Performance Modeling and Optimization of Parallel Out-of-Core Tensor Contractions*

Xiaoyang Gao[1] Swarup Kumar Sahoo[1] Qingda Lu[1] Gerald Baumgartner[2] Chi-Chung Lam[1]
J. Ramanujam[3] P. Sadayappan[1]
[1]Dept. of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210
{gaox,sahoo,luq,clam,saday}@cse.ohio-state.edu
[2]Dept. of Computer Science, Louisiana State University, Baton Rouge, LA 70803
gb@csc.lsu.edu
[3]Dept. of Electrical & Computer Engineering, Louisiana State University, Baton Rouge, LA 70803
jxr@ece.lsu.edu

Abstract

The Tensor Contraction Engine (TCE) is a domain-specific compiler for implementing complex tensor contraction expressions arising in quantum chemistry applications modeling electronic structure. This paper develops a performance model for tensor contractions, considering both disk I/O as well as inter-processor communication costs, to facilitate performance-model driven loop optimization for this domain. Experimental results are provided that demonstrate the accuracy and effectiveness of the model.

1 Introduction

The development of effective performance-model driven program transformation strategies for optimizing compilers is a challenging problem. We face this problem in the context of a domain-specific compiler targeted at a class of computationally demanding applications in quantum chemistry [2, 3]. A synthesis system is being developed for transformation into efficient parallel programs, of a high-level mathematical specification of a computation expressed as tensor contraction expressions. A tensor contraction is essentially a generalized matrix product involving multi-dimensional arrays. Often, the tensors are too large to fit in memory, so that out-of-core solutions are required. The optimization of a computation involving a collection of tensor contractions requires an accurate performance model for the core operation: a single tensor contraction, modeling both disk I/O costs and inter-processor communication costs. In this paper we address the problem of developing a performance model for parallel out-of-core tensor contractions.

The approach presented in this paper may be viewed as an example of the telescoping languages approach described in [15]. The telescoping languages/libraries approach aims at facilitating a high-level *scripting* interface for a domain-specific computation to the user, while achieving high performance that is portable across machine architectures, and compilation time that only grows linearly with the size of the user script. With this approach, library functions are pre-analyzed and appropriate annotations are included to provide information on performance characteristics. If user programs make heavy use of these library functions, the optimization of the user "script" is achieved using the performance characterization of the library functions, without requiring extensive analysis of the "expanded" program corresponding to inlined code for library functions. In a distributed computer, for efficient execution of out-of-core tensor contractions, two dominant overhead costs need to be reduced: inter-processor communication cost and local disk access cost. Many factors affect these costs, including the communication pattern, the parallel algorithms, data partitioning methods, loop orders, disk I/O placements and tile size selection. They are inter-related and can not be determined independently. The number of possible combinations is exceedingly large and searching them all is impractical. In this paper, we provide an approach, which can model the relationship between the space of possible structures and efficiently prune the search space to find the best solution in reasonable time.

This paper is organized as follows. In the next section, we introduce the main concepts and specify the parallel system supported by the algorithm. Section 3 discusses the impact of loop order and the placement of disk I/O statements. Algorithms used in *outside communication* pattern and *inside communication* pattern (defined in Section 2.1) are discussed in Section 4 and Section 5 respectively. Section 6 presents results from the application of the new algorithm to an example abstracted from NWChem [22]. We discuss related work in Section 7. Conclusions are provided in Section 8.

2 Preliminaries

Consider the following tensor contraction expression

$$C(a,b,c,d) = \sum_{m,n} A(a,b,m,n) \times B(c,d,m,n), \qquad (2.1)$$

where A and B are input arrays and C is the output array; m, n are the summation indices. If all indices range over N, $O(N^6)$ arithmetic operations will be required to compute this.

Notice that a tensor contraction is essentially a generalized

*Supported in part by the National Science Foundation through the Information Technology Research program (CHE-0121676 and CHE-0121706), and NSF grants CCR-0073800 and EIA-9986052.

matrix multiplication. The above expression can written as

$$C(I,J) = A(I,K) \times B(J,K), \qquad (2.2)$$

where $I \equiv \{a,b\}$, $J \equiv \{c,d\}$ and $K \equiv \{m,n\}$ are index sets considered as "macro-indices." This notation will be used in the rest of the paper. Consider a distributed-memory computer with P processors in which every processor has M words of local memory and unlimited local disk. If a processor needs data from the local disk of other processors, the required data will be first read by the owner processor and then communicated to the requesting processor. The inter-processor network bandwidth is denoted as B_c, and the local disk to memory bandwidth is B_d. Arrays A,B and C are either evenly distributed or fully replicated among all processors. An index set *dist* is used to represent the distribution pattern of an array. For example, if array A is distributed by index i and j, then $A.dist$ is $\langle i,j \rangle$. If A is replicated on all processors, then $A.dist$ is $\langle \rangle$.

The original size of an array is denoted as *array.size*. In a parallel algorithm, the size of an array required for local computation is denoted as *array.localsize*. If all required data can fit into memory, there is no disk I/O involved. Otherwise the array whose *array.localsize* larger than $M/3$ will be put on disk.

Because data sets are very large, we assume that both communication cost and disk I/O cost are dominated by the volume of data movement, not the number of data movements. The communication cost and disk I/O cost, can be calculated simply by dividing the transfered volume with the transferring bandwidth.

Three parallel matrix multiplication algorithms, *rotation, replication,* and *accumulation* are used. They will be discussed and compared in Section 2.2. The choice of the parallel algorithm decides the communication pattern. In the rotation algorithm, computation is executed in several steps. Each processor *circular-shifts* its local data with neighbors between computations. In the replication algorithm, one operand is *broadcast* to all processors. In the accumulation algorithm, the partial result of the entire target array is *reduced* among all processors. These communication patterns can be implemented by corresponding communication routines. Communication routines on out-of-core data will be carried out in several steps and results in extra disk access cost.

2.1 Communication Methods

When there is insufficient memory to hold all the remote data for the next computation to be performed locally on a processor, we can use one of two broad approaches to handling the out-of-core data: 1) first perform disk-to-disk transfer so that all remote data needed by a processor for its portion of the computation is first moved to its local disk, or 2) intersperse disk I/O with computation on in-core sub-arrays. We call the first method as the **outside communication method**, and the latter as the **inside communication method**. In the **outside communication method**, communication and local computation are separated from each other. All remote data for the next computation is fetched before the start of the computation and stored on disk. It may cause redundant disk access, but achieves minimal communication cost. With the **inside communication method**, communication and local computation are interleaved together. When one block of data is copied into memory, the owner processor performs computations on it, and passes it to other processors requiring it. When other processors receive remote data, they perform computations on it, and discard it without writing it to disk. This approach incurs extra communication cost, but minimizes disk access cost. Examples of these two communication methods are shown in Figure 1 and Figure 2. The choice of the communication method introduces a trade-off between communication cost and disk access cost. Thus, when available local memory is large enough to hold all the remote data, we can directly select the outside communication method.

2.2 Parallel Algorithms and Distribution Indices

Many approaches have been proposed for implementing parallel matrix multiplication. In this framework, three simple and common parallel algorithms are considered as the basis for an individual tensor contraction: rotation, replication and accumulation. Implementation details of these parallel algorithms are discussed next.

1. **Rotation:** We use a generalization of Cannon's algorithm as the primary template. In this approach, a logical view of the P processors as a two-dimensional $\sqrt{P} \times \sqrt{P}$ grid is used. To apply rotation parallel algorithm, each array is two-dimensional cyclic-block distributed along the two processor dimensions. A triplet $\{\ i,j,k\ \}$ formed by one index from each index set I,J,and K defines a distribution $\langle i,j \rangle$ for the result array C, and distribution $\langle i,k \rangle$ and $\langle k,j \rangle$ for the input arrays A and B. The computation is carried out in $\sqrt{P}$ steps. One processor holds a sub-block of array A,B and C respectively at any moment, performs a sub-matrix multiplication on them and transfers blocks A and B to its neighbors after the computation is done.

