Local second order Møller-Plesset theory with a single threshold using orthogonal virtual orbitals: A distributed memory implementation

Tianyi Shi,[†] Zhenling Wang,[‡,¶] Abdulrahman Aldossary,[§] Yang Liu,[†] Xiaoye S. Li,[*,†] and Martin Head-Gordon[*,‡,¶]

[†]*Applied Mathematics and Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA*

[‡]*Department of Chemistry, University of California, Berkeley, California 94720, USA*

[¶]*Chemical Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA*

[§]*Department of Chemistry, University of Toronto, Toronto, ON M5S 1A1, Canada*

E-mail: xsli@lbl.gov; mhg@cchem.berkeley.edu

Abstract

In order to alleviate the computational burden associated with superlinear compute scalings with molecular size in electron correlation methods, researchers have developed local correlation methods that wisely treat relatively small contributions as zeros but still yield accurate energy approximation. Such local correlation techniques can also be combined with parallel computing resources to obtain further efficiency and scalability. This work focuses on the distributed memory parallel implementation of a local

correlation method for second order Møller-Plesset (MP2) theory. This method also only has a single threshold to control the dropping of terms, and accuracy of different computing kernels in the algorithm. The process partitioning strategy and distributed parallel implementation with message passing interface (MPI) are discussed. In particular, the algorithm relies on a fixed sparsity pattern matrix multiplication and a corresponding distributed conjugate gradient solver, which exhibits almost linear scaling in both strong and weak scaling analysis. Numerical experiments on a range of molecules, including linear chains and molecules with 2 and 3-dimensional character, are reported. For example, with only 32 MPI ranks, this MP2 implementation can calculate the correlation energy of vancomycin in def2-TZVP basis within 0.003% accuracy ($10^{-6.5}$ threshold) in half an hour, where the same problem is unfeasible to solve with sequential or pure shared memory implementations.

1 Introduction

Computational quantum chemistry calculations have become an indispensable "third leg" of chemical research, joining theory and experiment in diverse studies of molecular structure, properties, and reactivity. The most widely used quantum chemistry methods are those based on Kohn-Sham (KS) density functional theory (DFT),[1] which provides an in-principle exact framework to describe molecules in their electronic ground states. In practice, approximate density functionals are necessary,[2] and the most accurate functionals today[3,4] are those on the fifth, or highest rung of the Jacob's Ladder classification.[5] Fifth rung functionals can be grouped into three types,[6] each of which involves the evaluation of an orbital-dependent correlation energy correction to the dressed mean-field treatment typical of rungs four and lower. Since both exchange and correlation are hybrids between wavefunction theory and DFT, they are often called double hybrid (DH) functionals.[4]

The DH orbital correlation term is usually of the second order Rayleigh-Schrödinger per-

turbation theory form, which reduces to the well-known second-order Møller-Plesset (MP2) theory,[7] when used with a mean-field Hartree-Fock reference, rather than a KS reference. MP2 is still widely used in its own right because it is free of the self-interaction errors (SIE) and self-correlation errors that sometimes cause poor performance in density functionals.[8] Examples include intermolecular interactions involving ions,,[9,10] ionic liquids,[11,12] short-range amino acid contacts,[13] as well as accurate treatment of hydrogen bonding.[14–16] Additionally it is worth mentioning that modifications to MP2 have been introduced to improve its accuracy, beginning with spin-component scaling,[17,18] as well as attenuation of long-range contributions.[19,20] More recently regularization of the potentially divergent contributions associated with small occupied-virtual energy gaps has attracted attention.[21–24] However, it is the rung 5 functionals that are the most successful modification.[25–27]

From a computational point of view, MP2 (by which we mean both conventional MP2 and the corresponding PT2 of rung five density functionals) is the simplest and the most economical model for wavefunction-based electron correlation. The computational scaling rises as $\mathcal{O}(M^5)$ with molecule size, M, if MP2 is implemented with canonical molecular orbitals (MOs) that diagonalize the Fock operator, but are delocalized over the entire molecule. Use of the resolution of the identity (RI) approximation[28–31] replaces 4-center two-electron repulsion integrals (ERIs) by linear combinations of 3- and 2-center ERIs, and means that the storage required for MP2 theory rises only as $\mathcal{O}(M^3)$. By comparison, mean-field theories require $\mathcal{O}(M^3)$ computation asymptotically and $\mathcal{O}(M^2)$ storage, so the MP2 step dominates for large molecules.

Physically, there is no reason why MP2 compute costs should rise $\mathcal{O}(M^5)$ with M. Electrons are short-sighted,[32] and long-range correlations (dispersion forces) decay as R^{-6} or faster.[33] Computationally, it is just a consequence of delocalized MOs, and therefore for molecules with HOMO-LUMO gaps, it is possible to spatially localize the MOs before the MP2 procedure. It is then a matter of identifying the subset of significant amplitudes that

contribute to the correlation energy. That is the central challenge of local correlation theory which began with the work of Pulay and Saebø.[34,35] While a rich variety of local correlation methods have since been proposed,[36–38] it is methods descended from the Pulay-Saebø approach,[39] combined with the use of pair natural orbitals (PNOs) and the RI approximation,[40] that have seen widest use to date. The resulting PNO-LMP2 methods[41,42] achieve linear scaling for gapped systems once the physical dimensions significantly exceed the size of localized orbitals. This clearly depends directly on the target accuracy. While we cannot review such methods in detail, a variety of alternative methods with complementary strengths have also been developed in recent years[43–46]

We recently proposed a promising approach to local MP2 (LMP2) that has several distinctive features.[47,48] First, the algorithm employs only a single threshold to control accuracy and compute cost. This makes it relatively straightforward for a user to select the optimum choice for a given application: as we demonstrated higher precision is required for problems that involve delicate global energy differences, which may be wasted in other contexts such as a potential energy surface scan. Second, the algorithm avoids the use of projected atomic orbitals (AOs) and PNOs for the virtual space and instead uses a single set of valence-virtual / hard-virtual (VV-HV) orthogonal orbitals (based on earlier work[49]). This simplifies implementation, and, intriguingly, leads to lower memory demand.[47] The initial implementation that we recently reported employs shared memory parallelism via OpenMP (OMP).[48] In this work, we seek to go beyond that limitation and report an optimized distributed memory implementation.

The motivation for doing so is that improvements in compute power over the past decade or more have come almost entirely from increased parallelism, as limits on heat dissipation have prevented growth in clock frequencies. Parallelism has developed at three broad levels: first, the resources available to a given processor, and the number of instructions it can issue per cycle; second, at the shared memory parallel level, and third at the distributed

memory parallel level. Only by exploiting all three levels of parallelism can a program exploit the distributed memory resources to expand the size of problem that can be solved, and the distributed compute resources that can reduce time to solution. Furthermore, a linear scaling algorithm is the ideal candidate for parallelism. If a reference single node job, which approaches the memory limit controlling maximum job size, has linear scaling, then each doubling of molecular problem size and compute resources in principle enables solution of a problem that *cannot* be solved without the doubling of compute resources. In the ideal case, this weak scaling test would require only constant elapsed time.

There has been considerable previous effort to develop parallel MP2 algorithms, some of which was summarized in a major review on parallel many-body methods,[50] as well as other reviews.[7] Early efforts focused on traditional canonical (i.e. non-local) MP2[51,52] with a focus on RI-MP2.[53–56] An implementation of the K-computer demonstrated performance into the petaflop regime,[57,58] and an implementation on the Summit machine has been recently reported,[59] using a fragment MO approach. As fragment approaches consistent with a set of dense MP2 calculations, they are relatively attractive targets for parallelization.[60–63] Other fragment-like approaches include the MP2 version of the divide-expand-consolidate (DEC) approach[64] which has been implemented in massively parallel fashion.,[65–67] as well as the cluster-in-molecule (CIM) approach.[68–71] Local correlation methods present a greater challenge for parallelization because they are relatively unstructured calculations rather than a structured assembly of multiple dense MP2 calculations. Nevertheless, a medium-scale parallel implementation has been reported for the PNO-LMP2 method.[42] DLPNO calculations at the double-hybrid DFT level can also be parallelized.[72]

Our main contribution in this paper is to develop a distributed memory parallel implementation of the single threshold MP2 using message passing interface (MPI). We treat the local MP2 algorithm as three sub-problems: 1) construction of local density fitting, 2) generation of the ERI tensor, and 3) solve of the amplitude tensor. We then design process

partitioning schemes and parallel strategies separately for these three steps. In addition, we use a novel linear algebra formulation to express the problem, so that understanding and development of this parallel sparsity-enforced algorithm become simpler. In numerical experiments with both 1D and higher-dimensional molecules, our distributed parallel MP2 implementation showcases good scalability. Particularly, the solver step exhibits almost linear scaling in weak scaling tests on linear alkanes with up to 120 used MPI ranks. Furthermore, with larger memory resources, our implementation extends the capabilities of the MP2 method, which now can calculate correlation of molecules that are previously considered infeasible for sequential or pure shared memory algorithms, such as linear alkane $C_{600}H_{1202}$ and vancomycin in def2-TZVP basis with 10^{-7} accuracy. Finally, compared to the previous pure OpenMP implementation, this distributed memory parallel solver can achieve a 50x speedup, especially when 64 to 128 MPI ranks are involved.

The outline of this paper is as follows. We start with a mathematical formulation of the MP2 method and our sparsity-enforced algorithm in section 2. Then, we describe our distributed memory implementation in section 3, including a couple of data distribution schemes and a distributed conjugate gradient (CG) method. Next, we present the performance of this parallel algorithm in section 4 through some examples. Finally, in section 5, we summarize the benefits of our algorithm and discuss some potential future directions.

2 Theory

In this section, we first introduce some linear algebra and chemical notation. Then, we formulate the MP2 energy as a result that requires solving a linear system and introduce our solver that maintains a fixed sparsity pattern. Using this solver, we can achieve significant computational and storage efficiency. In both theory and implementation, we limit ourselves to the case of restricted closed shell orbitals, although there is no obstacle, apart from effort,

in generalizing to unrestricted orbitals.

2.1 Notation

Throughout this manuscript, we use n_{occ} and n_{virt} to denote the number of elements in occupied and virtual MO basis respectively. We always use lowercase letters $1 \leq i, j \leq n_{\text{occ}}$ to denote occupied orbitals, and $1 \leq a, b \leq n_{\text{virt}}$ to denote virtual orbitals.

