

Article

GANSU: A GPU-Native Quantum Chemistry Framework for Efficient Hartree–Fock and Post-HF Calculations

Yasuaki Ito ^{1,*} , Satoki Tsuji ^{1,2} , Koji Nakano ¹  and Akihiko Kasagi ²¹ School of Advanced Science and Engineering, Hiroshima University, Higashi-Hiroshima 739-8527, Japan; tsuji.satoki@fujitsu.com (S.T.); nakano@cs.hiroshima-u.ac.jp (K.N.)² Fujitsu Limited, Kawasaki 211-8588, Japan; kasagi.akihiko@fujitsu.com

* Correspondence: yasuaki@cs.hiroshima-u.ac.jp

Abstract

GPU-accelerated quantum chemistry programs can dramatically reduce the time required for electronic structure calculations, yet most existing implementations either retrofit GPU kernels onto legacy CPU codebases or optimize individual kernels without addressing workflow-level integration overhead. We present GANSU (GPU Accelerated Numerical Simulation Utility), an open-source quantum chemistry framework written entirely in CUDA/C++ that integrates GPU-accelerated kernels for electron repulsion integrals, Fock matrix construction, and post-Hartree–Fock (post-HF) methods into a unified, GPU-resident execution pipeline. The key design principle is to eliminate host–device data transfers between computational stages by keeping all intermediate data, including density matrices, integral buffers, and Fock matrix replicas, on the GPU throughout the self-consistent field (SCF) iteration, combined with runtime-selectable integral strategies (stored ERI, resolution-of-the-identity, and Direct-SCF) that adapt to system size and available memory. On an NVIDIA H200 GPU, GANSU achieves end-to-end speedups of up to $52\times$ over PySCF for SCF, $45\times$ for MP2 on molecules with up to 470 basis functions, and $44\times$ for FCI, while outperforming GPU4PySCF by up to $34\times$ for FCI, across a range of molecular systems with up to 650 basis functions. The framework further provides analytical energy gradients and geometry optimization with nine algorithms, all operating within the same GPU-resident data flow. These results demonstrate that workflow-aware kernel integration, not just kernel-level optimization, is essential for realizing the full potential of GPU acceleration in scientific computing. GANSU is publicly available under the BSD-3-Clause license.

Keywords: GPU acceleration; quantum chemistry; Hartree–Fock; CUDA; resolution of identity; post-Hartree–Fock



Academic Editor: Antonio Gil Bravo

Received: 27 March 2026

Revised: 25 April 2026

Accepted: 26 April 2026

Published: 28 April 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The rapid growth of high-performance computing (HPC) systems has fundamentally changed the way large-scale scientific simulations are designed and executed. In particular, modern GPU accelerators provide massive thread-level parallelism, high memory bandwidth, and specialized hardware units, which together enable large performance gains for data-parallel workloads. Achieving such performance, however, requires not only optimized computational kernels but also the careful coordination of kernel execution, memory usage, and data movement at the application level.

Electronic structure calculations in computational quantum chemistry present a representative example of such challenges. Hartree–Fock (HF) and related self-consistent

field (SCF) methods consist of multiple computational stages, including electron repulsion integral (ERI) evaluation, Fock matrix construction, and iterative diagonalization and convergence control [1]. While each stage exhibits a high degree of parallelism, overall performance is determined by how efficiently these components are orchestrated within the SCF workflow. Consequently, HF calculations provide an ideal testbed for studying end-to-end GPU acceleration in complex scientific applications.

In recent years, machine learning surrogate models have emerged as a powerful approach for accelerating molecular property prediction in fields such as materials science and drug discovery [2,3]. Training such models requires large-scale datasets of quantum chemical energies, forces, and other electronic properties for diverse molecular configurations. Because the cost of generating these datasets is dominated by the underlying *ab initio* calculations, the throughput of the quantum chemistry engine directly limits the scale and diversity of the training data. High-performance GPU-accelerated frameworks are therefore essential not only for individual calculations but also as data generation engines for AI-driven molecular design pipelines.

A significant body of prior work has demonstrated that individual quantum chemistry kernels can be efficiently accelerated on GPUs. Examples include GPU implementations of ERI evaluation based on the Head–Gordon–Pople (HGP) algorithm [4] and the McMurchie–Davidson (MD) method [5,6], three-center ERI evaluation for resolution-of-the-identity (RI) approximations [7], dynamic Schwarz screening of negligible integrals [8], Boys function computation [9,10], one-electron integral evaluation including nuclear attraction integrals [11,12], and Fock matrix construction with efficient reduction strategies [13]. These kernel-level optimizations have achieved impressive speedups and constitute an essential foundation for GPU-accelerated quantum chemistry. However, optimizing isolated kernels is only a first step; practical performance depends critically on how these kernels are integrated into a complete application workflow.

In realistic SCF calculations, kernel orchestration introduces additional challenges, such as synchronization overheads, intermediate data storage, memory reuse, and load imbalance across different stages of the computation. Inefficient coordination can easily negate the benefits of highly optimized kernels, particularly on modern GPUs where memory hierarchy and execution concurrency play a dominant role. Despite this, relatively few studies have examined in detail the impact of kernel integration and workflow-level design on the overall performance of GPU-accelerated quantum chemistry codes.

In a series of previous studies, we developed multiple GPU-optimized computational kernels relevant to HF and post-HF calculations. For two-electron repulsion integrals, we implemented GPU-accelerated kernels based on the HGP algorithm with warp-based thread assignment [4] and the MD method with multiple CUDA mapping strategies [5,6]. We also developed dynamic Schwarz screening techniques that exploit asynchronous CUDA block scheduling to eliminate negligible integral contributions [8], and efficient three-center ERI kernels for RI-based formulations [7]. For the Boys function, a fundamental numerical primitive required by all integral algorithms, we proposed a bulk evaluation method using a gridded Taylor expansion with GPU-optimized lookup tables [9,10]. For one-electron integrals, we developed a hybrid Obara–Saika/McMurchie–Davidson approach with CUDA stream-based concurrent execution [11,12]. For Fock matrix construction, we introduced a distributed atomic reduction (DAR) method based on replicated Fock matrices to mitigate memory contention [13]. We also extended our GPU acceleration efforts to post-HF methods, including Møller–Plesset perturbation theory at the MP2 and MP3 levels [14], RI-based restricted MP2 (RI-RMP2) [15], and full configuration interaction (FCI) [16]. These works focused primarily on kernel-level performance and demonstrated speedups ranging from $3.75\times$ (Fock matrix DAR) to $4500\times$ (ERI evaluation) through careful

parallelization and algorithmic reformulation. Building on these results, the next logical step from an HPC standpoint is to integrate these kernels into a unified software framework and to evaluate their collective performance within a full SCF workflow.

We present GANSU, an integrated GPU-native quantum chemistry framework designed from the outset with HPC considerations in mind. Rather than introducing new quantum chemical methods, our contribution lies in the systematic integration and orchestration of the aforementioned GPU-accelerated kernels within an end-to-end SCF execution model. GANSU adopts a GPU-first design philosophy, explicit control over memory allocation and reuse, and modular kernel interfaces, enabling the flexible composition of computational components while minimizing integration overhead. The entire framework is implemented in CUDA [17,18] and C++ from scratch, without reliance on legacy Fortran codebases, and is publicly available as open-source software under the BSD-3-Clause license [19].

The main contributions of this work are as follows:

- We design and implement GANSU, an open-source GPU-native quantum chemistry framework that integrates multiple independently developed kernels into a unified SCF workflow with explicit memory management and minimal host–device data transfer.
- We demonstrate end-to-end SCF performance of up to $52\times$ faster than PySCF (CPU) and competitive with GPU4PySCF, scaling from small molecules (25 basis functions) to cholesterol (650 basis functions) on an NVIDIA H200 GPU.
- We integrate post-HF methods (MP2, MP3, RI-RMP2, FCI) into the same GPU-resident data flow, achieving end-to-end speedups of up to $45\times$ over PySCF for MP2 and $44\times$ for FCI, and up to $34\times$ over GPU4PySCF for FCI.
- We provide GPU-accelerated analytical energy gradients and geometry optimization with nine optimizer algorithms, enabling potential energy surface exploration entirely on the GPU.
- We analyze the performance breakdown of the SCF workflow, quantifying how ERI evaluation, Fock matrix construction, and other components scale with system size, providing insights into GPU application design beyond quantum chemistry.

The remainder of this paper is organized as follows. Section 2 reviews related quantum chemistry software and positions GANSU in context. Section 3 describes the overall architecture and design principles of the GANSU framework. Section 4 provides a brief overview of Hartree–Fock theory. Section 5 outlines the implemented computational kernels from a parallel computing perspective. Section 6 focuses on the integration of these kernels into an end-to-end SCF workflow. Section 7 describes the post-Hartree–Fock capabilities of GANSU. Section 8 presents the analytical energy gradient implementation and geometry optimization. Section 9 presents performance evaluations at both the kernel and application levels. Finally, Section 10 concludes the paper and discusses directions for future work.

2. Related Work

A number of quantum chemistry software packages are widely used in the community, and several have incorporated GPU acceleration to varying degrees. In this section, we review representative packages and clarify how GANSU differs from existing approaches.

2.1. CPU-Based Quantum Chemistry Packages

Gaussian [20] is one of the most widely used commercial quantum chemistry packages, supporting a broad range of electronic structure methods. Gaussian 16 includes limited GPU support for certain DFT calculations, but its core architecture remains CPU-

oriented. GAMESS [21] is a well-established open-source package that has introduced GPU-accelerated Fock matrix construction through the LibCChem library [22,23], though its overall framework is built around legacy Fortran code. NWChem [24] provides massively parallel CPU execution with limited GPU offloading through the Tensor Contraction Engine. Psi4 [25] is an open-source package emphasizing modularity and interoperability, primarily CPU-based with limited GPU support through external plugins. Q-Chem [26] is a commercial package offering GPU acceleration via the BrianQC plugin [27], which uses compiler-based code generation to accelerate two-electron integral evaluation.

PySCF [28,29] is a Python-based open-source framework that has gained widespread adoption due to its modularity and ease of extension. While PySCF itself is CPU-based, the GPU4PySCF extension [30] provides NVIDIA GPU acceleration for DFT and HF calculations using the Rys quadrature algorithm [31] for integral evaluation. GPU4PySCF achieves significant speedups for density-fitted DFT calculations and supports analytical gradients on GPUs.

2.2. GPU-Accelerated Quantum Chemistry Packages

TeraChem [32] is a pioneering GPU-native quantum chemistry package developed by Ufimtsev and Martinez [33,34]. It was designed from the ground up for GPU execution and supports HF, DFT, TDDFT, CASSCF, and ab initio molecular dynamics. TeraChem demonstrated the viability of GPU-first quantum chemistry and remains one of the most mature GPU-native codes available. However, it is a proprietary commercial package, and its integral evaluation kernels are based on algorithms optimized for earlier GPU architectures.

QUICK [35,36] is an open-source GPU-accelerated package supporting HF and DFT calculations on both NVIDIA (CUDA) and AMD (HIP) GPUs, with multi-GPU capabilities. It focuses primarily on DFT and QM/MM calculations and integrates with the AMBER molecular dynamics package. FermiONS++ [37] employs a hybrid CPU/GPU integral engine with support for linear-scaling methods, targeting both NVIDIA and AMD GPUs via OpenCL.

Early work on GPU-accelerated molecular integrals includes the recurrence relation-based approach of Miao and Merz [38] and the GPU cluster implementation by Asadchev et al. [22] within GAMESS. These works demonstrated the potential of GPUs for integral evaluation but focused primarily on specific kernels rather than end-to-end framework integration.

2.3. Positioning of GANSU

GANSU differs from the existing packages in several key aspects. First, unlike packages that retrofit GPU acceleration onto existing CPU-oriented codebases (e.g., Gaussian, GAMESS, NWChem, and PySCF/GPU4PySCF), GANSU is implemented entirely in CUDA and C++ from scratch, with a GPU-first design philosophy that avoids legacy constraints. This approach enables fine-grained control over memory management, kernel orchestration, and data flow that is difficult to achieve in retrofitted architectures.

Second, while TeraChem shares the GPU-native design philosophy, it is a commercial closed-source package. GANSU is released under the BSD-3-Clause open-source license [19], enabling community access, modification, and extension. GANSU also incorporates multiple ERI algorithms (HGP [4], MD [5], and Rys quadrature [31]), allowing flexible algorithm selection depending on the molecular system and basis set characteristics. In contrast, most existing packages rely on a single ERI algorithm.

Third, GANSU provides a comprehensive set of GPU-accelerated kernels that spans the entire HF workflow, from Boys function evaluation [10] through one-electron integrals [12], two-electron integrals [4,5,7], integral screening [8], and Fock matrix construction [13], as

well as post-HF methods including MP2, MP3 [14], RI-RMP2 [15], and FCI [16]. Each kernel has been independently developed, benchmarked, and published, and the present work focuses on their integration into a unified workflow.

Fourth, from an HPC perspective, GANSU emphasizes the study of kernel integration overhead and workflow-level optimization, which has received relatively little attention in the literature compared to individual kernel optimization. By treating kernel orchestration, memory reuse, and data locality as first-class design concerns, GANSU provides both a practical quantum chemistry tool and a platform for investigating GPU application design principles.

3. Overview of the GANSU Framework

This section provides an overview of the GANSU framework, focusing on its design philosophy, software architecture, and execution model from a high-performance computing perspective. GANSU is designed as a GPU-native quantum chemistry framework that integrates multiple optimized computational kernels into a unified self-consistent field (SCF) workflow.

