DOI: 10.14529/jsfi260105

MPI+OpenMP Implementation of Resolution-of-the-Identity Hartree—Fock Method Exploiting Permutational Symmetry of Three-Center Electron Repulsion Integrals

Iurii V. Kashpurovich[1,2], *Alexander V. Oleynichenko*[3,2], *Vladimir V. Stegailov*[1,2,4]

We report a high-performance implementation of the resolution-of-the-identity Hartree—Fock method that fully exploits the permutational symmetry of three-center electron repulsion integrals (ERIs). The present implementation adopts a hybrid MPI+OpenMP parallelization strategy. Two different algorithmic approaches (with and without the preliminary transformation of ERIs) are analyzed and compared. A custom data layout introduced previously is employed. Designed to efficiently leverage the permutational symmetry of ERIs, it minimizes not only inter-node communication but also local memory traffic. Other extensive low-level and algorithmic optimizations are proposed and discussed. Reasonable parallel scaling is demonstrated by performance benchmarks on a chlorophyll dimer $(C_{55}H_{72}O_5N_4Mg)_2$ in an aqueous environment of 48 molecules (322 atoms overall, 3700 and 11896 functions in main and auxiliary basis sets, respectively). Peak speedups of 84× and 71× on 128 threads are achieved for the ERI calculation and the exchange matrix construction, respectively, within the algorithm involving the preliminary transformation.

Keywords: restricted Hartree—Fock method, resolution-of-the-identity, density fitting, three-center electron repulsion integrals, MPI, OpenMP.

Introduction

The Hartree—Fock (HF) method, also known as the self-consistent field (SCF) method [8], is a basic approach to solving the many-body electronic structure problem in modern quantum chemistry. It serves both as a starting point for more advanced electron correlation methods and as a conceptual and software base for density functional theory (DFT). The computational complexity of a straightforward SCF algorithm scales as $O(N^4)$ with a system of N atoms, and numerous computational techniques and their program implementations with reduced scaling were proposed in the last decades to overcome this SCF deficiency. One of the most prominent modern approaches is the resolution-of-the-identity (RI) approximation, also known as the density fitting technique (DF) [4, 5, 9, 10, 12, 15, 16, 20, 28, 30, 31, 33, 34, 36, 38, 39, 42, 44, 47, 49]. Working expressions of the RI-HF theory employ three- and two-center electron repulsion integrals (ERIs) defined as

$$(\mu\nu|B) = \int \frac{\chi_\mu(\mathbf{r}_1)\chi_\nu(\mathbf{r}_1)\phi_B(\mathbf{r}_2)}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2, \tag{1}$$

$$V_{BC} = \int \frac{\phi_B(\mathbf{r}_1)\phi_C(\mathbf{r}_2)}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2, \tag{2}$$

where the basis functions χ_μ, $\mu = 1, \ldots, N_{\text{AO}}$ and the auxiliary basis functions ϕ_B, $B = 1, \ldots, N_{\text{aux}}$ are typically represented by atom-centered Gaussian-type orbitals (atomic or-

[1]Joint Institute for High Temperatures of Russian Academy of Sciences, Moscow, Russian Federation
[2]Moscow Institute of Physics and Technology, Moscow, Russian Federation
[3]Petersburg Nuclear Physics Institute named by B.P. Konstantinov of NRC "Kurchatov Institute", Gatchina, Russian Federation
[4]HSE University, Moscow, Russian Federation

bitals, AOs). Array (1) is also referred to as RI tensor. In addition to reducing the computational scaling of the SCF method to $O(N^\alpha)$ ($\alpha \in [3.0, 4.0)$), the RI approximation allows one to store all ERIs (including $(\mu\nu|B)$) in local or distributed random access memory (RAM) for most practical problems of interest [15, 39, 45] as well as aggregate high-performance tensor contraction engines (such as BLAS [3] libraries) and hardware accelerators (such as GPUs [4, 5, 42, 53]).

The first distributed RI-SCF implementation [12] stored all ERIs (1) to disk using a compression. Then the integrals were fully unpacked at each SCF iteration and computations with dense tensors were performed. Up to now, permutational symmetry of $(\mu\nu|B)$ with respect to indices μ and ν was accounted for only to efficienly store the RI tensor in memory. In the recently reported implementation presented by our group [25] it was also exploited for the Fock matrix assembly stage, although at the cost of additional technical complications. There are other challenges as well. For example, for systems consisting of hundreds of atoms conventional HF calculations (not employing the RI approximation) can be even faster than those using RI-HF [22]. It is because of difficulties to deal with the sparsity of intermediate arrays within RI-SCF procedure, while the scaling of the straightforward "conventional" HF approach can be reduced to $O(N^2)$, if one employs a proper integral screening [7, 14, 18]. Other issues are the lack of GPU-accelerated RI-HF implementations [4, 5, 42, 53] and of massively-parallel implementations of the two-component quasirelativistic RI-SCF method.

Our goal is to gradually fill the specified gaps within the new BUFO program package for quantum chemistry simulations. This paper reflects the current status of our progress and summarizes several previous developments [25–27] aimed at the efficient parallel implementation of the RI-HF method. Careful benchmarking of parallel algorithms for electronic structure calculations is instrumental in achieving the best performance of supercomputer systems [41]. Here we present two different MPI+OpenMP algorithms of the RI-HF method and conduct performance tests using two HPC systems: a 2-socket server with shared memory and a 32-node supercomputer with distributed memory. The key feature of the presented implementation is storing three-center ERIs (1) in the distributed RAM using the permutationally-adapted memory layout in order to minimize not only communications between processes, but also local memory movement [25].

The paper is organized as follows. Section 1 outlines the theoretical background of the spin-restricted RI-HF method. Different approaches for its implementation for distributed memory architectures and actual algorithms are described in Section 2. In Section 3 we highlight the parallel computers employed to test the scalability of the developed implementations and also describe molecular system for benchmark calculations in details. Results are presented in Section 4. Conclusion provides some remarks and specifies future directions of work.

1. Theoretical Background

Throughout this paper, we adopt the following index labeling conventions:

1. i: occupied molecular orbitals, with range N_{occ};
2. μ, ν, ρ, σ: atomic basis functions (AOs), with range N_{AO};
3. A, B, P: auxiliary RI basis functions, with range N_{aux}.

For the sake of simplicity, restricted Hartree—Fock (RHF) theory is considered further, though our approaches can be extended to unrestricted and two-component quasirelativistic version of the method as well. The primary goal of the SCF algorithm is to determine expansion

coefficients $C_{i\mu}$ defining spatial molecular orbitals (MOs) $\varphi_i(\mathbf{r})$,

$$\varphi_i(\mathbf{r}) = \sum_{\mu=1}^{N_{\text{AO}}} C_{i\mu}\chi_\mu(\mathbf{r}). \tag{3}$$

The coefficients $C_{i\mu}$ are needed to construct density matrix $D_{\mu\nu}$

$$D_{\mu\nu} = 2\sum_{i=1}^{N_{\text{occ}}} C_{i\mu}C_{i\nu}. \tag{4}$$

These coefficients are obtained by diagonalizing the Fock matrix $F_{\mu\nu}$

$$F_{\mu\nu} = h_{\mu\nu} + J_{\mu\nu} - K_{\mu\nu}, \tag{5}$$

where $h_{\mu\nu}$ stands for the core Hamiltonian. The Coulomb and exchange matrices $J_{\mu\nu}$ and $K_{\mu\nu}$ within the RI approximation are constructed according to the expressions

$$J_{\mu\nu} = \sum_{\rho,\sigma}^{N_{\text{AO}}}\sum_{B,C}^{N_{\text{aux}}} D_{\rho\sigma}(\mu\nu|B)V^{-1}_{BC}(C|\rho\sigma), \tag{6}$$

$$K_{\mu\nu} = \frac{1}{2}\sum_{i}^{N_{\text{occ}}}\sum_{\rho,\sigma}^{N_{\text{AO}}}\sum_{B,C}^{N_{\text{aux}}} C_{i\sigma}C_{i\rho}(\mu\sigma|B)V^{-1}_{BC}(C|\rho\nu). \tag{7}$$

The V^{-1}_{BC} matrix is a symmetric positive definite matrix, so it can be factorized using the Cholesky decomposition with factor L_{BP}. The latter can be used to transform three-center ERIs

$$\widetilde{(\mu\nu|P)} = \sum_{B}^{N_{\text{aux}}} (\mu\nu|B)L_{BP} \tag{8}$$

in order to get simplified expressions for Coulomb and exchange matrices

$$J_{\mu\nu} = \sum_{\rho,\sigma}^{N_{\text{AO}}}\sum_{P}^{N_{\text{aux}}} D_{\rho\sigma}\widetilde{(\mu\nu|P)}\widetilde{(P|\rho\sigma)}, \tag{9}$$

$$K_{\mu\nu} = \frac{1}{2}\sum_{i}^{N_{\text{occ}}}\sum_{\rho,\sigma}^{N_{\text{AO}}}\sum_{P}^{N_{\text{aux}}} C_{i\sigma}C_{i\rho}\widetilde{(\mu\sigma|P)}\widetilde{(P|\rho\nu)}. \tag{10}$$

2. Algorithm Design

2.1. High-Level Description

In this work, two different implementations of the RI-RHF method are presented. The first (RI-JK) refers to Eqs. (6) and (7) further, while the second one (RI-TJK) to Eqs. (9) and (10). Corresponding high-level view to both algorithms is sketched in Algorithm 1, so they are composed of stages:

- (line 1) scheduling of integration;
- (line 2) calculating ERIs: note that we adopt the row-major ordering convention in this work, so the last index is the fastest-varying; two algorithms initially store three-center ERIs using different layouts; the upper index p identifies the dimension being partitioned between processes;

- (line 3) performing the transformation (8) and the global transposition of $(\widetilde{\mu\nu|B})$ only in case of the RI-TJK (i.e., $(\widetilde{B|\mu\nu}) \equiv (B|\mu\nu)$ for the RI-JK) and calculating the inverse V_{BC}^{-1} and its Cholesky decomposition L_{BC} for both algorithms;
- (line 4) SCF procedure.

