Enabling Multireference Calculations on Multi-Metallic Systems with Graphic Processing Units

Valay Agarawal,[†] Rishu Khurana,[†,‡] Cong Liu,[‡] Matthew R. Hermes,[*,†] Christopher Knight,[*,¶] and Laura Gagliardi[*,†,§]

†*Department of Chemistry, University of Chicago*

‡*Chemical Sciences and Engineering Division, Argonne National Laboratory*

¶*Computational Science Division, Argonne National Laboratory*

§*Pritzker School of Molecular Engineering, University of Chicago*

E-mail: mrhermes@uchicago.edu; knightc@anl.gov; lgagliardi@uchicago.edu

Abstract

Modeling multimetallic systems efficiently enables faster prediction of desirable chemical properties and design of new materials. This work describes an initial implementation for performing multireference wave function method localized active space self-consistent field (LASSCF) calculations through the use of multiple graphics processing units (GPUs) to accelerate time-to-solution. Density fitting is leveraged to reduce memory requirements, and we demonstrate the ability to fully utilize multi-GPU compute nodes. Performance improvements of 5-10x in total application runtime were observed in LASSCF calculations for multimetallic catalyst systems up to 1200 AOs and an active space of (22e,40o) using up to four NVIDIA A100 GPUs. Written with performance portability in mind, comparable performance is also observed in early runs on the Aurora exascale system using Intel Max Series GPUs.

1 Introduction

Computational modeling and prediction of molecules and materials with multiple transition metals require methods that can accurately and efficiently model electron correlation. Single reference methods such as Hartree Fock (HF),[1] density functional theory (DFT),[2] Møller-Plesset perturbation theory (MP2)[3] and coupled cluster (CC)[4–7] are often inadequate to model the multiconfigurational electronic structure of transition-metal systems in their ground or excited states. Multireference (MR) methods excel at capturing electron correlation in these systems, however, for extended systems containing transition metals, they usually face two challenges: a large number of correlated electrons and a large number of inactive orbitals, typically from ligands. The traditional MR method, complete active space self-consistent field (CASSCF),[8] is not feasible because the cost scales exponentially with the size of the active space (AS), which in turn grows rapidly with multiple metal centers present. Controlling the size of the active space through automatic reduction in configuration space functions (or Slater determinants) using methods such as selected configuration interaction (SCI)[9–14] or density matrix renormalization group (DMRG)[15] has been explored, aiming to lower computational cost and enhance the practical viability of MR calculations. Another class of methods involves the reduction of the Hilbert space through fragmentation techniques guided by chemical intuition, such as the restricted active space SCF (RASSCF),[16,17] generalized active space SCF (GASSCF),[18–21] cluster mean field (cMF)[22] or localized active space self-consistent field (LASSCF).[23,24] These methods exhibit reduced scaling with respect to the size of the total active space compared to CASSCF. In this work, we focus on LASSCF that scales exponentially with the size of the active space of each fragment but polynomially with the size of the total AS.

The presence of inactive orbitals increases the cost of CASSCF polynomially, and several techniques are employed to control this cost. A common technique is to simply reduce the number of inactive orbitals through truncation of atoms. Some examples are replacing tertiary amines ligands with ammonia,[25] or benzoates with formates.[26–28] This approach

proves to be effective for qualitative studies of target properties. However, when conducting quantitative investigations of properties,[29] or aiming to fine-tune them with ligands,[30] it becomes essential to model all inactive orbitals. The second technique uses embedding methods such as DMET[31] and QDET[32] but these are not feasible when dealing with large AS typical in multimetallic systems. LASSCF offers a promising solution to deal with systems with large AS however, the cost of modeling a system with large AS and a large number of inactive orbitals with LASSCF is still limiting and improving the time-to-solution is vital to using LASSCF for such systems.

Any meaningful improvement in computational speed is achieved by first accelerating the calculation bottleneck. Focusing on fundamental operations instead of the entire method as a whole also has the potential to provide algorithm-independent acceleration if multiple algorithms utilize the same functionality. One way to improve computational speed is through the use of specialized hardware for core computational tasks, such as matrix multiplication. Graphics Processing Units (GPUs) are designed to perform a large number of (simple) calculations simultaneously, making them ideal candidates for some scientific computing applications. Modern supercomputers derive most of their floating point operations from GPUs as observed in the recent TOP500[33] list, where the top 5 systems deployed AMD, Intel, and NVIDIA GPUs. The heterogeneity of GPU vendors and software ecosystems today requires application developers to be mindful of performance portability for users to effectively use modern computing resources. Although most of the GPU-accelerated quantum chemistry codes have relied on NVIDIA GPUs,[34–39] progress has been made to adapt these implementations to other GPU vendors.[40–42]

Efforts on GPU-accelerated quantum chemistry codes for large systems have been mainly focused on integral generation[35,42] and single-reference methods, primarily HF,[36] DFT[43] and its relativistic counterpart[44], MP2[36,45,46] and CC[47–50]. GPU accelerated HF, DFT and MP2 have been applied on large systems. Specifically, HF calculations have been performed on organic molecules, water and glycine clusters,[51–54] DFT calculations have been performed

on organic molecules,[43] model systems,[55] solids[56], and gold clusters for relativistic DFT[44], MP2 calculations have been performed on glycine[46] and water clusters.[57] Among multiconfigurational methods, CASSCF,[58–60] DMRG,[61,62] non-orthogonal CI[63] and auxiliary field quantum Monte Carlo[64] have been implemented to utilize GPUs. Within the traditional CASSCF method AS up to (26e, 23o)[65] can be achieved, but (18e,18o) is considered the limit for routine calculations.[66] GPU-accelerated variational 2-body reduced density matrix (v2RDM) CASSCF[67] has been used with (50e,50o) AS for polyacenes. GPU-accelerated DMRG can treat AS with (113e,76o) for FeMo cofactor[61] and (114e,73o) for P-cluster,[62] but only to solve the CI problem on GPUs. We also highlight the adoption of GPU accelerated DMRG as a CI solver for CASSCF.[68]

The memory bottleneck of electronic structure methods such as HF, DFT, MP2, and CASSCF is the processing of 4-center 2-electron (4c-2e) electron repulsion integrals (ERIs) which scales as $\mathcal{O}(N_{\text{ao}}^4)$, where N_{ao} is the number of atomic orbitals. While utilizing direct-SCF principles,[69] where integrals are calculated on the fly, are common, within the realm of methods using precomputed integrals, this scaling makes the study of large systems quickly intractable with an increasing number of orbitals, often in the case of many inactive orbitals. Weakly correlated systems like water clusters or organic compounds have sparse ERIs and methods exploiting the sparsity of ERIs on GPUs[70,71] can achieve significant speedups while keeping memory requirements low. However, sparsity may not exist when solving for organometallic compounds of heavier atoms with diffused orbitals Another way to reduce memory requirements is to approximate the 4c-2e integrals using Cholesky density fitting techniques[72] to decompose a 4c-2e ERI into a 3c-2e Cholesky vector. This reduces the memory requirement to $\mathcal{O}(N_{\text{aux}}N_{\text{ao}}^2)$ where N_{aux} is the number of auxiliary orbitals and is usually 5-10x of N_{ao}. Other methods to approximate ERI like chain of spheres[73] and semi-numerical-K[74] have also been developed. Although various single-reference methods with density fitting techniques have been explored in GPU accelerated computational chemistry codes, the use of density fitting with multi-reference methods on GPUs is less common,

except the work by Mullinax et al.[67]

In this paper, we present an initial implementation of LASSCF with density fitting technique capable of leveraging multiple GPUs on a node. We show a 7-10x reduction in time-to-solution for up to 1200 atomic orbitals (AOs) and the AS of (22e,40o) for a realistic system and up to (64e,64o) for a model system. By focusing on core mathematical operations and a seamless integration, we are able to accelerate HF and CASSCF calculations as well without additional effort. Finally, we show the benefits of developing performance portable software by running and competitive performance on both Intel and NVIDIA GPUs, and without any tuning for Intel GPUs.

2 Theory

2.1 Notation

Orbitals are indicated as s_{K_n} where s represents the type of orbital, $K, L, \ldots$ the fragment index and $n = 1, 2, \ldots$ distinguishes multiple indices of the same type and fragment in a given expression. When the orbitals span the entire molecule (as opposed to a particular fragment), the orbitals are represented by s_n. The symbols μ, p, e, a, v and d represent atomic, molecular, embedding, active, virtual (or unoccupied) and doubly occupied orbitals respectively. As an example, a_{K_n} is the n^{th} active orbital of K^{th} fragment. The active-space wave function for the entire molecule is denoted by Ψ_A and the active-space wave function of a fragment is denoted by Ψ_{A_K}. The auxiliary basis orbitals are represented by P. E represents an energy, $\{E^{(1)}\}_{\text{CI}}$ represents the energy derivative with respect to the CI vector and $\{E^{(1)}\}_x^y$ represents the derivative of the energy with respect to orbital rotation between orbitals x and y. There are N_{ao} atomic orbitals (AOs), N_{mo} molecular orbitals (MOs), N_{aux} auxiliary orbitals, N_{ac} total active orbitals, N_{ac_f} active orbitals in each fragment and N_{emb_f} embedding orbital in each fragment. Repeated internal indices are implicitly summed. Table 1 contains all the notations used throughout the manuscript.

Table 1: All notation used in this work.

Symbol	Meaning	Counts
$K, L, ...$	Fragment Index	n_K
K_n	n^{th} Orbital index on K fragment	
s_n	Full system orbital of type $s \in \{\mu, p, v, d\}$	
s_{K_n}	Fragment orbital of type $s \in \{p, e, v, d\}$	
μ	Atomic orbital	N_{ao}
p	Molecular orbital	N_{mo}
a	Active space orbital	N_{ac}
e	Embedding space orbital	N_{emb_f}
a_K	Fragment active space orbital	N_{ac_f}
v	Virtual orbital	N_{vir}
d	Doubly occupied orbital	N_{occ}
P	Auxiliary orbital	N_{aux}
$\{E^{(1)}\}_x^y$	$\frac{\partial E_x}{\partial y}$ for orbitals x, y	
$\{E^{(1)}\}_{\text{CI}}$	$\frac{\partial E}{\partial CI}$	
Ψ_{A_K}	Fragment active-space wave function	

2.2 LASSCF algorithm

The LASSCF wave function is expressed as a direct product of fragment active space wave functions and a single determinant of the doubly occupied orbitals $|\phi_D\rangle$:

$$|\text{LAS}\rangle = \wedge_K^{\text{fragments}} |\Psi_{A_K}\rangle \wedge |\phi_D\rangle \,, \tag{1}$$

The energy of the LAS wave function is given as

$$E_{\text{LAS}} = \langle \text{LAS}| \, \hat{H} \, |\text{LAS}\rangle \tag{2}$$

LASSCF optimizes the fragment wave function Ψ_{A_K} by minimizing the fragment energy

$$E_K = \langle \Phi_{E_K}| \wedge \langle \Psi_{A_K}| \, \hat{H}_{E_K} \, |\Psi_{A_K}\rangle \wedge |\Phi_{E_K}\rangle \tag{3}$$

where $|\Phi_{E_K}\rangle$ is the determinant containing embedding space inactive orbitals. At conver-

gence, $E_K = E_{\mathrm{LAS}}$. The embedding space Hamiltonian $\hat{H}_{E_K}$ is

$$\begin{aligned}\hat{H}_{E_K} = (h_{e_{K_1}}^{e_{K_2}} + \{v^{jk}\}_{e_{K_1}}^{e_{K_2}} - \{v_\sigma^{\mathrm{self}}\}_{e_{K_1}}^{e_{K_2}})\hat{c}_{e_{K_1}}^\dagger \hat{c}_{e_{K_2}} \\ + \frac{1}{2} g_{e_{K_2} e_{K_4}}^{e_{K_1} e_{K_3}} \hat{c}_{e_{K_{1\sigma}}}^\dagger \hat{c}_{e_{K_{3\tau}}}^\dagger \hat{c}_{e_{K_{4\tau}}} \hat{c}_{e_{K_{2\sigma}}}\end{aligned} \tag{4}$$

omitting an irrelevant constant (Ref.[24] Eq. 12-14), where

$$\{v^{jk}\}_{\mu_1}^{\mu_2} = g_{\mu_2\mu_4}^{\mu_1\mu_3} D_{\mu_4}^{\mu_3} - g_{\mu_4\mu_2}^{\mu_1\mu_3} \{\gamma_\sigma\}_{\mu_4}^{\mu_3} \tag{5}$$

$$\{v^{(\mathrm{self})}\}_{e_{K_1}}^{e_{K_2}} = g_{e_{K_2} e_{K_4}}^{e_{K_1} e_{K_3}} D_{e_{K_4}}^{e_{K_3}} - g_{e_{K_4} e_{K_2}}^{e_{K_1} e_{K_3}} \{\gamma_\sigma\}_{e_{K_4}}^{e_{K_3}} \tag{6}$$

are the effective embedding potential and effective self potential respectively. $\hat{c}_{e_{K_n}}^\dagger$ and $\hat{c}_{e_{K_n}}$ are the creation and the annihilation operators with σ and τ spin; h and g are 1- and 2- electron molecular Hamiltonian matrix elements, respectively; and D and γ_σ are the spin-summed and spin-separated 1-body reduced density matrices, respectively. Eq. 3 can be minimized with a standard CASSCF solver with N_{emb_f} "atomic" orbitals (embedding orbitals form the basis), N_{ac_f} active orbitals and the $\hat{H}_{E_K}$ Hamiltonian. This corresponds to solving the equation

$$\{E^{(1)}\}_{\mathrm{CI}} = \{E^{(1)}\}_{a_K}^{e_K} = 0 \tag{7}$$

The total number of fragment embedding orbitals (N_{emb_f}) can be up to $2(N_{\mathrm{ac}_f} + N_{F_K})$ where N_{ac_f} is the number of user-selected fragment active orbitals and N_{F_K} is the number of user-selected fragment orbitals.

