The Journal of Chemical Physics ARTICLE scitation.org/journal/jcp

GQCP: The Ghent Quantum Chemistry Package

Cite as: J. Chem. Phys. 155, 084802 (2021); doi: 10.1063/5.0057515
Submitted: 21 May 2021 • Accepted: 7 July 2021 •
Published Online: 26 August 2021

Laurent Lemmens, Xeno De Vriendt, Daria Tolstykh, Tobias Huysentruyt, Patrick Bultinck,[a] and Guillaume Acke

AFFILIATIONS

Ghent Quantum Chemistry Group, Department of Chemistry, Ghent University, Krijgslaan 281 (S3), B-9000 Gent, Belgium

[a]Author to whom correspondence should be addressed: Patrick.Bultinck@UGent.be

ABSTRACT

The Ghent Quantum Chemistry Package (GQCP) is an open-source electronic structure software package that aims to provide an intuitive and expressive software framework for electronic structure software development. Its high-level interfaces (accessible through C++ and Python) have been specifically designed to correspond to theoretical concepts, while retaining access to lower-level intermediates and allowing structural run-time modifications of quantum chemical solvers. GQCP focuses on providing quantum chemical method developers with the computational "building blocks" that allow them to flexibly develop proof of principle implementations for new methods and applications up to the level of two-component spinor bases.

I. INTRODUCTION

Software in the computational molecular sciences community is currently undergoing transformations in order to manage its ever-increasing complexity and to prepare itself for the advent of exascale hardware architectures.[1,2] In an attempt to solve both problems at once, modularity has emerged as a software paradigm. Underlying this paradigm shift is the engineering principle of the *separation of concerns*,[3] in which the full software stack is composed of a set of modular components that each perform a specialized task. This modular design allows for developments to be made in each module independently, which greatly decreases maintenance efforts,[4] and targeting new kinds of hardware can be offloaded to specialized adapter modules instead of requiring large-scale rewrites at the level of the solution of the quantum chemical problem itself.[5]

Many software libraries are currently targeting modularity,[4,6–13] but with it comes the requirement of interoperability between different modules. In an attempt to overcome this obstacle, there has been a recent focus on providing programmatic communication through APIs (application programming interfaces) that specify instructions and transmit data,[4,11,13–21] and some libraries have adopted support for custom-written plug-ins.[14,18,20,21] Another recent trend is allowing access to lower-level intermediates, which are typically the matrix representations of one- and two-electron integrals.[11,14,18,20–23]

Even though many general-purpose as well as specific niche software packages exist, there is still room for software that specializes in the rapid prototyping of new proof of principles, especially if it incorporates features that are not yet omnipresent in the current electronic structure software ecosystem. Among others, such features are as follows: the possibility to modify the algorithmic structure of quantum chemical solvers without requiring the re-compilation of the whole project, the ability to work with real and complex orbitals, and the flexibility to use restricted, unrestricted or generalized[24,25] orbitals.

In the open-source Ghent Quantum Chemistry Package (GQCP), we have implemented computational objects that bear semantic correspondence with the concepts that appear in the electronic structure theory, while keeping the aforementioned features in mind. Rather than offering highly optimized and specialized routines, our goal is to deliver intuitive and reusable software components (both in C++ and Python) in a domain-specific language, in order to enable method developers to focus on the high-level formulation of prototypes for new computational procedures. By implementing core "building blocks," our design embodies the hypothesis that new research can emerge from the combination of carefully thought-out high-level software objects. Furthermore, we offer the flexibility of allowing run-time structural modifications of quantum chemical solvers and we provide access to all lower-level intermediates. It is this focus on expressiveness that renders our package complementary to other related initiatives such as Psi4,[20] PySCF,[21] and ChronusQ.[26]

In order to elucidate the two-component concepts that lie at the core of GQCP, we first provide a didactical introduction to

the theoretical framework behind spinor bases (Sec. II). We then give an overview of how we have implemented the underlying concepts, how we have targeted flexible algorithmic design, and provide some illustrative applications that highlight the capabilities of GQCP (Sec. III). We also detail how we adhere to the current best practices in quantum chemical software engineering (Sec. IV).

II. THEORETICAL CONCEPTS

In electronic structure theory, solving the Schrödinger equation

$$\hat{\mathcal{H}}|\Psi\rangle = E|\Psi\rangle, \tag{1}$$

amounts to finding the eigenvectors $|\Psi\rangle$ of the Hamiltonian of the system, $\hat{\mathcal{H}}$. Unfortunately, only for the smallest systems the Hamiltonian can be diagonalized, and thus, approximate methods are required. In such methods, the wave function is parameterized by a set of model parameters, which define the approximate ground state $|\Psi(\mathbf{p})\rangle$ (and possibly approximate excited states) after an optimization procedure.[27]

Lykos and Pratt[28] report how Löwdin considers that approximate methods pose a dilemma with respect to the symmetry properties imposed by the constants of motion $\hat{\Lambda}$ (with $[\hat{\mathcal{H}}, \hat{\Lambda}] = 0$) of the system. Indeed, as the approximations $|\Psi(\mathbf{p})\rangle$ are not necessarily eigenfunctions of the Hamiltonian, imposing those symmetries on approximate wave function models implies placing constraints that will restrict their variational freedom and hence increase the energy. On the other hand, breaking such symmetry requirements leads to lower energy solutions that are mixtures of different symmetry types, which among others may lead to worse performance for properties other than the energy.[29–33]

As the total projected spin $\hat{S}_z$ is a symmetry of the spinless field-free molecular Hamiltonian, wave function models are usually chosen to be eigenvectors of $\hat{S}_z$. In order to accomplish this, the one-electron functions that are used in the wave function model are then chosen to be eigenfunctions of individual electron projected spin. These one-electron functions are often called *spin-orbitals*,[27]

$$\phi_{p\alpha}(\mathbf{x}) = \phi_p^{\alpha}(\mathbf{r})\alpha(\omega) \tag{2}$$

$$\phi_{p\beta}(\mathbf{x}) = \phi_p^{\beta}(\mathbf{r})\beta(\omega), \tag{3}$$

and solely consist of a space part multiplied by a spin part. Here, the space–spin coordinate $\mathbf{x} = (\mathbf{r}, \omega)$ collects the space and spin coordinates $\mathbf{r}$ and ω, respectively. Inspired by Löwdin's argument,[28] the most general admissible form for the one-electron functions would however be one that represents an expansion in the complete spin basis. Such one-electron functions are *spinors*,[24,25,27]

$$\phi_P(\mathbf{x}) = \phi_P^{\alpha}(\mathbf{r})\alpha(\omega) + \phi_P^{\beta}(\mathbf{r})\beta(\omega), \tag{4}$$

which simultaneously hold spin-α and spin-β character. Such spinors arise naturally when the Hamiltonian is no longer spin-free, for example, when the spin–orbit coupling interaction is included,[33–36] or when the presence of an external magnetic field explicitly requires the treatment of electron spin;[33,37–39] although, for uniform magnetic fields applied along the z-axis, spin-orbitals are sufficient. Spinors are furthermore used in so-called broken-symmetry theories and their application has led to lower energies,[29–33] but the lower energy does come at the cost of the loss of true quantum numbers. As this may turn out to lead to poor chemical properties other than the energy, this conundrum has sparked recent approaches toward symmetry restoration.[40–45]

In the following sections, we explore the theory of second quantization in terms of spinors and contrast it with the simplifications that arise when using a spin-orbital basis. The distinctions that appear in these formalisms are discussed at the level of the orbitals, one- and two-electron operators and density matrices, in order to prepare the discussion of the semantic design of GQCP's reusable software objects in Sec. III.

A. Spinors are two-component orbitals

In the spin basis, we can alternatively represent the spinor (4) as the two-component function,

$$\boldsymbol{\phi}_P(\mathbf{r}) = \begin{pmatrix} \phi_P^{\alpha}(\mathbf{r}) \\ \phi_P^{\beta}(\mathbf{r}) \end{pmatrix}. \tag{5}$$

The scalar product of two spinors can be calculated using the multiplication of the row vector $\boldsymbol{\phi}_P^{\dagger}$ and the column vector $\boldsymbol{\phi}_Q$,

$$\int \mathrm{d}\mathbf{r}\boldsymbol{\phi}_P^{\dagger}(\mathbf{r})\boldsymbol{\phi}_Q(\mathbf{r}) = \int \mathrm{d}\mathbf{r}\phi_P^{\alpha *}(\mathbf{r})\phi_Q^{\alpha}(\mathbf{r}) + \int \mathrm{d}\mathbf{r}\phi_P^{\beta *}(\mathbf{r})\phi_Q^{\beta}(\mathbf{r}), \tag{6}$$

equivalent to the scalar product

$$\int \mathrm{d}\mathbf{x}\phi_P^{*}(\mathbf{x})\phi_Q(\mathbf{x}) = \int \mathrm{d}\mathbf{r}\phi_P^{\alpha *}(\mathbf{r})\phi_Q^{\alpha}(\mathbf{r}) + \int \mathrm{d}\mathbf{r}\phi_P^{\beta *}(\mathbf{r})\phi_Q^{\beta}(\mathbf{r}), \tag{7}$$

where we have used the orthonormality of the spin functions α and β. In an orthonormal basis of M spinors $\{\boldsymbol{\phi}_P(\mathbf{r})|P = 1, \ldots, M\}$, the corresponding elementary creation and annihilation operators, $\hat{a}_P^{\dagger}$ and $\hat{a}_P$, obey the usual fermionic anticommutation relations[27]

$$\left[\hat{a}_P, \hat{a}_Q^{\dagger}\right]_+ = \delta_{PQ} \tag{8}$$

and

$$\left[\hat{a}_P, \hat{a}_Q\right]_+ = 0. \tag{9}$$

The associated fermionic field operators

$$\hat{\boldsymbol{\phi}}^{\dagger}(\mathbf{r}) = \sum_P \boldsymbol{\phi}_P^{\dagger}(\mathbf{r})\hat{a}_P^{\dagger} \tag{10}$$

and

$$\hat{\boldsymbol{\phi}}(\mathbf{r}) = \sum_P \boldsymbol{\phi}_P(\mathbf{r})\hat{a}_P \tag{11}$$

can be used to express one- and two-electron operators in a spinor basis, which is a procedure that is commonly referred to as the *quantization*[27,46,47] of the one- and two-electron operators.