2. **Replication:** In this scheme, each processor locally holds one full input array and a strip of the other two arrays. In order to achieve good performance, we always replicate the smaller operand. Without loss of generality, we assume the size of array A is less than the size of B. Thus, to use the replication parallel algorithm, array A is replicated on all processors, $A.dist = \langle \rangle$, and arrays B and C are distributed by the same dimensions belonging to the index set J, $B.dist = C.dist = \langle j \rangle, j \in \mathbf{J}$. Replication communication can be modeled as an all-to-all broadcast communication operation, whose communication cost is a topology-dependent function. To simplify the problem, we assume that the interconnection network is completely connected in the rest of the paper. Thus, we use the expression

$$Replicate(S) = (S.size)/B_c \qquad (2.3)$$

to calculate the replication cost.

3. **Accumulation:** In order to apply the accumulation parallel algorithm, two operands are distributed by the same summation indices, $A.dist = B.dist = \langle k \rangle, k \in \mathbf{K}$, and the target array is replicated on all processors, $C.dist = \langle \rangle$. Every processor executes a partial matrix multiplication and accumulates the result at last. The accumulation can be modeled as an all-to-all reduction communication operation, whose communication cost depend on the inter-

processor topology. In the completely-connected network, the all-to-all reduction cost is

$$Reduce(S) = (S.size \times log(P))/B_c \qquad (2.4)$$

If the distribution of the input or output arrays are not suitable for a specific parallel algorithm, we need to rearrange the data before or after executing the parallel algorithm. The redistribution procedure is separated from the computation procedure.

The pseudocode of these three parallel algorithms using the inside communication method are shown in Figure 1. The corresponding pseudocode for the outside communication method are shown in Figure 2. Arrays A, B and C are out-of-core arrays that are distributed using a block-cyclic distribution among *P* processors in order to render the *Collective* disk I/O operations load-balanced. *Collective* disk I/O operations operate on global tiles, which consist of a set of local tiles. The corresponding *local* disk I/O operation is indicated under the *collective* disk I/O operation. In the pseudocode, the loop order of the It, Jt and Kt loops is not determined, all the disk I/O statements and message passing statements are placed inside the innermost loop. However, after the loop structure is defined, these data movement statements will be inserted at the appropriate places in the actual program.

2.3 The Overall Problem Definition

In this section, we define the problem of efficient execution of tensor contraction as follows.

For a given tensor contraction expression and some machine parameters, including the number of processors, the amount of physical available memory for every processor, the inter-processor network bandwidth, and the local disk to memory bandwidth, our goal is to determine:

- the communication method;
- the parallel algorithm and distributed indices;
- the order of the loops and disk I/O placements; and
- the tile sizes for each dimension

such that the total communication cost and the disk access cost are reduced.

For the input and output arrays, the algorithm can be used in either of these modes:

- the distribution of the input and output arrays are *unconstrained*, and can be chosen by the algorithm to optimize the communication cost; or
- the input and output arrays have a *constrained* distribution on to the processors in some pre-specified pattern.

The parallel execution can be decoupled into three stages:

1. redistribute the input arrays;
2. compute the tensor contraction expression in parallel; and
3. redistribute the output array.

PA	Distribution Constraints.
Rotation	$A.dist = \langle i,k \rangle, B.dist = \langle j,k \rangle, C.dist = \langle i,j \rangle$
Replication A	$B.dist = C.dist = \langle j \rangle$
Replication B	$A.dist = C.dist = \langle i \rangle$
Accumulation	$A.dist = B.dist = \langle k \rangle$

Table 1: Arrays distribution constraint fro different parallel algorithm

The total execution time is the sum of the execution times in the three stages. Because we only use load-balanced parallel algorithms, the computations are always evenly distributed among all the processors. We can ignore the calculation component, and consider only the communication overhead and the disk I/O cost. The total overhead cost for a specific parallel algorithm, which is denoted as *PA*, can be calculated by:

$$\begin{aligned} Overhead(PA) &= Redist(A, A.dist1, A.dist2) \\ &+ Redist(B, B.dist1, B.dist2) \\ &+ Redist(C, C.dist1, C.dist2) \\ &+ Computation(A, B, C, PA), \end{aligned}$$

where A.dist1 and B.dist1 are the initial distribution of the input arrays A and B, and C.dist2 is the expected distribution of the output array C. A.dist2, and B.dist2 are operand distribution patterns required for *PA*. C.dist1 is the target distribution pattern generated by *PA*. A.dist2, B.dist2 and C.dist1 must be compatible with each other by the distribution constraints of *PA*. The distribution constraints for different parallel algorithms is shown in Table 1.

If the initial distribution is the same as the final distribution, data re-arrangement is not necessary, and the redistribution cost is zero. Otherwise, the redistribution cost is the sum of the communication cost and the disk I/O cost, which depend on the redistribution scheme and machine specific interprocessor topology.

When a parallel algorithm is chosen for matrix multiplication, suitable distribution methods of the input and output arrays are decided as well. However, in a multi-dimensional tensor contraction expression, many distribution methods can be applied in a specific parallel algorithm. The choice of the distribution method will affect the redistribution cost in stages one and three. The overhead of parallel execution in stage two can be calculated independent of the distribution method. Here, we determine the parameters of the parallel out-of-core implementation that optimizes the total overhead cost. The choice of the distribution method that allows for optimizing the redistribution cost will be discussed later.

3 Loop Order and Disk I/O Placements

In this section, we will concentrate on the loop order and the placement of disk I/O statements. We will consider only the order of tiling loops since different orders of the intra-tile loops will not significantly affect the execution time.

Consider the tensor contraction expression given in expression (1). After tiling, the loops It, Jt, Kt will be the tiling loops as shown in Figures 2 and 1. Note that It, Jt, Kt are not single indices, but index vectors, i.e., they each consist of several loop indices. Orderings of these tiling loops will depend upon the order of the placement of the disk I/O statements. There

```
for It,Jt,Kt
  Collective Read $A_{Ii,Ki}$
  (Local Read $A_{Ii/\sqrt{P},Ki/\sqrt{P}}$ )
  Collective Read $B_{Ki,Ji}$
  (Local Read $B_{Ki/\sqrt{P},Ji/\sqrt{P}}$ )
  Collective Read $C_{Ii,Ji}$
  (Local Read $C_{Ii/\sqrt{P},Ji/\sqrt{P}}$)
  for $\sqrt{P}$ Rotations
    $C_{Ii,Ji}$ += $A_{Ii,Ki}$*$B_{Ki,Ji}$
    Circular-shift in-core $A_{Ii,Ki}$
    Circular-shift in-core $B_{Ki,Ji}$
  Collective Write $C_{Ii,Ji}$
  (Local Write $C_{Ii/\sqrt{P},Ji/\sqrt{P}}$)
```

(a): Rotation

```
for It,Jt,Kt
  Collective Read $A_{Ii,Ki}$
  (Local Read $A_{Ii,Ki}/P$)
  A2A In-Core Broadcast $A_{Ii,Ki}$
  Collective Read $B_{Ki,Ji}$
  (Local Read $B_{Ki,Ji/P}$)
  Collective Read $C_{Ii,Ji}$
  (Local Read $C_{Ii,Ji/P}$)
  $C_{Ii,Ji}$ += $A_{Ii,Ki}$*$B_{Ki,Ji}$
  Collective Write $C_{Ii,Ji}$
  (Local Write $C_{Ii,Ji/P}$)
```

(b): Replication

```
for It,Jt,Kt
  Collective Read $A_{Ii,Ki}$
  (Local Read $A_{Ii,Ki/P}$)
  Collective Read $B_{Ki,Ji}$
  (Local Read $B_{Ki/P,Ji}$)
  Local Read $C_{Ii,Ji}$
  $C_{Ii,Ji}$ += $A_{Ii,Ki}$*$B_{Ki,Ji}$
  All-Reduct In-Core $C_{Ii,Ji}$
  Local Write $C_{Ii,Ji}$
```