These CASSCF subproblems or tasks require repeated calculations of effective potentials and ERIs with specific indices pattern of $g_{p_2 a_2}^{p_1 a_1}$ and $g_{a_1 a_2}^{p_1 p_2}$ where a and p respectively refer to active and embedded space orbitals for orbital optimization.[75,76] For a realistic system, N_{emb_f} and N_{ac_f} do not scale with the system size since fragments necessary for modeling multireference nature would remain similar. Cases where larger fragments may need to be

considered, such as extended aromatic systems will have N_{emb_f} and N_{ac_f} scale with the system size.

Once all embedded CASSCF tasks are completed, the active orbitals of all fragments are not necessarily orthogonal since they are optimized independently and must be reorthogonalized. The active orbitals and CI vectors are then frozen and the whole molecule's inactive orbitals are optimized. Additionally, the energy is minimized with respect to rotation between two active subspaces (K, L) and their CI vectors. This corresponds to solving the equation

$$\{E^{(1)}\}_{\mathrm{CI}} = \{E^{(1)}\}_{a_L}^{a_K} = \{E^{(1)}\}_{d}^{v} = 0 \tag{8}$$

Eq. 8 is satisfied through a second order algorithm that optimizes

$$\mathbf{E}^{(1)} + \mathbf{E}^{(2)}\mathbf{x} = \vec{0}, \tag{9}$$

where $\mathbf{x}$ is a unitary generator containing all orbital and CI transformation variables, $\mathbf{E}^{(2)}$ is the second-order derivative. We omit details of exact forms of derivatives in Eq. 8 and refer the reader to ref. 23. Eq. 8 requires a HF-like effective potential repeatedly and ERIs of index pattern $g_{a_{L_1} a_{N_1}}^{a_{K_1} a_{M_1}}$.

The overall LASSCF calculation is converged when

$$\{E^{(1)}\}_{p_1}^{p_2} = 0 \tag{10}$$

$$\{E^{(1)}\}_{\mathrm{CI}} = 0 \tag{11}$$

is satisfied. Evaluating the gradient on the right-hand side of Eq. (10) requires the evaluation of ERIs of index pattern $g_{a_{K_1} a_{M_1}}^{p_1 a_{L_1}}$ once per cycle.

Diagrammatically, the algorithm can be represented as in Figure 1. In the algorithm, the orange boxes represent the CASSCF tasks; the green boxes represent processing before and

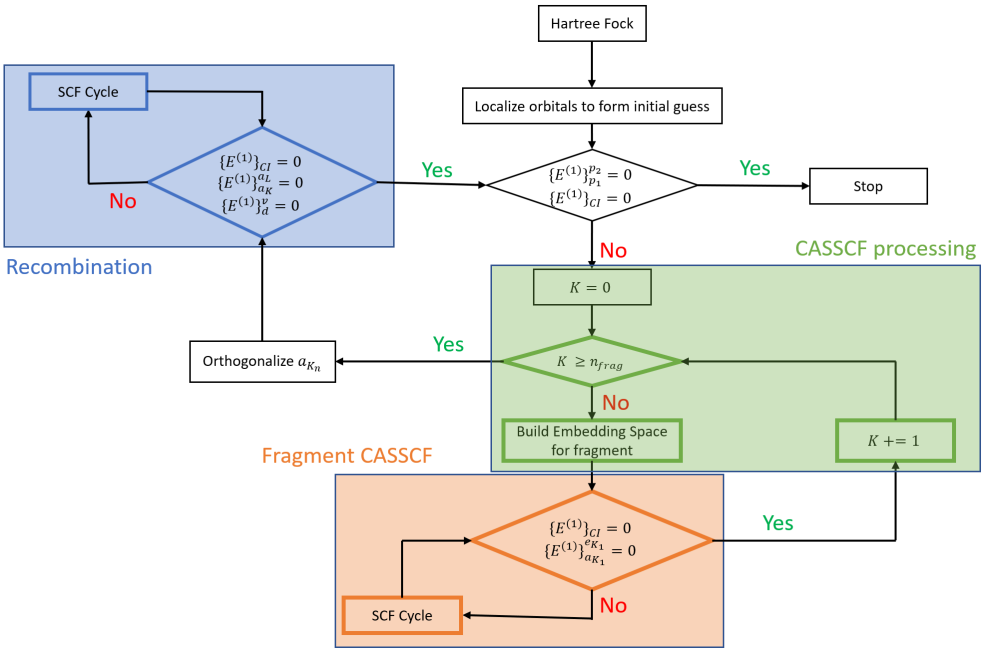


Figure 1: Flowchart of LASSCF algorithm. Highlighted boxes represented various phases of calculations. The green box represents constructing an embedding space for each fragment, orange box for performing a CASSCF task and blue box for active orbitals of a fragment are relaxed with respect to other fragments.

after a fragment CASSCF, e.g. embedding potential in Eq. 5; and the blue boxes represent the recombination algorithm.

2.3 Identification of computational bottlenecks

We used LASSCF within a density-fitting framework that splits a 4c-2e integral into 3c-2e integral[72] given as

$$\begin{aligned} g^{\mu_1\mu_3}_{\mu_2\mu_4} &\approx (\mu_1\mu_2|P)(P|Q)^{-1}(Q|\mu_3\mu_4) \\ &= ((\mu_1\mu_2|P)T)(T(Q|\mu_3\mu_4)) = b^P_{\mu_1\mu_2} b^P_{\mu_3\mu_4} \end{aligned} \tag{12}$$

where $b^P_{\mu_1\mu_2}$ are the three-center two-electron Cholesky vectors of size $\mathcal{O}(N_{\text{aux}}N^2_{\text{ao}})$, and T is $(P|Q)^{-\frac{1}{2}}$. We profiled CPU-only LASSCF calculations using simple Python timer functions within PySCF to collect the total time spent during the various stages of the algorithm.

This was done for all systems described in section 4, and results for the iron-sulfur cluster are shown as a representative example in figure 2.

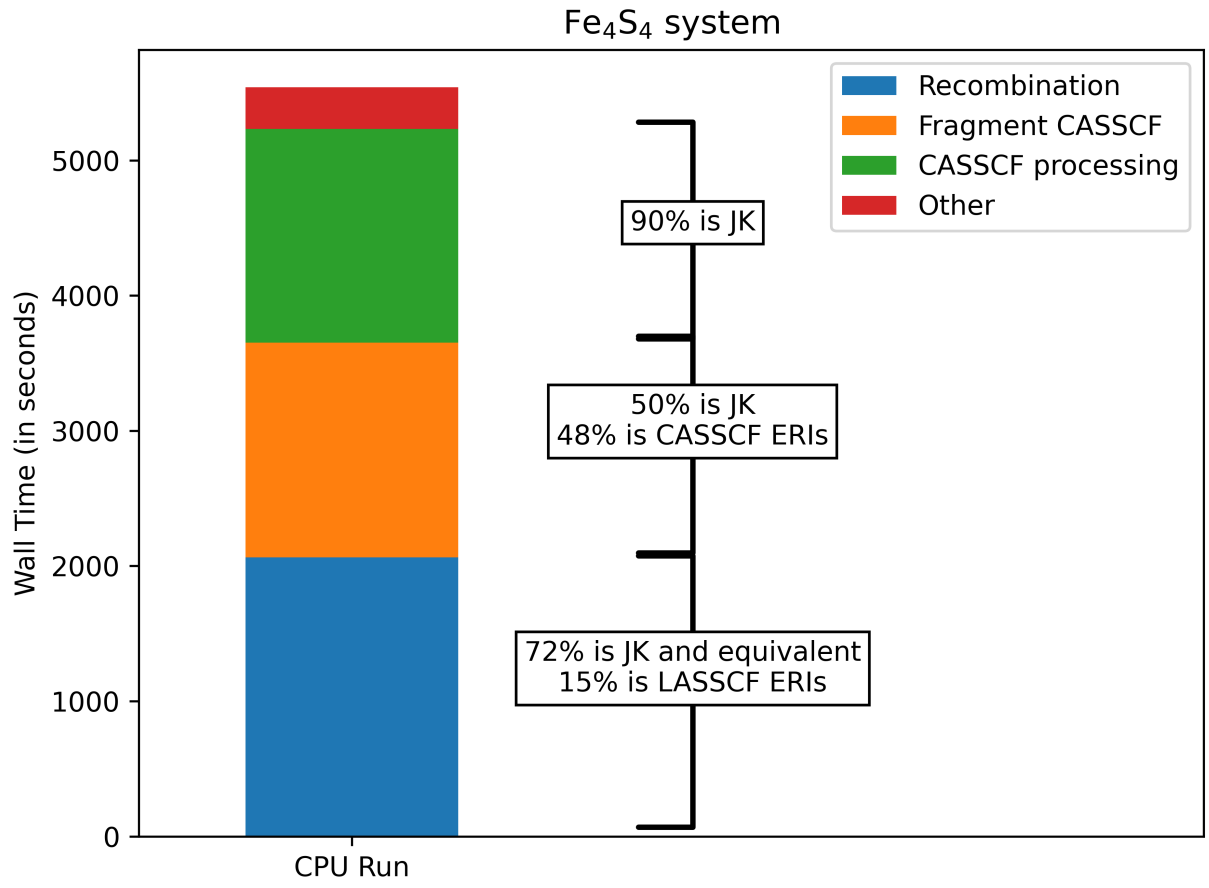


Figure 2: CPU run profile for iron-sulfur cluster. Computer details: Polaris compute node: One Milan CPU with 32 cores.

All embedded CASSCF calculations (in the orange boxes of Fig. 1) combined accounted for about 33% of the total wall time. Within fragment CASSCFs, the evaluation of effective potentials consumed 50% of the CASSCF wall time, while the computation of ERIs ($g^{p_1 a_1}_{p_2 a_2}$ and $g^{p_1 p_2}_{a_1 a_2}$) accounted for another 48%. The pre- and post- processing of CASSCF fragments (shown in the green boxes of Fig. 1) contributed 28% of the total wall time, with over 90% of this time spent on evaluating effective potentials. Finally, the recombination of all fragments (blue boxes of Fig. 1) consumed 37% of the total wall time with 72% dedicated to effective potential evaluations and 15% to LASSCF ERIs ($g^{p_1 a_{M_1}}_{a_{L_1} a_{N_1}}$). A similar breakdown was

observed for the other systems. Analyzing where time was spent in the code was repeated throughout the software development process to ensure priority was given to the next largest time-consuming step in the calculation.

2.4 Analysis of computational bottlenecks

2.4.1 Calculation of effective potentials

The calculation of effective potentials is performed as follows:

$$\{v^{(jk)}\}_{ij} = J_{ij} - \{K_\sigma\}_{ij} \tag{13}$$

where J_{ij} are calculated as

$$v_P = b^P_{ij} D_{ij} \tag{14}$$

$$J_{ij} = v_P b^P_{ij} \tag{15}$$

and

$$v^P_{ik} = b^P_{ij} \{\gamma_\sigma\}^j_k \tag{16}$$

$$\{K_\sigma\}_{ij} = v^P_{ik} b^P_{jk} \tag{17}$$

Here i, j represent both atomic orbitals and embedded orbitals since effective potentials are calculated with both. The computation of the Coulomb matrix J and exchange matrix K is collectively referred to as a "JK call". The calculation of J has a complexity of $\mathcal{O}(N_{\text{aux}} N^2_{\text{ao}})$ if performed for the full system (or $\mathcal{O}(N_{\text{aux}} N^2_{\text{emb}_f})$ for a single fragment) for both steps, namely Eq. 14 and Eq. 15 with output of size $\mathcal{O}(N^2_{\text{ao}})$ (or $\mathcal{O}(N^2_{\text{emb}_f})$). The exchange matrix ($K$), on the other hand, has a complexity of $\mathcal{O}(N_{\text{aux}} N^3_{\text{ao}})$ (or $\mathcal{O}(N_{\text{aux}} N^3_{\text{emb}_f})$) for both steps (Eq. 16 and Eq. 17), an order of magnitude more expensive than J, but with the same size of output

($\mathcal{O}(N_{\text{ao}}^2)$ (or $\mathcal{O}(N_{\text{emb}_f}^2)$)) as J. Performing singular value decomposition (SVD) on the density matrix could reduce the complexity of and memory requirements for the calculation of the K matrix. We show later that JK is no longer the largest computational bottleneck after being GPU-accelerated. That said these algorithmic improvements will be implemented in the future to further reduce the time-to-solution. For each effective potential calculation, the input is a density matrix and the AO (or embedded basis) Cholesky vectors and the output is J and K. For each call, we ensure that the relevant Cholesky vectors and density matrix are present on the GPUs, we calculate J and K, and transfer them back to the CPU.

