Computer Physics Communications 238 (2019) 254–261

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Efficient computation of Coulomb and exchange integrals for multi-million atom nanostructures☆

Piotr T. Różański *, Michał Zieliński

Institute of Physics, Faculty of Physics, Astronomy and Informatics, Nicolaus Copernicus University, ul. Grudziądzka 5, 87-100, Toruń, Poland

ARTICLE INFO

Article history:
Received 28 August 2018
Received in revised form 4 December 2018
Accepted 11 December 2018
Available online 31 December 2018

Keywords:
Quantum dots
Atomistic calculations
Linear scaling
Fast Fourier transform

ABSTRACT

Atomistic modeling of nanostructures such as quantum dots or nanowires often involves numbers of atoms reaching and even exceeding well beyond 1 million. Such a large quantity of atoms presents a very complex computational challenge especially at a stage of many-body calculation where numerous Coulomb matrix elements need to be calculated. Here we present a practical solution to this problem by performing calculations in the momentum space and utilizing fast Fourier transform combined with a memory-efficient way to compute the convolution that overcomes the problem of spurious interactions with quasi-charges from other super-cells. Finally, calculation of multiple integrals is optimized by reducing the problem to finding a minimum vertex cover of a graph. All these algorithms are implemented and presented here in a self-contained and highly parallelized computer program named Coulombo. Coulombo demonstrates quasi-linear scaling of computational time of Coulomb matrix elements with respect to the number of points in the computational box and, at the same time, significantly reduced memory demand. The proposed solution can have potential applications not only in the realm of nano-physics, but could be applied to other mesoscopic simulations or large-scale quantum chemistry problems.

Program summary

Program Title: Coulombo
Program Files doi: http://dx.doi.org/10.17632/98bhm5zbrd.1
Licensing provisions: CC BY 4.0
Programming language: C++
Nature of problem: Computing the Coulomb matrix elements (Coulomb and exchange integrals), while being a demanding computational task, is a necessary step in a range of quantum mechanical calculations. For example, in the field of nanostructures, such as quantum dots and nanowires, this stage of calculation even after numerous approximations is at least an $O(N^2)$ operation (a summation over all pairs of N atoms or grid points in the analyzed system). Moreover, calculating the full Coulomb matrix usually requires computation of thousands of such elements, thus presenting a formidable computational challenge.
Solution method: We provide a ready-to-use implementation for calculating Coulomb matrix elements for a given set of input wavefunctions. This implementation is based on the approach introduced in [1], by using a fast Fourier transform to compute the convolution. In this work we further significantly improved the method by eliminating the need to extend the computational domain with padding, thus reducing the memory consumption by a factor of 8. The optimal computational plan for each run is prepared by calculating a minimum vertex cover on a graph representing the subset of requested Coulomb matrix elements.
Additional comments including restrictions and unusual features: The implementation is fully parallelized in a distributed-memory model, using MPI and parallel routines from FFTW [2]. A minimum vertex cover is computed by our greedy approximation algorithm, which we found to perform significantly better than the standard text-book heuristic.
References:
[1] P. T. Różański & M. Zieliński, "Linear scaling approach for atomistic calculation of excitonic properties of 10-million-atom nanostructures", Phys. Rev. B 94 (2016) 045440

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: rozanski@fizyka.umk.pl (P.T. Różański).

https://doi.org/10.1016/j.cpc.2018.12.011

[2] M. Frigo & Steven G. Johnson, "The Design and Implementation of FFTW3", Proceedings of the IEEE 93 (2), 216–231 (2005), Invited paper, Special Issue on Program Generation, Optimization, and Platform Adaptation

1. Introduction

In recent years, atomistic [1–7] approaches for calculations of electronic and optical spectra of nanostructures have established themselves as methods of choice whenever one needs to account for low lattice symmetry [8–11], few monolayer thin interfaces [12–14], or chemical composition intermixing and resulting alloying [15–18]. Contrary to methods based on the continuous media approaches, such as the effective mass approximation, atomistic methods, including the empirical tight binding [3,4,19–22], are naturally able to account for all of the above effects.

However, since the systems are defined atom-by-atom, the key challenge for atomistic calculations is a large number of atoms to be accounted for, and thus the high computational complexity. For example, recent atomistic simulation of single phosphorus dopant in silicon involved a computation box exceeding 3 million atoms [23], whereas realistic simulations of crystal phase quantum dots [13,24] embedded in a large section of an indium phosphate nanowire may even lead to about 10 million atoms in the modeling.

Computations for such systems are a formidable challenge already at the single particle stage [6,24], yet the real struggle begins at the following many-body part of the calculation, where interactions between single particles are to be accounted for. For example, in nanowire quantum dots the calculation of an interacting electron–hole pair (exciton) may involve the calculation of the order of 10^4 Coulomb and exchange integrals over a domain consisting of 1 million atoms [14,24,25]. Thus any practical atomistic approach aimed at the description of large systems, must have good scaling properties with the number of atoms. In case of the empirical tight binding where a linear combination of atomic orbitals (LCAO) is used, the straightforward calculation of Coulomb matrix elements would lead to a very unfavorable scaling of $O(\tilde{N}^4)$, where $\tilde{N}$ is number of atoms. After a series of approximations this can be reduced to a quadratic-like $O(\tilde{N}^2)$ scaling. This seems acceptable yet it is still highly impractical since going from $\sim$1 thousand atoms (a small nanocrystal) simulation to 1 million atom case (a nanowire quantum dot in a segment of nanowire) would lead to the increase of the computational time by six orders of magnitude. Therefore, with any non-linear polynomial scaling scheme, going well beyond 1 million atoms seems highly challenging if not intractable.

In our recent work we have presented a solution to this problem. First we perform a single particle stage of the calculation using the empirical tight binding method. Then we convert the LCAO wave-functions to a three dimensional grid with no more than 30 grid point per atom to obtain converged many-body spectra. This transfer from LCAO to a relatively coarse grid allows for utilization of Fourier space methods on a grid (similarly to [26]) namely the fast Fourier transform (FFT). The combined overall time of both single particle and many-body calculation of Coulomb matrix elements scales as $O(N \log N)$, where N is number of grid points linearly related to the number of atoms $\tilde{N}$, and thus leading to a quasi-linear scaling in terms of the number of atoms.