(c): Accumulation

Figure 1: Pseudocode of Inside Communication Method

```
for $\sqrt{P}$ Rotations
  for It,Jt,Kt
    Collective Read $A_{Ii,Ki}$
    (Local Read $A_{Ii/\sqrt{P},Ki/\sqrt{P}}$ )
    Collective Read $B_{Ki,Ji}$
    (Local Read $B_{Ki/\sqrt{P},Ji/\sqrt{P}}$ )
    Collective Read $C_{Ii,Ji}$
    (Local Read $C_{Ii/\sqrt{P},Ji/\sqrt{P}}$)
    $C_{Ii,Ji}$ += $A_{Ii,Ki}$*$B_{Ki,Ji}$
    Collective Write $C_{Ii,Ji}$
    (Local Write $C_{Ii/\sqrt{P},Ji/\sqrt{P}}$)
  Circular-shift Out-of-Core $A_{I,K}$
  Circular-shift Out-of-Core $B_{K,J}$
```

(a): Rotation

```
A2A Broadcast Out-of-Core $A_{I,K}$
for It,Jt,Kt
  Local Read $A_{Ii,Ki}$
  Collective Read $B_{Ki,Ji}$
  (Local Read $B_{Ki,Ji/P}$)
  Collective Read $C_{Ii,Ji}$
  (Local Read $C_{Ii,Ji/P}$)
  $C_{Ii,Ji}$ += $A_{Ii,Ki}$*$B_{Ki,Ji}$
  Collective Write $C_{Ii,Ji}$
  (Local Write $C_{Ii,Ji/P}$)
```

(b): Replication

```
for It,Jt,Kt
  Collective Read $A_{Ii,Ki}$
  (Local Read $A_{Ii,Ki/P}$)
  Collective Read $B_{Ki,Ji}$
  (Local Read $B_{Ki/P,Ji}$)
  Local Read $C_{Ii,Ji}$
  $C_{Ii,Ji}$ += $A_{Ii,Ki}$*$B_{Ki,Ji}$
  Local Write $C_{Ii,Ji}$
All-Reduct Out-of-Core $C_{I,J}$
```

(c): Accumulation

Figure 2: Pseudocode of Outside Communication Method

are three disk read statements corresponding to the three arrays A, B, and C. We need to consider six cases for the placement of read statements: *ABC*, *ACB*, *BAC*, *BCA*, *CAB*, *CBA*.

Consider the case where read statements are in the order *ABC* as shown in Figure 3. The three read statements will divide the tiling loops into four parts: D_1, D_2, D_3, and D_4. Each of these parts will contain some loops from each of the index vectors It, Jt, Kt. Let D_i contain index vectors It_i, Jt_i, Kt_i as shown in Figure 3(a). Considering the loops in part D_1, we note that if Jt_1 is non-empty, then disk I/O for A will be unnecessarily repeated several times. So Jt_1 should be empty to reduce the total volume of disk access for A. Considering the loops in part D_2, we note that if It_2 is non-empty, then disk I/O for B will be unnecessarily repeated several times. So It_2 should be empty to reduce the total volume of disk access for B. Further, if Kt_2 is non-empty, it will increase the memory requirement for the disk read of A. So, we can make Kt_2 empty and move those loops to part D_1 without increasing the overhead of disk I/O. Similarly, considering the loops in part D_3, we note that Kt_3 should be empty to reduce the total volume of disk access for C and that the loops in Jt_3 should be moved to part D_2 to reduce the memory requirement for disk access of B. Continuing in this fashion, we decide to put loops in It_4 in part D_3 and loops in Jt_4 in part D_2.

The simplified code is shown in Figure 3(b). Note, that the particular loops put in It_1 and It_1 will not affect the minimum Overhead cost, but they will determine whether the conditions under which we can achieve the minimum Overhead cost are satisfied or not. This will be explained in detail in later sections.

```
for $It1$, $Jt1$, $Kt1$
  Read $A$
  for $It2$, $Jt2$, $Kt2$
    Read $B$
    for $It3$, $Jt3$, $Kt3$
      Read $C$
      for $It4$, $Jt4$, $Kt4$
        $C+=A\times B$
```

(a) Initial groups

```
for $It1$, $Kt1$
  Read $A$
  for $Jt1$
    Read $B$
    for $It2$
      Read $C$
      for $Kt2$
        $C+=A\times B$
```

(b) After cleanup

Figure 3: Loop groups

4 Overhead Minimization for the Outside Communication Method

In this section, we analyze each of the three parallel algorithms (rotation, replication and accumulation) with the **outside communication** pattern and determine the minimal *Overhead* cost achievable along with the conditions under which this will be possible. In the expressions used in this and the next section, A, B, C will denote the sizes of arrays A, B, C, respectively; the terms I, J, K and It_1, Jt_1, Kt_1 will denote the corresponding loop bounds. The total number of processors will be denoted by P and the local memory available for the tiles of each array, which we assume to be one-third of the local memory per processor, is denoted by M. The combined memory of all processors is, therefore, MP.

4.1 Rotation

Let us consider the tensor contraction code with disk I/O placement order *ABC*, the outside communication pattern, and rotation type of parallelism as shown in Figure 2. The tiling loops are ordered as discussed in the previous section. Our goal is to determine the tile sizes (or the number of tiles) that will minimize the *Overhead* cost, including disk I/O cost and communication cost.

In this case, each of the three arrays are partitioned equally among the P processors. So we have $A.localsize = A/P$, $B.localsize = B/P$, and $C.localsize = C/P$. The communication corresponds to shifting the A and B arrays to adjacent processors. These communications happen $\sqrt{P}$ times and each of these also involves disk operations. Therefore, the total communication volume $\mathcal{V} = \sqrt{P} \times (\frac{A}{P} + \frac{B}{P}) = (\frac{A}{\sqrt{P}} + \frac{B}{\sqrt{P}})$. The disk access volume during communication $\mathcal{D}_1 = 2 \times (\frac{A}{\sqrt{P}} + \frac{B}{\sqrt{P}})$, since the disk is accessed twice, once for reading and once for writing. It is clear that these two terms are independent of the tile sizes. The disk access volume during the computation $\mathcal{D}_2 = \sqrt{P} \times (\frac{A}{P} + \frac{B}{P} \times It_1 + 2 \times \frac{C}{P} \times Kt_1) = \frac{A}{\sqrt{P}} + \frac{B}{\sqrt{P}} \times It_1 + 2 \times \frac{C}{\sqrt{P}} \times Kt_1$. The total disk access volume $\mathcal{D} = \mathcal{D}_1 + \mathcal{D}_2$. For simplicity, in the calculations below $\mathcal{D}$ will include only those two parts that depend on the number of tiles (or tile sizes).

It is clear that this term depends on the number of tiles. To minimize Overhead cost, we will have to minimize the disk access volume during the computation and hence It_1, Kt_1 should be made 1. But this is not possible due to the constraint that the tiles of array A, B and C should fit into memory. Here we assume that each of these array tiles occupies one third of the memory. The constraints involving tiles can be expressed as follows.

$$It_1 \times Kt_1 \geq \frac{A}{M \times P} \tag{4.1}$$

$$Jt_1 \times Kt_1 \geq \frac{B}{M \times P} \tag{4.2}$$

$$It_1 \times It_2 \times Jt_1 \geq \frac{C}{M \times P} \tag{4.3}$$

Note that only Eqn. 4.1 involves both It_1 and Kt_1, which we want to be 1. We will try to minimize $\mathcal{D}$ under the constraint of Eqn. 4.1. The other two equations can be simultaneously satisfied by using a large value the the unconstrained variables It_2 and Jt_1. Since we are trying to reduce the values of It_1 and Kt_1 while satisfying Eqn. 4.1, the Eqn. 4.1 can be written as $It_1 \times Kt_1 = \frac{A}{M \times P}$. With this modification, we can substitute the value of Kt_1 in the equation for $\mathcal{D}$ to get,

$$B \times It_1^2 - \sqrt{P} \times \mathcal{D} \times It_1 + \frac{2 \times C \times A}{M \times P} = 0 \tag{4.4}$$

