



GPU-Accelerated Analytic Coulomb- and Exchange Gradients for Hartree Fock and Density Functional Theory

Life Sciences

Published by ChemRxiv

2025

terraquantum.swiss

23 March 2026

GPU-Accelerated Analytic Coulomb- and Exchange Gradients for Hartree Fock and Density Functional Theory

Vladimir Mironov¹, Alexander Paul², Dmitry Morozov^{2,3}, Ruslan Mallaev, Michael Perelshtein², Roman Ellerbrock¹

1. Terra Quantum (Switzerland)

2. Terra Quantum (Switzerland)

3. University of Jyväskylä

Abstract

We present a GPU-accelerated software package for the evaluation of analytic two-electron energy and gradient contributions in Hartree-Fock (HF) and Density Functional theory (DFT) calculations. The implementation is provided as a Python library with a C++ backend, enabling straightforward integration into modern computational chemistry and drug-discovery workflows. The code supports single-point energy and nuclear gradient evaluations on both single- and multi-GPU systems, and employs MPI-based parallelization with dynamic load balancing in multi-node environments. We report comprehensive benchmarks demonstrating favorable scaling with respect to system size, as well as high throughput for batched evaluations relevant to molecular dynamics, geometry optimization, and large-scale virtual screening. Parallel execution of a single system was carried out on up to 24 A100 GPUs. The implementation builds on optimized GPU-enabled variants of the LibintX and GauXC libraries to efficiently compute density-fitted Coulomb, semi-numerical exchange, and exchange–correlation contributions.

Keywords

Density functional theory, Density-fitting, semi-numerical exchange, CUDA, Python

GPU-Accelerated Analytic Coulomb- and Exchange Gradients for Hartree Fock and Density Functional Theory

Vladimir Mironov,^{*,†} Dmitry Morozov,^{†,‡} Alexander C. Paul,[†] Ruslan A. Mallaev,[†] Michael Perelshtein,[†] and Roman Ellerbrock^{*,†}

[†]*Terra Quantum AG, Kornhausstrasse 25, 9000 St. Gallen, Switzerland*

[‡]*Nanoscience Center and Department of Chemistry, University of Jyväskylä, P.O. Box 35, 40014 Jyväskylä, Finland*

E-mail: vmi@terraquantum.swiss; re@terraquantum.swiss

Abstract

We present a GPU-accelerated software package for the evaluation of analytic two-electron energy and gradient contributions in Hartree-Fock (HF) and density functional theory (DFT) calculations. The implementation is provided as a Python library with a C++ backend, enabling straightforward integration into modern computational chemistry and drug-discovery workflows. The code supports single-point energy and nuclear gradient evaluations on both single- and multi-GPU systems, and employs MPI-based parallelization with dynamic load balancing in multi-node environments.

We report comprehensive benchmarks demonstrating favorable scaling with respect to system size, as well as high throughput for batched evaluations relevant to molecular dynamics, geometry optimization, and large-scale virtual screening. Parallel execution of a single system was carried out on up to 24 A100 GPUs. The implementation builds on optimized GPU-enabled variants of the LibintX and GauXC libraries to efficiently

compute density-fitted Coulomb, semi-numerical exchange, and exchange-correlation contributions.

Introduction

Modern quantum-chemical simulations are routinely employed to investigate the structure and dynamics of molecules and materials, yet their computational cost still severely limits the accessible time and length scales. HF and Kohn-Sham DFT^[1] remain the standard tools in computational chemistry^[2] and are generally viewed as the most affordable *ab initio* methods. However, even for these comparatively inexpensive approaches, the dominant step exhibits a formal scaling of $\mathcal{O}(N^4)$ with respect to the number of basis functions, due to the need to evaluate two-electron repulsion integrals, which is required to construct the Fock (or Kohn-Sham) matrix.^[3] It remains the principal bottleneck even when integral screening and sparsity are exploited. The computation cost of the HF and DFT analytic gradients with atomic-centered basis sets scales similarly because the evaluation of the “wavefunction” (or Pulay) forces involves first derivatives of two-electron integrals.^[4]

The rapid adoption of graphics processing units (GPUs) in high-performance computing has fundamentally changed how such electronic-structure methods are implemented, enabling large reductions in time-to-solution as originally pioneered by TeraChem.^[5,6] A broad range of strategies has been pursued to exploit these architectures. For Gaussian-type orbitals (GTO)-based methods, state-of-the-art codes now provide graphics processing unit (GPU)-accelerated implementations of electron-repulsion integrals and Fock (Kohn-Sham) matrix builds, enabling modeling of systems comprising hundreds to thousands of atoms.^[7-10] For a comprehensive review of the historical developments, we refer the reader to Ref. [11]

Recently, many works^[10,12-14] have focused on methods that reduce the computational cost of quantum chemistry calculations on GPUs by using approximations like density fitting (DF) or Cholesky decomposition^[15,16] and semi-numerical exchange techniques such as semi-

numerical linear-scaling exchange (Sn-link)^{[17][18]} and chains-of-spheres exchange (COSX)^[19] methods. These methods simplify the evaluation of two-electron Coulomb and exchange terms in HF and hybrid DFT calculations. Several GPU implementations^{[12][13][20]} have shown that these approaches can significantly speed up HF and DFT calculations for large molecules. Authors of CP2K program also presented their scalable multi-GPU implementation of two-electron integrals for a large molecular systems, employing hybrid Gaussian-Plane-Wave (GPW) scheme in combination with DF.^[21]

An efficient and scalable HF and DFT implementation was recently demonstrated by Williams-Young et al.^[13], combining LibintX and GauXC libraries and using DF-Coulomb calculation and semi-numerical exchange techniques. However, that work focuses only on the energy calculations. Many applications – e.g., locating stationary points, exploring reaction paths, and molecular dynamics – require analytic nuclear gradients. Although GauXC includes infrastructure for exchange-correlation (XC) derivatives, derivatives of the Coulomb (J) and exact-exchange (K) contributions are not available in these libraries. Here we implement analytic derivatives of the two-electron energy terms (J and K) within LibintX and GauXC, enabling HF and hybrid DFT gradients on single- and multi-GPU systems. By employing reduced-scaling DF and semi-numerical exchange kernels, the implementation enables high-throughput evaluation of energies and analytic nuclear gradients for small- and medium-sized molecules, as well as systematic studies of large organic systems on high-performance computing (HPC) hardware.

Accessibility and modular software design are key drivers for the rapid development and widespread adoption of modern numerical frameworks. This has been demonstrated particularly clearly by recent advances in large language models, where unprecedented computational scale is achieved through Python-centric ecosystems that emphasize modular software design and ease of experimentation. In quantum chemistry, Python interfaces have seen increasing adoption in recent years, enabling tighter integration with data-driven workflows and high-level numerical libraries. However, compared to machine-learning frameworks, the

availability of performance-critical components exposed through flexible Python application programming interfaces (APIs) remains limited.

In this work, we provide a Python API that exposes GPU-accelerated Coulomb and exchange energy and gradient kernels in a modular and scalable manner. The interface is designed to support rapid method development and high-throughput execution within Python-based workflows, while retaining performance on single- and multi-GPU systems. To further lower barriers to adoption, we provide precompiled Conda packages for common architectures, enabling straightforward installation and integration into existing Python environments.

The paper is organized as follows. In the [Background](#) section, we give a brief introduction to HF and Kohn-Sham DFT. The main kernels of the calculation – two-electron integrals and the XC functional integrations – are described, as well as the equations for their derivatives. In the [Implementation](#) section the details of the DFT energy and gradient algorithms are given. The description of benchmarks and their results are given in sections [Benchmarks](#) and [Results and Discussion](#) respectively. Finally, the concluding remarks are given in the last section.

Background

Both HF and Kohn-Sham DFT methods rely on an self-consistent field (SCF) procedure to determine a set of molecular orbitals (MOs) that represent the electronic structure of a molecule. In HF, the occupied MOs define an electronic wavefunction in the form of a single Slater determinant; in Kohn-Sham DFT a formally similar set of orbitals is used to represent the electronic density. Molecular orbitals are expanded as linear combinations of atomic orbitals (AOs), which transforms the SCF equations into the generalized eigenvalue problem:

$$\mathbf{FC} = \mathbf{SC}\epsilon, \tag{1}$$

where ϵ is a diagonal matrix of the orbital energies, $\mathbf{F}$ is the Fock (Kohn-Sham) matrix, $\mathbf{C}$ is the matrix of MO expansion coefficients in the AO basis, $\mathbf{S}$ is the overlap matrix in the AO basis. In the SCF procedure Equation (1) is solved iteratively because $\mathbf{F}$ depends on $\mathbf{C}$.

The closed-shell Fock (Kohn-Sham) matrix can be computed as a sum of several components:

$$\mathbf{F}^{\text{HF}} = \mathbf{h}^{\text{core}} + 2\mathbf{J} - \mathbf{K} \quad (2)$$

$$\mathbf{F}^{\text{KS}} = \mathbf{h}^{\text{core}} + 2\mathbf{J} - \alpha\mathbf{K} + \mathbf{F}^{\text{XC}} \quad (3)$$

where $\mathbf{h}^{\text{core}}$ is the so-called core Hamiltonian, which comprises electronic kinetic energy and electron-nuclei attraction; $\mathbf{J}$ is the Coulomb interaction of an electron with the mean field of all electrons in the system; $\mathbf{K}$ is the exchange operator; and $\mathbf{F}^{\text{XC}}$ is the XC functional integral. The coefficient α depends on the type of the XC functional: $\alpha = 0$ for pure XC functionals, and between 0 and 1 for hybrid DFT. All components except $\mathbf{h}^{\text{core}}$ depend on the MO coefficients and have to be recomputed at each step of the SCF procedure.

