Automated Operation Minimization of Tensor Contraction Expressions in Electronic Structure Calculations

Albert Hartono,[1] Alexander Sibiryakov,[1] Marcel Nooijen,[3] Gerald Baumgartner,[4]
David E. Bernholdt,[6] So Hirata,[7] Chi-Chung Lam,[1] Russell M. Pitzer,[2]
J. Ramanujam,[5] and P. Sadayappan[1]

[1] Dept. of Computer Science and Engineering
[2] Dept. of Chemistry
The Ohio State University, Columbus, OH, 43210 USA
[3] Dept. of Chemistry, University of Waterloo, Waterloo, Ontario N2L BG1, Canada
[4] Dept. of Computer Science
[5] Dept. of Electrical and Computer Engineering
Louisiana State University, Baton Rouge, LA 70803 USA
[6] Computer Sci. & Math. Div., Oak Ridge National Laboratory, Oak Ridge, TN 37831 USA
[7] Quantum Theory Project, University of Florida, Gainesville, FL 32611 USA

Abstract. Complex tensor contraction expressions arise in accurate electronic structure models in quantum chemistry, such as the Coupled Cluster method. Transformations using algebraic properties of commutativity and associativity can be used to significantly decrease the number of arithmetic operations required for evaluation of these expressions, but the optimization problem is NP-hard. Operation minimization is an important optimization step for the Tensor Contraction Engine, a tool being developed for the automatic transformation of high-level tensor contraction expressions into efficient programs. In this paper, we develop an effective heuristic approach to the operation minimization problem, and demonstrate its effectiveness on tensor contraction expressions for coupled cluster equations.

1 Introduction

Currently, manual development of accurate quantum chemistry models is very tedious and takes an expert several months to years to develop and debug. The Tensor Contraction Engine (TCE) [2, 1] is a tool that is being developed to reduce the development time to hours/days, by having the chemist specify the computation in a high-level form, from which an efficient parallel program is automatically synthesized. This should enable the rapid synthesis of high-performance implementations of sophisticated ab-initio quantum chemistry models, including models that are too tedious for manual development by quantum chemists. An important first step in the synthesis process of the TCE is that of algebraic manipulation of the input tensor contraction expressions, to find an equivalent form with minimized operation count.

We illustrate the operation minimization problem using simple examples. Consider the following tensor contraction expression involving three tensors t, f and s, with a and c representing virtual orbital indices with range V, and i and j representing occupied orbital indices with range O. Computed as a single nested loop computation, the number of multiply-accumulate operations needed would be O^2V^2.

(1) $r_i^a {+}{=} \sum_{c,k} t_i^c f_c^k s_k^a,$ (cost O^2V^2)

However, by performing a two-step computation with an intermediate I, it is possible to compute the result using $2OV^2$ operations:

(2) $I_c^a = \sum_k f_c^k s_k^a,$ (cost OV^2)

(3) $r_i^a {+}{=} \sum_c t_i^c I_c^a,$ (cost OV^2)

Another possibility using O^2V computations, which is more efficient when $V > O$ (as is usually the case), is shown below:

(4) $I_i^k = \sum_c t_i^c f_c^k,$ (cost O^2V)

(5) $r_i^a {+}{=} \sum_k I_i^k s_k^a,$ (cost O^2V)

The above example illustrates the problem of single-term optimization, also called strength reduction: find the best sequence of two-tensor contractions to achieve a multi-tensor contraction. Different orders of contraction can result in very different operation costs; for the above example, if the ratio of V/O were 10, there is an order of magnitude difference in the number of arithmetic operations for the two choices.

For more complex expressions with several tensors to be contracted, the number of possible ways of forming intermediates is exponential in the number of tensors. The single-term optimization problem is a generalization of the well known matrix-chain multiplication problem, but while the latter has a simple polynomial time dynamic programming solution, the former problem has been shown to be NP-complete [7].

Let us next consider an expression with two terms:

(6) $r_{ij}^{ab} {+}{=} \sum_{c,d} t_i^c s_j^d v_{cd}^{ab} + \sum_{c,d} u_{ij}^{cd} v_{cd}^{ab},$ (cost $2O^2V^4$)

If each term were individually optimized via strength reduction, we would have:

(7) $I_{id}^{ab} = \sum_c t_i^c v_{cd}^{ab},$ (cost OV^4)

(8) $r_{ij}^{ab} {+}{=} \sum_d s_j^d I_{id}^{ab} + \sum_{c,d} u_{ij}^{cd} v_{cd}^{ab},$ (cost $O^2V^3 + O^2V^4$)

However, a better approach to reducing the overall operation cost would be as follows:

(9) $I_{ij}^{cd} = t_i^c s_j^d + u_{ij}^{cd},$ (cost O^2V^2)

(10) $r_{ij}^{ab} {+}{=} \sum_{c,d} I_{ij}^{cd} v_{cd}^{ab},$ (cost O^2V^4)

Thus it can be seen that single-term optimization (strength reduction) is not an optimal strategy. We have to look at the expression in the global context to determine the optimal evaluation. Evaluation of the binary terms in an expression constitute an intrinsic cost to evaluating the tensor product. Little can be done (except for instances of factorization of the form $AC + BC \to (A+B)C$) to reduce the cost of the binary terms. However, the cost of evaluating terms that are ternary or higher can be greatly reduced by combining them with the binary terms that have to be evaluated anyway. In the present example, the additional cost of evaluating the ternary term is reduced from $OV^4 + O^2V^3$ to O^2V^2. The expensive O^2V^4 multiplication that would be counted in single term optimization disappears as the multiplication has to be done anyway in the binary term.