Since the calculation of full K at once requires at least $2N_{\text{aux}}N_{\text{ao}}^2$ memory (one to store original AO basis ERI and one for the intermediate), it becomes taxing on the available computer memory. Most modern software, including PySCF, perform this calculation in blocks of size N_{blk} along P. We currently have kept the PySCF default $N_{\text{blk}} = 240$ for this work. Initial investigations showed smaller blocks help in load balancing, and examples of such explorations are presented in SI (Fig. SS7 and SS8). Tuning based on the number of GPUs or dynamically changing N_{blk} could help to ensure work is evenly distributed across GPUs. Dynamically changing the blocksize would require some data reorganization across GPUs (leveraging GPU-GPU links), but ensuring consistency with the rest of the PySCF code will be important.

The Cholesky vectors are often stored in triangular form of size $N_{\text{aux}}N_{\text{ao}}(N_{\text{ao}} + 1)/2$. After transferring a block of this lower-triangular matrix to the GPU, it must be unpacked to the square-matrix form before proceeding. Eq. 16 generates an intermediate (v_{ik}^P) of the same size as this block. Furthermore, in order to carry out Eq. (17) efficiently, the Cholesky vector factor must be transposed. Therefore, the total peak memory usage for the build of $\{K_\sigma\}_{ij}$ is $N_{\text{aux}}N_{\text{ao}}^2/2 + 3N_{\text{blk}}N_{\text{ao}}^2$. We will discuss data transfer optimizations later.

2.4.2 Calculation of ERIs in fragment CASSCF

The CASSCF orbital optimization requires explicit ERI ($g^{p_{K_1} p_{K_2}}_{a_{K_1} a_{K_2}}$ and $g^{p_{K_1} a_{K_1}}_{p_{K_2} a_{K_2}}$)[75,76] given by

$$b^P_{e_{K_2} p_{K_1}} = b^P_{e_{K_1} e_{K_2}} M^{p_{K_1}}_{e_{K_1}} \tag{18}$$

$$b^P_{p_{K_1} p_{K_2}} = b^P_{e_{K_2} p_{K_1}} M^{p_{K_2}}_{e_{K_2}} \tag{19}$$

$$g^{p_{K_1} a_{K_1}}_{p_{K_2} a_{K_2}} = b^P_{p_{K_1} p_{K_2}} b^P_{a_{K_1} a_{K_2}} \tag{20}$$

$$g^{p_{K_1} p_{K_2}}_{a_{K_1} a_{K_2}} = b^P_{p_{K_1} a_{K_1}} b^P_{p_{K_2} a_{K_2}} \tag{21}$$

Here, M is the matrix of MO coefficients that transforms from AO basis to MO basis. We refer to this kernel as AO2MO.

The complexity of the first two steps (Eq. 18 and Eq. 19) is $\mathcal{O}(N_{\text{aux}} N^3_{\text{emb}_f})$ and of the last two steps (Eq. 20 and Eq. 21) is $\mathcal{O}(N_{\text{aux}} N^2_{\text{emb}_f} N^2_{\text{ac}_f})$. If the first two steps are performed on the GPU and the next two on the CPU, we transfer $b^P_{p_{K_1} p_{K_2}}$ (of size $\mathcal{O}(N_{\text{aux}} N^2_{\text{emb}_f})$) back to the CPU. A GPU memory allocation of $2N_{\text{blk}} N^2_{\text{emb}_f}$ is required to perform the calculation.

If, instead, we perform Eq. 18, Eq. 19 and Eq. 20 on the GPU and Eq. 21 on the CPU, we pull $g^{p_{K_1} a_{K_1}}_{p_{K_2} a_{K_2}}$ and $b^P_{p_{K_1} a_{K_1}}$ ($N^2_{\text{emb}_f} N^2_{\text{ac}_f} + N_{\text{aux}} N_{\text{emb}_f} N_{\text{ac}_f}$ data) back and require a memory allocation of $2N_{\text{blk}} N^2_{\text{emb}_f} + N^2_{\text{emb}_f} N^2_{\text{ac}_f}$ on GPUs. Since $N_{\text{aux}} \gg N^2_{\text{ac}_f}$ and $N_{\text{emb}_f} \gg N_{\text{ac}_f}$, this approach significantly reduces data transfer compared to the previous option, where only Eq. 18 and Eq. 19 were executed on the GPU. Furthermore, given that the JK call already requires $3N_{\text{blk}} N^2_{\text{ao}}$ memory and $N_{\text{ao}} \gg N_{\text{emb}_f}$ and $N_{\text{blk}} = 240 > N^2_{\text{ac}_f}$ for most practical cases, this strategy does not impose any additional memory overhead beyond what is required for the JK call.

Finally, performing all 4 steps on a GPU would transfer $2 \times N^2_{\text{emb}_f} N^2_{\text{ac}_f}$ data back to CPU and require a minimum GPU memory allocation of $2N_{\text{blk}} N^2_{\text{emb}_f} + 2N^2_{\text{ac}_f} N^2_{\text{emb}_f}$. In most practical cases, the data transferred is smaller than in the previous approach, as $N_{\text{emb}_f} N_{\text{ac}_f} < N_{\text{aux}}$. Currently, we execute the first three steps on the GPU and the final step on the CPU because Eq. 21 is not relatively expensive. This approach makes CASSCF ERIs no longer a

bottleneck of LASSCF (see Sec. 4), although further optimizations may be explored in the future.

3 Implementation

The `mrh` research code,[77] which implements LASSCF, builds upon the PySCF[66] code. No modification or special version of PySCF is required and users of `mrh` can simply import the relevant PySCF modules and objects needed for LASSCF calculations. Within the LASSCF algorithm, low level functionalities, such as JK calls, are transparently accessed from PySCF. In designing the framework for GPU-accelerated LASSCF calculations, there was a strong emphasis on maintaining a similar level of transparency with PySCF to reduce the installation burden on users while enabling them to fully leverage GPU accelerators. Additionally, there was a strong emphasis on enabling performance-portable calculations, allowing users to take full advantage of GPU accelerators from various vendors, including AMD, Intel, and NVIDIA. This approach aims to support a broader user community. From a software development perspective, the goal was to minimize code duplication while maintaining flexibility in optimizing the mapping of LASSCF and `mrh` to GPUs. This was achieved, in part, by relying on vendor-optimized matrix-multiply routines to ensure good performance.

The majority of the GPU-related code is self-contained in a stand-alone C/C++ library `libgpu`. Python bindings for the C++ functions are created using the header-only pybind11[78] library, facilitating data exchange between the two programming models. This lightweight Python/C++ interface helps keep the `mrh` code organized, as it is only utilized during GPU-enabled runs. The core development of multiconfigurational algorithms in the Python `mrh` code continues uninterrupted, while the algorithms can be patched to leverage accelerated core functionalities for speedup. A key benefit of this development approach is that other algorithms can also take advantage of the same core functionalities for acceleration, without additional development effort.

Adding new GPU-accelerated functionality is straightforward, and as demonstrated below, the overall application speedup is already promising across several multi-metallic systems on NVIDIA and Intel GPUs. We prioritized which functionalities to accelerate using profiling tools, and NVIDIA's profiling tools such as NVIDIA Nsight Systems[79] on Polaris and Intel's profiling tools such as VTune and Advisor with profilers such as THAPI and iprof to ensure multiple GPUs were utilized effectively. These decisions enabled the development of mini-apps for quicker and independent prototyping and optimization of GPU-accelerated kernels. The trade-off of focusing only on key, separate computational tasks is potentially additional data transfers to ensure the remaining CPU code runs correctly.

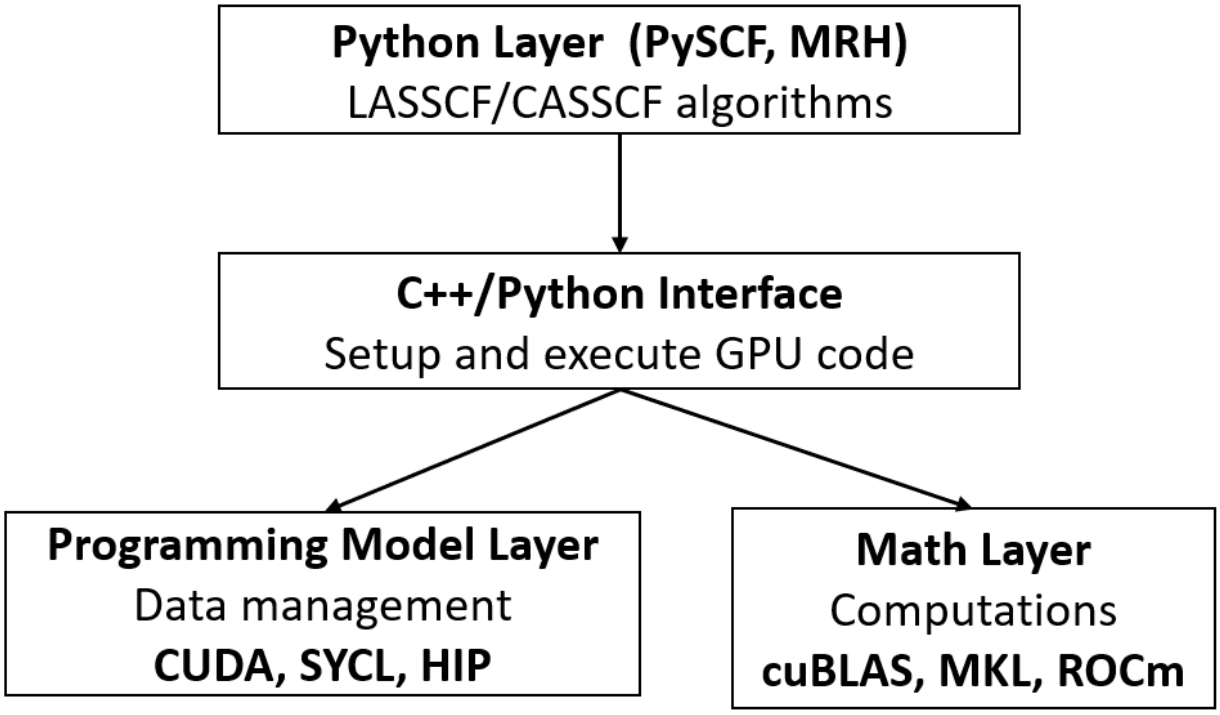


Figure 3: General code structure for offloading computational hotspots.

3.1 Python layer

The LASSCF algorithm leverages several functionalities provided by PySCF. For example, the CASSCF task defined by eq. 3 is solved directly by PySCF drivers, and all effective

potential calculations of LASSCF are performed with PySCF's JK engine.

An example file of how to run LASSCF is provided in Fig. 4. First, we instantiate PySCF's `mol` object with the geometry, charge and basis set information and perform a ROHF with density fitting. The LASSCF object `las` then uses the PySCF's mean field object (`mf`) along with, for this example, two fragments defined with `no1,no2` orbitals and `ne1,ne2` electrons localized on `atom_list1, atom_list2`, respectively. The orbitals are localized on each fragment, and used to perform the LASSCF calculation.

```
from pyscf import gto, scf
mol=gto.M(atom='geom.xyz', basis=basis)
mf=scf.ROHF(mol).density_fit().run()
from mrh.my_pyscf.mcscf.lasscf_async import
↪  LASSCF
las=LASSCF(mf, (no1, no2), (ne1, ne2))
lo=las.set_fragments_
↪  ((atom_list1,atom_list2),mf.mo_coeff)
las.kernel(lo)
```

Figure 4: A sample code to run a LASSCF calculation with a `geom.xyz` geometry file, two fragments of active spaces with `(ne1,ne2)` electrons, `(no1,no2)` orbitals, localized on `(atom_list1,atom_list2)` atom numbers. LASSCF is started with an initial guess orbitals `lo`.

