Performance Optimization of Tensor Contraction Expressions for Many-Body Methods in Quantum Chemistry[†]

Albert Hartono,[‡] Qingda Lu,[‡] Thomas Henretty,[‡] Sriram Krishnamoorthy,[‡] Huaijian Zhang,[‡] Gerald Baumgartner,[§] David E. Bernholdt,[⊥] Marcel Nooijen,[¶] Russell Pitzer,[‡] J. Ramanujam,[§] and P. Sadayappan*[,‡]

The Ohio State University, Columbus, Ohio, Louisiana State University, Baton Rouge, Louisiana, Oak Ridge National Laboratory, Oak Ridge, Tennessee, University of Waterloo, Waterloo, Ontario, Canada

Received: June 1, 2009; Revised Manuscript Received: September 3, 2009

Complex tensor contraction expressions arise in accurate electronic structure models in quantum chemistry, such as the coupled cluster method. This paper addresses two complementary aspects of performance optimization of such tensor contraction expressions. 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. The identification of common subexpressions among a set of tensor contraction expressions can result in a reduction of the total number of operations required to evaluate the tensor contractions. The first part of the paper describes an effective algorithm for operation minimization with common subexpression identification and demonstrates its effectiveness on tensor contraction expressions for coupled cluster equations. The second part of the paper highlights the importance of data layout transformation in the optimization of tensor contraction computations on modern processors. A number of considerations, such as minimization of cache misses and utilization of multimedia vector instructions, are discussed. A library for efficient index permutation of multidimensional tensors is described, and experimental performance data is provided that demonstrates its effectiveness.

Introduction

Users of current and emerging high-performance parallel computers face major challenges to both performance and productivity in the development of their scientific applications. The manual development of accurate quantum chemistry models typically can take an expert months to years of tedious effort to develop and debug a high-performance implementation. One approach to alleviate the burden on application developers is the use of automatic code generation techniques to synthesize efficient parallel programs from high-level specification of computations expressed in a domain-specific language. The Tensor Contraction Engine (TCE)[1−3] effort resulted from a collaboration between computer scientists and quantum chemists to develop a framework for automated optimization of tensor contraction expressions, which form the basis of many-body and coupled cluster methods.[4−7] In this paper, we describe two complementary optimization approaches that were developed for the TCE but that are available as independent software components for use by developers of other computational chemistry suites.

The first step in the TCE's code synthesis process is the transformation of input tensor contraction expressions into an equivalent form with minimal operation count. Input equations representing a collection of tensor contraction expressions typically involve the summation of tens to hundreds of terms, each involving the contraction of two or more tensors. Given a single-term expression with several tensors to be contracted, instead of a single nested loop structure to compute the result, it is often much more efficient to use a sequence of pairwise contractions of tensors, with explicit creation of temporary intermediate tensors. This optimization problem can be viewed as a generalization of the matrix chain multiplication problem. However, although the matrix-chain optimization problem has a polynomial time solution, the multitensor contraction problem has been shown to be *NP*-hard:[8] a combinatorial number of possibilities for pairwise two-tensor contractions must be considered. With tensor contraction expressions involving the summation of tens to hundreds of terms, there are opportunities for further reduction in computational cost by recognizing common subexpressions in the sequence of pairwise two-tensor contractions for computing the multitensor contraction terms. Quantum chemists have addressed the operation optimization problem for specific models,[7,9] but to the best of our knowledge, a general approach to optimization of arbitrary tensor contraction expressions was not addressed prior to the TCE effort. In the first part of this paper, we discuss a generalized treatment of the operation minimization problem for tensor contraction expressions.

The second part of the paper addresses an important issue pertaining to achieving a high fraction of processor peak performance when computing operation-minimized tensor contraction expressions. Achieving high performance on current and emerging processors requires the generation of highly optimized code that exploits the vector instruction set of the machine (e.g., SSE, AVX, etc.), minimizes data movement costs between memory and cache, and minimizes the number of register loads/stores in loops. The current state-of-the-art in compiler technology is unable to achieve anywhere close to machine peak in compiling loop-level code representing a multidimensional tensor contraction. Hence, the approach taken

[†] Part of the "Russell M. Pitzer Festschrift".

* Corresponding author. E-mail: saday@cse.ohio-state.edu.

[‡] The Ohio State University.

[§] Louisiana State University.

[⊥] Oak Ridge National Laboratory.

[¶] University of Waterloo.

10.1021/jp9051215 CCC: $40.75 © 2009 American Chemical Society
Published on Web 09/28/2009

in quantum chemistry codes is to morph a tensor contraction problem into a matrix multiplication problem and then use highly tuned matrix multiplication libraries available for nearly all systems. In general, this requires a layout transformation of the tensors into a form in which all contracted indices of the tensors are grouped together in the transformed view. Theoretically, the computational complexity of the data layout transformation step is linear in the number of elements in the tensor, whereas the computational complexity of the subsequent matrix multiplication has a higher computational complexity. However, in practice, the use of a straightforward loop code to perform the layout transformation results in significant overhead. In the second part of this paper, we discuss the development of an efficient tensor layout transformation library.

The rest of the paper is organized as follows: The next section elaborates on the operation minimization problem, followed by a section that describes the algorithmic approach to operation minimization. Experimental results that demonstrate its effectiveness are presented in the section after that. The following section describes the layout transformation problem, summarizing an approach (described in detail elsewhere[10]) to efficient transposition of 2D arrays, and the generalization of the 2D transposition routines to multidimensional tensor layout transformation along with experimental results from incorporation of the layout transformation routines into NWChem. We then discuss related work in the section following that, leading to the conclusion section.

Operation Minimization of Tensor Contraction Expressions

A tensor contraction expression comprises a sum of a number of terms, where each term represents the contraction of two or more tensors. We first illustrate the issue of operation minimization for a single term before addressing the issue of optimizing across multiple terms. Consider the following tensor contraction expression involving three tensors—t, f and s—with indices x and z that have range V, and indices i and k that have range O. Distinct ranges for different indices is a characteristic of the quantum chemical methods of interest, where O and V correspond to the number of occupied and virtual orbitals in the representation of the molecule (typically, $V \gg O$). Computed as a single nested loop computation, the number of arithmetic operations needed would be $2O^2V^2$.

$$r_i^x = \sum_{z,k} t_i^z f_z^k s_k^x \text{ (cost} = 2O^2V^2)$$

However, by performing a two-step computation with an intermediate, I, it is possible to compute the result using $4OV^2$ operations:

$$I_z^x = \sum_k f_z^k s_k^x \text{ (cost} = 2OV^2);$$
$$r_i^x = \sum_z t_i^z I_z^x \text{ (cost} = 2OV^2)$$

Another possibility using $4O^2V$ computations, which is more efficient when $V > O$ (as is usually the case in quantum chemistry calculations), is shown below:

$$I_i^k = \sum_z t_i^z f_z^k \text{ (cost} = 2O^2V);$$
$$r_i^x = \sum_k I_i^k s_k^x \text{ (cost} = 2O^2V)$$

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

With complex tensor contraction expressions involving a large number of terms, if multiple occurrences of the same subexpression can be identified, it need be computed only once, stored in an intermediate tensor, and used multiple times. Thus, common subexpressions can be stored as intermediate results that are used more than once in the overall computation. Manual formulations of computational chemistry models often involve the use of such intermediates. The class of quantum chemical methods of interest, which include the coupled cluster singles and doubles (CCSD) method,[7,9] are most commonly formulated using the molecular orbital basis (MO) integral tensors. However, the MO integrals are intermediates, derived from the more fundamental atomic orbital basis (AO) integral tensors. Alternate "AO-based" formulations of CCSD have been developed in which the more fundamental AO integrals are used directly, without fully forming the MO integrals.[11] However, it is very difficult to manually explore all possible formulations of this type to find the one with minimal operation count, especially since it can depend strongly on the characteristics of the particular molecule being studied.