The goal of operation minimization is to find an optimal or near-optimal factorization of the input tensor contraction expression to evaluate the ternary+ terms, given the presence of the binary terms that carry an intrinsic basic cost. In this paper, we develop algorithms for operation minimization. The solution presented in this paper is an extension of the techniques first reported in [10]. The conceptually simplest approach is to use an exhaustive search algorithm that is guaranteed to determine the optimal factorization. However, its runtime grows exponentially with the number of terms in the tensor contraction expression, making it infeasible for use on the more complex coupled cluster equations. We then develop heuristic search strategies for the operation minimization problem. The best algorithm is found to be a random-descent heuristic, which is then used to explore the generated solutions for a range of values of V/O. The results validate the effectiveness of the use of an automated approach to generating operation-minimal factorizations for large tensor contraction expressions.

2 Related Work

Compilers use common subexpression elimination to reduce the number of arithmetic operations [4]. However they do not consider algebraic properties such as associativity and distributivity. Computer algebra systems typically contain factorization algorithms, e.g., for finding roots of polynomials [3]. Similarly, an algorithm based on factor graphs can be used to factor functions of many variables into products of local functions [5]. However, the emphasis of these approaches is mainly on symbolic manipulation instead of on minimizing operation counts based on index range information. Winograd [12] addressed the general problem of evaluating multiple expressions that share common variables using the minimum number of arithmetic operations. Miller [8] suggested several analytical and numerical techniques for reducing the operation count in computational electromagnetic applications.

The work presented in this paper builds upon methods developed in a recent thesis by Sibiryakov [10]. The problem of strength reduction for arbitrary tensor contraction expressions was addressed in [7, 6]. We are not aware of other work that has addressed in a general manner the operation minimization problem that we consider in this paper. In developing efficient implementations of electronic structure methods such as the coupled cluster methods

[9, 11], quantum chemists have used domain heuristics for strength reduction and factorization for specific kinds of tensor contraction expressions, but have not developed approaches to solve the operation minimization problem for arbitrary tensor contraction expressions.

3 Operation Minimization Algorithms

In this section, we outline several algorithms for the operation minimization problem. We first start with the explanation on the single term optimization then continue on different approaches of multi term optimization.

3.1 Single Term Optimization

Considering first a single "product" term in a tensor contraction expression, as seen from expression (1), if the term has more than two tensors, there are many alternatives in evaluating just that product term. Although the single-term optimization problem is NP-complete, a dynamic programming algorithm is actually very effective in practice in the context of the tensor contraction expressions encountered in high-accuracy electronic structure methods.

Fig. 1 shows the dynamic programming algorithm for the single term optimization. The algorithm works bottom-up to find the optimal sequence of contractions for every subset of the given tensor set. This information is stored in a storage table of size $2^n - 1$, the number of all possible subsets of a given set of n tensors. Starting with the smallest possible subsets first, optimal solutions are incrementally generated for successively larger subsets. At step i, all possible $\binom{n}{i}$ subsets of length i are considered. For each such subset, all $2^{n-1} - 1$ ways to divide it into two parts are considered, and the cost of computing it with such a split is evaluated as the sum of the costs of the two parts (already in the table) and the cost of contraction of the two parts. Provided with all these operations, we can see that the complexity of the single term optimization of a product term $T_1 T_2 T_3 \cdots T_n$ is given by the following polynomial expression:

$$\sum_{i=1}^{n} \left(\binom{n}{i} \times \left(2^{i-1} - 1\right) \right) = \frac{1}{2} \times \left(3^n - 2^{n+1} + 1\right) = \Theta(3^n)$$

