Enabling DFT Simulations of Large Metallic Systems by Integrating the PEXSI Method into CP2K

Patrick Seewald

Master Thesis

Conducted in the group of Prof. Dr. Joost VandeVondele

Co-supervised by Mohammad Hossein Bani-Hashemian

ETH Zurich, Department of Materials

September 2014 - March 2015

Abstract

The pole expansion and selected inversion (PEXSI) method, accessible as a software library, was made available for use within the general purpose atomistic simulation package CP2K. The hope was to enable efficient large-scale simulations of metallic systems for which the diagonalisation approach to solve the Kohn-Sham equations is the limiting factor in an electronic structure calculation. PEXSI is a finite temperature approach to solve the Kohn-Sham equations in an orbital-free formalism and can fully replace the diagonalisation step. It can be efficiently parallelised and has a computational cost that scales at most quadratically with respect to the system size. The CP2K-PEXSI implementation was carefully tested in terms of accuracy and efficiency. Benchmarks were performed to compare PEXSI with the linear scaling implementation of CP2K for insulating systems, and with standard diagonalisation for metallic systems. We found that PEXSI is consistently faster than standard diagonalisation for quasi-2D systems if a large-scale parallelisation is applied (often more than 1000 processors are required). The minimal system size and the required number of processors at which PEXSI becomes more efficient than diagonalisation depend on the sparsity and the dimensionality of a system. For condensed 3D bulk systems we found that in the current implementation, CP2K-PEXSI is not significantly more efficient than diagonalisation even for an optimal parallelisation and large system sizes. On the CP2K side, the performance of PEXSI could possibly be improved by a more accurate distance screening criterion to more selectively tune the sparsity of the overlap matrix.

Contents

Chapter 1

Introduction

A key ingredient for realistic simulations of material properties of interest in chemistry, physics, material science and biology is an accurate description of the electronic structure of the system. Due to the inherent quantum nature of electrons, this step involves solving the electronic Schrödinger equation. For larger systems containing hundreds or more atoms, Kohn-Sham density functional theory (KS DFT) is the only available approximative method that is feasible and at the same time sufficiently accurate for a wide range of applications.

The computational cost of traditional algorithms for solving KS DFT scales cubically with respect to the system size. These algorithms are thus not able to handle larger systems containing thousands of atoms, even on the most powerful computers. The cubically scaling steps that dominate the computational cost for large systems are solving the Kohn-Sham equations by diagonalisation and the orthogonalisation of the wave functions [1].

Efficient algorithms solving KS DFT with a computational cost scaling linearly with the system size have been developed and implemented, enabling simulations of systems containing millions of atoms [2]. One restriction of linear scaling algorithms is that they rely on the nearsightedness principle, ensuring a sparse density matrix. Consequently, they can treat insulating systems only, i.e. systems with a sufficiently large band gap. For metallic systems without a gap, efficient alternatives to the diagonalisation of the Kohn-Sham matrix are rare.

Recently, an approach based on a pole expansion of the Fermi-Dirac function combined with selected inversion (PEXSI) has been proposed as an alternative to diagonalisation [3]. The advantage of this method over linear scaling is that it is based on the finite temperature Fermi-Dirac function and does not assume a sparse density matrix. Consequently, PEXSI is expected to perform equally well

for insulating as for metallic systems. Compared with standard diagonalisation, the PEXSI approach has a favourable parallel scalability and scales at most quadratically with respect to the system size.

This work is concerned with the integration of the PEXSI method, available as a library, into the atomistic simulation package CP2K. Besides the practical benefit of making PEXSI available for general-purpose electronic structure calculations with CP2K, such an integration allows for a direct comparison with the methods already implemented in CP2K. Based on such a comparison, the potential strengths and limitations of the PEXSI method for applications can be discussed.

This Master thesis is organized as follows: First, Kohn-Sham density functional theory is introduced, followed by a description of algorithms and their implementation in software packages, focusing on CP2K and PEXSI (Chapter 2). The integration of PEXSI into CP2K is described (Chapter 3), discussing conceptual aspects and giving an overview over the current state of implementation. An introduction to CP2K-PEXSI is given (Chapter 4) with a description of all relevant input parameters that need to be considered for a CP2K-PEXSI calculation. Recommendations on how to optimally set up a calculation are backed up with experiments testing the accuracy and efficiency of CP2K-PEXSI. A comparison of PEXSI with the other SCF methods (linear scaling and diagonalization) implemented in CP2K follows, considering the parallel scalability and the computation time in dependence of the system size for metallic and insulating quasi-2D and 3D bulk systems (Chapter 5). Chapter 6 summarises the obtained results, discusses strengths and limitations of the CP2K-PEXSI approach for applications and points out possible future development.

Chapter 2

Theory

2.1 Kohn-Sham density functional theory

Kohn-Sham density functional theory (DFT) is the most commonly applied electronic structure approach for systems of interest in chemistry and material science. This method is applicable to large systems while still being able to deal with the quantum nature of these systems. In the following, the finite temperature formulation of Kohn-Sham density functional theory is briefly discussed.

For most applications, the nuclei of atoms can be treated classically and can be held fixed for the electronic structure problem (known as *Born-Oppenheimer approximation*). The goal is then to solve the electronic Schrödinger equation which is a differential equation in a 3^{N_e} dimensional Hilbert space, where N_e is the total number of electrons with coordinates $\{\boldsymbol{r}_i\}$:

$$\hat{H}(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_\mathrm{e}})\Psi(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_\mathrm{e}}) = E\Psi(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_\mathrm{e}}). \tag{2.1}$$

Here, E is the total energy and $\hat{H}$ is the Hamiltonian of the electronic system which has the form

$$\hat{H} = \sum_{i=1}^{N_\mathrm{e}} \left(-\frac{1}{2}\nabla^2 + V(\boldsymbol{r}_i) \right) + \sum_{i<j}^{N_\mathrm{e}} \frac{1}{\boldsymbol{r}_i - \boldsymbol{r}_j}. \tag{2.2}$$

The term $V(\boldsymbol{r}_i)$ is the Coulomb energy including electron i and all nuclei. Due to the exponential scaling of the dimension of the Hilbert space with the number of electrons, direct numerical methods to solve this equation become very expensive and for realistic systems containing many atoms, approximations that reduce the degrees of freedom of the electronic system are necessary.

The main idea of DFT is to express all observables as functionals of the electronic density $\rho(\boldsymbol{r})$ instead of the wave function $\Psi(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_\mathrm{e}})$. The theoretical basis for such a reformulation is given by two theorems by Hohenberg and Kohn [4], leading to the *Kohn-Sham* formulation of DFT [5].

The first theorem states that the ground state of a system is uniquely specified by the ground state electronic density $\rho(r)$ and, therefore, the ground state energy is a functional of the density

$$E_0 = E[\rho] = \int d^3r V(\boldsymbol{r})\rho(\boldsymbol{r}) + F[\rho], \tag{2.3}$$

where the *Hohenberg-Kohn functional* F is not known analytically. The second Hohenberg-Kohn theorem proves that the ground state can be found by minimising $E[\rho]$ with respect to ρ.

The Hohenberg-Kohn theorems were extended by Mermin to finite electronic temperatures $T > 0$ [6]. His treatment implies that for finite temperatures and fixed number of particles, the proper variational functional is the *Helmholtz free energy* [7, 8]

$$\mathcal{F}[\rho] = E[\rho] - TS[\rho]. \tag{2.4}$$

Despite the strict validity of both Hohenberg-Kohn theorems, DFT is an approximative method as approximations have to be introduced for the unknown Hohenberg-Kohn functional $F[\rho]$. The Hohenberg-Kohn functional can be written as a sum of known and unknown contributions $F[\rho] = E_\mathrm{h}[\rho] + T[\rho] + E_\mathrm{xc}[\rho]$ with the following terms: the known classical expression for the Coulomb interaction $E_\mathrm{h}[\rho]$ (Hartree term), the unknown kinetic energy term $T[\rho]$ and the unknown exchange and correlation term $E_\mathrm{xc}[\rho]$ which can be considered a quantum-mechanical correction to the Hartree term.

In Kohn-Sham DFT, the kinetic energy term is approximated by the kinetic energy of a system of non-interacting electrons with the same ground state density. From this, the Kohn-Sham equations can be derived as an effective single-electron Schrödinger equations

$$H[\rho(\boldsymbol{r})]\Psi_i(\boldsymbol{r})) = \varepsilon_i \Psi_i(\boldsymbol{r}). \tag{2.5}$$

Here, H is the Kohn-Sham Hamiltonian given by

$$H[\rho(\boldsymbol{r})] = -1/2\nabla^2 + V(\boldsymbol{r}) + V_h(\boldsymbol{r}) + V_{xc}(\boldsymbol{r}), \tag{2.6}$$

containing the following potential energy terms: the electron-nuclei Coulomb interaction $V(\boldsymbol{r})$, the Coulomb interaction $V_h(\boldsymbol{r})$ between one electron and the density $\rho(\boldsymbol{r}')$, and the exchange-correlation potential $V_{\mathrm{xc}}(\boldsymbol{r})$ which is derived from $E_{\mathrm{xc}}[\rho]$.

For the exchange-correlation potential $V_{\mathrm{xc}}(\boldsymbol{r})$ approximations exist such as generalised gradient approximations (GGA) and meta-GGAs. The former parameterises $E_{\mathrm{xc}}[\rho]$ in dependence of ρ and $\nabla\rho$, while the latter additionally includes derivatives of higher order and the kinetic energy density.

Within this framework, the electron density $\rho(r)$ can be derived from the distribution of the (fictitious non-interacting) electrons over Kohn-Sham orbitals

$$\rho(\boldsymbol{r}) = \sum_i |\Psi_i(\boldsymbol{r})|^2 f_i. \tag{2.7}$$

The fractional occupation numbers f_i are given by the Fermi-Dirac distribution

$$f_i = f_\beta(\varepsilon_i - \mu) = \frac{2}{1 + e^{\beta(\varepsilon_i - \mu)}} \tag{2.8}$$

with the inverse temperature $\beta = 1/(k_{\mathrm{B}}T)$. The entropy S in the Helmholtz free energy (2.4) takes the form

$$S = -2k_{\mathrm{B}} \sum_i (\tilde{f}_i \log \tilde{f}_i + (1 - \tilde{f}_i) \log(1 - \tilde{f}_i)), \tag{2.9}$$

where we abbreviated $\tilde{f}_i = f_i/2$.

In the standard approach, the Kohn-Sham equations Eq. (2.5) are solved by finding for given potential terms the eigenfunctions $\Psi_i(\boldsymbol{r})$, $i = 1, \ldots, N_{\mathrm{e}}$ with eigenvalues ε_i. The solution to the full DFT problem has to consider that the Kohn-Sham Hamiltonian depends on the same electronic density that is a solution to the Kohn-Sham equations. This requires the *self-consistent field* (SCF) method that, starting from an initial guess for the Kohn-Sham matrix, iteratively solves the Kohn-Sham equations by updating the Kohn-Sham matrix with the density from previous iterations until convergence is achieved.

2.2 Methods to solve the Kohn-Sham equations

For solving the Kohn-Sham equations Eq. (2.5) numerically, the problem must be discretised first, usually done so by expanding the Kohn-Sham orbitals $\Psi_i(\boldsymbol{r})$

into a linear combination of a finite number of basis functions $\varphi_i(\boldsymbol{r})$

$$\Psi_i(\boldsymbol{r}) = \sum_{j=1}^{N} c_{ij}\varphi_j(\boldsymbol{r}). \tag{2.10}$$

The truncation of this expansion to a finite, preferably small number of basis functions is only accurate if the shape of the basis functions can well reproduce the properties of the wave function. A good approximation is obtained by choosing basis functions that resemble local atomic orbitals, e.g. Gaussian type orbitals centered at the atomic positions.

Inserting the expansion Eq. (2.10) into the Kohn-Sham equations Eq. (2.5) leads to a generalised eigenvalue problem

$$\boldsymbol{H}\boldsymbol{c} = \boldsymbol{S}\boldsymbol{c}\boldsymbol{\varepsilon}. \tag{2.11}$$

Here, $\boldsymbol{H}$ is the Kohn-Sham matrix with entries $H_{ij} = \langle\varphi_i|\hat{H}|\varphi_j\rangle$, $\boldsymbol{S}$ is the overlap matrix $S_{ij} = \langle\varphi_i|\varphi_j\rangle$ and $\varepsilon_{ij} = \varepsilon_i\delta_{ij}$. The form of $\boldsymbol{H}$ and $\boldsymbol{S}$ depends on the chosen basis set. For orthogonal basis functions, $\boldsymbol{S}$ is the identity matrix and the problem reduces to an ordinary eigenvalue problem. Localised basis functions are not orthogonal. An advantage of localised basis functions is however that both $\boldsymbol{S}$ and $\boldsymbol{H}$ are sparse for systems containing many atoms. This is because the contribution of a pair of functions that is separated by a sufficiently large distance can be neglected.

The sparsity of $\boldsymbol{H}$ and $\boldsymbol{S}$ is exploited by fast algorithms that operate on non-negligible matrix elements only. These approaches are necessary to enable DFT calculations on large systems for which traditional diagonalisation algorithms working on full matrices come to a limit. Two methods making use of matrix sparsity will be discussed and compared in this work: *linear scaling SCF* as it is currently implemented in the program package CP2K and the *Pole EXpansion and Selected Inversion* (PEXSI) method.

2.2.1 Traditional diagonalisation approach

The first step of solving the SCF equations by diagonalisation is a transformation from the generalised eigenvalue problem Eq. (2.11) to an ordinary eigenvalue problem $\boldsymbol{H}'\boldsymbol{c}' = \boldsymbol{c}'\boldsymbol{\varepsilon}$. This can be done by applying a transformation matrix $\boldsymbol{U}$ with $\boldsymbol{S} = \boldsymbol{U}^T\boldsymbol{U}$ (Cholesky decomposition) or, alternatively, $\boldsymbol{U} = \boldsymbol{S}^{\frac{1}{2}}$ (symmetric orthogonalisation). The explicit terms in the ordinary eigenvalue problem are

then given by $\boldsymbol{H}' = (\boldsymbol{U}^T)^{-1}\boldsymbol{H}\boldsymbol{U}^{-1}$ and $\boldsymbol{c}' = \boldsymbol{U}\boldsymbol{c}$.

The eigenvalue problem needs to be solved partially for the lowest eigenvalues and eigenvectors corresponding to occupied Kohn-Sham orbitals. The solution of the standard eigenvalue problem is the most expensive step and scales with $\mathcal{O}(N_\mathrm{e}^3)$. A standard implementation is provided by ScaLAPACK [9].

2.2.2 Direct evaluation of the density matrix

Alternative approaches avoid diagonalisation of the Kohn-Sham matrix by direct evaluation of the density matrix as a matrix function of the Kohn-Sham matrix $\boldsymbol{H}$. The density $\rho(\boldsymbol{r})$ is the diagonal $\rho(\boldsymbol{r}) = P(\boldsymbol{r}, \boldsymbol{r})$ of the real space density matrix given by

$$P(\boldsymbol{r}, \boldsymbol{r}') = \sum_i \psi_i(\boldsymbol{r}) f_\beta(\varepsilon_i - \mu)\psi_i^*(\boldsymbol{r}') = \sum_{jk} \varphi_j(\boldsymbol{r}) P_{jk} \varphi_k^*(\boldsymbol{r}'). \tag{2.12}$$

The second equality introduces the density matrix $\boldsymbol{P}$ in basis representation with entries

$$P_{jk} = \sum_i c_{ji} f_\beta(\varepsilon_i - \mu) c_{ik}. \tag{2.13}$$

The matrix function establishing the link between the sought density matrix $\boldsymbol{P}$ and the given matrices $\boldsymbol{H}$ and $\boldsymbol{S}$ can be derived as

$$\boldsymbol{P} = f_\beta(\boldsymbol{S}^{-1}\boldsymbol{H} - \mu\boldsymbol{I})\boldsymbol{S}^{-1}. \tag{2.14}$$

Some methods evaluate the density matrix for zero temperature, in this case $f_\beta(\varepsilon - \mu) = 2\Theta(\varepsilon - \mu)$ and

$$P_{jk} = 2\sum_{i=1}^{N_e} c_{ji} c_{ik}, \tag{2.15}$$

$$\boldsymbol{P} = 2\Theta(\boldsymbol{S}^{-1}\boldsymbol{H} - \mu\boldsymbol{I})\boldsymbol{S}^{-1}, \tag{2.16}$$

where Θ is the Heaviside step function. For fixed number of electrons, the chemical potential μ is determined by the trace conservation relation

$$Tr[P(x, x')] = Tr[\boldsymbol{P}\boldsymbol{S}] = N_\mathrm{e}. \tag{2.17}$$

2.2.3 Linear scaling algorithms

Instead of performing diagonalisation, the expression (2.16) can be used to compute the zero temperature density matrix as a matrix function of the Kohn-Sham matrix. This can be achieved by expanding the density matrix in a polynomial $\mathcal{P}$ of the Kohn-Sham matrix, $\boldsymbol{P} = \mathcal{P}(\boldsymbol{H})$. In this case, the most important operation is matrix-matrix multiplication which can be efficiently implemented for sparse matrices.

