The Efficient Computation of Ownership Sets in HPF

Pramod G. Joisha, *Student Member*, *IEEE*, and Prithviraj Banerjee, *Fellow*, *IEEE*

Abstract—Ownership sets are fundamental to the partitioning of program computations across processors by the owner-computes rule. These sets arise due to the mapping of arrays onto processors. In this paper, we focus on how ownership sets can be efficiently determined in the context of the HPF language and show how the structure of these sets can be symbolically characterized in the presence of arbitrary array alignment and array distribution directives. Our starting point is a system of equalities and inequalities due to Ancourt et al. [1] that captures the array mapping problem in HPF. We arrive at a refined system that enables us to efficiently solve for the ownership set using the Fourier-Motzkin Elimination technique and that requires the course vector as the only auxiliary vector. The formulation makes it possible to enumerate the elements of the ownership set exactly once, a feature that is very beneficial when such sets are applied to handle `DO` loops qualified by HPF's `INDEPENDENT` directive. We develop important and general properties pertaining to HPF alignments and distributions and show how they can be used to eliminate redundant communication due to array replication. Polynomial-time schemes that determine whether the ownership set of a *particular* processor, with respect to some array, is the empty set or whether the ownership set of *every* processor, with respect to some array, is the empty set, are presented. We show how distribution directives with unspecified processor meshes can be efficiently handled at compile time. We also show how to avoid the generation of communication code when pairs of array references are ultimately mapped onto the same processors. Experimental data demonstrating the improved code performance that the latter optimization enables is presented and discussed.

Index Terms—HPF, array alignment, array distribution, ownership set, Fourier-Motzkin Elimination technique, parallelizing compiler.

1 Introduction

In languages such as High Performance Fortran (HPF) [10], array mappings guide the partitioning of program computations across processors. They are specified by the programmer in terms of annotations called *directives*. The actual mapping process typically involves two steps: Arrays are first *aligned* with a template and templates are then *distributed* over a virtual mesh of processors. The alignment operation, performed via the `ALIGN` directive, assigns every array element to at least a single template cell. The distribution operation, done using the `DISTRIBUTE` directive, associates every template cell with exactly one processor. In this way, array elements are eventually mapped onto processors.

In an automated code generation scenario, the compiler decides the processors on which to execute the various compute operations occurring in a program. In allocating program computations to processors, the compiler uses the mapping information associated with the data. A possible scheme, known as the *owner-computes rule* [15], is to allow only the owner of the left-hand side reference in an assignment statement to execute the statement. By the owner-computes rule, expressions that use data located on the same processor can be evaluated locally on that processor, without the need for interprocessor communication. When the need to transfer remotely located data arises, the compiler produces the relevant communication code. Hence, the owner-computes rule leads to the notion of an array's *ownership set*, which is the set of all its elements mapped onto a processor by virtue of the alignment and distribution directives.

Since the assignment of computations to processors is determined by the allocation of data to processors, one of the aims of the HPF compilation problem is to find a suitable way of representing the mapping information. Given such a representation, the next issue that must be addressed is how can it be used to realize the owner-computes rule. Does the proposed framework provide insights into the nature of the array mapping problem? Does the representation reveal general properties that can be leveraged to generate efficient code? In this paper, we investigate these questions in the context of a recent representation proposed by Ancourt et al. [1].

1.1 An Example

Fig. 1 shows a four-point stencil computation that occurs over a two-dimensional grid of 1022×1022 points. The value at every grid point is updated with the average of its four neighbors. Once the entire grid is updated this way, the process is repeated with the new values.

The example introduces an abstract two-dimensional array `T` of size 1024×1024, called the template, against which the arrays `A` and `B` are aligned. The alignment directives align the array elements `A(i,j)` and `B(i,j)` with the template cell `T(i,j)`. The `DISTRIBUTE` directive maps blocks of template cells in `T` onto a two-dimensional virtual processor mesh `P`. The extent of a block along a dimension is determined by the extents of `T` and `P` along that

• *The authors are with the Center for Parallel and Distributed Computing, Electrical and Computer Engineering Department, Technological Institute, 2145 Sheridan Road, Northwestern University, Evanston, IL 60208-3118. E-mail: {pjoisha, banerjee}@ece.nwu.edu.*

Manuscript received 15 Sept. 1999; accepted 17 Jan. 2001.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 110599.

```
      program example

      REAL A(2:1023, 2:1023)
      REAL B(1:1024, 1:1024)
      INTEGER i, j, k

!HPF$ TEMPLATE T(1:1024, 1:1024)
!HPF$ PROCESSORS P(2, 2)
!HPF$ ALIGN A(i, j) WITH T(i, j)
!HPF$ ALIGN B(i, j) WITH T(i, j)
!HPF$ DISTRIBUTE T(BLOCK, BLOCK) ONTO P

!C    Arrays A and B are initialized here.

      do k = 0, 99
         do j = 2, 1023
            do i = 2, 1023
               A(i, j) = (B(i-1, j)+B(i+1, j)+B(i, j-1)+B(i, j+1))/4
            enddo
         enddo
         do j = 2, 1023
            do i = 2, 1023
                B(i, j) = A(i, j)
            enddo
         enddo
      enddo

      end
```

Fig. 1. A four-point stencil computation in HPF.

dimension. Since the extents along each dimension in this case are 1024 and 2, respectively, a block size of

$$(1024/2) \times (1024/2) = 512 \times 512$$

is used. Thus, the cells of the template T in the region $\mathtt{T}(512p_1 + l_1,\ 512p_2 + l_2)$ get mapped onto the processor $\mathtt{P}(p_1, p_2)$, where $0 \leq l_1, l_2 < 512$ and $0 \leq p_1, p_2 < 2$. We could have also chosen a $\mathtt{CYCLIC}(B)$ distribution along a particular processor dimension instead of a BLOCK distribution; in this case, the distribution is allowed to "wrap around" the processor dimension. Because $\mathtt{A(i,j)}$ and $\mathtt{B(i,j)}$ are aligned with $\mathtt{T(i,j)}$, the (BLOCK, BLOCK) distribution results in each of the four quadrants in A and B being mapped onto the corresponding "quadrant processor" in P. Hence, a specific processor would execute only those statement instances in Fig. 1 whose left-hand sides lie in the quadrant of A or B owned by it.

1.2 Related Work

The problem of array alignment and array distribution has been extensively studied and numerous structures that describe the mapping of arrays to processors have been suggested and examined [7], [4], [2], [13], [14], [3]. Early representations focused on BLOCK distributions alone and were incapable of conveniently describing the general $\mathtt{CYCLIC}(B)$ distribution. This deficiency was addressed in subsequent work by using techniques ranging from finite state machines, virtual processor meshes to set-theoretic methods [6], [8], [12]. However, these schemes primarily concentrated on enumerating local memory access sequences and handling array expressions. A generalized view of the HPF mapping problem was subsequently presented by Ancourt et al. [1] who showed how a system of equalities and inequalities could be used to mathematically express the *regular* alignment and distribution of arrays to processors. These systems were then used to formulate ownership sets and compute sets for loops qualified by the INDEPENDENT directive and parametric solutions for the latter were provided based on the Hermite Normal Form [1].

1.3 Contributions

In this paper, we investigate the ownership set formulation in the Ancourt et al. framework and show how it can be refined to a form that requires a course vector as the only auxiliary vector and that also enables the efficient enumeration of its constituent elements. This property is desirable when ownership sets are applied to handle DO loops qualified by HPF's INDEPENDENT directive. Our approach to solving for the ownership set is based on the Fourier-Motzkin Elimination (FME) technique and, in that respect, we deviate from [1]. We also formulate an efficient polynomial-time test using which redundant communication due to array replication can be avoided. We present a sufficient condition called the *mapping test* that eliminates the need for generating communication code for certain right-hand side array references in an assignment statement. These two optimizations in turn depend upon whether the ownership set of a *particular* processor with respect to some array is the empty set or whether the ownership set of *every* processor with respect to some array is the empty set. We discuss how these decisions can be arrived at in polynomial time, once the symbolic representation of the ownership set is known. The mapping test often results in marked performance improvements and we substantiate this by presenting experimental data. Finally, we discuss how our schemes permit the efficient handling at compile time of distributions in which the processor

```
REAL A(-1:20, 3:40, 0:20)

!HPF$ TEMPLATE T(0:99, 0:99, 0:99)
!HPF$ PROCESSORS P(1:9, 1:9)
!HPF$ ALIGN A(:, *, k) WITH T(2*k+1, 2:44:2, *)
!HPF$ DISTRIBUTE T(*, CYCLIC(4), BLOCK(13)) ONTO P
```

Fig. 2. Original fragment.

```
REAL A(-1:20, 3:40, 0:20)

!HPF$ TEMPLATE T(0:99, 0:99, 0:99)
!HPF$ PROCESSORS P(1:9, 1:9)
!HPF$ ALIGN A(i, j, k) WITH T(2*k+1, (i+1)*2+2, l)
!HPF$ DISTRIBUTE T(*, CYCLIC(4), BLOCK(13)) ONTO P
```

Fig. 3. Modified fragment.

meshes are unspecified. The techniques mentioned in this paper have been incorporated into a new version of the PARADIGM compiler [9] using *Mathematica*[1] as the symbolic manipulation engine.

