Modular implementation of linear and cubic-scaling approaches based on the orbital minimization method in electronic structure codes using atomic orbitals

Irina V. Lebedeva,1,2,3,4 Alberto García,4 Emilio Artacho,1,5,6,7 and Pablo Ordejón2

1CIC nanoGUNE BRTA, 20018 Donostia-San Sebastián, Spain
2Catalan Institute of Nanoscience and Nanotechnology - ICN2, CSIC and BIST, Campus UAB, 08193 Bellaterra, Spain
3Simune Atomistics, Avenida de Tolosa 76, 20018 Donostia-San Sebastián, Spain
4Institut de Ciencia de Materiais de Barcelona (ICMAB-CSIC), 0893 Bellaterra, Spain
5Donostia International Physics Center DIPC, 20018 Donostia-San Sebastián, Spain
6Theory of Condensed Matter, Cavendish Laboratory, University of Cambridge, Cambridge CB3 0HE, UK

We present a code modularization approach to design efficient and massively parallel cubic and linear-scaling solvers in electronic structure codes. The modular implementation of the orbital minimization method (OMM), in which linear algebra and parallelization issues are handled via external libraries, is demonstrated in the SIESTA code. The DBCSR (Distributed Block Compressed Sparse Row) library is used for algebraic operations with sparse matrices, while ScaLAPACK (Scalable Linear Algebra PACKage) is used for dense matrices. The MatrixSwitch and libOMM libraries, recently developed within the Electronic Structure Library (ESL), facilitate switching between different matrix formats. We show results comparing the performance of several cubic-scaling algorithms, and also demonstrate the parallel performance of the linear-scaling solvers, and their supremacy over the cubic-scaling solvers for system sizes of several hundreds of atoms.

INTRODUCTION

The success of electronic structure theory1 in modeling new materials and devices2–6 has stimulated the development of hundreds of electronic structure codes. Historically almost all of these individual software packages are written in distinct ways, although many tasks performed are similar. Except for numerical and performance related libraries such as basic linear algebra subroutines (BLAS)7 higher-level linear algebra utilities (serial Linear Algebra PACKage - LAPACK8 and its parallel counterpart, Scalable LAPACK - ScaLAPACK9), message passing interface (MPI) level9 etc., significant parts of the codes are replicated with some variations. Electronic structure packages are growing rapidly incorporating more and more new features. Also the codes have to adapt to the constant hardware evolution, which in the case of monolithic code architecture implies significant efforts on re-engineering. In this situation, it seems more efficient to change the traditional monolithic paradigm of software development to the modular one in which common tasks arise. In addition, such an approach allows to separate tasks related to high-level routines focused on the calculation of physical properties from underlying routines for parallelization and algebra. Significant efforts (e.g., the European project MAX10) are underway to stimulate the paradigm change in software design. Here we show the benefits of modularization on SIESTA11–15 as example.

SIESTA11–15 was specifically designed for linear-scaling calculations16–18 in which the computational time grows linearly with the number of atoms19,20. Such methods make possible calculations of large systems at a considerably less computational cost compared to common cubic-scaling approaches. SIESTA uses strictly localized atomic-like functions for basis sets in which the Hamiltonian and overlap matrices, H and S, are sparse. If, additionally, the confinement of the wavefunctions is imposed, the coefficient matrix C expanding wavefunctions in the basis is also sparse. Reducing the problem of solving the Kohn-Sham equations to the minimization of a properly constructed energy functional within the Ordejón-Mauri16–18 and Kin19 approaches, the inversion of the overlap matrix is avoided and only expressions involving products and sums of sparse H, S and C matrices need to be computed, all in linear-scaling effort.

The linear-scaling solvers in SIESTA, although available from the start12 are not widely used in practice. One of the reasons is that the implementation of these physical methods involved also coding of the algebra and parallelization of sparse matrices, which inevitably increased the code complexity and hindered progress. Recent efforts on linear-scaling methods have produced the Distributed Block Compressed Sparse Row (DBCSR) library22 that efficiently handles algebraic operations for sparse matrices and is massively parallelized23,24. Using this library, we have implemented an improved and more reliable version of linear-scaling solvers in SIESTA (Fig. 1).

Another recent initiative that has helped re-designing SIESTA is the Electronic Structure Library (ESL), a collaboration platform for shared software development. We use ESL’s libOMM library25 distributed within the omm-bundle26. It encodes the Ordejón-Mauri16–18 and Kin19 functionals, originally without the additional approximation of wave-function confinement, rendering dense C matrices and cubic scaling. Such an approach provides an alternative to conventional cubic-scaling methods, which can be faster in long simulations by avoiding computationally expensive orthonormalization and using history on previous steps. We refer to unconstrained minimization methods of suitable energy functionals, with either linear or cubic scaling, as the orbital minimization method (OMM)9,10,15,18. In libOMM25,26–28 the minimization is customarily performed via conjugate gradients (CG). The parameters of the quartic function describing the energy dependence along the search direction are computed analytically16,17.