To avoid modifying the PySCF library, we monkey-patched the JK operation (eq. 14-17) and AO2MO functionalities (eq. 18 - 21) using decorators on existing PySCF functions, redirecting these calls to our GPU accelerated versions. This is identical to how `gpu4pyscf`[39,80] modifies code executed at runtime. Additionally, we patched the `mol` object in PySCF to include a `use_gpu` flag, which readily enables branching for GPU usage without cluttering code too much. To minimize changes on the Python side of `mrh`, we used this flag to create if-else branches that direct the code to utilize the GPU-accelerated functionalities where applicable.

A sample code for running a GPU-accelerated calculation is provided in fig. 5. We import the `libgpu` library (subsection 3.2) to access GPU functionalities and call `patch_pyscf` that performs the monkey-patching described above. We initialize the GPU device(s) and pass the `use_gpu` flag in the SCF and LASSCF methods to enable the GPU-accelerated branches. Finally, after retrieving the relevant statistics for GPU usage, we free all allocated memory and release the GPUs. Overall, running a GPU-accelerated calculation is relatively straightforward requiring minimal changes to the Python script.

Figure 5: A sample code to run a GPU-accelerated LASSCF calculation. The LASSCF setup is similar as previous example, with `libgpu` imported for instructing code to use GPU-accelerated library, `gpu4mrh` being used to patch `pyscf` functionalities. `use_gpu` is tagged to LASSCF method that ensures all GPU accelerated branches are used.

```
from mrh.my_pyscf import libgpu
import pyscf
from gpu4mrh import patch_pyscf
from pyscf import gto, scf
gpu = libgpu.libgpu_init()
mol = gto.M(atom='geom.xyz', basis=basis,
↪  use_gpu = gpu)
mf = scf.ROHF(mol).density_fit().run()
from mrh.my_pyscf.mcscf.lasscf_async import
↪  LASSCF
las = LASSCF(mf, (no1, no2), (ne1, ne2),
↪  use_gpu = gpu)
lo = las.set_fragments_
↪  ((atom_list1,atom_list2),mf.mo_coeff)
las.kernel(lo)
libgpu.libgpu_destroy_device(gpu)
```

3.2 C++ layer

The C++ layer is written with portability in mind such that developers (and users) can take advantage of various accelerators without rewriting algorithms. The GPU accelerated C/C++ library `libgpu` is bound to python and exchanges data through pybind11, avoiding deep copies and instead using pointers to data when possible. The library has a concise programming model layer that is responsible for data management and streams/queues for scheduling work on the GPUs. The underlying programming model can be switched out to CUDA, SYCL, or HIP depending on the GPU accelerator. Similarly, math library abstractions for routine matrix multiplications (GEMM and Batched GEMM) are used. Other mathematical operations like transposes and packing/unpacking data were developed as hand-written kernels using CUDA as much of the initial development leveraged NVIDIA GPUs. These hand-written CUDA kernels are then translated with SYCLomatic[81] and HIPify[82] to generate the corresponding SYCL and HIP versions respectively. Only minor modifications to the translated code are made to resolve compilation issues and for the purpose of this paper they have not been optimized further. The high-level algorithms implemented in libgpu, such as JK, leverage the interfaces (programming model and math) such that they only need to be written once, and automatically run on any supported GPU once the needed kernels have been translated. The goal with developing the software in this manner was not to recreate a robust abstraction framework like Kokkos,[83–85] but remain agile while understanding how best to leverage GPUs in LASSCF calculations and keeping additional dependencies to a minimum. It is straightforward to add support for and explore external optimized libraries for select tasks, such as MAGMA[86] and cuTENSOR,[87] without needing to edit the quantum chemistry algorithms. There is also a CPU backend available, but that is only used to aid in the initial development of new algorithms. The performance of the CUDA version on NVIDIA A100 GPUs and the SYCL version on Intel Max Series GPUs is discussed in section 4. Results from the HIP version will be discussed in a future work.

3.3 Algorithms for Performance

The JK and AO2MO functions are fundamental operations that are called thousands and hundreds of times, respectively, within a single LASSCF calculation. Because the implementation is not fully GPU-resident, the GPU portions of the calculation are interleaved with CPU operations. These functions require the same Cholesky vectors as input, and naively transferring them to the GPU every call would be extremely inefficient, as the data transfer time would negate any performance gains. While on-the-fly integral generation techniques on the GPU are available, we opted to store all Cholesky vectors blockwise on the GPUs and use an efficient hashing scheme to access vector blocks. This hashing technique utilizes all GPUs on a node without relying on MPI.

To fully utilize all GPUs, we must ensure that kernels are launched asynchronously on GPUs, and any barriers are removed. Operations involving GPU memory or interactions with CPU pageable memory would typically require the CPU to wait until the operation completes before starting new tasks, such as scheduling additional work on the GPUs. To minimize this delay, new allocations are kept to a minimum, and existing allocations are reused whenever possible. Intermediate results from different kernels executed on GPUs are stored in a common scratch space. Data transfer is performed using pinned memory as necessary to optimize performance.

4 Results and Discussion

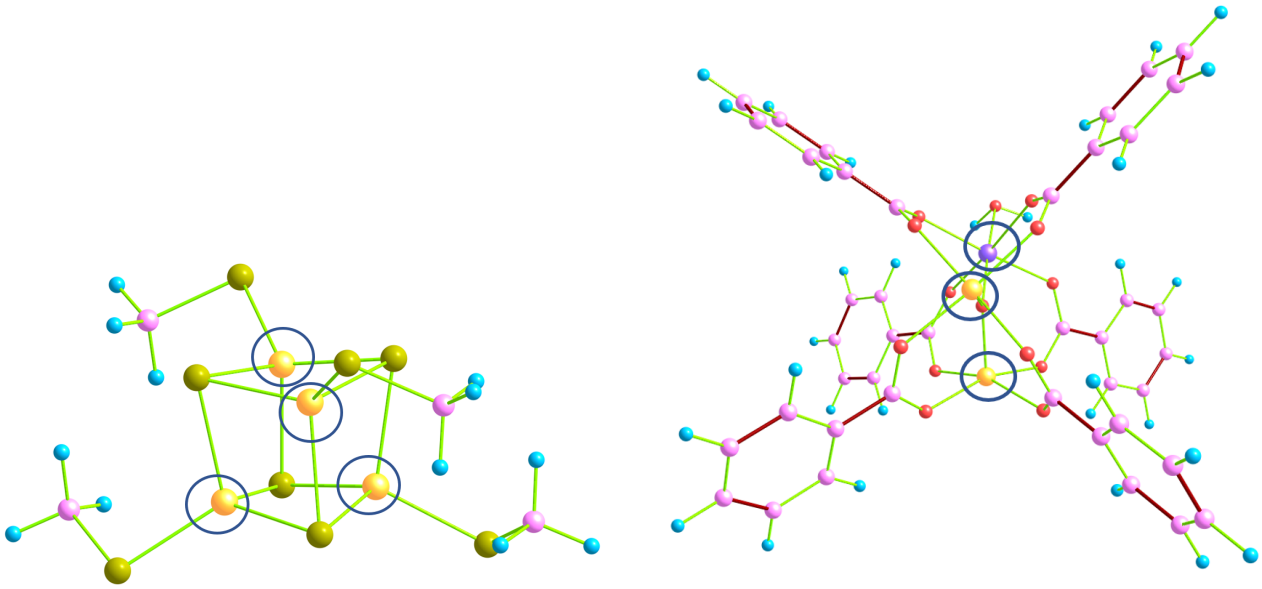
The calculations were performed on two of Argonne's Leadership Computing Facility (ALCF) supercomputers: Polaris and Aurora. Polaris compute nodes have 1 AMD “Milan" 32-core CPU and 4 NVIDIA A100 40GB GPUs. All A100s are linked to each other with NVLink. Aurora compute nodes have 2 Intel Xeon Max Series 52-core CPUs and 6 Intel Data Center Max Series GPUs code-named "Ponte Vecchio (PVC)". All PVCs are linked to each other with Xe link. Each Intel GPU in Aurora consists of two physical stacks that can be separately

targeted to run independent work. All Aurora runs reported here distributed work across individual stacks (i.e. 12 separate queues of overlapping work when running on 6 GPUs). CPU runs are performed on Polaris with a single CPU using 32 threads. Polaris' GPU runs are performed with a single CPU using 32 threads and up to 4 A100 GPUs. Aurora runs are performed using a single CPU with 32 threads only, with up to 6 PVC GPUs. We refer readers to the technical paper on Polaris supercomputer for specific details on the technical paper.[88] Technical paper for Aurora is currently under preparation. Briefly, the Polaris node Milan CPU's peak performance is 1.4 Tera floating point operations (TFLOPs), and each A100 has peak performance of 19.5 TFLOPs.

4.1 Systems under study

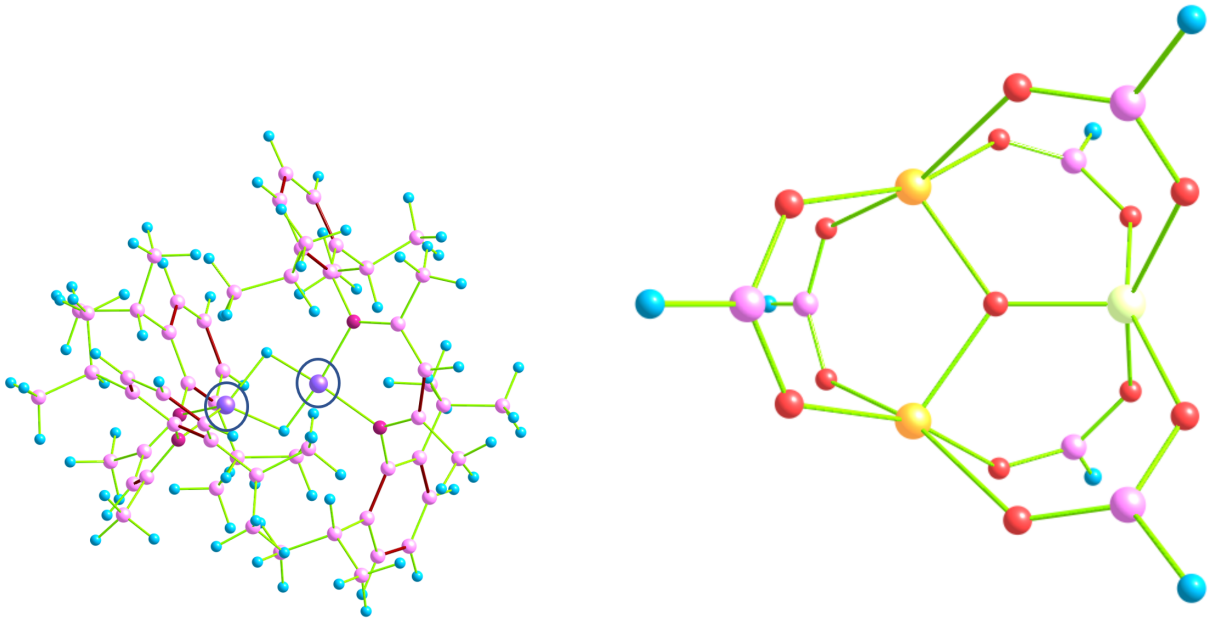
We used polyacetylenes of form $(C_2H_2)_nH_2$, where n is the number of monomers, for basic profiling, testing accuracy, understanding effects of performance with varying system parameters and limits of LASSCF calculations within given resources. This system can however be misleading and be an unreasonable approximation for the workload of a chemically meaningful system. This can lead make priorities skewed for acceleration when the goal is to accelerate LASSCF for meaningful systems. Three multi-metallic systems were considered to assess the performance of the GPU-accelerated LASSCF implementation. As a stress test of the algorithm, we were able to perform at least one LASSCF iteration for a (96e,96o) calculation for a 48 fragment polyacetylene with a GPU accelerated run within an hour.

The first system under consideration is an iron-sulfur (Fe_4S_4) cluster (Fig 6(a)). We refer to this as system **A** in the remainder of the manuscript. Conventionally, the iron sulfur tetramer is thought of as two iron sulfur dimers with $Fe^{+2.5}$ interacting antiferromagentically to form a singlet ground state, we designate two Fe centers as Fe^{+2} and the other two as Fe^{+3}. The local active spaces for Fe^{+3} and Fe^{+2} are (5e,10o) and (6e,10o) respectively, which include the 5 and 6 $3d$ electrons and all the $3d$ and $4d$ orbitals of each Fe center. This results in a total active space of (22e,40o), which is a typical size for systems with multiple



(a) System **A**

(b) System **B**



(c) System **C**

(d) System **D**

Figure 6: Systems under study. (a) Fe_4S_4 cluster: System **A**, (b) Fe_2Ni trimetallic MOF node: System **B**, (c) Homobimetallic nickel hydride catalyst: System **C**, and (d) $AlFe_2$ trimetallic oxo complex: System **D**. Color: Atom - Yellow: Fe, Purple: Ni, Lime Green: Al, Olive Green: S, Pink: C, Red: O, Blue: H.

metal atoms, such as the MoFe cofactor[61,62] and Photosystem II.[89] The Fe and S atoms are described using a cc-pVTZ basis set, while for C and H, a cc-pVDZ basis is used. This gives the total number of orbitals as 660 and the total number of auxiliary orbitals as 4236.