3.1. Design Philosophy

The design of GANSU is guided by three core principles: GPU-first execution, explicit resource management, and modular kernel integration. First, GANSU adopts a GPU-first execution model in which the most computationally intensive components of Hartree–Fock calculations are executed on GPUs by default. Rather than treating the GPU as an accelerator invoked selectively, GANSU is designed such that GPU execution is the primary mode, and CPU resources are used mainly for control logic and lightweight tasks.

Second, GANSU emphasizes explicit control over memory allocation, data layout, and memory reuse. On modern GPU architectures, memory hierarchy and data movement play a critical role in overall performance. To minimize unnecessary data transfers and allocation overheads, GANSU manages intermediate data structures explicitly and reuses GPU-resident buffers across multiple stages of the SCF iteration whenever possible. This design choice is particularly important for integrating multiple kernels that operate on shared data structures, such as density matrices and intermediate integral buffers.

Third, GANSU is built around modular kernel integration. Individual computational kernels, including electron repulsion integral evaluation [4,5], Boys function computation [10], one-electron integrals [11,12], and Fock matrix construction [13], are implemented as independent modules with well-defined interfaces. This modularity allows different kernel implementations or algorithms to be combined flexibly while maintaining a consistent execution model. This approach facilitates kernel orchestration and makes it possible to measure integration overheads directly.

3.2. Strategy-Based Software Design

A key architectural feature of GANSU is the systematic use of the Strategy design pattern [39] to achieve runtime flexibility without sacrificing performance. In quantum chemistry applications, the choice of computational algorithm depends on multiple factors, including the electronic structure of the molecule, the size of the system, and the available GPU memory. By encapsulating each algorithmic variant behind a common abstract interface and selecting the concrete implementation at runtime, GANSU enables the flexible composition of computational strategies while preserving a clean separation of concerns.

GANSU applies the Strategy pattern to four independent axes of algorithmic variation:

3.2.1. Hartree–Fock Variant Strategy

The abstract base class HF defines pure virtual methods for the core SCF operations, including Fock matrix construction, density matrix computation, energy evaluation, and convergence checking. Three concrete strategies, RHF, UHF, and ROHF, implement these methods according to the specific mathematical formulations of restricted, unrestricted, and restricted open-shell Hartree–Fock theory, respectively. This design ensures that the SCF control layer can operate uniformly on any HF variant through a single polymorphic interface.

3.2.2. ERI Computation Strategy

The abstract base class ERI defines the interface for electron repulsion integral evaluation, including methods for precomputation, Fock matrix construction, and post-HF energy computation. Four concrete strategies are provided:

- ERI_Stored: Precomputes and stores all ERIs in GPU memory for small molecules;
- ERI_RI: Employs the resolution-of-the-identity approximation with three-center integrals;
- ERI_Direct: Recomputes ERIs on-the-fly with Schwarz screening for large systems;
- ERI_RI_Direct: Combines the RI approximation with direct integral computation.

Each ERI strategy is further specialized for the HF variant (e.g., ERI_Stored_RHF and ERI_RI_UHF), forming a two-dimensional strategy matrix that for the allows independent selection of the HF method and the integral computation approach.

3.2.3. Convergence Acceleration Strategy

The SCF convergence procedure is encapsulated in a separate strategy hierarchy with two concrete implementations: constant or optimized damping and DIIS (direct inversion in the iterative subspace) [40,41]. Each HF variant maintains its own convergence strategy object, selected at runtime based on user configuration.

3.2.4. Initial Guess Strategy

The generation of the initial density matrix is handled by a dedicated strategy hierarchy supporting four methods: *core* Hamiltonian diagonalization, generalized Wolfsberg–Helmholtz (GWH) approximation, density matrix input from a previous calculation, and the superposition of atomic densities (SAD). This separation allows new initial guess methods to be added without modifying the SCF workflow.

3.2.5. Geometry Optimization Strategy

For gradient-based geometry optimization (Section 8), the abstract base class GeometryOptimizer defines the interface for search direction computation and Hessian updates, with nine concrete strategies: three quasi-Newton methods (BFGS, DFP, and SR1), four conjugate gradient variants (Fletcher–Reeves, Polak–Ribière, Hestenes–Stiefel, and Dai–Yuan), steepest descent, and GDIIS (geometry DIIS) [42]. A factory method selects the optimizer at the runtime.

All strategy objects are managed through `std::unique_ptr` smart pointers and instantiated by a HFBUILDER factory class based on input parameters. This design ensures deterministic resource management and enables clean runtime composition of independent algorithmic choices. By treating computational method selection as a first-class design concern, GANSU achieves a level of configurability comparable to established quantum chemistry packages while maintaining a lean, GPU-optimized codebase.

3.3. Software Architecture

Figure 1 illustrates the layered architecture of the GANSU framework. The layers are organized from low-level numerical primitives at the bottom to high-level analysis and output at the top, reflecting the data dependency structure: each layer consumes the results produced by the layers below it.

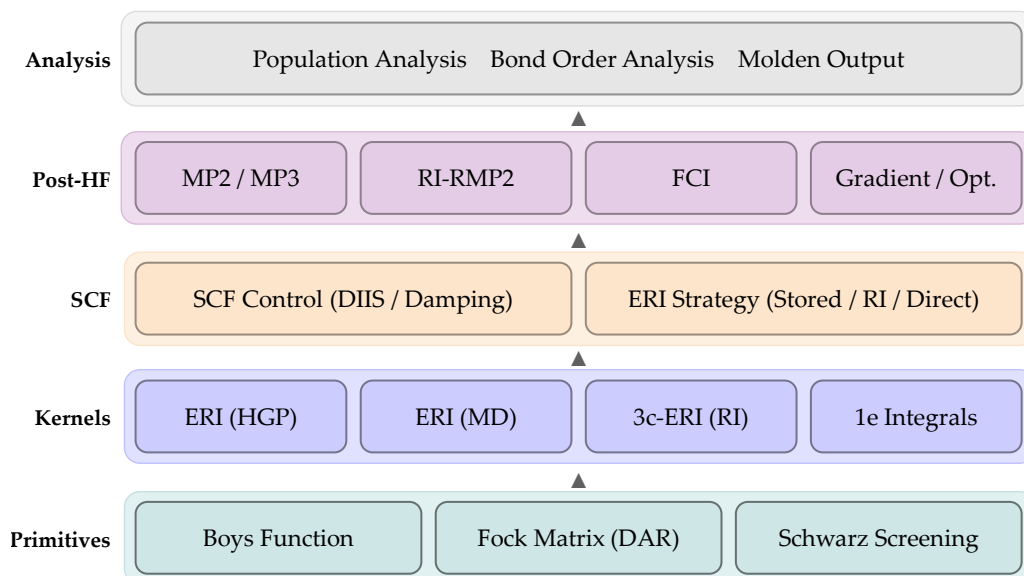


Figure 1. Architecture of the GANSU framework. Layers from bottom to top: numerical primitives, integral kernels, SCF control and ERI strategy selection, post-HF methods and gradient/optimization, and analysis/output.

The integral kernel layer includes GPU implementations of four-center ERIs via the HGP [4], MD [5], and Rys quadrature [31] algorithms, three-center ERIs for RI approximations [7], one-electron integrals including overlap, kinetic energy, and nuclear attraction integrals [11,12], and Schwarz-based integral screening [8]. These kernels constitute the dominant computational workload and are designed to exploit fine-grained parallelism on GPUs.

The numerical primitive layer provides supporting components such as Boys function evaluation [10] and the reduction operations required for Fock matrix construction [13]. These primitives are shared across multiple higher-level kernels and are optimized for bulk execution on GPUs. By centralizing such primitives, GANSU avoids redundant implementations and enables consistent performance optimization across the framework.

The SCF control layer coordinates the execution of computational kernels within each SCF iteration. This layer is responsible for invoking kernel execution, managing data dependencies, and controlling convergence procedures such as damping and direct inversion in the iterative subspace (DIIS). While the control logic itself is executed on the CPU, it is designed to minimize synchronization points and to maximize the amount of work performed on the GPU between host–device interactions.

The post-HF module layer provides GPU-accelerated implementations of electron correlation methods, including conventional MP2 and MP3 [14], RI-RMP2 [15], and full configuration interaction [16]. These modules consume the molecular orbital coefficients, orbital energies, and integral data produced by the SCF workflow, and are designed to operate entirely within the GPU-resident data flow established by the preceding HF calculation.

Finally, the analysis and output layer handles post-processing tasks such as population analysis and data export. These components are intentionally separated from the

performance-critical path of the SCF workflow, ensuring that additional analysis does not interfere with the main computational kernels.

3.4. Execution Model and Data Flow

The execution model of GANSU follows an iterative SCF workflow in which multiple GPU kernels are executed repeatedly until convergence is achieved. Within a single SCF iteration, integral evaluation, Fock matrix construction, and matrix operations are performed in a coordinated manner. Intermediate data, such as density matrices and integral buffers, are retained on the GPU across kernel invocations to reduce data movement overhead.

As described in Section 3.2, GANSU provides multiple ERI computation strategies that can be selected at runtime according to the system size and available GPU memory. The ERI_Stored strategy stores the full ERI tensor in GPU memory for small molecules, avoiding repeated integral evaluation across SCF iterations. The ERI_RI strategy employs the resolution-of-the-identity approximation [7,43] to reduce both memory requirements and computational cost. The ERI_Direct strategy recomputes integrals on-the-fly in each iteration, combined with Schwarz screening [8,44] to eliminate negligible contributions. The ERI_RI_Direct strategy combines the RI approximation with direct computation for systems where neither approach alone is optimal.

GANSU exposes parallelism at multiple levels. Fine-grained parallelism is exploited within individual kernels using GPU threads and thread blocks, while coarse-grained parallelism arises from batching independent tasks such as shell-pair or auxiliary basis processing. Although the current implementation focuses on single-GPU execution, the execution model is designed to be extensible to multi-GPU and distributed-memory environments.

3.5. Implementation and Availability

GANSU is implemented entirely in CUDA and C++17, with all computational kernels written from scratch to fully exploit GPU architectures without inheriting constraints from legacy CPU-oriented codebases. The framework depends on the NVIDIA CUDA Toolkit (version 12.9 or later), cuBLAS and cuSOLVER for GPU-accelerated linear algebra, and CMake (version 3.31 or later) for build configuration. It targets GPUs with CUDA Compute Capability 8.0 or higher.

The integral kernels support angular momentum up to i -orbitals ($l = 6$), enabling the use of large basis sets including correlation-consistent and ANO-type basis sets. GANSU supports Hartree–Fock variants (RHF, UHF, and ROHF) and post-HF methods (MP2, MP3, RI-RMP2, and FCI), analytical energy gradients for RHF and UHF, and geometry optimization with nine algorithms (BFGS, DFP, SR1, GDIIS, four conjugate gradient variants, and steepest descent), as well as population analysis, bond order analysis, and Molden format output for orbital visualization.

GANSU is publicly available as open-source software under the BSD-3-Clause license [19]. The repository includes the source code, build instructions, and example input files.

By explicitly addressing kernel orchestration, memory reuse, and workflow-level integration, the GANSU framework provides a foundation for the efficient end-to-end GPU acceleration of Hartree–Fock and post-HF calculations. The following sections describe the implemented computational kernels and their integration into the SCF workflow in more detail.

4. Preliminaries

This section provides a brief overview of the basic concepts and terminology of Hartree–Fock (HF) theory [45,46] required to understand the subsequent discussion. A

more comprehensive introduction aimed at computer scientists can be found in [1]. The goal is not to present a detailed theoretical treatment, but to clarify the computational objects, representations, and variants of HF calculations considered in this work.

4.1. Atomic and Molecular Orbitals

In molecular electronic structure calculations, the electronic wave function is represented using molecular orbitals (MOs). Each molecular orbital is expressed as a linear combination of atomic orbitals (AOs), which are typically chosen as Gaussian-type basis functions [47] in practical calculations. In this representation, atomic orbitals serve as the fundamental computational units for evaluating integrals, while molecular orbitals are obtained by solving an eigenvalue problem within the atomic orbital basis.

In practice, most operations in HF calculations are performed in the atomic orbital representation. Electron repulsion integrals, one-electron integrals, and Fock matrix elements are all defined with respect to pairs or quadruples of atomic orbitals. Molecular orbitals are constructed repeatedly during the self-consistent field (SCF) procedure by diagonalizing matrices defined in the atomic orbital basis.

4.2. Closed-Shell and Open-Shell Systems

HF calculations can be broadly classified according to the electronic structure of the system. In closed-shell systems, all occupied molecular orbitals are doubly occupied by electrons with opposite spins. In open-shell systems, one or more molecular orbitals are singly occupied, leading to unpaired electrons and spin polarization. Open-shell systems can be further distinguished by how spin symmetry is treated: in unrestricted treatments, separate spatial orbitals are used for spin-up and spin-down electrons, allowing complete flexibility at the cost of spin contamination; in restricted open-shell treatments, doubly occupied orbitals share a common spatial description while singly occupied orbitals are treated separately, preserving spin symmetry.

This distinction has important computational consequences. Closed-shell calculations involve a single density matrix and a single Fock matrix. Unrestricted open-shell calculations require two independent sets of density and Fock matrices for the alpha and beta spin components. Restricted open-shell calculations introduce an intermediate level of complexity, with separate density matrices for closed-shell and open-shell orbital spaces and a modified Fock operator that couples these spaces. As a result, GANSU provides three HF variants (RHF, UHF, and ROHF), each implemented as a distinct strategy (see Section 3.2) to accommodate these different computational requirements.

4.3. Hartree–Fock Variants

Several variants of HF theory are commonly used in practice to handle different types of electronic structures. In restricted Hartree–Fock (RHF) [48], the same spatial molecular orbitals are used for both spin-up and spin-down electrons, making RHF suitable for closed-shell systems. In unrestricted Hartree–Fock (UHF) [49], separate spatial orbitals are introduced for different spin components, allowing flexible treatment of open-shell systems at the cost of potential spin contamination.