Each step is discussed in detail in the following subsections. However, the focus is on the three-center ERIs, as they constitute the core of the RI approximation.

2.2. Scheduling ERIs Calculation

First of all, let us note that modern highly efficient libraries for the calculation of integrals (like those used in this work `libcint` [43] and `libgrpp` [37]) operate on shells of basis functions, where the shell groups functions sharing the same exponent and angular momentum. We will further assume that N_{AO} basis functions are grouped into $N_{\text{AO}}^{\text{sh}}$ shells, while N_{aux} functions into $N_{\text{aux}}^{\text{sh}}$.

Scheduling is needed to overcome challenges of the integration:

- computational workload balancing;
- fixed pattern of storing integrals.

The better the first one is resolved, the better scalability would be achieved. Then it seems that all $(N_{\text{AO}}^{\text{sh}})^2 N_{\text{aux}}^{\text{sh}}$ triplets of shells should be distributed dynamically between processes. Although this approach can easily lead to $O(N_{\text{AO}}^2 N_{\text{aux}})$ communication volume. However, dealing with the second challenge allows to reduce this communication, while imposing restrictions on the granularity of the distribution of shell triplets. Now, these triplets should be grouped in order to fill the whole block of the RI tensor. These blocks are composed of subblocks depicted in Fig. 1 as regions bounded by red lines (note that integrals are stored row-wise). So, depending on the algorithm, the blocks can be:

- Groups of rows of size N_{aux} in case of the RI-JK (see Fig. 1a). Note that red lines in Fig. 1a also show removed rows (they are removed in order to store permutationally-unique integrals only).
- Groups of triangles of the matrix $N_{\text{AO}} \times N_{\text{AO}}$ in case of the RI-TJK (see Fig. 1b). The calculation in this case may start and proceed from either upper or lower triangle. Each group is formed appropriately.

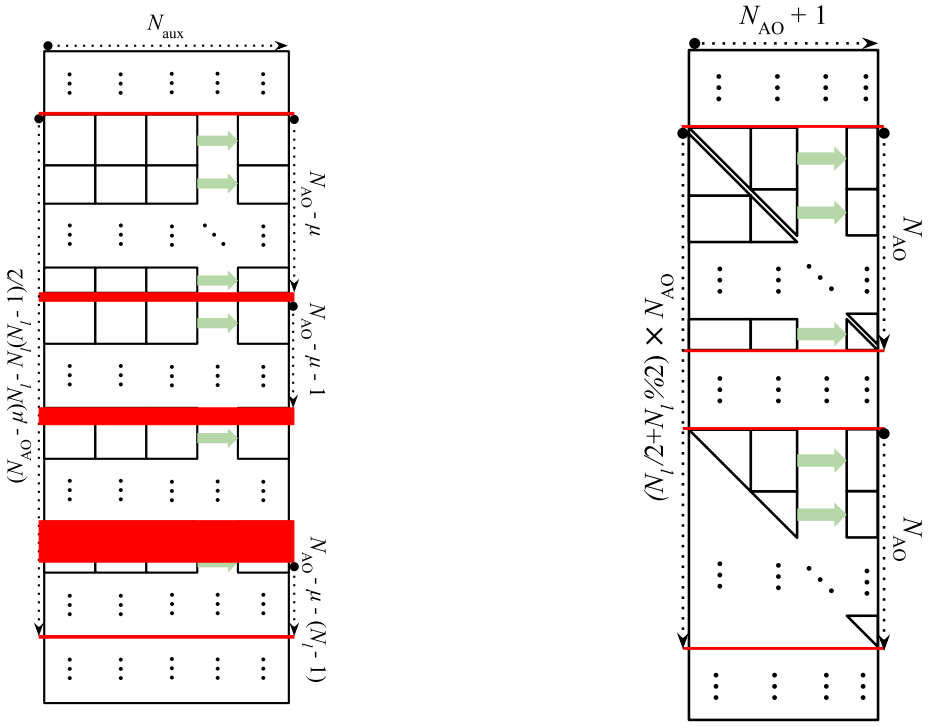
Thus, either $N_{\text{AO}}^{\text{sh}}$ (or even $(N_{\text{AO}}^{\text{sh}})^2$) or $N_{\text{aux}}^{\text{sh}}$ basis shells can be distributed initially in a static manner like in this work: a shell of $N_{\text{AO}}^{\text{sh}}$ or $N_{\text{aux}}^{\text{sh}}$ composed of the largest number of functions is identified and pushed to the process task queue with the smallest current number of ERIs expected to be calculated. This distribution provides predictable data placement of integrals in memory and a compromise between task granularity and subsequent communication volume. As a result of scheduling stage, shells are specifically ordered in the global workspace, so the

Algorithm 1 RI-RHF method

Input: Number of main basis shells: $N_{\text{AO}}^{\text{sh}}$; number of auxiliary basis shells: $N_{\text{aux}}^{\text{sh}}$

Output: Coefficients of MOs (Eq. (3)): $C_{i\mu}$

1: `scheduler`, $\texttt{ord}_{\text{AO}}$, $\texttt{ord}_{\text{aux}}$ ← ScheduleGlobalIntegrationShellOrder($N_{\text{AO}}^{\text{sh}}$, $N_{\text{aux}}^{\text{sh}}$)
2: $h_{\mu\nu}$, $(\mu^{(p)}\nu|B)$ or $(B^{(p)}|\mu\nu)$, V_{BC} ← CalculateERIs(`scheduler`, $\texttt{ord}_{\text{AO}}$, $\texttt{ord}_{\text{aux}}$)
3: $(\widetilde{B^{(p)}|\mu\nu})$, V_{BC}^{-1}, L_{BC} ← TransformERIs($(\mu^{(p)}\nu|B)$ or $(B^{(p)}|\mu\nu)$, V_{BC})
4: $C_{i\mu}$ ← SCFProcedure($h_{\mu\nu}$, $(\mu\nu|\widetilde{B^{(p)}})$, V_{BC}^{-1}, L_{BC})



(a) Row-majored $(\mu\nu|B)$. N_l is the size of the shell μ_l; μ is the number of the first function from μ_l in the basis

(b) Permutationally-adapted $(B|\mu\nu)$. N_l is the size of the shell B_l

Figure 1. Memory layout of the RI tensor (1) and scheme of ERI calculation in batches

corresponding $\texttt{ord}_{\text{AO}}$ and $\texttt{ord}_{\text{aux}}$ arrays of them are returned to perform integration (Alg. 1, line 1) within `CalculateERIs` function (Alg. 1, line 2).

Ideally, the dynamic `scheduler` should also be returned and then passed to `CalculateERIs` function to balance computational workload during calculation (especially if prescreening techniques are applied [21, 48]). It is not so easy to implement such a `scheduler` efficiently. Different approaches are tried within conventional HF [1, 11, 17, 23, 35, 54] and RI-HF calculations [6, 12, 42]. The best algorithms should combine the features of static and dynamic load-balancing. Nevertheless, our research does not focus on dynamic scheduling only proposing and discussing suitable granularity and initial distribution.

2.3. Calculation of Three-Center ERIs

After distributing shells $N_{\text{AO}}^{\text{sh}}$ or $N_{\text{aux}}^{\text{sh}}$ between processes, each process p has a number of shells either $\mu_l \in \{\mu_{l_0}^{(p)} \dots \mu_{l_n}^{(p)}\}$ in case of RI-JK or $B_l \in \{B_{l_0}^{(p)} \dots B_{l_n}^{(p)}\}$ in case of RI-TJK (in both cases $n = n(p)$, so it depends on initial static distribution of shells). These sets were extracted from the $\texttt{ord}_{\text{AO}}$ and $\texttt{ord}_{\text{aux}}$ arrays. Each shell represents the specific task of performing integration and filling the corresponding subblock of the RI tensor (see Fig. 1: the arrows indicate the progress of filling the RI tensor by batches, shown as rectangles and triangles). Both algorithms have a similar structure that is described in Algorithm 2 for the first of them. For each shell μ_l (line 1) or B_l of size N_l either $(N_{\text{AO}} - \mu)N_l - (N_l - 1)N_l/2$ rows[5] (line 2) or $N_l/2 + N_l\%2$ triangles[6] of the RI tensor should be filled with integrals, respectively. Then a block of `size` rows or triangles

[5] μ is the number of the first function from the shell μ_l in basis

[6] "%" represents the remainder of the division

Algorithm 2 Calculation of three-center ERIs (1)

Input: initial main basis shells distribuiton $\{\mu_{l_0}^{(p)} \dots \mu_{l_n}^{(p)}\}$ on given process p; number of main basis shells: $N_{\text{AO}}^{\text{sh}}$; number of auxiliary basis shells: $N_{\text{aux}}^{\text{sh}}$
Output: $(\mu^{(p)}\nu|B)$, $\mu \leq \nu$

1: **for** $\mu_l \in \{\mu_{l_0}^{(p)} \dots \mu_{l_n}^{(p)}\}$ **do**
2: `size` $\leftarrow$ GetBlockSize(μ_l)
3: **for** $\nu_l \in \{\mu_l \dots N_{\text{AO}}^{\text{sh}}\}, B_l \in \{0 \dots N_{\text{aux}}^{\text{sh}}\}$ **do** in parallel (OMP)
4: `batch` $\leftarrow$ CalculateBatchERIs(μ_l, ν_l, B_l)
5: $(\mu\nu|B^{(p)}) \leftarrow$ PlaceBatchERIsToRITensor($(\mu\nu|B^{(p)})$, `batch`, `offset`, μ_l, ν_l, B_l)
6: **end for**
7: `offset` $\leftarrow$ UpdateOffset(`size`)
8: **end for**

corresponding to shell μ_l or B_l is filled in the `for` loop (lines 3–6) parallelized using dynamic OpenMP policy in current implementations. Finally, after the block is ended, the `offset` is updated (line 7) to continue filling from the correct new position. For the first algorithm it is just increased by `size`, while for the second one lower or upper triangle should be identified as well in order to proceed.