Density-fitting Method for Computing two-electron Coulomb Matrix

The Coulomb term $\mathbf{J}$ in HF and Kohn-Sham DFT methods can be efficiently evaluated using the DF approximation, which is based on the resolution of the identity of four-center electron repulsion integrals.^{[15][22]} The basic idea is to approximate the pair density, formed by the product of atomic orbital basis functions, by a linear combination of auxiliary basis functions:

$$\phi_\mu(\mathbf{r})\phi_\nu(\mathbf{r}) \approx \sum_P C_{\mu\nu}^P \phi_P(\mathbf{r}), \quad (4)$$

where ϕ are basis functions, and $C_{\mu\nu}^P$ are the fitting coefficients. Greek indices (μ , ν , etc.) correspond to the regular basis set, the capital Latin letters (P and Q) denote the auxiliary

basis set.

The coefficients $C_{\mu\nu}^P$ are obtained by:

$$C_{\mu\nu}^P = \sum_Q (\mu\nu|Q)(Q|P)^{-1}, \quad (5)$$

where $(\mu\nu|Q)$ are three-center two-electron Coulomb integrals between regular and auxiliary basis functions, $(Q|P) = (P|Q)$ are elements of the symmetric matrix given by the two-center two-electron Coulomb integrals over auxiliary functions, and $(Q|P)^{-1}$ are the elements of the matrix inverse to $(Q|P)$.^[15] The integrals can be computed as follows:

$$(\mu\nu|Q) = \iint \frac{\phi_\mu(\mathbf{r})\phi_\nu(\mathbf{r})\phi_Q(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r} d\mathbf{r}', \quad (6)$$

$$(P|Q) = \iint \frac{\phi_P(\mathbf{r})\phi_Q(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r} d\mathbf{r}'. \quad (7)$$

The Coulomb matrix elements, $J_{\mu\nu}$, are computed as follows:

$$J_{\mu\nu} = \sum_{\lambda\sigma} D_{\lambda\sigma}(\mu\nu|\lambda\sigma) \approx \sum_{PQ} (\mu\nu|P)(P|Q)^{-1} \sum_{\lambda\sigma} (Q|\lambda\sigma)D_{\lambda\sigma}, \quad (8)$$

or, in contracted form,

$$J_{\mu\nu} \approx \sum_P (\mu\nu|P)v_P, \quad (9)$$

with the Coulomb potential fitting coefficients given by

$$v_P = \sum_Q (P|Q)^{-1} \sum_{\lambda\sigma} (\lambda\sigma|Q)D_{\lambda\sigma} = \sum_Q (P|Q)^{-1}x_Q \quad (10)$$

The density-fitted Coulomb matrix (DF-J) can be computed using the sequence of steps shown in Algorithm [1](#).

To reduce memory consumption, it is possible to avoid the explicit storage of the three-center Coulomb integral tensor $(\mu\nu|P)$ by performing its contraction with the density matrix

Algorithm 1 The DF Coulomb matrix calculation algorithm.

Input: Cholesky factors $\mathbf{L} \in \mathbb{R}^{n_{aux} \times n_{aux}}$ of the auxiliary matrix $\mathbf{V} \in \mathbb{R}^{n_{aux} \times n_{aux}}$, density matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, basis sizes n , auxiliary basis size n_{aux} , a function that evaluates 3-center 2-electron integrals $(\mu\nu|P)$

Output: Coulomb matrix $\mathbf{J} \in \mathbb{R}^{n \times n}$

Temporaries: $\mathbf{x}, \mathbf{v} \in \mathbb{R}^{n_{aux}}$

```

1: for  $Q \in [n]$  do
2:    $x_Q \leftarrow \sum_{\lambda\sigma} (\lambda\sigma|Q) D_{\lambda\sigma}$  ▷ Form the auxiliary potential
3: end for
4:  $\mathbf{v} \leftarrow \text{solveSLE}(\mathbf{L}, \mathbf{x})$  ▷ Fitted density  $v_P = (P|Q)^{-1}(\lambda\sigma|Q) D_{\lambda\sigma}$ 
5: for  $P \in [n_{aux}]$  do
6:    $J_{\mu\nu} \leftarrow \sum_P (\mu\nu|P) v_P$  ▷ Build the Coulomb matrix
7: end for

```

and the Coulomb fitting coefficients on-the-fly. This approach requires recomputation of the three-center two-electron integrals twice per SCF iteration. However, it reduces the intermediate storage requirements from $\mathcal{O}(N^2 N_{aux})$ to $\mathcal{O}(N_{aux})$, which can be advantageous in large-scale calculations. Additionally, certain two-electron evaluation methods permit the contraction of the density matrix with intermediate quantities before the final integrals are formed, thereby reducing the total number of floating-point operations.^[23–25] In the McMurchie-Davidson scheme, the density matrix is contracted with coefficients of the Cartesian GTOs expansion in the basis of Hermitian GTOs.

Density-fitting reduces the computational scaling and memory requirements associated with the four-center Coulomb integrals from $\mathcal{O}(N^4)$ to $\mathcal{O}(N^3)$. This is particularly advantageous for large systems and pure DFT calculations. Although a similar expansion may be applied to the exact exchange term,^[22] the resulting algorithm does not improve the formal computational scaling of exchange matrix construction, which remains $\mathcal{O}(N^4)$. Alternative approaches, like semi-numerical exact exchange^[25] or pseudo-spectral and related methods,^[26–28] can be applied to reduce the cost of exchange matrix calculations.

Semi-Numerical Calculation of the Exchange Matrix

The exact exchange matrix $K_{\mu\nu}$ constitutes a major computational bottleneck in HF and hybrid DFT calculations. Semi-numerical methods mitigate this cost by evaluating parts of the exchange contribution on a spatial integration grid.^{[17][19]}

The central idea is to express the two-electron integrals required for the exchange matrix using a mixed analytic-numerical approach:

$$\begin{aligned} K_{\mu\nu} &= \sum_{\lambda\sigma} D_{\lambda\sigma}(\mu\lambda|\nu\sigma) \\ &= \sum_{\lambda\sigma} D_{\lambda\sigma} \sum_i w_i \phi_\mu(\mathbf{r}_i) \phi_\lambda(\mathbf{r}_i) \int \frac{\phi_\nu(\mathbf{r}) \phi_\sigma(\mathbf{r})}{|\mathbf{r} - \mathbf{r}_i|} d\mathbf{r}, \end{aligned} \tag{11}$$

where ϕ_μ , ϕ_ν , ϕ_λ , and ϕ_σ denote atomic-orbital basis functions, $\mathbf{D}$ is the density matrix, and $\{\mathbf{r}_i, w_i\}$ are the points and weights of a real-space quadrature. In practice, atom-centered grids combined with standard numerical quadrature schemes are employed to approximate the real-space integrals.^{[18][19]} Screening based on density-matrix elements and basis-function overlaps can substantially reduce the number of non-negligible contributions, lowering the effective scaling well below the formal $\mathcal{O}(N^4)$ limit and, in favorable cases, approaching linear scaling. This mixed analytic–numerical formulation is particularly effective for hybrid DFT calculations, where the evaluation of exchange terms dominates the overall computational cost.

Building on this formulation, several practical algorithms have been developed to compute the exchange matrix efficiently, most notably the COSX^[19] and Sn-link^[18] methods, which are now widely adopted in modern quantum-chemistry packages. In these approaches, the exchange matrix expression in eq. (11) is typically reorganized to expose intermediate quantities and reduce computational overhead. One commonly used reformulation proceeds

as

$$\begin{aligned}
K_{\mu\nu} &= \sum_i w_i \phi_\mu(\mathbf{r}_i) \sum_\sigma \int \frac{\phi_\nu(\mathbf{r}) \phi_\sigma(\mathbf{r})}{|\mathbf{r} - \mathbf{r}_i|} d\mathbf{r} \sum_\lambda \phi_\lambda(\mathbf{r}_i) D_{\lambda\sigma} \\
&= \sum_i w_i \phi_\mu(\mathbf{r}_i) \sum_\sigma A_{\nu\sigma i} F_{\sigma i} \\
&= \sum_i \phi_\mu(\mathbf{r}_i) R_{\nu i},
\end{aligned} \tag{12}$$

where $F_{\sigma i} = \sum_\lambda \phi_\lambda(\mathbf{r}_i) D_{\lambda\sigma}$, $A_{\nu\sigma i} = \int \phi_\nu(\mathbf{r}) \phi_\sigma(\mathbf{r}) |\mathbf{r} - \mathbf{r}_i|^{-1} d\mathbf{r}$, $R_{\nu i} = w_i \sum_\sigma A_{\nu\sigma i} F_{\sigma i}$. The program usually runs over 3D locality-based clusters (batches) of grid points. Non-negligible basis functions are evaluated at the quadrature points. Then, $F_{\sigma i}$, $A_{\nu\sigma i}$ are computed and $R_{\nu i}$ is formed. Finally, the contribution to the exchange matrix from the current batch is computed. As a result, the exchange build can be expressed predominantly in terms of dense linear algebra operations. As in the two-electron case, the evaluation of integrals $A_{\mu\nu i}$ may be fused with their contraction with $F_{\sigma i}$ to improve performance.^{[12][13][29]}

Nuclear Derivatives

Energy gradients are essential to optimize molecular structures or perform molecular dynamics simulation. The Coulomb and exact exchange energy derivatives are among the most time-consuming components of the gradient. The following subsections detail their evaluation.

