Low-Scaling Electronic Structure Methods Based on Sparse Tensor Contraction

Dissertation
zur
Erlangung der naturwissenschaftlichen Doktorwürde (Dr. sc. nat.)
vorgelegt der
Mathematisch-naturwissenschaftlichen Fakultät
der
Universität Zürich
von
Patrick Seewald
von
Männedorf ZH

Zürich, 2021

Contents

Abstract

Linear algebra with tensors of rank 2–4 is ubiquitous in electronic structure theory. The occuring tensors and matrices often have a sparse representation (in which most entries are negligibly small) if a localized basis is chosen. This work is concerned with a general implementation of sparse tensor contractions laying the foundation for low-scaling algorithms for RPA, GW and Hartree-Fock Exchange. These methods are based on the resolution-of-the-identity (RI) approximation with a local metric, ensuring sparse integral tensors, and have an effective quadratic scaling with system size. The development of a generic and abstract tensor API overcomes the manual implementation and optimization of application-specific code and enables systematic optimizations based on the concepts of multi-dimensional tensors and tall-and-skinny matrices. We demonstrate that the low-scaling implementation extends the applicability of RPA to systems with more than 5000 electrons described with a correlation-consistent triple-zeta basis. Systems of this size can no longer be calculated with canonical RPA due to memory constraints. Our low-scaling RPA variant is not limited to systems with convenient properties (such as a large band gap or low dimensionality) but is of advantage also for densely packed solid state systems as is shown by the example of $6 \times 6 \times 2$ unit cells of bulk anatase with 864 atoms. The novel RI-based approach to Hartree-Fock Exchange can be one order of magnitude more expensive than the direct implementation for sparse systems and adapted basis sets but can be advantageous for dense systems with large and accurate basis sets.

An analytical method (MME) for periodic electron repulsion integrals (ERIs) has been derived in order to facilitate the use of Gaussian-type basis functions for periodic systems, as an alternative to the numerical GPW method. The MME method relies on a minimax approximation of the Fourier-transformed Coulomb potential by a sum of Gaussians combined with Ewald-like summation techniques for rapid convergence. It enables all-electron RI-MP2/RPA calculations for periodic systems and accelerates Image Charge Augmented QM/MM for adsorbates on metals.

Zusammenfassung

Lineare Algebra mit Tensoren von Rang 2–4 ist allgegenwärtig in der Theorie der elektronischen Struktur. Die darin vorkommenden Tensoren und Matrizen haben oft eine dünnbesetzte Darstellung (in der die meisten Einträge vernachlässigbar klein sind), wenn eine lokalisierte Basis gewählt wird. Diese Arbeit beschäftigt sich mit einer allgemeinen Implementierung von dünnbesetzten Tensorkontraktionen, welche den Grundstein legt für niedrigskalierende Algorithmen für RPA, GW and Hartree-Fock-Austausch. Diese Methoden basieren auf der Resolution-of-the-Identity (RI) Approximation mit einer lokalen Metrik, was dünnbesetzte Integrale gewährleistet, und skalieren effektiv quadratisch mit der Systemgrösse. Die Entwicklung einer generischen und abstrakten Tensor-Programmierschnittstelle überwindet die manuelle Implementierung und Optimierung von anwendungsspezifischem Code und ermöglicht systematische Optimierungen basierend auf den Konzepten von multidimensionalen Tensoren und hoch-und-dünnen Matrizen. Wir zeigen dass die niedrigskalierende Implementierung die Anwendung von RPA auf Systeme mit mehr als 5000 Elektronen ermöglicht, beschrieben mit einer korrelationskonsistenten tripel-zeta Basis. Systeme dieser Grösse können nicht mehr mit kanonischem RPA berechnet werden wegen zu hohem Speicherbedarf. Unsere niedrigskalierende Variante von RPA ist nicht limitiert auf Systeme mit günstigen Eigenschaften (wie eine grosse Bandlücke oder niedrige Dimensionalität), sondern ist von Vorteil auch für dicht gepackte Festkörper, wie gezeigt wird am Beispiel von $6 \times 6 \times 2$ Einheitszellen von Anatas mit 864 Atomen. Der neue RI-basierende Ansatz für Hartree-Fock-Austausch kann eine Grössenordnung teurer sein als die direkte Implementierung für dünnbesetzte Systeme und angepasste Basissätze, jedoch kann er von Vorteil sein für dichte Systeme mit einer grossen und genauen Basis.

Eine analytische Methode (MME) für periodische Elektronabstossungsintegrale (ERIs) wurde hergeleitet um die Verwendung von Gauss'schen Basisfunktionen für periodische Systeme zu erleichtern, als eine Alternative zur numerischen GPW-Methode. Die MME Methode basiert auf einer Minimax-Approximation des Fourier-transformierten Coulomb-Potentiales durch eine Summe von Gaussfunktionen kombiniert mit Ewald-ähnlichen Summierungstechniken für schnelle Konvergenz. Sie ermöglicht All-Elektronen RI-MP2/R-PA Berechnungen für periodische Systeme und beschleunigt Spiegelladung-erweitertes QM/MM für Adsorbaten auf Metallen.

Chapter 1

Introduction

Electronic Structure methods beyond the density functional theory (DFT) approximation are systematic improvements for the description of material properties ranging from molecules to solid state systems, however at significant higher computational costs than DFT. As computers become more powerful, these intrinsically expensive methods can be extended to larger and more realistic systems. Equally important are algorithmic improvements that reduce computational cost and memory footprint, ideally at a reduced scaling with system size. A promising strategy for reducing system size scaling is the use of a sparse representation of matrices and tensors (in which most elements are negligibly small) by expressing them in terms of local atom-centered basis functions and by the use of the resolution-of-identity (RI) approximation with a local metric.

Linear algebra with tensors of rank 2–4 is ubiquitous in electronic structure method. If the occuring tensors and matrices have a sparse representation, an implementation specialized for sparse tensors is required in order to derive computational savings from sparsity. The availability of a sparse tensor library could accelerate code development since many algorithms can be expressed naturally in terms of tensor contraction and tensor transformations (including data transposition and redistribution). Such operations are often tediously developed on a case-by-case basis and a generalized tensor library could simplify this process by providing generic operations that translate concise symbolic notations into optimized code. This library could also inspire novel low-scaling / sparse algorithms that were previously avoided in favor of dense algorithms only due to the availability of efficient libraries for dense linear algebra. Dense linear algebra has the natural advantage of being able to use high performance computing ressources more efficiently. This advantage is however less important than scaling with system size: if sparse algorithms have an improved scaling with system size they always outperform the dense variant given a large enough system size, irrespective of the absolute performance at which the respective operations can be performed.

This work is concerned with the implementation of such a library based on the existing DBCSR library [1] for block-sparse matrix multiplications. The DBCSR library focuses on concurrency using distributed memory parallelism and maximizing performance relying on optimizations for small dense blocks (including CPU and GPU backends). The tensor generalization of this library should maintain performance at the same time as providing an abstract API for general tensor-based operations.

Applications of such a sparse tensor library include already published algorithms for low-scaling RPA [2] and GW [3] and an analogous RI-based approach for Hartree-Fock Exchange. The use of a local metric together with localized Gaussian-type basis functions are the key ingredients enabling a sparse formulation of these methods. The dominant scaling with respect to system size is $O(N^2)$ for all methods and all systems, even truly 3-dimensional condensed phase systems with a small band gap.

Low-scaling RPA and GW have already been demonstrated to scale to large systems containing thousands of atoms where the largest system sizes considered were 864 water molecules for bulk water with basis sets of cc-TZV2P quality [2] and 1734 atoms for Graphene nanoribbons with basis sets of aug-DZVP quality [3]. The algorithm is a prime example of sparse tensor contraction and the availability of a sparse tensor library would not only greatly simplify the implementation but would also allow for systematic tensor-based optimizations. In contrast to the initial implementation in which the parallel layout of large matrices is a user input to a calculation, the final implementation should be ideally parameter-free, except for a parameter to control the truncation of small tensor elements.

For Hartree-Fock Exchange low-scaling algorithms do not require the use of RI and integral screening is sufficient to achieve a scaling close to $O(N)$ in system size [4]. The resolution of identity approach can however still be an improvement for condensed phased systems with high quality basis sets for which the number of integrals to be calculated and stored grows very large already for medium-sized systems despite integral screening.

Chapter 2

Theory

2.1 Introduction

This chapter introduces the basic theoretical concepts that serve as a basis for the algorithmic methods developed in the following chapters. The work described in this thesis finds its main applicability in ab initio electronic structure methods on the level of Hartree-Fock Exchange, electron correlation (RPA/MP2) and GW. These methods are thus the primary focus of this chapter and for a more general introduction into the topic we refer to standard textbooks [5–8].

A systematic classification of the presented method is given by Perdew et al. [9]. As methods satisfy more and more constraints of the exact limit, they get more accurate but also more expensive. This classification sets apart ab initio methods from semiempirical methods that typically fail to reproduce exact limits. The resulting hierarchy of methods consists of 5 steps in Jacob's ladder of density functional approximations: the first 3 steps are reserved to the DFT world (LSD/LDA, GGA and meta-GGA), the 4th rung adds exact (Hartree-Fock) exchange. The MP2 & RPA methods belong to the 5th rung as they correctly describe non-local dynamic electron correlation. Since the required computational costs increase rapidly on the fourth and fifth run, applications for realistic systems are typically restricted to the first three rungs. On the upside there is increasing interest in reducing the costs of the higher rungs to make them applicable to larger systems.

The following index notation and abbreviations are used throughout this thesis: $i, j, k, \dots$ refer to canonical occupied molecular orbitals (MOs), $a, b, c, \dots$ to canonical virtual MOs, $\mu, \nu, \lambda, \dots$ to atomic orbital basis set functions (AOs) and $P, Q, R, \dots$ to auxiliary resolution-of-the-identity (RI) basis set functions.

2.2 Hartree-Fock Exchange and Density Functional Theory

2.2.1 Hartree-Fock Exchange

The Hartree-Fock equation is an approximation of the wave function $\Psi(\mathbf{x}) = \Psi(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$ and the energy of a system of N electrons in a stationary state under the following assumptions:

- Born-Oppenheimer approximation: due to the nuclei being much heavier than the electrons, the electronic and nuclear degrees of freedom can be separated such that only the N electronic degrees of freedom are explicitly included in the Hamiltonian.
- The wavefunction can be expressed as an antisymmetrized product (Slater-determinant) of N orthonormal one-electron functions (orbitals) $\psi_i(\mathbf{x})$.

We emphasize that the number of electrons N is the number of valence electrons in case pseudopotentials [10] are employed to parametrize the effective potential of the core electrons instead of including them explicitly. Here only the second approximation is specific to Hartree-Fock and methods that go beyond Hartree-Fock (such as RPA & MP2) can be viewed as corrections to the second approximation. From these approximations the ground state energy of the electronic system can be obtained by minimizing the expression of the energy w.r.t. the orbitals $\psi_i(\mathbf{x})$ (based on the variational principle) under the constraint that the orbitals ψ_i are orthonormal. The Hartree-Fock energy can be expressed as

$$E^{\mathrm{HF}} = \min_{\psi_i} \langle \Psi | H^{\mathrm{HF}} | \Psi \rangle = \min_{\psi_i} \left(\sum_i^N H_i(\mathbf{R}) + \frac{1}{2} \sum_{i,j,i\neq j}^N (J_{ij} - K_{ij}) \right) \tag{2.1}$$

with the following one- and two-electron integrals

$$H_i(\mathbf{R}) = \int \psi_i^*(\mathbf{r}) \left[-\frac{1}{2}\nabla^2 + v(\mathbf{r}, \mathbf{R}) \right] \psi_i(\mathbf{r}) d\mathbf{r} \tag{2.2}$$

$$J_{ij} = \int\int \psi_i(\mathbf{r}_1)\psi_i^*(\mathbf{r}_1) \frac{1}{|\mathbf{r}_2 - \mathbf{r}_1|} \psi_j^*(\mathbf{r}_2)\psi_j(\mathbf{r}_2) d\mathbf{r}_1 d\mathbf{r}_2 \tag{2.3}$$

$$K_{ij} = \int\int \psi_i^*(\mathbf{r}_1)\psi_j(\mathbf{r}_1) \frac{1}{|\mathbf{r}_2 - \mathbf{r}_1|} \psi_i(\mathbf{r}_2)\psi_j^*(\mathbf{r}_2) d\mathbf{r}_1 d\mathbf{r}_2 \tag{2.4}$$

and the external potential $v(\mathbf{r}, \mathbf{R})$ collecting the electron-nuclei interations. The Hartree term J_{ij} describes the classical Coulomb interaction of electron charge distributions whereas the Exchange term K_{ij} describes the quantum mechanical Pauli repulsion of electrons.

The orbitals $\psi_i(\mathbf{r})$ are expanded into a set of basis functions $\phi_\mu(\mathbf{r})$

$$\psi_k(\mathbf{r}) = \sum_\mu C_{\mu k}\phi_\mu(\mathbf{r}) \tag{2.5}$$

where a finite, incomplete basis set is used in practice. It is advantageous to use basis functions centered around the atom position for an accurate description of the chemically active regions. Locality of basis functions can also be exploited to design efficient algorithms based on integral screening and sparse linear algebra. The functions $\psi_k(\mathbf{r})$ and $\phi_\mu(\mathbf{r})$ are referred to as molecular orbitals (MO) and atomic orbitals (AO), respectively. The density matrix is defined as

$$P_{\mu\nu} = \sum_i C_{\mu i}C^*_{\nu i}n_i \tag{2.6}$$

n_i being the occupation number of MO i. Without loss of generality, we consider restricted closed-shell system (each orbital is occupied by two electrons with opposite spin). We assume insulating, non-metallic systems for which the occupation number is a step function

$$n_i = \begin{cases} 2, & \text{if } i \leq N/2 \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

The electron density $\rho(\mathbf{r})$ can then be expressed in terms of the AOs as

$$\rho(\mathbf{r}) = 2\sum_i^{N/2} \psi_i^*(\mathbf{r})\psi_i(\mathbf{r}) = \sum_{\mu\nu} P_{\mu\nu}\phi_\mu(\mathbf{r})\phi_\nu^*(\mathbf{r}) \tag{2.8}$$

In this parametrization of the wave function in terms of a finite AO basis, the energy needs to be minimized w.r.t. the coefficients $C_{\mu k}$. The Hartree-Fock energy is the sum of Hartree energy E_H and Exchange energy E_x^HF. The superscript HF indicates that the exact exchange is specific to Hartree-Fock theory in contrast to the Hartree energy which is identical to density functional theory. The final expression for the Hartree-Fock energy reads as

$$E^\mathrm{HF} = E_\mathrm{H} + E_\mathrm{x}^\mathrm{HF} = \sum_{\mu\nu} P_{\mu\nu}(H^\mathrm{H}_{\mu\nu} + \Sigma^\mathrm{x}_{\mu\nu}) \tag{2.9}$$

with the Hartree matrix given by

$$H^{\mathrm{H}}_{\mu\nu} = \langle \mu | V^{h}(\mathbf{r}) | \nu \rangle \,, \tag{2.10}$$

$$V_{\mathrm{H}}(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' \tag{2.11}$$

and the Fock matrix given by

$$\Sigma^{\mathrm{x}}_{\mu\nu} = -\frac{1}{2} \sum_{\lambda\sigma} (\mu\lambda|\sigma\nu) P_{\lambda\sigma} \tag{2.12}$$

The shorthand notation $(\mu\lambda|\sigma\nu)$ refers to 4-center Electron Repulsion Integrals (ERIs) given by

$$(\mu\lambda|\sigma\nu) = (\phi_\mu\phi_\lambda|\phi_\sigma\phi_\nu) = \int\int \phi^*_\mu(\mathbf{r}_1)\phi^*_\lambda(\mathbf{r}_1)\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|}\phi_\sigma(\mathbf{r}_2)\phi_\nu(\mathbf{r}_2) d\mathbf{r}_1 d\mathbf{r}_2 \tag{2.13}$$

Eq. (2.9) must be solved for the density matrix which can be done either self-consistently by the Self-Consistent-Field (SCF) method or by using orbital transformation (OT) [11]. With the SCF method, given an initial density matrix, an initial Fock matrix can be built. A new density matrix can be obtained by diagonalization of the Fock matrix. Self-consistency is achieved by updating the density matrix in each diagonalization step, using a suitable density-mixing scheme to achieve smooth convergence. With the OT method, a minimization procedure is applied that minimizes the energy E^{HF} w.r.t. the MO coefficients.

2.2.2 Kohn-Sham Density Functional Theory

An alternative approach to solve the electronic Schrödinger equation approximately is Kohn-Sham Density Functional Theory (in the following abbreviated as DFT). DFT is based on the Hohenberg-Kohn theorems [12] that state that the electron many-body problem can be equivalently phrased in terms of the electron density $\rho(\mathbf{r})$ (depending only on one degree of freedom) instead of the wave function $\Psi(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$. To practically solve the resulting minimization problem, one-electron orbitals (similar as in Hartree-Fock theory) are introduced by assuming that a non-interacting system of electrons exist with the same ground state density as the non-interacting system.

The Kohn-Sham equation [13] can be written in a form analogous to Eq. (2.9)

$$E^{\mathrm{KS}} = \sum_{\mu\nu} P_{\mu\nu} H^{\mathrm{KS}}_{\mu\nu} \tag{2.14}$$

with

$$\begin{aligned} H^{\mathrm{KS}}_{\mu\nu} &= \langle\mu|H^{\mathrm{KS}}|\nu\rangle \ , \\ H^{\mathrm{KS}} &= -\frac{1}{2}\nabla^2 + V(\mathbf{r},\mathbf{R}) + V_{\mathrm{eff}}(\mathbf{r}) \ , \\ V_{\mathrm{eff}}(\mathbf{r}) &= V_{\mathrm{H}}(\mathbf{r}) + V_{\mathrm{xc}}(\mathbf{r}) \end{aligned}$$

Here V_{H} corresponds to the Hartree term in Hartree-Fock Eq. (2.11) and V_{xc} is the exchange-correlation potential. The exchange-correlation potential V_{xc} is conceptually introduced as the unknown potential that would make Kohn-Sham DFT an exact theory. By definition it includes the exchange term from Hartree-Fock Eq. (2.12). Correlation is then the electron interaction missing in the Hartree-Fock approximation. Since in DFT $V_{\mathrm{xc}}(\mathbf{r})$ is an unknown quantity it can only be approximated. In contrast to Hartree-Fock, DFT can not exactly describe exchange. Exact exchange is non-local in nature in the sense that the exchange potential depends explicitly on all orbitals, whereas in DFT it is only a function of $\mathbf{r}$. As a further disadvantage, DFT introduces a self-interaction error since each electron interacts with the total density of all electrons including itself. What makes DFT compelling is that it can approximate the correlation which is entirely missing in Hartree-Fock, and that it is at least one order of magnitude cheaper than Hartree-Fock.

2.2.3 Hybrid functionals

The close formal resemblance of DFT and Hartree-Fock can be used to incorporate exact exchange into density functional theory. Hybrid functionals replace a certain fraction of approximate DFT exchange with exact exchange in V_{xc}

$$E^{\mathrm{hybrid}}_{\mathrm{xc}} = \alpha E^{\mathrm{HF}}_{\mathrm{x}} + (1-\alpha) E^{\mathrm{KS}}_{\mathrm{x}} + E^{\mathrm{KS}}_{\mathrm{c}} \tag{2.15}$$

2.3 MP2 & RPA Electron Correlation with RI

MP2 and RPA can both be viewed as corrections to the Hartree-Fock or DFT energies to add the missing electron correlation. The second order Møller-Plesset energy [14–20] is obtained by Rayleigh-Schrödinger perturbation theory in which the zeroth order Hamiltonian is the Fock operator. The closed shell MP2 correlation energy E^{MP2} is obtained as

$$E^{\mathrm{MP2}} = -\sum_{ij,ab}^{\mathrm{occ,vir}} \frac{(ia|jb)[2(ia|jb) - (ib|ja)]}{\epsilon_a + \epsilon_b - \epsilon_i - \epsilon_j} \tag{2.16}$$

Indices i, j refer to occupied and a, b to virtual MOs and $\epsilon_i, \epsilon_a, \ldots$ to the corresponding Hartree-Fock or DFT orbital energies. The shorthand notation $(ia|jb)$ refers to 2-electron electron repulsion integrals (ERIs) as defined in Eq. (2.13). In order to avoid the expensive calculation of 2-electron ERIs the Resolution of the Identity (RI) Approximation is applied to save memory and computational cost.

2.3.1 Resolution of the Identity (RI) Approximation

The computation and storage of two-electron (four-center) integrals can be a major bottleneck in post-Hartree-Fock methods. The Resolution of the Identity (RI) approximation can be used to eliminate one center so that the maximum rank of integral tensors is reduced from four to three. Products of AO are expanded into auxiliary basis functions [21–29]

$$\phi_\mu(\mathbf{r})\phi_\nu(\mathbf{r}) = \sum_P C^P_{\mu\nu}\phi^{\mathrm{aux}}_P(\mathbf{r}) \tag{2.17}$$

so that two-electron electron repulsion integrals reduce to

$$(\mu\nu|\lambda\sigma) \approx \sum_{PQ} C^P_{\mu\nu}(P|Q)C^Q_{\lambda\sigma} \tag{2.18}$$

The expansion coefficients $C^P_{\mu\nu}$ are not uniquely determined and should be chosen to minimize the expansion error. A least-square fit to minimize the norm of the residual (the difference between an AO pair and its RI expansion) leads to the overlap RI metric [25, 28, 29]

$$C^P_{\mu\nu} = \sum_Q (\mu\nu Q)S^{-1}_{QP} \tag{2.19}$$

with the overlap integrals

$$(\mu\nu Q) = \int \phi_\mu(\mathbf{r})\phi_\nu(\mathbf{r})\phi_Q^{\text{aux}}(\mathbf{r})d\mathbf{r}, \tag{2.20}$$

$$S_{QP} = \int \phi_Q(\mathbf{r})\phi_P(\mathbf{r})d\mathbf{r} \tag{2.21}$$

Alternatively (and more accurately) the RI error of the original four-center integrals can be minimized leading to the Coulomb RI metric [21–23, 25, 26]

$$C_{\mu\nu}^P = \sum_Q (\mu\nu|Q)V_{QP}^{-1} \tag{2.22}$$

where the integrals V_{QP}^{-1} and $(\mu\nu|Q)$ are Coulomb integrals analogous to Eq. (2.13).

For a general potential $V_1(r)$ the 2-electron integrals of the form

$$(\mu\lambda|\sigma\nu)_1 = \int\int \phi_\mu^*(\mathbf{r}_1)\phi_\lambda^*(\mathbf{r}_1)V_1(|\mathbf{r}_2-\mathbf{r}_1|)\phi_\sigma(\mathbf{r}_2)\phi_\nu(\mathbf{r}_2)d\mathbf{r}_1d\mathbf{r}_2. \tag{2.23}$$

have an RI approximation

$$(\mu\lambda|\sigma\nu)_1 \approx \sum_{PQ} (\mu\lambda|Q)_2(Q|R)_2^{-1}(R|S)_1(S|P)_2^{-1}(P|\sigma\nu)_2 \tag{2.24}$$

where the subscript i in $(\cdots)_i$ denotes the operator $V_i(r)$ of the integral. Choosing $V_1 = V_2$ is most accurate. In practice a more local $V_2(r)$ (decaying quickly for increasing r) is beneficial to obtain a sparser representation of the 3-center ERIs. Choosing $V_2(r) = \delta(r)$ is the most local choice and recovers the overlap RI metric Eq. (2.19). Attenuated or truncated Coulomb potentials can serve as more reasonable compromises between accuracy and locality [28].

2.3.2 RI-MP2 & RI-RPA

Introducing the RI approximation with the Coulomb metric the 2-electron ERIs in the MP2 energy expression Eq. (2.16) are calculated as

$$(ia|jb) \approx \sum_P B_P^{ia}B_P^{jb} \tag{2.25}$$

with the tensor B_P^{ia} given by

$$B_P^{ia} = \sum_R (ia|R) L_{PR}^{-1} \tag{2.26}$$

and the triangular matrix L_{PR} obtained by the Cholesky decomposition of $(P|Q)$

$$(P|Q) = \sum_R L_{PR} L_{RQ}^T \tag{2.27}$$

Within the RI approximation the dRPA (direct RPA without exchange contributions) energy can be expressed as a frequency integral [30–33]

$$E^{\mathrm{RI-dRPA}} = \frac{1}{2} \int_{-\infty}^{+\infty} \frac{d\omega}{2\pi} \mathrm{Tr}(\ln(\mathbf{1} + \mathbf{Q}(\omega)) - \mathbf{Q}(\omega)) \tag{2.28}$$

The matrix $\mathbf{Q}(\omega)$ is given by

$$Q_{PR}(\omega) = 2 \sum_{ia} B_P^{ia} G_{ia}(\omega) B_R^{ia}, \tag{2.29}$$

$$G_{ia}(\omega) = (\epsilon_a - \epsilon_i)((\epsilon_a - \epsilon_i)^2 + \omega^2)^{-1} \tag{2.30}$$

$$\tag{2.31}$$

The frequency integral Eq. (2.28) can be most efficiently discretized by a minimax approximation [34].

2.4 Periodic Systems

In this section methods specific to periodic boundary conditions are reviewed. Conceptually periodic systems require **k**-point integration. Since the electronic structure methods presented here are optimized for large systems, only the Γ-point $\mathbf{k} = (0, 0, 0)$ is needed which is an approximation that holds in the limit of large simulation cells [35]. Whereas ERIs are traditionally calculated with standard analytical schemes [36], periodic boundary conditions imply virtually infinite lattice sums for non-local operators so that other approaches for ERI calculation need to be considered. One topic of this thesis is an analytical scheme for periodic ERIs. We therefore introduce here the numerical GPW method which is the current standard method to calculate periodic ERIs.

2.4.1 Periodic Hartree-Fock Exchange with Truncated Coulomb Operator

Periodic boundary systems are usually applied to facilitate the simulation of extended systems in the condensed or liquid phase. This formally requires integration over **k**-vectors. In practice the **k**-vector integration is discretized by a finite mesh of **k**-points. The HFX energy is however troublesome to compute due to an integratable singularity at $\mathbf{k} - \mathbf{k}'$. In CP2K this is solved by replacing the Coulomb operator in the Exchange integral Eq. (2.4) with a truncated Coulomb (TC) operator [37, 38]:

$$g_{\mathrm{TC}}(r) = \begin{cases} \frac{1}{r}, & \text{if } r \leq R_c \\ 0, & \text{otherwise} \end{cases} \tag{2.32}$$

The minimum required R_c to yield an accurate exchange energy is a system dependent quantity where convergence is more rapid for systems with a large band gap. This can be seen from the Exchange energy expression Eq. (2.9) by noting that the decay of the density matrix implies that 2-electron integrals with large separation between the centers of μ and ν or λ and σ can be neglected. The number of required **k**-points depends on the cell volume V and the cutoff radius R_c according to [37, 38]

$$N_k \approx \frac{4\pi}{3} R_c^3 \frac{1}{V} \tag{2.33}$$

A Γ-point only implementation is justified as long as the cell length L satisfies [38]

$$L > 1.61 R_c \tag{2.34}$$

The optimal value for the cutoff radius R_c depends on the band gap and unit cell of a system. For cubic unit cell the recommendation $R_c \leq L/2$ holds [38].

2.4.2 Periodic Electron Repulsion Integrals

An ERI over periodic functions $a^{\mathrm{P}}(\mathbf{r}), b^{\mathrm{P}}(\mathbf{r}), c^{\mathrm{P}}(\mathbf{r}), d^{\mathrm{P}}(\mathbf{r})$ and a general potential $V(\mathbf{r}_1, \mathbf{r}_2)$ has the form

$$(ab|cd) = \int_{\Omega} d\mathbf{r}_1 \int_{\mathbb{R}^3} d\mathbf{r}_2 a^{\mathrm{P}}(\mathbf{r}_1) b^{\mathrm{P}}(\mathbf{r}_1) V(\mathbf{r}_1 - \mathbf{r}_2) c^{\mathrm{P}}(\mathbf{r}_2) d^{\mathrm{P}}(\mathbf{r}_2) \tag{2.35}$$

with Ω the domain of the simulation cell and the superscript P indicating periodic repetition over all unit cells (summation over lattice vectors $\mathbf{R}$):

$$f^{\mathrm{P}}(\mathbf{r}) = \sum_{\mathbf{R}} f(\mathbf{r} - \mathbf{R}) \tag{2.36}$$

If the potential is non-local (such as the Coulomb potential), the summation over periodic images Eq. (2.36) is virtually infinite in Eq. (2.35) and a plane wave representation via Fast Fourier Transform is required (as will be explained in the next section). If the potential is local (such as truncated Coulomb Eq. (2.32)), a small number of periodic images is required and the ERI can be calculated more efficiently with analytical methods such as the Obara-Saika scheme [36].

2.4.3 Gaussian and Plane Wave (GPW) Method

The Gaussian and Plane Waves method is based on a representation of molecular orbitals in terms of both atom-centric Gaussian basis functions and plane waves [39]. A disadvantage of this method is its reliance on a smoothly varying electron density or basis functions with a large extent. Thus all-electron calculations are not possible and introduction of pseudopotentials [10] to describe the core electrons is a requirement.

We describe here the GPW-based calculation of ERIs over a potential $V(\mathbf{r})$. Without loss of generality, we restrict ourselves to 3-center ERI of the form $(ab|c)$. This is the largest number of centers within the RI approximation and the calculation of 2-center ERIs is analogous. The periodic 3-center ERI $(ab|c)$ is defined as

$$(ab|c) = \int_{\Omega} d\mathbf{r}_1 \int_{\mathbb{R}^3} d\mathbf{r}_2 a^{\mathrm{P}}(\mathbf{r}_1) b^{\mathrm{P}}(\mathbf{r}_1) V(\mathbf{r}_1 - \mathbf{r}_2) c^{\mathrm{P}}(\mathbf{r}_2) \tag{2.37}$$

The GPW scheme is based on an alternative representation of a function a in reciprocal lattice space via the Fourier series

$$a^{\mathrm{P}}(\mathbf{r}) \approx \frac{1}{V} \sum_{|\mathbf{G}| \leq G_c} \widehat{a}(\mathbf{G}) e^{i\mathbf{G}\cdot\mathbf{r}} \tag{2.38}$$

where $\widehat{a}$ is the plane wave representation of a given by the Fourier transform

$$\widehat{a}(\mathbf{G}) = \int_{\Omega} d\mathbf{r} a^{\mathrm{P}}(\mathbf{r}) e^{-i\mathbf{G}\cdot\mathbf{r}} = \int_{\mathbb{R}^3} d\mathbf{r} a(\mathbf{r}) e^{-i\mathbf{G}\cdot\mathbf{r}} \tag{2.39}$$

The only approximation in Eq. (2.38) is the truncation of the sum according to the cutoff G_c which is accurate if the function $a(\mathbf{r})$ is not too localized. Note that due to the numerical evaluation of the plane waves a is not restricted to Gaussian-type. Within the GPW method it is therefore convenient to apply any index transformation on the Gaussian basis function itself before transforming to plane waves, instead of contracting the ERIs. In RI-MP2/RPA, instead of calculating the 3-center ERI $(ia|R)$ the tensor B_P^{ia} Eq. (2.26) is evaluated directly by defining the third center as $c^\text{p}(\mathbf{r}) = \sum_R \phi_R(\mathbf{r}) L_{PR}^{-1}$.

We define $v^c(\mathbf{r})$ as the convolution

$$v^c(\mathbf{r}_1) = \int_{\mathbb{R}^3} d\mathbf{r}_2 V(\mathbf{r}_1 - \mathbf{r}_2) c^\text{P}(\mathbf{r}_2). \tag{2.40}$$

which can be expressed as a Fourier series with Fourier coefficients

$$\widehat{v}^c(\mathbf{G}) = \widehat{c}(\mathbf{G}) \widehat{V}(\mathbf{G}) \tag{2.41}$$

For the case of the Coulomb potential the Fourier transform $\widehat{V}(\mathbf{k})$ is known to be $4\pi/|\mathbf{k}|^2$. Within the GPW scheme $v^c(\mathbf{r})$ can thus be obtained by an inverse Fast Fourier Transform (FFT) $\widehat{v}^c(\mathbf{G}) \xrightarrow{\mathcal{F}^{-1}} v^c(\mathbf{r})$ [40]. The 3-center ERI $(ab|c)$ can then be expressed as

$$(ab|c) = \int_\Omega d\mathbf{r} a^\text{P}(\mathbf{r}) b^\text{P}(\mathbf{r}) v^c(\mathbf{r}) = \sum_{\mathbf{R}} \int_{\mathbb{R}^3} d\mathbf{r} a(\mathbf{r}) b(\mathbf{r} - \mathbf{R}) v^c(\mathbf{r}) \tag{2.42}$$

As shown in [39] this integral can be discretized over FFT real space mesh points $\mathbf{r}_i$

$$(ab|c) = \sum_{\mathbf{r}_i \subset \Omega} \tilde{g}_{ab}(\mathbf{r}_i) v^c(\mathbf{r}_i) \tag{2.43}$$

with

$$\tilde{g}_{ab}(\mathbf{r}_i) = \frac{1}{V} \sum_{|\mathbf{G}|<G_c} \widehat{g_{ab}}(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}_i} \tag{2.44}$$

and $g_{ab}(\mathbf{r})$ being the Gaussian product $a(\mathbf{r})b(\mathbf{r}-\mathbf{R})$. The numerical evaluation starts with calculating the s-type matrix elements of $v^c(\mathbf{r})$ and its numerical derivatives, from which all higher elements can be calculated by using recursion relations.

2.5 Implementation of Exact Exchange, RPA & GW in CP2K

This section provides a technical description of the reference implementation of exact exchange, RPA & GW in CP2K. This work attempts to provide alternate and possibly more efficient algorithms for these methods as introdued in Sec. 2.6. A concise summary of the reference implementations from the perspective of computational cost is thus appropriate.

2.5.1 Hartree-Fock Exchange

The dominant operation in Eq. (2.9) is the calculation of the 4-center ERIs $(\mu\lambda|\sigma\nu)$. A naive implementation would scale as $O(N^4)$ with respect to system size N in terms of both computational cost and memory footprint. By inspecting the full expression for the exchange energy from Eq. (2.9)

$$E_x^{\mathrm{HF}} = -\frac{1}{2}\sum_{\mu\nu\lambda\sigma} P_{\mu\sigma}P_{\nu\lambda}(\mu\nu|\lambda\sigma) \tag{2.45}$$

it can be seen that only those ERIs are needed that are contracted with non-vanishing elements of the density matrix. Combined with the Schwarz inequality the screening criterion can be formulated as [4]

$$P_{\mu\sigma}P_{\nu\lambda}(\mu\nu|\lambda\sigma) \leq \max\left(|P_{\mu\sigma}|, |P_{\nu\lambda}|\right) \times \sqrt{|(\mu\nu|\mu\nu)| \cdot |(\lambda\sigma|\lambda\sigma)|} \leq \epsilon_{\mathrm{Schwarz}}, \tag{2.46}$$

such that all ERIs $(\mu\nu|\lambda\sigma)$ that satisfy this inequality (given some screening threshold $\epsilon_{\mathrm{Schwarz}}$) can be skipped. Due to ERI symmetries the number of integrals can be further reduced by a factor of 8. Once the ERIs are calculated they can be reused in all SCF steps in the self-consistent SCF procedure if enough memory is available to store them in-core. The total execution time of a Hartree-Fock calculation is then dominated by the integral calculation in the first SCF step. The required memory can be reduced by a large factor by compressing the ERI data and decompressing them on-the-fly as needed during the SCF procedure. The scaling with basis size is $O(N^4)$. High quality basis sets thus make Hartree-Fock much more expensive, even more so if they are highly contracted, if they have a high angular momentum quantum number or if they are very diffuse since all these feature increase the number of primitive Gaussians that contribute to ERI calculation.

