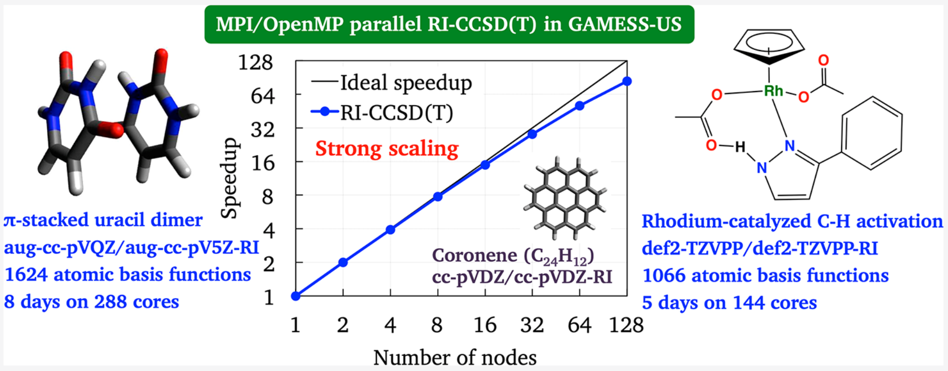
A Massively Parallel Implementation of the CCSD(T) Method Using the Resolution-of-the-Identity Approximation and a Hybrid Distributed/Shared Memory Parallelization Model

Dipayan Datta and Mark S. Gordon*

Cite This: https://doi.org/10.1021/acs.jctc.1c00389 Read Online



ABSTRACT: A parallel algorithm is described for the coupled-cluster singles and doubles method augmented with a perturbative correction for triple excitations [CCSD(T)] using the resolution-of-the-identity (RI) approximation for two-electron repulsion integrals (ERIs). The algorithm bypasses the storage of four-center ERIs by adopting an integral-direct strategy. The CCSD amplitude equations are given in a compact quasi-linear form by factorizing them in terms of amplitude-dressed three-center intermediates. A hybrid MPI/OpenMP parallelization scheme is employed, which uses the OpenMP-based shared memory model for intranode parallelization and the MPI-based distributed memory model for internode parallelization. Parallel efficiency has been optimized for all terms in the CCSD amplitude equations. Two different algorithms have been implemented for the rate-limiting terms in the CCSD amplitude equations that entail $\mathcal{O}(N_O^2N_V^4)$ and $\mathcal{O}(N_O^3N_V^3)$-scaling computational costs, where N_O and N_V denote the number of correlated occupied and virtual orbitals, respectively. One of the algorithms assembles the four-center ERIs requiring N_V^4 and $N_O^2N_V^2$-scaling memory costs in a distributed manner on a number of MPI ranks, while the other algorithm completely bypasses the assembling of quartic memory-scaling ERIs and thus largely reduces the memory demand. It is demonstrated that the former memory-expensive algorithm is faster on a few hundred cores, while the latter memory-economic algorithm shows a better strong scaling in the limit of a few thousand cores. The program is shown to exhibit a near-linear scaling, in particular for the compute-intensive triples correction step, on up to 8000 cores. The performance of the program is demonstrated via calculations involving molecules with 24−51 atoms and up to 1624 atomic basis functions. As the first application, the complete basis set (CBS) limit for the interaction energy of the π-stacked uracil dimer from the S66 data set has been investigated. This work reports the first calculation of the interaction energy at the CCSD(T)/aug-cc-pVQZ level without local orbital approximation. The CBS limit for the CCSD correlation contribution to the interaction energy was found to be −8.01 kcal/mol, which agrees very well with the value −7.99 kcal/mol reported by Schmitz, Hättig, and Tew [*Phys. Chem. Chem. Phys.* **2014**, *16*, 22167−22178]. The CBS limit for the total interaction energy was estimated to be −9.64 kcal/mol.

1. INTRODUCTION

The quest for accurate, reliable, and computationally affordable electronic structure methods is a primary research focus in chemistry and related fields. Although density functional theory (DFT) has long been a method of choice for the computational modeling of chemical problems of practical interest because of its favorable computational costs, the semiempirical nature of the DFT functionals has hindered their

Received: April 17, 2021

persistent success due to the lack of systematic improvability. Wave function based quantum-chemical methods, which aim at approximately solving the many-electron Schrödinger equation from first-principles, offer a powerful alternative to DFT. Coupled-cluster (CC) theory[1−3] has emerged as a broadly successful member of this family, especially in its CCSD(T)[4,5] variant with single and double excitations augmented with a noniterative correction for triple excitations. The pre-eminent success of CCSD(T) in achieving quantitative accuracy in the calculation of energies, geometric parameters, and spectroscopic properties of broad interest, in cases where the target electronic state can be adequately described by a single-determinant wave function, has established this method as a "gold standard" of quantum chemistry. Unfortunately, the steep $O(N^7)$ scaling of the computational cost of CCSD(T), where N is a measure of the system size, has confined its applicability to molecules consisting of only 15−20 atoms when using traditional sequential algorithms. Empowered by the advancements in massively parallel computer architectures, as well as by the recent developments in various cost-reducing approaches; however, the CC method has risen to prominence in computational chemistry during the past two decades.

Adapting a standard CCSD(T) implementation to modern parallel computer architectures is a viable strategy for expanding its applicability to large systems. Massively parallel CC implementations are available in several quantum chemistry programs, including NWChem,[6,7] GAMESS,[8] PQS,[9,10] ACESIII,[11] and Aquarius.[12] A parallel implementation of CCSD(T) not only speeds up the computation by distributing the steep-scaling floating point operations (FLOPs) over a number of parallel compute processes, but in addition makes use of the aggregate storage capacity of all nodes in a computer cluster for storing the requisite four-center two-electron repulsion integrals (ERIs), the N^4 scaling storage cost of which presents a memory bottleneck. The implementation in PQS employs distributed disk storage through the Array Files middleware,[13] while the implementations in NWChem and GAMESS employ the Global Arrays (GA) toolkit[14] and the Distributed Data Interface (DDI),[15,16] respectively, for storing large data arrays in the distributed random access memory (RAM) of a computer cluster. These parallel implementations have successfully expanded the applicability of the CCSD(T) method to calculations involving ∼1000−1500 atomic basis functions.[7,9,17] However, the use of the standard four-center ERIs imposes a performance limit to these parallel CCSD(T) codes. The implementations that employ a distributed memory model (GA or DDI) require a large number of compute nodes for storing the large four-center ERI matrix in order to make a large-scale CCSD(T) computation feasible. Since the network communication overhead associated with accessing data from the distributed memory increases rapidly with the number of nodes, the scalability of these codes is less than optimal when the number of nodes becomes very large.

The large memory demands and the communication bottlenecks of a standard CC program can be effectively reduced by employing approximate tensor factorization schemes.[18−24] The most widely used tensor factorization in the context of wave function based electron correlation methods concerns applying the density fitting (DF)/resolution-of-the-identity (RI) approximation[25−29] or alternatively the Cholesky decomposition (CD)[30−33] to the standard four-center ERI matrix. In both DF/RI and CD factorization schemes, the four-center ERIs are decomposed into products of three-index tensors, the memory requirement for which scales as N^3. Unlike the four-center ERIs, these three-index tensors can be stored in a replicated manner on the RAM of each compute node. Use of the aggregate disk storage or the distributed RAM of a large computer cluster then becomes unimportant. A suitable integral-direct algorithm can be designed that assembles the requisite four-center ERIs on the fly. Fully or partially integral-direct RI/CD-CC algorithms are becoming increasingly common,[34−41] as they offer an effective way to achieve improved parallel efficiency compared to standard CCSD(T) algorithms using four-center ERIs without compromising the accuracy. It is important to note that the DF/RI or the CD approximation does not reduce the computational scaling of the CC method, for example, the $O(N^6)$ scaling of CCSD. However, they offer a better factorization of the CCSD amplitude equations than do the standard four-center ERIs. This factorization can be utilized to arrive at a compact set of working equations, e.g., by introducing a t_1-transformed Hamiltonian,[37,41] thereby reducing the overall computational task.

Achieving true reduced scaling for the CC methods requires exploiting the short-range nature of dynamic electron correlation and employing localized orbitals to represent the ERIs as well as the cluster amplitudes. These localized orbitals substitute the traditionally used canonical Hartree−Fock (HF) orbitals that are spread over an entire molecule. A local CC method is thus capable of treating a large molecule as a whole at the expense of some accuracy (usually minor) and can in practice achieve linear scaling with respect to memory and computational costs. Following the pioneering work of Pulay and Saebø,[42−44] several local CC methods[45−55] have been put forth. The recently developed linear scaling domain-based local pair-natural orbital CC (DLPNO−CC) approach advocated by Neese and co-workers[56−58] has emerged as one of the most powerful local CC approaches because of its several extensions beyond ground state energy calculations. It is important to mention that the high efficiency of local CC approaches derives mainly from the compact description of the correlated wave function rather than from a massively parallel implementation, even though most local CC programs take advantage of parallel computer architectures.

While the present work is aimed at enabling applications to macromolecular systems consisting of a few thousand atoms described by a few hundred thousand basis functions, treating the entire system at the CC level of theory is not intended. Rather, the goal is to design a multilevel approach by combining the CCSD(T) method with the well-known fragment molecular orbital (FMO) approach.[59−64] Such a multilevel approach will facilitate the exploration of extended systems, while retaining the high accuracy of the CCSD(T) method. Li and Piecuch proposed an alternative multilevel approach,[65] which uses the local orbital framework of the cluster-in-molecule (CIM) scheme[51] and combines high-level methods, such as the completely renormalized CC [CR-CC(2,3)] theory[66,67] with the second-order Møller−Plesset perturbation theory (MP2) for treating the reactive and nonreactive parts of large molecular systems. One bottleneck of local orbital approaches like the CIM method is the need to localize the orbitals of the entire system. Findlater et al.[68] circumvented this problem by combining the CIM and FMO methods so that one only needs to localize the orbitals of smaller fragments, rather than those of the entire molecule.

An attractive feature of the FMO approach is that its design is ideally suited for taking advantage of massively parallel computers. A molecular system is divided into fragments, each of whose energy can be computed independently of the energies of the other fragments. The individual fragment calculations can then be distributed among separate groups of compute nodes. One has the option of treating all pairs of fragments (i.e., dimers) explicitly, resulting in the FMO2 method. Similarly, one can include trimers (FMO3) to capture explicit three-body effects. For a multilevel scheme based on the CCSD(T) method, maximum advantage of the coarse-grained parallelism of the FMO approach can be taken if a massively parallel CCSD(T) algorithm is designed, which not only distributes the computational task among the compute nodes in a group but also exploits the fine-grained parallelism that is feasible with modern multiprocessor CPUs. The goal of the present work is to develop such a massively parallel CCSD(T) algorithm for closed-shell molecules within the GAMESS (General Atomic and Molecular Electronic Structure System) program suite.[69,70] The DF/RI approximation for the ERIs is employed in the present implementation to gain parallel efficiency and reduce memory demands. However, the cluster amplitudes are treated in the conventional way without any tensor factorization. A combined FMO/RI-CCSD(T) method will be reported in the future.

The FMO implementation within GAMESS uses the Generalized Distributed Data Interface (GDDI)[71] parallelization model that partitions a number of nodes into groups, with each group processing one fragment (usually a monomer or a dimer). Within each group of nodes, one can implement fine-grained parallelism for a chosen electronic structure method, thereby enabling dual level parallelism. The GDDI model is based solely on the Message Passing Interface (MPI) library. A recent implementation of the MP2 method using the RI approximation combined with the FMO approach (FMO/RI-MP2)[72] has shown that the communication overhead between the intranode compute processes can be greatly reduced by adopting a shared-memory thread-based parallelization model using, e.g., OpenMP instead of MPI. The hybrid MPI/OpenMP model of FMO/RI-MP2 has been shown to achieve a 10× speedup in calculations involving medium to large water clusters compared to the pure MPI-driven GDDI model.[72] This hybrid MPI/OpenMP model is adopted also in the present work for the implementation of RI-CCSD(T).

The algorithmic considerations employed here are guided by two primary goals: to achieve good parallel scaling and to make large-scale applications of the RI-CCSD(T) method feasible even when computational resources are limited. An important consideration is to make the developed programs usable for the general computational chemistry community, while also taking advantage of the most powerful computers available. To accomplish these goals, a number of alternative algorithms have been implemented for the RI-CCSD part. In general, this work has focused on (i) reducing memory footprints at the expense of slightly increasing the number of operations, (ii) minimizing network communications and disk input/output (I/O) operations, and (iii) maintaining vectorization of the tensor contractions pertaining to the numerical evaluation of various terms. The reported implementations use a fully integral-direct algorithm and employ a set of amplitude-dressed three-index intermediates.[36,37,41] The (T) correction algorithm developed in this work completely bypasses the poorly scaling index permutation and vector addition operations following a recent suggestion by Nagy and Kállay.[41,54] Note that several alternative algorithms are possible for the implementation of RI-CCSD(T), each of which has its advantages and disadvantages. This work reports and tests only a few of them. Section 2 presents the new factorized RI-CCSD amplitude equations and the equations pertinent to the (T) correction. Section 3 describes the details of the proposed algorithm. Section 4 reports on the assessment of the parallel efficiency of the new code. Section 5 presents an accurate calculation of the basis set extrapolated interaction energy of the π-stacked uracil dimer, as well as a model application to study the energetics of a rhodium-catalyzed C−H bond activation reaction.

2. THEORETICAL BACKGROUND

The closed-shell coupled-cluster wave function is given by

$$|\Psi_{\mathrm{CC}}\rangle = \exp(\hat{T})|\Phi_0\rangle \tag{1}$$

where Φ_0 denotes a single-determinant reference state, usually a restricted HF (RHF) wave function, and the cluster operator $\hat{T}$ is defined as the sum of various n-electron excitation components given by

$$\hat{T}_n = \frac{1}{n!}\sum_{\substack{i,j,k,\cdots\\ a,b,c,\cdots}} t^{ijk\cdots}_{abc\cdots}\hat{E}^{abc\cdots}_{ijk\cdots} \tag{2}$$

In eq 2, $t^{ijk...}_{abc...}$ denote spatial orbital-based cluster amplitudes corresponding to excitations from the occupied orbitals $i,j,k,...$ in Φ_0 to the virtual or unoccupied orbitals $a,b,c,...$ and $\hat{E}^{abc...}_{ijk...}$ denote spin-summed second-quantized excitation operators.[73] The cluster amplitudes are determined via an iterative solution of the nonlinear CC amplitude equations, which are derived by substituting the *ansatz* of eq 1 into the electronic Schrödinger equation, followed by projecting the similarity transformed Hamiltonian operator $\exp(-\hat{T})\hat{H}\exp(\hat{T})$ onto various n-electron excited determinants. For the CCSD(T) method considered in the present work, one needs to solve the CCSD amplitude equations given by

$$R^{a}_{i} = \langle\Phi^{a}_{i}|e^{-(\hat{T}_1+\hat{T}_2)}\hat{H}e^{(\hat{T}_1+\hat{T}_2)}|\Phi_0\rangle = 0$$

$$R^{ab}_{ij} = \langle\Phi^{ab}_{ij}|e^{-(\hat{T}_1+\hat{T}_2)}\hat{H}e^{(\hat{T}_1+\hat{T}_2)}|\Phi_0\rangle = 0 \tag{3}$$

which determine the amplitudes t^{i}_{a} and t^{ij}_{ab}, respectively. In eq 3, $\{\Phi^{a}_{i}\}$ and $\{\Phi^{ab}_{ij}\}$ indicate singly and doubly excited determinants.

The residuals R^{a}_{i} and R^{ab}_{ij} are expressed in terms of the cluster amplitudes t^{i}_{a} and t^{ij}_{ab}, the Fock matrix elements f_{pq}, and the four-center ERIs $(pq|rs)$ with $p,q,r,...$, etc., denoting general (occupied or virtual) spatial orbital indices. The detailed expressions for the residuals are well-known in the literature.[2,73] A number of factorization schemes have been put forth for an efficient evaluation of the CCSD residuals by identifying common intermediates.[74−76]

In the DF/RI approximation,[25−29] the one-electron densities $\rho_{pq}(\mathbf{r}) = \phi_p(\mathbf{r})\phi_q(\mathbf{r})$ in the standard four-center ERIs

$$(pq|rs) = \int d\mathbf{r}_1\int d\mathbf{r}_2\phi_p(\mathbf{r}_1)\phi_q(\mathbf{r}_1)\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|}\phi_r(\mathbf{r}_2)\phi_s(\mathbf{r}_2) \tag{4}$$

are substituted with fitted densities $\bar{\rho}_{pq}(\mathbf{r})$ that are expanded in a preoptimized auxiliary basis set $\{\chi_P(\mathbf{r})\}$ as

$$\bar{\rho}_{pq}(\mathbf{r}) = \sum_{P}^{N_{\text{aux}}} C_{pq}^{P}\chi_P(\mathbf{r}) \tag{5}$$

with N_{aux} denoting the dimension of the auxiliary basis set. The fitting or expansion coefficients C_{pq}^{P} are in turn determined by minimizing the error between the actual and the fitted densities with a weight factor of $1/|\mathbf{r}_1 - \mathbf{r}_2|$. This leads to

$$C_{pq}^{P} = \sum_{Q} (pq|Q)[\mathbf{J}^{-1}]_{QP} \tag{6}$$

where the sum goes over the auxiliary basis functions. The elements of the Coulomb metric matrix $\mathbf{J}$ are defined as

$$J_{PQ} = \int d\mathbf{r}_1 \int d\mathbf{r}_2 \chi_P(\mathbf{r}_1)\frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}\chi_Q(\mathbf{r}_2) \tag{7}$$

and the three-center ERIs $(pq|P)$ as

$$(pq|P) = \int d\mathbf{r}_1 \int d\mathbf{r}_2 \phi_p(\mathbf{r}_1)\phi_q(\mathbf{r}_1)\frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}\chi_P(\mathbf{r}_2) \tag{8}$$

Substituting eqs 5 and 6 into eq 4 and using eq 8 one obtains

$$(pq|rs) = \sum_{P,Q} (pq|P)[\mathbf{J}^{-1}]_{PQ}(Q|rs) \tag{9}$$

The four-center ERIs $(pq|rs)$ can be expressed as products of three-center two-electron integrals, henceforth referred to as "RI integrals", as

$$(pq|rs) = \sum_{P}^{N_{\text{aux}}} B_{pq}^{P}B_{rs}^{P} \tag{10}$$

by factorizing the Coulomb metric matrix $\mathbf{J}$ via the Cholesky decomposition $\mathbf{J} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is a lower triangular matrix, and by defining the RI integrals as

$$B_{pq}^{P} = \sum_{Q} (pq|Q)[\mathbf{L}^{-1}]_{QP} \tag{11}$$

The accuracy of the RI approximation given by eq 10 depends on the quality of the auxiliary basis set and its dimension N_{aux}. The commonly used auxiliary basis sets have been optimized for individual atoms, atomic orbital (AO) bases, and for different levels of theory. Although auxiliary bases optimized for CC calculations are not available, it has been shown that the auxiliary bases optimized for MP2 calculations can be used also in CC calculations.[37] These basis sets, which are indicated with the "RI" extension to the AO basis name,[77−79] have been employed in this work.

