appeared in *Journal of Parallel and Distributed Computing*, vol. 21, no. 1, 1994, pp. 61-74

Data and Task Alignment in Distributed Memory Architectures*

Balaram Sinharoy
IBM Corporation
P. O. Box 950
Poughkeepsie, NY 12602

Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

Abstract

Different alignments of multi-dimensional arrays on a mesh-connected SIMD architecture result in different communication patterns during parallel program execution. In this paper a compile-time selection of data alignment that minimizes communication cost is discussed. First, it is shown that the selection is computationally as hard as a sub-class of the well-known problem of finding the closest vector in a lattice. The NP-hardness of the latter problem is proven. Then, two algorithms for exact minimum solution are discussed. Although the complexity of these algorithms is exponential, for small lattices often generated by parallel scientific computation the execution times of these algorithms may be acceptable. A polynomial-time algorithm for finding an approximate solution is also described. Finally, improvements in communication cost resulting from alignments for randomly generated graphs are presented.

Index Terms – Closest vectors in a lattice, communication cost, compiler optimization, complexity, data alignment, data distribution, SIMD architectures.

1 Introduction

A significant speedup can be achieved in a distributed memory parallel processing system by carefully distributing (or mapping) the data structures and the computation over the processors. In this paper we analyze the problem of data distribution over an array of processors. To distribute data structures optimally over an array of processors, one processor is allocated (at least conceptually) per data item of an array or other composite data structures. Operations involving two arrays can be performed entirely locally if the corresponding array elements are allocated to the same processor. Otherwise, a relatively expensive step, processor communication, takes place. The communication cost depends on the relative position of the two processors involved and the architecture under consideration. One of the major challenges in programming a distributed memory

*This work was partially supported by Office of Naval Research Grant N00014-93-1-0076 and by the National Science Foundation Grants CCR-8920694, and CDA-8805910. The content of this paper does not necessarily reflect the position or policy of the US Government – no official endorsement should be inferred or implied.

parallel computer is to determine the distribution of data structures among the processors that results in minimum communication cost.

For example, suppose arrays V1, V2, defined recursively by the equations in figure 1, are to

$$\begin{array}{ll} for \;\; i = 1, n & \\ \quad V1[i] = f(V2[i-2], V2[i-3]) & /*\ statement\ a_0\ */ \\ \quad V2[i] = g(V1[i+1], V1[i-1]) & /*\ statement\ a_1\ */ \end{array}$$

Figure 1: Variables $V1$ and $V2$ and statements a_0 and a_1 are to be aligned to minimize the communication cost.

be evaluated on an one-dimensional array of processors. Assume that consecutive elements of an array are mapped to adjacent processors. If $V2[i]$ is evaluated by the same processor that stores $V1[i+1]$, then the first argument of function "g" is conveniently located in the local memory. However the second argument has to be received from a processor two positions away thus incurring a communication cost of two units. The alignment problem is to determine the relative positions of the arrays $V1$ and $V2$ and the execution instances of the statements a_0 and a_1 that minimizes the communication cost.

In most mesh-connected architectures (such as Maspar and CM-2), the cost, C, of communicating a message of unit length between two processors, p_i and p_j, is proportional to the distance, d, between them[1]. Thus, $C(p_i, p_j) = \gamma * d(p_i, p_j)$, where γ denotes the time required to send a message of unit length between two directly connected processors. In an n-dimensional mesh connected architecture with four-nearest neighbor interconnection, the distance function is

$$d(< l_1, \ldots, l_n >, < m_1, \ldots, m_n >) = \sum_{i=1}^{n} |l_i - m_i|$$

where $|arg|$ denotes the absolute value of arg.

If the processors in the mesh-connected architecture communicate in a lock-step fashion (as in SIMD architectures), all the active processors simultaneously communicate the same amount of data in the same direction and to the same distance. For this type of traffic pattern, the distance metric, C, denotes the total time required by the active processors to communicate a message of unit length. The algorithms discussed in this paper determine the task and data alignment to minimize the total communication cost for such traffic patterns. For more general traffic patterns, the algorithms determine the alignment that minimizes the total distance travelled by the messages.

Let us consider a statement in the parallel program of the form:

$$a: \qquad v_l[i_1, i_2, \ldots, i_k] = \cdots\ v_r[e_1, e_2, \ldots, e_l]\ \cdots \tag{1}$$

where i_j are index variables and each subscripting expression e_j on the right hand side is a function over possibly many indexes. Compile-time analysis of the communication cost is independent of the language in which the program is written and can be performed in any traditional high-level programming languages or a parallel functional language [22]. What is important is the

[1]There is often a small "start up cost" associated with each communication, which is ignored in our analysis. In some architectures, there is a *global router*, communication through which is usually much costlier than communication through the mesh router, especially when the distance between the communicating processors is short.

form of subscripting expressions in variables present in the statement. In many parallel scientific computation, assignment statements, such as 1, are executed iteratively over the index domain. Thus there are potentially many instances of each statement at different index points which can be executed in parallel.

For each assignment statement "a", we define an alignment function $\boldsymbol{\alpha}_a$ that maps the index point representing an instance of the statement onto the position of the virtual processor domain that executes it (conceptually). Similarly, for each data structure v, we define an alignment function $\boldsymbol{\alpha}_v$. For the purpose of data alignment we assume that the number of dimensions of the virtual processor domain is equal to the largest dimension over all the data structures in the program. If the virtual processor domain and the actual processor domain have a different number of dimensions, then the compiler is expected to determine which dimension of variables and assignments to align with those of the processor architecture (thus the virtual processor domain is partitioned with planes orthogonal to some axes), after evaluating the communication cost of the best alignment for different selections of the projection dimensions to produce the most efficient code.

Without loss of generality, we can assume that the first n dimensions of variables v_l and v_r are aligned, where n is the number of dimensions of the processor array ($n \geq l, k$). Under these assumptions the communication cost of executing the statement is

$$C = \gamma * (d(\boldsymbol{\alpha}_l(\bar{i}), \boldsymbol{\alpha}_a(\bar{i})) + \sum_{r \in R_a} d(\boldsymbol{\alpha}_r(\bar{e}), \boldsymbol{\alpha}_a(\bar{i})))$$

where R_a is the multiset of all the variables that appear on the right hand side of statement "a" (the same variable may appear more than once), and l is the variable defined by the statement. To evaluate the distances in the above formula for n-dimensional mesh-connected architectures one need to know only the alignment in each dimension independently. The alignment function $\boldsymbol{\alpha}$ can be represented as a vector of elementary functions (for example, $\boldsymbol{\alpha}_l(\bar{i})$ is the tuple $< \alpha_{l1}(\bar{i}), \alpha_{l2}(\bar{i}), \ldots, \alpha_{ln}(\bar{i}) >$) and the resulting communication cost for the statement is

$$C = \gamma * (\sum_{j=1}^{n} |\alpha_{lj}(i_j) - \alpha_{aj}(i_j)| + \sum_{r \in R_a} \sum_{j=1}^{n} |\alpha_{rj}(e_j) - \alpha_{aj}(i_j)|)$$

A large class of parallel scientific computations can be modeled or expressed as Regular Iterative Algorithms (RIA) in which all subscripting expressions are of the form "$I - c$", where I is an iterative variable and c is an integer constant. Nested loop programs with regular data dependences, often referred to in the literature as *uniform dependence algorithms* [20] and the *uniform recurrence equations* considered in [9] can be converted easily to RIA. Many programs that implement discretized numerical solutions to differential and integral equations, digital signal processing problems, as well as linear algebraic and graph-theoretic problems can be conveniently written as RIAs [18]. A subclass of RIAs naturally leads to hardware systolic array implementations and conversely, any algorithm executed by a systolic array is a member of this subclass of RIAs [18]. The assignment statements for RIAs can be normalized to a form in which the left hand side variable is indexed by simple subscripts. In this case, for statement (1) we have

$$\forall j \leq n : e_j = i_j - c_{arj}$$

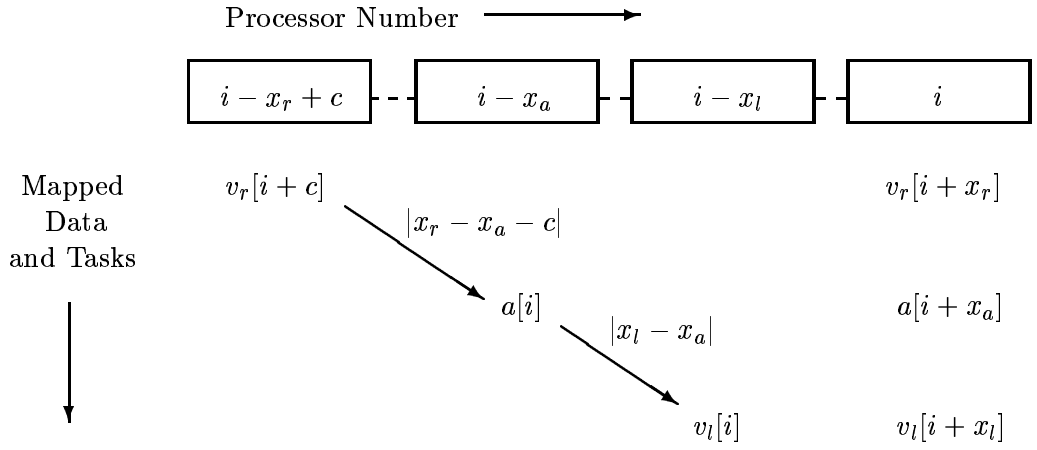

Figure 2: Inter-processor communication involved in executing the i-th instance of statement "a" in equation (2).