Restricted open-shell Hartree–Fock (ROHF) provides an alternative formulation for open-shell systems, in which doubly occupied and singly occupied orbitals are treated separately while maintaining a common spatial description for paired electrons. Although ROHF introduces additional complexity in the formulation of the Fock operator, it preserves spin symmetry and is widely used in quantum chemistry.

In GANSU, RHF, UHF, and ROHF calculations are supported within a unified SCF framework. While the mathematical expressions for the Fock matrix differ among these

variants, they share a common computational structure consisting of integral evaluation, Fock matrix construction, matrix diagonalization, and density matrix update.

4.4. Self-Consistent Field Procedure

HF calculations are carried out using an iterative SCF procedure. Starting from an initial guess for the electronic density, the Fock matrix is constructed, molecular orbitals are obtained by solving an eigenvalue problem, and a new density matrix is formed. This process is repeated until self-consistency is achieved according to predefined convergence criteria.

The SCF procedure repeats the same sequence of computational tasks in each iteration. The dominant cost arises from integral evaluation and Fock matrix construction, both of which involve large numbers of independent operations and data-intensive tensor contractions. Efficient execution therefore depends on kernel-level optimization as well as careful orchestration of these tasks within the iterative workflow.

The subsequent sections describe how these computational structures are mapped onto GPU-accelerated kernels and integrated into an efficient SCF workflow in GANSU.

5. Implemented Computational Kernels

This section describes the computational kernels implemented in GANSU, with an emphasis on the mathematical structures underlying Hartree–Fock (HF) calculations, their implications for GPU parallelization, and the specific optimization strategies developed in our prior work. Rather than providing detailed theoretical derivations, we introduce representative equations to clarify the computational patterns targeted by each kernel.

The dominant computational cost in HF calculations is the evaluation and contraction of electron repulsion integrals. With M basis functions, the number of unique four-center ERIs scales as $O(M^4)$, while Fock matrix construction requires contracting these integrals with the $O(M^2)$ density matrix. Schwarz screening reduces the effective scaling to $O(M^2)$ – $O(M^3)$ for spatially extended systems, and the RI approximation replaces the $O(M^4)$ four-center integrals with $O(M^2 N_{\text{aux}})$ three-center integrals and $O(M^2 N_{\text{aux}})$ matrix operations, where N_{aux} is the auxiliary basis size. Matrix diagonalization via cuSOLVER scales as $O(M^3)$ but constitutes a minor fraction of the total time for the system sizes considered here. Post-HF methods introduce higher scaling: MP2 requires $O(o^2 v^2 M)$ for the integral transformation and $O(o^2 v^2)$ for the energy evaluation, where o and v denote the numbers of occupied and virtual orbitals, respectively; MP3 scales as $O(o^3 v^3)$ and is reformulated using DGEMM to exploit vendor-optimized BLAS routines.

Table 1 summarizes the parallelization strategy and key GPU optimization techniques employed by each kernel.

5.1. Electron Repulsion Integral Kernels

The evaluation of electron repulsion integrals (ERIs) is a central computational task in HF calculations. An ERI is defined as

$$(\mu\nu|\lambda\sigma) = \iint \phi_\mu(\mathbf{r}_1)\phi_\nu(\mathbf{r}_1)\frac{1}{r_{12}}\phi_\lambda(\mathbf{r}_2)\phi_\sigma(\mathbf{r}_2) d\mathbf{r}_1 d\mathbf{r}_2, \quad (1)$$

where $\{\phi_\mu\}$ denote Gaussian basis functions [47]. Several recurrence-based algorithms have been developed for efficient ERI evaluation, including the Rys quadrature method [31], the Obara–Saika (OS) scheme [50], the McMurchie–Davidson (MD) method [51], the Head–Gordon–Pople (HGP) algorithm [52], and the PRISM algorithm [53]; see [54] for a comprehensive review. The four-index structure of ERIs naturally leads to a large number of independent integral evaluations, making them well suited for massively parallel execution on GPUs.

Table 1. Summary of parallelization granularity and GPU optimization techniques for each kernel.

Kernel	Parallelization Unit	Memory Optimization	Key Technique
ERI (HGP)	Shell-pair tile (warp)	Register accumulation	Warp-based assignment [4]
ERI (MD)	Shell block/thread	Shared memory (triple-buffer)	Four CUDA mapping strategies [5]
ERI (Rys)	Shell quartet (thread)	Register accumulation	Quadrature-based evaluation [31]
3c-ERI (RI)	Shell-pair tile (warp)	Register + warp shuffle	WBA with column-wise scan [7]
Schwarz screen	Task block	Global (atomicMin)	Dynamic block assignment [8]
Boys function	Input pair	Global (lookup table)	Gridded Taylor expansion [10]
1e integrals	Shell pair	Register (symbolic expand)	Hybrid OS/MD + CUDA streams [12]
Fock construction	ERI thread	Global (replicated Fock)	DAR + TLR * [13]
MP2/MP3	Summand (thread)	Register + warp shuffle	Hierarchical reduction [14]
RI-RMP2	Occupied slice	Global (chunked B_{ia}^P)	Dynamic chunk + streams [15]
FCI	Determinant string	Global (GPU-resident)	Unified sigma kernel [16]

* DAR: distributed atomic reduction; TLR: thread-local reduction.

GANSU incorporates multiple GPU-accelerated ERI kernels. The first is based on the Head–Gordon–Pople (HGP) algorithm [4,52], which evaluates ERIs through vertical and horizontal recurrence relations (VRR and HRR). In our GPU implementation, the recurrence relations are fully expanded and simplified symbolically using SymPy to eliminate recursive function overhead. The parallelization employs a *task matrix* representation, where rows and columns correspond to shell set pairs, and the matrix is partitioned into sub-matrices grouped by orbital angular momentum type to avoid warp divergence. We developed a *warp-based assignment* (WBA) method in which each warp of 32 threads processes a 32×32 tile of the task matrix, accumulating results in registers and performing warp shuffle-based reductions before writing to global memory via atomicAdd. This approach reduced atomicAdd operations by factors of up to 1/810 compared to the conventional thread-based assignment (TBA), achieving speedups of up to 1905× over sequential CPU execution [4].

The second ERI kernel is based on the McMurchie–Davidson (MD) method [5,6,51], which computes ERIs using Hermite Gaussian functions and auxiliary *R* coefficients evaluated via batch-parallel recurrence relations. We proposed four CUDA mapping strategies: basis-block mapping (BBM), basis-thread mapping (BTM), shell-block mapping (SBM), and shell-thread mapping (STM), each offering different trade-offs between intra-block parallelism and atomicAdd overhead. A *triple-buffering* technique for shared memory was developed to store only three consecutive batches of *R* values during the recurrence computation, reducing shared memory usage by up to 65% and enabling support for arbitrary angular momentum quantum numbers on the GPU. The MD-based kernels achieved speedups of up to 4500× over naive CPU implementations [5].

For large-scale calculations, GANSU employs dynamic integral screening based on the Schwarz inequality [8,44]:

$$|(\mu\nu|\lambda\sigma)| \leq \sqrt{(\mu\nu|\mu\nu)} \cdot \sqrt{(\lambda\sigma|\lambda\sigma)}. \quad (2)$$

Our GPU implementation performs screening at two granularity levels: the task-level screening of entire shell quartets and fine-grained primitive integral-level screening within non-skipped tasks. Shell pairs are sorted in descending order by their Schwarz upper bounds, so that significant integrals cluster in the upper-left region of the task matrix and negligible integrals cluster in the lower-right. A *dynamic block assignment* strategy exploits asynchronous CUDA block scheduling to propagate skip boundaries via atomicMin operations, eliminating more than 99.9% of upper-bound evaluations for skippable tasks. This approach achieved up to $18\times$ speedup over non-screening ERI computation and up to a 35.5% reduction in computation time compared to static task screening [8].

GANSU also incorporates three-center ERI kernels for the resolution-of-the-identity (RI) approximation [7,43]. In the RI formulation, four-center ERIs are approximated as

$$(\mu\nu|\lambda\sigma) \approx \sum_{PQ} (\mu\nu|P)(P|Q)^{-1}(Q|\lambda\sigma), \quad (3)$$

which reduces the computational complexity by replacing four-center integrals with three-center integrals and matrix operations. The three-center ERI kernels adapt the warp-based reduction techniques from the four-center case, with three specialized strategies: WBA, WBA with row-wise scan (WBA-RS), and WBA with column-wise scan (WBA-CS). The WBA-CS method achieved up to $2.76\times$ speedup over TBA with up to a 96% reduction in atomicAdd operations, and outperformed GPU4PySCF by up to $4.99\times$ [7].

In addition to the HGP and MD kernels, GANSU implements ERI evaluation based on the Rys quadrature method [31]. While the HGP and MD algorithms are efficient for low-angular momentum shells, their intermediate array sizes grow rapidly with angular momentum quantum number, making them impractical for d -type and higher shells. The Rys quadrature method avoids this combinatorial growth by expressing each ERI as a weighted sum over a small number of quadrature points, with the number of points determined by the total angular momentum of the shell quartet. GANSU uses the Rys quadrature kernel for shell quartets involving d or higher angular momentum functions, while the HGP and MD kernels handle lower angular momentum cases where their recurrence-based approaches are more efficient.

5.2. Boys Function Evaluation

The evaluation of ERIs relies on the repeated computation of the Boys function,

$$F_n(T) = \int_0^1 t^{2n} e^{-Tt^2} dt, \quad (4)$$

which appears as a fundamental numerical primitive in all Gaussian integral algorithms [47], including the HGP, MD, and Obara–Saika methods [1]. In practical HF calculations, a large number of Boys function values with different parameters must be evaluated.

In GANSU, Boys function evaluation is implemented as a device function called inline within ERI kernels, based on the algorithms developed in [9,10]. The kernel employs four numerical methods: origin Taylor expansion, *gridded Taylor expansion*, recurrence relations, and the semi-infinite interval approximation. It selects the optimal method for each input pair (n, T) based on precomputed accuracy boundaries. The gridded Taylor expansion

method, introduced in [10], expands the Boys function around evenly-spaced grid points using precomputed lookup tables stored in GPU global memory. With a grid interval of $d = 2^{-3}$, this method requires fewer than ten expansion terms (compared to 10–40 for the origin Taylor expansion), yielding a compact 65.5 KB lookup table covering $0 \leq n \leq 24$ and $0 \leq T \leq 32$.

The original work [10] implemented Boys function evaluation as a standalone bulk kernel with input sorting to avoid warp divergence, achieving a throughput of 17.7×10^9 evaluations per second on an NVIDIA A100 GPU. In GANSU, the same numerical algorithms are instead embedded as device functions within the ERI kernels, eliminating the overhead of a separate kernel launch and enabling the compiler to optimize register usage across integral recurrence and Boys function evaluation. The function supports both single evaluation (computing $F_n(T)$ alone) and *incremental evaluation* (computing all values $F_0(T), F_1(T), \dots, F_n(T)$), the latter being directly required by recursive integral algorithms. All the computed values satisfy an absolute error bound of 10^{-14} .

5.3. One-Electron Integrals

One-electron integrals, including overlap integrals, kinetic energy integrals, and nuclear attraction integrals, are required for constructing the core Hamiltonian matrix $H_{\mu\nu}$ in each SCF iteration. The nuclear attraction integrals are defined by

$$(\mu|V_{\text{nuc}}|\nu) = \sum_A \int \phi_\mu(\mathbf{r}) \frac{-Z_A}{|\mathbf{r} - \mathbf{R}_A|} \phi_\nu(\mathbf{r}) d\mathbf{r}, \quad (5)$$

where Z_A and $\mathbf{R}_A$ denote the charge and position of nucleus A .

GANSU implements GPU-accelerated kernels for all one-electron integrals using a *hybrid* Obara–Saika (OS) [50] and McMurchie–Davidson (MD) [51] approach [11,12]. For each combination of orbital angular momenta, the faster of the two recurrence methods is selected based on empirical benchmarks. The recurrence relations for both methods are fully expanded symbolically using SymPy, eliminating the recursive function overhead and achieving speedups of up to $385\times$ over unexpanded GPU implementations [11]. Shell pairs are sorted by angular momentum type, and separate CUDA kernels are launched for each shell-type combination. Multiple CUDA streams are used to execute these kernels concurrently, providing up to $2.2\times$ additional speedup through overlapping kernel execution [12]. The hybrid approach with CUDA stream multiplexing achieved overall speedups of up to $1635\times$ over sequential CPU execution for nuclear attraction integrals and up to $132.6\times$ over GPU4PySCF for the full set of one-electron integrals [11,12].

5.4. Fock Matrix Construction

The Fock matrix is constructed by contracting ERIs with the density matrix. For closed-shell systems, the Fock matrix elements are given by

$$F_{\mu\nu} = H_{\mu\nu} + \sum_{\lambda\sigma} P_{\lambda\sigma} \left[(\mu\nu|\lambda\sigma) - \frac{1}{2}(\mu\lambda|\nu\sigma) \right], \quad (6)$$

where $H_{\mu\nu}$ denotes the core Hamiltonian matrix and $P_{\lambda\sigma}$ is the density matrix.