Many linear scaling algorithms use an iterative approach, in which the density matrix $\boldsymbol{P}$ is obtained by reiterating a polynomial expansion of the Kohn-Sham matrix. These algorithms differ in their choice of the polynomial $\mathcal{P}$, McWeeny purification [10] for example uses $\mathcal{P}(X_n) = 3X_n^2 - 2X_n^3$, starting from $X_0 = \alpha(\boldsymbol{H} - \mu\boldsymbol{S}) + \beta\boldsymbol{S}$ (for some scalars α, β), converging to the step function (2.16). More advanced methods use polynomials with better convergence properties in combination with a correction to guarantee trace conservation (trace resetting purification [11]).

Linear scaling algorithms rely on a sparse representation of the matrices $\boldsymbol{S}, \boldsymbol{H}$ and $\boldsymbol{P}$ such that the number of non-zero elements grows linearly with the system size. The sparsity of $\boldsymbol{H}$ and $\boldsymbol{S}$ is ensured by localised basis functions. The sparsity of $\boldsymbol{P}$ additionally requires the validity of the nearsightedness principle.

The nearsightedness principle states that the density matrix $P(\boldsymbol{r}, \boldsymbol{r}')$ approaches zero if $\boldsymbol{r}$ and $\boldsymbol{r}'$ are sufficiently apart [1,12–14]. For insulating systems, the decay of $P(\boldsymbol{r}, \boldsymbol{r}')$ is exponential in $|\boldsymbol{r} - \boldsymbol{r}'|$. This decay property leads to a sparse $\boldsymbol{P}$, as long as a localised and well-conditioned set of basis functions is applied [15]. For metallic systems at zero temperature, the density matrix decays only algebraically (with $1/|\boldsymbol{r} - \boldsymbol{r}'|^k$ for some k). Linear scaling DFT is thus only applicable to insulating systems with a sufficiently large band gap.

2.2.4 Pole expansion and selected inversion

The pole expansion and selected inversion (PEXSI [3]) approach combines a Fermi operator expansion method with a method to calculate selected elements of a matrix inverse to efficiently evaluate selected elements of the finite temperature density matrix Eq. (2.14). This approach relies on the observation that not all entries of the density matrix are required. In fact, in the evaluation of the real space density

$$\rho(\boldsymbol{r}) = \sum_{ij} \varphi_i(\boldsymbol{r}) P_{ij} \varphi_j(\boldsymbol{r}), \tag{2.18}$$

only terms with indices (i, j) give a contribution for which $\varphi_i(\boldsymbol{r})\varphi_j(\boldsymbol{r}) \neq 0$. The advantage of a method imposing such a sparsity pattern on $\boldsymbol{P}$ lies in the fact that it can also efficiently treat systems where $\boldsymbol{P}$ is not sparse by value, e.g. metallic systems without a band gap. While linear scaling algorithms rely on the decay property of the density matrix, this approach does not depend on any physical properties but only on the locality of the basis functions.

Pole Expansion

The first step of the PEXSI method is to represent the finite temperature Fermi-Dirac function by a linear combination of functions that can be efficiently evaluated for matrices. The Pole expansion of the Fermi-Dirac function in the complex plane has the analytical form [16]

$$f_\beta(\varepsilon - \mu) \approx \mathrm{Im} \sum_{l=1}^{P} \frac{\omega_l^\rho}{\varepsilon - (z_l + \mu)}. \tag{2.19}$$

The complex weights ω_l^ρ and poles z_l are computed from analytical expressions. The number of required poles is proportional to $\log(\beta \Delta E)$, where ΔE is the difference between the largest and the smallest Kohn-Sham eigenvalue ε_i.

Provided this expansion for the density matrix (2.14), the remaining task is to evaluate selected elements $\{(\boldsymbol{H} - (z_l + \mu)\boldsymbol{S})^{-1}\}_{ij}$ for which $\varphi_i(\boldsymbol{r})\varphi_j(\boldsymbol{r}) \neq 0$, for all poles l.

Selected Inversion

Selected inversion [17–20] is the key to the favorable scaling of PEXSI compared to traditional $\mathcal{O}(N_\mathrm{e}^3)$ approaches. The algorithm applied to a sparse symmetric matrix $\boldsymbol{A}$ starts with an $\boldsymbol{LDL}^T$ factorisation of $\boldsymbol{A}$, with $\boldsymbol{L}$ a block lower diagonal matrix called the Cholesky factor, and $\boldsymbol{D}$ a block diagonal matrix. Then all elements A_{ij}^{-1} are calculated for which $L_{ij} \neq 0$. If $\boldsymbol{A} = \boldsymbol{H} - z\boldsymbol{S}$, the non-zero elements of $\boldsymbol{H}$ and $\boldsymbol{S}$ are a subset of the non-zero elements of $\boldsymbol{L}$. The selected inversion dominates the computational scaling of the PEXSI method. Its scaling is proportional to the number of non-zero elements in the Cholesky factor $\boldsymbol{L}$ and can be shown to be $\mathcal{O}(N_\mathrm{e}^2)$ for bulk 3D systems, $\mathcal{O}(N_\mathrm{e}^{3/2})$ for quasi-2D system and $\mathcal{O}(N_\mathrm{e})$ for quasi-1D systems [17].

2.3 Self-consistent solution

The solution of the Kohn-Sham equations must be self-consistent in the sense that the solution $\rho^{\text{out}}(\boldsymbol{r})$ must be equal to the input density $\rho^{\text{in}}(\boldsymbol{r})$ that was used to express the Kohn-Sham matrix $\boldsymbol{H}[\rho(r)]$. Such a solution can be achieved by reiterating the solution of the Kohn-Sham equations until convergence is achieved. The convergence towards the fixed point of $\rho(\boldsymbol{r})$ is however not guaranteed. A common problem is that oscillations in the density over subsequent SCF steps prevent convergence. Depending on the system, different mixing schemes have to be applied in order to damp oscillations and accelerate convergence.

Linear mixing adds a constant fraction of the output density ρ^{out} to the input density ρ^{in}

$$\rho^{\text{in}}_{n+1} = (1-\alpha)\rho^{\text{in}}_n + \alpha\rho^{\text{out}}_n,\ 0 < \alpha \leq 1. \tag{2.20}$$

Alternatively, the mixing can also be applied on the level of the Kohn-Sham matrix instead of the density. Linear mixing is the most simple approach but is not stable for all systems. Metallic systems show a more problematic convergence behaviour and a more sophisticated method needs to be applied.

Broyden mixing [21–23] is a quasi-Newton-Raphson method to minimise the residual $R_n = \rho^{out}_n - \rho^{in}_n$. This is done by approximating the Jacobian $J = \partial R/\partial\rho$ from the residual and the density of previous SCF iterations. The input density is then set according to Newton's method to

$$\rho^{\text{in}}_{n+1} = \rho^{\text{out}}_n - J^{-1}_n R_n, \tag{2.21}$$

with J_n the approximation to J in the n^{th} SCF iteration.

2.4 Molecular dynamics

Ab initio molecular dynamics (MD) simulates the motion of the nuclei exploiting the Born-Oppenheimer approximation in which the nuclei are considered as classical particles in a potential given by the electronic energy. In Born-Oppenheimer molecular dynamics [24], the equations of motion for an atom A are simply

$$M_{\text{A}}\ddot{\boldsymbol{R}}_{\text{A}}(t) = -\nabla_{\text{A}} E(\{\boldsymbol{R}_I\}), \tag{2.22}$$

where M_{A} and $\boldsymbol{R}_{\text{A}}$ are the mass and the coordinates of atom A, respectively. $E(\{\boldsymbol{R}_I\})$ is the electronic energy parametrically dependent on the coordinates

$\boldsymbol{R}_I$ of all atoms I.

These equations allow for a rather convenient separation of the electronic structure calculation and the propagation of the nuclei. For fixed atomic positions, the electronic energy is calculated by a sufficiently converged SCF cycle. This energy is then used to propagate the nuclei by a small time step (using for instance the Verlet algorithm), followed again by an update of the electronic energy.

For finite temperatures, the forces are more conveniently calculated by the gradient with respect to the Mermin free energy $\mathcal{F}$

$$M_{\mathrm{A}}\ddot{\boldsymbol{R}}_{\mathrm{A}}(t) = -\nabla_{\mathrm{A}}\mathcal{F}(\{\boldsymbol{R}_I\}). \tag{2.23}$$

It can be shown that this gives the same forces as Eq. (2.22) but with the advantage that $\mathcal{F}(\{\boldsymbol{R}_I\})$ is stationary with respect to variation in the occupation numbers [7,8]. From this the following expression for the forces can be derived [3, 25–27]

$$-\nabla_{\mathrm{A}}\mathcal{F}(\{\boldsymbol{R}_{\mathrm{I}}\}) = -\mathrm{Tr}\left[\boldsymbol{P}\nabla_{\mathrm{A}}\boldsymbol{H}\right] + \mathrm{Tr}\left[\boldsymbol{P}^{\mathrm{E}}\nabla_{\mathrm{A}}\boldsymbol{S}\right], \tag{2.24}$$

where $\boldsymbol{P}^{\mathrm{E}}$ is the energy weighted density matrix

$$P^{\mathrm{E}}_{jk} = \sum_i c_{ji}\varepsilon_i f_\beta(\varepsilon_i - \mu)c_{ik}. \tag{2.25}$$

2.5 CP2K / Quickstep

Quickstep [25, 28], the DFT module of CP2K, implements the *Gaussian and plane waves* (GPW) method [29, 30]. This approach combines the advantages of two representations of the density in terms of Gaussians and plane waves. The benefits of plane waves are that the calculation of Hartree potential is simple and Fast Fourier Transform (FFT) can be applied to efficiently convert between real and reciprocal space representation of the electronic density. On the other hand, the electronic density has the strongest variations in the vicinity of the atomic ions. Less basis functions are required to accurately solve the Kohn-Sham equations if a localised set of functions resembling atomic orbitals is used instead of plane waves. The other advantage of local atomic functions is that the overlap matrix and the Kohn-Sham Matrix are represented by sparse matrices (for sufficiently large systems).

Quickstep provides an up-to-date implementation of linear scaling SCF methods. These methods particularly benefit from the DBCSR sparse matrix-matrix

multiplication (see section 2.7.2). The linear scaling implementation is efficient even for large 3D bulk systems containing up to millions of atoms [2].

2.5.1 Basis functions

The basis functions $\varphi_i(\boldsymbol{r})$ employed to express the Kohn-Sham equations in matrix form are given in Quickstep as spherically contracted Gaussian functions [31, 32]. A basis function is a product of a spherical harmonic (representing the angular part of an atomic orbital) with a radial part given by a linear combination of Gaussians

$$\varphi_i(\boldsymbol{r}) = Y_{m_i}^{l_i}(\theta, \phi) r^{l_i} \sum_j c_{ij} \exp(-\alpha_j r^2). \tag{2.26}$$

The advantage of Gaussians over more accurate representations of atomic orbitals (Slater-type orbitals) is that matrix elements $\langle \varphi_i | \hat{O} | \varphi_j \rangle$ can be expressed analytically. All real space integrals needed for the calculation of the total energy, the Kohn-Sham matrix and the forces on the ions can thus be efficiently evaluated from analytical expressions.

2.5.2 Pseudopotentials

In the Gaussian and plane wave approach, the DFT problem is only solved for the valence electrons, assuming that the core electrons are well localised at the atomic nuclei. Their contribution to the total energy is described by a pseudopotential that is considered instead of the electron-nuclei Coulomb potential term of all-electron calculations. Quickstep provides the pseudopotentials of Goedecker, Teter and Hutter (*GTH pseudopotentials* [33, 34]) that are separated in a one-electron term (local part) and a two-electron term (nonlocal part) $V^{\mathrm{PP}}(\boldsymbol{r}, \boldsymbol{r}') = V_{\mathrm{loc}}^{\mathrm{PP}}(\boldsymbol{r}) + V_{\mathrm{nl}}^{\mathrm{PP}}(\boldsymbol{r}, \boldsymbol{r}')$.

2.5.3 Dual representation of the electronic density

The real space representation of the density in terms of atomic orbitals $\varphi_i(r)$ is given by equation (2.12) and can be written as

$$\rho(\boldsymbol{r}) = \sum_{ij} P_{ij} \varphi_i(\boldsymbol{r}) \varphi_j(\boldsymbol{r}). \tag{2.27}$$

In the plane wave representation, the density is expressed as

$$\rho(\boldsymbol{r}) = \frac{1}{\Omega} \sum_{G} \tilde{\rho}(\boldsymbol{G}) \exp(i\boldsymbol{G} \cdot \boldsymbol{r}), \tag{2.28}$$

where Ω is the volume of the unit cell and $\boldsymbol{G}$ are vectors in reciprocal space. Conversion between the two representations is achieved by expressing Gaussians numerically on a real space grid and Fast Fourier Transform.

In Kohn-Sham DFT, the electronic energy is given as a functional of the electronic density,

$$E[\rho] = T[\rho] + E_{\mathrm{V}}[\rho] + E_{\mathrm{h}}[\rho] + E_{\mathrm{xc}}[\rho], \tag{2.29}$$

and in the GPW approach, these terms take the following form:

- the electronic kinetic energy

$$T[\rho] = \sum_{ij} P_{ij} \langle \varphi_i(\boldsymbol{r}) | -\frac{1}{2}\nabla^2 | \varphi_j(\boldsymbol{r}) \rangle, \tag{2.30}$$

- the energy of the interaction between ionic cores (including inner-shell electrons) and valence electrons

$$E_{\mathrm{V}}[\rho] = \sum_{ij} P_{ij} \left(\langle \varphi_i(\boldsymbol{r}) | V_{\mathrm{loc}}^{\mathrm{PP}}(\boldsymbol{r}) | \varphi_j(\boldsymbol{r}) \rangle + \langle \varphi_i(\boldsymbol{r}) | V_{\mathrm{nl}}^{\mathrm{PP}}(\boldsymbol{r}, \boldsymbol{r}') | \varphi_j(\boldsymbol{r}') \rangle \right), \tag{2.31}$$

- and the energy of the classical interaction between valence electrons (Hartree energy)

$$E_{\mathrm{h}}[\rho] = 2\pi\Omega \sum_{G} \frac{\tilde{\rho}^*(\boldsymbol{G})\tilde{\rho}(\boldsymbol{G})}{\boldsymbol{G}^2}. \tag{2.32}$$

For the exchange-correlation functional $E_{\mathrm{xc}}[\rho]$ several approximate functionals based on generalised gradient approximations (GGA) and meta-GGA are implemented.

2.6 The PEXSI library

A general description of the PEXSI method has been given in section 2.6, here we mention some algorithmic and implementational details that are of interest for our application of PEXSI.

2.6.1 Finding the correct chemical potential

The density matrix calculation employing the pole expansion Eq. (2.19) requires the correct chemical potential for which $\mathrm{Tr}(\boldsymbol{P}[\mu]\boldsymbol{S}) = N_\mathrm{e}$ is satisfied. As the chemical potential is not known beforehand, the PEXSI method has to be initialised with an initial guess μ_0 and reiterated according to Newton's method until convergence in the number of electrons $N_\mathrm{e}(\mu_k)$ is achieved. To overcome the problem that Newton's method may not be robust for a bad initial guess of the chemical potential and in order to minimise the number of expensive PEXSI iterations, the PEXSI library provides an inertia counting procedure that calculates an estimate of the chemical potential.

Inertia counting [35] can best be illustrated for the zero temperature approximation ($\beta \to \infty$) of the number of occupied Kohn-Sham orbitals $N_\beta(\mu)$ which are the orbitals with eigenvalue below μ. The correct chemical potential satisfies $N_\infty(\mu) = N_\mathrm{e}/2$ and a good estimate can be obtained by root finding, given an efficient way to evaluate the function $N_\infty(\varepsilon)$. The number $N_\infty(\varepsilon)$ is equal to the number of negative eigenvalues of $\boldsymbol{H} - \varepsilon\boldsymbol{S}$ which can be obtained by performing an $\boldsymbol{LDL}^T$ factorisation of $\boldsymbol{H} - \varepsilon\boldsymbol{S}$. According to Sylvester's law of inertia, the number of negative entries in the diagonal matrix $\boldsymbol{D}$ is equal to the number of negative eigenvalues of $\boldsymbol{LDL}^T$.