where c_{arj} is the constant that appears at the j-th dimension of the r-th variable in equation a. The statement can be written as

$$a: \qquad v_l[\ldots, i_j, \ldots] = \cdots \; v_r[\ldots, i_j - c_{arj}, \ldots] \; \cdots \tag{2}$$

Since the array references in RIA are uncoupled in each dimension, the alignment function that needs to be considered is a shift along a dimension. For example, the shift along the j-th dimension of variable l, α_{lj}, is $i_j + x_{lj}$ (for $x_{lj} \in \mathbb{Z}$). Thus the communication cost for statement a in equation (2) is:

$$C = \sum_{j=1}^{n} C_j, \qquad \text{where} \qquad C_j = |x_{lj} - x_{aj}| + \sum_{r \in R_a} |x_{rj} - x_{aj} - c_{arj}|$$

Figure 2 shows the mapping of the variables and the assertion when the number of dimension is 1.

Since $\forall j \leq n: \; C_j \geq 0$, cost for each dimension can be minimized separately. Thus, under the stated assumptions the alignment problem can be reformulated as n problems of finding a minimum of the cost function for the j-th dimension, $j = 1, 2, \ldots, n$:

$$C_j = \sum_{a \in RIA} \sum_{r \in R_a \cup \{l_a\}} |x_{rj} - x_{aj} - c_{arj}| \tag{3}$$

where RIA is the set of all assertions in the regular iterative algorithm and $c_{al_aj} = 0$ for each dimension j and assignment a. Equation (3) is non-negative and is defined by differences of pairs of variables that belong to two disjoint classes (x_r's and x_a's). Hence (3) has infinitely many minimum points. The next section illustrates how the exploitation of this fact can lead to a reduction in the number of variables in (3).

For the program in figure 1, the communication cost function is

$$C = |x_2 - x_0| + |x_3 - x_0 + 2| + |x_3 - x_0 + 3| + |x_3 - x_1| + |x_2 - x_1 - 1| + |x_2 - x_1 + 1|$$

where x_0, x_1 are shifts of statements 0 and 1 and x_2, x_3 denote shifts of arrays V1 and V2, respectively. Allocating the i-th instance of both assignments and arrays to the i-th processor (i.e. setting all shifts to zero) incurs the communication cost of seven units per iteration step.

If *owner-computes* rule [6] is followed to derive task mappings from data mappings (i.e., i-th instance of an equation is executed at the processor at which the i-th instance of the variable being defined resides), then $x_0 = x_2$ and $x_1 = x_3$ and the communication cost is:

$$C = |x_1 - x_0 + 2| + |x_1 - x_0 + 3| + |x_0 - x_1 - 1| + |x_0 - x_1 + 1|$$

In this case, the minimum communication cost achievable is five at the family of points $\{[t, t-1, t, t-1] \mid t \in \mathbb{Z}\}$ and $\{[t, t-2, t, t-2] \mid t \in \mathbb{Z}\}$. If however, only data is aligned, that is, i-th instance of an equation is always executed at the i-th processor, then $x_0 = x_1 = 0$ and C reduces to

$$C = |x_2| + |x_3 + 2| + |x_3 + 3| + |x_3| + |x_2 + 1| + |x_2 - 1|$$

which reaches the minimum value of five for $x_2 = 0,\ x_3 = -2$.

The original cost function reaches the minimum of four at two families of points: $\{[t, t-1, t, t-2] \mid t \in \mathbb{Z}\}$ and $\{[t, t-2, t-1, t-2] \mid t \in \mathbb{Z}\}$. The alignment of tasks and data for minimum communication cost is shown in figure 3 (the point from the second family of solutions with $t = 0$ is chosen) where the inter-processor communications are shown with solid arrows.

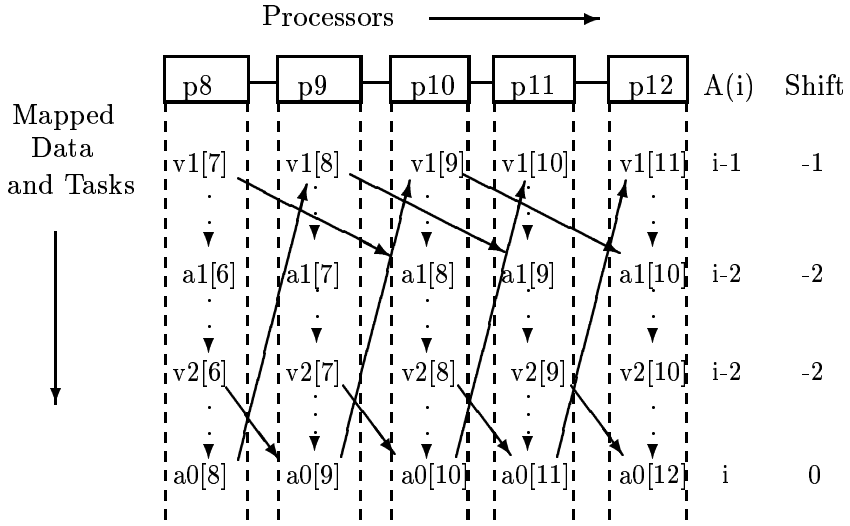

Figure 3: Local memory references (dotted arrows) and inter-processor communications (solid arrows) involved in executing the program in figure 1.

Data alignment has received a significant attention in recent years. However, most of the works are either limited to analyzing alignment for a single program statement or assumes owner-computes rule, which, as shown earlier, may not lead to the optimum alignment.

Gilbert and Schreiber [7] presented an algorithm for finding the minimum communication cost evaluation of expression over distributed processor arrays for architectures having a property called

"robustness". Robust architectures include hypercubes and the Connection Machine. Given the allocation of arrays on parallel architecture, Gilbert and Schreiber's algorithm determines the processors at which the temporary variables should reside and task evaluation should take place to minimize the communication cost. This algorithm applies, however, to one expression at a time, whereas the methods presented here are based on global program analysis and optimize data alignment for the entire parallel computation.

Knobe et al described a heuristic to determine the allocation preferences of each array occurrence from array definitions and usage patterns in the program [11] by building a *preference graph* incrementally. A cycle in the partially built preference graph indicates a potential conflict in alignment requirements which is resolved either by allowing communication or by serially allocating the dimension in conflict – thus reducing parallelism – as soon as it is detected. Their heuristic is also restricted by owner-computes rule. Unlike Knobe et.al.'s algorithm, the algorithm discussed in this paper analyzes the entire program before determining processor allocation and is not restricted by owner-computes rule.

Li and Chen [15] have described a heuristic based on component affinity graphs that determines which dimension of one array should be aligned with a particular dimension of another.

O'Boyle and Hedayat [1] discuss data alignment in a linear algebraic framework. Their condition of optimality takes into account only the number of non-local references made by each processor. Their algorithm selects the best alignment for each array independently of others. Hence, it does not address the problem that arises when these independently found alignments conflict with each other. In contrast, our approach measures communication cost in terms of the distance traveled by the message and the amount of data transferred.

Gupta [4] developed techniques based on the analysis of array references in various statements inside every loop in a Fortran 77 program for axis and stride alignment. His model is restricted by the owner-computes rule.

Although the results presented in this paper can be used in the compiler of any programming language, the methods developed have been implemented in the compiler of the parallel functional language, EPL [22]. In many equational languages, such as EPL, SISAL [21] and Crystal [2] the computation is specified by a set of recurrence equations, which defines a set of data structures. RIAs are therefore an easy and natural way of expressing parallel programs in such languages. In addition, thanks to the single assignment rule enforced by these languages, compiler analysis of the statements and their dependences are easier to perform than in traditional high-level languages [19].

The paper is organized as follows. In the following section a mathematical formulation of the alignment problem is given and its relation to a subclass of the closest vector problem is demonstrated. In section 3 this subclass of the closest vector problem is proven NP-hard. In section 4, two algorithms for an exact solution of the alignment problem are presented, although complexity of these algorithms is exponential, their performance is acceptable for small lattices. A polynomial-time algorithm for an approximate solution of the alignment problem and its performance is discussed in sections 5 and 6, respectively. Section 7 discusses how the result can be extended.

2 Mathematical Formulation

The alignment problem defined by formula (3) can be further simplified if some variables in the computation are independent of each other, as explained later. Consider graph $G = (V, E)$, where $V = \{v_1, \ldots, v_k\} \cup \{s_1, \ldots, s_m\}$ represents the data structures and assignment statements of an RIA and edges represent "appear in" relation between data structures and statements, $E = \{(s_j, v_i) : v_i \textit{ appears in } s_j\}$. Graph G, referred to as a *modified dependence graph* in this paper, is similar to the dependence graph of a program [12].