Coulomb Energy Derivatives

The evaluation of the nuclear derivatives of the Coulomb energy (E^J) within the density-fitting approximation follows directly from the analytical form of the DF Coulomb operator. The derivative of E^J with respect to the Cartesian coordinate $\xi \in \{x, y, z\}$ of atom A is given by:^[30]

$$\frac{\partial E^J}{\partial A_\xi} = \sum_P v_P \sum_{\mu\nu} D_{\mu\nu} \frac{\partial(\mu\nu|P)}{\partial A_\xi} - \frac{1}{2} \sum_{PQ} v_P v_Q \frac{\partial(P|Q)}{\partial A_\xi} \tag{13}$$

Thus, computing nuclear gradients requires evaluating the derivatives of the two-center and three-center electron–repulsion integrals with respect to the nuclear coordinates. Equation (13) can be formally separated into two contributions, corresponding to derivatives with respect to the primary basis-set centers and the auxiliary basis-set centers, respectively:

$$\frac{\partial E^{J,prim}}{\partial A_\xi} = \sum_{\mu\nu} D_{\mu\nu} \sum_P v_P \left(\frac{\partial(\mu\nu)}{\partial A_\xi} \middle| P \right) \quad (14)$$

$$\frac{\partial E^{J,aux}}{\partial A_\xi} = \sum_P v_P \sum_{\mu\nu} D_{\mu\nu} \left(\mu\nu \middle| \frac{\partial P}{\partial A_\xi} \right) - \frac{1}{2} \sum_{PQ} v_P v_Q \frac{\partial(P|Q)}{\partial A_\xi} \quad (15)$$

It is convenient to compute derivatives with respect to the primary and auxiliary basis sets separately in the direct DF-J-engine method. It allows in-place contraction of the intermediate Hermitian integrals with the density matrix $\mathbf{D}$ or fitted density v , leveraging the same two-step direct DF-J-engine scheme as in the Coulomb matrix calculation. Nuclear derivatives of the two-electron integrals can be evaluated analogously to the integrals themselves, using recursion relations or Rys quadrature.^{31,33} An overview of the McMurchie-Davidson (MD) method for computing two-electron integral nuclear derivatives which was used in this work is given in the Supporting Information.

Exchange Energy Derivatives

The contribution of the exchange term to the nuclear gradient is expressed as:

$$\frac{\partial E^K}{\partial A_\xi} = \sum_{\mu\nu} \frac{\partial K_{\mu\nu}}{\partial A_\xi} D_{\mu\nu}, \quad (16)$$

where the semi-numerical exchange matrix derivative can be computed as follows:

$$\begin{aligned}
\frac{\partial K_{\mu\nu}}{\partial A_\xi} &= \sum_{\lambda\sigma} D_{\lambda\sigma} \frac{\partial(\mu\lambda|\nu\sigma)}{\partial A_\xi} \\
&= \sum_i w_i \frac{\partial\phi_\mu(\mathbf{r}_i)}{\partial A_\xi} \sum_\sigma \int \frac{\phi_\nu(\mathbf{r})\phi_\sigma(\mathbf{r})}{|\mathbf{r} - \mathbf{r}_i|} d\mathbf{r} \sum_\lambda D_{\lambda\sigma} \phi_\lambda(\mathbf{r}_i) \\
&= \sum_i \frac{\partial\phi_\mu(\mathbf{r}_i)}{\partial A_\xi} R_{\nu i}
\end{aligned} \tag{17}$$

This semi-numerical scheme is similar to the computation of the exchange matrix in the energy evaluation, with the additional requirement of evaluating the nuclear derivatives of the basis functions at each grid point.^[19] Notably, no derivatives of one-electron integrals are required in this approach.

Implementation

We implemented Coulomb- and exact exchange energy derivatives as extensions to the LibintX and GauXC libraries, respectively. To exploit multi-node HPC systems, we added a distributed LibintX variant using message passing interface (MPI) parallelization. To improve the accessibility, we developed a Python APIs for the extended LibintX and GauXC libraries and develop pre-compiled versions for common architectures via package managers. The interfaces are designed to be compatible with the PySCF package,^[34] facilitating input preparation, analysis of results, and debugging within established Python-based workflows. Performance benchmarks for HF and DFT calculations were carried out using an internal Python-based quantum chemistry software. In the following, we will describe relevant parts including the numerically demanding operations in detail.

Implementation of the Coulomb Matrix

The integral-direct distributed J-engine implementation in the LibintX library^[13,35,36] computes the Coulomb matrix using batches of three-center two-electron integrals (Algorithm [1](#))

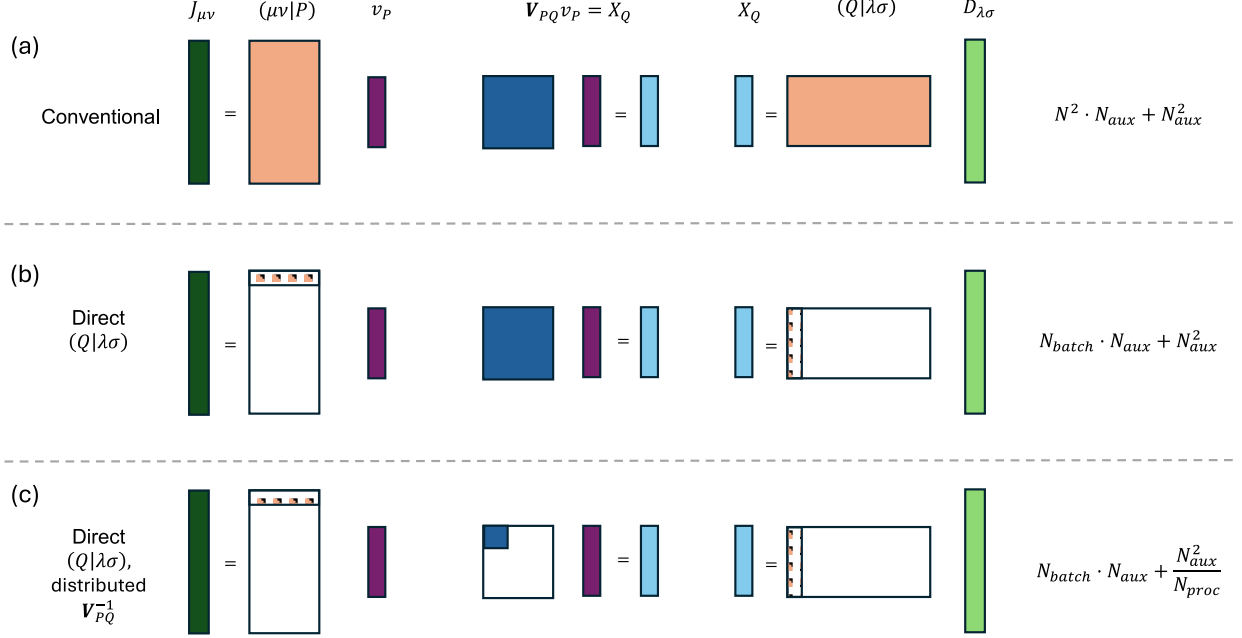


Figure 1: Schematic representation of the estimated memory requirements of the DF-J algorithm run in different modes. **(a)** shows consumption with full storage of all objects, **(b)** shows memory requirements for distributed 3-center integral calculation (LibintX and present work), and **(c)** shows memory requirements if additionally distributed V storage is used via cuSOLVERMp (present work). Shaded blocks denote integrals computed in the direct way. Note that gradient contributions follow an analogous scheme.

and Figure 1. Three-center integrals are never materialized. Instead, their contractions $(\lambda\sigma|Q)D_{\lambda\sigma}$ and $(\mu\nu|P)v_P$ are computed, which allows to significantly reduce the amount of calculations.^{23,25} As a consequence, three-center integral kernels must be called twice per SCF as indicated in panel (b) of Figure 1. The LibintX J-engine class provides methods for computing the three-center two-electron integrals contractions $(\lambda\sigma|Q)D_{\lambda\sigma}$ and $(\mu\nu|P)v_P$ (lines 2 and 6 of Algorithm 1); all other steps must be implemented by the user. Furthermore, for these operations it supports running on multiple GPUs installed on a single node. In this work, we developed complementary code that implements the remaining steps of Algorithm 1 and exposes them through a Python API.

Implementation of the Coulomb Energy Derivatives

For clarity, we denote the two-center Coulomb integrals matrix over the auxiliary basis as $\mathbf{V}$. Analytic equations often use the inverse two-center Coulomb matrix $\mathbf{V}^{-1}$ (see eq. (8)). In our implementation, the multiplication of $\mathbf{V}^{-1}$ with a vector x (Eq. (10)) is carried out by solving the linear system $\mathbf{V} v = x$, which is numerically more stable and accurate. By default, the linear system is solved on GPU using cuSOLVER (Fig. 1b). To alleviate device-memory constraints for large auxiliary bases, we optionally support cuSOLVERMp with distributed storage, trading increased communication overhead for reduced memory footprint (Fig. 1c). During J-engine initialization, a Cholesky factorization of $\mathbf{V}$ is computed once and retained either in device memory, when sufficient memory is available, or on the host otherwise. The stored Cholesky factors are subsequently reused to evaluate the fitted Coulomb potential coefficients v in all Coulomb matrix and gradient calculations, thereby amortizing the cost of the factorization across SCF iterations.