A resolution-of-the-identity scheme to Hartree-Fock (RI-HFX) has the potential advantage of being less sensitive to the number of primitive Gaussians since integral calculation should not be the bottleneck of such an implementation and the remaining operations scale with the number of contracted basis functions. Furthermore the 3-center quantities are much less heavy in memory so that storage should be less of a problem in a RI-HFX scheme. A RI-HFX implementation has the disadvantage that due to the indirect evaluation of the Fock matrix, avoiding the calculation of 4-center integrals, the strategy of imposing sparsity based on the decay of density matrix elements can not be applied.

2.5.2 RPA

The by far most dominant step in RPA is the contraction Eq. (2.29) $\mathbf{B}'(\omega)^T\mathbf{B}'(\omega)$ with $\mathbf{B}'(\omega) = \mathbf{G}(\omega)^{1/2}\mathbf{B}$ which is a multiplication of tall-and-skinny matrices that can be efficiently performed with a GPU-accelerated Scalapack [41]. The matrix $\mathbf{B}'$ grow as $O(N^3)$ in memory so that the feasible system size is limited by the amount of RAM provided even by the largest machines available (the largest system calculated so far being 512 water molecules).

A low-scaling implementation of RPA & GW promises to provide a reduced scaling of $O(N^3)$ w.r.t. both memory and execution time [2, 3]. Such an implementation relies on sparse tensor contractions which is a highly non-standard and customized operation. Therefore good performance and a stable, easy-to-use implementation can only be expected if more effort is spent on the development and optimization of numerical primitives that serve as the backbone of the actual RPA implementation. Even with an optimized implementation of sparse tensor contractions, it can not be expected that a high percentage of the peak performance of modern supercomputers will be achieved since sparse computations with heterogeneous block sizes are inherently problematic for GPUs. Despite this disadvantage in terms of absolute performance, the hope is that the reduced scaling will eventually win against an efficient quartic scaling RPA implementation and enable RPA for system sizes that were previously out of reach. The potential advantage of the low-scaling algorithms has already been demonstrated in [2], even though the comparison was performed on CPUs.

2.6 Low-scaling Algorithms

This section introduces alternative formulations of Hartree-Fock Exchange, RPA and GW that attempt to reduce the scaling and/or required memory. The presented algorithms have in common that they exploit sparsity by the use of a local RI metric and Gaussian basis functions. The tensor operations that are needed for an actual implementation will be discussed later in chapter 4.

2.6.1 RI Hartree-Fock Exchange

The Fock matrix $\Sigma^{\mathrm{x}}_{\mu\nu}$ can be written within the resolution-of-the-identity (RI) approximation as

$$\Sigma^{\mathrm{x}}_{\mu\nu} = -\frac{1}{2}\sum_{\lambda\sigma}(\mu\lambda|\sigma\nu)P_{\lambda\sigma} \overset{\mathrm{RI}}{\approx} -\frac{1}{2}\sum_{\lambda\sigma}\sum_{PQRS}(\mu\lambda|P)S^{-1}_{PR}V_{RS}S^{-1}_{SQ}(Q|\sigma\nu)P_{\lambda\sigma} \tag{2.47}$$

where the 2-center ERIs V_{RS} are integrals over the Hartree-Fock potential V_{x} which is the Coulomb operator for isolated systems and the truncated Coulomb operator with some cutoff radius for periodic systems. The 2-center ERIs S_{PR} and the 3-center ERIs $(\mu\lambda|P)$ are integrals over the RI metric V_{RI}. For accuracy V_{RI} should approximate V_{x} and for efficiency it should be local such that $(\mu\lambda|P) \approx 0$ if the centers of both functions μ, ν are far off from the center of P. More precisely the screening criterion

$$(\mu\lambda|P) \approx 0 \text{ if } (d_{\mu P} > d_{V_{\mathrm{RI}}} + d_{\mu} + d_P) \text{ and } (d_{\lambda P} > d_{V_{\mathrm{RI}}} + d_{\lambda} + d_P) \tag{2.48}$$

holds where $d_{\mu P}$ is the distance between the centers of μ and P, $d_{V_{\mathrm{RI}}}$ is the extent of the RI metric (equal to the cutoff radius in case of a truncated potential) and d_{μ}, d_{λ} and d_P are the radii of the functions μ, λ and P which are assigned according to some truncation threshold. The screening criterion

$$(\mu\lambda|P) \approx 0 \text{ if } d_{\mu\lambda} > d_{\mu} + d_{\lambda} \tag{2.49}$$

always holds independently of V_{RI}.

The only quantity that changes over the course of a self-consistent SCF procedure is the density matrix. As in canonical Hartree-Fock the quantities independent of the density matrix should be evaluated in the first SCF step only. The following contractions are thus computed in the first SCF step (mentioning also the formal scaling with system size):

$$\begin{array}{lc} \text{Contraction} & \text{scaling: cost/memory} \\ K_{PQ} = \sum_{RS} S^{-1}_{PR} V_{RS} S^{-1}_{SQ} & O(N^3)/O(N^2) \end{array} \tag{2.50}$$

$$M^{(1)}_{Q\lambda\mu} = \sum_{P} (\mu\lambda P) K_{PQ} \qquad O(N^2)/O(N^2) \tag{2.51}$$

The remaining contractions which are repeated in each SCF step are:

$$\begin{array}{lc} \text{Contraction} & \text{scaling: cost/memory} \\ M^{(2)}_{Q\lambda\nu} = \sum_{\sigma} (Q\sigma\nu) P_{\lambda\sigma} & O(N^2)/O(N^2) \end{array} \tag{2.52}$$

$$\Sigma^{\mathrm{x}}_{\mu\nu} = \sum_{Q\lambda} M^{(1)}_{Q\lambda\mu} M^{(2)}_{Q\lambda\nu} \qquad O(N^2)/O(N^2) \tag{2.53}$$

For the scaling relations we did not assume decaying density matrices. If density matrices become sparse for large systems, formal scaling of Eq. (2.52) and Eq. (2.53) is reduced to $O(N)$ (cost and memory).

The matrix K_{PQ} is dense irrespective of the chosen RI metric. Even though some sparsity remains in $M^{(1)}_{Q\lambda\mu}$ (the sparsity associated with criterion Eq. (2.49)), the quantity $M^{(1)}_{Q\lambda\mu}$ is expected to be the largest tensor (with the largest number of non-zeros). This tensor must be fully stored in memory so that it does not need to be recomputed in every SCF step. The large memory requirements can be reduced by a strategy combining a compression scheme with partial contraction in multiple batches: Each portion of $M^{(1)}_{Q\lambda\mu}$ is calculated and compressed immediately. The respective batches are decompressed when needed in each SCF step (where the contractions are also performed in multiple batches). The compression algorithm that was developed for the direct HFX implementation [4] can be reused for that purpose. The batches are defined by a static decomposition of the indices Q and λ. If each of the two indices is split into n_{mem} batches, the associated memory savings are n^2_{mem}. This strategy can thus reduce the memory footprint by a large factor, ideally without any additional computations or communications.

2.6.2 Low-scaling RI-RPA

The scaling of RPA and SOS-MP2 can be reduced from $O(N^4)$ to $O(N^3)$, or even better by alternative analytical formulations of the methods [34, 42–46]. Here we describe the

CP2K-specific implementation that was initially developed by J. Wilhelm [2]. The effective system size scaling is $O(N^2)$ ($O(N^3)$ in the limit of very large systems), independently of the properties of the system under consideration. The universal $O(N^2)$ scaling makes the low-scaling approach applicable to systems of all nature, including dense condensed-phase systems with a small (but non-vanishing) band gap. Low dimensionality or local electronic structure giving rise to sparse density matrices improve performance and bring down effective scaling to $O(N)$.

For low-scaling RPA, the matrix $Q_{PQ}(\omega)$ (2.29) is transformed to imaginary time [2, 46] $Q_{PQ}(\tau)$ by a cosine transformation

$$Q_{PQ}(\omega) = 2\int_0^\infty d\tau Q_{PQ}(\tau)\cos(\tau\omega) \tag{2.54}$$

and discretized on minimax grids for time and frequency with M grid points each

$$Q_{PQ}(\omega_k) = 2\sum_{j=1}^{M} \lambda_{kj} Q_{PQ}(\tau_j)\cos(\tau_j\omega_k) \tag{2.55}$$

where λ_{kj} are the integration weights. The tensor B_P^{ia} Eq. (2.26) is transformed from occupied-virtual molecular orbital pairs ia to pairs $\mu\nu$ of atomic orbital basis set functions. This decouples the sum over occupied and virtual orbitals and thereby reduces the formal scaling from quartic to cubic. Further requirements for a cubic scaling behavior are the use of localized atomic Gaussian basis functions and the localized overlap RI metric such that the occurring 3-center integrals are sparse.

The working equation for low-scaling RPA is

$$P_{RT}(\tau_j) = \sum_{\mu\sigma}\left(\sum_{\lambda}(\lambda\sigma|R)D^{\text{occ}}_{\mu\lambda}(\tau_j)\right)\left(\sum_{\nu}(\mu\nu|T)D^{\text{virt}}_{\nu\sigma}(\tau_j)\right) \tag{2.56}$$

with the pseudodensity matrices

$$D^{\text{occ}}_{\mu\lambda}(\tau_j) = \sum_{i}^{\text{occ}} C_{\mu i}C_{\lambda i}e^{-|(\epsilon_i-\epsilon_F)\tau_j|} \tag{2.57}$$

$$D^{\text{virt}}_{\nu\sigma}(\tau_j) = \sum_{a}^{\text{virt}} C_{\nu a}C_{\sigma a}e^{-|(\epsilon_a-\epsilon_F)\tau_j|} \tag{2.58}$$

The 3-center ERIs $(\lambda\sigma|R)$ are integrals over the RI metric V_{RI}.

The matrix $\mathbf{Q}(\tau_j)$ can then be obtained from $\mathbf{P}(\tau_j)$ by

$$\mathbf{Q}(\tau_j) = \mathbf{K}^{\mathrm{T}}\mathbf{P}(\tau_j)\mathbf{K} \tag{2.59}$$

with the matrix $\mathbf{K}$ defined by

$$\mathbf{K} = \mathbf{S}^{-1}\mathbf{L} \tag{2.60}$$

The Cholesky factor $\mathbf{L}$ is defined in Eq. (2.27) and $\mathbf{S}$ is the 2-center ERI over V_{RI}.

The expression Eq. (2.56) is evaluated for each time grid point j in terms of the following tensor contractions

Contraction	scaling: cost/memory	
$M^{\mathrm{occ}}_{\mu\sigma R}(\tau_j) = \sum_{\lambda} (\lambda\sigma\|R) D^{\mathrm{occ}}_{\mu\lambda}(\tau_j)$	$O(N^2)/O(N^2)$	(2.61)
$M^{\mathrm{virt}}_{\mu\sigma T}(\tau_j) = \sum_{\nu} (\mu\nu\|T) D^{\mathrm{virt}}_{\nu\sigma}(\tau_j)$	$O(N^2)/O(N^2)$	(2.62)
$P_{RT}(\tau_j) = \sum_{\mu\sigma} M^{\mathrm{occ}}_{\mu\sigma R}(\tau_j) M^{\mathrm{virt}}_{\mu\sigma T}(\tau_j)$	$O(N^2)/O(N^2)$	(2.63)
$Q_{PQ}(\tau_j) = \sum_{R} K_{RP} \sum_{T} K_{TQ} P_{RT}(\tau)$	$O(N^3)/O(N^2)$	(2.64)

For the scaling relations we did not assume decaying density matrices. If density matrices become sparse for large systems, formal scaling of the first 3 contractions is reduced to $O(N)$ (cost and memory). As for the RI-HFX method, the full storage of the large M tensors can be avoided by partitioning indices μ and σ into n_{mem} index ranges and by executing the corresponding n^2_{mem} tensor contraction batches consecutively, leading to a memory reduction by a factor of n_{mem}.

2.6.3 Low-scaling GW

All theories described so far relate to the ground state only and electronically excited states are not accessible. Green function techniques allow to go beyond the ground state description such that ionization energy, electron affinity and the fundamental band gap can be obtained. The G_0W_0 method approximates the quasiparticle wavefunctions by the MOs from Kohn-Sham DFT (including Hartree-Fock contribution / hybrid functionals) [47].

The eigenvalue problem of generalized Kohn-Sham (GKS) DFT (equivalent to Eq. (2.14) with the inclusion of exact exchange) can be written as

$$h^0\psi_n(\mathbf{r}) + \int d\mathbf{r}' v^{\text{xc}}(\mathbf{r},\mathbf{r}')\psi_n(\mathbf{r}') = \epsilon_n\psi_n(\mathbf{r}), \tag{2.65}$$

with h^0 containing the external and the Hartree potential as well as the kinetic energy. In Kohn-Sham DFT, the exchange-correlation potential $v^{\text{xc}}(\mathbf{r},\mathbf{r}')$ is a local potential $v^{\text{xc}}(\mathbf{r},\mathbf{r}') = \delta(\mathbf{r},\mathbf{r}')v^{\text{xc}}_{\text{KS}}(\mathbf{r})$. In Hartree-Fock theory, v^{xc} is equal to the exact exchange

$$\Sigma^{\text{x}}(\mathbf{r},\mathbf{r}') = -\sum_i^{\text{occ}} \frac{\psi_i(\mathbf{r})\psi_i(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} \tag{2.66}$$

After solving the GKS equations self-consistently, the G_0W_0 quasiparticle energies are obtained in effectively $O(N^2)$ computational cost according to

$$\epsilon_n^{G_0W_0} = \epsilon_n + \Sigma_n^{\text{x}} + \text{Re}\Sigma_n^{\text{c}}(\epsilon_n^{G_0W_0}) - v_n^{\text{xc}} \tag{2.67}$$

The low-scaling variant of GW has been originally developed by J. Wilhelm [3] based on the low-scaling RPA algorithm. The correlation self-energy Σ_n^{c} is calculated in imaginary time by

$$\Sigma_n^{\text{c}}(i\tau) = -\sum_{\nu P}\sum_{\mu} G_{\mu\nu}(i\tau)(n|\mu P)\sum_Q \tilde{W}^{\text{c}}_{PQ}(i\tau)(Q\nu|n) \tag{2.68}$$

where the 3-center ERIs $(n|\mu P)$ are integrals over the RI metric V_{RI} and the first index is contracted to the selected MOs n corresponding to the desired range of quasiparticle energies $\epsilon_n^{G_0W_0}$. The Green's function $\mathbf{G}(i\tau)$ can be written in terms of the pseudodensity matrices from low-scaling RPA

$$\mathbf{G}(i\tau) = \begin{cases} \mathbf{D}^{\text{occ}}(\tau), & \text{if } \tau < 0, \\ -\mathbf{D}^{\text{virt}}(\tau), & \text{if } \tau > 0 \end{cases} \tag{2.69}$$

and the scaled screened Coulomb interaction is defined as

$$\tilde{\mathbf{W}}^{\text{c}}(i\tau) = \mathbf{S}^{-1}\mathbf{W}^{\text{c}}(i\tau)\mathbf{S}^{-1} \tag{2.70}$$

The quantity $\mathbf{W}^{\text{c}}(i\tau)$ is the cosine transform from $\mathbf{W}^{\text{c}}(i\omega)$ defined by

$$\mathbf{W}^c(i\omega) = \mathbf{L}(\epsilon^{-1}(i\omega) - \mathbf{1})\mathbf{L}^T, \tag{2.71}$$

$$\epsilon(i\omega) = \mathbf{1} - \mathbf{L}^T\chi^0(i\omega)\mathbf{L}, \tag{2.72}$$

$$\chi^0(i\tau) = \mathbf{S}^{-1}\tilde{\chi}^0(i\tau)\mathbf{S}^{-1} \tag{2.73}$$

where $\tilde{\chi}^0(i\tau)$ is identical to $\mathbf{P}(\tau)$ in Eq. (2.56).

The static exchange self-energy is obtained by

$$\Sigma_n^{\mathrm{x}} = -\sum_{\nu P}\sum_{\mu} D_{\mu\nu}(n\mu P)\sum_{Q}\tilde{V}_{PQ}(Q\nu n) \tag{2.74}$$

with $D_{\mu\nu}$ the density matrix and $\tilde{\mathbf{V}} = \mathbf{S}^{-1}\mathbf{V}\mathbf{S}^{-1}$.

The scaling of Eq. (2.74) and Eq. (2.68) is $O(N^2)$ (not assuming sparse density matrices) and all matrix-matrix multiplications scale as $O(N^3)$, however with a much smaller prefactor such that the effective scaling is $O(N^2)$ even for large systems [3].

Chapter 3

Analytical MME Method for Periodic ERIs

The Gaussian and Plane Wave method allows the computation of matrix elements for periodic systems with the use of a Gaussian basis. The GPW method is the core ingredient of CP2K as it allows accurate and efficient electronic structure calculations of extended systems. It was originally developed for the calculation of the Kohn-Sham matrix elements and found later reuse for the periodic electron repulsion integrals (ERIs) appearing in the electron correlation methods RI-MP2 & RI-RPA.

The GPW method relies on a numerical representation of the density or basis functions in terms of plane waves with a finite cutoff. It therefore requires that the basis functions and the density are smoothly varying in real space. Only the valence electrons can thus explicitly be treated within the GPW scheme and the interaction with the ionic core is included via pseudopotentials. An extension of the GPW method Gaussian and *augmented*-plane-wave (GAPW) [48] was developed to enable all-electron calculations within the DFT formalism. No such equivalent exists yet for wavefunction correlation methods.

An alternative analytical method for 2- and 3-center periodic ERIs is proposed here, the Minimax-Ewald (MME) method based on two techniques:

- The Fourier-transformed Coulomb potential $1/\mathbf{k}^2$ is approximated by a sum of Gaussians using the minimax approximation of $1/x$ by exponential sums [49]

- Inspired by Ewald summation [50], the ERIs are expressed as a sum over periodic image cells which converges rapidly for both narrow and diffuse basis functions.

3.1 Periodic ERIs in the Reciprocal Lattice Representation

Periodic 2-center and 3-center ERIs $(a|b)$ and $(a|bc)$ are integrals of the form of Eq. (2.35):

$$(a|b) = \int_{\Omega} d\mathbf{r}_1 \int_{\mathbb{R}^3} d\mathbf{r}_2 \phi_a^{\mathrm{P}}(\mathbf{r}_1) V(\mathbf{r}_1 - \mathbf{r}_2) \phi_b^{\mathrm{P}}(\mathbf{r}_2) \tag{3.1}$$

$$(ab|c) = \int_{\Omega} d\mathbf{r}_1 \int_{\mathbb{R}^3} d\mathbf{r}_2 \phi_a^{\mathrm{P}}(\mathbf{r}_1) \phi_b^{\mathrm{P}}(\mathbf{r}_1) V(\mathbf{r}_1 - \mathbf{r}_2) \phi_c^{\mathrm{P}}(\mathbf{r}_2) \tag{3.2}$$

where the potential is assumed to be of Coulomb type $V(|\mathbf{r}_1 - \mathbf{r}_2|) = 1/|\mathbf{r}_1 - \mathbf{r}_2|$.

The domain Ω corresponds to the simulation cell with volume V. The simulation cell is spanned by 3 vectors $\mathbf{a_1}, \mathbf{a_2}, \mathbf{a_3}$. Lattice vectors are defined by $\mathbf{R} = \mathbf{h} \cdot \mathbf{s}$ with $\mathbf{h} = [\mathbf{a_1}, \mathbf{a_2}, \mathbf{a_3}]$ and $\mathbf{s}$ a vector of integers. For periodic boundary conditions the basis functions are periodically repeated over all image cells

$$\phi_a^{\mathrm{P}}(\mathbf{r}) = \sum_{\mathbf{R}} \phi_a(\mathbf{r} - \mathbf{R}) \tag{3.3}$$

The periodic basis functions can be expanded into a Fourier series

$$\phi_a^{\mathrm{P}}(\mathbf{r}) = \frac{1}{V} \sum_{\mathbf{G}} \widehat{\phi}_a(\mathbf{G}) e^{i\mathbf{G}\cdot\mathbf{r}} \tag{3.4}$$

with the Fourier coefficients

$$\widehat{\phi}_a(\mathbf{G}) = \int_{\Omega} d\mathbf{r} \phi_a^{\mathrm{P}}(\mathbf{r}) e^{-i\mathbf{G}\cdot\mathbf{r}} = \int_{\mathbb{R}^3} d\mathbf{r} \phi_a(\mathbf{r}) e^{-i\mathbf{G}\cdot\mathbf{r}} \tag{3.5}$$

Reciprocal lattice vectors are defined as $\mathbf{G} = 2\pi(\mathbf{h}^t)^{-1}\mathbf{g}$ with $\mathbf{g}$ an integer vector.

The basis functions are the product of a Gaussian radial part with a spherical harmonic and can be created by transformation of a set of simpler basis functions such as Cartesian Gaussians $C_{\mathrm{l},\alpha}(\mathbf{r})$ or Hermite Gaussians $H_{\mathrm{l},\alpha}(\mathbf{r})$:

$$H_{\mathrm{l},\alpha}(\mathbf{r}) = \prod_{k=1}^{3} H_{l_k,\alpha}(r_k) = \prod_{k=1}^{3} (-1)^{l_k} \left(\frac{\partial}{\partial r_k}\right)^{l_k} \exp(-\alpha r_k^2) \tag{3.6}$$

$$C_{\mathrm{l},\alpha}(\mathbf{r}) = \prod_{k=1}^{3} C_{l_k,\alpha}(r_k) = \prod_{k=1}^{3} r_k^{l_k} \exp(-\alpha r_k^2) \tag{3.7}$$

Following the work of Reine et al. [51] we choose to integrate over Hermite Gaussians and we express our basis functions as $\phi_a(\mathbf{r}) = H_{\mathbf{l},\alpha}(\mathbf{r}-\mathbf{A}), \phi_b(\mathbf{r}) = H_{\mathbf{m},\beta}(\mathbf{r}-\mathbf{B}), \phi_c(\mathbf{r}) = H_{\mathbf{k},\gamma}(\mathbf{r}-\mathbf{C})$ because, as will become apparent shortly, this choice leads to a simpler analytical form of the lattice sum expression of the ERIs. Moreover Hermite Gaussians are a convenient choice for ERI derivatives since differentiation by nuclear coordinates simply increments the l quantum number according to

$$\frac{\partial}{\partial A} H_{l,\alpha}(r-A) = H_{l+1,\alpha}(r-A) \tag{3.8}$$

Thus ERI derivatives of arbitrary order are available in the same scheme and there is no need for a specialized implementation of derivatives. It is also shown in [51] that Hermite Gaussians transform in exactly the same way to spherical-harmonic Gaussians as Cartesian Gaussians after rescaling them as $H_{l,\alpha}(r) \to 1/(2\alpha)^l H_{l,\alpha}(r)$.

The Coulomb potential is expressed in terms of its Fourier transform

$$\frac{1}{|\mathbf{r}|} = \frac{1}{2\pi^2} \int_{\mathbb{R}^3} d\mathbf{k} \frac{1}{\mathbf{k}^2} e^{i\mathbf{k}\cdot\mathbf{r}} \tag{3.9}$$

By inserting the Fourier transformed Coulomb potential Eq. (3.9) and the Fourier series Eq. (3.4) into Eq. (3.1) & Eq. (3.2) we obtain a discrete form of the ERIs as a reciprocal lattice sum:

$$(a|b) = \frac{4\pi}{V} \sum_{\mathbf{G}\neq 0} \frac{1}{|\mathbf{G}|^2} \widehat{\phi}_a(\mathbf{G}) \widehat{\phi}_b(-\mathbf{G}) \tag{3.10}$$

$$(ab|c) = \frac{4\pi}{V} \sum_{\mathbf{G}\neq 0} \frac{1}{|\mathbf{G}|^2} \widehat{\phi}_{ab}(\mathbf{G}) \widehat{\phi}_c(-\mathbf{G}) \tag{3.11}$$

with $\phi^{\mathrm{P}}_{ab}(\mathbf{r}) = \phi^{\mathrm{P}}_a(\mathbf{r})\phi^{\mathrm{P}}_b(\mathbf{r})$. We excluded the divergent $\mathbf{G} = 0$ component which is equivalent to subtracting the average from the functions ϕ_a, ϕ_b, ϕ_c so that their Fourier transforms vanish at $\mathbf{G} = 0$. Whether or not this modification affects the physical results (and needs to be corrected) depends on the method in which the ERIs are used [37, 52].

3.2 Derivation of the MME Method

2-center ERIs

The Fourier coefficient of a basis function $\widehat{\phi}_a$ can be expressed in terms of the Fourier coefficient of a Hermite Gaussian according to $\widehat{\phi}_a(\mathbf{G}) = e^{-i\mathbf{G}\cdot\mathbf{A}}\widehat{H}_{\mathbf{l},\alpha}(\mathbf{G})$. The Fourier coefficient of a Hermite Gaussian maps to a Cartesian Gaussian (and vice versa):

$$\begin{aligned}\widehat{H}_{\mathbf{l},\alpha}(\mathbf{G}) &= (-i)^l\left(\frac{\pi}{\alpha}\right)^{3/2} C_{\mathbf{l},(4\alpha)^{-1}}(\mathbf{G})\\ \widehat{C}_{\mathbf{l},\alpha}(\mathbf{G}) &= (-i)^l\left(\frac{\pi}{\alpha}\right)^{3/2} H_{\mathbf{l},(4\alpha)^{-1}}(\mathbf{G})\end{aligned} \tag{3.12}$$

For the derivation we used the Fourier transform of a Gaussian, the recurrence relation $H_{l,\alpha}(r) = -\frac{d}{dr}H_{l-1,\alpha}(r)$ together with the Fourier transform of the derivative $\widehat{f'}(k) = ik\widehat{f}(k)$. Using the product rule $C_{l,\alpha}(\mathbf{r})C_{m,\beta}(\mathbf{r}) = C_{l+m,\alpha+\beta}(\mathbf{r})$ Eq. (3.10) now reads

$$(a|b) = \frac{4\pi^4}{V}(-1)^l i^{l+m}\left(\frac{1}{\alpha\beta}\right)^{3/2}\sum_{\mathbf{G}\neq 0}\frac{1}{|\mathbf{G}|^2}\exp(-i\mathbf{G}\cdot(\mathbf{A}-\mathbf{B}))C_{\mathbf{l+m},(4\alpha)^{-1}+(4\beta)^{-1}}(\mathbf{G}) \tag{3.13}$$

where we introduced the short-hand notation $l = l_1 + l_2 + l_3$. The choice of Hermite Gaussian basis is crucial for the simple form of Eq. (3.13). If a Cartesian basis was used instead, the summand would contain a product of Hermite Gaussians which could not be reduced as elegantly to a single Gaussian.

The potential $1/|\mathbf{G}^2|$ can be eliminated by expanding it into a linear combination of Gaussians. This is mathematically justified by the minimax approximation of $1/x$ in terms of exponential functions which is well-studied [49]:

$$\frac{1}{x} \approx \sum_{i=1}^{n}\omega_i\exp(-\alpha_i x) \qquad \text{for } x\in[1,R_\mathrm{c}] \tag{3.14}$$

For a given number of points n and a given cutoff R_c for x the minimax approximation minimizes the maximum error $E_n(R_\mathrm{c})$ of Eq. (3.14) for $x \in [1, R_c]$. A larger cutoff R_c implies a larger minimax error. For each number of points n there is a critical value of R_c, so that the minimax error applies not only within $[1, R_c]$, but for the full range $[1,\infty)$. The minimax error decays exponentially with the square root of the number of points n and already 20 points are sufficient to obtain a minimax error $E_{20}(\infty) < 10^{-7}$. The actual error oscillates evenly between $-E_n(R_c)$ and $E_n(R_c)$, a feature favorable for error cancellation.

Approximation Eq. (3.14) can be applied to the Coulomb potential $1/|\mathbf{G}^2|$ by fixing the minimum G_{min} corresponding to the smallest reciprocal lattice vector $|\mathbf{G}| > 0$ for a given simulation cell. By the substitution $x = G^2/G^2_{\text{min}}$ we obtain

$$\frac{1}{|\mathbf{G}|^2} \approx \sum_{i=1}^{n} \tilde{\omega}_i \exp(-\tilde{\alpha}_i |\mathbf{G}|^2) \qquad \text{for } |\mathbf{G}| \in [G_{\text{min}}, G_{\text{c}}], \tag{3.15}$$

with the rescaled minimax coefficients $\tilde{\omega}_i = \omega_i/G^2_{\text{min}}$ and $\tilde{\alpha}_i = \alpha_i/G^2_{\text{min}}$. For convenience we omit the tilde from here on so that ω_i and α_i denote the rescaled coefficients. The error of this approximation relates to the minimax error as $\tilde{E}_n(|\mathbf{G}|) = 1/G^2_{\text{min}} E_n([G_{\text{max}}/G_{\text{min}}]^2)$, implying that larger systems require a larger cutoff R_c and a larger number of minimax points. However since the minimax error stays constant after the critical value of R_c has been reached, the error scales asymptotically with $L^2 \exp(-c\sqrt{n})$ (L being the cell length in one dimension). Consequently the number of minimax points should be scaled with the edge length L of the cell as $n \sim \log(L)^2$.

By inserting the minimax approximation Eq. (3.15) into Eq. (3.13) we obtain the working expression

$$(a|b) \approx \frac{4\pi^4}{V} (-1)^l i^{l+m} \left(\frac{1}{\alpha\beta}\right)^{3/2} \sum_{i=1}^{n} \omega_i \sum_{0<|\mathbf{G}|<G_c} \exp\left(-i\mathbf{G}\cdot(\mathbf{A}-\mathbf{B})\right) C_{\mathbf{l}+\mathbf{m},\tilde{\alpha}_i}(\mathbf{G}) \tag{3.16}$$

with the compound exponent $\tilde{\alpha}_i = \alpha_i + (4\alpha)^{-1} + (4\beta)^{-1}$. Using analytical properties of Gaussians and exponential functions this equation can be evaluated efficiently as long as the reciprocal lattice sum converges rapidly. This is the case only if one of the exponents is small enough. Since Eq. (3.16) has the form of a Fourier series it can be converted to an equivalent sum over direct lattice vectors by the means of the Poisson summation formula for a periodic function $f(\mathbf{R})$

$$\sum_{\mathbf{R}} f(\mathbf{R}) = \frac{(2\pi)^3}{V} \sum_{\mathbf{G}} \widehat{f}(\mathbf{G}) \tag{3.17}$$

In order to convert Eq. (3.16) to a direct lattice sum the term $\mathbf{G} = \mathbf{0}$ is formally included to complete the Fourier series. The finite sum (truncated by G_{c}) is extended over all reciprocal lattice vectors which is justified by the property that $1/|\mathbf{G}|^2$ is an upper bound for the minimax approximation for $|\mathbf{G}| > G_{\text{c}}$ [49]. Then Eq. (3.16) can be

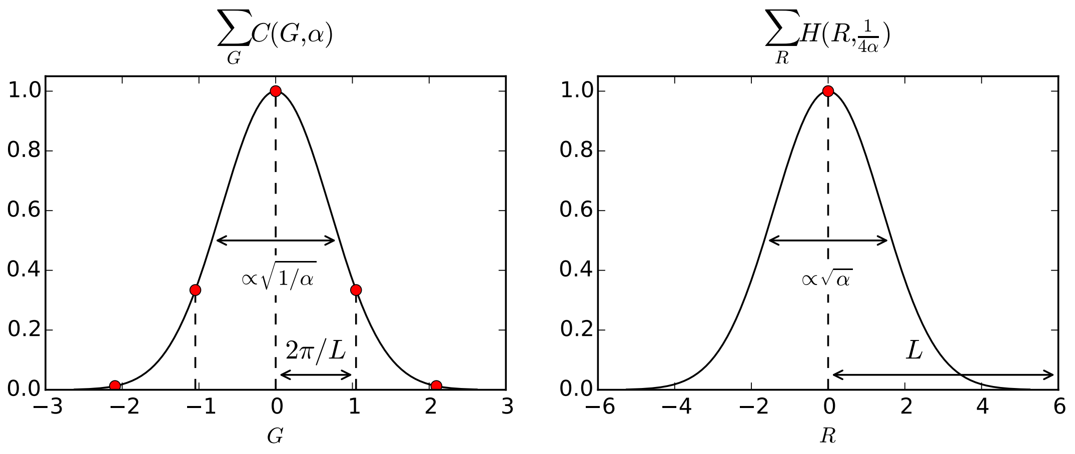


Figure 3.1: Improving convergence by converting the reciprocal lattice sum over a Gaussian (left panel) to a direct lattice sum over a Gaussian of inverse width (right panel).

converted to a direct lattice sum

$$(a|b) \approx 4\pi^4(-1)^m \left(\frac{1}{\alpha\beta}\right)^{3/2} \sum_{i=1}^{n} \omega_i \left[\left(\frac{1}{4\pi\tilde{\alpha}_i}\right)^{3/2} \sum_{\mathbf{R}} H_{\mathbf{l}+\mathbf{m},(4\tilde{\alpha}_i)^{-1}}(\mathbf{R}+\mathbf{A}-\mathbf{B}) - \frac{1}{V}\delta_{l+m,0}\right] \tag{3.18}$$

where the term $\frac{1}{V}\delta_{l+m,0}$ is the $\mathbf{G} = 0$ component implicitly included in the lattice sum which needs to be subtracted. We find that Eq. (3.18) typically converges faster since even diffuse basis functions extend over not too many periodic image cells, as illustrated in Fig. 3.1. The analytical conversion to a direct lattice sum is the core feature to enable efficient all-electron calculations since expression Eq. (3.18) converges especially fast for narrow basis functions with large exponents.

3-center ERIs

The Fourier coefficient of the product $\phi_{ab}^{\mathrm{P}}(\mathbf{r}) = \phi_a^{\mathrm{P}}(\mathbf{r})\phi_b^{\mathrm{P}}(\mathbf{r})$ can be calculated according to the second convolution theorem

$$\widehat{\phi}_{ab}(\mathbf{G}) = \frac{1}{V}\sum_{\mathbf{G}'} \widehat{\phi}_a(\mathbf{G}')\widehat{\phi}_b(\mathbf{G}-\mathbf{G}') \tag{3.19}$$

The discrete convolution coupling two sums over $\mathbf{G}$ and $\mathbf{G}'$ is rather inconvenient, especially when it comes to deriving a direct lattice evaluation analogous to Eq. (3.18). A more promising strategy is to expand the double-periodic function $\phi_{ab}^{\mathrm{P}}(\mathbf{r})$ into a sum over single-periodic functions $\phi^{\mathrm{P}}_{ab,\mathbf{R}}(\mathbf{r})$ according to

$$\phi^{\mathrm{P}}_{ab}(\mathbf{r}) = \sum_{\mathbf{R}'}\sum_{\mathbf{R}} \phi_a(\mathbf{r}-\mathbf{R})\phi_b(\mathbf{r}-\mathbf{R}') = \sum_{\mathbf{R}} \phi^{\mathrm{P}}_{ab,\mathbf{R}}(\mathbf{r}) \tag{3.20}$$

where the single-periodic product $\phi^{\mathrm{P}}_{ab,\mathbf{R}}$ is defined as the periodic repetition Eq. (2.36) of the product function $\phi_{ab,\mathbf{R}}(\mathbf{r}) = \phi_a(\mathbf{r}-\mathbf{R})\phi_b(\mathbf{r})$.

Inserting this expansion of $\phi^{\mathrm{P}}_{ab}(\mathbf{r})$ into Eq. (3.2) yields $(ab|c) = \sum_{\mathbf{R}}(ab,\mathbf{R}|c)$. The Fourier transform of the product function $\phi_{ab,\mathbf{R}}$ would yield a convolution that is best avoided by first expanding $\phi_{ab,\mathbf{R}}$ into a linear combination of single Hermite Gaussians by the ansatz

$$\phi_{ab,\mathbf{R}}(\mathbf{r}) = \prod_{k=1}^{3} H_{l_k,\alpha}(r_k - A_k - R_k)H_{m_k,\beta}(r_k - B_k) = \prod_{k=1}^{3}\sum_{t=0}^{l_k+m_k} F_t^{l_k,m_k} H_{t,\alpha+\beta}(r_k - P_k) \tag{3.21}$$

with the Gaussian product center $\mathbf{P} = (\alpha(\mathbf{A}+\mathbf{R})+\beta\mathbf{B})/(\alpha+\beta)$. The coefficients $F_t^{l_k,m_k}$ are generated for each combination of basis sets by simple recurrence relations derived in [53].