The coordinate representation (indicated by the superscript c) of a one-electron operator f^c for a system of N electrons is[27]

$$f^c(\mathbf{r}_1,\ldots,\mathbf{r}_N) = \sum_{i=1}^{N} f^c(\mathbf{r}_i), \tag{12}$$

where each individual one-electron operator $f^c(\mathbf{r}_i)$ is mathematically represented by a (2×2) matrix operator,

$$f^c(\mathbf{r}_i) = \begin{pmatrix} f^{c,\alpha\alpha}(\mathbf{r}_i) & f^{c,\alpha\beta}(\mathbf{r}_i) \\ f^{c,\beta\alpha}(\mathbf{r}_i) & f^{c,\beta\beta}(\mathbf{r}_i) \end{pmatrix}. \tag{13}$$

Each underlying one-electron operator $f^{c,\sigma\tau}(\mathbf{r}_i)$ is a scalar operator, which means that it only acts on one of the components of a spinor and not on the two-component spinors themselves. Using the field operators (10) and (11), the second-quantized analogue of (12) becomes

$$\hat{f} = \int \mathrm{d}\mathbf{r}\hat{\boldsymbol{\phi}}^\dagger(\mathbf{r})\, f^c(\mathbf{r})\, \hat{\boldsymbol{\phi}}(\mathbf{r}) \tag{14}$$

$$= \sum_{PQ} f_{PQ}\hat{a}_P^\dagger\hat{a}_Q, \tag{15}$$

where the parameters f_{PQ} are the one-electron integrals,

$$f_{PQ} = \int \mathrm{d}\mathbf{r}\boldsymbol{\phi}_P^\dagger(\mathbf{r})\, f^c(\mathbf{r})\, \boldsymbol{\phi}_Q(\mathbf{r}). \tag{16}$$

Analogously, a two-electron operator

$$g^c(\mathbf{r}_1,\ldots,\mathbf{r}_N) = \frac{1}{2}\sum_{i\neq j}^{N} g^c(\mathbf{r}_i,\mathbf{r}_j) \tag{17}$$

can be represented as

$$\hat{g} = \frac{1}{2}\iint \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\hat{\boldsymbol{\phi}}^\dagger(\mathbf{r}_1)\hat{\boldsymbol{\phi}}^\dagger(\mathbf{r}_2)\, g^c(\mathbf{r}_1,\mathbf{r}_2)\, \hat{\boldsymbol{\phi}}(\mathbf{r}_2)\hat{\boldsymbol{\phi}}(\mathbf{r}_1) \tag{18}$$

$$= \frac{1}{2}\sum_{PQRS} g_{PQRS}\, \hat{a}_P^\dagger\hat{a}_R^\dagger\hat{a}_S\hat{a}_Q, \tag{19}$$

where the two-electron parameters for the Coulomb repulsion operator are given by

$$g_{PQRS} = \sum_{\sigma\tau}(P\sigma Q\sigma|R\tau S\tau), \tag{20}$$

with $(P\sigma Q\sigma|R\tau S\tau)$ a Coulomb integral over the spinor components in Mulliken notation,

$$(P\sigma Q\sigma|R\tau S\tau) = \iint \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\phi_P^{\sigma *}(\mathbf{r}_1)\phi_Q^{\sigma}(\mathbf{r}_1) \times \frac{1}{\|\mathbf{r}_1-\mathbf{r}_2\|}\phi_R^{\tau *}(\mathbf{r}_2)\phi_S^{\tau}(\mathbf{r}_2). \tag{21}$$

Here, we have used σ and τ as generic labels for α and β.

Given a (normalized) wave function $|\Psi\rangle$, we define the one-electron density matrix (1-DM) as

$$D_{PQ} = \langle\Psi|\hat{a}_P^\dagger\hat{a}_Q|\Psi\rangle \tag{22}$$

and the two-electron density matrix (2-DM) as

$$d_{PQRS} = \langle\Psi|\hat{a}_P^\dagger\hat{a}_R^\dagger\hat{a}_S\hat{a}_Q|\Psi\rangle. \tag{23}$$

With these definitions, the expectation value of a one-electron operator can be written as

$$\langle\Psi|\hat{f}|\Psi\rangle = \sum_{PQ} f_{PQ}D_{PQ} \tag{24}$$

and the expectation value of a two-electron operator as

$$\langle\Psi|\hat{g}|\Psi\rangle = \frac{1}{2}\sum_{PQRS} g_{PQRS}\, d_{PQRS}. \tag{25}$$

B. Imposing spin symmetries leads to spin-orbital bases

When both components of a spinor are non-zero, it is not an eigenvector of the Pauli matrix σ_z. However, if K_α spinors only have a non-zero α-component and K_β spinors (with $M = K_\alpha + K_\beta$) only have a non-zero β-component, we can rearrange them as

$$\left\{\begin{pmatrix}\phi_1^\alpha(\mathbf{r})\\0\end{pmatrix},\ldots,\begin{pmatrix}\phi_{K_\alpha}^\alpha(\mathbf{r})\\0\end{pmatrix},\begin{pmatrix}0\\\phi_{K_\alpha+1}^\beta(\mathbf{r})\end{pmatrix},\ldots,\begin{pmatrix}0\\\phi_{K_\alpha+K_\beta}^\beta(\mathbf{r})\end{pmatrix}\right\} \tag{26}$$

and relabel them according to the label of their non-zero component,

$$\boldsymbol{\phi}_P(\mathbf{r}) \to \boldsymbol{\phi}_{p\sigma}(\mathbf{r}) = \begin{cases} \begin{pmatrix}\phi_p^\alpha(\mathbf{r})\\0\end{pmatrix} & 1\le P\le K_\alpha \quad 1\le p\le K_\alpha \\ \begin{pmatrix}0\\\phi_p^\beta(\mathbf{r})\end{pmatrix} & K_\alpha < P\le M \quad 1\le p\le K_\beta. \end{cases} \tag{27}$$

We note in passing that Eqs. (27) and (2) are alternative representations of a spin-orbital, in the same way that Eqs. (5) and (4) are alternative representations of a spinor.

The resulting compound lowercase indices $p\sigma$ are used as labels for *spin-orbitals*. Since these spin-orbitals have either of their components set to zero, they *are* eigenvectors of the Pauli matrix σ_z. In a spin-orbital basis, the field operators (10) and (11) can be rewritten as

$$\hat{\boldsymbol{\phi}}^\dagger(\mathbf{r}) = \sum_\sigma\sum_p \boldsymbol{\phi}_{p\sigma}^\dagger(\mathbf{r})\hat{a}_{p\sigma}^\dagger \tag{28}$$

and

$$\hat{\boldsymbol{\phi}}(\mathbf{r}) = \sum_\sigma\sum_p \boldsymbol{\phi}_{p\sigma}(\mathbf{r})\hat{a}_{p\sigma}, \tag{29}$$

with the associated elementary anticommutation relations

$$\left[\hat{a}_{p\sigma},\hat{a}_{q\tau}^\dagger\right]_+ = \delta_{\sigma\tau}\delta_{pq} \tag{30}$$

and

$$[\hat{a}_{p\sigma}, \hat{a}_{q\tau}]_+ = 0. \tag{31}$$

The expressions for the second-quantized representations of the one- and two-electron operators (12) and (17) then reduce to

$$\hat{f} = \sum_{\sigma\tau}\sum_{pq} f_{p\sigma,q\tau}\hat{a}^\dagger_{p\sigma}\hat{a}_{q\tau} \tag{32}$$

and

$$\hat{g} = \frac{1}{2}\sum_{\sigma\tau}\sum_{pqrs}(p\sigma q\sigma|r\tau s\tau)\hat{a}^\dagger_{p\sigma}\hat{a}^\dagger_{r\tau}\hat{a}_{s\tau}\hat{a}_{q\sigma}. \tag{33}$$

Similarly, 1- and 2-DMs reduce to

$$D^{\sigma\tau}_{pq} = \langle\Psi|\hat{a}^\dagger_{p\sigma}\hat{a}_{q\tau}|\Psi\rangle \tag{34}$$

and

$$d^{\sigma\sigma\tau\tau}_{pqrs} = \langle\Psi|\hat{a}^\dagger_{p\sigma}\hat{a}^\dagger_{r\tau}\hat{a}_{s\tau}\hat{a}_{q\sigma}|\Psi\rangle, \tag{35}$$

which can be used in the following expectation values:

$$\langle\Psi|\hat{f}|\Psi\rangle = \sum_{\sigma\tau}\sum_{pq} f_{p\sigma,q\tau}D^{\sigma\tau}_{pq} \tag{36}$$

and

$$\langle\Psi|\hat{g}|\Psi\rangle = \frac{1}{2}\sum_{\sigma\tau}\sum_{pqrs}(p\sigma q\sigma|r\tau s\tau)d^{\sigma\sigma\tau\tau}_{pqrs}. \tag{37}$$