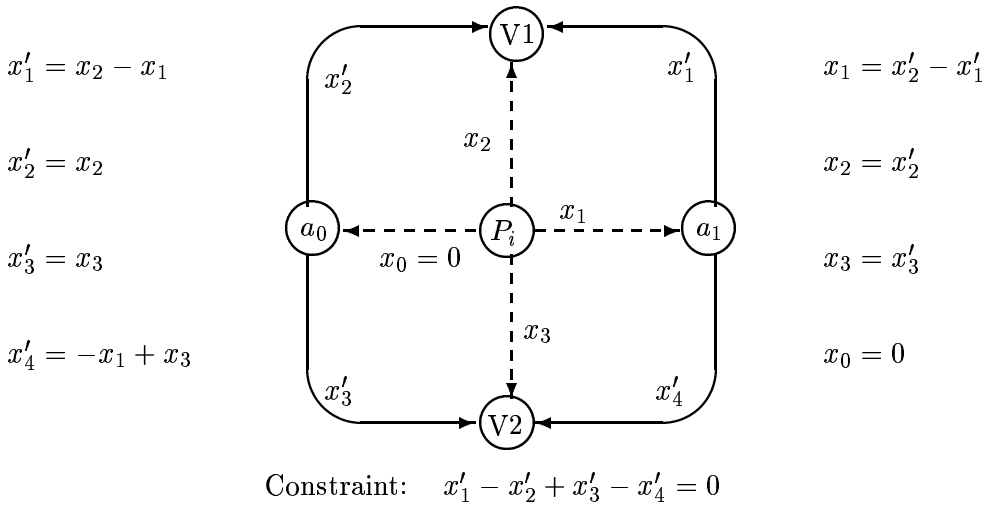

Figure 4: Relative and absolute alignments for the variables and statements in the program shown in figure 1.

Formula (3) uses absolute alignments (i.e., alignment with respect to the processor architecture) of data structures and assignment statements in the computation. Instead, alignments of every variable relative to the assignment statements in which it appears can be considered. Each edge (s_j, v_i) in the modified dependence graph can be labeled with a relative alignment $x'_k = x_{v_i} - x_{s_j}$. Figure 4 shows the modified dependence graph for the RIA in example 1 and the relationship of absolute alignments x_0, x_1, x_2, x_3 (represented by the dashed edges) to relative alignments x'_1, x'_2, x'_3, x'_4 (represented by the solid edges). Directions for the edges representing the relative alignments are from the statements to the variables. For example, variable $V1$ and statement a_1 has an absolute alignment of x_2 and x_1 with respect to the processor P_i, which means $V1$ is placed at processor P_{i+x_2} and a_1 is executed at processor P_{i+x_1}. Variable $V1$ has a relative alignment of x'_1 with respect to statement a_1, which means that if $a_1[i]$ is executed at processor P_{i+x_1} then $V1$ is stored at the local memory of the processor $P_{i+x_1+x'_1}$. Since $P_{i+x_1+x'_1}$ and P_{i+x_2} denote the same processor, we have $x'_1 = x_2 - x_1$, giving a relationship between the absolute and relative alignments.

The minimization problem (3) can be recast in terms of relative alignments x'_i, thereby simplifying the formula in (3). However, if there are cycles in the underlying graph of the modified dependence graph (i.e., the graph obtained from the modified dependence graph by replacing the directed edges with undirected edges) then variables $x'_1, x'_2, \ldots$ are not independent of each other.

Each cycle gives rise to a constraint of the form:

$$\sum_{i=1}^{|E|} a_i * x'_i = 0,\ a_i \in \{-1, 0, 1\} \tag{4}$$

where each a_i indicates whether the i-th edge belongs to the cycle, and if so whether the cycle traverses the corresponding edge along or against the direction of the edge. An example of such a constraint is shown in figure 4. Each constraint may be used to reduce the number of variables in the minimization problem.

The alignment problem can be generalized to allow the same label to appear on more than one edges. This can be useful in many situations. For example, the user may have some insight into the problem and to reduce the complexity of the compiler analysis phase, can provide alignment directives for some of the variables and statements. If the user provides the relative alignment between variable A and B, then the edge connecting A with node C in the modified dependency graph has the same variable in its label as the variable in the label on the edge connecting B with node C (the labels themselves may be of the form $x_i + c$, where c is a constant). Same variable may appear on more than one labels when array references more complicated than permitted by RIA is handled (see section 7) or when owner-compute rule is used.

Let $G' = (V', E')$ be a two-edge connected component of G with edges labeled $x_1, x_2, \ldots, x_n$. The subgraph G' gives rise to a minimization problem where the objective function is of the form

$$\sum_{i=1}^{n} \sum_{j=1}^{n_i} |x_i - c_{ij}| \tag{5}$$

where n_i is the number of times variable v_i appears in statement s_j, where the arc (s_j, v_i) is labeled with x_i[2]. This minimization problem can be solved separately from the rest of the graph G (and its remaining labels). If all two-edge connected components of modified dependence graph G consist of single nodes i.e., G is a forest, then lemma 1 (stated below) can be used to obtain the optimum alignment in $O(|V| * max|R_a|)$ where R_a denotes the set of array references that appear on the right hand side of statement "a".

Consider a spanning tree $T = (V', E_T)$ of the underlying graph of G'. Without loss of generality, we can assume that the edges of spanning tree T are labeled $\{x_1, x_2, \ldots, x_k\}$ where $k = |V'| - 1 (\leq |E'|)$. Each edge in $E' \setminus E_T$, when added to the spanning tree, produces a distinct cycle, traversal of which yields a constraint as in (4). Since all circuits in the connected component can be obtained from the cycles, the number of linearly independent constraints is

$$n = |E'| - |E_T| = |E'| - |V'| + 1$$

All these constraints can be written in the form of a dot product as

$$x_i = A_i \cdot X$$

where $i = k + 1, \ldots, n;\quad X = [x_1, x_2, \ldots, x_k]$, and $A_i \in \mathbb{Z}^k$.

[2]From now on we consider only relative alignments and denote them by $x_1, x_2, \ldots$.

If all edges of the spanning tree are labeled distinctly then $A_i \in \{-1, 0, 1\}^k$. In this case, each constraint can be used to replace a distinct variable in (3), thereby reducing the number of variables to $k = |V'| - 1$. Hence the alignment problem can be solved by finding a vector $[x_1, x_2, \ldots, x_k]$ that minimizes

$$\sum_{i=1}^{k} \sum_{j=1}^{n_i} |x_i - c_{ij}| + \sum_{i=1}^{m} |A_i \cdot X + b_i| \tag{6}$$

where $A_i \in \{-1, 0, 1\}^k$, $m = \sum_{i=k+1}^{n} n_i$ and $c_{ij}, b_i \in \mathbb{Z}$. As is explained lated the minimization problem is similar to finding the closest vector in a lattice.

Expression (6) can be written in the matrix notation as

$$|A \cdot X \ - \ C|$$

where A is of size $(\sum_i n_i + m) \times k$ and C is of size $(\sum_i n_i + m) \times 1$, such that each row of $A \cdot X - C$ is a distinct term of (6). Since there is at least one term $|x_i - c_{ij}|$ for each x_i, there are k rows in A that form the identity matrix of size $k \times k$. Thus the columns of A are linearly independent and $A \cdot X$ consequently is a lattice. The minimization problem expressed in equation 6 is therefore the closest vector problem.

In [23, 8], the closest vector problem is proven NP-hard for an arbitrary lattice in any norm. In [8], the three-dimensional matching problem [10] has been reduced to the closest vector problem. In the next section, the closest vector problem is proven to be NP-hard for the special type of lattice defined in (6) (in first norm). It should be noted however, that if all edges of graph G are labeled with distinct variables then A_i is in a proper subset of $\{-1, 0, 1\}^k$ and matrix A^T is known as *network matrix*. The complexity of the problem in this case is not known to the authors.

If there are no constraints (i.e, G has no cycles) then the problem reduces to finding such vector $X = [x_1, \ldots, x_k]$ that minimizes (5). Lemma 1 (also in [3]) suggests an efficient algorithm for finding such a vector.

Lemma 1 *x minimizes*

$$|x - c_0| + |x - c_1| + \cdots + |x - c_n|$$

where $c_0 \leq c_1 \leq \cdots \leq c_n$, if and only if $c_{\lceil \frac{(n-1)}{2} \rceil} \leq x \leq c_{\lceil \frac{n}{2} \rceil}$.

