise standard DFT calculations. Practically speaking, large matrices obtained in the primary basis are transformed into smaller matrices in the PAO basis prior to the SCF procedure, and back-transformed afterwards. From a computer science perspective, this approach could also be viewed as a lossy compression scheme for the data handled during the linear algebra in the SCF procedure, where the PAO basis set plays the role of a codebook. From a simulation methodology perspective, the method is similar to semi-empirical models based on minimal basis sets, with improved accuracy and quasi-automatic parameterization.

The performance of the method was demonstrated using MD simulations of liquid water, where it was shown that small basis sets yield structural and dynamical properties that outperform those of other minimal basis set approaches. Interestingly, very small samples of training data yielded satisfactory results. Compared to the standard approach, the number of flops needed in matrix-matrix multiplications decreased by over four orders of magnitude, leading to an effective 60-fold run-time speedup.

Finally, the PAO-ML method was used to simulate two TiO$_2$ nanoparticles solvated in liquid acetonitrile, which is an important component of dye sensitized solar cells. For this challenging non-homogeneous system, containing more than 75000 atoms, it was demonstrated that performing geometry relaxation and even molecular dynamics on a moderately sized part of a modern supercomputer is feasible. This opens the way to effective, DFT based, simulation of nanoscale systems and devices.

Clearly, the PAO-ML approach presented in this work can be further refined and extended. Possible directions for improvements include: a) Systematic storage and extension of reference data to yield a general purpose machine learned framework for large scale simulation, including a more rigorous quantification of the expected error, which will improve usability. b) Refined parameterization of the PAO basis sets, reducing the number of parameters needed and the enhancing the robustness of the method with larger PAO basis sets, which will improve accuracy. c) Extensions of the method to yield basis sets for fragments or molecules rather than atoms, which will increase accuracy and efficiency. d) More advanced machine learning techniques and alternative descriptors, which will allow for larger training sets and
improved transferability of reference results. These directions should be
explored in future work.

This thesis succeeded in pushing the envelope for large-scale DFT calcula-
tions. The chosen inter-disciplinary approach, combining know-how from
chemistry and computer science, holds great promises for future advances.
For example, the connection of basis sets and Shannon’s signal theory is
worth exploring. Furthermore, the emergence of novel heterogeneous com-
puter architectures pose an incredible challenge for existing codes. Previ-
ously hard-coded design decision have to be encapsulated to make them
eligible for change. In the case of the DBCSR library, the local block storage
format will have to be revised as more book-keeping work is moved to the
accelerators. Also the distribution of blocks across compute nodes might
become subject to change as heuristic communication avoidance strategies
are explored. In order to encapsulation such fundamental design decision,
while retaining good performance, more high-level programming tech-
niques, like code generation, will have to be introduced. These develop-
ments suggest that in the future the engineering portion of computational
chemistry will increase in comparison to its scientific aspects. A high engi-
neering ratio is no stranger to other fields such as e.g. high energy physics.
Despite these challenges, atomistic simulations have an exciting future ahead
as we are witnessing the dawn of the nano-age.
Following the belief that engineering is of growing importance in computational chemistry, I engaged in a number of side projects. In the following, I will shortly present five of my efforts of making CP2K more attractive and future-proof.

The CP2K program consists of over one million lines of code and is subject to perpetual change. In the early days of CP2K, most changes were additions of new and orthogonal functionalities. However, lately existing methods are increasingly extended or combined in novel ways. (The work in this thesis being an example.) Such changes often conflict with the assumptions that underlay the original design decisions. Therefore, these changes bear the thread of damaging existing abstractions built into the code. As a consequence, the complexity of the code base increases while its maintainability decreases. Hence, to keep CP2K long-term maintainable, active measures must be taken to counteract these entropic forces.

The universal approach against complexity is divide and conquer. Applied to software, this means to break the code base up into manageable pieces. Each piece of the code has to provide an abstraction by hiding its implementation details behind an application programming interface (API). Explicit APIs have the advantage of formalizing abstractions, which makes them more visible and easier to enforce. In order to encourage developers to use APIs, two activities should be made as easy as possible: Creating new APIs, and refactoring existing APIs. In the following two sections, I will outline how I worked towards these goals.
Packaging of CP2K’s Code Base

The Fortran language allows to declare objects as private within single source files. Unfortunately, the language does not provide any privacy mechanism for packages, i.e., groups of files. Therefore, I rewrote CP2K’s build system to provide an auxiliary mechanism for package privacy. Each sub-directory of the source folder is considered a package. Each package contains a small file, called package manifest, which lists files that are made publicly available to other packages. During the build process the package manifests are parsed and the package privacy is enforced.

Dividing the CP2K code base into packages is a laborious but not impossible task. It is made feasibly by the remarkable fact that the CP2K project adopted very early on an object oriented programming style. The packaging of CP2K also has other merits besides improving maintainability. The well defined APIs allow for a separation of concerns between computer engineer and scientists [121]. Furthermore, the packages allow to join forces with other quantum chemistry projects through shared libraries. An example for this is DBCSR, which is a package that is already available as a stand-alone library [59].

A good API is only useful if the provided abstraction is well documented and visible. Due to the lack of alternatives, I initiated the creation of a novel documentation generator specifically designed for modern Fortran. I collaborated with Sebastiano Caravati in the development of a partial Fortran parser and an accompanying HTML-generator. Figure A.1 shows a screenshot of the generated API-documentation.

![API documentation screenshot](https://apidoc.cp2k.org)

Figure A.1: Screenshot of CP2K’s API-documentation at https://apidoc.cp2k.org.
Continuous Integration for CP2K

The key enabler for refactoring is testing, as it gives developers the confidence to make larger changes. CP2K comes with over 2800 regression tests that cover around 85% of the code. However, running the test suite is time consuming, in particular when multiple configurations are tested. This common dilemma can be resolved by liberating the developers from the responsibility for thorough testing. That insight has led to the wide adoption of Continuous Integration (CI), where each code version in a central repository is tested automatically.

