Optimal Evaluation of Fortran-90 Array Expressions for Distributed Memory Machines

S.D. Kaushik, S.K.S. Gupta, C.-H. Huang, P. Sadayappan

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Abstract

The owner-computes strategy has been used for evaluation of Fortran-90 array expressions on distributed memory machines. This strategy simplifies code generation but is often expensive in terms of the total communication cost and size of temporary memory required for its implementation. In this paper, we propose the relaxing of the owner computes strategy, to reduce the total communication and temporary storage cost. We develop cost metrics for measuring the communication and memory cost associated with the evaluation of Fortran-90 array expressions on distributed memory machines. The *communication tree* is introduced as a useful representation for array expressions involving associative and commutative operators of one kind. Procedures for estimating communication and temporary memory costs for a communication tree are described. An efficient polynomial-time algorithm to determine an evaluation order which minimizes the communication cost is presented. We also present an efficient polynomial-time algorithm to determine the evaluation order which minimizes the memory cost involved in the evaluation of such expressions.

Keywords : Data distributions, Fortran-90, Array expressions, Arborescence, Binary expression trees.

1 Introduction

Distributed memory machines have shown potential to be the high performance computers of the next generation. Although several commercial distributed memory computers have been introduced over the past few years, use of these machines has been restricted. While distributed memory computers offer excellent potential for high performance, programming these machines is difficult, time-consuming and error-prone, due to the lack of effective programming and debugging tools. An approach gaining popularity is the use of a language model that provides a global address space along with mechanisms for explicit specification of data distributions for partitioning globally shared arrays. Definition of a language - High Performance Fortran (HPF) - with provisions for specification of data distribution directives is currently in progress. A compiler for such a language, bears the responsibility of generating the required communication commands to transfer data between the local memories of processors at appropriate points in the execution of the generated parallel program.

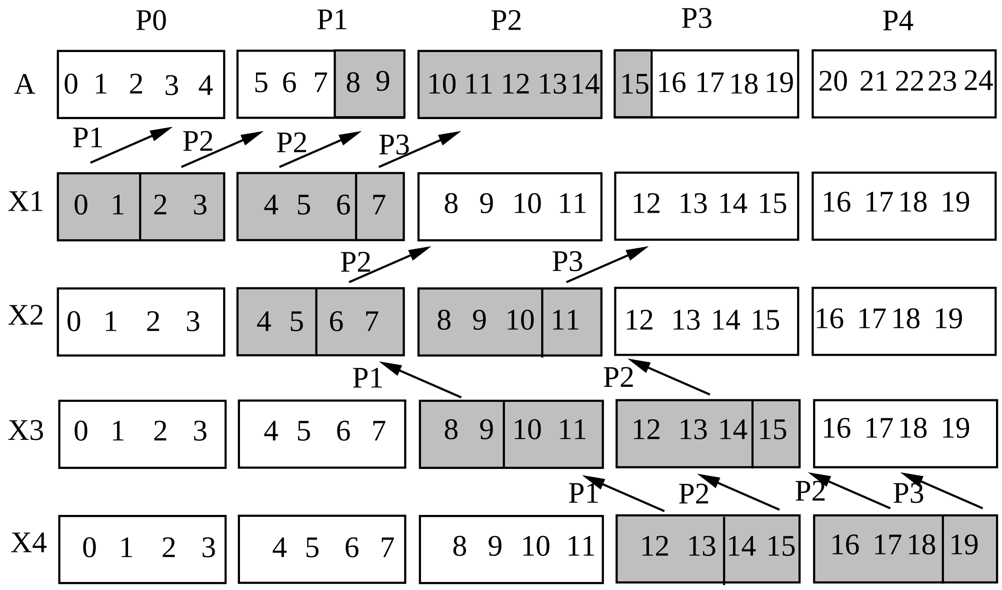

Figure 1: Execution of Fortran-90 array statement on a distributed memory machine. $A(8:15) = a_1 * X1(0:7) + a_2 * X2(4:11) + a_3 * X3(8:15) + a_4 * X4(12:19)$

This programming model has been incorporated in array-based languages like Fortran-90 [12, 20]. Extensions for Fortran-90 such as Fortran-90D [1], Fortran-90 Yale Extension [4] and Distributed Fortran-90 [19] have been proposed. In Fortran-90D, parallelism is represented largely in terms of parallel constructs such as array operations, forall loops and intrinsic functions.

Current compilers generate data parallel code based on the owner-computes philosophy [3, 7, 10, 11]. Under the owner computes strategy, the owner of a data element, i.e the processor on which the data element is located, performs the computation that assigns a value to that data element. The compiler inserts the necessary communication commands to obtain all the non-local data elements required to evaluate the value of the local data elements. For example, consider the Fortran-90 array statement.

$$A(8:15) = a_1 * X1(0:7) + a_2 * X2(4:11) + a_3 * X3(8:15) + a_4 * X4(12:19)$$

Arrays $A(0:25), X1(0:19), X2(0:19), X3(0:19)$ and $X4(0:19)$ are all aligned and distributed using a block distribution on 5 processors. The scalars a_1, a_2, a_3 and a_4 are replicated on all the processors. Fig. 1 illustrates the array elements involved in the assignment statement. Under the owner computes strategy each processor will obtain the necessary non-local values of $X1, X2, X3$ and $X4$. The communication performed to obtain non-local values is indicated by arrows with the destination processor number. For instance, processor $P1$ will obtain the values of $X1(0:1), X2(4:5), X3(8:9)$ and $X4(12:13)$. Using these values processor $P1$ evaluates the new value for $A(8:9)$. The operands of the addition operations are redistributed to form temporaries with the data distribution of $A(8:15)$. The partial results also have the distribution of $A(8:15)$. For instance the partial result $a_1 * X1(0:7) + a_2 * X2(4:11)$ will have the same data distribution as $A(8:15)$. We refer to A as the *owner* of the operation $+$. This means that the operands and the result of the operation $+$ have the data distribution of the array section A. The communication cost of the data redistribution is determined by the maximum number of messages a processor needs to send. The redistribution of $X1(0:7)$ to $A(8:15)$ has a communication cost of 2 as both processors $P0$ and $P1$ need to send 2 messages. The communication costs for redistribution of the array sections $X2, X3$ and $X4$ to the distribution of A are $1, 1$ and 2 respectively. Since the redistribution of each array section is performed sequentially, the total communication cost for the owner computes strategy is 6. Additional temporary storage would be needed at each processor to store the redistributed array sections and partial results. Processor $P2$ will require 10 units of temporary storage - 5 units to create local copies of the redistributed operands and 5 units to store the partial result.

In the evaluation of array expressions, the ease of code generation, the total communication cost and the total temporary storage cost can be used as goodness measures. The owner-computes philosophy simplifies the task of code generation for the compiler. However, the strategy is often expensive in terms of the total communication cost involved and size of temporary memory required. In the above array statement, if the owner computes strategy were relaxed and different "owners" chosen for the intermediate operations, then the total communication cost and temporary memory required can be reduced. The array statement can be executed as follows to yield a lower communication and temporary memory cost. The array sections $X1, X3$ and $X4$ are redistributed to the distribution of array section $X2$. Each redistribution has a communication cost of 1 unit. The expression $a_1 * X1(0:7) + a_2 * X2(4:11) + a_3 * X3(8:15) + a_4 * X4(12:19)$ is evaluated with $X2$ as the owner of all the addition operations. Finally the result which has the distribution of $X2$ is redistributed to the distribution of $A[8:15]$ and the values assigned to appropriate elements of $A[8:15]$. The total communication cost for the evaluation is 4 units and the maximum amount of temporary memory required during the evaluation is 8 units.

The general problem is as follows - Given n array sections of data $L, R_1, \ldots, R_n$, all of the same shape, we wish to evaluate an arithmetic expression $\mathcal{E}$ involving these arrays and arbitrary binary operators.

$$\mathcal{E} \quad : \quad L = R_1 \ op \ \ldots \ op \ R_n$$

The problem is to determine "owners" (among $L, R_1, \ldots, R_n$) of the individual operations in the expression. The objective is to minimize the total communication and temporary memory cost for the evaluation of the expression.

Efficient solutions to this problem will provide substantial practical benefits. The applicability of the solutions depends to great extent on the use in practice of array expressions with several operands. Stencil based methods for solving differential equations have expressions of more than 4 array operands with an associative and commutative operator of one kind. The number of operands depends on the size of the stencil. 7-point and 9-point stencils are often used. Image processing software also contains array expressions with several operands. Besides, combining array expressions within basic blocks would yield expressions involving a large number of array operands. Even if the arrays are properly aligned to minimize communication cost, the array sections accessed will have different distributions. Thus there exists a large potential for applying optimizations in the evaluation of array expressions on distributed memory machines.