This expression highlights that Fock matrix construction is dominated by large-scale reduction operations: the $O(M^4)$ ERI contributions must be accumulated into an $O(M^2)$ Fock matrix, creating severe memory contention when multiple GPU threads simultaneously update overlapping matrix elements via atomicAdd operations. To address this bottleneck, GANSU employs a *distributed atomic reduction* (DAR) method [13] based on replicated Fock matrices. Instead of directing all atomic updates to a single Fock matrix, GANSU allocates N_{Fock} replicas in GPU global memory, and each thread writes to a replica

determined by $k = t \bmod N_{\text{Fock}}$, where t is the thread index. The final Fock matrix is obtained by summing all the replicas. This approach distributes atomic operations across distinct memory addresses, reducing contention without reducing the total number of operations. The DAR method achieved up to $3.75\times$ speedup for mid-size molecules, with profiling showing up to 84.8% reduction in stalled warp cycles and up to $2.82\times$ improvement in L2 cache throughput [13]. The method is compatible with existing thread-local reduction (TLR) techniques, and the hybrid DAR+TLR approach provides up to $1.98\times$ additional speedup.

The full eight-fold symmetry of ERIs is exploited using the standard six-update equation per unique ERI, and primitive shells are sorted by Schwarz upper bounds to minimize warp divergence. Dedicated CUDA kernels are generated for each ERI sub-matrix type (e.g., [ss|ss], [sp|sp], and [pp|pp]) to ensure uniform workloads within warps.

6. SCF Workflow Integration

This section describes how the computational kernels introduced in Section 5 are integrated into an end-to-end self-consistent field (SCF) workflow in GANSU. By explicitly relating the mathematical structure of the SCF procedure to the execution model on GPUs, we clarify how kernel orchestration and memory reuse are achieved at the application level.

6.1. SCF Iteration Structure

Hartree–Fock calculations are based on a nonlinear eigenvalue problem that is solved iteratively. In each SCF iteration, the Fock matrix $\mathbf{F}$ is constructed from the current density matrix $\mathbf{P}$, and the Roothaan–Hall equation [48,55],

$$\mathbf{F}\mathbf{C} = \mathbf{S}\mathbf{C}\epsilon, \quad (7)$$

is solved to obtain updated molecular orbital coefficients $\mathbf{C}$ and orbital energies ϵ . Here, $\mathbf{S}$ denotes the overlap matrix.

The density matrix is then updated according to

$$P_{\mu\nu} = 2 \sum_i^{\text{occ}} C_{\mu i} C_{\nu i}, \quad (8)$$

for closed-shell systems, and the procedure is repeated until the convergence of the total energy or density matrix is achieved.

This iterative structure means that the same sequence of operations, namely integral evaluation, Fock matrix construction, matrix diagonalization, and density update, is executed repeatedly. Efficient SCF execution therefore requires minimizing the overheads associated with kernel invocation, data movement, and memory allocation across iterations.

6.2. Kernel Orchestration Within an SCF Iteration

In GANSU, a single SCF iteration is realized as an ordered sequence of GPU kernel executions coordinated by the SCF control layer. Electron repulsion integral kernels, Boys function evaluation, and Fock matrix construction kernels are invoked in a tightly coupled manner to maximize data locality and reduce the synchronization overhead.

Mathematically, the dominant operation within each iteration is the contraction

$$F_{\mu\nu}^{(k)} = H_{\mu\nu} + \sum_{\lambda\sigma} P_{\lambda\sigma}^{(k-1)} \left[(\mu\nu|\lambda\sigma) - \frac{1}{2}(\mu\lambda|\nu\sigma) \right], \quad (9)$$

where the superscript (k) denotes the SCF iteration index. In the Direct-SCF formulation [56], GANSU streams batches of ERIs through GPU kernels and accumulates their

contributions directly into GPU-resident Fock matrix replicas [13], with Schwarz screening [8] applied to eliminate negligible batches before computation. In the in-memory formulation, the precomputed ERI tensor is contracted with the updated density matrix using GPU-accelerated matrix operations.

By retaining both the density matrix and intermediate Fock matrix data on the GPU across kernel invocations, GANSU avoids repeated host–device data transfers. This design is particularly important because the contraction in Equation (9) involves large-scale reductions that are sensitive to memory bandwidth and synchronization costs.

6.3. Density Matrix Mixing and Convergence Control

To accelerate convergence of the SCF procedure, GANSU employs standard density matrix mixing techniques such as damping and direct inversion in the iterative subspace (DIIS) [40,41]. In the DIIS approach, error vectors are defined as

$$\mathbf{e}^{(k)} = \mathbf{F}^{(k)} \mathbf{P}^{(k)} \mathbf{S} - \mathbf{S} \mathbf{P}^{(k)} \mathbf{F}^{(k)}, \quad (10)$$

and an improved Fock matrix is constructed as a linear combination of previous Fock matrices,

$$\mathbf{F}^{\text{DIIS}} = \sum_i c_i \mathbf{F}^{(i)}, \quad (11)$$

where the coefficients $\{c_i\}$ are obtained by solving a small auxiliary linear system.

Although the DIIS coefficient computation itself is inexpensive and performed on the CPU, the resulting mixed Fock matrix is stored on the GPU and reused directly in subsequent kernel executions. As a result, GANSU keeps the computationally intensive parts of the SCF iteration on the GPU while limiting CPU–GPU interactions to lightweight control tasks.

6.4. Matrix Diagonalization and Data Locality

Solving the Roothaan–Hall equation requires diagonalization of the Fock matrix in an orthonormal basis. In GANSU, the generalized eigenvalue problem in Equation (7) is transformed using the overlap matrix,

$$\mathbf{F}' = \mathbf{X}^T \mathbf{F} \mathbf{X}, \quad (12)$$

where $\mathbf{X}$ satisfies

$$\mathbf{X}^T \mathbf{S} \mathbf{X} = \mathbf{I}. \quad (13)$$

The transformed Fock matrix $\mathbf{F}'$ is diagonalized to obtain updated molecular orbitals. Matrix operations involved in this step are delegated to GPU-accelerated linear algebra libraries (cuBLAS and cuSOLVER), allowing diagonalization and matrix transformations to be executed efficiently while keeping data resident on the GPU.

6.5. Workflow-Level Integration and Performance Implications

The performance advantage of GANSU over CPU-based implementations and naïve GPU ports arises from three complementary design principles that together eliminate the most common sources of overhead in GPU-accelerated scientific workflows.

6.5.1. Elimination of Redundant Data Movement

In a typical SCF iteration, the density matrix $\mathbf{P}$, Fock matrix $\mathbf{F}$, integral buffers, and molecular orbital coefficients $\mathbf{C}$ are all consumed or produced by successive kernels. A naïve implementation would transfer these matrices between the host and device at each kernel boundary. GANSU instead allocates all intermediate data on the GPU at the start of

the calculation and retains them across all SCF iterations, restricting CPU–GPU transfers to lightweight scalar quantities (energies and convergence flags). This design converts what would otherwise be a memory bandwidth-limited pipeline into a compute-limited one for small and medium systems.

6.5.2. Adaptive Bottleneck Mitigation

The dominant performance bottleneck shifts with system size. For small molecules ($M < 200$), ERI evaluation is compute-bound with high arithmetic intensity from recurrence relations, and the GPU achieves high FLOP utilization. As M grows, Fock matrix construction becomes memory-bound: the $O(M^4)$ integral contributions must be reduced into an $O(M^2)$ matrix through scattered atomic updates that saturate memory bandwidth rather than compute units. The DAR method [13] addresses this transition by distributing atomic updates across N_{Fock} -replicated Fock matrices, reducing memory contention by a factor proportional to N_{Fock} and improving the L2 cache throughput by up to $2.82\times$. For systems exceeding stored-ERI capacity, the RI approximation reduces the dominant cost from $O(M^4)$ to $O(M^2 N_{\text{aux}})$ three-center integrals followed by dense matrix operations that map efficiently to cuBLAS DGEMM.

6.5.3. Minimization of Kernel Launch and Synchronization Overhead

Recurrence relations in ERI and one-electron integral kernels are fully expanded symbolically at code generation time, eliminating recursive function calls and enabling the compiler to allocate all intermediates in registers. Shell pairs are sorted by angular momentum type so that threads within each warp follow identical control flow, avoiding warp divergence. Multiple independent kernels (e.g., overlap, kinetic, and nuclear attraction gradient kernels) are launched in separate CUDA streams for concurrent execution. Together, these choices minimize the fraction of GPU time spent on non-computational work: the “Other” category in the performance breakdown (Section 9) accounts for less than 8% of total SCF time across all tested molecules.

7. Post-Hartree–Fock Methods

While Hartree–Fock theory provides a mean-field description of electronic structure, the accurate treatment of electron correlation requires post-HF methods. GANSU provides GPU-accelerated implementations of Møller–Plesset perturbation theory [57] at the second and third orders and full configuration interaction, integrated within the same GPU-resident data flow as the SCF workflow.

7.1. MP2 and MP3 Correlation Energy

The second-order Møller–Plesset (MP2) correlation energy for a closed-shell system is given by

$$E^{(2)} = \sum_{i < j}^{\text{occ}} \sum_{a < b}^{\text{virt}} \frac{(ia|jb)[2(ia|jb) - (ib|ja)]}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}, \quad (14)$$

where i, j index occupied and a, b index virtual molecular orbitals, and ϵ are orbital energies. The four-index integrals $(ia|jb)$ are obtained by transforming the AO-basis ERIs to the MO basis.

GANSU implements the MP2 energy evaluation using a *hierarchical three-level reduction scheme* [14]. Each GPU thread evaluates one summand in registers; warp shuffle instructions aggregate partial sums within warps of 32 threads; atomicAdd to shared memory aggregates within CUDA blocks; and a single atomicAdd per block updates the global accumulator. With 1024 threads per block, this approach minimizes global memory

atomic operations and achieved speedups of up to $255.5\times$ over PySCF and $71.1\times$ over GPU4PySCF [14].

The third-order Møller–Plesset (MP3) correlation energy consists of three terms (4h2p, 2h4p, and 3h3p), each involving six-fold summations with $O(M^6)$ cost. GANSU reformulates all three terms into unified four-fold summations by factoring out the inner double summations as *four matrix multiplications*, which are evaluated using cuBLAS DGEMM [14]. This DGEMM-based reformulation delegates the most expensive $O(M^6)$ contractions to vendor-optimized routines, while the remaining four-fold summation is handled by a custom CUDA kernel with the same hierarchical reduction scheme as MP2. The MP3 implementation achieved speedups of up to $96.2\times$ over PySCF [14].

7.2. RI-RMP2 Correlation Energy

For systems where the full four-center MO integral tensor exceeds available GPU memory, GANSU provides an RI-based restricted MP2 (RI-RMP2) implementation [15]. The RI approximation introduces an intermediate tensor B_{ia}^P constructed from three-center integrals and the auxiliary basis metric, reducing memory requirements from $O(M^4)$ to $O(M^3)$.

The four-center MO integrals are reconstructed from the intermediate tensor via cuBLAS DGEMM:

$$(ia|jb) \approx \sum_P B_{ia}^P B_{jb}^P. \quad (15)$$

To further control memory usage, the occupied index i is sliced into chunks of length k , dynamically determined at runtime to maximize utilization of available GPU memory [15]. The correlation energy formula is decomposed into four terms with mutually exclusive index ranges, each computed by a separate CUDA kernel launched in different CUDA streams for concurrent execution. The RI-RMP2 implementation achieved speedups of up to $165.2\times$ over PySCF and $4.04\times$ over GPU4PySCF [15].

7.3. Full Configuration Interaction

For exact ground-state energy calculations within a given basis set, GANSU provides a fully GPU-accelerated full configuration interaction (FCI) implementation based on the Knowles–Handy algorithm [16,58]. The core operation is the sigma vector formation $\sigma = \mathbf{H}\mathbf{c}$ within the Davidson iterative eigensolver [59], where $\mathbf{H}$ is the full CI Hamiltonian matrix and $\mathbf{c}$ is the trial vector.

Unlike prior GPU implementations that accelerated only the sigma formation while leaving the Davidson procedure on the CPU, GANSU performs the *entire* Davidson eigensolver on the GPU, including orthogonalization, subspace diagonalization, and residual computation [16]. The sigma formation kernel uses unified thread assignment for both alpha- and beta-string contributions, eliminating excessive kernel launches and atomicAdd operations present in earlier implementations. All large vectors and matrices are allocated directly in GPU memory, with CPU–GPU transfers limited to the small ERI tensor before the Davidson loop. This fully GPU-resident approach achieved speedups of up to $35.4\times$ over PySCF and $15.9\times$ over GPU4PySCF [16], and was recognized with an Outstanding Paper Award.

8. Analytical Energy Gradients and Geometry Optimization

Beyond single-point energy calculations, GANSU provides GPU-accelerated analytical energy gradients and gradient-based geometry optimization. These capabilities allow users to locate equilibrium structures and stationary points on potential energy surfaces entirely within the GPU-resident workflow.

8.1. Analytical Energy Gradients

The total HF energy gradient with respect to nuclear coordinate R_{Ax} of atom A can be decomposed as

$$\frac{\partial E}{\partial R_{Ax}} = \frac{\partial V_{nn}}{\partial R_{Ax}} + \sum_{\mu\nu} P_{\mu\nu} \frac{\partial H_{\mu\nu}}{\partial R_{Ax}} + \frac{1}{2} \sum_{\mu\nu\lambda\sigma} P_{\mu\nu} P_{\lambda\sigma} \frac{\partial (\mu\nu|\lambda\sigma)}{\partial R_{Ax}} - \sum_{\mu\nu} W_{\mu\nu} \frac{\partial S_{\mu\nu}}{\partial R_{Ax}}, \quad (16)$$

where V_{nn} is the nuclear repulsion energy, $H_{\mu\nu}$ is the core Hamiltonian, $S_{\mu\nu}$ is the overlap matrix, $P_{\mu\nu}$ is the density matrix, and $W_{\mu\nu} = 2 \sum_k C_{\mu k} C_{\nu k} \epsilon_k$ is the energy-weighted density matrix. The four terms correspond to the nuclear repulsion gradient, the one-electron integral gradients (kinetic energy and nuclear attraction), the two-electron integral gradient, and the overlap gradient weighted by $\mathbf{W}$, respectively.