My first attempt at establishing CI in the CP2K project was using the popular Jenkins test automation software. However, I had to realize that a centralized architecture, which is typical for CI frameworks, does not go well together with CP2K’s distributed community. Therefore, as a second attempt, I wrote a python script that periodically collects testing reports from various URLs for central display. Figure A.2 shows a screenshot of the resulting dashboard. The loose architecture of the dashboard allows contributors to publish test reports at their own discretion and retain full control of the actual test execution. The CP2K community responded very positive to the dashboard and the number of provided test reports grew quickly. Today the dashboard collects test results from 40 different sources, covering a range of different configurations and architectures, including Piz Daint at CSCS and JUQUEEN at Jülich. It also lists valgrind reports, test coverage statistics, performance measurements, and compliance checks for coding conventions. Recently, the archived test reports were also data-mined, which led to further refinements, including e.g. the introduction of fixed reference values.

Figure A.2: Screenshot of the CP2K dashboard at https://dashboard.cp2k.org.
Profiling Time and Energy Consumption

CP2K has a strong emphasis on performance, which is why it comes with a built-in profiler. It is based on a hand selected set of annotated key routines, which leads to readily comprehensible profiling reports.

Originally, the CP2K profiler could only collect timing statistics on a per routine basis. However, it is also of interest from where a costly routine is called. Therefore, I rewrote CP2K’s profiler in order to track sub-routine calls explicitly. This allows for the generation of call graphs, such as the one shown in figure A.4. For visualization the open source tool KCachegrind was used.

While time and energy consumption used to be proportional, this strong correlation no longer holds due to the recent advent of dark silicon [122] and dynamic clocking techniques, such as Nvidia GPU Boost and Intel Turbo Boost. Therefore, the CP2K profiler was extended to also track energy consumption alongside the elapsed time. The energy consumption per routine is determined from the special files in the /sys/cray/pm_counters directory. These counters were recently introduced on Cray systems and provide live power information with a time resolution of approximately 10 Hz [123].

Finally, the Nvidia tools extension (NVTX) was used to tie CP2K’s profiler into the Nvidia profiler (nvprof). It allows to display routines running on the host CPU within the Nvidia visual profiler (nvvp). This feature greatly aids the interpretation of application level profiles, such as the one shown in figure A.3.

![Host Routines](image.png)

**Figure A.3:** Screenshot of the Nvidia visual profiler showing the profile of the TiO$_2$ nanoparticle benchmark. The top time line shows NVTX ranges for routines executed on the host CPU.
Figure A.4: Timing call graph of the MD simulation performed on 1024 nodes from section 4.4.8. The percentages shown in the graph are slightly distorted, because routines that account for less than 5% of the costs were neglected for clarity. The graphics was generated using the KCachegrind software.
A software package like CP2K does not only consist of its bare source code. Additionally, many other types of information are required to successfully compile, execute, and extend the code. Much of this know-how is spread across the wider community, including many non-developers. What was needed was an easy way of gathering and sharing this knowledge. Therefore setup the CP2K wiki to provide a central hub for documentation. In June 2013 the wiki replaced the five static pages that made up the CP2K website at the time. The wiki was made editable by the general public and the entire community was invited to contribute. This implied a cultural shift from a central authorship to a community driven organic website.

Since 2013 the CP2K wiki has acquired more than 200 pages of content. Popular categories include howtos, frequently asked questions, step-by-step exercises, tools, acronyms, and performance statistics. Also much of CP2K’s technical documentation has found its way into the wiki. This includes instructions on profiling, debugging, testing, and releases. A particularly successful channel for acquiring high quality content was to use the wiki for hosting exercise material for lectures and summer schools. In the future the wiki could also be more tightly integrated with the CP2K manual and the API-documentation.

Figure A.5: Screenshot of the DokuWiki-based CP2K website at https://www.cp2k.org.
Atomic Simulation Environment

Statically typed languages, like Fortran, are ideal for the implementation of large software packages. However, strong typing is less beneficial for small, prototypical, or short-lived programs. For these tasks, Python has found wide adoption by the scientific community. Python’s success is based, among other things, on its large standard library and high extensibility.

In the case of atomistic simulations, the existing simulation codes have to be made accessible from Python. The quasi-standard for providing this interface is the Atomic Simulation Environment (ASE). The ASE is a set of tools and Python modules for setting up, manipulating, running, visualizing and analyzing atomistic simulations. Its code is freely available under the GNU LGPL license [124]. The ASE library is structured around a large set of plugins, so-called calculators, for interacting with simulation codes. While most popular codes are supported, CP2K was for a long time absent from that list. I therefore developed the missing calculator to make CP2K available within the ASE.

The CP2K calculator is innovative in the way that it communicates via pipes instead of the more common file I/O. For this the CP2K distribution comes with a small helper program called CP2K-shell. It provides a simple interactive command line interface with a well defined, parsable syntax. When invoked, the CP2K calculator calls popen [125] to launch the CP2K-shell as a sub-process and attaches pipes to its stdin and stdout file handles. This even works together with MPI, because most MPI-implementations forward the stdin/stdout of the mpiexec process to the rank-0 process. The CP2K calculator also allows for multiple CP2K instances to be controlled from within the same Python process, simply by instantiating multiple calculator objects simultaneously. An in-depth review of the ASE, including my contributions, is in preparation [126].

![Supported Calculators](image)

**Figure A6:** Simulations packages supported by the Atomic Simulation Environment now also include CP2K. Screenshot from ASE project website [124].


Bibliography