The challenge in identifying cost-effective common subexpressions (also referred to as common subexpression elimination, or CSE) is the combinatorial explosion of the search space, since single-term optimization of different product terms must be treated in a coupled manner. The following simple example illustrates the problem.

Suppose we have two MO-basis tensors, v and w, which can be expressed as a transformation of the AO-basis tensor, a, in two steps. Using single-term optimization to form tensor v, we consider two possible sequences of binary contractions, as shown below, both of which have the same (minimal) operation cost. Extending the notation above, indices p and q represent AO indices, which have range $M = O + V$.

Sequence 1:

$$f_q^i = \sum_p a_q^p c_p^i \text{ (cost} = 2OM^2);$$
$$v_j^i = \sum_p f_p^i d_j^p \text{ (cost} = 2O^2M)$$

Sequence 2:

$$g_i^p = \sum_q a_q^p d_i^q \text{ (cost} = 2OM^2);$$
$$v_j^i = \sum_p g_j^p c_p^i \text{ (cost} = 2O^2M)$$

To generate tensor w, suppose that there is only one cost-optimal sequence:

$$f_q^i = \sum_p a_q^p c_p^i \text{ (cost} = 2OM^2\text{);}$$

$$w_x^i = \sum_p f_p^i e_x^p \text{ (cost} = 2OVM\text{)}$$

Note that the first step in the formation of w uses the same intermediate tensor f that appears in sequence 1 for v. Considering just the formation of v, either of the two sequences is equivalent in cost. But one form uses a common subexpression that is useful in computing the second MO-basis tensor, whereas the other form does not. If sequence 1 is chosen for v, the total cost of computing both v and w is $2OM^2 + 2O^2M + 2OVM$. On the other hand, the total cost is higher if sequence 2 is chosen ($4OM^2 + 2O^2M + 2OVM$). The $2OM^2$ cost difference is significant when M is large.

When a large number of terms exist in a tensor contraction expression, there is a combinatorial explosion in the search space if all possible equivalent-cost forms for each product term must be compared with each other.

In the first part of the paper, we address the following question: by developing an automatic operation minimization procedure that is effective in identifying suitable common subexpressions in tensor contraction expressions, can we automatically find more efficient computational forms? For example, with the coupled cluster equations, can we automatically find AO-based forms by simply executing the operation minimization procedure on the standard MO-based CCSD equations, where occurrences of the MO integral terms are explicitly expanded out in terms of AO integrals and integral transformations?

Operation Minimization with Common Subexpression Elimination

In this section, we describe the algorithm used to perform operation minimization, which uses single-term optimization together with common subexpression elimination (CSE). The exponentially large space of possible single-term optimizations, together with CSE, makes an exhaustive search approach prohibitively expensive. So we use a two-step approach to apply single-term optimization and CSE in tandem.

The algorithm is shown in Figure 2. It uses the single-term optimization algorithm, which is broadly illustrated in Figure 1 and described in greater detail in our earlier work.[12] It takes as input a sequence of tensor contraction statements. Each statement defines a tensor in terms of a sum of tensor contraction expressions. The output is an optimized sequence of tensor contraction statements involving only binary tensor contractions. All intermediate tensors are explicitly defined.

The key idea is to determine the "binarization" (determination of optimal sequence of two-tensor contractions) of more expensive terms before the less expensive terms. The most expensive terms contribute heavily to the overall operation cost and potentially contain expensive subexpressions. Early identification of these expensive subexpressions can facilitate their reuse in the computation of other expressions, reducing the overall operation count.

The algorithm begins with the *term set* to be optimized as the set of all the terms of the tensor contraction expressions on the right-hand side of each statement. The set of intermediates is initially empty. In each step of the iterative procedure, the binarization for one term is determined. Single-term optimization is applied to each term in the term set using the current set of intermediates, and the most expensive term is chosen to be "binarized" first. Among the set of optimal binarizations for the chosen term, the one that maximally reduces the cost of the remaining terms is chosen. Once the term and its binarizations are decided upon, the set of intermediates is updated, and the corresponding statements for the new intermediates are generated. The procedure continues until the term set is empty.

Evaluation of Operation Minimization

To illustrate the use of the automatic operation minimization algorithm, we consider the tensor expressions for a closed-shell CCSD T2 computation. Figure 3 shows the CCSD T2 equation, including the computation of the MO integrals (denoted v) and the expression for the double-excitation residual. We compare the optimized forms generated in two different ways: (1) with the conventional "separated" approach of first explicitly forming the MO integrals from AO integrals and then using the MO integrals for the CCSD T2 term and (2) using an "integrated" form in which significant MO integrals in the CCSD T2 equation are replaced by the expressions that produce them. Although some MO integrals may appear more than once in the T2 expression, the multiple expansion of such terms does not result in any unnecessary duplication of computation because of common subexpression elimination with the operation minimization algorithm.

We study two scenarios for evaluation of the CCSD T2 expression: (1) the typical mode, in which iterations of the residual calculation are performed with the t-amplitudes changing every iteration, but without change to the MO integrals (because the transformation matrices to convert AO integrals to MO integrals do not change), and (2) an orbital optimization (Brueckner basis) scenario in which the AO-to-MO transformation matrices change from iteration to iteration; that is, the MO integrals (if explicitly formed) must be recalculated for every iteration.

Since the operation minimization algorithm uses specific values for the number of occupied orbitals O and the number of virtual orbitals V, the optimized expressions that are generated could be different for different O and V values. The values for O and V depend on the molecule and quality of the simulation, but a typical range is $1 \leq V/O \leq 100$. To provide concrete comparisons, O was set to 10 and V values of 100, 500, and 1000 were used. Additional runs for O set to 100 and V values of 1000, 5000, and 10 000 were also evaluated, but the overall trends were similar, so that data is not presented here.

The standard CCSD computation proceeds through a number of iterations in which the MO integrals remain unchanged. At convergence, the amplitudes attain values such that the residual is equal to zero, and this typically takes 10−50 iterations. In some variants of CCSD, such as Brueckner basis orbital optimization, the MO integrals also change at each iteration, requiring the AO-to-MO transformation to be repeated. The optimized tensor expressions for these two scenarios can be very different. With the operation minimization system, all input terms can be tagged as either stable (i.e., unchanging from iteration to iteration) or volatile (i.e., changing every iteration). In addition, an expected number of iterations can be provided to the optimizer. The operation minimization algorithm seeks to find a transformed form of the input tensor expression that minimizes the total arithmetic cost for the expected number of iterations.