GANSU evaluates all four contributions analytically on the GPU. Dedicated CUDA kernels are provided for each component:

- The overlap gradient $\partial S_{\mu\nu} / \partial R_{Ax}$;
- The kinetic energy gradient $\partial T_{\mu\nu} / \partial R_{Ax}$;
- The nuclear attraction gradient $\partial V_{\mu\nu} / \partial R_{Ax}$, which requires differentiated Boys functions;
- The two-electron integral gradient $\partial (\mu\nu|\lambda\sigma) / \partial R_{Ax}$, computed in a single pass over all shell quartets with eight-fold symmetry;
- The nuclear repulsion gradient, evaluated via direct Coulomb force computation.

The one-electron derivative integrals are computed via the recursive differentiation of the McMurchie–Davidson expansion coefficients [51], with all recurrence relations fully expanded symbolically to eliminate function call overhead. The two-electron integral gradient is evaluated using the Rys quadrature method [31] for all angular momentum types, since nuclear coordinate differentiation increases the effective angular momentum by one, making the MD intermediate arrays impractically large. Multiple CUDA streams are used to overlap the execution of independent gradient kernels, and the partial gradient arrays are accumulated on the GPU using cuBLAS vector operations. Translational invariance ($\partial E / \partial R_B = -\partial E / \partial R_A$ for a two-center integral) is exploited to reduce the number of derivative evaluations.

The gradient computation follows the same Strategy pattern as the energy calculation: the base class HF declares a pure virtual method `compute_Energy_Gradient()`, and both RHF and UHF provide concrete implementations. The UHF gradient kernel accepts separate alpha and beta density matrices and computes the corresponding exchange contributions independently.

8.2. Geometry Optimization Performance

Using the analytical gradients, GANSU performs geometry optimization through a dedicated optimizer hierarchy that again follows the Strategy design pattern [39]. An abstract base class `GeometryOptimizer` defines the interface for search direction computation, step acceptance, and convergence checking, and a factory method selects the concrete optimizer at runtime based on user input.

Nine optimization algorithms are currently implemented:

- Quasi-Newton methods: BFGS (Broyden–Fletcher–Goldfarb–Shanno), DFP (Davidon–Fletcher–Powell), and SR1 (symmetric rank-1 update). These methods build an approximate inverse Hessian from successive gradient differences using rank-one or rank-two updates [60].
- Conjugate gradient methods: Fletcher–Reeves [61], Polak–Ribière [62], Hestenes–Stiefel [63], and Dai–Yuan [64] variants, each differing in the formula for the conjugate direction parameter β .

- Steepest descent: A baseline first-order method using the negative gradient as the search direction.
- GDIIS (geometry DIIS) [42]: A hybrid method that combines the BFGS Hessian update with DIIS extrapolation over a history of recent geometries and gradients, avoiding the need for a line search.

Line search is used for the quasi-Newton and conjugate gradient methods to determine the step length along the computed search direction. The GDIIS method instead determines the step from the DIIS subspace directly.

At each optimization step, GANSU performs a full SCF calculation at the current geometry, evaluates the analytical gradient, updates the nuclear coordinates according to the chosen optimizer, and repeats until convergence. Because both the SCF energy and the gradient are computed on the GPU, the optimization loop incurs minimal CPU–GPU data transfer beyond the updated nuclear coordinates and convergence status.

9. Performance Evaluation

This section presents a performance evaluation of the GANSU framework from both kernel-level and workflow-level perspectives. The objective is to quantify the performance of individual GPU-accelerated kernels as well as the end-to-end efficiency of the integrated self-consistent field (SCF) workflow.

9.1. Experimental Setup

All benchmarks were performed on a system equipped with an NVIDIA H200 GPU (141 GB HBM3e) and two Intel Xeon Platinum 8592+ CPUs (64 cores each; 128 cores total), with 2 TB of system DDR5 DRAM. GANSU uses the GPU as the sole computational device and does not employ CPU parallelism; the CPUs are used only for single-threaded SCF control logic. Double-precision arithmetic was employed throughout all calculations.

For comparison, we selected two reference implementations: PySCF [28,29] (version 2.11, CPU-based, and utilizing all 128 cores via multithreaded linear algebra) and GPU4PySCF [30] (version 1.5.2; GPU-accelerated on the same H200 GPU). All three implementations use identical molecular geometries, the same basis sets, and converge to the same SCF energy threshold (10^{-9} Hartree). For small and medium molecules, GANSU employs stored ERI, precomputing all integrals once and retaining them in GPU memory. For larger molecules that exceed stored-ERI memory capacity, GANSU uses the resolution-of-the-identity (RI) approximation; PySCF and GPU4PySCF use density fitting (DF-HF) for these molecules to ensure a fair comparison at the same level of integral approximation. The timings reported below are end-to-end wall-clock times from a user perspective, encompassing both integral computation and SCF convergence. Table 2 lists the molecular systems selected for evaluation, which range from small validation cases (H_2O ; 25 basis functions) to cholesterol (650 basis functions).

9.2. Kernel-Level Performance

Table 3 summarizes the performance of individual computational kernels implemented in GANSU, as reported in our prior publications. These results confirm that each kernel achieves large speedups while maintaining numerical accuracy sufficient for quantum chemical calculations.

The ERI evaluation kernels benefit from massive thread-level parallelism and the warp-based reduction strategies that minimize atomicAdd contention. The Boys function, evaluated inline within ERI kernels, achieves high throughput through the gridded Taylor expansion method. The one-electron integral kernels demonstrate the effectiveness of the hybrid OS/MD approach with CUDA stream-based concurrent execution. The Fock

matrix construction kernel shows that distributed atomic reduction effectively mitigates memory contention in the accumulation step. The integration of these kernels into GANSU preserves their individually optimized performance characteristics.

Table 2. Molecular systems used for benchmarking.

Molecule	Atoms	Basis Set	M	ERI (GANSU)
Water (H ₂ O)	3	cc-pVDZ	25	Stored
Methane (CH ₄)	5	cc-pVDZ	35	Stored
Ethanol (C ₂ H ₆ O)	9	6-31G*	57	Stored
Benzene (C ₆ H ₆)	12	6-31G*	102	Stored
Pentane (C ₅ H ₁₂)	17	cc-pVDZ	135	Stored
Naphthalene (C ₁₀ H ₈)	18	6-31G*	166	Stored
Decane (C ₁₀ H ₂₂)	32	6-31G*	194	Stored
Anthracene (C ₁₄ H ₁₀)	24	cc-pVDZ	260	Stored
Fullerene (C ₆₀)	60	STO-3G	300	Stored
Taxol (C ₄₇ H ₅₁ NO ₁₄)	113	STO-3G	361	RI
Dodecahedrane (C ₂₀ H ₂₀)	40	cc-pVDZ	400	RI
Sirolimus (C ₅₁ H ₇₉ NO ₁₃)	144	STO-3G	404	RI
Morphine (C ₁₇ H ₁₉ NO ₃)	40	cc-pVDZ	410	RI
Penicillin G (C ₁₆ H ₁₈ N ₂ O ₄ S)	42	cc-pVDZ	434	RI
C ₉₀ (C ₉₀)	90	STO-3G	450	RI
Oleic acid (C ₁₈ H ₃₄ O ₂)	54	cc-pVDZ	470	RI
Cyclosporine (C ₆₂ H ₁₁₁ N ₁₁ O ₁₂)	196	STO-3G	536	RI
Folic acid (C ₁₉ H ₁₉ N ₇ O ₆)	51	cc-pVDZ	575	RI
Cholesterol (C ₂₇ H ₄₆ O)	74	cc-pVDZ	650	RI

Table 3. Summary of kernel-level performance results from prior publications. Speedups are representative peak values.

Kernel	Peak Speedup	Reference
ERI (HGP, WBA)	1905× vs. seq. CPU	[4]
ERI (MD, STM)	4500× vs. seq. CPU	[5]
ERI screening	18× vs. no screening	[8]
Three-center RI-ERI	4.99× vs. GPU4PySCF	[7]
Boys function	21.7× vs. 72-core CPU	[10]
Nuclear attraction	1635× vs. seq. CPU	[11]
One-electron integrals	132.6× vs. GPU4PySCF	[12]
Fock matrix (DAR)	3.75× vs. baseline GPU	[13]

9.3. End-to-End SCF Performance

Table 4 compares the total wall-clock time for RHF/SCF convergence. The upper portion of the table shows molecules computed with stored ERI, while the lower portion shows larger molecules computed with RI approximation.

For small molecules (H₂O, CH₄, and $M < 50$), all three implementations complete in about 1 s or less; at this scale, kernel launch latency and initialization overhead dominate over actual computation, and no meaningful speedup distinction can be drawn. As the system size increases and the $O(M^4)$ ERI cost becomes dominant, the advantage of GANSU becomes pronounced. For stored-ERI molecules ($M \leq 300$), GANSU achieves speedups of up to 13× over PySCF (benzene) and 1–8× over GPU4PySCF. For molecules computed with RI approximation ($M = 361$ –650), GANSU achieves speedups of 11–52× over PySCF, with the largest speedup for sirolimus (52×). Compared to GPU4PySCF, GANSU is 1–3× faster for STO-3G RI molecules, where the RI integral cost dominates. For the largest cc-pVDZ molecules (folic acid and cholesterol), GPU4PySCF achieves comparable or slightly

faster performance than GANSU, as its highly optimized three-center integral engine compensates for the higher formal cost at these system sizes.

Table 4. End-to-end RHF/SCF wall-clock time (seconds).

Molecule	<i>M</i>	GANSU	PySCF	GPU4PySCF
<i>Stored ERI</i>				
Water (H ₂ O)	25	0.17	0.26	1.36
Methane (CH ₄)	35	0.17	0.89	0.59
Ethanol (C ₂ H ₆ O)	57	0.17	1.19	0.72
Benzene (C ₆ H ₆)	102	0.20	2.64	1.14
Pentane (C ₅ H ₁₂)	135	0.26	2.97	1.51
Naphthalene (C ₁₀ H ₈)	166	0.36	3.54	1.75
Decane (C ₁₀ H ₂₂)	194	0.43	3.54	1.88
Anthracene (C ₁₄ H ₁₀)	260	1.26	4.34	3.77
Fullerene (C ₆₀)	300	2.69	7.71	3.11
<i>RI approximation</i>				
Taxol (C ₄₇ H ₅₁ NO ₁₄)	361	1.01	50.45	2.83
Dodecahedrane (C ₂₀ H ₂₀)	400	0.86	9.57	0.87
Sirolimus (C ₅₁ H ₇₉ NO ₁₃)	404	1.27	65.86	3.31
Morphine (C ₁₇ H ₁₉ NO ₃)	410	1.02	15.89	1.25
Penicillin G (C ₁₆ H ₁₈ N ₂ O ₄ S)	434	1.26	16.08	1.34
C ₉₀ (C ₉₀)	450	1.90	85.97	4.22
Oleic acid (C ₁₈ H ₃₄ O ₂)	470	1.15	16.84	1.20
Cyclosporine (C ₆₂ H ₁₁₁ N ₁₁ O ₁₂)	536	2.58	130.13	5.62
Folic acid (C ₁₉ H ₁₉ N ₇ O ₆)	575	2.62	34.44	2.00
Cholesterol (C ₂₇ H ₄₆ O)	650	3.17	35.26	1.95

Table 5 presents the gradient computation times. The GANSU timings include both SCF convergence and gradient evaluation.

Table 5. Analytical gradient computation time (seconds), including SCF convergence.

Molecule	<i>M</i>	GANSU	PySCF	GPU4PySCF
Water (H ₂ O)	25	0.17	0.58	1.58
Benzene (C ₆ H ₆)	102	0.21	6.30	1.45
Naphthalene (C ₁₀ H ₈)	166	0.36	10.17	2.65
Decane (C ₁₀ H ₂₂)	194	0.43	9.33	2.10
Anthracene (C ₁₄ H ₁₀)	260	1.17	13.34	4.75

The gradient timings follow the same trend as the SCF results. For benzene, GANSU completes the gradient in 0.21 s compared to 6.30 s for PySCF (30×) and 1.45 s for GPU4PySCF (6.9×). For anthracene (*M* = 260), GANSU achieves 11.4× over PySCF and 4.1× over GPU4PySCF.

9.4. Integral-Direct SCF and Extended Code Comparisons

This subsection adds two comparisons to the preceding results: integral-direct SCF within GANSU, where four-center ERIs are recomputed every SCF iteration instead of being precomputed and stored, and the CPU-native package ORCA 6.1.1 [65,66] on the same 128-core CPU system as the PySCF runs. Table 6 shows the SCF compute times on a ten-molecule subset. The timings are internal compute times (SCF solve plus integral evaluation) and do not include process-level startup: GPU context initialization and basis loading for GANSU, and MPI launcher and basis loading for ORCA. These overheads are 1–3 s for GANSU and 20–25 s for ORCA on 128 ranks, and would otherwise swamp small-molecule comparisons; the convention matches Tables 4 and 5.

Table 6. SCF compute time (seconds, internal timer; startup excluded) extended with integral-direct SCF (GANSU) and ORCA 6.1.1 comparison. The “GANSU (stored/RI)” column reproduces the strategy used in Table 4: stored ERI for $M \leq 300$ and RI for $M > 300$. The “GANSU direct” column reports the integral-direct strategy ERI_Direct (four-center ERIs recomputed every SCF iteration with Schwarz screening). For $M > 300$ the recommended strategy is RI; direct is therefore omitted for these rows. ORCA runs use its default integral-direct SCF on all 128 CPU cores via MPI; the “ORCA RIJK” column uses the RIJK AutoAux keywords and is reported only for $M > 300$ where RI-level comparison is meaningful. ORCA 6.1 supports only spherical Gaussians (5d, 7f), whereas GANSU, PySCF, and GPU4PySCF use Cartesian (6d, 10f); the M column lists the Cartesian value.