1.4 Outline

The rest of this paper is organized as follows: In Section 2, we describe previous research that forms the foundations of our work. We discuss in this section how information pertaining to alignments and distributions can be compactly expressed as systems of equalities and inequalities. The ownership set is formally defined in this section. We refine the ownership set formulation in Section 3 and prove an important equivalence relation. In Section 4, we present the replication test and explain how it can be used to avoid redundant communication due to array replication. We also show in Section 4 how to ascertain in polynomial time whether the ownership set is empty or nonempty, given its symbolic FME solution. In Section 5, we describe the handling of distributions that lack an explicitly specified processor mesh. The mapping test is derived in Section 6 and we illustrate its workings using three examples. Finally, in Section 7, we report and analyze experimental data that demonstrates the performance benefits of the mapping test optimization.

2 Preliminaries

Fig. 2 shows an artificial code fragment comprising a declaration and a set of HPF directives. Since the first dimension of A and the single subscript-triplet expression conform, this fragment is equivalent to that shown in Fig. 3. The dummy variables i, j, k, and l in Fig. 3 satisfy the constraints $-1 \le i \le 20$, $3 \le j \le 40$, $0 \le k \le 20$, and $0 \le l \le 99$, respectively.

The alignment directives in Fig. 3 can be compactly expressed through the following collection of equalities and inequalities [1]:

$$\hat{\mathcal{R}}\boldsymbol{t} = \hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s}_0 - \hat{\mathcal{R}}\boldsymbol{l}_T, \tag{1}$$

$$\boldsymbol{a_l} \le \boldsymbol{a} \le \boldsymbol{a_u}, \tag{2}$$

$$\boldsymbol{0} \le \boldsymbol{t} \le \boldsymbol{u_T} - \boldsymbol{l_T}. \tag{3}$$

Similarly, the distribution directives in Fig. 3 can be represented by the following system [1]:

$$\hat{\pi}\boldsymbol{t} = \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} + \boldsymbol{l}, \tag{4}$$

$$\hat{\lambda}\boldsymbol{c} = \boldsymbol{0}, \tag{5}$$

$$\boldsymbol{0} \le \boldsymbol{p} < \hat{P}\boldsymbol{1}, \tag{6}$$

$$\boldsymbol{0} \le \boldsymbol{l} < \hat{C}\boldsymbol{1}, \tag{7}$$

where the column vector $\boldsymbol{t}$ in (4) satisfies (3). For the given example, the various matrices and vectors are

$$\hat{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \hat{\mathcal{A}} = \begin{pmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \end{pmatrix}, \boldsymbol{s_0} = \begin{pmatrix} 1 \\ 4 \end{pmatrix},$$
$$\boldsymbol{a_l} = \begin{pmatrix} -1 \\ 3 \\ 0 \end{pmatrix}, \boldsymbol{a_u} = \begin{pmatrix} 20 \\ 40 \\ 20 \end{pmatrix}, \boldsymbol{l_T} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \boldsymbol{u_T} = \begin{pmatrix} 99 \\ 99 \\ 99 \end{pmatrix},$$

and

$$\hat{\pi} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \hat{C} = \begin{pmatrix} 4 & 0 \\ 0 & 13 \end{pmatrix}, \hat{P} = \begin{pmatrix} 9 & 0 \\ 0 & 9 \end{pmatrix}, \hat{\lambda} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

While (1) is a consequence of the `ALIGN` directive, (4) and (5) are a result of the `DISTRIBUTE` directive. While (4) dictates the mapping of template cells onto processors, (5) indicates whether a particular processor dimension has a `BLOCK` or a `CYCLIC` distribution associated with it. Constraints on the array bounds vector $\boldsymbol{a}$ and the template cell vector $\boldsymbol{t}$ are given by (2) and (3), respectively. Finally, (6) and (7) describe the constraints that the processor identity vector $\boldsymbol{p}$ and the offsets vector $\boldsymbol{l}$ must satisfy. We shall use x, y, and z to represent the number of dimensions of the *alignee* [10], template, and processor mesh, respectively. Using this notation, the column vectors $\boldsymbol{a}$ and $\boldsymbol{t}$ consist of x and y elements respectively, while the column vectors $\boldsymbol{p}$, $\boldsymbol{l}$, and $\boldsymbol{c}$—called the *course vector*—have z elements each.

The ownership set of a processor $\boldsymbol{p}$, which is defined with respect to an array X, denotes those elements of X that are finally mapped onto $\boldsymbol{p}$ by virtue of the alignment and distribution directives. In set-theoretic notation, this set is:

$$\begin{aligned} \Delta_p(X) = \{\boldsymbol{a} | \exists \boldsymbol{c}, \boldsymbol{l}, \boldsymbol{t} \text{ such that} \\ & \hat{\mathcal{R}}\boldsymbol{t} = \hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}, \\ & \hat{\pi}\boldsymbol{t} = \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} + \boldsymbol{l}, \\ & \hat{\lambda}\boldsymbol{c} = \boldsymbol{0}, \\ & \boldsymbol{a_l} \le \boldsymbol{a} \le \boldsymbol{a_u}, \\ & \boldsymbol{0} \le \boldsymbol{t} \le \boldsymbol{u_T} - \boldsymbol{l_T}, \\ & \boldsymbol{0} \le \boldsymbol{l} < \hat{C}\boldsymbol{1}\}, \end{aligned} \tag{8}$$

where $\boldsymbol{0} \le \boldsymbol{p} < \hat{P}\boldsymbol{1}$. For instance, we could evaluate (8) to obtain $\Delta_p(B)$ for the example in Fig. 1. If we assume a $(\texttt{CYCLIC(256)}, \texttt{CYCLIC(256)})$ distribution (instead of the

1. *Mathematica* is a registered trademark of Wolfram Research, Inc.

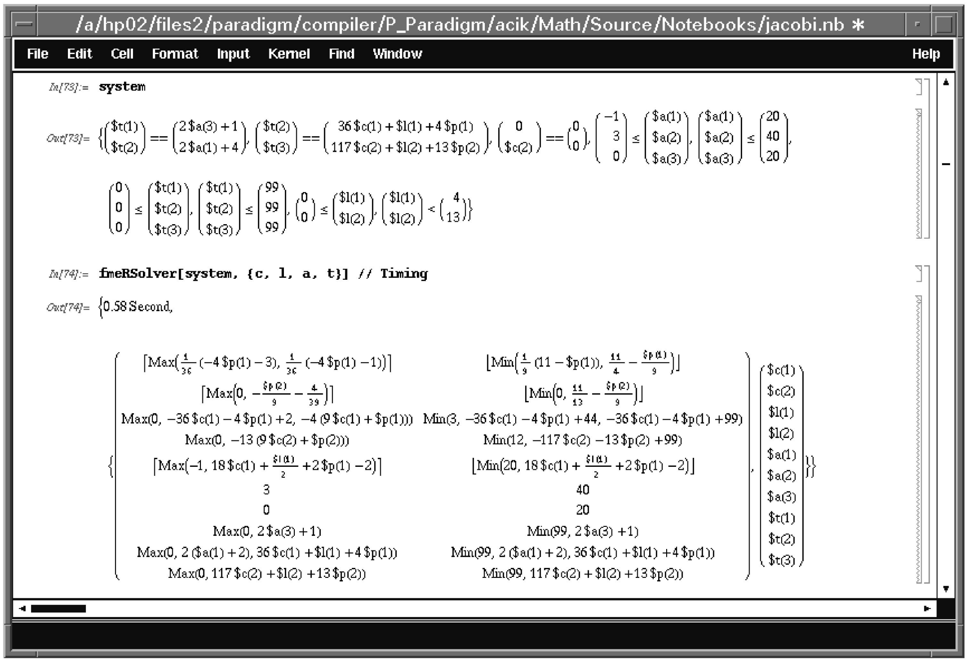


Fig. 4. An FME solver in *Mathematica*.