2.6.2 DFT solver

The PEXSI library provides a full Kohn-Sham DFT solver that computes the selected elements of the density matrix $\boldsymbol{P}$ for a given matrix pencil $(\boldsymbol{H}, \boldsymbol{S})$ in one wrapper routine. The solver takes care of the required steps to solution (inertia counting and PEXSI Newton iterations) internally. This design makes full use of the separation between the two algorithmic steps in the SCF method:

1. Calculating the density matrix $\boldsymbol{P}$ from $(\boldsymbol{H}, \boldsymbol{S})$ by solving Eq. (2.14).

2. Mapping the density matrix to the real space density $\rho(x)$ to calculate the energy by Eq. (2.4) and the new Kohn-Sham matrix $H_{ij} = \partial E/\partial P_{ij}$.

Step (1) is perfomed by PEXSI while step (2) requires a fully featured DFT code (as e.g. CP2K) that sets up the matrices $\boldsymbol{H}$ and $\boldsymbol{S}$, given a choice of basis functions and an exchange-correlation functional for a specific system. Thanks to this separation, an interface to the PEXSI library does not need to deal with the internals of the PEXSI code. Setting a few input options, passing the

PEXSI IN
Parallelisation: Number of processors per pole, dimension of process grid
Parameters: Electronic temperature, number of poles, target accuracy in number of electrons, initial guess for chemical potential
Matrix pencil $(\boldsymbol{H}, \boldsymbol{S})$
Exact number of electrons

PEXSI OUT
Density matrix $\boldsymbol{P}$
Energy weighted density matrix $\boldsymbol{P}^{\mathrm{E}}$
Entropic energy contribution $-TS$
Chemical potential μ
Converged number of electrons $\mathrm{Tr}(\boldsymbol{PS})$

Figure 2.1: Main input and output of PEXSI.

matrix data to PEXSI, calling the DFT driver and retrieving the results are all required steps to make full use of PEXSI.

The main input and output of PEXSI is listed in Fig. 2.1. The entropic energy contribution is needed for the free energy $\mathcal{F} = E - TS$ and is given by Eq. (2.9). Due to the orbital-free framework of the PEXSI method, S can not be evaluated via the occupation numbers. Instead, PEXSI calculates the free energy $\mathcal{F}$ and the total energy E as matrix functions such that $-TS = \mathcal{F} - E$ can be expressed as [19, 36]

$$-TS = Tr[(\boldsymbol{P}^{F} - \boldsymbol{P}^{\mathrm{E}})\boldsymbol{S}] + \mu N_{\mathrm{e}}, \tag{2.33}$$

where $\boldsymbol{P}^{\mathrm{E}}$ is the energy weighted density matrix defined in Eq. (2.25) and $\boldsymbol{P}^{\mathrm{F}}$ is the free energy density matrix

$$P^{\mathrm{F}}_{ik} = \sum_{i} c_{ji}\varepsilon_i f^{\mathrm{F}}_{\beta}(\varepsilon_i - \mu)c_{ik}, \tag{2.34}$$

with

$$f^{\mathrm{F}}_{\beta}(\varepsilon - \mu) = -2\beta^{-1}\ln(1 + \exp(\beta(\mu - \varepsilon))). \tag{2.35}$$

For a complete description of all parameters and arguments to PEXSI, the reader is referred to the online documentation of PEXSI [37].

2.6.3 Parallelisation Scheme

PEXSI is parallelised on two levels. The total number of available cores is split into subgroups where each subgroup performs the selected inversion of one pole at a time. Depending on the size and sparsity of the matrix, the selected

inversion scales efficiently to a number of processes per pole n_{pp} between 256 and 1024 [20, 37]. According to our experience, for accurate calculations, a typical number of poles $n_{\text{pole}} = 50$ or more is required for approximating the Fermi-Dirac function. A full parallelisation of PEXSI in which all poles are processed at once requires a total number of processes $n_{\text{proc}} = n_{\text{pp}} \cdot n_{\text{pole}}$. In this case, the total wall time is roughly the time needed to invert one pole (neglecting the generally cheap inertia counting and symbolic factorisation). Due to this two-level parallelism, PEXSI can scale almost perfectly to tens of thousands of processors.

2.6.4 LU factorisation

The PEXSI library depends on an external factorisation routine. In the current implementation, $\boldsymbol{LU}$ factorisation for a general matrix into a product of a block lower triangular matrix $\boldsymbol{L}$ and a block upper triangular matrix $\boldsymbol{U}$ is done instead of the $\boldsymbol{LDL}^T$ factorisation for symmetric matrices [35]. For the factorisation PEXSI requires the *SuperLU_DIST* library [38]. Furthermore a matrix reordering method is needed in order to reduce the number of additional non-zero elements (fill-in) in the Cholesky factor $\boldsymbol{L}$. For this step either *ParMETIS* [39] or *PT-Scotch* [40] software packages can be linked in.

2.7 Storage layouts for sparse matrices

Algorithms solving the Kohn-Sham equations by direct evaluation of the Fermi-Dirac function heavily rely on matrix operations that can make efficient use of the underlying sparsity of the matrices. Sparse matrices are represented in special formats such that only the non-zero matrix entries need to be explicitly stored. The down-side is a more complicated way of accessing matrix data because there is no direct correspondence between the matrix indices of an entry and its location in memory. Therefore a set of indices needs to be stored that map matrix positions to memory locations and vice versa. Distributed matrix formats store the matrix data on many distributed-memory nodes and an additional mapping is required that maps a given matrix position to the process where the matrix entry is stored.

A commonly applied format is the *Compressed Sparse Row* (CSR) format that is also used by PEXSI and - in a blocked form - by CP2K. The Compressed Sparse row (CSR) format and the block-compressed sparse row (BCSR) format

are described below, the representation of an actual example matrix in the two formats is depicted in Fig. 2.2. For comprehensibility, the same convention for the CSR index is used for both formats, slightly different from the CP2K implementation.

2.7.1 Compressed Sparse Row (CSR) format

The CSR format [41] stores the non-zero elements in a contiguous array `nzval`. The entries inside `nzval` are ordered in a row-wise fashion, i.e. sorted in ascending order with respect to $n \cdot i + j$, where n is the total number of rows, i is the row index and j is the column index. The index data is stored in two arrays `colind` and `rowptr`: the array `colind` stores the column index of each element in `nzval`. The array `rowptr` contains a compressed representation of the row index: it stores the position in `nzval` of all matrix elements that are first in a row. This storage layout requires only $2 \cdot nnz + n + 1$ memory locations instead of $n \cdot m$, where nnz is the number of non-zero elements, n is the number of rows and m is the number of columns.

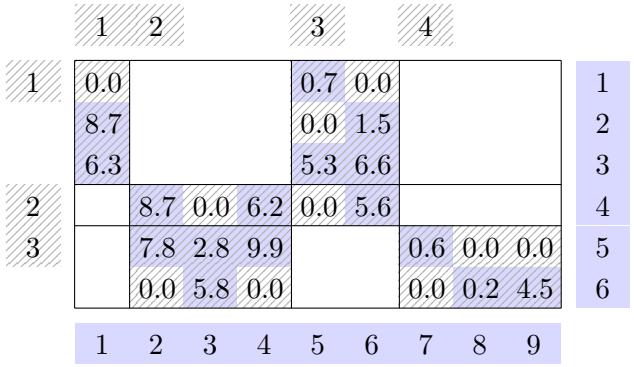
There exists also the column-wise storage layout - the CSC or Compressed Sparse Column format - that is formally used by PEXSI. But for symmetric matrices CSC format and CSR format are equivalent and for this work there is no need to distinguish between the two.

There are different possible choices for the distribution of sparse matrices over processors on a distributed memory machine. It is common to map the matrix rows and matrix columns to a two-dimensional process grid where each combination of grid rows and grid columns is mapped to a process.

2.7.2 Block-Compressed Sparse Row (BCSR) format of CP2K

CP2K is provided with its own library for sparse matrix-matrix multiplication [42]. Matrices are stored in a distributed BCSR format which is the CSR format referring to matrix blocks instead of single matrix entries. This storage layout is tailored to matrices of the form as they evolve in Kohn-Sham DFT with localised basis functions: each blocked row and blocked column is associated with an atom, containing the contributions from all basis functions centered at that atom.

Whether a block is zero or not is decided by assigning an interaction radius to each atom based on the spatial extension of the basis functions centered at



CSR format

`nzval`	0.7	8.7	1.5	6.3	5.3	6.6	8.7	6.2	5.6	7.8	2.8	9.9	0.6	5.8	0.2	4.5
`colind`	5	1	6	1	5	6	2	4	6	2	3	4	7	3	8	9
`rowptr`	1	2	4	7	10	14	17									

BCSR format

`nzval`	0.0	8.7	6.3	0.7	0.0	5.3	0.0	1.5	6.6	8.7	0.0	6.2	0.0	5.6	7.8	...
`blkptr`	1	4	10	13	15	21	27									
`colind`	1	3	2	3	2	4										
`rowptr`	1	3	5	7												

Figure 2.2: Representation of a sparse matrix in CSR and BCSR format. The upper panel shows the position of the non-zero elements in the uncompressed matrix. The row and column indices are labelled at the borders of the matrix - referring to matrix blocks in BCSR format and to matrix entries in CSR format. The matrix entries highlighted with a line-pattern are non-zero elements in BCSR format and the blue-shaded entries are non-zero elements in CSR format. The CSR non-zero elements form a subset of the BCSR non-zero elements. The two lower panels represent the same matrix in CSR and BCSR format. The array `nzval` stores the matrix entries. The arrays `colind` and `rowptr` are the CSR indices referring to matrix entries (CSR format) and matrix blocks (BCSR format). For the BCSR format, an additional index array `blkptr` is required that stores the position in `nzval` of the first entry in each block.

this atom. A block is set to zero if it corresponds to a pair of atoms that are separated by a distance greater than the sum of their interaction radii. In order to further improve sparsity, filtering is applied at different stages of matrix operations. Filtering removes all blocks whose Frobenius norm is below a given threshold.

On the algorithmic side, the representation of the sparsity in terms of blocks is favourable because matrix-matrix multiplication can be separated into two steps: multiplication on the level of the CSR indices in the first step, and multiplication of the block data in the second step. The first step exploits the sparsity but is not efficient due to the complicated data access involving a back-translation of the CSR index to actual row- and column-indices. The block-wise grouping enhances the efficiency of this step because only one look-up is required per block. In the second step, the actual multiplication of matrix data contained in full blocks is performed which can be implemented in a most efficient and cache-oblivious way, this step can also be transferred to GPU's.

Chapter 3

Integration of PEXSI into CP2K

3.1 Overview

PEXSI provides a minimal set of routines that need to be called in order to solve the Kohn-Sham equations. An interface to these routines is available in both C++ and Fortran programming languages. PEXSI already provides a full DFT solver such that it can be integrated to CP2K in a black-box like fashion: CP2K initialises PEXSI with some parameters (usually coming from user input) and calls the PEXSI DFT driver in the main SCF routine to obtain the density matrix and related quantities from the given matrix pencil $(\boldsymbol{H}, \boldsymbol{S})$. After finishing an SCF cycle, the PEXSI data is released. The main task that needs to be done on the CP2K-side is the conversion of the block-wise BCSR matrix format to the standard CSR format. In this conversion, care must be taken that the sparsity of the density matrix $\boldsymbol{P}$ (and by design of PEXSI, the sparsity of the matrices $\boldsymbol{H}$ and $\boldsymbol{S}$) is determined by the criterion $P_{ij} \neq 0$ if $\varphi_i(x)\varphi_j(x) \neq 0$.

3.2 Matrix format conversion

The matrix format of the PEXSI interface is CSR format that is distributed such that each processor holds $\lfloor N/P \rfloor$ consequentive rows where N is the number of rows and columns and P is the number of processors. The remaining $N - P \lfloor N/P \rfloor$ rows are appended to the data on the N^{th} processor [37]. The conversion should also offer the possibility to define a sparsity pattern for CSR matrices such that the non-zero elements in CSR format are a subset of the non-zero elements in the BCSR format. This is a natural feature to include in

the conversion because the BCSR format imposes a block-wise sparsity pattern. This is a restriction of the matrix format that is no longer present when a matrix is converted to the CSR format.

A conversion from the CP2K DBCSR (Distributed BCSR) format to the PEXSI CSR format must redistribute the matrix data among the processes and reorder the matrix data from block-wise to element-wise ordering. The most convenient way to achieve this is to redistribute the matrix data in a first step. Then the conversion between the two matrix formats can be performed locally on each process in a second step, without the need for communication. For the back conversion from CSR to DBCSR, the local reordering is done first, followed by the redistribution to the initial DBCSR matrix format. These two steps are depicted in Fig. 3.1 for the case of an artificial matrix distributed on 4 processors p_0, p_1, p_2, p_3.

3.2.1 Row-wise redistribution

The upper panel in Fig. 3.1 illustrates the redistribution of an arbitrarily distributed DBCSR matrix to a row-wise distribution in which each processor holds subsequent matrix rows. The distribution of DBCSR blocks over processes is set up by mapping the BCSR rows and columns to the rows and columns of the process grid. We will refer to this mapping by the terms *row distribution* and *column distribution*. In CP2K, the row and column distribution are set up randomly in order to achieve optimal load balance among the processes. The process grid is depicted above the example DBCSR matrix, each color corresponding to one of the 4 processes. The column distribution and row distribution are given by numbers on top and on the left side of the example matrix, respectively. The process grid for a row-wise distribution is always a grid of dimension $P \times 1$. In the current implementation, the matrix conversion offers three options for the number of consequentive rows on a processor, where P_{CSR} is the number of processes over which the CSR matrix shall be distributed, N is the total number of matrix rows and N_{block} is the total number of BCSR rows:

1. each processor holds $\lceil N/P_{\mathrm{CSR}} \rceil$ consequentive rows (see example)
2. each processor holds $\lfloor N/P_{\mathrm{CSR}} \rfloor$ consequentive rows (PEXSI matrix format)
3. each processor holds $\lfloor N_{\mathrm{block}}/P_{\mathrm{CSR}} \rfloor$ BCSR rows.

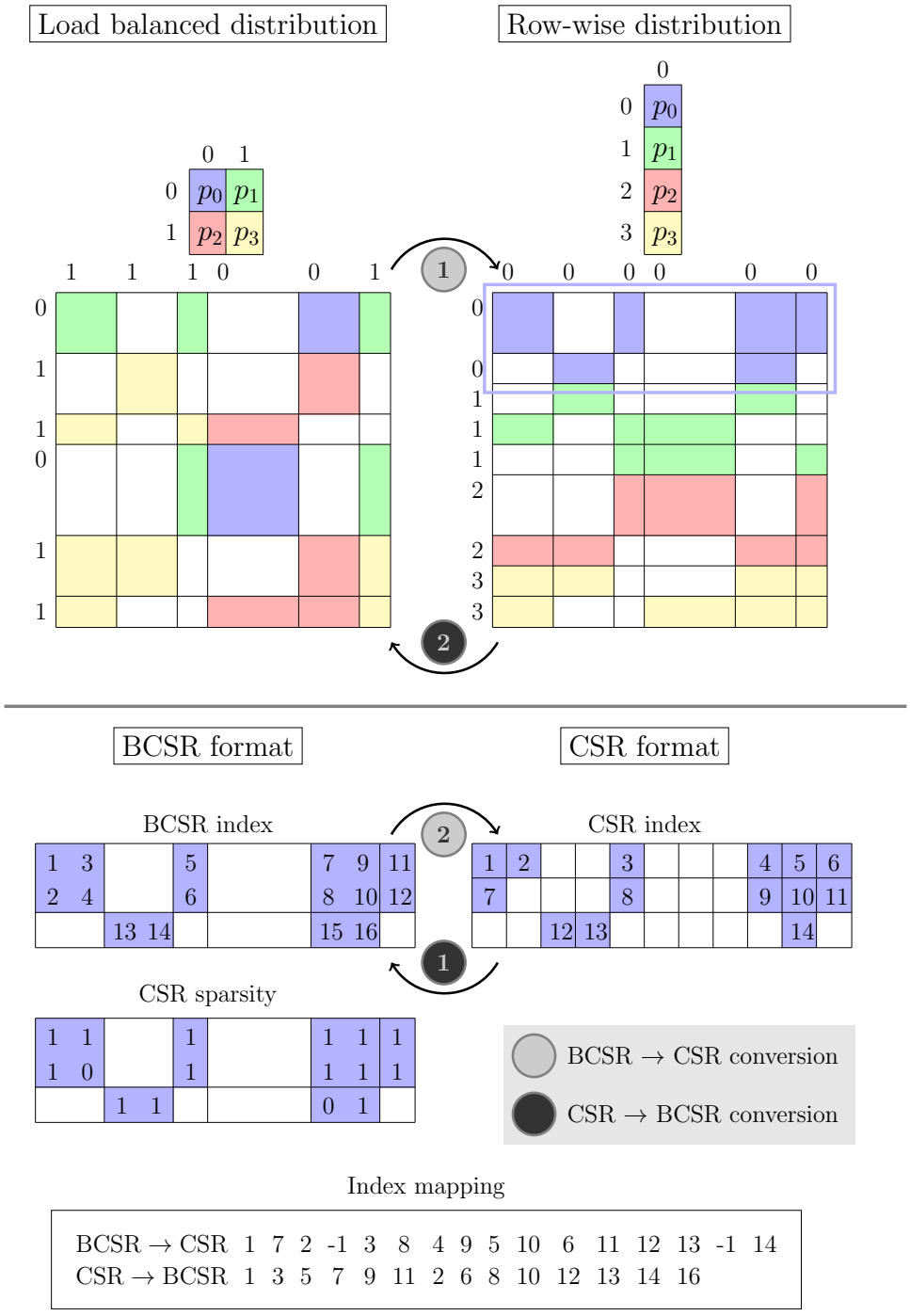


Figure 3.1: Depiction of the two steps of conversion between DBCSR and CSR format. *Upper panel:* Redistribution of a DBCSR matrix. *Lower panel:* Local conversion between BCSR and CSR format.