The Coulomb contribution to the nuclear gradient is computed using the modified $\mathbf{J}$ matrix construction (Algorithm 2). The Coulomb potential $\mathbf{x}$ and its derivatives with respect to the auxiliary basis functions $\partial\mathbf{x}/\partial C_\xi$ (Algorithm 2, lines 1 to 6) are evaluated simultaneously. Next the derivatives of Coulomb energy with respect to the primary basis set centers are obtained by contracting the corresponding derivatives of the three-center two-electron integrals with the fitted density $\mathbf{v}$ and density matrix $\mathbf{D}$ (Algorithm 2, lines 9 and 10). The auxiliary basis contribution to the J-gradient is computed separately (Algorithm 2, line 13). This procedure preserves the possibility to perform an early contraction with either the density matrix or the fitted Coulomb potential $\mathbf{v}$.³⁵ The details of the McMurchie-Davidson scheme of derivative integrals calculation are described in the Supporting Information.

It should be noted that our implementation does not use atomic gradient vectors directly. Instead, gradients are computed with respect to the atomic orbital centers, avoiding an explicit orbital-to-atom correspondence map. The atomic gradient vector can be reconstructed in a post-processing step on a higher level by accumulating the orbital contributions

Algorithm 2 Integral-direct Coulomb gradient calculation algorithm using density-fitting approximation.

Input: Cholesky factors $\mathbf{L} \in \mathbb{R}^{n_{aux} \times n_{aux}}$ of the auxiliary matrix $\mathbf{V} \in \mathbb{R}^{n_{aux} \times n_{aux}}$, two-center two-electron integral derivatives $\left(\frac{\partial P}{\partial A_\xi} \middle| Q\right) \in \mathbb{R}^{n_{aux} \times n_{aux} \times 3}$, density matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, basis sizes n , auxiliary basis size n_{aux} , number of atoms n_{atom} , and a function that evaluated 3-center 2-electron integrals $(\mu\nu|P)$ on demand in batches

Output: Primary Coulomb energy gradient contributions $\mathbf{G}^{prim} \in \mathbb{R}^{n \times 3}$ (14), auxiliary Coulomb energy gradient contributions $\mathbf{G}^{aux} \in \mathbb{R}^{n_{aux} \times 3}$ (15)

Temporaries: $\mathbf{x} \in \mathbb{R}^{n_{aux}}$, $\mathbf{dx} \in \mathbb{R}^{n_{aux} \times 3}$, $\mathbf{v} \in \mathbb{R}^{n_{aux}}$

- ▷ Build Temporary objects
- 1: **for all** $Q \in [n_{aux}]$ **do**
- 2: $x_Q \leftarrow \sum_{\mu\nu \in [n]} (\mu\nu|Q) D_{\mu\nu}$
- 3: **for all** $\xi \in [3]$ **do**
- 4: $dx_{Q\xi} \leftarrow \sum_{\mu\nu \in [n]} (\mu\nu|\partial Q/\partial A_\xi) D_{\mu\nu}$
- 5: **end for**
- 6: **end for**
- ▷ Solve linear equations using Cholesky factors
- 7: $\mathbf{v} \leftarrow \text{solveSLE}(\mathbf{L}, \mathbf{x})$
- ▷ Primary and auxiliary basis contributions to energy gradient
- 8: **for all** $(\mu, \xi) \in [n] \times [3]$ **do**
- 9: $G_{\mu\xi}^{prim} \leftarrow \sum_{\nu \in [n]} D_{\mu\nu} \sum_{Q \in [n_{aux}]} \left(\frac{\partial \mu}{\partial A_\xi} \nu \middle| Q\right) v_Q$
- 10: $G_{\nu\xi}^{prim} \leftarrow \sum_{\mu \in [n]} D_{\mu\nu} \sum_{Q \in [n_{aux}]} \left(\mu \frac{\partial \nu}{\partial A_\xi} \middle| Q\right) v_Q$ ▷ Prim. basis contribution (14)
- 11: **end for**
- 12: **for all** $(P, \xi) \in [n_{aux}] \times [3]$ **do**
- 13: $G_{P\xi}^{aux} \leftarrow \left(dx_{P\xi} - \frac{1}{2} \sum_{Q \in [n_{aux}]} \left(\frac{\partial P}{\partial A_\xi} \middle| Q\right) x_Q\right) x_P$ ▷ Aux. basis contribution (15)
- 14: **end for**
- 15: **return** $\mathbf{G}^{prim}, \mathbf{G}^{aux}$

associated with each atom.

To make use of multi-node HPC systems, we implemented a distributed, multi-node J-engine using MPI parallelization. Work is partitioned statically over $(\mu\nu|)$ pairs, with a dedicated load balancer instance for each total angular momentum case. The amount of work per task is unevenly distributed, due to differing contraction degree of the shell pairs. To mitigate this imbalance, the task list is shuffled deterministically using an identical RNG seed on all processes. After computation, the vector of fitted potential and the final matrix $\mathbf{J}$ (Algorithm 1 lines 2 and 6) are reduced globally across all ranks using `MPI.Allreduce`.

Implementation of Exact Exchange Energy Derivatives

GauXC^[13,37,38] is a high-performance library, which is capable of computing functional energy and derivatives, as well as semi-numerical exact exchange for a broad choice of 3D grids used in chemistry. It is parallelized with MPI and offers host and GPU implementations for most of the kernels. In this study we extended the GauXC library with the Python interface, which encapsulates low-level C++ classes used in GauXC. Additionally, we added code for computing semi-numerical exact exchange gradients.

The semi-numerical exchange gradients implementation strictly follows Equations (11) and (17). The corresponding routine is nearly identical to the K matrix calculation, with a few key differences. First, derivatives of the basis functions are evaluated on the integration grid. Second, instead of forming the exchange matrix, three derivative matrices containing the elements $\partial K_{\mu\nu}/\partial A_\xi$ are computed. Finally, the exchange contribution to the nuclear gradient is obtained as $\partial E^K/\partial A_\xi = \sum_{\mu\nu} D_{\mu\nu} \partial K_{\mu\nu}/\partial A_\xi$. As in the Coulomb energy derivative case, the implementation employs orbital-centered gradients rather than atomic gradients.

The Details of the SCF Driver Implementation

The SCF procedure employs direct inversion in the iterative subspace (DIIS)^[39] to accelerate convergence of the Hartree–Fock–Roothaan equations. All required input data for the HF and DFT calculations, including basis sets, one-electron integrals, two-center Coulomb integrals, and their nuclear derivatives, are generated using the PySCF integral module based on the `libcint` library.^[40] The SCF driver used in this work is intended to provide a reliable and transparent environment for benchmarking two-electron energy and gradient kernels. A fully featured production-level SCF program **TQ Ignite** which uses the backends presented in this work will be released separately as part of a broader quantum chemistry software package.

Benchmarks

To evaluate the performance of the proposed framework, we conducted a series of benchmarks in two representative regimes: (i) scaling a single system across one or multiple GPUs, and (ii) executing multiple small systems in parallel. The first regime reflects common workloads such as MD, QM/MM simulations, or structural relaxations. The second regime mirrors high-throughput scenarios, where geometry optimizations and subsequent energy evaluations are carried out for many small- to medium-sized molecules – an increasingly important use case for generating training data for machine learning models.

Molecular Dataset

Open Molecules 2025 (OMol25) is a large-scale dataset and benchmark comprising over 100 million single-point DFT calculations on molecular systems of up to 350 atoms, spanning biomolecules, metal complexes, electrolytes, and community datasets across the first 83 elements.^[41] It is designed to train molecular machine-learning models to predict chemical properties of molecular systems. Such datasets and benchmarks are especially relevant for computer-aided drug discovery, where accurate yet scalable modeling of proteins, nucleic acids, and small molecules, including conformer ranking and binding-mode refinement, are routine tasks in virtual screening and lead optimization workflows.

Our benchmark dataset is composed of a subset of 114 molecules OMol25-neutral dataset, four molecules used in the original LibintX and GauXC benchmark studies,^[13] and three additional protein molecules (valinomycin, ciclosporine, and gramicidine). The subset of OMol25-neutral molecules was selected to represent all range of molecular sizes from the complete dataset. The details of the selection procedure and the description of the molecules including their geometries in the subset are given in the Supporting Information. The seven additional molecular systems ranging from 113 atoms (Taxol) to 1231 atoms (Ubiquitin) are displayed in Table 1 and Figure 2. These systems span large drug-like molecules to small

proteins.

Benchmark Description

For each molecule we performed a set of synthetic benchmarks and recorded the wall time of the following computational kernels: Coulomb (J) build, exact exchange (K) build, and their corresponding gradient contributions. Each benchmark consists of an initial un-timed spin-up calculation followed by ten repeated calculations of a single kernel; we report the average runtime over these ten iterations.

Next benchmark was designed to simulate a high-throughput workload. For each molecule in the selected subset of OMol25-neutral dataset we carried out full PBE0/cc-pVDZ calculation followed by the calculation of the gradient. Two-electron quantities were computed using the code from the Ref. [13](#) and the current work. One-electron integrals and their gradients were computed using PySCF.[34](#) To study device utilization under load, we executed four benchmark series with 1, 2, 3, and 4 concurrent processes per GPU. We recorded timings for the GPU-accelerated kernels as well as the wall-clock time to complete each batch.

To estimate GPU utilization we collected `SMs Active [Throughput %]` performance counter over the duration of the OMol25 benchmark using NVIDIA Nsight Systems (v. 2025.6.1) software. The sampling frequency was set to 1000 Hz. The gross GPU utilization was computed as a time average over all samples including GPU idle periods. The net GPU utilization was computed as a time average only over the periods, when the GPU was active, i.e. the average over non-zero values of the collected metric.