Precise compromise between task granularity and subsequent communication volume can be achieved through grouping shells from external `for` loop (line 1) as well as tiling of the nested `for` loops (line 3), e.g., in a way that task would be characterized not only by shell number μ_l or B_l, but also by groups $\{\mu_{l_{g_1}}^{(p)} \dots \mu_{l_{g_n}}^{(p)}\}$ or $\{B_{l_{g_1}}^{(p)} \dots B_{l_{g_n}}^{(p)}\}$ as well as by tiles $\{\nu_{l_{t_1}}^{(p)} \dots \nu_{l_{t_n}}^{(p)}\}$ or $\{\mu_{l_{t_1}}^{(p)} \dots \mu_{l_{t_n}}^{(p)}\}$, respectively. Work on each task or even on several of them can be parallelized using OpenMP more efficiently then.

As a result of the algorithms part of the three-center ERIs is stored in process p memory. This part is $(\mu^{(p)}\nu|B)$ for RI-TJK and $(B^{(p)}|\mu\nu)$ for RI-JK. However, three-center ERIs are not distributed evenly between computational nodes by default. This is because the integration was scheduled by shells. Ideally, additional balancing of the three-center ERIs distribution during or after integration is needed for maximum efficiency of program execution, although it will be implemented in the future.

Finally, note that the proposed algorithms are easily adaptable to dynamic workflow: shell μ_l or B_l can be pushed to or extracted from the task queue (in fact, noted as $\{\mu_{l_0}^{(p)} \dots \mu_{l_n}^{(p)}\}$ or $\{B_{l_0}^{(p)} \dots B_{l_n}^{(p)}\}$ above) of the process p. This is also true for tasks created in a more complex manner.

2.4. Transformation and Transposition of ERIs

Before starting the SCF procedure, three-center integrals are stored either using permutationally-unique row-major or permutationally-adapted format (see Fig. 1). The purpose of the transformation (8) is to simplify subsequent SCF computations. In turn, this transformation requires $N_{\text{AO}}^2 N_{\text{aux}}^2$ arithmetic operations and subsequent distributed transposition of the $N_{\text{AO}}(N_{\text{AO}}+1)/2 \times N_{\text{aux}}$ matrix. In contrast, the alternative algorithm performs distributed transposition of the $N_{\text{occ}}N_{\text{AO}} \times N_{\text{aux}}$ matrix and $N_{\text{occ}}N_{\text{AO}}N_{\text{aux}}^2$ arithmetic operations subsequently in each iteration of the SCF during the construction of the exchange matrix. Thus, the

Algorithm 3 Transformation of three-center ERIs

Input: non-transformed row-majored three-center integrals $(\mu\nu|B^{(p)})$ (1); Cholesky factor L_{BP} of V_{BC}^{-1} (see Eq. (2)); initial distribution of rows $\overline{(\mu\otimes\nu)^{(p)}\dots(\mu\otimes\nu)^{(p+1)}}$ on process p; size of the main basis N_{AO}; size of the auxiliary basis N_{aux}

Output: $\widetilde{(P^{(p)}|\mu\nu)}$, $\mu \le \nu$

1: **for** $(\mu\otimes\nu) = \overline{(\mu\otimes\nu)^{(p)}\dots(\mu\otimes\nu)^{(p+1)}}$ **do** in parallel (OMP)
2: $\quad\widetilde{(\mu\nu|P^{(p)})} \leftarrow$ `TRMM`$[(\mu\nu|B^{(p)}), L_{BP}]$
3: **end for**
4: $\widetilde{(P^{(p)}|\mu\nu)} \leftarrow \widetilde{(\mu\nu|P^{(p)})}$
5: **for** $P = \overline{P^{(p)}/2\dots P^{(p+1)}/2}$ **do** in parallel (OMP)
6: $\quad$reshape $2\otimes N_{\mathrm{AO}}(N_{\mathrm{AO}}+1)/2$ of $\widetilde{(2P|\mu\nu)}$ into $N_{\mathrm{AO}}\otimes(N_{\mathrm{AO}}+1)$
7: **end for**

transformation (8) is justified if the transposition of $N_{\mathrm{AO}}(N_{\mathrm{AO}}+1)/2 \times N_{\mathrm{aux}}$ matrix is not the bottleneck and the number of SCF iterations is large enough.

The transformation and the transposition of three-center ERIs are described in Algorithm 3. The integrals were calculated and stored in rows of size N_{aux} in the previous stage (Fig. 1a), so now each row can be transformed according to Eq. (8). The corresponding iterations are shown in lines 1–3. They are parallelized with OpenMP. The matrix product is evaluated using the BLAS `TRMM` routine (line 2). Alternatively to parallelizing `for` loop, each `TRMM` operation can be parallelized as well. Then the whole RI tensor is transposed (line 4), so the column-majored RI tensor is obtained. To be stored using permutationally-adapted layout, finally, in lines 5–7 transformed integrals are reshaped into $N_{\mathrm{AO}}\times(N_{\mathrm{AO}}+1)$ matrices composed of a pair of triangles (Fig. 1b). If $N_{\mathrm{aux}}^{(p)}$ is odd, one of the last block triangles is simply filled with zeros.

Note that the algorithm can be greatly optimized by communication computation overlap: `for` loop in lines 1–3 can be divided into rounds, and at the end of each round transformed integrals can be put to the memory of other processes as well as got by given process. Thus, data transfer within the transposition (line 4) would also be divided into rounds being included in `for` loop. The optimal partition of this loop is such that one can perform a computation throughout the time of the message passing. Communication and computation would also be significantly reduced if three-center ERIs sparsity was taken into account. Finally, note that the overall message passing during the whole program execution might not be minimal if the transposition after the transformation (8) is performed, because it affects the amount of memory to be communicated within the SCF procedure. So, some checks need to be done in the future.

2.5. SCF Procedure

The SCF procedure is structured as always: calculation of Coulomb and exchange matrices (Eqs. (6) or (9) and (7) or (10)), construction of the Fock matrix (5) and its diagonalization to obtain molecular orbital expansion coefficients (3). Replicated data approaches as well as distributed ones are possible. For now, our implementations follow the replicated baseline for all objects except the RI tensor (1), which is evenly (or almost evenly) distributed between the computational nodes. The diagonalization of the Fock matrix is performed locally using OpenMP (although it can be implemented with MPI using libraries such as ELPA [32]). Note that iterative diagonalization methods such as Davidson or Lanczos [52] (and refs. therein) can be employed

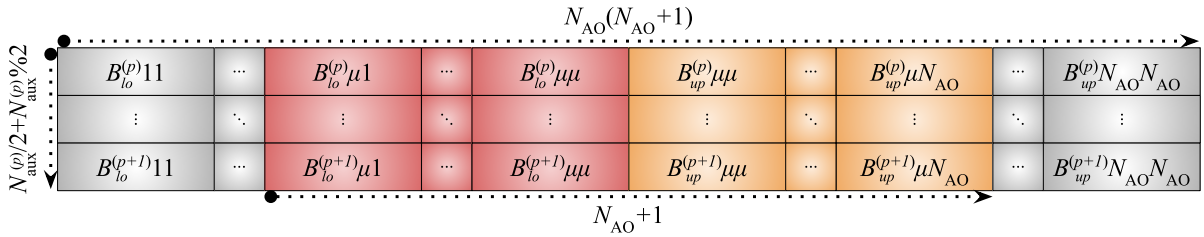


Figure 2. RI tensor representation for contractions within RI-J or RI-TJ (Alg. 4)

Algorithm 4 Coulomb matrix construction according to Eq. (6)

Input: permutationally-adapted three-center integrals $(B^{(p)}|\mu\nu)$ (1) (or $(\widetilde{B^{(p)}}|\mu\nu)$ (8)); density matrix $D_{\mu\nu}$ (4); inverse V_{BC}^{-1} of two-center integrals V_{BC} (2); size of the main basis N_{AO}; size of the auxiliary basis N_{aux}

Output: $J_{\mu\nu}$, $\nu \leq \mu$ (lower triangle)

1: **for** $\mu = \overline{1, N_{\text{AO}}}$ **do** in parallel (OMP)
2: $\quad V_{B_{\text{lo}}}^{(p)} \leftarrow \texttt{GEMV}[(B_{\text{lo}}^{(p)}|\mu\nu),\ D_{\mu\nu}],\ 0 \leq \nu \leq \mu$
3: $\quad V_{B_{\text{up}}}^{(p)} \leftarrow \texttt{GEMV}[(B_{\text{up}}^{(p)}|\mu\nu),\ D_{\mu\nu}],\ \mu \leq \nu \leq N_{\text{AO}}$
4: **end for**
5: $\widetilde{V}_C \leftarrow \texttt{GEMV}[\left(V_{BC}^{-1}\right)^{(p)},\ V_B^{(p)}]$
6: $\widetilde{V}_C^{(p)} \leftarrow \texttt{reduce}\ (\widetilde{V}_C)$ (MPI)
7: **for** $\mu = \overline{1, N_{\text{AO}}}$ **do** in parallel (OMP)
8: $\quad J_{\mu\nu}^{(p)} \leftarrow \texttt{GEMV}[(C_{\text{lo}}^{(p)}|\mu\nu),\ \widetilde{V}_{\text{lo}}^{(p)}],\ 0 \leq \nu \leq \mu$
9: $\quad J_{\nu\mu}^{(p)} \leftarrow \texttt{GEMV}[(C_{\text{up}}^{(p)}|\mu\nu),\ \widetilde{V}_{\text{up}}^{(p)}],\ \mu \leq \nu \leq N_{\text{AO}}$
10: **end for**
11: $J_{\mu\nu} \leftarrow \texttt{Allreduce}[J_{\mu\nu}^{(p)}]$ (MPI)

instead of full diagonalization. But let us focus on the calculation of the Coulomb and exchange matrices as the RI approximation is applied exactly here.