A straightforward strategy to implement the CCSD amplitude residuals given by eq 3 using the RI integrals would be to preassemble and store the four-center ERIs and substitute them in standard CCSD equations. This would allow one to reuse an already available implementation. However, the full advantage of the memory savings and an improved parallel efficiency offered by the RI integrals can only be exploited if the four-center ERIs are assembled on the fly. Note that the computational cost associated with the integral assembling according to eq 10 scales as $O(N^5)$, which is an additional cost in comparison to a CCSD implementation using standard four-center ERIs. Hence, care should be exercised to minimize the number of integral assembling steps. This can be effectively done by devising a new factorization scheme for the CCSD amplitude residuals that groups together the terms in eq 3, which require a given class of four-center ERIs, for example, $(ij|kl)$, $(ij|ab)$, $(ia|jb)$, ..., etc.; that is, (OO|OO), (OO|VV), (OV|OV), ..., where O and V stand for occupied and virtual orbitals, respectively. Further efficiency in the numerical evaluation of the CCSD residuals can be achieved by defining a set of amplitude-dressed RI integrals, the storage costs of which are identical with those of the bare RI integrals. The following dressed RI integrals and intermediates are used in this work

$$\tilde{t}_{ab}^{ij} = 2t_{ab}^{ij} - t_{ba}^{ij} \tag{12}$$

$$\mathcal{M}_0^P = \sum_{m,e} 2B_{me}^P t_e^m \tag{13}$$

$$\mathcal{M}_{ij}^{P,T_1} = \sum_{e} B_{je}^P t_e^i \tag{14}$$

$$\mathcal{M}_{ia}^{P,T_1} = \sum_{e} B_{ae}^P t_e^i \tag{15}$$

$$\mathcal{M}_{ia}^{P,T_2} = \sum_{m,e} B_{me}^P \tilde{t}_{ae}^{im} \tag{16}$$

$$\mathcal{W}_{ij}^P = B_{ij}^P + \mathcal{M}_{ij}^{P,T_1} \tag{17}$$

$$\mathcal{W}_{ia}^P = B_{ia}^P + \mathcal{M}_{ia}^{P,T_1} + \mathcal{M}_{ia}^{P,T_2} - \sum_{m} \mathcal{W}_{im}^P t_a^m \tag{18}$$

$$\mathcal{W}_{ab}^P = B_{ab}^P - \sum_{m} B_{mb}^P t_a^m \tag{19}$$

The use of t_1-dressed intermediates was recently proposed by DePrince and Sherrill[37] and was subsequently adopted by Gyevi-Nagy et al.[41] Epifanovsky et al.[36] also employed similar intermediates with an additional t_2-dressed intermediate as defined in eq 16. The use of these intermediates makes the CCSD amplitude residuals highly compact by folding onto themselves a large number of terms in the expression of R_{ij}^{ab}, which correspond to couplings between t_1 and t_2 amplitudes. One specific advantage of using these intermediates is that the (VV|VO)-type ERIs or intermediates labeled with three virtual and one occupied indices do not appear in the RI-CCSD residuals. These ERIs present a memory bottleneck in standard CCSD implementations employing four-index ERIs. Note that the proposed algorithm uses only molecular orbital (MO)-based RI integrals and intermediates and that the AO-to-MO transformation of the RI integrals is performed only once prior to the beginning of the RI-CC step. The RI-CCSD amplitude residuals used in this implementation are given in terms of the intermediates defined in eqs 12−19 as

$$\begin{aligned}
R_{ij}^{ab} = {} & \textstyle\sum_P \mathcal{W}_{ia}^P \mathcal{W}_{jb}^P && (ia)\otimes(jb)\ \text{group} \\
& +\textstyle\sum_{e,f}\sum_P \mathcal{W}_{ae}^P \mathcal{W}_{bf}^P t_{ef}^{ij} && \text{PPL group} \\
& +\textstyle\sum_{m,n}\left(I_{mn}^{ij} + \sum_P \mathcal{W}_{im}^P \mathcal{W}_{jn}^P\right)t_{ab}^{mn} && \text{HHL group} \\
& +\hat{\mathcal{P}}_{ij}^{ab}\Big[-\textstyle\sum_m(\mathcal{F}_{im} + \sum_e \mathcal{F}_{me}t_e^i)t_{ab}^{mj} && \text{Fock matrix terms} \\
& +\textstyle\sum_e(\mathcal{F}_{ae} - \sum_m \mathcal{F}_{me}t_a^m)t_{eb}^{ij} && \\
& -\textstyle\sum_{m,e}\left(I_{bm}^{je} - I_{mb}^{je} + \sum_P \mathcal{W}_{jm}^P \mathcal{W}_{be}^P\right)t_{ae}^{im} && O(N_O^3N_V^3)\ \text{group} \\
& -\textstyle\sum_{m,e}\left(-\frac{1}{2}I_{ma}^{je} + \sum_P \mathcal{W}_{jm}^P \mathcal{W}_{ae}^P\right)t_{eb}^{im}\Big] &&
\end{aligned} \tag{20}$$

$$R_i^a = f_{ia} - \sum_m \left(\mathcal{F}_{im} + \frac{1}{2}\sum_e \mathcal{F}_{me} t_e^i\right) t_a^m + \sum_e \left(\mathcal{F}_{ae} - \frac{1}{2}\sum_m \mathcal{F}_{me} t_a^m\right) t_e^i + \sum_{m,e} \mathcal{F}_{me} \tilde{t}_{ae}^{im} + \sum_P B_{ia}^P \mathcal{M}_0^P - \sum_m \sum_P B_{im}^P (\mathcal{M}_{ma}^{P,T_1} + \mathcal{M}_{ma}^{P,T_2}) + \sum_e \sum_P B_{ae}^P \mathcal{M}_{ie}^{P,T_2} \quad (21)$$

and the CCSD energy is given by

$$E_{\mathrm{CCSD}} = \sum_{i,a} 2 f_{ia} t_a^i + \sum_{\substack{i,j,\\a,b}} \sum_P (2B_{ia}^P B_{jb}^P - B_{ib}^P B_{ja}^P)(t_{ab}^{ij} + t_a^i t_b^j) \quad (22)$$

The terms in eq 20 have been labeled as "groups" since each of them includes multiple terms through the $\mathcal{W}_{pq}^P$ intermediates. The construction of the first term in eq 20 is analogous to assembling the $(ia|jb)$-type ERIs from the corresponding RI integrals. These terms have been labeled as the $(ia) \otimes (ib)$ group because of this analogy. The second and the third terms in eq 20 have been labeled according to the classes of the four-center intermediates involved in them, namely, the "particle−particle ladder" (PPL) group involves (VV|VV)-type intermediates and the "hole−hole ladder" (HHL) group involves the (OO|OO)-type intermediates. The last two terms in eq 20 have been labeled as the "$\mathcal{O}(N_O^3 N_V^3)$ group" according to the scaling of the computational cost associated with the tensor contractions involved in their evaluation.

The dressed Fock matrices appearing in eqs 20 and 21 are defined as

$$\mathcal{F}_{ia} = f_{ia} + \sum_P B_{ia}^P \mathcal{M}_0^P - \sum_m \sum_P B_{ma}^P \mathcal{M}_{mi}^{P,T_1} \quad (23)$$

$$\mathcal{F}_{ij} = (1 - \delta_{ij}) f_{ij} + \sum_P B_{ij}^P \mathcal{M}_0^P - \sum_m \sum_P B_{mi}^P \mathcal{M}_{mj}^{P,T_1} + \sum_e \sum_P B_{ie}^P \mathcal{M}_{je}^{P,T_2} \quad (24)$$

$$\mathcal{F}_{ab} = (1 - \delta_{ab}) f_{ab} + \sum_P B_{ab}^P \mathcal{M}_0^P - \sum_m \sum_P B_{ma}^P (\mathcal{M}_{mb}^{P,T_1} + \mathcal{M}_{mb}^{P,T_2}) \quad (25)$$

and the four-index I intermediates are defined as

$$I_{mn}^{ij} = \sum_{e,f} \sum_P B_{me}^P B_{nf}^P t_{ef}^{ij} \quad (26)$$

$$I_{bm}^{je} = \sum_{n,f} \sum_P B_{ne}^P B_{mf}^P t_{bf}^{jn} \quad (27)$$

$$I_{mb}^{je} = \sum_{n,f} \sum_P B_{ne}^P B_{mf}^P t_{fb}^{jn} \quad (28)$$

The permutation operator appearing in eq 20 is defined as

$$\hat{\mathcal{P}}_{ij}^{ab} X_{ij}^{ab} = X_{ij}^{ab} + X_{ji}^{ba} \quad (29)$$

where X_{ij}^{ab} is a generic notation for any two-body contribution to R_{ij}^{ab}.

In this implementation, the evaluation of the noniterative triples corrections to the CCSD energy follows the widely used the "*ijkabc*" algorithm proposed by Rendell et al.[80,81] For an elaborate discussion on alternative algorithms for the (T) correction, the reader is referred to ref 81. The algorithm in this work utilizes the 6-fold permutational symmetry of the spatial orbital-based triples amplitudes and the associated intermediates

$$X_{abc}^{ijk} = X_{bac}^{jik} = X_{acb}^{ikj} = X_{bca}^{jki} = X_{cab}^{kij} = X_{cba}^{kji} \quad (30)$$

by constructing all six-index tensors within restricted loops over occupied indices $i \geq j \geq k$. This allows one to store them as X^{ijk} (abc), which reduces the storage cost for a six-index tensor from $8N_O^3 N_V^3$ to $8N_V^3$ bytes (where N_O and N_V denote the number of correlated occupied and virtual orbitals, respectively). Throughout the paper, the tensor notation $A^{pqr\cdots}(stu\ldots)$ is used to indicate that the superscript indices p, q, r, ... are fixed inside a loop, while the tensor is stored in terms of the full set of indices s, t, u, ... shown within the parentheses.

The second-order triples amplitudes $t^{[2]}{}_{abc}^{ijk}$ are defined via

$$W^{ijk}(abc) \equiv D_{abc}^{ijk} t^{[2]}{}_{abc}^{ijk} = \hat{\mathcal{P}}_{ijk}^{abc}\left(-\sum_m \sum_P B_{ia}^P B_{jm}^P t_{bc}^{mk} + \sum_e \sum_P B_{ia}^P B_{be}^P t_{ec}^{jk}\right) \quad (31)$$

where the denominator D_{abc}^{ijk} is given by

$$D_{abc}^{ijk} = f_{ii} + f_{jj} + f_{kk} - f_{aa} - f_{bb} - f_{cc} \quad (32)$$

and the permutation operator $\hat{\mathcal{P}}_{ijk}^{abc}$ generates the sum of all six permutations given in eq 30 by its action on any one of them. Within the *ijkabc* algorithm, the (T) correction to the energy is given by[80]

$$\begin{aligned} E_{(T)} = \sum_{i\geq j\geq k}{}' \sum_{a\geq b\geq c} & 2[(Y^{ijk}(abc) - 2Z^{ijk}(abc)) \\ & \times (W^{ijk}(abc) + W^{ijk}(bca) + W^{ijk}(cab)) \\ & + (Z^{ijk}(abc) - 2Y^{ijk}(abc)) \times (W^{ijk}(bac) \\ & + W^{ijk}(acb) + W^{ijk}(cba)) + 3X^{ijk}(abc)] \\ & /[D_{abc}^{ijk}(1 + \delta_{ij} + \delta_{jk})(1 + \delta_{ab} + \delta_{bc})] \end{aligned} \quad (33)$$

In eq 33, the summation over occupied indices excludes the case $i = j = k$, which is indicated by the notation Σ'. The various intermediates are defined as

$$\begin{aligned} X^{ijk}(abc) = & W^{ijk}(abc)V^{ijk}(abc) + W^{ijk}(bca)V^{ijk}(bca) \\ & + W^{ijk}(cab)V^{ijk}(cab) + W^{ijk}(bac)V^{ijk}(bac) \\ & + W^{ijk}(acb)V^{ijk}(acb) + W^{ijk}(cba)V^{ijk}(cba) \end{aligned} \quad (34)$$

$$Y^{ijk}(abc) = V^{ijk}(abc) + V^{ijk}(bca) + V^{ijk}(cab) \quad (35)$$

$$Z^{ijk}(abc) = V^{ijk}(bac) + V^{ijk}(acb) + V^{ijk}(cba) \quad (36)$$

where the intermediate V is given by

$$V^{ijk}(abc) = W^{ijk}(abc) + B^P_{ia}B^P_{jb}t^k_c + B^P_{ia}B^P_{kc}t^j_b + B^P_{jb}B^P_{kc}t^i_a \tag{37}$$

It is important to note that a restricted inner loop over virtual indices a, b, c is used when computing the final energy contribution[80] via eq 33, even though the various six-index intermediates defined above are stored for the full set of virtual indices. The use of restricted loops $a \geq b \geq c$ reduces the computational task associated with the evaluation of $E_{(T)}$.

3. ALGORITHM

The detailed algorithms for the individual terms in the RI-CCSD residuals and for the (T) correction are presented in sections 3.1 and 3.2, while certain general features are outlined below. All of the new features have been implemented into the GAMESS program suite.[69,70]

In order to reduce storage costs for the t_2 amplitudes and the R^{ab}_{ij} residuals, the permutational symmetry of these spatial orbital-based tensors, e.g., $t^{ij}_{ab} = t^{ji}_{ba}$ is exploited by storing them with the index restriction $i \geq j$. This not only reduces the memory cost from $16N_V^2N_O^2$ to $8N_V^2N_O(N_O + 1)$ bytes, but also reduces the hard disk storage for the t_2 amplitudes. The only disk I/O operations in the new implementation involve reading and writing the amplitude vectors for a number of iterations, which is essential for ensuring fast convergence of the CCSD iterations via the direct inversion of the iterative subspace (DIIS) procedure. The t_2 amplitudes and the R^{ab}_{ij} residuals are the only four-index quantities that are stored in a replicated manner on each MPI rank. Therefore, the applicability limit of the code in terms of the system size is mostly determined by the size of the corresponding doubles amplitude ($\mathbf{T}_2$) and the residual matrices ($\mathbf{R}_2$). The evaluation of the various terms in the CCSD residuals requires a repeated symmetry unfolding of these two matrices. However, these are lower operation-count tasks compared to tensor contractions and the additional expenses are expected to be compensated for via OpenMP parallelization.

As noted above, the RI integrals and the amplitude-dressed intermediates defined in eqs 17−19 entail a cubic-scaling memory cost. The options of storing the bare RI integrals either in a replicated manner on each MPI rank or in the distributed memory of all MPI ranks using the DDI framework[15,16] of GAMESS have been implemented. The intermediates of eqs 17−19 are always stored in the replicated memory, as they need to be iteratively updated. A distributed storage of the four-center ERIs was an important algorithmic strategy for the massively parallel CCSD(T) implementation by Olson et al.,[8] while this is optional for the current RI-CCSD(T) code. Although the cubic-scaling RI integrals do not generally present a memory bottleneck, storing them in the distributed memory might be helpful when the available memory per node is small. This distributed storage entails an additional network communication cost associated with fetching the RI integrals repeatedly.

To reduce communication among the MPI ranks, the intermediates of eqs 12−19, the singles residual given in eq 21, and the dressed Fock matrices defined in eqs 23−25 are all constructed in a replicated manner on every MPI rank. The storage costs for the associated quantities are at most cubic scaling and they are constructed in OpenMP parallel batches with each batch requiring computational costs scaling as $O(N^3)$. Only the contributions to the doubles residual R^{ab}_{ij} given in eq 20 are constructed in a distributed manner, which requires a global reduction of the blocks of the $\mathbf{R}_2$ matrix constructed on various MPI ranks. In the GAMESS implementation, this reduction operation is performed only once per CCSD iteration after all contributions in eq 20 have been added to R^{ab}_{ij}. The remaining communication-bound steps include the global reduction of the RI-MP2 guess t_2 amplitudes prior to CCSD iterations, and the broadcasting of the updated amplitudes from the master MPI rank to all others in every iteration. The disk I/O bound DIIS extrapolation is performed in serial only on the master, which necessitates the broadcasting step.

3.1. Integral-Direct Parallel Implementation of the CCSD Amplitude Equations.

It is well-known that the particle−particle ladder (PPL) term in the CCSD doubles residual is the most expensive contribution.[6−8] This term presents both a memory bottleneck by requiring the $(ae|bf)$ class of ERIs [i.e., the (VV|VV) ERIs] and is also the time determining step in a CCSD iteration. The common approach to reduce the high memory demand is to employ four-center ERIs partially labeled with AO indices.[6,8,9,45] The computational time can be effectively reduced by employing symmetric and antisymmetric combinations of the t_2 amplitudes and the ERIs,[6,73,37,41] which reduces the number of FLOPs from $N_O^2N_V^4$ to $\sim N_O^2N_V^4/4$ by invoking the index restrictions $a \geq b$, $e \geq f$, $i \geq j$. This strategy is adopted in the new implementation. However, an AO-driven algorithm is not implemented since the PPL group of eq 20 is constructed from the dressed $\mathcal{W}^P_{ab}$ intermediates in lieu of bare RI integrals. These intermediates subsume contributions from terms that require the (VV|VO)-type ERIs (see, e.g., ref 40.). The current algorithm avoids an explicit assembling of these ERIs in the RI-CCSD step as they present additional memory bottlenecks.

Within the RI approximation, the PPL group is constructed via[37,41]

$$\mathcal{V}^{ef}_{a\geq b} = \sum_P \mathcal{W}^P_{ae}\mathcal{W}^P_{bf} \tag{38}$$

$$\mathcal{V}^{\pm e\geq f}_{a\geq b} = \mathcal{V}^{ef}_{a\geq b} \pm \mathcal{V}^{fe}_{a\geq b} \tag{39}$$

$$t^{+\,i\geq j}_{e\geq f} = t^{i\geq j}_{ef} + t^{i\geq j}_{fe}(1 - \delta_{ef});\; t^{-\,i\geq j}_{e\geq f} = t^{i\geq j}_{ef} - t^{i\geq j}_{fe} \tag{40}$$

$$\sigma^{\pm\,a\geq b}_{i\geq j} = \frac{1}{2}\sum_{e\geq f} \mathcal{V}^{\pm e\geq f}_{a\geq b} t^{\pm\,i\geq j}_{e\geq f} \tag{41}$$

$$R^{ab}_{i\geq j} = \sigma^{+\,a\geq b}_{i\geq j} + \sigma^{-\,a\geq b}_{i\geq j} \tag{42}$$

$$R^{ba}_{i\geq j} = \sigma^{+\,a\geq b}_{i\geq j} - \sigma^{-\,a\geq b}_{i\geq j} \text{ if } b \neq a \tag{43}$$

where the quantities $\mathcal{V}^{\pm}$ and $t^{\pm}$ indicate symmetric and antisymmetric combinations of the ERIs and the t_2 amplitudes, respectively, and $\sigma^{\pm}$ denote the intermediate contributions to the doubles residuals. The index restrictions applicable to each tensor are indicated explicitly in eqs 38-43.

The integral-direct algorithm described here requires integral assembling according to eq 38. In principle, a single integral assembling step should suffice to construct both $\mathcal{V}^{\pm}$ contributions of eq 39, and this would in turn reduce the integral assembling cost by a factor of 2 compared to the naïve algorithm that employs unrestricted loops over a and b. However, each of the $\mathcal{V}^{\pm}$ matrices entails a storage cost scaling as $N_V^2\,(N_V + 1)^2/4$. Even if these matrices are constructed in blocks over a number of MPI ranks as described below, the memory requirement per MPI rank for storing both $\mathcal{V}^{\pm}$ matrices simultaneously would be very high, and this would become a limiting factor for the system size that can be handled with the code. To keep the memory demand moderate, the $\mathcal{V}^{\pm}$ matrices are constructed one at a time. In effect, the algorithm constructs all symmetric contributions in eqs 39−41 first, digests them via eqs 42 and 43, and then constructs all antisymmetric combinations in the same memory space. In addition to memory savings for the $\mathcal{V}^{\pm}$ matrices, this strategy allows one to save quartic-scaling memory associated with two sets of the $t^{\pm}$ and $\sigma^{\pm}$ matrices as well. The downside is that the algorithm does not take advantage of the aforementioned 2-fold savings in the number of FLOPs associated with integral assembling. However, there is no need to compromise on the reduced number of FLOPs, $N_O^2N_V^4/4$, associated with the main contraction step of eq 41. Eq 41 entails two different contractions, which need to be evaluated separately anyway.