To evaluate the performance of the framework on multi-GPU HPC architectures, we carried out benchmarks on configurations ranging from 1 to 24 NVIDIA Tesla A100 40 GB devices. Each GPU was paired with 32 CPU cores and 128 GB of host memory. For testing we selected three biological systems of increasing size—valinomycin, crambin, and ubiquitin (Table [1](#)).

We used the cc-pVDZ[42](#) basis set together with the cc-pVDZ-jkfit[22](#) in all our tests.

Table 1: The description of molecules used in benchmarks. Basis set and auxiliary basis set are cc-pVDZ and cc-pVDZ-jkfit respectively.

Molecule	# atoms	# basis functions	# aux. basis functions	V^{-1} memory, GB
taxol	113	1185	6297	0.30
valinomycin	168	1620	8568	0.55
ciclosporine	196	1830	9660	0.70
olestra	453	4015	21 125	3.32
gramicidine	552	5480	29 032	6.28
crambin	642	6504	34 656	8.95
ubiquitin	1231	12 179	64 536	31.03

Single-point energy and gradient evaluations used the hybrid PBE0^[43] functional. Screening accuracy for the two-electron integrals and semi-numerical exchange energy were set to 10^{-12} and 10^{-8} respectively. In all benchmarks, two-center two-electron integrals and their derivatives are computed using the `libcint` library as provided by PySCF.^[40]

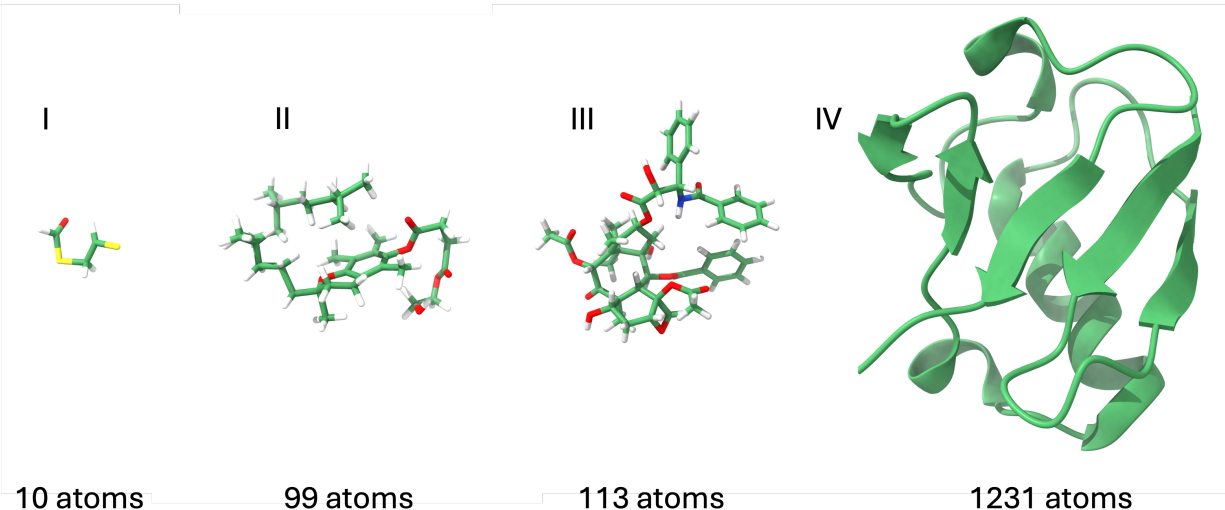


Figure 2: Exemplary molecules from the OMol25 dataset (ID 15254030 (I) and 30296213 (II)), and molecules used in the multi-GPU benchmark: Taxol (III) and Ubiquitin (IV). The corresponding basis set information is shown in Table [1](#).

Hardware Configuration

Single-GPU performance was evaluated on Google Cloud Platform (GCP) instances equipped with NVIDIA T4 (16 GB), V100 SXM2 (16 GB), and A100 SXM4 (80 GB) GPUs. The T4

and V100 benchmarks were performed on N1 machine types (8 vCPUs, 2 vCPU/core, 52 GB RAM), while the A100 benchmarks used an A2 machine type (12 vCPUs, 2 vCPU/core, 170 GB RAM). All cloud virtual machine instances ran Ubuntu 24.04.1 LTS with CUDA Toolkit 12.9.

Multi-GPU benchmarks were carried out using dual-socket AMD EPYC 7H12 (64 cores, 2.6 GHz) nodes, equipped with four NVIDIA Tesla A100-SXM4 40 GB devices and 512 GB of system memory. The Mellanox HDR InfiniBand interconnect providing node-to-node communication. Each node runs RHEL 8.10 operating system with 4.18.0 Linux kernel, NVIDIA driver 535.274.02, and CUDA toolkit 12.2.

Results and Discussion

Single-GPU Performance

Figure 3 reports the wall time as a function of basis size for both energy and gradient kernels evaluations. The panels show: (a,b) Coulomb kernel without screening, (c,d) Coulomb kernel with Schwarz screening at a 10^{-12} threshold, and (e,f) exact exchange kernel. In all cases, the computation time follows an asymptotic power-law dependence, as indicated by the fitted curves. When screening is applied, the two-electron Coulomb and exchange calculations have (sub-)quadratic asymptotic scaling with respect to the basis set size. A runtime plateau is visible for small systems, and it is more pronounced for Coulomb than for exchange integrals. As expected, Coulomb builds are consistently faster than exchange builds across all GPUs.

For the Coulomb kernels – both screened and unscreened – the gradient evaluation becomes more expensive for large systems. This behavior is anticipated, as gradient kernels require evaluating higher angular momentum intermediate integrals. Nevertheless, the gradient and energy evaluation times remain of similar magnitude. The exact exchange gradient kernel is also slightly more expensive than the energy kernel. On V100 and A100 GPUs, the runtime ratios $T_{\text{grad}}/T_{\text{energy}}$ fall in the ranges 1.2–1.5 for **J** and 1.1–1.15 for **K**.

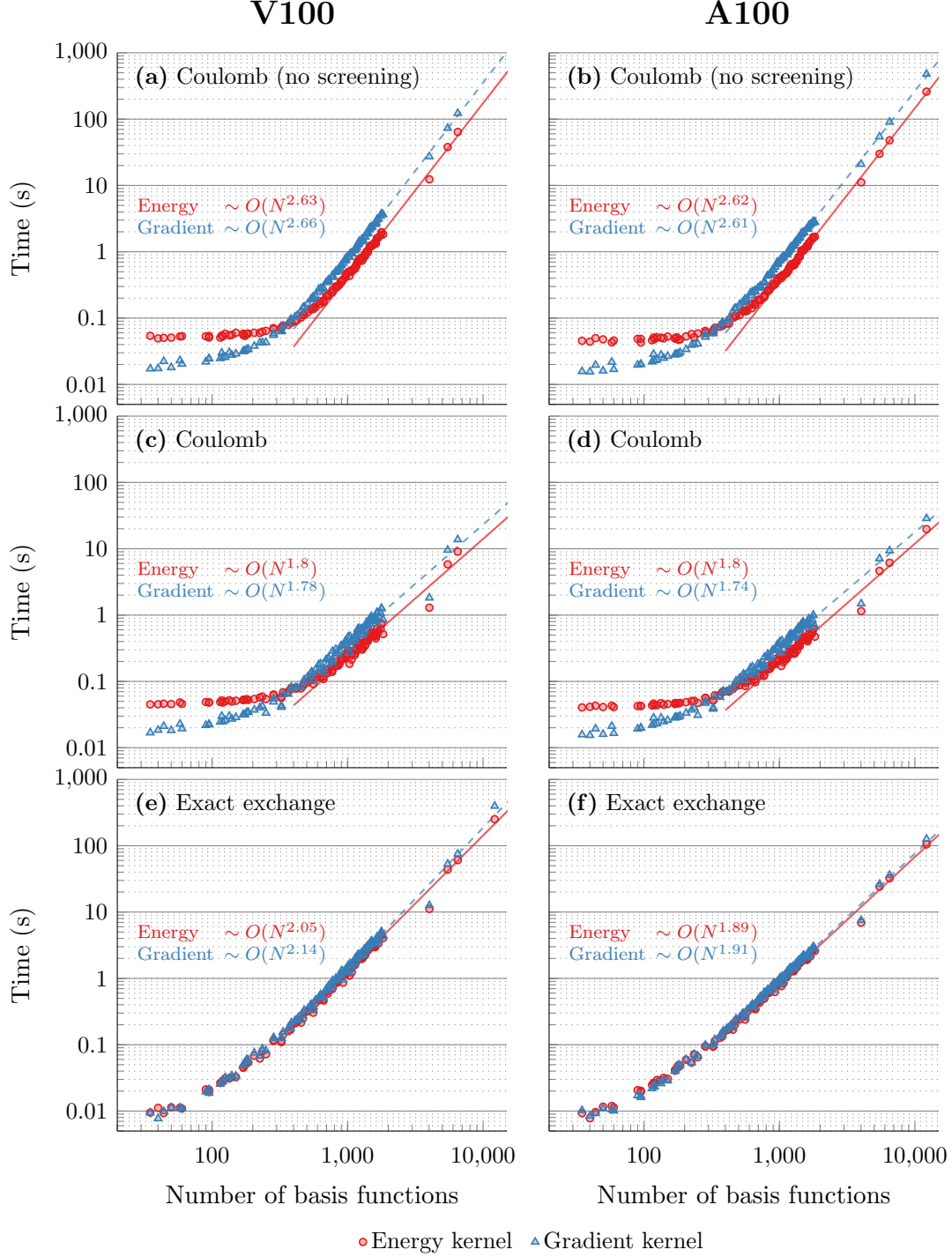


Figure 3: Single-GPU time to solution as a function of the system size for individual computational kernels on NVIDIA V100 (left) and A100 (right) GPUs. Each subplot illustrates the performance scaling of the following kernels: **(a)**, **(b)** density-fitting Coulomb (**J**) without screening; **(c)**, **(d)** density-fitting Coulomb with Schwarz screening turned on (screening threshold 10^{-12}); **(e)**, **(f)** semi-numerical exchange (**K**). The asymptotic lines demonstrate computational complexity of kernels depending on the system size.