Algorithm 4 outlines the construction of the Coulomb matrix according to Eq. (6) (RI-J). The alternative algorithm (RI-TJ) can be obtained by removing lines 5 and 6, so it requires less communication. The layout used introduces some additional difficulties compared to the straightforward implementation of the algorithm. Figure 2 presents two subblocks of the RI tensor to be contracted within the current iteration of the `for` loop in lines 1–4. These computations are parallelized using OpenMP. Lines 2 and 3 specify the axis for contraction. Elements of the first subblock are placed in lower triangles, while those of the second one are placed in upper triangles (see Figures 1b, 2 and 3). This fact is reflected by the lower indices "lo" and "up" of the auxiliary dimension index $B^{(p)}$. The step results into the local part of the intermediate vector $V_B^{(p)}$ on each process p. For vectorization purposes, it is better to store elements of this subvector with "lo" and "up" indices separately (combining them further if needed). Next, $V_B^{(p)}$ is contracted with the local submatrix $(V_{BC}^{-1})^{(p)}$ obtaining the partial sum of another entire intermediate vector $\widetilde{V}_C$ (line 5). The local parts $\widetilde{V}_C^{(p)}$ of $\widetilde{V}_C$ are then accumulated using the MPI `reduce` routine according to the initial partition along the auxiliary basis dimension $B^{(p)}$ (line 6). Lines 7–10 again refer to the subblocks shown in Fig. 2. Now, columns of these subblocks are contracted with $\widetilde{V}_C^{(p)}$, forming a partial sum $J_{\mu\nu}^{(p)}$. The "lo" and "up" indices were again involved. Note that for vectorization purposes in line 9 the upper term of $J_{\mu\nu}^{(p)}$ can be used (instead of the current approach). `for` loop

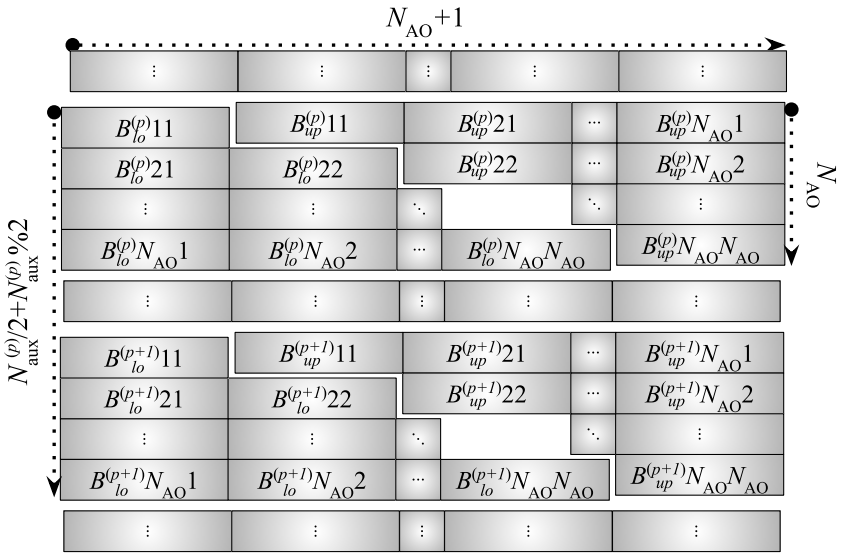


Figure 3. RI tensor representation for contractions within RI-K or RI-TK (Alg. 5)

Algorithm 5 Exchange matrix construction according to Eq. (7)

Input: permutationally-adapted three-center integrals $(B^{(p)}|\mu\nu)$ (1) (or $(\widetilde{P^{(p)}}|\mu\nu)$ (8)); MOs coefficients $C_{i\sigma}$ (3); Cholesky factor L_{BP} of V_{BC}^{-1} (see Eq. (2)); size of the main basis N_{AO}; size of the auxiliary basis N_{aux}

Output: $K_{\mu\nu}$, $\nu \leq \mu$ (lower triangle)

1: **for** $B = \overline{B^{(p)}/2 \ldots B^{(p+1)}/2}$ **do** in parallel (OMP)
2: $(2B|i\mu) \leftarrow$ `SYMM`$[C_{i\sigma}, (2B|\sigma\mu)]$
3: $(2B+1|i\mu) \leftarrow$ `SYMM`$[C_{i\sigma}, (2B+1|\sigma\mu)]$
4: **end for**
5: $(\mu\widetilde{i^{(p)}}|B) \leftarrow (B^{(p)}|i\mu)$
6: $(\mu\widetilde{i^{(p)}}|P) \leftarrow$ `TRMM`$[(\mu\widetilde{i^{(p)}}|B), L_{BP}]$
7: $K_{\mu\nu}^{(p)} \leftarrow$ `SYRK`$[(\mu\widetilde{i^{(p)}}|P)]$
8: $K_{\mu\nu} \leftarrow$ `Allreduce`$[K_{\mu\nu}^{(p)}]$ (MPI)

is parallelized with OpenMP again. Finally, all processes obtain the Coulomb matrix through the MPI `Allreduce` routine.

Algorithm 5 outlines the construction of the exchange matrix according to Eq. (7) (RI-K). The alternative algorithm (RI-TK) can be obtained just by removing lines 5 and 6, although the difference is also that the BLAS `SYRK` routine would be applied to the intermediate tensor part $(\widetilde{P^{(p)}}|i\mu)$ instead of $(\mu\widetilde{i^{(p)}}|P)$. Figure 3 shows a pair of triangles of the RI tensor to be contracted within the current iteration of the `for` loop in lines 1–4, parallelized using OpenMP. If the last triangle is zero-valued, it is more convenient to process the corresponding pair out of the `for` loop. After computations on lines 1–4, global transposition is done (line 5) in order to optimally perform contraction of the intermediate tensor with Cholesky factor L_{BP} of V_{BC}^{-1} (line 6). The partial sum of $K_{\mu\nu}$ is then obtained in line 7. Finally, all processes obtain an exchange matrix using the MPI `Allreduce` routine.

Algorithm 4 can be optimized by overlapping computation in line 5 with communication in line 6, although it is actual for a large number of processes. Algorithm 5 can also be modified in

this way by overlapping computation in lines 1–4 with communication in line 5. Both algorithms can be optimized using only lower triangles of symmetric matrices $J_{\mu\nu}$ and $K_{\mu\nu}$ for the final `Allreduce` operation. The latter can also overlap with the previous computations. All of these modifications are to be implemented in the future, as well as the optimization of Algorithm 3.

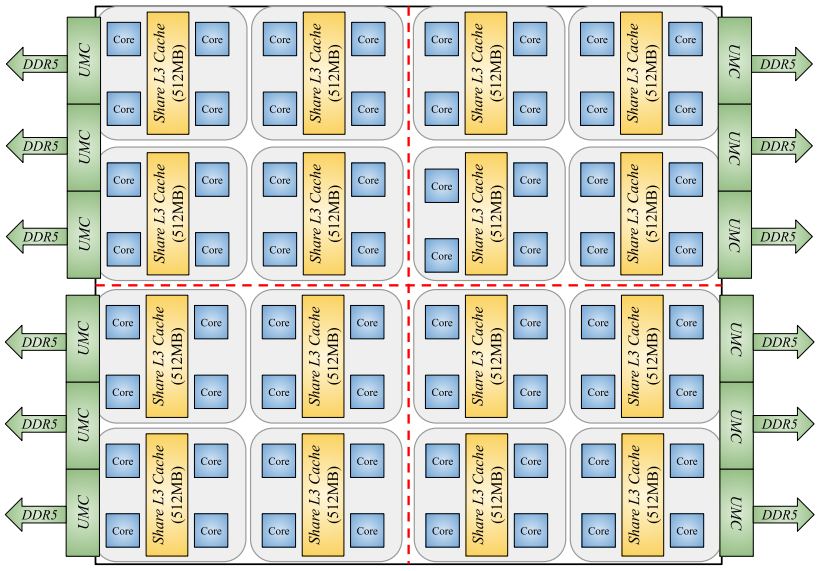
3. Computational Details

The MPI+OpenMP algorithms described in the previous section were implemented in the BUFO quantum chemistry simulation program. The newly developed code was benchmarked on 32× 6-core Intel Xeon E5-1650v3 @ 3.5GHz processors (Desmos at JIHT RAS [24, 40]) and 2× 64-core AMD EPYC 9535 processors (Irbis at MIPT). The first HPC server is a typical distributed memory system (see Fig. 4b and [46]), while the memory of the second workstation is shared between its processors (Fig. 4a and [2, 55]). So, two different strategies were adopted to run the hybrid MPI+OpenMP program using the `map-by` option of the OpenMPI library [13]. OpenMPI 5.0.0 and 4.1.6 were used for the Desmos and Irbis systems, respectively. MPI exchanges in case of the Desmos supercomputer were performed through Infiniband FDR, and in case of the Irbis server through shared memory. For Desmos each MPI process was bound to a processor with 6 cores sharing L3 cache. The number of OpenMP threads was varied only after all proccesors were occupied by MPI ranks. For the Irbis machine, each process was bound to the L3 cache shared by 4 cores of AMD EPYC 9535 processor. Different processor sets are possible for Irbis: NPS1, NPS2 and NPS4[7] (corresponding modes are reflected by red dotted lines in Fig. 4a; see also [55]). The simplest NPS1 was applied, so each processor is represented as a single NUMA node. At first, processes were bound to the caches of the first node. The remaining node was used only for tests with 32 MPI processes (and 1, 2 or 4 OpenMP threads). Alternatively, MPI processes can be gradually (with increase of their number) bound to L3 caches of different nodes again, enabling multithreading after 32 MPI processes would be distributed. Another policy is to reflect 2 processes to the NUMA nodes, increasing the number of threads up to 64 on each node. Although our code might not be optimal for such calculations due to the granularity used to calculate three-center ERIs (see Section 2.2) oriented to distributed computations rather than multithreaded on NUMA.