In the special case that the one-electron operator (13) is zero in its off-diagonal elements and equal in its diagonal elements, i.e., $f^{c,\alpha\alpha} = f^{c,\beta\beta}$, and the components of the α-spin-orbitals are equal to those of the β-spin-orbitals, its second-quantized representation can be written as

$$\hat{f} = \sum_{pq} f_{pq}\hat{E}_{pq}, \tag{38}$$

where we have introduced the one-electron singlet excitation operator[27,47]

$$\hat{E}_{pq} = \sum_{\sigma}\hat{a}^\dagger_{p\sigma}\hat{a}_{q\sigma}. \tag{39}$$

Similarly, in this case, the two-electron Coulomb operator can be represented by

$$\hat{g} = \frac{1}{2}\sum_{pqrs} g_{pqrs}\hat{e}_{pqrs}, \tag{40}$$

where we have introduced the two-electron singlet excitation operator[27]

$$\hat{e}_{pqrs} = \sum_{\sigma\tau}\hat{a}^\dagger_{p\sigma}\hat{a}^\dagger_{r\tau}\hat{a}_{s\tau}\hat{a}_{q\sigma}. \tag{41}$$

In this (restricted) basis, we define the orbital 1- and 2-DMs[27]

$$D_{pq} = \sum_{\sigma} D^{\sigma\sigma}_{pq} \tag{42}$$

and

$$d_{pqrs} = \sum_{\sigma\tau} d^{\sigma\sigma\tau\tau}_{pqrs}, \tag{43}$$

which are used in the following expectation values:

$$\langle\Psi|\hat{f}|\Psi\rangle = \sum_{pq} f_{pq}D_{pq} \tag{44}$$

and

$$\langle\Psi|\hat{g}|\Psi\rangle = \frac{1}{2}\sum_{pqrs}(pq|rs)d_{pqrs}. \tag{45}$$

C. Expanding spinors in scalar basis functions leads to coefficient matrices

By introducing a basis of K scalar basis functions $\{\chi_\mu(\mathbf{r}) \mid \mu = 1, \ldots, K\}$, we can express each of the components of the spinors as a linear combination of these basis functions,

$$\phi^\sigma_P(\mathbf{r}) = \sum_{\mu}\chi_\mu(\mathbf{r})C^\sigma_{\mu P}. \tag{46}$$

Since we use the same scalar basis for both components, the expansion coefficients $\{C^\sigma_{\mu P}\}$ can be arranged into the square $(M \times M)$-coefficient matrix $\mathbf{C}$ (with $M = 2K$),

$$\mathbf{C} = \begin{pmatrix} C^\alpha_{11} & \cdots & C^\alpha_{1K} & C^\alpha_{1(K+1)} & \cdots & C^\alpha_{1M} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ C^\alpha_{K1} & \cdots & C^\alpha_{KK} & C^\alpha_{K(K+1)} & \cdots & C^\alpha_{KM} \\ C^\beta_{11} & \cdots & C^\beta_{1K} & C^\beta_{1(K+1)} & \cdots & C^\beta_{1M} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ C^\beta_{K1} & \cdots & C^\beta_{KK} & C^\beta_{K(K+1)} & \cdots & C^\beta_{KM} \end{pmatrix}, \tag{47}$$

in which every column P collects the expansion coefficients of the spinor ϕ_P. In a spin-orbital basis, the coefficient matrix thus has a blocked diagonal form,

$$\mathbf{C} = \begin{pmatrix} \mathbf{C}^\alpha & \mathbf{0} \\ \mathbf{0} & \mathbf{C}^\beta \end{pmatrix}, \tag{48}$$

and with the further restriction that the components of the α-spin-orbitals are equal to those of the β-spin-orbitals, the spinor coefficient matrix can be written as

$$\mathbf{C} = \begin{pmatrix} \mathbf{C}' & \mathbf{0} \\ \mathbf{0} & \mathbf{C}' \end{pmatrix}, \tag{49}$$

with $\mathbf{C}' = \mathbf{C}^\alpha = \mathbf{C}^\beta$.

Using the expansion (46), we can calculate the one-electron integrals (16) as

$$f_{PQ} = \sum_{\sigma\tau}\sum_{\mu\nu} C^{\sigma *}_{\mu P} f^{\sigma\tau}_{\mu\nu} C^{\tau}_{\nu Q}, \tag{50}$$

where we have introduced the one-electron integrals in the underlying scalar basis as

$$f^{\sigma\tau}_{\mu\nu} = \int \mathrm{d}\mathbf{r}\chi^*_\mu(\mathbf{r})f^{c,\sigma\tau}(\mathbf{r})\chi_\nu(\mathbf{r}). \tag{51}$$

We should note that Eq. (50) is equivalent to the matrix multiplication

$$\mathbf{f} = \mathbf{C}^{\dagger}\begin{pmatrix} \mathbf{f}^{\alpha\alpha} & \mathbf{f}^{\alpha\beta} \\ \mathbf{f}^{\beta\alpha} & \mathbf{f}^{\beta\beta} \end{pmatrix}\mathbf{C}, \tag{52}$$

where $\mathbf{f}^{\sigma\tau}$ is the matrix representation of the scalar operator $f^{c,\sigma\tau}$ in terms of the underlying scalar basis. The two-electron Coulomb integrals (20) can be calculated using the expansion (46) as

$$g_{PQRS} = \sum_{\sigma\tau}\sum_{\mu\nu\rho\lambda} C^{\sigma *}_{\mu P} C^{\sigma}_{\nu Q} C^{\tau *}_{\rho R} C^{\tau}_{\lambda S} (\mu\nu|\rho\lambda), \tag{53}$$

in which $(\mu\nu|\rho\lambda)$ is a Coulomb integral over scalar basis functions,

$$(\mu\nu|\rho\lambda) = \iint \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 \chi^*_{\mu}(\mathbf{r}_1)\chi_{\nu}(\mathbf{r}_1)\frac{1}{\|\mathbf{r}_1 - \mathbf{r}_2\|}\chi^*_{\rho}(\mathbf{r}_2)\chi_{\lambda}(\mathbf{r}_2). \tag{54}$$

III. DESIGN AND APPLICATIONS

Before highlighting some of GQCP's illustrative capabilities later on in this section, we first detail GQCP's overarching design choices. We refer the reader to our documentation[82] for a more comprehensive and continuously updated overview of the APIs.

GQCP offers interfaces both in C++ (in its core library) and Python (through the use of Pybind11).[48] In general, we refer to real-valued (`double`) or complex-valued (`complex<double>`) types with the subscripts `_d` and `_cd` in our Python bindings. For example, the real-valued C++ type `ScalarRSQOneElectronOperator<double>` (cf. Sec. III A) is bound to the Python class `ScalarRSQOneElectronOperator_d`. In order to make this manuscript as readable as possible, we have opted not to pollute the *text* with these template parameters or subscripts, but the *code examples* do reflect the actual use of the APIs. Note that the code listings use GQCP's Python interface and implicitly assume that the `gqcpy` module has been imported:

```
1 import gqcpy
```

Listing 1: Importing the `gqcpy` Python module.

A. Orbital bases, operators, density matrices, and transformations

Based on the theoretical foundations covered in Sec. II, we have distinguished and implemented three distinct orbital bases. If the spinor coefficient matrix (47) is fully dense, we use the terminology *generalized spinor basis* with the corresponding class `GSpinorBasis`. If the spinor coefficient matrix contains off-diagonal zero blocks, we distinguish two kinds of spin-orbital bases. When the diagonal blocks are equal, the spin-orbital basis is considered to be *restricted* and the corresponding type is `RSpinOrbitalBasis`, and when they are not (necessarily), the spin-orbital basis is said to be *unrestricted* and is represented by an instance of `USpinOrbitalBasis`. Note that we have borrowed this nomenclature from three spin symmetry-related formulations of Hartree–Fock theory: RHF, UHF, and GHF.[49,50] We note that the restricted, unrestricted, and generalized distinctions are propagated throughout our APIs through the prefix `Y=R/U/G`.

An orbital basis object serves as the starting point for the second-quantized representation of one- and two-electron operators. The result of such a quantization procedure is an instance of `XYSQOneElectronOperator` or `XYSQTwoElectronOperator`, where `SQ` is used as an abbreviation of `SecondQuantized`. The prefix `X=Scalar, Vector, Tensor` is used in order to accommodate the tensorial structure of the operators. For example, second-quantized scalar operators such as the kinetic energy operator or the nuclear attraction operator can be represented by `ScalarYSQOneElectronOperator`. Vector operators, such as the dipole operator, the electronic spin operator, or the linear momentum operator can be represented by an instance of `VectorYSQOneElectronOperator`. Higher-order tensor operators, such as the quadrupole operator, can finally be represented by `TensorYSQOneElectronOperator`. Most of the methods that are usually only available for scalar operators are thus made available for higher-order tensor operators, such as basis transformations, the calculation of expectation values, and the evaluation in a Fock (sub)space.

In GQCP, the Hamiltonian is regarded as a distinct operator. For most quantum chemical applications, it suffices to represent it as a sum of one- and two-electron operators. To this end, GQCP provides the three types of `YSQHamiltonian` depending on which orbital basis the Hamiltonian is expressed in, allowing the user to choose the most applicable type in their routines. We also support the specialized representation of model Hamiltonians such as the Hubbard Hamiltonian.

Generally, quantum chemical routines optimize a set of model parameters (such as the amplitudes in coupled cluster theories or the expansion coefficients in configuration interaction theories), given a specification of the orbital basis and the one- and two-electron operators that define the Hamiltonian of the system. From these model parameters, the one- and two-electron density matrices (1- and 2-DMs) are used to calculate expectation values of second-quantized operators. GQCP provides `G1DM` and `G2DM` as types to represent the 1- and 2-DMs [cf. Eqs. (22) and (23)] expressed in a generalized spinor basis. Following the nomenclature as used by Helgaker *et al.*,[27] the *restricted* density matrices are referred to as `Orbital1DM` and `Orbital2DM`. To designate density matrices with possibly different components related to α- and β-spin, we use the terminology `SpinResolved1DM` and `SpinResolved2DM`.