The Fourier transform of $\phi^{\mathrm{P}}_{ab,\mathbf{R}}$ can now easily be evaluated according to

$$\widehat{\phi}_{PQ,\mathbf{R}}(\mathbf{G}) = \prod_{k=1}^{3}\sum_{t=0}^{l_k+m_k} C_t^{l_k,m_k} e^{-iG_kP_k}\widehat{H}_{t,\alpha+\beta}(G_k) \tag{3.22}$$

$$= \left(\frac{\pi}{\alpha+\beta}\right)^{3/2} e^{-i\mathbf{G}\cdot\mathbf{P}} \prod_{k=1}^{3}\sum_{t=0}^{l_k+m_k} (-i)^t F_t^{l_k,m_k} C_{t,(4(\alpha+\beta))^{-1}}(G_k)\,, \tag{3.23}$$

The minimax-approximated 3-center ERI in reciprocal lattice representation reads as

$$(PQ|R) \approx \frac{4\pi^4}{V}((\alpha+\beta)\gamma)^{-3/2}\sum_{i=1}^{n}\omega_i \times \tag{3.24}$$

$$\sum_{\mathbf{R}}\sum_{\mathbf{G}\neq 0} e^{i\mathbf{G}\cdot(\mathbf{C}-\mathbf{P})}\prod_{k=1}^{3}\sum_{t=0}^{l_k+m_k}(-1)^t i^{t+o_k} F_t^{l_k,m_k} C_{t+o_k,\tilde{\alpha}_i}(G_k) \tag{3.25}$$

with the compound exponent

$$\tilde{\alpha}_i = \frac{\alpha+\beta+\gamma}{4(\alpha+\beta)\gamma} + \alpha_i \tag{3.26}$$

Note that the coefficients $F_t^{l_k,m_k}$ and $\mathbf{P}$ are functions of $\mathbf{R}$ which is hidden by the abbreviated notation. Since the summand consists of a single Cartesian Gaussian, the Poisson summation formula Eq. (3.17) can be applied in the same way as for the 2-center case:

$$\begin{aligned}(ab|c) \approx & 4\pi^4(-1)^o((\alpha+\beta)\gamma)^{-3/2}\sum_{i=1}^{n}\omega_i \\ & \left([4\pi\tilde{\alpha}_i]^{-3/2}\sum_{\mathbf{R},\mathbf{R}'}\prod_{k=1}^{3}\sum_{t=0}^{l_k+m_k} F_t^{l_k,m_k} H_{t+o_k,(4\tilde{\alpha}_i)^{-1}}\left(C_k-P_k-R'_k\right)\right. \\ & \left. -\ \delta_{\mathbf{o},\mathbf{0}}\frac{1}{V}\prod_{k=1}^{3}C_0^{l_k,m_k}\right) \end{aligned} \tag{3.27}$$

3.3 Interaction Potential Types

So far we discussed ERIs for the Coulomb potential only. A generalization of the MME method to other potentials is possible as long as the potential (or its Fourier transform) can be approximated by a linear combination of Cartesian Gaussians. Here we discuss the generalization to Yukawa potential and shortrange/longrange Coulomb potential.

The Yukawa potential $V_\omega(\mathbf{r}) = e^{-\omega r}/|\mathbf{r}|$ has the Fourier transform $\mathcal{F}(V_\omega)(\mathbf{k}) = 4\pi/(\mathbf{k}^2+\omega^2)$. This potential can be mapped to the minimax approximation Eq. (3.14) by the substitution $x = (G^2+\omega^2)/(G_{\text{min}}^2+\omega^2)$ such that the minimax approximation has the exact same form as Eq. (3.15) with the coefficients $\tilde{\alpha}_i = \alpha_i/(G_{\text{min}}^2+\omega^2)$ and $\tilde{\omega}_i = \omega_i\exp(-\tilde{\alpha}_i\omega^2)/(G_{\text{min}}^2+\omega^2)$.

The Coulomb potential can be split into a short-range and a long-range part according to

$$\frac{1}{r} = \frac{\operatorname{erfc}(\omega r)}{r} + \frac{\operatorname{erf}(\omega r)}{r} \tag{3.28}$$

The Fourier transform of the long-range part is

$$\mathcal{F}\left(\frac{\operatorname{erf}(\omega r)}{r}\right)(\mathbf{k}) = \frac{4\pi}{\mathbf{k}^2}\exp(-\omega\mathbf{k}^2) \tag{3.29}$$

We insert the minimax approximation for $1/\mathbf{k}^2$ according to Eq. (3.15) to obtain

$$\mathcal{F}\left(\frac{\operatorname{erf}(\omega r)}{r}\right)(\mathbf{G}) \approx 4\pi\sum_{i=1}^{n}\tilde{\omega}_i\exp(-(\tilde{\alpha}_i+\omega)\mathbf{G}^2) \qquad \text{for } |\mathbf{G}| \in [G_{\text{min}}, G_{\text{c}}] \tag{3.30}$$

so that the range parameter ω leads to a shift in the exponent $\tilde{\alpha}_i$. The short-range part can be calculated indirectly by subtracting long-range ERIs from Coulomb ERIs.

3.4 Efficient Numerical Evaluation

Evaluation of Hermite Gaussians

The direct lattice sums Eq. (3.18) & Eq. (3.27) are the starting point of the implementation since these sums typically converge much faster than the reciprocal lattice sum. The Hermite Gaussians are evaluated by expanding them into Cartesian Gaussians according to

$$H_{l,\alpha}(x) = \sum_{k=0}^{l} D_{lk} C_{k,\alpha}(x) \tag{3.31}$$

The matrix $\mathbf{D}$ is involutory so that $\mathbf{D}$ is also the back transformation matrix from Hermite to Cartesian Gaussians (as can be seen by Fourier transforming Eq. (3.31) with use of Eq. (3.12)).

The transformation matrix elements D_{lk} can be found recursively starting from the derivative of a Cartesian Gaussian [53]

$$\frac{\partial}{\partial x} C_{k,\alpha}(x) = k C_{k-1,\alpha}(x) - 2\alpha C_{k+1,\alpha}(x) \tag{3.32}$$

to expand $H_{l+1,\alpha}(x)$ as

$$\begin{aligned} H_{l+1,\alpha}(x) &= -\frac{\partial}{\partial x} H_{l,\alpha}(x) = -\sum_{k=0}^{l} D_{lk} \frac{\partial}{\partial x} C_{k,\alpha}(x) = \\ &= \sum_{k=1}^{l} -D_{lk} k C_{k-1,\alpha}(x) + 2\alpha \sum_{k=0}^{l} D_{lk} C_{k+1,\alpha}(x) \end{aligned} \tag{3.33}$$

By comparing with the trivial expansion

$$H_{l+1,\alpha}(x) = \sum_{k=0}^{l+1} D_{l+1,k} C_{k,\alpha}(x) \tag{3.34}$$

the recurrence relation for the matrix elements D_{lk} is obtained

$$D_{l+1,k} = -(k+1) D_{l,k+1} + 2\alpha D_{l,k-1} \tag{3.35}$$

Lattice Summation

The working expressions to be implemented are given by Eq. (3.18) for 2-center ERIs and Eq. (3.27) for 3-center ERIs. For the case of orthorhombic simulation cells the lattice vectors are mutually orthogonal and the sum factorizes into the 3 Cartesian components. More precisely Eq. (3.18) and Eq. (3.27) can be described as a sum $\sum_{\mathbf{R}} f(\mathbf{R})$ with a summand that can be factorized as $f(\mathbf{R}) = f_1(R_1)f_2(R_2)f_3(R_3)$. The Cartesian components of the lattice sum can be separated according to

$$\begin{aligned}\sum_{\mathbf{R}} f(\mathbf{R}) &= \sum_{s_1}\sum_{s_2}\sum_{s_3} f(h\cdot\mathbf{s}) \\ &= \left(\sum_{s_1}(h_{11}s_1)\right)\left(\sum_{s_2}(h_{22}s_2)\right)\left(\sum_{s_3}(h_{33}s_3)\right) \\ &= \prod_{k=1}^{3}\sum_{R_k} f_k(R_k) \qquad (3.36)\end{aligned}$$

where we used that **h** is diagonal for orthorhombic cells. Since the angular momentum quantum numbers are restricted to small numbers, loops come with a significant overhead and a complete loop unrolling up to a certain maximum number is applied. Such an unrolling can be easily implemented and maintained by writing the loops in an external preprocessor language (Fypp [54]). The most expensive part in evaluating the lattice sums is the calculation of powers and exponentials for each lattice grid point. Due to the equidistant spacing of the grid points powers and exponentials can be evaluated recursively based on the value at the previous grid point, effectively replacing many *exp* and *pow* calls by much cheaper multiplication.

The number of grid points in the lattice sum over Gaussians is determined by assigning a static range to each Gaussian according to some truncation threshold. An accurate default is chosen for converging the lattice sums since performance does not strongly depend on the number of points thanks to the recursive evaluation of *exp* and *pow*.

Integral screening methods, relying on a cheap estimate of the magnitude of an integral, are important for performance to avoid the calculation of negligibly small integrals. 3-center ERIs $(ab|c)$ are small if the local basis functions a and b are sufficiently far apart. The minimax approximation allows for an efficient screening method since the cost of calculating an integral is proportional to the number of minimax points and a small number of points (such as 5) already gives a good estimate of the integral magnitude. After a coarse screening based on assigning a static radius to each basis function the screening is refined by calculating the integral with a small number of minimax points. If the result is below a certain threshold, the integral is approximated by this value. If the result is larger than the threshold the integral is recalculated with a larger and more accurate number of points.

Parameter Optimization

The generation of the minimax approximation is numerically problematic and should not be done dynamically at run time. Instead we use the pre-tabulated minimax approximations from [49] covering a sufficiently large number of different ranges for each number of points. The remaining problem is to pick a sufficiently large cutoff G_c translating to the fit range of the minimax approximation. This parameter G_c can not easily be converged by the user since both a too-large value and a too-small value lower the accuracy (in contrast to the GPW method where a large cutoff will always provide an exact reference value). This is due to a combination of two unrelated sources of errors: A too-small value of G_c leads to a truncation of significant contributions in the lattice sum and a too-large value reduces the accuracy of the minimax approximation. To ensure stability of the MME method and to make it parameter-free, a good default for G_c is derived systematically based on error estimates for both sources of errors in dependence of the basis parameters. The error estimates and the cutoff calibration procedure is described in detail in Appendix A.

3.5 Results

3.5.1 RI-MP2 & RI-RPA

The accuracy and performance of the newly developed MME is compared with GPW for RI-RPA calculations of water with the use of GTH pseudopotentials. For pseudopotential-based calculations the MME method allows calculations at an accuracy comparable to the GPW method, see Fig. 3.2. The MME method comes with a small overhead of roughly 20% in computational costs due to 3-center ERIs being more expensive in the MME approach, see Fig. 3.3. Since the ERIs are calculated over Gaussian functions, the transformation with the Cholesky factor L^{-1} Eq. (2.26) has to be performed explicitly after the ERI calculation whereas this transformation is implicit in the GPW scheme. This transformation step scales as $O(N^4)$ and starts to show for 128 water molecules, although adding only 20% overhead relative to the RPA energy calculation.

The GPW method is restricted to smoothly varying functions due to the FFT-based approach. This restriction is lifted in the MME method due to the analytical evaluation of Fourier transforms, enabling periodic all-electron RI-RPA/MP2 calculations. An at-

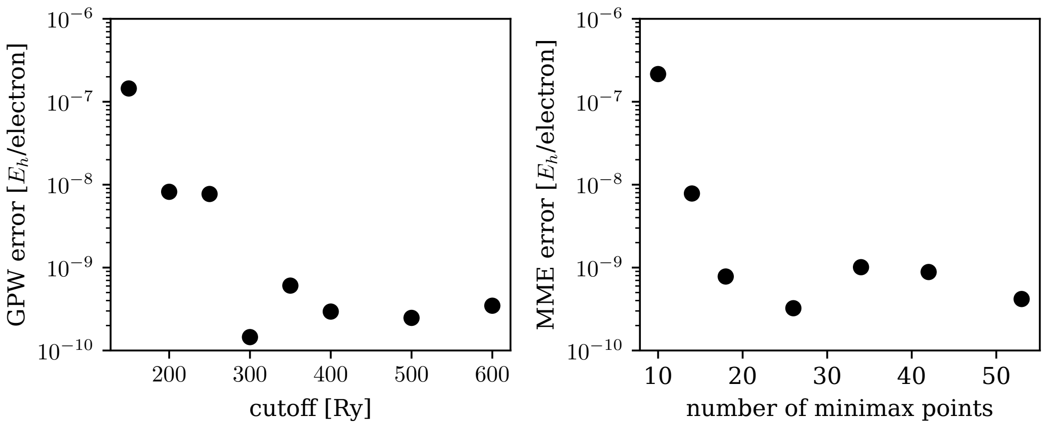


Figure 3.2: Comparison of the numerical accuracy of the MME method with the GPW method for an RPA calculation of a system of 32 water molecules. The results suggest that 20 minimax points are sufficient to converge the error to a value lower than 10^{-9} a.u./electron, comparable to the GPW error using a cutoff of 400 Ry. The reference energy has been obtained by a GPW-based calculation with a planewave cutoff / rel. cutoff of 1000 / 200 Ry.

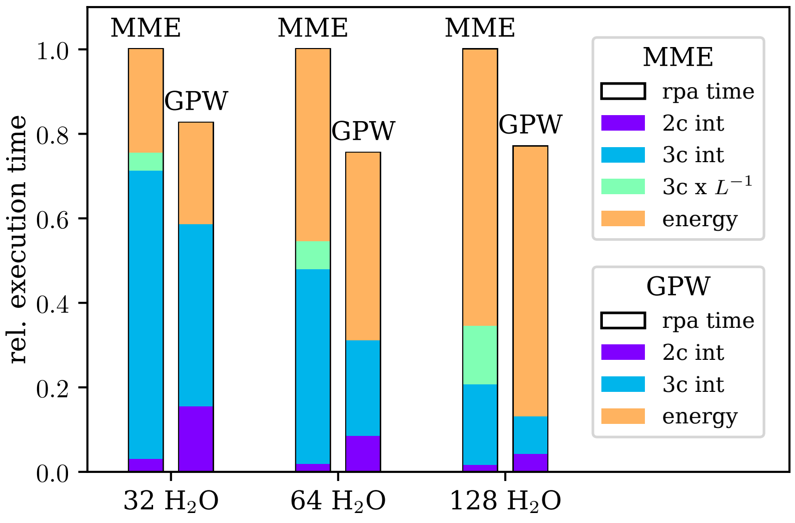


Figure 3.3: benchmarking MME (20 minimax points) vs. GPW method (400 Ry cutoff) for the RI-RPA method and water systems of different sizes between 32 and 128 molecules, using a cc-TZV2P basis. The 2-center ERIs are significantly cheaper due to the simple analytical formulation of the MME method. The same cannot be stated for 3-center integrals where the MME method has a small overhead over the GPW method. The total ERI calculation scales as $O(N^2)$ with system size. An additional integral transformation with L^{-1} is needed in the MME scheme scaling as $O(N^4)$.

tractive feature of the MME method is that the ERI calculation over narrow functions is feasible within the same scheme as for smooth functions and does not add any relative computational overhead. The only parameter is the number of minimax grid points and a sufficiently large cutoff is optimized automatically using the scheme derived in Appendix A.

The smooth convergence for accurate all-electron calculation is demonstrated in Fig. 3.4 and results for water systems of different size and basis set quality are shown in Fig. 3.5. The augmented correlation consistent orbital basis sets [55] and auxiliary basis sets [56] have been obtained from Basis Set Exchange [57].

The restriction to relatively small system sizes is not due to computational costs but due to large memory requirements. This is explained by inefficient ERI storage in the atomic orbital basis which could be overcome by a scheme calculating integrals in batches and immediately transforming each batch to molecular orbitals.

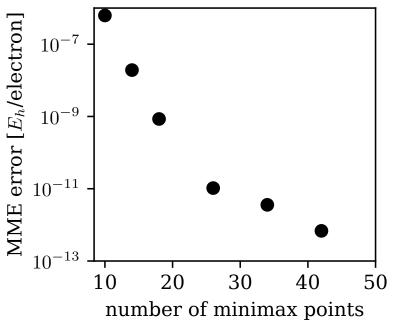


Figure 3.4: Convergence of the all-electron RPA energy (aug-cc-pVTZ basis, 32 water molecules) with the number of minimax points. The reference energy has been obtained with 53 minimax points.

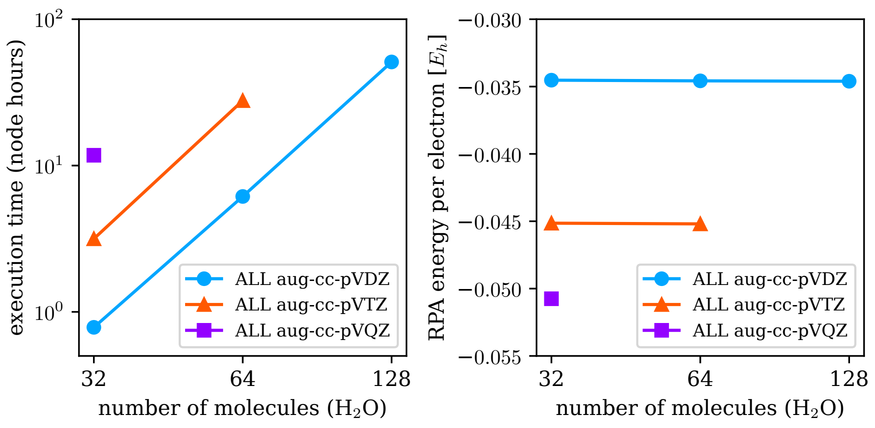


Figure 3.5: Demonstrating the capability of the MME ERI method to perform periodic all-electron RI-RPA calculations for the case of water systems of different sizes and different basis quality.

3.5.2 Image Charge Augmented QM/MM

The MME method can be applied as a fast method to calculate the charge distribution in a metal slab induced by an adsorbate. We briefly introduce the IC-QM/MM method by following the work of D. Golze [58] and demonstrate the use of the MME method to speed up the image charge integrals so that IC-QM/MM calculations can be performed at the cost of regular QM/MM calculations.

The image charge method [58] is a model to include polarization effects within the metal in QM/MM simulations of metal-adsorbate systems. Realistic systems of such interfaces consist of several layers in the metallic substrate and require sufficient lateral dimensions so that large simulation cells are needed. Quantum mechanical (QM) methods such as DFT are often not affordable. Classical molecular mechanics (MM) based on force fields are computationally much less expensive but cannot capture electronic effects.

Alternatively the electronic structure of the adsorbate can be described accurately by using a hybrid scheme combining QM and MM methods in which the adsorbate is treated by DFT and the metal by classical force fields. The interactions between adsorbate and metallic substrate are described at the MM level of theory. In the image charge augmented QM/MM scheme the charge distribution induced within the metallic substrate is modeled by a set of Gaussian charges (image charges) with constant width centered at the metal atoms. The image charges and the electrostatic response of the QM potential are determined self-consistently by imposing the constant-potential condition within the metal. Even though the electronic properties of the metallic substrate are not taken into account explicitly, the augmented QM/MM scheme can reproduce characteristic polarization effects of the adsorbate.

Within the IC-QM/MM approach the induced charge distribution is expressed in terms of the image charge Gaussians $g_a(\mathbf{r}, \mathbf{R}_a) = \exp\left(-\alpha|\mathbf{r} - \mathbf{R}_a|^2\right)$ by

$$\rho_m(\mathbf{r}) = \sum_a c_a g_a(\mathbf{r}, \mathbf{R}_a) \tag{3.37}$$

where $\mathbf{R}_a$ is the position of metal atom a. The Gaussian width is determined by a single parameter α that is set to a constant value. The expansion coefficients c_a are obtained by imposing that the electrostatic potential is constant in the metallic slab, equivalent to the minimization of the electronic energy w.r.t. the coefficients c_a

$$\frac{\partial E_{\mathrm{el}}}{\partial c_a} = \int (V_{\mathrm{h}}(\mathbf{r}) - V_0) g_a(\mathbf{r}) d\mathbf{r} + \sum_b c_b T_{ab} = 0 \tag{3.38}$$

where V_h is the Hartree potential and V_0 is a constant external potential. The matrix T_{ab} consists of ERIs between pairs of image charges

$$T_{ab} = \int\int \frac{g_b(\mathbf{r}', \mathbf{R}_b) g_a(\mathbf{r}, \mathbf{R})}{|\mathbf{r}' - \mathbf{r}|} d\mathbf{r} d\mathbf{r}' \tag{3.39}$$

The calculation of T_{ab} is typically more expensive than the actual QM/MM calculation if the GPW method is used. Fig. 3.7 shows timings of IC-QM/MM using the MME method, in comparison with the GPW method and regular QM/MM calculations without image charge correction. The much faster MME method renders the costs of the image charge correction negligible so that IC-QM/MM can be performed at about the same costs as regular QM/MM.

Both, IC-QM/MM and standard QM/MM calculations, are performed for a large flat tris-terpyridine-derived molecule (TTPB) on a Au(111) surface, see Fig. 3.6. In order to avoid superios interactions of the TTPB molecule on the surface, the metallic substrate is modeled laterally with a $p(12 \times 24)$ repetition of the unit cell and vertically by four metal layers. In order to decouple the periodic image along the vertical axis, a vacuum of 23 Å has been added above the TTPB molecule. We employ an asymmetric slab-structure, i.e. TTPB is only added on one side of the Au(111) slab. To assess the performance of the IC-QM/MM calculations dependent on the system size, the unit cell of the TTPB@Au(111) model has been repeated up to six times in one of the lateral dimensions. The number of SCF step has been set to 40.

The setup for the QM/MM and IC-QM/MM calcuations is the same as for the adsorbate@Au111 systems in Ref [58], except for the plane wave cutoff which is set to 300 Ry. The TTPB molecule is described at the DFT level of theory using the Perdew-Burke-Ernzerhof (PBE) functional [59], whereas the metallic substrate is described at the MM level. The interactions between the molecule and the metal are MM-based. Dispersion and Pauli repulsion are described by a modified Born-Mayer potential. Electrostatic interactions are accounted for by the IC correction. For more details, see Ref. [58].

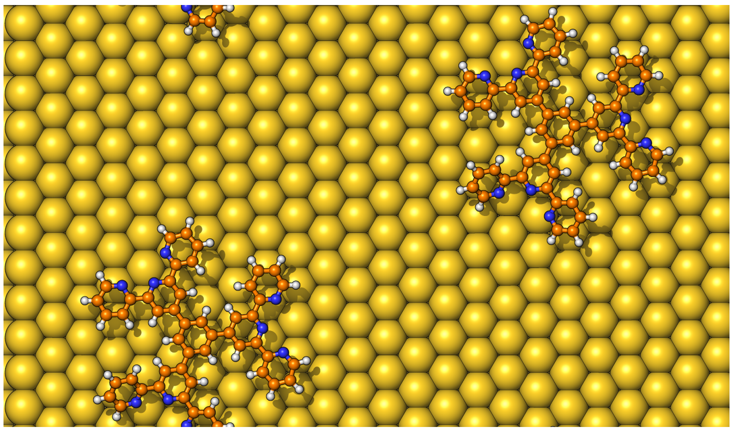


Figure 3.6: Orthorhombic unit cell of TTPB on Au(111). Color code: *orange*: C, *white*: H, *blue*: N, *yellow*: Au.

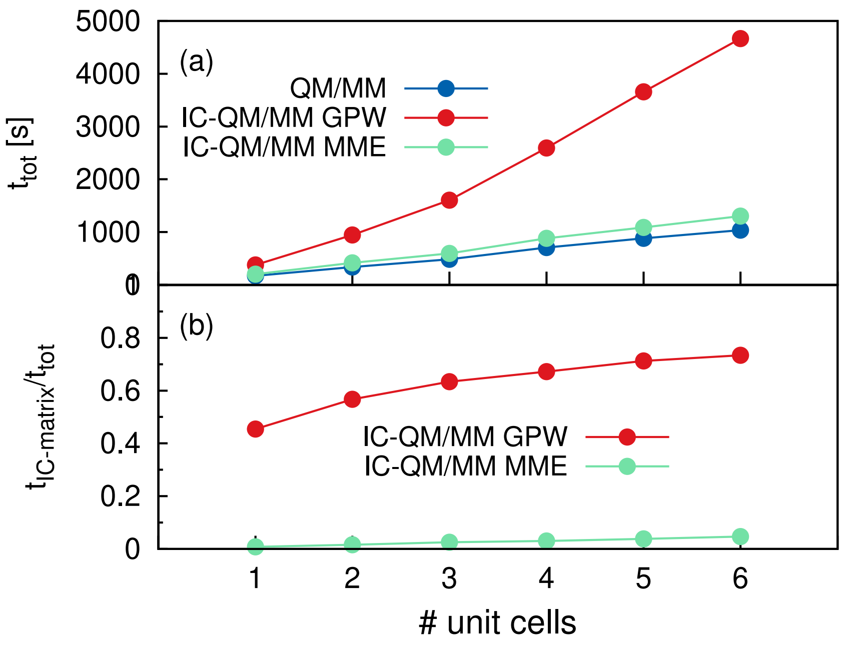


Figure 3.7: QM/MM and IC-QM/MM timings measured on a Cray XC30 system for TTPB@Au(111) repeating the unit cell in one of the lateral dimension. (a) Total execution time t_{tot} and (b) ratio $t_{\text{IC-matrix}}/t_{\text{tot}}$, where $t_{\text{IC-matrix}}$ is the time for calculating the IC-matrix.

3.6 Conclusion and Outlook

The MME method for electron repulsion integrals is a general analytical approach to calculate integrals over Gaussians with a polynomial factor and potentials of Coulomb type under periodic boundary conditions. This method facilitates the development of electronic structure methods that combine Gaussian basis functions with periodic boundary conditions. For local potentials the periodicity can be taken into account explicitly by summing over all image cells and the integrals can be calculated non-periodically by traditional methods such as Obara-Saika. The MME method fills the gap of analytical integration methods for the case that the potential is non-local. The method is generalizable to other types of potentials as was demonstrated for the case of attenuated-Coulomb and Yukawa potentials.

Compared with the numerical GPW method the MME method has the advantage of being applicable to narrow Gaussians that require a very high plane-wave cutoff at the same efficiency as Gaussians with an extended width. By means of this feature the MME method enables efficient all-electron RI-RPA and RI-MP2 calculations of periodic systems. The MME method is particularly efficient for the case of 2-center ERIs and yields significant speedups for methods relying on this type of integrals. This was demonstrated for the image-charge-augmented QM/MM method that extends QM/MM to metal-adsorbate systems where the charge induction in the metal can now be calculated at insignificant costs compared with the QM/MM part of the calculation.

Chapter 4

Sparse Tensor Contraction Framework

4.1 Introduction

In electronic structure methods, most algebraic expressions involve matrices and tensors and the operations performed can be generally classified as tensor contractions. Tensors derived from two-electron integrals have a rank of 4 and the introduction of the RI approximation reduces the maximum rank to 3. Under the assumption of local basis functions the atomic orbital representation of tensors is often sparse so that most elements are small and can be neglected.

Traditionally dense tensor contractions are implemented by a conversion to a matrix representation in order to delegate the calculation to a matrix multiplication library. This is motivated by the availability of optimized numerical libraries such as ScaLAPACK. However due to missing abstractions for multi-dimensional data, conversion operations need to be manually written for each individual contraction. Even mathematically simple operations such as index transpositions are troublesome and require a complete change in the data layout and a redistribution of all data.

For sparse tensor contractions the same approach is even more problematic in practice due to the bookkeeping of indices and block sizes which map a certain data element to its logical tensor index. The format of sparse tensors needs to be adapted to a heterogeneously structured sparsity pattern consisting of small dense blocks. A suitable mapping to a compatible sparse matrix format is less evident than in the dense case.

In order to implement the low-scaling algorithms described in Sec. 2.6 a tensor library must necessarily be optimized for large block-sparse tensors based on distributed memory parallelism. Several tensor libraries have been developed in recent years that provide a simple symbolic language for arbitrary tensor contractions, most notably the *libtensor* project [60], optimized for shared-memory parallelism, and the *Cyclops Tensor Framework* (*CTF*) [61] for large-scale distributed-memory applications. The libtensor project has been recently integrated into *CTF* to enable distributed-memory parallelization [62]. Both libraries have been extended to handle sparse data, *CTF* relying on an element-wise sparse layout [63] for generality but with limited sequential efficiency, and libtensor optimized for block sparsity [64] but not supporting distributed memory parallelism. Finally the *TiledArray* framework [65, 66] is optimized both for block-sparse data and large-scale parallelization using a task-based implementation of the 2d SUMMA algorithm. In contrast to statically distributed algorithms the task-based approach should be capable of hiding communication costs and load imbalance that arise due to data inhomogeneity attributed to block sparsity. They show large-scale applications exploiting block sparsity and low-rank sparsity within blocks [67–69]. Another project for large-scale block-sparse tensor contractions on multi-GPU nodes based on the task-focused PaRSEC runtime was recently reported [70].

Our approach towards a sparse tensor contraction library is based on the DBCSR matrix library [71]. The most distinctive feature of the DBCSR library compared with the libraries mentioned above are sequential performance optimizations specifically targeting multiplication of small blocks, including CPU and GPU backends.

4.2 Tensor Contractions as Matrix Multiplications

It can be easily seen that tensor contractions are mathematically equivalent to matrix-matrix multiplications, matrix-vector and vector-vector (inner or outer) products. A promising approach towards tensor contractions is to express them in terms of these simpler numerical primitives. A tensor contraction API could be implemented as a mapping between the abstract notation and the concrete evaluation using an existing linear algebra library.

For a systematic treatment we introduce a mathematical notation adapted from [61] that expresses the mapping between tensors and matrices systematically. Let $A(i_1, i_2, \ldots, i_{n_I})$ be the element of a tensor of rank n_I with index tuple $I = (i_1, i_2, \ldots i_{n_I})$. The shape is the size in each dimension and is referred to as $S = (I_1, I_2, \ldots, I_{n_I})$. We consider 3 tensors

A, **B**, **C** that appear in a contraction of the form

$$
\begin{aligned}
&C(i_1, \ldots, i_n, \ldots, i_{n_I}) = \\
&\sum_{k_1,\ldots,k_{n_K}} A(i_1, \ldots, i_n, k_1, \ldots, k_{n_K}) B(k_1, \ldots, k_{n_K}, i_{n+1}, \ldots, i_{n_I})
\end{aligned}
\tag{4.1}
$$

where the indices may be permuted arbitrarily and are ordered here just for simplicity of notation. The index K collects all indices that are contracted (summed over), appearing in both **A** and **B**, and the index I collects the indices remaining in the contraction result **C**.

We define the compound index $E = (i_1, \ldots, i_{n_I}, k_1, \ldots, k_{n_K})$ containing all indices appearing in the contraction. Let p_A, p_B and p_C be the functions that map the compound index E to indices of the respective tensor, e.g. $p_A(E) = (i_1, \ldots, i_n, k_1, \ldots, k_{n_K})$. These must be defined in a way that each index in E occurs in exactly two of the three tensor index tuples $p_A(E)$, $p_B(E)$ and $p_C(E)$. Then a general tensor contraction can be expressed in the closed-form notation

$$
C(p_C(E)) = \sum_{p_A(E)\cap p_B(E)} A(p_A(E)) \cdot B(p_B(E)) \tag{4.2}
$$

To clarify the notation, we consider the example

$$
C(i, j, k) = \sum_{l,m} A(l, m, i) B(k, m, j, l) \tag{4.3}
$$

In this case the general notation Eq. (4.2) can be recast by considering

$$
\begin{aligned}
E &= (i, j, k, l, m) && (4.4)\\
p_A(E) &= (l, m, i) && (4.5)\\
p_B(E) &= (k, m, j, l) && (4.6)\\
p_C(E) &= (i, j, k) && (4.7)
\end{aligned}
$$

and by noting that $p_A(E) \cap p_B(E) = \{l, m, i\} \cap \{k, m, j, l\} = \{l, m\}$.

In order to demonstrate the connection between tensor contraction and matrix multiplication, we define a one-to-one mapping between a tensor and its matrix representation: for a tensor $T(I)$ with index I, $M_T(\tilde{I})$ defines its matrix representation with 2-dimensional index $\tilde{I} = (i_1, i_2)$. The relation between tensor index I and matrix index $\tilde{I}$ is established by a one-to-one mapping

$$\tilde{I} = (m_{T1}(I), m_{T2}(I)) \tag{4.8}$$

where

$$m_{Ti} = f \circ \pi_{Ti} \circ s_{Ti} \tag{4.9}$$

is a composition of three functions:

- s_{Ti} selects an arbitrary subset of indices I
- π_{Ti} is an arbitrary index permutation that maps a set of indices to an ordered tuple
- f maps an index tuple to a single compound index (one-to-one), also known as *index folding*

The selection functions s_{T1} and s_{T2} must be defined so that each element of I must occur in either $s_{T1}(I)$ or $s_{T2}(I)$ but not both. It is permissible that s_{Ti} selects no indices or all indices in I. The index folding function f can be arbitrarily chosen as long as it's one-to-one. For simplicity we assume that f is the same function for all tensors $\mathbf{T}$ and for both matrix indices i, however this is not a strict requirement.

For the tensor contraction involving tensors $\mathbf{A}, \mathbf{B}, \mathbf{C}$ as defined in Eq. (4.2) and with contraction indices $I = p_A(E), J = p_B(E), K = p_C(E)$, we define the matrix mappings

$$\begin{aligned} s_{A1}(I) = s_{C1}(K) = I \cap K \\ s_{A2}(I) = s_{B1}(J) = I \cap J \\ s_{B2}(J) = s_{C2}(K) = J \cap K \end{aligned}$$

The corresponding permutation functions $\pi_{A1} = \pi_{C1}, \pi_{A2} = \pi_{B1}, \pi_{B2} = \pi_{C2}$ can be arbitrarily chosen. Then the tensor contraction Eq. (4.2) can be expressed as a matrix multiplication:

$$M_C(m_{C1}(K), m_{C2}(K)) = M_A(m_{A1}(I), m_{A2}(I)) \cdot M_B(m_{B1}(J), m_{B2}(J)) \tag{4.10}$$

For the example of Eq. (4.3) we have

$$M_C(f(i), f(j,k)) = M_A(f(i), f(l,m)) \cdot M_B(f(l,m), f(j,k)) \tag{4.11}$$

or establishing a short-hand notation which we will use from now on

$$M_C(i, jk) = M_A(i, lm) \cdot M_B(lm, jk) \tag{4.12}$$

As mentioned before, the selection function s_{Ti} may select zero indices, then the matrix representation of a tensor $T(I)$ is a row vector $V_T^{\mathrm{r}}(m_{T2}(I)) = M_T(, m_{T2}(I))$ or a column vector $V_T^{\mathrm{c}}(m_{T1}(I)) = M_T(m_{T1}(I),)$. Thus tensor contractions map not only to matrix-matrix multiplications but also to different products involving matrices and/or vectors:

- matrix-vector product

$$V_C^{\mathrm{r}}(m_{C2}(K)) = V_A^{\mathrm{r}}(m_{A2}(I)) \cdot M_B(m_{B1}(J), m_{B2}(J)) \tag{4.13}$$

$$V_C^{\mathrm{c}}(m_{C1}(K)) = M_A(m_{A1}(I), m_{A2}(I)) \cdot V_B^{\mathrm{c}}(m_{B1}(J)) \tag{4.14}$$

- vector outer product

$$M_C(m_{C1}(K), m_{C2}(K)) = V_A^{\mathrm{c}}(m_{A1}(I)) \cdot V_B^{\mathrm{r}}(m_{B2}(J)) \tag{4.15}$$

- vector inner product (S_C denoting a scalar)

$$S_C = V_A^{\mathrm{r}}(m_{A2}(I)) \cdot V_B^{\mathrm{c}}(m_{B1}(J)) \tag{4.16}$$

4.3 DBCSR Sparse Matrix Library

The DBCSR library implements sparse matrix primitives in the Distributed Block Compressed Sparse Row format. This format is derived from the Compressed Sparse Row format which allows to store non-zero elements only. This format is however inefficient for any matrix operation since every single scalar operation requires an indirect addressing step, mapping a certain index (row, column) to an address in the array storing all non-zero elements [72]. Often the sparsity structure consists of small dense blocks. In electronic structure methods the sparse matrix representation of quantities such as the density, the Hamiltonian or Electron Repulsion Integrals (ERIs) are defined in the atomic orbital (AO) basis where multiple basis functions are centered on the same atom. Defining a blocked matrix index in terms of atomic submatrices instead of single elements, indirect addressing steps occur only once for each block, such that its overhead is reduced dramatically and efficient dense compute kernels can be applied to the submatrix operations. The atomic blocks can have arbitrary sizes adapted to the number of basis functions per atom. Small and heterogeneous block sizes are problematic since established linear algebra libraries such as BLAS gain their performance from sufficiently large blocks.