We use capital letters to represent all linear algebra storage formats, including vectors, matrices, and tensors, in this paper. For example, given a dataset of $n_1 n_2$ elements, we can use X to represent these data as a vector of size $n_1 n_2 \times 1$, or a matrix with column major ordering of size $n_1 \times n_2$. We reserve some letters only for MP2 related concepts. Specifically, F_{occ} and F_{virt} are the Fock matrices corresponding to occupied and virtual MOs respectively, and J is used for the ERI matrix computed via double integrals

$$J_{a,i,b,j} = (ia|jb) = \int dr \int dr' \phi_a(r)\phi_b(r')\frac{1}{|r-r'|}\psi_i(r)\psi_j(r'),\ 1 \leq i, j \leq n_{\text{occ}},\ 1 \leq a, b \leq n_{\text{virt}}, \tag{1}$$

where ϕ and ψ are basis functions for virtual and occupied MOs. In addition, we use $K_{a,i,b,j} = (ib|ja)$, and T to represent the wave function amplitudes we aim to obtain in MP2.

For Fock matrices, we use subscripts such as $(F_{\text{occ}})_{i,j}$ and $(F_{\text{virt}})_{a,b}$ to access one specific element of the matrices. Comparatively, the ERI matrix $J \in \mathbb{R}^{n_{\text{occ}}n_{\text{virt}} \times n_{\text{occ}}n_{\text{virt}}}$ has a much larger size, and can be partitioned into submatrices in the form of

$$J = \begin{bmatrix} J_{1,1} & \cdots & J_{1,n_{\text{occ}}} \\ \vdots & & \vdots \\ J_{n_{\text{occ}},1} & \cdots & J_{n_{\text{occ}},n_{\text{occ}}} \end{bmatrix}. \tag{2}$$

In this scenario, the subscripts $J_{i,j} \in \mathbb{R}^{n_{\text{virt}} \times n_{\text{virt}}}$ for $1 \leq i, j \leq n_{\text{occ}}$ are used to denote

submatrices. This partition can be applied to K as well. Because of the symmetry in (1), we see that both J and K are symmetric matrices, and $J_{i,j} = K_{i,j}^T$ for all pairs of i and j.

2.2 MP2 method as a linear system solver

In this paper, we introduce a linear algebra formulation of the MP2 method. Unlike element-wise computations in existing literature, we use matrix representations, which offer a clean formulation and straightforward vectorization schemes in practice. The MP2 method computes the electron correlation energy by taking the inner product between the ERI matrix J and the wave amplitudes T

$$E_{\text{corr}} = J \cdot T, \tag{3}$$

where T is recovered via solving a linear system

$$\Delta T = 2J - K = C. \tag{4}$$

Here, we have

$$\Delta = I \otimes I \otimes I \otimes F_{\text{virt}} + I \otimes I \otimes F_{\text{occ}} \otimes I + I \otimes F_{\text{virt}} \otimes I \otimes I + F_{\text{occ}} \otimes I \otimes I \otimes I, \tag{5}$$

where $\otimes$ denotes the Kronecker product between two matrices. This matrix representation holds generally for any orthogonal orbital basis, whether the basis is canonical, pseudo-canonical, or non-canonical. One can easily verify that these equations match with element-wise computations in the scenario of canonical orbital basis. In fact, if the MO basis is not orthogonal so that we have nontrivial overlap between occupied or virtual MOs, we can simply replace the identity matrices in (5) by overlap matrices. In addition, we note that C is symmetric, and the partition (2) can be applied to C. As a result, we know T is symmetric, and can share the same partitioning. This allows us to design a solver to obtain T block by

block.

We use the method described in ref. 47 to aim for a solution T that has the same sparsity pattern as the right-hand-side C while solving (4). We achieve this via only solving for the values at the desired positions. For example, figure 1 (left) represents a linear system with a right-hand-side vector that has nonzeros at the first and third positions. In order to let the solution have the same sparsity pattern, we manually remove the second and fourth rows and columns in the matrix to obtain a reduced system as indicated in figure 1 (right). This leaves us with a reduced 2×2 dense system to get a sparse solution.

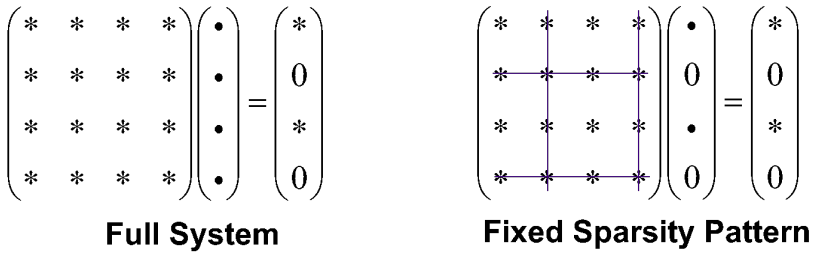

Figure 1: Left: Original linear system with a sparse right-hand-side. Right: A reduced linear system where rows and columns correspond to the zeros are removed, so that the solution has the same sparsity pattern as the right-hand side.

In practice, building Δ in (5) and taking submatrices for the reduced system is both computationally expensive and memory inefficient. Therefore, we use the iterative Krylov subspace method to keep the sparsity pattern unchanged throughout the iterations. In particular, since the Fock matrices F_{occ} and F_{virt} are both symmetric, we use the conjugate gradient (CG) method. This requires us to design a procedure that utilizes the Kronecker structure of Δ to efficiently perform an operation

$$Y = f(\Delta X), \tag{6}$$

where f is a linear operator to enforce Y share the same sparsity pattern as X. In MP2, this indicates that the sparsity pattern of C in (4) is carried over throughout the CG iterations.

2.3 Multiplication with a fixed sparsity pattern

In this section, we describe how to perform (6) efficiently without forming Δ explicitly. This is the basic building block in the CG method and sheds light on data distribution schemes discussed in the following section.

We start by analyzing the multiplication ΔX. In particular, we treat the four terms of Δ in (5) separately. With some manipulations, we find that

$$Y^{(1)} = (I \otimes I \otimes I \otimes F_{\text{virt}})X$$

is equivalent to

$$Y^{(1)}_{i,j} = F_{\text{virt}} X_{i,j}, \tag{7}$$

for $1 \leq i, j \leq n_{\text{occ}}$. Similarly, the multiplication

$$Y^{(3)} = (I \otimes F_{\text{virt}} \otimes I \otimes I)X$$

is equivalent to

$$\left(Y^{(3)}_{i,j}\right)^T = F_{\text{virt}} X^T_{i,j}, \quad 1 \leq i, j \leq n_{\text{occ}}. \tag{8}$$

Both (7) and (8) involve only multiplications with F_{virt}, and can be performed individually on each submatrix of X, which is ideal for parallelism.

Using the same technique to rewrite Kronecker products, we see that

$$Y^{(2)} = (I \otimes I \otimes F_{\text{occ}} \otimes I)X$$

is equivalent to

$$Y^{(2)}_{i,j} = \sum_{k=1}^{n_{\text{occ}}} (F_{\text{occ}})_{i,k} X_{k,j}, \quad 1 \leq i, j \leq n_{\text{occ}}. \tag{9}$$

In other words, each subblock (i, j) of $Y^{(2)}$ is a linear combination of blocks of X on block column j, with coefficients taken from the ith row of F_{occ}. We can also compute all subblocks on the same block column of $Y^{(2)}$ together, and rewrite (9) as

$$\begin{bmatrix} \left(Y^{(2)}_{1,j}\right)^T \\ \vdots \\ \left(Y^{(2)}_{n_{\text{occ}},j}\right)^T \end{bmatrix} = F_{\text{occ}} \begin{bmatrix} (X_{1,j})^T \\ \vdots \\ (X_{n_{\text{occ}},j})^T \end{bmatrix}, \tag{10}$$

where all matrices such as $Y^{(2)}_{1,j}$ and $X_{1,j}$ are now treated as a vector with the same elements, so the two constructed matrices on both sides of the equation have size $n_{\text{occ}} \times n^2_{\text{virt}}$. In this way, the multiplication is performed directly with F_{occ}. Finally, the last multiplication

$$Y^{(4)} = (F_{\text{occ}} \otimes I \otimes I \otimes I)X$$

is equivalent to

$$Y^{(4)}_{i,j} = \sum_{\ell=1}^{n_{\text{occ}}} (F_{\text{occ}})_{\ell,j} X_{i,\ell} = \sum_{\ell=1}^{n_{\text{occ}}} (F_{\text{occ}})_{j,\ell} X^T_{\ell,i} = \left(Y^{(2)}_{j,i}\right)^T, \quad 1 \le i, j \le n_{\text{occ}}, \tag{11}$$

where the second equality holds because both F_{occ} and X are symmetric. In this way, the multiplication result of (11) can be obtained via (10), which helps save computational effort and storage.

Now the large matrix-vector multiplication with Δ in (6) is transformed into smaller matrix-matrix multiplications with the Fock matrices F_{occ} and F_{virt}, we then need to design the linear operator f so that $f\left(Y^{(1)}\right), f\left(Y^{(2)}\right), f\left(Y^{(3)}\right)$, and $f\left(Y^{(4)}\right)$ share the same sparsity pattern as X, and thus the final solution to the reduced system (see Figure 1 (Right)) will share the same sparsity pattern as the right-hand-side. We can reformulate this problem as

finding a linear operator f such that

$$R = f(DS),$$

where D is a dense matrix, S is a sparse matrix, and R has the same sparsity pattern as S. In order to align with how the reduced system is formed in Figure 1, i.e., removing rows and columns corresponding to zeros, we define f separately for each column of S:

$$R_k = f(DS_k) = P^{(k)} D P^{(k)} S_k, \tag{12}$$

where R_k and S_k are the kth column of R and S, and $P^{(k)}$ is a diagonal matrix with

$$P^{(k)}_{p,p} = \begin{cases} 1, & S_{p,k} \neq 0 \\ 0, & \text{otherwise} \end{cases}.$$

For example, consider a simple 4×4 example

$$R = f\left(D \begin{bmatrix} * & & * & \\ * & * & & \\ & * & & \\ & & & * \end{bmatrix} \right),$$

where each '$*$' denotes a nonzero number in the sparse matrix S. Then from (12), we can

represent the operation to obtain the first column of R by

$$\begin{bmatrix} * & & * & \\ * & * & & \\ & * & & \\ & & & * \end{bmatrix} = f\left(\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}\begin{bmatrix} * & & * & \\ * & * & & \\ & * & & \\ & & & * \end{bmatrix}\right),$$

where the elements used and computed are labeled by $*$. Similarly, we can use the same technique to get the other three columns of R. With this multiplication kernel, we ensure that Y and X have the same sparsity pattern in (6), and the multiplication only involves submatrices of F_{occ} and F_{virt}.

3 Implementation

In this section, we present the implementation details on how we use distributed parallel programming to build J by local density fitting and perform the CG iterations. Specifically, we implement our algorithms with MPI,[73] which is a distributed memory framework. Throughout the following sections, we use 'local computations' to indicate calculations performed solely on one MPI rank, and 'communications' to denote the interactions among multiple ranks. Apart from message sending and receiving, communications also include a 'waiting' phase, where one or more processes remain idle until others finish the work so that further computations can take place. In general, the overall running time of a distributed parallel algorithm is the summation of local and communication time. To describe our parallel algorithm clearly, we refer to some standard MPI functions. These primarily include

- *Send*: the operation of sending some array of memory from one MPI rank to another.
- *Receive*: the operation of receiving the memory sent by *Send*.

