Tiling Multidimensional Iteration Spaces for Multicomputers

J. Ramanujam*
Dept. of Electrical and Computer Engineering
Louisiana State University, Baton Rouge, LA 70803–5901, USA.
Email: jxr@max.ee.lsu.edu

P. Sadayappan
Dept. of Computer and Information Science
The Ohio State University, Columbus, OH 43210–1277, USA.
saday@cis.ohio-state.edu

Abstract

This paper addresses the problem of compiling perfectly nested loops for multicomputers (distributed memory machines). The relatively high communication startup costs in these machines renders frequent communication very expensive. Motivated by this, we present a method of aggregating a number of loop iterations into *tiles* where the tiles execute atomically – a processor executing the iterations belonging to a tile receives all the data it needs before executing any one of the iterations in the tile, executes all the iterations in the tile and then sends the data needed by other processors. Since synchronization is not allowed during the execution of a tile, partitioning the iteration space into tiles must not result in deadlock. We first show the equivalence between the problem of finding partitions and the problem of determining the cone for a given set of dependence vectors. We then present an approach to partitioning the iteration space into deadlock-free tiles so that communication volume is minimized. In addition, we discuss a method to optimize the size of tiles for nested loops for multicomputers. This work differs from other approaches to tiling in that we present a method of optimizing grain size of tiles for multicomputers.

1 Introduction

Multicomputers (distributed memory message passing machines) are attractive due to their scalability, flexibility and performance but suffer from a lack of adequate programming support tools. Parallelizing compilers for such machines have received great attention recently. This paper addresses the problem of compiling nested loops for multicomputers. The message-passing paradigm employed in these machines makes program development significantly different from that for conventional shared memory machines. It requires that processors keep track of data distribution and that they communicate with each other by explicitly moving data around in messages. In addition, with current technology, the communication overhead is still at least an order of magnitude larger

*The author's research was supported in part by a grant from Louisiana Educational Quality Support Fund through contract LEQSF (1991-94)-RD-A-09.

than the corresponding computation. The relatively high communication startup costs in these machines renders frequent communication very expensive. This in turn calls for careful partitioning of the problem and efficient scheduling of the computations as well the communication.

Motivated by this concern, we present a method of partitioning nested loops and scheduling them on multicomputers with a view to "matching" (optimizing) the granularity of the resultant partitions. Nested loops are common in a large number of scientific codes and most of the execution time is spent in loops. For many nested loops, the amount of computation involved in executing a single iteration may not be enough to offset the communication startup overhead – processors may spend more time in interprocessor communication than in executing the computations. As a result, we present a method of aggregating a number of loop iterations into *tiles* where the tiles execute atomically – a processor executing the iterations belonging to a tile receives all the data it needs before executing any one of the iterations in the tile, executes all the iterations in the tile and then sends the data needed by other processors. Since synchronization is not allowed during the execution of a tile, partitioning the iterations into tiles must not result in deadlock. Given a perfectly nested loop to be executed on a multicomputer with given execution and communication costs, the tile shape and size must be chosen so as to optimize the performance; in addition, the tiles must be assigned to processors to minimize communication costs and reduce processor idle times. We first show the equivalence between the problem of finding partitions and the problem of determining the cone for a given set of dependence vectors. We then present an approach to partitioning the iteration space into deadlock-free tiles so that communication volume is minimized.

There have been tremendous advances in recent years in the area of intelligent routing mechanisms and efficient hardware support for interprocessor communication in multicomputers. These techniques reduce the communication overhead incurred by processor nodes that are incident on a path between a pair of communicating processors – this in turn greatly simplifies the task of mapping the partitions onto the processors. Therefore, we do not address the mapping problem here.

This paper presents a new approach to determine valid tiles, and to minimize communication volume as a result of tiling. In addition, we discuss a method to optimize the tile size for execution of nested loops. We begin with a discussion of background material and related work in the next section. Sections 3 and 4 show the equivalence of the problems of finding deadlock-free partitions and the problem of determining the set of extreme vectors for a given set of dependence vectors and present a linear programming formulation. In sections 5 and 6, we present a method based on linear programming for minimizing communication volume for K-nested loops. Section 7 presents a technique to optimize tile size; we summarize the work and discuss further avenues of research in section 8.

2 Background

2.1 Dependences

Good and thorough parallelization of a program critically depends on how precisely a compiler can discover the data dependence information [3, 6, 22, 38, 40, 41]. These dependences imply precedence constraints among computations which must be satisfied for a correct execution. Many algorithms exhibit regular data dependences, *i.e.* certain dependence patterns occur repeatedly over the duration of the computation.

Let S_x and S_y be two statements (not necessarily distinct) enclosed by perfectly nested loops. Data dependences determine which iterations of the loops can be executed concurrently. A `flow dependence` exists from statement S_x to statement S_y if S_x computes and writes a value that is subsequently (in sequential execution) read by S_y. A flow dependence implies that instances of S_x and S_y must execute as if some of the nest levels must be executed sequentially. Note that not all loop nest levels need to contribute to the dependence. An `anti-dependence` exists between S_x and S_y if S_x reads a value that is subsequently modified by S_y. An `output dependence` exists between S_x and S_y if S_x writes a value which is subsequently written by S_y.

2.2 Iteration Space Graph (ISG)

Dependence relations are often represented in *Iteration Space Graphs* (ISG's). For a K-nested loop with index set $(I_1, I_2, \ldots, I_K)$, the nodes of the ISG are points of a K-dimensional discrete Cartesian space; a directed edge exists between the iteration defined by $\vec{I_1}$ and the iteration defined by $\vec{I_2}$ whenever a dependence exists between statements in the loop constituting the iterations $\vec{I_1}$ and $\vec{I_2}$.

Many dependences that occur in practice have a constant distance in each dimension of the iteration space. In such cases, the vector $\vec{d} = \vec{I_2} - \vec{I_1}$ is called the `distance vector`. An algorithm has a number of such dependence vectors; the dependence vectors of the algorithm are written collectively as a `dependence matrix` $D = [\vec{d_1}, \vec{d_2}, \ldots, \vec{d_n}]$. In addition to the three types of dependence mentioned above, there is one more type of dependence known as `control dependence`. A control dependence exists between a statement with a conditional jump and another statement if the conditional jump statement controls the execution of the other statement. Of these data dependences, only flow dependence is inherent to the computation. In multicomputers, where data transfer and synchronization are both achieved through message passing, flow dependences correspond to communication; also, anti and output dependences can be removed through standard transformations [44]. Hence, we discuss only flow dependences. Dependence analysis is discussed in [2, 3, 6, 38, 40, 41].