DBCSR matrices are distributed over a two-dimensional grid of N_P MPI processes where the assignment of matrix indices to process coordinates can be chosen arbitrarily. To achieve good load balancing, the distribution should be aware of the different block sizes and should randomize over the structured sparsity pattern.

DBCSR consists of several layers according to the separation of concerns between *data exchange*, *data access*, *index operations* and *computations* [71, 73]:

- *Data exchange*: communication pattern used for transferring data for the multiplication based on Cannon's algorithm [74]

- *Data access*: In order to improve memory locality, the largest dimension in $\{m, n, k\}$ of a local sparse matrix multiplication is recursively divided until sufficiently small matrix dimensions have been obtained.

- *Index operations*: Submatrix multiplications involving blocks of ideally the same size are gathered into a stack. An on-the-fly filtering procedure optimizes away unneeded computations by skipping blocks with a pre-estimated norm falling below a certain filtering threshold.

- *Computations*: stacks are executed on the CPU or offloaded to the GPU which perform the small matrix multiplications.

In order to obtain good performance for the small matrix multiplications belonging to a stack, two libraries for small matrix multiplications have been developed: `LIBXSMM` [75] for CPUs and `LIBSMM_ACC` for GPUs. These libraries contain optimized kernels for arbitrarily small multiplication sizes. At the limit of large block sizes ($\{m, n, k\} > 80$) BLAS libraries become efficient and DBCSR directly calls cuBLAS or BLAS. Thanks to the generality of the sparsity structure, supporting dense blocks of arbitrary size, DBCSR performs well not only for sparse matrices, but also for nearly-dense or dense matrices where a performance comparable to `ScaLAPACK`'s `PDGEMM` [76] has been reported [71].

4.4 Generalizing DBCSR to Tensors

With the formalism of mapping tensor contractions to matrix multiplications Sec. 4.2 at hand, it's possible to use a matrix multiplication library as a backend for tensor contractions as long as the tensor-matrix mapping can be implemented efficiently at low computational overhead. The advantage of such an approach is that all tensor contractions can be reduced to a much smaller class of operations that are already efficiently implemented in the matrix library. This allows to obtain consistently good performance in a generic library design without case-by-case optimizations or implementations. A requirement is however that the shape and properties of the matrix representation of a tensor can be handled efficiently in the matrix library.

Regarding the properties of the matrix representations of tensors, the following observations need to be considered which are summarized here and addressed in details below

1. *Tall-And-Skinny (TAS) matrices*: due to the tensor index folding, one matrix dimension may be larger by several orders of magnitude than the other matrix dimension. For the example of a rank-3 tensor of shape (N, N, N), the matrix representation has shape (N^2, N), so that the ratio of the matrix dimensions increases linearly with system size N.

2. *Sparsity / data locality*: the index folding function f in Eq. (4.9) should preserve the sparsity pattern in terms of blocks such that each tensor block maps to a single matrix block.

3. *Block sizes*: Tensor blocks are generally much larger than matrix blocks since the number of elements per block scales as s^d where s is the block size in one dimension and d is the tensor rank.

4. *Parallelism*: The mapping between tensors and matrices should take into account the distribution such that switching between tensor and matrix representation does not involve data exchange.

Tall-And-Skinny (TAS) matrices

The matrix shape and occupancy affect the communication volume in the parallel matrix multiplication and the memory footprint of storing index-related metadata. Cannon's algorithm is known to be communication optimal for matrices with equal occupancies but not for rectangular or even tall-and-skinny matrices [77]. The memory footprint of storing index-related information (distribution and block sizes) scales linearly with the largest matrix dimension or as $O(N^2)$ for 3-rank tensors. Since this data is not distributed, the $O(N^2)$ scaling holds for the local data stored on each processor. Eventually, for large systems, the memory consumed by index metadata will exceed the memory required to store the actual matrix data. To conclude, the DBCSR matrix format is not appropriate for TAS matrices and a specialized format for matrices with one large dimension should be developed. A communication-avoiding parallel matrix multiplication algorithm that reduces communication volume compared to Cannon's algorithm is an important optimization for tensor contraction.

Data locality

Data locality can be ensured by defining a mapping reminiscent to the Z-order curve [78] which maps multidimensional data to one dimension while maintaining locality of the data points. In our case the mapping should take into account the block sizes to ensure that elements belonging to a block have contiguous indices also in the one-dimensional or linear representation. A direct way of mapping an index tuple $I = (i_1, i_2, \dots, i_{n_I})$ with associated shape $(I_1, I_2, \dots, I_{n_I})$ to a one-dimensional index is given by the mapping

$$f_{\mathrm{c1}}(I) = \sum_{k=1}^{n_I} \left(\prod_{l=1}^{k-1} I_l \right) (i_k - 1) + 1 \tag{4.17}$$

the subscript c1 indicating column-major order with indices starting at 1. The block-preserving mapping can then be defined by addressing an element indirectly with 2 indices: the index B of the block it belongs to and the index E of the element inside a specific block. Each of the two indices are independently folded by f_{c1} so that the linear indices are obtained by $\tilde{B} = f_{\mathrm{c1}}(B)$ and $\tilde{E} = f_{\mathrm{c1}}(E)$. This mapping ensures that blocks don't get split apart by the folding.

Block sizes

Large block sizes have the advantage of providing more operations per submatrix multiplication such that a higher performance can be achieved. On the downside large blocks make it more difficult to obtain a load-balanced distribution since there are less blocks per core. Then load imbalances due to sparsity and heterogeneous block sizes do not average out as easily. Splitting blocks into smaller sizes is advantageous for exploiting subblock sparsity by e.g. adapting block sizes to a set of basis functions with the same widths. The optimal block sizes will depend on the exact sparsity pattern and will be defined by the user. However optimized block sizes may render data access complicated when the block indices no longer corresponds to an intuitive property such as atom number. For manipulating blocks manually (as for creating and filling a tensor), different block sizes may be chosen than for performing contractions, and API routines should be provided for adapting block sizes dynamically.

Parallelism

DBCSR distributes blocks over a two-dimensional grid of N_P MPI processes. The N_P processes are factorized as $N_\mathrm{P} = P_1 \times P_2$ into P_1 rows and P_2 columns such that every process p is assigned the coordinate pair (p_1, p_2) with $p = p_2 + P_2 \cdot p_1$. Here we adopt the MPI convention of indices starting with 0 and row-major ordering of process grids. Matrix block coordinates (b_1, b_2) are mapped to processes by a distribution $D = (\mathbf{d}_1, \mathbf{d}_2)$, with $d_1(b_1)$ mapping row index b_1 to a process row p_1 and $d_2(b_2)$ mapping column index b_2 to a process column p_2. The distribution D can be chosen arbitrarily by the user of the library.

A generalization of matrix distribution to tensors is straightforward: for a tensor of rank n_I, the N_P processes are factorized as $N_\mathrm{P} = \prod_{k=1}^{n_I} P_k$. The process coordinates are $P = (p_1, \ldots, p_{n_I})$ with

$$p = f_{\mathrm{r0}}(P) = \sum_{k=1}^{n_I} \left(\prod_{l=k+1}^{n_I} P_l \right) p_k \tag{4.18}$$

the subscript r0 denoting row-major ordering with indices starting at 0. The distribution mapping block indices $B = (b_1, \ldots, b_{n_I})$ to process coordinates has the form $D = (\mathbf{d}_1, \ldots, \mathbf{d}_{n_I})$ with $d_k(b_k)$ mapping block index b_k to a process coordinate p_k.

The mapping between tensors and matrices should take into account the distribution so that switching between tensor and matrix representation does not involve data exchange. Thus a mapping between a 2-dimensional process grid for the matrix representation and a n_I-dimensional process grid for the tensor representation is established. This mapping is formally the same as the mapping between matrix and tensor block index Eq. (4.8) with the folding function f_{r0} Eq. (4.18).

4.5 Tall-and-Skinny (TAS) Matrices

The TAS matrix format was developed as an optimized format on top of DBCSR to store and multiply tall-and-skinny matrices with one large dimension efficiently. We first review limitations of the DBCSR matrix format and of the parallel multiplication algorithm implemented in DBCSR. Then we review parallel matrix multiplication algorithms for TAS matrices. Finally we present an optimized TAS matrix format that enables both reduced communication and efficient storage.

4.5.1 DBCSR Matrix Format

As the largest matrix dimension grows as $O(N^2)$, it is crucial to avoid holding its full index metadata (block sizes and block distribution) in local memory. The need for such an optimization can be exemplified by considering a tensor of block shape (N, N, N), $N = n_\mathrm{b} \cdot N_\mathrm{a}$ with n_b the number of blocks per atom and N_a the number of atoms. We restrict the available memory per core for index metadata to 1 GB which is generous given that ideally the majority of available memory should be reserved for storing the matrix data. The index metadata consists of 2 4-byte integer arrays of size N^2, the block distribution and block sizes. We assume $n_\mathrm{b} = 4$ blocks per atom such that the total memory corresponds to $2 \cdot N^2 \cdot 4$ bytes $= 128 \cdot N_\mathrm{a}^2$ bytes. Under these assumptions the memory limit of $\approx$ 1GB corresponds to $\approx$ 3000 atoms. This estimate is rather conservative, assuming a sophisticated memory management and assuming that only one TAS matrix layout is held in memory at a time which is not practical. Even so the limiting system size is below the maximum system size treated so far with low-scaling RPA consisting of 6000 atoms [79]. These hypothetical findings can be backed up by looking at the tensor dimensions of an actual calculation of low-scaling RPA for 1536 water molecules:[1] the dimensions of the largest tensor $M^{\mathrm{virt}}_{\mu\sigma T}$ are $(12288, 12288, 30720)$, mapped to the large matrix dimension of size $12288 \times 30720 \approx 3.8 \cdot 10^8$ (in terms of blocks), corresponding to 1.5 GB of local storage (for only one index array). We conclude that the DBCSR matrix format is not suitable for large sparse tall-and-skinny matrices since the memory excess of index management becomes unaffordable for large systems.

The $O(N^2)$ scaling of index metadata is an inherent issue of matrix representations of tensors, not of tensors themselves: in the tensor representation, index metadata consists of n_I vectors whose sizes scale as $O(N)$ where n_I is the tensor rank. Explicit storage of

[1]the even larger system of 6000 atoms has not been repeated in the new implementation due to our focus on realistic systems extended in all dimensions.

the matrix index data can be avoided by delegating the handling of index data to the tensor layer.

4.5.2 Data Exchange

An important aspect in designing efficient parallel algorithms for distributed-memory machines is the communication volume or bandwidth cost which is the amount of data that is exchanged (sent and received) by each processor. An algorithm is said to be communication optimal when it does not perform more communications than is necessary to solve the problem in parallel under the assumption of perfectly load-balanced data. Communication lower bounds for matrix multiplications were proven by [80]. Demmel et al. [81] reviewed communication optimality of different parallel matrix-matrix multiplication algorithms for arbitrary matrix dimensions, and suggested the CARMA algorithm as asymptotically optimal for all matrix dimensions. Even though they assume dense matrices, their result is transferrable to the case of sparse matrices by expressing communication costs as a function of the matrix occupancies instead of the matrix dimensions.

Another important aspect is the load balancing of the data which is trivial for the dense case but hard to achieve for sparse data with irregular block sizes. The DBCSR library relies on a grid-based distribution and the distribution is chosen adapted to the block sizes so that roughly the same number of elements is assigned to each process. Randomization over rows and columns should homogenize the sparsity structure of the matrix. Parallel matrix multiplication algorithms for the sparse case should be oblivious to the distribution to leave as much freedom as possible to load-balance the data.

The communication volume of the parallel multiplication algorithm depends on the occupancy of the matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ in $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$, assuming a random sparsity pattern. For the case of sparse tall-and-skinny matrices, occupancies can vary a lot for the 3 matrices, if for instance a tall-and-skinny matrix is multiplied with a small square matrix. Thus it is important that the chosen algorithm performs well for all possible combinations of varying occupancies in the 3 matrices.

Communication costs of parallel matrix multiplication algorithms

The CARMA algorithm [81] is communication-optimal for all matrix occupancies. This algorithm is based on splitting the largest dimension M, N, K involved in a matrix multiplication, yielding 2 smaller subproblems. A generalized version of this algorithm uses an arbitrary split factor s. The splitting is repeated recursively on the resulting subproblems.

By imposing an initial distribution of matrix elements that is adapted to the recursive splitting scheme, only the smallest matrix of a subproblem is ever communicated. To assess the bandwidth costs T_{CARMA} three cases need to be distinguished :

1. $N_\mathrm{P} < N_2/N_1$ (one large dimension):

$$T_{\mathrm{CARMA}} = O(N_1) \tag{4.19}$$

2. $N_2/N_1 < N_\mathrm{P} < N_3^2/(N_1N_2)$ (two large dimensions):

$$T_{\mathrm{CARMA}} = O\left(\sqrt{\frac{N_1N_2}{N_\mathrm{P}}}\right) \tag{4.20}$$

3. $N_3^2/(N_1N_2) < N_\mathrm{P}$ (three large dimensions):

$$T_{\mathrm{CARMA}} = O\left(\frac{(N_1N_2N_3)^{1/3}}{N_\mathrm{P}^{2/3}}\right) \tag{4.21}$$

These costs were originally derived for the dense case [81] and were generalized here to the sparse case by rephrasing the costs in terms of matrix occupancy instead of matrix dimensions. For case 3 we ignored memory restrictions that would increase communication volume.

The hierarchical CARMA distribution is fundamental different from a grid-based distribution as is used in DBCSR, and the initial distribution is rather restrictive: it requires that the two large matrices of a subproblem can be split so that each of the two subproblem is mapped to half of the processors. This requirement applies recursively down to some block size. This basically predetermines the distribution which conflicts with our requirement that an algorithm for sparse matrix multiplication should be oblivious to the distribution. However a simplified and grid-based variant of CARMA is still possible in which only one split step is performed.

We consider the bandwidth cost associated with Cannon's algorithm for a multiplication $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ given by [77]

$$T_{\mathrm{Cannon}} = \frac{N_A + N_B}{\sqrt{N_\mathrm{P}}} \tag{4.22}$$

and assume the unfavorable situation that the occupancies are $N_A \geq N_B \gg N_C$. The bandwidth cost of Cannon's algorithm is $O(N_A/\sqrt{N_\mathrm{P}})$. The bandwidth cost of CARMA is either $O(N_C)$ or $O(\sqrt{N_B N_C / N_\mathrm{P}})$, depending on the actual occupancy and parallelization, but in any case much lower than for Cannon. We conclude that Cannon's algorithm has bandwidth costs much larger than the optimum for the case that one or two matrices are much larger than the other matrices.

Hybrid TAS algorithm

This section presents a novel algorithm for multiplying sparse tall-and-skinny (TAS) matrices which combines Cannon's algorithm [74] with a simplified, non-recursive version of CARMA [81], which is equivalent to the dimension splitting technique going back to Frigo, Leiserson, Prokop and Ramachandran (1999) [82]. The main objective of such a hybrid algorithm is to lower communication costs of Cannon's algorithm at the same time as maintaining a grid-based approach. We call this algorithm TAS-hybrid because it is specifically optimized for tall-and-skinny matrices and uses a combination of dimension splitting with any other parallel matrix multiplication algorithm.

We first outline the TAS-hybrid parallel multiplication algorithm. We compare the bandwidth cost with Cannon's algorithm and with the communication lower bound. The dimension-splitting algorithm is oblivious to the parallel algorithm it is combined with, therefore we describe Cannon's algorithm only with respect to its communication volume. For a description of the DBCSR implementation of Cannon's algorithm, we refer to [71].

The communication scheme is illustrated in Fig. 4.1. Only the smallest of the three matrices needs to be communicated if the two other matrices have a compatible layout (either *tall-and-skinny* or *short-and-fat*) and if the rows or columns of the largest matrix dimension map to the same process coordinates. The split factor s is required to be a divisor of the respective process grid dimension. Since no assumptions are made for the grid-based distribution, the splitting of the matrix to disjoint process subgroups requires a row or column permutation so that the new rows or columns are assigned contiguously to subgroups. The permutations are only logical operations and do not require a change in the matrix layout, data reshuffling or data exchange. The resulting mapping between matrices and submatrices is depicted in Fig. 4.2.

We compare the communication volume of *TAS-hybrid* with *Cannon* and with the bandwidth lower bound of *CARMA*. For the matrix-matrix multiplication $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ the communication volume is expressed in terms of the occupancies N_A, N_B, N_C which we sort according to $N_1 \leq N_2 \leq N_3$. In order to discuss bandwidth costs independently of the ordering of N_A, N_B, N_C, we focus on the extreme cases of bandwidth of *Cannon*'s

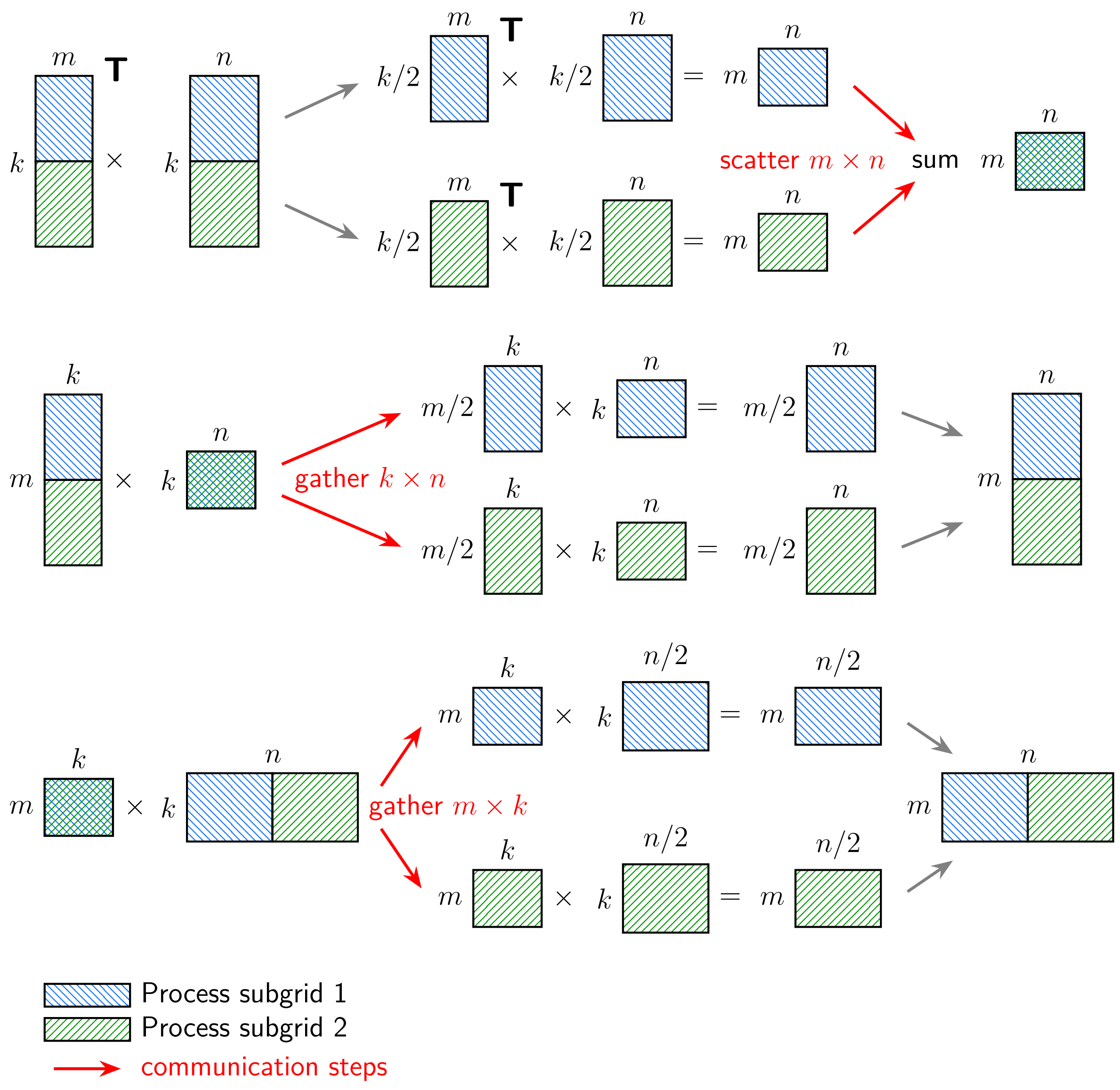


Figure 4.1: Grid-based communication scheme for TAS-hybrid algorithm with split factor s=2: the largest dimension is split into s parts. Depending on which of m, n, k is the largest dimension, three different cases arise. Grey arrows indicate mapping steps (grid-based dimension splitting, see Fig. 4.2) that do not involve communications. Red arrows indicate communication steps where data is exchanged between processors, see Fig. 4.3. A precondition for the former to not involve data exchange is that the two largest matrices are both in either *tall-and-skinny* layout (more rows than columns) or *short-and-fat* layout (more columns than rows), and that the distribution associated with the largest dimension is the same for both matrices. The multiplication of submatrices is performed in parallel on the process subgroups using any parallel matrix-matrix multiplication algorithm.

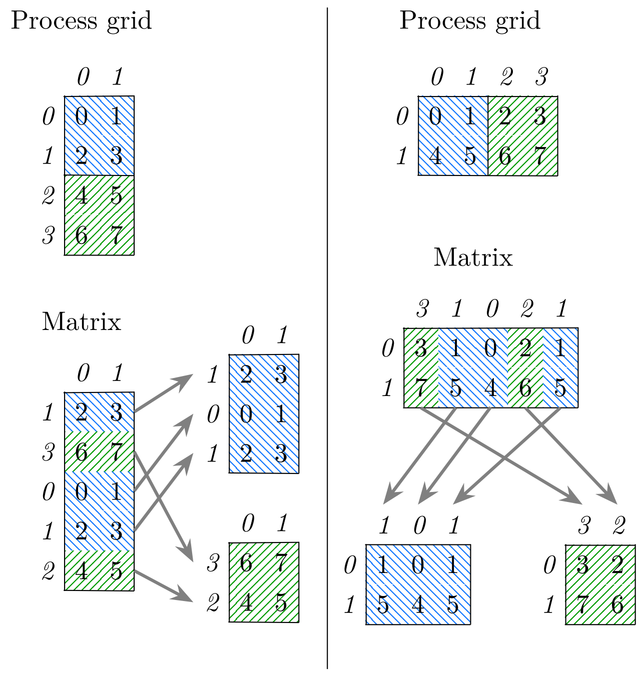


Figure 4.2: Grid-based scheme for splitting a matrix distributed on 8 processors into $s = 2$ submatrices on disjoint process subgroups. The process grid is split contiguously along its rows (*tall-and-skinny* matrices, left panel) or columns (*short-and-fat* matrices, right panel). The matrix rows / columns are labelled with the distribution, i.e. the process row / column they are mapped to (italic font). The matrix entries are annotated with the processor they reside on (regular font). The arrows indicate the mapping between matrix rows / columns and submatrix rows / columns.

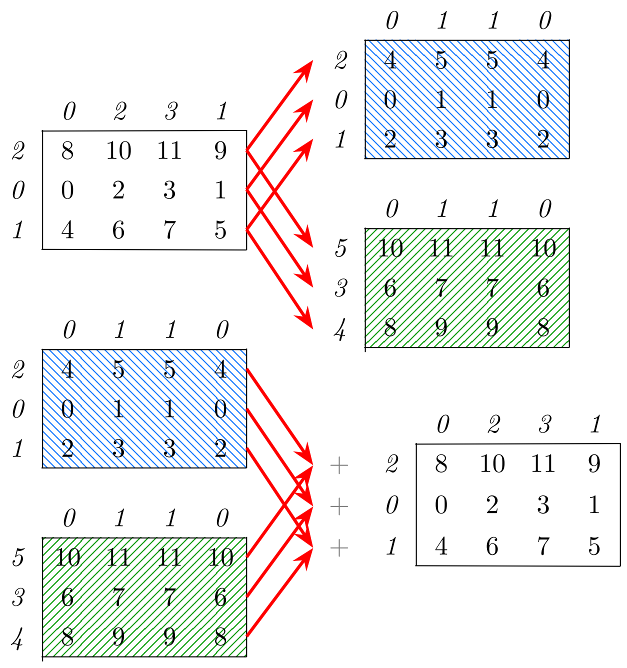


Figure 4.3: Grid-based scheme for the *gather* (top panel) and *scatter* (bottom panel) data exchange between a distributed matrix and $s = 2$ replicated matrices. The process grid for the distributed matrix can be chosen arbitrarily (here 3×4) and the process grid of the replicated matrices is dictated by the TAS-hybrid algorithm (here 3×2). For the scatter operations the two replicated matrices are merged by a sum.

algorithm $T_{\text{Cannon,min}} \leq T_{\text{Cannon}} \leq T_{\text{Cannon,max}}$ with

$$T_{\text{Cannon,min}} = \frac{N_1 + N_2}{\sqrt{N_{\text{P}}}} \tag{4.23}$$

$$T_{\text{Cannon,max}} = \frac{N_2 + N_3}{\sqrt{N_{\text{P}}}} \tag{4.24}$$

The favorable case $T_{\text{Cannon,min}} = O(N_2/\sqrt{N_{\text{P}}})$ is achieved if the output matrix $\mathbf{C}$ has the highest occupancy. If $\mathbf{A}$ or $\mathbf{B}$ has the highest occupancy, the bandwidth is $T_{\text{Cannon}} = O(N_3/\sqrt{N_{\text{P}}})$.

The *TAS-hybrid* is composed of two algorithms which we call *TAS-CARMA* (dimension splitting) and *TAS-Cannon* (multiplication of submatrices). For *TAS-CARMA*, there are s subgroup of processors, each subgroup consisting of N_{P}/s processors holding a full copy of the smallest matrix with occupancy N_1. Thus each processor of a subgroup holds $N_1 s/N_{\text{P}}$ data of the replicated matrix and each processor holds N_1/N_{P} data of the distributed matrix. The *gather* and *scatter* communication steps outlined in Fig. 4.3 perform the following data exchange[2]

Gather: each processor sends N_1/N_{P} data to each of s other processors and receives $N_1 s/N_{\text{P}}$ data.

Scatter: each processor sends $N_1 s/N_{\text{P}}$ data and receives N_1/N_{P} data from each of s other processors.

For both operations the communication volume is

$$T_{\text{TAS-CARMA}} = \frac{N_1 s}{N_{\text{P}}} \tag{4.25}$$

For the case that $N_{\text{p}} \leq N_2/N_1$, we choose $s = N_{\text{P}}$ and the multiplication of submatrices is local to each processor. This case is equivalent to the *CARMA* algorithm and thus also communication optimal. Otherwise the *TAS-hybrid* algorithm runs *Cannon* on N_{P}/s processes with the two largest submatrices split by a factor of s. The matrices thus have occupancies $N_{\tilde{1}} = N_1, N_{\tilde{2}} = N_2/s, N_{\tilde{3}} = N_3/s$. The bandwidth cost depends on the ordering of N_A, N_B, N_C:

[2]the terms *gather* and *scatter* stand for complex data movement operations that don't relate to the MPI routines MPI_Gather and MPI_Scatter but that are reminiscent of the MPI routines MPI_Allgather and MPI_Reduce_scatter, however with the crucial difference that data is not broadcast/reduced to/from each processor but to/from each subgroup of processors of size N_{P}/s.

1. $N_A = N_1, N_B = N_2 \implies N_{\tilde{A}} = N_1, N_{\tilde{B}} = N_2/s$

$$T_{\mathrm{TAS-Cannon}} = \frac{N_1 \cdot s + N_2}{\sqrt{N_\mathrm{P} \cdot s}} \leq \frac{N_2 + N_3}{\sqrt{N_\mathrm{P} \cdot s}} \tag{4.26}$$

2. $N_A = N_1, N_B = N_3 \implies N_{\tilde{A}} = N_1, N_{\tilde{B}} = N_3/s$

$$T_{\mathrm{TAS-Cannon}} = \frac{N_1 \cdot s + N_3}{\sqrt{N_\mathrm{P} \cdot s}} \leq \frac{2N_3}{\sqrt{N_\mathrm{P} \cdot s}} \tag{4.27}$$

3. $N_A = N_2, N_B = N_3 \implies N_{\tilde{A}} = N_2/s, N_{\tilde{B}} = N_3/s$

$$T_{\mathrm{TAS-Cannon}} = \frac{N_2 + N_3}{\sqrt{N_\mathrm{P} \cdot s}} \tag{4.28}$$

We introduced the condition $s \leq N_3/N_1$ for simplicity and we assumed without loss of generality $N_B \geq N_A$. An upper bound for the communication volume is thus given by

$$T_{\mathrm{TAS-Cannon}} \leq \frac{2N_3}{\sqrt{N_\mathrm{P} \cdot s}} \tag{4.29}$$

The communication volume of the *TAS-hybrid* algorithm is the sum

$$T_{\mathrm{TAS-hybrid}} = T_{\mathrm{TAS-CARMA}} + T_{\mathrm{TAS-Cannon}} \leq \frac{N_1 s}{N_\mathrm{P}} + \frac{2N_3}{\sqrt{N_\mathrm{P} \cdot s}} \tag{4.30}$$

The split factor s was not specified so far and is obtained by minimizing $T_{\mathrm{TAS-hybrid}}$ w.r.t. s under the conditions $s \leq N_\mathrm{P}$ and $s \leq N_3/N_1$, resulting in

$$s_{\mathrm{opt}} = \min\left(N_\mathrm{P}, \frac{N_3}{N_1}\right) \tag{4.31}$$

The memory overhead of $T_{\mathrm{TAS-hybrid}}$ over the minimum memory required for storage was not discussed so far and it is equal to $N_1 s$, the memory for creating s copies of the smallest matrix N_1 (ignoring the additional memory required by *Cannon*). With this choice of s_{opt} we have $N_1 s \leq N_3$ hence the total memory required is within a factor of 2 of the minimum required memory. Given this rather benign memory overhead, we don't introduce any memory constraint for the choice of s. A more promising strategy to reduce memory is to perform matrix multiplications in smaller batches (avoiding the full storage of an intermediate result in a series of multiplications) which will be discussed in Sec. 4.5.3.

Inserting s_{opt} into Eq. (4.30) we obtain the bandwidth costs

$$T_{\text{TAS-hybrid}} = \begin{cases} N_1 & \text{if } N_{\text{P}} < N_3/N_1 \\ \frac{N_3}{N_{\text{P}}} + 2\sqrt{\frac{N_1 N_3}{N_{\text{P}}}} & \text{if } N_{\text{P}} \geq N_3/N_1 \end{cases} \tag{4.32}$$

The bandwidth for the first case is lower than predicted from Eq. (4.30) because we removed *Cannon* communications due to *Cannon* being executed on a single processor. We calculate the ratio $T_{\text{TAS-hybrid}}/T_{\text{Cannon}}$ to decide in what cases the *TAS-hybrid* algorithm is favorable over *Cannon*'s algorithm

$$\begin{aligned} \frac{T_{\text{TAS-hybrid}}}{T_{\text{Cannon,min}}} &= O\left(\frac{\sqrt{N_1 N_3}}{N_1 + N_2}\right) = O\left(\sqrt{\frac{N_1}{N_2}}\sqrt{\frac{N_3}{N_2}}\right) \\ \frac{T_{\text{TAS-hybrid}}}{T_{\text{Cannon,max}}} &= O\left(\frac{\sqrt{N_1 N_3}}{N_2 + N_3}\right) = O\left(\sqrt{\frac{N_1}{N_3}}\right) \end{aligned} \tag{4.33}$$

If only the upper bound of *Cannon*'s bandwidth $T_{\text{Cannon,max}}$ is considered, *TAS-hybrid* has always lower communication volume. But inspection of *Cannon*'s lower bound bandwidth $T_{\text{Cannon,min}}$ shows that there is one case where *Cannon* is expected to perform better: If $N_2 = O(N_1)$, then $T_{\text{TAS-hybrid}}/T_{\text{Cannon,min}} = O(\sqrt{N_3/N_2})$. We summarize that *Cannon* has a lower asymptotic communication volume by a factor of $\sqrt{N_2/N_3}$ over *TAS-hybrid* only for the specific case that **C** has the highest occupancy and $N_2 \sim N_1$. *TAS-hybrid* has a lower asymptotic communication volume by a factor of $\sqrt{N_1/N_3}$ over *Cannon* if **A** or **B** have the highest occupancy or if $N_3 \sim N_2$.

Next we estimate the ratios $T_{\text{TAS-hybrid}}/T_{\text{CARMA}}$ as a measure of the deficiency of the *TAS-hybrid* algorithm w.r.t. the theoretical communication lower bound:

$$\frac{T_{\text{TAS-hybrid}}}{T_{\text{CARMA}}} = \begin{cases} O(1) & \text{if } N_{\text{P}} < N_2/N_1 \\ O\left(\sqrt{\frac{N_3}{N_2}}\right) & \text{if } N_2/N_1 < N_{\text{P}} < N_3^2/(N_1 N_2) \\ O\left(\left[\frac{N_1}{N_2}\right]^{1/4} N_{\text{P}}^{1/4}\right) & \text{if } N_3^2/(N_1 N_2) < N_{\text{P}} \end{cases} \tag{4.34}$$

We conclude by establishing a hierarchy of decreasing asymptotic communication cost between the three algorithms with TAS-hybrid performing in general better than Cannon but worse than CARMA:

1. *Cannon*: $T = O(N_3/\sqrt{N_{\text{P}}})$

2. *TAS-hybrid*: $T \to T/\sqrt{N_3/N_1}$

3. *CARMA*:

   - *2 large dimensions*: $T \to T/\sqrt{N_3/N_2}$
   - *3 large dimensions*: $T \to T/\left((N_1/N_2)^{1/4} \cdot N_\mathrm{P}^{1/4}\right)$

The communication reduction of *TAS-hybrid* over *Cannon* by a factor of $\sqrt{N_3/N_1}$ is substantial for the case $N_3 \gg N_1$. The shortcomings of *TAS-hybrid* is a factor of $\sqrt{N_3/N_2}$ increased communication volume over both *CARMA* and *Cannon*'s optimal case. Thus *TAS-hybrid* is most adapted to the situation $N_2 \sim N_3$ and $N_3 \gg N_1$. For the case of 3 large dimensions *TAS-hybrid* scales worse with the number of processes than *CARMA* by a factor of $N_\mathrm{P}^{1/4}$.

Not studied here but equally interesting is the combination of *TAS-hybrid* with an algorithm different from *Cannon*. We note that the remaining deficits of *TAS-hybrid* in communication costs are actually deficits of the *Cannon* algorithm - if it was combined with an algorithm that is communication optimal for either $N_3 = N_2$ or $N_2 = N_1$, the communication costs would be equal to *CARMA*.

The only reason we used *Cannon* is that it is well-established in the DBCSR library and showed good performance for sparse matrices, but *TAS-hybrid* is oblivous to the underlying algorithm for parallel matrix-matrix multiplication (only the split factor s_opt must be adapted to the underlying algorithm). The DBCSR library also comes with a *2.5D* algorithm [83] that has lower communication bounds than *Cannon* and it would be interesting to benchmark *TAS-hybrid* combined with *2.5D* against the *Cannon*-based version for different situations.