The number of processes P_{CSR} used for CSR matrices must not necessarily be the same as the total number of processes P (for PEXSI P_{CSR} is the number of processes per pole). In the case of the first two options, the target distribution requires a splitting of BCSR rows such that a BCSR row can be distributed over multiple processes. Conveniently, the DBCSR library of CP2K already provides a routine to redistribute a matrix according to an arbitrary row and column distribution and the remaining task is to set up a process grid, a row distribution and a column distribution that establish the target distribution.

3.2.2 Local conversion between BCSR and CSR matrix format

Once the matrix is redistributed, the conversion between the two matrix formats can be performed locally on each process and the task is to find a mapping between the two formats as they are depicted in Fig. 2.2. The relevant conversion data for an example matrix is illustrated in the lower panel of Fig. 3.1. The terms *BCSR index* and *CSR index* denote the order of the matrix entries as they are stored in the data array `nzval` of the two matrix formats, not to be mistaken for the terms CSR indices or BCSR indices, referring to the compressed representation of the row and column indices.

The first step of the local conversion is to create a mapping (called *index mapping*) between BCSR and CSR index. In the BCSR format, the matrix entries are sorted with respect to the number of the block they belong to (row-wise numbering of blocks) and for a given block, the elements are ordered in a column-wise fashion. In the CSR format, the ordering is row-wise. Practically, the index mapping is implemented by two vectors, one for each direction of conversion. The second step of the local conversion is to create the CSR `colind` and `rowptr` arrays.

Both steps must be aware of the sparsity pattern of the CSR matrix, i.e. that certain elements explicitly stored in BCSR format should be zeroed (removed) when going from BCSR to CSR and, conversely, that elements in BCSR blocks that are not present in CSR must be explicitly added. In practice, the sparsity pattern of a CSR matrix is represented by a BCSR matrix containing only 0 (for zero elements) and 1 (for non-zero elements). This matrix to which we refer as *CSR sparsity matrix* can, for instance, be obtained by applying a filtering threshold $\varepsilon_{\text{filter}}$ to set elements to 0 that have an absolute value below $\varepsilon_{\text{filter}}$. Due to the explicit specification of the CSR sparsity matrix at input,

the matrix conversion can deal with any other definition of sparsity, given an external routine that sets up the sparsity matrix. The sparsity matrix must have exactly the same block structure and BCSR indices as the BCSR matrix to be converted. It is redistributed in the same way as the BCSR matrix such that the local conversion can directly look up whether a BCSR matrix element is zero in CSR format or not. This information is then completely integrated into the index mapping. The CSR $\rightarrow$ BCSR mapping vector considers only explicitly present CSR matrix entries. The BCSR $\rightarrow$ CSR mapping vector contains a -1 for all BCSR matrix entries that are not present in CSR format.

The index mapping is created by iterating over the blocks of the BCSR matrix in a row-wise fashion. The DBCSR library provides an iterator that returns the block data as well as the indices needed to locate the block in the matrix. From the position of a matrix entry in a BCSR block, its position in the CSR format is explicitly calculated. This requires knowledge of the number of the matrix entries in all blocks stored in BCSR rows above the current block, the total number of non-zero matrix columns in the current BCSR row and the number of non-zero matrix columns on the left of the current block. The number of columns in each BCSR row is not directly accessible from local block data and must be obtained by a preceding iteration over BCSR blocks. In this approach, the sparsity pattern of the CSR matrix is first assumed to be the same as that of the BCSR matrix. In a second step, the index mapping is modified to correctly take the CSR sparsity into account.

After the index mapping has been established, the conversion from BCSR to CSR format can be performed. This conversion sends the data from BCSR `nzval` to the correct position in CSR `nzval` and calculates the CSR `colind` and `rowptr` arrays. These arrays are explicitly computed by iterating over BCSR blocks - in a similar way as for the calculation of the index mapping. In subsequent conversions for a conserved sparsity pattern, the CSR indices don't need to be recalculated and copying the data and the redistribution of the matrix are the only steps required.

3.2.3 Implementation

The CSR matrix type contains all data that is required to quickly convert from and to DBCSR format: the index mapping, the row-wise distributed intermediate DBCSR matrix, the CSR index as well as some convenient global matrix data (total number of non-zero elements, number of rows and columns

and the MPI communicator). The supported data types are complex and real, single and double precision.

The DBCSR matrix format is chosen according to the chemistry of the system and therefore, the conversion code can not derive a DBCSR matrix from a given CSR matrix. Thus it always requires a completely allocated DBCSR matrix format at input and can derive the CSR matrix from it. The creation of the CSR matrix from a given DBCSR matrix is separated from the actual conversion of the matrix data. The creation establishes the index mapping and allocates all fields of the CSR type. The conversion then merely needs to copy and redistribute the matrix data.

A conversion assumes that the pair of matrices to be converted exactly has the same sparsity pattern and is distributed in the same way as the matrices at the time of the creation of the CSR matrix. If the sparsity pattern changes or the matrix data is redistributed, the CSR matrix must be recreated. The fact that once created CSR matrices can be reused for subsequent conversions, assuming a conserved sparsity pattern, makes the conversion vulnerable in an environment where the matrix sparsity may change. However for PEXSI, this is the appropriate design because PEXSI relies on a fixed sparsity pattern and the conversion must guarantee that the CSR format does not change during an SCF cycle. Several safeguards are built into the matrix conversion that will detect any change of sparsity such that the program terminates with an error message if the sparsity changes. For instance, the DBCSR indices of the original DBCSR matrix are stored inside the CSR type and before each conversion, it is asserted that the DBCSR matrix to be converted has exactly the same indices.

The conversion can also handle DBCSR matrices in symmetric format (only the upper diagonal part is stored) by desymmetrising the matrix before conversion. Similarly it can return a symmetric matrix when converting from CSR to DBCSR format.

The matrix conversion tool has been tested by creating a set of random BCSR matrices, converting them to CSR format and back to BCSR format and checking that the original matrices are exactly restored. To test the sparsity refinement feature that removes certain matrix entries when going from BCSR to CSR format, the CSR sparsity has been defined by applying a filtering threshold $\varepsilon_{\text{filter}}$ to the elements of the BCSR matrix. It has been verified that the elements of the back-converted DBCSR matrix do not differ from the original DBCSR matrix by more than $\varepsilon_{\text{filter}}$. The validity of the CSR indices has been tested by creating a print routine that writes a CSR matrix to a file. The output was

compared with the output of a DBCSR print routine for a set of matrices.

3.3 Matrix sparsity and distance screening

PEXSI imposes a sparsity pattern to the density matrix $\boldsymbol{P}$ according to the locality of basis functions. More precisely, the criterion is that we only need the elements P_{ij} for which an $\boldsymbol{r}$ exists such that $\varphi_i(\boldsymbol{r})\varphi_j(\boldsymbol{r}) \neq 0$. By design of PEXSI, the sparsity of all matrices $\boldsymbol{S}$, $\boldsymbol{H}$ and $\boldsymbol{P}$ is exactly the same and the matrices differ by value only. The sparsity pattern of an example overlap matrix is depicted in Fig. 3.2.

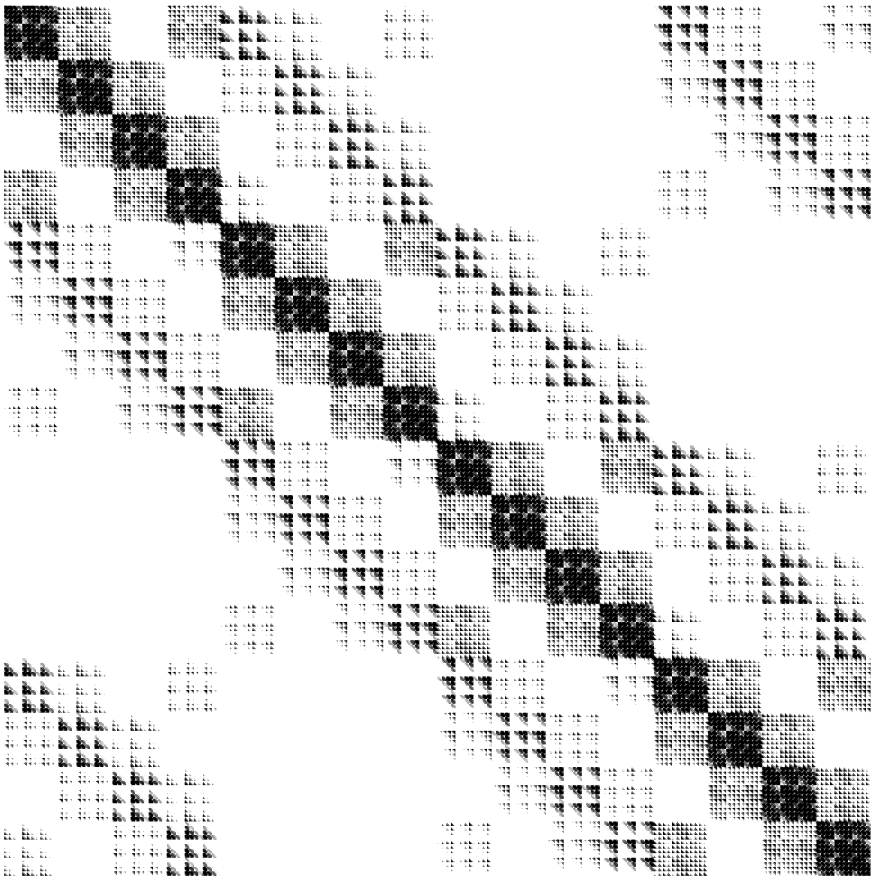


Figure 3.2: Sparsity of the overlap matrix for a 3-layer fcc(100) aluminium surface consisting of 768 atoms, treated with a DZVP basis set. In this case, the percentage of non-zero elements amounts to 13%. For larger systems the sparsity can drop down to less than 1%, a limit that is achieved for the case of this systems for 10000 atoms and more.

The CP2K-PEXSI interface needs to take care that the sparsity of $\boldsymbol{H}$ and $\boldsymbol{S}$ is in terms with this criterion. In practice, getting sparsity out of this criterion requires an approximation that effectively localises Gaussian basis functions

according to some truncation threshold. The sparsity pattern should be held fixed over an entire SCF cycle because the basis functions stay the same. Fixing sparsity also has some practical advantages: the matrix conversion needs to be initialised only once, i.e. only one CSR matrix needs to be created at the beginning of an SCF cycle and subsequent conversions only need to copy data as the matrix indices don't change. Similarly PEXSI can remember the form of the $\boldsymbol{LU}$ factorisation from previous steps.

The time required for the PEXSI DFT driver is proportional to the number of non-zero elements in the Cholesky factor which in turn depends on the sparsity of the overlap matrix and on the dimensionality of the system. Thus an accurate upper bound criterion for $|\varphi_i(\boldsymbol{r})\varphi_j(\boldsymbol{r})|$ is important to be able to optimally tune the sparsity of the matrices.

3.3.1 Sparsity in CP2K

In CP2K, matrix sparsity is obtained by the application of two different criteria:

1. *Distance screening*: to each primitive Gaussian basis function $\phi_i(\boldsymbol{r}) = r^{l_i} c_i \exp(-\alpha_i r^2)$, a radius r_i is assigned given some cutoff threshold ε according to $|\phi_i(\boldsymbol{r})| < \varepsilon$ for all $|\boldsymbol{r}| > r_i$. Elements of a matrix $\boldsymbol{M}$ of the form $M_{ij} = \langle\varphi_i|\hat{O}|\varphi_j\rangle$ are set to zero if there is no contribution from two Gaussians ϕ_i and ϕ_j that satisfy $r_i + r_j > d_{ij}$, where d_{ij} is the distance between the atomic centers of the two basis functions.

2. *Filtering*: an atomic block is removed if its Frobenius norm is below some threshold $\varepsilon_{\text{filter}}$ [42].

The first criterion is suitable to determine matrix sparsity for PEXSI because it is based on the locality of basis functions. Filtering should not be used in CP2K-PEXSI, because an atomic block being small by value does not necessarily imply an upper bound for the products of two basis functions.

As CP2K and especially its linear scaling implementation do not rely on a fixed sparsity pattern for all matrices, they can handle the sparsity of a matrix more flexibly and use the filtering method to improve the sparsity at any given point [2]. Consequently, the distance screening criterion is not meant to be highly selective and the respective cutoff threshold `eps_pgf_orb` is applied for a preselection of non-zero elements only. The main advantage of the CP2K implementation of distance screening compared to a more accurate criterion is its efficiency: it only needs to compute the radius of each basis function instead

of evaluating upper bounds for all pairs of basis functions. For PEXSI, an additional strict upper bound criterion applied to pairs of basis functions would be preferable but is not implemented in CP2K up to date. For the time being, the sparsity of PEXSI matrices is determined by an extension of the standard distance screening criterion of CP2K to the CSR matrix format.

3.3.2 Extending distance screening to CSR matrices

As mentioned before, the BCSR format of CP2K implies a sparsity pattern in terms of either full (non-zero) or empty (zero) atomic blocks. More precisely, an atomic block is set to zero if there is no pair of Gaussians that contributes to any matrix entry in this block. Thus the existence of an atomic block is essentially determined by the most extended Gaussian contained in a basis set. The extension of distance screening to single matrix elements will significantly improve the sparsity if the contracted basis functions $\varphi_i(\boldsymbol{r})$ contained in a basis set differ in their spatial extension. It does not introduce any additional approximation and the threshold applied is the same `eps_pgf_orb` that is also used by CP2K. The sparsity gain for an example overlap matrix is illustrated in Fig. 3.3.

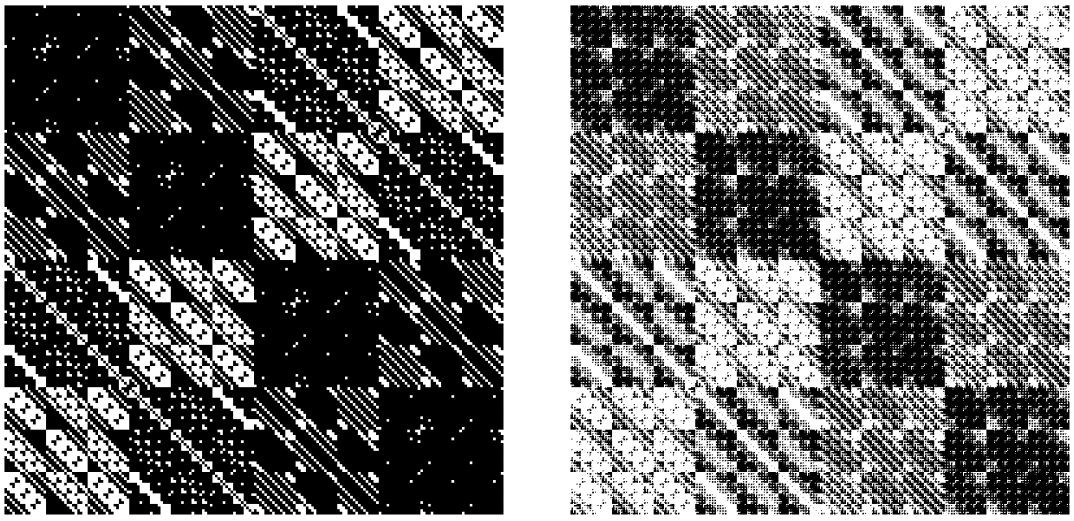


Figure 3.3: Sparsity of the overlap matrix in BCSR format (left handside) and CSR format (right handside) for a 3-layer fcc(100) aluminium surface consisting of 192 atoms. In BCSR format, the non-zero elements (black) are defined in terms of full atomic blocks while the CSR format allows for a refinement of the non-zero elements to single basis functions. For this example, the percentage of BCSR non-zero elements is 76% and the percentage of CSR non-zero elements is 51%.