Proof The lemma is obviously true for $n = 1$ and $n = 2$. Assuming that the lemma is true for $n = k - 2$, $k \geq 4$, we prove that it is true for $n = k$. Consider

$$f = |x - c_0| + |x - c_1| + \cdots + |x - c_k|$$

$$g = |x - c_1| + |x - c_2| + \cdots + |x - c_{k-1}|$$

Let S be the set of x's which minimize g. By the inductive hypothesis $S = \{x : c_{\lceil \frac{k-1}{2} \rceil} \leq x \leq c_{\lceil \frac{k}{2} \rceil}\}$. The set of x's which minimize $|x - c_0| + |x - c_{k+1}|$ is $S' = \{x : c_0 \leq x \leq c_{k+1}\}$. Since $S \subseteq S'$, so the set of x's which minimize f is S. □

Lemma 1 can easily be generalized to theorem 1. Like lemma 1, theorem 1 can be applied in multi-dimensional cases as well when the variables are decoupled as in equation 5.

Theorem 1 *x minimizes the function*

$$m_0|x - c_0| + m_1|x - c_1| + \cdots + m_n|x - c_n|$$

where $m_i \in \mathbb{R}_{>0}$ for $i = 0, 1, \ldots, n$ and $c_0 \leq c_1 \cdots \leq c_n$, if and only if $c_l \leq x \leq c_u$, where $l = \max_k(\sum_{i=0}^{k} m_i \leq \frac{\sum_{i=0}^{n} m_i}{2})$ and $u = \min_k(\sum_{i=0}^{k} m_i \geq \frac{\sum_{i=0}^{n} m_i}{2})$.

3 The Minimization Problem is NP-hard

As demonstrated in the previous section, finding the optimum alignment necessitates finding a vector of integers $[x_1, x_2, \ldots, x_k] \in \mathbb{Z}^k$ that minimizes (6). In this section an NP-complete problem F is constructed. This problem is then reduced to the problem of minimizing (6).

Problem F: *Let*

$$f(Y) = D \cdot Y + \sum_{i=1}^{m} |A_i \cdot Y + b_i|$$

where $D \in (\mathbb{Z}_{>0} \cup 0)^k$, $-1 \leq b_i \leq 2k - 1$ and $A_i \in \{-1, 0, 1\}^k$ are given. Is there a vector $Y = [y_1, y_2, \ldots, y_k] \in \{0, 1\}^k$ such that for a given constant c,

$$f(Y) \leq c$$

Lemma 2 *Problem F is NP-complete.*

Proof Clearly the problem is in NP. We reduce the satisfiability problem to problem F. The satisfiability problem is to determine if there is a vector $X = [x_1, x_2, \ldots, x_k] \in \{0, 1\}^k$ for which a given Conjunctive Normal Form (CNF) is satisfiable, i.e.,

$$\prod_{i=1}^{m} \sum_{j=1}^{k} (h_{ij} x_j + h'_{ij} \bar{x_j}) = 1$$

where $h_{ij}, h'_{ij} \in \{0, 1\}$ and '+' stands for boolean 'or' and juxtaposition denotes boolean 'and'.

Replacing $\bar{x_j}$ by $(1 - x_j)$ and using arithmetics instead of boolean algebra, b'_i and $A'_i \in \{-1, 0, 1\}^k$ can be found such that for each i, $i \in \{1, \ldots, m\}$

$$-1 + \sum_{j=1}^{k} (h_{ij} x_j + h'_{ij} \bar{x_j}) = A'_i \cdot X + b'_i$$

where $X = [x_1, \ldots, x_k]$ and $-1 \leq b'_i \leq k - 1$. Hence the CNF is satisfiable if and only if there is a vector $X = [x_1, x_2, \ldots, x_k] \in \{0, 1\}^k$ which satisfies the following m inequalities:

$$A'_i \cdot X + b'_i \geq 0 \quad , \quad i \in \{1, \ldots, m\}$$

i.e., if and only if

$$\sum_{i=1}^{m} |A'_i \cdot X + b'_i| = \sum_{i=1}^{m} (A'_i \cdot X + b'_i)$$

i.e., if and only if

$$\sum_{i=1}^{m} |A'_i \cdot X + b'_i| \leq \sum_{i=1}^{m} (A'_i \cdot X + b'_i)$$

Let $\sum_{i=1}^{m}(A'_i \cdot X + b'_i) = \sum_{i=1}^{k} d_i x_i + e$. Replacing x_i by $1 - y_i$ for all i's for which $d_i > 0$, and by y_i otherwise, and transferring everything but the constants to the left we get, after simplification

$$D \cdot Y + \sum_{i=1}^{m} |A_i \cdot Y + b_i| \leq c$$

where $D \in (\mathbb{Z}_{>0} \cup 0)^k$, $-1 \leq b_i \leq 2k-1$, $A_i \in \{-1, 0, 1\}^k$ and $Y = (y_1, \ldots, y_k) \in \{0, 1\}^k$. Thus for every CNF we can construct an instance of problem F in polynomial time. Hence problem F is NP-complete. □

Theorem 2 *The problem of finding a vector of integers $X = [x_1, x_2, \ldots, x_k]$ that minimizes function g defined as*

$$g = \sum_{i=1}^{k} \sum_{j=1}^{n_i} |x_i - c_{ij}| + \sum_{i=1}^{m} |A_i \cdot X + b_i| \tag{7}$$

where[3] *$m \geq 0, n_i \geq 1, b_i \in \mathbb{Z}, c_{ij} \in \mathbb{Z}, A_i \in \{-1, 0, 1\}^k$ is NP-hard*[4].

Proof We reduce problem F to the minimization problem. Let

$$f(x_1, x_2, \ldots, x_k) = \sum_{i=1}^{k} d_i |x_i| + \sum_{i=1}^{m} |A_i \cdot X + b_i|$$

where $D = [d_1, d_2, \ldots, d_k] \in (\mathbb{Z}_{>0} \cup 0)^k$. The main difficulty in reducing problem F to the minimization problem is that problem F requires the solution to be in the domain $\{0, 1\}^k$, whereas the integral minimum point of the minimization problem may lie outside this domain. To guarantee the solution of the minimization problem to lie in the domain, we construct another function f' from f as follows. Let

$$f(0, 0, \ldots, 0) = p$$

and

$$f'(x_1, x_2, \ldots, x_k) = f(x_1, x_2, \ldots, x_k) + p \sum_{i=1}^{k} |x_i| + p \sum_{i=1}^{k} |x_i - 1| - kp$$

Clearly,

$$f'(x_1, x_2, \ldots, x_k) = f(x_1, x_2, \ldots, x_k)$$

[3]The theorem holds for n_i greater than any fixed integer.

[4]Following the standard terminology [16], we call a problem NP-hard if there is an NP-complete problem that can be polynomially reduced to it. Hence, an NP-hard problem is at least as difficult as any NP-complete problem and P=NP does not necessarily imply that an NP-hard problem can be solved in polynomial time.

for $0 \leq x_i \leq 1, i = 1, 2, \ldots, k$ and

$$f'(x_1, x_2, \ldots, x_k) \geq f(x_1, x_2, \ldots, x_k) + 2p > f'(0, 0, \ldots, 0)$$

for other integral values of x_i's. So the integral minimum point of f' must lie in the domain $0 \leq x_1 \leq 1, \ldots, 0 \leq x_k \leq 1$. Now $p = \sum_i |b_i| \leq m(2k-1)$, a polynomial in m and k and in the specified domain $|x_i| = x_i$. So problem F reduces in polynomial time to the problem of minimizing f', which is a subproblem of the problem of minimizing g, in which $c_{ij} = 0$ for all i, j. Hence the problem of determining integral minimum point of g is NP-hard. □

4 Finding the Exact Solution

Although the problem considered is NP-hard, both the number of variables and the magnitude of constants in the alignment problems created by many RIAs are small enough to make the algorithms for exact solution feasible.

4.1 Enumeration

It is not difficult to give an upper and lower bound for each of the variables wherein the minimum of (6) must lie. Let

$$p = \sum_{i=1}^{k} \sum_{j=1}^{n_i} |c_{ij}| + \sum_{i=1}^{m} |b_i| \ ,$$

$$l_i = \min_j (c_{ij}) \quad and \quad u_i = \max_j (c_{ij})$$

It is clear that, if for some i, $x_i < -p + l_i$ or $x_i > p + u_i$ then the value of the function to be minimized is greater than p. So for the minimum point $X_{min} = [x_{min,1}, \ldots, x_{min,k}]$ we have $-p + l_i \leq x_{min,i} \leq p + u_i$ for $i = 1, 2, \ldots, k$.

To obtain a tighter bound, let

$$s_i = \min_{x_i} \sum_{j=1}^{n_i} |x_i - c_{ij}|$$

Note that each s_i can be found using lemma 1. Let

$$q_i = p - \sum_{\substack{j \\ j \neq i}} s_j$$

We have at minimum point X_{min}

$$\sum_{j=1}^{n_i} |x_{min,i} - c_{ij}| + \sum_{\substack{j=1 \\ j \neq i}}^{k} s_j \leq p$$

So,

$$\sum_{j=1}^{n_i} |x_{min,i} - c_{ij}| \leq q_i$$

However, if $x_{min,i} \geq u_i$ then

$$\sum_{j=1}^{n_i} |x_{min,i} - c_{ij}| = n_i x_{min,i} - \sum_{j=1}^{n_i} c_{ij}$$

Hence,

$$x_{min,i} \leq max\left(u_i, \left\lfloor \frac{q_i + \sum_{j=1}^{n_i} c_{ij}}{n_i} \right\rfloor\right)$$

A lower bound can be obtained in a similar way. Thus at the minimum point

$$min\left(l_i, \left\lceil \frac{-q_i + \sum_{j=1}^{n_i} c_{ij}}{n_i} \right\rceil\right) \leq x_{min,i} \leq max\left(u_i, \left\lfloor \frac{q_i + \sum_{j=1}^{n_i} c_{ij}}{n_i} \right\rfloor\right) \quad , \quad 1 \leq i \leq k \tag{8}$$

By enumerating all points in the parallelohedra given by (8) we can find the minimum point(s). The complexity of this method is $O(c^k)$ where c is the largest range for any variable x_i in (8).