Next, we performed LASSCF on the model of a node of the MIL-127 metal-organic framework (MOF) (system **B**) used in propylene oligomerization.[90] The structure (Fig. 6(b)) consists of one Ni^{+2} and two Fe^{+3} centers, bridged by six benzoate linkers and one H_2O molecule. The active spaces of Ni^{+2} and Fe^{+3}, (8e,10o) and (5e,10o) respectively, include the $3d$ electrons and $3d$ and $4d$ orbitals, resulting in a total active space of (18e,30o). The basis set used for Ni, Fe and central O is cc-pVTZ[91] and cc-pVDZ[91] is used for the rest of the atoms. The total number of orbitals in this system is 1164 and the total number of auxiliary orbitals is 8905.

The third test system is a homobimetallic nickel hydride catalyst (system **C**) shown in Fig. 6 (c). The system has two Ni^{+2} centers, corresponding to an active space of (8e,10o) on each atom which includes all $3d$ electrons and orbitals and $4d$ orbitals. This compound can be immobilized on silica support to be used for hydrogenation of unsaturated hydrocarbons.[92] The Ni atoms were modeled with the def2-TZVP basis while all other atoms with the def2-SVP basis. This gives 1378 orbitals in total and the total number of auxiliary orbitals is 10296.

The rationale for selecting these systems is to explore cases with different number of fragments, active space and basis set sizes. From **A** to **C** the number of fragments and total active space size decreases, while the number of basis functions increases. Strictly speaking, performance from **A** to **C** is not directly comparable as the total active space, number of fragments, number of basis functions and complexity of each part (number of iterations in each SCF cycle of CASSCF and recombination) changes, making it difficult to predict the time-to-solution going from **A** to **B** to **C**.

After those heuristics, we present the results denoting the end-to-end time to converge a LASSCF calculation for **A** and **B** with the same initial guess both with CPU-only and

GPU-accelerated implementations. The results presented below focus exclusively on performance profiling and the differences between GPU and CPU implementations. We computed total electronic energies for fixed geometries; these calculations do not explore any chemical properties of the systems. Their sole purpose is to demonstrate the feasibility of such computations.

While end-to-end time of a converged LASSCF calculation provides a way to evaluate an implementation's practical usefulness, it may be difficult to perform a detailed analysis of the performance of the accelerated implementation using this data for two reasons. First, the overall optimization may require different number of LASSCF cycles with the same input due to different convergence paths, including overcoming local minima, leading to different workloads (see SI fig. S1-S6). Second, the workload for various parts of a single LASSCF cycle, like CASSCF or recombination steps can vary substantially cycle to cycle (see SI fig S1-S6). For these reasons, we also present more robust tests of CPU-only and GPU-accelerated LASSCF iterations where two parameters, namely, maximum number of recombination cycles and total number of LASSCF cycles are arbitrarily fixed at 10 to give similar workloads for all runs for a given system.

We also performed a CASSCF calculation (not LASSCF) on a Al-Fe oxide cluster (system **D**), as shown in Fig. 6. The rational for these calculations is that, as discussed in Subsection 2.3, the effective potentials and CASSCF ERIs dominate the computational cost in fragment-based calculations. Therefore, accelerating these functionalities should also result in performant CASSCF calculations. We used an (11e,10o) active space, including all $3d$ orbitals and electrons for the two Fe centers. Hydrogen and carbon atoms were modeled with the cc-pVDZ[91] basis set, while oxygen, aluminum, and iron were modeled with the cc-pVTZ[91] basis set, resulting in a total of 674 atomic basis functions and a total of 3998 auxiliary basis functions.

Finally, for completeness, we demonstrate how varying the number of GPUs impacts the performance of LASSCF (system **A**) and CASSCF (system **D**). Analyzing the calculation

profiles helps further identify potential serialization and its effect on performance.

The atomic coordinates of the systems, input files, output files, and a sample analyzing script for all calculations can be found in ref. 93.

4.2 LASSCF scaling heuristics with polyacytelene systems

The plots for all runs are presented in SI Sec. V. As we increase the basis set from 6-31g to cc-pVDZ to cc-pVTZ while maintaining the same number of fragments and the total active space size where a total of ((24,24) active space is partitioned into 12 fragments of (2,2), the processing cost escalates most rapidly. This is because it predominantly comprises almost entirely of JK calls that scale with system size. Recombination costs go up with the system size, but contain other computations as well, making the scaling not so dramatic. Finally, CASSCF costs also increase, but tend to do so slowly since the embedding size does not increase linearly with system size. Consequently, we also see an increase in the acceleration as JK calls are highly accelerated.

Changing fragment active space sizes while keeping the total active space constant, i.e., a (24,24) active space being split into 2 spaces of (12,12), 3 spaces of (8,8), 4 spaces of (6,6), 6 spaces of (4,4) and 12 spaces of (2,2) increased the cost of CASSCF processing linearly with the number of fragments. CASSCF costs decrease due to decrease in FCI costs and possible simplification of fragment leading to a lower number of CASSCF iterations being performed. Recombination costs should remain about the same.

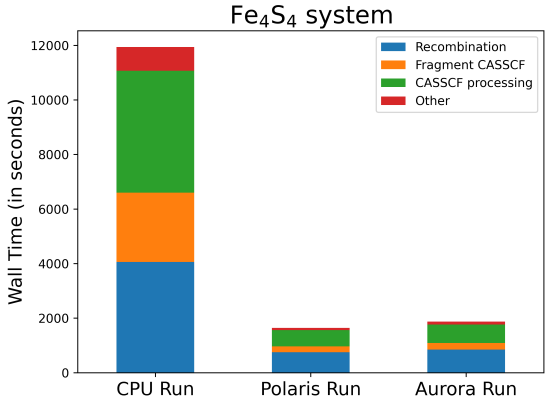
Finally, while varying the total active space as 2 spaces of (4,4), 2 spaces of (6,6), 2 spaces of (8,8) and 2 spaces of (12,12), we expect the cost of LASSCF to grow due to increase in FCI costs. However, the FCI problems are still very small compared to other parts of the algorithm. Therefore, no clear heuristics can be derived for this system.

4.3 Performance of LASSCF and CASSCF for multi-metallic systems

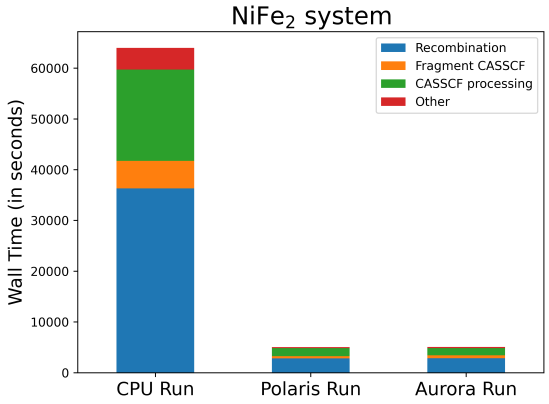
The performance of LASSCF and CASSCF till convergence is presented in Fig. 7. The figure presents a CPU-only run on a Polaris node, a GPU-accelerated run with Polaris (4 A100 GPUs) and with Aurora (4 PVC GPUs). Only 4 of the 6 GPUs on an Aurora node were used in these plots in an attempt to make an apples-to-apples comparison. The results are further discussed in the first two and third rows of table 2 for LASSCF and CASSCF respectively. We present the wall time for all runs, their speedups, and GPU active time, which shows how heavily GPUs are being used. Finally, we discuss the speedup of individual parts of LASSCF in table 3 in the first two columns.

LASSCF to Convergence: System **A** LASSCF is accelerated by about 7x for both architectures, and **B** LASSCF is accelerated by about 13x for both architectures. Going from **A** to **B**, the workload on recombination and CASSCF processing increases significantly due to increase in the N_{ao}, resulting in JK operations with substantially more work and ultimately leading to a higher speedups. Fragment CASSCFs, on the other hand, are similarly accelerated because the individual fragments are similar in embedding size and have similar active spaces.

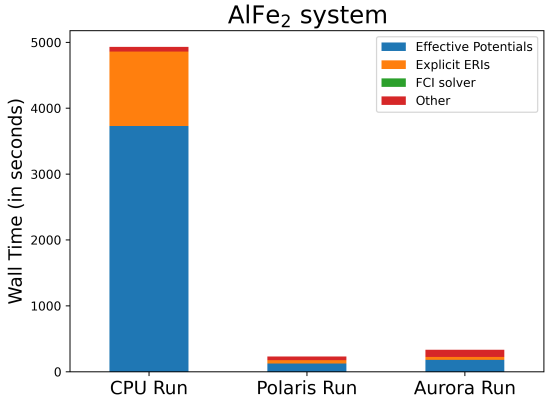
CASSCF to convergence: For system **D**, profiling CASSCF in manner similar to Fig. 2 showed that the effective potentials take about 77% of the wall time, and ERIs take about 22% of the wall time (Fig. 7(c)), the functionalities that we accelerated for LASSCF. Other parts of the calculation, including the exact FCI solver (not visible as a small green bar) are relatively very small for this system. The GPU-runs are accelerated by 15-20x. Since over 98% of the CASSCF runtime was from kernels that were accelerated, we see a very high relative speedup. This speedup is significantly higher compared to fragment CASSCFs ran within LASSCF since each cycle performs multiple CASSCFs of varying sizes and complexity, and as the number of LASSCF cycles goes on, fragment CASSCFs tend to converge very quickly in 3-4 iterations, leading to a much lower workload and hence



(a) System **A** LASSCF performance with Polaris and Aurora Nodes.



(b) System **B** LASSCF performance with Polaris and Aurora Nodes.



(c) System **D** CASSCF performance with Polaris and Aurora Nodes.

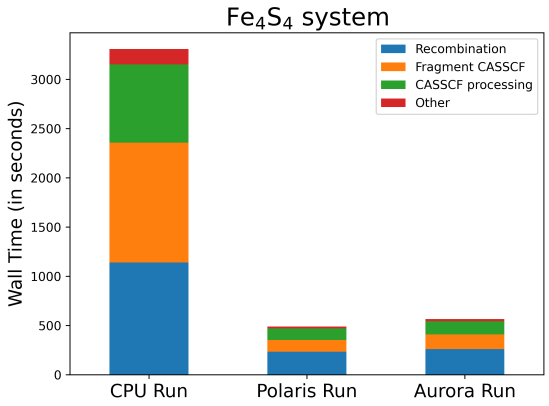
Figure 7: Calculation profiles for: (a) LASSCF on system **A** with (22e,40o) active space,(b) LASSCF on system **B** with (18e,30o) active space, (c) CASSCF on system **D** with (11e,10o) active space, with Polaris and Aurora nodes. Resources used: Polaris: Milan CPU 32 Threads + 4 A100 GPUs; Aurora: Max Series CPU 32 Threads + 6 PVC GPUs with 2 queues per GPU.

lower speedups. Additionally, the embedding space size is significantly smaller in LASSCF's CASSCFs, leading to smaller workloads. For a smaller CASSCF problem, we have presented absolute performance metrics in SI section VII.

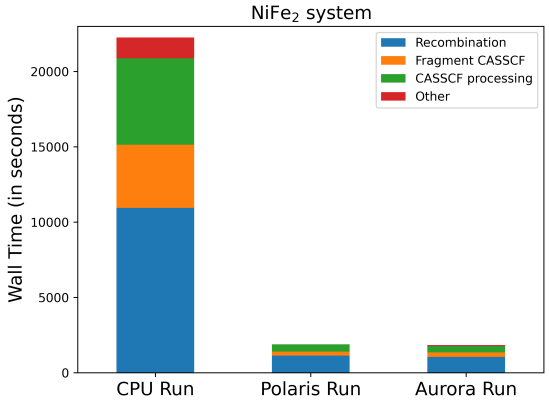
Controlled Workload LASSCF: For a more controlled workload comparison, each LASSCF run for **A**, **B** and **C** were executed with identical parameters: LASSCF maximum cycles were fixed to 10, CASSCF maximum cycles were fixed at 50, and recombination maximum cycles were fixed at 10. CASSCF or recombination may converge earlier than the maximum cycles but this setup limits unreasonably large number of iterations in one phase of the calculation, which may happen due to a variety of reasons such as local minimas or moving out of convergence basin altogether for a specific iteration. The results are presented in Fig. 8 and further discussed in the last three rows of table 2 and last three columns of table 3.

For system **A** (fig. 8(a)) and **B** (fig. 8(b)), the processing has the same speedup as running till convergence because the number of calls is the same for both CPU and GPU runs. We believe that the recombination speedup from unit runs are much closer to actual performance because the workload can vary substantially in production runs due to numerical convergence of the SCF algorithms.