3.2 Exhaustive Search

We first describe the multi term optimization with an exhaustive search algorithm that systematically evaluates all possible factorizations of the input tensor contraction expression to determine the form with lowest operation count. This search is implemented recursively using memoization, which is equivalent to a dynamic programming approach implemented in a top-down manner. The algorithm of exhaustive search is shown on Fig. 2.

```
SINGLE-TERM-OPT($T_1 T_2 \ldots T_n$)
1.  $table \leftarrow$ allocate a new table
2.  for $i \leftarrow 1$ to $n$ do
3.      $S \leftarrow$ set of all possible subsets of $\{T_1, T_2, \ldots, T_n\}$ with length of $i$
4.      for each set $\{T_{x_1}, T_{x_2}, \ldots, T_{x_i}\} \in S$
5.          SINGLE-TERM-SEARCH($\{T_{x_1}, T_{x_2}, \ldots, T_{x_i}\}, table$)
6.  $value \leftarrow$ getValue($table, T_1 T_2 \ldots T_n$)
7.  return $value.formula$

SINGLE-TERM-SEARCH($\{T_{x_1}, T_{x_2}, \ldots, T_{x_n}\}, table$)
1.  if $n = 1$
2.      $value.formula \leftarrow T_{x_1}$
3.      $value.cost \leftarrow 0$
4.  else if $n = 2$
5.      $value.formula \leftarrow (T_{x_1} \times T_{x_2})$
6.      $value.cost \leftarrow$ CONTRACTION-COST$(\{T_{x_1}, T_{x_2}\})$
7.  else
8.      $minCost \leftarrow \infty$
9.      $D \leftarrow$ set of all possible ways to divide $\{T_{x_1}, T_{x_2}, \ldots, T_{x_n}\}$ into two parts
10.     for each $< T_{i_1} T_{i_2} \ldots T_{i_a}, T_{j_1} T_{j_2} \ldots T_{j_b} > \in D$
11.         $left \leftarrow$ getValue($table, T_{i_1} T_{i_2} \ldots T_{i_a}$)
12.         $right \leftarrow$ getValue($table, T_{j_1} T_{j_2} \ldots T_{j_b}$)
13.         $T_{left} \leftarrow$ the intermediate tensor of $left.formula$
14.         $T_{right} \leftarrow$ the intermediate tensor of $right.formula$
15.         $curCost \leftarrow$ CONTRACTION-COST$(\{T_{left}, T_{right}\}) + left.cost + right.cost$
16.         if $curCost < minCost$
17.             $value.formula \leftarrow (left.formula \times right.formula)$
18.             $value.cost \leftarrow curCost$
19.             $minCost \leftarrow curCost$
20. $key \leftarrow T_{x_1} T_{x_2} \ldots T_{x_n}$
21. insertIntoTable($table, key, value$)
```

Fig. 1. Single Term Optimization

Considering a particular tensor as a factor, exhaustive search enumerates all possible factorizations, which grows exponentially. If a factor appears in n terms of a tensor contraction expression, the number of possible factorizations with respect to that factor is $2^n - n$. For instance, all possible factorizations of an expression $AB + AC + AD$ are $AB + AC + AD$, $A(B + C) + AD$, $A(B + D) + AC$, $A(C + D) + AB$, and $A(B + C + D)$.

As in standard dynamic programming, a storage table is maintained with solutions for subexpressions; hence, we need not re-evaluate subexpressions that have been previously considered. The procedure FIND-OPTIMAL-SUB-EXP shown on Fig. 3 describes the process of the storage table lookup. Matching two equivalent expressions requires generating canonical forms of both expressions. If a canonical form of a subexpression is found as a key in the storage table, the corresponding entry value, which is the optimal solution of the

```
START-EXHAUSTIVE($E$)
1.  $table \leftarrow$ allocate a new table
2.  return EXHAUSTIVE($E, table$)

EXHAUSTIVE($E, table$)
1.  $factors \leftarrow$ set of all factors in $E$
2.  $best.cost \leftarrow \infty$
3.  EXHAUSTIVE-SEARCH($E, factors, table, best$)
4.  return $best.formula$

EXHAUSTIVE-SEARCH($E, factors, table, best$)
1.  if $factors$ is empty
2.      apply each term in $E$ with SINGLE-TERM-OPT
3.      if OP-COUNT($E$) $<$ $best.cost$
4.          $best.cost \leftarrow$ OP-COUNT($E$)
5.          $best.formula \leftarrow E$
6.  else
7.      $f \leftarrow$ remove the first element of $factors$
8.      $S \leftarrow$ set of all terms in $E$ that contains $f$
9.      $C \leftarrow$ set of all possible subsets of $S$
10.     for each $c \in C$ where $c$ contains more than one term
11.         $E' \leftarrow$ replica of $E$
12.         $f' \leftarrow$ replica of $f$
13.         $factors' \leftarrow$ replica of $factors$
14.         remove all the terms, which are members of $c$, from $E'$
15.         remove $f'$ from every term in $c$
16.         $optimalSubexp \leftarrow$ FIND-OPTIMAL-SUB-EXP($c, table$, EXHAUSTIVE)
17.         $newTerm \leftarrow f' \times optimalSubexp$
18.         $E' \leftarrow E' + newTerm$
19.         $T \leftarrow$ the intermediate tensor of the subexpression $optimalSubexp$
20.         $factors' \leftarrow factors' \cup \{T\}$
21.         EXHAUSTIVE-SEARCH($E', factors', table, best$)
22.     EXHAUSTIVE-SEARCH($E, factors, table, best$)
```

Fig. 2. Exhaustive Search

```
FIND-OPTIMAL-SUB-EXP($F, table, searchProc$)
1.  $canonicForm \leftarrow$ the canonical form of $F$
2.  if $table$ contains a key which is equal to $canonicForm$
3.      $exp \leftarrow$ getValue($table, canonicForm$)
4.      $optimalSubexp \leftarrow$ replica of $exp$
5.      rename the indices of $optimalSubexp$ based on the indices of $F$
6.  else
7.      $optimalSubexp \leftarrow searchProc(F, table)$
8.      insertIntoTable($table, canonicForm, optimalSubexp$)
9.  return $optimalSubexp$
```

Fig. 3. Table Lookup

subexpression, is fetched and replicated. The indices of replica may differ from the original subexpression; thus, renaming indices of the replica is required.

