Title
A General Sparse Tensor Framework for Electronic Structure Theory

Journal
Journal of Chemical Theory and Computation, 13(3)

ISSN
1549-9618

Authors
Manzer, Samuel
Epifanovsky, Evgeny
Krylov, Anna I
et al.

Publication Date
2017-03-14

DOI
10.1021/acs.jctc.6b00853

Peer reviewed

A General Sparse Tensor Framework for Electronic Structure Theory

Samuel Manzer,[†] Evgeny Epifanovsky,[‡] Anna I. Krylov,[¶] and Martin Head-Gordon[*,†]

† *Kenneth S. Pitzer Center for Theoretical Chemistry, Department of Chemistry, University of California, Berkeley, and, Chemical Sciences Division, Lawrence Berkeley National Laboratory, Berkeley CA 94720, USA.*

‡*Q-Chem Inc, 6601 Owens Drive, Suite 105, Pleasanton, California 94588, USA*

¶*Department of Chemistry, University of Southern California, Los Angeles, California 90089, USA*

E-mail: mhg@cchem.berkeley.edu

Abstract

Linear-scaling algorithms must be developed in order to extend the domain of applicability of electronic structure theory to molecules of any desired size. However, the increasing complexity of modern linear-scaling methods makes code development and maintenance a significant challenge. A major contributor to this difficulty is the lack of robust software abstractions for handling block-sparse tensor operations. We therefore report the development of a highly efficient symbolic block-sparse tensor library in order to provide access to high-level software constructs to treat such problems. Our implementation supports arbitrary multi-dimensional sparsity in all input and output tensors. We avoid cumbersome machine-generated code by implementing all functionality as a high-level symbolic C++ language library, and demonstrate that our

implementation attains very high performance for linear-scaling sparse tensor contractions.

1 Introduction

Block-sparse tensors are central mathematical objects in electronic structure theory, the quintessential example of which is the four-center, two-electron integral tensor. The sparsity of the two-electron integral tensor and its blocked structure arise from the application of common integral screening algorithms, the cornerstone of modern self-consistent field (SCF) calculations.[1,2] More expensive correlated electronic structure methods transform the two-electron integrals into the molecular orbital basis, requiring additional approximations to preserve sparsity in the computation.[3–5] Such codes can become very cumbersome to develop and maintain.

Consider, for example, a universally sparse transformation of the three-center atomic orbital basis electron repulsion integrals, $(\mu\nu|Q)$, into a localized occupied[6,7] basis and virtual molecular orbital[8,9] basis (indices i and a below), a common step in local correlation methods:[8]

$$(ia|Q) = \sum_{\mu\in\{\mu\}_i} \sum_{\nu\in\{\nu\}_a} (\mu\nu|Q)\, c_{\mu i} c_{\nu a} \tag{1}$$

where Q is an auxiliary basis index, as used in the resolution of the identity (RI) approximation,[10,11] and the AO integrals are explicitly given as:

$$(\mu\nu|Q) = \int d\mathbf{r}_1 \int d\mathbf{r}_2 \omega_\mu(\mathbf{r}_1)\omega_\nu(\mathbf{r}_1)\, {\mathbf{r}_{12}}^{-1} \omega_Q(\mathbf{r}_2) \tag{2}$$

The restricted domain of the summations in Eq. 1 indicates that the transformations are sparse. When one index determines the range of another index, as the chosen molecular orbital index i controls which μ index entries are non-zero and thus must be traversed by

the summation, we say that these two indices are *coupled*. Any number of indices may be coupled together for varying physical reasons; for example, the μ, ν, i, and a indices from Eq. 1 are all coupled. The pairs (μ, i) and (ν, a) are coupled by the sparsity of the molecular orbital coefficients, while the pair (μ, ν) is coupled by atomic orbital overlap sparsity.

When a code consists of a long series of such sparse operations, including many different coupled sparse indices with varying types of sparsity, performing such simple refactoring operations as changing the storage order of the indices in several of the contractions becomes an extremely difficult task. While the RI approximation is applied in many contexts in order to reduce the dimensionality of the integral tensors, the developer is still forced to work with tensors containing at least three indices. The increasing sophistication of modern electronic structure methods makes the development of software infrastructure to efficiently handle these objects ever more necessary. While historically most electronic structure methods have been laboriously implemented by manually composing loops and matrix operations, interest in more suitable software paradigms has been steadily growing.

The early work of Windus and Pople[12] is perhaps the first recognizably modern approach to tensors in electronic structure software. In more recent times, the development of tensor infrastructure has been dominated by the Tensor Contraction Engine (TCE) project.[13] The capabilities of the TCE span the entire process of tensor expression derivation, factorization, and code generation.[13,14] The TCE has been used to implement extremely complex electronic structure methods, with a particular emphasis on high-order coupled cluster methods.[13,15,16] In an alternate line of development, Epifanovsky et al. have developed the libtensor project, a high-level symbolic C++ library to describe block tensor operations,[17] , following an early implementation of the C++ block-tensor library powering coupled-cluster methods in Q-Chem since 1998. A similar library project with a greater emphasis on concurrency is the Cyclops Tensor Framework of Solomonik et al.[18] Indeed, libtensor was recently extended to massively parallel architectures via fusion with Cyclops. Finally, the TiledArray project of Valeev and Calvin is an actively developed high-level library.[19]

Quantum chemistry has always had a pressing need for software tools to treat sparse tensors, which appear in virtually all reduced-scaling implementations of electron-correlation methods, and also in recently developed high-performance mean-field methods.[20,21] The TCE was designed to exploit sparsity arising from spatial, spin, and index-permutational symmetries.[13] These TCE features have been employed to design sophisticated treatments of quantum chemical problems that make effective use of these types of sparsity.[22] Other early efforts to provide rapid software prototyping capabilities for certain classes of sparse problems[23,24] came from Parkhill and Head-Gordon.[25] This was followed later by a different formulation from Kats and Manby.[26] Recently, there has been a surge of interest in the area of sparse tensor tools. Pinski et al. have formalized the nomenclature and provided an implementation of several commonly used sparse operations.[27] Lewis et al. have described a block-sparse tensor library that also derives cost savings from exploitation of rank sparsity within individual blocks.[28] Finally, Solomonik et al. have extended the Cyclops Tensor Framework to handle a degree of sparsity, and demonstrated this capability in the case of massively parallel contraction of a sparse tensor with a dense tensor.[29]

In this work, we shall describe our block-sparse tensor library, an extension of the libtensor project.[17] We shall provide the first detailed description of an implementation capable of handling fully-general multi-dimensional sparsity in tensor contractions. Our library has the unique ability to represent universally sparse contractions, in which the output and all input tensors have uniquely defined sparse structures. This capability is provided with the added benefit of a high-level, symbolic C++ implementation. We shall present concrete numerical evidence that the ability to handle multi-dimensional sparsity provides significant cost reductions relative to utilizing only pair-wise sparsity, and that the low-overhead of our implementation allows very high floating-point performance to be obtained. In addition, we will discuss the algorithms by which our library resolves an inline symbolic tensor expression into nested loops that perform the minimal number of arithmetic operations in a black-box, fully automated fashion.