Figure 4 shows the output generated by the integrated optimization of the AO-to-MO transform and the CCSD T2 expression (for an expected number of iterations, T, of 10). Seventeen new intermediates are generated, labeled using capital

```
SINGLE-TERM-OPT-CSE($E, is$)
if $E$ is a single-tensor expression
  then return $\{\langle E, \emptyset \rangle\}$
  else \* $E$ is a multiple-tensor contraction expression (i.e., $E_1 * \ldots * E_n$) * \
       $\{\langle p_1, is_1\rangle, \langle p_2, is_2\rangle, \ldots\} \leftarrow$
           set of pairs of optimal binarizations of $E$ and its corresponding intermediate set
           (the given intermediate set $is$ is used to find effective common subexpressions)
       return $\{\langle p_1, is_1\rangle, \langle p_2, is_2\rangle, \ldots\}$
```

Figure 1. Single-term optimization algorithm with common subexpression elimination.

```
OPTIMIZE($stmts$)
$MSET \leftarrow$ set of all terms obtained from RHS expressions of $stmts$
$is \leftarrow \emptyset$ \* the set of intermediates * \
while $MSET \neq \emptyset$
do $M_{heaviest} \leftarrow$ the heaviest term in $MSET$
        (searched by applying SINGLE-TERM-OPT-CSE($M_i, is$) on each term $M_i \in MSET$)
   $PSET \leftarrow$ SINGLE-TERM-OPT-CSE($M_{heaviest}, is$)
   $\langle p_{best}, is_{best}\rangle \leftarrow$ NIL
   $profit \leftarrow 0$
   for each $\langle p_i, is_i\rangle \in PSET$
   do $cur\_profit \leftarrow 0$
      for each $M_i \in (MSET - \{M_{heaviest}\})$
      do $base\_cost \leftarrow$ op-cost of optimal binarization in SINGLE-TERM-OPT-CSE($M_i, is$)
         $opt\_cost \leftarrow$ op-cost of optimal binarization in SINGLE-TERM-OPT-CSE($M_i, is \cup is_i$)
         $cur\_profit \leftarrow cur\_profit + (base\_cost - opt\_cost)$
      if $(\langle p_{best}, is_{best}\rangle = \text{NIL}) \vee (cur\_profit > profit)$
        then $\langle p_{best}, is_{best}\rangle \leftarrow \langle p_i, is_i\rangle$
             $profit \leftarrow cur\_profit$
   $stmts \leftarrow$ replace the term $M_{heaviest}$ in $stmts$ with $p_{best}$
   $MSET \leftarrow MSET - \{M_{heaviest}\}$
   $is \leftarrow is \cup is_{best}$
return $stmts$
```

Figure 2. Global operation minimization algorithm.

$$
\begin{array}{llll}
v^{ij}_{kl} = a^{pq}_{rs} c^i_p c^j_q c^r_k c^s_l & v^{ij}_{ka} = a^{pq}_{rs} c^i_p c^j_q c^r_k c^s_a & v^{ia}_{jk} = a^{pq}_{rs} c^i_p c^a_q c^r_j c^s_k & v^{ij}_{ab} = a^{pq}_{rs} c^i_p c^j_q c^r_a c^s_b \\
v^{ia}_{jb} = a^{pq}_{rs} c^i_p c^a_q c^r_j c^s_b & v^{ia}_{bj} = a^{pq}_{rs} c^i_p c^a_q c^r_b c^s_j & v^{ab}_{ij} = a^{pq}_{rs} c^a_p c^b_q c^r_i c^s_j & v^{ia}_{bc} = a^{pq}_{rs} c^i_p c^a_q c^r_b c^s_c \\
v^{ab}_{ic} = a^{pq}_{rs} c^a_p c^b_q c^r_i c^s_c & v^{ab}_{cd} = a^{pq}_{rs} c^a_p c^b_q c^r_c c^s_d & &
\end{array}
$$

$$
\begin{aligned}
r^{ab}_{ij} = {} & \tfrac{1}{2} v^{ji}_{ba} + f^c_a t^{bc}_{ji} - f_o o^i_k t^{ab}_{kj} + v^{jc}_{ba} t^c_i - v^{ij}_{kb} t^a_k + \tfrac{1}{2} v^{cd}_{ab} t^{dc}_{ji} + 2v^{ic}_{ak} t^{bc}_{jk} - v^{ic}_{ak} t^{bc}_{kj} - v^{ic}_{ka} t^{bc}_{jk} - v^{jc}_{ka} t^{bc}_{ki} \\
& + \tfrac{1}{2} v^{ji}_{kl} t^{ba}_{kl} - f^c_k t^c_i t^{ab}_{kj} - f^c_k t^{bc}_{ji} t^a_k + \tfrac{1}{2} v^{cd}_{ba} t^c_j t^d_i - v^{jc}_{bk} t^c_i t^a_k - v^{jc}_{ka} t^c_i t^b_k + v^{jc}_{kl} t^c_i t^{ba}_{kl} - v^{dc}_{ka} t^{bc}_{jk} t^d_i - v^{dc}_{kb} t^{ac}_{kj} t^d_i \\
& + 2v^{cd}_{ka} t^{bc}_{jk} t^d_i - v^{cd}_{ka} t^{bc}_{kj} t^d_i + \tfrac{1}{2} v^{ji}_{kl} t^b_k t^a_l - v^{cd}_{kb} t^{dc}_{ji} t^a_k + v^{ic}_{kl} t^{bc}_{jk} t^a_l + v^{jc}_{kl} t^{bc}_{ki} t^a_l - 2v^{ic}_{kl} t^{bc}_{jl} t^a_k + v^{ic}_{kl} t^{bc}_{lj} t^a_k \\
& + 2v^{dc}_{ka} t^{bc}_{ji} t^d_k - 2v^{ic}_{kl} t^c_l t^{ab}_{kj} - v^{cd}_{ka} t^{bc}_{ji} t^d_k + v^{ic}_{kl} t^c_k t^{ab}_{lj} + v^{cd}_{kl} t^{bc}_{ji} t^{ad}_{lk} - 2v^{cd}_{kl} t^{bc}_{ji} t^{ad}_{kl} + v^{cd}_{kl} t^{dc}_{il} t^{ab}_{kj} - 2v^{cd}_{kl} t^{dc}_{ik} t^{ab}_{lj} \\
& + \tfrac{1}{2} v^{cd}_{kl} t^{dc}_{ji} t^{ab}_{kl} + 2v^{cd}_{kl} t^{bc}_{jk} t^{ad}_{il} - 2v^{cd}_{kl} t^{bc}_{kj} t^{ad}_{il} + \tfrac{1}{2} v^{cd}_{kl} t^{bc}_{kj} t^{ad}_{li} - v^{cd}_{kl} t^{bc}_{jl} t^{ad}_{ik} + v^{cd}_{kl} t^{bc}_{lj} t^{ad}_{ik} + \tfrac{1}{2} v^{cd}_{kl} t^{ac}_{lj} t^{bd}_{ki} \\
& - v^{dc}_{kb} t^c_j t^d_i t^a_k + \tfrac{1}{2} v^{cd}_{kl} t^c_j t^d_i t^{ba}_{kl} + v^{jc}_{kl} t^c_i t^b_k t^a_l - 2v^{cd}_{kl} t^{bc}_{jk} t^d_i t^a_l + v^{cd}_{kl} t^{bc}_{kj} t^d_i t^a_l + v^{cd}_{kl} t^{bc}_{jl} t^d_i t^a_k + v^{cd}_{kl} t^{ac}_{lj} t^d_i t^b_k \\
& + v^{cd}_{kl} t^c_i t^d_k t^{ab}_{lj} - 2v^{cd}_{kl} t^c_i t^d_l t^{ab}_{kj} + \tfrac{1}{2} v^{cd}_{kl} t^{dc}_{ji} t^a_k t^b_l + v^{cd}_{kl} t^{bc}_{ji} t^d_k t^a_l - 2v^{cd}_{kl} t^{bc}_{ji} t^d_l t^a_k + \tfrac{1}{2} v^{cd}_{kl} t^c_j t^d_i t^b_k t^a_l \\
\text{residual}^{ab}_{ij} = {} & r^{ab}_{ij} + r^{ba}_{ji}
\end{aligned}
$$