The exhaustive search algorithm is guaranteed to find the operation-minimal factorization of the input expression, but since its time complexity grows exponentially with the number of terms, it may be impractical to use in optimizing expressions with a large number of terms. We therefore also implemented several heuristic search strategies for operation minimization.

3.3 Time-Limited Exhaustive Search with Tier-Based Partitioning

By imposing a time limit, we can avoid an indefinitely long search time that often occurs in exhaustive search. Each time exhaustive search exceeds the specified time limit, we suspend the search and store the result of the partially executed exhaustive search. Afterward the original expression is divided into two smaller groups each with half the original group's terms. These two subsets of terms can be individually factorized using the same time-limited exhaustive search and the partially factored terms can then be recombined. The cost of the combined expression is compared with the result of the timed exhaustive search that was previously interrupted. The result with the minimum operation count is returned. The splitting process is continued till each group of terms is successfully factorized within the time limit.

Prior to each splitting, it is essential to sort the expression terms in a decreasing order of term cost (as determined by single-term optimization), allowing higher-ordered terms to be placed and optimized in the same group after the splitting.

In the worst case, a splitting can occur whenever a time-limited exhaustive search is applied. We can view these recursive splittings as a binary tree in which each node represents one exhaustive search. The maximum number of nodes in such a binary tree will be $\sum_{i=0}^{log_2 N}(\frac{N}{2^i}) \approx 2N$, where N indicates the number of terms in the original expression. Therefore, the time complexity of time-limited exhaustive search is $O(TN)$, where T is the given time limit. For generating experimental results in this paper, the time limit used for exhaustive search was set to ten minutes. In addition, very similar results were also obtained with a shorter time limit of ten seconds, demonstrating the efficacy of the algorithm in finding a reasonable solution quickly.

We evaluated another approach to partitioning based on tier groups. Two terms are placed into the same tier group if they have the same number of tensors. The terms in the input expression are first partitioned into tier groups. Optimizing and combining tier groups is done incrementally. Suppose we have groups at tier 2, tier 3, and tier 4. We first optimize tier 2 with timed exhaustive search. The optimized tier 2 terms are then grouped together with the unoptimized tier 3 terms and then optimized with timed exhaustive search. Then this result is grouped with the unoptimized tier 4 terms and then optimized again with timed exhaustive search. The pseudocode of the algorithm is shown on Fig. 4.

```
START-TIMED-EXHAUSTIVE-TIER-BASED-PARTITION($E$)
1.  $table \leftarrow$ allocate a new table
2.  $S \leftarrow$ set of tier groups of $E$
3.  $firstTier \leftarrow$ remove the first element of $S$
4.  $optimal \leftarrow$ TIMED-EXHAUSTIVE-COST-BASED-PARTITION($firstTier, table$)
5.  for each $s \in S$
6.      $optimal \leftarrow optimal + s$
7.      $optimal \leftarrow$ TIMED-EXHAUSTIVE-COST-BASED-PARTITION($optimal, table$)
8.  return $optimal$

TIMED-EXHAUSTIVE-COST-BASED-PARTITION($E, table$)
1.  apply COST-BASED-SORT on $E$
2.  $timeLimit \leftarrow$ 10 minutes
3.  $optimal \leftarrow$ TIMED-EXHAUSTIVE($E, table, timeLimit$)
4.  if the timed exhaustive search executed on line 3 is completed within the given time limit
5.      return $optimal$
6.  else
7.      $left \leftarrow$ the first half of $E$'s terms
8.      $right \leftarrow$ the second half of $E$'s terms
9.      $optimalLeft \leftarrow$ TIMED-EXHAUSTIVE-COST-BASED-PARTITION($left, table$)
10.     $optimalRight \leftarrow$ TIMED-EXHAUSTIVE-COST-BASED-PARTITION($right, table$)
11.     $combined \leftarrow optimalLeft + optimalRight$
12.     if a partial result of $optimal$ exists and OP-COUNT($optimal$) $<$ OP-COUNT($combined$)
13.         return $optimal$
14.     else
15.         return $combined$
```

Fig. 4. Time-Limited Exhaustive Search with Tier-Based Partitioning

3.4 Direct Descent Search

Direct descent is a greedy algorithm that chooses the best local factorization at every step. All pairs of terms that can be combined using the distributive property of multiplication over addition are considered, and the transformation that provides the greatest reduction in operation count is chosen. For a particular factor, if the number of factorizable terms in the expression is n the total number of possible two-term factorizations is $n(n-1)/2$. At every step, all possible two-term factorizations are evaluated and the best one is chosen; this process is repeated until no more factorization is possible. Fig. 5 describes the algorithm of direct descent search. Based on the number of factorizations considered at each step, the runtime complexity of direct descent obviously grows polynomially with the degree of terms in the input expression.