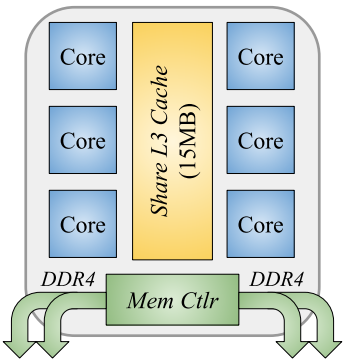
Other differences between the Desmos and Irbis benchmarks lie in the compilers and linear algebra libraries involved. In Desmos, the code was compiled with GCC 12.2.0 and linked with MKL 2025.0, while in Irbis, GCC 13.3.0 and OpenBLAS 3.30 were used. Both compilers support OpenMP 4.5, which is sufficient for all parallel constructs used in this work.

Currently, quantum chemistry calculations are widely used to simulate biomolecules and processes in living matter [29]. In our previous work [26], we already considered a chlorophyll molecule $C_{55}H_{72}O_5N_4Mg$ and its dimer in vacuum; the def2-SVP basis [50] supplemented by the def2-SVP-RIFIT auxiliary basis [19, 51] was used for orbital expansions (3). In this work, the dimer was additionally surrounded by 48 water molecules. Preliminary simulations of the equilibrium conformations of the chlorophyll dimer in a water solution (up to approximately 200 molecules) were performed using molecular dynamics (MD) with the potential [56]. The geometries were provided by Egor Igolnikov; then the number of H_2O molecules was reduced to 48 in order to fit in RAM of our computational facilities. Two main conformations of the chlorophyll dimer were observed in those simulations. The difference in their relative energy is determined

[7] NPS = NUMA (non-uniform memory access) node per socket.



(a) Principal scheme of the 64-core AMD EPYC 9535 processor (see also [2, 55])



(b) Principal scheme of the 6-core Intel® Xeon® E5-1650v3 @ 3.5GHz processor (see also [46])

Figure 4. Principal schemes workstation nodes used in benchmark calculations

mainly by the electrostatic attraction between the porphyrin rings and the interaction of the molecule tails with water. The chlorophyll molecule has a hydrophobic tail and a hydrophilic ring, so it is amphiphilic in general. In water, chlorophylls aggregate in pairs. On the one hand, π-stacking interactions of conjugated systems lead to their rendezvous; on the other hand, it is favorable to place the hydrophobic tails between the porphyrin rings to minimize their contact with water. Although the HF method does not account for electronic correlation, it might be the basis for more advanced calculations. So our tests were performed using one of the equilibrium conformations. The sizes used were $N_{\text{atoms}} = 322$, $N_{\text{occ}} = 722$, $N_{\text{AO}} = 3700$ (grouped into $N_{\text{AO}}^{\text{sh}} = 1792$) and $N_{\text{aux}} = 11896$ (grouped into $N_{\text{aux}}^{\text{sh}} = 4288$).

4. Results and Discussion

Performance tests required about 800 GB of RAM over all processes during the program execution. Our benchmarks on Desmos involved 32 MPI processes with 1 to 6 OpenMP threads. Less processes were not used because of memory limits. In contrast, there were no problems with the available memory workspace on the Irbis machine, so subsequently bindings of MPI processes from 1 to 32 were performed with relevant turn on of OpenMP multithreading.

Records of the type "$\#N_1$" or "$\#N_1$–N_2" in the table column name refer to line or lines of the corresponding algorithm described above in Section 2. "wall" means wall execution time. "Sp. (x)" stands for the speedup of calculation with the increase of the number N of parallel threads involved. The speedup is always evaluated from the wall time.

4.1. Scalability: Calculation of ERIs

Figure 5 and Table 1 show the strong scaling behavior of the integration algorithms within the RI-JK and RI-TJK approaches. Speedups in Tab. 1 are defined by the slowest process. Multithreading on Desmos performs well enough, giving up to 5.0× and 5.6× speedup for RI-JK

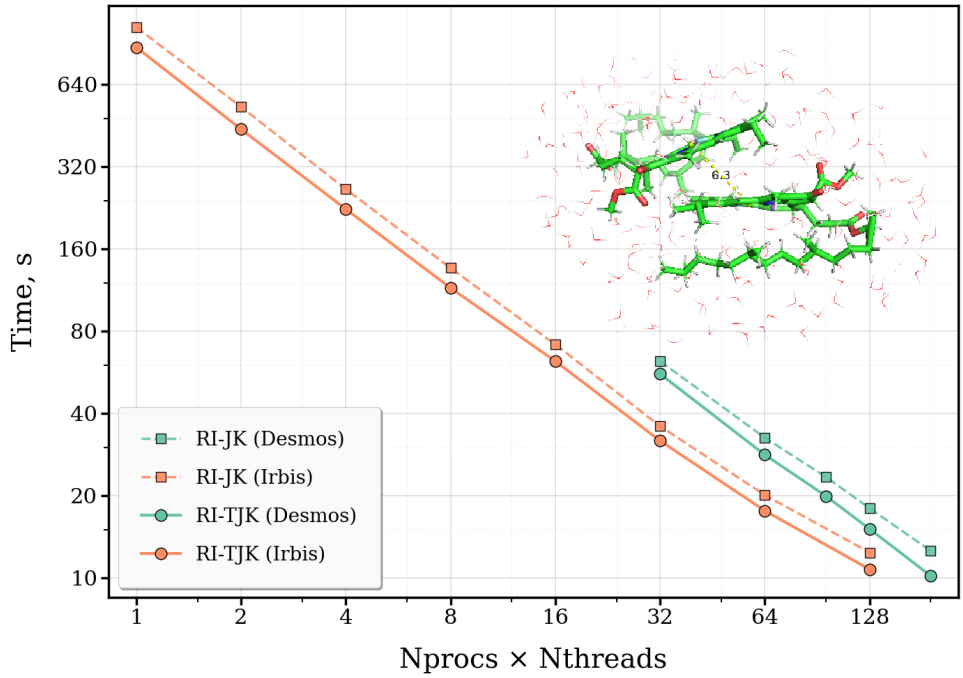


Figure 5. Scalability of the three-center ERIs calculation

Table 1. Breakdown of scalability of the three-center ERIs calculation

N	RI-JK				RI-TJK			
	Desmos		Irbis		Desmos		Irbis	
	Time, s	Sp. (×)	Time, s	Sp. (×)	Time, s	Sp. (×)	Time, s	Sp. (×)
1	–	–	1037.9	1.0×	–	–	877.1	1.0×
2	–	–	530.9	2.0×	–	–	439.7	2.0×
4	–	–	265.3	3.9×	–	–	224.2	3.9×
8	–	–	136.4	7.6×	–	–	115.4	7.6×
16	–	–	71.7	14.5×	–	–	62.2	14.1×
32	62.2	1.0×	36.0	28.8×	55.9	1.0×	31.8	28.0×
64	32.7	1.9×	20.1	51.5×	28.3	2.0×	17.6	49.7×
96	23.4	2.7×	–	–	20.0	2.8×	–	–
128	18.0	3.5×	12.4	83.6×	15.1	3.7×	10.8	81.4×
192	12.6	5.0×	–	–	10.2	5.6×	–	–

and RI-TJK, respectively, when going from 32 to 192 threads. It is not ideal (6.0×) because of the uneven initial static distribution at the stage of scheduling the calculation of three-centers of ERIs (see Alg. 1, line 1). The observed speedup of the RI-TJK implementation is slightly better. The probable reason for this is that $N^{\text{sh}}_{\text{aux}} \sim 2.5 N^{\text{sh}}_{\text{AO}}$, i.e., more integration tasks were distributed between processes. Though this consideration does not primarily apply to the analogous comparison of runnings on Irbis: multithreaded integration (i.e., when more than 32 threads are involved) scales similarly for both algorithms. Interestingly, the speedup of calculations with multithreading on Irbis is worse than the speedup up to 32 processes, when all L3 caches are bound to MPI ranks. Actually, the scalability is good up to 32 processes involved, resulting in 28.8× or 28.0× speedup. Better scalability can be achieved by means of dynamic scheduling, because

there is about 2.5 seconds between the fastest and the slowest processes. Nevertheless, scalability up to 32 processes is greater than that after turning on multithreading. This observation is true for other benchmarks as well (see the following subsections). The exact reason for that is unclear. Although in case of integration the probable reason is a bad decision to parallelize the nested `for` loop (see Alg. 2) with OpenMP instead of the external one.