Figure 3. Unoptimized input expressions for CCSD T2 and AO-to-MO transform.

$$
\begin{array}{llll}
A^{pq}_{ri} = a^{pq}_{sr} c^s_i & B^{pq}_{ij} = c^s_i A^{pq}_{sj} & C^{ip}_{aj} = c^s_a (c^i_q A^{qp}_{sj}) & D^{ap}_{ij} = c^a_q B^{pq}_{ij} \\
E^{pi}_{qr} = a^{sp}_{qr} c^i_s & v^{ij}_{kl} = c^i_p (c^j_q B^{pq}_{lk}) & v^{ij}_{ka} = c^j_q C^{iq}_{ak} & v^{ia}_{jk} = c^i_p D^{ap}_{kj} \\
v^{ij}_{ab} = c^r_a (c^s_b (c^j_q E^{qi}_{rs})) & v^{ia}_{jb} = c^a_q C^{iq}_{bj} & v^{ia}_{bj} = c^a_q (c^r_b (c^s_j E^{qi}_{rs})) & v^{ab}_{ij} = c^a_p D^{bp}_{ji} \\
F^{ai}_{jk} = v^{ad}_{ij} t^d_k & G^p_i = c^c_p t^c_i & H^{ab}_{ij} = v^{ab}_{ij} - 2v^{ab}_{ji} & I^{ap}_{ij} = c^s_a (a^{rq}_{ps} (c^c_r (c^d_q t^{dc}_{ij}))) \\
J^{ij}_{kl} = v^{cd}_{ij} t^{dc}_{kl} & K^{ij}_{kl} = t^c_i F^{cj}_{kl} & L^{pq}_{ri} = a^{sp}_{qr} G^s_i & M^a_i = v^{ad}_{ki} t^d_k - 2v^{ad}_{ik} t^d_k \\
N^{ij}_{kl} = v^{ic}_{jk} t^c_l & O^{ab}_{ij} = c^a_p (c^s_b (A^{pq}_{si} G^q_j)) & P^{ab}_{ij} = c^a_q (c^s_b (c^r_i L^{qr}_{sj})) & Q^{ia}_{jk} = v^{ia}_{jk} - 2v^{ia}_{kj}
\end{array}
$$

$$
\begin{aligned}
r^{ab}_{ij} = {} & \tfrac{1}{2} v^{ji}_{ba} + t^{ac}_{lj} (\tfrac{1}{2} v^{cd}_{kl} t^{bd}_{ki} - P^{cb}_{li}) - v^{jc}_{ka} t^{bc}_{ki} + t^{bd}_{ji} (f^d_a + t^{ac}_{lk} H^{dc}_{kl} + c^d_q (c^s_a ((2A^{pq}_{sk} - A^{qp}_{sk}) G^p_k))) \\
& + t^{bc}_{kj} (v^{cd}_{lk} t^{ad}_{il} - O^{ca}_{ki} + v^{cd}_{kl} (\tfrac{1}{2} t^{ad}_{li} - 2t^{ad}_{il}) - v^{ic}_{ak}) + t^{bc}_{jk} (2O^{ca}_{ki} - P^{ca}_{ki} + t^{ad}_{il} (2v^{cd}_{kl} - v^{cd}_{lk}) + 2v^{ic}_{ak} - v^{ic}_{ka}) \\
& + c^r_a (\tfrac{1}{2} I^{br}_{ji} + \tfrac{1}{2} c^s_b (G^q_i L^{qs}_{rj}) + c^s_b (E^{qj}_{sr} G^q_i)) + \tfrac{1}{2} t^{ab}_{kl} J^{kl}_{ji} + t^{ba}_{kl} (\tfrac{1}{2} K^{jk}_{li} + N^{jk}_{li} + \tfrac{1}{2} v^{ji}_{kl}) \\
& + t^b_k (t^a_l N^{jk}_{li} - v^{jc}_{ka} t^c_i + t^{ac}_{lj} F^{ck}_{li}) + t^{ab}_{kj} (t^{dc}_{il} H^{cd}_{kl} + t^c_i M^c_k - f_o o^i_k - f^c_k t^c_i + t^c_l Q^{ic}_{lk}) \\
& + t^a_k (t^{bc}_{lj} F^{cl}_{ki} - c^r_k I^{br}_{ji} - t^d_j P^{db}_{ki} + t^{bc}_{jl} (F^{ck}_{li} - 2F^{cl}_{ki}) + \tfrac{1}{2} t^b_l J^{kl}_{ji} + v^{jc}_{lk} t^{bc}_{li} + v^{ic}_{kl} t^{bc}_{lj} + t^{bc}_{jl} Q^{ic}_{lk} \\
& \quad + \tfrac{1}{2} t^b_l K^{jl}_{ki} - v^{jc}_{bk} t^c_i + t^{bc}_{ji} (M^c_k - f^c_k) - v^{ij}_{kb} + \tfrac{1}{2} v^{ji}_{lk} t^b_l) \\
\text{residual}^{ab}_{ij} = {} & r^{ab}_{ij} + r^{ba}_{ji}
\end{aligned}
$$

Figure 4. Integrated optimization of CCSD T2 with AO-to-MO transforms.

letters $A-Q$. Only 7 of the original 12 v integrals are explicitly computed in the optimized form, whereas the expression using the other v integrals has been transformed to use other intermediates to reduce total operation cost.

Table 1 provides detailed information about the computational complexity of the optimized expressions for the different cases considered, showing the coefficients for the various higher-order polynomial terms for the arithmetic cost (counting each floating point addition or multiplication as one operation; we note that this is different from the convention used in previous publications, such as ref 9, in which a multiply-add pair is counted as one operation rather than two).

TABLE 1: Coefficients of Leading Terms of Symbolic Cost Function; $O = 10$, $M = V + O$; ("sep" Denotes Separated Optimization of CCSD T2 Expression and AO-to-MO Transform; "int" Denotes Integrated Optimization of CCSD T2 and AO-to-MO Transform; T Denotes the Number of CCSD Iterations)