Here we present a computer code named Coulombo that is able to perform such quasi-linear (in time and memory) calculations of Coulomb and exchange integrals from real or complex wave-functions defined on a spatial, three-dimensional grid. The wave-functions are provided by a user as a set of simple files in a standard binary format. Since the program works with any regular three-dimensional grid (and not the particular tight-binding form input), the code is not problem specific and it can handle various wave-functions, e.g. of dopants or nanostructures.

One of the challenges of the calculations in the inverse space are implicitly assumed periodic boundary conditions imposed by the FFT. This leads to a spurious interaction of charges from a super-cell with their periodic copies from other cells. One way of avoiding this problem would be choosing a sufficiently large computational box, but the results typically converge slowly with respect to the grid (super-cell) volume, whereas simultaneous increase of computational volume unavoidably leads to an increased time of calculation and memory demands. Numerous techniques including multi-pole expansion [27–29] or truncated Coulomb interaction [29] have been proposed in the past to speed up this convergence. In our previous work we have decided to use a different approach by padding the grid with zeros. Due to this padding, the full convolution necessary in the calculation scheme is equivalent to a circular convolution, which in turn can be effectively computed using the fast Fourier transform algorithm. However this results in 8 times more grid points which significantly increases memory demands of the program [24].

Here we propose to overcome this problem by performing the FFT calculations on eight auxiliary sub-lattices without increased memory demand which would result from zero-padding, and at the same time without any increase of the computational time, thus retaining the quasi-linear scaling of the computations. The Coulombo code associated with this paper is provided with two basic schemes of dielectric screening (namely, static and Thomas–Fermi like), yet other models can be incorporated by a user. The code has the broad range of potential applications from nanostructures, dopants, up to more general quantum chemical problems.

2. Problem description

Coulomb matrix elements (elements of matrix representation of the Coulomb operator) are defined in any given base of single-particle states ψ_n, as [20,30]

$$E_{ijkl} = \int \frac{\psi_i^*(\vec{r}_1)\psi_j^*(\vec{r}_2)\psi_k(\vec{r}_2)\psi_l(\vec{r}_1)}{\epsilon(\vec{r}_1-\vec{r}_2)|\vec{r}_1-\vec{r}_2|}\, dr_1^3\, dr_2^3\,, \tag{1}$$

where i, j, k and l are single-particle functions indices, $\vec{r}_1$ and $\vec{r}_2$ span the entire computational domain, and ϵ stands for a position (distance) dependent dielectric function, which, in the simplest approximation, can often be replaced by a static dielectric constant.

If we assume that the wave-functions are represented on a regular (but not necessarily equidistant) three-dimensional grid, we can rewrite Eq. (1) as

$$E_{ijkl} = \sum_{\vec{r}_1}\sum_{\vec{r}_2} \frac{\psi_i^*(\vec{r}_1)\psi_j^*(\vec{r}_2)\psi_k(\vec{r}_2)\psi_l(\vec{r}_1)}{\epsilon(\vec{r}_1-\vec{r}_2)|\vec{r}_1-\vec{r}_2|}\, h_x^2\, h_y^2\, h_z^2 \tag{2}$$

and in this case, the computational effort needed to compute a single element E_{ijkl} scales as $O(N^2)$ where N is the number of grid points, as it requires a double summation over the entire computational domain.

Moreover, the upper bound on the number of all E_{ijkl} integrals for a single-particle basis consisting of M wave-function, is equal to M^4. Therefore, the total cost of computing all integrals in the pessimistic case for a basis of M single-particle states, scales as $O(M^4N^2)$ in the straightforward approach. Therefore significant optimizations are needed for this method to be feasible.

3. Optimizing the computation of a single integral

Eq. (2) can be rewritten as

$$E_{ijkl} = \sum_{\vec{r}_1} \sum_{\vec{r}_2} \rho_{il}(\vec{r}_1)\, G(\vec{r}_1 - \vec{r}_2)\, \rho_{jk}(\vec{r}_2)\, h_x^2\, h_y^2\, h_z^2 \tag{3}$$

by introducing two "quasi-densities" $\rho_{il} = \psi_i^* \psi_l$ and $\rho_{jk} = \psi_j^* \psi_k$ formed by pairs of wavefunction and "interaction function" $G(\Delta r) = (\epsilon_0 \epsilon(\Delta r)|\Delta r|)^{-1}$ depending on the precise model for dielectric function ϵ (assuming it depends on the distance only).

Now, if we introduce the "quasi-potential" V_{jk} generated by one of the quasi-densities ρ_{jk}, we can rearrange the equation as

$$E_{ijkl} = \sum_{\vec{r}_1} \rho_{il}(\vec{r}_1)\, V_{jk}(\vec{r}_1)\, h_x^2\, h_y^2\, h_z^2 \tag{4}$$

where V_{jk} is defined as

$$V_{jk}(\vec{r}_1) = \sum_{\vec{r}_2} G(\vec{r}_1 - \vec{r}_2)\, \rho_{jk}(\vec{r}_2) \tag{5}$$

which, in turn, can be rewritten as $V_{jk} = G \otimes \rho_{jk}$ where $\otimes$ denotes the full (non-periodic) convolution.

The efficient calculation of this convolution with the help of the FFT, similarly to the approach used in [24], requires padding and effectively extending the computational box twice in each direction (8 times in volume). Quasi-density ρ_{jk} is padded with zeros, while the interaction function G is padded in the way that makes it symmetric in every dimension, i.e.

$$\forall_{j_x \in \{i_x, 2N_x - i_x\}} \forall_{j_y \in \{i_y, 2N_y - i_y\}} \forall_{j_z \in \{i_z, 2N_z - i_z\}} G(j_x, j_y, j_z) = G(i_x, i_y, i_z)\,, \tag{6}$$

where the notation $j \in \{i, 2N - i\}$ means that j is either i or $2N - i$.