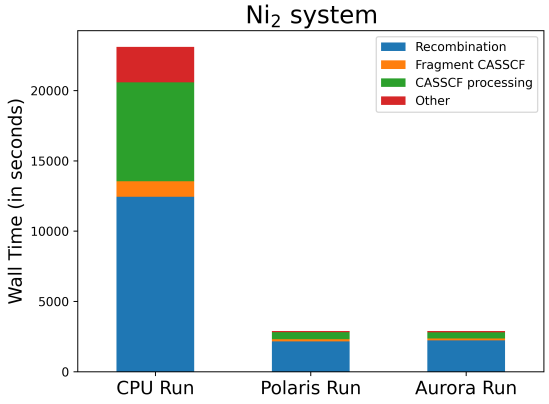
From **A** to **C**, the processing step gets more accelerated due to it being mostly a JK call that gets accelerated with increased workload. The recombination step performance increases from **A** to **B**, and decreases from **B** to **C**. From **A** to **B**, it's simply an increased workload leading to higher speedup. However, for **C**, the calculations have highly variable workloads among runs. For the specific run presented, the CPU run perform 37 total SCF cycles during the recombination step over 10 LASSCF cycles, compared to 51 of Polaris and 55 of Aurora run. Rerunning the same calculations may change the number of cycles significantly. Comparing per recombination SCF cycle speedups in system **C**, we get an approximate speedup of 8x, much closer to **B**. For fragment CASSCFs, we take all of them to have about the same speedups because the embedding space size and active space size is similar for



(a) System **A** LASSCF performance with Polaris and Aurora Nodes.



(b) System **B** LASSCF performance with Polaris and Aurora Nodes.



(c) System **C** LASSCF performance with Polaris and Aurora Nodes.

Figure 8: Calculation profiles for: (a) LASSCF on system **A** with (22e,40o) active space,(b) LASSCF on system **B** with (18e,30o) active space, (c) LASSCF on system **C** with (11e,10o) active space, with Polaris and Aurora nodes for controlled workloads. Resources used: Polaris: Milan CPU 32 Threads + 4 A100 GPUs; Aurora: Max Series CPU 32 Threads + 6 PVC GPUs with 2 queues per GPU.

all systems. The difference in cost comes from two areas. First, the number of CASSCF steps can differ significantly in the runs with same input based on convergence patterns, and second, often the CASSCF can converge quickly, sometimes in one or two cycles, within a LASSCF cycle, leading to insufficient workloads for performance to be apparent.

We note that a higher speedup is generally characterized by higher amount of time spent utilizing the GPUs, which is expected for larger workloads and when the CPU-GPU data transfers are minimized. The overall GPU usage is still low with considerable work still being done on the CPU, but we are achieving 7-10x speedups for LASSCF with 25% active time, and about 20x speedup for CASSCF with 50% active time. Individual bottleneck acceleration and overall acceleration does not exactly match the peak efficiency of processing power provided because of several reasons that include modest GPU utilization, CPU-only kernels, single-GPU kernels, communications overheads with GPU, and serialization and load imbalance with multi-GPU kernels. Effort is underway to further improve performance by moving more of the computation to GPUs and make existing computations more efficient.

4.4 Performance scaling with multiple GPUs

For scaling tests, the Polaris runs are performed on 1 CPU with 32 cores and varying the number of GPUs as 1, 2 and 4. The Aurora runs are performed on 1 CPU with 32 cores and number of GPUs are varied as 1, 2, 4 and 6. We perform the same CASSCF run on system **D** as in (fig. 7(c)) subsection and the same LASSCF run on system **A** (fig. 8(a)). For each run, we plot the time for each phase of the calculation in Fig. 9 and the relevant statistics are presented in Table 4.

LASSCF and CASSCF scaling (Fig. 9), in the case of the Polaris node, performance increases visibly when going from 1 GPU to 4 GPU; however, the scaling is not ideal. For the Aurora node, end-to-end time decreases from 1 GPU to 2 GPUs, remains constant with 4 GPUs, and then increases again on 6 GPUs. The accumulation of results from multiple GPUs to CPU is serialized in this implementation and is one of the main reasons for the

Table 2: Performance of LASSCF and CASSCF on Polaris and Aurora nodes. Speedup shown in GPU Run Wall Time column in parenthesis. % active time for GPU is shown in GPU Time Column.

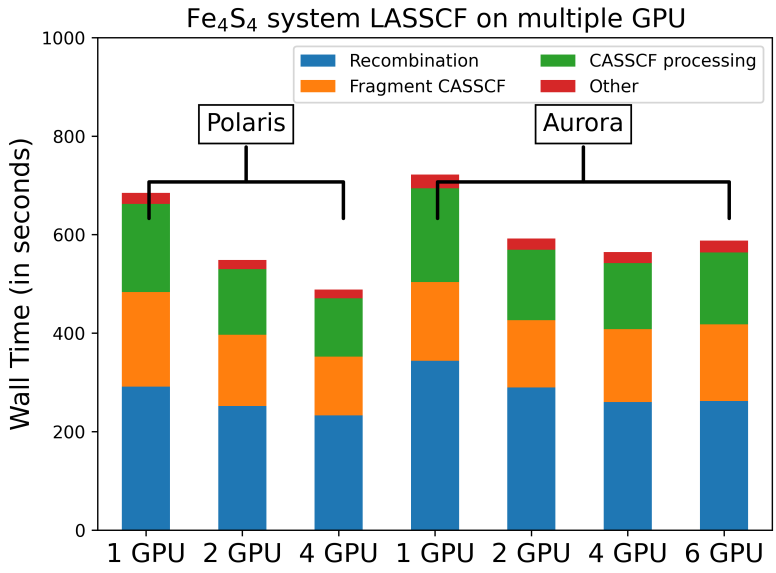
		CPU Only	GPU Wall Run Time (s)		GPU Time (s)	
Method	System	Wall Time (s)	Polaris	Aurora	Polaris	Aurora
LASSCF convergence	**A**	11938	1644 (7.3x)	1927 (6.2x)	445 (26%)	736 (38%)
	B	63993	5045 (12.6x)	5044 (12.6x)	1994(39%)	2057 (40.7%)
CASSCF	**D**	4931	230 (21.4x)	332 (14.9x)	146 (63%)	195 (59%)
LASSCF unit	**A**	3307	488 (6.8x)	564 (5.9x)	131 (27%)	204 (36%)
	B	22260	1842 (12.1x)	1833 (12.1x)	747 (40%)	753 (41%)
	C	23105	2895 (8x)	2214 (10.5x)	1217 (42%)	998 (45%)

slowdown on 6 Aurora GPUs (i.e. 12 separate SYCL queues). This effect is compounded by load imbalance, where an unevenly distributed workload, such as 1 GPU getting three blocks of work and the rest getting two blocks would still cost three blocks' worth of time. A similar effect is seen in CASSCF runs but the penalty is more pronounced since the GPU accelerated kernels have started to account for the majority of run time. Removing these deficiencies is a work currently in progress to improve the code.

Table 3: Speedup of individual parts of LASSCF with Polaris and Aurora Nodes

		Convergence		Units		
LASSCF part	Node	**A**	**B**	**A**	**B**	**C**
Processing	Polaris	7.5	11.2	6.7	11.6	14.2
	Aurora	6.6	12.7	5.9	14.0	16.1
CASSCF	Polaris	11.8	13.0	10.2	16.0	7.0
	Aurora	10.6	9.5	8.2	13.7	8.5
Recombination	Polaris	5.4	12.8	4.9	9.6	5.8
	Aurora	4.8	12.8	4.4	10.4	5.6

(a) LASSCF runs on system **A** with varying number of NVIDIA and Intel GPUs



(b) CASSCF runs on system **D** with varying number of NVIDIA and Intel GPUs

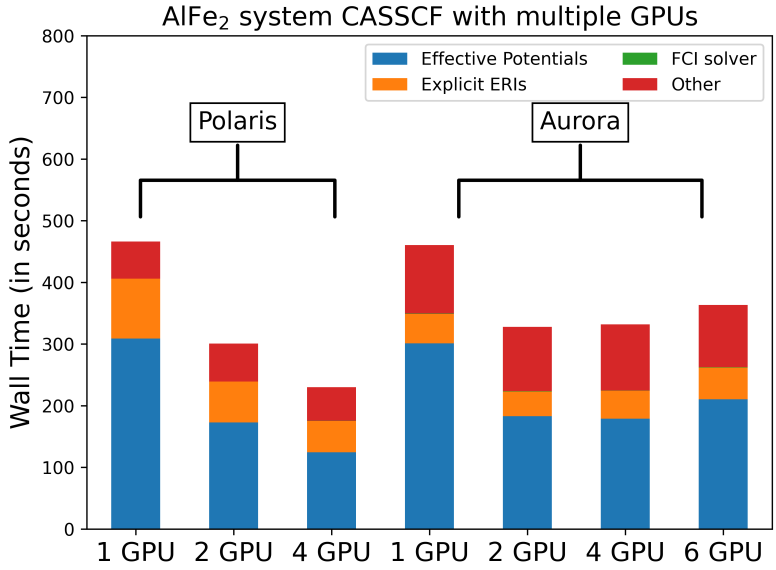


Figure 9: Scaling of (a) LASSCF and (b) CASSCF with varying GPUs. Resources used: Polaris: Milan CPU 32 Threads + 1/2/4 A100 GPUs; Aurora: Max Series CPU 32 Threads + 1/2/4/6 PVC GPUs with 2 queues per GPU.

Table 4: Scaling performance of LASSCF and CASSCF with multiple GPUs

Method	n_{GPU}	Wall Time (s)		GPU Time (s) (Active time %)	
		Polaris	Aurora	Polaris	Aurora
CASSCF on **D**	1	466	461	374 (80%)	320 (69%)
	2	301	328	209 (69%)	195 (59%)
	4	230	332	146 (63%)	195 (59%)
	6	–	363	–	234 (64%)
LASSCF on **A**	1	685	722	323 (47%)	363 (50%)
	2	549	592	184 (33%)	229 (39%)
	4	488	565	131 (26%)	204 (36%)
	6	–	587	–	228 (39%)

5 Conclusions

We have demonstrated an overall speedup of up to 10x for the LASSCF method using GPUs and up to 30x speedup in some fundamental operations. We were also able to run at least a full LASSCF cycle with a (96e,96o) active space on a polymer chain. We were able to converge iron-sulfur tetramer with up to (22e,40o) active space and 660 orbitals and also able to converge a MIL-127 MOF node with an (18e,30o) active space and about 1200 atomic orbitals. Focusing on speeding up low-level computational bottlenecks has yielded up to 20x speedup of CASSCF algorithm without additional effort. We have used density fitting algorithms to keep memory cost low and followed a code development philosophy to more easily explore GPU algorithms and retain some degree of portability. To that end, we demonstrate portable performance with NVIDIA A100 GPUs and Intel Max Series GPUs without further tuning (as well as on an AMD MI250 GPU not discussed here). Work continues in moving more bottlenecks to GPU-accelerated algorithms and improving the scaling efficiency of leveraging all GPUs on larger compute nodes.

6 Outlook

Within the paradigm of making the program more performant with the same resources, i.e., a single node with multiple GPUs, optimization of individual kernels and GPU accelerated versions of more kernels is high on our priority list. Examples of such kernels within CASSCF would be matrix-vector products during Hessian calculations and within LASSCF, CASSCF bottlenecks, along with matrix-vector products during LASSCF Hessians, transition density matrices and reorthogonalization algorithms would be ideal targets depending on the current cost. Additionally, we currently solve the CI problem exactly with FCI on CPUs. If this step becomes a bottleneck, GPU accelerated version of variations of FCI solvers such as non orthogonal CI,[63] DMRG,[61,62] v2RDM[67] are available. These could be transparently switched out in case of using NVIDIA GPUs. Further work is needed to solve the CI

problems on Intel and AMD GPUs. All of the new development highlighted above will be guided by profiling efforts that hint at the next biggest bottlenecks. To access larger systems, available memory on a GPU to store Cholesky vectors becomes a bottleneck. Since transferring Cholesky vectors every iteration is not a reasonable solution, the storage can be replaced by GPU-accelerated integral generation schemes[40,42] or distributed parallelism via MPI can be used to leverage multiple compute nodes. The work can be easily switched to using single precision in all operations to reduce memory cost, most likely, without the loss of accuracy as has been shown by several others for HF,[94] MP2,[45,95–97]and CC.[97,98] With GPUs being optimized for lower than single precision datatypes such as BFloat16 (BF16), floating point 16-bit precision (FP16), and even integers, there is the potential for significant gains if reduced precision datatypes can be leveraged. While we do not perform calculations related to spin ladders in this paper, differences in energy of spin states can be on the order of 1 cm $^{-1}$, which is about $2 \times 10^{-5} E_h$ on an order of $1 \times 10^{3} E_h$, a precision higher than FP16 or BF16, and nearing the edge of FP32, which may suggest the need for mixed-precision algorithms. Algorithmic advances and optimizations to incorporate reduced precision data types is becoming ever more important considering the GPU roadmaps from vendors. With all of the GPU-specific code being abstracted within the programming model and device software layers supporting the different GPU vendors, we have been able to compile and run the code on newer generations of GPUs, such as NVIDIA GH200, with minimal effort. For maximum performance it will be important to fully leverage reduced datatypes whether directly or via software emulation. It will also be important to ensure individual GPUs are still fully utilized when running science workloads and additional levels of parallelism may need to be explored (e.g. executing multiple streams/queues per device similar to what we currently do in Aurora).