(BLOCK, BLOCK) distribution shown in Fig. 1), this turns out to be:

$$\begin{aligned}\Delta_p(B) = \Big\{ \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} | \exists \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}, \begin{pmatrix} l_1 \\ l_2 \end{pmatrix}, \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \text{ such that} \\ \left\lceil -\frac{255}{512} - \frac{p_1}{2} \right\rceil \le c_1 \le \left\lfloor \frac{1023}{512} - \frac{p_1}{2} \right\rfloor, \\ \left\lceil -\frac{255}{512} - \frac{p_2}{2} \right\rceil \le c_2 \le \left\lfloor \frac{1023}{512} - \frac{p_2}{2} \right\rfloor, \\ \max\big(0, -256(2c_1 + p_1)\big) \\ \le l_1 \le \min(255, 1023 - 512c_1 - 256p_1), \\ \max\big(0, -256(2c_2 + p_2)\big) \\ \le l_2 \le \min(255, 1023 - 512c_2 - 256p_2), \\ \max(0, 512c_1 + l_1 + 256p_1) \\ \le t_1 \le \min(1023, 512c_1 + l_1 + 256p_1), \\ \max(0, 512c_2 + l_2 + 256p_2) \\ \le t_2 \le \min(1023, 512c_2 + l_2 + 256p_2), \\ \max(1, 1 + t_1) \le a_1 \le \min(1024, 1 + t_1), \\ \max(1, 1 + t_2) \le a_2 \le \min(1024, 1 + t_2)\Big\},\end{aligned}$$

where

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \le \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} < \begin{pmatrix} 2 \\ 2 \end{pmatrix}.$$

3 Ownership Sets Revisited

To solve (8) symbolically, we apply the Fourier-Motzkin Elimination (FME) technique [5], eliminating the variables corresponding to the unknown vectors $\boldsymbol{c}$, $\boldsymbol{l}$, $\boldsymbol{t}$, and $\boldsymbol{a}$. During the elimination process, the vector $\boldsymbol{p}$, which denotes the processor's identity in a Cartesian processor mesh, is treated like any other constant; it therefore manifests in the bound expressions of the solution system. When resolved at runtime to a particular processor's identity, the solution system will represent that processor's ownership set. Fig. 4 shows the outcome when the FME technique is applied on the system corresponding to $\Delta_p(A)$ for the example in Fig. 3. We represent the chosen elimination order by the *elimination vector* ϵ. The elimination vector ϵ is a column vector whose last element denotes the first unknown to be eliminated and whose first element denotes the last unknown to be eliminated. In Fig. 4, $\boldsymbol{\epsilon}^T = (\boldsymbol{c}^T, \boldsymbol{l}^T, \boldsymbol{a}^T, \boldsymbol{t}^T)$.

Note that the actual region of interest is the set of points corresponding to the solution vector $\boldsymbol{a}$. We can scan the elements of $\Delta_p(A)$ by constructing a loop nest in which the outermost loop corresponds to $c(1) (i.e., c_1) having $\lceil \max\big((-4\$\mathtt{p(1)} - 3)/36, (-4\$\mathtt{p(1)} - 1)/36\big) \rceil$ as its lower bound and $\lfloor \min\big((11 - \$\mathtt{p(1)})/9, (11/4) - (\$\mathtt{p(1)}/9)\big) \rfloor$ as its upper bound. Nested within $c(1) are the loops corresponding to $c(2), $l(1), $l(2), $a(1), $a(2), $a(3), $t(1), $t(2), $t(3) and in that order. Thus, the innermost loop that corresponds to $t(3) (i.e., t_3) would have

$$\max(0, 117\$\mathtt{c(2)} + \$\mathtt{l(2)} + 13\$\mathtt{p(2)})$$

as its lower bound and

$$\min(99, 117\$\mathtt{c(2)} + \$\mathtt{l(2)} + 13\$\mathtt{p(2)})$$

as its upper bound. Within the body of such a loop nest, $a(1) and $a(2) (i.e., a_1 and a_2) will be the subscripts of the elements of the array A owned by processor $\boldsymbol{p}$.

3.1 Refinement

Since the elements of $\boldsymbol{c}$, $\boldsymbol{l}$, and $\boldsymbol{t}$ serve only as auxiliary variables in (8), an important question that crops up at this juncture is whether formulations for $\Delta_p(X)$ exist that require fewer auxiliary variables. A lesser number of

Lemma 3.1: Consider the following definition for $\Delta'_p(X)$:

$$\Delta'_p(X) = \{\boldsymbol{a} | \exists \boldsymbol{c}, \boldsymbol{t} \text{ such that } \begin{array}{l} \hat{\mathcal{R}}\boldsymbol{t} = \hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}, \\ \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} \leq \hat{\pi}\boldsymbol{t} < \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1}, \\ \hat{\lambda}\boldsymbol{c} = \boldsymbol{0}, \\ \boldsymbol{a_l} \leq \boldsymbol{a} \leq \boldsymbol{a_u}, \\ \boldsymbol{0} \leq \boldsymbol{t} \leq \boldsymbol{u_T} - \boldsymbol{l_T}\}, \end{array} \tag{9}$$

where $\boldsymbol{0} \leq \boldsymbol{p} < \hat{P}\boldsymbol{1}$. Then,

$$\Delta_p(X) = \Delta'_p(X).$$

Fig. 5. Lemma 3.1.

Lemma 3.2: Consider the following definition for $\Delta^*_p(X)$:

$$\Delta^*_p(X) = \{\boldsymbol{a} | \exists \boldsymbol{c} \text{ such that } \begin{array}{l} \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p}) \leq \hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}) \leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1} - \boldsymbol{1}), \\ \boldsymbol{0} \leq \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}), \\ \hat{\lambda}\boldsymbol{c} = \boldsymbol{0}, \\ \boldsymbol{a_l} \leq \boldsymbol{a} \leq \boldsymbol{a_u}, \\ \boldsymbol{0} \leq \hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T} \leq \hat{\mathcal{R}}(\boldsymbol{u_T} - \boldsymbol{l_T})\}, \end{array} \tag{10}$$

where $\boldsymbol{0} \leq \boldsymbol{p} < \hat{P}\boldsymbol{1}$. Then,

$$\Delta'_p(X) = \Delta^*_p(X).$$

Fig. 6. Lemma 3.2.

unknowns translates to a smaller loop nest depth and, hence, a more efficient scan of $\Delta_p(X)$. This is especially important when compute sets derived from ownership sets are used to partition loops [9]. Reducing the number of unknowns also improves the overall timing of the FME solver.

The first improvement that can be done is removing the offsets vector $\boldsymbol{l}$. This can be accomplished in a straightforward manner and Lemma 3.1 shows how. The system in Lemma 3.1, shown in Fig. 5, is an improvement over that in (8) because there is a reduction in the number of unknowns (by $\|\boldsymbol{l}\|$) and in the number of inequalities (by $2\|\boldsymbol{l}\|$).[2]

3.1.1 Removing t

A key observation to make in (9) is that the replicated dimensions of a template do not affect the elements of $\boldsymbol{a}$. This is because, by premultiplying $\boldsymbol{t}$ with $\hat{\mathcal{R}}$, rows that correspond to replicated dimensions in $\boldsymbol{t}$ get elided. What is a "replicated dimension?" We refer to those dimensions of a template that contain a $*$ or an unmatched dummy variable in the alignment specification as the template's *replicated dimensions* (see [10]). The remaining dimensions are called its *aligned dimensions*.