Only with this specific padding of ρ_{jk} and G, the full convolution (that we need to calculate) is equivalent to the circular convolution computed with the FFT [31] and, therefore, we avoid introducing any unwanted periodicity or spurious boundary conditions. The padding effectively overcomes the problem of artificial interactions with quasi-charges from other (periodic) super-cells due to periodicity assumed in the FFT, yet at a cost of a significant memory demand.

Memory demand can be estimated based on the simple example of a moderate computational box of 50 nm width, with a grid step of 0.8 Å as used in [24]. Storing a single wave-function as a three-dimensional array of complex values requires a memory of

$$\left(\frac{500\,\text{Å}}{0.8\,\text{Å}}\right)^3 \cdot 16\,\text{bytes} \approx 3.9\,\text{GB}\,. \tag{7}$$

While this is still in range of mid- to high-end personal computers, increasing the memory demand by a factor of 8 would restrict the availability of the method to high-performance computing.

To avoid this eight-fold increase in the memory consumption, we introduce the memory-efficient way to compute the convolution, which takes advantage of the specific symmetries of the padding. This method is further described in detail in the following Section 3.1.

3.1. Memory-efficient fast convolution method

Eq. (5) can be represented as a full convolution $V = G \otimes \rho$, which, in turn, can be calculated with the help of the fast Fourier transform as an element-wise multiplication

$$\mathcal{F}V = \mathcal{F}G\, \mathcal{F}\rho\,, \tag{8}$$

where $\mathcal{F}V$, $\mathcal{F}G$ and $\mathcal{F}\rho$ represent Fourier transforms of V, G and ρ, respectively, after being extended in every dimension as described in the previous section.

Due to the specific symmetric padding of G (as it is both real and symmetric), the fast Fourier transform will also be both real and symmetric. Therefore, it can be calculated as a discrete cosine transform, without a need to extend the computational domain of G. In this case, while the "logical" size of the domain is still $[2N_x, 2N_y, 2N_z]$, the "physical" size (affecting the computation time and memory consumption) is only $[N_x + 1, N_y + 1, N_z + 1]$.

Directly from the definition of discrete Fourier transform of ρ, taking into account that values of ρ are zero outside the box $[N_x, N_y, N_z]$, we obtain

$$(\mathcal{F}\rho)(k_x, k_y, k_z) = \sum_{[N_x \times N_y \times N_z]} \rho(n_x, n_y, n_z) \exp\left(-2\pi \mathrm{i} \left(\frac{k_x n_x}{2N_x} + \frac{k_y n_y}{2N_y} + \frac{k_z n_z}{2N_z}\right)\right). \tag{9}$$

We can now divide the domain of $\mathcal{F}\rho$ into eight sub-lattices of size $[N_x, N_y, N_z]$ each:

$$\begin{aligned}
\mathcal{F}\rho_{000}(k_x, k_y, k_z) &= \mathcal{F}\rho(2k_x, 2k_y, 2k_z) \\
\mathcal{F}\rho_{001}(k_x, k_y, k_z) &= \mathcal{F}\rho(2k_x, 2k_y, 2k_z + 1) \\
\mathcal{F}\rho_{010}(k_x, k_y, k_z) &= \mathcal{F}\rho(2k_x, 2k_y + 1, 2k_z) \\
&\dots \\
\mathcal{F}\rho_{111}(k_x, k_y, k_z) &= \mathcal{F}\rho(2k_x + 1, 2k_y + 1, 2k_z + 1)
\end{aligned} \tag{10}$$

Substituting Eq. (10) into Eq. (9) we obtain

$$\begin{aligned}
&\mathcal{F}\rho_{\xi\upsilon\zeta}(k_x, k_y, k_z) \\
&= \sum_{[N_x \times N_y \times N_z]} \rho(n_x, n_y, n_z) \exp\left(-2\pi \mathrm{i} \left(\frac{\xi n_x}{2N_x} + \frac{\upsilon n_y}{2N_y} + \frac{\zeta n_z}{2N_z}\right)\right) \\
&\quad \exp\left(-2\pi \mathrm{i} \left(\frac{k_x n_x}{N_x} + \frac{k_y n_y}{N_y} + \frac{k_z n_z}{N_z}\right)\right),
\end{aligned} \tag{11}$$

which means that every sub-lattice of $\mathcal{F}\rho$ can be calculated using fast Fourier transform performed on ρ (without padding) multiplied by specific phase coefficient $w_{\xi\upsilon\zeta}$

$$\mathcal{F}\rho_{\xi\upsilon\zeta} = \mathcal{F}(\rho \exp(-2\pi \mathrm{i}\, w_{\xi\upsilon\zeta})) \tag{12}$$

where

$$w_{\xi\upsilon\zeta}(n_x, n_y, n_z) = \frac{\xi n_x}{2N_x} + \frac{\upsilon n_y}{2N_y} + \frac{\zeta n_z}{2N_z}\,. \tag{13}$$

The inverse Fourier transform of V, based on Eq. (8), can be written as

$$\begin{aligned}
V(n_x, n_y, n_z) = \sum_{[2N_x \times 2N_y \times 2N_z]} &\mathcal{F}G(k_x, k_y, k_z)\, \mathcal{F}\rho(k_x, k_y, k_z) \\
&\exp\left(2\pi \mathrm{i} \left(\frac{k_x n_x}{2N_x} + \frac{k_y n_y}{2N_y} + \frac{k_z n_z}{2N_z}\right)\right)
\end{aligned} \tag{14}$$

which, using Eq. (10), can be re-written as a summation over all sub-lattices of $\mathcal{F}\rho$:

$$\begin{aligned}
V(n_x, n_y, n_z) = \sum_{\xi\upsilon\zeta} \exp\left(2\pi \mathrm{i} \left(\frac{\xi n_x}{2N_x} + \frac{\upsilon n_y}{2N_y} + \frac{\zeta n_z}{2N_z}\right)\right) \sum_{[N_x \times N_y \times N_z]} &\\
\mathcal{F}G(2k_x + \xi, 2k_y + \upsilon, 2k_z + \zeta)\, \mathcal{F}\rho_{\xi\upsilon\zeta}(k_x, k_y, k_z) &\\
\exp\left(2\pi \mathrm{i} \left(\frac{k_x n_x}{N_x} + \frac{k_y n_y}{N_y} + \frac{k_z n_z}{N_z}\right)\right),&
\end{aligned} \tag{15}$$