To access computational resources beyond one node, we plan to use MPI. Currently, in LASSCF, all fragment CASSCFs are performed sequentially but are fundamentally independent of each other. This presents an opportunity for MPI parallelization distributing

fragment CASSCF tasks across compute nodes in a load-balanced fashion. Additionally, the recombination step can also be split up into smaller workloads that focus on recombining two or more fragments at a time instead of all the fragments together, and be parallelized with MPI. Load balancing of LASSCF is arguably more complicated than methods like HF, DFT, MP2 or CASSCF in general because there are several phases of calculation that can have varying complexity and scaling depending on the system under study. A single large fragment or large number of fragments makes CASSCF expensive, a large number of fragments could make recombination challenging, a system with few active space orbitals and a lot of inactive orbitals could mean a cheap CASSCF but expensive recombination.

Finally, post LASSCF methods like LASSI or PDFT allow us to correct LASSCF wave function when direct product of active spaces gives qualitatively incorrect wave function. [25,99,100] Acceleration of those calculations will reduce simulation time drastically in performing chemically meaningful calculations of complex multi-metallic systems.

7 Acknowledgments

This work is supported as part of the Computational Chemical Sciences Program, under Award DE-SC0023382, funded by the U.S. Department of Energy, Office of Basic Energy Sciences, Chemical Sciences, Geosciences, and Biosciences Division. This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of

Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan

8 Conflict of Interest

The authors declare no competing conflict of interest.

9 Supporting Information

The supporting information contains

1. References to geoemtry
2. Converged active space orbitals and their occupancy.
3. Iteration wise calculation profile for all LASSCF runs
4. Calculation profiles for polyenes with controlled variation of parameters
5. Discussion on use of profiling to create more performant algorithms.

10 Data Availability

Data is available on ref. 93.

References

(1) Szabo, A.; Ostlund, N. S. *Modern quantum chemistry: introduction to advanced electronic structure theory*; Courier Corporation, 1996.

(2) Kohn, W.; Sham, L. J. Self-consistent equations including exchange and correlation effects. *Phys. Rev.* **1965**, *140*, A1133.

(3) Møller, C.; Plesset, M. S. Note on an approximation treatment for many-electron systems. *Phys. Rev.* **1934**, *46*, 618.

(4) Čížek, J. On the correlation problem in atomic and molecular systems. Calculation of wavefunction components in Ursell-Type expansion Using Quantum-Field theoretical methods. *J. Chem. Phys.* **1966**, *45*, 4256–4266.

(5) Čížek, J. On the use of the cluster expansion and the technique of diagrams in calculations of correlation effects in atoms and molecules. *Adv. Chem. Phys* **1969**, *14*, 35–89.

(6) Čížek, J.; Paldus, J. Correlation problems in atomic and molecular systems III. Rederivation of the coupled-pair many-electron theory using the traditional quantum chemical methods. *Int. J. Quantum Chem.* **1971**, *5*, 359–379.

(7) Bartlett, R. J.; Musiał, M. Coupled-cluster theory in quantum chemistry. *Reviews of Modern Physics* **2007**, *79*, 291.

(8) Roos, B. O.; Taylor, P. R.; Sigbahn, P. E. A complete active space SCF method (CASSCF) using a density matrix formulated super-CI approach. *Chem. Phys.* **1980**, *48*, 157–173.

(9) Huron, B.; Malrieu, J. P.; Rancurel, P. Iterative perturbation calculations of ground and excited state energies from multiconfigurational zeroth-order wave functions. *J. Chem. Phys.* **1973**, *58*, 5745.

(10) Cimiraglia, R.; Persico, M. Recent advances in multireference second order perturbation CI: The CIPSI method revisited. *J. Comput. Chem.* **1987**, *8*, 39.

(11) Miralles, J.; Castell, O.; Caballol, R.; Malrieu, J. P. Specific CI calculation of energy differences: Transition energies and bond energies. *Chem. Phys.* **1993**, *172*, 33.

(12) Neese, F. A spectroscopy oriented configuration interaction procedure. *J. Chem. Phys.* **2003**, *119*, 9428.

(13) Tubman, N. M.; Lee, J.; Takeshita, T. Y.; Head-Gordon, M.; Whaley, K. B. A deterministic alternative to the full configuration interaction quantum Monte Carlo method. *J. Chem. Phys.* **2016**, *145*, 044112.

(14) Holmes, A. A.; Tubman, N. M.; Umrigar, C. J. Heat-bath configuration interaction: An efficient selected configuration interaction algorithm inspired by heat-bath sampling. *J. Chem. Theory Comput.* **2016**, *12*, 3674.

(15) White, S. R. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.* **1992**, *69*, 2863.

(16) Malmqvist, P. A.; Rendell, A.; Roos, B. O. The restricted active space self-consistent-field method, implemented with a split graph unitary group approach. *J. Phys. Chem.* **1990**, *94*, 5477.

(17) Olsen, J.; Roos, B. O.; Jörgensen, P.; Jensen, H. J. A. Determinant based configuration interaction algorithms for complete and restricted configuration interaction spaces. *J. Chem. Phys.* **1988**, *89*, 2185.

(18) Olsen, J.; Roos, B. O.; Jörgensen, P.; Jensen, H. J. A. Determinant based configuration interaction algorithms for complete and restricted configuration interaction spaces. *J. Chem. Phys.* **1988**, *89*, 2185.

(19) Ma, D.; Li Manni, G.; Gagliardi, L. The generalized active space concept in multiconfigurational self-consistent field methods. *J. Chem. Phys.* **2011**, *135*, 044128.

(20) Vogiatzis, K. D.; Li Manni, G.; Stoneburner, S. J.; Ma, D.; Gagliardi, L. Systematic expansion of active spaces beyond the CASSCF limit: A GASSCF/SplitGAS benchmark study. *J. Chem. Theory Comput.* **2015**, *11*, 3010.

(21) Odoh, S. O.; Manni, G. L.; Carlson, R. K.; Truhlar, D. G.; Gagliardi, L. Separated-pair approximation and separated-pair pair-density functional theory. *Chem. Sci.* **2016**, *7*, 2399.

(22) Jiménez-Hoyos, C. a.; Scuseria, G. E. Cluster-based mean-field and perturbative description of strongly correlated fermion systems. Application to the 1D and 2D Hubbard model. *Phys. Rev. B* **2015**, *92*, 085101.

(23) Hermes, M. R.; Pandharkar, R.; Gagliardi, L. Variational localized active space self-consistent field method. *J. Chem. Theory Comput.* **2020**, *16*, 4923.

(24) Hermes, M. R.; Gagliardi, L. Multiconfigurational self-consistent field theory with density matrix embedding: The localized active space self-consistent field method. *J. Chem. Theory Comput.* **2019**, *15*, 972.

(25) Pandharkar, R.; Hermes, M. R.; Cramer, C. J.; Gagliardi, L. Localized Active Space-State Interaction: a Multireference Method for Chemical Insight. *J. Chem. Theory Comput.* **2022**, *18*, 6557–6566.

(26) Mavrandonakis, A.; Vogiatzis, K. D.; Boese, A. D.; Fink, K.; Heine, T.; Klopper, W. Ab initio study of the adsorption of small molecules on metal–organic frameworks with oxo-centered trimetallic building units: the role of the undercoordinated metal ion. *Inorg. Chem.* **2015**, *54*, 8251–8263.

(27) Vitillo, J. G.; Bhan, A.; Cramer, C. J.; Lu, C. C.; Gagliardi, L. Quantum chemical characterization of structural single Fe (II) sites in MIL-type metal–organic frameworks for the oxidation of methane to methanol and ethane to ethanol. *ACS Catal.* **2019**, *9*, 2870–2879.

(28) Khurana, R.; Agarawal, V.; Liu, C. Exploring the Computational Aspects of Propylene Oligomerization Catalysis Using M2M Type Trimetallic MOF Nodes. *J. Phys. Chem. C* **2024**, *128*, 16986–16995.

(29) Sharma, P.; Truhlar, D. G.; Gagliardi, L. Magnetic coupling in a tris-hydroxo-bridged chromium dimer occurs through ligand mediated superexchange in conjunction with through-space coupling. *J. Am. Chem. Soc.* **2020**, *142*, 16644–16650.

(30) Campanella, A.; Gin, A.; Sung, S.; Jackson, C.; Martinez, R.; Ozarowski, A.; Bhowmick, I.; Zadrozny, J. Amplifying the Temperature Sensitivity of Zero-Field Splitting in Mn (II) Through Ligand Tuning. **2023**,

(31) Knizia, G.; Chan, G. K.-L. Density matrix embedding: A simple alternative to dynamical mean-field theory. *Phys. Rev. Lett.* **2012**, *109*, 186404.

(32) Ma, H.; Govoni, M.; Galli, G. Quantum simulations of materials on near-term quantum computers. *npj Comput. Mater.* **2020**, *6*, 85.

(33) Top 500 Supercomputers - June 2024. `https://top500.org/lists/top500/2024/06/`, Accessed: 2024-06-17.

(34) Yasuda, K. Accelerating density functional calculations with graphics processing unit. *J. Chem. Theory Comput.* **2008**, *4*, 1230–1236.

(35) 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.

(36) Galvez Vallejo, J. L.; Snowdon, C.; Stocks, R.; Kazemian, F.; Yan Yu, F. C.; Seidl, C.; Seeger, Z.; Alkan, M.; Poole, D.; Westheimer, B. M.; others Toward an extreme-scale electronic structure system. *J Chem. Phys.* **2023**, *159*.

(37) Spiga, F.; Girotto, I. phiGEMM: a CPU-GPU library for porting Quantum ESPRESSO on hybrid systems. 2012 20th Euromicro international conference on parallel, distributed and network-based processing. 2012; pp 368–375.

(38) Hohenstein, E. G.; Luehr, N.; Ufimtsev, I. S.; Martínez, T. J. An atomic orbital-based formulation of the complete active space self-consistent field method on graphical processing units. *J. Chem. Phys.* **2015**, *142*.

(39) 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.

(40) Kussmann, J.; Ochsenfeld, C. Employing opencl to accelerate ab initio calculations on graphics processing units. *J. Chem. Theory Comput.* **2017**, *13*, 2712–2716.

(41) Andrade, X.; Aspuru-Guzik, A. Real-space density functional theory on graphical processing units: Computational approach and comparison to Gaussian basis set methods. *J. Chem. Theory Comput.* **2013**, *9*, 4360–4373.

(42) Alkan, M.; Pham, B. Q.; Del Angel Cruz, D.; Hammond, J. R.; Barnes, T. A.; Gordon, M. S. LibERI—A portable and performant multi-GPU accelerated library for electron repulsion integrals via OpenMP offloading and standard language parallelism. *J Chem. Phys.* **2024**, *161*.

(43) Williams-Young, D. B.; Asadchev, A.; Popovici, D. T.; Clark, D.; Waldrop, J.; Windus, T. L.; Valeev, E. F.; de Jong, W. A. Distributed memory, GPU accelerated Fock construction for hybrid, Gaussian basis density functional theory. *J. Chem. Phys.* **2023**, *158*, 234104.

(44) Storchi, L.; Bellentani, L.; Hammond, J.; Orlandini, S.; Pacifici, L.; Antonini, N.; Belpassi, L. Acceleration of the Relativistic Dirac–Kohn–Sham Method with GPU: A Pre-Exascale Implementation of BERTHA and PyBERTHA. *J. Chem. Theory Comput.* **2025**, *21*, 3460–3475.

(45) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-Bedolla, C.; Aspuru-Guzik, A. Accelerating resolution-of-the-identity second-order Møller- Plesset quantum chemistry calculations with graphical processing units. *J. Phys. Chem. A* **2008**, *112*, 2049–2057.

(46) Stocks, R.; Palethorpe, E.; Barca, G. M. High-performance multi-GPU analytic RI-MP2 energy gradients. *J. Chem. Theory Comput.* **2024**, *20*, 2505–2519.

(47) Ma, W.; Krishnamoorthy, S.; Villa, O.; Kowalski, K. GPU-based implementations of the noniterative regularized-CCSD (T) corrections: applications to strongly correlated systems. *J. Chem. Theory Comput.* **2011**, *7*, 1316–1327.

(48) DePrince III, A. E.; Hammond, J. R. Coupled cluster theory on graphics processing units I. The coupled cluster doubles method. *J. Chem. Theory Comput.* **2011**, *7*, 1287–1295.