- *Allreduce*: the operation of reducing values from all ranks to one value and storing it on all ranks.

Note that although a processor is allowed to send messages to itself, in this manuscript we assume that communications occur with at least two MPI ranks.

3.1 Data distribution

We start the discussion of our distributed memory parallel implementation with data distribution to processes, which is essential in achieving load balance among MPI ranks. A good load balance helps minimize the idle time of processes and is thus one of the keys to good performance. To design a proper process partitioning strategy, we first describe the major tasks distributed at different stages of MP2.

3.1.1 Sparse maps and integral lists

In local correlation theory, only nontrivial elements of an object are computed and stored. "Sparse map", introduced by Pinski and coworkers,[41,74] is used to keep track of the indices of these significant parts. Suppose there are two lists $A = \{\alpha_1, \ldots, \alpha_m\}$ and $B = \{\beta_1, \ldots, \beta_n\}$, and each element $\alpha_a \in A$ for $1 \leq a \leq m$ has a corresponding subset B_a of B, then the sparse map $A \to B = \bigcup_{a=1}^{m} B_a$ contains a list of elements of B for all elements of A. Note that A and B do not necessarily need to be two different lists.

Sparse maps can be concatenated via traversing through all the elements. For two sparse maps $A \to B$ and $B \to C$, concatenating them yields an $A \to B \to C$ sparse map for lists A and C. In particular, when $B = A$, we call $A \to A \to C$ an "extended" sparse map and shorten the notation to $A \to\to C$. Notice that $A \to\to C$ contains more elements of C compared to $A \to C$. In our algorithm, we use five basic sparse maps $i \to j, i \to a, i \to P, i \to \mu$, and $a \to \nu$, where i and j are used to denote the set of all occupied orbitals, a for

the set of all virtual orbitals, P for all auxiliary orbitals, and μ and ν for all atomic orbitals.

In our algorithm, we implement robust local density fitting (LDF) to calculate $(ia|jb)$ elements:

$$(ia|jb) = -\sum_{i\to P,\ j\to Q} C_{ia}^{P}\,(P|Q)\,C_{jb}^{Q} + \sum_{i\to P} C_{ia}^{P}\,(jb|P) + \sum_{j\to Q} C_{jb}^{Q}\,(ia|Q)\,, \tag{13}$$

where $(ia|P)$ represents the interaction among occupied MO, corresponding virtual MO, and corresponding auxiliary orbitals, and can be computed with

$$(ia|P) = \int dr \int dr' \phi_i(r)\phi_j(r)\frac{1}{|r-r'|}\phi_P(r'),\ 1 \le i,j \le n_{\text{occ}},\ 1 \le P \le n_{\text{aux}}. \tag{14}$$

We generate $(ia|P)$ via an exhaustive search of two sparse maps $i \to\to a$ and $i \to\to P$. As extende maps are larger than the direct ones, we use them here for a more accurate representation of $(ia|P)$. More intuitions and details of using extended sparse maps can be found in.[41,48,74] In addition, the fitting coefficient C_{ia}^{P} is

$$C_{ia}^{P} = \sum_{i\to Q} V_{PQ}^{-1}\,(ia|Q), \tag{15}$$

where $(P|Q)$ is the fitting metric and can be computed similar to (14), and V_{PQ} is a symmetric submatrix of $(P|Q)$, with row and column indices from the sparse map $i \to P$. With $(ia|P)$ calculated, C_{ia}^{P} can be found by solving a symmetric linear system. Finally, (13) indicates that the ERI matrix can be computed with matrix-matrix multiplications among $(ia|P)$, $(P|Q)$, and C_{ia}^{P}. More details are explained in our preceding paper.[48] From here on we focus on our design on distributing data and computing jobs among several nodes.

From section 2, we discover that the multiplication in CG is performed with a partitioning of the ERI matrix with respect to the occupied orbital basis, so we generate the ERI matrix block by block. Therefore, the sparse maps and integral lists are also stored according to

the occupied MO basis. As a result, during the phase of sparse map construction, each MPI rank deals with a subset of the occupied MO basis and later communicates with each other to handle all the sparse maps for the entire occupied MO basis.

We then encounter the most time-consuming step of the construction of the integral list $(ia|P)$. In practice, to achieve better distributed memory load balance, for each auxiliary orbital P, we conduct a preliminary computation of the number of $(ia|P)$ to be generated using the number of elements in the sparse maps $i \twoheadrightarrow a$ and $i \twoheadrightarrow P$. This allows us to assign auxiliary orbitals to the MPI ranks in a way that each rank computes roughly the same number of $(ia|P)$ elements.

3.1.2 ERI matrix and CG iterations

From section 2.3, we know multiplications (7) and (10) take place on a single ERI block in (2), and multiplication (8) involves all blocks on the same block column. This suggests that we need two process grids separately for ERI generation and CG solver, and we aim to achieve load balance within each process partition and minimize communication to move from the ERI grid to the CG grid.

In order to minimize communication costs for performing multiplication (8), the most straightforward way to assign the processes is 1D block cyclic partitioning[75] with respect to the occupied MO basis. It is then natural to let the processes generate ERI blocks on the block column they handle so that the ERI grid and the CG grid are in fact combined into one. However, since only the lower triangular half of the ERI blocks need to be generated due to symmetry, the standard 1D block cyclic partitioning leads to load imbalance for the $(n_{\text{occ}}+1)n_{\text{occ}}/2$ blocks. Instead, for B processes, we use a special 1D block cyclic process grid with period $2B$ instead of B in the standard implementation. Mathematically, suppose $n_{\text{occ}} = k(2B)$ for simplicity for some integer k, then the pth process, for $0 \leq p \leq B-1$, handles block columns $p, 2B-1-p, 2B+p, 4B-1-p, \cdots$. In this way, each process deals

with $2k$ block columns and $k(n_{\text{occ}}+1)$ blocks. In the special case that the ERI matrix is dense, each block contains exactly n_{virt}^2 elements, so this load-balanced partitioning of blocks indicates that all MPI ranks hold an equal number of elements. If n_{occ} is not a multiple of $2B$, then the process with the most duties handles at most one more block column and $B-1$ more blocks than the rest of the processes. We also discover that this partition strategy works well in practice when the ERI matrix is sparse. For example, consider the simple molecule CH_4, which has 4 occupied MOs thus 4 block columns and 16 ERI blocks in (2). Figure 2 (Right) shows the 1D block cyclic process grid with 2 MPI ranks, and Figure 2 (Left) shows how the processes generate the corresponding ERI blocks. In this example, Process 0 deals with block columns 0 and 3, while Process 1 deals with block columns 1 and 2. As a result, both processes generate 5 blocks on and below the diagonal in the ERI matrix construction (red). In addition, for the upper triangular blocks, 1 block can be obtained directly with a transpose (green) because it is owned by the same process, and 2 blocks need to be acquired through communications (blue).

Figure 2: 1D block cyclic partitioning schemes for CH_4. Left: the process grid for ERI generation, where submatrices of ERI as in (2) are assigned to two processes. The numbers represent the process the block belongs to. The red font indicates that the block is generated, the green font indicates that the block is also on the same process and can be discovered through transpose, and the blue font indicates that the block is obtained from communication. Right: the process grid for CG solver, where block columns are assigned to the two processes to enable multiplication (8).

With this partitioning, the multiplication for all processes at each CG iteration can be summarized in 3 steps: 1) *Receive* blue block data from the previous iteration from other

processes; 2) Perform (7), (10), and (8) on the blocks and block columns; 3) *Send* blue block data back to the processes communicated with in step 1). Roughly speaking, at each CG iteration, each process needs to send $k(n_{\text{occ}}+1)/B$ blocks to every other process, and receive the same amount accordingly.

We can also design the ERI grid first, and partition the processes for the CG solver accordingly. Since the sparse ERI blocks are constructed as a dense matrix first with the two integral lists, and then post-processed to remove small numbers, an intuitive ERI grid can be built based on the number of elements generated in the dense version of the blocks. Via sorting the number of elements in each block, we can ensure that the processes work with roughly the same number of blocks and elements in the ERI grid. In addition, we assume implicitly that the same process owns the symmetric upper triangular counterparts to avoid communication in ERI generation. This scheme is referred to as the 2D round-robin process partitioning[75] in this article. Figure 3 (Left) shows this ERI grid for the same example CH_4 with 2 MPI ranks. In this case, each process generates 5 blocks on and below the diagonal in the ERI matrix construction (red) and knows 3 blocks above the diagonal (green) via transpose.

In order to minimize data transfer between the ERI grid and CG grid to perform multiplication (8), we let each block column belong to the process that owns the most number of blocks on that column, while requiring all processes handle roughly the same number block columns. In other words, suppose there are B processes, and $n_{\text{occ}} = kB$ for an integer k, then each process constructs $k(n_{\text{occ}}+1)/2$ blocks in the ERI grid and holds k block columns in the CG grid. If n_{occ} is not a multiple of B, this partitioning scheme still ensures that the busiest process only handles at most 1 more block column and constructs at most 1 more ERI block than the rest of the processes. Figure 3 (Right) shows the CG grid corresponding to the ERI grid in Figure 3 (Left). As the 4 blocks on each block column are distributed evenly to the processes, we can assign any 2 block columns to each process. In the figure,

Figure 3: 2D round-Robin partitioning schemes for CH_4. Left: the process grid for ERI generation, where submatrices of ERI as in (2) are assigned to two processes. The numbers represent the process the block belongs to. The red font indicates that the block is generated, and the green font indicates that the block is also on the same process and can be discovered through transpose. Right: the process grid for CG solver, where block columns are assigned to the two processes to enable multiplication (8).

Process 0 performs multiplications in CG on block columns 0 and 1, and Process 1 performs multiplications on block columns 2 and 3.

Similarly, the multiplication for all processes at each CG iteration can also be summarized in 3 steps: 1) For each block column, *Receive* data not on this process from the previous iteration from other processes; 2) Perform (7), (10), and (8) on the blocks and block columns; 3) *Send* data back to the processes communicated with in step 1). Same as the 1D partitioning, we can find that each process needs to send and receive approximately $k(n_{\text{occ}}+1)/B$ blocks to every other process.

We notice that in practice, the two partitioning schemes have similar performance. This is reflected in Table 1 (see section 4 for a detailed description of the columns of this table), which contains timing results of running LMP2 on the alkane $C_{220}H_{442}$ with 1D and 2D process partitioning using 44 MPI and 16 OMP threads per MPI. The computational efficiency of the 1D and 2D process partitioning are on the same level for all phases in LMP2. We can thus conclude that both process grids are competent with large problems using plenty of computing cores. For the rest of this paper, we conduct experiments using the 2D process grid, but similar timing and storage results should be expected for the 1D grid as well.