	standard iteration						Brueckner basis					
	$V = 100$		$V = 500$		$V = 1000$		$V = 100$		$V = 500$		$V = 1000$	
leading terms of symbolic cost function	sep	int	sep	int	sep	int	sep	int	sep	int	sep	int
VM^4	2	0	2	0	2	0	$2T$	0	$2T$	0	$2T$	0
V^2M^3	2	0	2	0	2	0	$2T$	0	$2T$	0	$2T$	0
V^3M^2	2	0	2	0	2	0	$2T$	0	$2T$	0	$2T$	0
V^4M	2	0	2	0	2	0	$2T$	0	$2T$	0	$2T$	0
O^2M^4	0	$2T$	0	$2T$	0	$2T$	0	$2T$	0	$2T$	0	$2T$
O^2V^4	$2T$	0	$2T$	0	$2T$	0	$2T$	0	$2T$	0	$2T$	0
OM^4	2	$2T + 4$	2	$2T + 4$	2	$2T + 4$	$2T$	$6T$	$2T$	$6T$	$2T$	$6T$
OVM^3	2	2	2	0	2	0	$2T$	0	$2T$	0	$2T$	0
OV^2M^2	4	0	2	0	2	0	$2T$	0	$2T$	0	$2T$	0
OV^3M	4	0	4	0	4	0	$4T$	0	$4T$	0	$4T$	0
O^3V^3	$20T$	$16T$	$20T$	$16T$	$20T$	$16T$	$22T$	$18T$	$22T$	$18T$	$22T$	$18T$
O^3V^2M	0	0	0	0	0	0	0	0	0	$2T$	0	$2T$
OV^4	$2T$	0	$2T$	0	$2T$	0	$2T$	0	$2T$	0	$2T$	0
O^2M^3	4	$6T + 6$	4	$8T + 8$	4	$8T + 8$	$4T$	$14T$	$4T$	$14T$	$4T$	$14T$
O^2VM^2	6	$12T + 8$	6	$12T + 8$	6	$12T + 8$	$6T$	$18T$	$6T$	$18T$	$6T$	$18T$
O^2V^2M	8	$8T + 8$	8	$8T + 8$	8	$8T + 8$	$8T$	$16T$	$8T$	$16T$	$8T$	$16T$
O^2V^3	$10T$	$4T$	$10T$	$4T$	$10T$	$4T$	$14T$	$4T$	$14T$	$4T$	$14T$	$4T$
reduction factor	1		2.46		4.24		2.51		13.75		28.46	

The first six columns in Table 1 correspond to the standard CCSD model; the last six columns correspond to optimization for the Brueckner CCSD model. Alternate columns, labeled "sep" and "int", provide the coefficients of cost terms for the resulting expressions using separated and integrated optimization, respectively. Considering the first two columns (for $V = 500$), it is clear that the optimized expressions are very different. Some table entries have constant values, and others are scaled by T. A constant value implies that the corresponding term is evaluated only once (for example, the MO integrals in the expressions derived by separated optimization), whereas the entries scaled by T are executed repeatedly during every CCSD iteration. Since a single table is used for displaying the polynomial complexity terms for different expressions, we also have some zero entries when terms do not apply to a particular optimized expression.

With separated optimization, the optimized form has several contractions with computational complexity in the fifth power of V/M (for $V \gg O$, M is very close to V) arising from the explicit computation of the MO integrals. In contrast, integrated optimization produces optimized expressions without any terms involving the fifth power of V/M instead trading them for an $O(OM^4)$ term that is computed T times (once every CCSD iteration). When $O \times T$ is less than V, despite being recomputed every iteration, such a term has lower cost than the one-time explicit computation of the MO integrals. The last row of the table shows the ratio of total arithmetic operation count using the separated versus integrated optimization. For $V = 100$, both optimized expressions essentially have the same cost, but for the higher values of V, it can be seen that the integrated optimization produces a much more efficient form than integrated optimization, with the benefit increasing as V increases.

The right half of Table 1 shows the computational complexity terms for the optimized expressions for the Brueckner CCSD model, in which the AO integral transformation must be performed for every CCSD iteration. For both the separated approach and the integrated approach, each term is therefore scaled by T. Again, the optimized forms are clearly very different for separated versus integrated optimization. Relative to the standard CCSD scenario, for the Brueckner CCSD mode, the benefit of integrated optimization over separated optimization is significantly higher.

So far, the comparisons of different optimized forms have all been generated by the automated operation minimization algorithm. But how effective is the automatic optimization when compared with manually optimized formulations? To answer this question, we generated an optimized version of just the CCSD T2 equations and compared the complexity of the generated terms with a highly optimized closed-shell CCSD T2 form developed by Scuseria et al.[9] The optimized form produced by the automatic minimizer is shown in Figure 5. The computational complexity of the most significant terms is $2O^2V^4 + 20O^3V^3 + 10O^4V^2$ operations (counting each floating-point addition or multiplication as a separate operation). The manually optimized implementation from Scuseria et al.[9] is $(1/2)O^2V^4 + 8O^3V^3 + 2O^4V^2$ A close examination of the optimized forms shows that the difference is mainly due to two reasons. First, our compiler-generated expressions exploit antisymmetry but not another form of symmetry ("vertex" symmetry) that is used in the optimized form from Scuseria et al.: $A_{ij}^{ab} = A_{ji}^{ba}$. The most significant contraction causing the $O(O^2V^4)$ complexity is essentially the same contraction in both optimized forms, but is implemented by Scuseria et al. with one-fourth of the operation count due to maximal exploitation of such symmetry. Second, a close examination of the form of optimized equations in Scuseria et al.[9] demonstrates the need for more sophisticated intermediate steps (e.g., one that involves adding and subtracting a term to an intermediate term that significantly enhances overall possibility of reduction in operation count). We are in the process of incorporating vertex symmetry and enhancing the operation count minimization capability of our compiler using more sophisticated steps.

Implementing Tensor Contractions using Tuned Matrix Multiplication

Consider the following tensor contraction expression,

$$E[i,j,k] = \sum_{a,b,c} A[a,b,c]\, B[a,i]\, C[b,j]\, D[c,k]$$

where all indices range over N, and a, b, and c are contraction indices. The direct way to compute this would require $O(N^6)$

$$
\begin{aligned}
&A^{a}_{i} = v^{ad}_{ki} t^{d}_{k} \quad B^{ai}_{bj} = v^{ad}_{ib} t^{d}_{j} \quad C^{ai}_{jk} = v^{ad}_{ij} t^{d}_{k} \quad D^{ij}_{kl} = v^{cd}_{ij} t^{dc}_{kl} \qquad E^{ab}_{ij} = v^{ad}_{ki} t^{bd}_{jk} \\
&F^{ij}_{kl} = t^{c}_{i} C^{cj}_{kl} \quad G^{a}_{i} = v^{ad}_{il} t^{d}_{l} \quad H^{ai}_{bj} = v^{ca}_{ib} t^{c}_{j} \quad I^{ij}_{kl} = 0.5 v^{ij}_{kl} + v^{ic}_{kl} t^{c}_{j} \\
&r^{ab}_{ij} = 0.5 v^{ji}_{ba} + 0.5 v^{cd}_{ab} t^{dc}_{ji} + 2 t^{ad}_{il} (v^{cd}_{kl} (t^{bc}_{jk} - t^{bc}_{kj})) + t^{bc}_{kj} (0.5 v^{cd}_{kl} t^{ad}_{li} + E^{ca}_{ki} - v^{ic}_{ak} - B^{ck}_{ai}) + 0.5 t^{ab}_{kl} D^{kl}_{ji} \\
&\quad + t^{bd}_{ki} (v^{jd}_{kl} t^{a}_{l} - v^{jd}_{ka} + 0.5 v^{cd}_{kl} t^{ac}_{lj}) + t^{bc}_{jl} (2 v^{ic}_{al} - E^{ca}_{li} - H^{cl}_{ai} + 2 B^{cl}_{ai} - v^{ic}_{la} + t^{a}_{k} (v^{ic}_{lk} - 2 v^{ic}_{kl} - 2 C^{cl}_{ki} + C^{ck}_{li})) \\
&\quad + t^{ad}_{kj} (t^{b}_{l} C^{dl}_{ki} - H^{dk}_{bi}) - t^{b}_{k} (v^{jc}_{ka} t^{c}_{i}) + t^{ba}_{kl} (I^{ji}_{kl} + 0.5 F^{jk}_{li}) + t^{d}_{i} (v^{jd}_{ba} + 0.5 v^{cd}_{ba} t^{c}_{j}) \\
&\quad + t^{ab}_{kj} (v^{ic}_{lk} t^{c}_{l} - f_{o} o^{i}_{k} - 2 t^{c}_{i} G^{c}_{k} - 2 v^{ic}_{kl} t^{c}_{l} + t^{c}_{i} (A^{c}_{k} - f^{c}_{k}) + t^{dc}_{il} (v^{cd}_{kl} - 2 v^{cd}_{lk})) \\
&\quad + t^{bc}_{ji} (f^{c}_{a} + t^{d}_{k} (2 v^{dc}_{ka} - v^{cd}_{ka}) + t^{a}_{k} (A^{c}_{k} - f^{c}_{k} - 2 G^{c}_{k}) + v^{cd}_{kl} (t^{ad}_{lk} - 2 t^{ad}_{kl})) \\
&\quad + t^{a}_{k} (0.5 t^{b}_{l} D^{kl}_{ji} - t^{d}_{j} H^{dk}_{bi} + t^{bc}_{lj} C^{cl}_{ki} - v^{cd}_{kb} t^{dc}_{ji} + v^{ic}_{kl} t^{bc}_{lj} + 0.5 t^{b}_{l} F^{jl}_{ki} - v^{jc}_{bk} t^{c}_{i} - v^{ij}_{kb} + t^{b}_{l} I^{ji}_{lk}) \\
&\text{residual}^{ab}_{ij} = r^{ab}_{ij} + r^{ba}_{ji}
\end{aligned}
$$