The types discussed above are dependent on the orbital basis in which they are expressed. In order to change their representation to that related to another orbital basis, GQCP provides the API `transformed(T)`, with `T` a suitable `YTransformation`. In this way, a `GTransformation` can be used to basis-transform generalized objects, but not unrestricted or restricted objects.

A schematic overview of the concepts introduced in this section and their most important relations is given in Fig. 1. Code listing 2 combines them in a simple example calculation of the GHF dipole moment. Note that in order to make this example as conceptually simple as possible, we assume that the user has calculated the GHF model parameters in the variable `ghf_parameters`, for example as a result of the calculation in code listing 4.

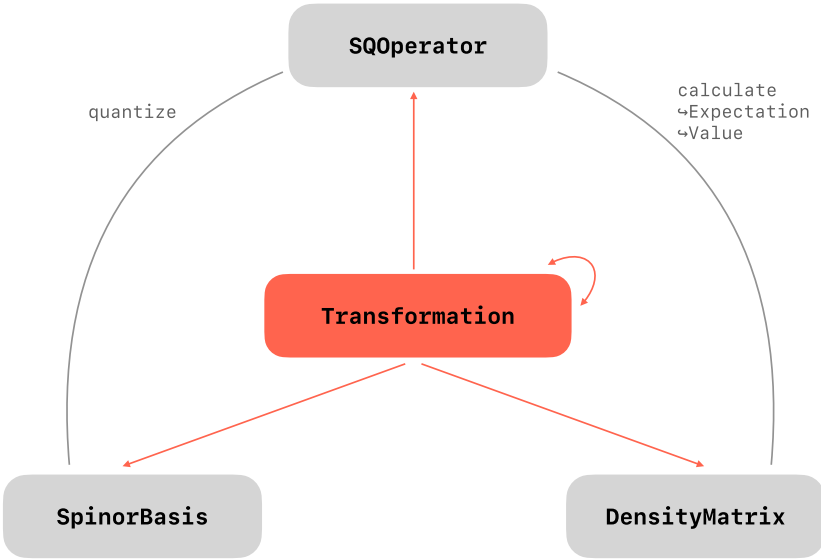

FIG. 1. Four high-level classes that GQCP offers and their quintessential relations. `SpinorBases` are linked to `SQOperators` through a `quantize` call, while `SQOperators` and `DensityMatrices` may be combined through `calculate ExpectationValue`. All four types are basis-transformable through the use of a `Transformation`.

```
# Set up the molecular system and spinor basis.
molecule = gqcpy.Molecule.ReadXYZ("CO.xyz", charge=0)
spinor_basis = gqcpy.GSpinorBasis_d(molecule, "6-31G**")

# (The optimization of the GHF wave function in the variable 'ghf_parameters' is omitted.)

# Calculate the electronic dipole moment in the AO basis.
dipole_op = spinor_basis.quantize(gqcpy.ElectronicDipoleOperator()) # 'op' for 'operator'.
D = ghf_parameters.calculateScalarBasis1DM() # The 1-DM in AO basis.
dipole_moment = dipole_op.calculateExpectationValue(D)
```

Listing 2: Having optimized the GHF wave function model in the variable `ghf_parameters` (omitted in this example), a user can use this code snippet to calculate the GHF dipole moment.

The optimized GHF coefficients can be used to transform the operator and density matrix objects so they are expressed with respect to the GHF MOs. Code example 3 shows this behavior and calculates the GHF dipole moment, but now uses the density matrix and dipole operator expressed with respect to the GHF MOs. Note that the results obtained after executing code snippet 3 are equal to those obtained after executing code snippet 2.

```
# Transform the dipole operator and density matrix from their AO representation to an MO representation, and calculate the dipole moment in the MO basis.
```

```
C = ghf_parameters.expansion() # The GHF spinor expansion coefficients.
dipole_op.transform(C)
D.transform(C)
dipole_moment = dipole_op.calculateExpectationValue(D)
```

Listing 3: Instead of calculating the GHF dipole moment through quantities expressed in the AO basis, users can calculate the GHF dipole moment using the dipole operator and density matrix in their MO representation. Note that listings 2 and 3 produce the same result.

Since the design of the computational objects reflects the theoretical framework behind spinor bases, GQCP can be used as a tool to explore electronic structure concepts in education,[11,51] both from a user's point of view and from a method developer's perspective.

B. The algorithm framework

Another way that GQCP targets development flexibility is by focusing on modifiable algorithms. Since almost all computational methods use an interactive scheme to solve a set of equations, being able to intervene in these iterative cycles both at compile-time and at run-time drastically increases the sustainability, reproducibility, and the speed of the development of new computational methods. In contrast to only being able to modify heuristic parameters such as particular thresholds, subspace dimensions, and damping factors, GQCP allows for an intuitive way to modify the *structure* of algorithms and solvers, opening up otherwise black-box quantum chemical methods.

In GQCP, algorithms are formulated as sequential applications of their constituting steps that can each modify a so-called environment, which acts as a calculation "cache" and collects the previously calculated intermediates. Pictorially, this is represented in Fig. 2. This reformulation not only allows for a more rapid implementation of modifications to particular solvers in the C++ core library language, but also provides the means for users to define their custom algorithmic steps in Python and modify algorithms at runtime by injecting them in the basic algorithm. We emphasize that such modifications do not require a re-compilation of the whole project. For an illustrative application of this design, we refer the reader to Sec. III C.

SDE's (Software Development Environment)[14] *module* design, where each module performs a particular task related to a quantum chemical method (such as calculating the energy, calculating the Fock matrix, etc.) is similar to what GQCP offers in its `Algorithm` framework: modules each perform their own functionality and have access to state, similarly to how different `Steps` may access an `Environment` and sequentially build up an `IterativeAlgorithm`. The main difference is that in GQCP, algorithms are not designed based on inheritance, but as subsequent applications of their underlying steps. As an example, in SDE, one would override the behavior of a plain `FockBuilder` with DIIS acceleration, while in GQCP, one would insert the DIIS step *in* the plain SCF solver to allow the acceleration to take place. Similarly, a density-damped solver may be engineered by inserting the dampening step in a plain SCF solver.

As an example of how solvers and environments are typically combined in a quantum chemical method, we discuss the calculation of the GHF energy and spinor expansion coefficients. The corresponding code example is found in listing 4. In the first section, the set-up is performed by instantiating a molecule, orbital basis, and Hamiltonian. Then, a solver and an environment (with an initial guess that solves the general diagonalization problem for the core Hamiltonian) are prepared and are provided to the `GHF QCMethod`, whose `optimize` call returns a `QCStructure`. This quantum chemical structure encapsulates (energy, optimized parameters)-pairs and allows for a general representation of ground-state and excited state optimized parameters. The code example concludes with the extraction of the ground state (electronic) energy and the corresponding optimized GHF model parameters from the computed `QCStructure`.

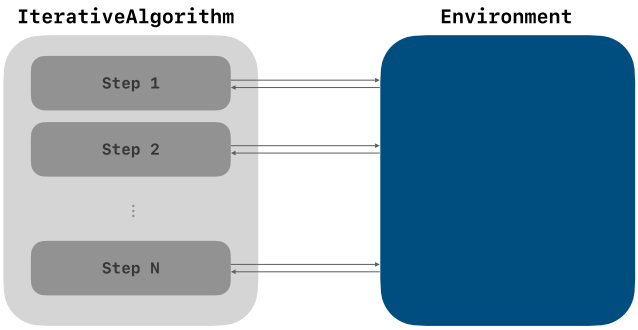

FIG. 2. In GQCP, algorithms are formulated as (iterative) applications of its constituent steps, each with read-write capabilities to a calculation environment that encapsulates the algorithm's state.

```
# Set up the molecular Hamiltonian in the AO basis.
molecule = gqcpy.Molecule.ReadXYZ("CO.xyz", charge=0)
spinor_basis = gqcpy.GSpinorBasis_d(molecule, "6-31G**")
hamiltonian = spinor_basis.quantize(gqcpy.FQMolecularHamiltonian(molecule))
```

```

# Use a solver and environment to optimize the GHF wave function model.
N = molecule.numberOfElectrons()
S = spinor_basis.quantize(gqcpy.OverlapOperator())
solver = gqcpy.GHFSCFSolver_d.Plain()
environment = gqcpy.GHFSCFEnvironment_d.WithCoreGuess(N, hamiltonian, S)
qc_structure = gqcpy.GHF_d.optimize(solver, environment)

# Extract the electronic energy and the optimized GHF model parameters.
electronic_energy = qc_structure.groundStateEnergy()
ghf_parameters = qc_structure.groundStateParameters()
```

Listing 4: A complete code example that calculates the GHF energy and optimized model parameters for CO in a 6-31G** basis set. This code example can be modified to use complex orbitals by replacing the occurrences of `_d` with `_cd`.

C. ROHF through constrained UHF

A typical application of the flexibility of GQCP's algorithmic framework is found in the formulation of restricted open-shell Hartree–Fock (ROHF) as constrained unrestricted Hartree–Fock (CUHF). In this formalism, Tsuchimochi and Scuseria have reproduced ROHF energies through a constrained UHF procedure by replacing the α- and β-Fock matrices according to the ROHF constraint after each UHF SCF cycle.[52]

In GQCP, the UHF SCF algorithm is formulated as an instance of `IterativeAlgorithm`, which entails a recurring application of the UHF SCF cycle and convergence checks until the algorithm is considered converged or has reached the maximum number of allowed iterations. In order to implement the aforementioned ROHF replacements, a custom modification function should be implemented and wrapped into the appropriate `FunctionalStep` class before inserting it at the correct position in the iterative step cycle of the algorithm.