3.4 CP2K-PEXSI interface

PEXSI has been integrated into the linear scaling SCF part of the CP2K code. The main responsibility of the interface to PEXSI is to determine a valid set of PEXSI options from the user input, to convert the matrices to the PEXSI CSR format and to print some PEXSI related output.

The input to CP2K-PEXSI is nearly identical to the options that are required by the PEXSI library. This gives the user flexibility while reasonable default values should reduce the complexity of setting up a CP2K-PEXSI calculation. PEXSI options that can be decided internally are not passed to the user (e.g. symbolic factorisation is only done in the first SCF step).

PEXSI has a two-level parallelism that uses a subgroup of all processes for the inversion of one pole such that parallelism over poles is achieved. The choice of the number of processes per pole is up to the user with the restriction that the number of processes per pole must divide the total number of processes without remainder. In CP2K-PEXSI the user input is the minimum number of processes per pole and the actual number of processes per pole is then determined as the smallest number greater than or equal to the user input such that a valid parallelisation of PEXSI is achieved.

Listing 3.1 mentions the most important steps in a CP2K-PEXSI calculation. Most of the steps in the CP2K-part were readily available and could be combined with PEXSI without profound changes to the CP2K source code. The only significant modification concerns the density mixing that was implemented in the standard SCF code of CP2K and not accessible in the linear scaling part of CP2K. The standard mixing method in the linear scaling environment is a linear mixing of the Kohn-Sham matrix. This works well for insulating systems but is not stable for metallic systems, requiring more advance mixing schemes on the level of the density matrix. The density mixing methods of CP2K (particularly Broyden mixing, see section 2.3) had to be refactored and integrated into the linear scaling code in order to make them available for PEXSI.

Inertia counting can be turned on or off by the user. For systems with a large band gap, the chemical potential μ returned by the previous SCF step is sufficiently exact and thus the inertia counting can be turned off for subsequent SCF steps. In the first SCF step, inertia counting is always performed. In PEXSI, inertia counting is automatically reinvoked if a PEXSI Newton iteration gives a difference in μ that is larger than some threshold. In our experience, turning off inertia counting works well also for metallic systems because the

found chemical potential should be already rather exact after a few SCF steps.

Listing 3.1 CP2K-PEXSI pseudocode for one SCF cycle

Distance screening: define sparsity pattern of all matrices $\boldsymbol{H}, \boldsymbol{S}, \boldsymbol{P}, \ldots$ by truncation of $\varphi_i(x)$.
Initialise CSR conversion with the overlap matrix S.
WHILE $|\mathcal{F}_{\text{new}} - \mathcal{F}_{\text{old}}| > \varepsilon_{\text{SCF}}$
  Calculate an upper bound ΔE for the spectral radius of $\boldsymbol{S}^{-1}\boldsymbol{H}$ using Arnoldi method.
  Convert $\boldsymbol{H}$ and $\boldsymbol{S}$ to the PEXSI CSR format.
  IF first SCF step
    Activate PEXSI inertia counting.
    Activate PEXSI symbolic factorisation.
  ELSE
    Activate PEXSI inertia counting only if requested by the user.
    Deactivate PEXSI symbolic factorisation.
  ENDIF
  CALL PEXSI DFT driver
    IN: $\boldsymbol{H}, \boldsymbol{S}, \Delta E$
    OUT: $\boldsymbol{P}, \boldsymbol{P}^{\text{E}}, -TS, \mu$
  Check convergence of PEXSI (error in the number of electrons).
  Convert $\boldsymbol{P}$ and $\boldsymbol{P}^{\text{E}}$ to the CP2K BCSR matrix format.
  Set μ as initial guess for the next SCF step.
  Compute the density $P \to \rho$ with Eq. (2.12).
  ρ mixing (if requested): $\rho^{\text{new}} = f_{\text{mix}}(\rho^{\text{new}}, \rho^{\text{old}})$.
  Construct $\mathcal{F} = E(\rho) - TS$ and $H_{ij} = \partial E / \partial P_{ij}$.
  $\boldsymbol{H}$ mixing (if requested): $\boldsymbol{H}^{\text{new}} = (1-\alpha)\boldsymbol{H}^{\text{new}} + \alpha \boldsymbol{H}^{\text{old}}$.
END WHILE
Calculate the forces $\nabla_{\text{A}}\mathcal{F} = f(\boldsymbol{P}, \boldsymbol{P}^{\text{E}}, \boldsymbol{H}, \boldsymbol{S})$ according to Eq. (2.22).

PEXSI needs an estimate for the upper bound ΔE for the eigenvalues ε_i of the generalised eigenvalue problem $(\boldsymbol{H}, \boldsymbol{S})$. This knowledge is required in order to correctly place the poles in the expansion of the Fermi function. An efficient way to obtain a good estimate for ΔE is to selectively calculate the maximum eigenvalue with the Arnoldi method.

Molecular dynamics also works out of the box thanks to the fact that PEXSI returns the additional quantities $\boldsymbol{P}^{\text{E}}$ and $-TS$ required to calculate the forces. These terms merely need to be passed to the corresponding variables in the Quickstep environment of CP2K.

The PEXSI interface is designed for the case of spin-restricted calculations in which each state is occupied with two electrons. The extension to the spin-unrestricted case (one electron per state) is achieved by scaling quantities with factors of 2 and 1/2 at the input and output of PEXSI.

As discussed before, the matrix conversion relies on a given sparsity and distribution of the BCSR matrices that must not change in an SCF cycle. For usual CP2K calculations this requirement does not strictly hold and the exact format of a matrix might change. The matrices $\boldsymbol{H}$ and $\boldsymbol{S}$ are therefore first copied to a template matrix with the correct format before conversion. This and the general design of the conversion tool reduce the dependency of the CP2K-PEXSI interface on the given CP2K data format such that only the format of the overlap matrix at initialization of the conversion tool is relevant.

The installation of CP2K-PEXSI including all dependencies is facilitated by a script that is shipped with the source code of CP2K that installs CP2K and PEXSI and automatically takes care of the linkage.

Chapter 4

Introduction to CP2K-PEXSI

4.1 Input

CP2K input that is relevant for the PEXSI part of a calculation is listed in Listing 4.1. For a complete documentation of the CP2K input, the reader is referred to the online documentation [43] and the example input files that come with the source code of CP2K. The input consists of keywords that are grouped into different sections. A CP2K-PEXSI calculation is requested by setting the keyword LS_SCF in the QS (Quickstep) section to .TRUE. and inside the LS_SCF subsection, the PURIFICATION_METHOD has to be set to PEXSI. The keyword EPS_PGF_ORB is the cutoff threshold for the radius of the primitive Gaussians which defines the matrix sparsity. Two subsections have been added in LS_SCF especially for PEXSI: PEXSI and RHO_MIXING. The PEXSI subsection contains all the relevant new input for PEXSI, and the RHO_MIXING section is used to request mixing on the level of the density instead of linear mixing of the Kohn-Sham matrix (necessary to achieve convergence for metallic systems).

In the following, a few crucial points to consider when requesting a CP2K-PEXSI calculation are explained. The PEXSI default values have chosen such that PEXSI is most efficient up to an accuracy of 10^{-6} a.u. per atom. Usually the only keywords that need to be set explicitly are EPS_PGF_ORB and MIN_RANKS_PER_POLE. In our experience, setting EPS_PGF_ORB to 1.0E-4 is appropriate for a requested accuracy of 10^{-6} a.u. per atom. The keyword MIN_RANKS_PER_POLE controls together with the total number of MPI ranks the parallelisation of PEXSI. It should be chosen according to the number of available processes and the minimum number of processes required to perform the selected inversion. As the parallel scalability over poles is almost perfect,

Listing 4.1 PEXSI-related CP2K input

```
&FORCE_EVAL
  &DFT
    &QS
      LS_SCF .TRUE.
      EPS_PGF_ORB 1.0E-4
    &END QS
    &LS_SCF
      PURIFICATION_METHOD PEXSI
      &PEXSI
        MIN_RANKS_PER_POLE 256
        NUM_POLE 60
        TEMPERATURE 300
        NUM_ELECTRON_PEXSI_TOLERANCE 0.001
        ⋮
      &END PEXSI
      &RHO_MIXING
        METHOD BROYDEN_MIXING
        ⋮
      &END RHO_MIXING
    &END LS_SCF
  &END DFT
&END FORCE_EVAL
```

the recommendation is to restrict the number of processes for selected inversion to a rather small value such that the parallelisation over poles can be exploited to a larger degree.

The most efficient way to run PEXSI in terms of the total wall time is to completely parallelise over the poles, in this case MIN_RANKS_PER_POLE is set to $n_{\text{proc}}/n_{\text{pole}}$ where n_{proc} is the total number of MPI ranks and n_{pole} is the total number of poles. An integer fraction of this number can be used if not enough cores for a full parallelisation are available. Note that it is beneficial for reaching peak performance of PEXSI to use a square number for MIN_RANKS_PER_POLE. As mentioned before, if MIN_RANKS_PER_POLE is not consistent with the total number of MPI ranks (i.e. if it does not divide the number of MPI ranks without remainder), this number is automatically increased to the next valid parallelisation.

If a higher accuracy if requested, the number of poles should be increased (NUM_POLE keyword) and especially for metallic systems, the convergence threshold in the number of electrons (NUM_ELECTRON_PEXSI_TOLERANCE) should be set to a smaller value. Raising the temperature to a higher value will

not affect the accuracy of the calculation for systems with a sufficiently large band gap but in this case, a smaller number of poles is needed to accurately represent the Fermi-Dirac function (the pole expansion requires a number of terms proportional to $\log(\beta\Delta E)$).

4.2 Efficiency and accuracy

For making efficient use of PEXSI, its parameters need to be tuned in such a way that optimal performance is achieved for a desired target accuracy. Here the following input options relevant for a CP2K-PEXSI calculation are discussed in terms of accuracy and computational cost:

- the truncation threshold `eps_pgf_orb` for distance screening,
- the number of poles to approximate the Fermi-Dirac function,
- the PEXSI tolerance in number of electrons,
- the electronic temperature and
- the parallelisation of PEXSI.

The options for inertia counting are not discussed, but we point out that using a sufficiently tight threshold for inertia counting can help keeping the number of expensive PEXSI Newton iterations relatively low. The CP2K default values for inertia counting should be accurate enough and stable for all systems we consider here, such that there is usually no need to change the default.

4.2.1 Distance screening

Fig. 4.1 shows the energy error, the time per SCF iteration and the sparsity in dependence of the truncation threshold for primitive Gaussian basis functions. In the current implementation, this `eps_pgf_orb` criterion fully determines the sparsity of the matrices $\boldsymbol{H}, \boldsymbol{S}$ and $\boldsymbol{P}$ when a CP2K-PEXSI calculation is performed. Using a higher truncation threshold leads to a significantly reduced number of non-zero elements and thus to savings in the computational costs. However, the `eps_pgf_orb` criterion seems to be unstable for thresholds greater than 10^{-4} as the error in energy quickly increases and thresholds greater 10^{-3} even cause a calculation to abort.

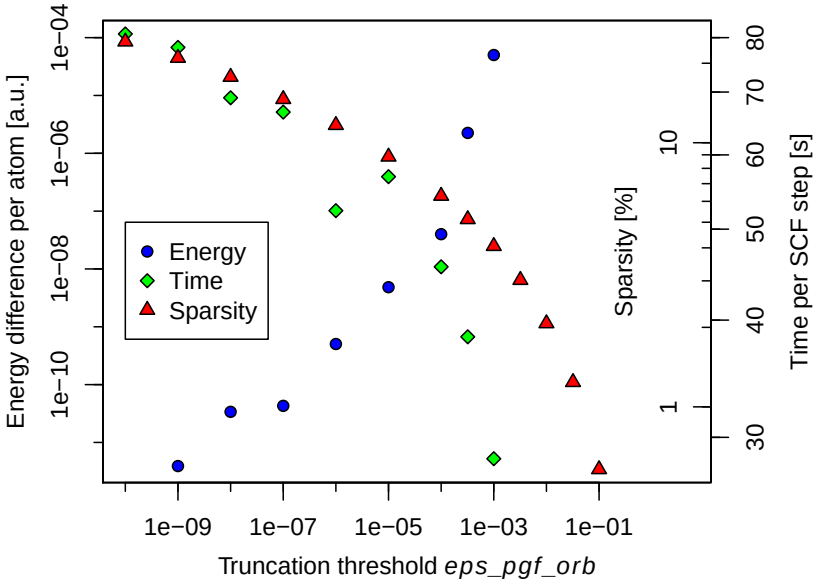


Figure 4.1: Error in energy, time per PEXSI SCF iteration and the sparsity (percentage of number of non-zero elements) of $\boldsymbol{H}, \boldsymbol{S}$ and $\boldsymbol{P}$ in dependence of the truncation threshold `eps_pgf_orb` for primitive Gaussian basis functions. The calculations were performed for the case of liquid water with 768 atoms, using a TZV2P basis set. The energy reference is for `eps_pgf_orb` $= 10^{-10}$. Choosing a higher threshold than `eps_pgf_orb` $= 10^{-3}$ causes calculations to abort.

4.2.2 Parallelisation

PEXSI has two levels of parallelism, the parallelisation over poles and the parallel selected inversion performed for each pole. As all poles can be processed independently, the parallelisation over poles is expected to be more efficient than exploiting the parallelism on the level of selected inversion. This is confirmed by Fig. 4.2 where we plot the PEXSI time in dependence of the number of processes per pole, for a fixed total number of processes. If more processes per pole are used, less poles can be inverted in parallel. Processing all poles in parallel is most efficient and increasing the number of processes for selected inversion leads to a significant increase in computation time. Considering that selected inversion scales well to 256 processes per pole for this specific system, the regime of almost perfect parallelisation of PEXSI extends to 12'800 processes (at full parallelisation for 50 poles).

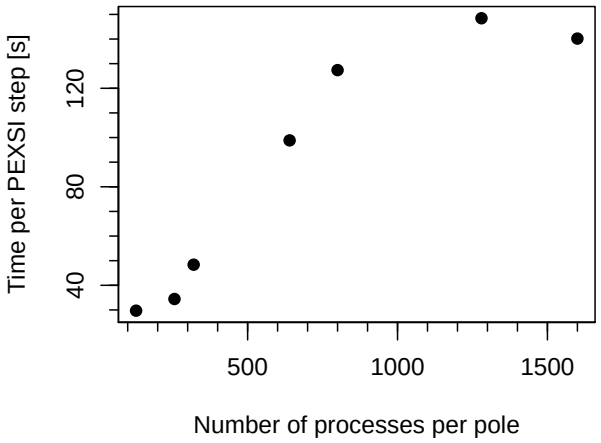


Figure 4.2: Performance of PEXSI in dependence of the number of processes per pole used for selected inversion. The system is liquid water consisting of 6144 atoms, parallelised over a total number of 6400 processes. We use 50 poles such that full parallelisation over poles is achieved at 128 processes per pole.

4.2.3 Insulating system

Fig. 4.4 shows the accuracy of the converged energy and the time per SCF step for an insulating system in dependence of the temperature and the number of poles. Due to the large band gap, the determination of an exact value of the chemical potential is not crucial for obtaining an accurate result and a value of 0.01 for the tolerance in the number of electrons can be safely used. As expected, a larger number of poles yields more accurate calculations but increases the total computation time. The electronic temperature is not considered as a physical parameter because for insulating system, raising the temperature does not significantly change the occupation of orbitals. However, raising the temperature leads to a smoother Fermi-Dirac function that is better approximated by the pole expansion and thus less poles are needed (the number of required terms is proportional to $\log(\beta\Delta E)$). It was found that using a tighter tolerance threshold for the number of electrons (0.001) will give an even more accurate result of 10^{-10} a.u. per atom for the case of 80 poles and a temperature of 100 K.

