Cyclops Tensor Framework: reducing communication and eliminating load imbalance in massively parallel contractions

Edgar Solomonik
Devin Matthews
Jeff Hammond
James Demmel

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-11
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-11.html

February 13, 2013

Cyclops Tensor Framework: reducing communication and eliminating load imbalance in massively parallel contractions

Edgar Solomonik
Univ. of California, Berkeley
Dept. EECS
solomon@eecs.berkeley.edu

Devin Matthews
Univ. of Texas, Austin
Dept. Chem. and Biochem.

Jeff R. Hammond
Argonne National Laboratory
Leadership Computing Facility

James Demmel
Univ. of California, Berkeley
Dept. EECS

***Abstract*—Cyclops (cyclic-operations) Tensor Framework (CTF) [1] is a distributed library for tensor contractions. CTF aims to scale high-dimensional tensor contractions such as those required in the Coupled Cluster (CC) electronic structure method to massively-parallel supercomputers. The framework preserves tensor structure by subdividing tensors cyclically, producing a regular parallel decomposition. An internal virtualization layer provides completely general mapping support while maintaining ideal load balance. The mapping framework decides on the best mapping for each tensor contraction at run-time via explicit calculations of memory usage and communication volume. CTF employs a general redistribution kernel, which transposes tensors of any dimension between arbitrary distributed layouts, yet touches each piece of data only once. Sequential symmetric contractions are reduced to matrix multiplication calls via tensor index transpositions and partial unpacking. The user-level interface elegantly expresses arbitrary-dimensional generalized tensor contractions in the form of a domain specific language. We demonstrate performance of CC with single and double excitations on 8192 nodes of Blue Gene/Q and show that CTF outperforms NWChem on Cray XE6 supercomputers for benchmarked systems.**

I. Introduction

Quantum chemistry is the field of science focused on the application of quantum mechanics to the study of chemical problems. While far from the only tool used to study such problems, quantum chemistry plays a role in elucidating the design of new materials for energy capture and storage, the mechanisms of combustion and atmospheric processes, and the interaction of molecules with many kinds of radiation, which is fundamental to probing many forms of matter at the atomistic scale. A major barrier to the application of *all* quantum chemistry methods is their steep computational cost. As a result, high-performance computers have been used for computational quantum chemistry for more than 40 years and enormous effort has been put into designing algorithms and developing software that enables the efficient use of such resources. Among the most common methods of quantum chemistry are quantum many-body (QMB) methods, which attempt to explicitly solve the Schrödinger equation using a variety of ansätze. The explicit treatment of electrons in molecules leads to a steep computational cost, which is nonetheless often of polynomial complexity, but with the benefit of systematic improvement via a series of related ansätze. The Coupled Cluster (CC) family of methods [1], [2] is currently the most popular QMB method in chemistry due to its high accuracy, polynomial time and space complexity, and systematic improvability. This paper focuses on the fundamental kernels of Coupled Cluster – tensor contractions – and demonstrates a completely new algorithmic approach that has the potential to enable these applications on state-of-the-art architectures while achieving a high degree of efficiency in computation, communication and storage.

We present a general parallel decomposition of tensors that efficiently decomposes packed tensor data with any partial or full tensor symmetry of any dimension. Our mapping algorithms rearrange the tensor layouts to suit any given tensor contraction. This tensor decomposition needs minimal padding and has a regular decomposition that allows the algorithm to be mapped to a physical network topology and executed with no load imbalance. Since this decomposition is completely regular, we automatically search over many possible mappings and decompositions at run-time, explicitly calculating the memory usage, communication cost, and padding they require and selecting the best one. Our mapping framework considers mappings that unpack or replicate the tensor data, which yields theoretically optimal communication cost.

We implemented these mapping algorithms in Cyclops Tensor Framework (CTF), a distributed tensor contraction library. We expose the generality of this framework via an elegant interface that closely corresponds to Einstein notation, capable of performing arbitrary dimensional contractions on symmetric tensors. This interface is a domain specific language well-suited for theoretical chemists. To demonstrate correctness and performance we implemented a Coupled Cluster method with single and double excitations using this infrastructure.

The contributions of this paper are

- a communication-optimal tensor contraction algorithm
- a cyclic tensor decomposition for symmetric tensors
- an automatic topology-aware mapping framework

[1]Software and documentation publicly available under a BSD license: http://www.eecs.berkeley.edu/~solomon/cyclopstf/index.html

- a load-balanced virtualization scheme
- a scalable implementation of Coupled Cluster

In this paper, we will start by giving a brief overview of Coupled Cluster, then detail related work. We define the Coupled Cluster Singles and Doubles (CCSD) equations that are realized by our implementation. After presenting this theory, we will discuss the algorithm we use to parallelize the CCSD tensor contractions. The implementation of the algorithm, Cyclops Tensor Framework, will be detailed. We will present Blue Gene/Q and Cray results for CCSD running on top of CTF, which show good weak scaling and outperform NWChem [3].

II. Background

Coupled Cluster (CC) is a method for computing an approximate solution to the time-independent Schrödinger equation of the form

$$\mathbf{H}|\Psi\rangle = E|\Psi\rangle,$$

where $\mathbf{H}$ is the Hamiltonian, E is the energy, and Ψ is the wave-function. In CC, the approximate wave-function is defined in exponential form

$$|\Psi\rangle = e^{\mathbf{T}}|\Phi_0\rangle$$

where $|\Phi_0\rangle$ is a one-electron reference wave-function, usually a Hartree-Fock Slater determinant. The $\mathbf{T}$ operator in CC has the form

$$\mathbf{T} = \mathbf{T_1} + \mathbf{T_2} + \mathbf{T_3} \dots$$

$$\mathbf{T_n} = \sum_{\substack{a_1 \dots a_n \\ i_1 \dots i_n}} t_{i_1 \dots i_n}^{a_1 \dots a_n} a_{a_1}^\dagger \dots a_{a_n}^\dagger a_{i_n} \dots a_{i_1}$$

where $\mathbf{T_n} \equiv \{t_{i_1 \dots i_n}^{a_1 \dots a_n}\}$ is a $2n^{th}$ rank (dimension) [2] tensor representing the set of amplitudes for all possible excitations of n electrons from occupied orbitals in the reference to virtual (unoccupied) orbitals. Each $\mathbf{T}_n$ is computed via a series of tensor contractions on tensors of rank $r \in \{2, 4, \dots 2n+2\}$. The specific tensor contractions depend on the variation of CC and can be derived by various algebraic or diagrammatic methods [4]. Using the truncated operator $\mathbf{T} = \mathbf{T_1} + \mathbf{T_2}$ gives the method commonly known as CCSD (Coupled Cluster Singles and Doubles) [5]. Removal of $\mathbf{T_1}$ gives the basic CCD method, while further addition of $\mathbf{T_3}$ gives the CCSDT (T - triples) method [6], [7] and $\mathbf{T_4}$ gives the CCSDTQ (Q - quadruples) method [8].

Computationally, tensor contractions can be reduced to matrix multiplication via index reordering (transposes). This approach is efficient and commonly used for contractions on fully dense tensors. However, the tensors which arise in CC usually have high-dimensional structure. In particular, permutational symmetry or skew-symmetry (anti-symmetry) among a set of indices implies that any reordering of the index set within the tensor will give the same value (with a potential sign change for anti-symmetry). For example, elements of the 2-particle Hamiltonian v_{ij}^{ab} are skew-symmetric in a, b and in i, j. This permutation symmetry arises from the requirement that the wave-function for fermions (bosons) be antisymmetric (symmetric) under the interchange of particles. So, we have

$$v_{ij}^{ab} = -v_{ij}^{ba} = v_{ji}^{ba} = -v_{ji}^{ab}$$

where the unique part of the tensor needs to be stored is $v_{i<j}^{a<b}$. The coupled cluster amplitudes T_n are also skew-symmetric for some spins, and can have symmetries among groups of n indices.

In general, permutational symmetry of n indices implies that only one of every $n!$ values in the full tensor is unique. This implies that it suffices to store only $1/n!$ of the tensor data. In higher-order methods such as CCSDT and CCSDTQ, which have 3-dimensional and 4-dimensional symmetries as well as multiple symmetric index groups in some tensors, the storage reduction provided by exploiting symmetry is significant (4-36 times less for various tensors in CCSDT and 16-576 times less for CCSDTQ). Further, any symmetry preserved within a contraction (e.g. the output C contains indices that were symmetric in operands A or B), reduces the computational cost relative to a non-symmetric contraction.