A code example that uses the CUHF algorithm on molecular oxygen in a 6-31G basis set is given in listing 5. The initial guess for the coefficient matrices solves the generalized diagonalization problem for the core Hamiltonian and the final variable `cuhf_energy` contains the converged CUHF electronic energy. The definition of the CUHF modification step and the corresponding definition of the CUHF solver are detailed in listing 9 in the Appendix.

```
# Set up the molecular Hamiltonian in the AO basis.
molecule = gqcpy.Molecule.ReadXYZ("O2.xyz", charge=0)
spin_orbital_basis = gqcpy.USpinOrbitalBasis_d(molecule, "6-31G")
hamiltonian = spin_orbital_basis.quantize(gqcpy.FQMolecularHamiltonian(
    molecule))

# Determine the number of alpha and beta electrons for a triplet oxygen
    calculation.
N = molecule.numberOfElectrons()
N_alpha = N // 2 + 1
N_beta = N // 2 - 1

# Set up the environment and optimize the UHF wave function model. The
    definition of the CUHF solver is omitted and can be found in the
    appendix.
```

```
S = spin_orbital_basis.overlap() # The overlap matrix in the AO basis.
environment = gqcpy.UHFSCFEnvironment_d.WithCoreGuess(N_alpha, N_beta,
    hamiltonian, S)
cuhf_energy = gqcpy.UHF_d.optimize(CUHF_solver, environment).
    groundStateEnergy()
```

Listing 5: Example code that performs a CUHF calculation on triplet molecular oxygen in a 6-31G basis set.

As far as other Hartree–Fock methods are concerned, GQCP provides real- and complex-valued restricted, unrestricted, and generalized Hartree–Fock (R/U/GHF) with internal and external stability analyses.[49,50] Plain, DIIS[53,54] and dampening solvers have been implemented in such a way that they can easily be modified or extended by the user as explained in Sec. III B.

D. CCSD with a GHF reference

Consider the scripting code in listing 6 that calculates the CCSD correlation energy using a GHF reference determinant. Note that, in order to simplify the code listings, we have opted not to include the code that performs the optimization of the HF spinors or spin-orbitals. For RHF, UHF and GHF code examples, we refer the reader to our website.[82] After having transformed the quantities from the atomic to the molecular orbital basis, the CCSD environment is initialized by supplying an `OrbitalSpace` that encapsulates occupied and virtual orbital indices. Finally, the optimization of the CCSD wave function model and the retrieval of the correlation energy are performed in the usual manner through the `optimize` call.

```
# Transform from AO to MO basis.
C = ghf_parameters.expansion()
spinor_basis.transform(C) # Now corresponds to the GHF MOs.
hamiltonian.transform(C)

# Prepare the solver and environment.
M = spinor_basis.numberOfSpinors()
solver = gqcpy.CCSDSolver_d.Plain()
environment = gqcpy.CCSDEnvironment_d.PerturbativeCCSD(hamiltonian,
    ghf_parameters.orbitalSpace()) # Initialize the CCSD amplitudes from MP2
     theory.

# Optimize and retrieve the CCSD correlation energy.
ccsd_qc_structure = gqcpy.CCSD_d.optimize(solver, environment)
ccsd_correlation_energy = ccsd_qc_structure.groundStateEnergy()
```

Listing 6: Example code that calculates the CCSD correlation energy based on a GHF reference determinant.

Only minor modifications are required in order to calculate the CCSD correlation energy with a UHF determinant as a reference (cf. listing 7) or with an RHF determinant as a reference (cf. listing 8). One of the modifications is related to the re-formulation of the spin-orbital bases as generalized spinor bases, while the other required change corresponds to a different kind of ordering of the occupied and virtual orbitals.

In the GHF method formulation, the MOs are sorted with increasing orbital energy, while in UHF (and RHF), they are separated with respect to their spin. Therefore, the correct instance of `OrbitalSpace` is initialized through the GHF occupation number vector (ONV). In the GCCSD example, the GHF ONV would correspond to a set of `1`s (denoting the occupied orbitals) followed by a set of `0`s (denoting the virtual orbitals), while in the RCCSD and UCCSD examples, the GHF ONV corresponds to a set of `1`s and a set of `0`s, once related to the alpha spin-orbitals and once to the beta spin-orbitals. As an example, after a GHF calculation, the GHF ONV would be represented as `110000` for two electrons in six orbitals and as `100|100` after an UHF or RHF calculation, where the alpha and beta orbitals are conceptually separated by the vertical bar `|`.

```
# Transform from AO to MO basis.
C = uhf_parameters.expansion()
spin_orbital_basis.transform(C)

# Re-formulations related to generalized spinor bases.
spinor_basis = gqcpy.GSpinorBasis_d.FromUnrestricted(spin_orbital_basis)
hamiltonian = spinor_basis.quantize(gqcpy.FQMolecularHamiltonian(molecule))

# Prepare the solver and environment.
solver = gqcpy.CCSDSolver_d.Plain()

M = spinor_basis.numberOfSpinors()
N = molecule.numberOfElectrons()
ghf_onv = gqcpy.SpinUnresolvedONV.GHF(M, N, uhf_parameters.
    spinOrbitalEnergiesBlocked()) # First the alpha orbital energies, then
    the beta orbital energies.
environment = gqcpy.CCSDEnvironment_d.PerturbativeCCSD(hamiltonian, ghf_onv
    .orbitalSpace()) # Initialize the CCSD amplitudes from MP2 theory.

# Optimize and retrieve the CCSD correlation energy.
ccsd_qc_structure = gqcpy.CCSD_d.optimize(solver, environment)
ccsd_correlation_energy = ccsd_qc_structure.groundStateEnergy()
```

Listing 7: Example code that calculates the CCSD correlation energy with a UHF reference determinant.

```
# Transform from AO to MO basis.
C = rhf_parameters.expansion()
spin_orbital_basis.transform(C)

# Re-formulations related to generalized spinor bases.
```

```
spinor_basis = gqcpy.GSpinorBasis_d.FromRestricted(spin_orbital_basis)
hamiltonian = spinor_basis.quantize(gqcpy.FQMolecularHamiltonian(molecule))

# Prepare the solver and environment.
solver = gqcpy.CCSDSolver_d.Plain()

M = spinor_basis.numberOfSpinors()
N = molecule.numberOfElectrons()
ghf_onv = gqcpy.SpinUnresolvedONV.GHF(M, N, rhf_parameters.
    spinOrbitalEnergiesBlocked()) # First the alpha orbital energies, then
    the beta orbital energies.
environment = gqcpy.CCSDEnvironment_d.PerturbativeCCSD(hamiltonian, ghf_onv
    .orbitalSpace()) # Initialize the CCSD amplitudes from MP2 theory.

# Optimize and retrieve the CCSD correlation energy.
ccsd_qc_structure = gqcpy.CCSD_d.optimize(solver, environment)
ccsd_correlation_energy = ccsd_qc_structure.groundStateEnergy()
```

Listing 8: Example code that calculates the CCSD correlation energy with an RHF reference determinant.

Since the change of reference determinant requires the transformation of the operators and spin-orbital basis to their generalized equivalents, the GCCSD, UCCSD, and RCCSD methods formulated in this manner have exactly the same computational scaling since no simplifications are used with respect to spin symmetry. However, with GQCP's aim in mind, the increase in flexibility and decrease in development time outweigh the minor decrease in execution speed.

As far as coupled-cluster (CC) methods are concerned, GQCP has implemented CCD (CC doubles) and CCSD (CC singles and doubles) both for RHF, UHF, and GHF reference determinants. Furthermore, we have implemented pair coupled-cluster doubles (pCCD)[55,56] and orbital-optimized pCCD (OO-pCCD) using the dense orbital Hessian.

IV. SOFTWARE DEVELOPMENT IN GQCP

We strictly adhere to scientific software best practices as promoted by the institutions MolSSI,[5,57,58] CECAM,[13,59] and MaX.[60] We spent considerable time refactoring our code, making sure that the APIs are semantically as clear as possible and that technical debt does not accumulate beyond reasonability.

GQCP's focus has been to identify reusable software objects and, as a consequence, we have not attempted any forms of deliberate (i.e., hand-coded) parallelization, or targeted heterogeneous architectures specifically. By designing components that specifically separate the *model* (i.e., the abstract formulation of a solution) from *computation* (i.e., how the specifics are implemented under the hood),[5,17] we focus on the implementation of the high-level characteristics of the problem at hand, while delegating the optimization and hardware acceleration to experts in their respective fields.[4,17,61] Currently, we rely on underlying libraries such as Eigen[62] (with interfaces to BLAS implementations) or Libint2[22] (for molecular integral calculations) for achieving parallelization in particular subcalculations. By relying on lower-level adapter-like modules,[61,63–70] our APIs can, in principle, proceed to scale to the exascale regime. As GQCP's focus is to provide useful generalizations, GQCP could serve as an *initiative* to further improve inter-module communication between the modules in the current electronic structure software ecosystem.[11,13,14,18,20,21,71,72]

Due to GQCP's Python bindings, a prototype can typically be developed in a Jupyter Notebook,[73] which allows for a flexible and high-level discussion between developers. Upon careful consideration related to possible generalizations, the actual functionality of the prototype is implemented in the C++ library, accompanied by corresponding Python bindings and relevant unit tests. Adhering to the Gitflow workflow,[74] this work is typically done in a so-called *feature* branch and is merged with the *develop* branch only after an approving review by peer developers (who typically formulate feedback on the code style, documentation, etc.) and if the continuous integration processes do not report any failures for the full set of unit tests.