2.3 Extreme vectors

Based on results in number theory and integer programming [4, 36], given a set of n distinct dependence vectors ($n \geq K$), one can find a set of K vectors say $\vec{e_1}, \ldots, \vec{e_K}$ such that any dependence vector $\vec{d_i}$, $i = 1, \ldots, n$ can be expressed as a nonnegative linear combination of vectors $\vec{e_1}, \ldots, \vec{e_K}$, *i.e*

$$c_i \vec{d_i} = \sum_{j=1}^{K} a_{i,j} \vec{e_j}$$

where $i = 1, \ldots, n$ and $a_{i,j} \geq 0$ and $c_i > 0$. The set of vectors $e_1, \ldots, e_K$ are referred to as `extreme vectors`. Extreme vectors are not necessarily unique and this fact is used to advantage in the choice of tile shapes (partitions) in later sections.

2.4 Related work on compiling programs for multicomputers

A number of research groups have focused on compiling programs for multicomputers augmented with user-defined data decomposition [5, 8, 14, 18, 19, 20, 21, 43]. Balasundaram and others [5] at

Rice are working on interactive parallelization tools for multicomputers that provide the user with feedback on the interplay between data decomposition and task partitioning on the performance of programs. Koelbel et al. [18, 19, 20] address the problem of automatic process partitioning of programs written in a language called Kali along with user-specified data partitions. A group led by Kennedy at Rice University [8, 14] is studying similar techniques for compiling a version of Fortran enhanced with data decomposition specifications (called Fortran D) for distributed memory machines such as Intel iPSC/860; they show how some transformations could be used to improve performance. Rogers and Pingali [27, 28] present a method which, given a sequential program and its data partition, performs task partitions to enhance locality of references. Zima et al. [21, 43] discuss SUPERB, an interactive system for semi-automatic transformation of FORTRAN 77 programs into parallel programs for the SUPRENUM machine, a loosely-coupled hierarchical multiprocessor.

2.5 Related work in tiling and other memory optimizations

Callahan et al. [9] discuss loop unroll-and-jam in the context of register allocation for arrays. Chen et al. [10, 11] describe how Crystal, a functional language, addresses the issue of programmability and performance of parallel supercomputers. Gallivan et al. [12] and Gannon et al. [13] discuss problems associated with automatically restructuring data so that it can be moved to and from local memories in the case of hierarchical shared memory machines. They present a series of theorems that enable one to describe the structure of disjoint sub-lattices accessed by different processors, and how to use this information to make "correct" copies of data in local memories, and how to write the data back to the shared address space when the modifications are complete. Neither paper addresses the automatic derivation of transformations. In the context of hierarchical shared memory systems, Irigoin and Triolet [15] present a method, which divides the iteration space into clusters referred to as *supernodes*, with the goals of vector computation within a node, data re-use within a partition and parallelism between partitions. The procedure works from a new data dependence abstraction called dependence cones. The paper does not address the problem of automatically choosing partitions. King and Ni [16, 17] discuss the grouping of iterations for execution on multicomputers; they present a number of conditions for valid tile formation in the case of two-dimensional iteration spaces. Mace [24] proves that the problem of finding optimal data storage patterns for parallel processing (the "shapes" problem) is NP-complete, even when limited to one- and two-dimensional arrays; in addition, efficient algorithms are derived for the shapes problem for programs modeled by a directed acyclic graph (DAG) that is derived by series-parallel combinations of tree-like subgraphs. Schreiber and Dongarra [35] discuss a method of choosing a subset of the dependence vectors as extreme vectors for tiling. Wolf and Lam [37] propose an algorithm for improving data locality of a loop nest via compound transformations. Wolfe [39, 42] discusses a technique referred to as *iteration space tiling* which divides the iteration space of loop computations into *tiles* (or blocks) of some size and shape so that traversing the tiles results in covering the whole space. Optimal tiling for a memory hierarchy finds tiles such that all data for a given tile will fit into the highest (fastest) level of memory hierarchy and exhibit high data reuse, reducing total memory traffic. In addition, in the context of loop unrolling, partitioning of iteration space graphs is discussed by Nicolau [25]. None of these approaches attempt explicit minimization of communication.

3 Tiling of Multidimensional Iteration Spaces

As mentioned above, the relatively high communication startup costs in distributed memory machines renders frequent communication very expensive. For example, the message startup time on Intel iPSC/860 is about 80 μs and it takes 1 μs to transfer a double precision floating point number neighboring nodes (once communication is set up); access to local memory takes negligible time. Hence, we focus on collecting iterations together into *tiles* where each tile executes atomically with no intervening synchronization or communication; as a result, we are able to amortize the high message startup cost over larger messages at the expense of processor idle time.

Each tile defines an atomic unit of computation comprising of a number of iterations. Thus no synchronization or communication must be necessary during the execution of the tile. This imposes a constraint that the partitioning of the iteration space into tiles does not result in deadlock, *i.e.,* there should be dependence cycles among the tiles. To keep code generation simple, it is also necessary that all tiles are identical in shape and size except near the boundaries of the iteration space.

3.1 Equivalence of tiling planes and extreme vectors

Tiles in K-dimensional (K-D) spaces are defined by K families of parallel hyperplanes (or planes) each of which is a $K-1$ dimensional hyperplane. Tiles so defined are parallelopipeds (except for those near the boundary of the iteration space) and each tile is a K-D subset of the iteration space. Thus, the shape of the tiles is defined by the families of planes and the size of the tiles is defined by the distance of separation between adjacent pairs of parallel planes in each of the K families.

Example 1:
for $i = 2$ to N do
  for $j = 2$ to $N - 1$ do
    $A[i, j] \leftarrow A[i - 1, j + 1] + A[i - 1, j] + A[i, j - 1]$
  endfor
endfor

Figure 1(a) shows the iteration space graph defined by the loop in example 1. The distance vectors are $(1, -1)$, $(1, 0)$ and $(0, 1)$. Tiles in this ISG are defined by 2 families of lines. Figure 1(b) shows tiles of size 2×2 in this ISG. In the case of arbitrary dependence graphs, clustering the tasks into atomic collections might result in deadlock [33, 34]. Sarkar [33] has formulated the condition for the absence of deadlock in this case as the *convexity* constraint and has presented a heuristic that derives deadlock-free convex partitions of arbitrary DAG's. Irigoin and Triolet [15] present these conditions for iteration space graphs.