Although the original libOMM library provides cubic...
Calculating CdS.

Allocating the auxiliary matrix A.

Allocating the I matrix.

Allocating the SW matrix.

Allocating the CdS matrix of type m_storage and size n x m (the number of wavefunctions x the basis size) consisting of blocks of size b_m x b_m.

Allocating the SW matrix.

Allocating the I matrix.

Allocating the auxiliary matrix A.

Calculating CdS.

Listing 1. An example of the calculation of the total charge in the electronic structure code SIESTA. The red rectangular box corresponds to SIESTA. Blue ellipses indicate the libraries used. The libraries in the dashed frame belong to the Electronic Structure Library (ESL). The arrows demonstrate calls to the libraries.

RESULTS

Overview of OMM approaches

In density functional theory (DFT), the problem of finding the ground state of a many-electron system is reduced to an energy minimization for the system of 2n non-interacting electrons moving in an effective potential and described by one-particle states \( | \psi_i \rangle \) \( (i = 1, \ldots, n) \) each of which is occupied by two electrons of opposite spin (assuming no spin polarization, for simplicity). The set of states \( \{| \psi_i \rangle \} \) is one of the many possible bases in the occupied subspace of the Hilbert space of the system and can be chosen orthonormal or not. In the latter case, the overlap matrix \( S_W \) with the elements \( (S_W)_{ij} = \langle \psi_i | \psi_j \rangle \) is not the identity matrix \( (S_W)_{ij} = \delta_{ij} \) and the density matrix operator that determines the projection onto the occupied subspace is then given by

\[
\hat{\rho} = \frac{1}{2} \sum_{i,j=1}^{n} | \psi_i \rangle \langle \psi_j | S_W^{-1} | \psi_j \rangle \langle \psi_i | ,
\]

(1)

involving the inverse of \( S_W \). The corresponding band structure energy becomes

\[
E = \text{Tr} \left[ \hat{H} \hat{\rho} \right] = 2 \text{Tr} \left[ S_W^{-1} H_W \right].
\]

(2)

where \( \hat{H} \) is the Hamiltonian operator and \( H_W \) is the corresponding matrix with the elements \( (H_W)_{ij} = \langle \psi_i | \hat{H} | \psi_j \rangle \). Note...
that the traces in Eq. (2) are taken on spaces of different dimensions: the size of the basis set for the first, and of the occupied states in the second. Also, the second equality holds for zero temperature.

In the basis of \( m \) functions \( \{|\phi_i\rangle\} \) (strictly localized atomic orbitals in SIESTA)

\[
|\phi_i\rangle = \sum_{\mu=1}^{m} C_i^\mu |\phi_\mu\rangle,
\]

where we refer to \( C \) as the coefficient matrix. Then \( H_W = C^\dagger H C \) and \( S_W = C^\dagger S C \), where \( H_{ij} = \langle \phi_i | H | \phi_j \rangle \), \( S_{ij} = \langle \phi_i | S | \phi_j \rangle \) and \( C^\dagger \) is the Hermitian conjugate of \( C \). The energy functional in Eq. (2) is minimized to find the ground state energy. The most common approach is direct diagonalization of the Hamiltonian matrix \( H \) (an \( m \times m \) matrix for the basis set of size \( m \)). Energy and charge density are then obtained using the wavefunctions and energies of the \( n \) lowest eigenstates. In contrast, in the iterative approaches, the energy is minimized with respect to variations in the states \( \{|\phi_i\rangle\} \). Here one needs to calculate the inverse of the overlap matrix \( S_W^{-1} \) or impose the orthonormality condition \( S_W \phi_{ij} = \delta_{ij} \). In any case, the computational time increases as \( O(n^3) \) with the system size, while the memory required to store the wavefunctions grows as \( O(n^2) \).

In OMM approaches, \[ \text{the expensive orthonormalization step is avoided via the modification of the energy functional in such a way that it automatically induces the orthonormalization of the wavefunctions during minimization:} \]

\[
\tilde{E} = 2 \text{Tr} \left[ (I_W + (I_W - S_W))H_W \right] = 2 \text{Tr} \left[ (2I_W - C^\dagger S C)C^\dagger H C \right] .
\]

This expression can be derived from consideration of Lagrange multipliers or expansion of the inverse overlap matrix to first order in the deviation from the identity \( S_W^{-1} \approx I_W + (I_W - S_W) \). The solution obtained from Eq. (4) is the same as from Eq. (2).