The dimensions of a template that are not mapped onto any processor dimension are said to be *collapsed*; the remaining dimensions of the template are referred to as its *distributed dimensions*. If a template dimension is collapsed and not replicated, that template dimension can only affect the values of $\boldsymbol{a}$ through the equality

$$\hat{\mathcal{R}}\boldsymbol{t} = \hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}$$

and the constraint $\boldsymbol{0} \leq \boldsymbol{t} \leq \boldsymbol{u_T} - \boldsymbol{l_T}$. To then find the corresponding elements of $\boldsymbol{a}$, we need only consider the pair:

$$\begin{array}{l} \boldsymbol{0} \leq \hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T} \leq \boldsymbol{u_T} - \boldsymbol{l_T}, \\ \boldsymbol{a_l} \leq \boldsymbol{a} \leq \boldsymbol{a_u}. \end{array}$$

Similarly, by considering

$$\begin{array}{l} \hat{\mathcal{R}}\boldsymbol{t} = \hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}, \\ \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} \leq \hat{\pi}\boldsymbol{t} < \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1}, \end{array}$$

template dimensions that are both aligned and distributed can be eliminated. Therefore, it is worth investigating whether a new system can be constructed for the ownership set that has a lesser number of unknowns than that required by (9). As Lemma 3.2 in Fig. 6 will show, such a formulation indeed does exist.

The new system defined in (10) has exactly the same number of inequalities as the system in (9); however, the gain is a reduction of $\|\boldsymbol{t}\|$ in the number of unknowns.

3.1.2 Further Refinements

From the standpoint of efficiently scanning the ownership set, the system in (10) has further scope for improvement. If we were to consider those processor dimensions j along which replicated template dimensions are distributed, the corresponding c_js are only constrained by

$$\boldsymbol{0} \leq \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}).$$

This means that for a given $\boldsymbol{a}$, there could exist more than one $\boldsymbol{c}$ for which the system in (10) holds. Hence, if the solution system for (10) were used to scan the ownership set, members could get enumerated more than once. To

2. The notation $\|v\|$ indicates the number of rows in the column vector v.

Lemma 3.3: Consider the following definition for $\Delta''_p(X)$:

$$\begin{aligned}\Delta''_p(X) = \{\boldsymbol{a} | \exists \boldsymbol{c} \text{ such that} \\ & \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{p}) \le \hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{A}\boldsymbol{a}+\boldsymbol{s_0}-\hat{\mathcal{R}}\boldsymbol{l_T}) \le \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{p}+\hat{C}\boldsymbol{1}-\boldsymbol{1}), \\ & \boldsymbol{0} \le \hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{p} \le \hat{\pi}(\boldsymbol{u_T}-\boldsymbol{l_T}), \\ & (\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\boldsymbol{c} = \boldsymbol{0}, \\ & \hat{\lambda}\boldsymbol{c} = \boldsymbol{0}, \\ & \boldsymbol{a_l} \le \boldsymbol{a} \le \boldsymbol{a_u}, \\ & \boldsymbol{0} \le \hat{A}\boldsymbol{a}+\boldsymbol{s_0}-\hat{\mathcal{R}}\boldsymbol{l_T} \le \hat{\mathcal{R}}(\boldsymbol{u_T}-\boldsymbol{l_T})\},\end{aligned} \tag{11}$$

where, $\boldsymbol{0} \le \boldsymbol{p} < \hat{P}\boldsymbol{1}$. Then, for every $\boldsymbol{a} \in \Delta''_p(X)$, there exists exactly one $\boldsymbol{c}$ that satisfies the system in Equation (11).

Fig. 7. Lemma 3.3.

remedy this problem, we extend the system in (10) by the equation $(\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\boldsymbol{c} = \boldsymbol{0}$. Lemma 3.3 in Fig. 7 ensures the validity of such a transformation.

An important implication of Lemma 3.3 is that if the FME technique is now applied to the system in (11), then, whatever be the order of elimination of the unknown variables (corresponding to the elements of $\boldsymbol{c}$ and $\boldsymbol{a}$), the associated loop nest will scan every member of the ownership set (i.e., $\boldsymbol{a}$) exactly once. To see why this is so, let ϵ represent one such elimination order. Suppose

$$\theta = \|\boldsymbol{\epsilon}\| = x + z,$$

where x and z denote the number of dimensions of the alignee and processor mesh, respectively. The integer bound expressions returned by the FME solver can be used to construct a loop nest that scans $\Delta''_p(X)$. The outermost loop in this nest matches ϵ_1, while the innermost loop matches ϵ_θ. Consider an iteration point $\boldsymbol{\rho}$ of such a loop nest and let $\boldsymbol{\varrho}$ be any other iteration point of the same loop nest. Thus, $\boldsymbol{\rho}$ and $\boldsymbol{\varrho}$ also represent solutions to the system in (11). Since every iteration point of a loop nest is distinct, let $\boldsymbol{\rho}$ and $\boldsymbol{\varrho}$ differ in the ith position. If $\epsilon_i \equiv a_n$, then the $\boldsymbol{a}$ that corresponds to $\boldsymbol{\rho}$ obviously differs from the $\boldsymbol{a}$ that corresponds to $\boldsymbol{\varrho}$. If instead $\epsilon_i \equiv c_j$, then the $\boldsymbol{c}$ that corresponds to $\boldsymbol{\rho}$ differs from the $\boldsymbol{c}$ that corresponds to $\boldsymbol{\varrho}$. But from Lemma 3.3, only one $\boldsymbol{c}$ can satisfy the system for a given $\boldsymbol{a}$. Thus, the corresponding values for $\boldsymbol{a}$ in $\boldsymbol{\rho}$ and $\boldsymbol{\varrho}$ must also be different. That is, the $\boldsymbol{a}$ associated with the iteration point $\boldsymbol{\rho}$ must be different from the $\boldsymbol{a}$ associated with any other iteration point $\boldsymbol{\varrho}$ of the same loop nest. In other words, every member of the ownership set gets enumerated exactly once.

3.1.3 Comparisons

From a mathematical perspective, the systems in (8), (9), (10), and (11) are all identical—they refer to the same ownership set in all cases. However, they differ in such metrics as the number of inequalities to be solved, the number of variables to be eliminated (consequently, the number of auxiliary parameters required to scan the set), and, finally, in the way the members of the set get enumerated. Table 1 summarizes these metrics. As can be seen from the table, the original system in (8) requires the elimination of the largest number of variables and handles the largest number of inequalities. If a loop nest were generated from its FME solution system to scan $\Delta_p(X)$, it is not guaranteed that every member will get enumerated exactly once. This inability to ensure the "uniqueness" of each enumerated member has serious repercussions if the ownership set is used to generate other sets. For instance, we could use the compute sets defined in [1] to handle loops qualified by the `INDEPENDENT` directive. In [1], these sets were defined using a formulation of the ownership set similar to that in (8). If the FME solution system to such a compute set formulation were scanned, certain iterations in the set could get enumerated more than once for certain alignment/distribution combinations. However, if the formulation in (11) is used, this problem can be avoided. The FME approach that we adopt to solve for these sets is quite different from the approach in [1] where a parametric solution based on the Hermite Normal Form was exploited for the same purpose.

In Fig. 8, we show the result of applying the FME technique to the formulation in (11). Though the system has the same number of inequalities as the

TABLE 1
Comparisons of the Ownership Set Formulations

System	Number of inequalities[d]	Number of unknowns[e]
Equation (8)	$2(x+y+3z+m)$	$x+y+2z$
Equation (9)	$2(x+y+2z+m)$	$x+y+z$
Equation (10)	$2(x+y+2z+m)$	$x+z$
Equation (11)	$2(x+y+3z+m)$	$x+z$

d. When the FME technique is applied, equalities of the form $f(c,l,t,a) = g(c,l,t,a)$ get replaced by a pair of inequalities of the form $f(\boldsymbol{c},\boldsymbol{l},\boldsymbol{t},\boldsymbol{a}) \le g(\boldsymbol{c},\boldsymbol{l},\boldsymbol{t},\boldsymbol{a})$ and $f(\boldsymbol{c},\boldsymbol{l},\boldsymbol{t},\boldsymbol{a}) \ge g(\boldsymbol{c},\boldsymbol{l},\boldsymbol{t},\boldsymbol{a})$.

e. To recapitulate, x is the number of dimensions of the alignee, y is the number of dimensions of the template, z is the number of dimensions of the processor mesh, and m is the number of aligned dimensions of the template.

