International Conference on Supercomputing 1992, Washington.

Deriving Good Transformations for Mapping Nested Loops on Hierarchical Parallel Machines in Polynomial Time

K G Kumar,* D Kulkarni† and A Basu
Center for Development of Advanced Computing
2/1 Brunton Road, Bangalore 560 025, INDIA

Abstract

We present a computationally efficient method for deriving the most appropriate transformation and mapping of a nested loop for a given hierarchical parallel machine. This method is in the context of our systematic and general theory of unimodular loop transformations for the problem of iteration space partitioning [7]. Finding an optimal mapping or an optimal associated unimodular transformation is NP-complete. We present a polynomial time method for obtaining a 'good' transformation using a simple parameterized model of the hierarchical machine. We outline a systematic methodology for obtaining the most appropriate mapping.

1 Introduction

Efficient execution of many programs on parallel machines demands a methodology for partitioning the iteration spaces of the nested loops that form the core of their computations. Most existing approaches treat transformations like loop interchange (permutation), outer and inner loop parallelization, reversal etc. independently. Only recently have there been attempts [2, 4, 5, 10] to treat the transformations in the unified framework of unimodular matrix transformations. A even more general and comprehensive theory has been presented in our recent work [7].

There is a growing trend today towards building machines that are hierarchical in the sense of supporting more than one possible physical level of interaction among processes. It is therefore most appropriate and relevant to target our theory and techniques to partitioning of iteration spaces to enable their most efficient mapping on such hierarchically defined multiprocessors.

In this paper, we present a computationally efficient technique for deriving the most suitable iteration space partitioning transformation and mapping for a given hierarchical parallel machine in the context of our recent theory of unimodular transformations and mappings for hierarchical machines. We introduce the problem of loop partitioning as one of obtaining suitable unimodular transformations. We present our abstract hierarchical and formally define the mapping problem. We then show the means of guaranteeing the deadlock freedom for a chosen of a simple mapping. We describe the derivation of an *otimal direction* and our method for obtaining a good sub-optimal and polynomially computable direction and transformation templates. We finally present the overall strategy for choosing the best transformation.

2 The Problem of Loop Partitioning

The goal of loop partitioning is identified to be one or a combination of outer loop parallelization, inner loop parallelization and vectorization, depending on the architecture in question. The choice between outer and inner loop parallelism depends on the communication costs. Outer loop parallelization calls for additional communication costs to satisfy the dependences that might still exist at the outer loops. Where as, inner loop parallelization usually involves barrier synchronizations at a relatively low communication cost. If the length of sequentialization thus enforced is minimized, it certainly is more advantageous to perform inner loop parallelization to the fullest extent. This is however not very practical since the number of parallel iterations involved is then a product of all the loop bounds and is hence too large for a closely coupled machine.

2.1 Unimodular Transformations for N–D Iteration Spaces

The concept of unimodular transformations, introduced by Banerjee [2] and by Wolf and Lam in [10], is a uni-

*Currently visiting the Department of Computing & Information Sciences, Queen's University, Kingston, Canada.
†Currently with the University of Toronto, Canada.

form and general framework for viewing Lamport's general hyperplane method [8], loop reversal, loop skewing and loop permutaions. The general goal of a method that looks for unimodular transformations is to find a unimodular matrix U ($|\ det(U)\ | = 1$) that characterizes the form of the loop after transformations. The new loop indices in the transformed loop are obtained in terms of the old loop indices by a matrix multiplication.

We consider only perfectly nested loops with constant loop limits and arbitrary nesting levels. We assume that all data dependences between iterations are flow dependences. An iteration $j = (j_1, ..., j_n)$ is flow dependent on iteration $i = (i_1, ..., i_n)$ if a statement in j reads a value modified by a statement in i; and $d = (j_1 - i_1, ..., j_n - i_n)$ is the corresponding dependence distance vector.

Henceforth, by a *dependence matrix* we mean an integer matrix whose rows correspond to the dependence distance vectors in the iteration space in question. We denote the dependence matrix by D.

Definition 1 (Internalization) *We say that an N–D iteration space* $\Re$ internalizes *a dependence vector d at level $r \le N$, if the outermost $N - r$ loops of $\Re$ are independent of each other with respect to d.*

Internalization of vector d at level $r \le N$ in other words means that d is confined to the innermost r loops.

Theorem 1 *Given an N–D iteration space and r dependence vectors which correspond to a dependence matrix D, the minimum level required to internalize all the dependences is k the rank of D, and the unimodular transformation to internalize them exactly at level k exists when $k < N$.*

The proof of this theorem which is available in [7], is constructive and is based on the following algorithm.

Algorithm A1:

We start with the pair of matrices D and U which is initially the N by N identity matrix I. For every column operation on the D matrix, we perform the same operation on the U matrix. In the algorithm below, by the elimination of an element in position (i,j) by another in position (i,j'), we mean the following:

Let a_{ij} and $a_{ij'}$ be the elements in the positions (i, j) and (i, j') respectively. If $a_{ij} \ge a_{ij'}$, then subtract n times the j'^{th} column from the j^{th} column where, $|n|.|a_{ij'}| \le |a_{ij}| < (|n| + 1).|a_{ij'}|$. Otherwise, interchange the j^{th} and j'^{th} columns and repeat the action. This goes on until we have zero in the position (i,j).