Within the same approximation, the density matrix of Eq. (1) is computed as

\[
\rho = C(I_W + (I_W - S_W))C^\dagger = 2C(2I_W - C^\dagger S C)C^\dagger
\]

and the forces on atom \( I \) as

\[
F_I = -\text{Tr} \left[ \rho \frac{\partial H}{\partial R_I} \right] + \text{Tr} \left[ \rho_E \frac{\partial S}{\partial R_I} \right],
\]

where we refer to \( \rho_E = 2CH \eta C^\dagger \) as the “energy density”.

If the basis functions and wavefunctions are chosen to be strictly localized, the Hamiltonian, overlap and coefficient matrices, \( H, S \) and \( C \), are sparse and \( O(n) \) scaling with system size is achieved. Note that this is not the case for Eqs. (1) and (2) as the inverse of \( S \) is not sparse (although sub-cubic scaling can be achieved using selected inversion to compute just the needed elements of the inverse). In the case of periodic systems, localized wavefunctions are close to the Wannier functions that decay exponentially with the distance from the center of localization in insulators and in metals at a finite temperature. Imposing localization constraints on the wavefunctions, however, leads to a deviation from the exact solution of Eqs. (2) and (4). Also the localized wavefunctions obtained are not strictly orthonormal and do not comply with the system symmetries. However, the degree of approximation can be controlled with the cutoff radius \( R_C \) for the wavefunctions.

In the Ordejón-Mauri functional the localization of the wavefunctions gives rise to many shallow local minima and flat regions in which the algorithm can be trapped for a long time during the energy minimization. This problem is solved in the Kim functional by including unoccupied states and introducing a chemical potential \( \eta \), i.e. the energy separating occupied and unoccupied states. The corresponding functional is obtained by (1) an eigenspectrum shift \( H \rightarrow H - \eta \mathcal{S} \), (2) changing dimensions of \( C \) from \( m \times n \) to \( m \times n' \), where \( n' > n \), and (3) changing the energy functional in Eq. (4) as \( \tilde{E} \rightarrow \tilde{E} + \eta \mathcal{E} \), and energy density \( \rho_E \) in Eq. (6) as \( \rho_E \rightarrow \rho_E + \eta \mathcal{S} \). It should be noted, however, that although the multiple-minima problem is solved in the Kim functional, it is sometimes hard to choose a proper value for \( \eta \). It should always lie within the band gap, but the bands can move up and down during self-consistency or molecular dynamics (MD), \( \eta \) possibly getting into the valence or conduction bands and, as a result, converging to an erroneous solution. Care should be taken to ensure that the solution reproduces the correct number \( 2n \) of electrons.

If the localization constraints on the wavefunctions are removed, the exact solution of Eqs. (2) and (4) is obtained.

In this case, however, one energy minimization can demand many CG iterations. This relates to the problem of length-scale or kinetic energy ill-conditioning. The efficiency of the CG algorithm depends on the ratio of the maximal and minimal extremal curvatures of the function minimized, which in OMM are determined by the maximal and minimal eigenvalues of the Hamiltonian. The eigenspectrum of the Hamiltonian is broad given the large kinetic energy of high-energy eigenstates. Although such states contribute negligibly to the ground-state solution, the problem becomes ill-conditioned and the convergence is slow. It is, however, possible effectively to reduce the width of the eigenspectrum by suppressing the kinetic energy contribution of high-energy states through preconditioning by which the CG gradient matrix is multiplied by the preconditioning matrix. Even in this case, however, one energy minimization can demand many CG iterations. This relates to the problem of length-scale or kinetic energy ill-conditioning. The efficiency of the CG algorithm depends on the ratio of the maximal and minimal extremal curvatures of the function minimized, which in OMM are determined by the maximal and minimal eigenvalues of the Hamiltonian. The eigenspectrum of the Hamiltonian is broad given the large kinetic energy of high-energy eigenstates. Although such states contribute negligibly to the ground-state solution, the problem becomes ill-conditioned and the convergence is slow. It is, however, possible effectively to reduce the width of the eigenspectrum by suppressing the kinetic energy contribution of high-energy states through preconditioning by which the CG gradient matrix is multiplied by the preconditioning matrix.
The list of the principal input parameters should be set to CSR (compressed sparse row) format. The density matrix \( \rho \), as an input, the Hamiltonian and overlap matrices in the basis \( H \) and \( S \), and the information on the system geometry. SIESTA uses for matrices the \( H \) and \( S \) as an input the Hamiltonian and overlap matrices in the basis format. The density matrix is used to update \( H \) for the next SCF step outside the solver. At the end of each MD step, the solver is called again to compute the energy density \( E \) and \( S \), which are received by the solver in the \( CSR \) format. The density matrix \( \rho \) is the output, also in \( CSR \) (see Eq. (3)). This matrix is used to update \( H \) for the next SCF step outside the solver. At the end of each MD step, the solver is called again to compute the energy density matrix \( \rho_E \), which, along with \( \rho \), is later used to calculate forces (see Eq. (4)) and stresses. The scheme of the \( \rho_E \) calculation is analogous to that of \( \rho \) shown in Fig. 2.