The A100 provides a speedup of $\sim 1.2 \times$ over the V100 for Coulomb kernels. That is consistent with the theoretical FP64 peak performance of V100 and A100 GPUs (7.8 and 9.7 TFLOPS respectively). The observed performance difference for exchange kernels is bigger (up to $1.5\text{--}2.0 \times$ speedup), especially when benchmarking bigger systems. We attribute this to the larger device memory (which allows for larger batches), higher memory bandwidth, as well as faster device-host interconnection available on A100.

Across both energy and gradient evaluations, the most computationally demanding component is the semi-numerical exchange step. Its cost is approximately an order of magnitude higher than that of the Coulomb contribution. The observed scaling behavior is sub-quadratic for both **J** and **K** kernels with respect to the number of basis functions.

The practical limitation of the current Coulomb kernels implementation is the size of the two-center two-electron integrals matrix **V**. Currently, the code is capable of running systems with $\approx 45\,000$ auxiliary basis functions on V100 (16 GB) and up to $\approx 100\,000$ auxiliary basis functions for A100 with 80 GB.

High-Throughput Single-GPU Performance

In high-throughput settings, parallelization over multiple GPUs is trivial; therefore it is only relevant how efficiently a single GPU can be occupied. For small molecules, individual calculation typically underutilizes an NVIDIA A100 GPU. It is therefore advantageous to execute multiple calculations concurrently in batches in order to hide latencies and more effectively saturate the GPU. We performed PBE0 energy and gradient calculations on the subset of OMol25-neutral dataset; the results are summarized in Table 2. We evaluated batched execution with up to four concurrent tasks per GPU.

A rough estimate based on the CPU-only code suggests that around 25% speedup can be obtained if GPU kernels fully overlap with CPU-only work. In practice, however, the GPU utilization is less than this estimate. Profiling with NVIDIA Nsight Systems shows, that even for the largest molecules in our OMol25 subset, the gross and net GPU utilizations

were only about 35 % and 85 %, respectively. For the whole subset the gross and net GPU utilization were 33 % and 82 % respectively. Here, gross GPU utilization is the time-average over the entire run; net utilization is the time-average over periods of nonzero GPU activity.

Consistent with these observations, allowing four processes to run concurrently on a single GPU results in an almost $2\times$ reduction in wall-clock time for the full batch. This demonstrates that batched multi-process execution is an effective strategy for improving throughput in this workflow.

Based on computations of the representative subset, we estimated the computational resources required to process the entire OMol25-neutral dataset. The results are summarized on Figure 4. This estimate was obtained as a weighted sum using timings measured on a representative subset of molecules and the distribution of the basis set sizes in the full dataset. Single-point DFT energies and their properties can be computed in ≈ 6500 h, with an additional ≈ 1000 h required for gradient evaluations. Therefore, the full calculation is practically possible on contemporary GPU clusters and economically feasible even on commercial cloud infrastructures.

A primary cause of low GPU utilization is host-side time spent in PySCF routines (one-electron integrals, two-center two-electron integrals, their derivatives, and the initial guess). We expect OMol25 screening runtimes to decrease further if these builds are offloaded to the GPU. As this work focuses on **J**, **K**, and XC matrices and their gradients, we will focus on this improvement in future work.

Multi-GPU Performance

The results for the individual two-electron energy and gradient kernels are shown in Figures 5 to 7. As in the single-GPU case, execution time is driven by the exact-exchange (*K*) kernel. The next-largest cost is the XC kernel for small molecules, but shifts to the Coulomb (*J*) kernel as system size grows.

The best parallel scaling is observed for the exchange energy and gradient kernels, which

Table 2: Performance metrics of major GPU-accelerated functions for different numbers of processes per GPU. Values represent compute time, i.e. the time difference between start and end times of the function, summed over all runs in the batch. Total compute time is the sum of the initial guess, DFT energy, and gradient execution times per molecule. Wall clock time represents the total elapsed time from batch execution. Non-GPU time share represents the total compute time by CPU-only kernels and host overhead relative to total compute time.

Function	# processes per GPU			
	1	2	3	4
snK exchange, s	1330.7	1709.9	2032.8	2379.3
XC, s	1064.9	1621.4	2270.1	2980.9
Coulomb, s	450.4	554.2	691.0	835.3
snK derivative, s	112.4	141.2	177.5	208.8
XC gradient, s	95.9	151.4	206.7	280.5
Initialization, s	81.6	104.9	121.9	157.8
Coulomb gradient, s	37.7	50.5	58.5	63.8
Solve SCF eqs., s	26.5	64.5	95.0	139.9
DIIS error vector, s	14.1	25.1	31.8	42.1
Non-GPU time share, %	24.0	24.1	24.9	25.7
Total compute time, s	4234.4	5831.9	7567.9	9537.7
Total wall clock time, s	4352.9	2993.8	2585.1	2434.6

also dominate the overall runtime. For ubiquitin and crambin, the Coulomb energy kernel shows diminishing parallel efficiency beyond 8 GPUs, although it remains among the least expensive components of the calculation. In contrast, the Coulomb gradient kernel exhibits better scaling, largely because only the gradient vector must be reduced among processes, rather than the full Coulomb matrix. As a result, communication overhead is reduced substantially. The Coulomb energy scaling limitations of the current $\mathbf{J}$ matrix build can be overcome by using NCCL^[44]-based reduction instead of MPI-based as suggested in Ref. [13] for the exact exchange matrix, or alternatively adopting a distributed Fock matrix calculation scheme. We note that we observed an unexpected performance drop for evaluating XC for Valinomycin on 4 GPUs. This behavior is reproducible and might be related to anomalies of the memory utilization in this setting.

A further factor limiting scaling efficiency is the computation of $\mathbf{V}^{-1}x$. Our current ap-

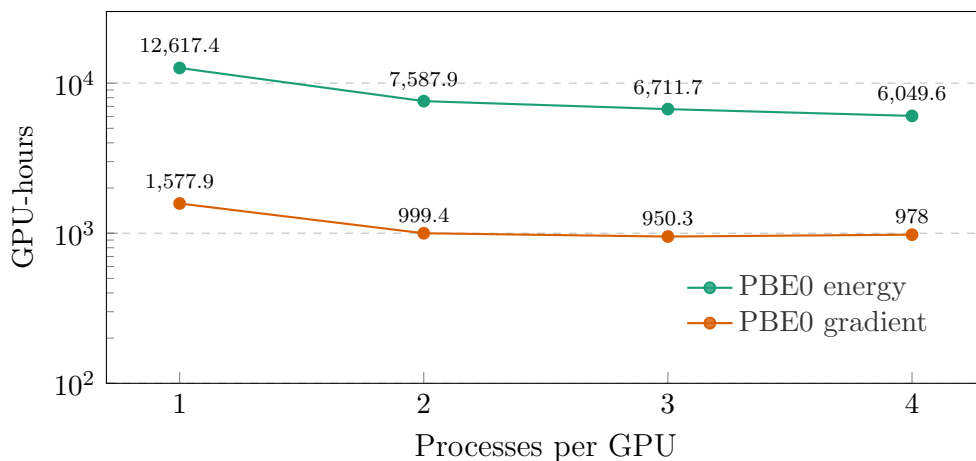


Figure 4: Estimated total GPU-hours required to process the entire OMol25-neutral dataset (34 335 828 structures) as a function of the number of concurrent processes per GPU. Results are shown separately for PBE0 single-point energy evaluations and nuclear gradient computations.

proach solves a linear system on the root process using precomputed Cholesky factors of the Coulomb metric. An alternative strategy is to employ iterative linear solvers, which require repeated $\sum_Q (P|Q) v_Q^i$ matrix-vector multiplications for trial vectors $\mathbf{v}^i$ until convergence. This approach offers two advantages: the contractions can be parallelized trivially across GPUs, and the method supports matrix-free implementations, reducing memory requirements. We expect this strategy to improve both scalability and memory efficiency in future versions of the code.