Work in the literature has been restricted to optimizing the evaluation of array expressions on vector machines and massively parallel SIMD architectures such as the Connection Machine, AMT DAP. Mace and Wagner [17, 18] determine globally optimal storage patterns for efficient access during vector operations. Knobe, Lukas and Steele [14, 15] have addressed the expression evaluation problem for data parallel SIMD architectures. Techniques for automatic layout of arrays to reduce the communication are discussed. Gilbert and Schreiber [8] address the issue of evaluating expressions involving arrays of the same shape located at various positions in a SIMD multi-computer connected in a regular fashion. Using a metric that describes the cost of moving arrays, they determine the intermediate positions at which to carry out the individual operations. They provide an efficient algorithm to find the minimum cost evaluation procedure for an arbitrary expression for "robust metrics". Li and Chen [16] address the problem of aligning arrays on a data parallel architecture to minimize communication. Chen and Wu [4] describe a general algebraic compiler optimization technique that reduces communication overhead for Fortran-90 implementations on massively parallel machines. However, the evaluation on MIMD architectures, of array expressions consisting of array sections distributed using regular data distributions, has not been addressed in the literature.

In this paper, we focus on array expressions involving *an associative and commutative operator of one kind.* We provide an efficient algorithm to find the owners (among the operand array sections) of intermediate operations such that the resulting evaluation procedure, allowing associative and commutative rearrangement, has minimum communication cost. We also provide an efficient algorithm to determine the evaluation order and owners of intermediate operations such that the temporary memory cost is minimized.

The paper is organized as follows. Section 2 describes in detail the execution of the Fortran-90 array statement. Metrics for the estimation of communication and memory cost associated with a single data redistribution are developed. In Section 3, the *communication tree* is introduced as a useful representation for expression evaluation on distributed memory machines. A communication cost metric for the communication tree is developed. A procedure for evaluation the memory cost for the communication tree and determination of evaluation order is provided. In Section 4, we describe the algorithm for determining the communication tree which minimizes the communication cost for an array expression. In Section 5, we describe the algorithm for determining the communication tree which minimizes the memory cost for an array expression. Conclusions and future directions are given in Section 6.

2 Expression Evaluation Cost Metrics

In this section, we describe the Fortran-90 array statement and the associated communication and temporary memory allocation required for its implementation. We develop communication cost and temporary memory cost metrics for the execution such statements. Consider two arrays $A(0:N_t-1)$ and $B(0:N_f-1)$ distributed over P_t and P_f processors respectively. The regular data distributions used are the block distribution, cyclic distribution and the block-cyclic distribution [7, 10]. Fig. 2 illustrates all the three types of data distributions for an array of size 16 distributed over 4 processors. The simplest form of the array statement is

$$A(l_1 : u_1 : s_1) \quad = \quad B(l_2 : u_2 : s_2) \tag{1}$$

On a distributed memory machine, all the elements involved in the computation of an element $A(i)$ may not be on the same processor and execution of the statement requires communication. The owner-computes rule is used - the processor that owns the element of array A performs the

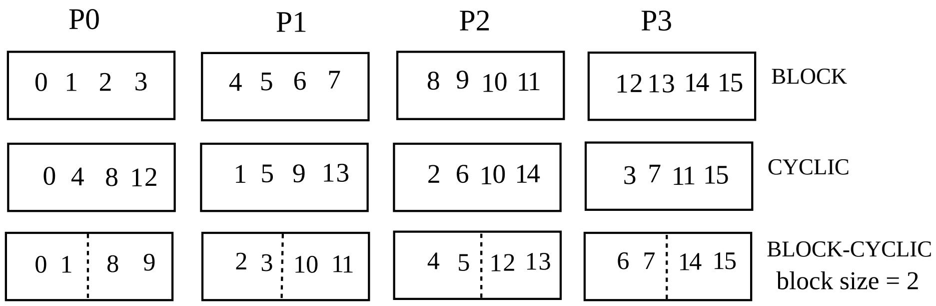

Figure 2: Data Distribution for Single Dimension Arrays

necessary computation to evaluate its value. For example, consider two arrays $A(0:24)$ and $B(0:24)$ distributed on 5 processors both using a block distribution and the execution of the array statement (Fig. 3(a)).

$$A(1:5) = B(4:20:4)$$

A is referred to as the target array and B as the source array. A is referred to as the owner of the assignment operation. Corresponding elements are linked by arrows. Some of the arrows cross processor boundaries and require data communication. For instance, processors $P1, P2$ and $P3$ communicate with processor $P0$ and processor $P4$ with processor $P1$. Fig 3(b) illustrates the allocated temporary memory to store redistributed values of $B(4:20:4)$. Processor $P0$ requires 4 units of temporary memory and processor $P1$ requires 1 unit. Processors $P2, P3$ and $P4$ don't require any additional temporary memory. Note that the value of $B(4)$ is replicated on processor $P0$. This simplifies the generated code as all required values can be accessed in a regular fashion. Thus the amount of temporary storage required at a processor depends on the number of data elements of the target array section located on that processor. We now develop the cost metrics for the simple array statement in Eq. 1.

Communication Cost

The typical communication model used in the evaluation of communication cost is as follows. The cost of communicating a message of size n is

$$t_c(n) = t_s + n * t_e + t_d + n * t_p$$

- t_s - is the startup cost for message transmission.
- t_e - is the transmission cost for a single data element.
- t_d - is a function of the distance to be traversed and depends on the network topology .
- t_p - is the time required for local data movement/ permutation involved while transferring data from the user space to the system space for message transmission.

We wish to establish a cost metric which is independent of the interconnection topology of the underlying architecture. Typically, for distributed memory machines the startup cost, t_s is the dominating factor in the communication cost. Hence the number of messages sent is used as a

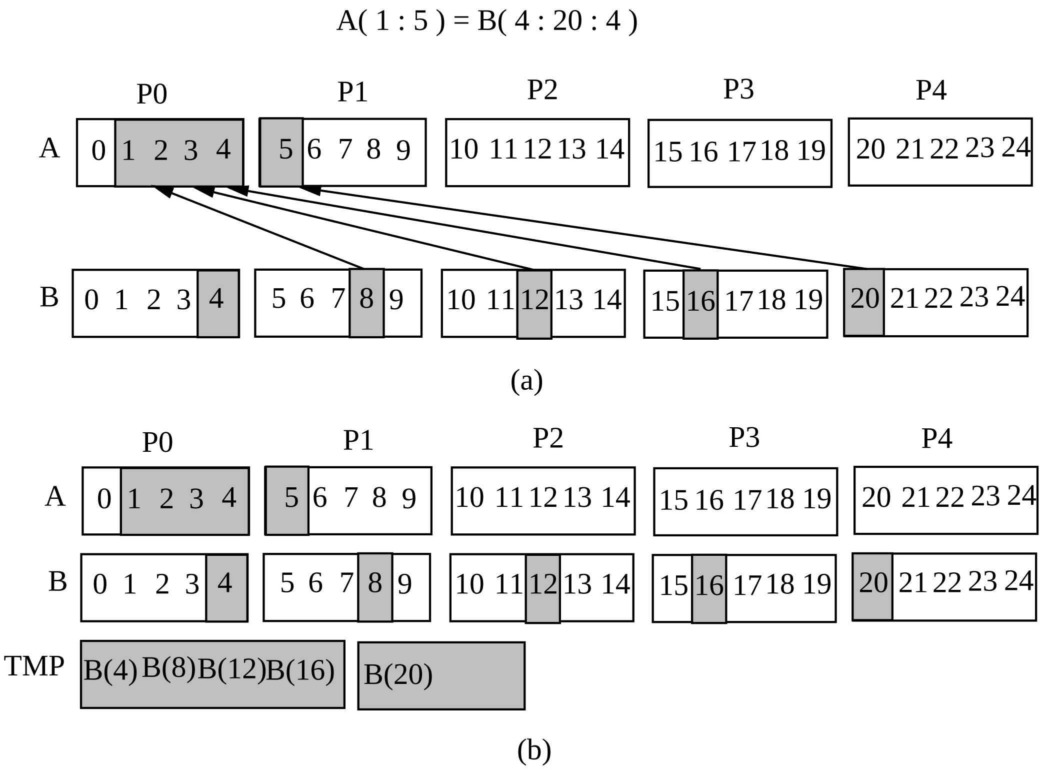

Figure 3: Fortran-90 Array Statement $A(1:5) = B(4:20:4)$(a) Communication Performed (b) Temporary Memory required.

quantitative measure to evaluate communication cost. The maximum number of communication messages sent by a single processor among all the processors involved in the data redistribution is used as the cost measure. This communication cost assumes the interconnection topology to be ideal. The processor with the largest number of sends will incur the largest delay in the redistribution. However, the algorithms presented in the paper are independent of the communication measures chosen.