To use the implemented OMM solver, SolutionMethod in the SIESTA input parameters should be set to BLOMM (OMM with Block matrices). The list of the principal input parameters related to the solver is given in Table 1.

### TABLE 1. Principal input parameters for the revised OMM solver (SolutionMethod BLOMM) and their default values.

| Input parameter       | Default value | Description                                      |
|-----------------------|---------------|--------------------------------------------------|
| OMM.UseSparse         | true          | Whether to use sparse matrices                   |
| OMM.UseKimFunctional  | true          | Whether to use the Kim (or Ordejón-Mauri) functional |
| OMM.Use2D             | true          | Whether to distribute matrices on a 2D process grid |
| OMM.WriteCoeffs       | false         | Whether to write the LWFs (C^† matrix) to the restart file |
| OMM.RelTol            | \( 10^{-9} \) | The tolerance for the energy convergence in conjugate-gradient (CG) iterations. |
| OMM.BlockSizeC        | \( b_{WF} = b_{BF} N_{WF}/N_{BF} \) | The block size for LWFs (rows of the \( C^† \) matrix). By default, equals the block size for the basis functions \( b_{BF} \) (input parameter BlockSize) multiplied by the ratio of the total number \( N_{WF} \) of LWFs to the basis set size \( N_{BF} \) |
| OMM.Eta               | 0 eV          | The chemical potential                           |
| OMM.ReLWF             | 9.5 Bohr      | The cutoff radius \( R_c \) for LWFs determining the sparsity pattern of the \( C^† \) matrix |
| OMM.ReLWFINit         | 0 Bohr        | The initial cutoff radius \( R_c \) for LWFs. It is the same as OMM.ReLWF if set to 0 |
| OMM.Extrapolate       | false         | Whether to estimate LWFs at the next by the linear extrapolation of the results of two last molecular dynamics (MD) steps |
| OMM.Precon            | -1            | The number of self-consistent-field (SCF) steps for which to apply the preconditioning. If negative, the preconditioning is applied at all SCF steps |
| OMM.PreconFirstStep   | OMM.Precon    | OMM.Precon for the first MD step |
| OMM.TPreconScale      | 10 Ry         | The scale \( \tau_T \) for the kinetic energy preconditioning (see Eq. (7)) |
| OMM.Cholesky          | false         | Whether to apply the Cholesky factorization |

**Modular solver architecture**

**Solver input and output**

A scheme of the implemented OMM solver is shown in Fig. 2. At each self-consistent-field (SCF) step, the solver receives as an input the Hamiltonian and overlap matrices in the basis of strictly localized atomic orbitals, \( H \) and \( S \), and the information on the system geometry. SIESTA uses for matrices the compressed sparse row (CSR) format, that is the matrix information is stored in local one-dimensional (1D) arrays containing data values, column indices, indices of the first nonzero elements of local rows, and numbers of nonzero elements in each local row. The blocks of rows are distributed on a 1D process grid (Fig. 3a). Therefore, \( H \) and \( S \) are received by the solver in the CSR format. The density matrix \( \rho \) is the output, also in CSR (see Eq. (5)). This matrix is used to update \( H \) for the next SCF step outside the solver. At the end of each MD step, the solver is called again to compute the energy density matrix \( \rho_E \), which, along with \( \rho \), is later used to calculate forces (see Eq. (6)) and stresses. The scheme of the \( \rho_E \) calculation is analogous to that of \( \rho \) shown in Fig. 2.

To use the implemented OMM solver, SolutionMethod in the SIESTA input parameters should be set to BLOMM (OMM with Block matrices). The list of the principal input parameters related to the solver is given in Table 1.

The solver uses the libOMM library\(^9\),\(^25\),\(^30\) to perform the CG minimization of the energy functional given by Eq. (4). As an input, the libOMM library requires \( H \) and \( S \) as well as the initial guess for \( C^† \), in one of the MS formats\(^9\),\(^29\),\(^33\),\(^34\). As an output, it provides the converged \( C^† \), and \( \rho \) or \( \rho_E \) in the same format. The \( pddbc \) format is used for parallel calculations with dense matrices. To run such calculations with SIESTA, the input parameter OMM.UseSparse should be set to false (Table 1). In this case, all matrix elements are stored and algebraic operations are performed using the ScaLAPACK library\(^2\). The matrix is divided into 2D blocks distributed on a 2D or 1D process grid (OMM.UseSparse true or false, respectively). For parallel calculations with sparse matrices (OMM.UseSparse true), the \( pdcsr \) format is used. The matrix is also divided into 2D blocks, which are distributed on a 1D or 2D process grid (Figs. 3a and 3b). However, in this case, zero blocks are not stored. The algebraic operations are performed by the DBCSR library\(^2\),\(^26\). At the moment, libOMM supports only equal rectangular blocks.