4.5.3 TAS Matrix Format

Previously we identified two needs that should be satisfied by the dedicated TAS matrix format for tall-and-skinny matrices:

- *Storage*: no replicated storage of index-related data scaling with $O(N_\mathrm{max})$, N_max being the large matrix dimension.

- *Multiplication Algorithm*: by splitting the large matrix dimension with a factor of s, s submatrices on disjount subgroups of processors should be obtained in DBCSR format.

Both requirements can be met by storing TAS matrices as DBCSR submatrices on disjoint process subgroups, split along the large dimension. It is desirable to have an API for TAS matrices that is mostly identical to the DBCSR API. This can be achieved most effectively by designing the TAS format as a wrapper library on top of DBCSR, inheriting most of the functionality of the DBCSR library. The TAS format is then reduced to a logical index mapping that associates rows or columns of the DBCSR submatrices to rows or columns of the TAS matrix. There are two layouts depending on whether the matrix is split along the rows (*tall-and-skinny* type) or along the columns (*short-and-fat* type).

Index data

The DBCSR format refers to submatrices and hence the TAS format must implement its own meta data describing the global structure of the matrix, most importantly block sizes and the parallel layout. The implementation should allow for an arbitrary distribution on a grid and arbitrary block sizes but the associated data should not be stored as explicit arrays in memory. Hence the way the index data is obtained is delegated to the user of the library. The TAS implementation implements index data as abstract derived types with type-bound procedures to obtain a certain value for a certain index. The user of the library implements block sizes and distribution as extensions of this abstract index type. The TAS implementation then calls this external implementation to obtain block size or process grid coordinate for a specific index on-the-fly. This trades storage for computations. However the associated computational costs are negligible in practice because for locally stored blocks, their index data can be calculated once and stored in local memory. The indirect indexing is then only needed during communication operations to receive blocks from other processes.

Data Exchange

Methods of the DBCSR library can be inherited only if they are local operations, acting on single matrix blocks. Non-local operations such as multiplication and redistribution are implemented based on the TAS matrix format. Three different data exchange operations provide the building blocks for the parallel matrix multiplication algorithm:

Replicate (gather): Replicates a distributed matrix to all s subgroups of processors

Merge (scatter): Merge the s replicated matrices to one distributed matrix by sum

Redistribute: Redistribute a distributed matrix to a different parallel layout

The first two communication patterns were already described in Fig. 4.3 and are the ingredients for the parallel matrix multiplication algorithm depicted in Fig. 4.1. The *redistribute* operation is additionally required since the algorithm assumes that the two largest matrices are both in either *tall-and-skinny* or *short-and-fat* layout, and that the distribution associated with the large dimension is the same for both matrices. If this is not the case, the *redistribute* operation is needed to redistribute the second largest matrix to a distribution compatible with the largest matrix. This operation is not only provided for ease-of-use but is strictly needed if the same matrix is involved in different multiplications, since a different layout may be required for each of the multiplications. It is instructive to compare the communication bounds of the matrix multiplication $T_{\text{TAS-hybrid}}$ Eq. (4.32) with the communication volume of redistributing one or both of the largest matrices, contributing an additional communication volume of $T_{N_2} = N_2/N_{\text{P}}$ and/or $T_{N_3} = N_3/N_{\text{P}}$:

$$T_{N_2} > T_{\text{TAS-hybrid}} \iff N_{\text{P}} < N_2/N_1 \tag{4.35}$$

$$T_{N_3} > T_{\text{TAS-hybrid}} \iff N_{\text{P}} < N_3/N_1 \tag{4.36}$$

We conclude that complete redistribution may require more communication than matrix multiplication for the case of tall-and-skinny matrices. The implementation thus avoids the redistribution of the two large matrices involded in a matrix multiplication if not strictly needed.

Practical considerations for TAS-hybrid algorithm

There are several practical complications in the implementation that were not yet covered by the idealized description of the multiplication algorithm so far. A more complete description is given in Alg. 1. We point out 3 issues:

1. The result occupancy N_C is required to get s_{opt} Eq. (4.31) but is unknown
2. The split factor s must be a divisor of the respective process grid dimension
3. Cannon's algorithm performs best only for square process subgrids [71]

Issue 1 requires an estimate of the occupancy of $\mathbf{C}$ prior to the multiplication, based on the occupancies of $\mathbf{A}, \mathbf{B}$. Such an estimate is obtained by replacing each block in $\mathbf{A}, \mathbf{B}$ with its Frobenius norm. These smaller matrices are then multiplied at much lower costs than the multiplication of the original matrices. For all elements present in the

Algorithm 1 TAS multiplication algorithm

```
procedure MultiplyTAS($A, B, C, P$)    ▷ $C = A \cdot B$
    ▷ $A, B, C$: TAS matrices
    ▷ $P$: Process grid $(P_1, P_2)$ with $N_\mathrm{P} = P_1 \times P_2$ processors

    ▷ Estimate optimal split factor $s$:
    $N_A \leftarrow$ Occupancy($A$); $N_B \leftarrow$ Occupancy($B$)    ▷ Number of non-zero elements
    $A_\mathrm{Small} \leftarrow$ BlockNorms($A$); $B_\mathrm{Small} \leftarrow$ BlockNorms($B$)    ▷ Small matrices containing block norms
    MultiplyTAS($A_\mathrm{Small}, B_\mathrm{Small}, C_\mathrm{Small}, P$)
    $N_C \leftarrow \sum_{ij}$(BlockSize($C_{ij}$) if $C_{\mathrm{small},ij} > 0$)    ▷ Estimate Occupancy of $C$
    $N_1 \leftarrow$ Min($N_A, N_B, N_C$)
    $N_2 \leftarrow$ Median($N_A, N_B, N_C$)
    $N_3 \leftarrow$ Max($N_A, N_B, N_C$)
    $s \leftarrow$ Min $\left(N_\mathrm{P}, \frac{N_3}{N_1}\right)$

    ▷ Split process grid:
    if largest TAS matrix in row layout then
        $s \leftarrow$ SplitToGrid($s, P_1, P_2$)    ▷ Adapt $s$ so that it divides $P_1$ and $P_1/s \approx P_2$
        $\tilde{P} \leftarrow \left(\frac{P_1}{s}, P_2\right)$
    else if largest TAS matrix in column layout then
        $s \leftarrow$ SplitToGrid($s, P_2, P_1$)
        $\tilde{P} \leftarrow \left(P_1, \frac{P_2}{s}\right)$
    end if

    ▷ Parallel Multiplication Algorithm:
    if $N_A = N_1$ then
        $\tilde{A} \leftarrow$ Replicate($A, \tilde{P}$)    ▷ Send full copy of matrix to each subgrid $\tilde{P}$
    else
        $\tilde{A} \leftarrow$ Submatrix($A, \tilde{P}$)    ▷ Extract submatrix on local subgrid $\tilde{P}$
    end if
    if $N_B = N_1$ then
        $\tilde{B} \leftarrow$ Replicate($B, \tilde{P}$)
    else
        $\tilde{B} \leftarrow$ Submatrix($B, \tilde{P}$)
    end if
    if $N_C \neq N_1$ then
        $\tilde{C} \leftarrow$ Submatrix($C, \tilde{P}$)
    end if
    $\tilde{P}_\mathrm{sq} \leftarrow$ SquareGrid($\tilde{P}$)    ▷ Optimize subgrid so that $\tilde{P}_{\mathrm{sq},1} \approx \tilde{P}_{\mathrm{sq},2}$
    if $\tilde{P}_\mathrm{sq} \neq \tilde{P}$ then
        $\tilde{A}_\mathrm{sq} \leftarrow$ Redistribute($\tilde{A}, \tilde{P}_\mathrm{sq}$)
        $\tilde{B}_\mathrm{sq} \leftarrow$ Redistribute($\tilde{B}, \tilde{P}_\mathrm{sq}$)
        $\tilde{C}_\mathrm{sq} \leftarrow$ CreateDBCSR($\tilde{C}, \tilde{P}_\mathrm{sq}$)
        MultiplyDBCSR($\tilde{A}_\mathrm{sq}, \tilde{B}_\mathrm{sq}, \tilde{C}_\mathrm{sq}$)    ▷ Multiply DBCSR submatrices
        $\tilde{C} \leftarrow$ Redistribute($\tilde{C}_\mathrm{sq}, \tilde{P}$)
    else
        MultiplyDBCSR($\tilde{A}, \tilde{B}, \tilde{C}$)    ▷ Multiply DBCSR submatrices
    end if
    if $N_C = N_1$ then
        $C \leftarrow$ Merge($\tilde{C}, P$)    ▷ Distribute and merge submatrices by sum
    end if
end procedure
```

result matrix the respective block sizes of the original matrix $\mathbf{C}$ are summed up to give an estimate of the $\mathbf{C}$ occupancy. Due to the submultiplicativity of the Frobenius norm this gives an upper bound of the occupancy of $\mathbf{C}$ (given some filter threshold ϵ to discard small blocks in $\mathbf{C}$).

The two issues 2 and 3 impose additional restrictions on the choice of the process grid $P = (P_1, P_2)$: for a given split factor s the dimensions P_1, P_2 must be chosen so that $P_1 = \tilde{P}_1 s$ and $\tilde{P}_1 \approx P_2$ or $P_2 = \tilde{P}_2 s$ and $P_1 \approx \tilde{P}_2$ (depending on the layout). Condition 2 is less of a problem since the communication cost Eq. (4.30) is not very sensitive to the exact choice of s. Condition 3 on the other hand can not be met in practice since the optimal split factor is unknown when the process grids are created. Hence an additional redistribution of the submatrices may be needed, mapping them to a more square subgrid. The associated communication costs equal to T_{N_3} Eq. (4.36) are not negligible in practice.

We remark that the multiplication algorithm based on the partitioning of the processors into ideally square subgrids gives best performance if the total number of processors consists of mostly small prime factors.

Batched multiplication

In a series of multiplications of TAS matrices, densification of the result matrices leads to growing memory requirements, which is often only temporary if the final result matrices are small in size. A natural memory optimization is the decomposition of a large matrix dimension into n_{mem} batches. The multiplication is then performed consecutively for each batch and large intermediate TAS matrices are never fully held in memory. Since the smallest matrix is roughly the same in size for each consecutive step, batched multiplication increases communication by a factor of n_{mem} in the TAS multiplication. A mechanism is implemented adapting data exchange to the batched multiplication so that communication is performed only in either the first or last batch. This optimization is activated by API methods that define the scope of the batched multiplication and that are called before the first and after the last batch is processed. The only communication overhead for batched multiplication is then only due to *Cannon*'s part of Eq. (4.32) amounting to a factor of $\sqrt{n_{\mathrm{mem}}}$.

4.6 Tensor API

The main practical challenge of implementing a tensor library is the large number of different cases that arise concerning different combinations of tensor ranks. Partly this challenge has already been addressed by identifying tensor contraction as isomorphic to matrix multiplications and by separating the tensor API from the implementation and optimization in terms of tall-and-skinny (TAS) matrices. This leads to the design decision to consider tensors as a special representation of matrices, so that the tensor descriptor can be implemented as a set of mapping functions acting on indices instead of data.

The general design of the DBCSR tensor library is illustrated in Fig. 4.4 and is based on 3 nested layers:

- **Tensor**: abstract API for tensors of arbitrary rank, mappings to represent tensors as matrices
- **TAS**: communication and memory optimizations for tall-and-skinny matrices with one large dimension
- **DBCSR**: storage and multiplication of submatrices on subgroups of processors

Tensor data is fully distributed and stored in the DBCSR format. Index information (block sizes and distribution) is replicated and maintained in the tensor format. Conversions between the 3 representations are implemented by on-the-fly index and data transformations.

A tensor of rank n is represented on a n-dimensional process grid. Process grid coordinates and tensor indices are folded to the matrix representation by the folding functions described in Sec. 4.4. All tensor data is stored in the DBCSR matrix format and is reshaped to the tensor format whenever a block is accessed. The index data is maintained in the tensor layer due to the compact index size. Index access in the TAS layer is implemented by derived types simulating function objects that calculate the matrix index based on the tensor-specific index folding function.

API methods

We do not describe in further details the tensor procedures that map trivially to DBCSR matrix procedures such as methods to create or destroy tensor objects, methods to obtain or insert blocks or methods to iterate over blocks. These methods are written as simple wrappers that convert between tensor and matrix index before or after calling the respective DBCSR matrix method. In case of data access, the wrappers also reshape a given matrix block in 2d format to the shape of the respective tensor block (or vice versa).

More interesting are the methods specific to tensor contraction. Two operations serve as the building blocks for any more complex contraction:

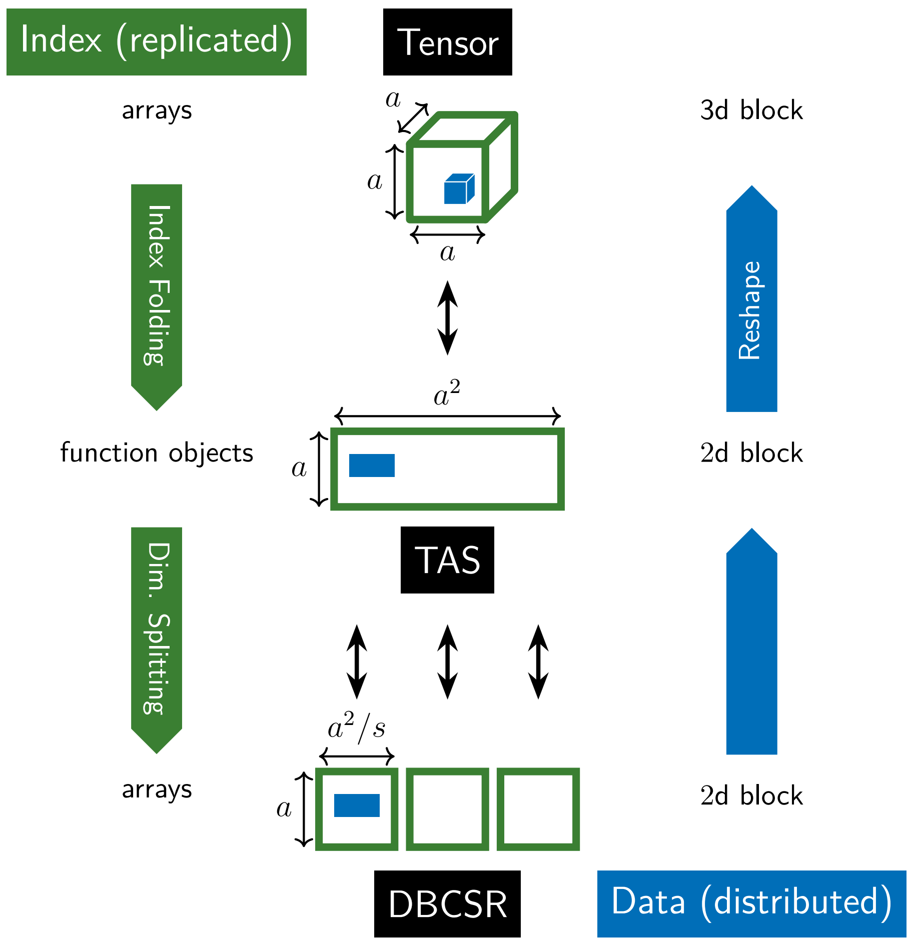


Figure 4.4: Schematic overview of the DBCSR tensor implementation consisting of the *Tensor API*, the *TAS (Tall-And-Skinny) matrix* layer and the *DBCSR* backend. The index data (green) containing block sizes and the parallel distribution is replicated on each core and is maintained in the compact tensor format. The data (blue) is distributed and stored in the DBCSR matrix format. Storing a large TAS matrix in terms of smaller submatrices leads to reduced communication costs in the parallel multiplication algorithm and circumvents the problem of storing large replicated index data in favor of flexible function objects. The connection between the layers is established with logical index mappings: *Index folding* mapping a multi-dimensional tensor index to a 2-dimensional matrix index and *Dimension splitting* resulting in s DBCSR submatrices on disjoint process subgroups. Data is accessed in terms of blocks and a local reshape operation converts on-the-fly between matrix blocks and n-dimensional tensor blocks. The rank of $n = 3$ is chosen only for illustrative purposes and the tensor implementation can deal with arbitrary ranks ≥ 2.

Copy: Copy a tensor from one layout to a different layout with optional index permutation and optional sum

Contract: Perform a contraction involving 3 tensors

The *Contract* procedure implements contraction according to Eq. (4.2). The *Copy* procedure is provided to transform a tensor between different layouts, including change of matrix representation, change of distribution and change of block sizes. Typically the *Copy* method is invoked between two contractions to convert a tensor that is involved in both contractions to a layout compatible with the second contraction. Block sizes are chosen according to an intuitive description to facilitate interfacing between DBCSR and the application code, however smaller block sizes may be beneficial for contraction since they allow to exploit sub-block sparsity. Changing block sizes is thus a crucial optimization that can easily be implemented with the *Copy* method. Finally, the *Copy* method can be used to implement arbitrary sums of 2 tensors including index permutations. If symmetries are present in a tensor, only the unique elements need to be calculated and the *Copy* method can then be used to desymmetrize by adding the transposed data.

In the current implementation only those types of contractions are implemented that map to a matrix-matrix multiplication. An extension to matrix-vector type contractions would rely on the same communication patterns and could be implemented in a future version of the library.

An example for the translation of a complex tensor contraction to the two tensor API procedures is given in Alg. 2. The Fortran source code for this example is listed in Appendix B.

Algorithm 2 Tensor contraction
$C(n,o) = C(n,o) + \sum_{i,j,k,l,m} A(i,j,k) \cdot A(l,m,k) \cdot B(i,l,n) \cdot [B(m,o,j) + B(o,m,j)]$

```
$D(i,j,l,m) \leftarrow \sum_k A(i,j,k) \cdot A(l,m,k)$                ▷ Contract
$E(j,m,n) \leftarrow \sum_{i,l} D(i,j,l,m) \cdot B(i,l,n)$          ▷ Contract
$F(j,m,o) \leftarrow B(m,o,j) + B(o,m,j)$                           ▷ Copy
$C(n,o) \leftarrow C(n,o) + \sum_{j,m} E(j,m,n) \cdot F(j,m,o)$     ▷ Contract
```

Matrix representation

The tensor API was designed to match the DBCSR matrix API as close as possible, consisting of natural generalizations of all methods to ranks ≥ 2. One important concept that is only present in the tensor API is the notion of mapping tensors to matrices. Conceptually this should be fully internal to the library and the user of the library should not be bothered to understand how tensors relate to their matrix representations. However the matrix representation needs to be adapted to the tensor contraction according to Eq. (4.10). If the tensors involved in a contraction don't have a compatible matrix representation, an additional redistribution is required prior to invoking the matrix multiplication. This is to be avoided since the communication volume of redistribution (given by N_3/N_P) potentially exceeds the communication costs of TAS matrix multiplication Eq. (4.32). Thus, for optimal performance, the user should provide the matrix representation when creating a tensor. The matrix representation is uniquely determined by specifying the subset of tensor indices that are mapped to each of the two matrix dimensions. Given 3 tensors involved in a contraction $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ with contraction indices $I = p_A(E), J = p_B(E), K = p_C(E)$ (according to Eq. (4.2)) the matrix indices should be chosen as follows:

$\mathbf{A}: \ (I \cap K, I \cap J)$ or $(I \cap J, I \cap K)$

$\mathbf{B}: \ (I \cap J, J \cap K)$ or $(J \cap K, I \cap J)$

$\mathbf{C}: \ (I \cap K, J \cap K)$ or $(J \cap K, I \cap K)$

The exact order of the matrix indices is dictated by the TAS multiplication algorithm which requires that the largest of the three matrix dimensions (occuring in two of the three matrices) should correspond to either the rows in both matrices or to the columns in both matrices. For instance if $I \cap J$ is the largest dimension it should map to either the rows of both $\mathbf{A}$ and $\mathbf{B}$ or to the colums of both matrices (not imposing any index order on $\mathbf{C}$). If the matrix indices are chosen arbitrarily, a compatible layout is automatically chosen in the contraction procedure, albeit at the expense of a redistribution of both large matrices.

Parameters and Optimizations

The tensor API was developed with the purpose of facilitating translation of arbitrary sparse tensor contractions from mathematical notation to optimized code. Ideally all optimizations should be internal to the library so that the user is concerned only with expressing equations in a high-level language. Well-established tensor contraction libraries such as *TensorFlow* [84], *PyTorch* [85] and the massively parallel *Cyclops Tensor Framework* [61] provide domain-specific languages based on Einstein notation in which tensor contractions can be expressed naturally. Our approach for sparse data sacrifices some of the abstraction for the sake of a communication-avoiding scheme. This implies some additional complexity in the library API:

- the user controls for each tensor its matrix representation that should be compatible with the contraction to be performed. The two largest tensors should both map to either *tall-and-skinny* or *short-and-fat* matrices
- the tensor contraction API is not based on Einstein notation but on the mapping to matrix multiplication Eq. (4.10)

Alternatively the matrix representation could be abstracted away from the API, in this case there is an additional cost of redistributing all tensors per contraction amounting to

$$T = \frac{N_A + N_B + N_C}{N_\mathrm{P}} \tag{4.37}$$

and exceeding the matrix multiplication communication costs Eq. (4.35). In our matrix-aware API, redistribution of a tensor is required only if the same tensor occurs in two different contractions. Even in this case only one redistribution is needed compared to two redistributions in a matrix-oblivious API. The design decision of a matrix-aware tensor API is thus justified by the performance gain outweighing the additional complexity. A more user-friendly and fully abstract tensor API could still be implemented on top of the current expert API.

Two optimizations have already been discussed for the TAS matrix implementation:

Batched contraction: different indices are split into contiguous ranges and the corresponding tensor batches are contracted consecutively for memory reduction.

Process grid optimization: performance of the TAS matrix multiplication is sensitive to the chosen process grid dimensions and the optimal process grid may be different for each contraction. A suitable process grid can be proposed by the TAS matrix multiplication algorithm and can be obtained after a contraction has been performed.

Both optimizations are enabled automatically by calling procedures that define the scope (start and end) of a batched contraction. All batches contain roughly the same amount of data and thus the process grid can automatically and continuously be optimized over the iterations of a batched contraction. Alg. 3 shows tensor contraction performed in batches for the same contraction as Alg. 2. The complete Fortran source code is provided as a standalone tensor example that comes with the DBCSR source code [1].

Algorithm 3 Batched tensor contraction
$C(n,o) = C(n,o) + \sum_{i,j,k,l,m} A(i,j,k) \cdot A(l,m,k) \cdot B(i,l,n) \cdot [B(m,o,j) + B(o,m,j)]$

```
$N_{\text{mem}}$                                              ▷ number of batches for each index
$i_{\text{mem}} = (1, 1+\text{s}, \dots, n_i)$                 ▷ split $i$ into $N_{\text{mem}}$ ranges by $N_{\text{mem}} + 1$ index offsets
$j_{\text{mem}}, l_{\text{mem}}, m_{\text{mem}}$               ▷ split $j, l$ and $m$ into $N_{\text{mem}}$ ranges each

BatchedStart($A$), $\cdots$, BatchedStart($F$)                 ▷ enables optimizations for batched contraction
for $n_J = 1, N_{\text{mem}}$ do
    $J \leftarrow [j_{\text{mem}}(n_J), j_{\text{mem}}(n_J + 1) - 1]$          ▷ range of $j$ belonging to batch $n_J$
    for $n_M = 1, N_{\text{mem}}$ do
        $M \leftarrow [m_{\text{mem}}(n_M), m_{\text{mem}}(n_M + 1) - 1]$      ▷ range of $m$ belonging to batch $n_M$
        $E(j,m,n) = 0$
        for $n_I = 1, N_{\text{mem}}$ do
            $I \leftarrow [i_{\text{mem}}(n_I), i_{\text{mem}}(n_I + 1) - 1]$  ▷ range of $i$ belonging to batch $n_I$
            for $n_L = 1, N_{\text{mem}}$ do
                $L \leftarrow [l_{\text{mem}}(n_L), l_{\text{mem}}(n_L + 1) - 1]$   ▷ range of $l$ belonging to batch $n_L$
                $D(i,j,l,m) \leftarrow \sum_k A(i,j,k) \cdot A(l,m,k)$     for $i \in I, j \in J, l \in L, m \in M$
                $E(j,m,n) \leftarrow E(j,m,n) + \sum_{i \in I, l \in L} D(i,j,l,m) \cdot B(i,l,n)$     for $j \in J, m \in M$
            end for
        end for
        $F(j,m,o) \leftarrow B(m,o,j) + B(o,m,j)$                  for $j \in J, m \in M$
        $C(n,o) \leftarrow C(n,o) + \sum_{j \in J, m \in M} E(j,m,n) \cdot F(j,m,o)$     for $j \in J, m \in M$
    end for
end for
BatchedEnd($A$), $\cdots$, BatchedEnd($F$)
```

All relevant parameters of the tensor API are summarized in Tab. 4.1. Parameters related to sparsity (*block sizes* and *filtering threshold*) have not been mentioned so far because they are not specific to the tensor implementation. Using smaller block sizes may expose more sparsity but leads to less operations per block such that the performance may suffer if blocks are too small. The best block size depends on the nature of the sparsity pattern. The filtering threshold needs to be set empirically to meet the accuracy requirements of the application.

Generic implementation

The generic implementation of operations on data of different ranks poses challenges because the built-in Fortran array type and array syntax is specific to the rank of an array. Thus rank-generic algorithms necessarily lead to code duplication with a specific implementation for each combination of data type and rank. Given 4 data types (complex & real in single & double precision) and 3 different ranks (between 2 and 4), 12 specialized implementations are needed for each data operation. This asks for a template engine or code generator. Since Fortran does not come with templates or a powerful-enough

Parameter	internal (automatic)	external (user)
process grid	optimized for batches	initial process grid
distribution	default distribution	custom distribution
block sizes	–	adapted to sparsity pattern
filtering threshold	–	converged to accuracy needs
contraction batches	–	custom index partitioning
TAS parameters	split factor s	compatible matrix layouts

Table 4.1: Parameters of the tensor API categorized as *internal* and *external*. The parameters *process grid* and *distribution* define the parallel layout of tensors. The parameters *block sizes* and *filtering threshold* relate to the sparsity structure of tensors. *Contraction batches* reduce the memory footprint. For best performance the parallel *TAS matrix multiplication* requires a compatible matrix representation.

preprocessor, we deploy the Fypp preprocessor which combines a macro language with the evaluation of Python expressions [54]. All rank-specific code is expressed in terms of Fypp macros and the DBCSR tensor API can be compiled for arbitrary maximum rank specified by a configuration variable.

4.7 Validation

In this section we discuss the performance of the DBCSR tensor library and we validate parameters that affect performance and memory footprint of the library. Target applications of the library are any algorithms that can be formulated in terms of sparse tensor contractions.

The sparsity structure of a tensor has however an important effect on how the sparsity propagates over contractions. A homogeneous random sparsity structure in a multiplication of two large matrices does not conserve sparsity and the result matrix may be dense, even if the two other matrices are very sparse (reminiscent of the birthday paradox [86]). The notion of sparse matrix multiplication is only meaningful if the sparsity has a structure that is conserved over multiple multiplication steps. Such a structure is present if tensor elements decay with increasing difference between indices as is naturally the case if tensors are represented in terms of a localized basis.

The different ways of index blocking is a large tuning space where the index order and the block sizes affect the sparsity (favoring small block sizes) and the performance of the library (favoring large block sizes).

Due to the above-mentioned reasons there exists no simple example or test case that is representative for all possible applications of the library, even though the library is general (for sparse and dense tensors of arbitrary shape and rank) and is not favoring any specific application. We thus validate the library by directly applying it the low-scaling RPA algorithm introduced in Sec. 2.6.2.

We consider the two contractions

$$\text{a)} \sum_{\lambda} D^{\text{virt}}_{\mu\lambda}(\lambda\sigma|R) = M^{\text{virt}}_{\mu\sigma R} \tag{4.38}$$

$$\text{b)} \sum_{\mu\sigma} M^{\text{occ}}_{\mu\sigma R} M^{\text{virt}}_{\mu\sigma T} = P_{RT} \tag{4.39}$$

In terms of matrices, contraction a) is an example for the case ■ × ▬ = ▬ (■ symbolizing a small matrix and ▬ symbolizing a large tall-and-skinny matrix). Contraction b) represents the case ▬ × ▮ = ■. These two cases are representative for the two different algorithmic situations for tall-and-skinny multiplications where in case a) the result matrix is the largest and in case b) the result matrix is the smallest (see Fig. 4.1). These two cases also cover the two communication scenarios in Eq. (4.33), case a) favorable for Cannon and case b) favorable for the TAS algorithm (occupancies $N_3 \geq N_2 \geq N_1$):

$$\text{a)}\ T_{\text{Hybrid}-\text{s}}/T_{\text{Cannon}} = O(\sqrt{N_1/N_2}\sqrt{N_3/N_2}) \tag{4.40}$$

$$\text{b)}\ T_{\text{Hybrid}-\text{s}}/T_{\text{Cannon}} = O(\sqrt{N_1/N_3}) \tag{4.41}$$

All validation tests and benchmarks have been performed on the Piz Daint supercomputer of CSCS (Swiss National Supercomputing Centre) with Cray XC50 architecture. One node has 12 cores and 64 GB of RAM. In all calculations a flat MPI setup (12 ranks per node) has been found superior to a hybrid MPI/OpenMP setup. The GPU backend of DBCSR has been disabled since it showed inferior performance to the LIBXSMM [75] CPU backend.

The TAS matrix multiplication algorithm (introduced in Sec. 4.5.2) has been validated by modifying the split factor s so that $s = s_{\text{opt}}/p$ with $s_{\text{opt}} = N_3/N_1$ the predicted optimal split factor and p the parameter to be varied in the tests. There are 3 important limits to be evaluated:

DBCSR limit ($s = 1$ or $p = s_{\text{opt}}$): equivalent to Cannon's algorithm (no communication within TAS)

Hybrid DBCSR-TAS ($s = s_{\text{opt}}$ or $p = 1$): the predicted optimum minimizing communication costs

TAS limit ($s = N_\mathrm{P}$ or $p = s_\mathrm{opt}/N_\mathrm{P}$): mapping each DBCSR subproblem to a single core (no communication within DBCSR)

The test results presented in Fig. 4.5 confirm that the predicted optimum $p = 1$ performs always better than the DBCSR and TAS limits. For case a) the speedup of the DBCSR-TAS algorithm over Cannon's algorithm is not significant and the measured optimum value of p is shifted towards the DBCSR limit. The shift can be explained by the inequality in Eq. (4.26) where for simplicity and generality the communication upper bound for Cannon's algorithm was replaced by a less tight bound. In this example the ratios of Eq. (4.40) $N_2/N_1 \approx 72$ and $N_3/N_2 \approx 18$ are more in favor of the DBCSR-TAS algorithm than of the DBCSR limit. This leads us to the conclusion that the DBCSR-TAS algorithm may perform significantly worse than the DBCSR limit if tested on an example with $N_2 \approx N_1$. A switch to $s = 1$ for this case may then significantly improve performance, however we did not find this optimization relevant or worthwhile for our applications and thus a simple and general scheme using always the same value for s is preferred. For case b) we find that the DBCSR-TAS algorithm performs better by a factor of 4 compared to the DBCSR limit. For this case s_opt is indeed the measured optimum and we conclude that our performance model based on communication cost estimates holds in practice. We note that neither the TAS limit nor the DBCSR limit exhibit good performance for the general case, justifying the choice of a hybrid algorithm that combines the two algorithms instead of relying on the DBCSR-internal implementation of Cannon's algorithm [74] or of the TAS limit [86].

Next we test the batching scheme that reduces memory usage at ideally low overhead in execution time. A memory reduction of n_mem can be achieved by splitting both indices μ and σ in Eq. (4.39) into n_mem batches, performing the n^2 contractions consecutively. As shown in Fig. 4.6 a memory reduction by a factor of 8 can be achieved at only 50% overhead in terms of execution time.

Of more practical relevance is an experiment in which the number of cores is reduced reciprocally to the number of batches such that the memory used per core stays approximately constant. This shows the capability of simulating larger systems within given memory constraints. This experiment is a part of the strong scaling benchmark Fig. 4.7 in the range between 16 and 512 nodes, where a batching of $n_\mathrm{mem} = 32$ is applied for 16 nodes. The batching is reduced linearly with increasing number of nodes until no batching is applied for 512 nodes. Surprisingly roughly the same performance is obtained on 16 nodes as on 512 nodes (73% relative to the best performance measured), suggesting that the overhead for heavy batching for 16 nodes is comparable to the effect of non-ideal

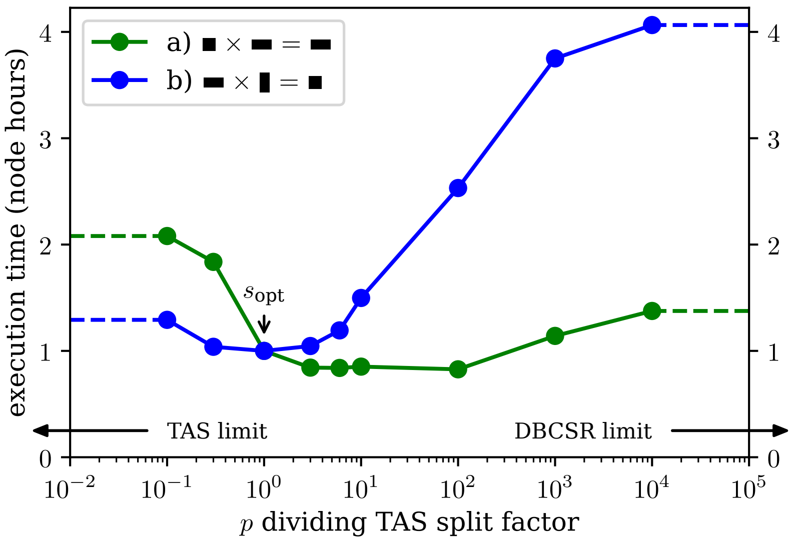


Figure 4.5: Comparing total execution time for different choices of split factors s for a system of 128 water molecules on 3072 compute cores. The slit factor s is controlled by the parameter p so that $s = s_{\text{opt}}/p$ with s_{opt} the predicted split factor minimizing total communication volume. Case a) with matrices of occupancies $5.3 \cdot 10^7, 3.8 \cdot 10^9, 7.0 \cdot 10^{10}$ corresponds to the situation that the result matrix is the largest and case b) with occupancies $3.3 \cdot 10^{10}, 7.0 \cdot 10^{10}, 3.0 \cdot 10^8$ to the contrary situation that the result matrix is the smallest. The two limits marked with dashed lines correspond to the DBCSR limit equivalent to Cannon's algorithm and the TAS limit where all communications are done in the TAS library layer. The hybrid algorithm exploiting both levels of parallelism at $p = 1$ outperforms both limits, most importantly the speedup over Cannon's algorithm is a factor of 4 for case b).

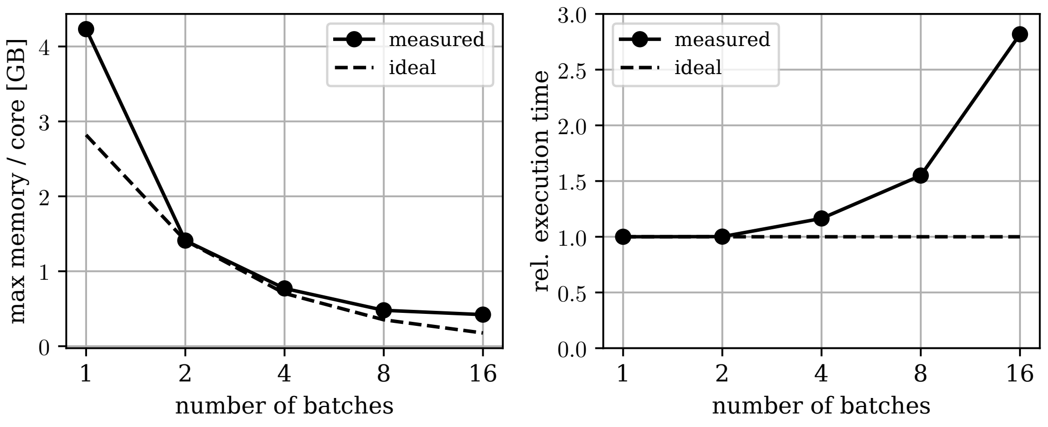


Figure 4.6: Memory savings (left panel) and overhead in execution time (right panel) by performing tensor contractions in n_{mem} batches for a system of 128 water molecules on 1536 compute cores. Ideally the relative memory reduction should be $1/n_{\text{mem}}$ and the execution time should stay constant.

parallel scaling for 512 nodes. We conclude that the batching scheme allows a memory reduction of up to a factor of 16 without compromising performance by more than 10%. Real strong scaling is demonstrated in the range between 512 nodes (6144 cores) and 2048 nodes (24576 cores), showing a steady drop of performance by roughly 20% in each step doubling the number of nodes.