Definition 2.1 (Data Distribution of Array Sections) *Let $SECT(N)$ represent the set of sections of arrays of size N. $SECT(N) = [0 : N-1] \times [0 : N-1] \times [0 : N-1]$. Let $DIST(P)$ represents the set of regular data distributions over P processors.*

- $DIST(P) = \{Block(P), Cyclic(P), Block-Cyclic(P,B)\}$ *where*
  - $Block(P)$: *represents the block distribution on P processors.*
  - $Cyclic(P)$: *represents a cyclic distribution on P processors.*
  - $Block-cyclic(P,B)$: *represents a block-cyclic distribution on P processors with block size B.*

Then the set of all distributed array sections is given by $ARRSECT(N,P) = DIST(P) \times SECT(N)$

Definition 2.2 (Communication Cost) *Let the target array $A[0 : N_t - 1]$ be distributed over P_t processors using a distribution D_t^A and let the source array $B[0 : N_f - 1]$ be distributed over P_f processors using a distribution D_f^B. Let $ARR^A = (D_t^A, (l_1 : u_1 : s_1))$ and $ARR^B = (D_f^B, (l_2 : u_2 : s_2))$ be the corresponding distributed array sections. Let $sends : P_f \times ARRSECT(N_f, P_f) \times ARRSECT(N_t, P_t) \rightarrow \mathcal{N}$ be defined by ($\mathcal{N}$ is the set of positive integers)*

- $sends(p_i, ARR^A, ARR^B)$: *is the number of messages sent from processor p_i, $0 \leq p_i \leq P_f$ to perform the data redistribution.*

The communication cost function $CC : ARRSECT(N_f, P_f) \times ARRSECT(N_t, P_t) \rightarrow \mathcal{N}$ is

$$CC(ARR^B, ARR^A) \quad = \quad max_{p_i} \left(sends(p_i, ARR^B, ARR_A) \right) \quad 0 \leq p_i \leq P_f - 1$$

In Fig. 3, the communication cost is $CC(ARR^B, ARR^A) = 1$ as all the processors send at most 1 message. The communication cost function is not commutative, i.e.

$$\exists \quad ARR_1, ARR_2 \;\; s.t. \;\; CC(ARR_1, ARR_2) \neq CC(ARR_2, ARR_1)$$

For example, in Fig. 3, $CC(ARR^B, ARR^A) = 1$ and $CC(ARR^A, ARR^B) = 3$. The communication cost function satisfies the triangle inequality, i.e.

$$CC(A_1, A_2) + CC(A_2, A_3) \geq CC(A_1, A_3) \quad \forall \quad A_1, A_2, A_3 \in ARRSECT(\mathcal{N}, \mathcal{N})$$

This follows from the fact that CC represents the cost of the best possible means of redistributing data from distribution ARR_1 to distribution ARR_3. If $CC(ARR_1, ARR_2) + CC(ARR_2, ARR_3) <$

$CC(ARR_1, ARR_3)$ were to hold then clearly distribution ARR_2 would be used as an intermediate distribution while redistributing from ARR_1 to ARR_3.

In [9] closed form expressions for the exact data sets (sets of data values to be communicated) and processor sets (sets of processors to communicate to) for the Fortran-90 array statement have been derived. Using these expressions, closed form expressions for the communication cost for all possible regular distribution pairs can be developed and extended to multidimensional arrays. These communication costs can be evaluated at compile time.

Memory Cost

A processor requires a number of temporary memory locations equal to the number of data elements of the target array section located on that processor (if any redistribution is required at all). Local values required in the array statement are replicated in the temporary memory to simplify the generated code. The temporary memory cost for the data redistribution corresponds to the maximum size of temporary storage required by a processor involved in the data redistribution.

Definition 2.3 (Memory Cost) *Let the target array $A[0 : N_t - 1]$ be distributed over P_t processors using a distribution D_t^A and let the source array $B[0 : N_f - 1]$ be distributed over P_f processors using a distribution D_f^B. Let $ARR^A = (D_t^A, (l_1 : u_1 : s_1))$ and $ARR^B = (D_f^B, (l_2 : u_2 : s_2))$ be the corresponding distributed array sections. Let $iters : P_t \times ARRSECT(N_f, P_f) \times ARRSECT(N_t, P_t) \rightarrow \mathcal{N}$ be*

- *$iters(p_i, ARR^B, ARR^A)$: is the amount of temporary storage required on processor p_i, $0 \leq p_i \leq P_t - 1$ to store the redistributed values of $B[l_2 : u_2 : s_2]$.*

The memory cost function $\mathcal{MC} : ARRSECT(N_f, P_f) \times ARRSECT(N_t, P_t) \rightarrow \mathcal{N}$ is defined as follows.

$$\mathcal{MC}(ARR^B, ARR^A) \quad = \quad max_{p_i}\left(iters(p_i, ARR^B, ARR^A)\right) \qquad 0 \leq p_i \leq P_t - 1$$

It is important to note that the memory cost is independent of the source array section (unless the source and target array sections are identically distributed in which case no temporary memory is required). If the source array section differs from the target array section, then the portion of the source array section required locally is created in temporary memory. This may lead to duplication of some elements already present locally on the processor, but it simplifies the generated code. Since the memory cost depends only on the target array section, $\mathcal{MC}(ARR^X, ARR^A)$ is the same for all source array sections ARR^X (unless $ARR^X = ARR^A$). The memory cost will be henceforth be referred to as $\mathcal{MC}(ARR^A)$.

The memory cost function is not commutative, i.e.

$$\exists ARR_1, ARR_2 \quad s.t. \;\; \mathcal{MC}(ARR_1, ARR_2) \neq \mathcal{MC}(ARR_2, ARR_1)$$

For example, in Fig. 3 $\mathcal{MC}(ARR^B, ARR^A) = 4$ and $\mathcal{MC}(ARR^A, ARR^B) = 1$ The definition of the memory cost function as the maximum number of elements of the target array section on a

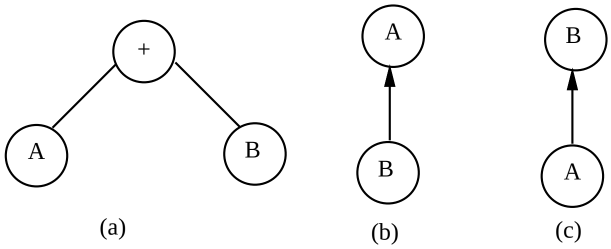

Figure 4: (a) Expression Tree for $A + B$. (b) Communication Tree when A is the owner of operation $+$. (c) Communication Tree when B is the owner of operation $+$.

processor guarantees that the triangle inequality is satisfied, i.e.

$$\mathcal{MC}(A_1, A_2) + \mathcal{MC}(A_2, A_3) \geq \mathcal{MC}(A_1, A_3)) \quad \forall \quad A_1, A_2, A_3 \in ARRSECT(\mathcal{N}, \mathcal{N})$$

Similar to the evaluation of the communication cost, closed form expressions for the memory cost for all possible pairs of regular data distributions can be derived and extended to multi-dimensional arrays.

3 Cost Evaluation for Array Expressions

In this section, we develop a representation to characterize the mapping and scheduling of expression evaluation on a distributed memory machine which includes information about the owner of each computation. This representation should be useful in the evaluation of the communication and temporary memory cost associated with the evaluation. No common subexpressions are allowed, i.e. all the subexpressions are assumed to be independent. This guarantees that the expression is represented as a binary expression tree rather than a directed acyclic graph. However, the binary expression tree is not sufficient for representing expression evaluation on a distributed memory machine. Each operator would have to be assigned an owner. Scalars and constants do not affect the communication cost and need not be represented.

Communication Trees

Communication trees are used for representing the evaluation of array expressions on distributed memory machines. The result of an operation has got the same distribution as the owner of the operation. Consider the evaluation of $A + B$ where A and B are array sections with different data distributions. If A is the owner of the operation $+$ then the result $A + B$ will have the same distribution as A. Similarly if B were the owner of the operation then $A + B$ would have the same distribution as B. Hence, it makes sense to associate the operation $+$ and the result $A + B$ with the owner of the operation. If each array section is represented by a node then the operation $A + B$ could be represented as illustrated in Fig. 4(b) and (c). Fig. 4(b) corresponds to the case when A is the owner of the operation. Node A represents the operation $+$ and also the result $A + B$ of the operation. Fig. 4(c) corresponds to the case when B is the owner of the operation. The formal definition of a communication tree requires specification of some graph structures.