On a push to the develop or master branch, the project's GitHub Actions[83] automate both the creation of a new Docker

container image and a new conda installer, made public through Docker Hub[75,76] and Anaconda,[77,78] respectively. Existing users can then proceed to use newly implemented functionalities simply by rebuilding the Docker image on their infrastructure, or re-installing the conda package. Users who require HPC systems for their calculations can install GQCP through the EasyBuild[79] installation framework, or can alternatively convert the Docker container to a Singularity[80] container. As far as provisioning is concerned for developers, we provide a specific Docker container image for development, which covers the proper inclusion and linkage of all our dependencies, so that they can immediately focus on development rather than manually provisioning their systems. We note that we more than welcome feature requests, bug reports, and pull requests from the community through our GitHub repository.[81]

V. SUMMARY

The Ghent Quantum Chemistry Package (GQCP) is an open-source electronic structure library that focuses on providing computational objects that correspond to the theoretical concepts behind two-component spinor bases. The resulting electronic structure building blocks can be used through both a C++ and a Python interface.

GQCP's main area of focus is to allow rapid prototyping of novel computational methods by providing a flexible software framework that allows method developers to focus on the high-level formulation of the problem at hand. Furthermore, GQCP allows structural modifications of quantum chemical solvers at run-time and allows access to all lower-level intermediates and raw data types.

As the design of the objects itself semantically resembles the underlying theoretical concepts, GQCP is a helpful educational tool for exploring electronic structure concepts and can serve as an initiative to improve programmatic communication in-between modules of the current electronic structure software ecosystem.

AUTHORS' CONTRIBUTIONS

L.L. is the software lead and visualization lead; he wrote the original draft and equally contributed in editing the draft. X.D.V. and D.T. equally contributed in the software development, reviewed the manuscript, and supported in editing. T.H. equally contributed in the software development. P.B. is the funding acquisition lead; he reviewed the manuscript and equally contributed in editing. G.A. is the editing lead; he equally contributed in the funding acquisition and software development and reviewed the manuscript.

ACKNOWLEDGMENTS

L.L. acknowledges support from an FWO Ph.D. fellowship (Grant No. 1126619N). Parts of this research were also funded by the FWO (Research Project No. G031820N). The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, FWO, and the Flemish Government—Department EWI.

APPENDIX: THE CUHF REPLACEMENT FUNCTION

```python
def cuhf_function(environment):
    # Read the relevant intermediates from the environment.
    F = environment.fock_matrices[-1]
    D = environment.density_matrices[-1]
    N = environment.N # Number of alpha and beta electrons.

    # Form the closed-shell Fock matrix and the UHF modification in the AO
       basis.
    F_cs = (F.alpha + F.beta) / 2.0
    Delta_UHF = (F.alpha - F.beta) / 2.0

    # Form the modified Fock matrices.
    F_aa = F_cs + Delta_UHF
    F_bb = F_cs - Delta_UHF

```

```
# Find the natural occupation numbers and vectors by diagonalizing the charge-density matrix in an orthonormal basis. For the orthonormal basis, we use the Löwdin basis of the alpha basis functions.
X = spin_orbital_basis.lowdinOrthonormalization().alpha # X transforms from the AO basis to the Löwdin basis.

P_AO = (D.alpha + D.beta) / 2.0
P_MO = P_AO.transformed(X)

natural_occupation_numbers, V = np.linalg.eigh(P_MO.matrix()) # Use NumPy for the diagonalization.

# Construct the Langrange multipliers to add them to the 'constrained' Fock matrices. They should be constructed in the basis of the natural occupations.
V = gqcpy.UTransformationComponent_d(V) # V transforms from the Löwdin basis to the natural occupations.

Delta_UHF_NO = Delta_UHF.transformed(X).transformed(V) # In the natural occupation (NO) basis.
Delta_UHF_NO = Delta_UHF_NO.parameters()

Lambda_NO = np.zeros(np.shape(Delta_UHF_NO))
Lambda_NO[:N.beta, N.alpha:] = -Delta_UHF_NO[:N.beta, N.alpha:]
Lambda_NO[N.alpha:, :N.beta] = -Delta_UHF_NO[N.alpha:, :N.beta]
Lambda_NO = gqcpy.ScalarUSQOneElectronOperatorComponent_d(Lambda_NO)

Lambda_AO = Lambda_NO.transformed(V.inverse()).transformed(X.inverse()) # In the AO basis.

# Overwrite the most recent UHF Fock matrix with the CUHF modification.
F_alpha_constrained = F_aa + Lambda_AO
F_beta_constrained = F_bb - Lambda_AO

F_constrained = gqcpy.ScalarUSQOneElectronOperator_d(F_alpha_constrained, F_beta_constrained)

environment.replace_current_fock_matrix(F_constrained)
```

```
cuhf_step = gqcpy.FunctionalStep_UHFSCFEnvironment_d(cuhf_function,
    description="Replace the alpha- and beta- UHF Fock matrices by their
     constrained counterparts.")

cuhf_solver = gqcpy.UHFSCFSolver_d.Plain(threshold=1.0e-04,
    maximum_number_of_iterations=1000)
cuhf_solver.insert(cuhf_step, 2)
```

Listing 9: A Python function that modifies the UHF Fock matrices according to the ROHF constraint and its subsequent insertion as an algorithmic step into a plain UHF SCF solver.

DATA AVAILABILITY

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

REFERENCES

[1]C. D. Sherrill, D. E. Manolopoulos, T. J. Martínez, and A. Michaelides, "Electronic structure software," J. Chem. Phys. **153**, 070401 (2020).

[2]M. S. Gordon and T. L. Windus, "Editorial: Modern architectures and their impact on electronic structure theory," Chem. Rev. **120**, 9015–9020 (2020).

[3]E. W. Dijkstra, "On the role of scientific thought," in *Selected Writings on Computing: A Personal Perspective* (Springer-Verlag, 1982), pp. 60–66.

[4]P. Giannozzi, O. Baseggio, P. Bonfà, D. Brunato, R. Car, I. Carnimeo, C. Cavazzoni, S. de Gironcoli, P. Delugas, F. F. Ruffino, A. Ferretti, N. Marzari, I. Timrov, A. Urru, and S. Baroni, "Quantum ESPRESSO toward the exascale," J. Chem. Phys. **152**, 154105 (2020).

[5]A. Krylov, T. L. Windus, T. Barnes, E. Marin-Rimoldi, J. A. Nash, B. Pritchard, D. G. A. Smith, D. Altarawy, P. Saxe, C. Clementi, T. D. Crawford, R. J. Harrison, S. Jha, V. S. Pande, and T. Head-Gordon, "Perspective: Computational chemistry software and its advancement as illustrated through three grand challenge cases for molecular science," J. Chem. Phys. **149**, 180901 (2018).

[6]S. Lehtola, C. Steigemann, M. J. T. Oliveira, and M. A. L. Marques, "Recent developments in libxc—A comprehensive library of functionals for density functional theory," SoftwareX **7**, 1–5 (2018).

[7]R. Di Remigio, A. H. Steindal, K. Mozgawa, V. Weijo, H. Cao, and L. Frediani, "PCMSolver: An open-source library for solvation modeling," Int. J. Quantum Chem. **119**, e25685 (2019).

[8]M. Scheurer, P. Reinholdt, E. R. Kjellgren, J. M. H. Olsen, A. Dreuw, and J. Kongsted, "CPPE: An open-source C++ and Python library for polarizable embedding," J. Chem. Theory Comput. **15**, 6154–6163 (2019).

[9]A. H. Romero, D. C. Allan, B. Amadon, G. Antonius, T. Applencourt, L. Baguet, J. Bieder, F. Bottin, J. Bouchet, E. Bousquet, F. Bruneval, G. Brunin, D. Caliste, M. Côté, J. Denier, C. Dreyer, P. Ghosez, M. Giantomassi, Y. Gillet, O. Gingras, D. R. Hamann, G. Hautier, F. Jollet, G. Jomard, A. Martin, H. P. C. Miranda, F. Naccarato, G. Petretto, N. A. Pike, V. Planes, S. Prokhorenko, T. Rangel, F. Ricci, G.-M. Rignanese, M. Royo, M. Stengel, M. Torrent, M. J. van Setten, B. Van Troeye, M. J. Verstraete, J. Wiktor, J. W. Zwanziger, and X. Gonze, "ABINIT: Overview and focus on selected capabilities," J. Chem. Phys. **152**, 124102 (2020).

[10]L. E. Ratcliff, W. Dawson, G. Fisicaro, D. Caliste, S. Mohr, A. Degomme, B. Videau, V. Cristiglio, M. Stella, M. D'Alessandro, S. Goedecker, T. Nakajima, T. Deutsch, and L. Genovese, "Flexibilities of wavelets as a computational basis set for large-scale electronic structure calculations," J. Chem. Phys. **152**, 194110 (2020).

[11]J. M. H. Olsen, S. Reine, O. Vahtras, E. Kjellgren, P. Reinholdt, K. O. H. Dundas, X. Li, J. Cukras, M. Ringholm, E. D. Hedegård, R. D. Remigio, N. H. List, R. Faber, B. N. C. Tenorio, R. Bast, T. B. Pedersen, Z. Rinkevicius, S. P. A. Sauer, K. V. Mikkelsen, J. Kongsted, S. Coriani, K. Ruud, T. Helgaker, H. J. A. Jensen, and P. Norman, "Dalton project: A Python platform for molecular- and electronic-structure simulations of complex systems," J. Chem. Phys. **152**, 214115 (2020).