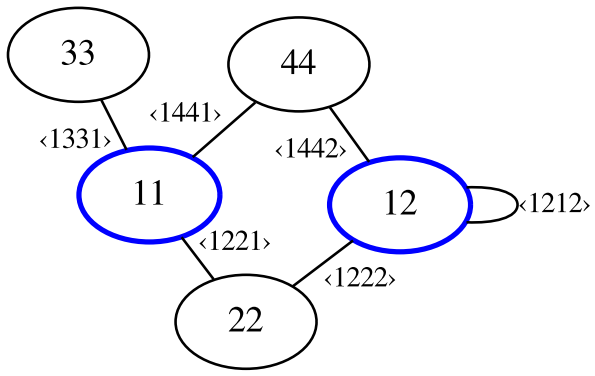

Fig. 1. Example of graph with vertices representing quasi-densities (products of wavefunctions) and edges representing integrals to be calculated. Vertices belonging to the minimum vertex cover are outlined. $\langle ijkl \rangle = E_{ijkl}$ as defined in Eq. (2).

and calculated using the inverse fast Fourier transform, as

$$V = \sum_{\xi\upsilon\zeta} \mathcal{F}^{-1} \left(\mathcal{F}G_{\xi\upsilon\zeta} \, \mathcal{F}\rho_{\xi\upsilon\zeta} \right) \exp(2\pi \mathrm{i} \, w_{\xi\upsilon\zeta}) \tag{16}$$

where $\mathcal{F}G_{\xi\upsilon\zeta}$ is a sub-lattice of $\mathcal{F}G$ defined as

$$\mathcal{F}G_{\xi\upsilon\zeta}(k_x, k_y, k_z) = \mathcal{F}G(2k_x + \xi, 2k_y + \upsilon, 2k_z + \zeta) . \tag{17}$$

The above formulation allows to sequentially calculate different $(\xi\upsilon\zeta)$ terms and accumulate them into a single storage of size $[N_x \times N_y \times N_z]$, thus eliminating the need for an eight-fold extension of storage used for a computational domain.

4. Optimizing many-integral computation

The straightforward optimization in the case of computing multiple integrals is based on a fact that the Fourier transform of G needs to be computed only once and it does not depend on the actual wave-functions.

Also, depending on the nature of the physical system being analyzed, we may want to calculate either all possible Coulomb matrix elements, or only the specific subset of them. Planning the computation in the optimal order allows to re-use some of the intermediate data without a significant increase in the memory consumption.

Directly from the formulation of Eq. (1) stem certain symmetries, i.e. $E_{ijkl} = E_{jilk} = E^*_{lkji} = E^*_{klij}$, which effectively reduce the number of integrals to be computed by a factor of four. Also, since the most expensive step is the convolution itself, we can take advantage of the fact that the result of the convolution, i.e. the quasi-potential V_{jk}, can be re-used for all integrals E_{*jk*}.

In order to minimize the number of convolutions and thus the number of the FFT computations, we construct an undirected graph, in which each vertex represents an unordered pair of wavefunctions. Two vertices $\{i, l\}$ and $\{j, k\}$ are connected by an edge if and only if there is a need to calculate any of the integrals between these two quasi-densities i.e. one of: $E_{ijkl}, E_{ikjl}, E_{ljki}, E_{lkji}, E_{jilk}, E_{jlik}, E_{kilj}$, E_{klij}. An example of such a graph is presented in Fig. 1.

On this graph, we compute a minimum vertex cover using an approximation algorithm described in detail in Appendix. Each vertex $\{j, k\}$ in the minimum vertex cover represents a quasi-density ρ_{jk} for which the convolution should be performed. For each such quasi-density, the quasi-potential V_{jk} is computed, and then used to compute all integrals represented by the adjacent edges. Every such integral can be calculated either directly (E_{*jk*} and E_{j**k}) or with the help of the complex conjugate (E_{*kj*} and E_{k**j}).

5. Implementation details

Implementation of the method, named Coulombo, is written in C++ (2011 standard) with hybrid MPI+OpenMP parallelization. The main executable (*coulombo*) accepts parameters from the command line, performs computations and writes results to the standard output. Diagnostic and error messages are written to the standard error stream. Description of all the parameters accepted by the program and standard usage examples are included in *README* file provided with the source code.

Program requires two external dependencies, which must be available on the host system at compile time:

1. Fast Fourier Transform implementation *FFTW* (version 3),
2. *armadillo* C++ linear algebra library.

To use the program, the user has to provide the precomputed wave-functions in *armadillo* binary format. The paths to the files with wave-function data should be passed as the command-line arguments to the program, one file per wavefunction (both complex and real data are supported). In case the user wants to include spin in the calculations (if the flag *spin* is given), the program will parse two consecutive data files per wavefunction, representing the spin-up and spin-down parts of the wavefunction. Therefore, the total number of input files must be even in case of using the program with *spin* flag. When including spin in the calculations, quasi-density is calculated from the spin-up ($\psi^{\uparrow}$) and spin-down ($\psi^{\downarrow}$) parts as

$$\rho_{jk} = \psi_j^{\uparrow *} \psi_k^{\uparrow} + \psi_j^{\downarrow *} \psi_k^{\downarrow} . \tag{18}$$

MPI parallelization of Coulombo is based on dividing the computational domain, in a balanced way, among all MPI nodes. With this approach, the total memory consumption of the program for any given set of parameters does not depend on the number of allocated nodes. Due to the internal storage order in *armadillo*, the domains are divided along the last (z) dimension.