3.5 Random Descent Search

Random descent search is a modified version of direct descent search that attempts to avoid some local minima by making random choices for two-term factorization at the initial steps.

```
START-DIRECT-DESCENT($E$)
1.  $table \leftarrow$ allocate a new table
2.  return DIRECT-DESCENT($E, table$)

DIRECT-DESCENT($E, table$)
1.  $factors \leftarrow$ set of all factors in $E$
2.  return DIRECT-DESCENT-SEARCH($E, factors, table$)

DIRECT-DESCENT-SEARCH($E, factors, table$)
1.  while $factors$ is not empty do
2.      $best.profit \leftarrow -\infty$
3.      for each $f \in factors$
4.          $S \leftarrow$ set of all terms in $E$ that contains $f$
5.          if $|S| = 0$ or $|S| = 1$
6.              $factors \leftarrow factors - \{f\}$
7.          else
8.              $C \leftarrow$ set of all possible subsets of $S$ that each subset contains exactly two terms
9.              $bestPair \leftarrow$ pair of terms in $C$ that has the greatest reduction cost
10.             if PROFIT-COUNT($f, bestPair$) $\leq 0$
11.                 $factors \leftarrow factors - \{f\}$
12.             else if PROFIT-COUNT($f, bestPair$) $> best.profit$
13.                 $best.profit \leftarrow$ PROFIT-COUNT($f, bestPair$)
14.                 $best.pair \leftarrow bestPair$
15.                 $best.factor \leftarrow f$
16.     if $best.profit \neq -\infty$
17.         remove all the terms, which are members of $best.pair$, from $E$
18.         remove $best.factor$ from every term in $best.pair$
19.         $optimalSubexp \leftarrow$ FIND-OPTIMAL-SUB-EXP($best.pair, table$, DIRECT-DESCENT)
20.         $newTerm \leftarrow best.factor \times optimalSubexp$
21.         $E \leftarrow E + newTerm$
22.         $T \leftarrow$ the intermediate tensor of the subexpression $optimalSubexp$
23.         $factors \leftarrow factors \cup \{T\}$
24. return $E$
```

Fig. 5. Direct Descent Search

These random moves are then followed by direct descent moves. Through experimentation, it was found best to make the number of random moves to be one fourth of the total number of terms in the input expressions. Using too many random factorizations at the initial steps was found to give poorer results; too few random factorizations at the initial steps did not contribute a significant improvement. In order to further minimize the operation count, we first execute a direct descent search and store its factorization result; after that, one hundred attempts of random descent are repeated. The best result from these one hundred tries is compared with the result of direct descent that was initially executed. The result with the minimum operation count is returned. The pseudocode of random descent search can be seen on Fig. 6.

```
START-RANDOM-DESCENT($E$)
1.  $table \leftarrow$ allocate a new table
2.  $E' \leftarrow$ replica of $E$
3.  $dirExp \leftarrow$ DIRECT-DESCENT($E', table$)
4.  $best.cost \leftarrow$ OP-COUNT($dirExp$)
5.  $best.formula \leftarrow dirExp$
6.  $attempts \leftarrow 100$
7.  for $i \leftarrow 1$ to $attempts$ do
8.      $E' \leftarrow$ replica of $E$
9.      $randExp \leftarrow$ RANDOM-DESCENT($E', table$)
10.     if OP-COUNT($randExp$) $< best.cost$
11.         $best.cost \leftarrow$ OP-COUNT($randExp$)
12.         $best.formula \leftarrow randExp$
13. return $best.formula$

RANDOM-DESCENT($E, table$)
1.  $factors \leftarrow$ set of all factors in $E$
2.  return RANDOM-DESCENT-SEARCH($E, factors, table$)

RANDOM-DESCENT-SEARCH($E, factors, table$)
1.  $randomSteps \leftarrow \frac{1}{4} \times$ total number of terms in $E$
2.  while $factors$ is not empty and $randomSteps > 0$ do
3.      $f \leftarrow$ an element chosen randomly from $factors$
4.      $S \leftarrow$ set of all terms in $E$ that contains $f$
5.      if $|S| = 0$ or $|S| = 1$
6.          $factors \leftarrow factors - \{f\}$
7.      else
8.          $C \leftarrow$ set of all possible subsets of $S$ that each subset contains exactly two terms
9.          if none of term pairs in $C$ has a positive reduction cost
10.             $factors \leftarrow factors - \{f\}$
11.         else
12.             $randomPair \leftarrow$ a randomly-chosen pair of terms in $C$ that has a positive reduction cost
13.             remove all the terms, which are members of $randomPair$, from $E$
14.             remove $f$ from every term in $randomPair$
15.             $optimalSubexp \leftarrow$ FIND-OPTIMAL-SUB-EXP($randomPair, table$, DIRECT-DESCENT)
16.             $newTerm \leftarrow f \times optimalSubexp$
17.             $E \leftarrow E + newTerm$
18.             $T \leftarrow$ the intermediate tensor of the subexpression $optimalSubexp$
19.             $factors \leftarrow factors \cup \{T\}$
20.             $randomSteps \leftarrow randomSteps - 1$
21. return DIRECT-DESCENT-SEARCH($E, factors, table$)
```