Scheme 1 depicts the hybrid MPI/OpenMP parallel algorithm for the PPL group, where the two contributions

Scheme 1. Algorithm A for the Evaluation of the PPL Group in Eq 20

```
1  do $ab = 1, N_V(N_V+1)/2$ // MPI parallel
2    MPI blocks: $(ab_{start}^{\mathrm{MyMPIrank}}, ab_{end}^{\mathrm{MyMPIrank}})$ // OpenMP parallel
3        do $ab^{\mathrm{MyThreadID}} = ab_{start}^{\mathrm{MyThreadID}}, ab_{end}^{\mathrm{MyThreadID}}$
4          $\mathcal{V}^{a\geq b}(e,f) = \sum_P \mathcal{W}^a(e,P)\mathcal{W}^b(f,P)$
5          $\mathcal{V}^{\pm}{}_{a\geq b}^{e\geq f} = \mathcal{V}^{a\geq b}(e,f) \pm \mathcal{V}^{a\geq b}(f,e)$
6        end do
7        $\sigma^{\pm}{}_{i\geq j}^{a\geq b} = \frac{1}{2}\sum_{e\geq f}\mathcal{V}^{\pm}{}_{a\geq b}^{e\geq f}t^{\pm}{}_{e\geq f}^{i\geq j}$
8        do $ab^{\mathrm{MyThreadID}} = ab_{start}^{\mathrm{MyThreadID}}, ab_{end}^{\mathrm{MyThreadID}}$
9          $R^{ab}(i\geq j) \mathrel{+}= \sigma^{+}{}_{i\geq j}^{a\geq b} + \sigma^{-}{}_{i\geq j}^{a\geq b}$
10         $R^{ba}(i\geq j) \mathrel{+}= \sigma^{+}{}_{i\geq j}^{a\geq b} - \sigma^{-}{}_{i\geq j}^{a\geq b}$ if $b \neq a$
11       end do
12   end OpenMP parallel
13 end do // end MPI parallel
```

involving the $\mathcal{V}^{\pm}$, $t^{\pm}$, and $\sigma^{\pm}$ matrices have been combined for brevity. This algorithm distributes the restricted outer loops over a and b among MPI ranks. The four-index intermediates of eq 39 are assembled in blocks on various MPI ranks, which in turn permits scaling down the memory cost for storing (one of) the $\mathcal{V}^{\pm}$ matrices as $N_V^2(N_V + 1)^2/4n_{\mathrm{MPI}}$, where n_{MPI} denotes the number of MPI ranks. On each MPI rank, the computational tasks are further parallelized via OpenMP. As shown in Scheme 1, each OpenMP thread assembles subblocks of the $\mathcal{V}^{\pm}$ matrices (line 4) at a computational cost scaling as $O(N_V^2N_{\mathrm{aux}})$. The tensor contractions of eq 41 are performed via matrix−matrix multiplications using the basic linear algebra subprogram (BLAS) DGEMM, which contributes to the efficiency of this algorithm.

Despite the high efficiency of the above algorithm, the $N_V^2(N_V + 1)^2/4$ scaling memory requirement associated with the storage of the $\mathcal{V}^{\pm}$ matrices may become overwhelming in large-scale applications. As discussed above, the applicability of this algorithm depends on the number of MPI ranks that can be assigned to a calculation. For calculations involving $N_V \geq$ 1000, a large number of MPI ranks will be needed to effectively scale down the high memory demand. On a typical small computer cluster that does not have a very large number of compute nodes or has a small to moderate amount of main memory (RAM) per node (e.g., up to 125 GB), the use of the above algorithm will impose a limit to large-scale applications.

Since an important focus of the current RI-CCSD(T) development is to make the program usable for the broad computational chemistry community, a second algorithm for constructing the PPL group has been implemented, which is shown in Scheme 2. In the remainder of this Article, the

Scheme 2. Algorithm B for the Evaluation of the PPL Group in Eq 20

```
1  do $ab = 1, N_V(N_V+1)/2$ // MPI parallel
2    MPI blocks: $(ab_{start}^{\mathrm{MyMPIrank}}, ab_{end}^{\mathrm{MyMPIrank}})$ // OpenMP parallel
3        do $ab^{\mathrm{MyThreadID}} = ab_{start}^{\mathrm{MyThreadID}}, ab_{end}^{\mathrm{MyThreadID}}$
4          $\mathcal{V}^{a\geq b}(e,f) = \sum_P \mathcal{W}^a(e,P)\mathcal{W}^b(f,P)$
5          $\mathcal{V}^{\pm a\geq b}(e\geq f) = \mathcal{V}^{a\geq b}(e,f) \pm \mathcal{V}^{a\geq b}(f,e)$
6          $\sigma^{\pm a\geq b}(i\geq j) = \frac{1}{2}\sum_{e,f}\mathcal{V}^{\pm a\geq b}(e\geq f)t^{\pm}{}_{e\geq f}^{i\geq j}$
7          $R^{ab}(i\geq j) \mathrel{+}= \sigma^{+a\geq b}(i\geq j) + \sigma^{-a\geq b}(i\geq j)$
8          $R^{ba}(i\geq j) \mathrel{+}= \sigma^{+a\geq b}(i\geq j) - \sigma^{-a\geq b}(i\geq j)$ if $b \neq a$
9        end do
10   end OpenMP parallel
11 end do // end MPI parallel
```

algorithm outlined in Scheme 1 is referred to as algorithm A and the one outlined in Scheme 2 is called algorithm B. Algorithm B uses an identical loop structure and computational work distribution among MPI ranks and OpenMP threads as algorithm A. The primary difference lies in assembling the intermediates of eq 39 and performing the contractions for a *fixed* pair of outer indices a and b. In other words, each OpenMP thread assembles single columns of the $\mathcal{V}^{\pm}$ matrices, contracts them with $t^{\pm}$, and finally updates a single row of the residual matrix $\mathbf{R}_2$ at a time. The contractions of eq 41 are thus evaluated via matrix-vector multiplication using the level-2 BLAS procedure DGEMV. This algorithm entails only $N_V(N_V + 1)/2$ scaling memory cost for storing single columns of each of the $\mathcal{V}^{\pm}$ matrices and is thus much more memory-economic than algorithm A. The reduced memory demand is utilized for storing both of the $\mathcal{V}^{\pm}$ matrices simultaneously unlike in algorithm A. The reduced vectorization of the contraction steps makes this algorithm intrinsically slower than algorithm A. However, algorithm B should scale as well as algorithm A in the parallel sense, as the distribution of the computational task is similar in both cases. Section 4 compares the parallel scaling of these two algorithms and also their relative memory requirements in a fairly large application.

Apart from the PPL group, the last two terms in eq 20 also present computational bottlenecks in calculations involving large systems. The PPL group dominates the computational time for CCSD iterations when large AO basis sets are employed for small to moderate-sized molecules. However, as the number of correlated electrons (hence also N_O) increases, the $O(N_O^3N_V^3)$ scaling terms become nearly as expensive as the PPL group. Furthermore, these terms entail four-index intermediates as defined in eqs 27 and 28. Even within the

RI approximation, these intermediates cannot be reduced to three-index quantities[36,37,41] similar to the ones used for the PPL group. Therefore, a careful optimization of the algorithm is needed for an efficient parallel implementation of these terms that bypasses the actual storage of four-index intermediates or assembles them in a distributed manner.

Scheme 3. Algorithm A for the Evaluation of the $O(N_O^3N_V^3)$ Group in Eq 20

```
1 do $em = 1, N_O N_V$ // MPI parallel
2   do $em^{\text{MyMPIrank}} = em^{\text{MyMPIrank}}_{start}, em^{\text{MyMPIrank}}_{end}$ // OpenMP parallel
3     $I_{jb,me} = \sum_P \mathcal{W}^m(j,P)\mathcal{W}^e(b,P)$
4     $I_{ja,me} = I_{jb,me}$
5     $\mathcal{V}_{me,nf} = \sum_P B^e(n,P)B^m(f,P)$
6   end do
7   do $j = 1, N_O$ // OpenMP parallel
8     $\tilde{t}^j(nf,b) = t^{jn}_{bf} - t^{jn}_{fb}$
9     $I_{jb,me} += \sum_{n,f} \mathcal{V}_{me,nf}\tilde{t}^j(nf,b)$
10    $I_{ja,me} += -\frac{1}{2}\sum_{n,f} \mathcal{V}_{me,nf}t^j(nf,a)$
11  end do
12  do $i = 1, N_O$ // OpenMP parallel
13    $\mathcal{W}^i(jb,a) = -\sum_{m,e} I_{jb,me}t^i(me,a)$
14    $\mathcal{W}^i(ja,b) = -\sum_{m,e} I_{ja,me}t^i(me,b)$
15    $R^{ab}_{i\geq j} += \mathcal{W}^i(jb,a); R^{ba}_{i\geq j} += \mathcal{W}^i(ja,b)$ if $i \geq j$
16    $R^{ba}_{i\geq j} += \mathcal{W}^i(jb,a); R^{ab}_{i\geq j} += \mathcal{W}^i(ja,b)$ if $i \leq j$
17  end do
18 end do // end MPI parallel
```

Scheme 3 represents the algorithm for the $O(N_O^3N_V^3)$ group. In this algorithm, the four-index intermediates given in eqs 27 and 28 are constructed in blocks by splitting the loops over the contracted index pair (m, e) among a number of MPI ranks, n_{MPI}. Hence, the memory required for storing them scales as $N_O^2N_V^2/n_{\text{MPI}}$. The computational costs associated with the assembling of integrals (or the $\mathcal{W}$ intermediates) shown in lines 3,5 of Scheme 3 scale as $O(N_ON_VN_{\text{aux}})$. Importantly, the I intermediates are directly constructed with the correct index permutations that are necessary for contracting them with t_2 amplitudes. This is an advantage of an integral-direct RI-CCSD algorithm. If standard four-center ERIs were employed or if they were assembled prior to CCSD iterations, index permutations would be necessary before each contraction. These index permutations would have to be performed repeatedly in each iteration. Alternatively, the same class of four-center ERIs can be stored with multiple index permutations as in the parallel CCSD(T) code of Olson et al.[8] This program stores 7 permuted arrays of size $8N_O^2N_V^2$ bytes in the distributed memory. The costs incurred in both choices are completely avoided in the present algorithm. All contractions shown in lines 9,10 and 13,14 of Scheme 3 are performed via matrix–matrix multiplications within OpenMP parallel loops over occupied indices i or j requiring an $O(N_O^2N_V^3)$ scaling of the computational cost. The symmetry unfolding of the $\mathbf{T}_2$ matrix is also performed within the same loops. All tensors or their blocks required in this algorithm are either fully available or assembled on each MPI rank, which eliminates network communication altogether.

The above algorithm for the $O(N_O^3N_V^3)$ group involves the distributed assembling of two quartic memory-scaling I intermediates. If algorithm A is employed for the PPL group, the memory required for storing one of the $\mathcal{V}^{\pm}$ matrices dominates the memory need for the full RI-CCSD(T) calculation. It is then trivial to store both I intermediates. For the memory-economic algorithm B for the PPL group, on the other hand, a complementary low memory demanding algorithm is needed for the $O(N_O^3N_V^3)$ group. This is the purpose for the development of a second algorithm for these terms, which is shown in Scheme 4. The GAMESS RI-CCSD(T) program combines the efficient but memory-expensive algorithms described in Schemes 1 and 3 into "algorithm A", and the memory-economic but slower algorithms shown in Schemes 2 and 4 into "algorithm B" in actual calculations depending upon the input options. Because of its higher computational efficiency, algorithm A has been chosen as the default option.

Scheme 4. Algorithm B for the Evaluation of the $O(N_O^3N_V^3)$ Group in Eq 20

```
1 do $e = 1, N_V$ // MPI parallel
2   do $e^{\text{MyMPIrank}} = e^{\text{MyMPIrank}}_{start}, e^{\text{MyMPIrank}}_{end}$ // OpenMP parallel
3     do $m = 1, N_O$
4       $I^e(m,jb) = \sum_P \mathcal{W}^m(j,P)\mathcal{W}^e(b,P)$
5       $I^e(m,ja) = I^e(m,jb)$
6       $\mathcal{V}^e(m,nf) = \sum_P B^e(n,P)B^m(f,P)$
7     end do
8     do $jn = 1, N_O^2$
9       $\tilde{t}^{jn}(f,b) = t^{jn}_{bf} - t^{jn}_{fb}$
10      $I^e(m,jb) += \sum_f \mathcal{V}^e(m,nf)\tilde{t}^{jn}(f,b)$
11      $I^e(m,ja) += -\frac{1}{2}\sum_f \mathcal{V}^e(m,nf)t^{jn}(f,a)$
12    end do
13    do $ij = 1, N_O^2$
14      $\mathcal{W}^{ij}(b,a) = -\sum_m I^e(m,jb)t^{ie}(m,a)$
15      $\mathcal{W}^{ij}(b,a) = -\sum_m t^{ie}(m,b)I^e(m,ja)$
16      $R^{i\geq j}(a,b) += \mathcal{W}^{ij}(b,a)$ if $i \geq j$
17      $R^{i\geq j}(b,a) += \mathcal{W}^{ij}(b,a)$ if $i \leq j$
18    end do
19  end do // end OpenMP parallel
20 end do // end MPI parallel
```

The algorithm shown in Scheme 4 bypasses the storage of any quartic memory-scaling tensor by performing all integral assembling and contraction steps within a loop over the contracted virtual index e. This loop is parallelized via the hybrid MPI/OpenMP scheme. The I intermediates of eqs 27 and 28 are stored as cubic memory-scaling arrays. The remaining tensors used in this algorithm also require at most cubic-scaling memory. Note that the inner loops in Scheme 4 are not parallel. Within such an algorithm, reducing the computational scaling of the contraction steps (lines 10,11 and 14,15) as much as possible was found to be beneficial for the efficiency. This is done by performing these contractions within loops over combined indices (j, n) or (i, j) (lines 8,13 of Scheme 4), which results in an $O(N_ON_V^2)$ scaling of the computational cost for each contraction. It is important to note that vectorization of the tensor contractions is not lost in this algorithm compared to the algorithm of Scheme 3. All contractions are performed via matrix–matrix multiplications

Scheme 5. Algorithm for the (T) Correction

```
1  Assemble all $(ij|ka)$ integrals. Memory need: $8N_O^3N_V$ bytes.
2  $do\ i = 1, N_O$ // OpenMP parallel
3      Memory need: $8(i \times N_V^3)$ bytes.
4      Assemble $(ab|ci)$ integrals if memory is available.
5  $end\ do$
6  The $(ab|ci)$ integrals are available for $i_{len} \le N_O$ occupied orbitals.
7  $do\ ijk = 1, N_O(N_O+1)(N_O+2)/6 - N_O$ // MPI parallel
8      Assemble $(bc|ai)$, $(ac|bj)$, $(ab|ck)$ integrals.
       if not already assembled. Memory need for each integral: $8N_V^3$ bytes.
9      $do\ b = 1, N_V$ // OpenMP parallel
10         $W^{ijk,b}(a,c) = -\mathbf{T}_2'^{bj}(a,m)\mathbf{V}^{ik}(m,c) - \mathbf{T}_2'^{bi}(a,m)\mathbf{V}^{jk}(m,c)$
                 $+\mathbf{V}^{ib}(a,e)\mathbf{T}_2'^{jk}(e,c) + \mathbf{V}^{jb}(a,e)\mathbf{T}_2'^{ik}(e,c)$
11     $end\ do$
12     $do\ c = 1, N_V$ // OpenMP parallel
13         $W^{ijk,c}(a,b) \mathrel{+}= -\left(\mathbf{V}^{ki}(m,a)\right)^\dagger\left(\mathbf{T}_2'^{cj}(b,m)\right)^\dagger - \left(\mathbf{V}^{ji}(m,a)\right)^\dagger\left(\mathbf{T}_2'^{ck}(b,m)\right)^\dagger$
                 $-\mathbf{T}_2'^{ck}(a,m)\mathbf{V}^{ij}(m,b) - \mathbf{T}_2'^{ci}(a,m)\mathbf{V}^{kj}(m,b)$
                 $+\mathbf{V}^{ic}(a,e)\mathbf{T}_2'^{kj}(e,b) + \left(\mathbf{T}_2'^{ki}(e,a)\right)^\dagger\left(\mathbf{V}^{jc}(b,e)\right)^\dagger$
                 $+\mathbf{V}^{kc}(a,e)\mathbf{T}_2'^{ij}(e,b) + \left(\mathbf{T}_2'^{ji}(e,a)\right)^\dagger\left(\mathbf{V}^{kc}(b,e)\right)^\dagger$
14     $end\ do$
15     Assemble $(ia|jb)$, $(ia|kc)$, and $(jb|kc)$ ERIs for fixed indices $i, j$, or $k$.
16     Construct $V^{ijk}(abc)$ according to eq 37.
17     Compute $E_{(T)}^{ijk}$ according to eq 33 within OpenMP parallel loops $a \ge b \ge c$.
18 $end\ do$ // end MPI parallel
```

in both algorithms. However, each OpenMP thread has to perform more work in the algorithm of Scheme 4.

The remaining terms in eqs 20 and 21 are computationally less expensive and do not require an extensive optimization. Both algorithms A and B use the same implementation for these terms. The HHL group of eq 20 is constructed within an MPI/OpenMP parallel loop over a contracted occupied index m. This allows one to store all tensors with at most cubic-scaling memory. The algorithm imposes the index restriction $i \ge j$ for storing the I intermediate defined in eq 26, since this restriction is compatible with the storage of the R_{ij}^{ab} residual. This reduces the memory need for storing this I intermediate by a factor of 2. Furthermore, this loop restriction allows the construction of the I intermediate without a symmetry unpacking of the $\mathbf{T}_2$ matrix. The pertinent contractions are performed with a computational cost scaling as $O(N^5)$.

In every CCSD iteration, the first step is to construct the intermediates of eqs 13−16 one at a time and digest them immediately by evaluating all quantities to which they contribute, namely, the dressed Fock matrices of eqs 23−25, the singles residual R_i^a defined in eq 21, and the $\mathcal{W}_{ij}^P$ and $\mathcal{W}_{ia}^P$ intermediates of eqs 17 and 18. The $\mathcal{W}_{ab}^P$ intermediate does not depend on the intermediates of eqs 13-16, and therefore, no memory is allocated for $\mathcal{W}_{ab}^P$ until the other intermediates have been consumed and discarded. Once all terms in eqs 23−25 are constructed, the dressed Fock matrices are digested into both R_i^a and R_{ab}^{ij} residuals and discarded. Following this order of operations allows the construction of the full singles residual alongside the construction of the intermediates of eqs 13−16 and the dressed Fock matrices. The next step is to compute the $(ia) \otimes (jb)$ group of eq 20 and discard the $\mathcal{W}_{ia}^P$ intermediate. The only remaining intermediate at this stage is $\mathcal{W}_{ij}^P$, which is needed for both HHL and the $O(N_O^3N_V^3)$ groups of eq 20. On the other hand, the $\mathcal{W}_{ab}^P$ intermediate is required for the PPL and $O(N_O^3N_V^3)$ groups. Hence, one needs to store both $\mathcal{W}_{ij}^P$ and $\mathcal{W}_{ab}^P$ intermediates simultaneously. The PPL and the $O(N_O^3N_V^3)$ groups are first constructed and the $\mathcal{W}_{ab}^P$ intermediate is discarded. The final step in the construction of the CCSD residuals is to add the HHL group, followed by a global reduction of the R_{ab}^{ij} residuals from all MPI ranks.