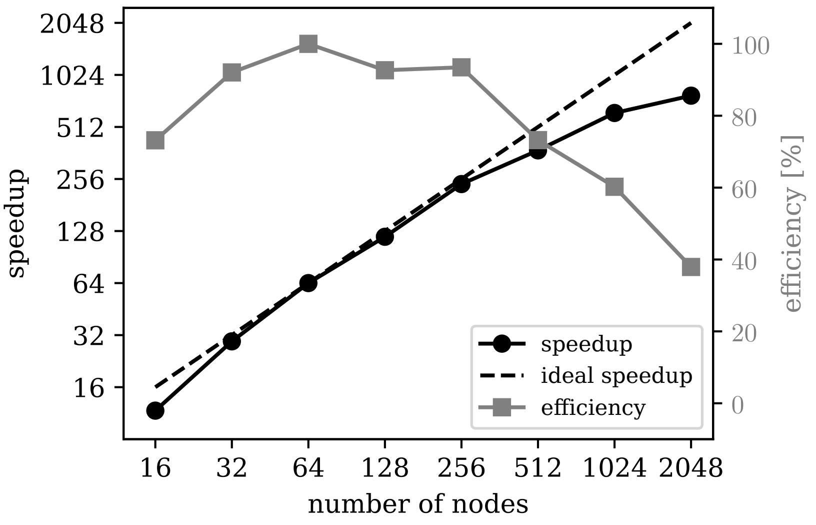


Figure 4.7: Strong scaling experiment for 256 water molecules. The number of batches was set to 32 for the lowest node count (16 nodes) and then decreased linearly with the number of nodes up to 512 nodes. The deficiencies in scaling towards lower node counts is thus attributed to an overhead associated with heavy batching and does not reflect the parallel performance. Real strong scaling (without batching) is demonstrated in the range between 512 and 2048 nodes.

Due to the use of Gaussian basis functions sparsity is an approximation controlled by the filter threshold ϵ_{filter}, discarding blocks with a Frobenius norm $< \epsilon_{\text{filter}}$. Both performance and accuracy of a calculation are affected by ϵ_{filter} as shown in Fig. 4.8. The effect on accuracy is system-dependent in electronic structure methods and systems with a more delocalized electronic structure need tighter thresholds. For an algorithm composed of multiple contractions of different shapes and sparsities the effect of ϵ_{filter} on the result may vary for each contraction. A system-specific optimization of ϵ_{filter} adapted to the required accuracy for each type of contraction is thus recommended.

Sparsity is best exploited by defining blocks of heterogeneous sizes, each block corresponding to a bunch of functions centered at the same position. Even more sparsity is exposed if the functions belonging to the same center are sorted with respect to their widths such that each block contains functions of similar extent. Smaller block sizes en-

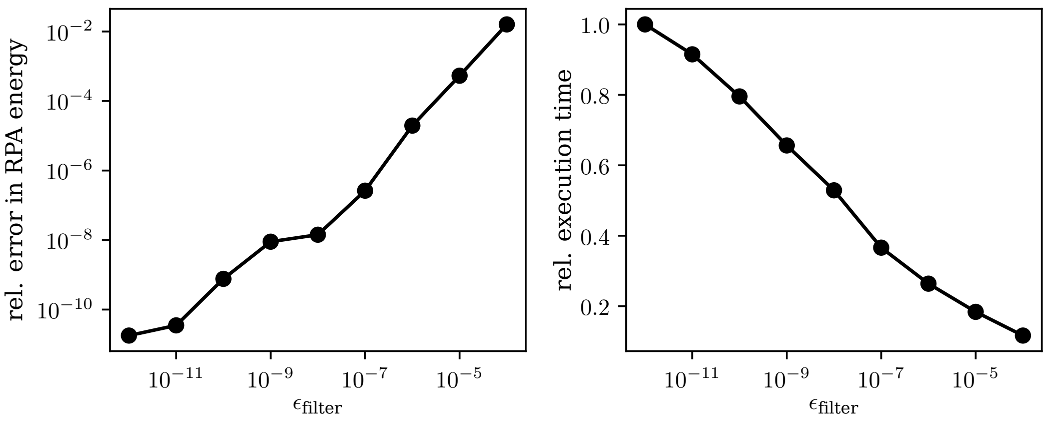


Figure 4.8: Convergence of the RPA energy with ϵ_{filter} controlling the truncation of small tensor elements (left panel) and the effect on computational cost (right panel). The energy converges quickly such that an accuracy gain of 2–3 orders of magnitude comes at a cost of only a factor of 2 in execution time. The system is 128 water molecules.

able a more refined screening, thus reducing the tensor occupancy. Small blocks however decrease the arithmetic intensity and decrease floating point performance. After defining elementary blocks with functions of the same width, block sizes smaller than some minimum size are combined into larger blocks and the minimum block size is optimized for performance. Execution time as a function of minimum block size has been measured for two different systems and as shown in Fig. 4.9, a minimum size of 5 is a reasonably good default.

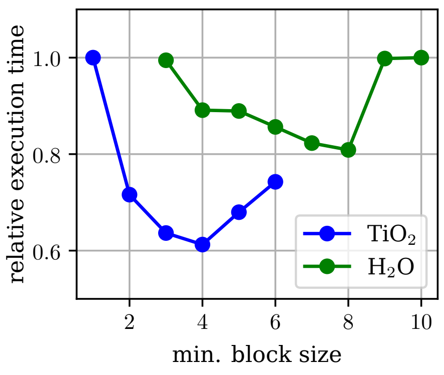


Figure 4.9: Effect of the minimum tensor block size (in each dimension) on performance for 2 different systems. Each elementary block contains exactly one set of Gaussian basis functions sharing the same exponents.

In contrast to the DBCSR matrix library where performance gains were reported from coalescing atomic blocks [71], the tensor library has best performance if subatomic blocks on the level of single basis sets are exploited. This behaviour is due to the higher rank of tensor blocks: if s is the minimum block size in one dimension, tensor blocks of rank 3 are of minimum total size s^3, much larger than the minimum matrix block size s^2, making a smaller default for s less problematic for tensors.

More insights into the overall performance and scalability of the tensor library can be obtained by profiling the three library layers for different system sizes as depicted in Fig. 4.10. The amount of time spent in each layer stays approximately the same when increasing system size, where about 70% (H_2O) or 80% (TiO_2) is spent in the DBCSR core library. The larger portion of time spent in the Tensor & TAS layers for H_2O is due to this system being sparser than TiO_2, so that less time is spent in computations. The overhead of data exchange in the tensor layer is small but not insignificant ($< 20\%$ for H_2O and $< 10\%$ for TiO_2), justifying our careful optimizations to avoid tensor redistributions. The vast majority of time spent in the DBCSR core library confirms that the complex tensor abstractions based on on-the-fly data and index conversions between Tensor, TAS and DBCSR representations do not impede overall performance of the library.

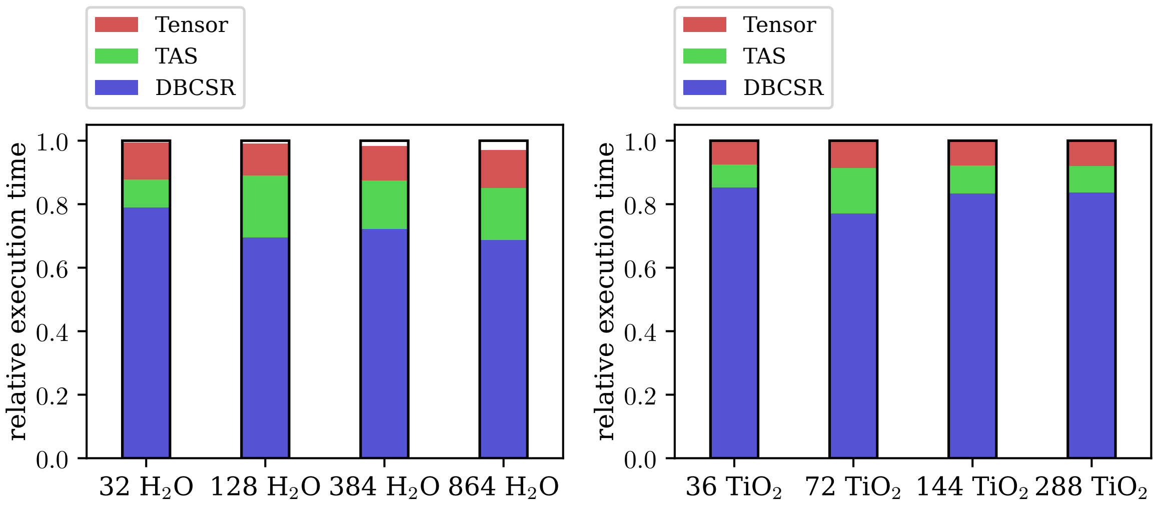


Figure 4.10: Profiling of the 3 layers of the DBCSR tensor library in dependence of the system size for 2 different systems (left panel: H_2O, right panel: TiO_2). The time spent in the *Tensor* layer corresponds to the redistribution to bring tensors into a contraction-compatible layout. The *TAS* layer takes care of the communication steps optimized for tall-and-skinny matrices, effectively reducing the amount of communication in the *DBCSR* layer. The *DBCSR* layer performs parallel multiplication of submatrices on subgroups and executes the local multiplication.

Chapter 5

Results for Low-Scaling RPA, GW & HFX

Promising applications of the DBCSR tensor library are RI-based formulations of Hartree-Fock Exchange, RPA and GW in the electronic structure code CP2K. The main root for sparse tensors is the use of a local RI metric (preferrably the overlap metric) leading to sparse 3-center integral tensors. Low-scaling RPA and GW were initially developed by Jan Wilhelm [2, 3] who provided a first implementation without the comfort of having a general sparse tensor contraction library at hand. The initial implementation was based on the DBCSR matrix library and tensor operations were implemented as static code specifically developed and optimized for each expression of Eq. (2.61). The initial implementation already contained the essential concepts that inspired the development of a general tensor library, including the concept of reducing communications by splitting large tall-and-skinny matrices and memory reduction by performing contractions in batches. However the lack of a tensor abstraction made the code very cumbersome to develop, optimize and maintain. The development of the DBCSR tensor library can thus be seen as a generalization and systematic optimization of the original low-scaling RPA code [2].

The new implementation is stripped down to a translation of mathematical expressions to operations provided by the DBCSR tensor API, only requiring a few hundred lines of code, all performance optimizations being internal to the DBCSR tensor library and automatic. The effort of providing an abstract tensor API does not only facilitate high-level code development but should also improve performance because important optimizations require a generic approach and could not have been provided in a direct implementation. These new optimizations include

- heterogeneous block sizes in all tensor dimensions that can be chosen arbitrarily adapted to the structure of basis sets.
- automatic optimization of the parallelization strategy minimizing communications.

Especially the second feature greatly improves ease-of-use, ensuring that calculations always run at best performance, in contrast to the initial implementation where optimal performance required running several preparatory calculations testing different choices of parameters.

A RI-based implementation of Hartree-Fock Exchange is based on the same tensor framework, demonstrating the transferability of the library to other tensor-based algorithms. Even though the RI-based algorithm is scaling worse with respect to system size than the direct evaluation of the Hartree-Fock energy, we demonstrate that dense solid state systems with a large basis are prohibitively expensive in the direct Hartree-Fock approach but feasible in the RI-based implementation.

5.1 Low-scaling RI-RPA and GW

We demonstrate improvements in the low-scaling algorithms for RPA and GW compared with the initial publications [2, 3] both in terms of performance and usability. The switch to the DBCSR tensor API facilitated the application of low-scaling RPA & GW and made performance more reliable since previously manually set parameters (mostly MPI group sizes) are now automatic.

The low-scaling RPA algorithm has been introduced in Sec. 2.6.2 and here we give a short summary of the equations, their scaling in memory and execution time, and their implementation in terms of operations provided by the DBCSR tensor library. The dominant part of the calculation is the evaluation of tensor contractions for each time grid point j

$$\begin{aligned}
&\text{Contraction} && \text{scaling: cost/memory} \\
M^{\text{occ}}_{\mu\sigma R}(\tau_j) &= \sum_{\lambda} (\lambda\sigma|R) D^{\text{occ}}_{\mu\lambda}(\tau_j) && O(N^2)/O(N^2) \qquad (5.1) \\
M^{\text{virt}}_{\mu\sigma T}(\tau_j) &= \sum_{\nu} (\mu\nu|T) D^{\text{virt}}_{\nu\sigma}(\tau_j) && O(N^2)/O(N^2) \qquad (5.2) \\
P_{RT}(\tau_j) &= \sum_{\mu\sigma} M^{\text{occ}}_{\mu\sigma R}(\tau_j) M^{\text{virt}}_{\mu\sigma T}(\tau_j) && O(N^2)/O(N^2) \qquad (5.3)
\end{aligned}$$

$$Q_{PQ}(\tau_j) = \sum_R K_{RP} \sum_T K_{TQ} P_{RT}(\tau) \qquad O(N^3)/O(N^2) \tag{5.4}$$

This can be translated to high-level code based on the DBCSR tensor API according to the pseudo code given in Alg. 4. The actual implementation performs the contractions in batches by splitting the atomic orbital indices μ, σ into n_{mem} batches each.

Algorithm 4 Implementation of low-scaling RPA in terms of the operations *Contract* & *Copy* of the DBCSR tensor library

```
$M^{\text{occ}}(R\sigma, \mu) \leftarrow (R\sigma, \lambda) \times D^{\text{occ}}(\lambda, \mu)$          ▷ Contract
$M^{\text{occ}}(R, \mu\sigma) \leftarrow M^{\text{occ}}(R\sigma, \mu)$          ▷ Copy
$M^{\text{virt}}(T\mu, \sigma) \leftarrow (T\mu, \nu) \times D^{\text{virt}}(\nu, \sigma)$          ▷ Contract
$M^{\text{virt}}(T, \mu\sigma) \leftarrow M^{\text{virt}}(T\mu, \sigma)$          ▷ Copy
$P(R, T) \leftarrow M^{\text{occ}}(R, \mu\sigma) \times M^{\text{virt}}(T, \mu\sigma)^{\text{T}}$          ▷ Contract
```

We compare performance of different implementations and configurations of RPA as implemented in CP2K:

- the canonical variant of RI-RPA based on RI with the Coulomb metric and a minimax quadrature for the frequency integration scaling as $O(N^4)$ [41]
- the legacy implementation of low-scaling RPA scaling effectively as $O(N^2)$ [2]
- the new tensor-based implementation of low-scaling RPA

The execution time of canonical RPA is dominated by dense matrix multiplications based on ScaLAPACK PDGEMM. We choose the best possible configuration for each RPA variant given the supercomputing power of the Piz Daint Cray XC50 machine at CSCS. Thus we run the canonical variant of RPA with a GPU-accelerated version of PDGEMM as provided by the COSMA library [87]. The COSMA library was chosen over other GPU-accelerated ScaLAPACK implementations since it reaches the highest percentage of the GPU peak performance on Piz Daint compared with other PDGEMM implementations [88].

Even though the DBCSR library has a sophisticated GPU backend [73] that is specifically optimized for small blocks of heterogeneous size, we found that better performance can be achieved for tensors by using the LIBXSMM [75] CPU backend. More investigations are needed to state if GPU performance can be improved. Generally speaking the basic unit of computation performed on a GPU is a stack, collecting small block multiplications of the same shape. Since GPU performance is best for large stacks, the number of different block sizes shouldn't be too large and best performance is achieved

method	system	basis	$(\alpha_{\min}, l)$	# grid points	$O(N^3)$ RPA ϵ_{filter} for M / P
RPA	H_2O	cc-TZV2P	$(0.15, 1)$	10	10^{-07} / 10^{-07}
RPA	TiO_2	cc-TZV2P	$(0.13, 2)$	10	10^{-08} / 10^{-07}
G_0W_0	GNR	aug-DZVP	$(0.11, 1)$	12	10^{-11} / 10^{-10}

Table 5.1: Parameters of all RPA/G_0W_0 calculations. Coordinates and basis sets are taken from the supplementary informations of [2, 3, 89]. We also list the most diffuse basis function with exponent α_{min} and its maximum l quantum number. The number of grid points refers to the minimax grids for imaginary time and frequency. The main input for low-scaling RPA are two filter thresholds, the first one applying to the tensors M^{virt} and M^{occ} Eq. (5.1) and the second one applying to the P matrix Eq. (5.3).

with large blocks. For matrix-based algorithms and atomic block sizes the number of different block sizes corresponds to the number of atom types N_a (e.g. $N_a = 2$ for H_2O). For tensor-based algorithms best performance was found by using subatomic blocks so that there are at least two different block sizes per atom or $2N_a$ distinct elementary block sizes. Tensors have blocks of at least rank 3 where 2 dimensions are mapped to a single matrix dimension. Thus we have $(2N_a)^2$ different combined block sizes or 16 distinct block sizes for H_2O which is inherently problematic for GPU performance. We conclude by stating that efficiently exploiting GPUs for the DBCSR tensor library is an open and challenging problem and here we rely on the CPU-based configuration of DBCSR.

The benchmarks are performed for the three different systems liquid water (H_2O) [2], bulk anatase (TiO_2) [89] and graphene nanoribbons (GNR) [3]. Both H_2O and TiO_2 are true bulk systems and system size was in increased regularly in all 3 dimensions in order to assess scaling with respect to system size. All calculations employ Goedecker-Teter-Hutter pseudopotentials and start from a PBE-based DFT wavefunction. The parameters for the 3 calculations are listed in Tab. 5.1. A full documentation of the parameters for each system size is given in Appendix C. The performance of the low-scaling algorithm depends strongly on the chosen filtering thresholds and we have converged these parameters to an error in RPA energy smaller than $5 \cdot 10^{-6}$ E_h per electron for all systems (including the error due to the overlap instead of the Coulomb RI metric). A better accuracy could be achieved by using an attenuated or truncated Coulomb metric, however this would reduce sparsity and increase computational costs by possibly a factor of 2.

The node count has been adapted to each method, using the minimum number of nodes required in order to have sufficient amount of memory. For canonical RPA the required memory scales as $O(N^3)$ and the node count has been scaled accordingly. For

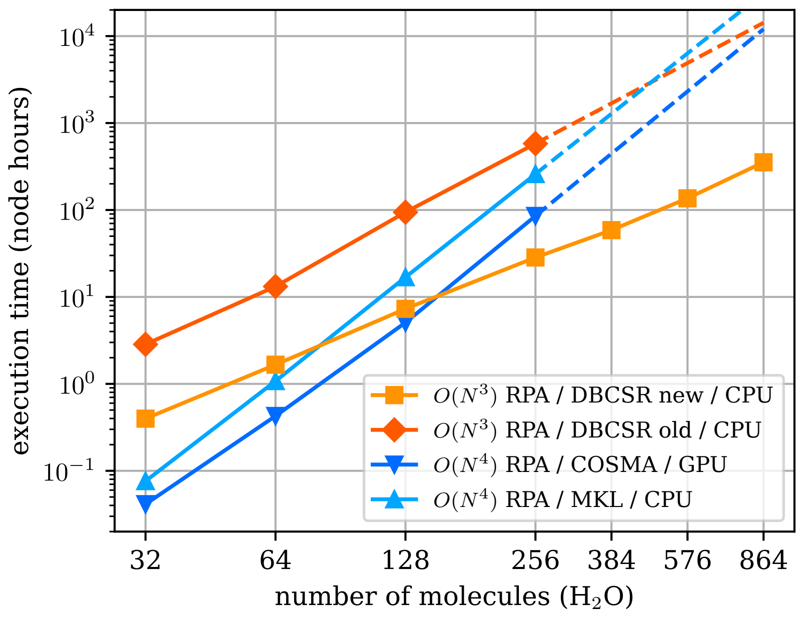


Figure 5.1: weak scaling benchmark comparing different implementations of RPA for different system sizes of bulk water. The two low-scaling $O(N^3)$ calculations refer to different implementations of the same algorithm, the old implementation based on the DBCSR matrix library and the new implementation based on the tensor API of the DBCSR library. The canonical $O(N^4)$-scaling RPA implementation is mostly PDGEMM-based and we consider an MKL-based configuration and a 3-times faster GPU-accelerated configuration using the COSMA library. The crossover between the best canonical and the best low-scaling implementation of RPA lies shortly above 128 molecules.

low-scaling RPA the memory scales as $O(N^2)$ and by using the feature of batched contraction the node count can be kept relatively low even for large systems. As previously shown (Fig. 4.7) 256 water molecules can be calculated on only 16 nodes with the low-scaling implementation where for the same system 256 nodes are required for canonical RPA with MKL. The COSMA library seems to rely on excess memory to minimize communication so that a 2–4 times higher node count seems to be required compared with MKL. Ultimately memory constraints limit the largest systems within reach of canonical RPA where we were restricted to use a maximum node count of 2400 nodes.

The benchmark results for the 3 systems are shown in Fig. 5.1, Fig. 5.2 and Fig. 5.3. The effective scaling of the low-scaling RPA variant is $O(N^2)$ for all systems, comprising all sparse tensor contractions that strongly dominate for not too large systems. The $O(N^3)$ scaling Eq. (5.4) starts to show only for the largest water system as analyzed in Fig. 5.4. The comparison with the legacy implementation of low-scaling RPA for water in Fig. 5.1 reveals that the new implementation is more than one order of magnitude

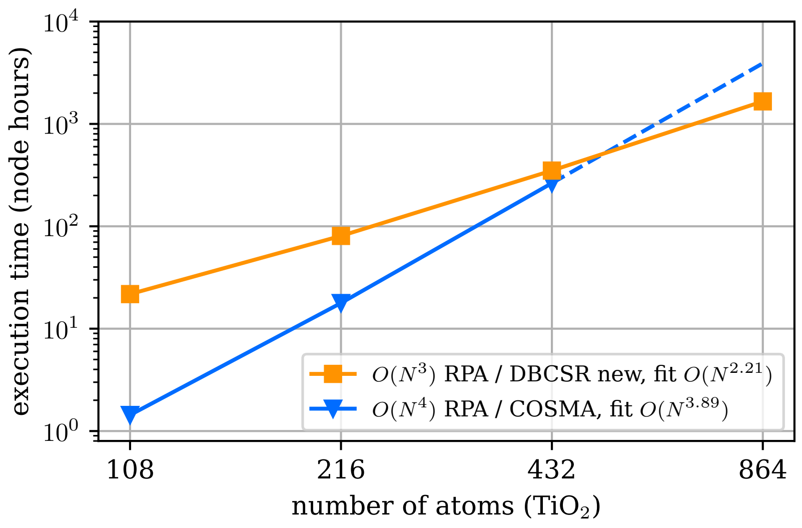


Figure 5.2: weak scaling benchmark comparing low-scaling $O(N^3)$ with canonical $O(N^4)$ implementation of RPA for bulk anatase. This system is less favorable than water for low-scaling due to a more delocalized electronic structure, requiring a tighter ϵ_{filter} and exhibiting no sparsity in the pseudo-density matrices. Nevertheless the low-scaling variant is roughly 2.4 faster than the extrapolated costs of the $O(N^4)$ RPA for the largest system. More importantly it is questionable whether 864 atoms could be run with the canconical RPA due to the large memory footprint.

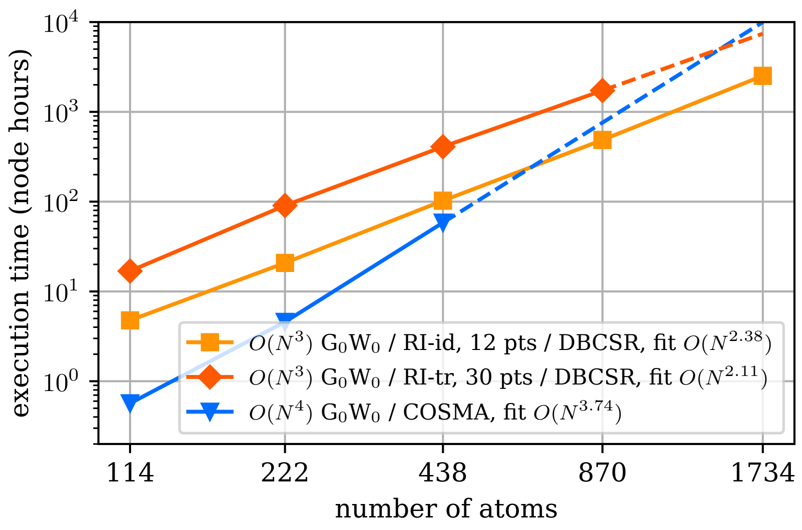


Figure 5.3: weak scaling benchmark comparing low-scaling $O(N^3)$ with canonical $O(N^4)$ implementation of G_0W_0 for graphene nanoribbons of varying horizontal length. Due to the limited accuracy (0.1 eV) of the calculation with the overlap metric (RI-id), a more accurate calculation has been performed with the truncated Coulomb metric (RI-tr) and 30 minimax points. Both variations of low-scaling G_0W_0 scale to larger system sizes than the canonical variant which is restricted to 438 atoms due to memory constraints.

faster, adding the word of caution that the subpar scaling w.r.t. system size of the legacy implementation suggests that it was not run with optimal choice of parameters. By taking the benchmark from [2] as reference, the speedup of the new implementation can be estimated more realistically by only considering the crossover with the CPU-based canonical RPA, suggesting a speedup by a factor of 6 to 7. An empirical comparison for the two other systems reveals a speedup of a factor 2.3 for TiO_2 and a factor 2.8 for the graphene nanoribbon system. The more pronounced speedup for water can be attributed to water being more sparse than the other two systems so that the optimizations of block sizes and communications have a stronger effect on the total performance.

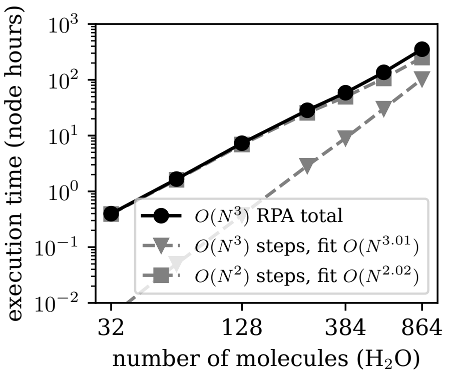


Figure 5.4: Assessment of the system-size scaling of low-scaling RPA. All sparse tensor based operations are scaling as $O(N^2)$ and strongly dominating, only for the largest system the $O(N^3)$ scaling steps have a noticeable (though small) effect on the total time.

Comparing low-scaling vs. canonical RPA, the water system is highly advantageous for the low-scaling variant: already for systems larger than 128 molecules the low-scaling variant performs much better. Up to 864 molecules can be computed at affordable computational costs. Due to the onset of the $O(N^3)$ scaling step computational costs are expected to increase steeply for even larger systems. For TiO_2 and the graphene nanoribbon system the advantage of the low scaling algorithm is less pronounced in terms of computational costs because these systems require a tighter threshold for ϵ_{filter} and an even tighter threshold is required to converge the quasiparticle energies for G_0W_0.

The most limiting factor of the canonical RPA is memory scaling as $O(N^3)$ so that the maximum system that could be calculated was TiO_2 with 432 atoms / 3456 electrons on 2400 nodes with 64 GB of RAM each (which is the maximum allocation possible on Piz Daint). The low-scaling RPA has a much lower memory footprint due to a better $O(N^2)$ scaling and due to the batching feature which allows to reduce memory by a large factor. The theoretical memory consumption estimated from the largest tensor for both RPA variants is illustrated in Fig. 5.5. For all systems considered here the low-scaling RPA could be extended to systems 2- to 4-times larger than the maximum system size feasible in the canonical variant, eventually being limited not by memory but by the computational costs. The largest Graphene nanoribbon system consisting of 1734 atoms is clearly out of reach for canonical RPA - if we ignore the overhead of COSMA in terms of memory and extrapolate the MKL calculation of 222 atoms on 32 nodes, 1734 atoms would require 15000 Piz Daint nodes.

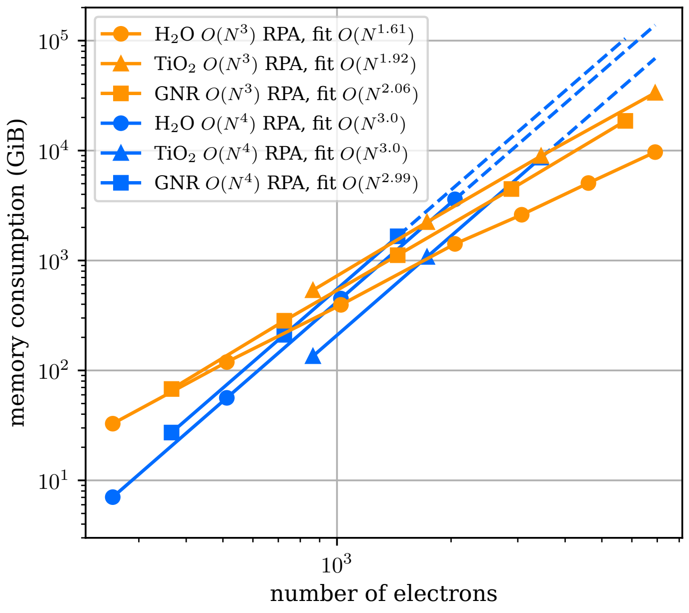


Figure 5.5: Ideal RPA memory consumption approximated by the number of elements in the largest tensor. Actual memory consumption may be larger but within a constant factor of these estimates. The fitted scaling laws match closely the theoretical expectation of $O(N^3)$ for canonical RPA and $O(N^2)$ for low-scaling RPA (water being sparser due to decaying pseudodensity matrices). Memory is the most limiting factor for canonical RPA and the low-scaling variant relaxes the memory requirements for large systems. The batching feature of the low-scaling variant was not taken into account here and allows to further reduce memory by a large factor.

For *GW* calculations the choice of the overlap metric and a small number of grid points is a compromise on the accuracy of the quasiparticle energy levels as shown in Fig. 5.6: the induced error on the zigzag and transport gap is on the order of 0.1eV in comparison with a more accurate calculation using a truncation radius of 2Å and 30 grid points. The more accurate variant has been benchmarked as well in Fig. 5.3 and is a constant factor of 4 more expensive.

The theoretical scaling with system size of the low-scaling RPA is revealed by counting the number of floating point operations performed in DBCSR. The results in Fig. 5.7 suggest that the scaling is exactly $O(N^2)$ for TiO_2 and GNR which is equivalent to the expected scaling if the pseudodensity matrices are dense. For water the scaling shifts

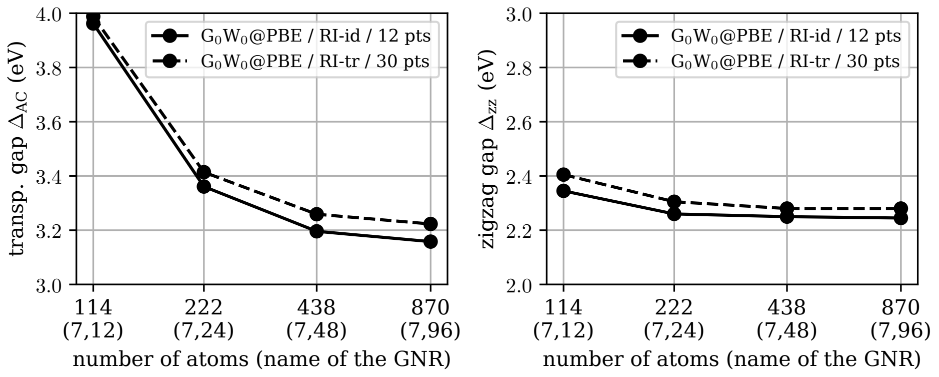


Figure 5.6: transport gap Δ_{AC} ([HOMO − 1] − [LUMO + 1]) (left panel) and zigzag gap (HOMO − LUMO) Δ_{zz} (right panel) for graphene nanoribbons of varying horizontal length. Both gaps saturate when increasing the ribbon length. The results were obtained with two different configurations of low-scaling G_0W_0: an exact calculation with 30 minimax points and a truncated Coulomb RI metric with truncation radius of 2Å, and a 4-times less expensive calculation with 12 minimax points and the identity RI metric. The difference in both gaps (Δ_{zz} and Δ_{AC}) is less than 0.1eV, demonstrating that meaningful results can be obtained already from the cheaper calculation.

slightly towards $O(N)$ for large system sizes due to the onset of sparse pseudodensity matrices. Note that this study includes only the FLOPs performed in DBCSR and does not include the FLOPs of the $O(N^3)$ operations.

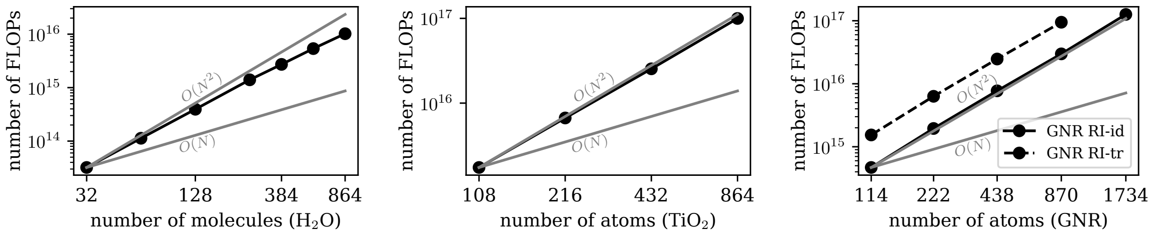


Figure 5.7: Scaling with system size of the total number of floating point operations of the dominating sparse tensor contractions. The expected scaling limits are depicted by grey lines. TiO_2 and GNR match the $O(N^2)$ limit (upper grey line) exactly. For H_2O the scaling evolves gradually into $O(N)$, however still being close to a $O(N^2)$ scaling for the system sizes considered.

The fitted scaling based on execution time of $O(N^{2.0})$ for water, $O(N^{2.2})$ for TiO_2 and $O(N^{2.4})$ for GNR suggests non-ideal weak scaling of DBCSR. This is more thoroughly evaluated in Fig. 5.8, showing the performance in terms of FLOP/s for the different system sizes. The fact that the performance regression is more pronounced for the two denser systems suggests that it does not have to do with sparsity but hints at limited scalability with number of processors. Indeed the total communication costs Eq. (4.32) reveal that if N_P is increased proportionally to N_3, the communication cost increases as $T = O(\sqrt{N_1}) = O(N) = O(\sqrt{N_\mathrm{P}})$ (assuming $N_1, N_2, N_3 = O(N^2)$). The same weak-scaling behaviour has already been reported for the DBCSR library in [71, 90]. Even if a communication-optimal matrix multiplication algorithm was used, the communication would increase as $T = O(N)$ Eq. (4.20) or $T = O(N^{2/3})$ Eq. (4.20), so that we conclude that the degradation in weak scaling is not due to the choice of algorithm and no evident solution exists to improve scaling with number of processors.

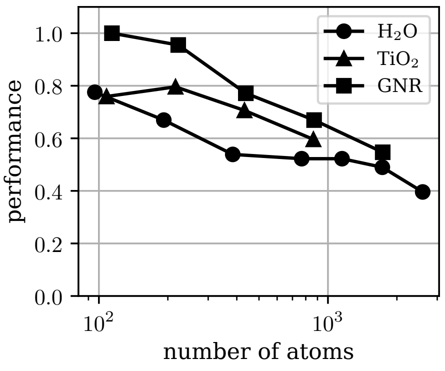


Figure 5.8: Performance (measured in FLOP/s, relative scale) for different system sizes. Performance gradually decreases towards larger system sizes so that the largest systems run at roughly half of the performance that is ideally expected.