The algorithm involves the following steps:

Step 1: Initialize row count $i = 1$; Initialize right margin $rm = N$

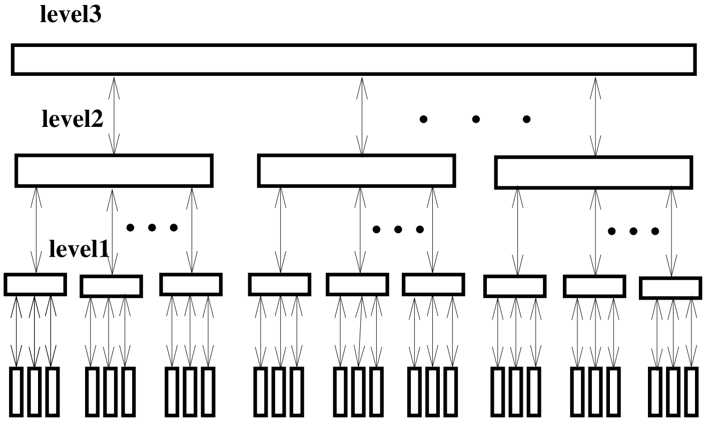

Figure 1: The Hierarchical Architecture Model

Step 2: Initialize column count $j = 1$;
Step 3: If $j < rm$ then Eliminate (i, j) using $(i, j + 1)$; Go to step 4;
Otherwise go to step 5;
Step 4: Increment j to $j + 1$; Go to step 3;
Step 5: Increment i to $i + 1$; If $i \le N$ and $i \le r$ then decrement rm to $rm - 1$ and go to step 2
Otherwise Stop.
End of Algorithm A1

It is easy to show [1] that these operations preserve unimodularity of U. In general, algorithm A1 has $O(N^2)$ or $O(r^2)$ (whichever is smaller) basic steps as its complexity upper bound, where N a basic step corresponds to the elimination operation.

Theorem 2 *Given a set S of positive dependence vectors in an N–D iteration space which have a rank k where $1 \le k \le N$, there exists a choice of a subset S' of atmost k vectors from S such that, any transformation U that internalizes all dependences in S' at level $N - k + 1$ internalizes all dependences of S also at level k and transforms all the vectors in S to positive vectors.*

The interested reader is refered to [7] for this proof. It is shown that an initial ordering of the dependences in D always exists such that algorithm A1 will then produce the desired result.

3 Architectural Model

The architectural model adopted in this work is that of a collection of processors with a hierarchy of communication media, that organizes the processors into hierarchical clusters. Inter-cluster communication cost increases with the level of the cluster in the hierarchy, but is uniform over a certain level. Figure 1 illustrates such a machine with three levels in its hierarchy.

The model is sufficiently general to describe most contemporary parallel architectures. These may be described by variations in the depth of the model (number of different communication levels) and the nature of the communication media. The communication media at the lower end of the spectrum could be caches, vector registers or physically shared address spaces, and at the higher end, could be switch based or bus based communication links. Conventional shared memory and distributed memory machines thus become interesting special cases. Example of a hybrid where each processor possesses the capability to do vector processing using multiple functional units or other cycle saving schemes is iWARP. Other examples of hybrids would be the CEDAR, SUPRENUM and DASH architectures.

4 Problem of Mapping Iteration Spaces

The problem of mapping a program on a real hierarchical architecture is in general one of mapping the parallelism inherent in it onto the various levels so as to result in the best performance, given the resource constraints and communication cost associated with each level. We now introduce some definitions.

Definition 2 (Mapping Definition) *A* mapping definition *is a tuple, whose first element is an integer indicating the hierarchy level in the machine and the second element is an "s" or a "p" corresponding to sequential or parallel.*

Definition 3 (Mapping) *Given an N–D iteration space, a* mapping *of the space on a hierarchical machine is an N-tuple of mapping definitions, one corresponding to each level of the iteration space, such that the level of the i^{th} element of a mapping is greater than or equal to the level of its j^{th} element whenever $i < j$.*

Definition 4 *When the j^{th} element of a mapping has its second field as "s", the j^{th} level of the iteration space is said to be mapped* sequentially. *Otherwise, when this field is "p", it is said to be mapped* parallely.

Definition 5 (Mapping Level) *The* mapping level *$ml(j)$ corresponding to the j^{th} level of a nested loop is the level of the machine hierarchy on which this loop level is mapped.*

Example 1: The mapping $(<2,p>, <2,s>, <1,s>, <1,p>)$ corresponds to executing levels 1 and 4 in parallel on machine levels 1 and 2 respectively and 2 and 3 in sequence, again on levels 1 and 2.

Definition 6 (Sub-Vector) *A* sub-vector *is an n-tuple of consecutive elements of a dependence vector.*

Definition 7 (Sequential Sub-Vector) *A sub-vector is a* sequential sub-vector *iff all its elements correspond to levels which are mapped sequentially and elements on either side of it are either non-existent or mapped parallely.*

Example 2: The mapping of example 1 gives rise to the sequential sub-vectors (1, -2) and (-1, -3) in dependences (3, 1, -2, -3) and (2, -1, -3, 4) respectively.

Definition 8 *A sequential sub-vector is said to be positive iff it is lexicographically positive in the same sense as a dependence vector.*

It turns out that practically every mapping in conventional use including those for hierarchical machines adopts a *single sequential layer* pattern and we call this mapping definition a *simple mapping.*

Definition 9 (Simple Mapping) *A mapping of an N–D space is said to be* simple *iff it results in exactly one sequential sub-vector corresponding to every dependence vector in the space.*

The mapping of example 2 is a simple mapping.

In the sequel, we refer only to simple mappings although, with a little effort, most of the results may be shown to hold for general mappings. The definition below is needed for the discussions in the sequel.

Definition 10 (Internalization Reference) *The* internalization reference *of a simple mapping is the starting nest level of the sequentialization.*

5 Transformations for Deadlock Free Mapping

A deadlocking situation is one wherein the dependences demand an order of execution of the iterations that is in conflict with the order of sequentialization in the mapping.