Table 1: Timing (seconds) of distributed memory LMP2 with 1D and 2D process partitioning on $C_{220}H_{442}$ using 44 MPI and 16 OMP threads per MPI.

Scheme	LDF	ERI	Solve	Total
1D	29	166	128	323
2D	30	177	127	333

3.2 Distributed conjugate gradient

With the 2D round-Robin process partitioning and matrix multiplication routines to maintain the sparsity pattern introduced in section 2.3, we can propose a sparsity-enforced CG solver for (4). Because of the Kronecker product structure of Δ, its diagonal elements are combinations of those of F_{occ} and F_{virt}. Therefore, we can build Jacobi preconditioners easily for each ERI block. Algorithm 1 summarizes this preconditioned CG solver to find the wave function amplitudes matrix T. This is a straightforward distributed memory implementation of the CG method[76] with the special matrix-vector multiplication we design to maintain sparsity pattern. In particular, lines 2-7 specify the communication patterns to perform this operation and are used repeatedly. In addition, coefficients used in CG iterations are computed on each MPI rank and are reduced in lines 9-10. In practice, Algorithm 1 converges in only a couple of iterations for our test cases.

4 Results and Discussion

In this section, we present some results on running our parallel MP2 on a range of molecular systems. In particular, we focus on linear alkane chains with def2-TZVP basis set and the frozen core approximation in section 4.2, a couple of molecules with 3D structures and various basis sets in section 4.3, and some molecules emerging in applications in section 4.4. Our distributed memory algorithm uses the 2D round-Robin process partitioning strategy (see

Algorithm 1 Preconditioned CG for solving (4) on one MPI process.

Require: Fock matrices F_{occ} and F_{virt}, desired accuracy ϵ, right-hand-side C_p, initial guess X_p, and set of block columns W handled by this process.

Ensure: Solution X_p on this process (overwriting the initial guess)

1: Construct Jacobi preconditioners Q_p for the blocks on this process.
2: *Send* $\tilde{X}_p$, blocks in X_p that are not in W, to corresponding processes.
3: *Receive* $\bar{X}_p$, blocks in B that are not part of X_p, from other processes.
4: Perform (7) and (8) on X_p.
5: Perform (10) on W.
6: *Send* transpose of blocks appeared in line 3 to the same list of processes.
7: *Receive* transpose of blocks appeared in line 2 from the same list of processes to finish computing $Z_p = f(\Delta X_p)$ while keeping the fixed sparsity pattern.
8: Set $R_p = C_p - Z_p$, $P_p = Q_p^{-1} R_p$, and $Y_p = P_p$.
9: Compute $\alpha_0 = R_p \cdot Y_p$ and $\gamma_0 = ||R_p||^2$.
10: *Allreduce* to sum over all α_0 and γ_0 from different processes to get α and γ.
11: **while** $\sqrt{\gamma} > \epsilon$ **do**
12: Repeat lines 2 to 7 to compute $Z_p = f(\Delta P_p)$.
13: Compute $P_p \cdot Z_p$ and *Allreduce* to sum and get β, and set $\nu = \alpha/\beta$.
14: Set $X_p = X_p + \nu P_p$, $R_p = R_p - \nu Z_p$, and $Y_p = Q_p^{-1} R_p$.
15: Set $\beta = \alpha$.
16: Repeat lines 9 to 10 to get α and γ.
17: Compute $\mu = \alpha/\beta$ and set $P_p = Y_p + \mu P_p$.
18: **end while**

section 3.1), and is implemented in MiniQC, a lite version of Q-Chem[77] with hybrid MPI and OpenMP parallelism. All the experiments are conducted on the CPU nodes of Perlmutter, a Cray EX supercomputer hosted by NERSC at Lawrence Berkeley National Laboratory with 2 AMD EPYC 7763 CPUs per node (i.e. $2 \times 64 = 128$ cores per node).

In the experiments, we separate our timing results into 3 parts: 1) Local density fitting (LDF) which contains the construction of sparse maps and preliminary information; 2) ERI which is comprised of integral list formation and ERI generation; and 3) Solve which includes the fixed sparsity CG solver and the correlation energy computation. These three labels appear in the legends throughout the graphs in this section.

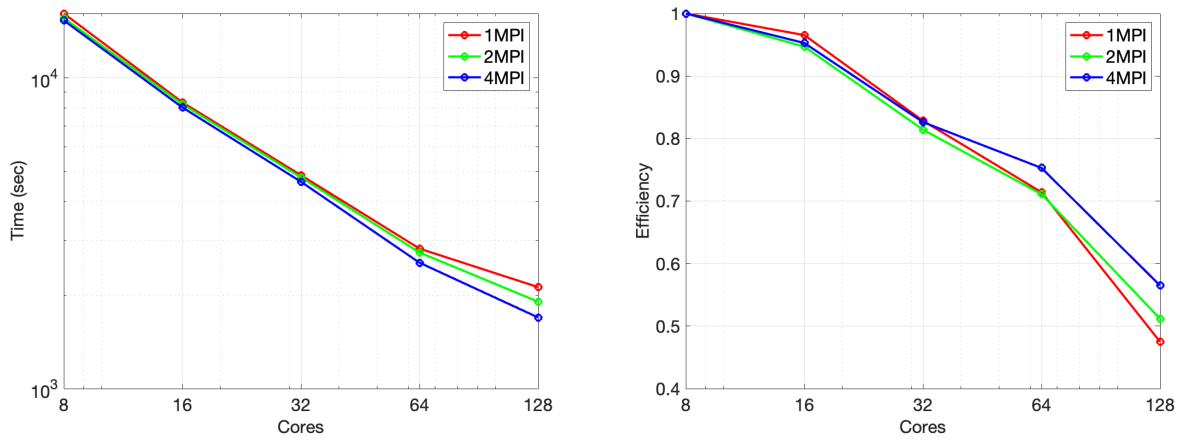

Figure 4: OpenMP strong scaling (Left) and corresponding parallel efficiency (Right) of running distributed LMP2 on 1 node for alkane $C_{220}H_{442}$. Each curve shows the result of utilizing different number of cores on the node with different MPI ranks. Left: We see the best performance when all 128 cores are used, and MPI parallelism is better than OpenMP parallelism. Right: The timing of using 8 cores is set to be the benchmark, and we see 4 MPI ranks achieve best parallel efficiency when more cores are used.

4.1 MPI and OpenMP setup

In our algorithm, OpenMP is used on each MPI rank. In other words, if we use B MPI ranks with M OMP threads per MPI, then the total number of computing cores used is BM. Figure 4 shows the time (Left) and corresponding parallel efficiency (Right) of running our codes on $C_{220}H_{442}$ using 1 MPI rank, 2 MPI ranks, and 4 MPI ranks on 1 node with different number of OMP threads per MPI respectively. The tests with 8 cores used are set to be the benchmark for computing parallel efficiency. One can find that our algorithm achieves the best performance while using all the cores on the node. In addition, if the number of computing cores utilized is fixed, using 4 MPIs achieves the best parallel efficiency, especially with a larger number of cores. This suggests that distributed memory parallelism is better than shared memory parallelism in our algorithm, at the expense of higher storage costs.

Because of the memory usage of our algorithm, we can afford to use 4 MPI ranks with 32 OMP threads each on 1 node for smaller-sized problems (in terms of molecule structure, basis functions, and desired accuracy), but we can only use 1 MPI per node for larger test

cases. Table 2 shows the OpenMP strong scaling results of our codes performed on $C_{400}H_{802}$ with $10^{-6.5}$ accuracy using 16 MPI ranks on 16 nodes. Here, the $10^{-6.5}$ threshold indicates that we treat elements smaller than $10^{-6.5}$ in ERI as zeros to create numerical sparsity in the LMP2 method. We find that using all 128 cores on each node gives the best timing, so we should use all possible computing cores if the number of MPI ranks is fixed.

In different scenarios, however, fully utilized nodes do not exhibit optimal time to solution because of e.g. memory bandwidth limitations. For example, Table 3 shows the time and energy consumption of running our codes on $C_{400}H_{802}$ using a total of 512 computing cores, where we can freely choose the combination of MPI ranks and OMP threads used. In other words, we are allowed to use as many nodes as we want. Only the first data point (4 MPIs with 128 OMP each) in Table 3 fully uses the 4 nodes requested, but this combination has the worst timing performance. This aligns with our previous observation that our MPI parallelism is better than OpenMP parallelism to some extent. As a result, in experiments with fixed computing cores such as weak scaling, one can use more MPIs with under-utilized nodes for faster solutions of large problems. The trade-off is worse energy efficiency and higher storage costs, which is not a zero sum game.

Energy efficiency is clearly an important metric to consider. Perlmutter jobs report an estimate of energy consumption based on the elapsed time, the number of nodes employed, and the fraction of each node's cores that are used (see Table 3), and the stats show that using fewer highly-occupied nodes leads to much lower energy consumption for fixed total number of cores. For less than 20% more energy consumption, execution time can be reduced by over 30% by changing from 4 fully utilized nodes to 8 half-utilized nodes, which is favorable. However to reduce execution time another 15% (using 32 nodes at 1/8-utilization) requires almost 300% more energy (than with 8 nodes). Clearly optimizing an objective function that combines energy use and time-to-solution is potentially worthwhile in future work.

Table 2: OpenMP strong scaling of distributed LMP2. Timing (seconds) of alkane $C_{400}H_{802}$ on 16 nodes using 16 MPI ranks and different numbers of OMP threads per MPI. Best efficiency is achieved with fully used nodes.

OMP	32	64	128
Time (sec)	936	645	600

Table 3: Fixed computing cores test of distributed LMP2. Timing (seconds) and energy (mega-joules) of alkane $C_{400}H_{802}$ using 512 cores with different combinations of MPI and OMP numbers. Each MPI uses 1 node. Best timing performance is obtained with 32 MPI ranks and under-utilized nodes, while lowest energy consumption is achieved with 4 MPI ranks and fully-utilized nodes.

Node	4	8	16	32	64
MPI	4	8	16	32	64
OMP	128	64	32	16	8
Time (sec)	1538	1030	936	865	891
Energy (MJ)	3.75	4.43	7.67	13.05	27.83

4.2 Linear Alkane Chains

The first set of experiments focuses on 1D alkane chains, which was one focus of the results reported in our earlier implementation that did not employ distributed memory parallelism.[48] In the calculations, we use the def2-TZVP basis set and the frozen core approximation.