Fig. 6. Random Descent Search

4 Experimental Results

We have implemented the algorithms for searching a formula with minimum number of operations as described previously. They were tested on complex tensor contraction expressions that appear in the “coupled cluster” family of quantum chemical methods. We used the coupled cluster equations including just single and double excitations (CCSD) and also single, double, and triple excitations (CCSDT) as representatives of the many different computational chemistry methods based on tensor contraction expressions. These methods involve coupled equations which determine the single excitation amplitudes (referred to here the “T1” equation), double excitation amplitudes (T2), and in the case of CCSDT, triple excitations (T3).

Table 1 shows the number of terms in each of the equations, along with the number of arithmetic operations of evaluating the equation. The number of arithmetic operations depends upon O and V, which vary depending on the molecule and desired quality of the simulation, but a typical range is $1 \leq V/O \leq 100$. To provide concrete comparisons, we set O to 10 and V to 100, giving the numerical values in Table 1. This is representative of calculations of modest size that could be done on a workstation and will be used throughout this paper unless otherwise noted.

In order to focus on the effects of the optimization algorithms, we eliminate the binary terms from the input equations and consider only the ternary and higher terms, as described in the introduction. The optimal cost of the binary terms of each equation can be seen in the last column of Table 1.

Table 2 illustrates the results obtained by optimizing the five equations with the algorithms described previously. Exhaustive search results are shown only for the T1 equations because

Table 1. Characteristics of the fully unfactorized input equations used in this experiment

Equation	Number of terms	Operation count (no optimization)	Operation count (with single term optimization only)	Operation count of optimal binary terms
CCSD T1	14	$1.78 \cdot 10^{10}$	$3.11 \cdot 10^{8}$	$2.24 \cdot 10^{8}$
CCSD T2	31	$4.90 \cdot 10^{13}$	$3.58 \cdot 10^{10}$	$2.27 \cdot 10^{10}$
CCSDT T1	15	$2.08 \cdot 10^{10}$	$2.31 \cdot 10^{9}$	$2.22 \cdot 10^{9}$
CCSDT T2	37	$6.13 \cdot 10^{13}$	$3.00 \cdot 10^{11}$	$2.45 \cdot 10^{11}$
CCSDT T3	47	$9.34 \cdot 10^{16}$	$3.26 \cdot 10^{13}$	$2.26 \cdot 10^{13}$

Table 2. The operation counts of expressions optimized by different algorithms

Equation	Ternary+ operation count				
	Single Term	Exhaustive	Timed Exhaustive with Tier-Based Partitioning	Direct Descent	Random Descent
CCSD T1	$8.65 \cdot 10^{7}$	$4.73 \cdot 10^{7}$	$4.73 \cdot 10^{7}$	$4.73 \cdot 10^{7}$	$4.73 \cdot 10^{7}$
CCSD T2	$1.31 \cdot 10^{10}$	–	$5.18 \cdot 10^{9}$	$5.33 \cdot 10^{9}$	$5.14 \cdot 10^{9}$
CCSDT T1	$8.65 \cdot 10^{7}$	$4.73 \cdot 10^{7}$	$4.73 \cdot 10^{7}$	$4.73 \cdot 10^{7}$	$4.73 \cdot 10^{7}$
CCSDT T2	$5.52 \cdot 10^{10}$	–	$5.57 \cdot 10^{9}$	$5.57 \cdot 10^{9}$	$5.37 \cdot 10^{9}$
CCSDT T3	$9.94 \cdot 10^{12}$	–	$9.80 \cdot 10^{11}$	$9.17 \cdot 10^{11}$	$9.17 \cdot 10^{11}$

Table 3. Ternary+ operation count for CCSD T2

Optimized	Actual V/O					
for V/O	1	2	3	5	10	100
2	$7.41 \cdot 10^6$	$4.52 \cdot 10^7$	$1.46 \cdot 10^8$	$7.20 \cdot 10^8$	$7.47 \cdot 10^9$	$4.41 \cdot 10^{13}$
5	$1.05 \cdot 10^7$	$5.88 \cdot 10^7$	$1.73 \cdot 10^8$	$7.08 \cdot 10^8$	$5.14 \cdot 10^9$	$4.67 \cdot 10^{12}$
10	$1.07 \cdot 10^7$	$5.95 \cdot 10^7$	$1.74 \cdot 10^8$	$7.13 \cdot 10^8$	$5.15 \cdot 10^9$	$4.67 \cdot 10^{12}$