4.2.4 Metallic system

The accuracy and efficiency of PEXSI for the case of a metallic system is investigated in Fig. 4.5. In contrast to the insulating system considered before, changing the electronic temperature affects the energy due to a change in the occupation of orbitals. Due to the better approximation of the Fermi-Dirac function at higher temperature, less PEXSI iterations and consequently less time are required for an SCF step. For these calculations we employed a rather loose threshold for inertia counting (0.025 a.u. for μ) to illustrate the effect of the temperature and a tighter threshold may damp the effect of the temperature on the number of PEXSI iterations.

4.2.5 Molecular dynamics

Fig. 4.3 shows the kinetic energy and the constant of motion (kinetic energy + potential energy) over a CP2K-PEXSI MD simulation in the NVE ensemble of liquid water, DZVP basis, 96 atoms. The drift in the constant of motion is 5.5×10^{-7} a.u. per ps and atom. A standard SCF calculation for the same system yields a drift of 1.0×10^{-7} a.u. per ps and atom. This proves that the forces are correctly implemented in CP2K-PEXSI and that PEXSI is already quite accurate for the applied settings (60 poles, 0.001 tolerance in number of electrons).

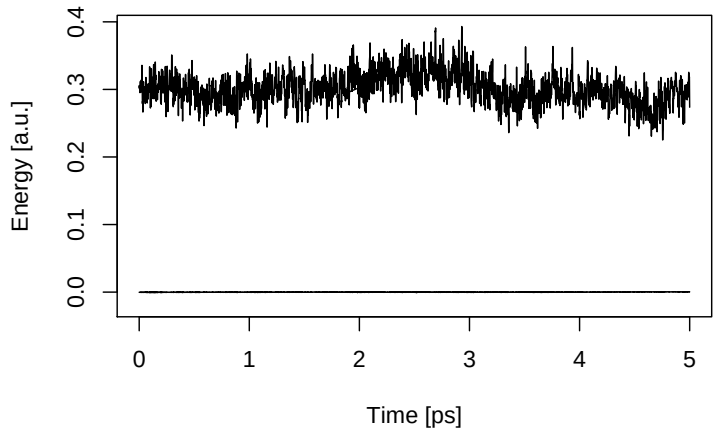


Figure 4.3: The kinetic energy (top) and the constant of motion (bottom) over a MD NVE simulation of liquid water containing 96 atoms at room temperature. A DZVP basis is used and the SCF convergence threshold is set to $\varepsilon_{\mathrm{SCF}} = 10^{-10}$ a.u. per atom. One timestep is 0.5 fs. For PEXSI, 60 poles and a tolerance threshold of 0.001 in number of electrons are applied, yielding a rather exact conservation of the constant of motion with a drift of only 5.5×10^{-7} a.u. per ps and atom.

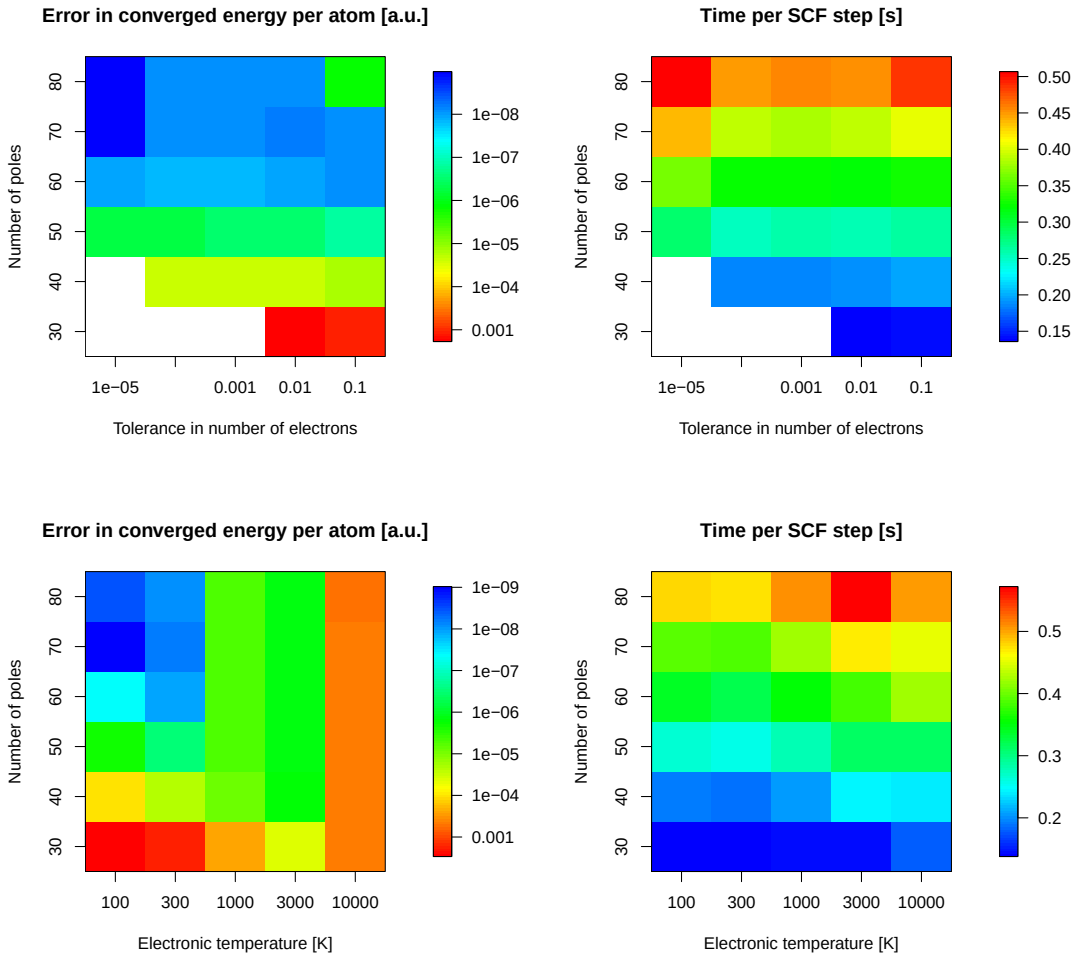


Figure 4.4: Accuracy and efficiency in dependence of the PEXSI parameters number of poles, tolerance in number of electrons and electronic temperature for the case of an insulating system (liquid water with 96 atoms, DZVP basis). For the upper panels, the temperature was set to 300 K and for the lower panels, the tolerance in number of electrons was set to 0.01. The results refer to converged SCF calculations using linear mixing of the Kohn-Sham matrix. The reference energy is obtained by a linear scaling calculation for the same system at zero temperature. White spots represent failed calculations due to an insufficient number of poles.

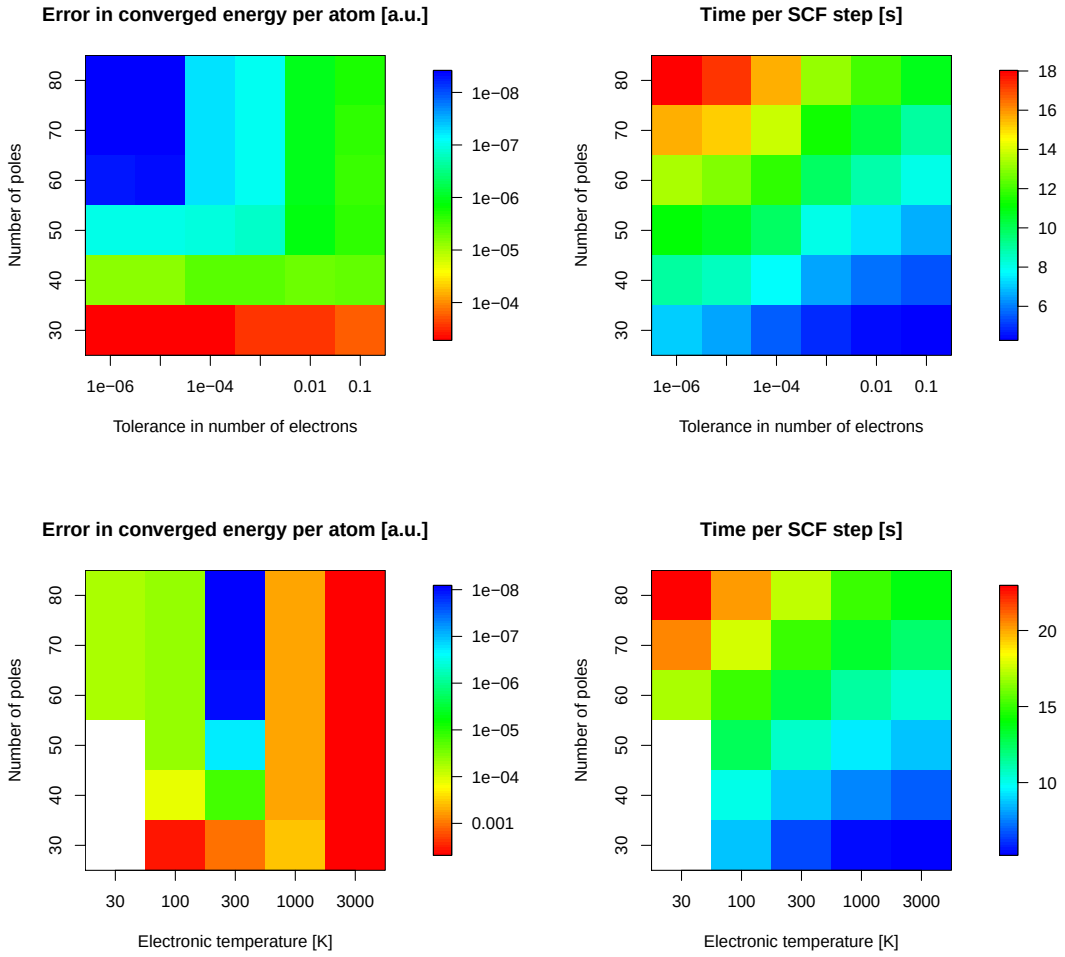


Figure 4.5: Accuracy and efficiency in dependence of the PEXSI parameters number of poles and electronic temperature for the case of a metallic system (3-layer fcc(100) aluminium surface, 192 atoms, DZVP basis). The results refer to converged SCF calculations using Broyden density mixing. The reference energy is obtained by a standard CP2K SCF calculation for the same system at an electronic temperature of 300 K.

Chapter 5

Comparison of PEXSI with other CP2K SCF methods

In the following, the three methods PEXSI, linear scaling and standard diagonalisation are compared with respect to their parallel scalability and the scaling of their computational costs with system size. The parallel scalability is investigated by measuring CPU time vs. the number of processes at fixed system size. CPU time is defined as the total wall time multiplied with the number of processes. Perfect parallel scalability implies that CPU time stays the same when the number of processes is increased. For the scaling with system size, wall time vs. number of atoms is evaluated at fixed number of processes. We perform diagonalisation with ScaLAPACK 2.0.2 [9]. The calculations are performed on the Cray XC30 machine Piz Daint of the Swiss National Supercomputing Centre, employing only MPI parallelisation (no GPU's, no threads).

All comparisons measure the computational time of the first two SCF steps. A comparison between PEXSI, linear scaling and diagonalisation has to take into account that PEXSI needs more Newton iterations in the first SCF steps to find the correct chemical potential. In subsequent iterations, the chemical potential from previous iterations is already accurate such that less PEXSI iterations are required. One fully converged SCF cycle was carried out for each system to derive the mean number of required PEXSI iterations per SCF step (assuming that this does not depend on the system size). To make a fair comparison, the measured time for the first two SCF steps was then rescaled to the mean number of PEXSI iterations per SCF step.

5.1 Systems and parameters

In order to explore the full domain of applicability of PEXSI, insulating and metallic systems and quasi-2D and 3D systems are considered: liquid water representing a sparse 3D insulating system, a fcc(100) aluminium surface with 3 atomic layers representing a metallic 2D system, monolayer graphene representing a thin semi-metallic 2D system, silicon representing a semi-conducting 3D system and diamond at high temperature which behaves as a 3D metallic system. The properties as well as the results of a full SCF calculation for these systems are listed in Tab. 5.1. For all systems, the number of poles for PEXSI is 50 and for the inertia counting procedure, a threshold of $\Delta\mu = 0.005$ a.u. is applied. The distance screening criterion `eps_pgf_orb` is set to 10^{-4}. The full SCF calculations are carried out for a target accuracy of $\varepsilon_{\mathrm{SCF}} = 10^{-7}$ a.u. per electron. The calculations for water are performed with different choices of basis sets: DZVP-GTH, DZVP-MOLOPT-SR-GTH and TZV2P-GTH as implemented in CP2K. For all other systems the CP2K basis set DZVP-GTH-PADE-CONFINED is applied which is the preferred CP2K DZVP basis in terms of sparsity.

Table 5.1: Properties, PEXSI parameters and results from a converged SCF calculation for all systems considered here. The abbreviations refer to the following quantities: the electronic temperature T_{el}, the number of basis functions per atom N_{Basis} (number of valence electrons $\times$ number of basis functions per electron), the number of processes per pole n_{pp}, the number of atoms N_{A}, the PEXSI tolerance in number of electrons ΔN_{e}, the number of required SCF steps N_{scf} to convergence, the average number of PEXSI iterations per SCF step N_{PEXSI} and the energy error ΔE_{SCF} per atom in a.u. as obtained by a comparison with a standard CP2K SCF or linear scaling calculation.

System	Dim.	T_{el} [K]	Basis	N_{Basis}	n_{pp}	N_{A}	ΔN_{e}	N_{scf}	N_{PEXSI}	ΔE_{SCF}
H_2O	3D	300	DZVP	$6 \times 13, 1 \times 5$	256	769	0.01	7	1.0	2.5×10^{-7}
			TZV2P	$6 \times 22, 1 \times 9$				7	1.1	1.8×10^{-5}
			DZVP-M	$6 \times 13, 1 \times 5$				8	1.0	1.3×10^{-6}
Al	2D	300	DZVP	3×13	256	432	0.001	22	3.1	6.2×10^{-6}
C	2D	300	DZVP	4×13	256	648	0.001	17	3.4	3.4×10^{-6}
Si	3D	300	DZVP	4×13	256	216	0.01	15	2.2	2.0×10^{-6}
C	3D	3000	DZVP	4×13	640	512	0.001	20	2.3	4.3×10^{-6}

The number of PEXSI iterations over a full SCF cycle is presented in Fig. 5.1 for all systems. Water requires much less PEXSI iterations due to its large band gap. For the other systems, PEXSI needs considerably more iterations in the first SCF steps and after the tenth SCF step, never more than 3 PEXSI iterations are required.

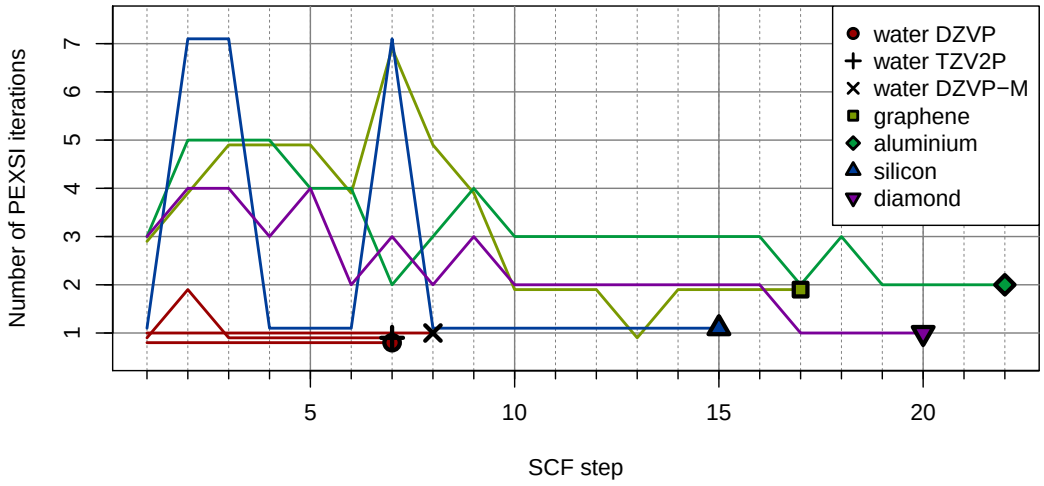


Figure 5.1: Number of PEXSI Newton iteration needed to find the density matrix at the correct chemical potential over an SCF cycle. The much faster convergence of water is explained by the application of a different mixing method (mixing of Kohn-Sham matrix instead of density mixing).