3.2. Parallel Implementation of the Perturbative Triples Correction. The perturbative triples correction is the most compute-intensive step in an RI-CCSD(T) calculation with the numerical evaluation of the $W^{ijk}(abc)$ intermediate defined in eq 31 being the major computational task. Within the restricted loop structure of the $ijkabc$ algorithm, one needs to account for the 6-fold permutational symmetry of the triples amplitudes explicitly. This leads to a total of 12 tensor contractions through eq 31.

A common approach to implement the $ijkabc$ algorithm is to store the $W^{ijk}(abc)$ intermediate as a vector of length N_V^3. The terms on the right-hand side of eq 31 are evaluated via matrix−matrix multiplications. One specific drawback of this algorithm is the difficulty of adding the results of matrix multiplications under different permutations of eq 30 to correct locations of the $W^{ijk}(abc)$ array. This is illustrated with the first three permutations of eq 30 and only the first term on the right-hand side of eq 31.

$$\text{permutation}\begin{pmatrix} ijk \\ abc \end{pmatrix}: \omega(abc) = -\mathbf{T}_2^i(ab,\, m)\mathbf{V}^{jk}(m,\, c) \tag{44}$$

$$\text{permutation}\begin{pmatrix} jik \\ bac \end{pmatrix}: \omega(bac) = -\mathbf{T}_2^j(ba,\, m)\mathbf{V}^{ik}(m,\, c) \tag{45}$$

$$\text{permutation}\begin{pmatrix} ikj \\ acb \end{pmatrix}: \omega(acb) = -\mathbf{T}_2^i(ac,\, m)\mathbf{V}^{kj}(m,\, b) \tag{46}$$

where the $\mathbf{V}$ matrix stores the four-center ERIs $(jm|kc)$. Unlike the $\omega(abc)$ intermediate of eq 44, the $\omega(bac)$ and $\omega(acb)$ intermediates of eqs 45 and 46 cannot be directly added to $W^{ijk}(abc)$ without permuting the compound vector indices bac or acb to the correct order abc. A simple way to add the contributions of eqs 44–46 is to accumulate the initial results of matrix multiplications into an auxiliary array ω of length N_V^3 as shown above, followed by an index permutation operation on ω and a vector addition (via DAXPY) to the $W^{ijk}(abc)$ array. This strategy is suitable for a parallel implementation, since it is possible to define 12 independent parallelizable tasks, each consisting of a matrix multiplication, an index permutation, and a vector addition. Alternatively, the vector addition step may be skipped and the results of matrix multiplications may be directly added[8] to $W^{ijk}(abc)$ after index permutations. However, the various permutations then become mutually dependent, thereby making parallelization difficult.

The index permutation and vector addition operations involved in both of the above cases are much less efficient than the matrix multiplications.[10] In a compute-intensive process, such as the (T) correction, these tasks might become competitive with the more efficient matrix multiplications and are likely to obviate much of the parallel efficiency. Recently, Nagy and Kállay[41,54] proposed an algorithm that partially or completely eliminates these poorly scaling tasks by exploiting the intrinsic permutational symmetries of the spatial orbital-based ERI and $\mathbf{T}_2$ tensors. A similar strategy is employed in the implementation described here.

The key idea is to store the $W^{ijk}(abc)$ intermediate as a three-index array $W^{ijk}(a,b,c)$ of the same length N_V^3 as the corresponding vector, and to evaluate the various contributions of eq 31 within a loop over a virtual index, for example, b. Since this index is fixed within the loop, $W^{ijk}(a,b,c)$ can be practically treated as a two-index array. While explicit permutations of the compound vector indices are necessary when adding the results of matrix multiplications to the vector $W^{ijk}(abc)$, these results can be directly added to a two-index array via simple matrix transpositions. In fact, no separate matrix transposition is necessary and the results of the matrix multiplications can be directly brought into the correct index order, for example, $W^{ijk,b}(a,c)$ simply by controlling the first two arguments of the DGEMM routine (that are related to matrix transposition) and by altering the order of the ERI and $\mathbf{T}_2$ matrices in the DGEMM call. This strategy can be applied to all 12 contributions, thereby eliminating all index permutations and vector additions. Scheme 5 illustrates the implemented algorithm for the (T) correction schematically, where the inner loops over virtual indices are OpenMP parallel. A minimum of two inner loops is required for accumulating all 12 contributions of eq 31 within the OpenMP parallel algorithm in order to avoid race conditions.

The first term on the right-hand side of eq 31 involves a summation over the occupied index m, which in turn requires the full symmetry unpacked $\mathbf{T}_2$ matrix. Since repeated symmetry unpacking within the $i \geq j \geq k$ loops would further increase the already large computational load, the symmetry unpacking is performed prior to the $i \geq j \geq k$ loops. The symmetry unpacked t_2 amplitudes are stored into a single $\mathbf{T}_2'$ matrix that requires $8N_O^2N_V^2$ bytes of memory. Both the original symmetry packed $\mathbf{T}_2$ matrix and the residual matrix $\mathbf{R}_2$ are discarded, as they are no longer needed. Moreover, the $\mathbf{T}_2'$ matrix is stored as a four-index array T_2' (a, b, i, j). This allows the algorithm to access both t_{ij}^{ab} and t_{ij}^{ba} amplitudes for the fixed indices j and b (within loops) as the submatrices $\mathbf{T}_2'$ $(:,b,:,j)$ and $\mathbf{T}_2'$ $(b,:,:,j)$ without any index permutation. The ERIs needed for evaluating the right-hand side of eq 31 are also stored in a similar way.

Repeated integral assembling operations within the $i \geq j \geq k$ loops are likely to slow down the computation even with MPI/OpenMP parallelization. Therefore, the best strategy is to assemble the integrals outside the $i \geq j \geq k$ loops. Two classes of four-center ERIs are needed for computing the contributions of eq 31, namely, $(ij|ka)$ and $(ab|ci)$ [i.e., the (OO|OV) and the (VV|VO) classes]. The storage cost for the (OO|OV) ERIs scales as $N_O^3N_V$. These ERIs are fully preassembled and stored in the algorithm described here. The storage cost for the (VV|VO) ERIs scales as $N_ON_V^3$. For large-scale applications, storing the full (VV|VO) ERI matrix on each MPI rank would be tremendously memory-expensive. For this reason, they are preassembled for a limited number of occupied orbitals $i_{\text{len}} \leq N_O$ as long as memory is available (lines 2–6 of Scheme 5). As stated above, the large quartic memory-scaling residual matrix $\mathbf{R}_2$, as well as all intermediates required for the CCSD iterations, are discarded once the iterations have converged. The memory thus released is used for storing a fraction of the preassembled (VV|VO) ERIs, while the remaining ERIs need to be assembled within the $i \geq j \geq k$ loops. Reducing the integral assembling task partly in this way, or fully in the case of small to moderate-sized applications, is highly beneficial for reducing the wall time for the triples correction step.

Note that the memory demand for the full RI-CCSD(T) calculation is determined by the RI-CCSD step. No additional memory is allocated for triples corrections. If algorithm A is employed for RI-CCSD, the memory required for the evaluation of the PPL group is large enough to allow the storage of the preassembled (VV|VO) ERIs for about 40–60% of the occupied orbitals even in fairly large applications (e.g., for calculations involving 1066 AO basis functions reported in section 5.2, these ERIs could be stored for up to 49% of the occupied orbitals). For the memory-economic algorithm B, on the other hand, the memory demand for the RI-CCSD step is small, and therefore, a smaller number of (VV|VO) ERIs can be preassembled. Thus, the choice of the algorithm for RI-CCSD influences the efficiency (wall time) of the (T) correction by controlling the fraction of the preassembled (VV|VO) ERIs.

The minimum memory requirement for the perturbative triples correction amounts to the sum of (a) $8(N_O^2 + N_V^2 + N_ON_V)N_{\text{aux}}$ bytes for storing the RI integrals, (b) $8(N_O + N_V)$ bytes for storing the orbital energies, (c) $8(N_O^2N_V^2 + N_ON_V)$ bytes for storing the symmetry unpacked $\mathbf{T}_2'$ and $\mathbf{T}_1$ amplitude matrices, (d) $8N_O^3N_V$ bytes for storing the full set of preassembled (OO|OV) ERIs, and (e) four three-index arrays each of size $8N_V^3$ bytes. In this minimal memory route, all (VV|

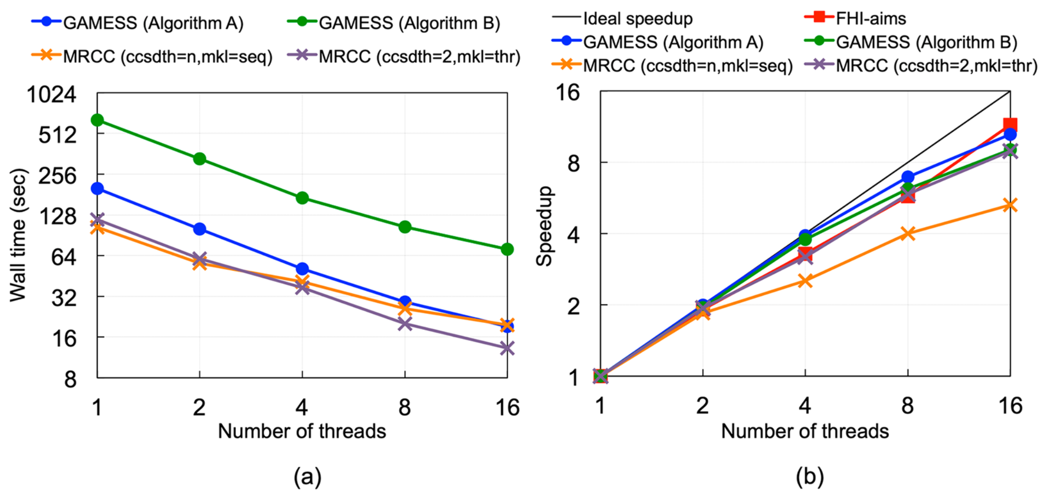


Figure 1. Multithreaded performance of the RI-CCSD iteration for different implementations tested with a $(H_2O)_{10}$ cluster using the cc-pVDZ/cc-pVDZ-RI basis sets. (a) Wall times and (b) speedup values relative to the wall time for the single thread computation. Two different settings were employed in the calculations using MRCC, namely, (i) using the sequential BLAS and avoiding nested OpenMP parallelism (denoted as ccsdth = n, mkl = seq) and (ii) using the threaded BLAS and assigning two OpenMP threads to the outer parallel regions (denoted as ccsdth = 2, mkl = thr).

VO) ERIs are assembled within the $i \geq j \geq k$ loops. Three arrays of size $8N_V^3$ bytes are used for storing these ERIs for the indices i, j, and k, while the fourth one is used for storing the $W^{ijk}(a,b,c)$ matrix. The construction of the $V^{ijk}(a,b,c)$ intermediate of eq 37 requires the (OV|OV) class of ERIs. However, these ERIs are needed only for specific occupied index pairs (i, j), (j, k), and (k, i) unlike the full set of (OO|OV) or (VV|VO) ERIs. In the new algorithm, the (OV|OV) ERIs are assembled within the $i \geq j \geq k$ loops after all contributions to eq 31 have been added. This facilitates the reuse of the three arrays of size $8N_V^3$ bytes for storing the assembled (OV|OV) ERIs, which were initially used for storing the (VV|VO) ERIs. For example, the $\mathbf{V}^{ij}$ (a, b) ERI matrix of size $8N_V^2$ bytes is stored as $\mathbf{V}^{ij}(a, b, 1)$ in a three-index array.

4. PERFORMANCE ANALYSIS

In this section, the parallel efficiency of the RI-CCSD(T) program in GAMESS is analyzed and the relative memory requirements for algorithms A and B for the RI-CCSD iteration are compared. All calculations were performed on two local computer clusters, namely, "cj" at Ames Laboratory and "Nova" at the high-performance computing facility of the Iowa State University. Each compute node of cj consists of two 12-core Intel Xeon E5-2695 v2 processors and has 126 GB of RAM, while the compute nodes of Nova are equipped with two 18-core Intel Skylake 6140 Xeon processors and have either 192 or 384 GB RAM capacities. The compute nodes of cj and Nova are interconnected through the InfiniBand FDR (56 Gbps bandwidth) and EDR (100 Gbps bandwidth) networks, respectively. Most calculations related to performance and accuracy benchmarks were performed on cj, while the large-scale calculations reported in sections 4.3 and 5 were performed on Nova. In addition, the multinode performance of the program was tested on the US supercomputer "Theta" at the Argonne Leadership Computing Facility (ALCF). Theta is a Cray XC40 supercomputer comprised of 64-core Intel KNL 7230 compute nodes and has 192 GB of memory per node. The GAMESS program suite was compiled with the Intel compiler version 18.3, the Intel MPI library, and the serial version of the MKL. The implemented RI-CCSD(T) program has been made available through the September 2020 public release of GAMESS (https://www.msg.chem.iastate.edu/games/index.html).

The RI-CCSD(T) program uses the DDI framework[15,16] of GAMESS and avoids explicit use of MPI directives. The DDI model is general and can be compiled with any standard MPI library. The DDI runs an equal number of compute processes and data servers on each node.[15] The former execute the actual computation, while the latter are involved in communicating data stored in the distributed memory. The numbers of MPI ranks per node mentioned in the following sections refer only to the compute processes. For all calculations, the OpenMP threads were bound either to physical cores (on Nova) or to hardware threads (two and four per physical core on cj and Theta, respectively) with OMP_PROC_BIND = close. The MPI processes were pinned onto the domains of OpenMP threads via I_MPI_PIN_DOMAIN = omp, with the domain size specified via OMP_NUM_THREADS.

The correlation-consistent cc-pVXZ[82] and aug-cc-pVXZ[83] (X = D, T, Q) basis sets were employed in all benchmark calculations used for testing the accuracy and the parallel performance of the RI-CCSD(T) program along with the matching cc-pVXZ-RI and aug-cc-pVXZ-RI auxiliary basis sets.[78] For the large-scale calculations reported in sections 4.3 and 5.2, the triple-ζ valence basis set with polarization functions, def2-TZVPP,[84] was used along with the def2-TZVPP-RI[77] auxiliary basis set. All calculations employed the frozen core approximation.

4.1. Multithreaded Performance.

The multithreaded performance of the RI-CCSD(T) program was tested with a $(H_2O)_{10}$ cluster using the cc-pVDZ/cc-pVDZ-RI basis sets. Although this is a very small system for testing the parallel efficiency, it was chosen here for comparison with certain other parallel RI-CCSD(T) implementations reported recently,[40,41] namely, those in the FHI-aims[85] and the MRCC program suites.[86] The MRCC program suite was compiled on cj with the same compiler and libraries as was used for GAMESS to ensure that the results from both programs are obtained with identical hardware settings. The wall times for the RI-CCSD iteration and the speedup values relative to the single-thread computation obtained with the GAMESS and MRCC implementations are compared in Figure 1. In addition, Figure

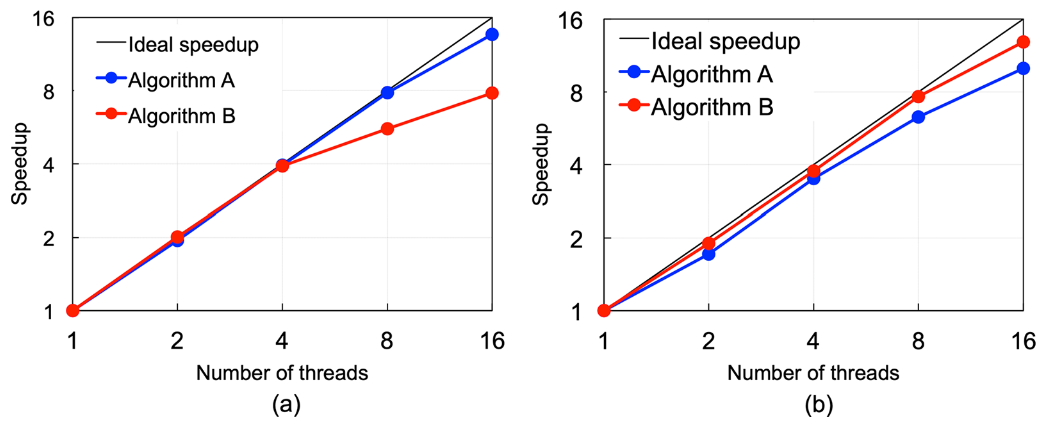


Figure 2. Multithreaded performance of the two most expensive steps in the RI-CCSD iteration in the GAMESS implementation: (a) the PPL group and (b) the $O(N_O^3N_V^3)$ groups tested with a $(H_2O)_{10}$ cluster and the cc-pVDZ/cc-pVDZ-RI basis sets.

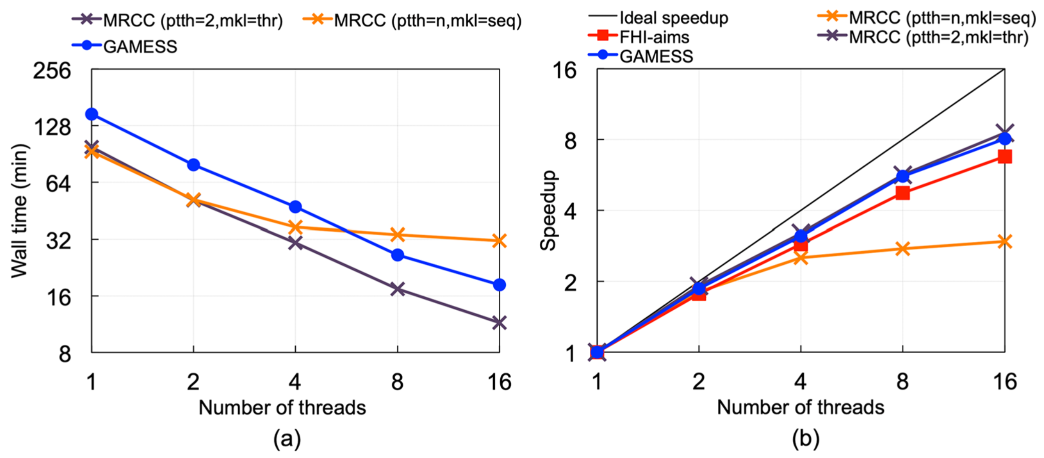


Figure 3. Multithreaded performance of the (T) correction for different implementations tested with a $(H_2O)_{10}$ cluster using the cc-pVDZ/cc-pVDZ-RI basis sets. (a) Wall times and (b) speedup values relative to the wall time for the single thread computation. Two different settings were employed in the calculations using MRCC, namely, (i) using the sequential BLAS and avoiding nested OpenMP parallelism (denoted as ptth = n,mkl = seq), and (ii) using the threaded BLAS and assigning two OpenMP threads to the outer parallel regions (denoted as ptth = 2,mkl = thr).