2 Theory

2.1 Sparsity in Electron Structure Theory

Sparsity occurs in many different forms in various mathematical contexts across the field of electronic structure theory. The simplest, "lowest common denominator" form of sparsity is *element-wise sparsity*, tensor sparsity that occurs without any additional structure or pattern. Element-wise sparsity is tracked using data structures that record significant tensor elements on an individual basis, such as the compressed-sparse-row[30] storage formats. While exploiting such sparsity reduces the operation count of electronic structure computations, it is advantageous to take advantage of higher-level sparse structure when possible.[31,32]

Many of the most crucial tensors in electronic structure theory exhibit *block sparsity*, in which the significant tensor elements are clustered together in blocks of varying size and dimensionality. Perhaps the most famous manifestation of block sparsity in the context of electronic structure theory is found in the two-electron integral tensor. The two-electron integrals have a natural block-sparse structure that arises from the shell structure of widely-used Gaussian basis sets. The molecular orbital coefficient matrices in localized orbital basis sets also exhibit block-sparse structure due to the use of atom-centered functions in the atomic orbital basis. After the application of appropriate numerical thresholds to define the block-sparse structure of such *root tensors*, the sparse structure of subsequent tensors may be derived using methods that we shall discuss. We summarize by noting that due to the common occurrence of "atom-centered function groups that decay with distance," block-sparse tensors of varying forms are ubiquitous in electronic structure theory.

Electronic structure tensors often also contain higher-level "structured sparsity", which can arise for very different reasons than the decay-with-distance phenomena described above. Point group and spin symmetries give rise to significant sparsity in many electronic structure algorithms, and general purpose libraries have been devised to exploit these properties in a wide class of computations.[17] While the focus of this work is on the exploitation of small-

block sparsity such as that found in the two-electron integral tensor, the algorithms discussed herein can also be used to treat more structured sparsity through appropriate block definition.

2.2 Block-Sparse Tensor Storage Format

Unfortunately, implementation of block-sparse tensor contractions has historically required time-consuming and error-prone manual coding of each individual contraction. However, several recent high-level library projects have utilized block sparsity in some form,[26,28] and our work continues in this vein. In order to obtain high floating point efficiency through the use of existing highly optimized matrix multiply routines, our library stores tensors using the well-established "block-major order."[12,17,26,28]

In order to give a clear definition of "block-major order," we wish to clarify several terms. Throughout this document, we will use the terms "index" and "subspace" somewhat interchangeably. In general, a subspace of a tensor is a one dimensional object with a certain dimension (size) and with a particular blocking pattern. An index is a label applied to the elements of a subspace, and indices are grouped inside parentheses to indicate that the subspaces that they label are coupled by sparsity in some way. To illustrate our usage of these terms, we would say that the tensor $I_{(\mu\nu)Q}$ is composed of three subspaces, and that the indices μ and ν are coupled to form a sparse *index group*. An index group is either a single dense index or any set of indices coupled by sparsity to one another. We will only drop the parentheses convention for sparse index groups in the case of tensors with superscripted indices.

We now define block-major order as follows. First, the subspaces of the tensor are subdivided into blocks; for example, in our implementation of the PARI-K algorithm,[21] indices corresponding to atomic-orbitals are subdivided by shell (a set of Gaussians with the same center and contraction scheme), those corresponding to auxiliary basis indices are subdivided by atom, and occupied molecular-orbital basis indices are not subdivided at all and referenced as a single block. A given block of the tensor is specified by a tuple

of block indices, hereafter referred to as an "index tuple." Elements of the tensor whose absolute indices fall within the bounds of a given index tuple are stored contiguously in memory. Only non-zero blocks are stored. By convention, the distance between tensor elements corresponding to consecutive block index values of a given subspace decreases with increasing subspace index (this is analogous to "row-major" order for matrices). An example of a tensor stored in this format is shown in Fig. 1.

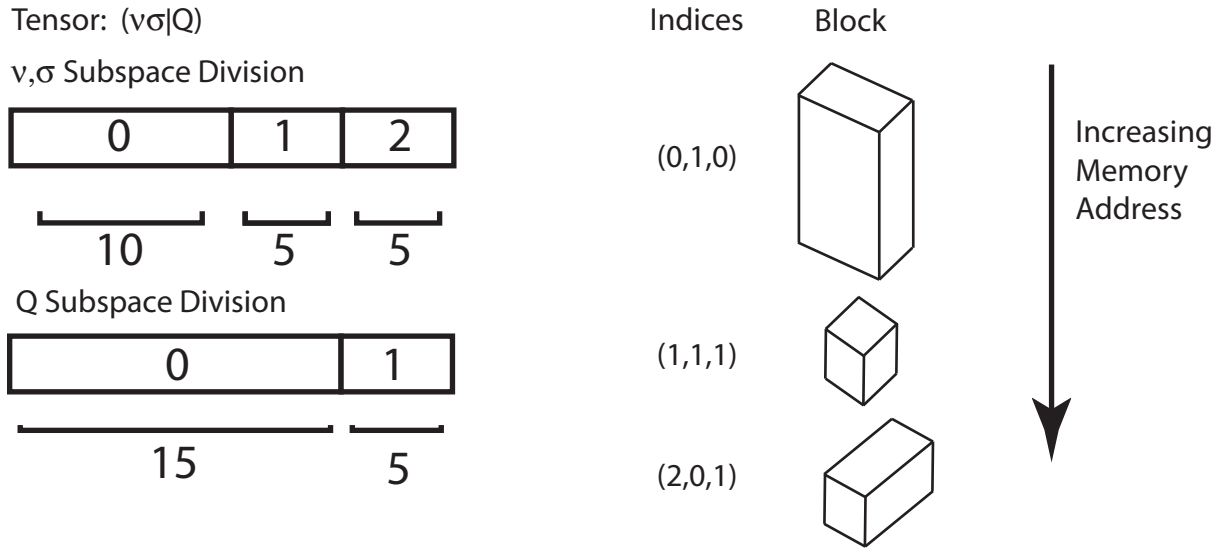

Figure 1: An example of the block-major sparse tensor storage format used in our library. The example is a third rank tensor of AO integrals (i.e. like Eq. 2) with formal dimensions of (20,20,20). As shown on the left, the ranges of the first two indexes (ν and σ) are divided into 3 blocks (of size 10, 5, and 5 respectively), while the third index (Q) is divided into 2 blocks of size 15 and 5. In this fictional tensor whose 8000 elements are divided into 18 blocks, only the three index tuples (0,1,0),(1,1,1), and (2,0,1) correspond to non-zero blocks. The sizes of the three 3 non-zero blocks are illustrated on the right, and they are stored consecutively in memory.

It is somewhat more cumbersome to compute offsets into block-major tensors than typical "row-major" or "column-major" arrays, and this difficulty is compounded when said tensors are sparse. We detail our offset-computation algorithm fully in Supporing Information, but here note only that there are many different efficient approaches through which one can compute the offsets of blocks in sparse tensors, including simply lookup of index tuples using a hash table. Given this fact, in the algorithms described in the following sections, we simply

refer generically to the process of "computing block offsets."

2.3 Tensor Sparsity

Historically, the exploitation of sparsity in electronic structure has focused on pair-wise sparse couplings between indices - for example, the overlap sparsity between two atomic orbital indices $\mu\lambda$ in the two-electron integrals,[33] or the locality of the expansion coefficients $c_{(\mu i)}$ of local molecular orbitals.[34] Pinski et al.[27] and Riplinger et al.[35] have discussed methods for tracking the sparsity relationships between two indices, using a "sparse map" data structure which they describe as two nested lists.[35] While the infrastructure they propose is powerful, its pair-wise nature leads to certain inevitable limitations. For example, in order to formulate a local RI expansion using coefficients $C_{i\overline{\mu}Q}$ of the localized occupied-virtual pair $i\overline{\mu}$, Pinski et al. use a pair-wise sparse map between i and the auxiliary basis index Q. However, it would be more natural in this context to map the $i\overline{\mu}$ pair itself to a unique local subset of auxiliary basis functions.