Theorem 3 *A mapping of an N–D iteration space with a set S of dependence vectors in it is deadlock free iff all resulting sequential sub-vectors corresponding to every dependence is positive.*

Proof: From the definition of a sequential sub-vector, we see that the levels which are above (or loops which nest) the levels of this sequential sub-vector, if any, are all executing parallely. This implies that elements of the sequential sub-vector alone are responsible for any ordering restrictions among the iterations nested by it. Thus, it is adequate if the sequential sub-vector is lexicographically positive. □

Example 3: The dependence (3, 1, -2, -3) is non deadlocking whereas the dependence (2, -1, -3, 4) deadlocks for the mapping of example 2.

Theorem 4 below establishes the existence of a transformation in every case to enable a non-deadlocking simple mapping that internalizes as many dependences as can be contained within a hyperplane of dimension k', the internalization reference of the mapping.

Theorem 4 *Let k' be the internalization reference of a simple mapping involving an N–D space with a set S of dependences. There exists a transformation such that any n dependences can be internalized at level $N-k'+1$ where, n is the size of the largest subset of S whose rank is less than or equal to $N-k'+1$ and the simple mapping would be deadlock free.*

Proof: The proof is again constructive. We have seen in section 3 that we can find a transformation U' that internalizes all dependences of S at level k while preserving positivity, k being the rank of S. The algorithm below derives a transformation U from U' that is suitable for a deadlock free simple mapping with internalization reference k'.

Algorithm A2:

Step 1: Initialize column count $j = N-1$; Initialize row count lower bound $i_l = 1$; Determine the row index i of the first non-zero element from the top (if any) of column j of D;
If it exists, row count lower bound $i_l = i$;
else $i_l = r+1$; (where $r = |S|$)
Step 2: Determine the row index i of the first non-zero element from the top (if any) of column $j-1$ of D;
If it exists, row count upper bound $i_l = i-1$;
else $i_l = |S|$;
Step 3: For p ranging from i_l to i_u, we determine $c_p = \lceil |a_{p,N}|/a_{p,N-1} \rceil$. We define this to have a value 0 when $a_{p,N}$ is positive and a value 1 when $a_{p,N}$ is zero. Determine $n = max(c_p)$ for p ranging from i_l to i_u.
Step 4: Add n times column $N-1$ to column N of D.
Step 5: Decrement j to $j-1$;
If $j < N-k+1$ go to step 6
else $i_l = i_u + 1$; Go to step 2;
Step 6: Interchange columns k' and N; Stop.

End of Algorithm A2 □

The complexity of this algorithm in the worst case is clearly $O(N)$ column additions and $O(r)$ integer divisions.

Example 4: Consider the set of dependences $S = \{(2,1,3), (3,1,-2), (1,-2,-1), (4,1,-7)\}$. The transformation U' and D' from algorithm A1 are:

$$U' = \begin{bmatrix} 5 & 1 & 0 \\ -13 & -2 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad D' = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 2 & 1 \\ 30 & 5 & -2 \end{bmatrix}$$

The results of algorithm A2 on the above are:

$$U = \begin{bmatrix} 5 & 5 & 1 \\ -13 & -12 & -2 \\ 1 & 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \\ 30 & 28 & 5 \end{bmatrix}$$

6 Optimal Transformation for Simple Mapping

Definition 11 (Optimal Transformation) *We say a transformation is optimal for a simple mapping iff the total length of the sequential component in terms of the number of iterations is minimal.*

The time of execution is clearly proportional to the length of this sequential component. For $1 \leq j \leq N$, the j^{th} column of the transformation matrix represents the direction in the original space corresponding to the direction of the j^{th} loop traversal in the transformed space. Theorem 5 below guarantees a valid transformation corresponding to any choice of the direction of the k'^{th} loop.

Theorem 5 *Given any direction, we can construct a unimodular matrix with that direction vector as the k'^{th} column.*

Proof: Let the direction vector be defined by $\overline{u} = (u_1, u_2, \ldots, u_N)$ where g.c.d$(u_1, u_2, \ldots, u_N)$ is unity. Let for some $1 \leq i \leq N$, u_i be a non-zero element of $\overline{u}$. Clearly, there has to be at least one. Consider the set of vectors $(u_i, 0, \ldots, 0, -u_1, 0, \ldots, 0)$, $(0, u_i, 0, \ldots, 0, -u_2, 0, \ldots, 0)$, ..., $(0, \ldots, 0, -u_N, 0, \ldots, 0, u_i)$ where, the negation of the elements $u_1, u_2, \ldots, u_N$ all appear in the ith position. Clearly, each of the above vectors is normal to $\overline{u}$ and it is obvious that the rank of the matrix formed with the above vectors as the rows is $N-1$. Now, algorithm A1 provides us a method of obtaining a unimodular transformation whose first column is normal to all of them and therefore is the same as $\overline{u}$. Interchanging columns 1 and k' of this transformation gives us the desired transformation. □

A Simple Special Case: A simple special case is when one of the elements of $\overline{u}$, say the n^{th}, is ± 1. The desired unimodular matrix in this case may be obtained by replacing the n^{th} column of the identity matrix by $\overline{u}$ and interchanging the k'^{th} and nth columns.