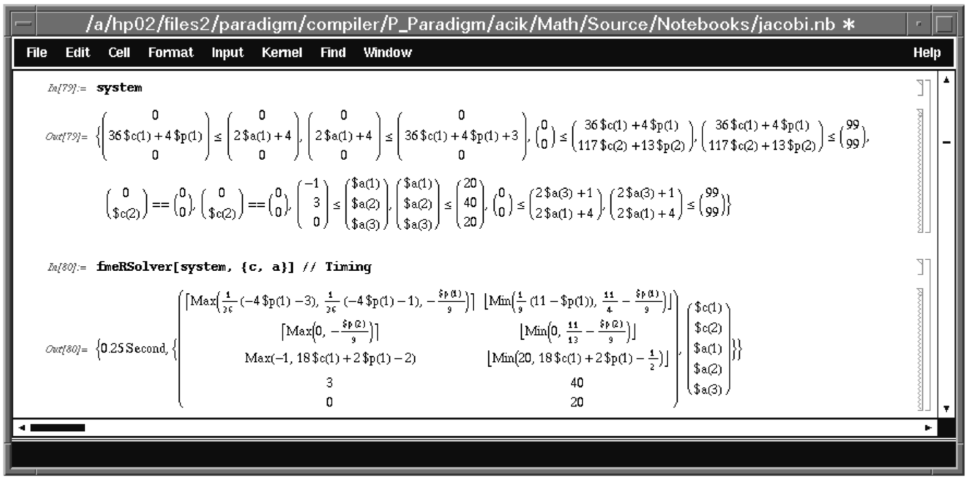


Fig. 8. The FME solver applied on the modified ownership set formulation.

Lemma 3.4: If $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\boldsymbol{p}-\boldsymbol{q}) = \mathbf{0}$, and $\Delta''_p(X) \neq \emptyset$, $\Delta''_q(X) \neq \emptyset$, then $\Delta''_p(X) = \Delta''_q(X)$.

Fig. 9. Lemma 3.4.

Lemma 3.5: If $\Delta''_p(X) \cap \Delta''_q(X) \neq \emptyset$, then $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\boldsymbol{p}-\boldsymbol{q}) = \mathbf{0}$.

Fig. 10. Lemma 3.5.

Theorem 1. For a given array X, the ownership sets across processors must be either equal or disjoint. That is,

$$\Delta''_p(X) = \Delta''_q(X),$$
or
$$\Delta''_p(X) \cap \Delta''_q(X) = \emptyset.$$

Fig. 11. Theorem 1.

formulation in (8), the number of unknowns to be eliminated is lesser by 5 variables, and this results in a significant reduction in the time required to solve the system from 0.58 seconds to 0.25 seconds.

3.2 An Equivalence Relation

It is interesting to enquire into the nature of ownership sets across processors. That is, for an arbitrary alignment/distribution combination, can these sets partially overlap? Or, are they equal or disjoint? The answers to these questions can be used to devise an efficient runtime test that avoids redundant communication due to array replication (see Section 4).

The expression $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T$ is a square diagonal matrix of size $z \times z$. It is easy to see that the principal diagonal elements of this matrix are either 0 or 1. It is also easy to see that the jth principal diagonal element is 0 if and only if the template dimension distributed on the jth processor dimension is a replicated one. To review, replicated dimensions are those dimensions of a template that contain either a $*$ or an unmatched dummy variable in the alignment specification. Thus, if an array X is aligned to a template T that is then distributed onto a processor mesh P, Lemma 3.4 states, in Fig. 9, that the ownership sets with respect to X of two processors $\boldsymbol{p}$ and $\boldsymbol{q}$ in P will be the same if (1) $\boldsymbol{p}$ and $\boldsymbol{q}$ own at least one element of the alignee array X, and (2) $\boldsymbol{p}$ and $\boldsymbol{q}$ match in at least those dimensions on which the aligned dimensions of the template T are distributed.

The reverse is also true; if the ownership sets of two processors with respect to an array overlap, then their coordinates along those dimensions on which the aligned dimensions of the template are distributed must match. This is what Lemma 3.5 states in Fig. 10. The above two lemmas can be used to prove Theorem 1 in Fig. 11:

Proof. Suppose $\Delta''_p(X)$ and $\Delta''_q(X)$ are not disjoint. Then $\Delta''_p(X) \cap \Delta''_q(X) \neq \emptyset$. Hence, from Lemma 3.5, we get

$$\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\boldsymbol{p}-\boldsymbol{q}) = \mathbf{0}. \quad \text{(I.1)}$$

Since we have assumed that $\Delta''_p(X) \cap \Delta''_q(X) \neq \emptyset$, then $\Delta''_p(X) \neq \emptyset$ and $\Delta''_q(X) \neq \emptyset$. By Lemma 3.4, this fact and (I.1) therefore imply that

$$\Delta''_p(X) = \Delta''_q(X).$$

Thus, either $\Delta''_p(X) \cap \Delta''_q(X) = \emptyset$ or $\Delta''_p(X) = \Delta''_q(X)$ must be true. □

Let us define a binary relation $\sim$ on a mapped array such that given two array elements β and γ, $\beta \sim \gamma$ if and only if β and γ are mapped onto the same processors. The rules of HPF ensure that for any legal ALIGN/DISTRIBUTE combination, every element of the mapped array will reside on at least one processor [10]. Hence, $\sim$ must be reflexive. Also, if $\beta \sim \gamma$, $\gamma \sim \beta$ is obviously true. Therefore, $\sim$ is symmetric. Finally, if $\beta \sim \gamma$ and $\gamma \sim \delta$ are true, then from Theorem 1, $\beta \sim \delta$. That is, $\sim$ is transitive. Hence, the ALIGN

for each $\mathbf{0} \le \boldsymbol{q} < \hat{P}\mathbf{1}$
    if $\boldsymbol{p} \ne \boldsymbol{q}$ then
        send $\Delta_p(Y) \cap \nabla_q\big(S', Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})\big)$ to $\boldsymbol{q}$
    endif
endfor
	for each $\mathbf{0} \le \boldsymbol{q} < \hat{P}\mathbf{1}$
	    if $\boldsymbol{p} \ne \boldsymbol{q}$ then
	        receive $\Delta_q(Y) \cap \nabla_p\big(S', Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})\big)$ from $\boldsymbol{q}$
	    endif
	endfor

Fig. 12. Send and receive actions at a processor *p*.

for each $\mathbf{0} \le \boldsymbol{q} < \hat{P}\mathbf{1}$
    if $(\Delta_q(Y) = \emptyset) \vee \big(\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\boldsymbol{q} - \boldsymbol{p}) \ne \mathbf{0}\big)$ then
        send $\Delta_p(Y) \cap \nabla_q\big(S', Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})\big)$ to $\boldsymbol{q}$
    endif
endfor
	for each $\mathbf{0} \le \boldsymbol{q} < \hat{P}\mathbf{1}$
	    if $(\Delta_p(Y) = \emptyset) \vee \big(\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\boldsymbol{p} - \boldsymbol{q}) \ne \mathbf{0}\big)$ then
	        receive $\Delta_q(Y) \cap \nabla_p\big(S', Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})\big)$ from $\boldsymbol{q}$
	    endif
	endfor

Fig. 13. Send and receive actions with the replication test optimization.

and DISTRIBUTE directives for a mapped array define an equivalence relation on that array.

4 The Replication Test

Let $\nabla_p\big(S', Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})\big)$ indicate those elements of a right-hand side array reference $Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})$ in an assignment statement S' that $\boldsymbol{p}$ views on account of its compute work in S'. Thus, $\boldsymbol{p}$ would have to potentially fetch these elements from a remote processor $\boldsymbol{q}$ and the set of elements to be received would be $\Delta_q(Y) \cap \nabla_p\big(S', Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})\big)$. Likewise, $\boldsymbol{p}$ would have to potentially send the elements in $\Delta_p(Y) \cap \nabla_q\big(S', Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})\big)$ to a remote processor $\boldsymbol{q}$ because $\boldsymbol{q}$ may in turn view the elements owned by $\boldsymbol{p}$. Pseudocode fragments in Fig. 12 illustrate how such data exchange operations can be realized.