1 compares the speedup values obtained with the FHI-aims code. The wall times for the FHI-aims code were taken from ref 40. Hence, they are not directly comparable. However, the speedup values can be expected to be independent of hardware settings used for compiling the program.

It is important to note that the MRCC implementation uses nested OpenMP parallelism and threaded BLAS, none of which are used in the GAMESS implementation. Therefore, a fair comparison would require using sequential BLAS for the MRCC implementation and associating all OpenMP threads only with the outer parallel regions. In Figure 1, these results are indicated as "MRCC (ccsdth=n,mkl=seq)" with n = 1−16, where "ccsdth" (or "ccsdthreads") is the number of OpenMP threads associated with the outer parallel regions. As Figure 1 indicates, this setting does not lead to the best performance for the MRCC implementation. Hence, a second set of results was obtained with threaded BLAS and by assigning two OpenMP threads to the outer parallel regions (except for the single thread calculation). These results are indicated as "MRCC (ccsdth=2,mkl=thr)" in Figure 1.

The wall times for the RI-CCSD iteration using the GAMESS algorithm B are 3−4 times larger than those for the algorithm A. Figure 1a indicates that the wall times for these two algorithms do not become closer as the number of OpenMP threads increases. However, as hypothesized in section 3.1, both algorithms are found to exhibit a similar strong scaling with 10.5× and 9× speedups on 16 threads for algorithms A and B, respectively (see Figure 1b). Figure 1a shows that the MRCC implementation takes nearly half as much wall time per RI-CCSD iteration as does the GAMESS algorithm A when the former is invoked with threaded BLAS and nested OpenMP parallelism. This may be attributed to the fact that the GAMESS algorithm A constructs the symmetric and antisymmetric combinations defined in eqs 39−41 in two steps in order to save memory. Figure 1b indicates that both GAMESS algorithms for the RI-CCSD iteration show good speedups that are comparable to the implementations in other program suites, namely, between 9× and 11× with 16 threads.

Figure 2 shows the speedups for the two most expensive steps involved in the RI-CCSD iteration, namely, the construction of the PPL and the $O(N_O^3N_V^3)$ groups. The algorithm A for the PPL group shows an excellent strong scaling with a 13× speedup on 16 threads. For algorithm B, the speedup for the PPL group with 16 threads is relatively small (8×). For the $O(N_O^3N_V^3)$ group, both algorithms show similar speedups with values between 10 and 12.8 on 16 threads.

The multithreaded performance of the (T) correction for different implementations is shown in Figure 3. The wall times

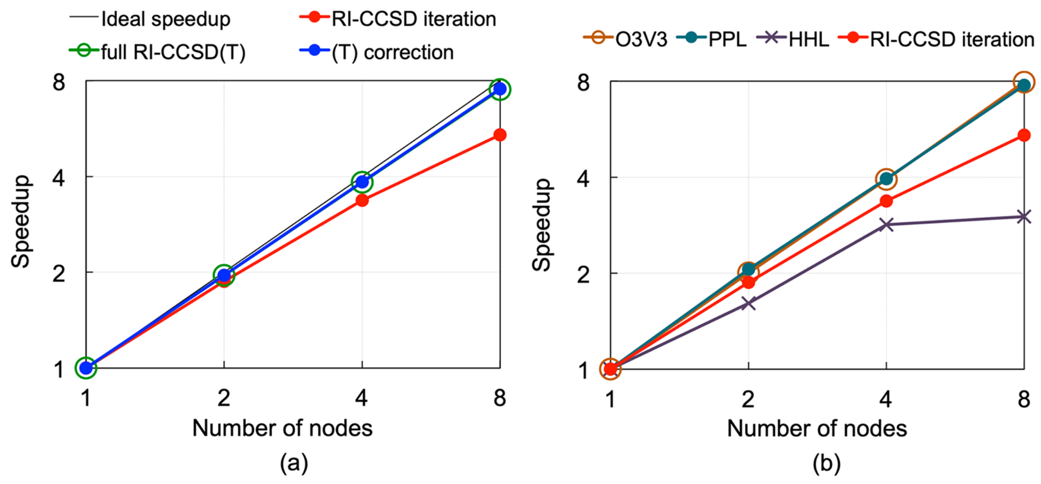


Figure 4. Multinode performance of the RI-CCSD(T) program tested on cj for the coronene molecule using the cc-pVDZ/cc-pVDZ-RI basis sets. (a) Speedups for the full RI-CCSD(T) calculation relative to the single node computation are shown along with the RI-CCSD iteration and the (T) correction. (b) Speedups for three representative steps in RI-CCSD iteration, namely, the evaluation of the PPL, $O(N_O^3N_V^3)$ (denoted as O3V3 in the plot), and the HHL groups are shown separately.

for the MRCC implementation were obtained with (i) sequential BLAS and associating all OpenMP threads (1−16) with the outer parallel region, and with (ii) threaded BLAS and assigning two OpenMP threads to the outer parallel region. In Figure 3, these results are indicated as "MRCC (ptth=n,mkl=seq)" and "MRCC (ptth=2,mkl=thr)", respectively, with "ptth" (or "ptthreads") indicating the number of OpenMP threads associated with the outer parallel region.

The GAMESS algorithm for the (T) correction relies on the MPI-based parallelism for distributing the major bulk of the computational load, viz., the parallelization of the outer loops $i \geq j \geq k$. Only the inner loops over virtual indices are parallelized via OpenMP (see Scheme 5). Therefore, it is unlikely that the (T) correction step of the GAMESS implementation can be greatly expedited solely by invoking OpenMP parallelism while using a single MPI rank. For the MRCC implementation of the (T) correction, a part of the outer loops $i \geq j \geq k$ is parallelized via OpenMP in addition to parallelizing the inner virtual loops. This is possible due to the use of nested OpenMP. Hence, the wall times for the MRCC implementation invoked with nested OpenMP are expected to be shorter than those for the GAMESS implementation. Figure 3a reflects this trend. However, both algorithms show a similar strong scaling for the (T) correction (see Figure 3b).

Figures 1 and 3 indicate that the MRCC implementation[41] is faster than the GAMESS implementation on a single core. The sequential algorithms for the two programs have been designed to be consistent with different parallelization strategies. The MRCC implementation has been designed to achieve maximum efficiency through nested OpenMP parallelism and threaded BLAS, which contribute both to reducing wall times and to improving the strong scaling. This becomes clear when one compares the performances of the MRCC algorithm with and without nested OpenMP and threaded BLAS. The design goal of the GAMESS algorithm is to achieve an excellent strong scaling of the most compute-intensive steps on a few hundred up to a few thousand cores. While nested OpenMP and threaded BLAS expedite the computation on a given number of cores, achieving an optimum strong scaling on such a wide range of cores is likely to require careful adjustments of the thread distribution between the outer and the inner parallel regions depending upon the number of cores employed. Since optimizing the thread distribution is a nontrivial task, in particular for a nonexpert user, the nested OpenMP parallelism was not employed in the GAMESS implementation.

It is evident from the present results that both the RI-CCSD iteration and the (T) correction steps of the GAMESS implementation show a strong scaling that is comparable to the MRCC implementation. The results reported in the following sections indicate that the GAMESS implementation, which is built upon a somewhat slower sequential algorithm than the MRCC implementation, is successful in achieving a very good strong scaling for the most demanding PPL and $O(N_O^3N_V^3)$ groups and the (T) correction both on ~100 and on ~8000 cores. For enhancing the efficiency further, adapting the current RI-CCSD(T) program to a heterogeneous architecture based on CPUs and graphics processing units (GPUs) will be explored in the future. This is ideal for both accelerating the computation and achieving high scalability. Having a logically simple code without runtime adjustable parameters is also beneficial for this purpose.

4.2. Multinode Performance.

The multinode performance of the RI-CCSD(T) program was tested with the coronene ($C_{24}H_{12}$) molecule using the cc-pVDZ/cc-pVDZ-RI basis sets. Unless explicitly mentioned, algorithm A was used for the RI-CCSD iteration. The first set of results shown in Figure 4 was obtained with cj using up to eight compute nodes and one MPI rank per node. The number of OpenMP threads was chosen to be 24 per MPI rank in all calculations. The speedup values relative to the wall time for the single node computation are shown in Figure 4. An excellent near-linear speedup is observed for the (T) correction with a parallel efficiency (speedup/number of nodes) of 94.3% on eight nodes (i.e., on 192 cores). The parallel efficiency of the RI-CCSD iteration is 67.6% on eight nodes. Figure 4(b) indicates that the PPL and the $O(N_O^3N_V^3)$ groups show excellent speedups, while the HHL group exhibits somewhat poor strong scaling.

While the performance of the RI-CCSD(T) program in GAMESS is satisfactory on a small number of nodes, for large-scale applications it is important to analyze the strong scaling in the limit of a large number of compute nodes. This test was

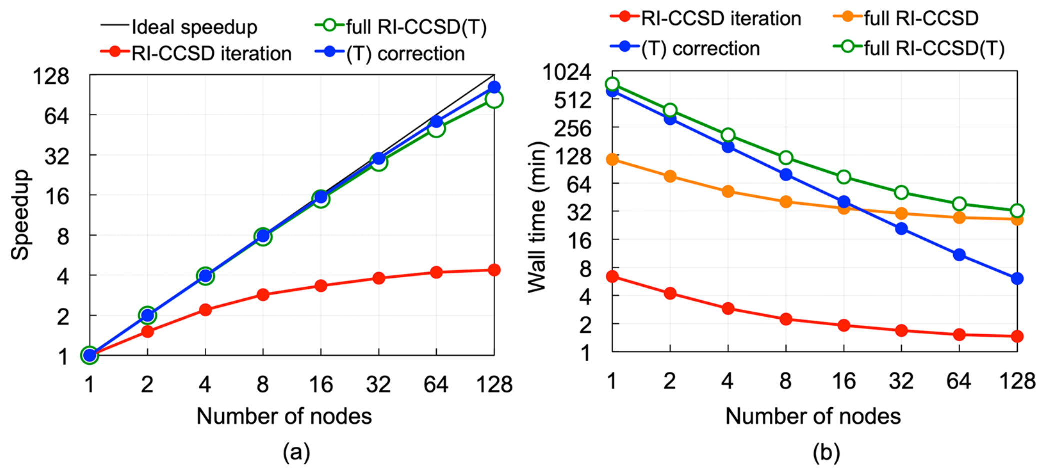


Figure 5. Multinode performance of the RI-CCSD(T) program tested on ALCF Theta for the coronene molecule using the cc-pVDZ/cc-pVDZ-RI basis sets. (a) Speedups relative to the single node computation and (b) wall times for the full RI-CCSD(T) calculation are shown along with the RI-CCSD iteration and the (T) correction.

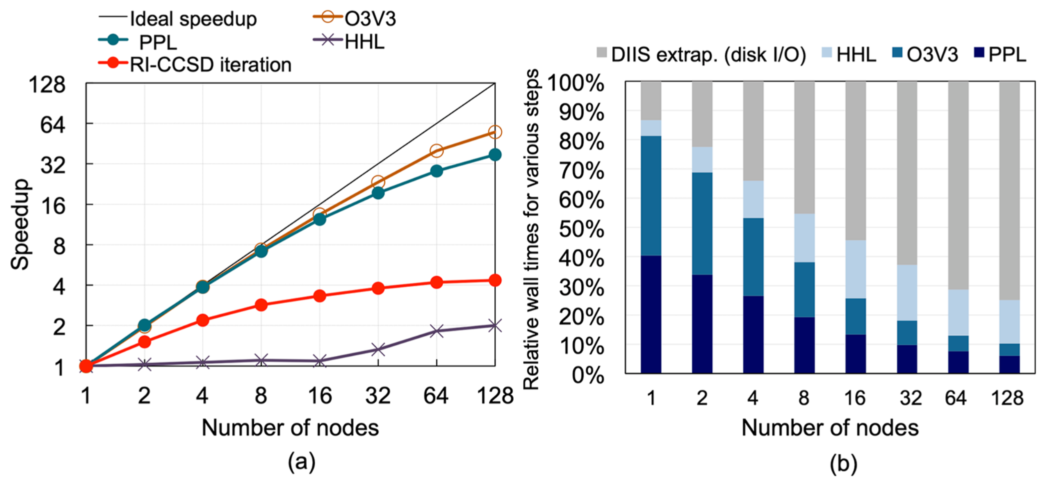


Figure 6. Multinode performance of the RI-CCSD iteration tested on ALCF Theta for the coronene molecule using the cc-pVDZ/cc-pVDZ-RI basis sets. (a) Speedups and (b) relative wall times for the various parallel and sequential steps.

performed on the supercomputer Theta using the same molecule (coronene) and basis sets as in the above benchmark. In this case, up to 128 nodes were employed with one MPI rank per node and 64 threads per MPI rank. Sufficient memory was allocated for all of these calculations, such that the full set of (VV|VO) ERIs could be preassembled outside the $i \geq j \geq k$ loops (see Scheme 5) irrespective of the number of nodes employed. This is important for treating the calculations performed with different numbers of nodes on an equal footing. The speedup values relative to the single node computation and the wall times are shown in Figure 5.

Figure 5a indicates that the (T) correction shows an excellent near-linear speedup with a parallel efficiency of 80.4% on 128 nodes (i.e., on 8192 cores). It may be concluded from Figures 4a and 5a that the (T) correction algorithm in the GAMESS implementation shows an excellent strong scaling both on a few hundred, as well as on a few thousand, cores. Furthermore, Figure 5a shows that the strong scaling of the full RI-CCSD(T) calculation is nearly identical with that of the (T) correction step. Note that the strong scaling of the full RI-CCSD(T) calculation was computed on the basis of the average wall time for a single RI-CCSD iteration and the wall time for the (T) correction. This is because the number of iterations required to converge the cluster amplitudes to a certain threshold is molecule-dependent and does not reflect on the parallel efficiency of the code.

One finds from Figure 5a that the RI-CCSD iteration shows a much poorer strong scaling than the (T) correction and that there is practically no speedup beyond 32 nodes. This is consistent with the fact, as shown by the orange curve in Figure 5b, that the total time to converge the CCSD iterations to the standard threshold of 10^{-7} (18 iterations were needed in the present case) exceeds the wall time for the (T) correction on 32 or a larger number of nodes. In this range, the wall time for the full RI-CCSD(T) calculation is dominated by the RI-CCSD part [the green curve in Figure 5b]. However, the wall time per RI-CCSD iteration remains shorter than the (T) correction throughout the entire range of nodes, as indicated by the red curve in Figure 5b.

Since the system size is fixed in this benchmark, the observed poor strong scaling might be an outcome of the lack of enough parallelizable tasks when using a large number of cores. It is thus interesting to study the weak scaling of the RI-CCSD iteration. In a weak-scaling analysis, one increases both the number of cores and the system size by the same factor, such that the computational workload per processor remains

Table 1. Number of Active Cores Involved in the Computation of the PPL, $O(N_O^3N_V^3)$, and the HHL Groups versus the Actual Numbers of Compute Cores Employed[a]

number of loop iterations		PPL group		$O(N_O^3N_V^3)$ group		HHL group	
		$N_V\ (N_V + 1)/2 = 50721$		$N_ON_V = 17172$		$N_O = 54$	
number of MPI ranks employed	number of threads employed	number of active MPI ranks	number of active threads	number of active MPI ranks	number of active threads	number of active MPI ranks	number of active threads
1	64	1	64	1	64	1	54
2	64	2	64	2	64	2	27
4	64	4	64	4	64	4	14/13
8	64	8	64	8	64	8	7/6
16	64	16	64	16	64	16	4/3
32	64	32	64	32	64	32	2/1
64	64	64	64	64	64	54	1
128	64	128	64	128	64	54	1

[a]The estimates are given for the coronene molecule using the cc-pVDZ/cc-pVDZ-RI basis sets, for which $N_O = 54$ and $N_V = 318$.

constant. Since the amounts of computational task associated with various terms in the RI-CCSD equations are widely different, increasing the system size would lead to different increments in the computational workload for different terms. This makes designing a weak scaling test complicated. To have a clear idea, it is useful to analyze the strong scaling of the three representative term groups, namely, PPL, $O(N_O^3N_V^3)$, and HHL.

Figure 6a indicates that the most steep-scaling and rate-limiting PPL and also the $O(N_O^3N_V^3)$ groups show much better speedups compared to the overall RI-CCSD iteration, whereas the HHL group practically does not show any speedup. Figure 6b further indicates that the much better strong scaling of the PPL and $O(N_O^3N_V^3)$ groups make them less time-consuming when using a large number of cores and the poorly scaling terms such as the HHL group as well as the sequential DIIS extrapolation become dominant on the relative time scale.

The trend observed in Figure 6a can be explained by comparing the number of iterations in the MPI/OpenMP parallel do loops in the algorithms for the three representative groups, namely, N_O for the HHL group as compared to much larger $N_V\ (N_V + 1)/2$ for the PPL group and N_ON_V for the $O(N_O^3N_V^3)$ group (when using algorithm A of Scheme 3). Table 1 shows the number of loop iterations for the system under consideration along with the number of active cores (MPI ranks and OpenMP threads). It is evident that all cores remain active in the computation of the PPL and the $O(N_O^3N_V^3)$ groups. For the HHL group, on the other hand, the number of active threads decreases from 54 to 1 as the number of active MPI ranks increases from 1 to 54, since the total number of active cores cannot exceed the number of parallelized loop iterations, $N_O = 54$. Thus, a large number of cores remain idle during the computation of the HHL group, and this explains its poor strong scaling. Note that a similar situation would arise also for the $O(N_O^3N_V^3)$ group if algorithm B of Scheme 4 is employed, since $N_V = 318$ is much smaller than N_ON_V. However, for small-scale applications such as the present one, the more efficient algorithm A would always be viable because of its low memory demand and its use is recommended.

The present results indicate that even for a small molecule and basis set as considered in this strong scaling analysis, the amounts of computational workload for the PPL and the $O(N_O^3N_V^3)$ groups are large enough to engage up to ∼8000 cores. The poor strong scaling of the RI-CCSD iteration is thus caused by the low-cost terms, for example, the HHL group. Therefore, it is reasonable to design a weak scaling test that would ensure that all compute cores are active during the computation of the HHL group. This is possible if a set of molecules with an increasing number of atoms is chosen for the benchmark, while using the same basis set for each molecule. This would lead to an increment in N_O (also in N_V) and the number of cores must be increased proportionately.

The weak scaling benchmark test considered in this work consisted of water clusters, $(H_2O)_n$ with $n = 6-30$, all of which were described by the cc-pVDZ/cc-pVDZ-RI basis sets. Table 2 shows the values of N_O and the number of cores used for

Table 2. Weak Scaling Benchmark for the RI-CCSD Iteration Using $(H_2O)_n$ Clusters Described by cc-pVDZ/cc-pVDZ-RI Basis Sets[a]

$(H_2O)_n$	N_O	number of cores
$(H_2O)_6$	24	24
$(H_2O)_{12}$	48	48
$(H_2O)_{18}$	72	72
$(H_2O)_{24}$	96	96
$(H_2O)_{30}$	120	120

[a]The lengths of the MPI/OpenMP parallelized do loops in the algorithm for the HHL group (N_O) are shown along with the number of compute cores employed.

different water clusters. Figure 7a shows the plot of the scaled speedup $S_P = P + (1 - P)\alpha$ according to Gustafson's law,[87] where α is the serial fraction, as a function of the number of cores P. The only sequential step in the RI-CCSD iteration is the DIIS extrapolation. The serial fraction α was computed as the ratio of the wall time for the DIIS extrapolation to the wall time for the full RI-CCSD iteration. A 90× speedup is observed for $(H_2O)_{30}$ on 120 cores, which is much better than the strong scaling observed in Figure 5. Note, however, that the weak scaling is not linear. This is not surprising, as the DIIS extrapolation involves disk I/O operations. As shown in Figure 7b, the data size involved in these operations increases with the system size, and the serial fraction α increases proportionately.