Definition 12 (Optimum Direction)
Let $m_1, m_2, \ldots, m_N$ represent the size of the iteration space along the dimensions $1, 2, \ldots, N$ respectively and $\overline{u} = (u_1, u_2, \ldots, u_N)$ a candidate direction for optimality. It can be easily shown that the maximum diagonal to the iteration space along this direction has lengths $\sum_{i=1}^{N} m_i.|u_i|$. Also, let $S = \{d_1, d_2, \ldots d_r\}$ be the set of dependences where for $1 \leq i \leq N$, $d_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,N})$. The conditions for optimality may now be stated as the choice $\overline{u}$ such that $\sum_{i=1}^{N} m_i.|u_i|$ is minimized while for all $1 \leq i \leq N$ $\sum_{i=1}^{N} a_i.u_i$ is positive

Example 5: Consider a 4–D iteration space with $m_1 = 10$, $m_2 = 5$, $m_3 = 5$ and $m_4 = 10$. Let the dependences be {(0, 2, 0, -1), (0, 0, 1, -3), (1, 3, -1, 2), (2, -4, 3, 1)}. It may be established that the optimum direction is $\overline{u} = (0, 1, 2, 0)$. Optimal transformation for a simple mapping with internalization reference 2 can then be derived using algorithm A1 or the simpler method as

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Clearly, the formulation of the problem of finding the optimum direction above corresponds to an integer programming problem and its solution is in the general case NP-Complete.

Theorem 6 *Given a rectangular N–D iteration space, we can always derive a unimodular transformation that has an optimum direction as its k'^{th} column and internalizes a set of dependences of the space of rank k at level $k+1$ when $k < N$ and at level N when $k = N$.*

Proof: Our proof is again constructive. We have seen how a unimodular transformation U' with the optimum direction $\overline{u}$ as its k'^{th} column can be derived. Let D' be the transformed dependence matrix obtained using the transformation U'. Now, by applying algorithm A1 to D' and U' ignoring their k'^{th} columns, clearly we obtain the desired matrices. □

Example 6: Let us consider the same set of four dependences of example 5 for which we derived the optimal transformation for a simple mapping with internalization reference 2, and whose transformed dependence matrix was

$$D' = \begin{bmatrix} 0 & 2 & 0 & -1 \\ 0 & 2 & 1 & -3 \\ 1 & 1 & -1 & 2 \\ 2 & 2 & 3 & 1 \end{bmatrix}$$

Applying algorithm A1 on the truncated matrix in this case would clearly not lead to any further internalization. Thus, the desired final transformation U is the same as U'. All the dependences are internalized at level 4 and the first two dependences (with rank 2) at level 3.

In some special situations we could reduce D' to the last k columns using algorithm A1 either by involving the k'^{th} column without changing it or using columns other than the k'th only.

Example 7: Let us again consider the same set of four dependences of example 5. We first interchange rows 1 and 3 and then apply algorithm A1 treating the 2nd column as the last. This would result in the following final dependence matrix which clearly internalizes the first three dependences at level 3.

$$D' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & -1 \\ 0 & 2 & -1 & -1 \\ -13 & 2 & 26 & 7 \end{bmatrix}$$

Column 2 remains unaltered here by algorithm A1 since the first element on it is 1 which divides any other element.

7 Polynomial-Time Sub- Optimal Transformations

Let $\overline{d} = (a_1, a_2, \ldots, a_N)$ be any dependence. Clearly, we are interested in a direction which has a positive product with $\overline{d}$. Since we are looking for integer solutions, the region of interest corresponding to this dependence therefore may be defined by the constraint $a_1.i_1 + a_2.i_2 + \ldots + a_N.i_N \geq 1$. Each dependence in the space gives rise to such a constraint and our region for a solution is the intersection of all these regions. Note that this region is not only non-empty, it is guaranteed to have infinite number of integer points.

Definition 13 (m-Starting) *We say a dependence is* m-starting *if its m^{th} element is the first non-zero element starting from the left.*

Now if m be the maximum starting value among all dependences in the space, then there has to be a non-zero value in our solution for at least one of the elements from u_m to u_N.

Corresponding to any dependence $\overline{d}$, consider the equation

$$a_1.i_1 + a_2.i_2 + \ldots + a_N.i_N = 1 \qquad (1)$$

Definition 14 (Intercept along Original Axis)
The intercept *along an original axis, say $i_j = 1/a_j$, is the solution for i_j when we make all the other dimensions zero in Equation 1 and is defined only when $a_j \neq 0$.*

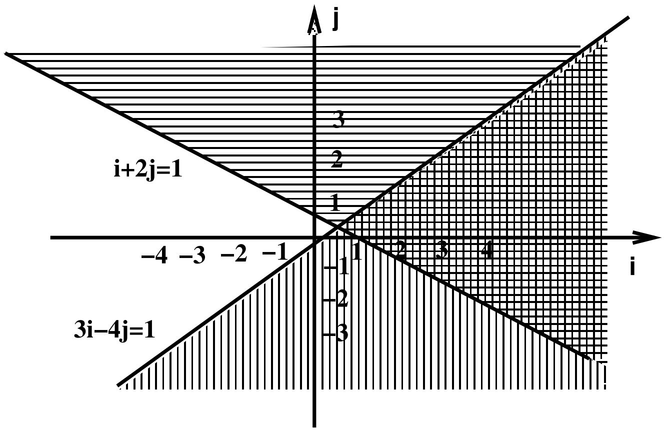

Figure 2: Region of Solutions

Definition 15 (Conflict-Free Dimension for Original Axis) *When the intercepts along an original axis are all non-zero and of the same sign, then we declare that dimension as* conflict-free *for the original axis.*

Example 8: Figure 2 illustrates a 2–D space with two dependences (1, 2) and (3, -4) which define the regions $i_1 + 2.i_2 \geq 1$ and $3.i_1 - 4.i_2 \geq 1$ respectively. The dependences are consistent along the i-axis but inconsistent along the j-axis. The region of solution are the shaded regions.

We now generalize the above notion of intercept to define the intercept along a displaced axis.