The equations implemented in the libOMM library are compatible with all OMM flavours discussed in the previous section, including the Ordejón-Mauri and Kim functionals, with and without localization constraints (OMM.UseSparse true and false, respectively). However, to make the libOMM library functional for sparse matrices, some parts of the code have been reformulated. Now block-size information is passed to the MS library during the allocation of intermediate mat-
FIG. 2. Scheme of the revised OMM solver. Blocks within the solver are shown in blue. The rest of the SIESTA code is shown as a red block. Arrows indicate data flow. Hamiltonian, overlap and density matrices are denoted as $H$, $S$ and $\rho$. The Hermitian conjugate of the coefficient matrix of expansion of the wavefunctions in the basis of localized atomic orbitals is denoted as $C^\dagger$. SIESTA matrices use the csr format. They are converted to MatrixSwitch (MS) format for calculation of $\rho$ and $S$ with the help of the libOMM library. The restart file for $C^\dagger$ can be read once at the first molecular dynamics (MD) step.

ces required for the CG minimization using $m\_allocate()$ (see section 6S of Supplementary Information). Also sparsity is imposed on the gradient matrix $G$ (with the elements $G_{ij}^\mu = \partial E/\partial (C_{ij}^\mu)$) following the sparsity pattern of the initial guess for $C$. Already during $G$ calculation only matrix elements that fit into the sparsity pattern are computed in the contributions to $G$ that are given by products of matrices (using $\text{keep\_sparsity} = \text{true}$ option of $m\_multiply()$). In the rest of the contributions, the nonzero elements that do not fit into the sparsity pattern are omitted and no longer stored, while zero elements within the sparsity pattern are stored as zeros. The sparsity of the density ($\rho$) and energy density ($\rho_E$) matrices is assumed to be the same as of the overlap matrix $S$ and only elements of these matrices that fit into the sparsity pattern are computed. Additionally, the expression for the calculation of $\rho_E$ has been corrected as compared to the previous libOMM version\cite{16} in accordance with Eqs. (4) - (6) and Ref.\cite{17}. The Cholesky factorization (OMM.UseCholesky true) and kinetic energy preconditioning (switched on using the input parameters OMM.Precon and OMM.FirstStepPrecon) are available only for dense matrices.

MatrixSwitch extension

In order to incorporate the libOMM library into SIESTA within the OMM solver, the following steps are required (see Fig. 2): (1) matrix format conversion from/to the csr format to/from the MS formats and (2) initialization and update of $C^\dagger$, according to the current geometry of the system. The matrix format conversion is realized using calls to MS subroutines $m\_register\_csr()$ and $m\_copy()$ (see section 6S of Supplementary Information). The first of this subroutines has been added to the MS library and the second one has been extended to allow the conversion from/to the csr format to/from the pdcsr and pdldbc formats. The conversion is performed as follows (see also Fig. 1S of Supplementary Information). First the pointers to arrays of the csr matrix and its block size are passed to MS. Then a pdcsr/pdldbc matrix distributed on the 1D process grid with the same block size for rows as the initial csr matrix is filled in element by element (Fig. 3). The missing elements of the pdddbc matrix or within nonzero blocks of the pdcsr matrix are filled with zeros. Note that to speed up the conversion and guarantee linear scaling, column and row indices of nonzero blocks of the pdcsr matrix should be passed to the DBCSR library before filling the values via the call to $m\_reserve\_blocks()$ (see section 6S of Supplementary Information). Once the 1D-distributed pdcsr/pdldbc matrix is ready, it can be redistributed on a 2D process grid (OMM.Use2D true, Table I). In the case when the final matrix is distributed on the 1D process grid (OMM.Use2D false) and has the same block size for rows at the initial csr matrix, the last step is omitted.

The conversion from pdcsr and pdldbc to csr is implemented in a similar way. It is assumed that the row and column indices of nonzero elements of the csr matrix are already known. Only the values of the matrix elements are restored.

The initialization of the sparse $C^\dagger$ matrix at the first MD step and its update at following steps are carried out in the same way as in the original implementation\cite{16,17} of the linear-scaling solver in SIESTA (section 2S of Supplementary Information). The input and output of $C^\dagger$ is performed using the subroutines $m\_read()$ and $m\_write()$ added to the MS library (sections 3S and 6S of Supplementary Information).