The challenge in exploiting high-dimensional symmetry is that the contractions can no longer be trivially reduced to dense matrix multiplication. Further, since the number of possible as well as encountered (partial) permutational symmetries grows factorially with tensor dimension, it is difficult to generalize and tiresome to specialize. As a result, most implementations exploit tensor symmetry to a limited extent and perform redundant work and communication by unpacking or padding tensors.

III. Previous work

We provide an overview of existing applications and known algorithms for distributed memory CC and tensor contractions. We also discuss parallel numerical linear algebra algorithms, in particular 2.5D algorithms [9], which will serve as a key motivation for the design of Cyclops Tensor Framework.

A. NWChem and TCE

NWChem [3] is a computational chemistry software package developed for massively parallel systems. NWChem includes implementations of CC and tensor contractions, which are of interest in our analysis. We will detail the parallelization scheme used inside NWChem and use it as a basis of comparison for the Cyclops Tensor Framework design.

NWChem uses the Tensor Contraction Engine (TCE) [10], [11], [12], to automatically generate sequences of tensor contractions based on a diagrammatic representation of CC schemes. TCE attempts to form the most efficient sequence of contractions while minimizing memory usage of intermediates (computed tensors that are neither inputs nor outputs). We note that TCE or a similar framework can function with any distributed library which actually executes the contractions. Thus, TCE can be combined with Cyclops Tensor Framework since they are largely orthogonal components. However, the tuning decisions done by such a contraction-generation layer should

[2] We will use the term dimension to refer to tensor rank or order.

be coupled with performance and memory usage models of the underlying contraction framework. In addition, one of the present authors is working on a new, more flexible generator for CC contractions which could be more tightly coupled to CTF.

To parallelize and execute each individual contraction, NWChem employs the Global Arrays (GA) framework [13]. Global Arrays is a partitioned global-address space model (PGAS) and allows processors to access (fetch) data which may be laid out physically on a different processor. Data movement within GA is performed via one-sided communication, thereby avoiding synchronization among communicating nodes, while fetching distributed data on-demand. NWChem performs different block tensor sub-contractions on all processors using GA as the underlying communication layer to satisfy dependencies and obtain the correct blocks. Since this dynamically scheduled scheme is not load balanced, NWChem uses dynamic load balancing among the processors. Further, since distribution is hidden by GA, the communication pattern is irregular and possibly unbalanced. Cyclops Tensor Framework attempts to eliminate the scalability bottlenecks of load imbalance and irregular communication, by using a regular decomposition which employs a structured communication pattern well-suited for torus network architectures.

B. ACES III and SIAL

The ACES III package uses the SIAL framework [14], [15] for distributed memory tensor contractions in coupled-cluster theory. Like the NWChem TCE, SIAL uses tiling to extract parallelism from each tensor contraction. However, SIAL has a different runtime approach that does not require active-messages, but rather uses intermittent polling (between tile contractions) to respond to communication requests, so SIAL can be implemented using MPI two-sided communication. To-date, ACES III has not implemented arbitrary-order tensor contractions or methods beyond CCSD(T), so no direct comparison can be made for such cases.

C. MRCC

MRCC [16] is a program suite which performs arbitrary-order calculations for a variety of CC and related methods. Parallelism is enabled to a limited extent by either using a multi-threaded BLAS library or by parallel MPI features of the program. However, the scaling performance is severely limited due to highly unordered access of the data and excessive inter-node communication. MRCC is currently the only tenable solution for performing any type of CC calculation beyond CCSDTQ, and the lack of scalability presents a serious bottleneck in many calculations.

MRCC uses a string-based approach to tensor contractions which originated in the development of Full CI codes. In this method, the tensors are stored using a fully-packed representation, but must be partially unpacked in order for tensor contractions to be performed. The indices of the tensors are then represented by index "strings" that are pre-generated and then looped over to form the final product. The innermost loop contains a small matrix-vector multiply operation (the dimensions of this operation are necessarily small, and become smaller with increasing level of excitation as this loop involves only a small number of the total indices). The structured communication, storage, and contractions algorithms that we propose in the Cyclops Tensor Framework could then present a significant improvement in both raw efficiency and parallelization of tensor contractions relative to MRCC.

D. Lower bounds on communication

Since tensor contractions are closely related to matrix multiplication (MM), it is of much interest to consider the best known distributed algorithms for MM. Ideally, the performance achieved by any given tensor contraction should approach the efficiency of matrix multiplication, and generally the latter is an upper-bound. In particular, we would like to minimize the communication (number of words of data moved across the network by any given processor) done to contract tensors. Since any operation within a tensor contraction maps to three tensors the same lower bound argument that works for matrix multiplication applies to tensor contractions.

Given a matrix multiplication or contraction that requires F/p multiplications on p processors, with M words of memory on each processor, it is known that some processor must communicate at least

$$W = \Omega\left(\frac{F}{p \cdot \sqrt{M}} - M\right) \tag{1}$$

words of data [17], [18], [19]. If the tensor or matrices are of size $S = \Theta(M \cdot p)$, the communication lower bound is

$$W_{2D} = \Omega\left(\frac{F}{\sqrt{p \cdot S}} - \frac{S}{p}\right).$$

We label this lower bound as W_{2D} because it is achieved by matrix multiplication algorithms that are most naturally described on a 2D processor grid. In particular, blocked Cannon's algorithm [20] and SUMMA [21], [22] achieve this communication bandwidth lower bound. We can also see that, assuming the initial data is not replicated and load-balanced, there is an absolute (memory-size insensitive) lower-bound [23], [24],

$$W_{3D} = \Omega\left(\frac{F}{p^{2/3} \cdot \sqrt{S}} - \frac{S}{p}\right)$$

This communication lower-bound can be achieved by performing 3D blocking on the computational graph rather than simply distributing the matrices. An old algorithm known as 3D matrix multiplication has been shown to achieve this communication cost [25], [21], [23], [26].

However, in practice, most applications run with some bounded amount of extra available memory. 2.5D algorithms minimize communication cost for any amount of physical memory. In particular, if all operand tensors or matrices are of size $S = \Theta(M \cdot p/c)$, where $c \in [1, p^{1/3}]$, the communication lower bound is

$$W_{2.5D} = \Omega\left(\frac{F}{\sqrt{p \cdot c \cdot S}} - \frac{S}{p}\right)$$

Using adaptive replication this communication lower-bound can be achieved for matrix multiplication as well as other dense linear algebra kernels via the algorithms presented in [9]. Its also important to note that 2.5D algorithms can map very efficiently to torus network architectures as demonstrated in [27]. We demonstrate an algorithm and an implementation of a tensor contraction framework that does no more communication for each contraction than these lower bounds.

IV. Arbitrary-order Coupled Cluster

Coupled Cluster is an iterative process, where in each iteration, the new set of amplitudes $\mathbf{T}'$ are computed from the amplitudes from the previous iteration $\mathbf{T}$ and from the Hamiltonian $\mathbf{H} = \mathbf{F} + \mathbf{V}$. The diagonal elements of the one-particle Hamiltonian $\mathbf{F}$ are separated out as a factor $\mathbf{D}$, giving a final schematic form similar to a standard Jacobi iteration (although $\mathbf{V}$ still contains diagonal elements),

$$\mathbf{T}' = \mathbf{D}^{-1}\left[(\mathbf{F}' + \mathbf{V})(\mathbf{1} + \mathbf{T} + \frac{1}{2}\mathbf{T}^2 + \frac{1}{6}\mathbf{T}^3 + \frac{1}{24}\mathbf{T}^4)\right].$$

The expansion of the exponential operator is complete at fourth order due to the fact that the Hamiltonian includes only one- and two-particle parts. The specific tensors which compose $\mathbf{F}'$, $\mathbf{V}$, and $\mathbf{T}$ are

$$\begin{aligned}
\mathbf{F}' &= (1-\delta_{ab})f_b^a + f_i^a + f_a^i + (1-\delta_{ij})f_j^i, \\
\mathbf{V} &= v_{cd}^{ab} + v_{ci}^{ab} + v_{bc}^{ai} + v_{bj}^{ai} + v_{ij}^{ab} + \\
&\quad v_{ab}^{ij} + v_{jk}^{ai} + v_{ak}^{ij} + v_{kl}^{ij}, \\
\mathbf{T} &= \mathbf{T_1} + \mathbf{T_2} + \cdots + \mathbf{T_n} \\
&= t_i^a + t_{ij}^{ab} + \cdots + t_{i_1\ldots i_n}^{a_1\ldots a_n},
\end{aligned}$$

where the $abcdef\ldots$ indices refer to virtual orbitals while $ijklmn\ldots$ refer to occupied orbitals. The contractions which must be done can be derived using either algebraic or diagrammatic techniques, however the result is a sequence of contractions such as

$$z_i^a = \frac{1}{2}\sum_{efm} v_{ef}^{am} t_{im}^{ef}, \text{or}$$

$$z_{ij}^{ab} = \frac{1}{4}\sum_{efmn} v_{ef}^{mn} t_{ij}^{ef} t_{mn}^{ab}.$$

Contractions which involve multiple $\mathbf{T}$ tensors are factored into a sequence of contractions involving one or more intermediates, such that each contraction is a binary tensor operation.