For tiles in K-D iteration spaces to be legal, dependence vectors crossing a boundary between any pair of tiles must all cross from a given tile to the other i.e. the source iterations of all the dependence vectors must be in the same tile and their sink in the other. In the ensuing discussion, we assume that the dependence matrix for a K-nested loop has rank K and that the number of dependence vectors is n. A vector $\vec{h_i} = (h_{i,1}, h_{i,2}, \ldots, h_{i,K})$ defines a family of hyperplanes (in a K-dimensional space)

$$h_{i,1}x_1 + h_{i,2}x_2 + \cdots + h_{i,K}x_K = c$$

Figure 1: ISG and tiling for example 1

for various values of c.

Each tile boundary is defined by a vector perpendicular to it. In K dimensions, any K vectors $\vec{h}_1, \ldots, \vec{h}_K$ (where $\vec{h}_i$ is the vector perpendicular to the ith boundary) define legal tiles if

$$\vec{h}_i \cdot \vec{d}_j \geq 0, \qquad i = 1, \ldots, K \tag{1}$$

for all $\vec{d}_j$ in the set of dependence vectors. One can also define an equivalent condition:

$$\vec{h}_i \cdot \vec{d}_j \leq 0, \qquad i = 1, \ldots, K. \tag{2}$$

Each of the $K-1$ dimensional planes is a boundary whose perpendicular is one of the $\vec{h}_i$. Condition (1) is from [15].

To form K-dimensional tiles from a K-dimensional iteration space (assuming that the dependence matrix D (a $K \times n$ matrix) whose columns are dependence vectors is of rank K), the vectors perpendicular to the K tile boundaries must be linearly independent. Let H denote the $K \times K$ matrix whose rows are the vectors $\vec{h}_1, \ldots, \vec{h}_K$. Thus, the rank of H must be K in order to define K-D tiles. We restate condition (1) using matrix notation which lets us describe succinctly the relation between tile boundaries (vectors perpendicular to them) and extreme vectors and cast the problem of finding tile boundaries in terms of finding extreme vectors and *vice-versa*.

Condition (1) states that in K dimensions, any K vectors $\vec{h}_1, \ldots, \vec{h}_K$ (where $\vec{h}_i$ is the vector perpendicular to the ith boundary) define legal tiles if

$$\vec{h}_i \cdot \vec{d}_j \geq 0, \qquad i = 1, \ldots, K$$

for all $\vec{d}_j, j = 1, \ldots, n$ in the set of dependence vectors. In matrix notation, let $D^+ = HD$. The tiling condition implies that, for tiles to be legal

$$D^+_{ij} \geq 0 \qquad i = 1, \ldots, K \qquad j = 1, \ldots, n. \tag{3}$$

All entries in D^+ are nonnegative. Again using matrix notation, if tiles defined by $\vec{h}_i$ are K-dimensional tiles, then $\vec{h}_i$ must be linearly independent; this means H must have rank K i.e., H must be nonsingular. As a result, the inverse of matrix H must exist. Thus, the dependence matrix D can written as

$$D = H^{-1}D^+, \qquad \text{where} \quad D^+_{ij} \geq 0, \quad i = 1, \ldots, K \quad j = 1, \ldots, n \tag{4}$$

Since every entry in D^+ is nonnegative, the above relation implies that every dependence vector (every column of the matrix D) can be expressed as *nonnegative* linear combinations of the columns of the matrix H^{-1}. Thus the columns of H^{-1} are extreme vectors (by definition) for the set of dependence vectors $\vec{d}_j$ which constitute the columns of matrix D. Thus, tiles are either defined by a set of extreme vectors (for a given set of dependence vectors) or by a set of valid tiling planes given by a set of vectors perpendicular to the tiling planes. We note here that H is not unique. In section 5, we show a linear programming formulation of the problem of finding H which is lower triangular with unit diagonal. An integer matrix with determinant ± 1 is referred to as a *unimodular* matrix [6]; the inverse of a unimodular matrix is also a unimodular matrix. The algorithm in section 5 finds a unimodular H; hence H^{-1} is also integral.

4 Extreme vectors for K-dimensional iteration spaces

In 2-dimensional iteration spaces, the extreme vectors are a subset of the dependence vectors. The algorithm described below finds the extreme vectors in $O(n)$ time where n is the number of dependence vectors. Let the components of each dependence (distance) vector $\vec{d_i}$ be $(d_{i,1}, d_{i,2})$ where $d_{i,1}$ is the distance along the outer loop and $d_{i,2}$ is the distance along the inner loop. Find the ratio $r_i = \frac{d_{i,2}}{d_{i,1}}$ for all the dependence vectors (including the signs). The vectors with the highest and lowest values (one vector for each) form a set of extreme vectors. Note that if $d_{i,1} = 0$ for any $\vec{d_i}$, then that vector is an extreme vector. If there is more than one vector with $d_{i,1} = 0$, we choose the vector with the smallest value of $d_{i,2}$ from among those. The largest and the smallest values among the r_i are found in $O(n)$ time.

In the case of higher dimensional iteration spaces, a subset of the dependence vectors need not themselves be the extreme vectors. This is illustrated through the following example:

Example 2:

for $i = 2$ to N do
  for $j = 2$ to N do
    for $k = 2$ to $N - 1$ do
      $A[i, j, k] \leftarrow A[i-1, j, k] + A[i, j-1, k] + A[i, j, k-1] + A[i-1, j-1, k+1]$
    endfor
  endfor
endfor

The loop has 4 distance vectors $\vec{d_1} = (1, 0, 0)$, $\vec{d_2} = (0, 1, 0)$, $\vec{d_3} = (0, 0, 1)$ and $\vec{d_4} = (1, 1, -1)$. For any choice of three of the distance vectors as extreme vectors, the fourth vector cannot be expressed as a nonnegative linear combination of the three vectors *i.e.,* the fourth does not lie in the positive hull of the other three. In this case, an extreme vector set consists of the following vectors $(0, 0, 1)$, $(1, 0, -1)$ and $(0, 1, 0)$ of which only two are dependence (distance) vectors.