Definition 3.1 (Arborescence) *A directed graph G is called an arborescence if*

- *G contains no circuit - neither directed nor semicircuit.*
- *In G there is precisely one vertex R of zero in-degree. This vertex R is the root of the arborescence.*

A *spanning arborescence* in a connected digraph is a spanning tree that is an arborescence. An arborescence has the following properties [6].

- An arborescence is a directed tree in which every vertex other that the root has an in-degree of exactly one.
- In an arborescence, there is a directed path from the root R to every other vertex. Conversely a circuitless digraph G is an arborescence if there is a vertex R in G such that every vertex is accessible from R and R is not accessible from any other vertex.

An arborescence is traditionally called an *out-tree.* Reversing the direction of the edges gives an *in-tree.* The in-tree has got similar properties as the out-tree. We formally define the communication tree for an array expression $\mathcal{E}$, consisting of an associative and commutative operator *op* where

$$\mathcal{E} : L = R_1 \ \ op \ \ \ldots \ \ op \ \ R_n$$

Definition 3.2 (Communication Tree) *A communication tree $T = (V, E)$ for the expression $\mathcal{E}$ is an in-tree where $V = \{V_L\} \cup V_{RHS}$. V_L, the node representing the target array section L, is the root of the in-tree and $V_{RHS} = \{V_{R_1}, \ldots V_{R_n}\}$. Each V_{R_i}, $\ 1 \leq i \leq n$ represents a source array section.*

Every vertex $v \in V$ represents an array section of the array expression. In addition, every internal node represents partial results of computation. An edge $e = (v_i, v_j)$ signifies redistribution of data corresponding to the of array section v_i to the data distribution of v_j and an operation owned by the array section v_j. Thus an internal node v_j with unit indegree due to an edge (v_j, v_i) represents an operation v_i *op* v_j or v_j *op* v_i. An internal node v with $indegree(v) = k$ due to directed edges $(v_1, v), \ldots, (v_k, v)$ corresponds to k operations involving operands $v, v_1, \ldots v_k$. The order in which these operations are performed doesn't affect the communication cost and is not represented in the communication tree. The root of the communication tree corresponds to the target array section. Along with other operations corresponding to incoming edges, the root implicitly represents an assignment operation to the target array section. Since every vertex of the communication tree has an outdegree of exactly one, no source array section or partial result is redistributed to two owners. In the in-tree, there exists a directed path from every vertex to the root. This implies that values of every array section reach the target array section, either directly (single edge path) or indirectly (through intermediate nodes as a partial result).

The communication tree contains information about the owner of each intermediate operation. It complements the expression tree for expression evaluation on a distributed memory machine. The communication tree is sufficient for evaluating the communication and memory cost. The expression

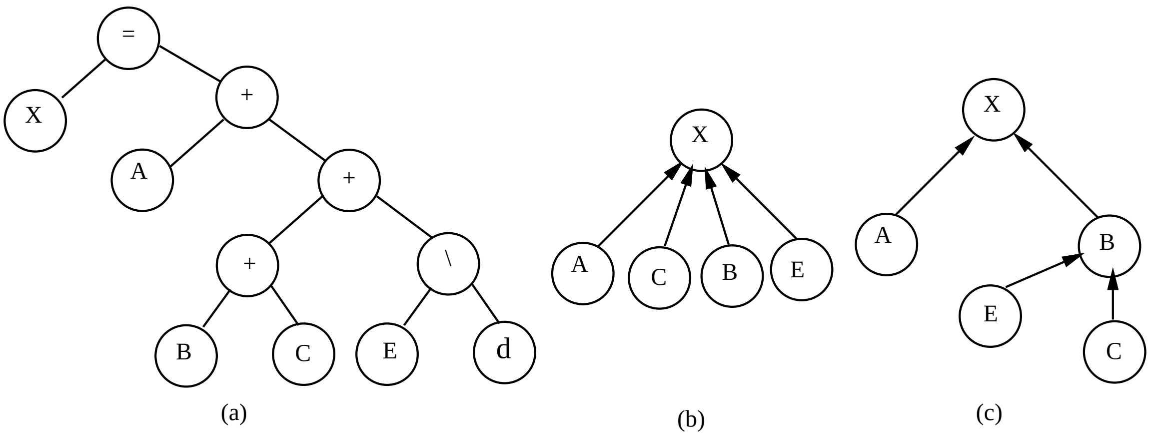

Figure 5: (a) An Expression Tree for $X[0:5] = A[1:6] + B[0:5] + C[2:7] + E[0:5]/d$ (d is a scalar). (b) Communication Tree corresponding owner computes strategy. (c) Communication Tree with owner computes strategy relaxed.

tree represents the restrictions on the order of evaluation. Fig. 5 illustrates an expression tree and some possible communication trees. Fig. 5(b) corresponds to the standard owner computes strategy. Fig. 5(c) corresponds to an evaluation order in which B is the owner of $B+C$ and $B+C+E/d$ and X is the owner of $A+B+C+E/d$. If an expression contains operators with different precedences then a particular order must be imposed on the evaluation of the expression and some communication trees may not be valid. However, this problem does not arise is expressions containing an associative and commutative operator of one kind. Henceforth the communication tree will be used to represent the evaluation of an expression on distributed memory machine. A reference to the name of an array section will mean the distribution of the array section, i.e. $CC(V_i, V_j)$ would refer to the communication cost for data redistribution from the data distribution of array section V_i to the data distribution of array section V_j.

3.1 Communication Cost

Each edge in the communication tree represents data communication in which potentially all the processors are involved. Hence at any instance, only the communication corresponds to a single edge can be performed. Therefore the communication cost for each edge in the tree should be summed to give the total communication cost for the communication tree.

Definition 3.3 (Communication Cost) *Let τ be the set of communication trees representing the evaluation of the expression $\mathcal{E}$. Let $Dist : V \rightarrow ARRSECT(\mathcal{N}, \mathcal{N})$ give the data distributions of the source and target array sections. The communication cost for a communication tree $T = (V, E) \in \tau$ is given by $\mathcal{C} : \tau \rightarrow \mathcal{N}$ where*

$$\mathcal{C}(T) \quad = \sum_{(V_x, V_y) \in E} CC(Dist(V_x), Dist(V_y))$$

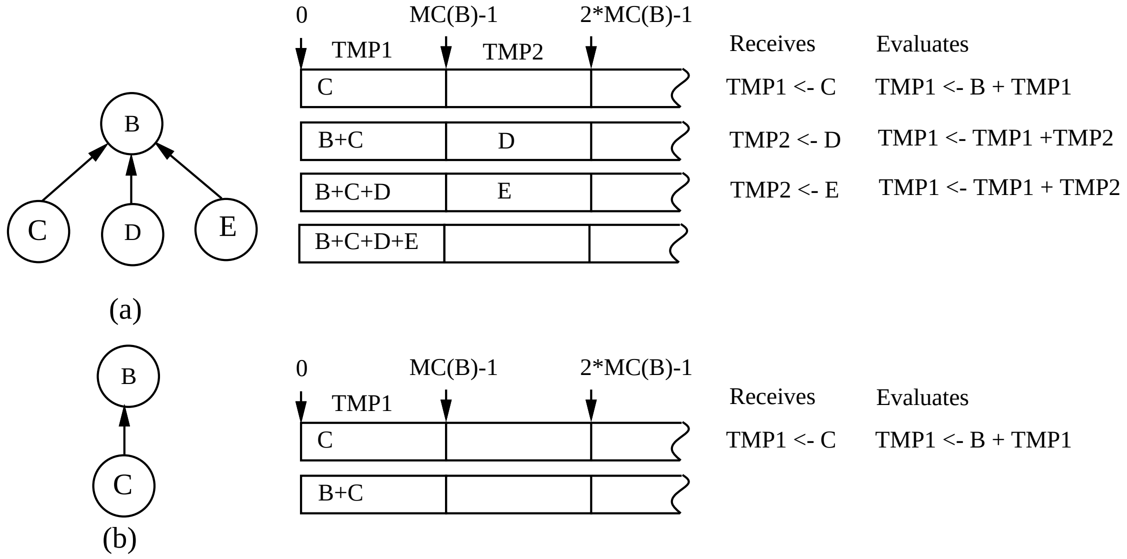

Figure 6: Memory requirement for evaluation of an internal node v in the communication tree (a) $indegree(v) > 1$ (b)$indegree(v) = 1$

The fact that communication cost for the communication tree is the sum of the communication costs of all the edges allows for an efficient algorithm to find an evaluation order with optimal cost. It is important to note that the above cost evaluation is exact and involves no approximations.