Molecule	M	GANSU (Stored/RI)	GANSU Direct	PySCF	GPU4 PySCF	ORCA Direct	ORCA RIJK
<i>Stored ERI regime ($M \leq 300$)</i>							
Ethanol	57	0.17	1.07	1.19	0.72	1.29	—
Benzene	102	0.20	1.28	2.64	1.14	1.64	—
Pentane	135	0.26	1.99	2.97	1.51	2.07	—
Naphthalene	166	0.36	4.29	3.54	1.75	2.46	—
Decane	194	0.43	4.22	3.54	1.88	2.49	—
Anthracene	260	1.26	21.7	4.34	3.77	4.91	—
Fullerene	300	2.69	11.9	7.71	3.11	13.3	—
<i>RI regime ($M > 300$)</i>							
Taxol	361	1.01	—	50.45	2.83	9.80	100.7
Morphine	410	1.02	—	15.89	1.25	15.9	33.5
Cholesterol	650	3.17	—	35.26	1.95	35.1	81.8

The ERI_Direct strategy recomputes four-center ERIs with Schwarz screening on every SCF iteration. This removes the GPU-memory cost of storing the full ERI tensor, but pays N_{iter} times the integral cost. For the small and medium molecules in Table 6, GANSU direct takes 1–22 s, which is 4–17 $\times$ the stored-ERI time (1.07 s vs. 0.17 s for ethanol; 21.7 s vs. 1.26 s for anthracene). The ratio tracks the number of SCF iterations: the GPU kernel speedups of Table 3 apply once per run for stored ERI but N_{iter} times for direct. This is why the paper uses stored ERI for $M \leq 300$ and RI for $M > 300$: integral-direct SCF doesn’t reveal new GPU speedup, it just pays the ERI cost more often.

The anthracene/fullerene ordering flips between columns. Anthracene (cc-pVDZ; $M = 260$) is faster than fullerene (STO-3G; $M = 300$) in the stored-ERI, PySCF, and ORCA direct columns, but slower in the GANSU direct and GPU4PySCF columns. The per-iteration cost is not a pure function of M ; it also depends on shell angular momentum (for direct SCF with explicit ERIs) and on auxiliary-basis size (for density-fitting or RI methods). A single cartesian ($dd|dd$) quartet has $6^4 = 1296$ integral components, compared with 1 for ($ss|ss$); cc-pVDZ has d shells on every carbon, and STO-3G has none. A direct-SCF code that recomputes all four-center ERIs every iteration therefore pays much more per iteration on anthracene/cc-pVDZ than on fullerene/STO-3G, even though STO-3G fullerene has a larger M . Density-fitting codes see the same effect through the auxiliary basis, which grows with orbital-basis angular momentum. Stored-ERI and RI strategies (Table 4) amortize the basis-specific integral cost over all iterations, leaving Fock construction (which scales with M) to dominate, so the M -ordering reappears. ORCA direct keeps the M -ordering as well, because COSX [67] evaluates exchange by grid integration instead of ($\ell\ell|\ell\ell$) ERIs and thereby flattens the angular-momentum dependence. The reversal is a fingerprint of algorithms in which basis-specific per-iteration constants dominate; integral caching erases it.

Compared with ORCA direct on the same 128-core CPU system, GANSU direct is roughly comparable for $M \leq 135$ (1.28 s vs. 1.64 s for benzene at 6-31G*) and slower for larger molecules (21.7 s vs. 4.9 s for anthracene). The direct-SCF pipeline of ORCA has been optimized for over a decade: COSX [67] for the exchange contribution, the SHARK

integral library, density-based screening on top of Schwarz, and multipole expansions for long-range. The ERI_Direct strategy in GANSU, in contrast, implements only the baseline formulation (four-center ERIs with Schwarz screening); porting these specialized techniques to GPUs is a matter for future work.

What matters for the overall claim is that GANSU stored/RI beats ORCA direct (with COSX and the rest) by $3.9\text{--}15.6\times$ and ORCA RIJK with AutoAux by $25.8\text{--}99.7\times$ on the same 128-core CPU hardware. For cholesterol ($M = 650$), GANSU RI at 3.17 s is $11.1\times$ faster than ORCA direct (35.1 s) and $25.8\times$ faster than ORCA RIJK (81.8 s). All multipliers use internal compute times, so they reflect computational work rather than process launching. Integral caching on the GPU therefore wins not only against a plain direct-SCF baseline but also against a mature CPU-native direct-SCF pipeline: the speedup comes from caching and reuse, not from per-iteration kernel tuning in the direct regime.

One basis-set detail: ORCA 6.1 supports only spherical Gaussians (5d and 7f), whereas GANSU, PySCF, and GPU4PySCF use Cartesian (6d and 10f) as in the rest of Section 9. The basis function count in ORCA is therefore smaller for shells with $\ell \geq 2$ (96 spherical vs. 102 Cartesian for benzene/6-31G*; 5–15% typical), which is well below the speedups reported above.

9.5. Post-HF Performance

Table 7 summarizes the kernel-level performance of post-HF methods as reported in prior publications.

Table 7. Summary of post-HF kernel-level performance. Speedups are representative peak values.

Method	vs. PySCF	vs. GPU4PySCF	Reference
MP2	$255.5\times$	$71.1\times$	[14]
MP3	$96.2\times$	—	[14]
RI-RMP2	$165.2\times$	$4.04\times$	[15]
FCI	$35.4\times$	$15.9\times$	[16]

Table 8 presents the end-to-end MP2 timings, which include both the SCF convergence phase and the MP2 correlation calculation.

Table 8. End-to-end MP2 computation time (seconds), including SCF.

Molecule	M	GANSU	PySCF	GPU4PySCF
<i>Stored ERI</i>				
Water (H ₂ O)	25	0.10	0.51	2.28
Methane (CH ₄)	35	0.18	0.85	2.36
Benzene (C ₆ H ₆)	120	0.25	5.00	3.19
Naphthalene (C ₁₀ H ₈)	190	0.53	7.51	5.38
Decane (C ₁₀ H ₂₂)	260	0.95	7.06	7.06
Anthracene (C ₁₄ H ₁₀)	260	1.32	12.78	9.44
Fullerene (C ₆₀)	300	2.89	20.42	6.77
<i>RI approximation</i>				
Taxol (C ₄₇ H ₅₁ NO ₁₄)	361	1.21	50.81	4.63
Dodecahedrane (C ₂₀ H ₂₀)	400	1.00	11.06	1.73
Sirolimus (C ₅₁ H ₇₉ NO ₁₃)	404	1.53	69.26	5.56
Morphine (C ₁₇ H ₁₉ NO ₃)	410	1.19	16.75	2.16
Penicillin G (C ₁₆ H ₁₈ N ₂ O ₄ S)	434	1.54	17.39	2.52
C ₉₀ (C ₉₀)	450	2.44	91.47	7.39
Oleic acid (C ₁₈ H ₃₄ O ₂)	470	1.61	18.33	2.38

The MP2 speedups grow with system size, consistent with the $O(o^2v^2M)$ scaling of the integral transformation step, which dominates for larger molecules and maps efficiently to GPU-parallel execution. For small molecules (H_2O and CH_4), all three implementations complete within 2.5 s. For medium molecules ($M = 100\text{--}300$), GANSU is $7\text{--}20\times$ faster than PySCF and $2\text{--}13\times$ faster than GPU4PySCF, as the integral transformation begins to dominate over SCF overhead. For fullerene/STO-3G ($M = 300$), GANSU is $7.1\times$ faster than PySCF and $2.3\times$ faster than GPU4PySCF. For RI molecules, GANSU achieves speedups of up to $45\times$ over PySCF (sirolimus; $M = 404$) and $3\text{--}4\times$ over GPU4PySCF. The cc-pVDZ RI molecules (dodecahedrane through oleic acid) are also evaluated, with GANSU achieving $11\text{--}14\times$ over PySCF and $1.5\text{--}1.8\times$ over GPU4PySCF. These end-to-end results complement the kernel-level benchmarks in Table 7, which report speedups of up to $255\times$ over PySCF for the MP2 kernel alone.

Table 9 presents the end-to-end FCI timings. Because the FCI determinant space grows factorially with the number of electrons and orbitals, both the computational cost and memory usage increase explosively with the system's size. The FCI benchmarks therefore use small molecules with minimal basis sets ($M \leq 18$), in contrast to the SCF and MP2 evaluations above.

Table 9. End-to-end FCI computation time (seconds), including SCF.

Molecule	Basis Set	M	GANSU	PySCF	GPU4PySCF
Methane (CH_4)	STO-3G	9	0.14	1.11	0.76
Water (H_2O)	6-31G	13	0.19	5.12	2.90
Ethylene (C_2H_4)	STO-3G	14	0.44	16.71	10.01
Carbon dioxide (CO_2)	STO-3G	15	0.23	6.99	2.72
Sodium hydroxide (NaOH)	STO-3G	15	0.48	21.10	16.47
Ammonia (NH_3)	6-31G	15	0.45	17.43	10.50
Sodium chloride (NaCl)	STO-3G	18	0.55	17.37	8.51
Calcium oxide (CaO)	STO-3G	18	0.94	38.99	27.72

GANSU achieves speedups of $8\text{--}44\times$ over PySCF and $5\text{--}34\times$ over GPU4PySCF across all tested molecules. The largest speedup over PySCF is observed for NaOH ($44\times$), while the largest over GPU4PySCF is also for NaOH ($34\times$). These speedups are substantially larger than the MP2 results because the FCI Davidson eigensolver in GANSU runs entirely on the GPU, including subspace operations and residual computation [16], whereas PySCF and GPU4PySCF perform these steps with higher overhead. All molecules complete in under 1 s on GANSU, compared to 1–39 s for PySCF and 0.8–28 s for GPU4PySCF.

9.6. Geometry Optimization

As described in Section 8, GANSU provides nine optimizer algorithms (BFGS, DFP, SR1, GDIIS, CG-FR, CG-PR, CG-HS, CG-DY, and steepest descent). While PySCF and GPU4PySCF support analytical gradient computation (as compared in Table 5), they do not provide a built-in geometry optimizer; a direct optimization timing comparison is therefore omitted. In our tests, all small-molecule optimizations (CH_4 , NH_3 , and CO_2) converge in under 1 s on the H200 GPU.

9.7. Scaling Behavior

The SCF timings in Table 4 reveal how performance scales with system size. For stored-ERI molecules ($M \leq 300$), the wall-clock time grows from 0.17 s ($M = 25$) to 2.69 s ($M = 300$), a $16\times$ increase for a $12\times$ increase in basis size, indicating near-quadratic effective scaling that is significantly better than the formal $O(M^4)$ cost because (1) Schwarz screening eliminates the vast majority of negligible integrals, and (2) the precomputed

ERI tensor is reused across all SCF iterations without recomputation. For RI molecules ($M = 361$ – 650), the wall-clock time ranges from 0.86 s to 3.17 s; the RI approximation reduces the dominant cost to three-center integrals and dense matrix operations [7], both of which map efficiently to GPU DGEMM and achieve near-peak throughput. PySCF scales less favorably: its wall-clock time increases from 9.57 s ($M = 400$) to 130.13 s ($M = 536$) for RI molecules, reflecting the higher overhead of CPU-based integral evaluation.

9.8. Numerical Validation

Table 10 compares the converged RHF energies across all three implementations. All calculations use Cartesian Gaussian basis functions (six components for d shells, ten for f , etc.) to ensure a consistent basis set definition across programs. The energies agree to within 10^{-7} – 10^{-8} Hartree for small molecules (H_2O , CH_4 , and anthracene) and to within 10^{-6} Hartree for fullerene. These residual differences arise from minor variations in integral evaluation algorithms and convergence procedures, and are well below chemical accuracy (1.6×10^{-3} Hartree). PySCF and GPU4PySCF produce identical energies to the reported precision, confirming that GPU acceleration itself does not introduce numerical error.

Table 10. RHF energy comparison (Hartree) for numerical validation.

Molecule	GANSU	PySCF	GPU4PySCF
Water (H_2O)	−76.0234907	−76.0234908	−76.0234908
Methane (CH_4)	−40.1985958	−40.1985959	−40.1985959
Benzene (C_6H_6)	−230.7029054	−230.7029055	−230.7029055
Fullerene (C_{60})	−2243.9343511	−2243.9343526	−2243.9343526
Anthracene ($\text{C}_{14}\text{H}_{10}$)	−536.0409108	−536.0409112	−536.0409112

9.9. Performance Breakdown

Finally, we analyze the distribution of execution time across different components of the SCF workflow. Figure 2 shows a breakdown of the total SCF wall-clock time into ERI evaluation, Fock matrix construction, matrix diagonalization, one-electron integrals, and other overheads for four representative molecules spanning a range of system sizes.