We shall discuss why it is necessary in many contexts to utilize a higher-dimensional description, which we shall refer to as "tensor sparsity," in order to capture the full measure of sparsity in many of the tensors appearing in electronic structure. As a strong example, we consider the fit coefficients, $C_{(\mu\lambda Q)}$ arising in the pair resolution of the identity approximation (PARI),[20] in which a product of Gaussians, $|\mu_A\lambda_B)$ is approximately represented by a fit, $|\widetilde{\mu_A\lambda_B})$, which is an expansion in terms of the union of the sets of auxiliary basis functions centered on the respective atoms, A and B, of the two functions:

$$|\widetilde{\mu_A\lambda_B}) = \sum_{Q}^{A\cup B} C_{(\mu\lambda Q)}|Q) \tag{3}$$

Clearly, the sparse structure of this tensor cannot be described simply by pair-wise maps; the range of the Q index depends upon both other indices. One might expect that issues of this nature could be handled within the existing pair-wise frameworks by relabeling the pair

of indices $(\mu\nu)$ as a single "combined pair index." However, this does not yield convenient data structures for use in tensor operations in which only one index from the specified index pair couples to the sparsity present in a *different* tensor. We may readily obtain an example of such a problem by considering the transformation of one index of the PARI fit coefficients into a basis of localized molecular orbitals with the MO coefficients $c_{(\lambda j)}$:

$$D_{(\mu jQ)} = \sum_{\lambda} C_{(\mu\lambda Q)} c_{(\lambda j)} \tag{4}$$

For this and many other problems, we propose the use of a *sparse tree* data structure, an example of which is shown in Fig. 2. In this data structure, each non-zero index tuple corresponds to a specific lowest-level node of the tree. The individual block index values for the different indices are nested at each appropriate level; thus the index tuples $(0, 3, 6)$ and $(0, 4, 2)$ nest under the common node corresponding to block index 0 of the first subspace. Each significant value of a given subspace index corresponds to a separate subtree. Thus, the entire list of index tuples may be generated by depth-first traversal of the tree, where at each traversal step the path from the root to the leaf node is the current index tuple. The "list of lists" utilized by Pinski et al.[27] and Riplinger et al.[35] may be regarded as a two-dimensional special case of this data structure in some sense. However, we have found that in order to obtain high performance it is necessary to minimize the number of independent memory allocations involved in the construction of a sparse tree. As such, while we find that conceptualizing our data structure as a tree and referring to it as such yields a better understanding of its fundamental operations, the actual implementation stores the tree as a single contiguous allocation of block index tuples, as shown in Fig. 2.

In a subsequent section, we shall define a set of (nearly) linear-scaling operations on these sparse tree data structures. Using these operational definitions, we shall then prove that no combination of pair-wise sparse maps can recover the full sparsity present in a tensor description; only the tensor description can minimize the number of arithmetic operations

in a sparse problem. We shall provide numerical examples of the size of the cost reduction for practical calculations in our Results section.

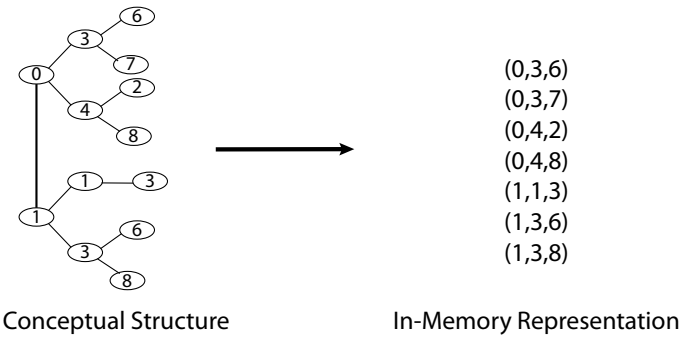

Figure 2: The conceptual representation of our sparse tree data structure for the blocks that contain nonzero elements, and its literal multi-dimensional representation, for an example of a third-rank tensor.

2.4 Sparse operations on tree data structures

We now specify a standardized set of operations that may be performed on these sparse tree data structures. The first operation we wish to discuss is "permutation", in which we permute the order of the levels in the tree. This may be trivially implemented with $O(N \log N)$ scaling by swapping the corresponding entries in each index tuple, followed by re-sorting the index tuples into lexicographic order using standard sorting algorithms. The "inversion" operation of Pinski et al., in which the order of the subspace indices in a pair-wise map is reversed, may be regarded as a special case of tree permutation. An example of the permutation operation is shown in Fig. 3. This operation is used in many situations, particularly to bring all sparsity information in a tensor contraction into agreement with a common loop ordering.

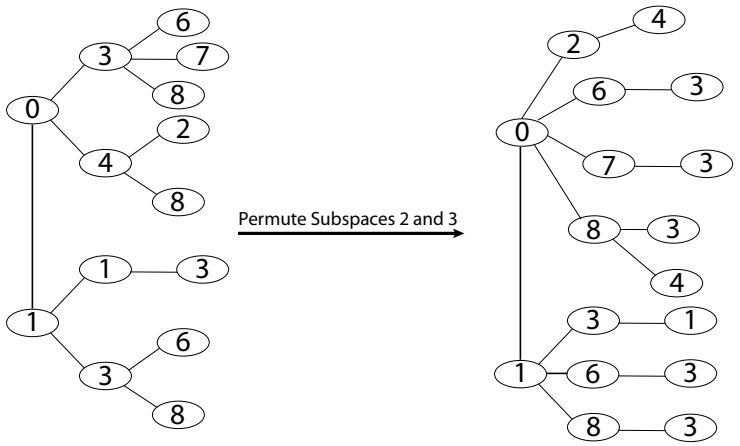

Figure 3: Permutation of indices 1 and 2 (i.e. second and third columns) of a sparse tree.

The second operation that we shall discuss is "fusion", in which the sparsity of two tensors may be combined to yield a higher-dimensional object. The following example illustrates the utility of this operation. Consider a standard sparse matrix product:

$$\mathbf{C}_{(ij)} = \sum_k \mathbf{A}_{(ik)}\mathbf{B}_{(kj)} \tag{5}$$

All indices in the above equation refer to blocks rather than elements, and the parentheses are used to denote non-zero blocks. What is the range of the k index in the inner loop of the contraction in Eq. 5? Clearly, it depends upon the particular iteration of the outer loops over the i and j indices. We may obtain the appropriate set of k index values for a given set of outer indices by *fusing* the trees describing A and B, meaning that we join the lowest level of tree A to the highest level of tree B, as shown in Fig. 4.