In addition to MPI, Coulombo also supports OpenMP parallelization. To use OpenMP (or hybrid MPI+OpenMP) parallelization, one should specify the *threads-per-node* command line parameter. Otherwise, OpenMP will not be used as the environment variable OMP_NUM_THREADS is ignored by the program.

The execution of the program starts by sequentially loading all data files in the first (root) node, and scattering them to subsequent MPI nodes. In order to increase the performance of the FFTW library, a small zero-padding is introduced, so all the dimensions of the zero-padded computational box could be written in form $2^i 3^j 5^k$ for some natural i, j, k. Finding such dimensions is performed by class *Round*, consisting of a single static template method, *Round::up*. Zero-padding is performed as part of MPI "scatter" operation, by taking advantage of "sub-array" MPI types.

After the data has been split across all MPI nodes, the optimal plan for computation is generated as described in Section 4, based on the list of integrals requested by the user in *integrals* command-line parameter. Since generation of the plan takes negligible time, as compared to other stages of the calculations, it is performed redundantly by all nodes.

The user may specify a comma-separated list of integral specifications. Each specification may either be a single integral (e.g. `1111`) or a pattern matching one or more integrals (e.g. `ijji`, `1jj1`, `*12*`) consisting of:

- numbers (1–9) designating specific wavefunction in the same order as they appear in the command line,
- letters (a–z) designating any wavefunction, but the repeated letter in any given match always corresponds to the same wavefunction,
- asterisk sign (*) designating any wavefunction, with the repeated asterisks being independent.

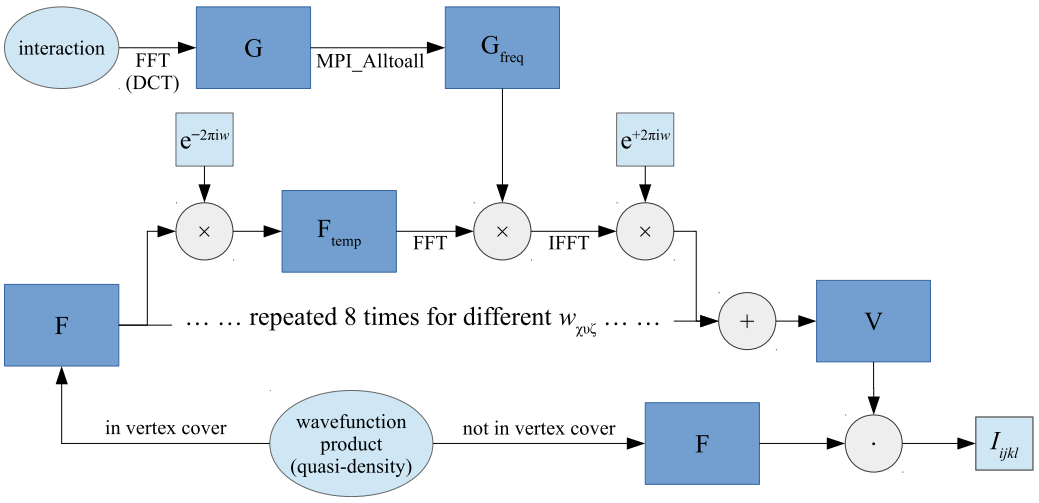

Fig. 2. Visualization of the implemented method presenting flow of data through the internal buffers (F, G, V, etc. marked with dark background). Cross-product signs (×) mark the element-wise multiplication, while the dot-product sign (·) involves a summation over all grid points. IFFT = Inverse Fast Fourier Transform.

For example, pattern `ijji` matches integrals `1111`, `1221`, `2332` etc. while pattern `i1i2` matches integrals `1112`, `2122`, `3132` etc. Patterns with no repeated letters (e.g. `ijkl`) are equivalent to `****`, which is the default settings—if the parameter *integrals* is not given, all integrals are computed.

Following the generated computation plan, all nodes synchronously perform a sequence of steps, where each step can be either computing (and locally storing) a quasi-potential V_{jk} from a given quasi-density ρ_{jk}, or computing an element-wise product between pre-computed potential V_{il} and a given quasi-density ρ_{jk}, resulting in the value for the integral E_{ijkl}.

Memory-efficient fast convolution method described in Section 3.1 is implemented in class *CoulombCalculator*, which internally allocates all the needed buffers at its instantiation. Visualization of computation scheme and flow of data through internal buffers are presented in Fig. 2. For calculating fast Fourier transforms (and the discrete cosine transform in case of interaction function G), we use specialized distributed-memory parallel routines from FFTW.

There are three caveats resulting from this data distribution:

1. Due to the internal storage format, computational box is distributed among MPI nodes in the z direction, while FFTW expects the multi-dimensional data to be distributed in the x direction. This is easily solved by permuting the order of the dimensions passed to FFTW routines.
2. Since we use a specialized version of the parallel FFT, output data have y and z dimensions swapped, and therefore, all computed Fourier transforms are distributed in the y direction. This is properly accounted for in *CoulombCalculator*, and does not constitute a problem, since we never need to directly combine real-space with momentum-space data.
3. Due to the formulation of Eqs. (16) and (17), the required y range of $\mathcal{F}G$ needed by each node differs from the range for other momentum-space domains $\mathcal{F}V$ and $\mathcal{F}\rho$. Therefore, after computing discrete cosine transform of G, the all-to-all communication round is performed, re-distributing the required slices of the $\mathcal{F}G$ domain between all MPI nodes.