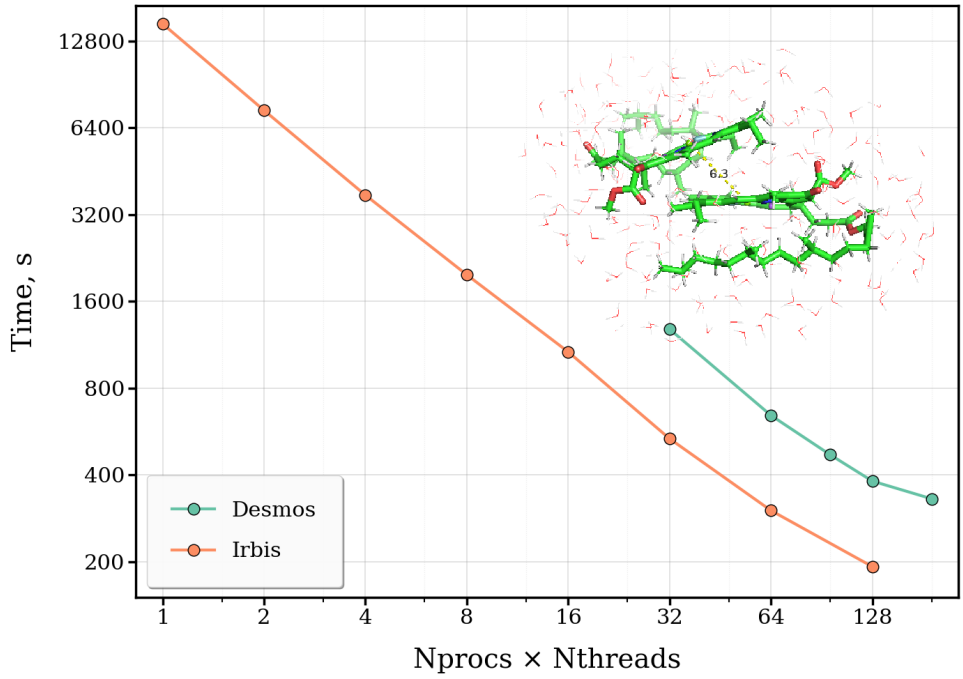


Figure 6. Scalability of the three-center ERIs transformation

Table 2. Breakdown of scalability of the three-center ERIs transformation

N	Desmos Time, s #1–3	Desmos Time, s #4–7	Desmos Time, s wall	Desmos Sp. (×)	Irbis Time, s #1–3	Irbis Time, s #4–7	Irbis Time, s wall	Irbis Sp. (×)
1	–	–	–	–	14435.7	267.4	14703.2	1.0×
2	–	–	–	–	7216.5	148.6	7365.2	2.0×
4	–	–	–	–	3654.4	86.2	3740.6	3.9×
8	–	–	–	–	1926.8	51.9	1978.7	7.4×
16	–	–	–	–	1029.4	39.2	1068.6	13.8×
32	1255.6	25.9	1281.5	1.0×	508.9	25.1	534.1	27.5×
64	624.7	18.4	643.1	2.0×	280.5	20.2	300.7	48.9×
96	451.9	17.5	469.4	2.7×	–	–	–	–
128	366.8	13.6	380.4	3.4×	171.8	20.2	192.1	76.6×
192	318.0	12.3	330.3	3.9×	–	–	–	–

When comparing the Desmos and Irbis performance, it can be seen that the computation time on 32 processes differs by approximately a factor of two. This can be attributed to the twofold advantage of the Irbis CPU cores in terms of FLOP/s due to the AVX-512 instruction set compared to AVX2 available for the Desmos' CPUs.

4.2. Scalability: Transformation of ERIs

Figure 6 and Table 2 show the strong scaling behavior of the three-center ERI transformation (8) and their subsequent transposition for the RI-TJK approach. The `TRMM` operations speedup is great, nearly ideal up to 32 processes, while the scaling of the transposition step is

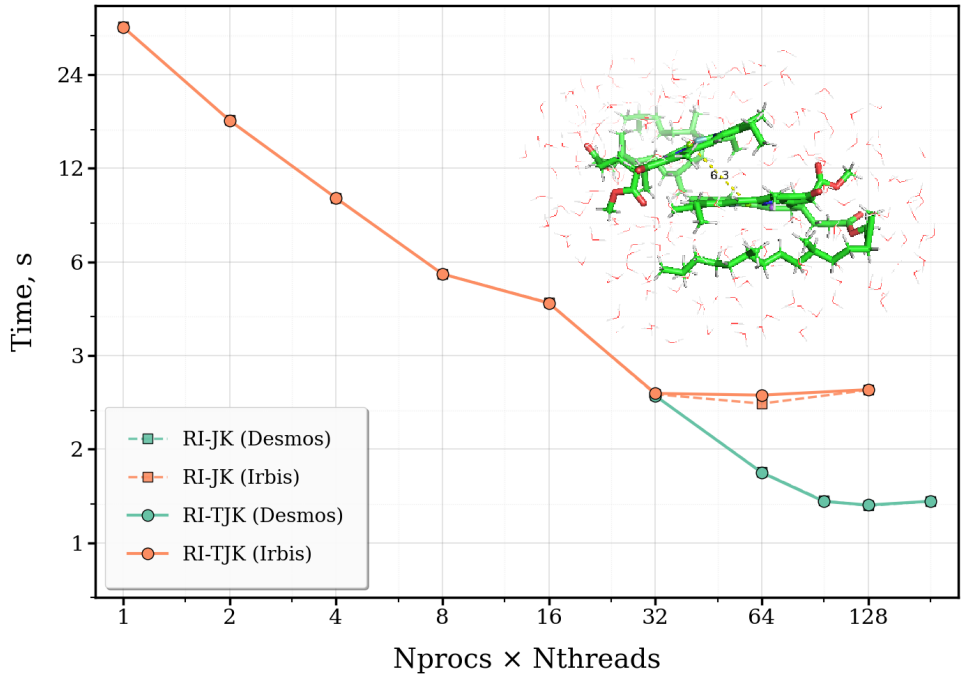


Figure 7. Scalability of the RI-J/TJ algorithm

Table 3. Breakdown of scalability of the RI-J/TJ algorithm

N	RI-J/TJ (Desmos) Time, s #1–4	#7–10	wall	Sp. (×)	RI-J/TJ (Irbis) Time, s #1–4	#7–10	wall	Sp. (×)
1	–	–	–	–	17.21	16.96	34.18	1.0×
2	–	–	–	–	8.61	8.50	17.12	2.0×
4	–	–	–	–	4.86	4.77	9.64	3.6×
8	–	–	–	–	2.71	2.78	5.49	6.2×
16	–	–	–	–	2.18	2.25	4.44	7.7×
32	1.05	1.18	2.23	1.0×	1.10	1.16	2.25	15.2×
64	0.59	0.68	1.27	1.8×	1.00	1.10	2.10	16.3×
96	0.47	0.55	1.02	2.1×	–	–	–	–
128	0.45	0.53	0.99	2.2×	1.08	1.25	2.32	14.7×
192	0.45	0.57	1.02	2.1×	–	–	–	–

not so good. The former operations are not ideally scaled because the rebalancing of the RI tensor after the parallel evaluation of ERIs was not implemented for now. So after the RI tensor is computed, it is still unevenly distributed between MPI processes. The transposition step requires much less time, but can become a bottleneck, when more and more processes are involved. For the computation on 128 threads of Irbis it already takes about 10% of the wall time. This obsta-

cle significantly limits scalability, though the speedup of multithreaded calculations is generally worse again even in spite of the transposition step. The same conclusions about multithreading are also true for Desmos. The reasons for this are currently unclear.

It can be noted that the time needed for the transformation step on 32 processes of Desmos and Irbis again differs by approximately a factor of two. Actually, the transposition step in both tests is done in the same time. So, Irbis offers twice the computational performance of Desmos, but its memory bandwidth is not proportionally higher. As a result, it reaches the memory wall sooner.

4.3. Scalability: the SCF Procedure

Figure 7 and Table 3 show the strong scaling behavior of the RI-J and RI-TJ algorithms (lines #5 and #6 are out of interest here, because corresponding transformations were significantly faster than for other steps). Neither algorithm exhibits satisfactory scalability across the full range of the number of threads. The possible reason for performance degradation is the large intermediate buffer used for the reduction of $J^{(p)}_{\mu\nu}$ using OpenMP. Being allocated on the stack in current implementations, this buffer induces significant cache pressure, resulting in frequent cache misses and limited data reuse. Even the twofold superiority typical for Irbis on 32 processes is not observed. This situation is to be resolved in the future, though RI-J and RI-TJ still take much less time than RI-K and RI-TK.

Figure 8 and Tables 4 and 5 show the strong scaling behavior of the RI-K and RI-TK algorithms. Now, the typical picture of nearly ideal speedup up to 27.1× and 28.0× on 32 processes of Irbis is observed, as well as more moderate scalability after that. RI-K scalability is worse, because there is no three-center ERIs rebalancing in current RI-K implemetation and the transposition is memory-bound. The transformation within RI-K takes about 5% of the execution time. Benchmarks performed on Desmos show that multithreading can be efficiently incorpo-

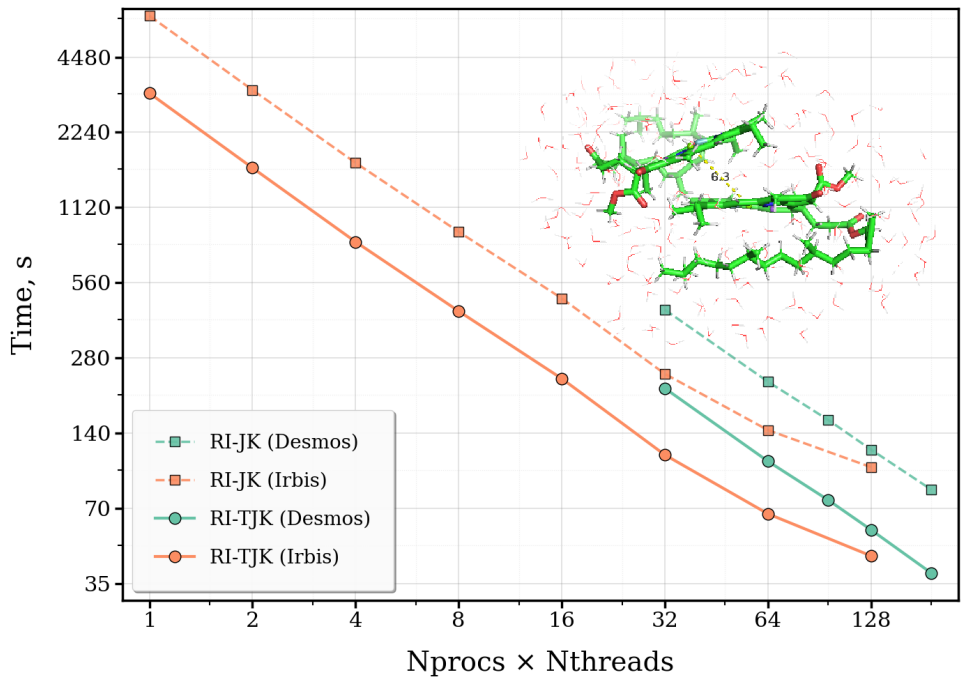


Figure 8. Scalability of the RI-K/TK algorithm

Table 4. Breakdown of scalability of the RI-K algorithm

N	RI-K (Desmos) Time, s					Sp.	RI-K (Irbis) Time, s					Sp.
	#1–4	#5	#6	#7	wall	(×)	#1–4	#5	#6	#7	wall	(×)
1	–	–	–	–	–	–	2138.4	64.3	3296.3	1053.7	6552.7	1.0×
2	–	–	–	–	–	–	1070.7	38.2	1666.7	532.7	3308.2	2.0×
4	–	–	–	–	–	–	543.2	22.6	853.4	272.8	1692.0	3.9×
8	–	–	–	–	–	–	287.1	13.8	452.6	142.7	896.2	7.3×
16	–	–	–	–	–	–	154.2	10.6	242.8	78.1	485.7	13.5×
32	141.8	6.7	220.2	68.4	437.1	1.0×	76.4	6.6	120.5	38.7	242.2	27.1×
64	72.3	5.5	111.4	36.2	225.4	1.9×	44.3	6.2	70.7	23.1	144.2	45.4×
96	50.4	4.9	77.8	25.3	158.4	2.8×	–	–	–	–	–	–
128	38.2	4.6	58.6	19.1	120.6	3.6×	29.8	6.1	50.4	16.4	102.6	63.9×
192	25.5	4.6	40.3	13.0	83.4	5.2×	–	–	–	–	–	–