The above quadratic equation will have a real solution under the condition that the quadratic curve discriminant $P \times \mathcal{D}^2 - 4 \times B \times (\frac{2 \times C \times A}{M \times P}) \geq 0$. In other words, for any real value of It_1, the minimum achievable value of $\mathcal{D}$ is $\frac{1}{P}\sqrt{\frac{8 \times A \times B \times C}{M}}$. This minimum value of $\mathcal{D}$ can be achieved with $It_1 = I \times \sqrt{\frac{2}{M \times P}}$ and $Kt_1 = \frac{K}{\sqrt{2 \times M \times P}}$. In order to satisfy Equations 4.2 and 4.3, we need to choose values of Jt_1 and It_2 that satisfy the conditions $Jt_1 \geq J \times \sqrt{\frac{2}{M \times P}}$ and $It_2 \geq 1$. Hence, the minimum total disk access volume is

$$2 \times (\frac{A}{\sqrt{P}} + \frac{B}{\sqrt{P}}) + \frac{A}{\sqrt{P}} + \frac{1}{P}\sqrt{\frac{8 \times A \times B \times C}{M}}. \tag{4.5}$$

If these values are not integers, the number of tiles will be set to the ceiling. There are two special cases if values of It_1 or Kt_1 are less than 1.

- *Case 1:* $I < \sqrt{\frac{M \times P}{2}}$

  In this case, we select the values as $It_1 = 1$, $Kt_1 = \frac{A}{M \times P}$, $Jt_1 \geq \frac{J}{I}$, $It_2 \geq \frac{I^2}{M \times P}$. The minimum total disk access volume during the computation in this case will be $\frac{A}{\sqrt{P}} + \frac{B}{\sqrt{P}} + 2 \times \frac{C}{\sqrt{P}} \times \frac{A}{M \times P}$.

- *Case 2:* $K < \sqrt{2 \times M \times P}$

  In this case, we select the values as $It_1 = \frac{A}{M \times P}$, $Kt_1 = 1$, $Jt_1 \geq \frac{B}{M \times P}$, $It_2 \geq \frac{M \times P}{K^2}$. The minimum total disk access volume during the computation in this case will be $\frac{A}{\sqrt{P}} + \frac{B}{\sqrt{P}} \times \frac{A}{M \times P} + 2 \times \frac{C}{\sqrt{P}}$.

We performed the analysis for the other five disk placement orders in a similar fashion. The results are shown in Table 2 and the details can be obtained from the associated technical report [11].

4.2 Replication

For this case, let us consider the tensor contraction code with disk I/O placement order *ABC*, outside communication pattern, and replication type of parallelism as shown in Figure 2. The tiling loops are ordered as discussed in the previous section. As in the case of rotation, our goal is to determine the tile sizes to minimize the Overhead cost.

In this case, the arrays B and C are partitioned equally among the P processors whereas A is replicated on all processors. So we have $A.localsize = A$, $B.localsize = B/P$, $C.localsize = C/P$. In this case, communication corresponds to broadcasting array A. Therefore, the total communication volume $\mathcal{V} = A$. The disk access volume during communication $\mathcal{D}_1 = A$. Also in this case the above two terms are independent of the tile sizes. The disk access volume during the computation $\mathcal{D}_2 = A + \frac{B}{P} \times It_1 + 2 \times \frac{C}{P} \times Kt_1$. The total disk access volume $\mathcal{D} = \mathcal{D}_1 + \mathcal{D}_2$. For simplicity, in the calculations below $\mathcal{D}$ will include only the parts that depend on the number of tiles.

It is clear that $\mathcal{D}$ depends on the number of tiles. To minimize the Overhead cost, we will have to minimize the disk access volume during the computation and hence It_1, Kt_1 should be set to 1. But this is not possible due to the constraint that the tiles of arrays A, B, and C fit into memory. The size constraints involving tiles can be expressed as follows.

$$It_1 \times Kt_1 \geq \frac{A}{M} \tag{4.6}$$

$$Jt_1 \times Kt_1 \geq \frac{B}{M \times P} \tag{4.7}$$

$$It_1 \times It_2 \times Jt_1 \geq \frac{C}{M \times P} \tag{4.8}$$

Our analysis here is similar to that for the case of rotation (Section 4.1). We will try to minimize $\mathcal{D}$ under the constraint of

Eqn. 4.6. The other two equations can be simultaneously satisfied by using a large value for the unconstrained variables It_2 and Jt_1. Since we are trying to reduce the values of It_1 and Kt_1 while satisfying Eqn. 4.6, the Eqn. 4.6 can be written as $It_1 \times Kt_1 = \frac{A}{M}$. With this modification, we can substitute the value of Kt_1 in the equation for $\mathcal{D}$ to get

$$B \times It_1^2 - P \times \mathcal{D} \times It_1 + \frac{2 \times C \times A}{M} = 0. \tag{4.9}$$

From the above equation, it should be clear that for any real value of It_1, the minimum achievable value of $\mathcal{D}$ is $\frac{1}{P}\sqrt{\frac{8 \times A \times B \times C}{M}}$. This minimum value of $\mathcal{D}$ can be achieved with $It_1 = I \times \sqrt{\frac{2}{M}}$ and $Kt_1 = \frac{K}{\sqrt{2 \times M}}$, $Jt_1 \geq \frac{J}{P} \times \sqrt{\frac{2}{M}}$ and $It_2 \geq 1$. These values satisfy all the constraints. Hence, the minimum total disk access volume is

$$A + A + \frac{1}{P}\sqrt{\frac{8 \times A \times B \times C}{M}} \tag{4.10}$$

If these values are not integers, the number of tiles will be set to the ceiling. There are two special cases if the values of It_1 or Kt_1 are less than 1. The analysis for these cases can be done as shown in Section 4.1.

We did the analysis for the other five disk placement orders as above. The results are shown in Table 2 and details can be obtained from [11].

4.3 Accumulation

In this section, we deal with the accumulation type of parallelism. Consider the tensor contraction code with accumulation type of parallelism as shown in Figure 2. Again our goal is to determine the tile sizes that will minimize the total Overhead cost.

In this case, arrays A and B are partitioned equally among the P processors whereas C is replicated on all processors. So we have $A.localsize = A/P$, $B.localsize = B/P$, $C.localsize = C$. In this case, the communication involves an All-Reduce operation of array C. Therefore, total communication volume $\mathcal{V} = C \times \log P$. The disk access volume during communication $\mathcal{D}_1 = C$. Again the total communication cost is independent of the tile sizes. The disk access volume during the computation $\mathcal{D}_2 = \frac{A}{P} + \frac{B}{P} \times It_1 + 2 \times C \times Kt_1$. The total disk access volume $\mathcal{D} = \mathcal{D}_1 + \mathcal{D}_2$. For simplicity, in the calculations below $\mathcal{D}$ will include only those parts that depend on the number of tiles.

It is clear that $\mathcal{D}$ depends on the number of tiles. As in the previous sections, to minimize the disk access volume during the computation, It_1 and Kt_1 should be made 1. But this is prevented by the constraint that the tiles of array A, B, and C should fit into memory. The constraints involving tiles in this case can be expressed as follows.

$$It_1 \times Kt_1 \geq \frac{A}{M \times P} \tag{4.11}$$

$$Jt_1 \times Kt_1 \geq \frac{B}{M \times P} \tag{4.12}$$

$$It_1 \times It_2 \times Jt_1 \geq \frac{C}{M} \tag{4.13}$$

We do the analysis similar to that in the previous subsections. We will try to minimize $\mathcal{D}$ under the constraint of Eqn. 4.11. The other two equations are simultaneously satisfied by using a large value for the unconstrained variables It_2 and Jt_1. As before, the Eqn. 4.11 can be written as $It_1 \times Kt_1 = \frac{A}{M \times P}$. Now substituting the value of Kt_1 in the equation for $\mathcal{D}$ we get

$$\frac{B}{P} \times It_1^2 - \mathcal{D} \times It_1 + \frac{2 \times C \times A}{M \times P} = 0 \tag{4.14}$$

From this equation it is clear that, for any real value of It_1, the minimum achievable value of $\mathcal{D}$ is $\frac{1}{P}\sqrt{\frac{8 \times A \times B \times C}{M}}$. This minimum value of $\mathcal{D}$ can be achieved with $It_1 = I \times \sqrt{\frac{2}{M}}$, $Kt_1 = \frac{K}{P \times \sqrt{2 \times M}}$, $Jt_1 \geq J \times \sqrt{\frac{2}{M}}$, and $It_2 \geq 1$. Hence, the minimum total disk access volume is

$$C + \frac{A}{P} + \frac{1}{P}\sqrt{\frac{8 \times A \times B \times C}{M}} \tag{4.15}$$

If these values are not integers, the number of tiles will be set to the ceiling. There are two special cases if the values of It_1 or Kt_1 are less than 1. Again, the analysis for these cases can be done as shown in the previous subsections.