Conclusion

In this work, we implemented analytic two-electron energy derivatives for density-fitted Coulomb and semi-numerical exchange kernels as extensions to the LibintX and GauXC libraries, enabling efficient HF and hybrid DFT calculations on single- and multi-GPU architectures. We demonstrated the performance of the new implementation across small, medium, and large molecular systems and analyzed its scaling behavior with increasing system size. We conducted high-throughput benchmarks that emulate screening workloads

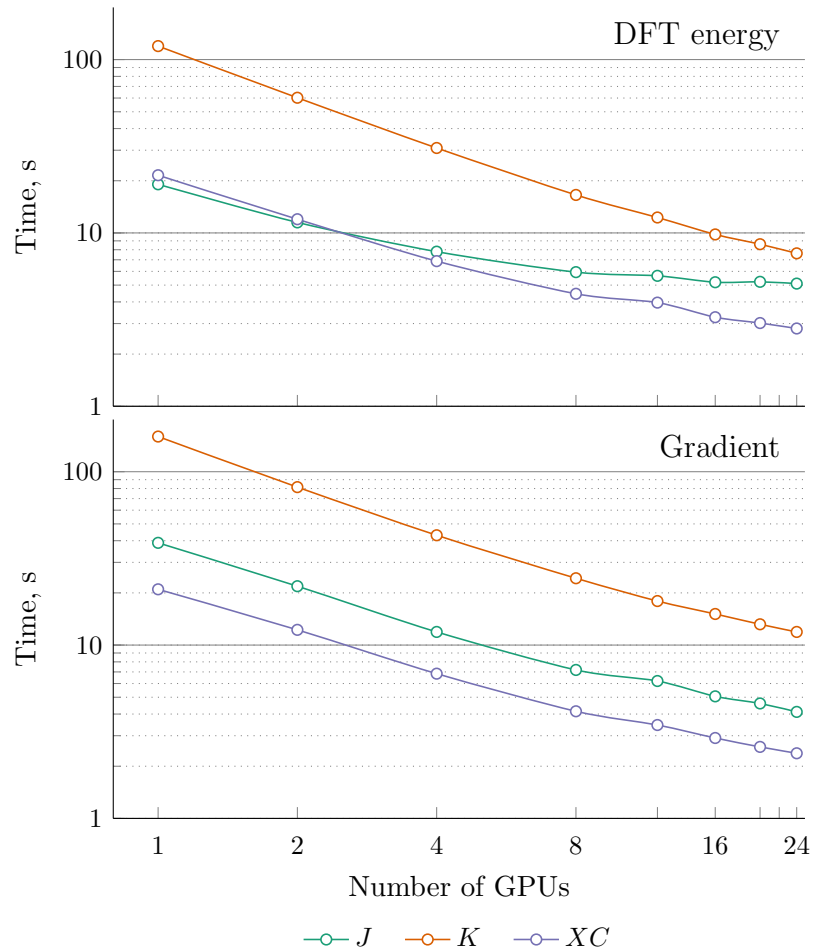


Figure 5: Scaling performance of two-electron computational kernels on multiple NVIDIA A100 GPUs for the **Ubiquitin** system: density-fitting Coulomb (J), semi-numerical exchange (K), and DFT exchange–correlation contribution (XC). Solid lines represent measured timings. The top panel contains timings for the energy kernels and the bottom panel contains for the gradient kernels.

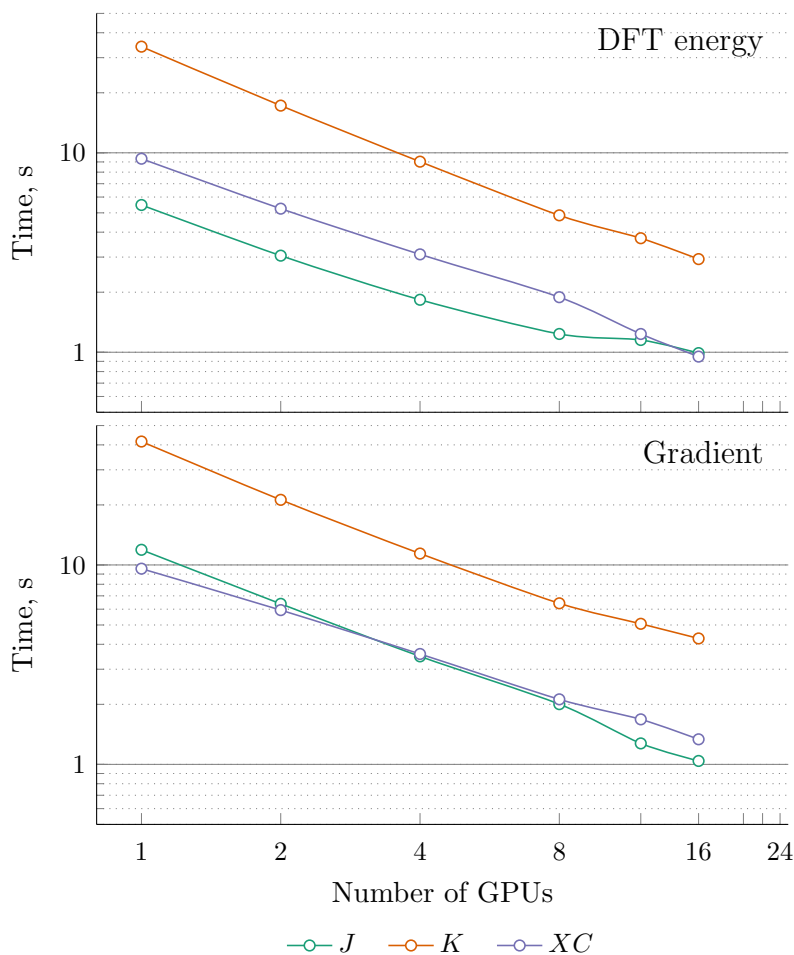


Figure 6: Scaling performance of two-electron computational kernels on multiple Nvidia A100 GPUs for the **Crambin** system: density-fitting Coulomb (J), semi-numerical exchange (K), and DFT exchange–correlation contribution (XC). Solid line represent measured timings. Top panel contains timings of energy kernels, bottom panel contains timings of gradient kernels.

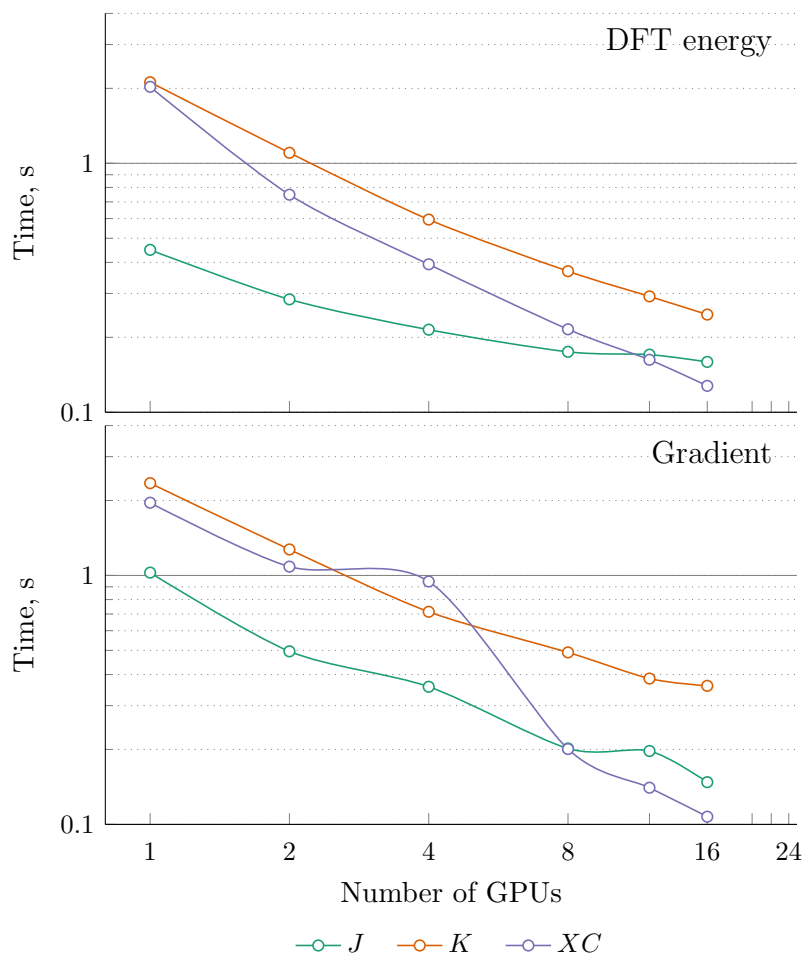


Figure 7: Scaling performance of two-electron computational kernels on multiple Nvidia A100 GPUs for the **Valinomycin** system: density-fitting Coulomb (J), semi-numerical exchange (K), and DFT exchange–correlation contribution (XC). Solid line represent measured timings. Top panel contains timings of energy kernels, bottom panel contains timings of gradient kernels.

via batched, concurrent jobs per GPU. The benchmark results show favorable scaling and strong parallel efficiency, which is maintained for up to 24 NVIDIA A100 GPUs, leading to substantial reductions in wall time for large-scale electronic-structure calculations.

Furthermore, we demonstrate that the implementation is well suited for high-throughput workloads representative of drug-discovery pipelines, including batched evaluations of large collections of small- and medium-sized molecules. Using timing data obtained for representative molecular subsets together with the empirical distribution of basis-set sizes, we estimate the computational cost of processing the OMol25-neutral benchmark. These estimates indicate that complete DFT single-point energies, molecular properties, and analytic gradients for the entire dataset can be computed within realistic wall times on modern GPU clusters, and at moderate cost even when using commercial cloud platforms. Overall, this work demonstrates that GPU-accelerated density-fitted Coulomb and semi-numerical exchange kernels with analytic gradients effectively bridge the gap between high-level electronic-structure accuracy and the throughput requirements of contemporary large-scale molecular screening, molecular dynamics and QM/MM simulations.

Acknowledgement

We thank Katerina Tsarova for her excellent organizational support and project coordination. We also thank Anya Schukin, Ben Orrell, Cedric Kring, Steven Beckers, Bernadette Behnke, Jeff Titus, and the entire Terra Quantum team for their valuable contributions to packaging and deploying the library, as well as for their support throughout this work.

Supporting Information Available

The Supporting Information provides details of the two-electron gradient algorithms and the molecular systems used in the benchmarks. The source code for the LibintX-based gradient implementation is available online at <https://github.com/terra-quantum-public/TQ-libintx>.

Installation instructions for the Python API via package managers are available online at <https://chemistry-docs.terraquantum.io/quantum-chemistry/index.html>.