Definition 16 (Displaced Axis) *A displaced i_j^{th} axis corresponds to assigning a constant integer value different from 0 to at least one of the other indices different from i_j in Equation 1.*

Example 9: In a 3–D space, assignment of a value 1 to i_1 and i_2 would define a displaced axis i_3.

Let $\overline{c}_r = (c_1, c_2, \ldots, c_{j-1}, c_{j+1}, \ldots, c_N)$ represent the $N-1$ constant values assigned to the dimensions other than the jth. Let $\overline{d}_r$ be the reduced $N-1$ element vector obtained by removing the jth element from $\overline{d}$.

Definition 17 (Intercept along Displaced Axis) *The intercept of a dependence $\overline{d}$ along the displaced axis i_j is defined as $i_j = (1 - \overline{c}_r.\overline{d}_r)/a_j$.*

Note that when we have a case where a_j is zero, if $0 \geq (1 - \overline{c}_r.\overline{d}_r)$ is invalid, then the dimension in question cannot be a candidate for conflict-free intercepts.

Example 10: The intercept of the dependence (3, -2, 1) on the displaced axis i_3 of example 9 would be $i_3 = 0$.

Definition 18 (Conflict-Free Dimension for Displaced Axis)

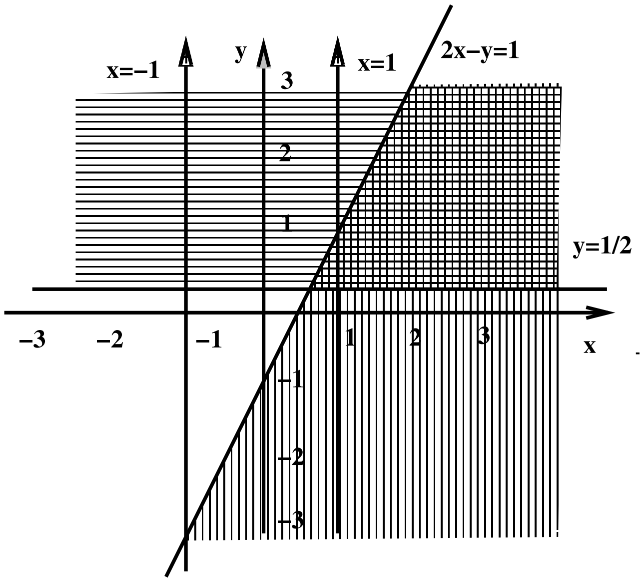

Figure 3: Conflict Along Axes

When the intersection of the regions of solutions corresponding to all dependences have at least one integer point (are non-conflicting) on the displaced axis, then we declare the corresponding dimension as conflict-free *for the displaced axis.*

Example 11: Figure 3 illustrates a 2–D space with two dependences (0, 2) and (2, -1) which are conflict-free along the displaced y-axis with $x = 1$ but are conflicting along the displaced y-axis with $x = -1$.

Definition 19 (Conflict-Free Point) *A* conflict-free point *on an axis is an integer point on the axis which is contained in the intersection of the regions of solutions corresponding to all dependences.*

Clearly, when the dimension is conflict-free for the axis in question, the intersection is non-empty and a conflict-free point exists.

Example 12: The point (1, 1) is a conflict-free point on the displaced y-axis with $x = 1$ in example 11.

Definition 20 (m-Fullness) *A dimension is said to be* m-full *iff for all m-starting dependences, all elements of that dimension are non-zero and have the same sign, and for all $m < n \leq N$, all n-starting dependences are such that their elements of that dimension are either zero or of the same sign as the m-starting ones above.*

Note that the m^{th} dimension is always m-full when we consider only positive dependences. Further, a dimension which is conflict-free for the original axis is m-full for any m.

Definition 21 (m-Conflict-Free Point) *A* conflict-free point *on an axis is an integer point* *on the axis which is contained in the intersection of the regions of solutions corresponding to all n-starting dependences, for $m \leq n \leq N$.*

Clearly, m-conflict-free points exist for all m-full dimensions.
The following two lemmas and the theorem below form the basis of the algorithm we present later for deriving a sub-optimal direction vector.

Lemma 1 *Let j be a conflict-free dimension for the original axis. Then the vector $\overline{u}$ whose j^{th} element is s and all other elements are zero has a positive product with all dependences where, s is +1 if the dimension is originally positive and -1 if it is negative.*

Proof: This is a consequence of the fact that the j^{th} elements of all dependences are of the same sign when the j^{th} dimension is conflict-free for the original axis. The choice of $\overline{u}$ is therefore bound to give the j^{th} element in the positive case and its negation in the negative case. □

Lemma 2 *Let j be a conflict-free dimension for a displaced axis with $\overline{c}_r = (c_1, c_2, \ldots, c_{j-1}, c_{j+1}, \ldots, c_N)$ representing the $N-1$ constant values assigned to the dimensions other than the jth. If s is a conflict-free point on the displaced axis, then $\overline{c} = (c_1, c_2, \ldots, c_{j-1}, s, c_{j+1}, \ldots, c_N)$ has a positive product with all dependences.*

Proof: By virtue of the definition of a conflict-free dimension for a displaced axis, the inequalities $i_j \geq (1 - \overline{c}_r.\overline{d}_r)/a_j$ corresponding to all the dependences are consistent. Further, since s is a conflict-free point, $s = (1 - \overline{c}_r.\overline{d}_r)/a_j$ is valid corresponding to all the dependences. That is to say, $\overline{c}.\overline{d} \geq 1$ corresponding to all dependences. □

Theorem 7 *Let $\overline{c}_{m+1} = (c_1, c_2, \ldots, c_N)$ be a vector whose product with all n-starting vectors is positive for $n > m$. Then, for any m-full dimension j such that $c_j = 0$, if s is a m-conflict-free point on the displaced j axis, assigning the value s to c_j in $\overline{c}_{m+1}$ will yield a vector whose product with all n starting vectors is positive for $n \geq m$.*

Proof: Let S_m be the set of dependences which are n-starting where $n \geq m$, and let j be a m-full dimension. For any dependence, we have two cases depending on whether its j^{th} element is a zero or not.