Opt. V/O	Leading terms of cost function in symbolic form
2	$2O^3V^3 + 2O^4V^2 + \underline{4OV^4} + 10O^2V^3 + 10O^3V^2 + 6O^4V + \ldots$
5	$4O^3V^3 + 4O^4V^2 + 6O^2V^3 + 8O^3V^2 + 6O^4V + \ldots$
10	$4O^3V^3 + 4O^4V^2 + 6O^2V^3 + 8O^3V^2 + 6O^4V + \ldots$

Table 4. Ternary+ operation count for CCSDT T2

Optimized	Actual V/O					
for V/O	1	2	3	5	10	100
2	$7.85 \cdot 10^6$	$4.77 \cdot 10^7$	$1.54 \cdot 10^8$	$7.52 \cdot 10^8$	$7.70 \cdot 10^9$	$4.43 \cdot 10^{13}$
5	$9.88 \cdot 10^6$	$5.89 \cdot 10^7$	$1.78 \cdot 10^8$	$7.55 \cdot 10^8$	$5.63 \cdot 10^9$	$5.28 \cdot 10^{12}$
10	$1.09 \cdot 10^7$	$6.13 \cdot 10^7$	$1.80 \cdot 10^8$	$7.40 \cdot 10^8$	$5.37 \cdot 10^9$	$4.88 \cdot 10^{12}$

Opt. V/O	Leading terms of cost function in symbolic form
2	$2O^3V^3 + 2O^4V^2 + \underline{4OV^4} + 12O^2V^3 + 12O^3V^2 + 6O^4V + \ldots$
5	$4O^3V^3 + 2O^4V^2 + 12O^2V^3 + 16O^3V^2 + 6O^4V + \ldots$
10	$4O^3V^3 + 4O^4V^2 + 8O^2V^3 + 10O^3V^2 + 6O^4V + \ldots$

Table 5. Percentage of ternary+ operation count

V/O	1	2	3	5	10	100
%ternary+ (CCSD)	51.72	42.50	36.98	30.47	18.47	2.26
%ternary+ (CCSDT)	18.94	13.57	10.45	7.08	3.87	0.33

they are the only ones small enough for this approach to be feasible. All heuristic algorithms find the optimal factorization in small cases (i.e., the T1 equations), and in the other cases produce very similar results, which have less than one percent differences. The direct descent results illustrate its tendency to get stuck in local minima and not find an optimal factorization. Random descent sometimes offers improvement and consistently performs the best of all of the algorithms described.

To examine the behavior of the random descent search in more detail, we examined the effect of varying the V/O ratio. Changing this ratio will change the actual costs of each term and may even change the optimal factorization. Tables 3 and 4 illustrates these effects. Operation counts are shown for V/O ratios ranging from 1 to 10, for factorizations of the

CCSD T2 and CCSDT T2 equations that have been optimized explicitly for V/O ratios of 2, 5, and 10. This is representative of how the results of this optimization are likely to be used in practice: code will be automatically generated for selected values of V/O spanning the range of interest, and at runtime, the best available version will be selected based on the actual V and O for the molecule under study. The results clearly illustrate the value of tailoring the factorization to the V/O of interest. For example, using the factorization that is optimal for $V/O = 2$ for a calculation in which $V/O = 100$ yields an operation count that is an order of magnitude larger than in the more optimal case using the algorithm optimized for $V/O = 10$. Conversely, using an algorithm optimized for the large $V/O = 10$ ratio is clearly not optimal if V/O is small, e.g. 1. In Table 3 the symbolic operation count is given for the CCSD T2 equation for the various factorization solutions. For the larger V/O ratios of 5 and 10, no terms are used that scale as V^4 or OV^4, as are present (and underlined) in the factorization scheme optimized for $V/O = 2$. Instead the algorithm prefers to use more terms that scale as O^3V^3 (four instead of two), and there is an interesting tradeoff between terms that formally scale as N^5 vs. N^6, where N indicates O or V.

Similar conclusions can be drawn from Table 4. Comparing the optimized algorithms for $V/O = 2$ and $V/O = 10$ for various values of actual V/O ratios we clearly see that for small V/O values, the $V/O = 10$ solution is about 30% more costly than the $V/O = 2$ solution, while at the other end of the spectrum the $V/O = 10$ solution is about an order of magnitude more efficient. Similar tradeoffs as in case of the CCSD T2 equations are at work to determine the best overall factorization scheme, depending on the actual V and O values.