The computational cost of PEXSI is proportional to the number of non-zero elements in the Cholesky factor (or $\boldsymbol{L}+\boldsymbol{U}$ in $\boldsymbol{LU}$ factorisation). The sparsity of the Cholesky factor depends on the dimensionality of the system and the sparsity of the overlap matrix. Its number of non-zero elements increases with $\mathcal{O}(N_\mathrm{e}^2)$ for 3D bulk systems and $\mathcal{O}(N_\mathrm{e}^{3/2})$ for quasi-2D systems. The number of non-zero elements vs. the system size for the overlap matrix and the Cholesky factor is depicted in Fig. 5.2. The matrix sparsity is tabulated in Tab. 5.2. It can clearly be seen that quasi-2D systems (aluminium and graphene) have a generally sparser Cholesky factor than 3D systems. Graphene has a sparser representation than aluminium because it is very thin. Comparing 3D systems, liquid water has a favourable sparsity because it is separated into weakly-interacting molecules while condensed systems as diamond and silicon are much less sparse.

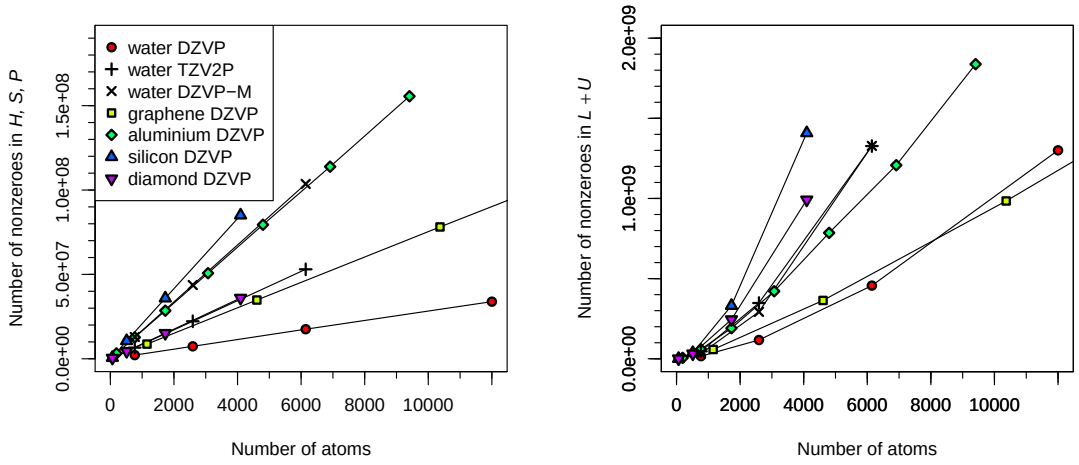


Figure 5.2: Number of matrix non-zero elements in dependence of the system size. The left panel depicts the number of non-zero elements in the overlap matrix. In the right panel, the number of non-zero elements in the matrix $\boldsymbol{L}+\boldsymbol{U}$ of PEXSI's $\boldsymbol{LU}$ factorisation is plotted.

Table 5.2: Percentage of non-zero elements in the matrices $\boldsymbol{H}, \boldsymbol{S}, \boldsymbol{P}$ and in $\boldsymbol{L}+\boldsymbol{U}$ for different system sizes (number of atoms) N_A.

(a) H_2O 3D DZVP

N_A	$\boldsymbol{S}$	$\boldsymbol{L}+\boldsymbol{U}$
768	6.3	47.2
2592	1.9	29.7
6144	0.8	20.6
12000	0.4	15.4

(b) H_2O 3D TZV2P

N_A	$\boldsymbol{S}$	$\boldsymbol{L}+\boldsymbol{U}$
768	6.3	44.2
2592	1.9	29.1
6144	0.8	19.8

(c) H_2O 3D DZVP-M

N_A	$\boldsymbol{S}$	$\boldsymbol{L}+\boldsymbol{U}$
768	37.1	93.1
2592	11.1	74.2
6144	4.7	59.8

(d) Al 2D DZVP

N_A	$\boldsymbol{S}$	$\boldsymbol{L}+\boldsymbol{U}$
192	50.8	93.0
768	12.7	59.5
1728	5.7	37.8
3072	3.2	26.4
4800	2.0	20.2
6912	1.4	15.0
9408	1.0	12.3

(e) C 2D DZVP

N_A	$\boldsymbol{S}$	$\boldsymbol{L}+\boldsymbol{U}$
1152	3.9	25.8
4608	1.0	10.1
10368	0.4	5.4
18432	0.2	3.4
28800	0.2	2.5

(f) Si 3D DZVP

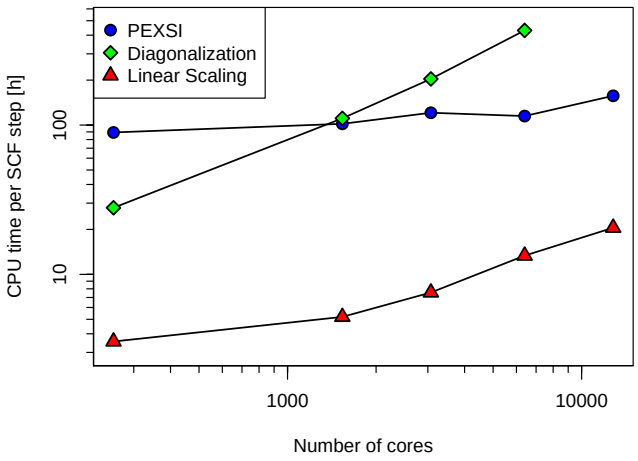
N_A	$\boldsymbol{S}$	$\boldsymbol{L}+\boldsymbol{U}$
64	90.8	98.6
512	24.0	84.0
1728	7.1	65.7
4096	3.0	49.7

(g) C 3D DZVP

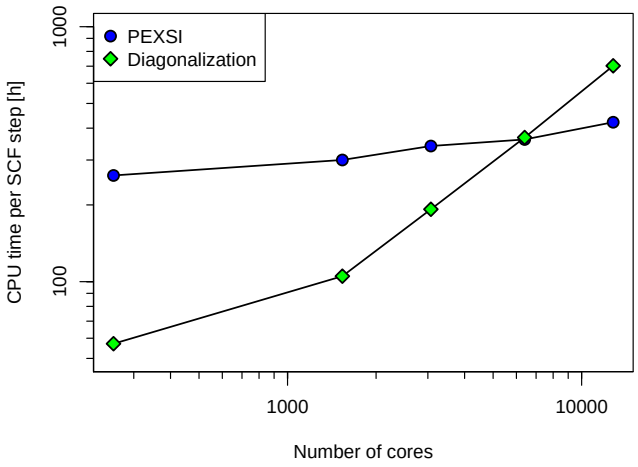
N_A	$\boldsymbol{S}$	$\boldsymbol{L}+\boldsymbol{U}$
64	67.4	96.8
512	10.1	71.3
1728	3.0	48.5
4096	1.3	35.0

5.2 Parallel scalability

The parallel scalability of PEXSI, linear scaling and diagonalisation for liquid water and aluminium surface is presented in Fig. 5.3. For PEXSI, the number of processes per pole is held constant such that the parallelisation over poles, not the parallel scalability of selected inversion is investigated. PEXSI is generally slower than linear scaling and faster than diagonalisation for more than 1000 cores (water) or more than 7000 cores (aluminium). Due to PEXSI's favourable scaling with system size, for larger systems a smaller number of cores will be sufficient for making efficient use of PEXSI. The parallel scalability of PEXSI is very good due to the parallelisation over independent poles. At the maximum number of cores (12800), full parallelisation of PEXSI is achieved such that the regime of almost perfect parallel scalability comes to an end at this number (see also Fig. 4.2). The parallel scalability of diagonalisation is not very good in general but in terms of computational resources, diagonalisation is almost always more economical than PEXSI because diagonalisation is more efficient at a smaller number of cores.



(a) Liquid water, 6144 atoms, DZVP basis.



(b) 2D aluminium with 3 atomic layers, 4800 atoms, DZVP basis.

Figure 5.3: Parallel scalability comparing PEXSI with diagonalisation and linear scaling as implemented in CP2K for two systems. Complete parallelisation of PEXSI is achieved at the largest number of cores (12800). CPU time is defined as total wall-time times number of cores.

5.3 Scaling with system size

5.3.1 Bulk liquid water

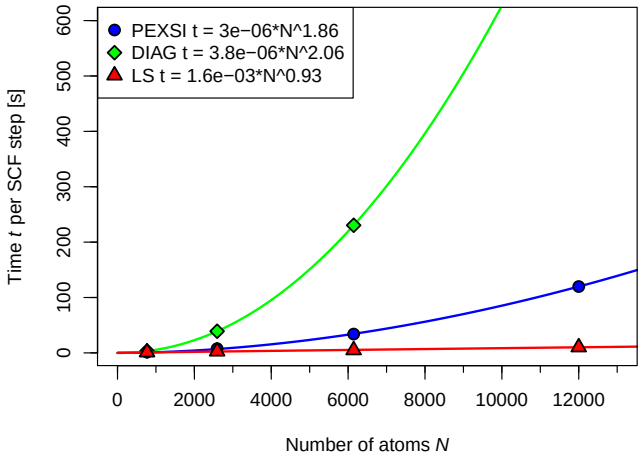
The benchmarks for variation of the system size for liquid water are presented in Fig. 5.4 for different choices of basis sets. In order to verify the expected algorithmic scaling with respect to system size, a power law was fitted to each benchmark curve. The expected scaling is $\mathcal{O}(N_\mathrm{e}^3)$ for standard diagonalisation, $\mathcal{O}(N_\mathrm{e}^2)$ for PEXSI and $\mathcal{O}(N_\mathrm{e})$ for linear scaling. Due to cache and parallelisation effects, the observed scaling can significantly differ from the algorithmic scaling. Standard diagonalisation can effectively use more processors for larger system size, leading to an effective scaling that is better than the asymptotical $\mathcal{O}(N_\mathrm{e}^3)$ scaling for a large number of processors.

The basis set has a great influence on the performance of PEXSI and linear scaling. Employing a larger basis set (TZV2P) leads to more non-zero elements in the overlap matrix and thus PEXSI and linear scaling have a larger cost (see also Fig. 5.2). Standard diagonalisation is not affected as much by the basis even though a larger basis set leads to a larger matrix size. For PEXSI, the crucial criterion for optimal performance is the localisation of basis functions. For linear scaling, the condition number of the overlap matrix is important because this determines the sparsity of S^{-1} that needs to be calculated. The molopt basis set of CP2K [44] for instance is optimised with respect to the condition number of the overlap matrix but has a less sparse overlap matrix than standard DZVP basis. Consequently, linear scaling performs well with molopt basis while PEXSI performs worse than with standard DZVP basis.

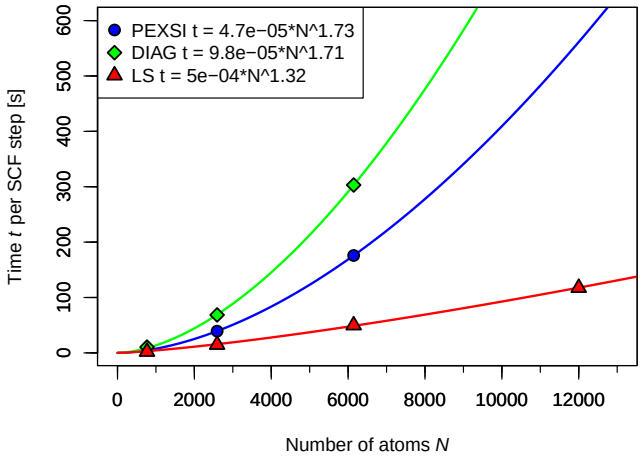
Note that in the benchmarks considered here, 6400 cores and 256 processes per pole are applied (see Tab. 5.1). 25 poles can be processed in parallel and two consecutive steps are required to invert all 50 poles. Consequently, a speed-up of PEXSI by roughly a factor of 2 can be gained by doubling the number of cores.

5.3.2 Quasi-2D systems

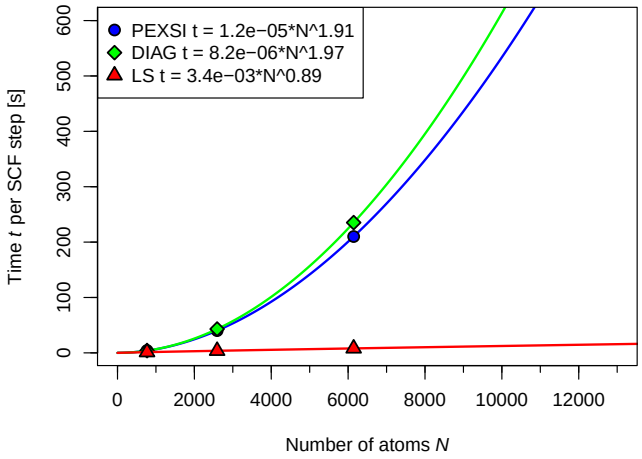
The scaling with respect to system size for the quasi-2D systems is depicted in Fig. 5.5, comparing PEXSI with standard diagonalisation. Note that linear scaling is not applicable to these systems because of their metallic or semi-metallic nature. The expected scaling of PEXSI is $\mathcal{O}(N_\mathrm{e}^{1.5})$. Because the monolayer graphene system has a very sparse representation, PEXSI performs much bet-



(a) Water, DZVP-GTH basis



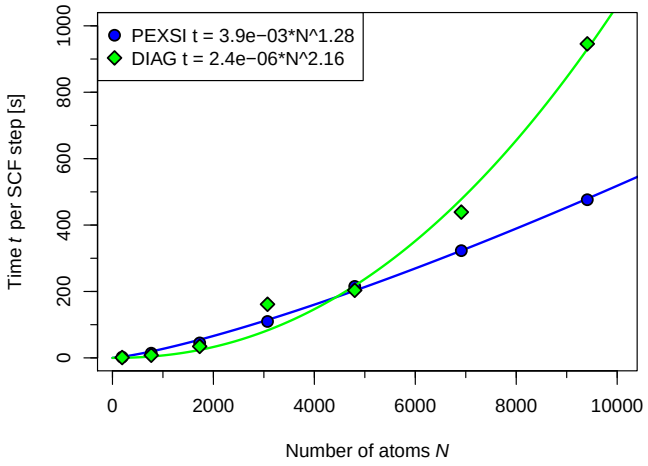
(b) Water, TZV2P-GTH basis



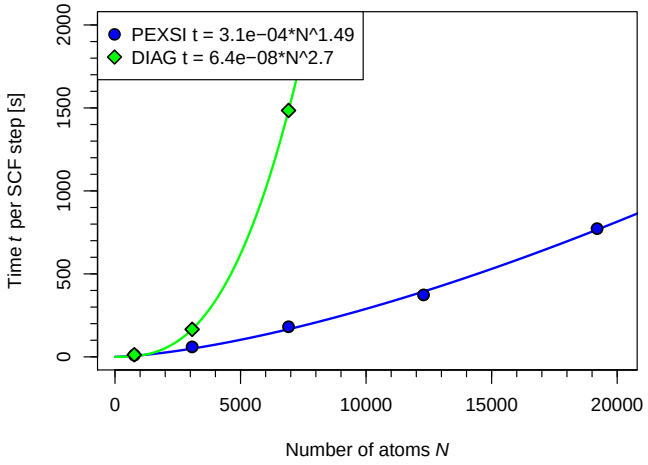
(c) Water, DZVP-MOLOPT-SR-GTH basis

Figure 5.4: Scaling of the computational costs with respect to system size for liquid water and different choices of basis sets, comparing PEXSI, diagonalisation and linear scaling. The benchmarks are performed on 6400 cores.

ter than standard diagonalisation. The aluminium system is less sparse but PEXSI becomes more efficient than diagonalisation for 5000 atoms and more. Again, the poles are processed in two consecutive steps for the applied parallelisation.