The strong scaling of the algorithms A and B for the PPL group were compared in the limit of a large number of compute cores using the coronene molecule and the cc-pVDZ/cc-pVDZ-RI basis sets as described before. The results are shown in Figure 8. It is evident from Figure 8a that the wall

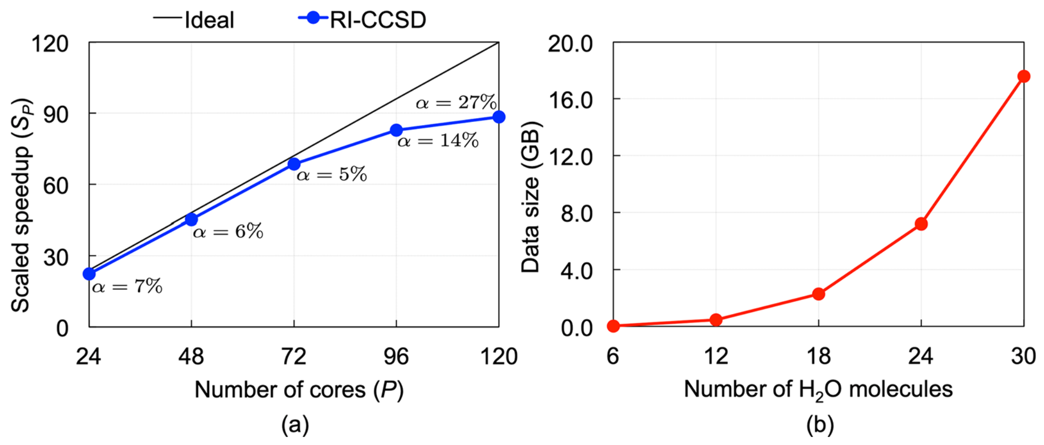


Figure 7. Weak scaling of the RI-CCSD iteration tested with water clusters, $(H_2O)_n$ with $n = 6-30$, using the cc-pVDZ/cc-pVDZ-RI basis sets. (a) Plot of the scaled speedup as a function of the number of cores. The quantity α indicates the serial fraction in % (see text). (b) Data size involved in the disk I/O operations in the DIIS extrapolation for different water clusters.

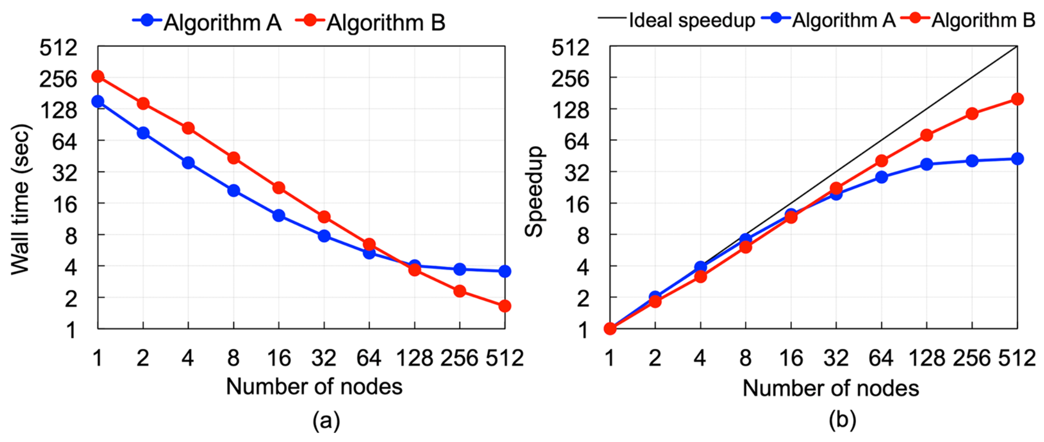


Figure 8. Relative performances of the algorithms A and B for the PPL group tested on ALCF Theta for the coronene molecule using the cc-pVDZ/cc-pVDZ-RI basis sets. (a) Wall times and (b) speedups relative to the single-node computation.

Table 3. Memory Requirements and Wall Times for the RI-CCSD Algorithms A and B for the Retinol Molecule Using the def2-TZVPP/def2-TZVPP-RI Basis Sets[a]

		memory requirement per MPI rank		wall time per RI-CCSD iteration	
		algorithm A	algorithm B	algorithm A	algorithm B
number of basis functions	1071				
N_O	58	288 GB	91 GB	46 m	3 h
N_V	992				
N_{aux}	2496				

[a]All calculations employed nine compute nodes, one MPI rank per node, and 18 OpenMP threads per MPI rank.

times for the two algorithms become comparable when using 128 nodes, and algorithm B indeed becomes faster if the number of nodes is further increased to 256 or 512. Furthermore, Figure 8b indicates that algorithm B shows a better speedup than algorithm A in the limit of 256 or 512 nodes even though its strong scaling is slightly inferior when a small number of nodes are used. Hence, there is a crossover point that occurs for the present calculation at 16 nodes. These results imply that the intrinsic inefficiency of algorithm B due to reduced vectorization of the tensor contraction (eq 41) may be well compensated for by employing a large number of cores. The rationale for this observation is obtained from a close look at the steps involved in the construction of the PPL group. Both algorithms A and B include memory bandwidth-bound symmetrization and antisymmetrization operations (eqs 39, 40, 42, and 43) that cannot be vectorized, although they have been parallelized via OpenMP. In the case of algorithm A, these less efficient operations become dominant over the more efficient vectorized tensor contraction when a large number of cores are employed, and this leads to a poor speedup in this limit. The less vectorized tensor contraction used in algorithm B (via DGEMV), on the other hand, is more comparable in efficiency with the memory bandwidth-bound operations. This uniform efficiency of all steps involved in algorithm B results in a better combined parallel efficiency, which becomes apparent particularly in the limit of a large number of cores.

4.3. Memory Requirements for the RI-CCSD Algorithms A and B.

The relative memory requirements and the wall times for the RI-CCSD algorithms A and B were assessed using the retinol molecule and the def2-TZVPP/def2-TZVPP-RI basis sets as a test case. Table 3 gives an estimate of the size of the calculation. Algorithm A requires 1814 GB of total memory for storing one of the $\mathcal{V}^{\pm}$ and one of the $\sigma^{\pm}$ matrices defined in eqs 39 and 41. It is then necessary to adjust the number of MPI ranks in such a way that the size of the blocks of these matrices to be stored on each MPI rank fits into the RAM capacity of a node. The use of one MPI rank per compute node is recommended for large-scale applications, such that the entire node memory is available for storing large matrices and in addition, all logical cores on a node can be assigned to OpenMP threads. The reported calculations were performed on the "wide" nodes of Nova that have a RAM capacity of 384 GB, and nine compute nodes were employed. This led to 288 GB of memory requirement per MPI rank. For these calculations, 18 OpenMP threads per node were employed. On a total of 162 cores, each RI-CCSD iteration took an average wall time of 46 min with algorithm A. Assigning a larger number of MPI ranks would further reduce both the memory requirement and the wall time for the RI-CCSD iteration.

With an identical number of cores, algorithm B required 3 h of wall time per RI-CCSD iteration. The redeeming feature of this algorithm is evident from Table 3. Algorithm B requires only 91 GB of memory per MPI rank for the present calculation. As already stated, algorithm B does not assemble or store any four-index ERI matrix in a distributed manner. Therefore, the memory required for this algorithm does not need to be scaled down by assigning a certain number of nodes unlike in the case of algorithm A. In principle, therefore, this calculation can even be performed on a single multiprocessor node using algorithm B. Assigning more than one node would add to the efficiency of this algorithm, which is beneficial also for the (T) correction. Thus, algorithm B makes fairly large calculations, for example, those involving ∼1000 atomic basis functions, feasible on a standard computer cluster consisting of only 10−20 compute nodes and a moderate amount of node memory, e.g., up to 125 GB. Such computer clusters are widely available and used. Note that the major source of the difference in the wall times for the two algorithms is the time required to evaluate the PPL group, viz., 28 min with algorithm A and 2.3 h with algorithm B. The evaluation of the $O(N_O^3N_V^3)$ group took between 7 and 12 min using both algorithms.

On the basis of the results reported in sections 4.2 and 4.3, it is relevant to point out a potential advantage of algorithm B. The memory demand of an algorithm is a concern when adapting a CPU-only program to the hybrid CPU/GPU architecture, since the memory availability for the GPUs is usually much smaller than for CPUs, for example, only 16−32 GB for the NVIDIA Tesla V100 GPUs. The low memory demand of algorithm B makes it more amenable for GPU adaptation than algorithm A. Of course, further modifications are needed for a proper distribution of the associated big data. Moreover, as Figure 8 demonstrates, algorithm B scales better than algorithm A in the limit of a large number of cores. Typically, a few thousand cores are needed to achieve this improved scaling. While such a large number of cores is usually not available unless one uses a supercomputer, adapting the RI-CCSD(T) program to a hybrid CPU/GPU architecture can effectively overcome this barrier. Thus, from the scaling perspective as well, algorithm B may be anticipated to be more suitable for taking advantage of the many-folded acceleration provided by the GPUs. For the CPU-only implementation reported in the present work, the use of algorithm A is recommended when sufficient computational resources are available. The applications reported in section 5 employed algorithm A unless otherwise mentioned.

5. APPLICATIONS

5.1. Accuracy Benchmarks. The accuracy of the new RI-CCSD(T) program was tested by computing errors relative to CCSD(T) energies obtained with standard four-center ERIs. For the latter, the available parallel CCSD(T) program[8] in GAMESS was employed. These accuracy benchmarks serve two purposes, namely, they test the correctness of the new implementation and also provide an assessment of the error committed by employing the RI approximation. Two sets of calculations were performed to benchmark the accuracy: (a) using the CYCONF test set[88] for the relative energies of 11 lowest-energy conformers of cysteine and (b) the ISOL22 test set[89] for the isomerization energies of 22 large organic molecules (containing up to 51 atoms). The aug-cc-pVTZ/aug-cc-pVQZ-RI basis sets were used for the CYCONF test set, while the smaller cc-pVDZ/cc-pVTZ-RI basis sets were employed in calculations involving the ISOL22 test set. Table 4 reports the mean absolute errors (MAEs) and standard deviations of the RI-CCSD(T) energies relative to standard CCSD(T) energies, which are on the order of 10^{-2}−10^{-3} kcal/mol for both test sets. It has been verified that the errors diminish upon a systematic enlargement of the size of the auxiliary basis set.

Table 4. Mean Absolute Errors and Standard Deviations (in kcal/mol) of the RI-CCSD(T) Energies Relative to CCSD(T) Energies Obtained with Standard Four-Center ERIs

test set	MAE	standard deviation
CYCONF	0.005	0.007
ISOL22	0.016	0.024

The accuracy of the new RI-CCSD(T) implementation was further tested by computing the π−π interaction energy of two uracil molecules, a test system from the S66 data set of Hobza and co-workers.[90,91] The availability of accurate benchmark results for the S66 data set obtained with high-level methods such as CC theory[48,90−92] is important for assessing the performance of other computational methods. The uracil dimer is particularly a challenging system, as CC calculations using large quadruple-ζ basis sets are usually intractable. Such large basis results are required for estimating the complete basis set (CBS) limit of the interaction energy. Although explicitly correlated CC calculations[38,48,92] can achieve the CBS limit with triple-ζ quality basis sets, only the CCSD method can benefit from the explicitly correlated treatment. It has been shown that the triples correction should preferably be obtained from conventional CCSD(T) calculations.[93] Composite approaches[90−92] are commonly employed to alleviate the problem due to high computational expenses of CC calculations, where progressively smaller basis sets are used for MP2, CCSD, and CCSD(T) levels of theory.

For the π-stacked uracil dimer, the CBS estimate of the interaction energy is known to be quite sensitive to the choice of the basis set used for CC calculations. For instance, the value of −9.83 kcal/mol was obtained with the aug-cc-pVDZ basis used for the CCSD(T) calculation,[90] and the value of −9.75 kcal/mol was obtained with a two-point extrapolation[94] based on the CCSD(T) results computed with the heavy-augmented (haug) cc-pVXZ (X = D,T) basis sets.[91] The present work reports benchmark results for the interaction energy of the π-stacked uracil dimer obtained at the RI-CCSD and RI-CCSD(T) levels using the aug-cc-pVXZ/aug-cc-pV(X+1)Z-RI (X = T, Q) basis sets for all atoms. In view of the small errors introduced by the RI approximation as seen from Table 4, the current results can be considered reliable estimates for those obtained in conventional CCSD(T) calculations.

Table 5 reports the CBS estimate for the CCSD correlation contribution to the interaction energy. It is interesting to report this result separately, since several different values have been suggested in the literature. A value of −8.50 kcal/mol was estimated from the reported results of Hobza and co-workers.[91] In 2014, Schmitz et al. reported the value of −7.99 kcal/mol[48] based on the local explicitly correlated PNO-CCSD[F12] method, which differs by 0.5 kcal/mol from the previously reported result. Later, Peng et al. reported a value of −8.30 kcal/mol[38] based on explicitly correlated CCSD calculations using aug-cc-pVQZ and cc-pVQZ-F12 basis sets. Clearly, none of these results agree within 0.1 kcal/mol. It is, thus, useful to obtain a CBS estimate for the CCSD correlation contribution to the interaction energy using the conventional CCSD method. The present work fulfills this need. The counterpoise-corrected RI-CCSD/CBS estimate reported in Table 5 was obtained via a two-point extrapolation[94] from the results obtained with aug-cc-pVXZ (X = T, Q) basis sets. The latter two results were obtained directly from the dimer and monomer energies computed at the RI-CCSD/aug-cc-pVXZ level without using a composite approach. The reported RI-CCSD/CBS estimate of −8.01 kcal/mol agrees to within 0.02 kcal/mol with the value reported by Schmitz et al.,[48] while the other two values differ by lager amounts.

Table 5. CCSD Correlation Contribution to the Interaction Energy (in kcal/mol) of the π-Stacked Uracil Dimer

aug-cc-pVTZ	aug-cc-pVQZ	CBS/best estimate	source
−7.693[a]	−7.878[a]	**−8.014**[b]	this work
		−8.495	ref 91
		−8.30 ± 0.02	ref 38
		−7.985	ref 48

[a]Obtained by directly computing the difference between the RI-CCSD correlation contributions to the dimer and the monomer energies for a given basis. All of the current RI-CCSD results include counterpoise correction. [b]The CBS estimate was obtained with two-point extrapolation[94] using the aug-cc-pVXZ (X = T,Q) values.

To obtain the CBS estimate for the total interaction energy, a composite approach similar to that of Hobza and co-workers[90,91] was employed. This approach can be described by the following equation

$$\begin{aligned}\Delta E(\text{RI-CC/CBS}) = {} & \Delta E(\text{HF/aug-cc-pVQZ}) \\ & + \Delta E^{\text{corr}}(\text{RI-MP2/CBS}) \\ & + \Delta E^{\text{corr}}(\text{RI-CC/CBS}) \end{aligned} \tag{47}$$

The notable difference of the present composite approach with that of Hobza and co-workers is that the former employs identical basis sets for the RI-MP2 and RI-CCSD(T) calculations. The RI-MP2/CBS and RI-CC/CBS [where RI-CC is either RI-CCSD or RI-CCSD(T)] estimates were obtained via two-point extrapolation[94] of the correlation contributions calculated with the aug-cc-pVXZ (X = T,Q) basis sets. The HF(aug-cc-pVQZ) contribution was computed to be 0.378 kcal/mol. The total interaction energies are reported in Table 6. The counterpoise-corrected RI-CCSD(T)/CBS estimate for the interaction energy is −9.64 kcal/mol, which differs from the two reference values reported by Hobza and co-workers, namely, −9.83[90] and −9.75[91] kcal/mol by 0.2 and 0.1 kcal/mol, respectively. However, the current RI-CCSD(T)/CBS estimate agrees very well with the value of −9.67 kcal/mol reported by Martin and co-workers,[92] which was obtained with a composite approach based on explicitly correlated CCSD and conventional (T) calculations. The current RI-CCSD/CBS value of −7.64 kcal/mol for the total interaction energy agrees to within 0.07 kcal/mol with the value reported by Schmitz et al.[48]

Table 6. Total Interaction Energy (in kcal/mol) of the π-Stacked Uracil Dimer

high-level method	CBS/best estimate	source
CCSD	**−7.64**[a]	this work
CCSD	−7.92 ± 0.02	ref 38
CCSD	−7.57	ref 48
CCSD(T)	**−9.64**[a]	this work
CCSD(T)	−9.83	ref 90
CCSD(T)	−9.75	ref 91
CCSD(T)	−9.67	ref 92

[a]Obtained using the composite method described in eq 47. All of the current RI-CC results include counterpoise corrections.

Coupled-cluster calculations on the uracil dimer using a quadruple-ζ basis set are challenging. Even with a triple-ζ basis, the CCSD(T) calculation can take up to 2 weeks.[92] Peng et al.[38] recently reported the first explicitly correlated CCSD results on this system using the aug-cc-pVQZ and cc-pVQZ-F12 basis sets. To the best of our knowledge, the present work reports the first benchmark CCSD(T) result on this system using the aug-cc-pVQZ basis set. This is also the largest computation considered in the present work, which involved 1624 AO basis functions and 3519 auxiliary basis functions. The dimer calculation had $N_O = 42$ and $N_V = 1566$, while the monomer calculation using the dimer basis had an even larger number of virtual orbitals, namely, $N_V = 1595$ ($N_O = 21$). For these calculations, it was necessary to employ the memory-economic algorithm B for the RI-CCSD iterations. The dimer calculation required 8 days on 288 cores (16 compute nodes, one MPI rank per node, and 18 OpenMP threads per MPI rank). The (T) correction took only ∼3.3 days and a larger fraction of the time was consumed by the RI-CCSD step. This is a special case for which the PPL group is much more compute-intensive than the (T) correction, since the length of the MPI/OpenMP parallel loop for the PPL group is about 93 times larger than the $i \geq j \geq k$ loops for the (T) correction. Considering the fact that no local orbital approximation was employed in the RI-CC calculations, 8 days of runtime is reasonable. The monomer calculation using the dimer basis required only 1.5 days on 288 cores.