3.2 Memory Cost

Consider an internal node v of the communication tree, with $indegree(v) > 1$ representing $n+1$ associative and commutative operations. These operations can be performed using two temporary arrays each of size $\mathcal{MC}(v)$. One temporary array is used to accumulate the result of the operation while the other temporary is used to receive the next array section. If the node had $indegree(v) = 1$, then a single temporary array of size $\mathcal{MC}(v)$ would be required to evaluate the operation at v. For example, consider the evaluation of the node B in Fig. 6(a). The operation represented by the node B can be performed using temporary memory of size $2 * \mathcal{MC}(B)$. The contents of temporary memory are illustrated at various points in the evaluation of the node. At any instant, at most 2 temporary arrays, each of size $\mathcal{MC}(B)$ are required. Similarly, if the node B had indegree 1 then it can be evaluated using temporary memory of size $\mathcal{MC}(B)$. This is illustrated in Fig. 6(b).

Estimation of temporary memory size required for the evaluation of the entire communication tree is more complicated. The maximum amount of temporary memory required for the evaluation of a node does not depend only on the node being evaluated. Partial results from other computations need to be stored in temporary memory too. The order of evaluation will affect the maximum size of temporary memory required for the evaluation. For example, consider the communication tree in Fig. 7(a). The numbers of the left of each node represent the memory cost of the node. If the left subtree of the node X were evaluated before the right subtree then the maximum amount of temporary memory required at any instant during the evaluation of the tree would be 11 units. Evaluation of $A + B$ would require 3 units. This partial result is redistributed to the distribution

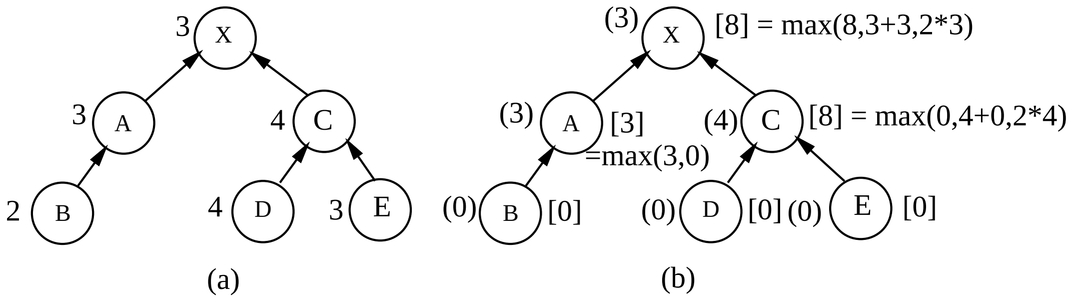

Figure 7: Evaluation of Memory cost for a communication tree (a) Communication Tree with Memory costs (b) Labeling algorithm

of X and occupies 3 units of memory. Now evaluation of the node C requires $2 * 4 = 8$ units of memory. Thus at this instant $3 + 8 = 11$ units of temporary memory are required. On the other hand, if the right subtree were evaluated first then the evaluation of the entire tree can be performed using at most 8 units of temporary memory.

The total temporary memory cost for the communication tree can be evaluated as follows. The communication tree is labeled by visiting nodes in a bottom-up fashion so that a node is not visited until all of its children are labeled. The label of a node denotes the minimum size of temporary storage, required to evaluate the tree rooted at the node. The algorithm is as follows.

If n is a leaf then
    A $label(n) = 0;$ $\quad result(n) = 0;$
else
begin /* n is an internal node */
    B $result(n) = \mathcal{MC}(n);$
    let $n_1, \ldots, n_k$ be the children of n ordered by *label*
    so that $label(n_1) \geq label(n_2) \geq \cdots \geq label(n_k)$
    if $k = 1$ then /* n has got only one child */
        C $label(n) = max(result(n), label(n_1));$
    else /* n has more than one child */
        D $label(n) = max(label(n_1), label(n_2) + result(n), 2 * result(n));$
end

Definition 3.4 (Memory Cost) *The memory cost of the communication tree $T = (V, E)$ with root $v_{root} \in V$ is given by $\mathcal{M}(T) = label(v_{root})$.*

Using the labeling algorithm, we also obtain an evaluation order for the communication tree which requires the minimum memory cost. Intuitively this corresponds to evaluating the subtree rooted at the child with the largest *label* first. If two children of the node have the same *label*, (i.e. the evaluation of the subtrees rooted at both nodes requires same amount of memory) then either subtree can be evaluated first. The algorithm is similar to the labeling algorithm used to find the minimum number of registers required to evaluate an arithmetic expression represented by a binary expression tree [2]. Some results follow directly from the algorithm.

Lemma 3.1 *Let $T = (V, E)$ be a communication tree representing the expression $\mathcal{E}$. If $v \in V$ is an internal node with children $v_i \in V,\ \ 1 \leq i \leq k$ then $label(v) \geq label(v_i),\ \forall\ \ 1 \leq i \leq k$.*

Lemma 3.2 *Let $T = (V, E)$ be a communication tree representing the expression $\mathcal{E}$. For every internal node $v \in V$, with children $v_i \in V,\ \ 1 \leq i \leq k$ such that $k \geq 2$, $label(v) \geq 2 * \mathcal{MC}(v)$.*

Theorem 3.1 *Let $T = (V, E)$ be a communication tree representing the expression $\mathcal{E}$.*

(i) If $\exists v_i \in V$ s.t. $indegree(v_i) \geq 2$ then the memory cost for the communication tree $\mathcal{M}(T) \geq 2 * \mathcal{MC}(v_i)$.

(ii) If $\forall v_i \in V,\ \ indegree(v_i) \leq 1,$ then the memory cost for the communication tree $\mathcal{M}(T) \geq max\,(\mathcal{MC}(v_i))\quad v_i \in V$. (The communication tree is a linear chain).

Proof: Both (i) and (ii) follow from Lemma. 3.1, Lemma. 3.2, Def. 3.4 and by noting that every node in the communication tree is a descendant of the root. □.

Corollary 3.1 *Let $T = (V, E)$ be a communication tree with root V_{root} representing the expression $\mathcal{E}$. Then the memory cost of the tree $\mathcal{M}(T) \geq \mathcal{MC}(V_{root})$.*

Proof: V_{root} has at least a unit indegree. By Theorem. 3.1, $\mathcal{M}(T) \geq max(\mathcal{MC}(V_{root}), \mathcal{MC}(V_i), V_i \in V)$, i.e. $\mathcal{M}(T) \geq \mathcal{MC}(V_{root})$ □.

The memory cost evaluated represents the upper bound on the actual temporary storage that would be used. To evaluate the actual memory cost one would have to also take into account the processor which has the largest data points for each node of the communication tree

4 Communication Optimal Evaluation of Expressions with an Associative and Commutative Operator of one kind

In this section, we focus on minimizing the communication cost for the evaluation of the array expression $\mathcal{E}$.

$$\mathcal{E} \quad : \quad L = R_1 \ op\ \cdots\ op\ R_n$$

where *op* is an associative and commutative operator and each $R_i, 0 \leq i \leq n$ is an array section. We construct an intermediate structure called the *communication graph* to obtain a communication tree with minimum communication cost.

Definition 4.1 (Communication Graph) *A communication graph $G = (V, E)$ for the array expression $\mathcal{E}$ is a weighted directed graph where $V = \{V_L\} \cup V_{RHS}$. V_L is a node representing the target array section and $V_{RHS} = \{V_{R_1}, \ldots V_{R_n}\}$. Each V_{R_i} represents a source array section. $E = (V_{RHS} \times V_{RHS}) \cup E'$ where $E' = \{(V_i, V_L) | V_i \in V_{RHS}\}$. A weight function $w : E \rightarrow \mathcal{N}$ assigns each directed edge $e = (V_i, V_j) \in E$ a weight $w(e) = \mathcal{CC}(V_i, V_j)$.*

The weights represent the communication cost for the redistribution between the two array sections. For the communication graph G, representing $\mathcal{E}$, $|V| = n+1$ and $|E| = n^2 + n$. Every expression $\mathcal{E}$ will have a unique communication graph G. The communication graph forms a useful structure because of the following property.

Theorem 4.1 *Let $G = (V, E)$ be the communication graph for the expression $\mathcal{E}$. Every spanning in-tree of G is a communication tree representing $\mathcal{E}$.*

Proof: Consider a spanning in-tree of the communication graph G. Due to the construction of G, V_L is accessible from any other node V_{R_i} while no node V_{R_i} can be reached from V_L. Thus V_L is the root of the spanning in-tree. The vertex set of the in-tree includes all the array sections for the expression $\mathcal{E}$ and the target array section is the root. Hence by Def. 3.2 the spanning in-tree is a communication tree for the expression $\mathcal{E}$. □

The converse also holds.