(a) Aluminium surface with 3 atomic layers, DZVP basis



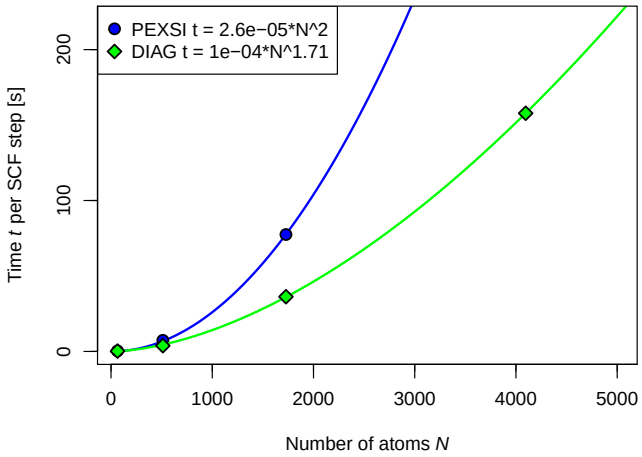
(b) Monolayer graphene, DZVP basis

Figure 5.5: Scaling of the computational costs with respect to system size for quasi-2D systems comparing PEXSI and diagonalisation. The benchmarks are performed on 6400 cores.

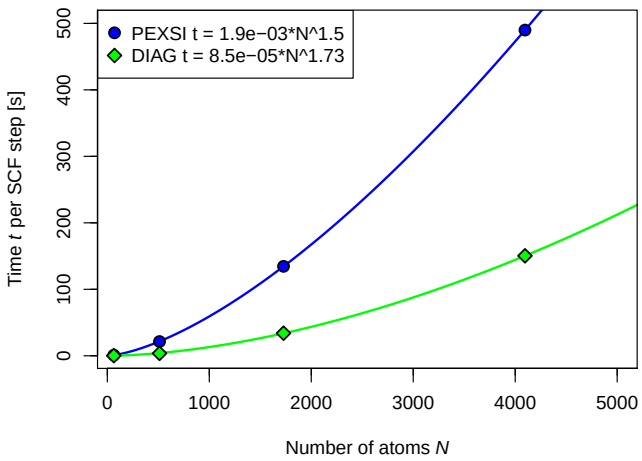
5.3.3 Condensed bulk systems

The last systems considered are condensed 3D systems with metallic character (diamond at an electronic temperature of 3000 K) or semi-conducting character

(silicon at room temperature). Compared to the liquid water systems, these systems are less sparse and consequently, the costs of a PEXSI calculation are higher. The parallelisation of the diamond system was not chosen in an optimal way (640 processes per pole) such that 5 consecutive steps for the inversion of poles are needed. By reducing the number of processes per pole and increasing the total number of processes such that full parallelisation is achieved, a maximum speed-up of a factor of 5 can be expected for diamond with PEXSI. For silicon, a factor of 2 can be gained at full parallelisation. Even in this limit, PEXSI will not be significantly faster than standard diagonalisation.



(a) Silicon at 300 K, DZVP basis



(b) Diamond at 3000 K, DZVP basis

Figure 5.6: Scaling of the computational costs with respect to system size for 3D condensed systems with semi-conducting character (silicon) and metallic character (diamond). The benchmarks are performed on 6400 cores.

Chapter 6

Discussion and Conclusion

This Master thesis was mainly concerned with making the PEXSI (Pole EXpansion and Selected Inversion) method available in the atomistic simulation package CP2K. In a second step this method was evaluated by comparison with linear scaling and standard diagonalisation. The calculations performed for a wide range of different systems prove that the current implementation is stable and accurate for electronic structure calculations and ab initio molecular dynamics.

PEXSI is most useful for metallic systems and systems with a small band gap because for these systems, the linear scaling approach fails. For insulating systems, linear scaling clearly outperforms the PEXSI method. PEXSI was proven to be the most efficient method (in terms of total wall time) implemented in CP2K for large quasi-2D metallic systems containing a few thousand atoms, if more than 1000 processors are available. Large systems are advantageous because PEXSI scales at most quadratically with the system size. A massive parallelisation favours the high parallel scalability of PEXSI over independent poles up to 10'000 processors or even more. On the other hand, diagonalisation scales cubically with respect to the system size and can not make efficient use of a large number of processors. This being said, diagonalisation was shown to be generally cheaper in terms of CPU time because it is more efficient than PEXSI for a smaller degree of parallelisation.

PEXSI can be more efficient than diagonalisation also for 3D systems due to the favourable scaling of $\mathcal{O}(N_\mathrm{e}2)$ instead of $\mathcal{O}(N_\mathrm{e}^3)$ with respect to the system size (number of electrons N_e). In practice, a significant speed-up of PEXSI compared to diagonalisation could be observed only for liquid water but not for condensed systems due to an insufficiently sparse Cholesky factor. For quasi-2D system, PEXSI scales with $\mathcal{O}(N_\mathrm{e}^{1.5})$. The required system size and degree of

parallelisation where PEXSI becomes faster than diagonalisation depend on the sparsity of the system. For a 3 atomic layer thick aluminium surface, PEXSI is significantly faster for 3000 atoms and more if a parallelisation over more than 5000 processors is applied. For monolayer graphene, PEXSI almost immediately outperforms diagonalisation.

The results obtained in this work can not be directly compared to the SIESTA-PEXSI approach because SIESTA uses numerical basis functions [45, 46] while CP2K is based on Gaussian-type functions. Numerical orbitals are advantageous with respect to the sparsity of the overlap matrix because they can be chosen to be strictly local while the localisation of Gaussian-type functions is an approximation involving the neglect of small matrix elements. This being said, the sparsity obtained in CP2K is already comparable to the sparsity obtained with SIESTA. For instance for liquid water with 12000 atoms and DZVP basis, the sparsity is 0.29% (12% for Cholesky factor) in SIESTA [35] and 0.40% (15% for Cholesky factor) in CP2K. Some gain for the sparsity of the overlap matrix in CP2K can be expected by applying a more accurate screening criterion that sets matrix elements to zero based on an upper bound for the product of two primitive Gaussians instead of enforcing localised basis functions based on a truncation threshold. A further gain could be achieved by creating basis sets optimised with respect to the sparsity of the overlap matrix.

Diagonalisation is a direct and exact method while PEXSI has many parameters that must be tuned with respect to efficiency and accuracy. In the current implementation of CP2K-PEXSI, the user must decide on the parallelisation over poles, the number of poles to approximate the Fermi-Dirac function, the convergence threshold in number of electrons and the truncation threshold for the basis functions. Additionally the thresholds for the inertia counting procedure must be set sufficiently tight to reduce the number of expensive PEXSI Newton iterations. The accuracy and efficiency of CP2K-PEXSI was evaluated in dependence of all relevant parameters, for a metallic and an insulating system. The default values were chosen such that for most systems, they should give good accuracy and reasonable performance.

Some savings in computational costs could possibly be achieved by adapting the accuracy required for PEXSI to the accuracy that can be achieved within a certain SCF step instead of keeping the parameters fixed over an entire SCF cycle. An automatic determination of the PEXSI parameters as a function of the requested accuracy would be preferable over to the current implementation. This is however not a trivial task to achieve due to the complex interplay

between different parameters (for instance temperature and number of poles) and the system dependent implications (metallic vs. non-metallic, system size) of many parameters.

Currently, the main limitation for a broad application of the PEXSI approach to a wide range of systems is its high computational costs in the sense that PEXSI starts to be beneficial only at large system sizes and requires massive parallelisation. This limitation may be removed in the future by the availability of better high-performance computers.

Acknowledgements

I want to thank Prof. Joost VandeVondele for having offered me this most interesting topic for my Master thesis that gave me valuable insights into recent algorithmic developments in the field of Kohn-Sham density functional theory. I want to thank my supervisor Mohammad Hossein Bani-Hashemian for his support in many respects that contributed a lot to a successfully finished project. I want to thank Prof. Lin Lin of University of California, Berkeley, for providing the PEXSI library, in particular I want to point out the cleanly written interface that saved a lot of work during the integration to CP2K. I'm also grateful for the support I received from his side as well as his collaboration in the evaluation of the results received from CP2K-PEXSI.

Further thanks go to all members of the Nanoscale Simulations group for their help and advice. Last but not least, I want to thank Prof. Nicola Spaldin for the coffee supply.

Bibliography

[1] S. Goedecker, "Linear scaling electronic structure methods," *Reviews of Modern Physics*, vol. 71, no. 4, p. 1085, 1999.

[2] J. VandeVondele, U. Bortnik, and J. Hutter, "Linear scaling self-consistent field calculations with millions of atoms in the condensed phase," *Journal of Chemical Theory and Computation*, vol. 8, no. 10, pp. 3565–3573, 2012.

[3] L. Lin, M. Chen, C. Yang, and L. He, "Accelerating atomic orbital-based electronic structure calculation via pole expansion and selected inversion," *Journal of Physics: Condensed Matter*, vol. 25, no. 29, p. 295501, 2013.

[4] P. Hohenberg and W. Kohn, "Inhomogeneous electron gas," *Physical review*, vol. 136, no. 3B, p. B864, 1964.

[5] W. Kohn and L. J. Sham, "Self-consistent equations including exchange and correlation effects," *Physical Review*, vol. 140, no. 4A, p. A1133, 1965.

[6] N. D. Mermin, "Thermal properties of the inhomogeneous electron gas," *Physical Review*, vol. 137, no. 5A, p. A1441, 1965.

[7] R. M. Wentzcovitch, J. L. Martins, and P. B. Allen, "Energy versus free-energy conservation in first-principles molecular dynamics," *Physical Review B*, vol. 45, no. 19, p. 11372, 1992.

[8] M. Weinert and J. Davenport, "Fractional occupations and density-functional energies and forces," *Physical Review B*, vol. 45, no. 23, p. 13709, 1992.

[9] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.

[10] R. McWeeny, "Some recent advances in density matrix theory," *Reviews of Modern Physics*, vol. 32, no. 2, p. 335, 1960.

[11] A. M. Niklasson, C. Tymczak, and M. Challacombe, "Trace resetting density matrix purification in o (n) self-consistent-field theory," *The Journal of chemical physics*, vol. 118, no. 19, pp. 8611–8620, 2003.

[12] W. Kohn, "Density functional and density matrix method scaling linearly with the number of atoms," *Physical Review Letters*, vol. 76, no. 17, p. 3168, 1996.

[13] S. Goedecker and G. E. Scuseria, "Linear scaling electronic structure methods in chemistry and physics," *Computing in Science & Engineering*, vol. 5, no. 4, pp. 14–21, 2003.

[14] E. Prodan and W. Kohn, "Nearsightedness of electronic matter," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 33, pp. 11635–11638, 2005.

[15] P. E. Maslen, C. Ochsenfeld, C. A. White, M. S. Lee, and M. Head-Gordon, "Locality and sparsity of ab initio one-particle density matrices and localized orbitals," *The Journal of Physical Chemistry A*, vol. 102, no. 12, pp. 2215–2222, 1998.

[16] L. Lin, J. Lu, L. Ying, and W. E, "Pole-based approximation of the fermi-dirac function," *Chinese Annals of Mathematics, Series B*, vol. 30, no. 6, pp. 729–742, 2009.

[17] L. Lin, J. Lu, L. Ying, R. Car, and W. E, "Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems," *Communications in Mathematical Sciences*, vol. 7, no. 3, pp. 755–777, 2009.

[18] L. Lin, C. Yang, J. C. Meza, J. Lu, and L. Ying, "Selinv – an algorithm for selected inversion of a sparse symmetric matrix," *ACM Transactions on Mathematical Software (TOMS)*, vol. 37, no. 4, p. 40, 2011.

[19] L. Lin, C. Yang, J. Lu, and L. Ying, "A fast parallel algorithm for selected inversion of structured sparse matrices with application to 2d electronic structure calculations," *SIAM Journal on Scientific Computing*, vol. 33, no. 3, pp. 1329–1351, 2011.

[20] M. Jacquelin, L. Lin, and C. Yang, "Pselinv – a distributed memory parallel algorithm for selected inversion: the symmetric case," *arXiv:1404.0447*, 2014.

[21] P. Bendt and A. Zunger, "New approach for solving the density-functional self-consistent-field problem," *Physical Review B*, vol. 26, no. 6, p. 3114, 1982.

[22] J. E. Dennis, Jr and J. J. Moré, "Quasi-newton methods, motivation and theory," *SIAM review*, vol. 19, no. 1, pp. 46–89, 1977.

[23] G. Broyden, "A class of methods for solving simultaneous nonlinear equations," *Math. Comput*, vol. 19, pp. 577–593, 1965.

[24] J. Hutter, "Car–parrinello molecular dynamics," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 2, no. 4, pp. 604–612, 2012.

[25] J. VandeVondele, M. Krack, F. Mohamed, M. Parrinello, T. Chassaing, and J. Hutter, "Quickstep: Fast and accurate density functional calculations using a mixed gaussian and plane waves approach," *Computer Physics Communications*, vol. 167, no. 2, pp. 103–128, 2005.

[26] O. F. Sankey and D. J. Niklewski, "Ab initio multicenter tight-binding model for molecular-dynamics simulations and other applications in covalent systems," *Physical Review B*, vol. 40, no. 6, p. 3979, 1989.

[27] P. Pulay, "Ab initio calculation of force constants and equilibrium geometries in polyatomic molecules: I. theory," *Molecular Physics*, vol. 17, no. 2, pp. 197–204, 1969.

[28] M. Krack and M. Parrinello, "Quickstep: make the atoms dance," *High Performance Computing in Chemistry*, vol. 25, p. 29, 2004.

[29] G. Lippert, J. Hutter, and M. Parrinello, "A hybrid gaussian and plane wave density functional scheme," *Molecular Physics*, vol. 92, no. 3, pp. 477–488, 1997.

[30] G. Lippert, J. Hutter, and M. Parrinello, "The gaussian and augmented-plane-wave density functional method for ab initio molecular dynamics simulations," *Theoretical Chemistry Accounts*, vol. 103, no. 2, pp. 124–140, 1999.

[31] "CP2K basis sets." `http://www.cp2k.org/basis_sets`. Accessed: 2015-23-03.

[32] H. B. Schlegel and M. J. Frisch, "Transformation between cartesian and pure spherical harmonic gaussians," *International Journal of Quantum Chemistry*, vol. 54, no. 2, pp. 83–87, 1995.

[33] S. Goedecker, M. Teter, and J. Hutter, "Separable dual-space gaussian pseudopotentials," *Physical Review B*, vol. 54, no. 3, p. 1703, 1996.

[34] C. Hartwigsen, S. Gœdecker, and J. Hutter, "Relativistic separable dual-space gaussian pseudopotentials from h to rn," *Physical Review B*, vol. 58, no. 7, p. 3641, 1998.

[35] L. Lin, A. García, G. Huhs, and C. Yang, "SIESTA-PEXSI: massively parallel method for efficient and accurate ab initio materials simulation without matrix diagonalization," *Journal of Physics: Condensed Matter*, vol. 26, no. 30, p. 305503, 2014.

[36] A. Alavi, J. Kohanoff, M. Parrinello, and D. Frenkel, "Ab initio molecular dynamics with excited electrons," *Physical review letters*, vol. 73, no. 19, p. 2599, 1994.

[37] "Documentation of the PEXSI library." `http://www.pexsi.org`. Accessed: 2015-23-03.

[38] X. S. Li and J. W. Demmel, "SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems," *ACM Trans. Mathematical Software*, vol. 29, pp. 110–140, June 2003.

[39] G. Karypis and V. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 71–95, 1998.

[40] C. Chevalier and F. Pellegrini, "Pt-scotch: A tool for efficient parallel graph ordering," *Parallel Computing*, vol. 34, no. 6, pp. 318–331, 2008.

[41] J. Dongarra, "Compressed Row Storage (CRS)." `http://netlib.org/linalg/html_templates/node91.html`. Accessed: 2015-23-03.

[42] U. Borštnik, J. VandeVondele, V. Weber, and J. Hutter, "Sparse matrix multiplication: The distributed block-compressed sparse row library," *Parallel Computing*, vol. 40, no. 5, pp. 47–58, 2014.

[43] "CP2K input manual." `http://manual.cp2k.org/trunk/`. Accessed: 2015-23-03.

[44] J. VandeVondele and J. Hutter, "Gaussian basis sets for accurate calculations on molecular systems in gas and condensed phases," *The Journal of chemical physics*, vol. 127, no. 11, p. 114105, 2007.

[45] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, and D. Sánchez-Portal, "The siesta method for ab initio order-n materials simulation," *Journal of Physics: Condensed Matter*, vol. 14, no. 11, p. 2745, 2002.

[46] E. Artacho, E. Anglada, O. Diéguez, J. D. Gale, A. García, J. Junquera, R. M. Martin, P. Ordejón, J. M. Pruneda, D. Sánchez-Portal, *et al.*, "The siesta method; developments and applicability," *Journal of Physics: Condensed Matter*, vol. 20, no. 6, p. 064208, 2008.