5.2. Model Application. The formation of C−C bonds by functionalizing the otherwise inactive C−H bonds using

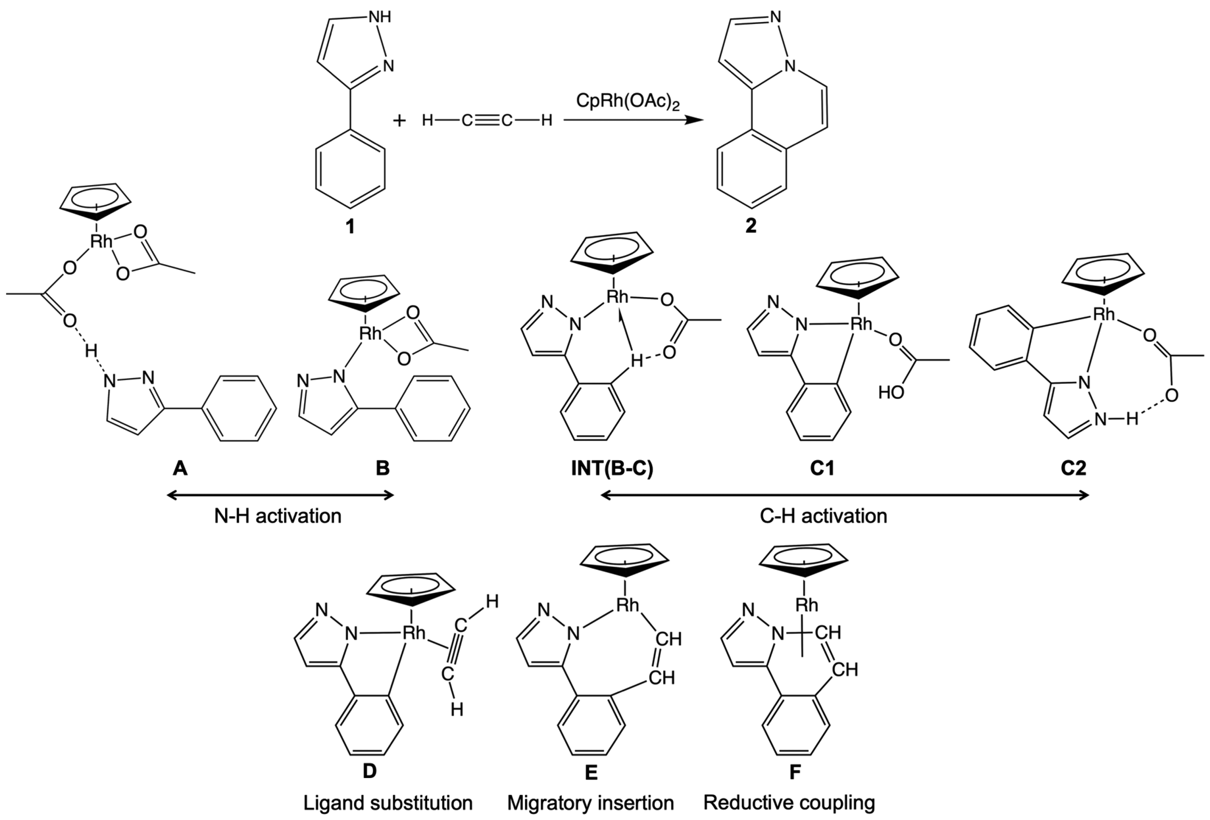


Figure 9. Representative steps on the reaction profile for the Rhodium-catalyzed C−H bond activation of 3-phenylpyrazole (**1**) with acetylene to form pyrazolo[5,1-*a*]isoquinoline (**2**).

Table 7. Gas-Phase Single-Point Energies on the Reaction Profile for the Rhodium-Catalyzed C−H Bond Activation in 3-Phenylpyrazole (1) with Acetylene Computed at the RI-CCSD(T)/def2-TZVPP/def2-TZVPP-RI Level at Geometries Optimized Using the BP86 Functional (from ref 97)

species	number of atoms	number of AO basis functions	N_O	N_V	N_{aux}	$\Delta E_{RI\text{-}CCSD(T)}$ (kcal/mol)	ΔE_{BP86}[a] (kcal/mol)	wall time	number of cores
A	44	1066	71	971	2524	0.0	0.0	4.5 days	144
INT(A-B)	44	1066	71	971	2524	−3.9	2.0	5 days	144
B	36	886	59	807	2100	13.7	19.6	38 h	144
TS(B-C)1	36	886	59	807	2100	28.0	33.1	52 h	144
INT(B−C)	36	886	59	807	2100	30.9	30.1	34 h	144
TS(B-C)2	36	886	59	807	2100	31.6	30.1	36 h	144
C1	36	886	59	807	2100	14.5	20.7	36 h	144
TS(C1-C2)	36	886	59	807	2100	13.9	21.8	35 h	144
C2	36	886	59	807	2100	3.6	9.6	38 h	144
D	32	796	52	726	1888	11.6	14.6	26 h	108
E	32	796	52	726	1888	−0.6	−6.0	25 h	108
F	32	796	52	726	1888	−30.6	−26.3	25 h	108

[a]BP86 energies are from ref 97.

transition metal catalysts[95,96] is an important and widely used synthetic route in organic chemistry. Accurate theoretical calculations are important for elucidating the mechanisms of the catalytic processes and characterizing various intermediates. Since the relative stabilities of various intermediates may strongly vary depending on the choice of substrates, it is often important to include reactants, catalysts, etc., used in the actual synthetic cycle into the computational models (see, e.g., ref 97), thereby making them computationally expensive. Semi-empirical and DFT methods have been the popular choices for such computational studies, while computational models based on coupled-cluster theory have been almost nonexistent until recently.[98]

To assess the applicability of the reported RI-CCSD(T) implementation in modeling organometallic reactions, single-point energy calculations were performed on selected points on the reaction profile for the C−H bond activation in 3-phenylpyrazole (**1**) with acetylene catalyzed by CpRh(OAc)$_2$ (see Figure 9). Algarra et al. reported a computational study[97] on this reaction by employing the BP86 functional[99] for optimizing geometries of all species and also for computing single-point energies. Table 7 reports the gas-phase single-point energies obtained at the RI-CCSD(T)/def2-TZVPP/

def2-TZVPP-RI level of theory computed at the BP86 optimized geometries reported in ref 97 along with the wall times required for full RI-CCSD(T) calculations. The Stuttgart/Dresden effective core potential SD(28, MWB)[100] was used for Rh. The primary focus of this study is to test the viability of the RI-CCSD(T) calculation in combination with a fairly large polarized triple-ζ valence basis set to model this reaction profile.

As outlined in ref 97, the reaction proceeds through the hydrogen-bonded adduct **A**. The first step is N−H activation, which involves Rh−N bond formation leading to **INT(A-B)** and the subsequent N−H bond cleavage induced by a leaving acetate group from the CpRh(OAc)$_2$ catalyst. N−H activation leads to the intermediate **B**, which serves as the primary substrate for C−H activation. C−H activation occurs through the intermediate **INT(B−C)**, which is formed via an agostic interaction[101] between Rh and the *ortho*-C−H bond of the phenyl group in 3-phenylpyrazole (**1**). The product **C1** of the C−H activation step rearranges to the more stable form **C2**. The subsequent ligand substitution (AcOH/C$_2$H$_2$) then forms **D**, which undergoes migratory insertion leading to **E**. The intermediate **E** then undergoes C−N reductive coupling and leads to **F**, in which the final product, pyrazolo[5,1-*a*]isoquinoline (**2**), is bound to the Rh center as a tetrahapto ligand.

Table 7 indicates that despite the fairly large size of the systems, these calculations can be accomplished within a reasonable amount of time. The H-bonded adduct **A** and the intermediate **INT(A-B)** have the largest size, namely, 44 atoms, and the calculations involve more than 1000 basis functions. The RI-CCSD(T) calculations on these two systems took ∼5 days, while for the other relatively smaller systems the calculations took 1−2 days. Note that only a moderate number of compute cores (less than 150) were employed in all calculations. Table 7 shows that certain calculations involving identical system size required different amounts of wall time. This is caused by varying numbers of CCSD iterations required for convergence. These results demonstrate that applications involving reasonably large organometallic systems can be performed with the reported GAMESS implementation of RI-CCSD(T) and that the calculations can be done within a reasonable amount of time even on small to moderate-sized computer clusters. It is also apparent that the DFT/BP86 relative energies are in remarkably good agreement with the much more reliable RI-CCSD(T) values except for a few species.

6. CONCLUSIONS AND OUTLOOK

A massively parallel CCSD(T) algorithm has been described and implemented within the GAMESS program suite using the resolution-of-the-identity (RI) approximation for the ERIs. The implementation uses the MPI-based distributed memory model for internode parallelism and the OpenMP-based shared memory model for intranode parallelism on multiprocessors nodes. The primary features of the reported RI-CCSD(T) algorithm include (a) the use of an integral-direct strategy to reduce memory bottlenecks, (b) the use of a set of amplitude-dressed three-center intermediates to cast the RI-CCSD amplitude equations in a compact quasi-linear form, (c) the use of permutational symmetry of the doubles amplitude and residual matrices to reduce storage costs and disk I/O, and (d) the use of permutational symmetries of the ERIs and the doubles amplitudes to eliminate all memory bandwidth-bound index permutations of triples amplitudes in the (T) correction step. The distribution of the computational workload for various terms in the RI-CCSD amplitude equations and the (T) correction has been optimized in such a way as to minimize network communications between MPI ranks.

Two different algorithms have been developed for the computationally most expensive PPL and the $O(N_O^3N_V^3)$ groups. The more memory-expensive algorithm A assembles four-center ERIs requiring N_V^4 and $N_O^2N_V^2$ -scaling memory costs in blocks on a number of MPI ranks. The use of this algorithm in large-scale applications requires employing an appropriate number of MPI ranks or compute nodes to scale down the associated quartic-scaling memory demand. The alternative algorithm B follows the strategy of storing at most cubic memory-scaling ERIs or intermediates and is, therefore, highly memory-economic. The memory-economic feature of this algorithm is achieved at the expense of reduced vectorization of the tensor contraction involved in the construction of the PPL group. It has been demonstrated that algorithm B is capable of making large-scale computations involving up to ∼1000 atomic basis functions feasible on standard computer clusters within a reasonable amount of time. Hence, algorithm B is expected to be more suitable for use by the general computational chemistry community.

The reported RI-CCSD(T) algorithm shows an excellent near-linear speedup, in particular for the (T) correction, in the range of both a few hundred cores and a few thousand cores. Although the overall RI-CCSD iteration does not scale as well as the (T) correction, the rate-limiting PPL group and the $O(N_O^3N_V^3)$ group have been shown to scale reasonably well. The algorithm B for the PPL group has been demonstrated to show a better strong scaling than algorithm A in the limit of a few thousand cores. The relatively poor strong scaling of the RI-CCSD iteration arises from the fact that the low-cost terms in the RI-CCSD equations, e.g., the HHL group, do not effectively speed up as the number of cores is increased. This is because several compute cores remain idle during their computation. However, the parallel scaling of the RI-CCSD iteration, as well as that of the HHL group, improves if the molecular size is increased proportionately to the increase in the number of cores, that is, the algorithm shows good weak scaling.

Benchmark calculations have been reported to assess the accuracy of the CC energies obtained with the RI-CCSD(T) program vis-à-vis the CCSD(T) energies based on standard four-center ERIs. Several large-scale applications have been reported on molecules consisting of 24−51 atoms described by 500−1000 atomic basis functions. This work reports the first calculation of the interaction energy of the π -stacked uracil dimer from the S66 benchmark set at the CCSD(T) level using the aug-cc-pVQZ/aug-cc-pV5Z-RI basis sets. Of particular interest is the CCSD correlation contribution to the interaction energy, for which varying values have been reported in the literature.[91,48,38] The CBS estimate calculated in this work via two-point extrapolation of the results obtained with aug-cc-pVXZ (X = T,Q) basis sets is −8.01 kcal/mol, which agrees very well with the value of −7.99 kcal/mol reported by Schmitz et al.[48] This is the largest calculation considered in this work, which employed 1624 atomic basis functions and took 8 days using algorithm B.

To extend the applicability of the reported RI-CCSD(T) program in GAMESS and also to enhance its efficiency, two different routes will be explored in the future. The first will be to combine the RI-CCSD(T) algorithm with the fragment molecular orbital (FMO) or the effective FMO (EFMO) implementations available in GAMESS to develop a multilevel approach for extended systems, wherein the actives sites will be treated with the high-level RI-CCSD(T) method and the larger environments will be treated either with the FMO approach or with the RI-MP2 approach, all within the fragmentation *ansatz*.[72] Second, the current CPU-only RI-CCSD(T) algorithm will be adapted to a hybrid CPU/GPU architecture. Work is underway to adapt the more compute-intensive steps, namely, the (T) correction and the evaluation of the PPL group in the RI-CCSD iteration. The initial analysis provided in this work indicates that algorithm B would be more suitable for GPU adaptation, both because of its low memory demand and also because of its superior strong scaling relative to algorithm A in the limit of a large number of cores.

AUTHOR INFORMATION

Corresponding Author

Mark S. Gordon − *Department of Chemistry and Ames Laboratory, Iowa State University, Ames 50011-2416 Iowa, United States of America;* orcid.org/0000-0001-6893-553X; Email: mark@si.msg.chem.iastate.edu

Author

Dipayan Datta − *Department of Chemistry and Ames Laboratory, Iowa State University, Ames 50011-2416 Iowa, United States of America;* orcid.org/0000-0003-0824-0837

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jctc.1c00389

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was supported in part by a Computational Chemical Sciences grant from the US Department of Energy, Office of Science, Basic Energy Sciences, Division of Chemical Sciences, Geosciences, and Biological Sciences. The work was performed at Ames Laboratory, which is operated by Iowa State University under contract No. DE-AC02-07CH11338. The computer cluster cj was provided by the Ames Laboratory. The work was also supported in part by a National Science Foundation software infrastructure SI2 grant, OCI-1047772. The authors gratefully acknowledge the Argonne Leadership Computing Facility at the Argonne National Laboratory for providing CPU time and technical support. D.D. thanks Dr. Robert J. Grandin of the department of IT services at Iowa State University for his valuable assistance in preparing the HPC cluster Nova to host large-scale coupled-cluster calculations.

REFERENCES

(1) 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.

(2) Bartlett, R. J.; Musiał, M. Coupled-Cluster Theory in Quantum Chemistry. *Rev. Mod. Phys.* **2007**, *79*, 291−352.

(3) Crawford, T. D.; Schaefer III, H. F. An Introduction to Coupled Cluster Theory for Computational Chemists. *Rev. Comp. Chem.* **2007**, *14*, 33−136.

(4) Raghavachari, K.; Trucks, G. W.; Pople, J. A.; Head-Gordon, M. A Fifth-Order Perturbation Comparison of Electron Correlation Theories. *Chem. Phys. Lett.* **1989**, *157*, 479−483.

(5) Bartlett, R. J.; Watts, J. D.; Kucharski, S. A.; Noga, J. Non-Iterative Fifth-Order Triple and Quadruple Excitation Energy Corrections in Correlated Methods. *Chem. Phys. Lett.* **1990**, *165*, 513−522.

(6) Kobayashi, R.; Rendell, A. P. A Direct Coupled Cluster Algorithm for Massively Parallel Computers. *Chem. Phys. Lett.* **1997**, *265*, 1−11.

(7) Anisimov, V. M.; Bauer, G. H.; Chadalavada, K.; Olson, R. M.; Glenski, J. W.; Kramer, W. T. C.; Aprà, A.; Kowalski, K. Optimization of the Coupled Cluster Implementation in NWChem on Petascale Parallel Architectures. *J. Chem. Theory Comput.* **2014**, *10*, 4307−4316.

(8) Olson, R. M.; Bentz, J. L.; Kendall, R. A.; Schmidt, M. W.; Gordon, M. S. A Novel Approach to Parallel Coupled Cluster Calculations: Combining Distributed and Shared Memory Techniques for Modern Cluster Based Systems. *J. Chem. Theory Comput.* **2007**, *3*, 1312−1328.

(9) Janowski, T.; Ford, A. R.; Pulay, P. Parallel Calculation of Coupled Cluster Singles and Doubles Wave Functions Using Array Files. *J. Chem. Theory Comput.* **2007**, *3*, 1368−1377.

(10) Janowski, T.; Pulay, P. Efficient Parallel Implementation of the CCSD External Exchange Operator and the Perturbative Triples (T) Energy Calculation. *J. Chem. Theory Comput.* **2008**, *4*, 1585−1592.

(11) Deumens, E.; Lotrich, V. F.; Perera, A.; Ponton, M. J.; Sanders, B. A.; Bartlett, R. J. Software Design of ACES III with the Super Instruction Architecture. *WIREs Comput. Mol. Sci.* **2011**, *1*, 895−901.

(12) Solomonik, E.; Matthews, D.; Hammond, J. R.; Stanton, J. F.; Demmel, J. A Massively Parallel Tensor Contraction Framework for Coupled-Cluster Computations. *J. Parallel Distrib. Comput.* **2014**, *74*, 3176−3190.

(13) Ford, A. R.; Janowski, T.; Pulay, P. Array Files for Computational Chemistry: MP2 Energies. *J. Comput. Chem.* **2007**, *28*, 1215−1220.

(14) Nieplocha, J.; Palmer, B.; Tipparaju, V.; Krishnan, M.; Trease, H.; Aprà, E. Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit. *Int. J. High Perf. Comput. Appl.* **2006**, *20*, 203−231.

(15) Fletcher, G. D.; Schmidt, M. W.; Bode, B. M.; Gordon, M. S. The Distributed Data Interface in GAMESS. *Comput. Phys. Commun.* **2000**, *128*, 190−200.

(16) Olson, R. M.; Schmidt, M. W.; Gordon, M. S.; Rendell, A. P. Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Approach. *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*; Phoenix, AZ, 2003; p 41.

(17) Yoo, S.; Aprà, E.; Zeng, X. C.; Xantheas, S. S. High-Level Ab Initio Electronic Structure Calculations of Water Clusters (H2O)16 and (H2O)17: A New Global Minimum for (H2O)16. *J. Phys. Chem. Lett.* **2010**, *1*, 3122−3127.

(18) Benedikt, U.; Auer, A. A.; Espig, M.; Hackbusch, W. Tensor Decomposition in Post-Hartree−Fock Methods. I. Two-Electron Integrals and MP2. *J. Chem. Phys.* **2011**, *134*, 054118.

(19) Benedikt, U.; Böhm, K.-H.; Auer, A. A. Tensor Decomposition in Post-Hartree−Fock Methods. II. CCD Implementation. *J. Chem. Phys.* **2013**, *139*, 224101.

(20) Kinoshita, T.; Hino, O.; Bartlett, R. J. Singular Value Decomposition Approach for the Approximate Coupled-Cluster Method. *J. Chem. Phys.* **2003**, *119*, 7756−7762.

(21) Hohenstein, E. G.; Parrish, R. M.; Martínez, T. J. Tensor Hypercontraction Density Fitting. I. Quartic Scaling Second-and Third-Order Møller-Plesset Perturbation Theory. *J. Chem. Phys.* **2012**, *137*, 044103.