Test results

To compare the performance of diagonalization and OMM with dense and sparse matrices, we have performed test MD simulations for single-layer boron nitride (BN) in different sizes. The computational details of the tests are given in section 4S of Supplementary Information. Fig. 4 demonstrates that the approaches in which the wavefunctions are not confined in space have much worse scaling with system size than the methods with localized wavefunctions within a cutoff radius $R_c$. The scaling of the former approaches is close to cubic for large systems (exceeding 1000 atoms in our calculations). It should be noted, however, that for small systems (within 1000 atoms) the scaling is sub-cubic. The reason is that for such systems the solver contribution to the total time plotted in Fig. 4 is comparable to the contributions of other parts of the code that have linear scaling with system size. Among the methods using dense matrices, OMM with applied preconditioning or Cholesky factorization, which improve convergence, shows a slightly better scaling compared to diagonalization or plain OMM. Also OMM using sparse matrices handled via the DBCSR library with no localization ($R_c \rightarrow \infty$) clearly has a better scaling than OMM using ScalAPACK. This is explained by the fact that the former, although having a dense coefficient matrix, still exploits the sparsity of the Hamiltonian and overlap.
FIG. 3. Example of matrix format conversion from the csr format used in SIESTA to pdcsr MatrixSwitch format handled with DBCSR: (a) a csr matrix distributed on the 1D process grid with 4 CPU cores, (b) pdcsr matrix with $2 \times 3$ blocks distributed on the same 1D process grid and (c) pdcsr matrix with $2 \times 3$ blocks distributed on the $2 \times 2$ 2D process grid. Arrows indicate steps 1 and 2 of subroutine m_copy() as explained in Fig. 1S from Supplementary Information. Small squares represent elements of the $8 \times 12$ matrix. Black squares are zero elements that are not stored. Red, yellow, blue and green squares correspond to elements stored on cores 1, 2, 3 and 4, respectively.

FIG. 4. Wall time (in hours) for 4 MD steps for a single boron nitride (BN) layer computed using different approaches vs number $N$ of atoms in the system: (black squares) diagonalization, (green triangles up) OMM with dense matrices (using ScalAPACK) and preconditioning using a kinetic-energy scale $\tau_E = 10$ Ry, (dark green triangles down) OMM with dense matrices with Cholesky factorization, (open green triangles up) plain OMM with dense matrices, (magenta diamonds) OMM with sparse matrices (using DBCSR) without wavefunction localization (wavefunction cutoff radius $R_C \to \infty$), (red circles) Ordejón-Mauri functional with $R_C = 4$ Å and (open red circles) Kim functional with $R_C = 4$ Å and chemical potential $\eta = -5.5$ eV. In all the cases without wavefunction localization, the Ordejón-Mauri functional is considered. The calculations are performed on 96 CPU cores. A double-zeta polarized (DZP) basis set is used. The block size is $b_{WF} = 6$ for the wavefunctions and $b_{BF} = 13$ for the basis functions.

In the range of system sizes considered, OMM with kinetic energy preconditioning is the fastest among the approaches without wavefunction localization, followed by OMM with the Cholesky factorization, diagonalization, and plain OMM (Fig. 4). The crossover between preconditioned dense OMM and the linear-scaling methods takes place for the system with about 1200 atoms. For the plain dense OMM and for diagonalization, the crossovers with linear-scaling methods occur earlier, at about 300 and 700 atoms, respectively.

Our timings for single-layer BN have confirmed that the Or-
dejón-Mauri and Kim approaches in which the wavefunctions are localized within a cutoff radius $R_C$ show linear scaling with system size (Fig. 5). The computational times corresponding to different parts of the solver (matrix conversion, libOMM library, initialization and update of the coefficient matrix, reading and writing of restart for localized wavefunctions) and other parts of the code such as the DHSCF subroutine for Hamiltonian update, all do change linearly upon increasing the system size. As a result, relative contributions of different parts of the code do not depend on the system size (Fig. 5). This is different from the cubic-scaling methods, in which the solver very early takes most of the computing time upon increasing the system size, since the rest of the code has linear scaling. It should also be noted that, for the systems considered, the solver takes only 40–50% of the computational time, comparable, for example, to the DHSCF subroutine for Hamiltonian update. Most of this time corresponds to the minimization of the energy functional given by Eq. (4) performed by the solver library libOMM. The matrix format conversion takes only 0.5–1.0% of the total time. Writing of the restart files for localized wavefunctions takes up to 0.3% of the time, and initialization and update of the coefficient matrix take a negligible time within 0.01%.