Table 5. Breakdown of scalability of the RI-TK algorithm

N	RI-TK (Desmos) Time, s			Sp.	RI-TK (Irbis) Time, s			Sp.
	#1–4	#7	wall	(×)	#1–4	#7	wall	(×)
1	–	–	–	–	2138.7	1074.3	3213.0	1.0×
2	–	–	–	–	1070.2	544.2	1614.4	2.0×
4	–	–	–	–	542.9	274.2	817.1	3.9×
8	–	–	–	–	287.0	144.5	431.5	7.5×
16	–	–	–	–	154.1	77.8	231.9	13.9×
32	141.7	70.3	212.1	1.0×	76.3	38.5	114.8	28.0×
64	72.2	36.1	108.3	2.0×	44.1	22.6	66.7	48.2×
96	50.4	25.4	75.7	2.8×	–	–	–	–
128	38.2	19.4	57.6	3.7×	29.8	15.5	45.3	70.9×
192	25.5	13.2	38.7	5.5×	–	–	–	–

rated: up to 5.2× and 5.5× for RI-K and RI-TK, respectively. The reason for the bad results of calculations employing multithreading on Irbis is again unclear. It can be noted that the overall time including all stages (except for the transposition) on 32 processes of Desmos and Irbis differs by approximately a factor of two as before. Also, the transposition step in the RI-K tests is done in the same time on Desmos and Irbis at this number of processes. It is consistent with the coincidence of transposition times observed in the previous section after the transformation of three-center ERIs (8).

Conclusion

We report a new hybrid MPI+OpenMP implementation of the resolution-of-the-identity restricted Hartree—Fock method extensively employing permutational symmetry of the ERI tensor to minimize explicit local memory movement during iterations of SCF procedure. Both versions of the proposed parallel algorithm (RI-JK and RI-TJK) were benchmarked on a system of two

chlorophyll molecules surrounded by 48 water molecules (3700 basis functions supplemented with 11896 auxiliary basis functions) to reveal their performance and scalability features; two different machines with different computer architectures were used throughout to perform these benchmarks.

The RI-JK algorithm allows us to avoid any global transpositions of three-center ERIs at the cost of performing more moderate transposition at each SCF iteration, whereas the RI-TJK algorithm performs one global transposition of these integrals only once before the start of the SCF procedure (see Tab. 2 and Tab. 4 for comparison). Fortunately, in both cases the communication computation overlap can greatly improve the overall efficiency of the algorithms when there would be too many communications. It should be true for the larger number of processes involved, because even for the case of 128 threads communication already takes about 5–10% of wall time. The contractions that can be actually overlapped also differ, because avoiding the pre-transformation (8) of formal complexity $O(N_{\mathrm{AO}}^2 N_{\mathrm{aux}}^2)$ before the SCF procedure leads to including $O(N_{\mathrm{AO}} N_{\mathrm{aux}}^2)$ arithmetic operations at each SCF iteration. Generally, the RI-JK algorithm is recommended when a rather small number of SCF iterations is expected, while RI-TJK is more beneficial for SCF calculations expected to converge slowly. For both scenarios, it was shown that the permutationally-adapted memory layout for storing three-center ERIs can be efficiently used, though some technical problems should be resolved especially in case of RI-J and RI-TJ algorithms. However, the permutationally-adapted memory layout removes the need to perform redundant copy operations in RI-K and RI-TK at each SCF iteration compared to conventional parallel RI-SCF implementations unpacking three-center ERIs each iteration. Another issue to be studied in detail in the future is how to deal with RI tensor sparsity using the permutationally-adapted layout. Performance evaluation shows that the proposed approaches to parallelization of the RI-SCF method are reasonably scalable on modern computational architectures (up to 70–80× speedup on 128 threads), though some problems should be identified and addressed in the future in order to obtain the scalability closer to the ideal one.

Acknowledgements

The authors thank Egor Igolnikov for providing configurations of the chlorophyll dimer in water from MD simulations.

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (State Assignment No. 075-00270-26-00).

References

1. Alexeev, Y., Kendall, R.A., Gordon, M.: The distributed data SCF. Computer Physics Communications 143(1), 69–82 (2002). https://doi.org/10.1016/S0010-4655(01)00439-8

2. AMD: AMD EPYC$^{\mathrm{TM}}$ 9005 Proceessor Architecture Overview. https://docs.amd.com/v/u/en-US/58462_amd-epyc-9005-tg-architecture-overview (2025), accessed: 2026-02-15

3. Blackford, L.S., Demmel, J., Dongarra, J., *et al.*: An updated set of basic linear algebra subprograms (BLAS). ACM Transactions on Mathematical Software 28(2), 135–151 (2002). https://doi.org/10.1145/567806.567807

4. Bussy, A., Schütt, O., Hutter, J.: Sparse tensor based nuclear gradients for periodic Hartree-Fock and low-scaling correlated wave function methods in the CP2K software package:

A massively parallel and GPU accelerated implementation. Journal of Chemical Physics 158(16), 164109 (2023). https://doi.org/10.1063/5.0144493

5. Bussy, A., Hutter, J.: Efficient periodic resolution-of-the-identity Hartree-Fock exchange method with k-point sampling and Gaussian basis sets. Journal of Chemical Physics 160(6), 064116 (2024). https://doi.org/10.1063/5.0189659

6. Calaminici, P., Domínguez-Soria, V.D., Geudtner, G., *et al.*: Parallelization of three-center electron repulsion integrals. Theoretical Chemistry Accounts 115(4), 221–226 (2005). https://doi.org/10.1007/s00214-005-0005-0

7. Dyczmons, V.: No N^4-dependence in the calculation of large molecules. Theoretical Chemistry Accounts 28(3), 307–310 (1973). https://doi.org/10.1007/BF00533492

8. Echenique, P., Alonso, J.L.: A mathematical and computational review of Hartree–Fock SCF methods in quantum chemistry. Molecular Physics 105(23–24), 3057–3098 (2007). https://doi.org/10.1080/00268970701757875

9. Eichkorn, K., Treutler, O., Öhm, H., *et al.*: Auxiliary basis sets to approximate Coulomb potentials. Chemical Physics Letters 242(4), 652–660 (1995). https://doi.org/10.1016/0009-2614(95)00838-u

10. Eichkorn, K., Weigend, F., Treutler, O., Ahlrichs, R.: Auxiliary basis sets for main row atoms and transition metals and their use to approximate Coulomb potentials. Theoretical Chemistry Accounts 97(1–4), 119–124 (1997). https://doi.org/10.1007/s002140050244

11. Foster, I., Tilson, J.L., Wagner, A., *et al.*: Toward high-performance computational chemistry: I. Scalable Fock matrix construction algorithms. Journal of Computational Chemistry 17(1), 109–123 (1996). https://doi.org/10.1002/(SICI)1096-987X(19960115)17:1<109::AID-JCC9>3.0.CO;2-V

12. Früchtl, H.A., Kendall, R.A., Harrison, R.J., Dyall, K.G.: An implementation of RI-SCF on parallel computers. International Journal of Quantum Chemistry 64(1), 63–69 (1997). https://doi.org/10.1002/(SICI)1097-461X(1997)64:1<63::AID-QUA7>3.0.CO;2-%23

13. Gabriel, E., Fagg, G.E., Bosilca, G., *et al.*: Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Lecture Notes in Computer Science, vol. 3241, pp. 97–104. Springer Berlin Heidelberg (2004). https://doi.org/10.1007/978-3-540-30218-6_19

14. Gill, P.M., Johnson, B.G., Pople, J.A.: A simple yet powerful upper bound for Coulomb integrals. Chemical Physics Letters 217(1), 65–68 (1994). https://doi.org/10.1016/0009-2614(93)E1340-M

15. Glebov, I.O., Poddubnyi, V.V.: An effective algorithm of the Hartree–Fock approach with the storing of two-electron integrals in the resolution of identity approximation. Russian Journal of Physical Chemistry A 98(4), 617–625 (2024). https://doi.org/10.1134/S0036024424040101

16. Guidon, M., Hutter, J., VandeVondele, J.: Auxiliary density matrix methods for Hartree-Fock exchange calculations. Journal of Chemical Theory and Computation 6(8), 2348–2364 (2010). https://doi.org/10.1021/ct1002225

17. Harrison, R.J., Guest, M.F., Kendall, R.A., *et al.*: Toward high-performance computational chemistry: II. A scalable self-consistent field program. Journal of Computational Chemistry 17(1), 124–132 (1996). https://doi.org/10.1002/(SICI)1096-987X(19960115)17:1<124::AID-JCC10>3.0.CO;2-N

18. Häser, M., Ahlrichs, R.: Improvements on the direct SCF method. Journal of Computational Chemistry 10(1), 104–111 (1989). https://doi.org/10.1002/jcc.540100111