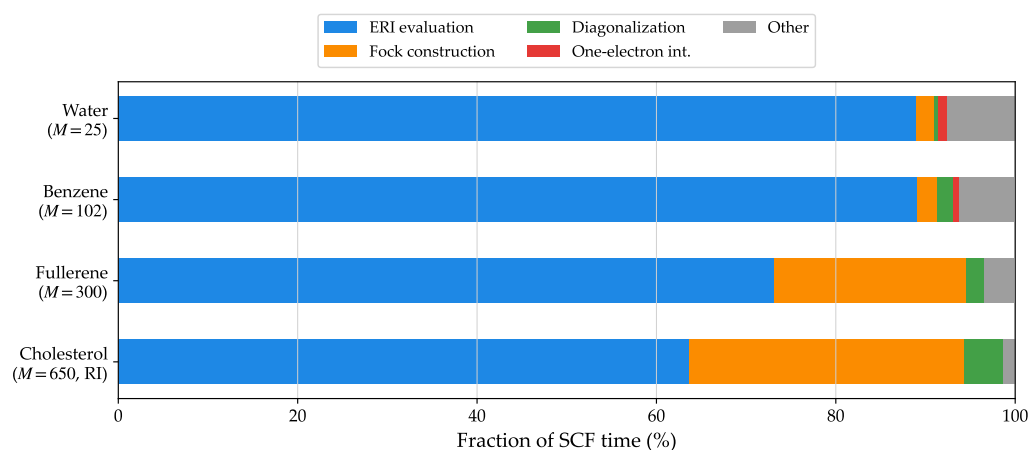


Figure 2. Breakdown of SCF wall-clock time by component, measured on an NVIDIA H200 GPU. ERI evaluation dominates for small molecules, while Fock matrix construction grows significantly for larger systems. Cholesterol uses the RI approximation ($M = 650$).

The breakdown reveals that ERI evaluation dominates the computational cost for small and medium molecules (89% for H_2O and benzene), while Fock matrix construction becomes increasingly significant for larger systems, reaching 31% for cholesterol ($M = 650$).

This shift occurs because ERI evaluation scales as $O(M^4)$ (or $O(M^2 N_{\text{aux}})$ with RI) but is performed only once in the stored-ERI strategy, whereas Fock matrix construction, which involves $O(M^2)$ reduction with $O(M^4)$ atomic updates, must execute in every SCF iteration and becomes memory bandwidth-limited as the number of concurrent atomic updates grows with M . Matrix diagonalization remains a minor contributor across all system sizes because the $O(M^3)$ dense eigenvalue problem maps efficiently to cuSOLVER. One-electron integrals are negligible for larger systems where stored ERI or RI strategies are used, as they are computed once and their $O(M^2)$ cost is dwarfed by the ERI and Fock steps. The “Other” category, which includes basis orthogonalization, energy evaluation, and density matrix construction, consistently accounts for less than 8% of the total time, confirming that kernel orchestration overhead is well controlled.

Overall, these results demonstrate that GANSU accelerates both individual computational kernels and the end-to-end workflow, confirming the effectiveness of the proposed integration strategy for GPU-accelerated Hartree–Fock and post-HF calculations.

9.10. Scope and Discussion

Several aspects of the current work merit discussion regarding scope and generalizability. All benchmarks were performed on a single NVIDIA H200 GPU with 141 GB HBM3e; the execution model is designed to be portable across CUDA-capable GPUs, though absolute performance will vary with memory bandwidth and compute capability. The largest system tested (cholesterol: 650 basis functions for SCF) demonstrates the practical range of single-GPU execution; multi-GPU and distributed-memory extensions, currently under development, will enable treatment of larger systems. GANSU uses double-precision arithmetic throughout to ensure numerical reproducibility; mixed-precision strategies could further improve throughput and represent a natural extension. For the SCF RI molecules, both GANSU and GPU4PySCF employ three-center integral approximations (RI and density fitting, respectively), providing a comparison at a similar level of integral approximation. The observed speedups of GANSU therefore reflect differences in kernel implementation and workflow integration rather than a fundamental difference in algorithmic cost. For FCI, the comparison is on equal footing and GANSU achieves speedups of up to $34\times$ over GPU4PySCF.

The 650 basis function upper-bound in Table 4 is a memory-capacity limit of a single NVIDIA H200 GPU (141 GB HBM3e) under the stored-ERI and RI strategies, not an algorithmic one. With RI, the largest GPU-resident object is the three-center integral tensor of size $O(M^2 N_{\text{aux}})$, which for cholesterol/cc-pVDZ takes about 90 GB in double precision. Three routes extend this limit. First, the ERI_Direct and ERI_RI_Direct strategies recompute integrals and eliminate the storage bound entirely. Second, larger-HBM GPUs scale the bound with device memory. Third, multi-GPU execution, under active development, will partition the tensor across devices. The 650-basis number is therefore a single-GPU stored/RI benchmark point, not a ceiling on applicability.

GANSU is written against the CUDA programming model and the NVIDIA math libraries (cuBLAS and cuSOLVER). Two paths to other GPU vendors are technically feasible. The first is direct porting via HIPIFY, which translates CUDA source to HIP for AMD GPUs and is already used by packages such as QUICK [35,36]. The second is rewriting the GPU-facing layers against a portable abstraction such as Kokkos, SYCL, or OpenMP target. The HIPIFY route preserves the existing CUDA kernel tuning and is the natural first step. Cross-vendor portability is an important direction, but a multi-vendor performance study is outside the scope of this paper.

10. Conclusions and Future Work

We presented GANSU, an integrated GPU-accelerated quantum chemistry framework designed to efficiently execute Hartree–Fock (HF) and post-HF calculations on modern GPU architectures. Unlike studies that focus on optimizing individual computational kernels in isolation, this work emphasized the integration and orchestration of multiple GPU-accelerated kernels, developed over a series of prior studies [4–16], within an end-to-end execution model.

By explicitly aligning the mathematical structure of HF theory with the execution characteristics of GPUs, GANSU enables efficient execution of electron repulsion integral evaluation, Boys function computation, one-electron integrals, and Fock matrix construction while minimizing data movement and synchronization overhead. The framework further extends to post-HF methods including MP2, MP3, RI-RMP2, and full CI, as well as analytical energy gradients and geometry optimization, all operating within a unified GPU-resident data flow. Performance evaluations demonstrated that the proposed framework preserves the high efficiency of individual kernels, achieving speedups of up to 4500 times for ERI evaluation, 21.7 times for Boys function computation over 72-core CPUs, and 132.6 times for one-electron integrals over GPU4PySCF, and, more importantly, achieves substantial end-to-end performance improvements: up to 52 times faster than PySCF for SCF, 45 times for MP2, and 44 times for FCI on an NVIDIA H200 GPU, with FCI speedups of up to 34 times over GPU4PySCF. These results highlight the importance of workflow-aware design in GPU-accelerated scientific applications.

The primary contribution of this work lies in showing how individually optimized GPU kernels can be composed into a practical, scalable application. The design principles adopted in GANSU, namely GPU-first execution, explicit memory management, and modular kernel integration, may also inform the development of GPU-native frameworks in other computational domains.

Several directions for future work are currently under development. First, extending the single-GPU execution model to multi-GPU and distributed-memory environments will enable simulations of larger molecular systems with improved strong and weak scaling. Second, excited-state methods such as configuration interaction doubles (CID), algebraic diagrammatic construction (ADC), and equation-of-motion coupled-cluster (EOM-CC) have been integrated into the framework as initial GPU-accelerated implementations; further optimization and validation of these methods for production use is ongoing. Third, density functional theory (DFT) algorithms, including exchange–correlation functional evaluation and grid-based numerical integration on GPUs, are under active development. GANSU is an evolving framework that continues to incorporate diverse quantum chemical methods and algorithms, with the goal of providing a high-performance GPU-native environment for a wide range of electronic structure calculations.

Our experience with GANSU confirms that the careful integration of GPU-accelerated kernels at the workflow level is essential for high performance in complex scientific applications. The design choices described here may inform the development of GPU-accelerated frameworks in other computational domains.

Author Contributions: Conceptualization, Y.I. and K.N.; methodology, Y.I.; software, Y.I. and S.T.; formal analysis, Y.I.; investigation, Y.I.; resources, Y.I., K.N., and A.K.; data curation, S.T.; writing—original draft, Y.I.; writing—review and editing, S.T.; supervision, K.N.; project administration, K.N. and A.K.; and funding acquisition, A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The GANSU source code is available at <https://github.com/Yasuaki-Ito/GANSU/> (accessed on 26 March 2026).

Acknowledgments: The authors thank Kanta Suzuki, Haruto Fujii, Nobuya Yokogawa, and Hong Gao for their contributions to the development of individual computational kernels in GANSU.

Conflicts of Interest: Authors Satoki Tsuji and Akihiko Kasagi were employed by the company Fujitsu Limited. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

GANSU	GPU Accelerated Numerical Simulation Utility
HF	Hartree–Fock
SCF	Self-consistent field
ERI	Electron repulsion integral
GPU	Graphics processing unit
HPC	High-performance computing
RI	Resolution of the identity
DIIS	Direct inversion in the iterative subspace
RHF	Restricted Hartree–Fock
UHF	Unrestricted Hartree–Fock
ROHF	Restricted open-shell Hartree–Fock
MD	McMurchie–Davidson
HGP	Head–Gordon–Pople
OS	Obara–Saika
MO	Molecular orbital
AO	Atomic orbital
MP2	Second-order Møller–Plesset perturbation theory
MP3	Third-order Møller–Plesset perturbation theory
FCI	Full configuration interaction
DAR	Distributed atomic reduction
WBA	Warp-based assignment
TBA	Thread-based assignment
TLR	Thread-local reduction

References

1. Ito, Y.; Tsuji, S.; Fujii, H.; Suzuki, K.; Yokogawa, N.; Nakano, K.; Kasagi, A. Introduction to Computational Quantum Chemistry for Computer Scientists. In *Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPSW), San Francisco, CA, USA, 27–31 May 2024*; IEEE: New York, NY, USA, 2024; pp. 273–282.
2. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, Australia, 6–11 August 2017*; PMLR, 2017; Volume 70, pp. 1263–1272.
3. Smith, J.S.; Isayev, O.; Roitberg, A.E. ANI-1: An Extensible Neural Network Potential with DFT Accuracy at Force Field Computational Cost. *Chem. Sci.* **2017**, *8*, 3192–3203. [[CrossRef](#)]
4. Suzuki, K.; Ito, Y.; Fujii, H.; Yokogawa, N.; Tsuji, S.; Nakano, K.; Kasagi, A. GPU Acceleration of Head–Gordon–Pople Algorithm. In *Proceedings of the 12th International Symposium on Computing and Networking (CANDAR), Naha, Japan, 26–29 November 2024*; IEEE: New York, NY, USA, 2024; pp. 115–124.
5. Fujii, H.; Ito, Y.; Yokogawa, N.; Suzuki, K.; Tsuji, S.; Nakano, K.; Parque, V.; Kasagi, A. Efficient GPU Implementation of the McMurchie–Davidson Method for Shell-Based ERI Computations. *Appl. Sci.* **2025**, *15*, 2572. [[CrossRef](#)]