Theorem 4.2 *Let $G = (V_G, E_G)$ be the communication graph for the expression $\mathcal{E}$. Every communication tree for $\mathcal{E}$ is a spanning in-tree of G.*

Proof : Let $G = (V_G, E_G)$ be the communication graph for $\mathcal{E}$. Consider a communication tree $T = (V_T, E_T)$ for $\mathcal{E}$. The construction of G implies that $V_G = V_T$ (both G and T represent the same expression $\mathcal{E}$) and $E_T \subset E_G$. Noting that (i) T is an in-tree (ii) every edge in E_T is in E_G and (iii) $V_G = V_T$ it follows that T is a spanning in-tree of G. □.

Clearly the communication tree corresponding to the conventional owner computes strategy is one of the spanning in-trees of the communication graph.

Corollary 4.1 *Let $G = (V, E)$ be the communication graph for the expression $\mathcal{E}$. A minimum spanning in-tree of G is the communication tree for $\mathcal{E}$ with minimum communication cost.*

Proof: Let T be a minimum spanning in-tree of G. By Theorem. 4.1 T is a communication tree for expression $\mathcal{E}$. Assume existence of a communication tree T' for $\mathcal{E}$ with communication cost such that $\mathcal{C}(T) \geq \mathcal{C}(T')$. By Theorem. 4.2 the communication tree T' is a spanning in-tree of G. Hence T' is a spanning in-tree for G which possibly has a lower cost than the minimum spanning in-tree for G. Hence either $\mathcal{C}(T) = \mathcal{C}(T')$ and T is a communication tree with minimum communication cost or we have a contradiction. □.

There exist several efficient algorithms for finding the minimum spanning arborescences of a directed graph [5, 6, 21, 13]. The complexity for a dense graph with n vertices is $\mathcal{O}(n^2)$ [21]. Given an expression $\mathcal{E}$ we construct the communication graph in $\mathcal{O}(n^2)$ steps and the communication tree with minimum communication cost can be found in $\mathcal{O}(n^2)$ steps.

For example consider the array statement in Section 1

$$A(8:15) = a_1 * X1(0:7) + a_2 * X2(4:11) + a_3 * X3(8:15) + a_4 * X4(12:19)$$

Fig. 8 shows the steps for obtaining the communication tree with minimum communication cost. Fig. 8(a) illustrates the communication graph and Fig. 8(b) illustrates the minimum spanning tree.

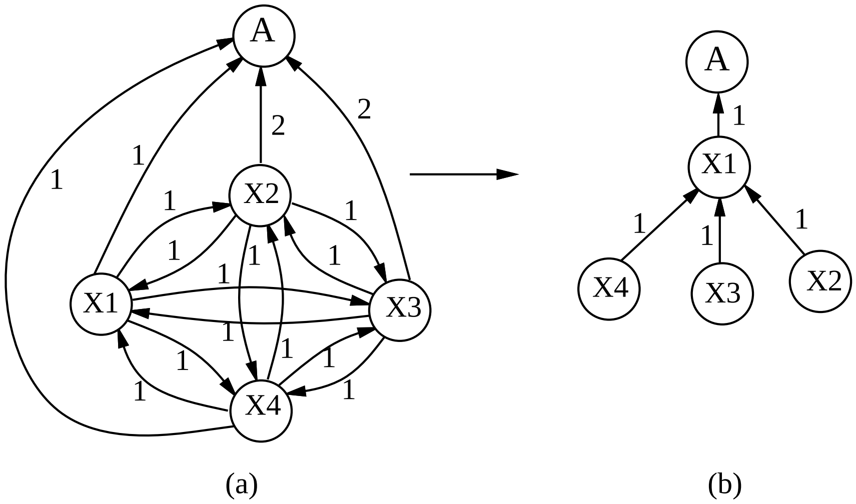

Figure 8: Finding the minimum communication cost communication tree for $A(8:15) = a_1 * X1(0:7) + a_2 * X2(4:11) + a_3 * X3(8:15) + a_4 * X4(12:19)$ (a) Communication graph (b) Minimum spanning in-tree

The conventional owner computes strategy has a communication cost of 6. The communication tree in Fig. 8(b) has a communication cost of 4.

5 Memory Optimal Evaluation of Expressions with an Associative and Commutative Operator of one kind

In this section, we focus on minimizing the temporary memory cost involved in the evaluation of array expression consisting of $n+1$ array sections $L, R_1, \ldots R_n$, and an associative and commutative operator.

$$\mathcal{E} : L = R_1 \;\; op \ldots op \;\; R_n$$

Let the nodes $V_1, \ldots V_n$ correspond to the source array sections $R_1, \ldots R_n$(not in order) such that $\mathcal{MC}(V_1) \leq \mathcal{MC}(V_2) \leq \ldots \leq \mathcal{MC}_{n-1}(V) \leq \mathcal{MC}(V_n)$. Let V_{root} be the node corresponding to the target array section L.

Theorem 5.1 *One of the three communication trees in Fig. 9 is the minimum memory cost communication tree for expression $\mathcal{E}$ with $n \geq 3$ operands.*

Proof: See Appendix. □.

The communication tree in Fig. 9(a) corresponds to the owner computes strategy and has got an exact memory cost of $2 * \mathcal{MC}(V_{root})$. The communication tree in Fig. 9(b) has got an exact memory cost of $max\ (\mathcal{MC}(V_{root}), 2 * \mathcal{MC}(V_1))$. The communication tree in Fig. 9(c) is a linear tree and has got a memory cost of $max(\mathcal{MC}(V_{root}), \mathcal{MC}(V_i), 0 \leq i < n) = max(\mathcal{MC}(V_{root}), \mathcal{MC}(V_{n-1}))$.

The algorithm for determining the communication tree with minimum memory cost requires $\mathcal{O}(n)$ steps. The steps are as follows

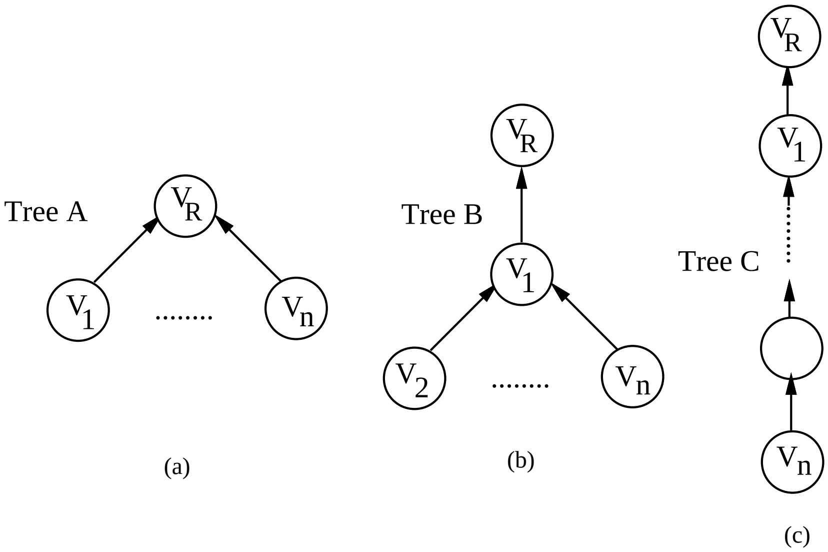

Figure 9: Trees with minimum memory cost

1. Determine the memory cost $\mathcal{MC}(V_i)$, $V_i \in V$ for the expression $\mathcal{E}$.

2. If $n < 3$ determine the cost of the 3 possible communication trees with 3 nodes. Select the tree with the minimum memory cost.

3. If $n \geq 3$ then find the node V_1 with the minimum memory cost $\mathcal{MC}(V_1)$ and the node V_{n-1} with the second largest memory cost $\mathcal{MC}(V_{n-1})$. Evaluate the cost of the three trees in Fig. 9 and find the tree with minimum memory cost.

6 Conclusions

This paper has addressed the issue of the relaxing the owner computes strategy for evaluation of Fortran-90 array expressions on distributed memory computers. Cost metrics for the measuring communication and memory cost associated with the evaluation of the array expression on distributed memory computers were developed. The communication tree was introduced as a useful representation for array expressions involving associative and commutative operators of one kind. Procedures for estimating the communication and memory costs for the communication tree were developed. A polynomial time algorithm to find the "owners" (among the operand array sections) of intermediate operations such that the resulting evaluation procedure, allowing associative and commutative rearrangement, has minimum communication cost was presented. An efficient algorithm to determine the evaluation order and owners of intermediate operations such that the temporary memory cost is minimized was also presented.