Contrast the above with the following example, where three of the dependence vectors form a set of extreme vectors:

Example 3:

for $i = 2$ to N do
  for $j = 2$ to $N - 1$ do
    for $k = 2$ to $N - 1$ do
      $A[i, j, k] \leftarrow A[i-1, j, k] + A[i, j-1, k] + A[i, j, k-1] + A[i-1, j+1, k+1]$
    endfor
  endfor
endfor

There are 4 distance vectors $\vec{d_1} = (1, 0, 0)$, $\vec{d_2} = (0, 1, 0)$, $\vec{d_3} = (0, 0, 1)$ and $\vec{d_4} = (1, -1, -1)$. We note that $\vec{d_1} = \vec{d_2} + \vec{d_3} + \vec{d_4}$. Therefore, $\vec{d_2}$, $\vec{d_3}$ and $\vec{d_4}$ form a set of extreme vectors.

4.1 Valid extreme vectors

In imperative languages like Fortran, C etc., a dependence distance vector is a valid vector if its first nonzero component is positive. Thus in 2 dimensions, for any given set of distance vectors, the columns of the matrix

$$\begin{bmatrix} 1 & 0 \\ -e_1 & 1 \end{bmatrix}$$

where $e_1 \geq 0$ will always be a set of valid extreme vectors for sufficiently large e_1. In three dimensions, the columns of the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ -e_1 & 1 & 0 \\ 0 & -e_2 & 1 \end{bmatrix}$$

will be a set of extreme vectors.

Consider example 2 again. There are 4 distance vectors $\vec{d_1} = (1, 0, 0)$, $\vec{d_2} = (0, 1, 0)$, $\vec{d_3} = (0, 0, 1)$ and $\vec{d_4} = (1, 1, -1)$. In this case,

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

is a valid set of extreme vectors. In general, in K-dimensions, we can always find a matrix E of the form

$$E = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -e_1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -e_2 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & 1 & 0 \\ 0 & 0 & \cdots & \cdots & -e_{K-1} & 1 \end{bmatrix}$$

where $e_i \geq 0$; the columns of the matrix E form a valid set of extreme vectors *i.e.,* every dependence vector can be expressed as nonnegative linear combinations of the columns of E. Such a matrix E is nonsingular and the entries in E^{-1} are:

$$E_{ij}^{-1} = \begin{cases} 0 & \text{if } j > i \\ 1 & \text{if } j = i \\ \prod_{k=j}^{i-1} e_k & \text{if } j < i \end{cases}$$

The matrix E^{-1} is one valid H, *i.e.,* the rows of E^{-1} define legal (deadlock-free) tiles. For example, in 4-dimensional iteration spaces, the matrix

$$E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -e_1 & 1 & 0 & 0 \\ 0 & -e_2 & 1 & 0 \\ 0 & 0 & -e_3 & 1 \end{bmatrix}$$

is a valid set of extreme vectors. The inverse of E is given below:

$$E^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ e_1 & 1 & 0 & 0 \\ e_1 e_2 & e_2 & 1 & 0 \\ e_1 e_2 e_3 & e_2 e_3 & e_3 & 1 \end{bmatrix}$$

Since E is the set of extreme vectors, the dependence matrix D can written as

$$D = ED^+, \qquad \text{where } D^+_{ij} \geq 0, \quad i = 1, \ldots, K \quad j = 1, \ldots, n \tag{5}$$

All entries in D^+ are nonnegative. Again using matrix notation, since E^{-1} exists, D^+ can be written as

$$D^+ = E^{-1}D$$

and

$$\left(E^{-1}D\right)_{ij} \geq 0 \qquad i = 1, \ldots, K \quad j = 1, \ldots, n$$

which can be written as

$$E_i^{-1} \cdot D_j \geq 0 \qquad i = 1, \ldots, K \quad j = 1, \ldots, n$$

where E_i^{-1} is the ith row of E^{-1} and D_j is the jth column of D. Thus the rows of E^{-1} are the perpendiculars to the tile boundaries and these rows are linearly independent.

Synchronization (and the accompanying data transfer) between tiles is required whenever there is a dependence between iterations which belong to different tiles. Since tiles are separated by tile boundaries (defined by $\vec{H}_i$), the dependence vector in this case must cross the tile boundary between the tiles. A nonzero entry in D^+ say D^+_{ij} implies that communication is incurred due to the jth dependence vector poking the ith tile boundary. The amount communication across a tile boundary defined by the ith row of H is a function of the sum of the entries in the ith row of the transformed dependence matrix D^+.

5 Tiling for minimal communication

Based on results from the previous sections, we formulate the problem of finding the tiling planes as (a linear programming problem) that of finding a transformation H such that

$$\sum_{k=1}^{K} H_{i,k} D_{k,j} \geq 0 \qquad i = 1, \ldots, K \quad j = 1, \ldots, n$$

with the constraint that

$$H_{i,k} = 0 \qquad \text{if } k > i.$$

The rows of H are the perpendiculars to the tiling planes and the columns of H^{-1} are the spanning (extreme) vectors. When the diagonal entries in H are 1 and the lower triangular entries are integers, an $l \times l \times \cdots \times l$ tile is defined as the region in the iteration space enclosed by the set of feasible solutions to:

$$\begin{array}{rcll} \vec{h}_i \cdot \vec{x} & \geq & c_i & \text{for } i = 1, \ldots, K \\ \vec{h}_i \cdot \vec{x} & < & c_i + l & \text{for } i = 1, \ldots, K. \end{array}$$

The communication volume for such an $l \times l \times \cdots \times l$ tile is:

$$\text{Communication volume} = l^{K-1} \sum_{i=1}^{K} \sum_{j=1}^{n} \sum_{k=1}^{K} H_{i,k} D_{k,j} \tag{6}$$

and hence is proportional to $\sum_{i=1}^{K}\sum_{j=1}^{n}\sum_{k=1}^{K} H_{i,k}D_{k,j}$. Thus, we formulate the problem of finding a set of valid tiling planes that minimizes communication volume as following linear programming problem: Find a $K \times K$ matrix H such that

$$\sum_{i=1}^{K}\sum_{j=1}^{n}\sum_{k=1}^{K} H_{i,k}D_{k,j}$$

is minimized subject to the constraints that

$$\begin{aligned}\sum_{k=1}^{K} H_{i,k}D_{k,j} &\geq 0 \quad i = 1,\ldots,K \quad j = 1,\ldots,n\\ H_{i,k} &= 0 \text{ if } k > i\\ H_{i,k} &= 1 \text{ if } k = i.\end{aligned}$$