(49) Boguslawski, K.; Brzęk, F.; Chakraborty, R.; Cieślak, K.; Jahani, S.; Leszczyk, A.; Nowak, A.; Sujkowski, E.; Świerczyński, J.; Ahmadkhani, S.; Kędziera, D.; Kriebel, M. H.; Żuchowski, P. S.; Tecmer, P. PyBEST: Improved functionality and enhanced performance. *Comput. Phys. Comm.* **2024**, *297*, 109049.

(50) Kriebel, M. H.; Tecmer, P.; Gałynska, M.; Leszczyk, A.; Boguslawski, K. Accelerating Pythonic Coupled-Cluster Implementations: A Comparison Between CPUs and GPUs. *J. Chem. Theory Comput.* **2024**, *20*, 1130–1142.

(51) Asadchev, A.; Gordon, M. S. New multithreaded hybrid CPU/GPU approach to Hartree–Fock. *J. Chem. Theory Comput.* **2012**, *8*, 4166–4176.

(52) Barca, G. M.; Poole, D. L.; Vallejo, J. L. G.; Alkan, M.; Bertoni, C.; Rendell, A. P.; Gordon, M. S. Scaling the hartree-fock matrix build on summit. SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. 2020; pp 1–14.

(53) Bussy, A.; Schütt, O.; Hutter, J. Sparse tensor based nuclear gradients for periodic Hartree–Fock and low-scaling correlated wave function methods in the CP2K software package: A massively parallel and GPU accelerated implementation. *J. Chem. Phys.* **2023**, *158*.

(54) Qi, J.; Zhang, Y.; Yang, M. A hybrid CPU/GPU method for Hartree–Fock self-consistent-field calculation. *J. Chem. Phys.* **2023**, *159*.

(55) Stopper, D.; Roth, R. Massively parallel GPU-accelerated minimization of classical density functional theory. *J. Chem. Phys.* **2017**, *147*.

(56) Jia, W.; Cao, Z.; Wang, L.; Fu, J.; Chi, X.; Gao, W.; Wang, L.-W. The analysis of a plane wave pseudopotential density functional theory code on a GPU machine. *Comput. Phys. Comm.* **2013**, *184*, 9–18.

(57) Snowdon, C.; Barca, G. M. An Efficient RI-MP2 Algorithm for Distributed Many-GPU Architectures. *J. Chem. Theory Comput.* **2024**, *20*, 9394–9406.

(58) Hohenstein, E. G.; Luehr, N.; Ufimtsev, I. S.; Martínez, T. J. An atomic orbital-based formulation of the complete active space self-consistent field method on graphical processing units. *J. Chem. Phys.* **2015**, *142*.

(59) Snyder, J. W.; Hohenstein, E. G.; Luehr, N.; Martínez, T. J. An atomic orbital-based formulation of analytical gradients and nonadiabatic coupling vector elements for the state-averaged complete active space self-consistent field method on graphical processing units. *J. Chem. Phys.* **2015**, *143*.

(60) Hohenstein, E. G.; Bouduban, M. E.; Song, C.; Luehr, N.; Ufimtsev, I. S.; Martínez, T. J. Analytic first derivatives of floating occupation molecular orbital-complete active space configuration interaction on graphical processing units. *J. Chem. Phys.* **2015**, *143*.

(61) Menczer, A.; van Damme, M.; Rask, A.; Huntington, L.; Hammond, J.; Xantheas, S. S.; Ganahl, M.; Legeza, O. Parallel implementation of the Density Matrix Renormalization Group method achieving a quarter petaFLOPS performance on a single DGX-H100 GPU node. *J. Chem. Theory Comput.* **2024**, *20*, 8397–8404.

(62) Xiang, C.; Jia, W.; Fang, W.-H.; Li, Z. Distributed Multi-GPU Ab Initio Density Matrix Renormalization Group Algorithm with Applications to the P-Cluster of Nitrogenase. *J. Chem. Theory Comput.* **2024**, *20*, 775–786.

(63) Straatsma, T.; Broer, R.; Faraji, S.; Havenith, R.; Suarez, L.; Kathir, R.; Wibowo, M.; De Graaf, C. GronOR: Massively parallel and GPU-accelerated non-orthogonal configuration interaction for large molecular systems. *J. Chem. Phys.* **2020**, *152*, 064111.

(64) Huang, Y.; Guo, Z.; Pham, H. Q.; Lv, D. GPU-accelerated Auxiliary-field quantum Monte Carlo with multi-Slater determinant trial states. *arXiv preprint arXiv:2406.08314* **2024**,

(65) Gao, H.; Imamura, S.; Kasagi, A.; Yoshida, E. Distributed implementation of full configuration interaction for one trillion determinants. *J. Chem. Theory Comput.* **2024**, *20*, 1185–1192.

(66) Sun, Q. et al. Recent developments in the PySCF program package. *J. Chem. Phys.* **2020**, *153*, 024109.

(67) Mullinax, J. W.; Maradzike, E.; Koulias, L. N.; Mostafanejad, M.; Epifanovsky, E.; Gidofalvi, G.; DePrince III, A. E. Heterogeneous CPU+ GPU algorithm for variational

two-electron reduced-density matrix-driven complete active-space self-consistent field theory. *J. Chem. Theory Comput.* **2019**, *15*, 6164–6178.

(68) Legeza, Ö.; Menczer, A.; Ganyecz, Á.; Werner, M. A.; Kapás, K.; Hammond, J.; Xantheas, S. S.; Ganahl, M.; Neese, F. Orbital optimization of large active spaces via AI-accelerators. 2025; `https://arxiv.org/html/2503.20700v1`.

(69) Almlöf, J.; Fægri Jr, K.; Korsell, K. Principles for a direct SCF approach to LICAO–MOab-initio calculations. *J. Comput. Chem.* **1982**, *3*, 385–399.

(70) Herault, T.; Robert, Y.; Bosilca, G.; Harrison, R. J.; Lewis, C. A.; Valeev, E. F.; Dongarra, J. J. Distributed-memory multi-GPU block-sparse tensor contraction for electronic structure. 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 2021; pp 537–546.

(71) Seritan, S.; Bannwarth, C.; Fales, B. S.; Hohenstein, E. G.; Isborn, C. M.; Kokkila-Schumacher, S. I. L.; Li, X.; Liu, F.; Luehr, N.; Jr., J. W. S.; Song, C.; Titov, A. V.; Ufimtsev, I. S.; Wang, L.-P.; Martínez, T. J. TeraChem: A graphical processing unit-accelerated electronic structure package for large-scale ab initio molecular dynamics. *Wiley Interdisciplinary Reviews: Computational Molecular Science* **2021**, *11*, e1494.

(72) Koch, H.; Sánchez de Merás, A.; Pedersen, T. B. Reduced scaling in electronic structure calculations using Cholesky decompositions. *J. Chem. Phys.* **2003**, *118*, 9481–9484.

(73) 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.

(74) Laqua, H.; Thompson, T. H.; Kussmann, J.; Ochsenfeld, C. Highly efficient, linear-scaling seminumerical exact-exchange method for graphic processing units. *J. Chem. Theory Comput.* **2020**, *16*, 1456–1468.

(75) Werner, H.-J.; Knowles, P. J. A second order multiconfiguration SCF procedure with optimum convergence. *J. Chem. Phys.* **1985**, *82*, 5053–5063.

(76) Werner, H.-J.; Meyer, W. A quadratically convergent multiconfiguration–self-consistent field method with simultaneous optimization of orbitals and CI coefficients. *J. Chem. Phys.* **1980**, *73*, 2342–2356.

(77) Hermes, M. R. https://github.com/MatthewRHermes/mrh Commit ID: 49a5950. 2018; `https://github.com/MatthewRHermes/mrh`.

(78) pybind11 Authors pybind11. `https://github.com/pybind/pybind11`, Accessed: 2025-02-18.

(79) developers, N. S. Nsight Systems. `https://developer.nvidia.com/nsight-systems`, Accessed: 2025-02-18.

(80) Wu, X.; Sun, Q.; Pu, Z.; Zheng, T.; Ma, W.; Yan, W.; Yu, X.; Wu, Z.; Huo, M.; Li, X.; Ren, W.; Gong, S.; Zhang, Y.; Gao, W. Enhancing GPU-acceleration in the Python-based Simulations of Chemistry Framework. 2024; `https://arxiv.org/abs/2404.09452`.

(81) Authors, S. Syclomatic. `https://github.com/oneapi-src/SYCLomatic`, Accessed: 2025-02-18.

(82) HIPify. `https://github.com/ROCm/HIPIFY`, `https://github.com/ROCm/HIPIFY`, Accessed: 2025-02-18.

(83) Edwards, H. C.; Trott, C. R.; Sunderland, D. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *J. Parallel Distrib. Comput.* **2014**, *74*, 3202–3216, Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

(84) Trott, C.; Berger-Vergiat, L.; Poliakoff, D.; Rajamanickam, S.; Lebrun-Grandie, D.; Madsen, J.; Al Awar, N.; Gligoric, M.; Shipman, G.; Womeldorff, G. The Kokkos Ecosystem: Comprehensive Performance Portability for High Performance Computing. *Comput. Sci. Eng.* **2021**, *23*, 10–18.

(85) Trott, C. R. et al. Kokkos 3: Programming Model Extensions for the Exascale Era. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 805–817.

(86) Nath, R.; Tomov, S.; Dongarra, J. Accelerating GPU Kernels for Dense Linear Algebra. Proceedings of the 2009 International Meeting on High Performance Computing for Computational Science, VECPAR'10. Berkeley, CA, 2010.

(87) cuTensor Authors cuTensor. `https://developer.nvidia.com/cutensor`, Accessed: 2025-02-18.

(88) Homerding, B.; Lenard, B.; Blackworth, C.; Holohan, C.; Kulyavtsev, A.; McPheeters, G.; Pershy, E.; Rich, P.; Waldron, D.; Zhang, M.; others Polaris and Acceptance Testing. 2023.

(89) Kurashige, Y.; Chan, G. K.-L.; Yanai, T. Entangled quantum electronic wavefunctions of the Mn4CaO5 cluster in photosystem II. *Nat. Chem.* **2013**, *5*, 660–666.

(90) Yeh, B.; Chheda, S.; Prinslow, S. D.; Hoffman, A. S.; Hong, J.; Perez-Aguilar, J. E.; Bare, S. R.; Lu, C. C.; Gagliardi, L.; Bhan, A. Structure and site evolution of framework Ni species in MIL-127 MOFs for propylene oligomerization catalysis. *J. Am. Chem. Soc.* **2023**, *145*, 3408–3418.

(91) Dunning Jr, T. H. Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen. *J. Chem. Phys.* **1989**, *90*, 1007–1023.

(92) Czerny, F.; Searles, K.; Sot, P.; Teichert, J. F.; Menezes, P. W.; Coperet, C.; Driess, M.

Well-defined, silica-supported homobimetallic nickel hydride hydrogenation catalyst. *Inorg. Chem.* **2021**, *60*, 5483–5487.

(93) Agarawal, V. GPU4LASSCF Data. 10.5281/zenodo.15242737, 10.5281/zenodo.15242737, Accessed: 2025-04-17.

(94) Dawson, W.; Ozaki, K.; Domke, J.; Nakajima, T. Reducing numerical precision requirements in quantum chemistry calculations. *J. Chem. Theory Comput.* **2024**, *20*, 10826–10837.

(95) Pokhilko, P.; Epifanovsky, E.; Krylov, A. I. Double precision is not needed for many-body calculations: Emergent conventional wisdom. *J. Chem. Theory Comput.* **2018**, *14*, 4088–4096.

(96) Olivares-Amaya, R.; Watson, M. A.; Edgar, R. G.; Vogt, L.; Shao, Y.; Aspuru-Guzik, A. Accelerating correlated quantum chemistry calculations using graphical processing units and a mixed precision matrix multiplication library. *J. Chem. Theory Comput.* **2010**, *6*, 135–144.

(97) Knizia, G.; Li, W.; Simon, S.; Werner, H.-J. Determining the numerical stability of quantum chemistry algorithms. *J. Chem. Theory Comput.* **2011**, *7*, 2387–2398.

(98) DePrince III, A. E.; Hammond, J. R. Coupled cluster theory on graphics processing units I. The coupled cluster doubles method. *J. Chem. Theory Comput.* **2011**, *7*, 1287–1295.

(99) Hermes, M.; Bhavnesh, J.; Agarawal, V.; Gagliardi, L. Localized Active Space State Interaction Singles. 2025; https://chemrxiv.org/engage/chemrxiv/article-details/67cb271581d2151a028627d9.

(100) Agarawal, V.; King, D. S.; Hermes, M. R.; Gagliardi, L. Automatic State Interac-

tion with Large Localized Active Spaces for Multimetallic Systems. *J. Chem. Theory Comput.* **2024**, *20*, 4654–4662.