In Section 3.2, we saw that if $\Delta_p(Y) \ne \emptyset$ and $\Delta_q(Y) \ne \emptyset$, then $\Delta_p(Y)$ and $\Delta_q(Y)$ are equal if and only if $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\boldsymbol{p} - \boldsymbol{q}) = \mathbf{0}$. This property can be used to avoid redundant communication due to array replication; the modified pseudocode fragments in Fig. 13 show this optimization.

Once the integer FME solution for the system of equalities and inequalities that describe the ownership set is obtained, computing whether $\Delta_p(Y)$ is the empty set for a given $\boldsymbol{p}$ incurs only an additional polynomial-time overhead. The key idea that enables this decision is that in the FME solution system for (11), a particular p_j will occur in the bound expressions for c_j.[3] That is, there will be an inequality pair of the form

$$f_j(p_j) \le c_j \le g_j(p_j)$$

in the solution system. In addition, there can *at most* be one more inequality pair in the solution system that also contains p_j in its bound expressions. This inequality pair will have the form $F_j(p_j, c_j) \le a_{n_j} \le G_j(p_j, c_j)$. Hence, if (11) has a solution $\boldsymbol{a}$ for a given $\boldsymbol{p}$, each of the z disjoint inequality groups

$$\begin{gathered} f_j(p_j) \le c_j \le g_j(p_j), \\ F_j(p_j, c_j) \le a_{n_j} \le G_j(p_j, c_j), \end{gathered}$$

in the solution system must *independently* admit a solution. The task of checking whether each of these groups has a solution for a particular p_j is clearly of quadratic complexity. Hence, the complexity of ascertaining whether $\Delta_p(Y)$ is nonempty for a given $\boldsymbol{p}$ is polynomial. Since the complexity of evaluating the condition $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\boldsymbol{p} - \boldsymbol{q}) \ne \mathbf{0}$ is $O(z)$, the overall runtime complexity of evaluating the Boolean predicates in Fig. 13, given the integer FME solution system for the ownership set (known at compile time), becomes polynomial.

Observe that in the absence of replication, $\hat{\mathcal{R}}^T\hat{\mathcal{R}}$ is the identity matrix; in this situation, $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\boldsymbol{p} - \boldsymbol{q}) \ne \mathbf{0}$ if and only if $\boldsymbol{p} \ne \boldsymbol{q}$. Hence, in the absence of replication, the test degenerates to the usual $\boldsymbol{p} \ne \boldsymbol{q}$ condition.

3. The symbols p_j and c_j indicate elements in $\boldsymbol{p}$ and $\boldsymbol{c}$, respectively.

5 Unspecified Processor Meshes

When a variable is eliminated from a system of inequalities by the FME technique, the method partitions the system into three sets: a set $\mathbf{S}_-$ in which the coefficients of the variable are negative, a set $\mathbf{S}_0$ in which the coefficients of the variable are zero and a set $\mathbf{S}_+$ in which the coefficients of the variable are positive [5]. Therefore, for those variables that are to be eliminated by the technique, knowledge regarding the signs of their coefficients must be available. In the context of the system in (11), this implies that the principal diagonal entries of the square diagonal matrices $\hat{C}$ and $\hat{P}$ can be symbolic. This is because these elements are known to be positive a priori and their actual values are of no concern until runtime. A useful consequence of this fact is that a processor mesh need not be provided in a DISTRIBUTE directive. Hence, the determination of the actual processor mesh can be postponed until runtime.

For instance, suppose that the distribution directive in Fig. 3 lacked a processor mesh—assume that it was

$$\mathtt{!HPF\$\ DISTRIBUTE\ T(*,CYCLIC(4),BLOCK(13))}.$$

Then, all that can be said about $\hat{P}$ is that it should have the form

$$\hat{P} = \begin{pmatrix} P_{11} & 0 \\ 0 & P_{22} \end{pmatrix},$$

where P_{11} and P_{22} are positive. If we apply the FME method on the system in (11) with this knowledge of $\hat{P}$, we obtain the solution shown in Fig. 14. Notice that on replacing P_{11} and P_{22} in this solution system by 9, we obtain the solution system shown in Fig. 8.

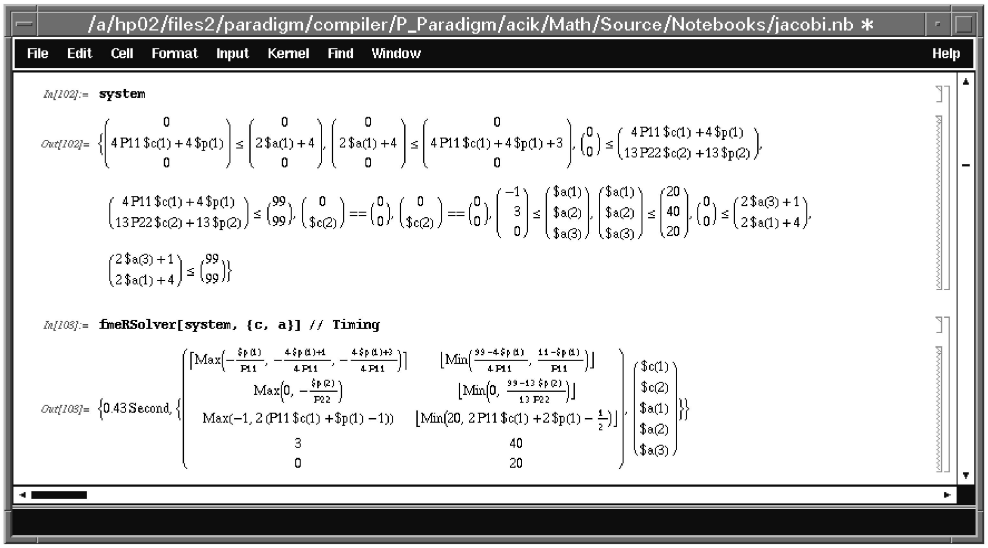


Fig. 14. Solving the ownership set with a symbolic $\hat{P}$.

Apart from being positive, since the third dimension of T is BLOCK distributed on the second dimension of P, the choice for P_{22} would have to be additionally constrained by the inequality $13P_{22} \geq 100$ (i.e., $C_{jj}P_{jj} \geq u_{k_j} - l_{k_j} + 1$ where k_j is the template dimension distributed in a BLOCK fashion on the jth processor dimension). More generally, the principal diagonal elements of $\hat{P}$ would have to be chosen so as to satisfy the following inequalities:

$$\hat{P}\mathbf{1} \geq \mathbf{1}, \tag{12}$$

$$\hat{\lambda}\hat{C}\hat{P}\mathbf{1} \geq \hat{\lambda}\hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T} + \mathbf{1}). \tag{15}$$

A possible strategy for determining the principal diagonal elements of $\hat{P}$ at runtime is presented in Fig. 15. In general, more than one distributee may be associated with a processor mesh and the handling of such a case is highlighted in the figure. The displayed pseudocode fragment assumes that two templates T and S are distributed on the processor mesh P. The procedure number_of_processors() is a generic system inquiry function that returns the number of physical processors available in the lower-level processor arrangement. The quantity ω—determined at compile time—equals the number of processor dimensions on which template dimensions are distributed in a CYCLIC(B) fashion *considering both* T and S. That is, ω equals the rank of $(\hat{I} - \hat{\lambda}_T)(\hat{I} - \hat{\lambda}_S)$. Note that, by this strategy, some of the physical processors may remain unused. More sophisticated schemes that utilize all of the physical processors and that also optimize with respect to some other criteria could be devised.