The equations, as written above, are termed the "spin-orbital" representation in that the indices are allowed to run over orbitals of either α or β spin, while only amplitudes with certain combinations of spin are technically allowed. Some programs use this representation directly, checking each amplitude or block of amplitudes individually to determine if it is allowed (and hence should be stored and operated upon). However, an alternative approach is the use the spin-integrated equations where each index is explicitly spin-α, $abij\ldots$, or spin-β, $\bar{a}\bar{b}\bar{i}\bar{j}\ldots$. For example, the second contraction above becomes,

$$\begin{aligned}
z_{ij}^{ab} &= \frac{1}{4}\sum_{efmn} v_{ef}^{mn} t_{ij}^{ef} t_{mn}^{ab}, \\
z_{\bar{i}\bar{j}}^{\bar{a}\bar{b}} &= \frac{1}{4}\sum_{\bar{e}\bar{f}\bar{m}\bar{n}} v_{\bar{e}\bar{f}}^{\bar{m}\bar{n}} t_{\bar{i}\bar{j}}^{\bar{e}\bar{f}} t_{\bar{m}\bar{n}}^{\bar{a}\bar{b}}, \\
z_{i\bar{j}}^{a\bar{b}} &= \sum_{e\bar{f}m\bar{n}} v_{e\bar{f}}^{m\bar{n}} t_{i\bar{j}}^{e\bar{f}} t_{m\bar{n}}^{a\bar{b}}.
\end{aligned}$$

While the number of contractions is increased, the total amount of data which must be stored and contracted is reduced compared to a naïve implementation of the spin-orbital method, and without the overhead of explicit spin-checking.

The amplitudes (and f and v integrals) have implicit permutational symmetry. Indices which appear together (meaning either both upper or lower indices of a tensor) and which have the same spin and occupancy may be interchanged to produce an overall minus sign. In practice this allows the amplitudes to be stored using the symmetric packing facilities built into CTF.

A. Interface for Tensor Operations

Cyclops Tensor Framework provides an intuitive domain specific language for performing tensor contractions and other tensor operations. This interface is implemented using operator overloading and templating in C++, with the end result that tensor contractions can be programmed in the exact same syntax as they are defined algebraically,

$$z_{i\bar{j}}^{a\bar{b}} = \sum_{e\bar{f}m\bar{n}} v_{e\bar{f}}^{m\bar{n}} t_{i\bar{j}}^{e\bar{f}} t_{m\bar{n}}^{a\bar{b}}$$

$$\Downarrow$$

```
W["MnIj"] = V["MnEf"] * T["EfIj"];
Z["AbIj"] = W["MnIj"] * T["AbMn"];
```

This interface naturally supports all types of tensor operations, not just contraction. The number and placement of the unique indices implicitly defines the operation or operations which are to be performed. For example, the repetition of an index within an input tensor which does not appear in the output tensor defines a trace over that index. Similarly, an index which appears in all three tensors defines a type of "weighting" operation while an index which appears multiple times in the input and once in the output will operate on diagonal or semi-diagonal elements of the input only. The weighting operation deserves special attention as it is required in CC to produce the new amplitudes $\mathbf{T}'$ from $\mathbf{Z} = \mathbf{H}e^{\mathbf{T}}$,

$$\mathbf{T}' = \mathbf{D}^{-\mathbf{1}}\mathbf{Z}$$

$$\Downarrow$$

```
T["AbIj"] = Dinv["AbIj"] * Z["AbIj"];
```

Additionally, Equation-of-Motion CC (EOM-CC) and many other related techniques have terms that require computation of only the diagonal elements of a tensor contraction or require replication of the result along one or more dimensions, both of which can be expressed easily and succinctly in this interface. For example, the diagonal tensor elements used in EOMIP-CCSD include terms such as,

$$\begin{aligned}\bar{H}^{ai\bar{j}}_{ai\bar{j}} &\leftarrow W^{i\bar{j}}_{i\bar{j}} \quad \text{and} \\ \bar{H}^{ai\bar{j}}_{ai\bar{j}} &\leftarrow \sum_{\bar{e}} v^{i\bar{j}}_{a\bar{e}} t^{a\bar{e}}_{i\bar{j}},\end{aligned}$$

which can be expressed in CTF as,

```
Hbar["AIj"]  += W["IjIj"];
Hbar["AIj"]  += V["IjAe"] * T["AeIj"];
```

B. Application to CCSD

The CCSD model, where $\mathbf{T} = \mathbf{T_1} + \mathbf{T_2}$, is one of the most widely used coupled cluster methods as it provides a good compromise between efficiency and accuracy, and is fairly straightforward to derive and implement. In particular, CCSD is only slightly more computationally expensive than the simpler CCD method [28] but provides greater accuracy, especially for molecular properties such as the gradient and those derived from response theory. Formally, CCD and CCSD have the same leading-order cost: $O(n_o^2 n_v^4)$, where n_o and n_v are the number of occupied and virtual orbitals, respectively.

The spin-orbital equations for CCSD are relatively simple, and are, in factorized form,

$$\begin{aligned}\tau^{ab}_{ij} &= t^{ab}_{ij} + \frac{1}{2} P^a_b P^i_j t^a_i t^b_j, \\ \tilde{F}^m_e &= f^m_e + \sum_{fn} v^{mn}_{ef} t^f_n, \\ \tilde{F}^a_e &= (1-\delta_{ae}) f^a_e - \sum_m \tilde{F}^m_e t^a_m - \frac{1}{2}\sum_{mnf} v^{mn}_{ef} t^{af}_{mn} \\ &+ \sum_{fn} v^{an}_{ef} t^f_n, \\ \tilde{F}^m_i &= (1-\delta_{mi}) f^m_i + \sum_e \tilde{F}^m_e t^e_i + \frac{1}{2}\sum_{nef} v^{mn}_{ef} t^{ef}_{in} \\ &+ \sum_{fn} v^{mn}_{if} t^f_n, \\ \tilde{W}^{mn}_{ei} &= v^{mn}_{ei} + \sum_f v^{mn}_{ef} t^f_i, \\ \tilde{W}^{mn}_{ij} &= v^{mn}_{ij} + P^i_j \sum_e v^{mn}_{ie} t^e_j + \frac{1}{2}\sum_{ef} v^{mn}_{ef} \tau^{ef}_{ij}, \\ \tilde{W}^{am}_{ie} &= v^{am}_{ie} - \sum_n \tilde{W}^{mn}_{ei} t^a_n + \sum_f v^{ma}_{ef} t^f_i \\ &+ \frac{1}{2}\sum_{nf} v^{mn}_{ef} t^{af}_{in}, \\ \tilde{W}^{am}_{ij} &= v^{am}_{ij} + P^i_j \sum_e v^{am}_{ie} t^e_j + \frac{1}{2}\sum_{ef} v^{am}_{ef} \tau^{ef}_{ij}, \\ z^a_i &= f^a_i - \sum_m \tilde{F}^m_i t^a_m + \sum_e f^a_e t^e_i + \sum_{em} v^{ma}_{ei} t^e_m \\ &+ \sum_{em} v^{ae}_{im} \tilde{F}^m_e + \frac{1}{2}\sum_{efm} v^{am}_{ef} \tau^{ef}_{im} - \frac{1}{2}\sum_{emn} \tilde{W}^{mn}_{ei} t^{ea}_{mn}, \\ z^{ab}_{ij} &= v^{ab}_{ij} + P^i_j \sum_e v^{ab}_{ie} t^e_j + P^a_b P^i_j \sum_{me} \tilde{W}^{am}_{ie} t^{eb}_{mj} \\ &- P^a_b \sum_m \tilde{W}^{am}_{ij} t^b_m + P^a_b \sum_e \tilde{F}^a_e t^{eb}_{ij} - P^i_j \sum_m \tilde{F}^m_i t^{ab}_{mj} \\ &+ \frac{1}{2}\sum_{ef} v^{ab}_{ef} \tau^{ef}_{ij} + \frac{1}{2}\sum_{mn} \tilde{W}^{mn}_{ij} \tau^{ab}_{mn}, \\ t^a_i &= \left(D^a_i\right)^{-1} z^a_i, \\ t^{ab}_{ij} &= \left(D^{ab}_{ij}\right)^{-1} z^{ab}_{ij},\end{aligned}$$

where the permutation operator $P^a_b[\ldots a \ldots b \ldots] = [\ldots a \ldots b \ldots] - [\ldots b \ldots a \ldots]$.