If the problem is formulated using absolute alignment (equation 3), enumeration need only be done along the co-ordinates that represent shifts of the statements. Given the value of these co-ordinates, the value of the other co-ordinates that result in the minimum value of equation (3) can be easily determined by applying theorem 1. Since the number of statements in an RIA is usually small, for small values of c (that is defined by the constants in the problem), the complexity of this algorithm may be acceptable.

4.2 Successive Replacement

The enumeration method discussed above has complexity $O(c^k)$ where c depends on the problem instance. In this section, it is shown that points which minimize the considered function form a convex set, and there is at least one minimum point that can be obtained by solving r ($\leq k$) linear equations generated by (6).

Consider $N = \sum_{i=1}^{k} n_j + m$ linear equations obtained from (6):

$$E_i \cdot X + e_i = 0 \tag{9}$$

such that if the i-th equation is satisfied, the i-th term in (6) becomes 0. For example, $E_i = [0, \ldots, 1, 0, \ldots, 0]$ for $i < N - m$ and $E_i = A_{N-i+m}$ otherwise. A rank of this set of linear equations is the maximum number of independent equations that can be selected from (9) and is equal to k. The i-th *hyperplane of (9)* is the set of points $\{X = [x_1, \ldots, x_k] | E_i \cdot X + e_i = 0\}$. A *straight line* in k dimensional space is defined here in the standard way as a set of points $X = [x_1, x_2, \ldots, x_k]$ that satisfy $k - 1$ independent linear equations with variables in X. The points on the line that yield the minimum of (6) are called the *local minimum* points with respect to this line.

Lemma 3 *Let l be a straight line in k dimensional space, then at least one of the local minimum points of line l lies on one of the hyperplanes of (9) (assuming that the line intersects at least one of the hyperplanes).*

Proof Let $p = [p_1, \ldots, p_k]$ and $q = [q_1, \ldots, q_k]$ be two distinct points on line l; we can assume without loss of generality that $p_1 < q_1$. Line l is therefore described by equations

$$x_i = \frac{p_i - q_i}{p_1 - q_1} x_1 + \frac{p_1 q_i - p_i q_1}{p_1 - q_1} \quad for \quad i = 2, 3, \ldots, k \tag{10}$$

Replacing each x_i $(i > 1)$ in (6) by the right hand side of equation (10) we get

$$\sum_{i=1}^{N} |a_i x_1 - e_i| \tag{11}$$

Since terms with $a_i = 0$ are constants, to find the minimum we need to consider only the following terms

$$\sum_{i,\ a_i \neq 0} |a_i||x_1 - e_i/a_i| \tag{12}$$

According to theorem 1, one local minimum point of (12) is defined by $x_1^{(0)} = e_i/a_i$, for some $i, (1 \leq i \leq N)$. Since this value of $x_1^{(0)}$ defines a point $X^{(0)}$ that satisfies equation $E_i \cdot X^{(0)} + e_i = 0$ then the local minimum point $X^{(0)}$ lies on the i-th hyperplane of (9). □

Theorem 3 *The set of points that minimizes (6) is convex and is defined by intersection of the halfspaces of (9).* [5]

Proof For all points, (6) is non-negative, therefore the set of points that minimizes (6) is nonempty. Obviously, the theorem holds if the set consists of a single point only. Otherwise, let $p = [p_1, \ldots, p_k]$ and $q = [q_1, \ldots, q_k]$ be two distinct minimum points such that $p_1 < q_1$. Consider the line in the k-dimensional space connecting p and q and therefore described by (10). The local minimum points on this line are the minimum points of (6). Replacing x_i $(i > 1)$ in (6) by the right hand side of (10) and applying theorem 1, we obtain the bounds $x_1^{(l)} < p_1 < q_1 < x_1^{(u)}$ for the local minimum point on this line. Thus x_1 is the minimum point of (6) if and only if $x_1^{(l)} \leq x_1 \leq x_1^{(u)}$ and there are such i, j that equations $E_i \cdot X^{(l)} + e_i = 0$ and $E_j \cdot X^{(u)} + e_j = 0$ hold. Let $s = [s_1, \ldots, s_k] = \lambda p + (1 - \lambda) q$ for some $0 < \lambda < 1$, be a point on the line segment between p and q. Since $x_1^{(l)} \leq p_1 < s_1 < q_1 \leq x_1^{(u)}$, s is also the minimum point of (6). Thus the minimum points form a convex set.

Since $x_1^{(l)}$ and $x_1^{(u)}$ lie on i-th and j-th hyperplane respectively, the line segment between them is a part of intersection of two (of the four) closed halfspaces defined by those hyperplanes. □

Theorem 4 *There is a minimum point of (6) that lies on k hyperplanes of (9), where k is the rank of (9). Moreover, if vectors $E_i \in \{-1, 0, 1\}^k$ in (9) are obtained from the minimization problem for a spanning tree with edges labeled distinctly (see section 2), then all coordinates of this minimum point are integers.*

[5] Each hyperplane of (6) defines two halfspaces, viz., $\{X : E_i \cdot X + e_i \leq 0\}$ and $\{X : E_i \cdot X + e_i \geq 0\}$.

Proof According to lemma 3, at least one of the minimum points lies on at least one of the hyperplanes of (9). Let the hyperplane defined by equation $E_{i1} \cdot X + e_{i1} = 0$ contains the minimum point. This equation can be used to replace one of the variables, say x_j, in (6) to obtain a new objective function $f_2(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k)$. Since f_2 is of the same form as (6), the same argument can be used to obtain another hyperplane defined, say, by $E'_{i2} \cdot X' + e'_{i2} = 0$ of f_2 on which the minimum lies, where $X' = [x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k]$. If $|E_{i2} \cdot X + e_{i2}|$ is the original term in (6) from which $|E'_{i2} \cdot X' + e'_{i2}|$ in f_2 is obtained by using $E_{i1} \cdot X + e_{i1} = 0$, then at least one of the minimum points lies on the intersection of the following two hyperplanes of (9)

$$E_{i1} \cdot X + e_{i1} = 0$$
$$E_{i2} \cdot X + e_{i2} = 0$$

By repeating this argument $k-1$ times, the first part of the theorem is proven.

If vectors E_i's satisfy the second condition of the theorem, then it can be shown[6] that the process of successive replacement mentioned above keeps coefficients of x_i's in the terms of the functions $f, f_2, \ldots$ in $\{-1, 0, 1\}$. Thus there is an integer solution to those k equations. □

It is clear that for the algorithm suggested by 4, the number of steps is less than $\binom{n}{k}$, hence its complexity can be stated as $O(2^n)$ or $O(n^k)$. Depending on both the number of hyperplanes and values of the constants in (6) this algorithm may be more or less effective than the enumeration method. Generally, to be effective, both algorithms require that the two-edge connected components in the modified dependence graph (see section 2) contain a small number of nodes (which determines k) and a relatively small number of edges (which determines n). The successive replacement method is independent of the range of constants in a problem instance.

5 Approximate Solution

In this section we propose a polynomial-time algorithm for finding an approximate solution to the minimization problem and a method for iterative refinement of this solution. Recall that we are to minimize

$$g_1 = \sum_{i=1}^{k} \sum_{j=1}^{n_i} |x_i - c_{ij}| + \sum_{i=1}^{m} |A_i \cdot X - b_i|$$

Consider

$$g_2 = \sum_{i=1}^{k} \sum_{j=1}^{n_i} (x_i - c_{ij})^2 + \sum_{i=1}^{m} (A_i \cdot X - b_i)^2$$

Let $X^{(r)} = [x_1^{(r)}, x_2^{(r)}, \ldots, x_k^{(r)}]$ be a rational solution[7] of k linear equations

$$\sum_{j=1}^{n_p} (x_p - c_{pj}) + \sum_{i=1}^{m} a_{ip}(A_i \cdot X - b_i) = 0 \quad , \quad 1 \le p \le k \tag{13}$$

[6] Because network matrices are totally unimodular, the hyperplanes form an integral polyhedra [16].

[7] For a set of linear equations $Ax = b$, where A is an $n \times m$ matrix of rank r there always exists a generalized inverse [17] of matrix A, denoted by A^+. The vector $X = A^+ b$ is the least square solution of $|Ax - b|_2$, where $||_2$ denotes the second norm. So there always exists a solution to (13).

where a_{ip} is the p-th element of A_i. These equations are obtained by differentiating g_2 with respect to $x_1, \ldots, x_k$.