The dependence of computational time on block size for the Kim functional with DBCSR are presented in Fig. 6. In the case of the double-zeta polarized (DZP) basis set, each boron and nitrogen atom hosts 3 localized wavefunctions and 13 basis functions. Accordingly the computational time drops significantly at block-size values $b_{WF}$ for the basis functions divisible by 13 (Fig. 6b). For such block sizes, the computational time grows upon increasing the block size (note that the growth continues beyond the block sizes shown in Fig. 6a) and has the minimum at $b_{WF} = 13$. The wavefunction block-size $b_{WF}$ dependence reaches the minimum at $b_{WF} = 6–10$. At small $b_{WF}$, a fast growth of the computation time is observed. It can be attributed to an increase in the number of nonempty blocks considered upon decreasing the block size. At large $b_{WF}$, computational time also grows but in a slower rate. This dependence can be explained by increasing the number of matrix elements that are stored and explicitly considered in matrix operations. Therefore, we find optimal block sizes both for the wavefunctions and basis functions of the order of 10. Furthermore, chemical considerations can be exploited when dividing matrices into blocks. Still the optimal choice of block sizes for complex systems is not straightforward and requires further investigation.

The CPU scaling of the libOMM solver library in calculations with sparse matrices using DBCSR is shown in Fig. 7. A similar CPU scaling is observed for systems of different size (Fig. 7a), with different block and basis set sizes. The computational time decreases by a factor of about 2.5 upon doubling the computational cost. Such a speedup is observed for CG energy minimization and subsequent calculation of the density matrix. It should be noted, however, that calls to libOMM for calculation of $\rho_F$ involving only two matrix multiplication operations show much better CPU scaling. This can be appreciated from a twice steeper slope of computational cost versus computational time as compared to the calls for energy minimization and calculation of the density matrix (Fig. 7b). It can, therefore, be expected that the solver parallelization might be further improved via proper code refactoring. The use of OpenMP, GPUs and the library for small matrix multiplication (LIBXSMM) are known to lead to a superior DBCSR performance which also requires investigation.

**Recommendations for OMM solver use**

The new modular implementation of the OMM solver makes it easier to disentangle technical problems in e.g. parallelization from drawbacks of the OMM method itself. Here we present the first implementation of the solver utilizing external libraries that represents the starting point for further performance improvement and method polishing. Ways to improve the solver performance were mentioned in the previous subsection. We briefly discuss now the drawbacks of the OMM method and how they can be addressed.
FIG. 7. (a) Computational cost (in CPU hours) vs wall time (in s), for one call to the solver library libOMM including one conjugate gradient (CG) iteration and calculation of the density matrix $\rho$ for different supercells of BN using the Kim functional: (black squares) $60 \times 60$, (red circles) $72 \times 72$ and (blue triangles) $90 \times 90$ (7200, 10368 and 16200 atoms, respectively). The number of CPU cores used is indicated. (b) Relative computational cost vs relative time for the calls to the solver library including one conjugate gradient iteration and calculation of the density matrix $\rho$ (closed symbols) or calculation of the energy density $\rho_E$ (open symbols) for different supercells of BN. Relative values are given with respect to the results for 192 CPU cores. The calculations are for a DZP basis, $R_C = 4$ Å, $\eta = -5.5$ eV, $b_{WF} = 6$, and $b_{BF} = 13$. Linear mixing with a mixing parameter of 0.1 is used.

One of the most important methodological problems of the OMM approach is in the minimization, which can require a large number of CG iterations. As shown in Fig. 8, the first SCF iteration from scratch is rather costly both for the linear and cubic-scaling OMM. For the linear-scaling methods, the first SCF iteration can include thousands of CG steps, followed by tens of SCF iterations with hundreds of CG steps each. After that each SCF step needs just a few CG iterations, becoming very fast. It should be noted that except for the very first SCF iterations, the linear-scaling and plain cubic-scaling OMM require roughly same numbers of CG steps. However, kinetic energy preconditioning or Cholesky factorization significantly reduce the number of CG iterations required, with a considerable computational-time reduction (see also Fig. 4).

Therefore, it is always recommended to use any of both ways to deal with kinetic energy ill-conditioning in dense OMM. The extension of these approaches to sparse matrices is not straightforward and requires further investigation.

Also starting from scratch, one can get into regions in parameter space where the energy functional does not have a minimum in the CG line minimization. To avoid this situation, we recommend using small cutoff radii for the initial guess of wavefunctions (OMM.RcLWFInit, Table 1) both for linear and cubic-scaling OMM. It is also recommended to preconverge the ground state using a small linear-mixing parameter. Starting from as low as 0.01 can be required for very large systems. It can then be gradually increased to normal values of 0.1 – 0.2. After getting close to the ground state, the use of other mixing schemes is possible. If the geometry of the system is far from the optimal one, a reduced step for geometry optimization may also be needed when starting. Note that the convergence with cutoff radius in linear-scaling OMM is achieved already for radii of several Å (see section 5S of Supplementary Information). For single-layer BN, $R_C = 4$ Å is sufficient for converged energy and forces within 10 meV/atom and 0.02 eV/Å, respectively.