5.2 RI Hartree Fock Exchange

The RI Hartree-Fock (RI-HFX) algorithm has been introduced in Sec. 2.6.1 and here we give a short summary of the equations, their scaling in memory and execution time, and their implementation in terms of operations provided by the DBCSR tensor library. The algorithm evaluates once per SCF procedure the contraction of the 3-center integrals with the matrix $\mathbf{K} = \mathbf{S}^{-1}\mathbf{V}\mathbf{S}^{-1}$

$$\begin{array}{cc} \text{Contraction} & \text{scaling: cost/memory} \\ M^{(1)}_{Q\lambda\mu} = \sum_P (\mu\lambda P) K_{PQ} & O(N^2)/O(N^2) \end{array} \tag{5.5}$$

In each SCF step the Fock matrix $\Sigma^{\mathrm{x}}_{\mu\nu}$ is evaluated using the density matrix $\mathbf{P}$ of the current SCF step

$$\begin{array}{cc} \text{Contraction} & \text{scaling: cost/memory} \\ M^{(2)}_{Q\lambda\nu} = \sum_\sigma (Q\sigma\nu) P_{\lambda\sigma} & O(N^2)/O(N^2) \end{array} \tag{5.6}$$

$$\Sigma_{\mu\nu} = \sum_{Q\lambda} M^{(1)}_{Q\lambda\mu} M^{(2)}_{Q\lambda\nu} \qquad O(N^2)/O(N^2) \tag{5.7}$$

It is not evident that this contraction order is the most efficient possible and the following alternate formulations have been thoroughly tested:

- Contracting the 3-center integrals with the molecular orbital coefficients instead of the density matrix, resulting in smaller though dense tensor contractions (scaling as $O(N^3)$ instead of $O(N^2)$) that could be more efficiently accelerated using GPUs

$$(\mu n P) = \sum_{\sigma} c_{n\lambda}(\mu\lambda P)$$
$$M_{Qn\mu} = \sum_{P} (\mu n P) K^{1/2}_{PQ}$$
$$\Sigma_{\mu\nu} = \sum_{Qn} M_{Qn\mu} M_{Qn\nu}$$

- Contracting the same 3-center integral tensor $(Q\lambda\mu)$ with the $\mathbf{K}$ and $\mathbf{P}$ matrix

$$M^{(2)}_{Q\lambda\nu} = \sum_{\sigma} M^{(1)}_{Q\sigma\nu} P_{\lambda\sigma}$$
$$\Sigma_{\mu\nu} = \sum_{Q\lambda} (\mu\lambda Q) M^{(2)}_{Q\lambda\nu}$$

  so that the other tensor $(\mu\lambda Q)$ remains very sparse, allowing for a prescreening reducing the number of elements in $M^{(2)}_{Q\lambda\nu}$: only those pairs of (Q, λ) need to calculated in $M^{(2)}_{Q\lambda\nu}$ for which non-zero entries in $(\mu\lambda Q)$ exist. This way of imposing a sparsity structure is equivalent to the notion of *emergent sparsity* as discussed by Manzer et al. [64]

Though conceptually interesting, both alternate variants turned out to perform worse so that the approach presented as first seems to be the most preferrable within a sparse contraction scheme.

The implementation of RI-HFX is analogous to low-scaling RPA and summarized in terms of pseudocode in Alg. 5. The actual implementation performs the contractions in batches by splitting the indices λ, Q into n_{mem} batches each.

Parameters for the calculations are listed in Tab. 5.2. A full documentation of the parameters for each system size is given in Appendix C. For both variants of Hartree-Fock the integral screening and filter thresholds have been set to the loosest values still

Algorithm 5 Implementation of RI-HFX in terms of the operations *Contract* & *Copy* of the DBCSR tensor library

```
$M^{(1)}(Q,\lambda\mu) \leftarrow K(Q,P) \times (P,\lambda\mu)$                    ▷ Contract
$M^{(1)}(\lambda Q,\mu) \leftarrow M^{(1)}(Q,\lambda\mu)$                           ▷ Copy
$M_c^{(1)} \leftarrow \text{compress}(M^{(1)}(\lambda Q,\mu))$                      ▷ Compress full tensor
while SCF not converged do
    $M^{(2)}(\nu Q,\lambda) \leftarrow (\nu Q,\sigma) \times P(\sigma,\lambda)$     ▷ Contract
    $M^{(2)}(\lambda Q,\nu) \leftarrow M^{(2)}(\nu Q,\lambda)$                      ▷ Copy
    $M^{(1)}(\lambda Q,\mu) \leftarrow \text{decompress}(M_c^{(1)})$                ▷ Decompress batch-wise
    $\Sigma^x(\mu,\nu) \leftarrow M^{(1)}(\lambda Q,\mu)^T \times M^{(2)}(\lambda Q,\nu)$   ▷ Contract
end while
```

ensuring stable SCF convergence to a target accuracy of 10^{-7}. For each calculation a full SCF procedure has been performed, restarted from a PBE-based wavefunction. The error induced by the RI approximation and the overlap metric is smaller than $5 \cdot 10^{-6} E_h$ per electron for all systems.

Benchmarks on 2 different systems are shown in Fig. 5.9 and Fig. 5.10: water representing a sparse system with localized electronic structure for which the combined Schwarz and density matrix based integral screening Eq. (2.46) are expected to work very well, and bulk anatase TiO_2 for which density matrix based screening fails. For H_2O we employ a TZVP basis set that was specifically optimized for direct HFX (favoring a small number of contractions per set). For TiO_2 the same cc-TZV2P basis was used as for RPA, which is problematic for direct HFX, due to the presence of a diffuse function with quantum number $l = 2$ in the basis of Titanium, causing a large number of integrals to be calculated. In order to assess the influence of basis set quality the basis set has been reduced to a TZV2P basis by removing the respective polarization functions. Additionally the $l = 2$ quantum number of the most diffuse primitive has been reduced

					RI-HFX	direct HFX
method	system	basis	(α_{min}, l)	Cutoff radius	ϵ_{filter} / $\epsilon_{\text{storage}}$	$\epsilon_{\text{schwarz}}$ / $\epsilon_{\text{storage}}$
HFX	H_2O	TZVP	$(0.15, 1)$	4.0	10^{-07} / 10^{-06}	10^{-06} / 10^{-07}
HFX	TiO_2	cc-TZV2P	$(0.13, 2)$	4.5	10^{-08} / 10^{-06}	10^{-09} / 10^{-10}
HFX	TiO_2	TZV2P	$(0.13, 1)$	4.5	10^{-08} / 10^{-06}	10^{-09} / 10^{-10}

Table 5.2: Parameters of all Hartree-Fock calculations. The main input for RI-HFX is a threshold ϵ_{filter} for sparse tensor contractions. For direct HFX $\epsilon_{\text{schwarz}}$ is the threshold for integral screening. Both implementations also have a threshold $\epsilon_{\text{storage}}$ for compression. All parameters have been optimized by trying out the largest value which still ensures stable SCF convergence. We also list the most diffuse basis function with exponent α_{min} and its maximum l quantum number.

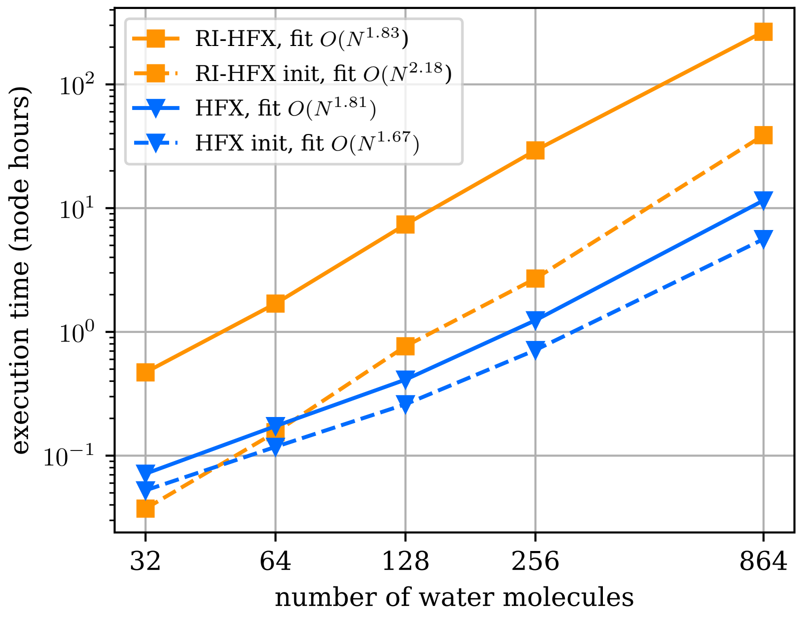


Figure 5.9: weak-scaling benchmark comparing the RI-HFX method with direct HFX: continuous lines for the fully converged SCF procedure, dashed lines for initializations taking place only in the first SCF iteration. The RI-based algorithm scales slightly worse with system size and is roughly one order of magnitude slower than the direct HFX. Performance of direct HFX declines for the largest system due to the memory-driven necessity of using threading (3 threads per rank).

to $l = 1$. The error in the Hartree-Fock energy induced by this modification of the basis set is as small as $7 \cdot 10^{-5}$ E_h per electron.

For water, the RI-HFX is roughly one order of magnitude more expensive for all system sizes. Direct HFX does not show ideal $O(N)$ scaling which is memory-driven by the replication of the density matrix. More precisely, the replication of the density matrix requires either more nodes or a hybrid parallelization employing a larger number of threads, reducing the overall performance. The effective scaling of RI-HFX is measured as $O(N^{1.8})$, close to the theoretical scaling of $O(N^2)$ and slightly worse than the scaling of direct HFX.

The performance comparison between RI-HFX and direct HFX depends on the number of SCF steps performed because for direct HFX, the dominant part of the calculation is performed only once in the first SCF step, whereas in RI-HFX the first step is the most expensive but all subsequent steps still need to perform relatively costly tensor contractions. Typically the subsequent steps each take roughly 1/3 of the time of the first SCF step in RI-HFX. It is interesting to also compare the time for initializations only taking place in the first SCF step (depicted by dashed lines in Fig. 5.9), corresponding to Eq. (5.5) for RI-HFX and to the integral calculation for direct HFX.

For TiO_2 Fig. 5.10 and the larger cc-TZV2P basis, direct HFX is limited by memory so that the integrals for systems larger than 216 atoms could not have been stored in core (given 2400 compute nodes with 64 GB each). This makes the full SCF procedure unaffordable so that only the first SCF step has been performed for direct HFX (depicted by dashed lines). The RI-HFX method requires much less memory (the largest calculation of 864 atoms being feasible on 512 compute nodes). If unlimited amount of memory was available, RI-HFX would still perform better in terms of execution time, taking into account imperfect strong scaling of direct HFX which does not show in Fig. 5.10 because a much smaller number of nodes have been used than necessary to store the integrals in-core.

For the reduced TZV2P basis, direct HFX performs better by more than a factor of 7 compared with the larger cc-TZV2P basis, even though the basis is smaller in size by only 25%. Direct HFX is very sensitive to the features of the basis and the costs increase rapidly with diffuse functions and functions with large values for l. For RI-HFX the reduced TZV2P basis leads to a speedup by a factor of only 2.3. This corresponds to a scaling of $O(N^3)$ with respect to basis set size and not to the expected $O(N^4)$ scaling, which can be attributed to the fact that the RI basis has not been optimized for the smaller basis (only the functions with largest l have been removed). Using an optimized RI basis, RI-HFX would perform slightly better. RI-HFX and direct HFX have similar computational costs for the reduced TZV2P basis. An advantage of RI-HFX is still the reduced memory footprint so that the largest calculation could be done on only 256 nodes, whereas direct HFX required 2400 nodes for the largest system.

A comparison of the theoretical memory consumption (estimated by the number of elements in the largest tensor) is given in Fig. 5.11. The memory use of direct HFX is highly system dependent because the screening depends on the decay of the density matrix and the number of diffuse basis functions. H_2O is thus favorable for direct HFX, exhibiting a linear memory scaling in contrast to the quadratic scaling for the case of TiO_2. For RI-HFX, memory consumption is very similar for the two systems because the sparsity of the largest tensor $M^{(1)}$ doesn't benefit from a sparse density matrix. For H_2O the memory consumption of RI-HFX is therefore much higher compared with direct HFX but for TiO_2, the RI-HFX approach uses roughly 2 orders of magnitude less memory than direct HFX.

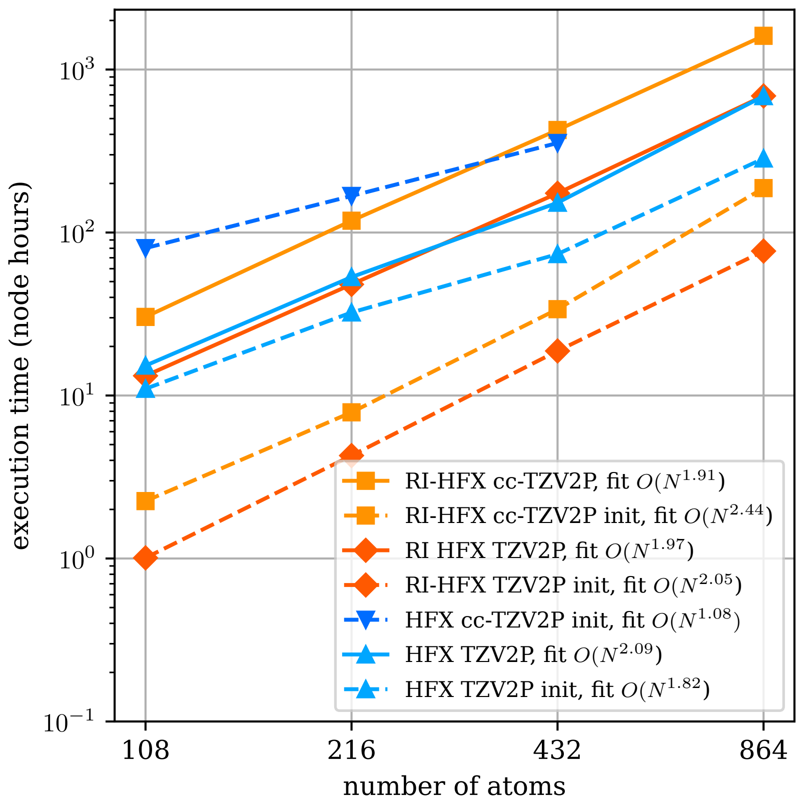


Figure 5.10: weak-scaling benchmark comparing RI-HFX with the direct implementation of HFX for anatase with a cc-TZV2P basis and a smaller TZV2P basis. Continuous lines for the fully converged SCF procedure, dashed lines for initializations taking place only in the first SCF iteration. The RI-based implementation performs better for cc-TZV2P basis and, more importantly, has a smaller memory footprint so that the largest system of 864 atoms can be calculated on 512 compute nodes. For direct HFX, the ERI storage requires much more memory and the limiting system size for the cc-TZV2P basis is already reached for 216 atoms (given 2400 compute nodes with 64GB of RAM each). For this reason, ERIs are not stored in-core and timings for only the first SCF step are reported (performed on a smaller number of nodes). For the reduced TZV2P basis, direct HFX becomes affordable for up to 864 atoms and both variants of HFX have roughly the same cost (direct HFX requiring 2400 nodes compared to 256 nodes for RI-HFX)

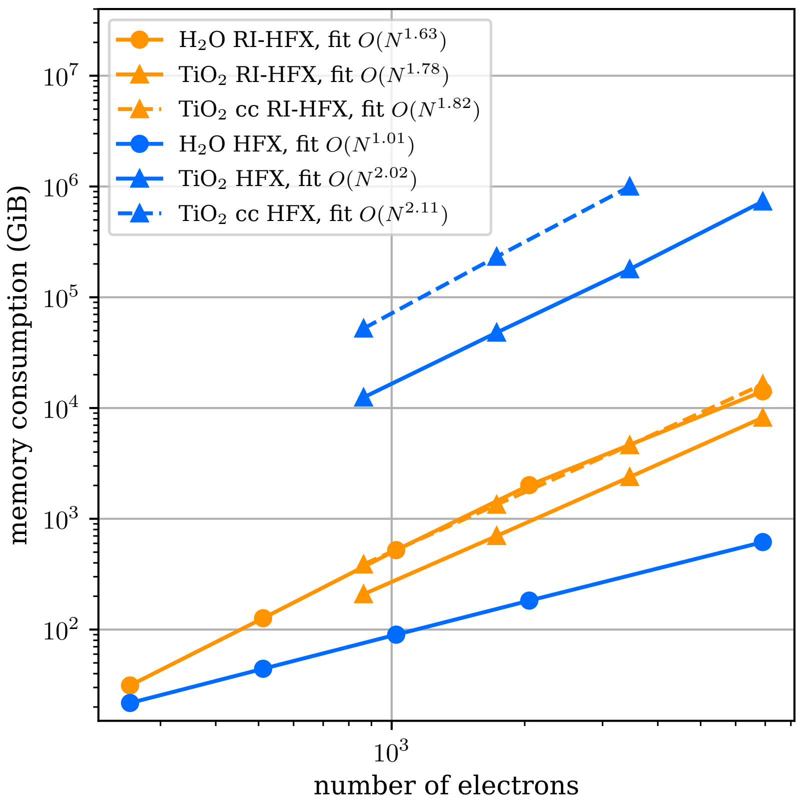
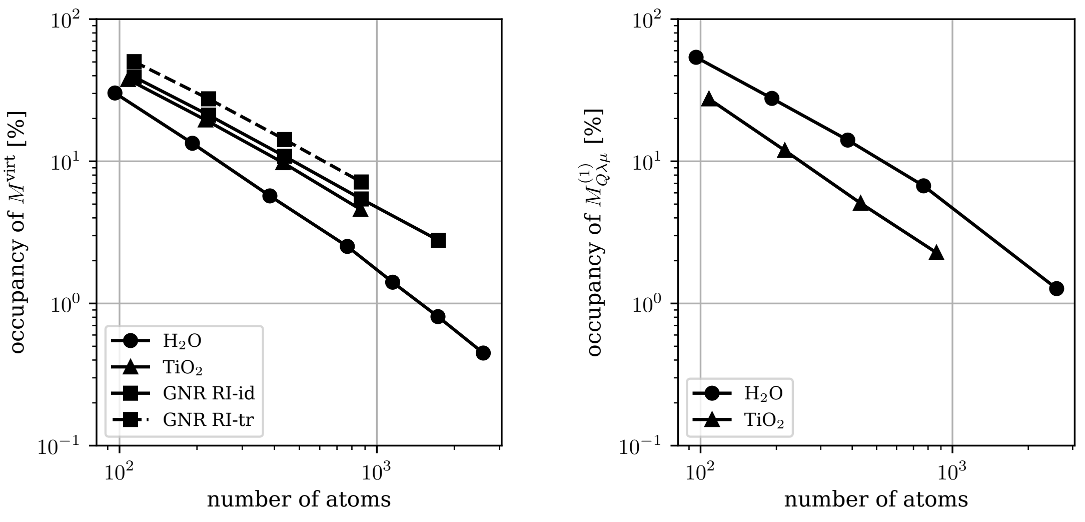


Figure 5.11: Comparison of memory use between RI-HFX and direct HFX where memory was approximated by the number of tensor elements to be stored (4-center ERIs for direct HFX and tensor $M^{(1)}$ for RI-HFX). Both algorithms use the same compression technique so that the comparison by counting tensor elements is reasonable. Due to density-matrix-based screening direct HFX memory scales linearly for water. For TiO2 the scaling is quadratic due to a non-decaying density matrix. The memory use of RI-HFX always scales quadratically.

5.3 Conclusion and Outlook

The benefit of exploiting an efficient tensor-based implementation of sparse linear algebra has been demonstrated for three electronic structure methods: Hartree-Fock Exchange, RPA and GW. The algorithms make efficient use of the locality of basis functions and the sparsity of 3-center integrals due to a RI expansion with a local metric. The overlap metric is the metric of choice for highest sparsity even if it is known to be less accurate than the Coulomb metric. The error induced by RI and the overlap metric is less than $5 \cdot 10^{-6} E_h$ per electron which is acceptable and is still small compared to orbital basis set errors. For GW the overlap metric is more problematic due to errors on the order of 0.1 eV for the quasiparticle energy levels which can be systematically improved by using a truncated Coulomb metric.

Making use of tensor sparsity to reduce the number of operations performed is key to the low-scaling behaviour of the algorithms presented here. Without the use of sparsity, the methods presented here would scale as $O(N^4)$ instead of $O(N^2)$. The occupancy of the largest tensors in both low-scaling RPA and RI-HFX is depicted in Fig. 5.12, revealing typical occupancies of less than 10% for the largest systems considered.



(a) RPA **(b)** HFX

Figure 5.12: Occupancy of the largest tensor, illustrating the effect of increasing sparsity and the importance of a sparse tensor library. For low-scaling RPA and very sparse systems (such as H_2O) occupancy goes down below 1%, otherwise occupancy between 1% and 10% is typical for large systems.

For RPA and GW, the reduction of memory and computational costs extends the applicability of these methods to system sizes that are clearly out of reach within a canonical formulation. This benefit has been demonstrated not only for sparse systems of linear or planar extension and/or with a large band gap, but also for bulk crystalline systems by the example of anatase. The comparison of low-scaling vs. canonical RPA has been performed with a configuration most favorable for canonical RPA, on Cray XC50 nodes maximizing the performance of canonical RPA by exploiting GPUs, even though the low-scaling implementation can currently not take advantage of GPUs. The benefit of low-scaling RPA would be even more pronounced (by a factor of 3) on CPU machines. It is not evident how sparse tensor contractions can efficiently be accelerated on GPUs since the tensor layout is optimized for small heterogeneous block sizes. Sacrificing some of the sparsity in favor of more homogeneous block sizes will probably not lead to overall better performance, even if GPUs could be exploited more efficiently.

For Hartree-Fock Exchange, the benefits of using an RI-based approach is less pronounced due to the already very efficient integral screening, exploiting sparsity much better than a tensor-based algorithm. However the RI approximation reduces the number of integrals by a large amount so that RI-HFX can deal better with accurate basis sets (containing diffuse functions and large l quantum numbers) and for densely packed systems. The auxiliary density matrix method (ADMM) [91] is already a more efficient approach to make Hartree-Fock Exchange calculations available for problematic basis sets. ADMM is an approach that is orthogonal to RI and a combination of the two methods could be of interest, however the small basis sets employed in ADMM are presumably better-suited for direct HFX. RPA calculations in the EXX/RPA formalism [52] require a single-point Hartree-Fock calculation with a typically large and accurate basis for which the RI approach is always better suited. Independently of the basis, RI-HFX is always less costly than the corresponding low-scaling RPA calculation (for the largest system of TiO_2, a RI-HFX single point calculation costs roughly 20% of a low-scaling RPA calculation).

Finally, we emphasize again that the implementation of both RI-HFX and low-scaling RPA / GW are of low complexity and consist mainly to calls to the DBCSR tensor library, providing a complete set of operations in terms of which these algorithms have been implemented. Other tensor-based algorithms (dense or sparse) could be implemented with little effort and we hope that the availability of a sparse tensor library will fuel the use of sparse tensor algorithms within electronic structure theory.

Appendix A

MME Cutoff Calibration

The cutoff calibration is based on an error estimate of 2-center ERIs in dependence of the cutoff $\mathbf{G}_{\text{max}}$ which determines the fit range in which the minimax approximation is valid according to Eq. (3.15). The total accuracy is affected by two sources of errors:

1. Minimax error Δ_{mm}: the error within the fit range $|\mathbf{G}| \leq G_{\text{max}}$ caused by the approximative expansion of the potential into a sum of Gaussians

2. Cutoff error: the error outside of the fit range $|\mathbf{G}| > G_{\text{max}}$ where the minimax approximation falls off faster than the potential it approximates

The error should refer to normalized basis functions in order to approximate the error of the final representation of the ERIs in the normalized spherical harmonics Gaussian basis. It is more convenient to estimate the error based on Hermite Gaussians which should be comparable with the magnitude of the error in the spherical harmonics Gaussian basis if the Hermite basis Eq. (3.6) is first normalized.

For the following we assume an orthorhombic simulation cell so that $\mathbf{h}$ is diagonal and the reciprocal lattice vectors are mutually orhogonal. For the normalization of Hermite Gaussians we use their relation to Hermite polynomials [53]

$$H_{l,\alpha}(x) = e^{-\alpha x^2} \alpha^{l/2} H_l(\sqrt{\alpha} x) \tag{A.1}$$

The normalization constant N so that $1/N H_{l,\alpha}(x)$ is normalized can be calculated according to

$$N^2 = \int_{-\infty}^{\infty} dx \left[\left(-\frac{d}{dx} \right)^l \exp(-\alpha x^2) \right]^2$$

$$
\begin{aligned}
&=(-1)^l \int_{-\infty}^{\infty} dx \left[\left(\frac{d}{dx}\right)^{2l} \exp(-\alpha x^2)\right] \exp(-\alpha x^2) \\
&=(-1)^l \int_{-\infty}^{\infty} dx H_{2l,\alpha}(x) \exp(-\alpha x^2) \\
&=(-\alpha)^l \int_{-\infty}^{\infty} dx H_{2l}(\sqrt{\alpha}x) \exp(-2\alpha x^2) \\
&=(-\alpha)^l (2\alpha)^{-1/2} \int_{-\infty}^{\infty} dx H_{2l}\left(\frac{x}{\sqrt{2}}\right) \exp(-x^2) \\
&=\left(\frac{\alpha}{2}\right)^l \sqrt{\frac{\pi}{2\alpha}} \frac{(2l)!}{l!} \\
&=(2\alpha)^{(l-1/2)} \Gamma(l+1/2).
\end{aligned}
$$

where the solution of the last integral is given in [92], Eq. 22.13.17.

We proceed by an expression for the Minimax error Δ_{mm} which is the error of the minimax approximated integral $(a|a)_{\text{mm}}$ compared with the integral $(a|a)$ with the exact potential but $|\mathbf{G}| < G_{\max}$:

$$
\begin{aligned}
\Delta_{\text{mm}} = |(a|a) - (a|a)_{\text{mm}}| = \\
\frac{4\pi^4}{V} \left(\frac{1}{\alpha}\right)^3 \left(\prod_{k=1}^{3} \left[(2\alpha)^{-(l_k-1/2)} (\Gamma(l_k+1/2))^{-1}\right]\right) \\
\left| \sum_{|\mathbf{G}|<G_{\max}} \Delta_V(\mathbf{G}) \exp(-i\mathbf{G}\cdot(\mathbf{A}-\mathbf{B})) C_{2\mathbf{l},(2\alpha)^{-1}}(\mathbf{G}) \right|
\end{aligned}
\tag{A.2}
$$

with

$$
\Delta_V(\mathbf{G}) = \frac{1}{|\mathbf{G}^2|} - \sum_i \omega_i \exp(-\alpha_i \mathbf{G}^2) \tag{A.3}
$$

whose maximum absolute value is the minimax error $E_{\text{mm}} = \max_{|\mathbf{G}|<G_{\max}}(|\Delta_V(\mathbf{G}|)$. The absolute value of the lattice sum in Eq. (A.2) can be estimated as

$$
\begin{aligned}
|\ldots| &\leq E_{\text{mm}} \sum_{|\mathbf{G}|<G_{\max}} \left|C_{2\mathbf{l},(2\alpha)^{-1}}(\mathbf{G})\right| \leq E_{\text{mm}} 8 \prod_{k=1}^{3} \sum_{0<G<G_{\max}} C_{2l_k,(2\alpha)^{-1}}(G) \\
&\leq E_{\text{mm}} 8 \prod_{k=1}^{3} \left(\frac{h_{kk}}{2\pi} \int_0^{\infty} dG C_{2l_k,(2\alpha)^{-1}}(G) + C_{2l_k,(2\alpha)^{-1}}(\sqrt{2l_k\alpha}) - \delta_{l_k,0}\right) \\
&\leq E_{\text{mm}} 8 \prod_{k=1}^{3} \left(\frac{h_{kk}}{4\pi} \Gamma(l_k+1/2)(2\alpha)^{l_k+1/2} + (2l_k\alpha)^{l_k} \exp(-l_k) - \delta_{l_k,0}\right),
\end{aligned}
$$

where the lattice sum was split into two parts for $G < G_0$ and $G > G_0$ with $G_0 = \operatorname{argmax}(C_{2l_k,(2\alpha)^{-1}}(G)) = \sqrt{2l_k\alpha}$. Then we integrated over the two domains such that each sum is the lower Darboux sum of the respective integral. The additional summand is an upper bound for the term closest to G_0 which has not been included in the Darboux sums. If $l_k = 0$ then $G_0 = 0$ and the extra term is excluded by subtracting $\delta_{l_k,0}$.

Collecting all terms we find

$$\Delta_{\mathrm{mm}} \leq \frac{32\pi^3}{V} E_{\mathrm{mm}} \prod_{k=1}^{3} \left(\frac{h_{kk}}{2\pi} + \sqrt{\frac{2}{\alpha\pi}} \frac{l_k!}{(2l_k)!} [(4l_k)^{l_k} \exp(-l_k) - \delta_{l_k,0}] \right) \tag{A.4}$$

The term $l!/(2l)![(4l)^l \exp(-l) - \delta_{l,0}] < 3/4$ for all l as can be verified numerically. The estimate for the minimax error can thus be written as

$$\Delta_{\mathrm{mm}} \leq \frac{32\pi^4}{V} E_{\mathrm{mm}} \prod_{k=1}^{3} \left(\frac{h_{kk}}{2\pi} + \frac{3}{4}\sqrt{\frac{2}{\alpha\pi}} \right) \tag{A.5}$$

For the cutoff error, we know that the minimax approximation of the potential is greater than 0 and smaller than the exact potential. For deriving an upper bound of the error induced by $G_{\max}$ we consider a hard cutoff treating the minimax approximation as 0 for $|\mathbf{G}| > G_{\max}$ or neglecting all terms $|\mathbf{G}| > G_{\max}$. An efficient estimate of the error in the potential for $\mathbf{G} > G_{\max}$ is given by the inequality of arithmetic and geometric means

$$\Delta_V(\mathbf{G}) \leq \frac{1}{|\mathbf{G}|^2} \leq \frac{1}{3} \left(\frac{1}{G_1^2 G_2^2 G_3^2} \right)^{1/3} \tag{A.6}$$

The bounding function factorizes in the 3 Cartesian directions which allows for an efficient evaluation of the cutoff error. The cutoff error is the error of the integral $(a|a)_{\mathrm{c}}$ calculated with a hard cutoff $G_{\max}$ compared with the exact integral and is given by

$$\begin{aligned} \Delta_{\mathrm{c}} = |(a|a) - (a|a)_{\mathrm{c}}| = \qquad (\mathrm{A.7}) \\ \frac{4\pi^4}{V} \left(\frac{1}{\alpha}\right)^3 \left(\prod_{k=1}^{3} \left[(2\alpha)^{-(l_k - 1/2)} (\Gamma(l_k + 1/2))^{-1} \right] \right) \\ \left| \sum_{|\mathbf{G}| \geq G_{\max}} \Delta_V(\mathbf{G}) \exp(-i\mathbf{G} \cdot (\mathbf{A} - \mathbf{B})) C_{2\mathrm{l},(2\alpha)^{-1}}(\mathbf{G}) \right| \\ \leq \frac{4\pi^4}{3V} \left(\frac{1}{\alpha}\right)^3 \left(\prod_{k=1}^{3} \left[(2\alpha)^{-(l_k - 1/2)} (\Gamma(l_k + 1/2))^{-1} \right] \right) \prod_{k=1}^{3} \sum_{G_k \geq G_{\max}} |G_k|^{2l_k - 2/3} \exp\left(\frac{-G_k^2}{2\alpha} \right) \end{aligned}$$

We note that the cutoff error Δ_c is the largest for basis functions that are extended in reciprocal space, i.e. that have a large exponent α and a large maximum number of l. For a given basis set the largest error Δ_c is found by a parameter search (complete sampling for l, golden section search for α). An upper bound for the total error is then given by the sum of minimax and cutoff error $\Delta = \Delta_\mathrm{mm} + \Delta_\mathrm{c}$. A good estimate for the cutoff minimizing the total error Δ can be found by bisection on the difference $\Delta_\mathrm{c} - \Delta_\mathrm{mm}$.