We did the analysis for the other five disk placement orders as above. The results are shown in Table 2 and details can be obtained from [11].

5 Overhead Minimization for the Inside Communication Method

In this section, we analyze each of the three parallel algorithms possible with the **inside communication** pattern and determine the minimal Overhead cost achievable along with the conditions under which this will be possible.

5.1 Rotation

Consider the tensor contraction code with disk I/O placement order *ABC*, inside communication pattern, and the rotation type of parallelism as shown in Figure 1. The tiling loop ordering is decided as before. The goal is to determine the tile sizes (or the number of tiles) that will minimize the total Overhead cost.

In this case, each of the three arrays are partitioned equally among the P processors in a block-cyclic fashion. So we have $A.localsize = A/P$, $B.localsize = B/P$, $C.localsize = C/P$. In this case, communication corresponds to shifting the A and B arrays to adjacent processors. This communication happens $\sqrt{P}$ times for each iteration of the tiling loops and each of these also involves disk operations. Therefore, the total communication volume $\mathcal{V} = \sqrt{P} \times (\frac{A \times Jt_1}{P} + \frac{B \times It_1 \times It_2}{P}) = (\frac{A \times Jt_1}{\sqrt{P}} + \frac{B \times It_1 \times It_2}{\sqrt{P}})$. Due to in-memory transfer there will not be any disk access as part of the communication. The total disk access volume $\mathcal{D} = \frac{A}{P} + \frac{B}{P} \times It_1 + 2 \times \frac{C}{P} \times Kt_1$. For simplicity in the calculations below, $\mathcal{D}$ will not include the component $\frac{A}{P}$, which is independent of the number of tiles.

First we will try to optimize $\mathcal{D}$ and $\mathcal{V}$ independently. To minimize the communication volume $\mathcal{V}$, It_1, It_2 and Jt_1 should be made 1. But this is not possible due to the constraint that the tiles of array A, B, and C should fit into memory. Again we assume that each of these array tiles occupies one-third of memory. The constraints involving tiles are the same as those shown in the rotation case of **outside communication**.

	ABC or *ACB*	*BAC* or *BCA*	*CAB* or *CBA*
Outside/ Rotation	$\mathcal{V} = \frac{A}{\sqrt{P}} + \frac{B}{\sqrt{P}}$ $\mathcal{D} = \frac{A}{\sqrt{P}} + \frac{\sqrt{8ABC}}{P\sqrt{M}} + \frac{2}{\sqrt{P}}(A+B)$	$\mathcal{V} = \frac{A}{\sqrt{P}} + \frac{B}{\sqrt{P}}$ $\mathcal{D} = \frac{B}{\sqrt{P}} + \frac{\sqrt{8ABC}}{P\sqrt{M}} + \frac{2}{\sqrt{P}}(A+B)$	$\mathcal{V} = \frac{A}{\sqrt{P}} + \frac{B}{\sqrt{P}}$ $\mathcal{D} = \frac{2C}{\sqrt{P}} + \frac{2\sqrt{ABC}}{P\sqrt{M}} + \frac{2}{\sqrt{P}}(A+B)$
Outside/ Replication	$\mathcal{V} = A$ $\mathcal{D} = 2A + \frac{\sqrt{8ABC}}{P\sqrt{M}}$	$\mathcal{V} = A$ $\mathcal{D} = \frac{B}{P} + \frac{\sqrt{8ABC}}{P\sqrt{M}} + A$	$\mathcal{V} = A$ $\mathcal{D} = \frac{2C}{P} + \frac{2\sqrt{ABC}}{P\sqrt{M}} + A$
Outside/ Accumulation	$\mathcal{V} = C\log P$ $\mathcal{D} = \frac{A}{P} + \frac{\sqrt{8ABC}}{P\sqrt{M}} + C$	$\mathcal{V} = C\log P$ $\mathcal{D} = \frac{B}{P} + \frac{\sqrt{8ABC}}{P\sqrt{M}} + C$	$\mathcal{V} = C\log P$ $\mathcal{D} = 3C + \frac{2\sqrt{ABC}}{P\sqrt{M}}$

Table 2: Communication and Disk Access Volume for the **Outside Communication** pattern

	$I \geq \sqrt{MP}, J \geq \sqrt{MP},$ $K \geq \sqrt{MP}$	$I < \sqrt{MP}, J \geq \sqrt{MP},$ $K \geq \sqrt{MP}$	$I \geq \sqrt{MP}, J < \sqrt{MP},$ $K \geq \sqrt{MP}$	$I \geq \sqrt{MP}, J \geq \sqrt{MP},$ $K < \sqrt{MP}$
ABC	$\mathcal{D} = \frac{A}{P} + 3\sqrt{\frac{ABC}{MP^3}}$ $\mathcal{V} = 2\sqrt{\frac{ABC}{MP^2}}$ (1)	Lower bound is same as (3)	Lower bound is higher than (6)	$\mathcal{D} = \frac{A}{P} + \frac{AB}{MP^2} + \frac{2C}{P}$ $\mathcal{V} = 2\sqrt{\frac{ABC}{MP^2}}$ (2)
ACB	Lower bound is same as (1)	$\mathcal{D} = \frac{A}{P} + \frac{B}{P} + \frac{2AC}{MP^2}$ $\mathcal{V} = \frac{AC}{MP\sqrt{P}} + \frac{B}{\sqrt{P}}$ (3)	Lower bound is higher than (6)	Lower bound is same as (2)
BAC	$\mathcal{D} = \frac{B}{P} + 3\sqrt{\frac{ABC}{MP^3}}$ $\mathcal{V} = 2\sqrt{\frac{ABC}{MP^2}}$ (4)	Lower bound is higher than (3)	Lower bound is same as (6)	$\mathcal{D} = \frac{B}{P} + \frac{AB}{MP^2} + \frac{2C}{P}$ $\mathcal{V} = 2\sqrt{\frac{ABC}{MP^2}}$ (5)
BCA	Lower bound is same as (4)	Lower bound is higher than (3)	$\mathcal{D} = \frac{A}{P} + \frac{B}{P} + \frac{2BC}{MP^2}$ $\mathcal{V} = \frac{A}{\sqrt{P}} + \frac{BC}{MP\sqrt{P}}$ (6)	Lower bound is same as (5)
CAB	$\mathcal{D} = \frac{2C}{P} + 2\sqrt{\frac{ABC}{MP^3}}$ $\mathcal{V} = 2\sqrt{\frac{ABC}{MP^2}}$ (7)	$\mathcal{D} = \frac{2C}{P} + \frac{AC}{MP^2} + \frac{B}{P}$ $\mathcal{V} = \frac{AC}{MP\sqrt{P}} + \frac{B}{\sqrt{P}}$ (8)	$\mathcal{D} = \frac{2C}{P} + \frac{A}{P} + \frac{BC}{MP^2}$ $\mathcal{V} = \frac{A}{\sqrt{P}} + \frac{BC}{MP\sqrt{P}}$ (9)	$\mathcal{D} = \frac{2C}{P} + 2\sqrt{\frac{ABC}{MP^3}}$ $\mathcal{V} = 2\sqrt{\frac{ABC}{MP^2}}$ (10)
CBA	Lower bound is same as (7)	Lower bound is same as (8)	Lower bound is same as (9)	Lower bound is same as (10)