Let $X^{(0)}$ be an integer approximation of $X^{(r)}$, i.e. $X^{(0)} = [\ldots, \lfloor x_i^r + \frac{1}{2} \rfloor, \ldots]$. For all X, including $X^{(m)}$ that minimizes g_1, we have $g_2(X) \geq g_2(X^{(r)})$. It is easy to show that $\sqrt{\sum_{i=1}^p y_i^2} \leq \sum_{i=1}^p |y_i| \leq \sqrt{p * \sum_{i=1}^p y_i^2}$. Hence

$$\sqrt{g_2(X^{(r)})} \leq \sqrt{g_2(X^{(m)})} \leq g_1(X^{(m)})$$

Thus $X^{(r)}$ approximates the minimum point $X^{(m)}$ with a relative error not larger than (assuming $g_1(X^{(m)}) \neq 0$)

$$\frac{g_1(X^{(r)})}{g_1(X^{(m)})} - 1 \leq \frac{g_1(X^{(r)})}{\sqrt{g_2(X^{(r)})}} - 1 \leq \frac{\sqrt{N * g_2(X^{(r)})}}{\sqrt{g_2(X^{(r)})}} - 1 = \sqrt{N} - 1$$

where $N = \sum_{i=1}^k n_i + m$ is the total number of terms in the objective function g_1. Similarly, $X^{(0)}$ approximates the minimum point with a relative error no larger than $\sqrt{N} * \sqrt{\frac{g_2(X^{(0)})}{g_2(X^{(r)})}} - 1$.

Obtained solution, $X^{(0)}$, if not optimal, can always be improved by the following iterative scheme adapted from the cyclic coordinate descent algorithm for non-linear programming [13]. The algorithm minimizes (6) cyclically with respect to the coordinate variables. Since (6) does not have continuous first partial derivatives at all points, many of the standard line search methods, such as Newton's method or the method of false position [13] cannot be used to find the local minimum along each coordinate. However, theorem 1 can be used to find such a local minimum at each step.

We start with the point $[x_1^{(0)}, x_2^{(0)}, \ldots, x_k^{(0)}]$ obtained above. Each iteration consists of k steps. Let at the end of the i-th iteration, $i = 0, 1, 2, \ldots$, the solution be at the point $[x_1^{(i)}, x_2^{(i)}, \ldots, x_k^{(i)}]$. At the j-th step of $(i+1)$-th iteration, substitute the variables

$$x_1, x_2, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k$$

in expression (6) with values

$$x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, x_{j+1}^{(i)}, \ldots, x_k^{(i)}$$

respectively, and obtain the optimum value $x_j^{(i+1)}$ for the variable x_j that minimizes the resulting expression (which, after substitution, has exactly one variable x_j) using theorem 1. Unfortunately this simple scheme may not converge to the global minimum as illustrated by the following example, even though the point where it stabilizes is "close" to the minimum point.

Example 1 Minimize

$$g_1 = |x_1 - 1| + |x_1 - 2| + |x_1 - 6| + |x_2 - 1| + |x_2 - 2| + |x_2 - 5| + |x_1 - x_2 - 1| + |x_1 - x_2 - 1|$$

Using the approximate method, it can be easily determined that $X^{(m)} = [\frac{58}{21}, \frac{61}{21}]$, from which the initial guess $X^{(0)} = [3, 3]$. At the end of the first iteration, $X^{(1)} = [4, 3]$ (where the expression evaluates to 12) and the simple iterative scheme terminates, even though the minimum points are [2,1] and [3,2] with the minimum value 10. □

To get a better understanding of the iteration scheme described above, a geometric interpretation of the process is helpful. Figure 5 shows the successive solution points at different steps of the iteration scheme for the problem in example 1. The iteration starts at point 1 in the figure. At step j of an iteration, a line is drawn through the current point parallel to the j-th dimension

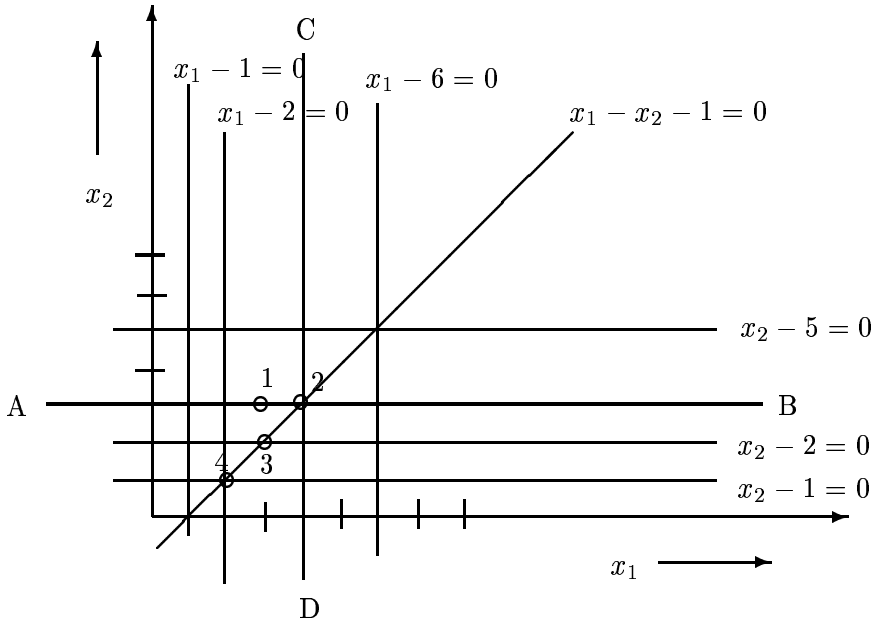

Figure 5: Geometric interpretation of the iterative scheme in section 5.

and a point (referred to as the local minimum point) on this line, at which the expression evaluates to the minimum among all the points on the line, is selected as the next point. The process of substituting values for x_i's ($i \neq j$) in g_1 is geometrically equivalent to determining all the points of intersection of this line with all the hyperplanes of g_1. Once these points are found, the a local minimum point on this line can be determined by selecting one of the middle-most point(s) of these points of intersection (see theorem 1). In figure 5, step 1 of iteration 1 determines the local minimum point on line AB. Since point 2 is the middle-most point (there are two identical lines, $x_1 - x_2 - 1 = 0$, due to the presence of two terms $|x_1 - x_2 - 1|$ in g_1), the solution moves from point 1 to point 2. In step 2 of iteration 1, a local minimum point on line CD is determined and it turns out that point 2 is still a local minimum point.

Example 1 shows that even though point 2 is a local minimum point for any line passing through it that is parallel to any of the axis, it is not the (global) minimum point. To be a (global) minimum point, a point must be a local minimum point for lines passing through it along any direction.

However, not all the directions need to be considered. Under the equivalence relation "intersects with the same set of hyperplanes", the set of directions can be divided into a finite number of equivalent classes. Only one direction from each class need to be considered, because they all result in the same set of intersection points at a given step of iteration. If at each iteration, local minimum points are determined along each of these directions, the modified algorithm (which may not be

efficient) will converge to the minimum point.

Continuing with example 1, the solution can be improved if the next line is chosen parallel to $x_1 - x_2 = 0$ (and passing through point 2), instead of being parallel to one of the axis, to determine the next local minimum point. Both point 3 as well as point 4 are such local minimum points and incidentally, they are also the minimum points.

6 Experimental Results

The approximate algorithm described in the previous section has been implemented and the performance of the algorithm is shown in table 1 and figures 5, 6 and 7. Only lines parallel to the axis are used in the experiments. Randomly generated graphs of different node sizes and different *edge probabilities* were used to find the modified dependence graph (see section 2). The number of references between two arrays (i.e., the *number of terms* in (6) having same coefficients but different constants) were chosen randomly from a normal distribution. The constants in the expression (6) were obtained by multiplying the quantity *range* with values drawn from a normal distribution with *mean* as shown in table 1 and standard deviation 1. In the table, *found* indicates the value of (6) upon termination of the algorithm, *at 0* indicates the value of (6) at the origin and *no of vars* indicates the number of variables in (6). Theorem 1 can be applied to each group of terms in (6) which differ only at the constant term to find the minimum for the sum of these terms in the group. In the table *lower bound* indicates the sum of the minimum obtained for each such groups in (6). Since we have no efficient way of determining the actual minimum points, the performance of our algorithm is compared with this lower bound. In the last column, *imp* denotes the improvement of the alignment method over no alignment and it is defined as

$$\frac{value\ at\ 0 - value\ found}{value\ found}$$

From this experiment the following observations can be made:

- The algorithm converges in a few iterations. The number of iterations increases slightly with an attendant increase in the number of variables in (6), but increases more rapidly with the increase in the dispersion of the constants in (6).
- As indicated in the above example, the algorithm may terminate at a point which is not a minimum point of (6). However a comparison of columns 5 and 6 shows that the algorithm terminates at a point at which the value of (6) is close to minimum.
- When the constants are drawn from a normal distribution with mean 0, there is an improvement of about 30-50%, although the value of the function at the minimum point cannot be much larger than that obtained by the algorithm, as is evident from the lower bound column. Thus, when the dependence vectors in a program lie randomly, there cannot be considerable improvement.
- When the mean of the distribution increases, the improvement as well as the number of iterations increase. Thus when the dependence vectors in a program show a tendency to

lie in a particular direction the improvement is much greater, although the algorithm takes longer time to terminate.