This fusion operation bears a resemblance to the "chaining" operation of Pinski et al., in which two pair-wise sparse maps with a common index are used to generate a pair-wise map between their un-shared indices. However, our fusion operation preserves the full multi-dimensional character of all input sparsity, yielding a higher-dimensional tree reflecting the new combined coupling. The space of loop iterations traversed in Eq. (5) corresponds to the elements of the resulting fused tree, and the set of k index blocks touched for given values

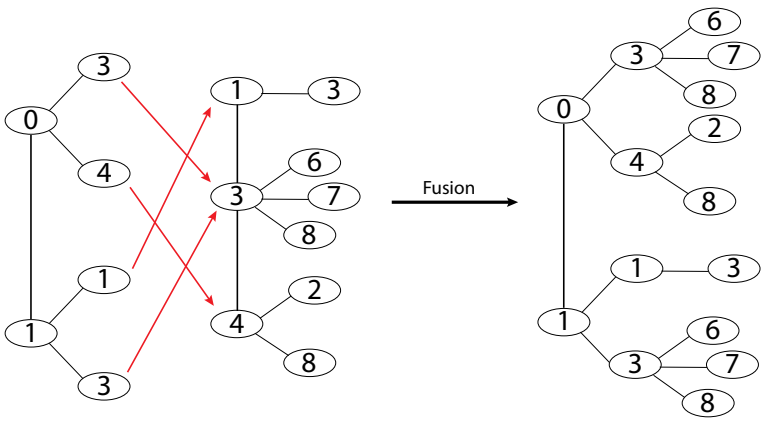

Figure 4: Fusion of two lower-dimensional sparse trees which share a common index to form a higher-dimensional tree. In this example, two two-dimensional tensors with a common index type (lowest level of tree A; highest level of tree B) are fused to make a three-dimensional sparse tensor. One application of the fusion operation can be seen from Eq (5): the fused 3-d tensor represents non-zero block contributions to the sparse matrix multiply.

of the outer loop indices is given by the appropriate set of leaf nodes.

The fusion operation is extended to multiple indices in the following manner: suppose that we wish to fuse two trees, containing m and n indices, respectively, along k shared indices. First, the right-hand side tree is permuted such that the k fused indices occupy the first k positions, while retaining the relative order of the un-fused indices. The process of fusion is then repeated as in Fig. 4, with arrows drawn matching left-hand side tree entries with the right-hand side tree entries whose first k indices match. This results in a new tree of order $n + m - k$.

Finally, we wish to discuss the "reduction" operation. Reduction involves the removal of a given subspace from a tree, yielding a lower dimensional tree. It is implemented in a near-linear scaling fashion in several stages. First, the appropriate entry is deleted from each index tuple. The index tuples are then re-sorted, and the duplicate entries which may have been introduced are removed. An example of the reduction operation is given in Fig. 5.

These three tree structure operations together provide an approach for implementing sparse tensor contraction in a general way. Referring back to Eq. (5) as a simple example,

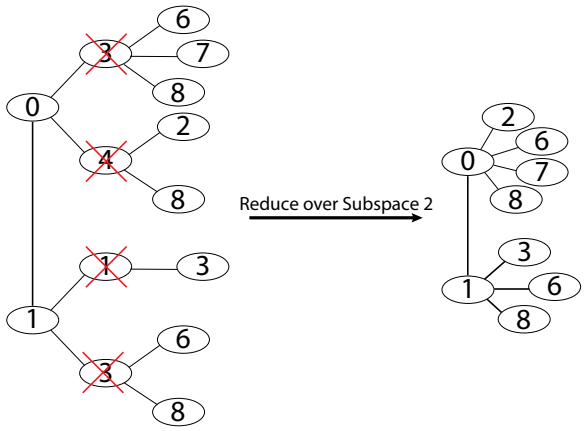

Figure 5: Reduction over the second subspace of a rank-3 tree to form a two-dimensional tree.

one first performs appropriate permutations of $\mathbf{A}$ and $\mathbf{B}$, followed by the fusion of their trees to implicitly define (though not explicitly form) a third rank tensor, $\mathbf{F}_{(ikj)} = \mathbf{A}_{(ik)} * \mathbf{B}_{(kj)}$. Non-zero blocks of the fused tensor implicitly contain element-wise multiplications between the non-zero blocks of $\mathbf{A}$ and $\mathbf{B}$, although we form only the tree describing the sparsity of this tensor rather than the elements themselves. Finally reduction over k blocks defines the block structure of the output product, $\mathbf{C}_{(ij)} = Tr_k\left[\mathbf{F}_{(ikj)}\right]$. While reduction is formally a partial trace (i.e. summation), in practice this recombined with the delayed element-wise multiplications associated with fusion and implemented as matrix multiplication.

2.5 Emergent Sparsity

We now wish to highlight an interesting special case of the exploitation of sparsity in the context of electronic structure theory, though it surely arises in other contexts also. If a given tensor (Tensor 1) is contracted with another tensor (Tensor 2) sharing two or more of its indices, it is clear that we should utilize the sparse coupling between those shared indices, as present in Tensor 2, not only in the latter contraction but also "pre-emptively" in the formation of Tensor 1. By doing so, only a small part of Tensor 1 must be constructed, resulting in great computational savings. If we do not, we would be forming elements of

Tensor 1 which, while they could be numerically significant, would be multiplied by zero values when they were actually utilized in the subsequent contraction. It is thus often advantageous when possible to impose a sparse structure on the output of a tensor contraction (in this example, Tensor 1) that is somewhat independent of the sparse structure of its input tensors.

The above discussion is perhaps a bit abstract, so we shall now provide a real-world example. In a separate publication, we discuss the concurrent use of the occ-RI-K[36] and PARI-K[21] approximations in the context of SCF for molecular interactions (SCF-MI).[37] The SCF-MI method is defined by the construction of a minimum-energy Slater determinant from absolutely localized molecular orbitals (ALMOs).[37] Each ALMO is expanded only in terms of the atomic orbitals residing on its respective molecular fragment; this results in a block-diagonal MO coefficient matrix, and gives many of the tensors in the method additional block sparse structure.

The mixed representation of the Hartree-Fock exchange matrix consisting of occupied orbitals (index i) and atomic orbitals (index ν) needed for the occ-RI-K method using non-orthogonal ALMOs is:

$$K_{i\nu} = \sum_{j} \left({}^{\bullet}_{i} | \nu {}^{j}_{\bullet} \right) \tag{6}$$

We have used the standard superscript notation to indicate that the first occurrence of the j index is contravariant.[38] Our new method utilizes the Dunlap robust fit functional to remove the first order fitting errors associated with the PARI-K approximation:[39]

$$(\mu\lambda|\nu\sigma) \approx (\widetilde{\mu\lambda}|\,\nu\sigma) + (\mu\lambda\,|\widetilde{\nu\sigma}) - (\widetilde{\mu\lambda}\,|\widetilde{\nu\sigma}) \tag{7}$$

In the context of evaluating the exchange matrix, the expression corresponding to the

third term in the Dunlap functional is therefore:

$$\left(\widetilde{i^{\bullet}_{\bullet}{}^{j}} \,|\, \widetilde{\nu j}\right) = \sum_{P} (D'')^{\bullet j}_{i\bullet P} \left(P|Q\right) D_{(\nu j Q)} \tag{8}$$

In this expression, $(D'')^{\bullet j}_{i\bullet P}$ corresponds to the fully MO transformed fit coefficients (with one of the MO indices contravariant due to the non-orthogonality of the absolutely localized molecular orbitals),[37] while $D_{(\nu j Q)}$ corresponds to the half-transformed fit coefficients. The combined expression is evaluated in two steps. In the first step, the fully MO transformed RI fit coefficients are contracted with the (non-sparse) auxiliary basis Coulomb integrals:

$$E^{\bullet j}_{i\bullet Q} = \sum_{P} (D'')^{\bullet j}_{i\bullet P} \left(P|Q\right) \tag{9}$$

This intermediate is Tensor 1 in the context of the earlier abstract discussion. Even if $(D'')^{\bullet j}_{i\bullet P}$ has only a linear number of non-zero elements, because $(P|Q)$ is non-sparse, the product has a quadratic number of significant elements (i.e index Q is uncoupled from i and j).