We start with the alkane $C_{400}H_{802}$, which has 1201 occupied orbitals and 15611 virtual orbitals. Figure 5 (Left) shows a strong scaling timing test with 8, 16, 32, 64, and 128 MPI ranks with 32 OMP threads per MPI. The numbers at the data points indicate the parallel efficiency for different stages of the algorithm, where the test using 8 MPI ranks is set to be the benchmark of computations. We can see that Solve scales best with respect to the number of MPI ranks used, and has the best parallel efficiency. Since LDF and ERI contain some sequential components to generate sparse maps and truncate $(ia|jb)$ elements, they have worse strong scalability compared with Solve. As a result, Total time scales worse compared to Solve. Nevertheless, since the serial part is fast, the Total time still exhibits

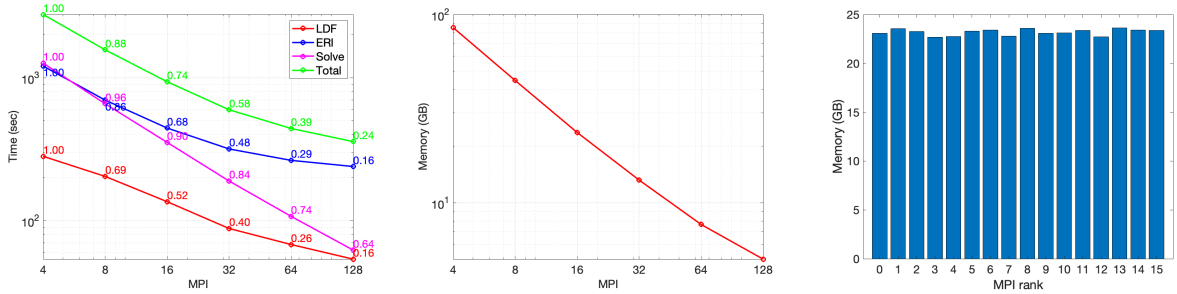

Figure 5: Left: MPI time strong scaling of our parallel MP2 method on the alkane $C_{400}H_{802}$. Numbers on the curves indicate the parallel efficiency if the test with 4 MPI ranks is set to be the benchmark. Solve shows almost linear scaling and best parallel efficiency. Middle: MPI memory strong scaling on the same alkane $C_{400}H_{802}$. Right: Memory usage on each MPI rank when 16 processes are used to run our algorithm on $C_{400}H_{802}$. The difference among processes is small, indicating good load balance.

good scaling. Figure 5 (Middle) shows the memory usage per MPI rank in the same strong scaling setup. From the curve, we can tell an almost perfect linear scaling for storage costs. Lastly, Figure 5 (Right) shows the storage cost on each MPI rank when we use 16 MPIs. One can see that every MPI rank uses roughly 23 GB, which indicates a good load balance for the chosen data distribution scheme.

We present the weak scaling result for a series of alkanes in Figure 6. In this experiment, we use 64 computing cores for every 10 carbon atoms. For better efficiency (see Figure 4 and Table 3), we choose 32 OMP per MPI, so there are 2 MPI ranks for every 10 Carbon. Therefore, the experiments range from $C_{10}H_{22}$ to $C_{600}H_{1202}$. For timing (see Figure 6 (Left)), similar to the strong scaling result, we see good scaling behaviors in the Solve stage. Especially, starting from moderately long chain lengths ($C_{40}H_{82}$), the timing curve scales almost linearly. Overall, the Total time behaves similarly to ERI construction, which contains sequential element truncation and nontrivial MPI communications. Still, the curves are close to a linear scaling pattern, so we can anticipate a good performance of our parallel algorithm when applied to even larger test cases. Because duplicated data with quadratic scaling of number of orbitals, such as Fock matrices, are stored on each MPI rank for easier computations, we expect the storage costs to also grow quadratically with the linear increase of

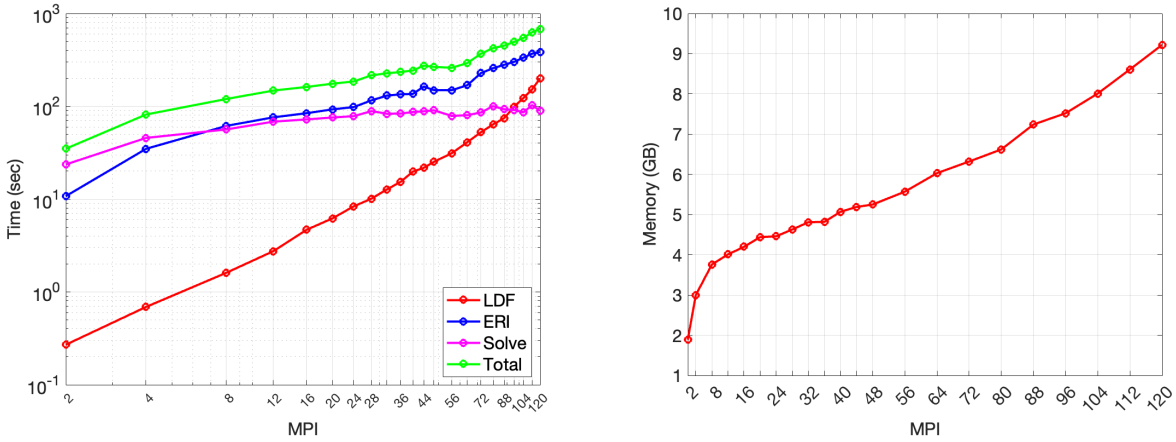

Figure 6: Weak scaling of time (Left) and storage cost (Right) of our algorithm on alkane chains from $C_{10}H_{22}$ to $C_{600}H_{1202}$. We use 2 MPI ranks with 32 OMP per MPI for every 10 carbon atoms. We see almost linear scaling for timing and quadratic scaling for memory consumption.

the number of MPIs. We can visualize this from the trend of the curve in Figure 6 (Right), when the number of MPI used is larger than roughly 20.

Furthermore, Figure 7 shows the timing (Left) and memory (Right) requirements when we apply our algorithm to a sequence of alkane chains with limited computing power and three levels of accuracy threshold. Specifically, we use 16 MPI ranks with 32 OMP per MPI on all alkanes C_nH_{2n+2}. For all levels of accuracy, even with a tight threshold of 10^{-7}, that delays the onset of linearity, we see good scaling for both timing and storage cost. The time drop occurring at $C_{560}H_{1122}$ for 10^{-5}, $C_{480}H_{962}$ for 10^{-6}, and $C_{360}H_{722}$ for 10^{-7} are the results of changing from 4 MPIs per node to 1 MPI per node.

Finally, Table 4 shows the floating point operation (flop) rate (number of flops per unit time), a performance metric other than time-to-solution, of using our algorithm on linear alkanes. In particular, we test on $C_{20}H_{42}$ and $C_{100}H_{202}$ with basis def2-TZVP and accuracy threshold $10^{-6.5}$, and report the flop rate (GFLOP/sec) of LDF, ERI, and Solve stages. Numbers in this table come from the profiling tool CrayPat [78] on Perlmutter. To get accurate flop rate on CrayPat, we test with 1 OMP thread per MPI rank. For both tests, we observe that ERI has the highest flop rate while Solve has the lowest. This is expected as we mainly

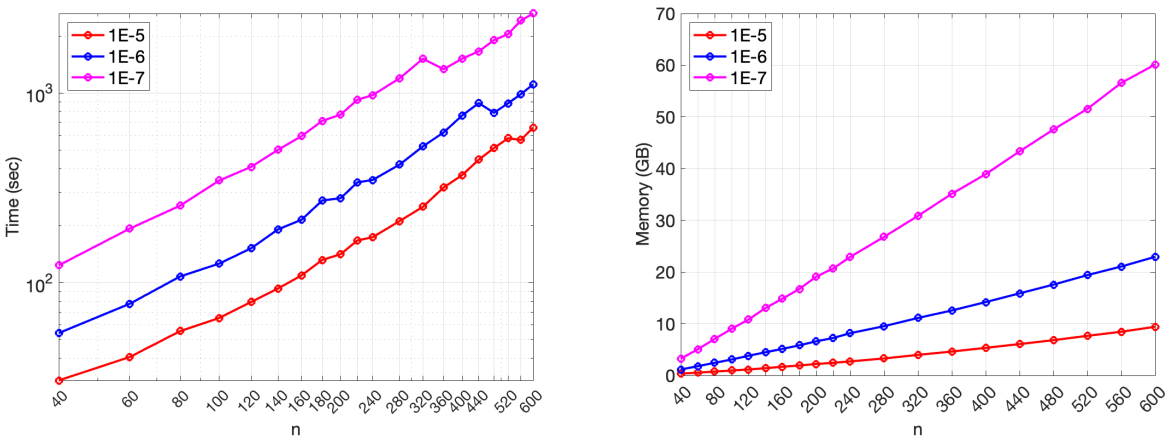

Figure 7: Time (Left) and storage cost (Right) of our algorithm on alkane chains with fixed computing power and three levels of threshold. All the experiments C_nH_{2n+2} use 16 MPI ranks and 32 OMP per MPI with a total of 512 computing cores, and we see good scaling for both timing and memory.

use BLAS3 routines such as matrix-matrix multiplications in ERI construction, and BLAS2 routines such as matrix-vector multiplications in the CG solver. We use a combination of BLAS2 and BLAS3 routines for sparse map generation, so the flop rate of LDF is in between ERI and Solve. With another test to run $C_{20}H_{42}$ with 1 MPI and 1 OMP, we observe that the flop rate we achieve for ERI is only about 10% of the Perlmutter optimal performance of 39.2 GFLOPs on one CPU core. Therefore, we conclude that our implementation right now is memory bound, and we will investigate further on this low flop rate performance in the near future.

Table 4: Aggregate flop rate of distributed LMP2 on $C_{20}H_{42}$ and $C_{100}H_{202}$. 1 OMP thread is used per MPI rank and the unit used is GFLOP/sec. ERI construction has the highest flop rate while CG Solve has the lowest flop rate.

Test	Node	MPI	LDF	ERI	Solve
$C_{20}H_{42}$	1	8	9.77	29.98	3.46
$C_{100}H_{202}$	4	16	45.15	76.99	6.80

4.3 Higher dimensional compounds

In this section, we test our algorithm on some molecules with 3-dimensional structures to further illustrate the efficiency and scalability of our algorithm. Figure 8 shows the testing systems, including compact non-covalent binding systems buckycatcher and circumcircumcoronene dimer (a fragment of graphene), two drug molecules paclitaxel and vancomycin, and a small protein crambin. Geometries used for these molecules are included in the Supplementary Information. Specifically, we use a strong scaling setting with 4, 8, 16, 32, and 64 MPI ranks and 32 OMP threads per MPI with different choices for the molecular basis and threshold levels (see Figure 9), and we see good strong scalability. The missing data point, which corresponds to using 4 MPI ranks for graphene, is the result of storage space shortage on one node for the partitioned sub-problem. Throughout this section, the memory we report is necessary for all calculations, but we see peak memory during constructions of sparse maps and the $(ia|P)$ integral list. This is the same obstacle that hinders us from using more MPI ranks on one node for large molecules. Therefore, one important ongoing task is to reduce memory usage with a detailed peak memory usage profile, so that we can handle large problems in certain conditions in the future with higher computational and memory efficiency.