Because of the locality of references, it can be fairly assumed that the modified dependence graph of a real program is no more complex than the randomly generated graphs used in the experiment. So it is reasonable to expect that the average performance of a real program will be no worse than the average performance shown here. The two-edge connected components in a modified dependence graph in real programs are often smaller in size than those in the randomly generated graphs, making it possible to solve a larger problem in the same amount of time.

Mean = 0; Range = 5; Edge Probability = 0.03							
No of Nodes	*No of terms*	*No of vars*	*at 0*	*lower bound*	*found*	*No of iters*	*imp (%)*
45	78.0	2.9	207.1	139.8	143.1	5.5	44.7
55	116.6	10.6	261.7	163.8	174.8	7.2	49.7
65	168.8	22.5	357.0	215.8	246.6	8.7	44.7
75	219.2	36.5	416.3	239.7	287.2	9.0	44.9
85	287.8	54.2	591.2	329.7	425.6	9.2	38.9
95	357.3	64.4	792.4	440.4	590.9	9.6	34.1
100	392.2	70.7	876.4	485.0	666.0	9.2	31.6
Mean = 5; Range = 5; Edge Probability = 0.03							
45	77.4	3.4	316.1	130.5	137.0	6.8	130.8
55	117.5	8.6	482.4	190.5	207.1	13	133.0
65	165.6	21.5	634.7	222.3	274.1	13.6	131.6
75	226.5	36.6	884.9	289.9	401.2	17.5	120.4
85	287.4	50.1	1136.9	364.1	546.9	17.1	107.9
95	359.0	66.9	1463.1	450.6	765.1	18.4	91.2
100	393.8	74.7	1566.0	482.9	847.4	18.5	84.8
No of Nodes = 25; Range = 5; Edge Probability = 0.2							
Mean	*No of terms*	*No of vars*	*at 0*	*lower bound*	*found*	*No of iters*	*imp (%)*
0	175.1	23.4	376.9	226.7	300.6	7.1	25.4
5	176.1	23.6	700.3	234.6	408.1	13.3	71.6
15	177.1	23.7	1935.1	247.1	782.8	27.0	147.2
20	175.0	22.9	2519.9	243.7	983.2	34.7	156.3
30	176.3	23.2	3793.2	243.9	1423.0	45.0	166.6
40	172.4	23.0	4971.7	242.6	1799.4	54.9	176.3
50	177.3	23.1	6270.2	245.1	2306.9	64.5	171.8
No of Nodes = 50; Range = 5; Edge Probability = 0.2							
0	171.9	29.6	355.3	203.1	247.3	9.2	43.6
10	163.9	27.4	1191.5	218.0	381.0	23.8	212.8
20	167.0	28.2	2450.3	223.0	598.0	42.3	309.7
30	166.5	29.0	3677.3	222.3	794.5	62.3	362.8
40	162.0	28.7	4656.0	213.6	929.5	83.5	400.9
50	163.1	28.6	5796.1	213.4	1104.1	92.5	425.0
No of Nodes = 25; Mean = 0; Edge Probability = 0.2							
Range	*No of terms*	*No of vars*	*at 0*	*lower bound*	*found*	*No of iters*	*imp (%)*
20	173.9	23.1	1609.5	942.5	1221.8	18.9	31.7
35	174.1	23.3	2846.4	1653.8	2145.1	26.8	32.7
50	178.3	23.5	4176.4	2445.2	3154.1	34.4	32.4
No of Nodes = 50; Mean = 0; Edge Probability = 0.05							
20	163.7	29.0	1471.6	833.0	985.5	20.1	49.3
35	168.9	31.2	2668.5	1504.6	1770.6	34.7	50.7
50	167.0	26.3	4123.6	2360.2	2718.3	40.6	51.7

Table 1: The improvement obtained by using the iterative algorithm and the number of iterations it takes to converge when applied on randomly generated graphs.

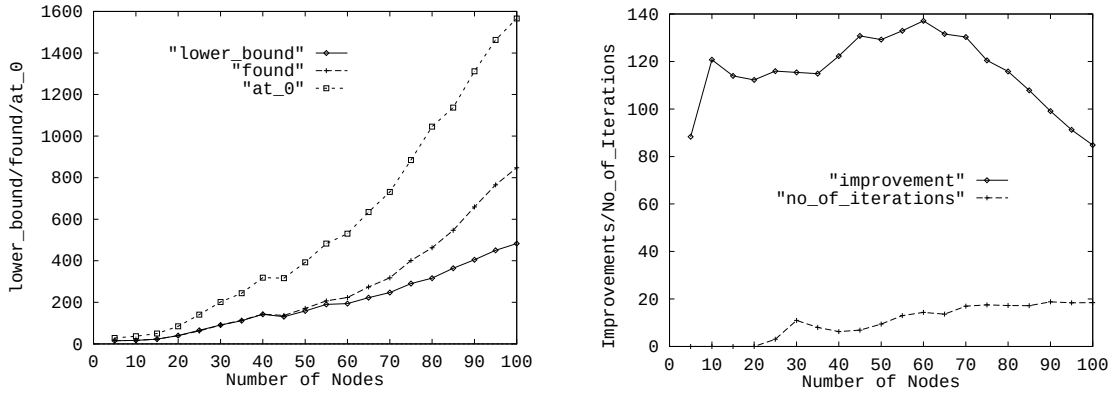

Fig 5. Performance of the algorithm as the number of nodes increases (Range = 5, Mean = 5, Edge Probability = 0.03).

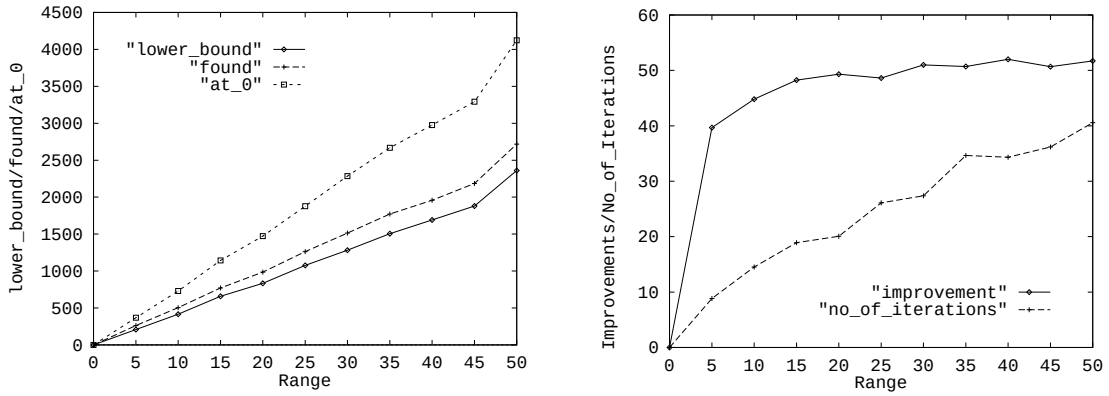

Fig. 6. Performance of the algorithm as the range of the constants increases (Number of nodes = 50, Mean = 0, Edge Probability = 0.05).

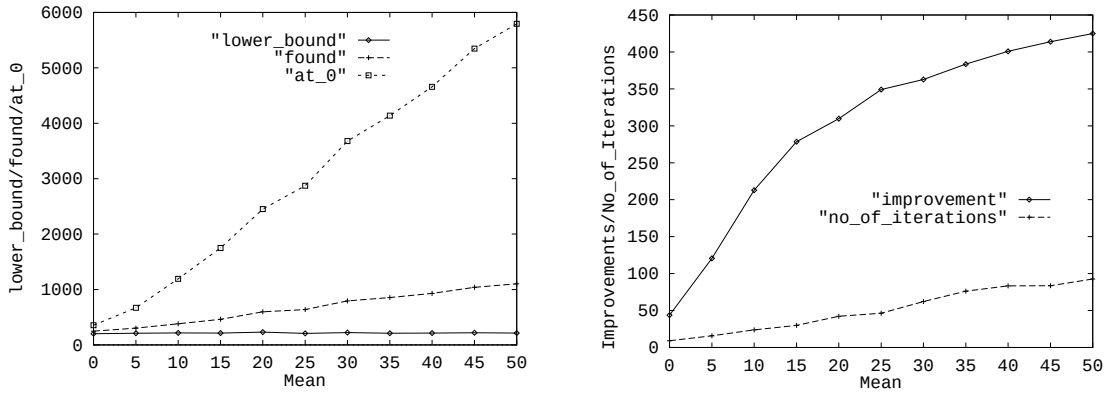

Fig. 7. Performance of the algorithm as the mean of the constants increases (Number of Nodes = 50, Range = 5, Edge Probability = 0.05).