In the second step, the resulting tensor, $E^{\bullet j}_{i\bullet Q}$, is contracted with the half-transformed fit coefficients (playing the role of Tensor 2) and used to decrement the economized exchange matrix in a contraction that has emergent sparsity:

$$K_{i\nu} \mathrel{-}= \sum_{jQ} E^{\bullet j}_{i\bullet Q} D_{(\nu j Q)} \tag{10}$$

The emergent sparsity comes from the fact that the summed index pair jQ is sparse in $D_{(\nu j Q)}$ (Tensor 2), even though it is dense in $E^{\bullet j}_{i\bullet Q}$. Thus in the contraction, for every j value, only a constant number of Q blocks will be significant. We therefore *impose* this sparse coupling between j and Q on $E^{\bullet j}_{i\bullet Q}$, rendering the first contraction linear-scaling to form just the required blocks.

Due to cases such as this, it desirable that any sparse tensor framework allow the imposition of arbitrary sparse structure on the output of any tensor contraction. Local correlation

models represent a related example where sparsity is imposed based on a model that is far more drastic than would be obtained based on strict numerical thresholding.

2.6 Tensor Sparsity vs. Pair-Wise Sparsity

Utilizing multi-dimensional sparsity brings additional code complexity if not handled appropriately. We therefore wish to provide definitive proof that exploiting sparsity in the full multi-dimensional form can reduce the operation count beyond what is possible through consideration of pair-wise sparsity alone, and at worst will recover the performance of the pair-wise implementation. Once again, we consider as an example the local density fitting expansion for the atomic orbital product $\mu\nu$. The most economical expansion naturally takes the form of some set of auxiliary functions Q centered near the $\mu\nu$ product. However, to avoid the additional complexity of dealing with high-dimensional sparsity, many local fitting codes such as the ARI-K algorithm of Sodt and Head-Gordon[40] simply associate the fit domain of Q values with a single AO index, either μ or ν. The *most economical* pair-wise approximate description of the multi-dimensional sparsity inherent in our RI coefficient tensor is obtained by reduction of the appropriate excluded tree index using the definition of the reduction operation from the previous section. For example, to obtain the (μ, Q) pair-wise map, we may reduce over the ν index of the (μ, ν, Q) tree. The use of any pair-wise map that does not contain all elements resulting from this reduction operation amounts to making further *physical* approximations with respect to the multi-dimensional case - approximations that may or may not be justified.

We consider now the problem of looping over the significant elements of the fit coefficient tensor $C_{(\mu\nu Q)}$ corresponding to the sparse tree shown in Fig. 6. While in this simplified example the simplest solution would be to loop over the tensor elements in order without consideration of sparsity at all, this is clearly not possible if another tensor must be accessed concurrently as, for example, in the case of contraction with another tensor $T_{\nu j}$. If we are restricted to using pair-wise maps within our nested loops, then the minimum-operation

solution that retains the same physical content as the multi-dimensional loop is to run one outer loop over its full index range and restrict every inner loop by the applicable pair-wise sparse map(s) obtained by reduction over the appropriate subspace(s) of the full multi-dimensional tree. We may recover the space of iterations performed by such nested pair-wise loops using our previously defined tree fusion operation. For example, given the loop nesting order μ, ν, Q, we first reduce each subspace of our tree individually to obtain the pair-wise maps (μ, ν), (μ, Q), and (ν, Q). Then we fuse the (μ, ν) and (ν, Q) trees to produce a new tree suitable for looping over all three indices. We then prune the resulting tree by performing *two-index* fusion between it and our (μ, Q) tree. In this case, the two-index fusion amounts to an "intersection" between the two trees, in the same sense as defined by Pinski et al. The resulting tree, the final tree in Fig. 6, represents the full space of iterations that would be performed by a set of nested loops over the pair-wise sparse maps. We have highlighted the red element to indicate an increase in the number of operations relative to the multi-dimensional case that we have started with. We further note that any fused tree composed from pair-wise maps that corresponds to a set of nested loops over all indices in the tensor must contain all entries from the original multi-dimensional tree. The combination of these two facts proves that the cost of the pair-wise implementation is an upper bound to the cost of the multi-dimensional implementation. We shall give concrete numerical examples of the cost savings associated with exploitation of multi-dimensionality in the Results section of this work.

2.7 Automatic Loop Generation: A Fully Symbolic Library

In the interest of providing easily extensible, maintainable implementations of electronic structure methods, we wish to provide a library that does not require the explicit coding of any loops over indices or cumbersome machine generated code. The libtensor[17] library has provided such functionality for some time for *dense* tensors; in the present work, we have extended this library to treat block-sparse tensors. Our library utilizes C++ operator over-

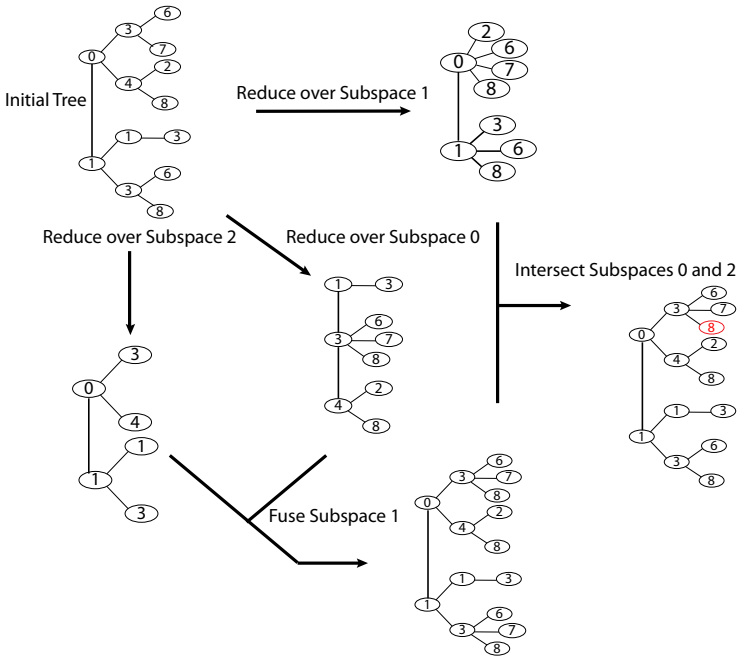

Figure 6: Example of advantage of multi-dimensional sparsity relative to pair-wise sparsity. The red element is erroneously traversed in nested loops over the combined pair-wise sparsity, but not over the multi-dimensional original.

loading to enable parsing of in-code symbolic sparse tensor expressions, allowing problems such as a fully sparse integral transformation to be converted into a single line of C++ code as shown in Fig. 7.

Symbolic Expression:

$$(\nu k|Q) = \sum_{\sigma} (\nu\sigma|Q)\, c_{\sigma k}$$

C++ Code:

```
H(nu|Q|k) = contract(sigma,I(nu|sigma|Q),C_mo(sigma|k));
```

Figure 7: Conversion of symbolic tensor contraction expression to C++ code, as deployed in our development version of Q-Chem.

Our algorithm begins by choosing a loop order that traverses all indices in the problem. The problem of optimally ordering sparse nested loops is very complex. In the interest of simplicity of the initial implementation, we only make one potential rearrangement to the index order given in the contraction expression; namely, we decide whether or not the loops over contracted or uncontracted indices should be the outer loops. We make a crude estimate of the performance tradeoff between these two choices by the comparing the product of the contracted dimensions with the uncontracted dimensions. This will favor reuse of input tensor elements in cases in which many indices are contracted, meaning that the output tensor is much lower order than the input tensors. This procedure is neither sophisticated nor optimal, and a more advanced treatment is desirable. However, we find that we are able to obtain good performance with the present implementation.