4.4 Comparison against reported timings using other algorithms

Finally, in this section, we run our LMP2 algorithm on some molecules for which timings have been reported using other local MP2 algorithms. We note that there are at least two intrinsic difficulties in making this type of comparison. First, different computer resources are employed in each case, and second, the cost of local MP2 calculations increases very strongly with the desired accuracy. For example, as we reported in our previous paper,[48] making our drop tolerance ten times smaller requires roughly ten times more computation.

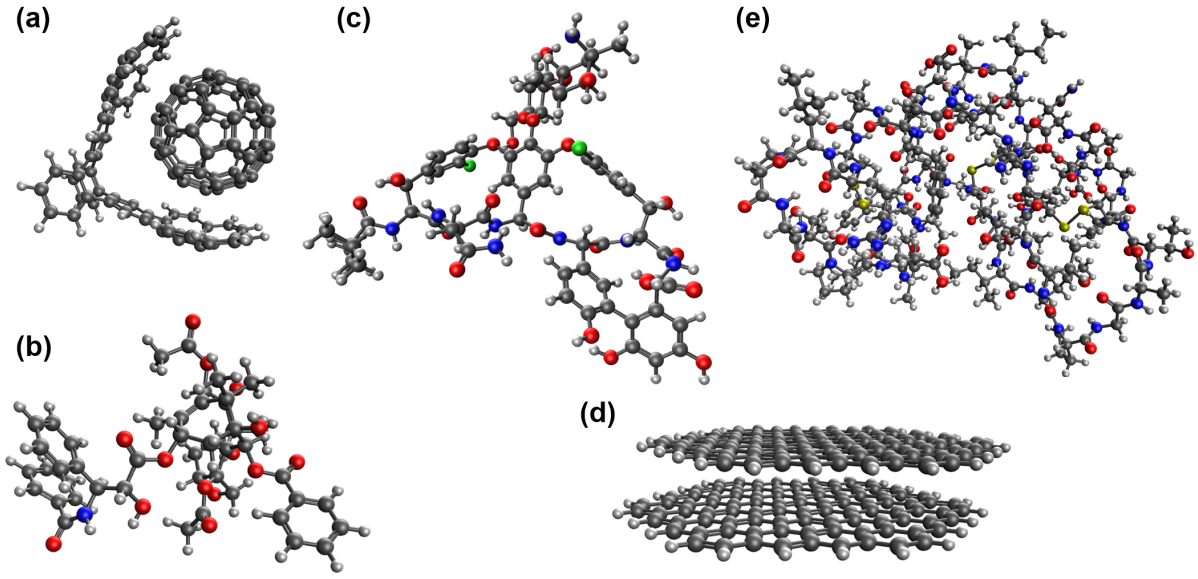

Figure 8: Testing 3-D molecules: (a) buckycatcher $C_{60}@C_{60}H_{28}$, (b) paclitaxel $C_{47}H_{51}NO_{14}$, (c) vancomycin $C_{66}H_{75}Cl_2N_9O_{24}$, (d) circumcircumcoronene dimer $C_{192}H_{48}$, (e) crambin $C_{200}H_{309}N_{55}O_{65}S_6$. Color codes for atoms: C, black; H, white; O, red; N, blue; Cl, green; S, yellow.

With these caveats in mind, we tested water clusters and DNA fragments[43] and glycine chains.[79] Figure 10 shows examples of the structures of these systems, and their geometries are included in the Supplementary Information, with structures taken from refs. 79,80. We performed our calculations in the cc-pVDZ basis, with an accuracy threshold of 10^{-6}.

Table 5 shows the timings (always with 32 OpenMP threads) and storage requirements with different parallel computing configurations. From the table, we see that a threshold of 10^{-6}, which typically recovers more than 99.99% of the correlation energy, takes 290 seconds on DNA_4, using 64×32 AMD Epyc Milan cores. For comparison, in ref. 43, the same molecule needs 47 minutes of elapsed time using only 6 cores, and recovers only 99.9% of the correlation energy. The distributed memory computing algorithm has reduced time to solution by over a factor of ten, while our tighter thresholding has enabled a factor of ten higher accuracy.

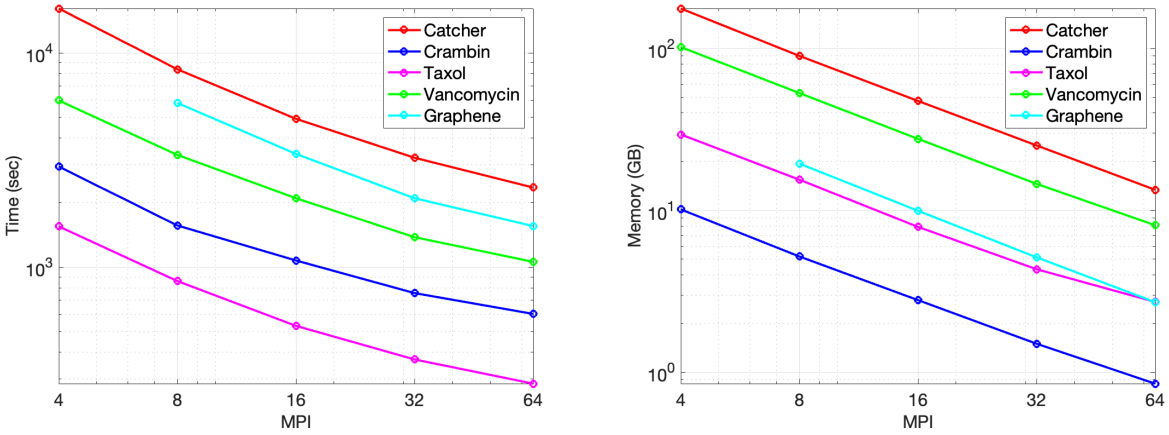

Figure 9: Strong scaling of time (Left) and storage cost (Right) of our algorithm on catcher, crambin, paclitaxel, vancomycin, and graphene. We use basis def2-SV(P) for crambin, and def2-TZVP for all other experiments. In terms of accuracy, we use 10^{-5} for crambin and graphene, 10^{-6} for catcher and paclitaxel, and $10^{-6.5}$ for vancomycin.

As another comparison, we completed the $(\text{gly})_{50}$ calculation in 31 seconds using 16 nodes (64 MPI processes) and 32 OpenMP threads. A state-of-the-art GPU implementation, which was also evaluating gradient information (perhaps a factor of 2-3 extra work), takes 110 seconds[79] using 8 NVIDIA Ampere A100 GPUs processors, which has, in principle, about two times higher peak performance in double precision (156 TFLOPS vs 80 TFLOPS). The GPU implementation relies on a fragment molecular orbital (FMO) approximation to accelerate the calculations, which involves manual partitioning of the system into a set of smaller jobs that are recombined to approximate the global calculation. This permits dense linear algebra straightforwardly, leading to high floating point efficiency in the GPU implementation. However, one cannot perform rigorous thresholding, as we do, and in extreme cases the FMO method may not be able to yield stable accuracy as it depends on fragmentation.[81] Overall, while direct comparison is difficult, it is clear that our current method yields very competitive times to solution as a result of the distributed memory implementation, with the advantage of easily controllable accuracy.

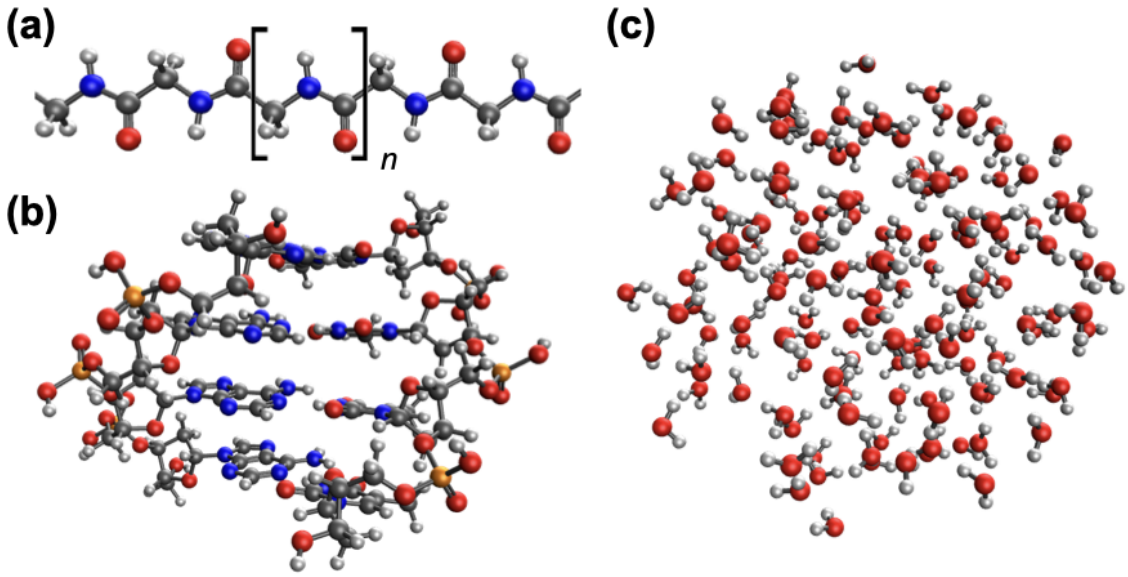

Figure 10: Molecular systems used to compare against previously reported local MP2 timings: (a) Polyglycine chains; $(gly)_n$, (b) DNA_4, (c) $(H_2O)_{142}$. Color codes for atoms: C, black; H, white; O, red; N, blue; P, orange.

5 Conclusions

In this manuscript, we have reported on a distributed memory parallel implementation of a recently introduced local MP2 method[48] to compute the MP2 electron correlation energy with numerical precision controlled via a single threshold. Our implementation consists of a distributed construction of sparse maps and ERI blocks, and a distributed conjugate gradient method to compute the amplitude tensors. In particular, we use two process partitioning strategies for different stages. We use 2D round-robin process partitioning to generate ERI blocks corresponding to a pair of occupied orbitals (i, j), and conduct some multiplications in CG. We use a different 1D cyclic block column partitioning for the remaining CG multiplications.

We tested our implementation on a sequence of linear alkanes and a set of molecules with 3D structures. Good scalability was observed on strong scaling tests, weak scaling

Table 5: Timing (seconds) and storage costs (megabytes) of realistic molecules using different parallel configurations.