5.1. Choosing the dielectric model

Coulombo supports both the static dielectric model with dielectric constant ϵ_r

$$\epsilon(\vec{r}) = \epsilon_r \tag{19}$$

as well as more realistic (in the context of nanostructure calculations) distance-dependent model proposed in [32] and defined as

$$\epsilon(\vec{r}) = \begin{cases} \epsilon_r \frac{qR}{\sinh[q(R-|\vec{r}|)]+q|\vec{r}|} & |\vec{r}| < R \\ \epsilon_r & |\vec{r}| \geq R \end{cases}, \tag{20}$$

where q and R are constants characterizing the material of the analyzed system: q is calculated as

$$q = \frac{2}{\sqrt{\pi}} \frac{(96\pi^2)^{\frac{1}{3}}}{a_{TF}} \tag{21}$$

where a_{TF} is a value of the command-line parameter *tf-lattice* and R can be found as a solution to the equation

$$\frac{\sinh(qR)}{qR} = \epsilon_r \,. \tag{22}$$

The program can be easily extended to handle other dielectric screening models, by inheriting from class *Interaction*.

6. Benchmarks

Benchmarks were performed on sets of six single-particle wavefunctions each computed on a three-dimensional cube of silicon atoms forming the zinc-blende lattice structure. The calculation was performed for cubes of different dimensions, and therefore the different number of atoms in the calculation. The overall (wall) time of calculation was measured after each individual calculation. Here we do not discuss the details (or computer codes) of the tight-binding calculation since it was discussed elsewhere [24], but rather focus on the most time-consuming stage: the calculation of the integrals.

The approximate relation between the number of atoms A and number of grid points $N = N_x N_y N_z$ is given as

$$A \approx 8N \left(\frac{h}{a_{Si}}\right)^3, \tag{23}$$

where h is a grid step (for the benchmark we assume $h_x = h_y = h_z$ and $N_x = N_y = N_z$) and a_{Si} is the lattice constant for silicon bulk crystal $a_{Si} = 5.431$ Å. A grid step of $h = 0.8$ Å was used throughout the benchmark process.

To validate the scaling of our method, we computed all possible ($6^4 = 1296$) Coulomb matrix elements for a series of systems ranging from $A = 10{,}000$ to $A = 1000{,}000$ (one million) atoms

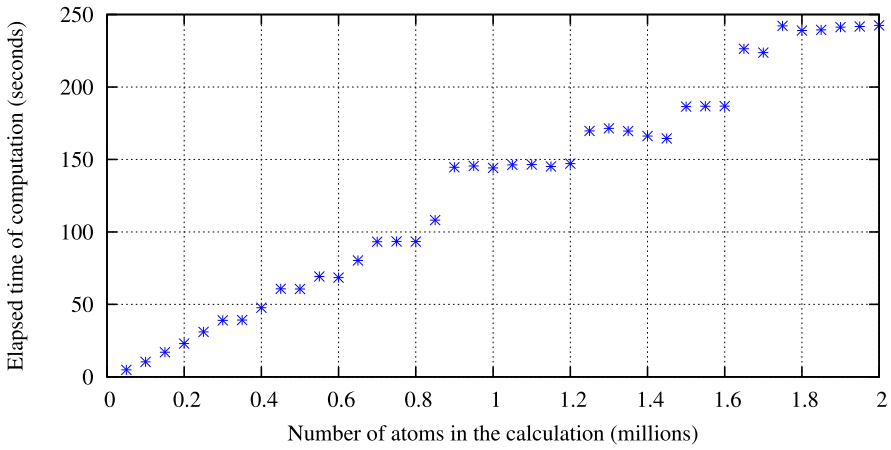

Fig. 3. Plot of the wall-clock calculation time for all 1296 Coulomb matrix elements for a series of different systems with varying number of atoms.

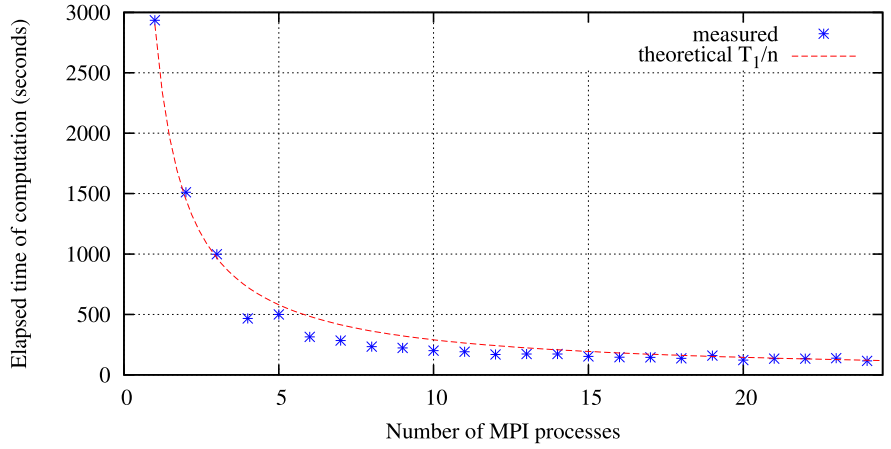

Fig. 4. Wall clock calculation time of all 1296 Coulomb matrix elements of a system of one million silicon atoms, versus a number of allocated MPI processes.

(corresponding to N varying from X to Y) using 16 MPI processes on a multi-CPU system with Intel® Xeon® E5-4640 processors. Wall clock time of the calculations is presented in Fig. 3 as a quasi-linear trend, with visible steps resulting from padding the computational box as described in Section 5. We would like to emphasize at this point that the computation of all matrix elements takes only a few minutes for multi-million-atom system, so the method is naturally suited (due to quasi-linear scaling) also for much larger systems, even up to 100 million atoms.

Finally, in order to evaluate the parallelization scheme used in Coulombo, we performed the series of calculations for the system with $A = 1000{,}000$ and with different number of MPI processes ranging from 1 to 25. Wall clock time of the calculations is presented in Fig. 4. The apparent super-linear speedup is not an artifact and it has been verified by repeated measurements. It can be attributed to the more efficient use of CPU cache, since each core is responsible only for a small part of the input data.

7. Conclusions

We have presented an efficient computer code aimed at quasi-linear scaling computation of Coulomb and exchange integrals for wave-functions defined on a grid. The code is effectively parallelized using MPI and it utilizes well-established FFTW and Armadillo libraries. The program takes as its input a set of particle wave functions given on a three-dimensional, spatial grid and saved as files in simple binary format. The output is presented as a list of (all or selected subset) calculated Coulomb matrix elements. The calculation of many-integral cases is optimized by finding a minimum vertex cover of a graph. The calculated integrals are free from spurious contribution from periodic super-cells, by using an effective padding technique that does not increase the size of the computational domain.