The algorithm presented finds the communication optimal evaluation order using "owners" from the operands of the array expression. The possibility of the existence of an intermediate distribution outside the operands distributions, which could lead to further cost reduction cannot be ruled out. Work on extending the above methodology to general expressions is currenly in progress.

References

[1] I. Ahmad, R. Bordawekar, Z. Bozkus, A. Choudhary, G. Fox, K. Parasuram, R. Ponnusamy, S. Ranka, and R. Thakur. Implementation and scalability of Fortan-90D intrinsic functions on distributed memory machines. Technical report, Northeast Parallel Architectures Centre, Jun. 1992.

[2] A. V. Aho, R. Sethi, and J.D. Ullman. *Compilers P rinciples, techniques and tools.* Addison-Wesley, Jun 1987.

[3] B. M. Chapman, P. Mehrotra, and H. P. Zima. Vienna Fortran – a Fortran language extension for distributed memory multiprocessors. In J. Saltz and P. Mehrotra, editors, *Language, Compilers and Runtime Environments for Distributed Memory Machines*, pages 39–62. North-Holland, Amsterdam, 1992.

[4] M. Chen and J.-J. Wu. Optimizing Fortran-90 programs for data motion on massively parallel systems. Technical Report YALEU/DCS/TR-882, Yale University, Dept. of Computer Science, Jan. 1992.

[5] Y.-J. Chu and T.-H. Liu. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396–1400, 1965.

[6] N. Deo. *Graph Theory with Applications to Engineering and Computer Science.* Prentice Hall of India, 1987.

[7] G. Fox, S. Hiranandani, K. Kennedy, C Koelbel, U. Kremer, C.-W. Tseng, and M-Y. Wu. Fortran-D Language Specification. Technical Report TR-91-170, Dept. of Computer Science, Rice University, Dec. 1991.

[8] J.R. Gilbert and R. Schreiber. Optimal expression evaluation on data parallel architectures. *Journal of Parallel and Distributed Computing*, 13:58–64, 1991.

[9] S.K. Gupta, S.D. Kaushik, S. Mufti, S. Sharma, C.-H. Huang, and P. Sadayappan. On the generation of efficient data communication for distributed memory machines. Technical Report OSU-CIS-RC-5/92-TR15, Dept. of Computer and Information Sciences, The Ohio State University, Jun. 1992.

[10] S. Hiranandani, K. Kennedy, and C.-W. Tseng. Compiler optimizations for Fortran-D on mimd distributed memory machines. In *Supercomputing '91*, pages 86–100, Nov. 1991.

[11] S. Hiranandani, K. Kennedy, and C.-W. Tseng. Compiler support for machine-independent parallel programming in Fortran D. In J. Saltz and P. Mehrotra, editors, *Language, Compilers and Runtime Environments for Distributed Memory Machines*, pages 139–176. North-Holland, Amsterdam, 1992.

[12] American National Standards Institute. *American Natioinal Standard for Information Systems Programming Language Fortran (Fortran 90)*, ANSI X3.198-1991 edition, Jan. 1990.

[13] R.M. Karp. A simple derivation of Edmonds' algorithm for optimum branchings. *Networks*, 1(3):265–272, 1971.

[14] K. Knobe, J. D. Lukas, and G.L. Steele Jr. Massively parallel data optimization. In *Proc. Frontiers '88 : The Second Symposium on the Frontiers of Massively Parallel Computation, George Mason University*, pages 551–558, Oct. 1988.

[15] K. Knobe, J.D. Lukas, and G.L. Steele Jr. Data optimization : Allocation of arrays to reduce communication on SIMD machines. *Journal of Parallel and Distributed Computing*, 8:102–118, 1990.

[16] J. Li and M. Chen. The data alignment phase in compiling programs for distributed memory machine. *Journal of Parallel and Distributed Computing*, 13(2):213–221, Oct. 1991.

[17] M. Mace. *Memory Storage Patterns in Parallel Processing.* Kluwer Academic Publishers, Boston, Massachusetts, 1987.

[18] M. Mace and R. A. Wagner. Globally optimum selection of memory storage patterns. In *International Conference of Parallel Processing*, pages 264–271, Jul. 1985.

[19] J. Merlin. Techniques for automatic parallelisation of 'Distributed Fortran 90'. Technical Report SNARC 92-02, Southampton Novel Architecture Research Centre, Jun. 1992.

[20] M. Metcalf and J. Reid. *Fortran 90 Explained.* Oxford University Press, 1990.

[21] R.E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–36, 1977.

7 Appendix

Theorem 7.1 *One of the three communication trees in Fig. 9 is the minimum memory cost communication tree for expression $\mathcal{E}$ with $n \geq 3$ operands.*

Proof: Let us first evaluate the temporary memory costs for each of the communication trees in Fig. 9 using the labeling algorithm. The tree in Fig. 9(a) corresponds to the owner computes strategy and has got a memory cost of $2 * \mathcal{MC}(V_{root})$. (We shall refer to this tree as Tree A). The communication tree in Fig. 9(b) has got a memory cost of $max\,(\mathcal{MC}(V_{root}), 2 * \mathcal{MC}(V_1))$. (We shall refer to this tree as Tree B). The tree in Fig. 9(c) is a linear tree and has got a memory cost of $max(\mathcal{MC}(V_{root}), \mathcal{MC}(V_i), 0 \leq i < n) = max(\mathcal{MC}(V_{root}), \mathcal{MC}(V_{n-1}))$. (We shall refer to this tree as Tree C). We consider the following cases

- $\underline{2 * \mathcal{MC}(V_{root}) \leq \mathcal{MC}(V_1)}$

  Tree A has got the minimum memory cost of $2 * \mathcal{MC}(V_{root})$. Since $\mathcal{MC}(V_1) \leq \mathcal{MC}(V_i), 1 < i \leq n$, we have $2 * \mathcal{MC}(V_{root}) \leq \mathcal{MC}(V_i), 1 \leq i \leq n$. This implies that $2 * \mathcal{MC}(V_{root}) \leq 2 * \mathcal{MC}(V_i), 1 \leq i \leq n$ and $2 * \mathcal{MC}(V_{root}) \leq max(\mathcal{MC}(V_i), 1 \leq i \leq n)$. Using Theorem. 3.1 the result follows. Any other tree will have a cost of at least $max(\mathcal{MC}(V_i), 1 \leq i \leq n)$ or $max(2 * \mathcal{MC}(V_i), 1 \leq i \leq n)$.

- $\underline{2 * \mathcal{MC}(V_1) > \mathcal{MC}(V_{n-1}) \& 2 * \mathcal{MC}(V_{root}) > \mathcal{MC}(V_1)}$

  $2 * \mathcal{MC}(V_1) > \mathcal{MC}(V_{n-1}) \& 2 * \mathcal{MC}(V_{root}) > \mathcal{MC}(V_1)$ implies that $2 * \mathcal{MC}(V_i) > \mathcal{MC}(V_{n-1})$ $\forall i,\ 1 \leq i < n-1$. Using Theorem. 3.1 any communication tree T with a node $V_k, 1 \leq k \leq$

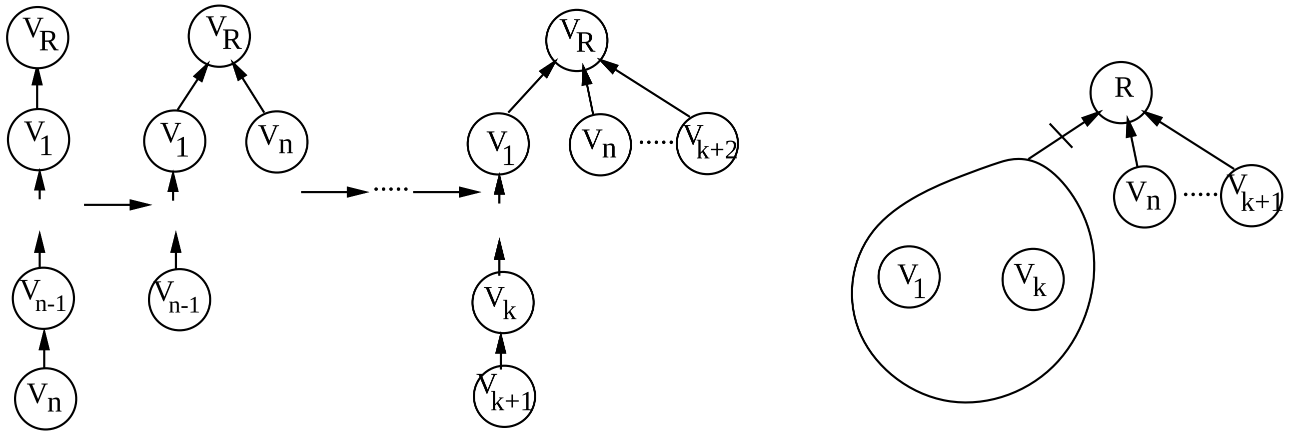

Figure 10: (a) Memory Cost reducing transformations (b) Family of trees with minimum memory cost

$n-1$ with $indegree(V_k) \geq 2$ has a cost $\mathcal{M}(T) \geq 2 * \mathcal{MC}(V_k) > \mathcal{MC}(V_{n-1})$. Hence in a communication tree with minimum memory cost each vertex $V_i, 1 \leq i \leq n-1$ should have $indegree(V_i) \leq 1$. We now consider two cases depending on the relative memory costs of the root node V_{root} and the node V_{n-1}.