[12]A. García, N. Papior, A. Akhtar, E. Artacho, V. Blum, E. Bosoni, P. Brandimarte, M. Brandbyge, J. I. Cerdá, F. Corsetti, R. Cuadrado, V. Dikan, J. Ferrer, J. Gale, P. García-Fernández, V. M. García-Suárez, S. García, G. Huhs, S. Illera, R. Korytár, P. Koval, I. Lebedeva, L. Lin, P. López-Tarifa, S. G. Mayo, S. Mohr, P. Ordejón, A. Postnikov, Y. Pouillon, M. Pruneda, R. Robles, D. Sánchez-Portal, J. M. Soler, R. Ullah, V. W.-z. Yu, and J. Junquera, "Siesta: Recent developments and applications," J. Chem. Phys. **152**, 204108 (2020).

[13]M. J. T. Oliveira, N. Papior, Y. Pouillon, V. Blum, E. Artacho, D. Caliste, F. Corsetti, S. de Gironcoli, A. M. Elena, A. García, V. M. García-Suárez, L. Genovese, W. P. Huhn, G. Huhs, S. Kokott, E. Küçükbenli, A. H. Larsen, A. Lazzaro, I. V. Lebedeva, Y. Li, D. López-Durán, P. López-Tarifa, M. Lüders, M. A. L. Marques, J. Minar, S. Mohr, A. A. Mostofi, A. O'Cais, M. C. Payne, T. Ruh, D. G. A. Smith, J. M. Soler, D. A. Strubbe, N. Tancogne-Dejean, D. Tildesley, M. Torrent, and V. W.-z. Yu, "The CECAM electronic structure library and the modular software development paradigm," J. Chem. Phys. **153**, 024117 (2020).

[14]R. M. Richard, C. Bertoni, J. S. Boschen, K. Keipert, B. Pritchard, E. F. Valeev, R. J. Harrison, W. A. de Jong, and T. L. Windus, "Developing a computational chemistry framework for the exascale era," Comput. Sci. Eng. **21**, 48–58 (2019).

[15]M. F. Herbst, M. Scheurer, T. Fransson, D. R. Rehn, and A. Dreuw, "adcc: A versatile toolkit for rapid development of algebraic-diagrammatic construction methods," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **10**, e1462 (2020).

[16]L. Belpassi, M. De Santis, H. M. Quiney, F. Tarantelli, and L. Storchi, "BERTHA: Implementation of a four-component Dirac–Kohn–Sham relativistic framework," J. Chem. Phys. **152**, 164118 (2020).

[17]P. Zulian, A. Kopaničáková, M. G. C. Nestola, A. Fink, N. A. Fadel, J. Vandevondele, and R. Krause, "Large scale simulation of pressure induced phase-field fracture propagation using Utopia," CCF Trans. High Perform. Comput. (published online) (2021).

[18]C. Peng, C. A. Lewis, X. Wang, M. C. Clement, K. Pierce, V. Rishi, F. Pavošević, S. Slattery, J. Zhang, N. Teke, A. Kumar, C. Masteran, A. Asadchev, J. A. Calvin, and E. F. Valeev, "Massively parallel quantum chemistry: A high-performance research platform for electronic structure," J. Chem. Phys. **153**, 044120 (2020).

[19]K. Guther, R. J. Anderson, N. S. Blunt, N. A. Bogdanov, D. Cleland, N. Dattani, W. Dobrautz, K. Ghanem, P. Jeszenszki, N. Liebermann, G. L. Manni, A. Y. Lozovoi, H. Luo, D. Ma, F. Merz, C. Overy, M. Rampp, P. K. Samanta, L. R. Schwarz, J. J. Shepherd, S. D. Smart, E. Vitale, O. Weser, G. H. Booth, and A. Alavi, "NECI: *N*-electron configuration interaction with an emphasis on state-of-the-art stochastic methods," J. Chem. Phys. **153**, 034107 (2020).

[20]D. G. A. Smith, L. A. Burns, A. C. Simmonett, R. M. Parrish, M. C. Schieber, R. Galvelis, P. Kraus, H. Kruse, R. Di Remigio, A. Alenaizan, A. M. James, S. Lehtola, J. P. Misiewicz, M. Scheurer, R. A. Shaw, J. B. Schriber, Y. Xie, Z. L. Glick, D. A. Sirianni, J. S. O'Brien, J. M. Waldrop, A. Kumar, E. G. Hohenstein, B. P. Pritchard, B. R. Brooks, H. F. Schaefer, A. Y. Sokolov, K. Patkowski, A. E. DePrince, U. Bozkaya, R. A. King, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill, "PSI4 1.4: Open-source software for high-throughput quantum chemistry," J. Chem. Phys. **152**, 184108 (2020).

[21]Q. Sun, X. Zhang, S. Banerjee, P. Bao, M. Barbry, N. S. Blunt, N. A. Bogdanov, G. H. Booth, J. Chen, Z.-H. Cui, J. J. Eriksen, Y. Gao, S. Guo, J. Hermann, M. R. Hermes, K. Koh, P. Koval, S. Lehtola, Z. Li, J. Liu, N. Mardirossian, J. D. McClain, M. Motta, B. Mussard, H. Q. Pham, A. Pulkin, W. Purwanto, P. J. Robinson, E. Ronca, E. R. Sayfutyarova, M. Scheurer, H. F. Schurkus, J. E. T. Smith, C. Sun, S.-N. Sun, S. Upadhyay, L. K. Wagner, X. Wang, A. White, J. D. Whitfield, M. J. Williamson, S. Wouters, J. Yang, J. M. Yu, T. Zhu, T. C. Berkelbach, S. Sharma, A. Y. Sokolov, and G. K.-L. Chan, "Recent developments in the PySCF program package," J. Chem. Phys. **153**, 024109 (2020).
[22]E. F. Valeev, Libint: A library for the evaluation of molecular integrals of many-body operators over Gaussian functions, http://libint.valeyev.net/.
[23]Q. Sun, "Libcint: An efficient general integral library for Gaussian basis functions," J. Comput. Chem. **36**, 1664–1671 (2015).
[24]H. Fukutome, "Unrestricted Hartree–Fock theory and its applications to molecules and chemical reactions," Int. J. Quantum Chem. **20**, 955–1065 (1981).
[25]P.-O. Löwdin and I. Mayer, "Some studies of the general Hartree-Fock method," Adv. Quantum Chem. **24**, 79–114 (1992).
[26]D. B. Williams-Young, A. Petrone, S. Sun, T. F. Stetina, P. Lestrange, C. E. Hoyer, D. R. Nascimento, L. Koulias, A. Wildman, J. Kasper, J. J. Goings, F. Ding, A. E. DePrince, E. F. Valeev, and X. Li, "The Chronus Quantum software package," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **10**, e1436 (2020).
[27]T. Helgaker, P. Jørgensen, and J. Olsen, *Molecular Electronic-Structure Theory* (John Wiley & Sons Ltd., Chichester, 2000).
[28]P. Lykos and G. W. Pratt, "Discussion on the Hartree-Fock approximation," Rev. Mod. Phys. **35**, 496–501 (1963).
[29]C. A. Jiménez-Hoyos, T. M. Henderson, and G. E. Scuseria, "Generalized Hartree–Fock description of molecular dissociation," J. Chem. Theory Comput. **7**, 2667–2674 (2011).
[30]I. W. Bulik, G. Scalmani, M. J. Frisch, and G. E. Scuseria, "Noncollinear density functional theory having proper invariance and local torque properties," Phys. Rev. B **87**, 035117 (2013).
[31]C. A. Jiménez-Hoyos, R. Rodríguez-Guzmán, and G. E. Scuseria, "Polyradical character and spin frustration in fullerene molecules: An ab initio non-collinear Hartree–Fock study," J. Phys. Chem. A **118**, 9925–9940 (2014).
[32]J. J. Goings, F. Ding, M. J. Frisch, and X. Li, "Stability of the complex generalized Hartree-Fock equations," J. Chem. Phys. **142**, 154109 (2015).
[33]J. J. Goings, F. Egidi, and X. Li, "Current development of noncollinear electronic structure theory," Int. J. Quantum Chem. **118**, e25398 (2018).
[34]D. Williams-Young, F. Egidi, and X. Li, "Relativistic two-component particle–particle Tamm–Dancoff approximation," J. Chem. Theory Comput. **12**, 5379–5384 (2016).
[35]F. Egidi, S. Sun, J. J. Goings, G. Scalmani, M. J. Frisch, and X. Li, "Two-component noncollinear time-dependent spin density functional theory for excited state calculations," J. Chem. Theory Comput. **13**, 2591–2603 (2017).
[36]M. Nakano, J. Seino, and H. Nakai, "Development of spin-dependent relativistic open-shell Hartree–Fock theory with time-reversal symmetry (I): The unrestricted approach," Int. J. Quantum Chem. **117**, e25356 (2017).
[37]S. Sen and E. I. Tellgren, "Non-perturbative calculation of orbital and spin effects in molecules subject to non-uniform magnetic fields," J. Chem. Phys. **148**, 184112 (2018).
[38]S. Sun, D. B. Williams-Young, T. F. Stetina, and X. Li, "Generalized Hartree–Fock with nonperturbative treatment of strong magnetic fields: Application to molecular spin phase transitions," J. Chem. Theory Comput. **15**, 348–356 (2019).
[39]S. Sen, K. K. Lange, and E. I. Tellgren, "Excited states of molecules in strong uniform and nonuniform magnetic fields," J. Chem. Theory Comput. **15**, 3974–3990 (2019).
[40]G. E. Scuseria, C. A. Jiménez-Hoyos, T. M. Henderson, K. Samanta, and J. K. Ellis, "Projected quasiparticle theory for molecular electronic structure," J. Chem. Phys. **135**, 124108 (2011).
[41]C. A. Jiménez-Hoyos, T. M. Henderson, T. Tsuchimochi, and G. E. Scuseria, "Projected Hartree–Fock theory," J. Chem. Phys. **136**, 164109 (2012).
[42]Y. Cui, I. W. Bulik, C. A. Jiménez-Hoyos, T. M. Henderson, and G. E. Scuseria, "Proper and improper zero energy modes in Hartree-Fock theory and their relevance for symmetry breaking and restoration," J. Chem. Phys. **139**, 154107 (2013).
[43]Y. Qiu, T. M. Henderson, J. Zhao, and G. E. Scuseria, "Projected coupled cluster theory," J. Chem. Phys. **147**, 064111 (2017).
[44]P. J. Lestrange, D. B. Williams-Young, A. Petrone, C. A. Jiménez-Hoyos, and X. Li, "Efficient implementation of variation after projection generalized Hartree–Fock," J. Chem. Theory Comput. **14**, 588–596 (2018).
[45]T. Tsuchimochi, Y. Mori, and S. L. Ten-no, "Spin-projection for quantum computation: A low-depth approach to strong correlation," Phys. Rev. Res. **2**, 043142 (2020).
[46]K. G. Dyall and K. J. Faegri, *Introduction to Relativistic Quantum Chemistry* (Oxford University Press, 2007).
[47]P. R. Surján, *Second Quantized Approach to Quantum Chemistry*, An Elementary Introduction (Springer-Verlag, 1989).
[48]See https://pybind11.readthedocs.io/en/stable/ for pybind11—Seamless interoperability between C++ and Python; accessed 23 June 2021.
[49]R. Seeger and J. A. Pople, "Self-consistent molecular orbital methods. XVIII. Constraints and stability in Hartree–Fock theory," J. Chem. Phys. **66**, 3045–3050 (1977).
[50]J. L. Stuber and J. Paldus, "Symmetry breaking in the independent particle model," in *Fundamental World of Quantum Chemistry* (Springer-Verlag, 2003), pp. 67–139.
[51]D. G. A. Smith, L. A. Burns, D. A. Sirianni, D. R. Nascimento, A. Kumar, A. M. James, J. B. Schriber, T. Zhang, B. Zhang, A. S. Abbott, E. J. Berquist, M. H. Lechner, L. A. Cunha, A. G. Heide, J. M. Waldrop, T. Y. Takeshita, A. Alenaizan, D. Neuhauser, R. A. King, A. C. Simmonett, J. M. Turney, H. F. Schaefer, F. A. Evangelista, A. E. DePrince, T. D. Crawford, K. Patkowski, and C. D. Sherrill, "Psi4NumPy: An interactive quantum chemistry programming environment for reference implementations and rapid development," J. Chem. Theory Comput. **14**, 3504–3511 (2018).
[52]T. Tsuchimochi and G. E. Scuseria, "Communication: ROHF theory made simple," J. Chem. Phys. **133**, 141102 (2010).
[53]P. Pulay, "Convergence acceleration of iterative sequences. The case of SCF iteration," Chem. Phys. Lett. **73**, 393–398 (1980).
[54]P. Pulay, "Improved SCF convergence acceleration," J. Comput. Chem. **3**, 556–560 (1982).
[55]T. M. Henderson, I. W. Bulik, T. Stein, and G. E. Scuseria, "Seniority-based coupled cluster theory," J. Chem. Phys. **141**, 244104 (2014).
[56]P. A. Limacher, P. W. Ayers, P. A. Johnson, S. De Baerdemacker, D. Van Neck, and P. Bultinck, "A new mean-field method suitable for strongly correlated electrons: Computationally facile antisymmetric products of nonorthogonal geminals," J. Chem. Theory Comput. **9**, 1394–1401 (2013).
[57]T. D. Crawford, On the development of sustainable software for computational chemistry, https://doi.org/10.6084/m9.figshare.790757.v1, 2013.
[58]See https://molssi.org for MolSSI—The molecular sciences software institute; accessed 23 June 2021.
[59]See https://www.cecam.org for CECAM (Centre Europeen de Calcul Atomique et Moleculaire - European Center of Atomic and Molecular Calculations); accessed 23 June 2021.
[60]See http://www.max-centre.eu for MAX—Materials Design at the Exascale; accessed 23 June 2021.
[61]A. Perera, R. J. Bartlett, B. A. Sanders, V. F. Lotrich, and J. N. Byrd, "Advanced concepts in electronic structure (ACES) software programs," J. Chem. Phys. **152**, 184105 (2020).
[62]G. Guennebaud and B. Jacob, Eigen v3, http://eigen.tuxfamily.org; accessed 23 June 2021.
[63]S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object-oriented numerical software libraries," in *Modern Software Tools for Scientific Computing* (Birkhäuser, 1997), pp. 163–202.
[64]M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the Trilinos project," ACM Trans. Math. Software **31**, 397–423 (2005).