19. Hellweg, A., Hättig, C., Höfener, S., Klopper, W.: Optimized accurate auxiliary basis sets for RI-MP2 and RI-CC2 calculations for the atoms Rb to Rn. Theoretical Chemistry Accounts 117, 587–597 (2007). https://doi.org/10.1007/s00214-007-0250-5

20. Hollman, D.S., Schaefer, H.F., Valeev, E.F.: Fast construction of the exchange operator in an atom-centred basis with concentric atomic density fitting. Molecular Physics 115(17–18), 2065–2076 (2017). https://doi.org/10.1080/00268976.2017.1346312

21. Hollman, D.S., Schaefer, H.F., Valeev, E.F.: A tight distance-dependent estimator for screening three-center Coulomb integrals over Gaussian basis functions. Journal of Chemical Physics 142(15), 154106 (2015). https://doi.org/10.1063/1.4917519

22. Huang, H., Sherill, C.D., Chow, E.: Techniques for high-performance construction of Fock matrices. Journal of Chemical Physics 152(2), 024122 (2020). https://doi.org/10.1063/1.5129452

23. Ishimura, K., Kuramoto, K., Ikuta, Y., Hyodo, S.A.: MPI/OpenMP Hybrid Parallel Algorithm for Hartree-Fock Calculations. Journal of Chemical Theory and Computation 6(4), 1075–1080 (2010). https://doi.org/10.1021/ct100083w

24. Ismagilov, T., Mukosey, A., Smirnov, F., *et al.*: Towards performance analysis of GPU-aware MPI over Angara interconnect. International Journal of High Performance Computing Applications 40(2), 240–253 (2026). https://doi.org/10.1177/10943420251411961

25. Kashpurovich, I.V., Oleynichenko, A.V., Stegailov, V.V.: Achieving the maximum performance of the resolution of the identity approximation in the Hartree-Fock method. In: Sokolinsky, L., Zymbler, M. (eds.) Parallel Computational Technologies. Communications in Computer and Information Science, vol. 2891, pp. 239–263. Springer (2026)

26. Kashpurovich, I.V., Oleynichenko, A.V., Stegailov, V.V.: Development of strategies for parallel implementation of the Hartree-Fock theory in resolution-of-the-identity approximation. Russian Journal of Physical Chemistry A 100(5), 1013–1036 (2026)

27. Kashpurovich, I.V., Oleynichenko, A.V., Stegailov, V.V.: NUMA-aware OpenMP algorithm for three-center electron repulsion integrals. In: Voevodin, V., Antonov, A., Nikitenko, D. (eds.) Supercomputing. Lecture Notes in Computer Science, vol. 16196, pp. 333–350. Springer, Cham (2026). https://doi.org/10.1007/978-3-032-13127-0

28. Laikov, D.N.: Fast evaluation of density functional exchange-correlation terms using the expansion of the electron density in auxiliary basis sets. Chemical Physics Letters 281(1–3), 151–156 (1997). https://doi.org/10.1016/s0009-2614(97)01206-2

29. Lankin, A.V., Norman, G.E.: Introduction to quantum mechanics of living matter. Russian Journal of Physical Chemistry A 99(6), 1416–1445 (2025). https://doi.org/10.1134/s0036024425700785

30. Le, H.A., Shiozaki, T.: Occupied-orbital fast multipole method for efficient exact exchange evaluation. Journal of Chemical Theory and Computation 14(3), 1228–1234 (2018). https://doi.org/10.1021/acs.jctc.7b00880

31. Manzer, S., Horn, P.R., Mardirossian, N., Head-Gordon, M.: Fast, accurate evaluation of exact exchange: the occ-RI-K algorithm. Journal of Chemical Physics 143(2), 024113 (2015). https://doi.org/10.1063/1.4923369

32. Marek, A., Blum, V., Johanni, R., *et al.*: The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science. Journal of Physics: Condensed Matter 26(21), 213201 (2014). https://doi.org/10.1088/0953-8984/26/21/213201

33. Mejía-Rodríguez, D., Köster, A.M.: Robust and efficient variational fitting of Fock exchange. Journal of Chemical Physics 141(12), 124114 (2014). https://doi.org/10.1063/1.4896199

34. Merlot, P., Kjærgaard, T., Helgaker, T., *et al.*: Attractive electron–electron interactions within robust local fitting approximations. Journal of Computational Chemistry 34(17), 1486–1496 (2013). https://doi.org/10.1002/jcc.23284

35. Mironov, V.A., Alexeev, Y., Keipert, K., *et al.*: An efficient MPI/OpenMP parallelization of the Hartree-Fock method for the second generation of Intel® Xeon Phi™ processor. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017. vol. 31, pp. 1–12. ACM (2017). https://doi.org/10.1145/3126908.3126956

36. Neese, F.: An improvement of the resolution of the identity approximation for the formation of the Coulomb matrix. Journal of Computational Chemistry 24(14), 1740–1747 (2003). https://doi.org/10.1002/jcc.10318

37. Oleynichenko, A.V., Zaitsevskii, A., Mosyagin, N.S., *et al.*: LIBGRPP: A library for the evaluation of molecular integrals of the generalized relativistic pseudopotential operator over Gaussian functions. Symmetry 15(1), 197 (2022). https://doi.org/10.3390/sym15010197

38. Reine, S., Tellgren, E., Krapp, A., *et al.*: Variational and robust density fitting of four-center two-electron integrals in local metrics. Journal of Chemical Physics 129(10), 104101 (2008). https://doi.org/10.1063/1.2956507

39. Shiozaki, T.: BAGEL : Brilliantly Advanced General Electronic-structure Library. Wiley Interdisciplinary Reviews: Computational Molecular Science 8(1), e1331 (2017). https://doi.org/10.1002/wcms.1331

40. Stegailov, V., Dlinnova, E., Ismagilov, T., *et al.*: Angara interconnect makes GPU-based Desmos supercomputer an efficient tool for molecular dynamics calculations. International Journal of High Performance Computing Applications 33(3), 507–521 (2019). https://doi.org/10.1177/1094342019826667

41. Stegailov, V., Smirnov, G., Vecher, V.: VASP hits the memory wall: Processors efficiency comparison. Concurrency and Computation: Practice and Experience 31(19), e5136 (2019). https://doi.org/10.1002/cpe.5136

42. Stocks, R., Palethorpe, E., Barca, J.: Multi-GPU RI-HF energies and analytic gradients – toward high-throughput ab initio molecular dynamics. Journal of Chemical Theory and Computation 20(17), 7503–7515 (2024). https://doi.org/10.1021/acs.jctc.4c00877

43. Sun, Q.: Libcint: An efficient general integral library for Gaussian basis functions. Journal of Computational Chemistry 36(22), 1664–1671 (2015). https://doi.org/10.1002/jcc.23981

44. Sun, Q.: Efficient Hartree-Fock exchange algorithm with Coulomb range separation and long-range density fitting. Journal of Chemical Physics 159(22), 224101 (2023). https://doi.org/10.1063/5.0178266

45. Sun, Q., Berkelbach, T.C., Blunt, N.S., *et al.*: PySCF: the Python-based simulations of chemistry framework. Wiley Interdisciplinary Reviews: Computational Molecular Science 8(1), e1340 (2017). https://doi.org/10.1002/wcms.134

46. Tugov, A.: Dedicated servers based on Intel Xeon E5v3 processors. https://selectel.ru/blog/vydelennye-servery-na-baze-processorov-intel-xeon-e5v3/ (2014), accessed: 2026-02-15

47. Vahtras, O., Almlöf, J., Feyereisen, M.: Integral approximations for LCAO-SCF calculations. Chemical Physics Letters 213(5–6), 514–518 (1993). https://doi.org/10.1016/0009-2614(93)89151-7

48. Valeev, E.F., Shiozaki, T.: Comment on "A tight distance-dependent estimator for screening three-center Coulomb integrals over Gaussian basis functions" [Journal of Chemical Physics 142, 154106 (2015)]. Journal of Chemical Physics 153(9), 097101 (2020). https://doi.org/10.1063/5.0020567

49. Weigend, F.: A fully direct RI-HF algorithm: Implementation, optimised auxiliary basis sets, demonstration of accuracy and efficiency. Physical Chemistry Chemical Physics 4(18), 4285–4291 (2002). https://doi.org/10.1039/b204199p

50. Weigend, F., Ahlrichs, R.: Balanced basis sets of split valence, triple zeta valence and quadruple zeta valence quality for H to Rn: Design and assessment of accuracy. Physical Chemistry Chemical Physics 7(18), 3297–3305 (2005). https://doi.org/10.1039/B508541A

51. Weigend, F., Häser, M., Patzelt, H., Ahlrichs, R.: RI-MP2: optimized auxiliary basis sets and demonstration of efficiency. Chemical Physics Letters 294(1–3), 143–152 (1998). https://doi.org/10.1016/s0009-2614(98)00862-8

52. Windom, Z.W., Bartlett, R.J.: On the iterative diagonalization of matrices in quantum chemistry: reconciling preconditioner design with Brillouin-Wigner perturbation theory. Journal of Chemical Physics 158(13), 134107 (2023). https://doi.org/10.1063/5.0139295

53. Wu, X., Sun, Q., Pu, Z., *et al.*: Enhancing GPU-Acceleration in the Python-Based Simulations of Chemistry Frameworks. Wiley Interdisciplinary Reviews: Computational Molecular Science 15(2), e70008 (2025). https://doi.org/10.1002/wcms.70008

54. Xing, L., Patel, A., Chow, E.: A new scalable parallel algorithm for Fock matrix construction. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium. pp. 902–914 (2014). https://doi.org/10.1109/IPDPS.2014.97

55. Xu, P., Yang, K.: Balanced Memory Configurations with 5th Generation AMD EPYC Processors. https://lenovopress.lenovo.com/lp2283-balanced-memory-configurations-with-5th-generation-amd-epyc-processors (2025), accessed: 2026-02-15

56. Zhang, L., Silva, D.A., Yan, Y., Huang, X.: Force field development for cofactors in the photosystem II. Journal of Computational Chemistry 33(25), 1969–1980 (2012). https://doi.org/10.1002/jcc.23016