Molecule	Node	MPI	LDF (s)	ERI (s)	Solve (s)	Total (s)	Memory (GB)
DNA_1	16	16	0.39	10.19	7.51	18.22	7.85
DNA_2	32	32	1.27	46.24	21.24	69.05	40.00
DNA_4	64	64	4.95	234.95	49.6	290.13	122.32
$(gly)_{10}$	8	32	0.34	7.38	2.63	10.38	5.2
$(gly)_{20}$	16	64	0.84	11.01	3.15	15.03	11.76
$(gly)_{30}$	16	64	1.61	15.71	4.92	22.28	18.54
$(gly)_{40}$	16	64	2.46	20.94	4.38	27.84	25.66
$(gly)_{50}$	16	64	3.83	22.48	4.88	31.26	33.13
$(H_2O)_{68}$	64	64	1.58	40.91	6.52	49.14	19.38
$(H_2O)_{142}$	64	64	9.09	241.35	15.17	266.07	58.27
$(H_2O)_{285}$	64	64	34.11	1080.12	31.31	1146.45	153

tests, and fixed computing resources tests. Relative to the original shared memory OpenMP implementation[48] there are two key advantages:

1. Leveraging the much larger memory resources associated with distributed computing pays great benefits when applied to an algorithm such as local MP2 whose memory requirements scale linearly with system size. This is a weak scaling benefit: we can treat molecules that simply cannot be handled with shared memory parallelism due to insufficient per-node memory. Examples include alkane chains beyond C_{200} as well as higher accuracy thresholds which lead to larger memory requirements.

2. The time to solution is greatly reduced for even those problems that can still be solved using shared memory parallelism. This strong scaling benefit is demonstrated here by showing speedups of more than 50 times versus the OpenMP-only implementation when 64 - 128 MPI ranks are employed. This leads to quite short time-to-solution,

such as less than an hour for vancomycin in a def2-TZVP basis and a tight threshold. Again this job could not be completed on a single node at all.

There is considerable scope for further development of our distributed memory parallel implementation of linear scaling MP2. One important next step is to extend it to improved versions of MP2, such as the size-consistent second-order Brillouin-Wigner theory,[23,24] and regularized MP2.[22] Another important next step is to extend the approach to parallel local MP3 and CCD/CCSD. Although additional developments of various tensor contractions for these higher-order methods are needed, our parallel amplitude solver code can be reused without many modifications, and the same is true for our process partitioning schemes for different stages of the calculations. Another very desirable future direction is to involve the usage of data sparsity in our algorithm. With the Kronecker products to represent Δ in the linear system, we can rewrite the linear system $\Delta t = c$ into a matrix or tensor Sylvester equations. With further investigations of low-rank or hierarchical low-rank structures of the ERI tensor and Fock matrices, we want to develop algorithms to find amplitude tensors with Sylvester equation solvers, by utilizing both numerical sparsity and data sparsity.

Conflicts of Interest

MHG is a part-owner of Q-Chem Inc, which is the software platform used to implement the algorithms described here.

Acknowledgements

This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing, and Office of Basic Energy Sciences, via the Scientific Discovery through Advanced Computing (SciDAC) program. This research used resources of

the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

References

(1) Kohn, W.; Becke, A. D.; Parr, R. G. Density functional theory of electronic structure. *J. Phys. Chem.* **1996**, *100*, 12974.

(2) Mardirossian, N.; Head-Gordon, M. Thirty years of density functional theory in computational chemistry: An overview and extensive assessment of 200 density functionals. *Mol. Phys.* **2017**, *115*, 2315–2372.

(3) Goerigk, L.; Grimme, S. Double-hybrid density functionals. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2014**, *4*, 576–600.

(4) Martin, J. M. L.; Santra, G. Empirical double-hybrid density functional theory: A 'third way' in between WFT and DFT. *Isr. J. Chem.* **2020**, *60*, 787–804.

(5) Perdew, J. P.; Schmidt, K. In *Density Functional Theory and Its Applications to Materials*; Van Doren, V., Van Alsenoy, C., Geerlings, P., Eds.; AIP Conference Proceedings; American Institute of Physics, 2001; Vol. 577; pp 1–20.

(6) Hait, D.; Head-Gordon, M. Communication: xDH double hybrid functionals can be qualitatively incorrect for non-equilibrium geometries: Dipole moment inversion and barriers to radical-radical association using XYG3 and XYGJ-OS. *J. Chem. Phys.* **2018**, *148*, 171102.

(7) Cremer, D. Møller-Plesset perturbation theory: from small molecule methods to meth-

ods for thousands of atoms. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2011**, *1*, 509–530.

(8) Cohen, A. J.; Mori-Sánchez, P.; Yang, W. Challenges for density functional theory. *Chem. Rev.* **2012**, *112*, 289–320.

(9) Lao, K. U.; Herbert, J. M. Accurate and efficient quantum chemistry calculations of noncovalent interactions in many-body systems: The XSAPT family of methods. *J. Phys. Chem. A* **2015**, *119*, 235–253.

(10) Lao, K. U.; Herbert, J. M. A simple correction for nonadditive dispersion within extended symmetry-adapted perturbation theory (XSAPT). *J. Chem. Theory Comput.* **2018**, *14*, 5128–5142.

(11) Zahn, S.; MacFarlane, D. R.; Izgorodina, E. I. Assessment of Kohn–Sham density functional theory and Møller–Plesset perturbation theory for ionic liquids. *Phys. Chem. Chem. Phys.* **2013**, *15*, 13664–13675.

(12) Izgorodina, E. I.; Seeger, Z. L.; Scarborough, D. L.; Tan, S. Y. Quantum chemical methods for the prediction of energetic, physical, and spectroscopic properties of ionic liquids. *Chem. Rev.* **2017**, *117*, 6696–6754.

(13) Villot, C.; Lao, K. U. Electronic structure theory on modeling short-range noncovalent interactions between amino acids. *J. Chem. Phys.* **2023**, *158*, 094301.

(14) Fanourgakis, G. S.; Apra, E.; Xantheas, S. S. High-level ab initio calculations for the four low-lying families of minima of $(H_2O)_{20}$. I. Estimates of MP2/CBS binding energies and comparison with empirical potentials. *J. Chem. Phys.* **2004**, *121*, 2655–2663.

(15) Yoo, S.; Apra, E.; Zeng, X. C.; Xantheas, S. S. High-level ab initio electronic struc-

ture calculations of water clusters $(H_2O)_{16}$ and $(H_2O)_{17}$: A new global minimum for $(H_2O)_{16}$. *J. Phys. Chem. Lett.* **2010**, *1*, 3122–3127.

(16) Rakshit, A.; Bandyopadhyay, P.; Heindel, J. P.; Xantheas, S. S. Atlas of putative minima and low-lying energy networks of water clusters n =3–25. *J. Chem. Phys.* **2019**, *151*, 214307.

(17) Grimme, S. Improved second-order Møller–Plesset perturbation theory by separate scaling of parallel-and antiparallel-spin pair correlation energies. *J. Chem. Phys.* **2003**, *118*, 9095–9102.

(18) Jung, Y.; Lochan, R. C.; Dutoi, A. D.; Head-Gordon, M. Scaled opposite-spin second order Møller–Plesset correlation energy: an economical electronic structure method. *J. Chem. Phys.* **2004**, *121*, 9793.

(19) Goldey, M.; Head-Gordon, M. Attenuating away the errors in inter- and intramolecular interactions from second-order Møller–Plesset calculations in the small aug-cc-pVDZ basis set. *J. Phys. Chem. Lett.* **2012**, *3*, 3592–3598.

(20) Goldey, M.; Dutoi, A.; Head-Gordon, M. Attenuated second order Møller-Plesset theory: Assessment and performance in the aug-cc-pVTZ basis. *Phys. Chem. Chem. Phys.* **2013**, *15*, 15869.

(21) Loipersberger, M.; Bertels, L. W.; Lee, J.; Head-Gordon, M. Exploring the limits of second- and third-order Møller–Plesset perturbation theories for noncovalent interactions: Revisiting MP2.5 and assessing the importance of regularization and reference orbitals. *J. Chem. Theory Comput.* **2021**, *17*, 5582–5599.

(22) Shee, J.; Loipersberger, M.; Rettig, A.; Lee, J.; Head-Gordon, M. Regularized second-order Møller–Plesset theory: A more accurate alternative to conventional MP2 for non-

covalent interactions and transition metal thermochemistry for the same computational cost. *J. Phys. Chem. Lett.* **2021**, *12*, 12084–12097.

(23) Carter-Fenk, K.; Head-Gordon, M. Repartitioned Brillouin-Wigner perturbation theory with a size-consistent second-order correlation energy. *J. Chem. Phys.* **2023**, *158*, 234108.

(24) Carter-Fenk, K.; Shee, J.; Head-Gordon, M. Optimizing the regularization in size-consistent second-order Brillouin-Wigner perturbation theory. *J. Chem. Phys.* **2023**, *159*, 171104.

(25) Mardirossian, N.; Head-Gordon, M. Survival of the most transferable at the top of Jacob's ladder: Defining and testing the ωB97M(2) double hybrid density functional. *J. Chem. Phys.* **2018**, *148*, 241736.

(26) Santra, G.; Calinsky, R.; Martin, J. M. Benefits of range-separated hybrid and double-hybrid functionals for a large and diverse data set of reaction energies and barrier heights. *J. Phys. Chem. A* **2022**, *126*, 5492–5505.

(27) Becke, A. D.; Santra, G.; Martin, J. M. A double-hybrid density functional based on good local physics with outstanding performance on the GMTKN55 database. *J. Chem. Phys.* **2023**, *158*, 151103.

(28) 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.

(29) Weigend, F.; Häser, M. RI-MP2: first derivatives and global consistency. *Theor. Chem. Acc.* **1997**, *97*, 331.

(30) Weigend, F.; Häser, M.; Patzelt, H.; Ahlrichs, R. RI-MP2: optimized auxiliary basis sets and demonstration of efficiency. *Chem. Phys. Lett.* **1998**, *294*, 143.

(31) DiStasio, Jr., R. A.; Steele, R. P.; Rhee, Y. M.; Shao, Y.; Head-Gordon, M. An improved algorithm for analytical gradient evaluation in resolution-of-the-identity second-order Møller-Plesset perturbation theory: Application to alanine tetrapeptide conformational analysis. *J. Comput. Chem.* **2007**, *28*, 839.

(32) Kohn, W. Density functional and density matrix method scaling linearly with the number of atoms. *Phys. Rev. Lett.* **1996**, *76*, 3168.

(33) Mackie, C. J.; Gonthier, J. F.; Head-Gordon, M. Compressed intramolecular dispersion interactions. *J. Chem. Phys.* **2020**, *152*, 024112.

(34) Pulay, P. Localizability of dynamic electron correlation. *Chem. Phys. Lett.* **1983**, *100*, 151–154.

(35) Saebo, S.; Pulay, P. Local treatment of electron correlation. *Annu. Rev. Phys. Chem.* **1993**, *44*, 213–236.

(36) Maslen, P.; Head-Gordon, M. Non-iterative local second order Møller–Plesset theory. *Chem. Phys. Lett.* **1998**, *283*, 102–108.

(37) Ayala, P. Y.; Scuseria, G. E. Linear scaling second-order Møller–Plesset theory in the atomic orbital basis for large molecular systems. *J. Chem. Phys.* **1999**, *110*, 3660–3671.