**DISCUSSION**

We have demonstrated how modularization simplifies the implementation of new solvers in electronic structure codes with the implementation of the OMM solver in the SIESTA code. Matrix algebra operations and parallelization are efficiently handled via external libraries. In particular, the implementation benefits from two ESP libraries:
The libOMM library is used to perform the minimization of the energy functional, while the MatrixSwitch library serves as an interface to low-level algebraic routines facilitating switching between different matrix formats. These libraries have been extended to make possible not only cubic-scaling but also linear-scaling OMM calculations. Now the energy functional minimization in libOMM can be carried out for sparse matrices with the DBCSR library in addition to dense matrices using ScaLAPACK. To facilitate incorporating libOMM into electronic structure codes based on atomic orbitals, MatrixSwitch has been also supplemented with subroutines for matrix format conversion and matrix reading and writing. The solver library libOMM can now be easily further developed in the MatrixSwitch language for the implementation of new solvers. Furthermore, the extended MatrixSwitch and libOMM libraries available through ESI can be used for implementation of linear and cubic-scaling OMM approaches in other codes.

To test the performance of the new OMM and traditional diagonalization solvers available in SIESTA, large-scale calculations have been performed for a BN layer. When sparse matrices and localized wavefunctions are used, linear scaling with system size is achieved in practice, as expected. Matrix conversion, reading and writing of restart files, as well as initialization and update of the localized wavefunctions take a small fraction of the computational time. For the linear-scaling methods that fraction does not depend on system size. The cubic-scaling OMM with kinetic energy preconditioning performs best for small systems, even better than diagonalization. For plain OMM, diagonalization, and cubic-scaling OMM with kinetic energy preconditioning, the crossovers with linear-scaling methods are observed at about 300, 700 and 1200 atoms, respectively. The best performance for the linear-scaling OMM with sparse matrices is achieved when the wavefunctions and basis functions are divided into blocks of sizes around 10, taking into account the chemical structure. The OMM solver is MPI-parallelized. When using the DBCSR library for algebraic operations with sparse matrices, the computational time decreases by a factor of 2.5 upon doubling the computational cost. It is expected that CPU scaling can be further improved via refactoring some operations in the libOMM library, using OpenMP and GPUs, etc.

To perform OMM calculations from scratch, it is recommended to start using a small linear-mixing parameter (down to 0.01), a small step for geometry optimization, and cutoff radii for the wavefunctions of a few Å. For the cubic-scaling OMM, the convergence becomes much faster with kinetic energy preconditioning or Cholesky factorization. The extension of these approaches to sparse matrices demands further investigation.

DATA AVAILABILITY

The data supporting the results presented in this article are available at the Mendeley Data repository at the following link https://data.mendeley.com/datasets/c8kz58bg5z/1.

CODE AVAILABILITY

The orderN branch of SIESTA including the developments described in the paper can be found at https://gitlab.com/irina_lebedeva/siesta/-/tree/orderN. The orderN branch will be incorporated to the SIESTA master branch and releases in the near future. The extended MatrixSwitch and libOMM libraries can be downloaded from the master branch of the omm-bundle at https://gitlab.com/ElectronicStructureLibrary/omm-bundle.

ACKNOWLEDGMENTS

The authors acknowledge the European Union MaX Center of Excellence (EU-H2020 Grant No. 824143), the Partnership for Advanced Computing in Europe, PRACE, for awarding us access to computational resources in Joliot-Curie at GENCI@CEA, France (EU-H2020 Grant No. 201921516) as well as technical and human support provided by IZO-SGI SGiker of the University of the Basque Country (UPV/EHU) and European funding (ERDF and ESF). ICN2 is supported by the Severo Ochoa program from Spanish MINECO (Grant No. SEV-2017-0706) and by Generalitat de Catalunya (CERCA Programme). ICMAB is supported by the Severo Ochoa program from Spanish MICIU (Grant No. CEX2019-000917-S). PO acknowledges support by Spanish MICIU, AEI and EU FEDER (Grant No. PGC2018-096955-B-C43). AG acknowledges support by Spanish MICIU, AEI and EU FEDER (Grant No. PGC2018-096955-B-C44) and by Generalitat de Catalunya (Grant No. 2017SGR1506). EA acknowledges funding from Spanish MICINN through grant PID2019-107338RB-C61/AEI/10.13039/501100011033, as well as a Maria de Maeztu award to Nanogune, Grant CEX2020-001038-M funded by MCIN/ AEI/10.13039/501100011033. We also thank Dr. David López-Durán for useful discussions and providing the introduction to the MatrixSwitch library.

AUTHOR CONTRIBUTIONS

AG, EA and PO designed the project. IVL extended the codes and performed the calculations. All the authors discussed the results and commented on the manuscript.

CONFLICT OF INTEREST

The authors declare no conflict of interest.