Figure 5. CCSD T2 expression optimized separately from AO-to-MO transform.

arithmetic operations. However, as discussed in the first part of the paper, algebraic transformations can be used to reduce the number of operations to $O(N^4)$.

$$T1[a, b, k] = \sum_{c} A[a, b, c]\, D[c, k]$$

$$T2[a, j, k] = \sum_{b} T1[a, b, k]\, C[b, j]$$

$$E[i, j, k] = \sum_{a} T2[a, j, k]\, B[a, i]$$

Each of the three contractions for the operation-optimized form is essentially a generalized matrix multiplication. Since highly tuned library generalized matrix multiplication (GEMM) routines exist, it is attractive to translate the computation for each 2-tensor contraction node into a call to GEMM if possible. For the above 3-contraction example, the first contraction can be implemented directly as a call to GEMM with A viewed as an $N^2 \times N$ rectangular matrix and D as an $N \times N$ matrix. The second contraction, however, cannot be directly implemented as a GEMM call because the contraction index b is the middle index of $T1$. GEMM can be directly used only when summation indices and nonsummation indices in the contraction can be collected into two separate contiguous groups. However, $T1$ can first be "reshaped" via explicit layout transformation; that is, $T1[a, b, k] \rightarrow T1r[a, k, b]$. GEMM can then be invoked with the first operand, $T1r$, viewed as an $N^2 \times N$ array and the second input operand, C, as an $N \times N$ array. The result, which has the index order $[a, k, j]$, would also have to be reshaped to form $T2[a, j, k]$. Considering the last contraction, it might seem that some reshaping would be necessary to use GEMM. However, GEMM allows one or both of its input operands to be transposed. Thus, the contraction can be achieved by invoking GEMM with B as the first operand, in transposed form, and $T2[a, j, k]$ as the second operand, with shape $N \times N^2$.

In general, a sequence of multidimensional tensor contractions can be implemented using a sequence of GEMM calls, possibly with some additional array-reordering operations interspersed. Since the multiplication of two $N \times N$ matrices requires $O(N^3)$ operations and reordering of a $P \times Q$ matrix requires only $O(PQ)$ data moves, it might seem that the overhead of the layout transformation steps would be negligible relative to the time for matrix multiplication. However, as shown in the next section, a simple nested loop structure to perform the layout transposition can result in significant overhead. The remaining sections of this paper address the development of an efficient index permutation library for tensors. The problem of efficient transposition of 2D matrices is first addressed and is then used as the core function in implementing generalized tensor layout transformation.

```
for i = 0 to N1-1
  for j = 0 to N2-1
    B[i][j] =  A[j][i]
```

Figure 6. A simple implementation of matrix transposition.

Index Permutation Library for Tensors

In this section, we first present an overview of the problem of efficient 2D matrix transposition (discussed in detail elsewhere[10]) and then discuss its use in optimizing arbitrary index permutations of multidimensional arrays. Consider the simple double-nested loop in Figure 6. Although transposition might seem such a straightforward operation, existing compilers are unable to generate efficient code. For example, the program in Figure 6 was compiled using the Intel C compiler with "-O3" option. On an Intel Pentium 4 with a 533 MHz front side bus, it achieved an average data transfer bandwidth of 90.3MB/s, for single-precision arrays, with each dimension ranging from 3800 to 4200. This is only 4.4% of the sustained copy bandwidth achieved on the machine by the STREAM memory benchmark.[13]

On modern architectures, the cache hierarchy, the memory subsystem, and SIMD vector instructions (like SSE) are key factors to performance of matrix transpose, and there is interplay among them. Cache provides fast data and instruction buffers to on-chip computation resources and is often organized into multiple levels, including level 1 (L1) cache, level 2 (L2) cache, and so on. A cache is organized as a set of cache blocks (lines) whose typical sizes range from 16 bytes to 128 bytes. If a data element has multiple accesses during its stay in cache, *temporal locality* is exploited. If different elements within a cache line are accessed, *spatial locality* is exploited. Translation lookaside buffer (TLB) is a special CPU cache that memory management hardware uses to improve virtual address translation speed. Matrix transposition lacks temporal locality and has a large cache footprint. The data access pattern for the code in Figure 6 involves row-wise access of B but column-wise access of A in the inner loop. This results in poor spatial locality for A. If the loops are interchanged, excellent spatial locality can be obtained for A, but array B will now have poor spatial locality. The strided access pattern for column-wise access can potentially result in a large number of conflict misses in cache and TLB misses.

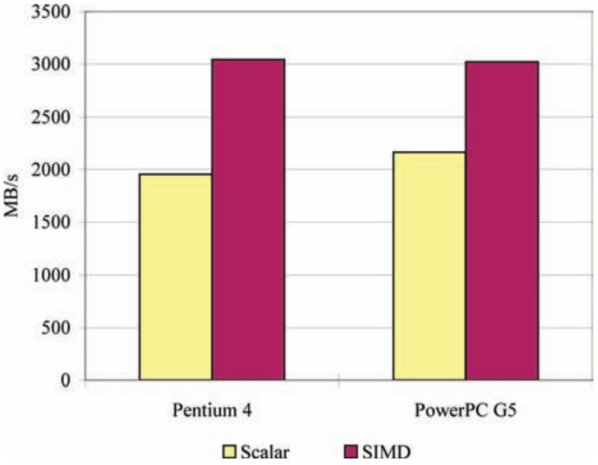

Figure 7. Improvement from using SIMD in memory copy.

Processors have adopted multimedia extensions characterized as single instruction multiple data (SIMD) units operating on packed short vectors. Examples of these SIMD extensions include SSE/SSE2/SSE3/SSE4 for Intel processors and VMX/AltiVec for PowerPC processors. The effective use of SIMD support can provide performance enhancement for matrix transposition in several ways.