Recasting these equations in terms of spin-integrated quantities adds additional contractions. Most terms in the expression for z^a_i and z^{ab}_{ij} are unchanged except for the addition of spin labels and a change in overall factor. However, terms which contain permutation operators become more complicated, especially term 3 of z^{ab}_{ij} (and the equation for its intermediate). The symmetry properties of the amplitudes allow for some simplification, though, as the explicit permutation operators can be implied by the symmetry relations in the specification of the output tensor. For example, if two antisymmetric matrices are multiplied together and the result is then antisymmetrized, the result, in terms of fully packed storage requires six separate operations. All of these operations can be represented by a single contraction call in the CTF interface if the output tensor is specified to have antisymmetry,

$$\begin{aligned}C[ab] &= P^a_b A[ac] \times B[cb], \\ C[a<b] &= \sum_c \big\{ A[a<c]B[c<b] - A[a<c]B[b<c] \\ &\quad - A[c<a]B[c<b] - A[b<c]B[c<a] \\ &\quad + A[b<c]B[a<c] + A[c<b]B[c<a] \big\}\end{aligned}$$

$$\Downarrow$$

```
/* A, B, and C are antisymmetric */
C["ab"] = A["ac"] * B["cb"];
```

An interface layer to automatically produce the necessary spin-integrated contractions has also been implemented, so that the code can be written entirely in terms of the simple spin-orbital quantities. With these simplifications, the total amount of code to perform a single CCSD iteration is only 41 lines.

C. Higher-order Coupled Cluster

Higher order CC methods (CCSDT, CCSDTQ, CCSDTQP, etc.) are theoretically very similar to CCD and CCSD, however, several important computational distinctions arise. First, as the order increases, the highest set of $\mathbf{T_n}$ amplitudes grows relatively much larger than the Hamiltonian elements and the

other $\mathbf{T}$ amplitudes. The computation time in terms of FLOPS is dominated by a handful of contractions involving this largest amplitude set. However, the sheer number of small contractions which must be done in addition can instead dominate the wall time if they are not performed as efficiently or do not parallelize as well. Thus, the efficiency of small tensor contractions and strong-scalability of the parallel algorithm become relatively much more important for higher order CC.

Second, since the total memory and/or disk space available for the computation is effectively constant, high orders of CC necessitate the use of a smaller number of occupied and virtual orbitals. This shrinks the length of each tensor dimension, threatening vectorization and increasing indexing overhead. CTF currently uses a sequential contraction kernel which uses a cyclic blocking with a fixed tile size to avoid vectorization problems for packed tensors. While extremely short edge lengths could still cause excessive overhead in this scheme (due to the padding needed to fill out the fixed-length tiles), good performance should be retained in most circumstances.

V. Tensor decomposition and contraction

We define a tensor contraction between $\mathbf{A} \in \mathbb{R}^{I_1 \times \ldots I_k}$, $\mathbf{B} \in \mathbb{R}^{I_1 \times \ldots I_l}$ into $\mathbf{C} \in \mathbb{R}^{I_1 \times \ldots I_m}$ as

$$c_{i_1 \ldots i_m} = \sum_{j_1 \ldots j_{k+l-m}} a_{i_1 \ldots i_{m-l} j_1 \ldots j_{k+l-m}} \cdot b_{j_1 \ldots j_{k+l-m} i_{m-l+1} \ldots i_m}.$$

Tensor contractions reduce to matrix multiplication via index folding. We define a folding

$$|i_1 \ldots i_n| = \{I_1 \times \ldots I_n\} \rightarrow \left[1 : \prod_{i=1}^{n} I_i\right],$$

for instance $|ijk| = \{i, j, k\} \rightarrow i + I_1 \cdot j + I_1 \cdot I_2 \cdot k$. Any contraction can be folded into matrix multiplication in the following manner,

$$c_{|i_1 \ldots i_{m-l}|, |i_{m-l+1} \ldots i_m|} = \sum_{|j_1 \ldots j_{k+l-m}|} a_{|i_1 \ldots i_{m-l}|, |j_1 \ldots j_{k+l-m}|} \cdot b_{|j_1 \ldots j_{k+l-m}|, |i_{m-l+1} \ldots i_m|}.$$

So here $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ can be treated simply as matrices, albeit, in general, the index ordering may have to be transposed. Tensors can also have symmetry, we denote antisymmetric (skew-symmetric) index groups as

$$t_{[i_1 \ldots i_j \ldots i_k \ldots i_n]} = -t_{[i_1 \ldots i_k \ldots i_j \ldots i_n]}$$

for any $j, k \in [1, n]$. For the purpose of this analysis, we will only treat antisymmetric tensors, for symmetric matrices the non-zero diagonals require more special consideration. Using the notation $I^{\otimes n} = \underbrace{I \times \cdots \times I}_{(n-1)\text{-times}}$, we denote a packed (folded) antisymmetric layout as a map from an index to a interval of size binomial in the tensor edge length

$$|i_1 < i_2 < \ldots i_n| = \{I^{\otimes n}\} \rightarrow \left[1 : \binom{I}{n}\right]$$

so given a simple contraction of antisymmetric tensors, such as,

$$c_{[i_1 \ldots i_{m-l}], [i_{m-l+1} \ldots i_m]} = \sum_{j_1 \ldots j_{k+l-m}} a_{[i_1 \ldots i_{m-l}], [j_1 \ldots j_{k+l-m}]} \cdot b_{[j_1 \ldots j_{k+l-m}], [i_{m-l+1} \ldots i_m]},$$

we can compute it in packed antisymmetric layout via

$$c_{|i_1 < \ldots i_{m-l}|, |i_{m-l+1} < \ldots i_m|} = (k+l-m)! \cdot \sum_{|j_1 < \ldots j_{k+l-m}|} a_{|i_1 < \ldots i_{m-l}||j_1 < \ldots j_{k+l-m}|} \cdot b_{|j_1 < \ldots j_{k+l-m}||i_{m-l+1} < \ldots i_m|}.$$

The above contraction is an example where all symmetries are *preserved*. Any preserved symmetries must be symmetries of the contraction graph $G = (V, E)$ where vertices are triplets defined by,

$$v_{a_1 \ldots a_k b_1 \ldots b_l c_1 \ldots c_m} = (a_{a_1 \ldots a_k}, b_{b_1 \ldots b_l}, c_{c_1 \ldots c_m}).$$

Broken symmetries are symmetries which exists in one of $\mathbf{A}$, $\mathbf{B}$, or $\mathbf{C}$, but not in $\mathbf{G}$. For example, we can consider the contraction

$$c_{[ij]kl} = \sum_{pq} a_{[ij][pq]} \cdot b_{pk[ql]}$$

which corresponds to contraction graph elements $v_{[ij]klpq}$. The symmetry $[ij]$ is preserved but the symmetries $[pq]$ and $[ql]$ are broken. While each preserved contraction allows a reduction in floating point operations, broken contractions allow only preservation of storage. The broken symmetries can be unpacked and the contraction computed as

$$c_{|i<j|kl} = 2 \cdot \sum_{pq} a_{|i<j|pq} \cdot b_{pkql}$$

or the broken symmetries can remain folded, in which case multiple permutations are required,

$$c_{|i<j|kl} = 2 \sum_{|p<q|} a_{|i<j||p<q|} \cdot b_{pk|q<l|} - a_{|i<j||p<q|} \cdot b_{pk|l<q|} - a_{|i<j||q<p|} \cdot b_{pk|q<l|} + a_{|i<j||q<p|} \cdot b_{pk|l<q|}$$

Our framework makes dynamic decisions to unpack broken symmetries in tensors or to perform the packed contraction permutations, based on the amount of memory available. We will show that in either case, our approach is communication-optimal. All preserved symmetries are always kept in packed layout, so no extra computation is preformed.