(22) Parrish, R. M.; Sherrill, C. D.; Hohenstein, E. G.; Kokkila, S. I. L.; Martínez, T. J. Communication: Acceleration of Coupled Cluster

Singles and Doubles via Orbital-Weighted Least-Squares Tensor Hypercontraction. *J. Chem. Phys.* **2014**, *140*, 181102.
(23) Parrish, R. M.; Zhao, Y.; Hohenstein, E. G.; Martínez, T. J. Rank Reduced Coupled Cluster Theory. I. Ground State Energies and Wavefunctions. *J. Chem. Phys.* **2019**, *150*, 164118.
(24) Schutski, R.; Zhao, J.; Henderson, T. M.; Scuseria, G. E. Tensor-Structured Coupled Cluster Theory. *J. Chem. Phys.* **2017**, *147*, 184113.
(25) Whitten, J. L. Coulombic Potential Energy Integrals and Approximations. *J. Chem. Phys.* **1973**, *58*, 4496−4501.
(26) Feyereisen, M.; Fitzgerald, G.; Komornicki, A. Use of Approximate Integrals in Ab Initio Theory. An Application in MP2 Energy Calculations. *Chem. Phys. Lett.* **1993**, *208*, 359−363.
(27) Vahtras, O.; Almlöf, J.; Feyereisen, M. W. Integral Approximations for LCAO-SCF Calculations. *Chem. Phys. Lett.* **1993**, *213*, 514−518.
(28) Weigend, F. A Fully Direct RI-HF Algorithm: Implementation, Optimised Auxiliary Basis Sets, Demonstration of Accuracy and Efficiency. *Phys. Chem. Chem. Phys.* **2002**, *4*, 4285−4291.
(29) Sodt, A.; Subotnik, J. E.; Head-Gordon, M. Linear Scaling Density Fitting. *J. Chem. Phys.* **2006**, *125*, 194109.
(30) Beebe, N. H. F.; Linderberg, J. Simplifications in the Generation and Transformation of Two-Electron Integrals in Molecular Calculations. *Int. J. Quantum Chem.* **1977**, *12*, 683−705.
(31) 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.
(32) Aquilante, F.; Lindh, R.; Bondo Pedersen, T. Unbiased Auxiliary Basis Sets for Accurate Two-Electron Integral Approximations. *J. Chem. Phys.* **2007**, *127*, 114107.
(33) Pedersen, T. B.; Aquilante, F.; Lindh, R. Density Fitting with Auxiliary Basis Sets from Cholesky Decompositions. *Theor. Chem. Acc.* **2009**, *124*, 1−10.
(34) Rendell, A. P.; Lee, T. J. Coupled-Cluster Theory Employing Approximate Integrals: An Approach to Avoid the Input/Output and Storage Bottlenecks. *J. Chem. Phys.* **1994**, *101*, 400−408.
(35) Hättig, C.; Weigend, F. CC2 Excitation Energy Calculations on Large Molecules Using the Resolution of the Identity Approximation. *J. Chem. Phys.* **2000**, *113*, 5154−5161.
(36) Epifanovsky, E.; Zuev, D.; Feng, X.; Khistyaev, K.; Shao, Y.; Krylov, A. I. General Implementation of the Resolution-of-the-Identity and Cholesky Representations of Electron Repulsion Integrals Within Coupled-Cluster and Equation-of-Motion Methods: Theory and Benchmarks. *J. Chem. Phys.* **2013**, *139*, 134105.
(37) DePrince, A. E., III; Sherrill, C. D. Accuracy and Efficiency of Coupled-Cluster Theory Using Density Fitting/Cholesky Decomposition, Frozen Natural Orbitals, and a t1-Transformed Hamiltonian. *J. Chem. Theory Comput.* **2013**, *9*, 2687−2696.
(38) Peng, C.; Calvin, J. A.; Pavošević, F.; Zhang, J.; Valeev, E. F. Massively Parallel Implementation of Explicitly Correlated Coupled Cluster Singles and Doubles Using TiledArray Framework. *J. Phys. Chem. A* **2016**, *120*, 10231−10244.
(39) Peng, C.; Calvin, J. A.; Valeev, E. F. Coupled-Cluster Singles, Doubles and Perturbative Triples with Density Fitting Approximation for Massively Parallel Heterogeneous Platforms. *Int. J. Quantum Chem.* **2019**, *119*, No. e25894.
(40) Shen, T.; Zhu, Z.; Zhang, I. Y.; Scheffler, M. Massive-parallel Implementation of the Resolution-of-Identity Coupled-cluster Approaches in the Numeric Atom-centered Orbital Framework for Molecular Systems. *J. Chem. Theory Comput.* **2019**, *15*, 4721−4734.
(41) Gyevi-Nagy, L.; Kállay, M.; Nagy, P. R. Integral-Direct and Parallel Implementation of the CCSD(T) Method: Algorithmic Developments and Large-Scale Applications. *J. Chem. Theory Comput.* **2020**, *16*, 366−384.
(42) Pulay, P. Localizability of Dynamic Electron Correlation. *Chem. Phys. Lett.* **1983**, *100*, 151−154.
(43) Saebø, S.; Pulay, P. Local Configuration Interaction: An Efficient Approach for Larger Molecules. *Chem. Phys. Lett.* **1985**, *113*, 13−18.
(44) Boughton, J. W.; Pulay, P. Comparison of the Boys and Pipek−Mezey Localizations in the Local Correlation Approach and Automatic Virtual Basis Selection. *J. Comput. Chem.* **1993**, *14*, 736−740.
(45) Hampel, C.; Werner, H.-J. Local Treatment of Electron Correlation in Coupled Cluster Theory. *J. Chem. Phys.* **1996**, *104*, 6286−6297.
(46) Schütz, M.; Werner, H.-J. Local Perturbative Triples Correction (T) with Linear Cost Scaling. *Chem. Phys. Lett.* **2000**, *318*, 370−378.
(47) Schwilk, M.; Ma, Q.; Köppl, C.; Werner, H.-J. Scalable Electron Correlation Methods. 3. Efficient and Accurate Parallel Local Coupled Cluster with Pair Natural Orbitals (PNO-LCCSD). *J. Chem. Theory Comput.* **2017**, *13*, 3650−3675.
(48) Schmitz, G.; Hättig, C.; Tew, D. P. Explicitly Correlated PNO-MP2 and PNO-CCSD and Their Application to the S66 Set and Large Molecular Systems. *Phys. Chem. Chem. Phys.* **2014**, *16*, 22167−22178.
(49) Scuseria, G. E.; Ayala, P. Y. Linear Scaling Coupled Cluster and Perturbation Theories in the Atomic Orbital Basis. *J. Chem. Phys.* **1999**, *111*, 8330−8343.
(50) Maslen, P. E.; Dutoi, A. D.; Lee, M. S.; Shao, Y.; Head-Gordon, M. Accurate Local Approximations to the Triples Correlation Energy: Formulation, Implementation and Tests of 5th-order Scaling Models. *Mol. Phys.* **2005**, *103*, 425−437.
(51) Li, S.; Ma, J.; Jiang, Y. Linear Scaling Local Correlation Approach for Solving the Coupled Cluster Equations of Large Systems. *J. Comput. Chem.* **2002**, *23*, 237−244.
(52) Li, W.; Piecuch, P.; Gour, J. R.; Li, S. Local Correlation Calculations Using Standard and Renormalized Coupled-Cluster Approaches. *J. Chem. Phys.* **2009**, *131*, 114109.
(53) Yang, J.; Chan, G. K. L.; Manby, F. R.; Schütz, M.; Werner, H.-J. The Orbital-Specific-Virtual Local Coupled Cluster Singles and Doubles Method. *J. Chem. Phys.* **2012**, *136*, 144105.
(54) Nagy, P. R.; Kállay, M. Optimization of the Linear-Scaling Local Natural Orbital CCSD(T) Method: Redundancy-Free Triples Correction Using Laplace Transform. *J. Chem. Phys.* **2017**, *146*, 214106.
(55) Nagy, P. R.; Kállay, M. Approaching the Basis Set Limit of CCSD(T) Energies for Large Molecules with Local Natural Orbital Coupled-Cluster Methods. *J. Chem. Theory Comput.* **2019**, *15*, 5275−5298.
(56) Riplinger, C.; Neese, F. An Efficient and Near Linear Scaling Pair Natural Orbital Based Local Coupled Cluster Method. *J. Chem. Phys.* **2013**, *138*, 034106.
(57) Riplinger, C.; Sandhöfer, B.; Hansen, A.; Neese, F. Natural Triple Excitations in Local Coupled Cluster Calculations with Pair Natural Orbitals. *J. Chem. Phys.* **2013**, *139*, 134101.
(58) Guo, Y.; Riplinger, C.; Becker, U.; Liakos, D. G.; Minenkov, Y.; Cavallo, L.; Neese, F. Communication: An Improved Linear Scaling Perturbative Triples Correction for the Domain Based Local Pair-Natural Orbital Based Singles and Doubles Coupled Cluster Method [DLPNO-CCSD(T)]. *J. Chem. Phys.* **2018**, *148*, 011101.
(59) Kitaura, K.; Sawai, T.; Asada, T.; Nakano, T.; Uebayasi, M. Pair Interaction Molecular Orbital Method: An Approximate Computational Method for Molecular Interactions. *Chem. Phys. Lett.* **1999**, *312*, 319−324.
(60) Kitaura, K.; Ikeo, E.; Asada, T.; Nakano, T.; Uebayasi, M. Fragment Molecular Orbital Method: An Approximate Computational Method for Large Molecules. *Chem. Phys. Lett.* **1999**, *313*, 701−706.
(61) Nakano, T.; Kaminuma, T.; Sato, T.; Akiyama, Y.; Uebayasi, M.; Kitaura, K. *Chem. Phys. Lett.* **2000**, *318*, 614−618.
(62) Nakano, T.; Kaminuma, T.; Sato, T.; Fukuzawa, K.; Akiyama, Y.; Uebayasi, M.; Kitaura, K. Fragment Molecular Orbital Method: Use of Approximate Electrostatic Potential. *Chem. Phys. Lett.* **2002**, *351*, 475−480.
(63) Fedorov, D. G. The Fragment Molecular Orbital Method: Theoretical Development, Implementation in GAMESS, and

Applications. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2017**, *7*, No. e1322.

(64) Gordon, M. S.; Fedorov, D. G.; Pruitt, S. R.; Slipchenko, L. V. Fragmentation Methods: A Route to Accurate Calculations on Large Systems. *Chem. Rev.* **2012**, *112*, 632−672.

(65) Li, W.; Piecuch, P. Multilevel Extension of the Cluster-in-Molecule Local Correlation Methodology: Merging Coupled-Cluster and Møller-Plesset Perturbation Theories. *J. Phys. Chem. A* **2010**, *114*, 6721−6727.

(66) Piecuch, P.; Włoch, M. Renormalized Coupled-Cluster Methods Exploiting Left Eigenstates of the Similarity-Transformed Hamiltonian. *J. Chem. Phys.* **2005**, *123*, 224105.

(67) Piecuch, P.; Włoch, M.; Gour, J. R.; Kinal, A. Single-Reference, Size-Extensive, Non-Iterative Coupled-Cluster Approaches to Bond Breaking and Biradicals. *Chem. Phys. Lett.* **2006**, *418*, 467−470.

(68) Findlater, A. D.; Zahariev, F.; Gordon, M. S. Combined Fragment Molecular Orbital Cluster in Molecule Approach to Massively Parallel Electron Correlation Calculations for Large Systems. *J. Phys. Chem. A* **2015**, *119*, 3587−3593.

(69) Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S. J.; Windus, T. L.; Dupuis, M.; Montgomery, J. J. A. General Atomic and Molecular Electronic Structure System. *J. Comput. Chem.* **1993**, *14*, 1347−1363.

(70) Barca, G. M. J.; Bertoni, C.; Carrington, L.; Datta, D.; De Silva, N.; Deustua, J. E.; Fedorov, D. G.; Gour, J. R.; Gunina, A. O.; Guidez, E.; Harville, T.; Irle, S.; Ivanic, J.; Kowalski, K.; Leang, S. S.; Li, H.; Li, W.; Lutz, J. J.; Magoulas, I.; Mato, J.; Mironov, V.; Nakata, H.; Pham, B. Q.; Piecuch, P.; Poole, D.; Pruitt, S. R.; Rendell, A. P.; Roskop, L. B.; Ruedenberg, K.; Sattasathuchana, T.; Schmidt, M. W.; Shen, J.; Slipchenko, L.; Sosonkina, M.; Sundriyal, V.; Tiwari, A.; Galvez Vallejo, J. L.; Westheimer, B.; Wloch, M.; Xu, P.; Zahariev, F.; Gordon, M. S. Recent Developments in the General Atomic and Molecular Electronic Structure System. *J. Chem. Phys.* **2020**, *152*, 154102.

(71) Fedorov, D. G.; Olson, R. M.; Kitaura, K.; Gordon, M. S.; Koseki, S. A New Hierarchical Parallelization Scheme: Generalized Distributed Data Interface (GDDI), and an Application to the Fragment Molecular Orbital Method (FMO). *J. Comput. Chem.* **2004**, *25*, 872−880.

(72) Pham, B. Q.; Gordon, M. S. Hybrid Distributed/Shared Memory Model for the RI-MP2Method in the Fragment Molecular Orbital Framework. *J. Chem. Theory Comput.* **2019**, *15*, 5252−5258.

(73) Scuseria, G. E.; Janssen, C. L.; Schaefer, H. F., III An Efficient Reformulation of the Closed Shell Coupled Cluster Single and Double Excitation (CCSD) Equations. *J. Chem. Phys.* **1988**, *89*, 7382−7387.

(74) Kállay, M.; Surján, P. R. Higher Excitations in Coupled-Cluster Theory. *J. Chem. Phys.* **2001**, *115*, 2945−2954.

(75) Engels-Putzka, A.; Hanrath, M. A Fully Simultaneously Optimizing Genetic Approach to the Highly Excited Coupled-Cluster Factorization Problem. *J. Chem. Phys.* **2011**, *134*, 124106.

(76) Datta, D.; Gauss, J. A Non-antisymmetric Tensor Contraction Engine for the Automated Implementation of Spin-Adapted Coupled Cluster Approaches. *J. Chem. Theory Comput.* **2013**, *9*, 2639−2653.

(77) Weigend, F.; Häser, M.; Patzelt, H.; Ahlrichs, R. RI-MP2: Optimized Auxiliary Basis Sets and Demonstration of Efficiency. *Chem. Phys. Lett.* **1998**, *294*, 143−152.

(78) Weigend, F.; Köhn, A.; Hättig, C. Efficient use of the Correlation Consistent Basis Sets in Resolution of the Identity MP2 Calculations. *J. Chem. Phys.* **2002**, *116*, 3175−3183.

(79) Hellweg, A.; Hättig, C.; Höfener, S.; Klopper, W. Optimized Accurate Auxiliary Basis Sets for RI-MP2 and RI-CC2 Calculations for the Atoms Rb to Rn. *Theor. Chem. Acc.* **2007**, *117*, 587−597.

(80) Rendell, A. P.; Lee, T. J.; Komornicki, A. A Parallel Vectorized Implementation of Triple Excitations in CCSD(T): Application to the Binding Energies of the AlH3, AlH2F, AlHF2 and AlF3 Dimers. *Chem. Phys. Lett.* **1991**, *178*, 462−470.

(81) Rendell, A. P.; Lee, T. J.; Komornicki, A.; Wilson, S. Evaluation of the Contribution from Triply Excited Intermediates to the Fourth-Order Perturbation Theory Energy on Intel Distributed Memory Supercomputers. *Theor. Chim. Act.* **1993**, *84*, 271−287.

(82) Dunning, 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.

(83) Kendall, R. A.; Dunning, T. H.; Harrison, R. J. Electron Affinities of the First-row Atoms Revisited. Systematic Basis Sets and Wave Functions. *J. Chem. Phys.* **1992**, *96*, 6796−6806.

(84) Weigend, F.; Ahlrichs, R. Balanced Basis Sets of Split Valence, Triple Zeta Valence and Quadruple Zeta Valence Quality for H to Rn: Design and Assessment of Accuracy. *Phys. Chem. Chem. Phys.* **2005**, *7*, 3297−3305.

(85) Blum, V.; Gehrke, R.; Hanke, F.; Havu, P.; Havu, V.; Ren, X.; Reuter, K.; Scheffler, M. Ab Initio Molecular Simulations with Numeric Atom-Centered Orbitals. *Comput. Phys. Commun.* **2009**, *180*, 2175−2196.

(86) Kállay, M.; Nagy, P. R.; Mester, D.; Rolik, Z.; Samu, G.; Csontos, J.; Csóka, J.; Szabó, P. B.; Gyevi-Nagy, L.; Hégely, B.; Ladjánszki, I.; Szegedy, L.; Ladóczki, B.; Petrov, K.; Farkas, M.; Mezei, P. D.; Ganyecz, Á. The MRCC Program System: Accurate Quantum Chemistry from Water to Proteins. *J. Chem. Phys.* **2020**, *152*, 074107.

(87) Gustafson, J. L. Reevaluating Amdahl's Law. *Commun. ACM* **1988**, *31*, 532−533.

(88) Wilke, J. J.; Lind, M. C.; Schaefer III, H. F.; Császár, A. G.; Allen, W. D. Conformers of Gaseous Cysteine. *J. Chem. Theory Comput.* **2009**, *5*, 1511−1523.

(89) Huenerbein, R.; Schirmer, B.; Moellmann, J.; Grimme, S. Effects of London Dispersion on the Isomerization Reactions of Large Organic Molecules: A Density Functional Benchmark Study. *Phys. Chem. Chem. Phys.* **2010**, *12*, 6940−6948.

(90) Řezáč, J.; Riley, K. E.; Hobza, P. S66: A Well-balanced Database of Benchmark Interaction Energies Relevant to Biomolecular Structures. *J. Chem. Theory Comput.* **2011**, *7*, 2427−2438.

(91) Řezáč, J.; Riley, K. E.; Hobza, P. Extensions of the S66 Data Set: More Accurate Interaction Energies and Angular-Displaced Nonequilibrium Geometries. *J. Chem. Theory Comput.* **2011**, *7*, 3466−3470.

(92) Kesharwani, M. K.; Karton, A.; Sylvetsky, N.; Martin, J. M. L. The S66 Non-Covalent Interactions Benchmark Reconsidered Using Explicitly Correlated Methods Near the Basis Set Limit. *Aust. J. Chem.* **2018**, *71*, 238−248.

(93) Sylvetsky, N.; Peterson, K. A.; Karton, A.; Martin, J. M. L. Toward a W4-F12 Approach: Can Explicitly Correlated and Orbital-Based Ab Initio CCSD(T) Limits Be Reconciled? *J. Chem. Phys.* **2016**, *144*, 214101.

(94) Halkier, A.; Helgaker, T.; Jørgensen, P.; Klopper, W.; Koch, H.; Olsen, J.; Wilson, A. K. Basis-set Convergence in Correlated Calculations on Ne, N2, and H2O. *Chem. Phys. Lett.* **1998**, *286*, 243−252.

(95) Shilov, A. E.; Shul'pin, G. B. Activation of C-H Bonds by Metal Complexes. *Chem. Rev.* **1997**, *97*, 2879−2932.

(96) Ackermann, L. Carboxylate-Assisted Transition-Metal-Catalyzed C-H Bond Functionalizations: Mechanism and Scope. *Chem. Rev.* **2011**, *111*, 1315−1345.

(97) Algarra, A. G.; Cross, W. B.; Davies, D. L.; Khamker, Q.; Macgregor, S. A.; McMullin, C. L.; Singh, K. Combined Experimental and Computational Investigations of Rhodium-and Ruthenium-Catalyzed C-H Functionalization of Pyrazoles with Alkynes. *J. Org. Chem.* **2014**, *79*, 1954−1970.

(98) Dohm, S.; Hansen, A.; Steinmetz, M.; Grimme, S.; Checinski, M. P. Comprehensive Thermochemical Benchmark Set of Realistic Closed-Shell Metal Organic Reactions. *J. Chem. Theory Comput.* **2018**, *14*, 2596−2608.

(99) Becke, A. D. Density-Functional Exchange-Energy Approximation with Correct Asymptotic Behavior. *Phys. Rev. A: At., Mol., Opt. Phys.* **1988**, *38*, 3098−3100.

(100) Andrae, D.; Häußermann, U.; Dolg, M.; Stoll, H.; Preuß, H. Energy-Adjusted ab initio Pseudopotentials for the Second and Third Row Transition Elements. *Theor. Chim. Acta* **1990**, *77*, 123−141.
(101) Brookhart, M.; Green, M. L. H.; Parkin, G. Agostic Interactions in Transition Metal Compounds. *Proc. Natl. Acad. Sci. U. S. A.* **2007**, *104*, 6908−6914.