(38) Lee, M. S.; Maslen, P. E.; Head-Gordon, M. Closely approximating second-order Møller–Plesset perturbation theory with a local triatomics in molecules model. *J. Chem. Phys.* **2000**, *112*, 3592–3601.

(39) Schütz, M.; Hetzer, G.; Werner, H. J. Low-order scaling local electron correlation methods. I. Linear scaling local MP2. *J. Chem. Phys.* **1999**, *111*, 5691–5705.

(40) Werner, H.-J.; Manby, F. R.; Knowles, P. J. Fast linear scaling second-order Møller-Plesset perturbation theory (MP2) using local and density fitting approximations. *J. Chem. Phys.* **2003**, *118*, 8149–8160.

(41) Pinski, P.; Riplinger, C.; Valeev, E. F.; Neese, F. Sparse maps—A systematic infrastructure for reduced-scaling electronic structure methods. I. An efficient and simple linear scaling local MP2 method that uses an intermediate basis of pair natural orbitals. *J. Chem. Phys.* **2015**, *143*, 034108.

(42) Werner, H.-J.; Knizia, G.; Krause, C.; Schwilk, M.; Dornbach, M. Scalable electron correlation methods I.: PNO-LMP2 with linear scaling in the molecular size and near-inverse-linear scaling in the number of processors. *J. Chem. Theory Comput.* **2015**, *11*, 484–507.

(43) Nagy, P. R.; Samu, G.; Kállay, M. An integral-direct linear-scaling second-order Møller–Plesset approach. *J. Chem. Theory Comput.* **2016**, *12*, 4897–4914.

(44) Kjærgaard, T.; Baudin, P.; Bykov, D.; Kristensen, K.; Jørgensen, P. The divide–expand–consolidate coupled cluster scheme. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2017**, *7*, e1319.

(45) Glasbrenner, M.; Graf, D.; Ochsenfeld, C. Efficient reduced-scaling second-order Møller–Plesset perturbation theory with Cholesky-decomposed densities and an attenuated Coulomb metric. *J. Chem. Theory Comput.* **2020**, *16*, 6856–6868.

(46) Li, W.; Wang, Y.; Ni, Z.; Li, S. Cluster-in-molecule local correlation method for dispersion interactions in large systems and periodic systems. *Acc. Chem. Res.* **2023**, *56*, 3462–3474.

(47) Wang, Z.; Aldossary, A.; Head-Gordon, M. Sparsity of the electron repulsion inte-

gral tensor using different localized virtual orbital representations in local second-order Møller–Plesset theory. *J. Chem. Phys.* **2023**, *158*, 064105.

(48) Wang, Z.; Aldossary, A.; Shi, T.; Liu, Y.; Li, X. S.; Head-Gordon, M. Local second-order Møller–Plesset theory with a single threshold using orthogonal virtual orbitals: theory, implementation, and assessment. *J. Chem. Theory Comput.* **2023**, *19*, 7577–7591.

(49) Subotnik, J. E.; Dutoi, A. D.; Head-Gordon, M. Fast localized orthonormal virtual orbitals which depend smoothly on nuclear coordinates. *J. Chem. Phys.* **2005**, *123*, 114108.

(50) Calvin, J. A.; Peng, C.; Rishi, V.; Kumar, A.; Valeev, E. F. Many-body quantum chemistry on massively parallel computers. *Chem. Rev.* **2020**, *121*, 1203–1231.

(51) Baker, J.; Pulay, P. An efficient parallel algorithm for the calculation of canonical MP2 energies. *J. Comput. Chem.* **2002**, *23*, 1150–1156.

(52) Ishimura, K.; Pulay, P.; Nagase, S. A new parallel algorithm of MP2 energy calculations. *J. Comput. Chem.* **2006**, *27*, 407–413.

(53) Bernholdt, D. E.; Harrison, R. J. Large-scale correlated electronic structure calculations: the RI-MP2 method on parallel computers. *Chem. Phys. Lett.* **1996**, *250*, 477–484.

(54) Hättig, C.; Hellweg, A.; Köhn, A. Distributed memory parallel implementation of energies and gradients for second-order Møller–Plesset perturbation theory with the resolution-of-the-identity approximation. *Physical Chemistry Chemical Physics* **2006**, *8*, 1159–1169.

(55) Katouda, M.; Nagase, S. Efficient parallel algorithm of second-order Møller–Plesset per-

turbation theory with resolution-of-identity approximation (RI-MP2). *Int. J. Quantum Chem.* **2009**, *109*, 2121–2130.

(56) Goldey, M.; DiStasio Jr, R. A.; Shao, Y.; Head-Gordon, M. Shared memory multiprocessing implementation of resolution-of-the-identity second-order Møller–Plesset perturbation theory with attenuated and unattenuated results for intermolecular interactions between large molecules. *Mol. Phys.* **2014**, *112*, 836–843.

(57) Katouda, M.; Nakajima, T. MPI/OpenMP hybrid parallel algorithm of resolution of identity second-order Møller–Plesset perturbation calculation for massively parallel multicore supercomputers. *J. Chem. Theory Comput.* **2013**, *9*, 5373–5380.

(58) Katouda, M.; Naruse, A.; Hirano, Y.; Nakajima, T. Massively parallel algorithm and implementation of RI-MP2 energy calculation for peta-scale many-core supercomputers. *J. Comput. Chem.* **2016**, *37*, 2623–2633.

(59) Barca, G. M.; Snowdon, C.; Vallejo, J. L. G.; Kazemian, F.; Rendell, A. P.; Gordon, M. S. Scaling correlated fragment molecular orbital calculations on Summit. SC22: International Conference for High Performance Computing, Networking, Storage and Analysis. 2022; pp 1–14.

(60) Fedorov, D. G.; Kitaura, K. Second order Møller-Plesset perturbation theory based upon the fragment molecular orbital method. *J. Chem. Phys.* **2004**, *121*, 2483–2490.

(61) Mochizuki, Y.; Yamashita, K.; Murase, T.; Nakano, T.; Fukuzawa, K.; Takematsu, K.; Watanabe, H.; Tanaka, S. Large scale FMO-MP2 calculations on a massively parallel-vector computer. *Chem. Phys. Lett.* **2008**, *457*, 396–403.

(62) 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.

(63) Pham, B. Q.; Gordon, M. S. Hybrid distributed/shared memory model for the RI-MP2 method in the fragment molecular orbital framework. *J. Chem. Theory Comput.* **2019**, *15*, 5252–5258.

(64) Kristensen, K.; Høyvik, I.-M.; Jansik, B.; Jørgensen, P.; Kjærgaard, T.; Reine, S.; Jakowski, J. MP2 energy and density for large molecular systems with internal error control using the divide-expand-consolidate scheme. *Phys. Chem. Chem. Phys.* **2012**, *14*, 15706–15714.

(65) Kristensen, K.; Kjærgaard, T.; Høyvik, I.-M.; Ettenhuber, P.; Jørgensen, P.; Jansik, B.; Reine, S.; Jakowski, J. The divide–expand–consolidate MP2 scheme goes massively parallel. *Mol. Phys.* **2013**, *111*, 1196–1210.

(66) Baudin, P.; Ettenhuber, P.; Reine, S.; Kristensen, K.; Kjærgaard, T. Efficient linear-scaling second-order Møller-Plesset perturbation theory: The divide–expand–consolidate RI-MP2 model. *J. Chem. Phys.* **2016**, *144*, 054102.

(67) Kjærgaard, T.; Baudin, P.; Bykov, D.; Eriksen, J. J.; Ettenhuber, P.; Kristensen, K.; Larkin, J.; Liakh, D.; Pawlowski, F.; Vose, A.; others. Massively parallel and linear-scaling algorithm for second-order Møller–Plesset perturbation theory applied to the study of supramolecular wires. *Comput. Phys. Commun.* **2017**, *212*, 152–160.

(68) Li, W.; Guo, Y.; Li, S. A refined cluster-in-molecule local correlation approach for predicting the relative energies of large systems. *Phys. Chem. Chem. Phys.* **2012**, *14*, 7854–7862.

(69) Li, W.; Ni, Z.; Li, S. Cluster-in-molecule local correlation method for post-Hartree–Fock calculations of large systems. *Mol. Phys.* **2016**, *114*, 1447–1460.

(70) Ni, Z.; Li, W.; Li, S. Fully optimized implementation of the cluster-in-molecule local

correlation approach for electron correlation calculations of large systems. *J. Comput. Chem.* **2019**, *40*, 1130–1140.

(71) Ni, Z.; Guo, Y.; Neese, F.; Li, W.; Li, S. Cluster-in-molecule local correlation method with an accurate distant pair correction for large systems. *J. Chem. Theory Comput.* **2021**, *17*, 756–766.

(72) Neugebauer, H.; Pinski, P.; Grimme, S.; Neese, F.; Bursch, M. Assessment of DLPNO-MP2 approximations in double-hybrid DFT. *J. Chem. Theory Comput.* **2023**, *19*, 7695–7703.

(73) Walker, D. W.; Dongarra, J. J. MPI: a standard message passing interface. *Supercomputer* **1996**, *12*, 56–68.

(74) Riplinger, C.; Pinski, P.; Becker, U.; Valeev, E. F.; Neese, F. Sparse maps—A systematic infrastructure for reduced-scaling electronic structure methods. II. Linear scaling domain based pair natural orbital coupled cluster theory. *J. Chem. Phys.* **2016**, *144*, 024109.

(75) Blackford, L. S.; Choi, J.; Cleary, A.; D'Azevedo, E.; Demmel, J.; Dhillon, I.; Dongarra, J.; Hammarling, S.; Henry, G.; Petitet, A.; others *ScaLAPACK users' guide*; SIAM, 1997.

(76) Shewchuk, J. R.; others. An introduction to the conjugate gradient method without the agonizing pain. 1994.

(77) Epifanovsky, E.; Gilbert, A. T.; Feng, X.; Lee, J.; Mao, Y.; Mardirossian, N.; Pokhilko, P.; White, A. F.; Coons, M. P.; Dempwolff, A. L.; others. Software for the frontiers of quantum chemistry: An overview of developments in the Q-Chem 5 package. *J. Chem. Phys.* **2021**, *155*, 084801.

(78) Kaufmann, S.; Homer, B. Craypat-Cray X1 performance analysis tool. Cray User Group (May 2003), 2003.

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

(80) Maurer, S. A.; Lambrecht, D. S.; Flaig, D.; Ochsenfeld, C. Distance-dependent Schwarz-based integral estimates for two-electron integrals: Reliable tightness vs. rigorous upper bounds. *J. Chem. Phys.* **2012**, *136*.

(81) Fukuzawa, K.; Watanabe, C.; Kurisaki, I.; Taguchi, N.; Mochizuki, Y.; Nakano, T.; Tanaka, S.; Komeiji, Y. Accuracy of the fragment molecular orbital (FMO) calculations for DNA: Total energy, molecular orbital, and inter-fragment interaction energy. *Comput. Theor. Chem.* **2014**, *1034*, 7–16.