Table 3: Communication and Disk Access Volume for the **Inside Communication** pattern with rotation type parallelism

	ABC or *ACB*	*BAC* or *BCA*	*CAB* or *CBA*
Inside/Replication	$\frac{A}{P} + RA + \frac{\sqrt{8ABC}}{P\sqrt{M}}$	$\frac{B}{P} + \sqrt{\frac{8ABC(1+RP)}{MP^3}}$	$\frac{2C}{P} + \sqrt{\frac{4ABC(1+RP)}{MP^3}}$
Inside/Accumulation	$\frac{A}{P} + \frac{2\sqrt{ABC(2+R\log P)}}{P\sqrt{M}}$	$\frac{B}{P} + \frac{2\sqrt{ABC(2+R\log P)}}{P\sqrt{M}}$	$(2+R\log P)C + \frac{2\sqrt{ABC}}{P\sqrt{M}}$

Table 4: Effective Communication and Disk Access Volume for the **Inside Communication** pattern with replication/accumulation type parallelism

Note that only Equation 4.3 involves all the variables whose values we want to reduce namely It_1, It_2, and Jt_1. We will try to minimize $\mathcal{V}$ under the constraint of Equation 4.3. The other two equations can be simultaneously satisfied by using a large value for the unconstrained variable Kt_1. Since we are trying to reduce the values of It_1, It_2, and Jt_1 while satisfying Equation 4.3, the Equation 4.3 can be written as $It_1 \times It_2 \times Jt_1 = \frac{C}{M \times P}$. With this modification, we can substitute the value of Kt_1 in the equation for $\mathcal{D}$ to get

$$B \times (It_1 \times It_2)^2 - \sqrt{P} \times \mathcal{V} \times (It_1 \times It_2) + \frac{A \times C}{M \times P} = 0 \quad (5.1)$$

The above quadratic equation will have a real solution when the condition, quadratic curve discriminant $P \times \mathcal{V}^2 - 4 \times B \times (\frac{A \times C}{M \times P}) \geq 0$ is true. In other words, for any real value of It_1 the minimum achievable value of $\mathcal{V}$ is $\frac{2}{P}\sqrt{\frac{A \times B \times C}{M}}$. This minimum value of $\mathcal{V}$ can be achieved with $It_1 = \frac{I}{\sqrt{M \times P}}$, $It_2 = 1$, $Jt_1 = \frac{J}{\sqrt{M \times P}}$, and $Kt_1 \geq \frac{K}{\sqrt{M \times P}}$ which also satisfies the Equations 4.1 and 4.2. Hence, the minimum total communication volume is

$$\frac{2}{P}\sqrt{\frac{A \times B \times C}{M}} \quad (5.2)$$

Now we will minimize the disk access volume independently. Note that It_1 and Kt_1 should be made 1 in this case. But this is not possible due to the constraint that the tiles of arrays A, B, and C should fit into the memory. We will try to minimize $\mathcal{D}$ under the constraint of Eqn. 4.1. The other two equations can be simultaneously satisfied by using a large value for the unconstrained variables It_2 and Jt_1. The Eqn. 4.1 in this case can be written as $It_1 \times Kt_1 = \frac{A}{M \times P}$. With this modification, we can substitute the value of Kt_1 in the equation for $\mathcal{D}$ to get

$$B \times It_1^2 - P \times \mathcal{D} \times It_1 + \frac{2 \times C \times A}{M \times P} = 0 \quad (5.3)$$

From this equation we can see that for any real value of It_1 the minimum achievable value of $\mathcal{D}$ is $\sqrt{\frac{8 \times A \times B \times C}{M \times P^3}}$. This minimum value of $\mathcal{D}$ can be achieved with $It_1 = I \times \sqrt{\frac{2}{M \times P}}$ and $Kt_1 = \frac{K}{\sqrt{2 \times M \times P}}$, $Jt_1 \geq J \times \sqrt{\frac{2}{M \times P}}$, and $It_2 \geq 1$. These values will also satisfy Equations 4.2 and 4.3. Hence, the minimum total disk access volume is

$$\frac{A}{P} + \sqrt{\frac{8 \times A \times B \times C}{M \times P^3}} \quad (5.4)$$

But it is obvious that the number of tiles does not match with that of the previous analysis to minimize communication volume. So we cannot optimize both the communication volume and disk access volume. We have computed the Overhead cost for both the cases and we choose the one which has the smaller Overhead cost. In this case we choose the number of tiles that optimizes the communication volume as this gives the least Overhead cost. The values of communication and disk access volume are as follows with these tile sizes:

$$\mathcal{V} = \frac{2}{P}\sqrt{\frac{A \times B \times C}{M}} \quad (5.5)$$

$$\mathcal{D} = \frac{A}{P} + 3\sqrt{\frac{A \times B \times C}{M \times P^3}} \quad (5.6)$$

There are three special cases if values of It_1, Jt_1, or Kt_1 are less than 1.

- *Case 1:* $I < \sqrt{M \times P}$, $J \geq \sqrt{M \times P}$, $K \geq \sqrt{M \times P}$,

  In this case, the least overhead that can be achieved is $\mathcal{V} = \frac{A \times C}{\sqrt{M \times P^3}} + \frac{B}{\sqrt{P}}$ and $\mathcal{D} = \frac{A}{P} + \frac{B}{P} + \frac{2 \times C \times A}{M \times P^2}$ with $It_1 = 1$, $It_2 = 1$, $Jt_1 = \frac{C}{M \times P}$, $Kt_1 = \frac{A}{M \times P}$. But with these values, Eqn. 4.2 is not satisfied. This is not a problem, though, as the lower bound achieved in this case is higher than the lower bound of some other cases as shown in the Table 3.

- *Case 2:* $I \geq \sqrt{M \times P}$, $J < \sqrt{M \times P}$, $K \geq \sqrt{M \times P}$,

  This case similar to case 1.

- *Case 3:* $I \geq \sqrt{M \times P}$, $J \geq \sqrt{M \times P}$, $K < \sqrt{M \times P}$,

  In this case, the least overhead that can be achieved is $\mathcal{V} = 2\sqrt{\frac{A \times B \times C}{M \times P^2}}$ and $\mathcal{D} = \frac{A}{P} + \frac{B}{P} \times \frac{A}{M \times P} + \frac{2 \times C}{P}$ with $It_1 = \frac{A}{M \times P}$, $It_2 = \frac{\sqrt{M \times P}}{K}$, $Jt_1 = \frac{J}{\sqrt{M \times P}}$, $Kt_1 = 1$. With these values, all the constraints are also satisfied.

We did the analysis for the other five disk placement orders as above. The results of the analysis are shown in Table 3 and details can be obtained from [11].

5.2 Replication

For this case, let us consider the tensor contraction code with disk I/O placement order *ABC*, an inside communication pattern, and the replication type of parallelism as shown in Figure 1.

In this case, arrays B and C are partitioned equally among the P processors whereas A is replicated on all processors. So we have $A.localsize = A$, $B.localsize = B/P$, $C.localsize = C/P$. In this case, communication corresponds to an in-core broadcast of array A. Therefore, the total communication volume $\mathcal{V} = A$, and it is independent of the tile sizes. The total disk access volume $\mathcal{D} = \frac{A}{P} + \frac{B}{P} \times It_1 + 2 \times \frac{C}{P} \times Kt_1$.

The constraints involving tiles are the same as those shown in the replication part of outside communication. We do the analysis similar to the ones for the earlier cases. The minimum achievable value of $\mathcal{D}$ can be computed as $\frac{A}{P} + \frac{1}{P}\sqrt{\frac{8 \times A \times B \times C}{M}}$. This minimum value of $\mathcal{D}$ can be achieved with $It_1 = I \times \sqrt{\frac{2}{M}}$ and $Kt_1 = \frac{K}{\sqrt{2 \times M}}$, $Jt_1 \geq \frac{J}{P} \times \sqrt{\frac{2}{M}}$ and $It_2 \geq 1$. These values satisfy all the constraints. The analysis for the special cases can be done in the earlier sections.