6 The Mapping Test

Consider an assignment statement S' bearing affine subscript expressions and contained in a loop nest characterized by the loop iteration vector $\boldsymbol{\alpha}$:

$$X(\hat{S}_x\boldsymbol{\alpha} + \boldsymbol{a_{0_x}}) = \cdots + Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a_{0_y}}) + \cdots.$$

The communication sets $\Delta_p(Y) \cap \nabla_q(S', Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a_{0_y}}))$ and $\Delta_q(Y) \cap \nabla_p(S', Y(\hat{S}_y\boldsymbol{\alpha} + \boldsymbol{a_{0_y}}))$ (shown in Fig. 12 and Fig. 13) that would be generated for the above assignment statement take into account the relative alignments and distributions of the left-hand side and right-hand side array references. If these communication sets are empty, no communication will occur at runtime. However, the overhead of checking at runtime whether a particular processor

set η = number_of_processors()
set $\hat{P} = \hat{I}$
set $\hat{\lambda}_T\hat{\lambda}_S\hat{P}\mathbf{1} = \hat{\lambda}_T\hat{\lambda}_S\mathtt{max}(\hat{C}_T^{-1}\hat{\pi}_T(\boldsymbol{u_T} - \boldsymbol{l_T} + \mathbf{1}), \hat{C}_S^{-1}\hat{\pi}_S(\boldsymbol{u_S} - \boldsymbol{l_S} + \mathbf{1}))$
set $\hat{\lambda}_T(\hat{I} - \hat{\lambda}_S)\hat{P}\mathbf{1} = \hat{\lambda}_T(\hat{I} - \hat{\lambda}_S)\hat{C}_T^{-1}\hat{\pi}_T(\boldsymbol{u_T} - \boldsymbol{l_T} + \mathbf{1})$
set $\hat{\lambda}_S(\hat{I} - \hat{\lambda}_T)\hat{P}\mathbf{1} = \hat{\lambda}_S(\hat{I} - \hat{\lambda}_T)\hat{C}_S^{-1}\hat{\pi}_S(\boldsymbol{u_S} - \boldsymbol{l_S} + \mathbf{1})$
if $|\hat{P}| > \eta$ then
    print "Error: Insufficient number of processors."
    exit
else
    set $(\hat{I} - \hat{\lambda}_T)(\hat{I} - \hat{\lambda}_S)\hat{P}\mathbf{1} = \lfloor(\frac{\eta}{|\hat{P}|})^{1/\omega}\rfloor$
endif

Fig. 15. Establishing $\hat{P}$ at runtime.

Theorem 2. Define two vectors $\boldsymbol{\zeta}$ and $\boldsymbol{\xi}$ for the assignment statement S':

$$\boldsymbol{\zeta} = \hat{\pi}_T \hat{\mathcal{R}}_x^T \big(\hat{\mathcal{A}}_x(\hat{S}_x \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_x}) + \boldsymbol{s}_{\mathbf{0}_x} - \hat{\mathcal{R}}_x \boldsymbol{l}_{\boldsymbol{T}}\big),$$
$$\boldsymbol{\xi} = \hat{\pi}_S \hat{\mathcal{R}}_y^T \big(\hat{\mathcal{A}}_y(\hat{S}_y \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y}) + \boldsymbol{s}_{\mathbf{0}_y} - \hat{\mathcal{R}}_y \boldsymbol{l}_{\boldsymbol{S}}\big).$$

Here we assume that `T` and `S` are the templates to which `X` and `Y` are respectively aligned, and that both `T` and `S` are distributed onto a processor mesh `P`. If $\hat{\mu} = \hat{\pi}_T \hat{\mathcal{R}}_x^T \hat{\mathcal{R}}_x \hat{\pi}_T^T$ and $\hat{\nu} = \hat{\pi}_S \hat{\mathcal{R}}_y^T \hat{\mathcal{R}}_y \hat{\pi}_S^T$, then

$$\begin{aligned} &\Delta_r(Y) \neq \emptyset \;\forall\; \mathbf{0} \leq \boldsymbol{r} < \hat{P}\mathbf{1} \\ &\wedge\, \hat{\nu} \leq \hat{\mu} \\ &\wedge\, \hat{\nu}(\lfloor \hat{C}_T^{-1}\boldsymbol{\zeta} \rfloor \bmod \hat{P}\mathbf{1}) = \lfloor \hat{C}_S^{-1}\boldsymbol{\xi} \rfloor \bmod \hat{P}\mathbf{1} \end{aligned} \tag{14}$$

is a sufficient condition for the array reference $Y(\hat{S}_y \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})$ to be identically mapped onto the same processor that owns $X(\hat{S}_x \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_x})$.

Fig. 16. Theorem 2.

should dispatch a section of its array to some other processor that views it exists, irrespective of whether data is actually communicated or not. This could result in the expensive runtime cost of communication checks, even in cases where it could be avoided such as when the elements of $X(\hat{S}_x \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_x})$ and $Y(\hat{S}_y \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})$ are ultimately mapped onto the same processor. If the compiler could detect such situations, it could refrain from generating communication code for such array reference pairs. This is what the mapping test attempts to do.

6.1 A Sufficient Condition

Thereom 2, shown in Fig. 16, states a sufficient condition for the mapping test that is proven below.

Proof. Suppose $X(\hat{S}_x \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_x}) \in \Delta''_p(X)$; as discussed earlier, a legal `ALIGN/DISTRIBUTE` combination will admit at least one such $\boldsymbol{p}$. Thus, there exists a $\boldsymbol{c}$ such that from the system in (11), we have

$$\begin{aligned} &\hat{\pi}_T \hat{\mathcal{R}}_x^T \big(\hat{\mathcal{A}}_x(\hat{S}_x \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_x}) + \boldsymbol{s}_{\mathbf{0}_x} - \hat{\mathcal{R}}_{\boldsymbol{x}} \boldsymbol{l}_{\boldsymbol{T}}\big) \\ &\quad = \hat{\pi}_T \hat{\mathcal{R}}_x^T \hat{\mathcal{R}}_x \hat{\pi}_T^T (\hat{C}_T \hat{P}\boldsymbol{c} + \hat{C}_T \boldsymbol{p}) + \boldsymbol{l}, \end{aligned} \tag{II.1}$$

where

$$\mathbf{0} \leq \boldsymbol{l} \leq \hat{\pi}_T \hat{\mathcal{R}}_x^T \hat{\mathcal{R}}_x \hat{\pi}_T^T (\hat{C}_T \mathbf{1} - \mathbf{1}). \tag{II.2}$$

By virtue of (11), $\hat{\mu}\boldsymbol{c} = \boldsymbol{c}$. Therefore, after some rearranging, (II.1) becomes $\hat{C}_T(\hat{P}\boldsymbol{c} + \hat{\mu}\boldsymbol{p}) + \boldsymbol{l} = \boldsymbol{\zeta}$. From (II.2), $\mathbf{0} \leq \hat{C}_T^{-1}\boldsymbol{l} \leq \hat{\mu}(\mathbf{1} - \hat{C}_T^{-1}\mathbf{1})$. Therefore,

$$\hat{P}\boldsymbol{c} + \hat{\mu}\boldsymbol{p} \leq \hat{P}\boldsymbol{c} + \hat{\mu}\boldsymbol{p} + \hat{C}_T^{-1}\boldsymbol{l} \leq \hat{P}\boldsymbol{c} + \hat{\mu}\boldsymbol{p} + \hat{\mu}(\mathbf{1} - \hat{C}_T^{-1}\mathbf{1}). \tag{II.3}$$

Since $\mathbf{1} \leq \hat{C}_T \mathbf{1}$, we have $\mathbf{0} < \hat{C}_T^{-1}\mathbf{1} \leq \mathbf{1}$. Thus,

$$\mathbf{0} \leq \mathbf{1} - \hat{C}_T^{-1}\mathbf{1} < \mathbf{1}. \tag{II.4}$$

Since $\hat{0} \leq \hat{\mu} \leq \hat{I}$ is always true, we get

$$\hat{0} \leq \hat{\mu}(\mathbf{1} - \hat{C}_T^{-1}\mathbf{1}) \leq \mathbf{1} - \hat{C}_T^{-1}\mathbf{1}$$

from the above. Using (II.4) again, we also get $\mathbf{0} \leq \hat{\mu}(\mathbf{1} - \hat{C}_T^{-1}\mathbf{1}) < \mathbf{1}$. Consequently, (II.3) becomes

$$\lfloor \hat{P}\boldsymbol{c} + \hat{\mu}\boldsymbol{p} \rfloor \leq \lfloor \hat{P}\boldsymbol{c} + \hat{\mu}\boldsymbol{p} + \hat{C}_T^{-1}\boldsymbol{l} \rfloor < \lfloor \hat{P}\boldsymbol{c} + \hat{\mu}\boldsymbol{p} + \mathbf{1} \rfloor,$$

or $\hat{P}\boldsymbol{c} + \hat{\mu}\boldsymbol{p} = \lfloor \hat{C}_T^{-1}\boldsymbol{\zeta} \rfloor$. Since $\mathbf{0} \leq \hat{\mu}\boldsymbol{p} \leq \boldsymbol{p} < \hat{P}\mathbf{1}$, we therefore have

$$\hat{\mu}\boldsymbol{p} = \lfloor \hat{C}_T^{-1}\boldsymbol{\zeta} \rfloor \bmod \hat{P}\mathbf{1}. \tag{II.5}$$