When the j^{th} element is a zero, it follows from the definition of m-fullness that the dependence cannot be m-starting. By our assumption therefore the product of $\overline{c}_{m+1}$ with this dependence is positive. Further, since its j^{th} element is zero this product will remain so no matter what is assigned to c_j.

Consider next the subset of all those dependences in S_m whose j^{th} elements are non-zero. In the inequality $a_1.i_1 + a_2.i_2 + \ldots + a_N.i_N \geq 1$, we assign the values of the elements $1, 2, \ldots, N$ of a dependence from this subset to $a_1, a_2, \ldots, a_N$ respectively and the values from $\overline{c}_{m+1}$ to all the indices except i_j. This yields a family of inequalities involving i_j, one corresponding to each of the dependences in the above subset. Owing to the assumption about the m-fullness of the jth dimension it follows that the j^{th} elements of all these dependences have the same sign. Therefore the above family of inequalities are consistent and we can certainly find a m-conflict-free point, say s, which when assigned to c_j will make the product of $\overline{c}_{m+1}$ with all these dependences positive. □

We now present our algorithm for deriving the sub-optimal direction vector. Our heuristics for achieving sub-optimality can be summarized as

- Picking from among candidate dimensions, always the one with the lowest weight.
- Seeking, as far as possible, solutions along the original co-ordinate axes. If this fails, then along a minimally displaced axis.
- Seeking a conflict-free dimension at the nearest opportunity.
- Choosing always the conflict-free point along a candidate axis that is closest to the origin

Algorithm A3:

Let us assume the original dependences are ordered in the matrix D to have a staircase form. Let $\overline{w} = (w_1, w_2, \ldots, w_N)$ be the vector of weights where for $1 \leq i \leq N$, $w_i = m_i$, the size of the iteration space along the ith dimension.
Step 1: Initialize the solution vector $\overline{u} = (u_1, u_2, \ldots, u_N) to (0, 0, \ldots, 0)$; Initialize m to the maximum starting value; Mark all dimensions as *not selected*.
Step 2: Look for a conflict-free dimension among the previously *not selected* dimensions;

If we happen to find some, choose the one with the minimum weight, say the jth; Determine the integer lower bound and upper bound lb_j and ub_j respectively for the conflict-free points along the jth axis; Let s_j be the point in the range that is closest to the origin. Assign the value s_j to u_j in $\overline{u}$; Stop.

If we fail to find any, among all previously *not selected* m-full dimensions, choose the one with the minimum weight, say the jth; Mark this dimension as *selected*; Determine the integer lower bound and upper bound lb_j and ub_j respectively for the m-conflict-free points along the j^{th} axis; Let s_j be the point in the range that

is closest to the origin. Assign the value s_j to u_j in $\overline{u}$; Decrement the starting value m to the next lower value; Go to step 2.
End of Algorithm A3

In the worst case, we may require to examine the conflict-freeness of a dimension $N.(N-1)/2$ times implying $O(r.N^3)$ multiplication operations.

Example 13: Let us take once again the initial dependence matrix of example 5. Initial maximum value of m is 3. Clearly, no dimension is conflict-free for original axis. The m-full dimension of minimum weight is 3 and minimal m-conflict-free point is 1. We now reduce m to 2. There are again no conflict-free ones among the remaining dimensions. The m-full dimension of minimum weight is 2 and minimal m-conflict-free point is 1. Again we reduce m to 1. There is now a conflict-free dimension, namely 1. The minimal conflict-free point is 1. Thus, the sub-optimal solution obtained is $\overline{u} = (1, 1, 1, 0)$ which is different from the optimal one $(0, 1, 2, 0)$.

7.1 Integrated Method for Sub-Optimal Transformation

Theorem 8 *At least one of the elements in the sub-optimal solution is a +1 or a -1.*

Proof: In fact, the first element of the sub-optimal solution to be fixed is +1 or -1. Either, we have some conflict-free dimensions for the original axes or all the intercepts have the same sign. The absolute value of the intercept corresponding to any dependence is therefore less than 1 and so, +1 or -1 is the minimal conflict-free point. □

We now present an algorithm which, given the original dependence matrix D, derives the unimodular transformation matrix whose k'^{th} column corresponds to the sub-optimal direction vector and the corresponding transformed dependence matrix.

Algorithm A4:

Step 1: Initialize U to I, the N by N identity matrix; Initialize m to the maximum starting value equal to the starting value of the first row of D; Mark all dimensions as *not selected*.
Step 2: Look for the dimensions j_x, if any, for which all dependences have the same sign;

If we happen to find none go to step 3;
else
Choose the one with the minimum weight, say the j^{th} and assign $j_r = j$; If j^{th} elements of all dependences are negative, multiply the j_r^{th} column of D and U by -1; Go to step 6.
Step 3: Look for the m-full dimension with the minimum weight, say the j^{th} and assign $j_r = j$; Mark this dimension j as *selected*; If j^{th} elements of all n-starting dependences, for $m \leq n \leq N$ are negative, multiply the j_r^{th} column of D and U by -1.
Step 4: Look for the previously *not selected* dimensions j_x, if any, for which the inequality $a_{j_x}.i_j \geq 1 - a_{j_r}$ is consistent for all the dependences;