7 Extensions of the Result

For more general types of index references, a good alignment may require transposing or permuting the index domains of the arrays. For example, $A[I,J] = B[J+1, I+2]+B[J-3, I-2]+B[J-2, I-1]$ requires B to be transposed, before its effective alignment with A can be found using the presented methods. Heuristic methods for index domain alignment ([15]) or language support (for example, HPF [5]) to allow the user to supply the best permutation of the index domains of the different arrays can be used for this purpose.

The method presented in the paper can be applied to programs involving more complicated array references than is permitted by RIA. Array references of the form $c - I$, where c is a constant or a loop independent value can be easily incorporated by reversing the distribution of the array along the dimension in which $c - I$ occurred and treat the reference as $I - c$ for our analysis.

For more complicated array references, consider aligning the arrays in the doall loops of figure 6 on a two-dimensional mesh. The third index domain of arrays A and B can be kept in the local memory of the processors. Let A_c and B_c, for $c \in \{10, \ldots, 20\}$, represent the array sections $A[I, J, c]$ and $B[I, J, c]$ for $10 \leq I, J \leq 100$. Our method can be applied to align array sections A_c with B_c, where the labels between A_c and B_c are $x - c$ for aligning the first dimension (similarly, $x' - 2c$ for aligning the second dimension), the same variable x being repeated for all values of c for the same dimension. If all the processors iterate on subscript K in a lock-step fashion, then for the best results, array C (which is of two dimension) should initially be aligned with array sections A_{10} and B_8 using our method, assuming the value of K to be 10 and then at each iteration moved by a vector [2,3] (coefficients of K in the subscripting expressions for array C).

$$
\begin{array}{l}
\textit{doall } I = 10, 100 \\
\quad \textit{doall } J = 10, 100 \\
\quad\quad \textit{doall } K = 10, 20 \\
\quad\quad\quad A[I, J, K] = B[I + K - 5, J + 2K - 3, K - 2] + C[I + 2K - 3, J + 3K - 1]
\end{array}
$$

Figure 6: A program with complicated subscripting expressions for which optimum alignment can be determined.

If an array has fewer dimensions than the processor domain of the parallel architecture then it is often replicated across the processor array and therefore needs to be aligned only in its own dimensions. Often, the data structure is wrapped around the processor array (this is commonly used in architectures that support wrap-around when the range of the data structure dimension is higher than the corresponding processor domain dimension). This solution reduces the size of local memory available to each processor, but the alignment technique presented remains applicable. If the target architecture does not support wrap-around, then a block of data is usually mapped to each processor. To apply our method in this case, each term in the objective function, expression (6), should be divided by the length of the block along the dimension being aligned.

Branching instructions in a program may cause some of the array references to be used more often than the others. A priori knowledge of the relative use of the array references to be made at run time, if available, can be used in our method (hence, further reduce the communication cost) by multiplying each term in the objective function with suitable coefficients. Sometimes it may be beneficial (for example, odd elements of an array may have subscripting expressions very different

than the even elements) to split an array into two or more (for example, odd vs even elements) to facilitate better alignment.

8 Conclusion

Scientific and engineering computations are numerically intensive, but often such computations display regularity in both the control flow patterns and the data structures used. Such computations are often comprised of iterative applications of numerical algorithms to all (or most of) the parts of the data structures, a characteristic very suitable for massively parallel processing.

The problem of aligning data structures and tasks for such computations in order to minimize communication cost in a distributed memory machine has been discussed in this paper. For programs with regular index references, the alignment problem has been shown to be equivalent to the problem of finding the closest vector in a restricted class of lattices. It is shown that this subclass of the closest vector problem is NP-hard. We have described two algorithms for finding the exact minimum. Although the complexity of these algorithms is exponential, the execution times should be acceptable, for small lattices generated by a large class of parallel scientific computations. In addition, a polynomial-time algorithm for finding a local minimum has been presented. The performance of this algorithm on randomly generated graphs has been measured and discussed.

The target machine assumed in the analysis is a mesh-connected architecture, which can be easily embedded on most distributed memory machines. The cost function of alignment is defined in terms of the distance between two processors. We focused on complexity and algorithms for alignment when the distance is defined in terms of the first norm, as is the case for four-nearest neighbor meshes. However, other network interconnections and various message passing strategies (such as grouping messages from different processors before forwarding them further or storing messages for later use) may require that different metrics be considered for the distance. Neither the complexity nor an efficient solution of the alignment problem for metrics other than the first norm considered here are known to the authors.

Acknowledgement We are very greatful to the anonymous referees for their valuable suggestions in improving the presentation of the material in this paper.

References

[1] M. O'Boyle and G.A. Hedayat, "Data Alignment: Transformation to Reduce Communication on Distributed Memory Architectures," *Proc. Scalable High Performance Computing Conference*, IEEE Computer Science Press, pp. 366-371, 1992.

[2] M. Chen, Y. Choo and J. Li, "Crystal: Theory and Pragmetics of Generating Efficient Parallel Code," in *Parallel Functional Languages and Compilers* by Boleslaw K. Szymanski, editor, Frontier Series, ACM Press, New York, New York, 1991.

[3] D. L. Farnsworth, "The Sum of Absolute Deviations and the Sum of Squared Deviations," in *The American Mathematical Monthly: The Official Journal of the Mathematical Society of*

America, v. 97, no. 12,Dec., 1990 pp 911-912. See also *Introduction to Mathematical Statistics*, 4th ed., Macmillan, New York, 1983.

[4] M. Gupta, "Automatic Data Partitioning on Distributed Memory Multicomputers," Ph. D. dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1992.

[5] High Performance Fortran Forum, *High Performance Fortran Language Specification*, version 1.0, May 1993.

[6] S. Hiranandini, K. Kennedy and C.-W. Tseng, "Compiler Support for Machine-Independent Parallel Programming in Fortran D," Technical Report, COMPTR 90-149, Department of Computer Science, Rice University, Feb. 1991.

[7] J. R. Gilbert, and R. Schreiber, "Optimal Expression Evaluation for Data parallel Architectures," in *Journal of Parallel and Distributed Computing*, Vol. 13, No. 1, Spetember 1991, pp. 58-64.

[8] R. Kannan, "Minkowski's convex body theorem and integer programming," in *Math. Operations Res.* Vol. 12, No. 3, August 1987, pp. 415-440

[9] R. M. Karp, R. E. Miller and S. Winograd, "The organization of computations for uniform recurrence equations", *JACM*, July 1967.

[10] R. M. Karp, "Reducibility among Combinatorial problems" in R. E. Miller and J. W. Thatcher (Eds.), *Complexity of Computer Computations.*, Plenum Press, New York,1972, pp. 85-103.

[11] K. Knobe, J. D. Lukas and G. L. Steele Jr., "Data Optimization: Allocation of Arrays to Reduce Communication on SIMD Machines," in *Journal of Parallel and Distributed Computing*, **8**, 1990, pp. 102-118.

[12] D. J. Kuck, R. H. Kuhn, D. A. Padua, B. R. Leasure and M. Wolfe, "Dependence Graphs and Compiler Optimizations," in *Conf. Record of the 8th ACM Symp. on the Principles of Programming Languages*(Williumsburgh), ACM Press, pp. 207-218.

[13] D. G. Luenberger, *Linear and Nonlinear Programming* , Addison-Wesley Publishing Company, 1984.

[14] H. W. Lenstra, "Integer Programming with a Fixed Number of Variables". First announcement (1979). *Math. Oper. Res.* **8, 4** 1983, pp.538-548.

[15] J. Li and M. Chen, "The Data Alignment Phase in Compiling Programs for Distributed Memory Machines," in *Journal of Parallel and Distributed Computing*, Vol. 13, No. 2, October 1991, pp. 213-221.

[16] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, John-Wiley and Sons, 1988.

[17] W. Pearson, *Handbook of Applied Mathematics,* Van Nostrand Reinhold, New York, 1983, p. 916-917

[18] S. K. Rao, *Regular Iterative Algorithms and their Implementations on Processor Arrays*, Ph. D. dissertation, Stanford University, Stanford, CA 1985.

[19] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors*, The MIT Press, Cambridge, Massachussetts, 1989.

[20] W. Shang and J. A. B. Fortes, "Independent Partitioning of Algorithms with Uniform Dependencies," in *IEEE Transactions on Computers*, Vol 41, No 2, February 1992.

[21] S. K. Skedzielewski, "Sisal," in *Parallel Functional Languages and Compilers* by Boleslaw K. Szymanski, editor, Frontier Series, ACM Press, New York, New York, 1991.

[22] Boleslaw K. Szymanski, editor *Parallel Functional Languages and Compilers*, Frontier Series, ACM Press, New York, New York, 1991.

[23] P. van Emde Boas, *Another NP-complete problem and the complexity of computing short vectors in a lattice*, Rep. 81-04, Math. Inst. Univ. Amsterdam, 1981.