A K-dimensional tile of size $r_1 l \times r_2 l \times \cdots \times r_K l$ is defined as the region in the iteration space enclosed by the set of feasible solutions to:

$$\begin{aligned}\vec{h_i}\cdot\vec{x} &\geq c_i \qquad \text{for } i = 1,\ldots,K\\ \vec{h_i}\cdot\vec{x} &< c_i + r_i l \qquad \text{for } i = 1,\ldots,K\end{aligned}$$

The values $r_1, r_2, \ldots, r_K$ are the aspect ratios of such a tile. The communication volume for an $r_1 l \times r_2 l \times \cdots \times r_K l$ tile is:

$$\text{Communication volume} = l^{K-1}\sum_{i=1}^{K}\prod_{q=1,q\neq i}^{K} r_q \sum_{j=1}^{n}\sum_{k=1}^{K} H_{i,k}D_{k,j} \tag{7}$$

In case there are rational entries in lower triangular H, the matrix H with normalized rows(all entries in a row of H are integers and their $gcd = 1$), has a determinant not equal to 1; for a discussion of the expression for communication volume in such cases, the reader is referred to the next section. For the rest of the paper, we will assume that $r_1 = r_2 = \cdots = r_K = 1$ unless otherwise stated.

The loop nest in example 2 has the following dependence matrix,

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

The problem of finding communication minimal tiling planes (or extreme vectors) is that of finding a matrix H of the form

$$H = \begin{bmatrix} 1 & 0 & 0\\ a & 1 & 0\\ b & c & 1\end{bmatrix}$$

such that

$$D^+ = HD = \begin{bmatrix} 1 & 0 & 0 & 1\\ a & 1 & 0 & a+1\\ b & c & 1 & b+c-1\end{bmatrix}$$

has nonnegative entries whose sum is minimized. Thus, the problem is:
Minimize $2a + 2b + 2c + 4$ subject to:

$$\begin{aligned} a &\geq 0 \\ a + 1 &\geq 0 \\ b &\geq 0 \\ c &\geq 0 \\ b + c - 1 &\geq 0 \end{aligned}$$

There are two solutions; the first is $a = 0, b = 1, c = 0$ for which the transformation H is

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

The extreme vectors are columns of H^{-1} given by:

$$H^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}.$$

The second solution is $a = 0, b = 0, c = 1$ for which the transformation H is

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

for which the extreme vectors are columns of H^{-1} given by:

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

In either case, we note that two of the three columns of H^{-1} are dependence vectors.

Next we discuss a more difficult example, where the dependence vectors form two separate cones. Let the set of dependence vectors be given by:

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

Thus we need to find a matrix H

$$H = \begin{bmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{bmatrix}$$

such that $5a + 5b + 5c + 12$ is minimized subject to

$$
\begin{array}{rcl}
a & \geq & 0 \\
2a + 1 & \geq & 0 \\
a + 2 & \geq & 0 \\
b & \geq & 0 \\
b + 1 & \geq & 0 \\
c & \geq & 0 \\
c + 1 & \geq & 0 \\
2b + c & \geq & 0 \\
b + 2c & \geq & 0
\end{array}
$$

The minimal solution is $a = b = c = 0$ which means

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In the next section, we discuss communication minimal extreme vectors in two dimensional iteration spaces.

6 Extreme vectors for minimal communication in 2-D spaces

The matrices H derived in section 5 are unimodular. An important consequence of unimodularity is that tiles can be derived by applying a sequence of elementary loop transformations, namely loop interchange, skewing and reversal [7, 31, 37] followed by strip mining [42]. Non-unimodular loop transformations [32] can be used to tiles from H matrices whose determinant $\neq 1$; the procedure described in [32] is complex and is based on deriving appropriate step sizes for the loops in the transformed loopnest.

In the case of 2-dimensional iteration spaces, we presented a method of finding extreme vectors in [29, 30] – the transformation matrix is not necessarily lower triangular and the determinant need not be 1 in such cases. For example, when the dependence vectors are given by:

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

the extreme vectors are $(1, -1)$ and $(1, 1)$ in which case a transformation matrix H is:

$$H = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

whose determinant is 2. From a transformational view, we need to find a nonsingular matrix H

$$H = \begin{bmatrix} 1 & h_1 \\ h_2 & 1 \end{bmatrix}$$

such that

$$D^+ = HD = \begin{bmatrix} 1 & 1-h_1 & 1+h_1 & 2+h_1 \\ h_2 & h_2-1 & h_2+1 & 2h_2+1 \end{bmatrix}$$

has nonnegative entries. The optimal solutions for linear programming problems occur at the *simplex* vertices or corners of the feasible region. Figure 2 shows the constraints and the feasible region. The corners of the feasible region for the above set of constraints are: $h_1 = -1, h_2 = 1$ and $h_1 = 1, h_2 = 1$. Since we require that H be nonsingular, its determinant must be nonzero *i.e.,* $1 - h_1h_2 \neq 0$, the second simplex point $h_1 = 1, h_2 = 1$ cannot be considered. Therefore, the solution is $h_1 = -1, h_2 = 1$. In general, we need to consider only those simplex vertices where the matrix H is nonsingular. The choice among those depends on the communication volume which is the topic of the ensuing discussion. When a set of tiling planes are given by $\vec{h_1} = (a, b)$ and $\vec{h_2} = (c, d)$ (a, b, c, d are integers) where the determinant of $H = ad - bc \neq 0$ every intersection of the 2 families of lines need not be a point in the iteration space. $\vec{h_1}$ defines a family of lines $ax_1 + by = c$ for different values of c. From theorem 5.4.2 from Banerjee's book [6] (page 81), it follows that for a given value of c, the equation has integer solution if and only if $gcd(a, b)$ divides c. For a given value of c say k_1, this means that there is at least one point in the iteration space that lies on the line $ax + by = k_1$. If for every value of c, there must be a point in the iteration space that lies on one of the family of lines, then $gcd(a, b)$ must divide c for all values of c – this implies that $gcd(a, b) = 1$. Let (x_0, y_0) be a point in the iteration space which lies on the intersection of lines $ax + by = k_1$ and $cx + dy = k_2$ for some specific values k_1 and k_2. We assume that $gcd(a, b) = 1$ and $gcd(c, d) = 1$ and that $ad - bc \neq 0$. Let the matrix H refer to

$$H = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Since $det(H) = ad - bc \neq 0$, H^{-1} exists and is:

$$H^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Since $ax_0 + by_0 = k_1$ and $cx_0 + dy_0 = k_2$, we can write

$$H \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}$$

We need to find the minimum value of s_1 such that the intersection of $ax + by = k_1 + s_1$ and $cx + dy = k_2$ is a point in the iteration space *i.e.* find the minimum value of s_1 such that one can find a point (x, y) (where x and y are integers) which satisfies

$$H \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} k_1 + s_1 \\ k_2 \end{bmatrix}$$