If we happen to find none go to step 5;
else
Choose the one with the minimum weight, say the jth; Let s_j be the integer point closest to the origin on the j^{th} dimension satisfying all these inequalities. Add s_j times the j^{th} column to the j_r^{th} column in matrices D and U; Go to step 6.
Step 5: Look for the previously *not selected* m-full dimensions with minimum weight, say the j^{th}; Mark this dimension as *selected*; Let s_j be the integer point closest to the origin on the j^{th} dimension satisfying all these inequalities. Add s_j times the j^{th} column to the j_r^{th} column in matrices D and U; Go to step 4.
Step 6: Interchange the j_r^{th} and k'th columns in matrices D and U; Stop.
End of Algorithm A4

In the worst case, we may require to examine the conflict-freeness of a dimension $N.(N-1)/2$ times. This implies a worst case of $O(N^3)$ division/comparison operations and $O(r.N)$ addition operations. This is a considerable improvement.

Example 14: Considering again the initial dependence matrix of examples 5, we could use algorithm A4 to obtain the sub-optimally transformed dependence matrix for internalization reference 2 and the corresponding tranformation matrix as below

$$D = \begin{bmatrix} 0 & 0 & 1 & -3 \\ 0 & 2 & 2 & -1 \\ 1 & 3 & 3 & 2 \\ 2 & -4 & 1 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

8 Overall Mapping Strategies

Our approach to obtain a sub-optimal mapping consists of determining template transformations from the computed sub-optimal directions and using the simple analytical performance model described below to choose the most appropriate permutation of the template.

8.1 Simplified Performance Model

Let p_a^i signify the parallelism available at level i of the machine which is equivalently the total number of elementary computing agents available at that cluster.

Similarly, let p_m^i signify the upper bound on the parallelism mapped onto level i of the machine. Let the vector $\overline{m} = (m_1, m_2, \ldots, m_N)$. Let U be a unimodular transformation matrix which internalizes the maximum possible number of dependences and whose k'^{th} column is the sub-optimal direction and whose columns are $\overline{u}_1, \overline{u}_2, \ldots, \overline{u}_N$.

For any machine level i, consider the set L_i of all loop dimension indices which the given simple mapping would map on levels 1 to i of the machine in parallel. The bound on the parallelism mapped onto level i of the machine would be $p_m^i = \prod_{j \in L_i} \overline{u}_j.\overline{m}$. We call the ratio $\lceil p_m^i / p_a^i \rceil$, the *parallel slackness* at level i to borrow a terminology from [9] and denote it by r^i. We take the parallel slackness of a level onto which no loop levels are mapped as being equal to the parallel slackness of its next sub-level if any. If no sub-levels exist, it is by default 1.

If t_s represents the total length of computation of the sequential component of the simple mapping, then the total length of computation for the whole space is given by $T_{comp} = t_s.r^M$, where M is the total number of levels of the machine. We assume a upper threshold r_t^i for the r^is and further, we assume that $p_a^M.r_t^M$ is much greater than the total parallelism defined by the simple mapping.

The total volume of communication contributed by dependence vector d_j can easily be shown to be equal to $\prod_{i=1}^{N}(m_i - a_i^j)$ where a_i^j is the ith component of the dependence d_j. The total weight of communication contributed by d_j would be therefore $C_w^j = \prod_{i=1}^{N}(m_i - a_i^j).w_j$, where w_j is the weight of a unit volume of communication corresponding to d_j. Let t_c^i be the cost per unit weight of communication at level i. The net contribution to the cost would however be $t_c^i.f$, where $1-f$ represents the overlap factor of communication by computation. We may thus define the total communication overhead as $T_{comm} = \sum_{j=1}^{r} C_w^j.t_c^{ml(k_j)}.f$ where k_j is the starting value of dependence d_j following the transformation.

The overall execution time of a simple mapping may now be defined as $T = T_{comp} + T_{comm}$.

8.2 The Methodology

Our methododology for mapping may now be described as comprising of the following steps.

1. *Determining the Sub-Optimal Direction*

The steps for this were presented in the last section.

2. *Computing the Transformation Template*

We define the transformation template U'' as a matrix whose columns correspond to the columns of the solution transformation, but in which the sub-optimal direction forms the N^{th} column.

We start with a unimodular matrix U' whose N^{th} column is the sub-optimal direction. We form a dependence matrix D whose rows are the original dependences ordered in the order of decreasing gross communication weight. We determine D', the product of D and U'. We now derive the transformation template U'' and the transformed dependence matrix D'' by applying algorithm A1 to U' and D' without involving the N^{th} columns of the two.

3. *Choosing the Best Mapping and Transformation*

Cost of a simple mapping: Let sm_l be a candidate simple mapping. Let $dl_1, dl_2, \ldots, dl_N$ be the lengths of the greatest diagonals of the space along the dimensions corresponding to columns $1, 2, \ldots, N$ respectively of U''. Let L_i^l represent the set of all loop dimension indices which sm_l would map on levels 1 to i of the machine in parallel. The bound on the parallelism mapped and the parallel slackness for level i of the machine would then be $p_m^i = \prod_{j \in L_i^l} dl_j$ and $r_i = \lceil p_m^i / p_a^i \rceil$ respectively. Thus, $T_{comp}^l = t_s.r^M$, where $t_s = t_{iter}.dl_N$ and M is the total number of levels in the machine.

Let k' be the internalization reference of mapping sm_l. The total communication overhead for the simple mapping sm_l would be $T_{comm}^l = \sum_{j=1}^{r} C_w^j.t_c^{ml(k_j)}.f$ where k_j is now the starting value of dependence d_j in a permutation of D'' obtained by inserting its N^{th} column in the k'^{th} position and pushing columns k' to $N-1$ one position to the right.