References

- (1) Becke, A. D. Perspective: Fifty years of density-functional theory in chemical physics. *The Journal of Chemical Physics* **2014**, *140*.
- (2) Grimme, S.; Schreiner, P. R. Computational Chemistry: The Fate of Current Methods and Future Challenges. *Angewandte Chemie International Edition* **2018**, *57*, 4170–4176.
- (3) Akimov, A. V.; Prezhdo, O. V. Large-Scale Computations in Chemistry: A Bird’s Eye View of a Vibrant Field. *Chemical Reviews* **2015**, *115*, 5797–5890.
- (4) Pulay, P. *Modern Electronic Structure Part II*; 1995; pp 1191–1240.
- (5) Ufimtsev, I. S.; Martínez, T. J. Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-Electron Integral Evaluation. *Journal of Chemical Theory and Computation* **2008**, *4*, 222–231.
- (6) Ufimtsev, I. S.; Martinez, T. J. Quantum Chemistry on Graphical Processing Units. 2. Direct Self-Consistent-Field Implementation. *Journal of Chemical Theory and Computation* **2009**, *5*, 1004–1015.
- (7) Miao, Y.; Merz, K. M. Acceleration of Electron Repulsion Integral Evaluation on Graphics Processing Units via Use of Recurrence Relations. *Journal of Chemical Theory and Computation* **2013**, *9*, 965–976.
- (8) Miao, Y.; Merz, K. M. Acceleration of High Angular Momentum Electron Repulsion Integrals and Integral Derivatives on Graphics Processing Units. *Journal of Chemical Theory and Computation* **2015**, *11*, 1449–1462.

- (9) Li, X.; Linares, M.; Norman, P. VeloxChem: GPU-Accelerated Fock Matrix Construction Enabling Complex Polarization Propagator Simulations of Circular Dichroism Spectra of G-Quadruplexes. *The Journal of Physical Chemistry A* **2025**, *129*, 633–642.
- (10) Wu, X.; Sun, Q.; Pu, Z.; Zheng, T.; Ma, W.; Yan, W.; Xia, Y.; Wu, Z.; Huo, M.; Li, X.; Ren, W.; Gong, S.; Zhang, Y.; Gao, W. Enhancing GPU-Acceleration in the Python-Based Simulations of Chemistry Frameworks. *WIREs Computational Molecular Science* **2025**, *15*.
- (11) Seritan, S.; Bannwarth, C.; Fales, B. S.; Hohenstein, E. G.; Isborn, C. M.; Kokkila-Schumacher, S. I. L.; Li, X.; Liu, F.; Luehr, N.; Snyder, J. W.; Song, C.; Titov, A. V.; Ufimtsev, I. S.; Wang, L.; Martínez, T. J. TeraChem : A graphical processing unit-accelerated electronic structure package for large-scale ab initio molecular dynamics. *WIREs Computational Molecular Science* **2021**, *11*.
- (12) Laqua, H.; Thompson, T. H.; Kussmann, J.; Ochsenfeld, C. Highly Efficient, Linear-Scaling Seminumerical Exact-Exchange Method for Graphic Processing Units. *Journal of Chemical Theory and Computation* **2020**, *16*, 1456–1468, PMID: 32053375.
- (13) Williams-Young, D. B.; Asadchev, A.; Popovici, D. T.; Clark, D.; Waldrop, J.; Windus, T. L.; Valeev, E. F.; de Jong, W. A. Distributed memory, GPU accelerated Fock construction for hybrid, Gaussian basis density functional theory. *The Journal of Chemical Physics* **2023**, *158*.
- (14) Stocks, R.; Palethorpe, E.; Barca, G. M. J. Multi-GPU RI-HF Energies and Analytic Gradients – Toward High-Throughput Ab Initio Molecular Dynamics. *Journal of Chemical Theory and Computation* **2024**, *20*, 7503–7515.
- (15) Vahtras, O.; Almlöf, J.; Feyereisen, M. Integral approximations for LCAO-SCF calculations. *Chemical Physics Letters* **1993**, *213*, 514–518.

- (16) Aquilante, F.; Boman, L.; Boström, J.; Koch, H.; Lindh, R.; de Merás, A. S.; Peder-
sen, T. B. *Linear-Scaling Techniques in Computational Chemistry and Physics: Meth-
ods and Applications*; Springer Netherlands, 2011; pp 301–343.
- (17) Kussmann, J.; Ochsenfeld, C. Pre-selective screening for matrix elements in linear-
scaling exact exchange calculations. *The Journal of Chemical Physics* **2013**, *138*.
- (18) Laqua, H.; Kussmann, J.; Ochsenfeld, C. Efficient and Linear-Scaling Seminumerical
Method for Local Hybrid Density Functionals. *Journal of Chemical Theory and Com-
putation* **2018**, *14*, 3451–3458.
- (19) Neese, F.; Wennmohs, F.; Hansen, A.; Becker, U. Efficient, approximate and paral-
lel Hartree-Fock and hybrid DFT calculations. A 'chain-of-spheres' algorithm for the
Hartree-Fock exchange. *Chemical Physics* **2009**, *356*, 98–109.
- (20) Palethorpe, E.; Barca, G. M. J. High-Performance, High-Angular-Momentum J Engine
on Graphics Processing Units. *Journal of Chemical Theory and Computation* **2025**, *21*,
9388–9403.
- (21) 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. *The Journal of
Chemical Physics* **2023**, *158*.
- (22) Weigend, F. A fully direct RI-HF algorithm: Implementation, optimised auxiliary basis
sets, demonstration of accuracy and efficiency. *Physical Chemistry Chemical Physics*
2002, *4*, 4285–4291.
- (23) Reza Ahmadi, G.; Almlöf, J. The Coulomb operator in a Gaussian product basis.
Chemical Physics Letters **1995**, *246*, 364–370.

- (24) White, C. A.; Head-Gordon, M. A J matrix engine for density functional theory calculations. *The Journal of Chemical Physics* **1996**, *104*, 2620–2629.
- (25) Neese, F. An improvement of the resolution of the identity approximation for the formation of the Coulomb matrix. *Journal of Computational Chemistry* **2003**, *24*, 1740–1747.
- (26) Friesner, R. A. Solution of self-consistent field electronic structure equations by a pseudospectral method. *Chemical Physics Letters* **1985**, *116*, 39–43.
- (27) Martinez, T. J.; Carter, E. A. *Modern Electronic Structure Theory*; World Scientific Publishing, 1995; Chapter 17, pp 1132–1165.
- (28) Hohenstein, E. G.; Parrish, R. M.; Martínez, T. J. Tensor hypercontraction density fitting. I. Quartic scaling second- and third-order Møller-Plesset perturbation theory. *The Journal of Chemical Physics* **2012**, *137*.
- (29) Asadchev, A.; Valeev, E. F. Memory-Efficient Recursive Evaluation of 3-Center Gaussian Integrals. *Journal of Chemical Theory and Computation* **2023**, *19*, 1698–1710.
- (30) Bykov, D.; Petrenko, T.; Izsák, R.; Kossmann, S.; Becker, U.; Valeev, E.; Neese, F. Efficient implementation of the analytic second derivatives of Hartree-Fock and hybrid DFT energies: A detailed analysis of different approximations. *Molecular Physics* **2015**, *113*, 1961–1977.
- (31) Helgaker, T.; Taylor, P. R. On the evaluation of derivatives of Gaussian integrals. *Theoretica Chimica Acta* **1992**, *83*, 177–183.
- (32) Reine, S.; Tellgren, E.; Helgaker, T. A unified scheme for the calculation of differentiated and undifferentiated molecular integrals over solid-harmonic Gaussians. *Physical chemistry chemical physics : PCCP* **2007**, *9*, 4771–4779.
- (33) Flocke, N.; Lotrich, V. Efficient electronic integrals and their generalized derivatives for

- object oriented implementations of electronic structure calculations. *Journal of Computational Chemistry* **2008**,
- (34) Sun, Q. et al. Recent developments in the PySCF program package. *The Journal of Chemical Physics* **2020**, 153.
- (35) Asadchev, A.; Valeev, E. F. 3-center and 4-center 2-particle Gaussian AO integrals on modern accelerated processors. *The Journal of Chemical Physics* **2024**, 160.
- (36) LibintX. Accessed 2025-12-18. <https://github.com/ValeevGroup/libintx>.
- (37) Williams-Young, D. B.; de Jong, W. A.; van Dam, H. J. J.; Yang, C. On the Efficient Evaluation of the Exchange Correlation Potential on Graphics Processing Unit Clusters. *Frontiers in Chemistry* **2020**, 8.
- (38) GauXC. Accessed 2025-12-18. <https://github.com/wavefunction91/GauXC>.
- (39) Pulay, P. Convergence acceleration of iterative sequences. The case of SCF iteration. *Chemical Physics Letters* **1980**, 73, 393–398.
- (40) Sun, Q. Libcint: An efficient general integral library for Gaussian basis functions. *Journal of Computational Chemistry* **2015**, 36, 1664–1671.
- (41) Levine, D. S. et al. The Open Molecules 2025 (OMol25) Dataset, Evaluations, and Models. 2025; <https://arxiv.org/abs/2505.08762>.
- (42) Dunning Jr, T. H. Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen. *The Journal of chemical physics* **1989**, 90, 1007–1023.
- (43) Adamo, C.; Cossi, M.; Barone, V. An accurate density functional method for the study of magnetic properties: the PBE0 model. *Journal of Molecular Structure: THEOCHEM* **1999**, 493, 145–157.

- (44) NCCL: Optimized primitives for inter-GPU communication. <https://github.com/NVIDIA/nvcc1>.