This can be written as:

$$H \begin{bmatrix} x \\ y \end{bmatrix} = H \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} s_1 \\ 0 \end{bmatrix}$$

or as

$$H \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} = \begin{bmatrix} s_1 \\ 0 \end{bmatrix}$$

Figure 2: Feasible solutions to the problem of finding communication minimal tiling planes

Therefore,

$$\begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} = H^{-1} \begin{bmatrix} s_1 \\ 0 \end{bmatrix} =$$
$$\frac{1}{det(H)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \begin{bmatrix} s_1 \\ 0 \end{bmatrix} = \frac{s_1}{det(H)} \begin{bmatrix} d \\ -c \end{bmatrix}$$

Since $gcd(c, d) = 1$ and at most one of $x - x_0$ and $y - y_0$ can be zero, it follows that s_1 has to be a multiple of $det(H)$ and hence the smallest value of s_1 is $det(H)$.

When $det(H) \neq 1$ for a nonsingular matrix H, an $l \times l$ tile is defined by the set of feasible solutions to

$$\begin{aligned} \vec{h_1} \cdot \vec{x} &\geq c_1 && (8) \\ \vec{h_1} \cdot \vec{x} &< c_1 + l && (9) \\ \vec{h_2} \cdot \vec{x} &\geq c_2 && (10) \\ \vec{h_2} \cdot \vec{x} &< c_2 + l\, det(H) && (11) \end{aligned}$$

or by the set of feasible solutions to

$$\begin{aligned} \vec{h_1} \cdot \vec{x} &\geq c_1 && (12) \\ \vec{h_1} \cdot \vec{x} &< c_1 + l\, det(H) && (13) \\ \vec{h_2} \cdot \vec{x} &\geq c_2 && (14) \\ \vec{h_2} \cdot \vec{x} &< c_2 + l. && (15) \end{aligned}$$

When $det(H)$ is prime, the above are the only two possibilities. When $det(H)$ is not prime, there are other ways of defining an $l \times l$ tile. This generalizes easily to higher dimensional iteration spaces.

Let R_i be the sum of the entries in the ith row of the transformed dependence matrix D^+, *i.e.,*

$$R_i = \sum_{j=1}^{n} D^+_{i,j}$$

In the case of two dimensional iteration spaces, the communication volume incurred due to tiles defined by H is

$$l \left(R_1 + \frac{1}{det(H)} R_2 \right)$$

for tiles defined by

$$\begin{aligned} \vec{h_1} \cdot \vec{x} &\geq c_1 \\ \vec{h_1} \cdot \vec{x} &< c_1 + l \\ \vec{h_2} \cdot \vec{x} &\geq c_2 \\ \vec{h_2} \cdot \vec{x} &< c_2 + l\, det(H) \end{aligned}$$

For tiles defined by

$$\begin{aligned} \vec{h_1} \cdot \vec{x} &\geq c_1 \\ \vec{h_1} \cdot \vec{x} &< c_1 + r_1\, l \\ \vec{h_2} \cdot \vec{x} &\geq c_2 \\ \vec{h_2} \cdot \vec{x} &< c_2 + r_2\, l \end{aligned}$$

(where r_1 and r_2 are aspect ratios) the communication volume is:

$$l\left(\frac{r_2}{det(H)}R_1 + \frac{r_1}{det(H)}R_2\right).$$

Thus, for 2-D iteration spaces, at each simplex vertex (each vertex defines a distinct transformation H) where $det(H)$ is nonzero, we need to evaluate the communication volume and choose the simplex (vertex) corner that minimizes communication volume.

7 Scheduling using the tile space graph

The tiles defined in section 3.1 are parallelopipeds that tessellate the K-dimensional iteration space. Given the dependence vectors in the ISG, we define the *Tile Space Dependence Graph* (TSG) which indicates the dependences (precedence constraints) among the tiles. We assume here that tile size is larger than the magnitude of any dependence vector *i.e.* the tile size along each dimension is larger than the largest of the corresponding components of the dependence vector. This assumption leads to the following:

- The source and sink of any dependence vector that crosses a tile boundary are neighboring tiles
- Legal tiles satisfy the convexity condition.

All dependence vectors in the Tile Space Dependence Graph (TSG) are K-tuples where each component is 0 or 1. Thus, there are at most $2^K - 1$ other tiles that a given tile can depend on. Lamport [23] derived a method of scheduling loop iterations represented as indices (points) in an iteration space by finding a family of parallel hyperplanes such that all indices lying on one hyperplane can be executed simultaneously. We refer to this family of hyperplanes as the *scheduling hyperplane.* For scheduling of tiles, the scheduling hyperplane must satisfy the conditions of the hyperplane theorem [23]. The scheduling hyperplane given by $\vec{1} = (1, 1, \ldots, 1)$, which is a K-dimensional vector with all components equal to 1, satisfies the said conditions.

7.1 Allocation of tiles to processors

In general we may have many more tiles than the number of processors. The tiles must be allocated so that interprocessor communication is minimized while the computation is load-balanced through time. We note here that the allocation scheme has a significant impact on the execution time. We now discuss a specific tile allocation scheme for nested loops. Our method of generation of tiles based on the derivation of H ensures that the tile space graph has dependences along orthogonal directions: $(1, 0, \ldots, 0)$,$(0, 1, \ldots, 0)$,. . .,$(0, \ldots, 0, 1, 0, \ldots, 0)$, . . .,$(0, \ldots, 0, 1)$. Hence choosing allocations of tiles along any one of the directions “internalizes” all communication in that direction. The advantage of the scheme under consideration is that a K-dimensional TSG is mapped onto a $K-1$ dimensional torus, which easily maps onto popular interconnection topologies of multicomputers. We are studying the impact of mapping the K-dimensional TSG onto an r-dimensional torus, for $1 \leq r < K - 1$. The interplay between load balance and communication depends on tile size and the tile allocation scheme; this is discussed next.