To illustrate the potential benefits of employing SIMD extensions in memory bandwidth-bound computations, Figure 7 shows the performance difference for memory copy using scalar versus SIMD instruction sets on an Intel Pentium 4 and a PowerPC G5.

The reader is referred to a prior publication[10] for details on the issues to be addressed for efficient implementation of the matrix transposition operation through explicit attention to various architectural factors. A combination of offline analysis and empirical search are used to determine the best choice of optimization parameters. The empirical search is performed once at library installation time and is similar to the ATLAS approach to generating an efficient BLAS library.[14,15] The code generator takes as input the architectural parameters and generates multiple versions of code optimized for different categories of problem instances; at library invocation time, a dynamic search tree is traversed to determine which version of the code is actually executed.

The matrix transposition approach (presented elsewhere[10]) can be used to optimize arbitrary index permutations of multidimensional arrays. When going from 2D matrix transposition to higher dimensions, several issues must be considered:

1. It is important to reduce the number of generated code versions while optimizing for different permutations. Instead of having $n!$ code versions for all possible permutations of n-dimensional arrays, we generate only n versions. By always accessing the source array or the destination array in a fixed order, we calculate the access stride of each loop into the other array. In such a way, one code version handles $(n - 1)!$ permutations.

2. The decrease in dimension sizes with the increasing dimensionality impairs the benefits from optimizations, such as loop tiling. Due to the limited benefit of second-level tiling and TLB tiling with reduced dimensions, we have only one level of tiling when dimensionality n is larger than 3.

3. The index calculation overhead must be effectively controlled to achieve high performance. Instead of relying on compiler-generated code, we identify loop invariants and generate efficient indexing code by strength reduction.[16]

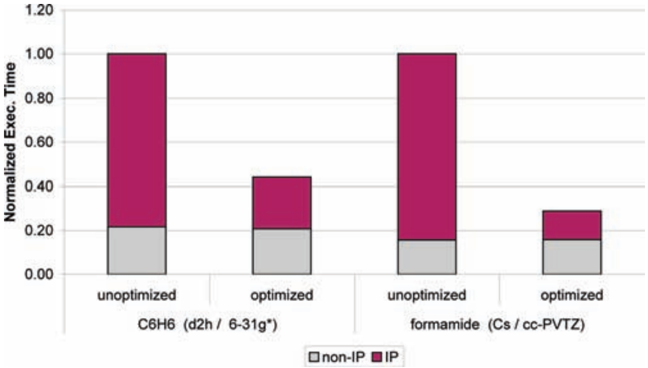

Figure 8. Normalized execution time of triples correction on a Pentium 4. Note: IP refers to index permutation.

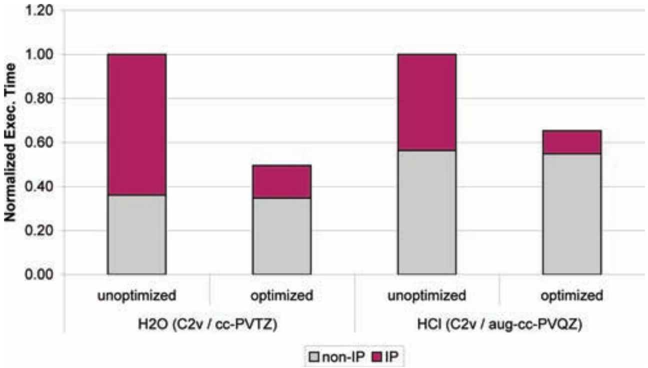

Figure 9. Normalized execution time of CCSDT on a Pentium 4. Note: IP refers to index permutation.

Following the optimization procedure with the above changes, we have developed a highly optimized index permutation library for both PowerPC and x86 architectures. We demonstrate the effectiveness of optimized index permutation operations by employing them in NWChem,[17] a widely used computational chemistry suite.

In NWChem, two variants of index permutation operations are used: without and with accumulation: (i) $A' = \text{permute}(A, p)$, where A is transformed to A' using index permutation p, and $A' = A' + c \times \text{permute}(A, p)$, where A is permuted, scaled by factor c and accumulated into A'.

Two representative computations are used in our evaluation: (1) the triples correction in the CCSD(T) computation and (2) the CCSDT computation. The experiments were conducted using NWChem version 4.7 on the same Pentium 4 platform used in the previous section. By replacing the original index permutation code in NWChem with the optimized version, significant performance improvements are obtained, as shown in Figures 8 and 9. We make the following observations:

1. The computation complexity of triples correction is $O(O^3V^4)$, whereas the index permutation cost of triples correction is $O(O^3V^3)$, which is mainly for symmetrization operations. However, the index permutation was found to dominate the computation of the triples correction. Our implementation offers overall speedups of 2.27 and 2.58 for the triples correction, respectively, for the two tested molecules. This improvement essentially comes from the index permutation speedups of 3.35 and 3.53, respectively.

2. The computation complexity of CCSDT is $O(O^3V^5)$; its index permutation cost is only $O(O^3V^3)$. The theoretical order complexity might suggest that the index permutation cost would be negligible; however, this is not the case, as seen from the experimental results: overall speedups of 2.02 and 1.74, respectively, are achieved for the two inputs.

Related Work

Common subexpression elimination is frequently used in traditional optimizing compilers.[16] Classical CSE techniques are focused on identifying the opportunities for value reuse; in most cases, such opportunities are rather limited, and they exist only for scalars. In contrast, the CSE problem in our work considers more complicated arithmetic structures and requires search for profitable alternatives in a large space of possible choices for value reuse. Algebraic properties (e.g., associativity) play a central role in our approach, but they are typically ignored in CSE techniques used in optimizing compilers.

Quantum chemists have proposed domain-specific heuristics for strength reduction and factorization for specific forms of tensor contraction expressions (e.g., for electronic structure methods, such as the coupled cluster methods[7,9,18]). For example, Scuseria et al.[9] and Janssen and Schaefer[18] developed a very highly optimized formulation for closed-shell CCSD equations. Common subexpression elimination is frequently employed in the manual formulation of quantum chemical methods. However, due to the complexity of the equations, it is prohibitively time-consuming to explore manually the large set of alternative formulations. With the help of the automated search techniques proposed here, it becomes feasible to explore a much larger space of possible formulations for operation minimization. Janssen and Schaefer[18] describe a common subexpression elimination algorithm for tensor contractions but do not present any experimental results.

Theoretical study and empirical evaluation of optimizing matrix transposition with cache performance considerations were conducted by Carter and Gatlin.[19,20] The authors conclude that, assuming conflict misses are unavoidable, it is impossible to be both cache- and register-efficient and employ an in-cache buffer. Other memory characteristics are not taken into account. Zhang et al.[21] focus on how to write an efficient bit-reversal program with loop tiling and data padding. Different implementations of matrix transposition were investigated by Chatterjee et al.,[22] with the conclusion that hierarchical nonlinear layouts are inherently superior to the standard layouts for the matrix transposition problem. We do not consider data padding or noncanonical layouts as an option, since we focus on generation of library routines that can be used with the standard data layouts used by quantum chemistry software suites, such as NWChem.