By applying all optimizations, the cost of computing all M^4 integrals for a single-particle basis consisting of M wave-functions defined on N grid points ($M \ll N$), has been reduced to $O(M^2 N \log N + M^4 N)$. Such scaling properties make the program most suitable for applications in nanostructure theory, namely calculations of excitonic spectra of nanowires and nanowire quantum dots, as well as modeling of dopants systems embedded in larger multi-million atom blocks of surrounding semiconductor material. Further applications, e.g. in quantum chemistry, are also possible.

Acknowledgments

The authors would like to thank Jacek Kobus for valuable comments. The authors acknowledge support from the Polish National Science Centre based on decision No. 2015/18/E/ST3/00583.

Appendix. Minimum vertex cover implementation

Finding a minimum vertex cover of a graph is one of the well-known NP-complete problems, which means that unless P=NP, it cannot be solved in polynomial time. Therefore, we use a greedy approximation algorithm:

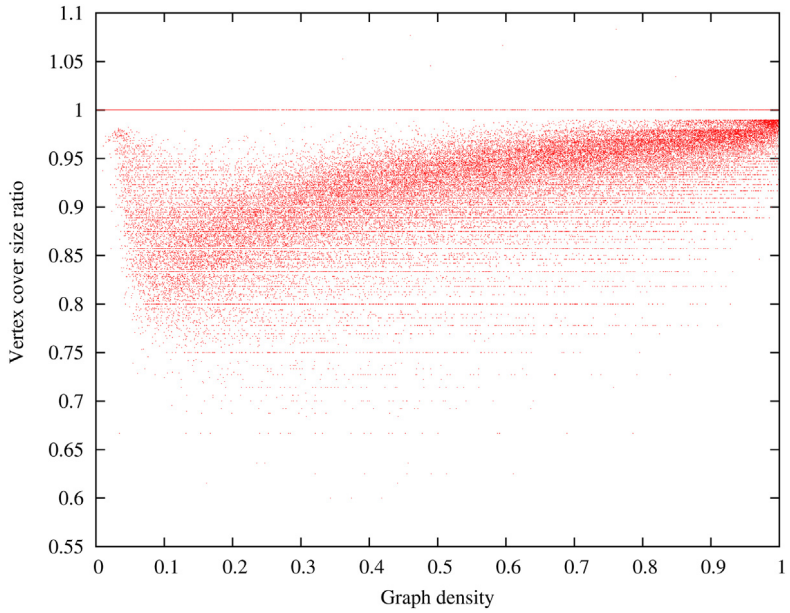

Fig. A.5. Performance evaluation of the proposed vertex cover heuristic. Each point represents a single randomly generated graph. Value on the vertical axis represents a ratio r between the size of the vertex cover obtained with our proposed heuristic and the size of the vertex cover obtained using the reference solution.

Data: undirected graph $(V_0, E_0) = G_0$
Result: vertex cover $C \subset V_0$
$C := \varnothing$;
$(V, E) = G := G_0$;
remove all vertices $v \in V$ with degree(v) = 0;
while *V is not empty* **do**
  if *there is a vertex $w \in V$ with degree(w) = 1* **then**
    v := the only neighbor of w;
  else
    v := vertex from G with a greatest degree;
  end
  $C := C \cup \{v\}$;
  remove v (and all the adjacent edges) from graph G;
end

This heuristic can be easily implemented with $O(|V| \log |V|)$ computational complexity, using a priority queue. In our implementation we used a *std::set* implementation from C++ Standard Template Library, which is guaranteed to keep the ordering of the elements (and therefore, allow the extraction of the "largest" element in $O(1)$ time), while the removing and adding elements is implemented in $O(\log |V|)$ time.

To compare the performance of the proposed method with a well-known reference approximation algorithm proposed in [33], we compared sizes of the vertex covers from both methods for a series of random graphs. Size of the vertex cover is a good performance measure (smaller size is better), so the ratio

$$r = \frac{\text{size of the vertex cover of the proposed method}}{\text{size of the vertex cover of the reference method}} \tag{A.1}$$

is a good indicator of the method's performance ($r < 1$ stands for better performance than the reference method).

We generated the r ratio for 100,000 random graphs of $|V|$ between 10 and 100. For each realization:

- we generate the size of the graph ($|V|$) by drawing from the discrete uniform distribution between 10 and 100,
- we generate the graph density D by drawing from the uniform distribution between 0 and 1,
- we iterate through all possible edges, and either add it to the graph (with probability D), or skip it (with probability $1 - D$).

We do not allow loops (edges $v \leftrightarrow v$) since they can be easily handled at the pre-processing step, as each node forming such a loop must be included in a vertex cover. Therefore, each graph with graph density equal to D will have, on average, $D\frac{V(V-1)}{2}$ edges.

A plot of the r values versus graph density D is presented in Fig. A.5. Apart from the set of cases where both methods perform exactly the same (line $r = 1$), the proposed heuristic performs significantly better than a reference solution. This is despite the fact that the greedy approach is usually presented as a sub-optimal solution to the minimal vertex cover problem. We assume that the usual worst-case behavior of the greedy approach is most likely fixed by introducing an additional step at every iteration for handling nodes with degree = 1.