A. Cyclic tensor decomposition

A blocked distribution implies each processor owns a contiguous piece of the original tensor. In a cyclic distribution, a cyclic phase defines the periodicity of the set of indices whose elements are owned by a single processor. For example, if a vector is distributed cyclically among 4 processors, each processor owns every fourth element of the vector. For a tensor of dimension d, we can define a set of cyclic phases $(p_0, p_1, \ldots, p_{d-1})$, such that processor $P_{i_0, i_1, \ldots, i_{d-1}}$ owns all tensor elements whose index $(j_0, j_1, \ldots, j_{d-1})$ satisfies

$$j_k = i_k \text{mod} (p_k)$$

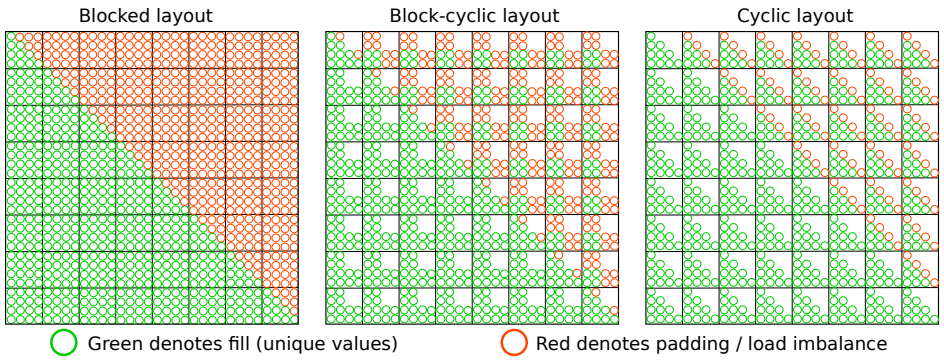


Fig. 1. The load-imbalance incurred or padding necessary for blocked, block-cyclic, and cyclic layouts.

for all $k \in \{0, 1, \ldots, d\}$. A block-cyclic distribution generalizes blocked and cyclic distributions, by distributing contiguous blocks of any size b cyclically among processors. Cyclic decompositions are commonly used in parallel numerical linear algebra algorithms and frameworks such as ScaLAPACK (block-cyclic) [29] and Elemental (cyclic) [30]. Our method extends this decomposition to tensors.

Like matrix multiplication, tensor contractions are invariant with respect to a similarity permutation on A and B,

$$PCP^T = PA \cdot BP^T = (PAP^T) \cdot (PBP^T)$$

This invariance means that we can permute the ordering of rows in columns in a matrix or slices of a tensor, so long as we do it to both A and B and permute PCP^T back to C. This property is particularly useful when considering cyclic and blocked distributions of matrices and tensors. We can define a permutation, P, that permutes a tensor elements from a blocked to a cyclic layout. Conversely, we can run an algorithm on a cyclic distribution and get the answer in a blocked distribution by applying a permutation, or run *exactly* the same algorithm on a blocked distribution and get the cyclic answer by applying a permutation.

The main idea behind Cyclops Tensor Framework is to employ a cyclic distribution to preserve packed symmetric structure in sub-tensors, minimize padding, and generate a completely regular decomposition, susceptible to classical linear algebra optimizations. Each processor owns a cyclic subtensor, where the choice of cyclic phases in each dimension has the same phase for all symmetric indices. By maintaining the same cyclic phase in each dimension, the algorithm ensures that each the sub-tensor owned by any processor has the same structure and structure as the whole tensor. Further, minimal padding on each sub-tensor allows for every subtensor to have the exact same shape only with different entries. Figure 1 demonstrates the difference in padding (or load-imbalance) required to store exactly the same sub-tensors on each processor. It is evident that only a cyclic layout can preserve symmetry as well as maintain load balance. Overall the amount of padding required for CTF is equivalent to setting the block size $b = p^{1/d}$, since we must add up the padding on each processor.

B. Distributed contraction of tensors

Our decomposition gives us tensors in cyclic layouts distributed over a torus topology. These tensors can be replicated over some dimensions, which means blocking of all indices is done. The distributed algorithm for tensor contractions can be efficiently defined as a generalized nested SUMMA algorithm. If the dimensions of two tensors with the same contraction index are mapped onto different torus dimensions, a SUMMA algorithm is done on the plane defined by the two torus dimensions. For each pair of indices mapped in this way, a nested level of SUMMA is done.

The communication cost of the recursive SUMMA algorithm is asymptotically the same as a single SUMMA algorithm done on a matrix of the same size as the tensor distributed on a 2D network with the same number of nodes as the higher-dimensional torus network. SUMMA is known to be communication-optimal given that no extra memory is utilized (weak-scaling regime). Thus, our recursive distributed contraction algorithm is also communication-optimal for a large problem size. To get optimality for irregularly shaped tensors, we also make the dynamic choice of which pair of tensors needs to be communicated (multicasts must be done on A and B, and reductions on C).

However, we are also interested in strong scaling, in order to compute small tensor contractions rapidly. Such efficiency is necessary for higher order CC methods which require many different contractions. To do strong scaling, one computes a problem of the same size on more processors and aims to reduce the time to solution. In this scaling regime, more memory must be available than the amount necessary to store the tensor operands and output. Therefore, we can replicate tensor data and avoid communication. We always replicate the smallest one of the three tensors involved in the contraction to minimize the amount of memory and communication overhead of replication.

By taking consideration of the size of all of the tensors and doing replication up to the given memory constraints, we obtain an algorithm that partitions the tensor data in a communication optimal fashion. In particular, if we must compute F multiplies to do a contraction, we exploit as much memory as possible and select an algorithm that minimizes the amount

of data communicated for each tensor. Since we exploit the maximum replication and employ an optimal algorithm for contraction along each pair of indices the communication bandwidth cost comes out to be

$$W = O\left(\frac{F}{p \cdot \sqrt{M}} - M\right)$$

where M is the memory. This $-M$ term is achieved by avoiding the migration of input tensors wherever possible. This communication cost matches the bandwidth lower bound for matrix multiplication (Equation 1). If the contraction involves r symmetric permutations due to broken symmetries,

$$W = O\left(r \cdot \left(\frac{F/r}{p \cdot \sqrt{M}} - M\right)\right) = O\left(\frac{F}{p \cdot \sqrt{M}} - rM\right).$$

Evidently, performing symmetric premutations is useful only if the data-input size approaches the total data-movement cost for the contraction, which is uncommon. Note that if adaptive replication was not done, performing unpacking could be faster than doing permutations, since the unpacked contraction would effectively utilize more available memory.

C. On-node contraction of tensors

To perform the on-node contraction, we perform non-symmetric transposes of the tensors. In particular, we move all dimensions which do not correspond to groups of symmetric indices whose symmetry is broken within the contraction. If symmetries are not broken, we can simply fold the symmetric indices into one bigger dimension linearizing the packed layout. We perform an ordering transposition on the local tensor data to move linearized dimensions forward and the broken symmetric dimensions in the back of the tensors. To do a sequential contraction, we can then iterate over the broken symmetric indices (or unpack the symmetry) and call matrix multiplication over the linearized indices. For instance, the contraction from the start of this section,

$$c_{[ij]kl} = \sum_{pqr} a_{[ij][pq]} \cdot b_{[pqk][rl]}$$

would be done as a single matrix multiplication for each block, if all the broken symmetries are unpacked. However, if all the broken symmetries are kept folded, the nonsymmetric transpose would push forward the folded index corresponding to $|i < j|$, so that the sequential kernel could iterate over $pqkrl$ and call a scale operation for each $|i < j|$.