Thus, the overall execution time of the simple mapping sm_l is $T^l = T_{comp}^l + T_{comm}^l$.

Choosing the best one: Our final task is the choice of the best among the *candidate* simple mappings, and the corresponding iteration space transformation.

It may easily be shown that the number of candidates corresponding to N loop levels and M machine levels would be $\sum_{i_1=1}^{M} \sum_{i_2=1}^{i_1} \ldots \sum_{i_N=1}^{i_{N-1}} 1$. This is roughly of the complexity $O(N.M^2)$.

We assume that the sequentially mapped level of the loop is the outermost one mapped onto the lowest machine level specified in the simple mapping.

Example 15: Consider a 4–D iteration space with the same initial dependence matrix as in example 15. Let the gross weights of the dependences be respectively $C_w^1 = 200$, $C_w^2 = 100$, $C_w^3 = 50$ and $C_w^4 = 20$. Let the lengths of the dimensions in the original space be $m_1 = 100$, $m_2 = 100$, $m_3 = 20$ and $m_4 = 20$.

The simple mapping candidates for mapping the above on a three level machine are {(1, 1, 1, 1), (2, 1, 1, 1), (2, 2, 1, 1), (2, 2, 2, 1), (2, 2, 2, 2), (3, 1, 1, 1), (3, 2, 1, 1), (3, 2, 2, 1), (3, 2, 2, 2), (3, 3, 1, 1), (3, 3, 2, 1), (3, 3, 2, 2), (3, 3, 3, 1), (3, 3, 3, 2), (3, 3, 3, 3)}. Let us suppose the parallelism available for the machine considered are described by $p_a^1 = 100$, $p_a^2 = 1000$ and $p_a^1 = 4000$. Also, let the costs associated with a unit weight of communication be given as $t_c^1 = 10$ units,

$t_c^2 = 20$ units and $t_c^3 = 50$ units. Finally, let the time for one iteration, $t_{iter} = 100$ units.

The template dependence and transformation may easily be derived as in example 17 to be the following:

$$U'' = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} D'' = \begin{bmatrix} 0 & 0 & 3 & 1 \\ 0 & 2 & 1 & 2 \\ 1 & 3 & -2 & 3 \\ 2 & -4 & -1 & 1 \end{bmatrix}$$

The greatest diagonals are computed to be $dl_1 = 100$, $dl_2 = 100$, $dl_3 = 20$ and $dl_4 = 220$. It may be verified that the best simple mapping corresponds to the tuple (3, 1, 1, 1). For the tuples that precede this one, the time for computation dominates quite significantly and for the onesthat follow it, the computation time remains the same but the communication time increases steadily. The required transformation corresponding to the chosen mapping is the following:

$$U = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

8.3 Overall Complexity of the Method

It can be seen that T_{comp} and T_{comm} involve $O(M^2)$ and an overall worst case $O(N.M^2.r)$ multiplication operations respectively. Determining the minimum T clearly involves at worst $O(N.M^2)$ comparisons. So the overall complexity is clearly bounded by $O(N.M^2.r)$ multiplication operations.

9 Conclusion

We have presented in this paper a computationally efficient methodology for the problem of mapping nested loops on hierarchical parallel machines in the context of a systematic and general theory of unimodular loop transformations presented by us in our recent work [6, 7]. We have started by describing optimal and sub-optimal directions in the iteration space and the derivation of a unimodular transformation involving a chosen direction. We have suggested a polynomial time method for computing a sub-optimal direction in the iteration space that would give us a transformation template and candiadte mappings. We have adopted a simple parameterized hierarchical machine model which is used for choosing the best mapping among these and an associated transformation.

References

[1] U. Banerjee, Dependence Analysis for Supercomputing, Kluwer Academic Publishers, 1988.

[2] U. Banerjee, Unimodular Transformations of Double Loops, proceedings of the third Workshop on Programming Languages and Compilers for Parallel Computing, Irvine, California, August 1-3, 1990.

[3] A. Basu, S. Srinivas, K.G. Kumar and A. Paulraj, A Model for Performance Prediction on Message Passing Multiprocessors Achieving Concurrency by Domain Decomposition, proceedings of CONPAR'90, Zurich, LNCS - 457, pp 75-85, September, 1990.

[4] M.L. Dowling, Optimal Code Parallelization using Unimodular Transformations, Parallel Computing, 16, pp 157-171, 1990.

[5] D. Kulkarni, K.G. Kumar, A. Basu, and A. Paulraj, Loop Partitioning for Distributed Memory Multiprocessors as Unimodular Transformations, Proceedings of the ACM International Conference on Supercomputing, June, 1991.

[6] K.G. Kumar, D. Kulkarni and A. Basu, Generalized Unimodular Loop Transformations for Distributed Memory Multiprocessors, Proceedings of the International Conference on Parallel Processing, Illinois, August, 1991.

[7] K.G. Kumar, D. Kulkarni and A. Basu, Mapping Nested Loops on Hierarchical Parallel Machines using Unimodular Transformations, Research Report, FG-TR-022, Centre for Development of Advanced Computing, Bangalore, India, December, 1991.

[8] L. Lamport, The Parallel Execution of DO Loops, Communications of the ACM, 17(2), 1974.

[9] L.G. Valiant, Optimally Universal Parallel Computers, Philosophical Transactions of Royal Society London, Series A, 326, pp 373-376, 1988.

[10] M. E. Wolf and M. S. Lam, A Loop Transformation Theory and an Algorithm to Maximize Parallelism, IEEE Transactions on Parallel and Distributed Systems, Vol.2, No.4, pp 452-471, October, 1991.