7.2 Optimizing tile size

Let the computation be defined by an K-dimensional ISG of size $n \times n \times \cdots \times n$. Let the tiles be defined by parallelopipeds whose sides are of size l along the orthogonal axes in the tile space; this results in rhomboidal partitions of the iteration space. We will show results assuming an unlimited number of processors. Finally, we assume that tiles are allocated among processors as described in the previous subsection.

The size of the tile is assumed to be $l \times l \times \cdots \times l$; in addition we assume that $n >> l$. Let the communication setup cost for a packet be s; let the cost of data transmission be w. Note that all these costs are normalized with respect to the cost of executing a single iteration. Thus the communication cost model is:

$$t_{comm} = s + w \times \text{length}$$

The cost of sending c values per iteration across a tile boundary to the processor which executes the neighbor tile is

$$s + wcl^{K-1}$$

The optimal tile size is one that minimizes the cost of execution of tiles. We derive an expression for the optimal tile size (l) by solving for zeros of the derivative of the cost with respect to l. Given that l^K nodes have to be executed in a tile, the cost of execution of the tiles is:

$$\begin{aligned} &\text{Time, } T = \\ &\quad \frac{Kn}{l}\left[l^K + (K-1)wcl^{K-1} + (K-1)s\right] \end{aligned}$$

Setting $dT/dl = 0$, we get

$$\begin{aligned} \frac{dT}{dl} &= 0 \\ \Longrightarrow \quad & \frac{Kn}{l}\left[Kl^{K-1} + wc(K-1)^2 l^{K-2}\right] \\ & -\frac{Kn}{l^2}\left[l^K + (K-1)wcl^{K-1} + (K-1)s\right] = 0 \end{aligned}$$

assuming $s \neq 0$. If $w \approx 0$ (data transmission cost per word is negligible), we have

$$l^K = s \text{ or } l_{opt} = \sqrt[K]{s}$$

If $s = 0$, then

$$\begin{aligned} & T = Knl^{K-1}\left[l + (K-1)wc\right] \\ \Longrightarrow \quad & l_{opt} = 1 \end{aligned}$$

Similar results can derived for an overlapped communication model.
For $K = 3$, one can derive analytical results for l_{opt} when $w \not\approx 0$. Setting $dT/dl = 0$, we get

$$l^3 + l^2 wc - s = 0$$

This equation can be solved analytically if $s \geq \frac{4w^3c^3}{27}$ [1]. For other cases, given the values of machine parameters (w, s) and the problem parameters (c, n, K), the optimal value of l can be obtained through numerical methods.

Thus, the tile boundaries induce a Tile Space Graph for which we have identified that the wavefront schedule $\vec{1}$ is a valid scheduling hyperplane and in the case of K-nested loops with dependence matrices whose rank is K, this is also the optimal schedule when unlimited processors are available for loop execution.

8 Summary

In this paper, we dealt with compiler support for parallelizing programs for coarse-grain multicomputers – we considered the class of programs expressible as tightly nested loops with a regular dependence structure. The relatively high communication startup costs in these machines renders frequent communication very expensive. We studied the effect of clustering communication and the ensuing loss of parallelism on performance and propose a method of aggregating a number of loop iterations into "tiles" where the tiles execute atomically. Since execution of tiles is atomic, it is important that dividing the iteration spaces of loops into tiles does not result in deadlock. We showed the equivalence between the problem of finding partitions and the problem of determining the extreme vectors (cone) for a given set of dependence vectors. We then presented an approach to partitioning the iteration space into deadlock-free tiles so that communication volume is minimized. We also presented a technique to optimize tile size for multicomputers. We are also investigating impact of tiling on the distribution of variables that do not induce dependences. Work is in progress on the problem of deriving communication minimal tiling for linear dependences (as those in LU decomposition) and for nontightly nested loops. We are also studying the use of tiling to minimize task scheduling overhead in parallel execution of loops on shared memory machines.

References

[1] M. Abramowitz and I. Stetgun. *Handbook of Mathematical Functions.* Dover Publications, New York, 1984.

[2] J. R. Allen. *Dependence Analysis for Subscripted Variables and its Applications to Program Transformations.* Ph.D. Dissertation, Department of Mathematical Sciences, Rice University (UMI 83-14916), Houston, Texas, April 1983.

[3] R. Allen and K. Kennedy. Automatic Translation of FORTRAN Programs to Vector Form. *ACM Trans. Programming Languages and Systems,* Vol. 9, No. 4, 1987, pp. 491-542.

[4] A. Bachem. The Theorem of Minkowski for Polyhedral Monoids and Aggregated Linear Diophantine Systems. *Optimization and Operations Research – Proc. of Workshop,* University of Bonn, October 1977, Lecture Notes in Economics and Mathematical Systems, Vol. 157, pages 1–14, Springer Verlag.

[5] V. Balasundaram, G. Fox, K. Kennedy and U. Kremer. An interactive environment for data partitioning and distribution. *Proc. 5th Distributed Memory Computing Conference (DMCC5),* Charleston, S. Carolina, pages 1160–1170, April 1990.

[6] U. Banerjee. *Dependence Analysis for Supercomputing,* Kluwer Academic Publishers, Boston, MA, 1988.

[7] U. Banerjee. Unimodular Transformation of Double Loops. *Advances in Languages and Compilers for Parallel Processing,* A. Nicolau et al., (Eds.), Pitman, London, 1991, pp. 192–219.

[8] D. Callahan and K. Kennedy. Compiling Programs for Distributed-Memory Multiprocessors. *The Journal of Supercomputing,* Vol. 2, Oct. 1988, pp. 151-169.

[9] D. Callahan, S. Carr and K. Kennedy. Improving Register Allocation of Subscripted Variables. *Proc. ACM SIGPLAN 90 Conf. Programming Language Design and Implementation,* pp. 53–65, June 1990.

[10] M. Chen, Y. Choo and J. Li. Compiling Parallel Programs by Optimizing Performance. *The Journal of Supercomputing,* 2, 1988, pp. 171-207.