Several studies focus on how to generate or optimize intraregister permutations. The generation of register-level permutations is addressed by Kudriavtsev and Kogge[23] for optimizing data permutations at the instruction level, with a focus on SSE instructions. Ren et al.[24] present an optimization framework to eliminate and merge SIMD data permutation operations with a high-level abstraction. Both studies propagate data organization along data-flow graphs and focus on reducing intraregister permutations. We manually generate various versions of microkernels and empirically choose the best one. However, the manual process need only be repeated once for every vector instruction set. The limited number of vector instruction sets allows this process to be applicable across a wide range of processor architectures.

Empirical search employed in library generators, such as ATLAS,[14,15,25] has drawn great interest because of the complexity of analytical modeling of optimal parameters for modern architectures. However, empirical global search is often too expensive to apply. Yotov et al.[26] present a strategy employing both model-driven analysis and empirical search to decide optimization parameters in matrix multiplication. Chen et al.[27] also present an approach to combining compiler models and empirical search, using matrix multiplication and Jacobi relaxation as two examples. Our work is similar in spirit but is applied to a computation that is bandwidth-limited and has no temporal locality. Matrix transposition is similar to the level 1 BLAS kernels optimized by Whaley and Whalley[28] using an empirical search-based approach, but the presence of strided memory access in matrix transposition makes it harder to exploit spatial locality.

Conclusions

This paper has addressed two complementary aspects of performance optimization for tensor contraction expressions that arise in many body methods in quantum chemistry: (1) algebraic transformations to optimize the number of arithmetic operations, and (2) efficient multidimensional tensor permutation to facilitate effective use of tuned matrix multiplication libraries to perform tensor contractions. The effectiveness of the developed optimization approaches has been demonstrated using examples from coupled cluster models.

Acknowledgment. We thank the referees for their valuable feedback. This work has been supported in part by the U.S. National Science Foundation through Grants 0121676, 0121706, 0403342, 0508245, 0509442, 0509467, 0541409, 0811457, 0811781, 0926687, and 0926688; the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL); and a Discovery Grant, 262942-03, from the Natural Sciences and Engineering Research Council of Canada. ORNL is managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract DE-AC-05-00OR22725.

References and Notes

(1) Baumgartner, G.; et al. *Proceedings of the IEEE* **2005**, *93*, 276–292.

(2) Auer, A.; et al. *Mol. Phys.* **2006**, *104*, 211–218.

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

(4) Lee, T. J.; Scuseria, G. E. Achieving chemical accuracy with coupled cluster theory. In *Quantum Mechanical Electronic Structure Calculations with Chemical Accuracy*; Langhoff, S. R., Ed.; Kluwer Academic: Norwell, MA, 1997; pp47−109.

(5) Martin, J. M. L. Benchmark Studies on Small Molecules. In *Encyclopedia of Computational Chemistry*; v. R. Schleyer, P., Schreiner, P. R., Allinger, N. L., Clark, T., Gasteiger, J. P., Kollman, H. F. S., Eds.; Wiley & Sons: Berne, Switzerland, 1998; Vol.1; pp 115−128.

(6) Bartlett, R.; Purvis, G. *Int. J. Quantum Chem.* **1978**, *14*, 561–581.

(7) Stanton, J.; Gauss, J.; Watts, J.; Bartlett, R. *J. Chem. Phys.* **1991**, *94*, 4334–4345.

(8) Lam, C.; Sadayappan, P.; Wenger, R. *Parallel Processing Letters* **1997**, *7*, 157–168.

(9) Scuseria, G.; Janssen, C.; Schaefer, H. *J. Chem. Phys.* **1988**, *89*, 7382–7387.

(10) Lu, Q.; Krishnamoorthy, S.; Sadayappan, P. Combining analytical and empirical approaches in tuning matrix transposition. *PACT '06: Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, Seattle, Washington, 2006; pp 233−242.

(11) Koch, H.; Christiansen, O.; Kobayashi, R.; Jørgensen, P.; Helgaker, T. *Chem. Phys. Lett.* **1994**, *228*, 233.

(12) Hartono, A.; Sibiryakov, A.; Nooijen, M.; Baumgartner, G.; Bernholdt, D.; Hirata, S.; Lam, C.; Pitzer, R.; Ramanujam, J.; Sadayappan, P. Automated Operation Minimization of Tensor Contraction Expressions in Electronic Structure Calculations. *Proceedings of the ICCS 2005 5th International Conference on Computational Science*, Atlanta, Georgia, 2005; pp 155−164.

(13) McCalpin, J. *IEEE Computer Society TCCA Newsletter* **1995**, 19–25.

(14) Whaley, R. C.; Petitet, A.; Dongarra, J. J. *Parallel Computing* **2001**, *27*, 3–35.

(15) Whaley, R. C.; Dongarra, J. Automatically Tuned Linear Algebra Software. *SuperComputing 1998: High Performance Networking and Computing*, 1998, CD-ROM Proceedings.

(16) Aho, A. V.; Lam, M. S.; Sethi, R.; Ullman, J. D. *Compilers: Principles, Techniques, and Tools*, 2nd ed.;Addison-Wesley, 2006.

(17) High Performance Computational Chemistry Group *NWChem, A computational chemistry package for parallel computers*, Version 3.3; Pacific Northwest National Laboratory: Richland, WA, 1999.

(18) Janssen, C.; Schaefer, H. *Theor. Chim. Acta* **1991**, *79*, 1–42.

(19) Carter, L.; Gatlin, K. S. Towards an Optimal Bit-Reversal Permutation Program. *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, Palo Alto, California, 1998; pp 544−555.

(20) Gatlin, K. S.; Carter, L. Memory Hierarchy Considerations for Fast Transpose and Bit-Reversals. *HPCA '99: Proceedings of the 5th International Symposium on High Performance Computer Architecture*, Orlando, Florida, 1999; pp 33−43.

(21) Zhang, Z.; Zhang, X. *SIAM J. Sci. Comput.* **2000**, *22*, 2113–2134.

(22) Chatterjee, S.; Sen, S. Cache-Efficient Matrix Transposition. *HPCA '00: Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, Toulouse, France, 2000; pp 195−205.

(23) Kudriavtsev, A.; Kogge, P. Generation of permutations for SIMD processors. *LCTES'05: Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, Chicago, Illinois, 2005; pp 147−156.

(24) Ren, G.; Wu, P.; Padua, D. Optimizing Data Permutations for SIMD Devices. *PLDI '06: Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation*, Ottawa, Ontario, Canada, 2006; pp118−131.

(25) Yotov, K.; Li, X.; Ren, G.; Garzaran, M. J. S.; Padua, D.; Pingali, K.; Stodghill, P. *Proceedings of the IEEE* **2005**, *93*, 358–386.

(26) Yotov, K.; Pingali, K.; Stodghill, P. Think globally, search locally. *ICS '05: Proceedings of the 19th Annual International Conference on Supercomputing*, Cambridge, Massachusetts, 2005; pp 141−150.

(27) Chen, C.; Chame, J.; Hall, M. Combining Models and Guided Empirical Search to Optimize for Multiple Levels of the Memory Hierarchy. *CGO '05: Proceedings of the International Symposium on Code Generation and Optimization*, San Jose, California, 2005; pp 111−122.

(28) Whaley, R. C.; Whalley, D. B. Tuning High Performance Kernels through Empirical Compilation. *Proceedings of the 2005 International Conference on Parallel Processing (34th ICPP'2005)*, Oslo, Norway, 2005; pp 89−98.

JP9051215