Appendix B

Tensor Contraction Example

Fortran source code for the tensor contraction example of Alg. 2, documenting the tensor API. Left-out code is marked with '...' Complete source code for this example is provided together with the DBCSR source code (https://github.com/cp2k/dbcsr).

```
type(dbcsr_t_pgrid_type) :: pgrid ! process grid
type(dbcsr_t_distribution_type) :: dist ! distribution
type(dbcsr_t_type) :: A_ijk, A_lmk, B_iln, C_no, D_ijlm, E_jmn, F_jmo ! tensors
type(dbcsr_t_iterator_type) :: iter ! block iterator

real(real64), dimension(:, :, :), allocatable :: blk_values_3d ! variable for tensor blocks

integer, dimension(3) :: pdims_3d, shape_3d, blk_ind_3d, blk_size_3d
integer, dimension(4) :: shape_4d, pdims_4d

integer, dimension(:), allocatable :: blk_size_i, blk_size_j, ...
integer, dimension(:), allocatable :: blk_ind_1, blk_ind_2, ...
integer, dimension(:), allocatable :: dist_1, dist_2, ...

! -- create process grid for rank 3 tensors --
pdims_3d = 0
call dbcsr_t_pgrid_create(mpi_comm_world, pdims_3d, pgrid)

blk_size_i = [...] ! block sizes along dimension i
blk_size_j = [...] ! block sizes along dimension j
! and so on for k, l, m, n, o

! -- create tensor A(i,j,k) --

shape_3d = [...] ! shape in terms of block sizes

! default distribution vectors in 3 dimensions
call dbcsr_t_default_distvec(shape_3d(1), pdims_3d(1), blk_size_i, dist_1)
call dbcsr_t_default_distvec(shape_3d(2), pdims_3d(2), blk_size_j, dist_2)
call dbcsr_t_default_distvec(shape_3d(3), pdims_3d(3), blk_size_k, dist_3)

! create distribution
```

```
call dbcsr_t_distribution_new(dist, pgrid, dist_1, dist_2, dist_3)
deallocate (dist_1, dist_2, dist_3)
call dbcsr_t_pgrid_destroy(pgrid)

! create tensor
call dbcsr_t_create(A_ijk, "A(ij,k)", dist, &
                    map1_2d=[1, 2], map2_2d=[3], &
                    blk_size_i, blk_size_j, blk_size_k)

call dbcsr_t_distribution_destroy(dist)

! −− fill tensor A(i,j,k) −−

! block indices corresponding to local non−zero blocks
blk_ind_1 = [...]
blk_ind_2 = [...]
blk_ind_3 = [...]
call dbcsr_t_reserve_blocks(A_ijk, blk_ind_1, blk_ind_2, blk_ind_3)

! iterate over local non−zero blocks
call dbcsr_t_iterator_start(iter, A_ijk)
do while (dbcsr_t_iterator_blocks_left(iter))
   call dbcsr_t_iterator_next_block(iter, blk_ind_3d, blk_size=blk_size_3d)
   allocate (blk_values_3d(blk_size_3d(1), blk_size_3d(2), blk_size_3d(3)))
   blk_values_3d = [...] ! fill block
   call dbcsr_t_put_block(A_ijk, blk_ind_3d, blk_size_3d, blk_values_3d)
   deallocate (blk_values_3d)
enddo
call dbcsr_t_iterator_stop(iter)

! −− copy A(i,j,k) to A(l,m,k) −−

call dbcsr_t_create(A_ijk, A_lmk, name="A(lm,k)")
call dbcsr_t_copy(A_ijk, A_lmk)

! −− create D(i,j,l,m) −−

pdims_4d = 0
call dbcsr_t_pgrid_create(mpi_comm_world, pdims_4d, pgrid)

shape_4d = [...]
call dbcsr_t_default_distvec(shape_4d(1), pdims_4d(1), blk_size_i, dist_1)
call dbcsr_t_default_distvec(shape_4d(2), pdims_4d(2), blk_size_j, dist_2)
call dbcsr_t_default_distvec(shape_4d(3), pdims_4d(3), blk_size_l, dist_3)
call dbcsr_t_default_distvec(shape_4d(4), pdims_4d(4), blk_size_m, dist_4)

call dbcsr_t_distribution_new(dist, pgrid, dist_1, dist_2, dist_3, dist_4)
deallocate (dist_1, dist_2, dist_3, dist_4)
call dbcsr_t_pgrid_destroy(pgrid)

call dbcsr_t_create(D_ijlm, "D(ij,lm)", dist, &
                    map1_2d=[1, 2], map2_2d=[3, 4], &
                    blk_size_i, blk_size_j, blk_size_l, blk_size_m)
```

```
call dbcsr_t_distribution_destroy(dist)

! —— omitted: creation of tensors B, C, E ——

! —— F(j,m,o) = B(m,o,j) + B(o,m,j) ——

! F(j,m,o) = B(m,o,j)
call dbcsr_t_copy(B_iln, F_jmo, order=[2, 3, 1])

! F(j,m,o) = F(j,m,o) + B(o,m,j)
call dbcsr_t_copy(B_iln, F_jmo, order=[3, 2, 1], summation=.true.)

! ———— D(i,j,l,m) = A(i,j,k) x A(l,m,k) ————

! The arguments of dbcsr_t_contract define the mappings to the matrix representation
! using the following convention:
! tensor_3(map_1, map_2) := alpha * tensor_1(notcontract_1, contract_1) x
!                                   tensor_2(contract_2, notcontract_2)
!                           + beta * tensor_3(map_1, map_2)
! ('x' standing for matrix-matrix multiplication)

call dbcsr_t_contract(alpha=dbcsr_scalar(1.0_real64), tensor_1=A_ijk, tensor_2=A_lmk, &
                      beta=dbcsr_scalar(0.0_real64), tensor_3=D_ijlm, &
                      contract_1=[3], notcontract_1=[1, 2], &
                      contract_2=[3], notcontract_2=[1, 2], &
                      map_1=[1, 2], map_2=[3, 4], &
                      filter_eps=filter_eps)

! ———— E(j,m,n) = D(i,j,l,m) x B(i,l,n) ————

call dbcsr_t_contract(dbcsr_scalar(1.0_real64), D_ijlm, B_iln, &
                      dbcsr_scalar(0.0_real64), E_jmn, &
                      contract_1=[1, 3], notcontract_1=[2, 4], &
                      contract_2=[1, 2], notcontract_2=[3], &
                      map_1=[1, 2], map_2=[3], &
                      filter_eps=filter_eps)

! ———— C(n,o) = C(n,o) + E(j,m,n) x F(j,m,o) ————

call dbcsr_t_contract(dbcsr_scalar(1.0_real64), E_jmn, F_jmo, dbcsr_scalar(1.0_real64), C_no, &
                      contract_1=[1, 2], notcontract_1=[3], &
                      contract_2=[1, 2], notcontract_2=[3], &
                      map_1=[1], map_2=[2], &
                      filter_eps=filter_eps)
```

Appendix C

Parameters for Benchmark Systems

H_2O cc-TZV2P # atoms (electrons, orb basis, RI basis)	$O(N^3)$ RPA error [E_h/electron]	# batches	# nodes (MPI,OMP)	$O(N^4)$ RPA # nodes COSMA (MPI,OMP)	# nodes MKL (MPI,OMP)
96 (256, 1824, 4352)	$3 \cdot 10^{-7}$	2	2 (12, 1)	2 (1, 12)	2 (12, 1)
192	$2 \cdot 10^{-6}$	2	8 (12, 1)	32 (1, 12)	4 (12, 1)
384	$3 \cdot 10^{-6}$	4	16 (12, 1)	128 (1, 12)	32 (12, 1)
768	$2 \cdot 10^{-6}$	4	32 (12, 1)	1024 (1, 12)	256 (12, 1)
1152		4	64 (12, 1)		
1728		4	128 (12, 1)		
2592 (6912, 49248, 117504)		4	256 (12, 1)		

The system size was increased in all 3 dimensions by taking supercells of (1,1,1), (2,1,1), (2,2,1), (2,2,2), (3,2,2), (3,3,2), (3,3,3) of a basic unit cell containing 32 H_2O of dimensions (9.8528, 9.8528, 9.8528) Å.

TiO_2 cc-TZV2P # atoms (electrons, orb basis, RI basis)	$O(N^3)$ RPA error [E_h/electron]	# batches	# nodes (MPI,OMP)	$O(N^4)$ RPA # nodes (MPI,OMP)
108 (864, 4068, 11520)	$3 \cdot 10^{-6}$	3	32 (12, 1)	64 (1, 12)
216	$3 \cdot 10^{-6}$	5	64 (12, 1)	256 (1, 12)
432	$3 \cdot 10^{-6}$	3	256 (12, 1)	2400 (1, 12)
864 (6912, 32544, 92160)		6	512 (12, 1)	

The systems correspond to (3,3,1), (6,3,1), (6,6,1), (6,6,2) unit cells of dimensions (3.782, 3.782, 9.502) Å.

GNR aug-DZVP # atoms (electrons, orb basis, RI basis)	$O(N^3)$ G_0W_0 # batches	 # nodes (MPI,OMP)	$O(N^4)$ G_0W_0 # nodes (MPI,OMP)
114 (366, 2118, 5160)	4	8 (12, 1)	64 (1, 12)
222	4	16 (12, 1)	256 (1, 12)
438	4	64 (12, 1)	
870	4	256 (12, 1)	
1734 (5766, 33078, 80940)	8	512 (12, 1)	

H_2O TZVP # atoms (electrons, orb basis, RI basis)	RI-HFX error $[E_h/\text{electron}]$	# SCF	# batches	direct HFX # nodes (MPI,OMP)	# nodes (MPI,OMP)
96 (256, 1280, 6816)	$1 \cdot 10^{-6}$	16	2	4 (12, 1)	4 (12, 1)
192	$1 \cdot 10^{-6}$	15	2	8 (12, 1)	4 (12, 1)
384	$2 \cdot 10^{-6}$	14	2	32 (12, 1)	8 (12, 1)
768	$5 \cdot 10^{-7}$	18	3	64 (12, 1)	16 (12, 1)
2592 (6912, 34560, 184032)	$5 \cdot 10^{-7}$	12	8	256 (12, 1)	128 (4, 3)

TiO_2 cc-TZV2P # atoms (electrons, orb basis, RI basis)	RI-HFX error $[E_h/\text{electron}]$	# SCF	# batches	direct HFX # nodes (MPI,OMP)	# nodes (MPI,OMP)
108 (864, 4068, 11520)	$3 \cdot 10^{-6}$	20	5	32 (12, 1)	128 (12, 1)
216		20	5	64 (12, 1)	256 (12, 1)
432		19	6	128 (12, 1)	512 (12, 1)
864 (6912, 32544, 92160)		3	4	512 (12, 1)	

For the largest system 3 SCF steps have been performed and the execution time has been extrapolated to 19 SCF steps to save ressources.

TiO_2 TZV2P # atoms (electrons, orb basis, RI basis)	RI-HFX error $[E_h/\text{electron}]$	# SCF	# batches	direct HFX # nodes (MPI,OMP)	# nodes (MPI,OMP)
108 (864, 3060, 10404)	$3 \cdot 10^{-6}$	21	2	32 (12, 1)	64 (12, 1)
216	$2 \cdot 10^{-6}$	20	2	64 (12, 1)	150 (4, 3)
432	$2 \cdot 10^{-6}$	19	3	128 (12, 1)	600 (4, 3)
864 (6912, 24480, 83232)	$5 \cdot 10^{-7}$	18	5	256 (12, 1)	2400 (1, 12)

Appendix D

Exemplary Inputs

Listing D.1: Exemplary input file for low-scaling RPA

```
&GLOBAL
   PROJECT     H2O-RPA
   RUN_TYPE    ENERGY
   EXTENDED_FFT_LENGTHS
&END GLOBAL
&FORCE_EVAL
   METHOD Quickstep
   &DFT
      BASIS_SET_FILE_NAME BASIS_RI_cc-TZ

      ! sort basis functions wit respect to their exponents,
      ! this improves sparsity for tensor-based low-scaling algorithms
      SORT_BASIS          EXP
      &SCF
         EPS_SCF 1.0E-6
         MAX_SCF 20
         &OUTER_SCF
            EPS_SCF  1.0E-6
            MAX_SCF  5
         &END
         &OT
            MINIMIZER        CG
            PRECONDITIONER   FULL_ALL
         &END
      &END SCF
      &XC
         &XC_FUNCTIONAL PBE
```

```
  &END XC_FUNCTIONAL
  &WF_CORRELATION
    &RI
      &RI_METRIC
        ! select overlap RI metric
        ! for highest sparsity
        POTENTIAL_TYPE IDENTITY
      &END
    &END
    &LOW_SCALING

      ! number of batches, reducing memory by this factor
      MEMORY_CUT 2

      ! Filtering threshold for first tensor contraction
      EPS_FILTER 1.0E-7

      ! Filtering threshold for second tensor contraction,
      ! expressed relatively to EPS_FILTER
      EPS_FILTER_FACTOR 1.0
    &END
    &RI_RPA

      ! Perform minimax grid based variant of RPA
      MINIMAX_QUADRATURE

      ! Number of grid points
      RPA_NUM_QUAD_POINTS 10
    &END
    &INTEGRALS

      ! Enable fast analytical 2-center integrals
      ERI_METHOD MME
      &WFC_GPW

        ! integral accuracy criterion affecting memory
        ! consumption of 3-center integral calculation
        EPS_GRID 1.0E-04
      &END
    &END
  &END
&END XC
```

```
    &END DFT
    &SUBSYS
        &CELL
            ABC 9.8528 9.8528 9.8528
            MULTIPLE_UNIT_CELL  1  1  1
        &END CELL
        &TOPOLOGY
            COORD_FILE_NAME H2O−32.xyz
            COORD_FILE_FORMAT cp2k
            MULTIPLE_UNIT_CELL  1  1  1
        &END TOPOLOGY
        &KIND H
            BASIS_SET          cc−TZ
            BASIS_SET RI_AUX   RI_TZ
            POTENTIAL          GTH−PBE−q1
        &END KIND
        &KIND O
            BASIS_SET          cc−TZ
            BASIS_SET RI_AUX   RI_TZ
            POTENTIAL          GTH−PBE−q6
        &END KIND
    &END SUBSYS
&END FORCE_EVAL
```

Listing D.2: Exemplary input file for low-scaling G_0W_0

```
&GLOBAL
    PROJECT GNR−GW
    RUN_TYPE ENERGY
    EXTENDED_FFT_LENGTHS
&END GLOBAL
&FORCE_EVAL
    METHOD Quickstep
    &DFT
        BASIS_SET_FILE_NAME ./BASIS
        SORT_BASIS EXP
        UKS
        MULTIPLICITY 1
        &SCF
            EPS_SCF 1.0E−6
            MAX_SCF 20
            &OUTER_SCF
                EPS_SCF  1.0E−6
        MAX_SCF  5
    &END
    &OT
        MINIMIZER CG
        PRECONDITIONER FULL_SINGLE_INVERSE
    &END
&END SCF
&XC
    &XC_FUNCTIONAL PBE
    &END XC_FUNCTIONAL
    &WF_CORRELATION
        &LOW_SCALING
            EPS_FILTER 1.0E-11
            MEMORY_CUT 4
        &END
        &INTEGRALS
            &WFC_GPW
                EPS_GRID 1.0E-6

                ! improves stability of
                ! Cholesky decomposition
                EPS_PGF_ORB_S 1.0E-20
            &END
        &END
        &RI
            &RI_METRIC

                ! select overlap RI metric
                POTENTIAL_TYPE IDENTITY

                ! alternatively use truncated Coulomb metric
                ! for more accurate GW levels
                !POTENTIAL_TYPE TRUNCATED
                !CUTOFF_RADIUS 2.0
            &END
        &END
        &RI_RPA
            MINIMAX_QUADRATURE
            RPA_NUM_QUAD_POINTS 12
            &GW
                CORR_OCC 15
                CORR_VIRT 15
```

```
                    CROSSING_SEARCH NEWTON
                    OMEGA_MAX_FIT 1.0
                    ANALYTIC_CONTINUATION PADE
                    RI_SIGMA_X
                &END GW
            &END RI_RPA
        &END
    &END XC
&END DFT
&SUBSYS
    &CELL
        ABC [angstrom] 440.0 22.0 12.0
        PERIODIC NONE
    &END CELL
    &KIND H
        BASIS_SET aug-DZVP-GTH
        BASIS_SET RI_AUX RI_aug_DZ
        POTENTIAL GTH-PBE-q1
    &END KIND
    &KIND C
        BASIS_SET aug-DZVP-GTH
        BASIS_SET RI_AUX RI_aug_DZ
        POTENTIAL GTH-PBE-q4
    &END KIND
    &KIND C1 ! Breaking spin-up/spin-down symmetry on
        ! specific atoms in order to find correct spin state
        ELEMENT C
        BASIS_SET aug-DZVP-GTH
        BASIS_SET RI_AUX RI_aug_DZ
        POTENTIAL GTH-PBE-q4
        &BS
            &ALPHA
                NEL 1
                L 1
                N 2
            &END
            &BETA
                NEL -1
                L 1
                N 2
            &END
        &END
        &END KIND
        &KIND C2
            ELEMENT C
            BASIS_SET aug-DZVP-GTH
            BASIS_SET RI_AUX RI_aug_DZ
            POTENTIAL GTH-PBE-q4
            &BS
                &ALPHA
                    NEL -1
                    L 1
                    N 2
                &END
                &BETA
                    NEL 1
                    L 1
                    N 2
                &END
            &END
        &END KIND
        &TOPOLOGY
            COORD_FILE_NAME struc.xyz
            COORD_FILE_FORMAT xyz
            &CENTER_COORDINATES
            &END
            ! generate reorder to preserve spin symmetry breaking
            &GENERATE
                REORDER
            &END GENERATE
        &END TOPOLOGY
    &END SUBSYS
&END FORCE_EVAL
```

Listing D.3: Exemplary input file for low-scaling RI-HFX

```
&GLOBAL
    PROJECT TiO2
    RUN_TYPE ENERGY
    EXTENDED_FFT_LENGTHS
&END GLOBAL
&FORCE_EVAL
    METHOD Quickstep
    &DFT
        BASIS_SET_FILE_NAME     ./BASIS_TiO2
```

```
SORT_BASIS              EXP
&QS
   EPS_DEFAULT 1.0E−12
   EPS_PGF_ORB 1.0E−20
   EPS_FILTER_MATRIX 0.0e0
&END QS
&SCF
   EPS_SCF 1.0E−7
   MAX_SCF 20
   &OUTER_SCF
      EPS_SCF 1.0E−07
      MAX_SCF 3
   &END
   &OT
      PRECONDITIONER FULL_SINGLE_INVERSE
      MINIMIZER        DIIS
   &END OT
&END SCF
&XC
   &XC_FUNCTIONAL NONE
   &END XC_FUNCTIONAL
   &HF
      &RI
         RI_METRIC IDENTITY

         ! reduce memory by this factor (should be a square number,
         ! number of batches is square root of this number)
         MEMORY_CUT 16

         ! filter criterion for tensors
         EPS_FILTER 1.0E−08

         ! filter criterion for storage,
         ! expressed relatively to EPS_FILTER
         EPS_STORAGE_SCALING 100
      &END
      &INTERACTION_POTENTIAL
         POTENTIAL_TYPE TRUNCATED
         CUTOFF_RADIUS 4.5
      &END
   &END HF
&END XC
```

```
    &END DFT
    &SUBSYS
        &CELL
            ABC 11.346 11.346 9.502
            MULTIPLE_UNIT_CELL 1 1 1
        &END CELL
        &TOPOLOGY
            COORD_FILE_NAME tio2.xyz
            COORD_FILE_FORMAT cp2k
            MULTIPLE_UNIT_CELL 1 1 1
        &END TOPOLOGY
        &KIND O
            BASIS_SET cc-TZ
            BASIS_SET RI_HFX RI_TZ
            POTENTIAL GTH-PBE-q6
        &END KIND
        &KIND Ti
            BASIS_SET cc-TZ
            BASIS_SET RI_HFX RI_TZ
            POTENTIAL GTH-PBE-q12
        &END KIND
    &END SUBSYS
&END FORCE_EVAL
```

Acknowledgements

First of all I would like to thank Professor Jürg Hutter for his support, for the fruitful discussions and for his ideas that helped to initiate and shape this work. I'm grateful for the possibility of attending various summer schools and conferences.

This thesis owes a great deal to the work by Jan Wilhelm who derived and implemented the theoretical framework of low-scaling RPA and GW. Without his experience in the implementation and optimization of tensor expressions, I could never have generalized these concepts to a tensor library.

Dorothea Golze contributed the calculations for the IC-QM/MM method and I'm deeply thankful for this rewarding collaboration.

Alfio Lazzaro and Ilia Sivkov were my collaborators in the DBCSR PASC project and I'm grateful for discussions that helped me understand the inner workings of the DBCSR library. I'm grateful to Alfio for sharing his experience with MPI and high-performance computing in general.

The group of Joost VandeVondele at CSCS developed the COSMA library and contributed to the DBCSR library. Marko Kabić is the main author of the COSMA library (which was used as a reference for benchmarking) and I'm thankful for the discussions I had with him and Joost about parallel matrix multiplication algorithms. I thank Shoshana Jakobovits from the same group for her valueable improvements on the GPU side of the DBCSR library and for the effort she put into documentation of DBCSR.

I'd like to thank Frederick Stein for contributing important ideas concerning the generalization of the MME method for electron repulsion integrals to other potentials, and for the motivating exchange of ideas we had while working on the MP2 / RPA code.

I want to thank Augustin Bussy and Maximilien Ambroise for reusing my work. Special thanks go to Maximilien Ambroise for contributing the C interface to the DBCSR tensor API. I hope that the DBCSR tensor API will help you make fast progress, and that the library meets your expectations.

Special thanks go to my officemate Tiziano for the technical support, for sharing his broad knowledge of computer science and software development, and for the many insightful discussions we had over the years.

I’m grateful to the CMSZH graduate school for the support I received, and for the organization of the annual retreat.

Finally, I thank all former and current members of the Hutter group for the great time at UZH. Staying in contact during an extended time of ‘home office’ helped me to stay motivated to work from home.

Bibliography

[1] The CP2K Developers Group. *DBCSR: Distributed Block Compressed Sparse Row matrix library*. 2020. `https://github.com/cp2k/dbcsr`.

[2] J. Wilhelm et al. *Large-Scale Cubic-Scaling Random Phase Approximation Correlation Energy Calculations Using a Gaussian Basis*. In: *Journal of Chemical Theory and Computation* 12.12 (2016), pp. 5851–5859.

[3] J. Wilhelm et al. *Toward GW Calculations on Thousands of Atoms*. In: *The Journal of Physical Chemistry Letters* 9.2 (2018), pp. 306–312.

[4] M. Guidon et al. *Ab initio molecular dynamics using hybrid density functionals*. In: *The Journal of Chemical Physics* 128.21 (2008), p. 214104.

[5] A. Szabo and N. S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Mineola: Dover Publications, Inc., 1996.

[6] R. G. Parr and W. Yang. *Density functional theory of atoms and molecules*. Oxford University Press USA, 1989.

[7] T. Helgaker, P. Jorgensen, and J. Olsen. *Molecular electronic-structure theory*. John Wiley & Sons, 2014.

[8] D. Marx and J. Hutter. *Ab initio molecular dynamics: basic theory and advanced methods*. Cambridge University Press, 2009.

[9] J. P. Perdew et al. *Prescription for the design and selection of density functional approximations: More constraint satisfaction with fewer fits*. In: *The Journal of Chemical Physics* 123.6 (2005), p. 062201.

[10] S. Goedecker, M. Teter, and J. Hutter. *Separable dual-space Gaussian pseudopotentials*. In: *Phys. Rev. B* 54 (3 July 1996), pp. 1703–1710.

[11] J. VandeVondele and J. Hutter. *An efficient orbital transformation method for electronic structure calculations*. In: *The Journal of Chemical Physics* 118.10 (2003), pp. 4365–4369.

[12] P. Hohenberg and W. Kohn. *Inhomogeneous Electron Gas*. In: *Phys. Rev.* 136 (3B Nov. 1964), B864–B871.

[13] W. Kohn and L. J. Sham. *Self-Consistent Equations Including Exchange and Correlation Effects*. In: *Phys. Rev.* 140 (4A Nov. 1965), A1133–A1138.

[14] C. Møller and M. S. Plesset. *Note on an Approximation Treatment for Many-Electron Systems*. In: *Phys. Rev.* 46 (7 Oct. 1934), pp. 618–622.

[15] J.-Q. Sun and R. J. Bartlett. *Second-order many-body perturbation-theory calculations in extended systems*. In: *The Journal of Chemical Physics* 104.21 (1996), pp. 8553–8565.

[16] M. Katouda and S. Nagase. *Efficient parallel algorithm of second-order Møller–Plesset perturbation theory with resolution-of-identity approximation (RI-MP2)*. In: *International Journal of Quantum Chemistry* 109.10 (2009), pp. 2121–2130.

[17] P. Y. Ayala, K. N. Kudin, and G. E. Scuseria. *Atomic orbital Laplace-transformed second-order Møller–Plesset theory for periodic systems*. In: *The Journal of Chemical Physics* 115.21 (2001), pp. 9698–9707.

[18] C. Pisani et al. *Local-MP2 electron correlation method for nonconducting crystals*. In: *The Journal of Chemical Physics* 122.9 (2005), p. 094113.

[19] M. Marsman et al. *Second-order Møller–Plesset perturbation theory applied to extended systems. I. Within the projector-augmented-wave formalism using a plane wave basis set*. In: *The Journal of Chemical Physics* 130.18 (2009), p. 184103.

[20] A. Grüneis, M. Marsman, and G. Kresse. *Second-order Møller–Plesset perturbation theory applied to extended systems. II. Structural and energetic properties*. In: *The Journal of Chemical Physics* 133.7 (2010), p. 074107.

[21] J. L. Whitten. *Coulombic potential energy integrals and approximations*. In: *The Journal of Chemical Physics* 58.10 (1973), pp. 4496–4501.

[22] J. W. Mintmire, J. R. Sabin, and S. B. Trickey. *Local-density-functional methods in two-dimensionally periodic systems. Hydrogen and beryllium monolayers*. In: *Phys. Rev. B* 26 (4 Aug. 1982), pp. 1743–1753.

[23] F. Weigend. *A fully direct RI-HF algorithm: Implementation, optimised auxiliary basis sets, demonstration of accuracy and efficiency*. In: *Phys. Chem. Chem. Phys.* 4 (18 2002), pp. 4285–4291.

[24] M. Feyereisen, G. Fitzgerald, and A. Komornicki. *Use of approximate integrals in ab initio theory. An application in MP2 energy calculations*. In: *Chemical Physics Letters* 208.5 (1993), pp. 359–363.

[25] O. Vahtras, J. Almlöf, and M. Feyereisen. *Integral approximations for LCAO-SCF calculations*. In: *Chemical Physics Letters* 213.5 (1993), pp. 514–518.

[26] B. I. Dunlap, J. W. D. Connolly, and J. R. Sabin. *On some approximations in applications of X theory*. In: *The Journal of Chemical Physics* 71.8 (1979), pp. 3396–3402.

[27] X. Ren et al. *Resolution-of-identity approach to Hartree–Fock, hybrid density functionals, RPA, MP2 and GW with numeric atom-centered orbital basis functions*. In: *New Journal of Physics* 14.5 (May 2012), p. 053020.

[28] Y. Jung et al. *Auxiliary basis expansions for large-scale electronic structure calculations*. In: *Proceedings of the National Academy of Sciences* 102.19 (2005), pp. 6692–6697.

[29] S. Reine et al. *Variational and robust density fitting of four-center two-electron integrals in local metrics*. In: *The Journal of Chemical Physics* 129.10 (2008), p. 104101.

[30] D. C. Langreth and J. P. Perdew. *Exchange-correlation energy of a metallic surface: Wave-vector analysis*. In: *Phys. Rev. B* 15 (6 Mar. 1977), pp. 2884–2901.

[31] G. E. Scuseria, T. M. Henderson, and D. C. Sorensen. *The ground state correlation energy of the random phase approximation from a ring coupled cluster doubles approach*. In: *The Journal of Chemical Physics* 129.23 (2008), p. 231101.

[32] W. Klopper et al. *Spin flipping in ring-coupled-cluster-doubles theory*. In: *Chemical Physics Letters* 510.1 (2011), pp. 147–153.

[33] H. Eshuis, J. Yarkony, and F. Furche. *Fast computation of molecular random phase approximation correlation energies using resolution of the identity and imaginary frequency integration*. In: *The Journal of Chemical Physics* 132.23 (2010), p. 234114.

[34] M. Kaltak, J. Klimeš, and G. Kresse. *Low Scaling Algorithms for the Random Phase Approximation: Imaginary Time and Laplace Transformations*. In: *Journal of Chemical Theory and Computation* 10.6 (2014), pp. 2498–2507.

[35] J. Hutter et al. *cp2k: atomistic simulations of condensed matter systems*. In: *WIREs Computational Molecular Science* 4.1 (2014), pp. 15–25.

[36] S. Obara and A. Saika. *Efficient recursive computation of molecular integrals over Cartesian Gaussian functions*. In: *The Journal of Chemical Physics* 84.7 (1986), pp. 3963–3974.

[37] J. Spencer and A. Alavi. *Efficient calculation of the exact exchange energy in periodic systems using a truncated Coulomb potential*. In: *Phys. Rev. B* 77 (19 May 2008), p. 193110.

[38] M. Guidon, J. Hutter, and J. VandeVondele. *Robust Periodic Hartree−Fock Exchange for Large-Scale Simulations Using Gaussian Basis Sets*. In: *Journal of Chemical Theory and Computation* 5.11 (2009), pp. 3010–3021.

[39] B. G. LIPPERT, J. HUTTER, and M. PARRINELLO. *A hybrid Gaussian and plane wave density functional scheme*. In: *Molecular Physics* 92.3 (1997), pp. 477–488.

[40] M. Del Ben, J. Hutter, and J. VandeVondele. *Second-Order Møller–Plesset Perturbation Theory in the Condensed Phase: An Efficient and Massively Parallel Gaussian and Plane Waves Approach*. In: *Journal of Chemical Theory and Computation* 8.11 (2012), pp. 4177–4188.

[41] M. Del Ben et al. *Enabling simulation at the fifth rung of DFT: Large scale RPA calculations with excellent time to solution*. In: *Computer Physics Communications* 187 (2015), pp. 120–129.

[42] D. Neuhauser, E. Rabani, and R. Baer. *Expeditious Stochastic Calculation of Random-Phase Approximation Energies for Thousands of Electrons in Three Dimensions*. In: *The Journal of Physical Chemistry Letters* 4.7 (2013), pp. 1172–1176.

[43] J. E. Moussa. *Cubic-scaling algorithm and self-consistent field for the random-phase approximation with second-order screened exchange*. In: *The Journal of Chemical Physics* 140.1 (2014), p. 014107.

[44] Y. Gao et al. *Sublinear scaling for time-dependent stochastic density functional theory*. In: *The Journal of Chemical Physics* 142.3 (2015), p. 034106.

[45] M. Kállay. *Linear-scaling implementation of the direct random-phase approximation*. In: *The Journal of Chemical Physics* 142.20 (2015), p. 204105.

[46] H. F. Schurkus and C. Ochsenfeld. *Communication: An effective linear-scaling atomic-orbital reformulation of the random-phase approximation using a contracted double-Laplace transformation*. In: *The Journal of Chemical Physics* 144.3 (2016), p. 031101.

[47] F. Hüser, T. Olsen, and K. S. Thygesen. *Quasiparticle GW calculations for solids, molecules, and two-dimensional materials*. In: *Phys. Rev. B* 87 (23 June 2013), p. 235132.

[48] G. Lippert, J. Hutter, and M. Parrinello. *The Gaussian and augmented-plane-wave density functional method for ab initio molecular dynamics simulations*. In: *Theoretical Chemistry Accounts* 103.2 (1999), pp. 124–140.

[49] D. Braess and W. Hackbusch. *Approximation of $1/x$ by exponential sums in $[1, \infty)$*. In: *IMA Journal of Numerical Analysis* 25.4 (Oct. 2005), pp. 685–697.

[50] P. P. Ewald. *Die Berechnung optischer und elektrostatischer Gitterpotentiale*. In: *Annalen der Physik* 369.3 (1921), pp. 253–287.

[51] S. Reine, E. Tellgren, and T. Helgaker. *A unified scheme for the calculation of differentiated and undifferentiated molecular integrals over solid-harmonic Gaussians*. In: *Phys. Chem. Chem. Phys.* 9 (34 2007), pp. 4771–4779.

[52] M. Del Ben, J. Hutter, and J. VandeVondele. *Electron Correlation in the Condensed Phase from a Resolution of Identity Approach Based on the Gaussian and Plane Waves Scheme*. In: *Journal of Chemical Theory and Computation* 9.6 (2013), pp. 2654–2671.

[53] T. Helgaker and P. R. Taylor. *Gaussian basis sets and molecular integrals*. In: *Modern Electronic Structure Theory: Part II*. 1995, pp. 725–856.

[54] B. Aradi. *Fypp — Python powered Fortran metaprogramming*. `https://github.com/aradi/fypp`. 2020.

[55] R. A. Kendall, T. H. Dunning, and R. J. Harrison. *Electron affinities of the first-row atoms revisited. Systematic basis sets and wave functions*. In: *J. Chem. Phys.* 96 (1992).

[56] F. Weigend, A. Köhn, and C. Hättig. *Efficient use of the correlation consistent basis sets in resolution of the identity MP2 calculations*. In: *J. Chem. Phys.* 116 (2002).

[57] B. P. Pritchard et al. *A New Basis Set Exchange: An Open, Up-to-date Resource for the Molecular Sciences Community*. In: *J. Chem. Inf. Model.* 59 (2019).

[58] D. Golze et al. *Simulation of Adsorption Processes at Metallic Interfaces: An Image Charge Augmented QM/MM Approach*. In: *J. Chem. Theory Comput.* 9.11 (2013), pp. 5086–5097.

[59] J. P. Perdew, K. Burke, and M. Ernzerhof. *Generalized Gradient Approximation Made Simple*. In: *Phys. Rev. Lett.* 77 (18 Oct. 1996), pp. 3865–3868.

[60] E. Epifanovsky et al. *New implementation of high-level correlated methods using a general block tensor library for high-performance electronic structure calculations*. In: *Journal of Computational Chemistry* 34.26 (2013), pp. 2293–2309.

[61] E. Solomonik et al. *A massively parallel tensor contraction framework for coupled-cluster computations*. In: *Journal of Parallel and Distributed Computing* 74.12 (2014), pp. 3176–3190.

[62] K. Z. Ibrahim et al. *Cross-scale efficient tensor contractions for coupled cluster computations through multiple programming model backends*. In: *Journal of Parallel and Distributed Computing* 106 (2017), pp. 92–105.

[63] E. Solomonik and T. Hoefler. *Sparse Tensor Algebra as a Parallel Programming Model*. In: *CoRR* abs/1512.00066 (2015).

[64] S. Manzer et al. *A General Sparse Tensor Framework for Electronic Structure Theory*. In: *Journal of Chemical Theory and Computation* 13.3 (2017), pp. 1108–1116.

[65] J. A. Calvin, C. A. Lewis, and E. F. Valeev. *Scalable Task-Based Algorithm for Multiplication of Block-Rank-Sparse Matrices*. In: *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. IA3 '15. Austin, Texas: Association for Computing Machinery, 2015.

[66] J. A. Calvin and E. F. Valeev. *Task-Based Algorithm for Matrix Multiplication: A Step Towards Block-Sparse Tensor Computing*. In: *CoRR* abs/1504.05046 (2015).

[67] C. Peng et al. *Massively Parallel Implementation of Explicitly Correlated Coupled-Cluster Singles and Doubles Using TiledArray Framework*. In: *The Journal of Physical Chemistry A* 120.51 (2016), pp. 10231–10244.

[68] C. Peng, J. A. Calvin, and E. F. Valeev. *Coupled-cluster singles, doubles and perturbative triples with density fitting approximation for massively parallel heterogeneous platforms*. In: *International Journal of Quantum Chemistry* 119.12 (2019), e25894.

[69] X. Wang, C. A. Lewis, and E. F. Valeev. *Efficient evaluation of exact exchange for periodic systems via concentric atomic density fitting*. In: *The Journal of Chemical Physics* 153.12 (2020), p. 124116.

[70] T. Herault et al. *Distributed-memory multi-GPU block-sparse tensor contraction for electronic structure (revised version)*. Research Report RR-9365. Inria - Research Centre Grenoble – Rhône-Alpes, Oct. 2020, p. 34.

[71] U. Borštnik et al. *Sparse matrix multiplication: The distributed block-compressed sparse row library*. In: *Parallel Computing* 40.5 (2014), pp. 47–58.

[72] R. Barrett et al. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, 1994.

[73] O. Schütt et al. *GPU-Accelerated Sparse Matrix–Matrix Multiplication for Linear Scaling Density Functional Theory*. In: *Electronic Structure Calculations on Graphics Processing Units*. John Wiley & Sons, Ltd, 2016. Chap. 8, pp. 173–190.

[74] L. E. Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis. Montana State University-Bozeman, College of Engineering, 1969.

[75] A. Heinecke et al. *LIBXSMM: Accelerating Small Matrix Multiplications by Runtime Code Generation*. In: *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2016, pp. 981–991.

[76] J. Choi et al. *ScaLAPACK: a portable linear algebra library for distributed memory computers — design issues and performance*. In: *Computer Physics Communications* 97.1 (1996), pp. 1–15.

[77] E. Solomonik et al. *Scaling Betweenness Centrality Using Communication-Efficient Sparse Matrix Multiplication*. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '17. Denver, Colorado: Association for Computing Machinery, 2017.

[78] G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. In: (1966).

[79] J. Wilhelm. *Low-Scaling Many-Body Perturbation Theory for Nanoscopic Systems*. PhD thesis. University of Zurich, 2017.

[80] D. Irony, S. Toledo, and A. Tiskin. *Communication lower bounds for distributed-memory matrix multiplication*. In: *Journal of Parallel and Distributed Computing* 64.9 (2004), pp. 1017–1026.

[81] J. Demmel et al. *Communication-Optimal Parallel Recursive Rectangular Matrix Multiplication*. In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 2013, pp. 261–272.

[82] M. Frigo et al. *Cache-oblivious algorithms*. In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 1999, pp. 285–297.

[83] A. Lazzaro et al. *Increasing the Efficiency of Sparse Matrix-Matrix Multiplication with a 2.5D Algorithm and One-Sided MPI*. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC '17. Lugano, Switzerland: Association for Computing Machinery, 2017.

[84] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. https://www.tensorflow.org/.

[85] A. Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035.

[86] I. Sivkov et al. *DBCSR: A Blocked Sparse Tensor Algebra Library*. 2019.

[87] G. Kwasniewski et al. *Red-Blue Pebbling Revisited: Near Optimal Parallel Matrix-Matrix Multiplication*. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '19. Denver, Colorado: Association for Computing Machinery, 2019, pp. 1–22.

[88] K. Marko. *COSMA: Communication-Optimal Matrix-Multiplication*. https://github.com/eth-cscs/COSMA. 2020.

[89] C. Spreafico and J. VandeVondele. *The nature of excess electrons in anatase and rutile from hybrid DFT and RPA*. In: *Phys. Chem. Chem. Phys.* 16 (47 2014), pp. 26144–26152.

[90] J. VandeVondele, U. Borštnik, and J. Hutter. *Linear Scaling Self-Consistent Field Calculations with Millions of Atoms in the Condensed Phase*. In: *Journal of Chemical Theory and Computation* 8.10 (2012), pp. 3565–3573.

[91] M. Guidon, J. Hutter, and J. VandeVondele. *Auxiliary Density Matrix Methods for Hartree−Fock Exchange Calculations*. In: *Journal of Chemical Theory and Computation* 6.8 (2010), pp. 2348–2364.

[92] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. Vol. 55. US Government printing office, 1948.

CMSZH

CHEMICAL MOLECULAR SCIENCES ZURICH

GRADUATE SCHOOL