[65]J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Z. Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. van der Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick, "The international exascale software project roadmap," Int. J. High Perform. Comput. Appl. **25**, 3–60 (2011).

[66]J. A. Calvin and E. F. Valeev, TiledArray: A general-purpose scalable block-sparse tensor framework, https://github.com/valeevgroup/tiledarray; accessed 23 June 2021.

[67]E. Solomonik, D. Matthews, J. R. Hammond, J. F. Stanton, and J. Demmel, "A massively parallel tensor contraction framework for coupled-cluster computations," J. Parallel Distrib. Comput. **74**, 3176–3190 (2014).

[68]J. A. Calvin, C. A. Lewis, and E. F. Valeev, "Scalable task-based algorithm for multiplication of block-rank-sparse matrices," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms, IA3'15* (Association for Computing Machinery, New York, NY, 2015).

[69]J. A. Calvin and E. F. Valeev, "Task-based algorithm for matrix multiplication: A step towards block-sparse tensor computing," arXiv:1504.05046 (2015).

[70]N. Jindal, V. Lotrich, E. Deumens, and B. A. Sanders, "Exploiting GPUs with the super instruction architecture," Int. J. Parallel Program. **44**, 309–324 (2016).

[71]D. E. Bernholdt, B. A. Allan, R. Armstrong, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. Mclnnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, and S. Zhou, "A component architecture for high-performance scientific computing," Int. J. High Perform. Comput. Appl. **20**, 163–202 (2006).

[72]E. Aprà, E. J. Bylaska, W. A. de Jong, N. Govind, K. Kowalski, T. P. Straatsma, M. Valiev, H. J. J. van Dam, Y. Alexeev, J. Anchell, V. Anisimov, F. W. Aquino, R. Atta-Fynn, J. Autschbach, N. P. Bauman, J. C. Becca, D. E. Bernholdt, K. Bhaskaran-Nair, S. Bogatko, P. Borowski, J. Boschen, J. Brabec, A. Bruner, E. Cauët, Y. Chen, G. N. Chuev, C. J. Cramer, J. Daily, M. J. O. Deegan, T. H. Dunning, M. Dupuis, K. G. Dyall, G. I. Fann, S. A. Fischer, A. Fonari, H. Früchtl, L. Gagliardi, J. Garza, N. Gawande, S. Ghosh, K. Glaesemann, A. W. Götz, J. Hammond, V. Helms, E. D. Hermes, K. Hirao, S. Hirata, M. Jacquelin, L. Jensen, B. G. Johnson, H. Jónsson, R. A. Kendall, M. Klemm, R. Kobayashi, V. Konkov, S. Krishnamoorthy, M. Krishnan, Z. Lin, R. D. Lins, R. J. Littlefield, A. J. Logsdail, K. Lopata, W. Ma, A. V. Marenich, J. M. del Campo, D. Mejia-Rodriguez, J. E. Moore, J. M. Mullin, T. Nakajima, D. R. Nascimento, J. A. Nichols, P. J. Nichols, J. Nieplocha, A. Otero-de-la-Roza, B. Palmer, A. Panyala, T. Pirojsirikul, B. Peng, R. Peverati, J. Pittner, L. Pollack, R. M. Richard, P. Sadayappan, G. C. Schatz, W. A. Shelton, D. W. Silverstein, D. M. A. Smith, T. A. Soares, D. Song, M. Swart, H. L. Taylor, G. S. Thomas, V. Tipparaju, D. G. Truhlar, K. Tsemekhman, T. Van Voorhis, Á. Vázquez-Mayagoitia, P. Verma, O. Villa, A. Vishnu, K. D. Vogiatzis, D. Wang, J. H. Weare, M. J. Williamson, T. L. Windus, K. Woliński, A. T. Wong, Q. Wu, C. Yang, Q. Yu, M. Zacharias, Z. Zhang, Y. Zhao, and R. J. Harrison, "NWChem: Past, present, and future," J. Chem. Phys. **152**, 184102 (2020).

[73]See https://jupyter.org for Project Jupyter; accessed 23 June 2021.

[74]See https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow for Gitflow Workflow; accessed 23 June 2021.

[75]See https://hub.docker.com for Docker Hub; accessed 23 June 2021.

[76]Docker Hub—The Ghent Quantum Chemistry Package, https://hub.docker.com/r/gqcg/gqcp, 2021.

[77]See https://anaconda.org for Anaconda.org; accessed 23 June 2021.

[78]See https://anaconda.org/gqcg/gqcp for Conda—The Ghent Quantum Chemistry Package; accessed 23 June 2021.

[79]See https://easybuild.io for EasyBuild; accessed 23 June 2021.

[80]See https://sylabs.io/singularity/ for Singularity; accessed 23 June 2021.

[81]See https://github.com/GQCG/GQCP for the GQCP GitHub repository; accessed 23 June 2021.

[82]See https://gqcg.github.io/GQCP for documentation on GQCP: The Ghent Quantum Chemistry Package; accessed 23 June 2021.

[83]See https://github.com/features/actions for GitHub Actions: Automate your workflow from idea to production; accessed 23 June 2021.