[11] M. Chen, Y. Choo and J. Li. Theory and pragmatics of compiling efficient parallel code. Technical Report YALEU/DCS/TR-760, Yale University, December 1989.

[12] K. Gallivan, W. Jalby and D. Gannon. On the Problem of Optimizing Data Transfers for Complex Memory Systems. *Proc. 1988 ACM International Conference on Supercomputing,* St. Malo, France, pp. 238-253.

[13] D. Gannon, W. Jalby and K. Gallivan. Strategies for Cache and Local Memory Management by Global Program Transformations. *Journal of Parallel and Distributed Computing,* Vol. 5, No. 5, October 1988, pp. 587-616.

[14] S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer and C. Tseng. An Overview of the Fortran D Programming System. Tech. Report Rice COMP TR91-154, Department of Computer Science, Rice University, March 1991.

[15] F. Irigoin and R. Triolet. Supernode Partitioning. in *Proc. 15th Annual ACM Symp. Principles of Programming Languages,* San Diego, CA, Jan. 1988, 319-329.

[16] C. King and L. Ni. Grouping in Nested Loops for Parallel Execution on Multicomputers. *Proc. International Conf. Parallel Processing,* 1989, Vol. 2, pp. II-31 to II-38.

[17] C. King, W. Chou and L. Ni. Pipelined Data-Parallel Algorithms: Part II – Design. *IEEE Trans. Parallel and Distributed Systems,* Vol. 1, No. 4, pages 486–499, October 1990.

[18] C. Koelbel, P. Mehrotra and J. van Rosendale. Semi-automatic Process Partitioning for Parallel Computation. *International Journal of Parallel Programming,* Vol. 16, No. 5, 1987, pp. 365-382.

[19] C. Koelbel, P. Mehrotra and J. van Rosendale. Supporting Shared Data Structures on Distributed Memory Machines. *Proc. Second ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPoPP),* SIGPLAN Notices, Vol. 25, No. 3, pages 177–186, March 1990.

[20] C. Koelbel. *Compiling programs for nonshared memory machines.* Ph.D. thesis, CSD-TR-1037, Purdue University, November 1990.

[21] U. Kremer, H. Bast, H. Gerndt and H. Zima. Advanced Tools and Techniques for Automatic Parallelization. *Parallel Computing,* Vol. 7, 1988, pp. 387-393.

[22] D. Kuck, R. Kuhn, B. Leasure, D. Padua and M. Wolfe. Dependence Graphs and Compiler Optimizations. *Proc. ACM 8th Annual Symposium on Programming Languages,* Williamsburg, VA, Jan. 1981, pp. 207-218.

[23] L. Lamport. The Parallel Execution of DO Loops. *Communications of the ACM,* Vol. 17, No. 2, Feb. 1974, pp. 83-93.

[24] M. Mace. Memory Storage Patterns in Parallel Processing. Kluwer Academic Publishers, Boston, MA, 1987.

[25] A. Nicolau. Loop Quantization: A Generalized Loop Unwinding Technique. *Journal of Parallel and Distributed Computing,* Vol. 5, No. 5, Oct. 1988, pp. 568-586.

[26] D. Padua and M. Wolfe. Advanced Compiler Optimizations for Supercomputers. *Communications of the ACM,* Vol. 29, No. 12, Dec. 1986, pp. 1184-1201.

[27] A. Rogers and K. Pingali. Process Decomposition Through Locality of Reference. *Proc. ACM SIGPLAN 89 Conference on Programming Language Design and Implementation,* Portland, OR, Jun. 1989, pp. 69-80.

[28] A. Rogers. *Compiling for locality of reference.* Ph.D. thesis, Cornell University, August 1990.

[29] J. Ramanujam and P. Sadayappan. Tiling of Iteration Spaces for Multicomputers. *Proc. 1990 International Conference on Parallel Processing,* Vol 2, pages 179-186, August 1990.

[30] J. Ramanujam. *Compile-time Techniques for Parallel Execution of Loops on Distributed Memory Multiprocessors.* Ph.D. Thesis, The Ohio State University, Dept. of Comp. & Inf. Sci., (UMI 91-11789) September 1990.

[31] J. Ramanujam. A linear algebraic view of loop transformations and their interaction. in *Parallel Processing for Scientific Computing,* D. Sorensen (Editor), SIAM Press, March 1991.

[32] J. Ramanujam. Unimodular and non-unimodular transformations of nested loops. Technical Report TR-91-12-03, Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA. December 1991.

[33] V. Sarkar and J. Hennessey. Partitioning Programs for Macro-Dataflow. *Proc. 1986 ACM Conf. on Lisp and Functional Programming,* pages 202–211, August 1986.

[34] V. Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors.* Pitman, London and the MIT Press, Cambridge, Massachusetts, 1989.

[35] R. Schreiber and J. Dongarra. Automatic Blocking of Nested Loops. Technical Report, University of Tennessee, Knoxville, TN, August 1990.

[36] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience series in Discrete Mathematics and Optimization, John Wiley and Sons, New York, 1986.

[37] M. Wolf and M. Lam. A Data Locality Optimizing Algorithm. *Proc. ACM SIGPLAN 91 Conf. Programming Language Design and Implementation,* pp. 30–44, June 1991.

[38] M. Wolfe. *Optimizing Supercompilers for Supercomputers.* Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Report No. 82-1105, (UMI 83-03027) October 1982.

[39] M. Wolfe. Iteration Space Tiling for Memory Hierarchies. in *Parallel Processing for Scientific Computing,* G. Rodrigue (Ed.) SIAM, Philadelphia PA, 1987, pp. 357-361.

[40] M. Wolfe and U. Banerjee. Data Dependence and Its Application to Parallel Processing. *International Journal of Parallel Programming,* Vol. 16, No. 2, 1987, pp. 137-178.

[41] M. Wolfe. *Optimizing Supercompilers for Supercomputers,* Pitman Publishing, London and MIT Press, 1989.

[42] M. Wolfe. More Iteration Space Tiling. in *Proc. Supercomputing 89,* Reno NV, Nov. 13-17, 1989, pp. 655-664.

[43] H. Zima, H. Bast and H. Gerndt. SUPERB: A Tool for Semi-automatic MIMD/SIMD Parallelization. *Parallel Computing,* Vol. 6, 1988, pp. 1-18.

[44] H. Zima and B. Chapman. *Supercompilers for parallel and vector supercomputers.* ACM Press Frontier Series, 1990.