References

[1] A.J. Williamson, L.W. Wang, A. Zunger, Phys. Rev. B 62 (2000) 12963–12977, http://dx.doi.org/10.1103/PhysRevB.62.12963.
[2] G. Bester, S. Nair, A. Zunger, Phys. Rev. B 67 (2003) 161306, http://dx.doi.org/10.1103/PhysRevB.67.161306.
[3] G. Klimeck, F. Oyafuso, T. Boykin, R. Bowen, P. von Allmen, Comp. Modeling in Eng. and Sci. (CMES) 3 (2002) 601, URL https://engineering.purdue.edu/gekcogrp/publications/pubs_src/J_2002_1_freePU.pdf.
[4] F.A. Zwanenburg, A.S. Dzurak, A. Morello, M.Y. Simmons, L.C.L. Hollenberg, G. Klimeck, S. Rogge, S.N. Coppersmith, M.A. Eriksson, Rev. Modern Phys. 85 (2013) 961–1019, http://dx.doi.org/10.1103/RevModPhys.85.961.
[5] W. Jaskólski, M. Zieliński, G.W. Bryant, J. Aizpurua, Phys. Rev. B 74 (2006) 195339, http://dx.doi.org/10.1103/PhysRevB.74.195339.
[6] M. Zieliński, Phys. Rev. B 86 (2012) 115424, http://dx.doi.org/10.1103/PhysRevB.86.115424.
[7] G.W. Bryant, M. Zieliński, N. Malkova, J. Sims, W. Jaskólski, J. Aizpurua, Phys. Rev. B 84 (2011) 235412, http://dx.doi.org/10.1103/PhysRevB.84.235412.
[8] G. Bester, A. Zunger, Phys. Rev. B 71 (2005) 045318, http://dx.doi.org/10.1103/PhysRevB.71.045318.
[9] M. Zieliński, Nanoscale Res. Lett. (ISSN: 1556-276X) 7 (1) (2012) 265, http://dx.doi.org/10.1186/1556-276X-7-265.
[10] M. Zieliński, J. Phys.: Condens. Matter 25 (46) (2013) 465301, URL http://stacks.iop.org/0953-8984/25/i=46/a=465301.
[11] M. Zieliński, Y. Don, D. Gershoni, Phys. Rev. B 91 (2015) 085403, http://dx.doi.org/10.1103/PhysRevB.91.085403.
[12] N. Akopian, G. Patriarche, L. Liu, J.-C. Harmand, V. Zwiller, Nano Lett. 10 (4) (2010) 1198–1201, http://dx.doi.org/10.1021/nl903534n.

[13] M. Bouwes Bavinck, K.D. Jöns, M. Zieliński, G. Patriarche, J.-C. Harmand, N. Akopian, V. Zwiller, Nano Lett. 16 (2) (2016) 1081–1085, http://dx.doi.org/10.1021/acs.nanolett.5b04217.
[14] M. Zieliński, Phys. Rev. B 88 (2013) 115424, http://dx.doi.org/10.1103/PhysRevB.88.115424.
[15] R. Singh, G. Bester, Phys. Rev. B 84 (2011) 241402, http://dx.doi.org/10.1103/PhysRevB.84.241402.
[16] R. Singh, G. Bester, Phys. Rev. Lett. 104 (2010) 196803, http://dx.doi.org/10.1103/PhysRevLett.104.196803.
[17] M. Zieliński, K. Gołasa, M.R. Molas, M. Goryca, T. Kazimierczuk, T. Smoleński, A. Golnik, P. Kossacki, A.A.L. Nicolet, M. Potemski, Z.R. Wasilewski, A. Babiński, Phys. Rev. B 91 (2015) 085303, http://dx.doi.org/10.1103/PhysRevB.91.085303.
[18] M. Zieliński, Phys. Rev. B 88 (2013) 155319, http://dx.doi.org/10.1103/PhysRevB.88.155319.
[19] J.-M. Jancu, R. Scholz, F. Beltram, F. Bassani, Phys. Rev. B 57 (1998) 6493–6507, http://dx.doi.org/10.1103/PhysRevB.57.6493.
[20] M. Zieliński, M. Korkusinski, P. Hawrylak, Phys. Rev. B 81 (2010) 085301, http://dx.doi.org/10.1103/PhysRevB.81.085301.
[21] W. Jaskólski, M. Zieliński, G.W. Bryant, Acta Phys. Pol. A 106 (2004) 193.
[22] M. Zieliński, Acta Phys. Pol. A 122 (2012) 312, URL http://przyrbwn.icm.edu.pl/APP/PDF/122/a122z2p15.pdf.
[23] M. Usman, J. Bocquel, J. Salfi, B. Voisin, A. Tankasala, R. Rahman, M.Y. Simmons, S. Rogge, L.C.L. Hollenberg, Nature Nanotechnol. 11 (2016) http://dx.doi.org/10.1038/nnano.2016.83, 763 EP –.
[24] P.T. Różański, M. Zieliński, Phys. Rev. B 94 (2016) 045440, http://dx.doi.org/10.1103/PhysRevB.94.045440.
[25] M. Świderski, M. Zieliński, Phys. Rev. B 95 (2017) 125407, http://dx.doi.org/10.1103/PhysRevB.95.125407.
[26] E. Nielsen, R. Rahman, R.P. Muller, J. Appl. Phys. 112 (11) (2012) 114304, http://dx.doi.org/10.1063/1.4759256.
[27] G. Makov, M.C. Payne, Phys. Rev. B 51 (1995) 4014–4022, http://dx.doi.org/10.1103/PhysRevB.51.4014.
[28] E.L. de Oliveira, E.L. Albuquerque, J.S. de Sousa, G.A. Farias, F.M. Peeters, J. Phys. Chem. C 116 (7) (2012) 4399–4407, http://dx.doi.org/10.1021/jp2088516.
[29] A. Franceschetti, H. Fu, L.W. Wang, A. Zunger, Phys. Rev. B 60 (1999) 1819–1829, http://dx.doi.org/10.1103/PhysRevB.60.1819.
[30] W. Sheng, S.J. Cheng, P. Hawrylak, Phys. Rev. B 71 (2005) 035316, http://dx.doi.org/10.1103/PhysRevB.71.035316.
[31] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes 3rd Edition: The Art of Scientific Computing, third ed., Cambridge University Press, 2007.
[32] R. Resta, Phys. Rev. B 16 (1977) 2717–2722, http://dx.doi.org/10.1103/PhysRevB.16.2717.
[33] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, in: Series of Books in the Mathematical Sciences, W. H. Freeman, 1979.