VI. Automatic mapping of contractions

Each contraction can place unique restrictions on the mapping of the tensors. In particular, our decomposition needs all symmetric tensor dimensions to be mapped with the same cyclic phase. Further, we must satisfy special considerations for each contraction, that can be defined in terms of indices (we will call them paired tensor dimensions) which are shared by a pair of tensors in the contraction. These considerations are

1) dimensions which are paired must be mapped with the same phase
2) for the paired tensor dimensions which are mapped to different dimensions of the processor grid (are mismatched)
   a) the mappings of two pairs of mismatched dimensions cannot share dimensions of the processor grid
   b) the subspace formed by the mappings of the mismatched paired dimensions must span all input data

We want to satisfy these constraints for a general case of any torus network of any dimension and shape, and be able to select an optimal mapping. The mapping framework of CTF achieves this.

A. Topology-aware network mapping

Any torus topology can be folded into a number of tori of smaller dimension. Depending on the dimensions of the tensors and the torus network, the tensor should be mapped to some folding of the network. Given a folding of the network, the optimal mapping should minimize the surface area of the sub-tensors. This mapping should have the longest indices of the largest tensor mapped to the longest processor grid dimensions. This implies a greedy index assignment algorithm can efficiently find the best mapping for a given folded network. Cyclops Tensor Framework defines all foldings for a given network and selects mappings onto them dynamically.

Once a tensor is defined or a tensor contraction is invoked, CTF searches through all topologies and selects the best mapping which satisfies the constraints. The search through mappings is done entirely in parallel among processors, then the best mapping is selected across all processors. The mapping logic is done without reading or moving any of the tensor data and is generally composed of integer logic that executes in a trivial amount of time with respect to the contraction. We construct a 'ghost' mapping for each valid topology and each ordering of tensors. The distributed contraction algorithm is constructed on each ghost mapping, and its communication and memory overheads are evaluated. If the ghost mapping is suboptimal it is thrown out without ever dictating data movement. Once a mapping is decided upon, the tensors are redistributed. Mappings in which tensors are replicated are also considered, with corresponding replication and reduction kernels generalizing 2.5D algorithms.

The best mapping can be selected according to a performance model. The amount of virtualization (to be described in the next section), communication, memory usage, and necessary redistributions is explicitly calculated for each mapping and the optimal one is selected. Generally, we attempt to not exceed the memory, not perform more than some factor of virtualization when possible, and minimize communication given those constraints. However, other performance models may be used and the best one may depend on the architecture and the scientific problem of interest.

B. Virtualization

Cyclops Tensor Framework performs virtualization to create a level of indirection between the task decomposition and the physical network topology. We provide a virtualization scheme that is guaranteed to generate a load balanced decomposition

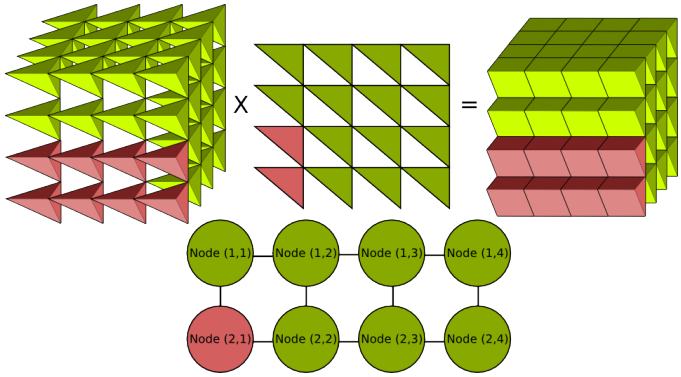


Fig. 2. Virtualization as used in CTF to perform contractions. This diagram demonstrates a mapping for a contraction of the form $c_{[kl]i} = \sum_j a_{[jkl]} \cdot b_{[ij]}$. In this case, we have a 4-by-2 processor grid, and a 4-by-4-by-4 virtual grid.

for any given tensor contraction (tensors of any symmetry, any dimension, and any index map defining the contraction). Further, we parameterize the virtual decomposition so that it is effectively a multiple of the processor grid, which insures that each processor owns the same number of sub-blocks. This scheme reduces the problem of mapping tensors with symmetry to mapping padded tensors with no symmetry. For example, in Figure 2, the 3D virtualized mapping is decomposed among the processors so that each processor is contracting a matrix of symmetric tensors with a vector of symmetric tensors into a matrix of symmetric tensors. The mapping is defined so that by the time the distributed contraction algorithm is executed, it need not be aware of the symmetry of the sub-tensors but only of their size.

We do not use a dynamically scheduled virtualization approach such as the overdecomposition embodied by the Charm++ runtime system [31]. Instead, we define the virtualization so that its dimensions are a multiple of the physical torus dimensions and generate a regular mapping. This approach maintains perfect load-balance and achieves high communication and task granularity by managing each virtualized sub-grid explicitly within each processor.

C. Redistribution of data

To satisfy each new set of restrictions for a contraction the mapping must change and tensor data must be reshuffled among processors according to the new mapping. Since the redistribution can potentially happen between every contraction, an efficient implementation is necessary. However, the data must first be given to CTF by the user application. We detail a scheme for input and output of data by key-value pairs, as well as a much more efficient algorithm for mapping-to-mapping tensor redistribution. Since Coupled Cluster and most other scientific applications are iterative and perform sequences of operations (contractions) on the same data, we assume input and output of data will happen less frequently than contractions.

To support general and simple data entry, CTF allows the user to write tensor data bulk-synchronously into the tensor object using key-value pairs. This allows the user to write data from any distribution that was previously defined, and to do so with any desired granularity (all datbe a at once or by chunks). Redistribution happens by calculating the cyclic phase of each key to determine which processor it belongs on. Once counts are assembled the data is redistributed via all-to-all communication. After this single redistribution phase, each processor should receive all data belonging to its sub-tensors, and can simply bin by virtual block then sort it locally to get it into the right order. This key-value binning scheme is essentially as expensive as a parallel sorting algorithm.

When transitioning between distributions, which we expect to happen much more frequently than between the application and user, we can take advantage of existing knowledge about the distribution. Between each distribution the cyclic phase along each dimension of the tensor can potentially change. This implies that each element might migrate from any given processor to another. Further, depending on the cyclic phase, the amount of padding could change along each dimension of the tensor. Additionally, due to the blocking schemes employed within each processor (to be described in the next section), the local ordering of the elements on each processor can change.

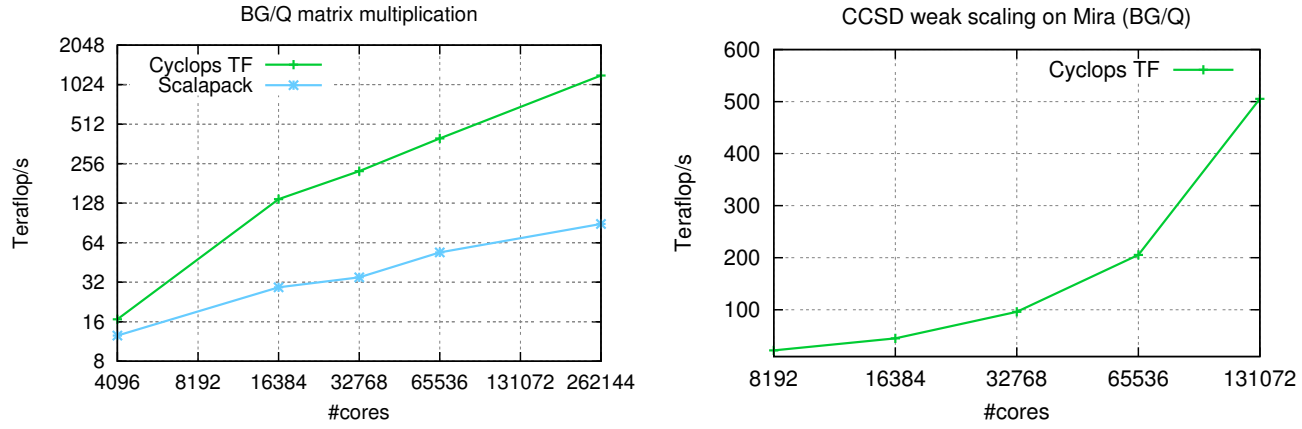
Our solution is to communicate the data while preserving the global ordering of elements in the communicated buffer. Each processor iterates over its local data in the order of the global index of the data, and computes the destination processor. The reverse process is performed in order for each processor to determine what data is received from which processor.

Redistributions require communication, however, they are a lower-order communication term with respect to a tensor contraction. Each piece of data must be migrated only once, and a trade-off between latency and bandwidth is made by the selection of an all-to-all algorithm.