Our automatic loop generation code must handle an arbitrary number of indices across both sparse and dense tensors. In order to generate a canonical set of loops over all indices, we must first fuse all coupled sparse trees in the input and output tensors. The algorithm performing this process is summarized in Fig. 8. A given loop will traverse either a single sparse tree produced by the above loop fusion step (and therefore possibly traverse many indices at once) or traverse a single index that is dense in its occurrences in all tensors

in the expression. While one could represent dense tensors as sparse trees with all block combinations included, this would introduce undesirable overhead.

Having obtained a canonical set of sparse trees, we prepare our nested loops to be run in two phases. First, we merge loops over indices that are coupled as members of the same sparse tree (note that they need not all be members of the same tensor) into a "loop group." Universally dense indices each constitute a loop group unto themselves. Once the loop groups have been formed, we match each loop group to the list of all (tensor, index group) pairs that it touches. Finally, we generate the list of offsets into each (tensor,index group) pair for every iteration of the loop group. The loop grouping process is summarized in Fig. 9. It should be noted that our design does not allow interspersing loops over dense indices between loops over coupled sparse indices. This is an undesirable limitation of our library, and one which we are currently working to correct.

Finally, we proceed to execute the nested loops themselves. Our implementation uses a recursive algorithm in which each loop group corresponds to a unique recursion level. Our loop list execution routine takes a [block kernel] parameter, allowing the same loop traversal algorithm to implement many different block-level operations such as permutation and contraction. The algorithm by which we traverse all tensors and perform the requested operations in terms of individual block operations is summarized in Fig. 10.

Initialize [*tree list*], list of all sparse trees in all tensors.
Permute all trees in [*tree list*] to match index loop ordering.
Loop over indices i:
    Form [*trees for loop*], list of [*tree list*] entries touched by i loop
    Loop over *tree* in [*trees for loop*]:
        Fuse *tree* to [*tree list*] entry indexed by [*trees for loop*] first element
        Remove *tree* from [*tree list*]

Figure 8: Algorithm for performing fusion of sparsity across all tensors to form canonical set of trees.

An unfortunate side-affect of this implementation is that the trees representing coupled sparsity across multiple tensors are held explicitly in memory and looped over. These trees

```
Loop over indices i:
    If i belongs to a tree in [tree list]:
        Add i loop to loop group for tree
    Else:
        Create new loop group for dense index i
    Add all (tensor,index group) pairs including i to loop group
Loop over loop groups g:
    Loop over index tuples entries of g:
        Loop over (tensor,index group) pairs touched by g:
            Set appropriate entries in offset arrays
```

Figure 9: Algorithm for grouping loops over indices by sparse coupling.

```
Loop over offset list entries for current loop group:
    Loop over (tensor,index group) pairs touched by loop group:
        Scale [outer size] and set [offset] for (tensor,index group)
    If inner loop in loop group:
        Compute block offsets
        Call [block kernel] on block pointers
    Else:
        Recurse down to next loop group
```

Figure 10: Nested loop execution algorithm for general sparse tensors.

can become very large due to their high dimensionality, in some cases comparable in size to some of the input tensors themselves. We have nonetheless successfully run jobs with over 10000 basis functions in augmented basis sets on nodes with 256-512GB of RAM, demonstrating that our implementation may be used in practical calculations. Work on an alternate formulation of the fusion steps that avoids this relatively high memory usage is in progress.

3 Results and Discussion

3.1 Multi-dimensional Sparsity: Numerical Performance

We shall now provide numerical examples of the advantages of utilizing the full multidimensional index coupling present in higher-order tensors. We examine a series of globular water clusters, an example of which is shown in Fig. 11. For each of these clusters, we compute the half-transformed PARI fit coefficient tensor $D_{(\mu jQ)}$ given in Eq. 4. We then zero all blocks of $D_{(\mu jQ)}$ in which no element of a given block exceeds a threshold of 10^{-6} (based on the second order errors of the Dunlap robust fit[21] we may use such a loose threshold to screen RI fit coefficients).

We now compare the operation count of a set of loops traversing the significant blocks of $D_{(\mu jQ)}$ using the full three-dimensional tree, versus the operation count yielded by using nested pair-wise sparse maps. As discussed in the Theory section, the nested loops over pair-wise sparse maps traverse a set of index tuples corresponding to the entries of a sparse tree created by a two step process; first, we generate three different two-dimensional sparse trees by reducing the full $D_{(\mu jQ)}$ tree over all three different indices. These trees formed by reducing over indices 0,1, and 2 are labeled (jQ),(μQ), and (μj), respectively. We then fuse these trees together in sequence over the appropriate shared indices to recreate a new "pair-wise" tree. As the numerical results in Table 1 demonstrate, the result is that we lose a very large factor of sparsity. Maintaining a fully multi-dimensional description of tensor

sparsity is therefore necessary in order to keep arithmetic operations to a minimum.

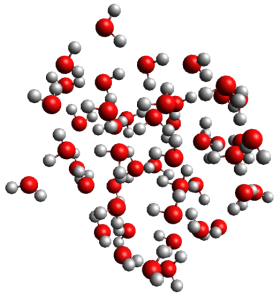

Figure 11: A 46-molecule water cluster used in our performance benchmarks.

Table 1: Percentage of non-zero elements in the half-transformed PARI fit coefficients $D_{(\mu jQ)}$ defined in Eq. 4, illustrated in terms of different sparse trees. Percentages are based on block tuples, not on elements. The discrepancy between the first column and the "pair-wise" column is the sparsity lost by utilizing only a pair-wise representation. The aug-cc-pVTZ[41] basis set was used in conjunction with the cc-pVTZ-JK RI fitting basis set,[42] and the half-transformed coefficients were screened with a threshold of 10^{-6}.

System	$D_{(\mu jQ)}$ % dense	(μj) % dense	(jQ) % dense	(μQ) % dense	pair-wise % dense
Water 46	1.0	68	96	34	28
Water 71	0.6	55	88	28	19
Water 127	0.3	41	71	20	10

3.2 Tensor Contraction Performance

We have already employed the sparse tensor library in this work in the context of implementing two high-performance applications,[21,36] interfacing with the Q-Chem electronic structure program,[43] and are in the process of reporting further applications in a separate publication. In order to highlight the generality of our library, we consider examples of three types of contractions: [sparse] = [sparse] $*$ [sparse], [sparse] = [sparse] $*$ [dense], and [dense] = [sparse] $*$ [sparse] Our chosen examples of these contraction types are chosen from the proceeding SCF-MI equations, and are summarized in Eqs. 11-13 in the order specified above.

$$D_{(\mu jQ)} = \sum_{\lambda} C_{(\mu\lambda Q)} c_{(\lambda j)} \tag{11}$$

$$E^{\bullet j}_{i\bullet Q} = \sum_{P} (D'')^{\bullet j}_{i\bullet P} (P|Q) \tag{12}$$

$$K_{i\nu} -= \sum_{jQ} E^{\bullet j}_{i\bullet Q} D_{(\nu jQ)} \tag{13}$$

As a further illustration of the benefits yielded by the abstract symbolic nature of our library, our implementation uses only the following literal C++ code to implement these three contractions with full exploitation of sparsity:

```
D_full(mu|Q|k) = contract(lambda,C_perm(mu|Q|lambda),C_mo(lambda|k));
E(i|j|Q) = contract(R,Dpp_truncated_full(i|R|j),V(Q|R));
L3(i|nu) = contract(j|Q,E(i|j|Q),D_truncated_full_perm(nu|j|Q));
```