The present computer optimized factorization can be contrasted with current (hand-written) implementations of coupled cluster methods. In traditional implementations, factorization is considered only at a symbolic level, trying to reduce the V exponent first, then the O exponent, then the factor in front of the cost term; typically, little attention is paid to terms beyond the highest order (in the sum of the O and V exponents). This approach doesn't fully consider the ratio of V/O, and the possibility that terms considered lower order might result in an operation count comparable to higher order terms with a different balance of O and V exponents. The equation parts of Tables 3 and 4 illustrates this idea, with the symbolic costs of the different factorizations found for different V/O ratios. Comparing, for example, the costs of the CCSD T2 equation factored for $V/O = 2$ and $V/O = 5$, we observe that N^6 terms have *higher* coefficients in $V/O = 5$ than in $V/O = 2$, while the OV^4 term has been entirely eliminated from $V/O = 5$. The larger V space means that it is more cost-effective to evaluate more N^6 terms with a lower V exponent than the N^5 term OV^4. A similar cross-over occurs in the CCSDT T2 equations between $V/O = 2$ and $V/O = 5$. Moreover, changes in term coefficients take place in the CCSDT T2 equations where $V/O = 5$ and $V/O = 10$. In the $V/O = 10$

equation the coefficient of the O^4V^2 term is higher than that in the $V/O = 5$ equation, while it is vice versa for the O^2V^3 term. This is an important result because it runs counter to the intuition (and accepted practice) of most quantum chemists. Let us note that it is not only the ratio V/O which determines the optimal factorization, but also their individual sizes as there is a trade-off between overall N^5 and N^6 terms in achieving optimal performance.

Interestingly, the random descent algorithm not necessarily finds the optimal factorization. This can be seen from Table 4 where the algorithm optimized for $V/O = 10$ performs better for the actual $V/O = 5$ ratio than the algorithm that was explicitly optimized for this case. This shows that there is still some room for improvement.

To put these results, obtained for the ternary and higher terms only, in proper perspective, Table 5 shows the percentage of the total computational cost due to the ternary and higher terms as a function of the V/O ratio. It is seen that the ternary+ cost is a sizable fraction of the overall calculation for lower V/O ratios, but it can rapidly become a rather small fraction of the overall cost if V/O is large, in particular for CCSDT. In such cases single term optimization might suffice. However it must be noted that these percentages do not tell the entire story. On modern hierarchical memory systems, the binary terms can typically be implemented with significantly greater efficiency than the ternary+ terms, so that a simple operation count underestimates their importance to the overall computation. Further, the present optimization scheme for factorization is quite generally applicable, and the efficiency gains can be expected to be very relevant for computational schemes that are not dominated by the binary terms. The factorization of more complicated, yet efficient theoretical models in quantum chemistry will be explored in our future work.

5 Conclusions

In this paper we presented heuristic and exhaustive search algorithms for operation minimization of complex tensor expressions occurring for example in quantum chemistry. It has been demonstrated that optimal factorization depends on the precise sizes of the index ranges of the tensors involved, and therefore very different computer implementations will be optimal for different size problems. We found that the random descent algorithm works best, although this search algorithm can be expensive for complicated cases. The time-limited exhaustive search with tier-based partitioning algorithm is often a cost-effective alternative; in addition, we believe this can provide a suitable starting point that can be subject to further optimizations.

Acknowledgments This work has been supported in part by the U.S. National Science Foundation, the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), and by a Discovery grant from the Natural Sciences and Engineering

Research Council of Canada. ORNL is managed by UT-Battelle, LLC for the US Dept. of Energy under contract DE-AC-05-00OR22725.

References

1. 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*, 93(2):276–292, February 2005.
2. G. Baumgartner, D.E. Bernholdt, D. Cociorva, R. Harrison, S. Hirata, C. Lam, M. Nooijen, R. Pitzer, J. Ramanujam, and P. Sadayappan. A high-level approach to synthesis of high-performance codes for quantum chemistry. In *Proc. of Supercomputing 2002*, November 2002.
3. B. Buchberger, G. Collins, and R. Loos, editors. *Computer Algebra: Symbolic and Algebraic Computation*. Springer-Verlag, New York, 1983.
4. C.N. Fischer and R.J. LeBlanc Jr. *Crafting a Compiler*. Benjamin/Cummings, 1991.
5. F. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, February 2001.
6. C. Lam. *Performance Optimization of a Class of Loops Implementing Multi-Dimensional Integrals*. PhD thesis, The Ohio State University, Columbus, OH, August 1999.
7. C. Lam, P. Sadayappan, and R. Wenger. On optimizing a class of multi-dimensional loops with reductions for parallel execution. *Parallel Processing Letters*, 7(2):157–168, 1997.
8. E.K. Miller. Solving bigger problems by decreasing the operation count and increasing computation bandwidth. *Proceedings of the IEEE*, 79(10):1493–1504, October 1991.
9. G.E. Scuseria, C.L. Janssen, and H.F. Schaefer III. An efficient reformulation of the closed-shell coupled cluster single and double excitation (CCSD) equations. *The Journal of Chemical Physics*, 89(12):7382–7387, 1988.
10. A. Sibiryakov. Operation Optimization of Tensor Contraction Expression. Master's thesis, The Ohio State University, Columbus, OH, August 2004.
11. J.F. Stanton, J. Gauss, J.D. Watts, and R.J. Bartlett. A direct product decomposition approach for symmetry exploitation in many-body methods. I. Energy calculations. *The Journal of Chemical Physics*, 94(6):4334–4345, 1991.
12. S. Winograd. *Arithmetic Complexity of Computations*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1980.