- $\underline{2 * \mathcal{MC}(V_{root}) > \mathcal{MC}(V_{n-1})}$

  In this case communication Tree C has got the least memory cost, since any communication tree T with either the root or a node $V_i, 1 \leq i \leq n-1$ has indegree ≥ 2 will have cost greater than Tree C.

- $\underline{2 * \mathcal{MC}(V_{root}) \leq \mathcal{MC}(V_{n-1})}$

  $2 * \mathcal{MC}(V_{root}) > \mathcal{MC}(V_{n-1})$ implies that $\exists \mathcal{MC}(V_k) < 2 * \mathcal{MC}(V_{root}) < \mathcal{MC}(V_{k+1})$, $1 \leq k \leq n-2$

  Consider the series of trees in Fig. 10(a). The leftmost tree, Tree A has a cost of $\mathcal{MC}(V_{n-1})$ $(2 * \mathcal{MC}(V_{root}) \leq \mathcal{MC}(V_{n-1}))$ implies that $max(\mathcal{MC}(V_{root}), \mathcal{MC}(V_{n-1})) = \mathcal{MC}(V_{n-1})$. The transformation leads to a reduction in temporary memory cost. The final tree has a cost of $max(2 * \mathcal{MC}(V_{root}), \mathcal{MC}(V_k)) = 2 * \mathcal{MC}(V_{root})$. A family of trees with the same cost will be of the form shown in Fig. 10(b). In this family, vertices $V_1, \ldots, V_k$ have an indegree of ≤ 1. Vertices $V_{k+1}, \ldots, V_n$ have an indegree of 0. The root has indegree ≥ 2. Tree A is a member of this family. Any tree with a different form will have greater cost. If T is a communication tree with a vertex V_c, $k+1 \leq c \leq n$ is s.t. $indegree(V_c) \geq 1$ then $\mathcal{M}(T) \geq \mathcal{MC}(V_c) > 2 * \mathcal{MC}(V_{root})$. Similarly if T is a communication tree with a vertex V_c, $1 \leq c \leq n$ is s.t. $indegree(V_c) \geq 2$ then $\mathcal{M}(T) \geq 2 * \mathcal{MC}(V_c) > 2 * \mathcal{MC}(V_{root})$.

- $\underline{2 * \mathcal{MC}(V_1) \leq \mathcal{MC}(V_{n-1})}$

  $2 * \mathcal{MC}(V_1) \leq \mathcal{MC}(V_{n-1})$ implies that $\exists k, 1 \leq k \leq n-2$, s.t. $\mathcal{MC}(V_k) \leq 2 * \mathcal{MC}(V_1) < \mathcal{MC}(V_{k+1})$ Since $2 * \mathcal{MC}(V_1) < \mathcal{MC}(V_{k+1})$, and $\mathcal{MC}(V_i) \leq \mathcal{MC}(V_{i+1}) \quad \forall i, \quad 1 \leq i \leq n-1$ we have $2 * \mathcal{MC}(V_1) < \mathcal{MC}(V_i) \quad \forall i, \quad k+1 \leq i \leq n-1$. We consider the following cases

  - $\underline{2 * \mathcal{MC}(V_1) \leq \mathcal{MC}(V_{n-1}) \ \& \ \mathcal{MC}(V_{root}) \leq \mathcal{MC}(V_1) \leq 2 * \mathcal{MC}(V_{root}) < 2 * \mathcal{MC}(V_1)}$

    $\mathcal{MC}(V_{root}) \leq \mathcal{MC}(V_1)$ implies that $2 * \mathcal{MC}(V_{root}) \leq 2 * \mathcal{MC}(V_1)$. A family of trees in which vertices $V_1, \ldots V_k$ have indegree ≤ 1 and vertices $V_{k+1}, \ldots, V_n$ have indegree 0,

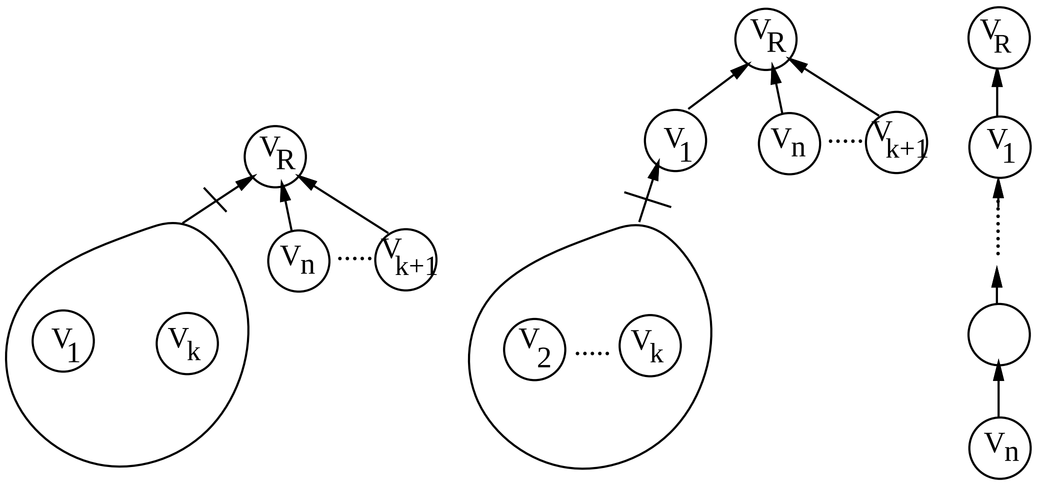

Figure 11:

and V_{root} has indegree ≥ 2 has the minimum cost of $2 * \mathcal{MC}(V_{root})$. (Fig. 11(a)). Any other tree T with a vertex $V_c, 1 \leq c \leq k$ s.t. $indegree(V_c) \geq 2$ will have a memory cost $\mathcal{M}(T) \geq 2 * \mathcal{MC}(V_c) \geq 2 * \mathcal{MC}(V_{root})$. Similarly any tree where any of the vertices $V_{k+1} \ldots V_n$ have indegree greater than 0 will have greater cost. Tree A is a member of this family. If $\mathcal{MC}(V_{root}) \leq \mathcal{MC}(V_1)$ then both V_{root} and V_1 can have indegree greater than 1.

- $\underline{2 * \mathcal{MC}(V_1) \leq \mathcal{MC}(V_{n-1}) \ \ \& \ \mathcal{MC}(V_1) < \mathcal{MC}(V_{root}) < 2 * \mathcal{MC}(V_1) \leq 2 * \mathcal{MC}(V_{root})}$
  A family of trees in which vertices $V_2, \ldots V_k$ have indegree ≤ 1, vertices $V_{k+1}, \ldots, V_n$ have indegree 0, V_1 has indegree ≥ 2 and V_{root} has unit indegree has the minimum cost of $2 * \mathcal{MC}(V_1)$.(Fig. 11(b))
- $\underline{2 * \mathcal{MC}(V_1) \leq \mathcal{MC}(V_{n-1}) \ \ \& \ \mathcal{MC}(V_{root}) > 2 * \mathcal{MC}(V_1)}$ Using Corollary 3.1, any tree T has a cost of at least $\mathcal{MC}(V_{root})$. The tree with exactly a memory cost of $\mathcal{MC}(V_{root})$ will have minimum memory cost. Tree B has such a cost and is thus a minimum memory cost tree. $\square$.

The proof also identifies the conditions for which each tree represents the tree with minimum cost. It should also be noted that there is a family of trees for some cases with minimum memory cost and the trees in Fig. 9 are one of the members of the appropriate family. Since the trees in Fig. 9 are not the unique trees with minimum cost the converse of Theorem. 5.1 does not hold. However a weaker form of converse holds and is as follows.

Theorem 7.2 *If $\mathcal{MC}(V_{root}) \leq 2 * \mathcal{MC}(V_1)$ then any communication tree for expression $\mathcal{E}$ with minimum memory cost belongs to one of the families of trees in Fig. 11.*