The result of the analysis for the other five disk placement orders are shown in Table 4 and details can be obtained from [11]. Note that the values shown in this table are the effective communication and disk access volume $EffVol = \mathcal{D} + R \times \mathcal{V}$, where $R = \frac{B_d}{B_c}$, where B_d is the disk bandwdith and B_c is the communication (network) bandwidth.

5.3 Accumulation

In this section, we deal with the accumulation type of parallelism. Consider the tensor contraction code with the accumulation type of parallelism as shown in Figure 1. In this case, arrays A and B are partitioned equally among the P processors whereas C is replicated on all processors. So we have

$A.localsize = A/P$, $B.localsize = B/P$, $C.localsize = C$. In this case, communication involves in-core All-Reduce operation of array C. Therefore, the total communication volume $\mathcal{V} = C \times Kt_1 \times \log P$. The total disk access volume $\mathcal{D} = \frac{A}{P} + \frac{B}{P} \times It_1 + 2 \times C \times Kt_1$. In this case, we can optimize the total overhead cost, which is $\frac{EffVol}{B_d}$, where *EffVol* is the effective communication and disk access volume given by (note that R is defined at the end of Section 5.2)

$$EffVol = \frac{A}{P} + \frac{B}{P} \times It_1 + C \times Kt_1 \times (2 + R \times \log P). \qquad (5.7)$$

Our goal is to minimize *EffVol* under the constraints involving tile sizes that are shown in the accumulation section of the previous section. We proceed as before and compute the minimum achievable value of *EffVol*, which is found to be $\frac{A}{P} + 2 \times \sqrt{\frac{ABC(2+R\times\log P)}{M\times P^2}}$. This minimum value is achieved with $It_1 = I \times \sqrt{\frac{(2+R\times\log P)}{M}}$, $Kt_1 = \frac{K}{P\times\sqrt{(2+R\times\log P)\times M}}$, $Jt_1 \geq J \times \sqrt{\frac{(2+R\times\log P)}{M}}$, and $It_2 \geq 1$.

The special cases are handled as before. The analysis for the other five disk placement orders are also done as above. The results are shown in Table 4 and details can be obtained from [11]. Again, note that the values in the table give the minimum value of *EffVol*.

6 Experiments

Our performance models for the various approaches to parallel out-of-core tensor contractions were evaluated on an Itanium-2 cluster at the Ohio Supercomputer Center. The configuration of the cluster is shown in Table 5. All the programs were compiled with the Intel Itanium Fortran Compiler for Linux. We considered three example computations.

(1) Square Matrix Multiplication:

$$C(I,J) += A(I,K) \times B(J,K) \qquad (6.1)$$

In order to limit the execution time we ran "scaled down" experiments by setting the available physical memory limit to 64Mbytes. All the array dimensions were set to 4000. The parallel programs were run on 4 processors. We implemented parallel programs for the six methods discussed earlier. Table 6 compares the predicted costs for I/O and communication with the measured costs for the different approaches. It can be seen that there is a good match between predicted and actual times, and that the difference in performance of the various methods is quite significant.

(2) 4-index transform: This expression (also referred to as the AO-to-MO transform) is commonly used to transform two-electron integrals from an atomic orbital (AO) basis to a molecular orbital (MO) basis.

$$T1[a,b,c,d] += A[a,b,c,p] \times B[p,d] \qquad (6.2)$$

The size of all dimensions was set to 800. The parallel program was run on 4 processors. Between the different algorithms, we can find the best solution to be outside replication. The predicted overheads for the different parallel algorithms are shown in Table 7.

(3) CCSD: We used a sub-expression from the CCSD (Coupled Cluster Singles and Doubles) model [1, 20, 21] for determine electronic structures.

$$T1[i,j] += A[i,a,b,c] \times B[a,b,c,j] \qquad (6.3)$$

The size of all dimensions was set to 800. The parallel program was run on 4 processors. The best solution on the current machine can be seen to be outside accumulation. The predicted values of different parallel algorithms are shown in Table 7.

The effective choice of parallel algorithms results in a noticeable improvement in the communication cost for most cases. The ratio of disk bandwidth and interprocessor network bandwidth determines which factor dominates the total execution time. In previous experiments, because the network is almost twenty times faster than the disk, the disk cost dominated. In such a situation, the inside rotation algorithm is the best. However, using our model, we are able to predict the best choice for a given machine and problem characteristic. Table 8 shows such an example for a given matrix multiplication and disk bandwidth, where $I = 160000$, $J = 160000$, $K = 160000$, and $B_d = 10MB/s$. If we use a 100M Ethernet as the interconnection network and run the program on 4 processors, then the best parallel algorithm is outside replication. If we use Myrinet and run the program on 4 processors, the best solution becomes inside rotation.

7 Related Work

The issues arising in optimizing locality in the context of tensor contractions has been previously addressed by us, focusing primarily on minimizing memory-to-cache data movement [8, 9]. This approach was extended to the disk-memory hierarchy in [16], where a greedy approach to disk read/write placement was taken. For each set of tile sizes, the algorithm places read/write statements immediately inside those loops at which the memory limit is exceeded. In [17], a set of candidate fusion structures with disk I/O placements was taken as input and the tile size search space was explored. The search space was divided into feasible and infeasible solution spaces and their boundary was shown to contain the optimal solution. An algorithm was developed to locate the boundary efficiently and a steepest ascent hill-climbing used to determine an efficient solution for the tile sizes.

There has been some work in the area of software techniques for optimizing disk I/O. These include parallel file systems, compile time [4–6, 12–14, 18, 19] and runtime libraries and optimizations [7, 23]. Bordawekar et al. [4, 5] discuss several compiler methods for optimizing out-of-core programs in High Performance Fortran. Bordawekar et al. [6] develop a scheduling strategy to eliminate additional I/O arising from communication among processors; this paper is among the few that address the impact of scheduling on disk I/O overhead in a parallel context. Solutions for choreographing disk I/O with computation are presented by Paleczny et al. [19]. They organize computations into groups that operate efficiently on data accessed in chunks. Compiler-directed prefetching is discussed by Mowry et al. [18]. ViC* (Virtual C*) [10] is a preprocessor that transforms out-of-core C* programs into in-core programs with appropriate calls to the ViC* I/O library. Kandemir et al. [12–14] develop file layout and loop transformations for reducing I/O. None of these techniques address performance modeling and optimization of of parallel out-of-core computations addressing both disk I/O costs and interprocessor communication overheads.

There has been some work in the design of out-of-core linear algebra libraries [24, 25]. But, we are not aware of any work that addresses the detailed modeling of disk I/O and

Node	OS	Compiler	Memory	Network Bandwidth	Disk Bandwidth
Dual 900MHz	Linux	efc	1GB	200MB/s	8MB/s

Table 5: Configuration of the Itanium 2 cluster at OSC

Parallel Method	Predicted Overhead (sec.)	Measured Overhead (sec.).
Inside/Rotation	25.28	28.8827
Outside/Rotation	75.42	70.1237
Inside/Replication	24.718	23.5525
Outside/Replication	56.9	63.1590
Inside/Accumulation	53.792	54.7575
Outside/Accumulation	73.792	78.6768

Table 6: Predicted and Empirical results

	4index			ccsd		
	Disk I/O Volume(MB)	Comm. Volume(MB)	Total Time(sec.)	Disk I/O Volume(MB)	Comm. Volume(MB)	Total Time(sec.)
Outside/Rotation	1024000.32	204800.32	829440	1024000	409600	839680
Outside/Replica.	307200.64	0.64	245760	512000	409600	438563
Outside/Accumula.	921600.16	819200	778240	204800.64	1.28	163840
Inside/Rotation	307200.16	205824.32	256051	204800.16	409600	184320
Inside/Replica.	-	-	-	921600.32	409600	766243
Outside/Accumula.	512000	1146880	466944	-	-	

Table 7: Predicted performance results on 4 processors for ccsd and 4index-transform

	I=J=K=160000 , 4 Processors		I = J = K =640000, 16 Processors	
	$B_c = 10MB/s$	$B_c = 200MB/s$	$B_c = 10MB/s$	$B_c = 200MB/s$
Outside/Rotation	281920	262464	**3855360**	3699712
Outside/Replication	**259683**	232168	4224000	3601408
Outside/Accumulation	318784	264307	6018048	4274790
Inside/Rotation	310240	**120240**	4040960	**1000960**
Inside/Replication	268248	123502	4636610	1330921
Inside/Accumulation	298304	243827	5690368	3947110

Table 8: When B_d=10MB/s, predicted best disk/communication overheads (in sec.)

inter-processor communication costs, in addition to an evaluation and optimization of the overall performance of parallel out-of-core computations.