As a test system for benchmarking these tensor contractions, we have chosen linear stacked benzene ring structures such as that shown in Fig. 13. The spacing between consecutive benzene rings is 3.8 Å. We avoid the traditional linear benchmarks such as alkanes in this case because ALMOs are traditionally defined only for non-bonded fragments. For stack lengths of ten, twenty, and thirty rings, we time the first SCF iteration starting from a guess ALMO coefficient matrix consisting of the block-diagonal combination of the monomer MO coefficient matrices. Our timing results are shown in Fig. 12. The floating point performance of each contraction for the largest stack was measured as 1.3 GFLOPS, 4.3 GFLOPS, and 2.0 GFLOPS, respectively. These rates correspond respectively to 25%, 80%, and 38% and of single-core peak floating point performance (5.4 GFLOPS, as gauged by the performance of a 10000x10000 dense matrix multiplication using the Intel MKL library) for the single

AMD Opteron 6376 processor on which these calculations were run. Thus, our tensor library captures the linear scaling behavior inherent in the sparse nature of these structures, and does so while *retaining high floating point performance*. This leaves the SCF-MI calculations dominated by the three-center AO integral calculation, the optimization of which we shall discuss in future work.

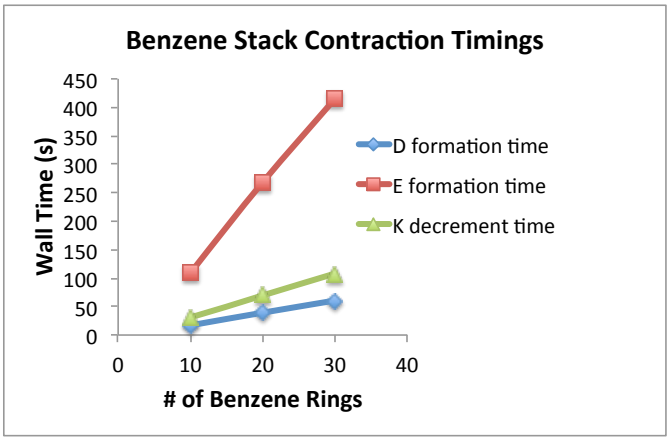

Figure 12: Timings for all three contractions from Eqs. 11-13. Benzene stack calculations used the aug-cc-pVTZ[41] basis set with an integral threshold of 10^{-10}. Reported times are for the first SCF iteration.

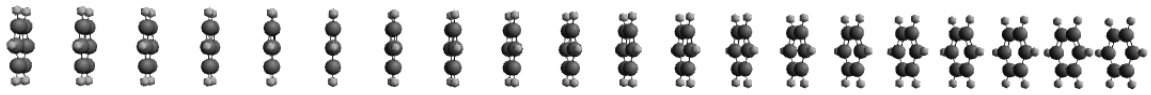

Figure 13: Stacked benzene molecules used to benchmark linear-scaling conractions

The library implementation presented in this work is not yet parallel. We believe the most natural approach to implementing shared-memory parallelism for the operations that we have described is to choose an index of the output tensor and divide its significant blocks among multiple processors. All of the loop generation and execution operations may then be performed as before, with each processor using a set of sparse trees that have been "pruned" so that the divided index falls within the appropriate range specified for that processor. Such parallelization should suffice for calculations of moderate size; extension to parallelization over more indices is technically more difficult but certainly possible.

4 Conclusions

We have described a highly abstract, high performance implementation of common block-sparse tensor operations. Subject to definition of appropriate block structures, our implementation is capable of utilizing truly arbitrary sparsity to reduce operation count without sacrificing floating-point performance. We have demonstrated the capabilities of this library in the context of a high-performance SCF-MI code fully described in a separate publication. The source code for our block-sparse library is available on GitHub as part of the libtensor project at URL https://github.com/epifanovsky/libtensor. Work on adding greater concurrency support to the library is also in progress. We hope that the capabilities of our library can aid in the development of more complex electronic structure methods that either exploit locality or impose local models.

5 Acknowledgements

Initial support for this work was provided by the Scientific Discovery through Advanced Computing (SciDAC) program funded by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Basic Energy Sciences, with additional support from Q-Chem Inc. through an NIH Subcontract from Grant 2 R44 GM096678-02, and a subcontract from MURI Grant W911NF-14-1-0359. MH-G and AIK are part-owners of Q-Chem Inc.

References

(1) Cremer, D.; Gauss, J. *J. Comput. Chem.* **1986**, *7*, 274–282.

(2) Häser, M.; Ahlrichs, R. *J. Comput. Chem.* **1989**, *10*, 104–111.

(3) Bartlett, R. J. *Annu. Rev. Phys. Chem.* **1981**, *32*, 359–401.

(4) Jung, Y.; Shao, Y.; Head-Gordon, M. *J. Comput. Chem.* **2007**, *28*, 1953–1964.

(5) Huntington, L. M. J.; Hansen, A.; Neese, F.; Nooijen, M. *J. Chem. Phys.* **2012**, *136*, 064101.

(6) Boys, S. F. *Rev. Mod. Phys.* **1960**, *32*, 296–299.

(7) Pipek, J.; Mezey, P. G. *J. Chem. Phys.* **1989**, *90*, 4916.

(8) Pulay, P. *Chem. Phys. Lett.* **1983**, *100*, 151–154.

(9) Subotnik, J. E.; Dutoi, A. D.; Head-Gordon, M. *J. Chem. Phys.* **2005**, *123*, 114108.

(10) Vahtras, O.; Almlöf, J.; Feyereisen, M. *Chemical Physics Letters* **1993**, *213*, 514–518.

(11) Feyereisen, M.; Fitzgerald, G.; Komornicki, A. *Chem. Phys. Lett.* **1993**, *208*, 359–363.

(12) Windus, T. L.; Pople, J. A. *Int. J. Quantum Chem.* **1995**, *56*, 485–495.

(13) Hirata, S. *J. Phys. Chem. A* **2003**, *107*, 9887–9897.

(14) Hirata, S. *Theor. Chem. Acc.* **2006**, *116*, 2–17.

(15) Hirata, S. *J. Chem. Phys.* **2004**, *121*, 51.

(16) Shiozaki, T.; Hirao, K.; Hirata, S. *J. Chem. Phys.* **2007**, *126*, 244106.

(17) Epifanovsky, E.; Wormit, M.; Kuś, T.; Landau, A.; Zuev, D.; Khistyaev, K.; Manohar, P.; Kaliman, I.; Dreuw, A.; Krylov, A. I. *J. Comput. Chem.* **2013**, *34*, 2293–2309.

(18) Solomonik, E.; Matthews, D.; Hammond, J. R.; Demmel, J. Cyclops tensor framework: reducing communication and eliminating load imbalance in massively parallel contractions. IEEE International Symposium on Parallel & Distributed Processing (IPDPS). 2013; pp 813–824.

(19) Calvin, J.; Valeev, E. Tiledarray: A massively-parallel,block-sparse tensor library written in C++. https://github.com/valeevgroup/tiledarray/. 2015.