6. Fujii, H.; Ito, Y.; Yokogawa, N.; Suzuki, K.; Tsuji, S.; Nakano, K.; Kasagi, A. A GPU Implementation of McMurchie–Davidson Algorithm for Two-Electron Repulsion Integral Computation. In *Proceedings of the 15th International Conference on Parallel Processing and Applied Mathematics (PPAM 2024)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2025; Volume 15579, pp. 210–224.
7. Suzuki, K.; Ito, Y.; Fujii, H.; Yokogawa, N.; Tsuji, S.; Nakano, K.; Parque, V.; Kasagi, A. Efficient GPU Implementations of Three-Center Two-Electron Repulsion Integrals. *Concurr. Comput. Pract. Exp.* **2025**, *37*, e70328. [[CrossRef](#)]
8. Tsuji, S.; Ito, Y.; Fujii, H.; Yokogawa, N.; Suzuki, K.; Nakano, K.; Kasagi, A. Dynamic Screening of Two-Electron Repulsion Integrals in GPU Parallelization. In *Proceedings of the 12th International Symposium on Computing and Networking Workshops (CANDARW), Naha, Japan, 26–29 November 2024*; IEEE: New York, NY, USA, 2024; pp. 211–217.
9. Tsuji, S.; Ito, Y.; Nakano, K.; Kasagi, A. Efficient GPU-Accelerated Bulk Evaluation of the Boys Function for Quantum Chemistry. In *Proceedings of the 11th International Symposium on Computing and Networking (CANDAR), Matsue, Japan, 28 November–1 December 2023*; IEEE: New York, NY, USA, 2023; pp. 49–58.
10. Tsuji, S.; Ito, Y.; Nakano, K.; Kasagi, A. GPU Acceleration of the Boys Function Evaluation in Computational Quantum Chemistry. *Concurr. Comput. Pract. Exp.* **2025**, *37*, e8328. [[CrossRef](#)]
11. Yokogawa, N.; Ito, Y.; Tsuji, S.; Fujii, H.; Suzuki, K.; Nakano, K.; Kasagi, A. Parallel GPU Computation of Nuclear Attraction Integrals in Quantum Chemistry. In *Proceedings of the 12th International Symposium on Computing and Networking Workshops (CANDARW), Naha, Japan, 26–29 November 2024*; IEEE: New York, NY, USA, 2024; pp. 163–169.
12. Yokogawa, N.; Ito, Y.; Tsuji, S.; Fujii, H.; Suzuki, K.; Nakano, K.; Parque, V.; Kasagi, A. GPU-Accelerated One-Electron Integral Computation for Quantum Chemistry. *Concurr. Comput. Pract. Exp.* **2026**, *38*, e70628. [[CrossRef](#)]
13. Tsuji, S.; Ito, Y.; Fujii, H.; Yokogawa, N.; Suzuki, K.; Nakano, K.; Parque, V.; Kasagi, A. GPU-Accelerated Fock Matrix Computation with Efficient Reduction. *Appl. Sci.* **2025**, *15*, 4779. [[CrossRef](#)]
14. Tsuji, S.; Ito, Y.; Yokogawa, N.; Suzuki, K.; Nakano, K.; Parque, V.; Kasagi, A. A GPU Implementation of the Second- and Third-Order Møller–Plesset Perturbation Theory. In *Proceedings of the 13th International Symposium on Computing and Networking Workshops (CANDARW), Yamagata, Japan, 25–28 November 2025*; IEEE: New York, NY, USA, 2025; pp. 90–96.
15. Suzuki, K.; Ito, Y.; Yokogawa, N.; Tsuji, S.; Nakano, K.; Parque, V.; Kasagi, A. GPU Acceleration of RI-RMP2 Correlation Energy Computation. In *Proceedings of the 13th International Symposium on Computing and Networking (CANDAR), Yamagata, Japan, 25–28 November 2025*; IEEE: New York, NY, USA, 2025; pp. 174–180.
16. Gao, H.; Ito, Y.; Nakano, K.; Imamura, S.; Kasagi, A.; Tsuji, S. Fully GPU-Accelerated Full Configuration Interaction for Exact Molecular Ground-state Energy Calculation. In *Proceedings of the 13th International Symposium on Computing and Networking (CANDAR), Yamagata, Japan, 25–28 November 2025*; IEEE: New York, NY, USA, 2025; pp. 48–57.
17. Nickolls, J.; Buck, I.; Garland, M.; Skadron, K. Scalable Parallel Programming with CUDA. *ACM Queue* **2008**, *6*, 40–53. [[CrossRef](#)]
18. NVIDIA Corporation. CUDA C++ Programming Guide. Version 12.9. 2025. Available online: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/> (accessed on 26 March 2026).
19. GANSU Development Team. GANSU: GPU Accelerated Numerical Simulation Utility. 2026. Available online: <https://github.com/Yasuaki-Ito/GANSU/> (accessed on 26 March 2026).
20. Frisch, M.J.; Trucks, G.W.; Schlegel, H.B.; Scuseria, G.E.; Robb, M.A.; Cheeseman, J.R.; Scalmani, G.; Barone, V.; Petersson, G.A.; Nakatsuji, H.; et al. *Gaussian 16 Revision C.01*; Gaussian, Inc.: Wallingford, CT, USA, 2016.
21. Barca, G.M.J.; Bertoni, C.; Carrington, L.; Datta, D.; De Silva, N.; Deustua, J.E.; Fedorov, D.G.; Gour, J.R.; Gunina, A.O.; Guidez, E.; et al. Recent developments in the general atomic and molecular electronic structure system. *J. Chem. Phys.* **2020**, *152*, 154102. [[CrossRef](#)]
22. Asadchev, A.; Allada, V.; Feber, J.; Gordon, M.S. Ab Initio Quantum Chemistry on GPU Cluster. *J. Chem. Theory Comput.* **2012**, *8*, 4166–4176. [[CrossRef](#)]
23. Barca, G.M.J.; Galvez-Vallejo, J.L.; Poole, D.L.; Rendell, A.P.; Gordon, M.S. High-Performance, Graphics Processing Unit-Accelerated Fock Build Algorithm. *J. Chem. Theory Comput.* **2020**, *16*, 7232–7238. [[CrossRef](#)]
24. Aprà, E.; Bylaska, E.J.; de Jong, W.A.; Govind, N.; Kowalski, K.; Straatsma, T.P.; Valiev, M.; van Dam, H.J.J.; Alexeev, Y.; Anchell, J.; et al. NWChem: Past, present, and future. *J. Chem. Phys.* **2020**, *152*, 184102. [[CrossRef](#)]
25. Smith, D.G.A.; Burns, L.A.; Simmonett, A.C.; Parrish, R.M.; Schieber, M.C.; Galvelis, R.; Kraus, P.; Kruse, H.; Di Remigio, R.; Alenaizan, A.; et al. PSI4 1.4: Open-Source Software for High-Throughput Quantum Chemistry. *J. Chem. Phys.* **2020**, *152*, 184108. [[CrossRef](#)]
26. Epifanovsky, E.; Gilbert, A.T.B.; Feng, X.; Lee, J.; Mao, Y.; Mardirossian, N.; Pokhilko, P.; White, A.F.; Coons, M.P.; Dempwolff, A.L.; et al. Software for the frontiers of quantum chemistry: An overview of developments in the Q-Chem 5 package. *J. Chem. Phys.* **2021**, *155*, 084801. [[CrossRef](#)]
27. Tornai, G.J.; Ladjanszki, I.; Rák, A.; Kis, G.; Cserey, G. Calculation of Quantum Chemical Two-Electron Integrals by Applying Compiler Technology on GPU. *J. Chem. Theory Comput.* **2019**, *15*, 5319–5331. [[CrossRef](#)] [[PubMed](#)]

28. Sun, Q.; Berkelbach, T.C.; Blunt, N.S.; Booth, G.H.; Guo, S.; Li, Z.; Liu, J.; McClain, J.D.; Sayfutyarova, E.R.; Sharma, S.; et al. PySCF: The Python-based simulations of chemistry framework. *WIREs Comput. Mol. Sci.* **2018**, *8*, e1340. [\[CrossRef\]](#)
29. Sun, Q.; Zhang, X.; Banerjee, S.; Bao, P.; Barbry, M.; Blunt, N.S.; Bogdanov, N.A.; Booth, G.H.; Chen, J.; Cui, Z.H.; et al. Recent developments in the PySCF program package. *J. Chem. Phys.* **2020**, *153*, 024109. [\[CrossRef\]](#) [\[PubMed\]](#)
30. Li, R.; Sun, Q.; Zhang, X.; Chan, G.K.L. Introducing GPU Acceleration into the Python-Based Simulations of Chemistry Framework. *J. Phys. Chem. A* **2025**, *129*, 1459–1468. [\[CrossRef\]](#)
31. Dupuis, M.; Rys, J.; King, H.F. Evaluation of Molecular Integrals over Gaussian Basis Functions. *J. Chem. Phys.* **1976**, *65*, 111–116. [\[CrossRef\]](#)
32. Seritan, S.; Bannwarth, C.; Fales, B.S.; Hohenstein, E.G.; Kokkila-Schumacher, S.I.L.; Luehr, N.; Snyder, J.W., Jr.; Song, C.; Titov, A.V.; Ufimtsev, I.S.; et al. TeraChem: Accelerating electronic structure and ab initio molecular dynamics with graphical processing units. *J. Chem. Phys.* **2020**, *152*, 224110. [\[CrossRef\]](#)
33. Ufimtsev, I.S.; Martinez, T.J. Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-Electron Integral Evaluation. *J. Chem. Theory Comput.* **2008**, *4*, 222–231. [\[CrossRef\]](#)
34. Ufimtsev, I.S.; Martinez, T.J. Quantum Chemistry on Graphical Processing Units. 3. Analytical Energy Gradients, Geometry Optimization, and First Principles Molecular Dynamics. *J. Chem. Theory Comput.* **2009**, *5*, 2619–2628. [\[CrossRef\]](#)
35. Manathunga, M.; Miao, Y.; Mu, D.; Gotz, A.W.; Merz, K.M., Jr. Parallel Implementation of Density Functional Theory Methods in the Quantum Interaction Computational Kernel Program. *J. Chem. Theory Comput.* **2020**, *16*, 4315–4326. [\[CrossRef\]](#)
36. Manathunga, M.; Jin, C.; Cruzeiro, V.W.D.; Miao, Y.; Mu, D.; Arumugam, K.; Keipert, K.; Aktulga, H.M.; Merz, K.M., Jr.; Gotz, A.W. Harnessing the Power of Multi-GPU Acceleration into the Quantum Interaction Computational Kernel Program. *J. Chem. Theory Comput.* **2021**, *17*, 3955–3966. [\[CrossRef\]](#)
37. Kussmann, J.; Ochsenfeld, C. Hybrid CPU/GPU Integral Engine for Strong-Scaling Ab Initio Methods. *J. Chem. Theory Comput.* **2017**, *13*, 3153–3159. [\[CrossRef\]](#)
38. Miao, Y.; Merz, K.M., Jr. Acceleration of Electron Repulsion Integral Evaluation on Graphics Processing Units via Use of Recurrence Relations. *J. Chem. Theory Comput.* **2013**, *9*, 965–976. [\[CrossRef\]](#)
39. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley: Boston, MA, USA, 1994.
40. Pulay, P. Convergence Acceleration of Iterative Sequences. The Case of SCF Iteration. *Chem. Phys. Lett.* **1980**, *73*, 393–398. [\[CrossRef\]](#)
41. Pulay, P. Improved SCF Convergence Acceleration. *J. Comput. Chem.* **1982**, *3*, 556–560. [\[CrossRef\]](#)
42. Császár, P.; Pulay, P. Geometry Optimization by Direct Inversion in the Iterative Subspace. *J. Mol. Struct.* **1984**, *114*, 31–34. [\[CrossRef\]](#)
43. Vahtras, O.; Almlöf, J.; Feyereisen, M.W. Integral Approximations for LCAO-SCF Calculations. *Chem. Phys. Lett.* **1993**, *213*, 514–518. [\[CrossRef\]](#)
44. Gill, P.M.W.; Johnson, B.G.; Pople, J.A. A Simple yet Powerful Upper Bound for Coulomb Integrals. *Chem. Phys. Lett.* **1994**, *217*, 65–68. [\[CrossRef\]](#)
45. Szabo, A.; Ostlund, N.S. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*; Dover Publications: Garden City, NY, USA, 1996.
46. Helgaker, T.; Jørgensen, P.; Olsen, J. *Molecular Electronic-Structure Theory*; John Wiley & Sons: Hoboken, NJ, USA, 2000.
47. Boys, S.F. Electronic Wave Functions—I. A General Method of Calculation for the Stationary States of Any Molecular System. *Proc. R. Soc. Lond. A* **1950**, *200*, 542–554. [\[CrossRef\]](#)
48. Roothaan, C.C.J. New Developments in Molecular Orbital Theory. *Rev. Mod. Phys.* **1951**, *23*, 69–89. [\[CrossRef\]](#)
49. Pople, J.A.; Nesbet, R.K. Self-Consistent Orbitals for Radicals. *J. Chem. Phys.* **1954**, *22*, 571–572. [\[CrossRef\]](#)
50. Obara, S.; Saika, A. Efficient Recursive Computation of Molecular Integrals over Cartesian Gaussian Functions. *J. Chem. Phys.* **1986**, *84*, 3963–3974. [\[CrossRef\]](#)
51. McMurchie, L.E.; Davidson, E.R. One- and Two-Electron Integrals over Cartesian Gaussian Functions. *J. Comput. Phys.* **1978**, *26*, 218–231. [\[CrossRef\]](#)
52. Head-Gordon, M.; Pople, J.A. A Method for Two-Electron Gaussian Integral and Integral Derivative Evaluation Using Recurrence Relations. *J. Chem. Phys.* **1988**, *89*, 5777–5786. [\[CrossRef\]](#)
53. Gill, P.M.W.; Pople, J.A. The PRISM Algorithm for Two-Electron Integrals. *Int. J. Quantum Chem.* **1991**, *40*, 753–772. [\[CrossRef\]](#)
54. Gill, P.M.W. Molecular Integrals over Gaussian Basis Functions. In *Advances in Quantum Chemistry*; Sabin, J.R., Zerner, M.C., Eds.; Academic Press: San Diego, CA, USA, 1994; Volume 25, pp. 141–205.
55. Hall, G.G. The Molecular Orbital Theory of Chemical Valency VIII. A Method of Calculating Ionization Potentials. *Proc. R. Soc. Lond. A* **1951**, *205*, 541–552. [\[CrossRef\]](#)
56. Almlöf, J.; Faegri, K., Jr.; Korsell, K. Principles for a Direct SCF Approach to LCAO-MO Ab-Initio Calculations. *J. Comput. Chem.* **1982**, *3*, 385–399. [\[CrossRef\]](#)

57. Møller, C.; Plesset, M.S. Note on an Approximation Treatment for Many-Electron Systems. *Phys. Rev.* **1934**, *46*, 618–622. [[CrossRef](#)]
58. Knowles, P.J.; Handy, N.C. A New Determinant-Based Full Configuration Interaction Method. *Chem. Phys. Lett.* **1984**, *111*, 315–321. [[CrossRef](#)]
59. Davidson, E.R. The Iterative Calculation of a Few of the Lowest Eigenvalues and Corresponding Eigenvectors of Large Real-Symmetric Matrices. *J. Comput. Phys.* **1975**, *17*, 87–94. [[CrossRef](#)]
60. Nocedal, J.; Wright, S.J. *Numerical Optimization*, 2nd ed.; Springer: New York, NY, USA, 2006.
61. Fletcher, R.; Reeves, C.M. Function Minimization by Conjugate Gradients. *Comput. J.* **1964**, *7*, 149–154. [[CrossRef](#)]
62. Polak, E.; Ribière, G. Note sur la convergence de méthodes de directions conjuguées. *Rev. Française Informat. Rech. Opérationnelle* **1969**, *3*, 35–43. [[CrossRef](#)]
63. Hestenes, M.R.; Stiefel, E. Methods of Conjugate Gradients for Solving Linear Systems. *J. Res. Natl. Bur. Stand.* **1952**, *49*, 409–436. [[CrossRef](#)]
64. Dai, Y.H.; Yuan, Y. A Nonlinear Conjugate Gradient Method with a Strong Global Convergence Property. *SIAM J. Optim.* **1999**, *10*, 177–182. [[CrossRef](#)]
65. Neese, F.; Wennmohs, F.; Becker, U.; Riplinger, C. The ORCA quantum chemistry program package. *J. Chem. Phys.* **2020**, *152*, 224108. [[CrossRef](#)]
66. Neese, F. Software update: The ORCA program system—Version 5.0. *WIREs Comput. Mol. Sci.* **2022**, *12*, e1606. [[CrossRef](#)]
67. Neese, F.; Wennmohs, F.; Hansen, A.; Becker, U. Efficient, approximate and parallel Hartree–Fock and hybrid DFT calculations. A ‘chain-of-spheres’ algorithm for the Hartree–Fock exchange. *Chem. Phys.* **2009**, *356*, 98–109. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.