VII. Parallel performance

A. Implementation details

The implementation uses no external libraries except for MPI [32], BLAS, and OpenMP. All code is tightly integrated and written in C/C++. Computationally expensive routines are



(a) Matrix Multiplication

(b) CCSD

Fig. 3. Figure 3(a) displays the strong scaling of matrix multiplication of 32K-by-32K square matrices. Figure 3(b) shows weak scaling of CCSD on Blue Gene/Q. The number of occupied orbitals ranged from 100 to 250 and the number of virtual orbitals ranged from 400 to 1000.

threaded and/or parallelized with MPI. Performance profiling is done by hand and with TAU [33].

B. Architectures

Cyclops Tensor Framework targets massively parallel architectures and is designed to take advantage of network topologies and communication infrastructure that scale to millions of nodes. Parallel scalability on commodity clusters should benefit significantly from the load balanced characteristics of the workload, while high-end supercomputers will additionally benefit from reduced inter-processor communication which typically becomes a bottleneck only at very high degrees of parallelism. We collected performance results on two state-of-the-art supercomputer architectures, IBM Blue Gene/Q and Cray XE6. We also tested sequential and multi-threaded performance on a Xeon desktop.

The sequential and non-parallel multi-threaded performance of CTF is compared to NWChem and MRCC. The platform is a commodity dual-socket quad-core Xeon E5620 system. On this machine, we used the sequential and threaded routines of the Intel Math Kernel Library. This platform, as well as the problem sizes tested reflect a typical situation for workloads on a workstation or small cluster, which is where the sequential performance of these codes is most important. Three problem sizes are timed, spanning a variety of ratios of the number of virtual orbitals to occupied orbitals.

The second experimental platform is 'Hopper', which is a Cray XE6 supercomputer, built from dual-socket 12-core "Magny-Cours" Opteron compute nodes. We used the Cray LibSci BLAS routines. This machine is located at the NERSC supercomputing facility. Each node can be viewed as a four-chip compute configuration due to NUMA domains. Each of these four chips have six super-scalar, out-of-order cores running at 2.1 GHz with private 64 KB L1 and 512 KB L2 caches. Nodes are connected through Cray's 'Gemini' network, which has a 3D torus topology. Each Gemini chip, which is shared by two Hopper nodes, is capable of 9.8 GB/s bandwidth. However, the NERSC Cray scheduler does not allocate contiguous partitions, so topology-aware mapping onto a torus cannot currently be performed.

The final platform we consider is the IBM Blue Gene/Q (BG/Q) architecture. We use the installations at Argonne and Lawrence Livermore National Laboratories. On both installations, IBM ESSL was used for BLAS routines. BG/Q has a number of novel features, including a 5D torus interconnect and 16-core SMP processor with 4-way hardware multi-threading, transactional memory and L2-mediated atomic operations [34], all of which serve to enable high performance of the widely portable MPI/OpenMP programming model. The BG/Q cores run at 1.6 GHz and the QPX vector unit supports 4-way fused multiply-add for a single-node theoretical peak of 204.8 GF/s. The BG/Q torus interconnect provides 2 GB/s of theoretical peak bandwidth per link in each direction, with simultaneous communication along all 10 links achieving 35.4 GB/s for 1 MB messages [35].

C. Results

We present the performance of a CCSD implementation on top of Cyclops Tensor Framework. The CCSD contraction code was extended from CCD in a few hours of work and is very compact. For each contraction, written in one line of code, CTF finds a topology-aware mapping of the tensors to the computer network and performs the necessary set of contractions on the packed structured tensors.

1) Sequential performance: The results of the sequential and multi-threaded comparison are summarized in Table I. The time per CCSD iteration is lowest for NWChem in all cases, and similarly highest for MRCC. The excessive iteration times for MRCC when the $\frac{n_v}{n_o}$ ratio becomes small reflect the fact that MRCC is largely memory-bound, as contractions are performed only with matrix-vector products. The multi-threaded speedup of CTF is significantly better than NWChem, most likely due to the lack of multi-threading of tensor transposition and other non-contraction operations in NWChem.

TABLE I
SEQUENTIAL AND NON-PARALLEL MULTI-THREADED PERFORMANCE COMPARISON OF CTF, NWCHEM, AND MRCC. ENTRIES ARE AVERAGE TIME FOR ONE CCSD ITERATION, FOR THE GIVEN NUMBER OF VIRTUAL (n_v) AND OCCUPIED (n_o) ORBITALS.

		$n_v = 110$ $n_o = 5$	$n_v = 94$ $n_o = 11$	$n_v = 71$ $n_o = 23$
NWChem	1 thread	6.80 sec	16.8 sec	49.1 sec
CTF	1 thread	23.6 sec	32.5 sec	59.8 sec
MRCC	1 thread	31.0 sec	66.2 sec	224. sec
NWChem	8 threads	5.21 sec	8.60 sec	18.1 sec
CTF	8 threads	9.12 sec	9.37 sec	18.5 sec
MRCC	8 threads	67.3 sec	64.3 sec	86.6 sec

TABLE II
CCSD ITERATION TIME ON 64 NODES OF HOPPER FOR n_v VIRTUAL ORBITALS AND n_o OCCUPIED ORIBTALS:

system	n_o	n_v	CTF	NWChem
w5	25	180	14 sec	36 sec
w7	35	252	90 sec	178 sec
w9	45	324	127 sec	-
w12	60	432	336 sec	-

2) Performance scalability: On the Cray XE6 machine, we compared the performance of our CCSD implementation with that of NWChem. We benchmarked the two codes for a series of water systems. In Table II, we detail the best time to solution achieved for each water problem by NWChem and CTF on 64 nodes of Hopper. The execution of NWChem was terminated if it did not complete a CCSD iteration by half an hour of execution, which is denoted by a dash in the table. NWChem completed CCSD for the w9 water system on 128 nodes, at the rate of 223 sec/iteration. On 128 nodes, CTF performed this task in 73 sec/iteration, 3-times faster than NWChem.

As a simple benchmark of the mapping framework, we compare the performance of matrix multiplication (which is a tensor contraction), done by CTF with the distributed matrix multiplication performance of ScaLAPACK [36] on Blue Gene/Q. CTF performs topology-aware mapping on the architecture and employs optimized collective communication. As a result (Figure 3(a)) CTF achieves significantly better strong scalability and achieves one petaflop/s on 16,384 nodes (262K cores).

The parallel weak scaling efficiency of our CCSD implementation on Blue Gene/Q is displayed in Figure 3(b). This weak scaling data was collected by doing the largest CCSD run that would fit in memory on each node count and normalizing the efficiency by the operation count. Going from 512 to 8192 nodes (130K cores), the efficiency actually increases, since larger CCSD problems can be done, which increases the ratio of computation over communication. We maintain high efficiency (30% of theoretical peak) to 8,192 nodes of BG/Q. The application was run with 4 MPI processes per node and 16 threads per process. Results at higher scales are expected to improve by reducing the number of MPI ranks and running with one process per node. However, this will require running with 32-64 threads, a challenge for index transposition and redistribution kernels. Investigation of better blocking and threading schemes for transposition kernels will be necessary.

TABLE III
A PERFORMANCE BREAKDOWN OF IMPORTANT KERNELS FOR A CCSD ITERATION DONE BY CTF ON A SYSTEM WITH $n_o = 200$ OCCUPIED ORIBTALS AND $n_v = 800$ VIRTUAL ORBITALS ON 4096 NODES (65K CORES) OF MIRA.

kernel	% of time	complexity	architectural bounds
matrix mult.	45%	$O(n_v^4 n_o^2/p)$	flops/mem bandwidth
broadcasts	20%	$O(n_v^4 n_o^2/p\sqrt{M})$	multicast bandwidth
prefix sum	10%	$O(p)$	allreduce bandwidth
data packing	7%	$O(n_v^2 n_o^2/p)$	integer ops
all-to-all-v	7%	$O(n_v^2 n_o^2/p)$	bisection bandwidth
tensor folding	4%	$O(n_v^2 n_o^2/p)$	memory bandwidth

Table III lists profiling data for a run of CTF on 4096 nodes (65K cores) of BG/Q. Nearly half the execution is spent in matrix multiplication, showing the relatively high efficiency of this calculation (24% of theoretical floating point peak). The prefix sum, data packing, and all-to-all-v operations are all part of tensor redistribution, which has a large effect on performance. The table lists the architectural bounds for each kernel, demonstrating that the application is stressing many components of the hardware with significant computations.