(20) Merlot, P.; Kjærgaard, T.; Helgaker, T.; Lindh, R.; Aquilante, F.; Reine, S.; Pedersen, T. B. *J. Comput. Chem.* **2013**, *34*, 1486–96.

(21) Manzer, S. F.; Epifanovsky, E.; Head-Gordon, M. *J. Chem. Theory Comput.* **2015**, *11*, 518–527.

(22) Baumgartner, G.; Auer, A.; Bernholdt, D. E.; Bibireata, A.; Choppella, V.; Cociorva, D.; Gao, X.; Harrison, R. J.; Hirata, S.; Krishnamoorthy, S.; Krishnan, S.; Lam, C.-C.; Lu, Q.; Nooijen, M.; Pitzer, R. M.; Ramanujam, J.; Sadayappan, P.; Sibiryakov, A. Synthesis of High-Performance Parallel Programs for a Class of Ab Initio Quantum Chemistry Models. Proc. IEEE. 2005; pp 276–292.

(23) Parkhill, J. A.; Lawler, K.; Head-Gordon, M. *J. Chem. Phys.* **2009**, *130*, 084101.

(24) Parkhill, J.; Head-Gordon, M. *J. Chem. Phys.* **2010**, *133*, 024103.

(25) Parkhill, J. A.; Head-Gordon, M. *Mol. Phys.* **2010**, *108*, 513–522.

(26) Kats, D.; Manby, F. R. *J. Chem. Phys.* **2013**, *138*, 144101.

(27) Pinski, P.; Riplinger, C.; Valeev, E. F.; Neese, F. *J. Chem. Phys.* **2015**, *143*, 034108.

(28) Lewis, C. A.; Calvin, J. A.; Valeev, E. F. *arXiv:1510.01156* **2015**, 1–11.

(29) Solomonik, E.; Hoefler, T. *arXiv:1512.00066* **2015**, 1–20.

(30) Gustavson, F. G. *ACM Transactions on Mathematical Software.* **1978**, *4*, 250–269.

(31) Challacombe, M. *Comput. Phys. Comm.* **2000**, *128*, 93–107.

(32) Saravanan, C.; Shao, Y.; Baer, R.; Ross, P. N.; Head-Gordon, M. *J. Comput. Chem.* **2003**, *24*, 618–22.

(33) Strout, D. L.; Scuseria, G. E. *J. Chem. Phys.* **1995**, *102*, 8448.

(34) Maslen, P. E.; Ochsenfeld, C.; White, C. A.; Lee, M. S.; Head-Gordon, M. *J. Phys. Chem. A* **1998**, *102*, 2215–2222.

(35) Riplinger, C.; Pinski, P.; Becker, U.; Valeev, E. F.; Neese, F. *J. Chem. Phys.* **2016**, *144*, 024109.

(36) Manzer, S.; Horn, P. R.; Mardirossian, N.; Head-Gordon, M. *J. Chem. Phys.* **2015**, *143*, 024113.

(37) Khaliullin, R. Z.; Head-Gordon, M.; Bell, A. T. *J. Chem. Phys.* **2006**, *124*, 204105.

(38) Head-Gordon, M.; Lee, M.; Maslen, P.; Voorhis, T. V.; Gwaltney, S. *Modern Methods and Algorithms of Quantum Chemistry.* **2000**, *3*, 593–638.

(39) Dunlap, B. *J. Molec. Struct. (THEOCHEM)* **2000**, *529*, 37–40.

(40) Sodt, A.; Head-Gordon, M. *J. Chem. Phys.* **2008**, *128*, 104106.

(41) Kendall, R. A.; Dunning, T. H.; Harrison, R. J. *J. Chem. Phys.* **1992**, *96*, 6796.

(42) Weigend, F. *Phys. Chem. Chem. Phys.* **2002**, *4*, 4285–4291.

(43) Shao, Y.; Gan, Z.; Epifanovsky, E.; Gilbert, A. T.; Wormit, M.; Kussmann, J.; Lange, A. W.; Behn, A.; Deng, J.; Feng, X.; Ghosh, D.; Goldey, M.; Horn, P. R.; Jacobson, L. D.; Kaliman, I.; Khaliullin, R. Z.; Kuś, T.; Landau, A.; Liu, J.; Proynov, E. I.; Rhee, Y. M.; Richard, R. M.; Rohrdanz, M. A.; Steele, R. P.; Sundstrom, E. J.; Woodcock, H. L.; Zimmerman, P. M.; Zuev, D.; Albrecht, B.; Alguire, E.; Austin, B.; Beran, G. J. O.; Bernard, Y. A.; Berquist, E.; Brandhorst, K.; Bravaya, K. B.; Brown, S. T.; Casanova, D.; Chang, C.-M.; Chen, Y.; Chien, S. H.; Closser, K. D.; Crittenden, D. L.; Diedenhofen, M.; DiStasio, R. A.; Do, H.; Dutoi, A. D.; Edgar, R. G.; Fatehi, S.; Fusti-Molnar, L.; Ghysels, A.; Golubeva-Zadorozhnaya, A.; Gomes, J.;

Hanson-Heine, M. W.; Harbach, P. H.; Hauser, A. W.; Hohenstein, E. G.; Holden, Z. C.; Jagau, T.-C.; Ji, H.; Kaduk, B.; Khistyaev, K.; Kim, J.; Kim, J.; King, R. A.; Klunzinger, P.; Kosenkov, D.; Kowalczyk, T.; Krauter, C. M.; Lao, K. U.; Laurent, A.; Lawler, K. V.; Levchenko, S. V.; Lin, C. Y.; Liu, F.; Livshits, E.; Lochan, R. C.; Luenser, A.; Manohar, P.; Manzer, S. F.; Mao, S.-P.; Mardirossian, N.; Marenich, A. V.; Maurer, S. A.; Mayhall, N. J.; Neuscamman, E.; Oana, C. M.; Olivares-Amaya, R.; ONeill, D. P.; Parkhill, J. A.; Perrine, T. M.; Peverati, R.; Prociuk, A.; Rehn, D. R.; Rosta, E.; Russ, N. J.; Sharada, S. M.; Sharma, S.; Small, D. W.; Sodt, A.; Stein, T.; Stück, D.; Su, Y.-C.; Thom, A. J.; Tsuchimochi, T.; Vanovschi, V.; Vogt, L.; Vydrov, O.; Wang, T.; Watson, M. A.; Wenzel, J.; White, A.; Williams, C. F.; Yang, J.; Yeganeh, S.; Yost, S. R.; You, Z.-Q.; Zhang, I. Y.; Zhang, X.; Zhao, Y.; Brooks, B. R.; Chan, G. K.; Chipman, D. M.; Cramer, C. J.; Goddard, W. A.; Gordon, M. S.; Hehre, W. J.; Klamt, A.; Schaefer, H. F.; Schmidt, M. W.; Sherrill, C. D.; Truhlar, D. G.; Warshel, A.; Xu, X.; Aspuru-Guzik, A.; Baer, R.; Bell, A. T.; Besley, N. A.; Chai, J.-D.; Dreuw, A.; Dunietz, B. D.; Furlani, T. R.; Gwaltney, S. R.; Hsu, C.-P.; Jung, Y.; Kong, J.; Lambrecht, D. S.; Liang, W.; Ochsenfeld, C.; Rassolov, V. A.; Slipchenko, L. V.; Subotnik, J. E.; Van Voorhis, T.; Herbert, J. M.; Krylov, A. I.; Gill, P. M.; Head-Gordon, M. *Mol. Phys.* **2015**, *113*, 184–215.