8 Conclusion and Future Work

This paper addressed the problem of developing performance models for a core computation – tensor contractions – in the context of a domain-specific compiler targeting a class of computations in quantum chemistry. The cost of disk I/O and interprocessor communication for different data partitions and tile sizes was modeled for various computational alternatives. The models were experimentally evaluated and the predictions were shown to match measured results. It was also seen that the optimal choice of parallel algorithm was dependent on the characteristics of both the tensor structure as well as machine parameters. Further work is in progress for using this framework to optimize tensor contraction expressions with a sequence of tensor contractions.

Acknowledgments

We would like to thank the Ohio Supercomputer Center (OSC) for the use of their computing facilities.

References

[1] R.J. Bartlett and G.D. Purvis. Many-body perturbation theory, coupled-pair many-electron theory and the importance of quadruple excitations for the correlation problem. *Int. J. Quantum Chem.*, 14:561–581, 1978.

[2] 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*, 2005.

[3] 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.

[4] R. Bordawekar. *Techniques for Compiling I/O Intensive Parallel Programs*. PhD thesis, Dept. of Electrical and Computer Eng., Syracuse University, April 1996.

[5] R. Bordawekar, A. Choudhary, K. Kennedy, C. Koelbel, and M. Paleczny. A model and compilation strategy for out-of-core data-parallel programs. In *Proc. of the Fifth ACM Symposium on Principles and Practice of Parallel Programming*, 1995.

[6] R. Bordawekar, A. Choudhary, and J. Ramanujam. Automatic optimization of communication in out-of-core stencil codes. In *Proc. of Tenth ACM International Conference on Supercomputing*, pages 366–373, 1996.

[7] Y. Chen, M. Winslett, Y. Cho, and S. Kuo. Automatic parallel I/O performance optimization. In *Proc. of Tenth Annual ACM Symposium on Parallel Algorithms and Architectures*, 1998.

[8] D. Cociorva, J. Wilkins, G. Baumgartner, P. Sadayappan, J. Ramanujam, M. Nooijen, D. E. Bernholdt, and R. Harrison. Towards automatic synthesis of high-performance codes for electronic structure calculations: Data locality optimization. In *Proc. of the Intl. Conf. on High Performance Computing*, volume 2228, pages 237–248. Springer-Verlag, 2001.

[9] D. Cociorva, J. Wilkins, C. Lam, G. Baumgartner, P. Sadayappan, and J. Ramanujam. Loop optimization for a class of memory-constrained computations. In *Proc. of the Fifteenth ACM International Conference on Supercomputing (ICS'01)*, pages 500–509, 2001.

[10] T. Cormen and A. Colvin. ViC*: A preprocessor for virtual-memory C*. Technical Report PCS-TR94-243, Dartmouth College, November 1994.

[11] X. Gao, S. Sahoo, Q. Lu, G. Baumgartner, J. Ramanujam, C. Lam, and P. Sadayappan. Compiler techniques for efficient parallelization of tensor contraction expressions. Technical Report OSU-CISRC-12/17–TR65, The Ohio State University, Columbus, OH, December 2004.

[12] M. Kandemir, A. Choudhary, and J. Ramanujam. An I/O conscious tiling strategy for disk-resident data sets. *The Journal of Supercomputing*, 21(3):257–284, 2002.

[13] M. Kandemir, A. Choudhary, J. Ramanujam, and R. Bordawekar. Compilation techniques for out-of-core parallel computations. *Parallel Computing*, 24(3-4):597–628, June 1998.

[14] M. Kandemir, A. Choudhary, J. Ramanujam, and M. Kandaswamy. A unified framework for optimizing locality, parallelism and communication in out-of-core computations. *IEEE Transactions of Parallel and Distributed Systems*, 11(7):648–668, July 2000.

[15] K. Kennedy, B. Broom, K. Cooper, J. Dongarra, R. Fowler, D. Gannon, L. Johnsson, J. M. Crummey, and L. Torczon. Telescoping languages: A strategy for automatic generation of scientific problem-solving systems from annotated libraries. *JPDC*, 61(12):1803–1826, December 2001.

[16] S. Krishnan. Datalocality optimization for synthesis of out-of-core programs. Master's thesis, The Ohio State University, Columbus, OH, September 2003.

[17] S. Krishnan, S. Krishnamoorthy, G. Baumgartner, D. Cociorva, C. Lam, P. Sadayappan, J. Ramanujam, D. E. Bernholdt, and V. Choppella. Data locality optimization for synthesis of efficient out-of-core algorithms. In *Proc. of the Intl. Conf. on High Performance Computing*, 2003.

[18] T. Mowry, A. Demke, and O. Krieger. Automatic compiler-inserted I/O prefetching for out-of-core applications. In *Proc. of Second Symposium on Operating Systems Design and Implementations*, pages 3–17, 1996.

[19] M. Paleczny, K. Kennedy, and C. Koelbel. Compiler support for out-of-core arrays on parallel machines. Technical Report 94509-S, Rice University, Houston, TX, December 1994.

[20] J.A. Pople, R. Krishnan, H.B. Schlegel, and J.S. Binkley. Electron correlation theories and their application to the study of simple reaction potential surfaces. *Int. J. Quantum Chem.*, 14:545–560, 1978.

[21] 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.

[22] T. P. Straatsma, E. Aprà, T. L. Windus, E. J. Bylaska, W. de Jong, S. Hirata, M. Valiev, M. Hackler, L. Pollack, R. Harrison, M. Dupuis, D. M. A. Smith, J. Nieplocha, V. Tipparaju, M. Krishnan, A. A. Auer, E. Brown, G. Cisneros, G. Fann, H. Früchtl, J. Garza, K. Hirao, R. Kendall, J. Nichols, K. Tsemekham, K. Wolinski, J. Anchell, D. Bernholdt, P. Borowski, T. Clark, D. Clerc, H. Dachsel, M. Deegan, K. Dyall, D. Elwood, E. Glendening, M. Gutowski, A. Hess, J. Jaffee, B. Johnson, J. Ju, R. Kobayashi, R. Kutteh, Z. Lin, R. Littlefield, X. Long, B. Meng, T. Nakajima, S. Niu, M. Rosing, G. Sandrone, M. Stave, H. Taylor, G. Thomas, J. van Lenthe, A. Wong, and Z. Zhang. *NWChem, A Computational Chemistry Package for Parallel Computers, Version 4.6*. Pacific Northwest National Laboratory, Richland, Washington 99352–0999, USA, 2004. `http://www.emsl.pnl.gov/docs/nwchem/`.

[23] R. Thakur, R. Bordawekar, A. Choudhary, R. Ponnusamy, and T. Singh. PASSION runtime library for parallel I/O. In *Proc. of Scalable Parallel Libraries Conference*, pages 119–128, 1994.

[24] Sivan Toledo. A survey of out-of-core algorithms in numerical linear algebra. In James Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms and Visualization*, pages 161–180. American Mathematical Society Press, Providence, RI, 1999.

[25] Sivan Toledo and Fred G. Gustavson. The design and implementation of solar, a portable library for scalable out-of-core linear algebra computations. In *IOPADS '96: Proceedings of the fourth workshop on I/O in parallel and distributed systems*, pages 28–40. ACM Press, 1996.