VIII. FUTURE WORK

Different types of sparsity in tensors will also be considered in Cyclops Tensor Framework. Tensors with banded sparsity structure can be decomposed cyclically so as to preserve band structure in the same way CTF preserves symmetry. Completely unstructured tensors can also be decomposed cyclically, though the decomposition would need to perform load balancing in the mapping and execution logic.

Cyclops Tensor Framework will also be integrated with a higher-level tensor manipulation framework as well as CC code generation methods. We have shown a working implementation of CCSD on top of CTF, but aim to implement much more complex methods. In particular, we are targeting the CCSDTQ method, which employs tensors of dimension up to 8 and gets the highest accuracy of any desirable CC method (excitations past quadruples have a negligible contribution).

ACKNOWLEDGMENTS

ES and DM were supported by a Department of Energy Computational Science Graduate Fellowship, grant number DE-FG02-97ER25308. We acknowledge funding from Microsoft (Award #024263) and Intel (Award #024894), and matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from ParLab affiliates National Instruments, Nokia, NVIDIA, Oracle and Samsung, as well as MathWorks. Research is also supported by DOE grants DE-SC0004938, DE-SC0005136, DE-SC0003959, DE-SC0008700, and AC02-05CH11231, and DARPA grant HR0011-12-2-0016. This research used resources of the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This research used resources of the National En-

ergy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] J. Čížek, "On the correlation problem in atomic and molecular systems. calculation of wavefunction components in Ursell-Type expansion using Quantum-Field theoretical methods," *The Journal of Chemical Physics*, vol. 45, no. 11, pp. 4256–4266, Dec. 1966.

[2] R. J. Bartlett and M. Musiał, "Coupled-cluster theory in quantum chemistry," *Reviews of Modern Physics*, vol. 79, no. 1, pp. 291–352, 2007.

[3] E. J. Bylaska et. al., "NWChem, a computational chemistry package for parallel computers, version 6.1.1," 2012.

[4] T. D. Crawford and H. F. Schaefer III, "An introduction to coupled cluster theory for computational chemists," *Reviews in Computational Chemistry*, vol. 14, p. 33, 2000.

[5] G. D. Purvis and R. J. Bartlett, "A full coupledcluster singles and doubles model: The inclusion of disconnected triples," *The Journal of Chemical Physics*, vol. 76, no. 4, pp. 1910–1918, Feb. 1982.

[6] Y. S. Lee, S. A. Kucharski, and R. J. Bartlett, "A coupled cluster approach with triple excitations," *Journal of Chemical Physics*, vol. 81, no. 12, p. 5906, 1984.

[7] J. Noga and R. J. Bartlett, "The full CCSDT model for molecular electronic structure," *Journal of Chemical Physics*, vol. 86, no. 12, p. 7041, 1987.

[8] S. A. Kucharski and R. J. Bartlett, "Recursive intermediate factorization and complete computational linearization of the coupled-cluster single, double, triple, and quadruple excitation equations," *Theoretica Chimica Acta*, vol. 80, no. 4-5, pp. 387–405, 1991.

[9] E. Solomonik and J. Demmel, "Communication-optimal 2.5D matrix multiplication and LU factorization algorithms," in *Lecture Notes in Computer Science, Euro-Par, Bordeaux, France*, Aug 2011.

[10] G. Baumgartner, A. Auer, D. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Hirata, S. Krishnamoorthy, S. Krishnan, C. Lam, Q. Lu, M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov, "Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 276 –292, feb. 2005.

[11] E. A. Auer, G. Baumgartner, D. E. Bernholdt, A. Bibireata, D. Cociorva, X. Gao, R. Harrison, S. Krishnamoorthy, H. Krishnan, C. chung Lam, Q. Lu, M. Nooijen, R. Pitzer, J. Ramanujam, and E. Sibiryakov, "Automatic code generation for many-body electronic structure methods: The tensor contraction engine," 2006.

[12] X. Gao, S. Krishnamoorthy, S. Sahoo, C.-C. Lam, G. Baumgartner, J. Ramanujam, and P. Sadayappan, "Efficient search-space pruning for integrated fusion and tiling transformations," in *Languages and Compilers for Parallel Computing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, vol. 4339, pp. 215–229.

[13] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global arrays: A nonuniform memory access programming model for high-performance computers," *The Journal of Supercomputing*, vol. 10, pp. 169–189, 1996, 10.1007/BF00130708.

[14] V. Lotrich, N. Flocke, M. Ponton, B. A. Sanders, E. Deumens, R. J. Bartlett, and A. Perera, "An infrastructure for scalable and portable parallel programs for computational chemistry," in *Proceedings of the 23rd international conference on Supercomputing*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 523–524.

[15] E. Deumens, V. F. Lotrich, A. Perera, M. J. Ponton, B. A. Sanders, and R. J. Bartlett, "Software design of aces iii with the super instruction architecture," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 1, no. 6, pp. 895–901, 2011.

[16] M. Kállay and P. R. Surján, "Higher excitations in coupled-cluster theory," *The Journal of Chemical Physics*, vol. 115, no. 7, p. 2945, 2001.

[17] H. Jia-Wei and H. T. Kung, "I/O complexity: The red-blue pebble game," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, ser. STOC '81. New York, NY, USA: ACM, 1981, pp. 326–333.

[18] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in linear algebra," *SIAM J. Mat. Anal. Appl.*, vol. 32, no. 3, 2011.

[19] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1017 – 1026, 2004.

[20] L. E. Cannon, "A cellular computer to implement the Kalman filter algorithm," Ph.D. dissertation, Bozeman, MT, USA, 1969.

[21] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar, "A three-dimensional approach to parallel matrix multiplication," *IBM J. Res. Dev.*, vol. 39, pp. 575–582, September 1995.

[22] R. A. Van De Geijn and J. Watts, "SUMMA: scalable universal matrix multiplication algorithm," *Concurrency: Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.

[23] A. Aggarwal, A. K. Chandra, and M. Snir, "Communication complexity of PRAMs," *Theoretical Computer Science*, vol. 71, no. 1, pp. 3 – 28, 1990.

[24] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds," in *Proceedinbgs of the 24th ACM symposium on Parallelism in algorithms and architectures*, ser. SPAA '12. New York, NY, USA: ACM, 2012, pp. 77–79. [Online]. Available: http://doi.acm.org/10.1145/2312005.2312021

[25] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," *SIAM Journal on Computing*, vol. 10, no. 4, pp. 657–675, 1981.

[26] S. L. Johnsson, "Minimizing the communication time for matrix multiplication on multiprocessors," *Parallel Comput.*, vol. 19, pp. 1235–1257, November 1993.

[27] E. Solomonik, A. Bhatele, and J. Demmel, "Improving communication performance in dense linear algebra via topology aware collectives," in *Supercomputing, Seattle, WA, USA*, Nov 2011.

[28] G. E. Scuseria and H. F. Schaefer III, "Is coupled cluster singles and doubles (CCSD) more computationally intensive than quadratic configuration interaction (QCISD)?" vol. 90, no. 7, pp. 3700–3703, 1989.

[29] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK user's guide*, J. J. Dongarra, Ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997.

[30] J. Poulson, B. Maker, J. R. Hammond, N. A. Romero, and R. van de Geijn, "Elemental: A new framework for distributed memory dense matrix computations," *ACM Transactions on Mathematical Software*, in press.

[31] L. V. Kale and S. Krishnan, "CHARM++: a portable concurrent object oriented system based on C++," in *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, ser. OOPSLA '93. New York, NY, USA: ACM, 1993, pp. 91–108.

[32] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. Cambridge, MA, USA: MIT Press, 1994.

[33] S. S. Shende and A. D. Malony, "The TAU parallel performance system," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, Summer 2006.

[34] R. Haring, M. Ohmacht, T. Fox, M. Gschwind, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, A. Gara, G.-T. Chiu, P. Boyle, N. Chist, and C. Kim, "The IBM Blue Gene/Q compute chip," *Micro, IEEE*, vol. 32, no. 2, pp. 48 –60, march-april 2012.

[35] D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker, "The IBM Blue Gene/Q interconnection network and message unit," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 26:1–26:10.

[36] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK user's guide*, J. J. Dongarra, Ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997.