That is, if $X(\hat{S}_x \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_x}) \in \Delta''_p(X)$, then $\boldsymbol{p}$ must fulfill (II.5). Similarly, if $Y(\hat{S}_y \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y}) \in \Delta''_q(Y)$, then $\boldsymbol{q}$ must satisfy (II.6):

$$\hat{\nu}\boldsymbol{q} = \lfloor \hat{C}_S^{-1}\boldsymbol{\xi} \rfloor \bmod \hat{P}\mathbf{1}. \tag{II.6}$$

It is also given that $\Delta''_r(Y) \neq \emptyset$ for all $\mathbf{0} \leq \boldsymbol{r} < \hat{P}\mathbf{1}$. Thus, from Lemma 3.4, every $\boldsymbol{r}$ that fulfills (II.6) also owns $Y(\hat{S}_y \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y})$. We now need to verify whether a $\boldsymbol{p}$ that is a solution for (II.5) is also a solution for (II.6). Since we are given that $\hat{\nu}(\lfloor \hat{C}_T^{-1}\boldsymbol{\zeta} \rfloor \bmod \hat{P}\mathbf{1}) = \lfloor \hat{C}_S^{-1}\boldsymbol{\xi} \rfloor \bmod \hat{P}\mathbf{1}$, using (II.5) and (II.6) we get

$$\hat{\mu}\hat{\nu}\boldsymbol{p} = \hat{\nu}\boldsymbol{q}. \tag{II.7}$$

But we are also given that $\hat{\nu} \leq \hat{\mu}$. Therefore, multiplying by $\hat{\nu}$ and noting that $\hat{\nu}^2 = \hat{\nu}$, we get

$$\hat{\nu} \leq \hat{\mu}\hat{\nu}. \tag{II.8}$$

Again, since $\hat{\mu} \leq \hat{I}$, we get on multiplying by $\hat{\nu}$,

$$\hat{\mu}\hat{\nu} \leq \hat{\nu}. \tag{II.9}$$

Hence, (II.8) and (II.9) imply that $\hat{\mu}\hat{\nu} = \hat{\nu}$. Therefore, (II.7) becomes $\hat{\nu}\boldsymbol{p} = \hat{\nu}\boldsymbol{q}$. Thus, $\boldsymbol{p}$ also fulfills (II.6). In other words, $Y(\hat{S}_y \boldsymbol{\alpha} + \boldsymbol{a}_{\mathbf{0}_y}) \in \Delta''_p(Y)$ is also true. □

The sufficient condition can be established at compile time. Predicate (14) constitutes the actual mapping test. Verifying whether $\hat{\nu} \leq \hat{\mu}$ and

$$\hat{\nu}(\lfloor \hat{C}_T^{-1}\boldsymbol{\zeta} \rfloor \bmod \hat{P}\mathbf{1}) = \lfloor \hat{C}_S^{-1}\boldsymbol{\xi} \rfloor \bmod \hat{P}\mathbf{1}$$

is an $O(z)$ operation. Establishing whether $\Delta_r(Y) \neq \emptyset$ for all $\mathbf{0} \leq \boldsymbol{r} < \hat{P}\mathbf{1}$ is a polynomial-time operation, given the symbolic representation of the ownership set (see Section 4). Thus, the overall time complexity for verifying the requirements of Theorem 2 is polynomial once the FME solution system for the ownership set is known.

The impact of the mapping test on runtimes can often be dramatic. To illustrate the savings, the runtimes for the ADI benchmark, for arrays of sizes $4 \times 1024 \times 2$ on a

1×4 mesh of processors with and without this optimization were 0.51 and 64.79 seconds, respectively! The large value of 64.79 seconds arose due to three assignment statements that were the sinks of loop-independent flow dependencies that were enclosed within a triply nested loop spanning an iteration space of $2048 \times 2 \times 1022$ points. Each of these three assignment statements included right-hand side array references that were finally distributed onto the same processor as the corresponding left-hand side array reference. Hence, in all, 18 communication checks (nine for `MPI_SEND` and another nine for `MPI_RECV`) per iteration were eliminated.

6.2 The Matrix Multiplication Benchmark

To demonstrate how the mapping test works, we begin with the Matrix Multiplication benchmark (from the Livermore Kernel 21 [11]), in which each of the templates `T` and `S` are simultaneously collapsed and replicated. The mapping test correctly determines that none of the right-hand side array references in the benchmark's single assignment statement require any communication code to be generated.

Consider the pair of references `C(i, j)` and `A(i, k)` in Fig. 17. By inspection, we have

$$\hat{\mathcal{A}}_c = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{s_{0_c}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

$$\hat{\pi_T} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \hat{C_T} = \begin{pmatrix} 512 & 0 \\ 0 & 512 \end{pmatrix},$$

$$\hat{\mathcal{A}}_a = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{s_{0_a}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

$$\hat{\pi_S} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \hat{C_S} = \begin{pmatrix} 512 & 0 \\ 0 & 512 \end{pmatrix},$$

$$\hat{\mathcal{R}}_c = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{a_{l_c}} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \boldsymbol{a_{u_c}} = \begin{pmatrix} 1024 \\ 1024 \end{pmatrix},$$

$$\hat{\lambda_T} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{l_T} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \boldsymbol{u_T} = \begin{pmatrix} 1024 \\ 1024 \end{pmatrix},$$

$$\hat{\mathcal{R}}_a = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{a_{l_a}} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \boldsymbol{a_{u_a}} = \begin{pmatrix} 1024 \\ 1024 \end{pmatrix},$$

$$\hat{\lambda_S} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{l_S} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \boldsymbol{u_S} = \begin{pmatrix} 1024 \\ 1024 \end{pmatrix},$$

$$\hat{P} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.$$

Thus,

$$\boldsymbol{\zeta} = \hat{\pi}_T \hat{\mathcal{R}}_c^T \big(\hat{\mathcal{A}}_c(\hat{S}_c \boldsymbol{\alpha} + \boldsymbol{a_{0_c}}) + \boldsymbol{s_{0_c}} - \hat{\mathcal{R}}_c \boldsymbol{l_T}\big) = \begin{pmatrix} i-1 \\ j-1 \end{pmatrix},$$

$$\boldsymbol{\xi} = \hat{\pi}_S \hat{\mathcal{R}}_a^T \big(\hat{\mathcal{A}}_a(\hat{S}_a \boldsymbol{\alpha} + \boldsymbol{a_{0_a}}) + \boldsymbol{s_{0_a}} - \hat{\mathcal{R}}_a \boldsymbol{l_S}\big) = \begin{pmatrix} i-1 \\ 0 \end{pmatrix},$$

$$\hat{\mu} = \hat{\pi}_T \hat{\mathcal{R}}_c^T \hat{\mathcal{R}}_c \hat{\pi}_T^T = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

$$\hat{\nu} = \hat{\pi}_S \hat{\mathcal{R}}_a^T \hat{\mathcal{R}}_a \hat{\pi}_S^T = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

To verify the first requirement, we need to determine the integer FME solution system for $\Delta_p(A)$. By applying the FME solver with (c_1, c_2, a_1, a_2) as the elimination order, we find this to be

```
      program matmul

      REAL A(1024, 1024), B(1024, 1024), C(1024, 1024)
      INTEGER i, j, k

!HPF$ TEMPLATE S(1024, 1024)
!HPF$ TEMPLATE T(1024, 1024)
!HPF$ PROCESSORS P(2, 2)
!HPF$ ALIGN A(i, *) WITH S(i, *)
!HPF$ ALIGN B(*, j) WITH S(*, j)
!HPF$ ALIGN C(i, j) WITH T(i, j)
!HPF$ DISTRIBUTE S(BLOCK, BLOCK) ONTO P
!HPF$ DISTRIBUTE T(BLOCK, BLOCK) ONTO P

!C
!C    We assume that the matrices A, B and C
!C    are already initialized.
!C

      do j = 1, 1024
         do k = 1, 1024
            do i = 1, 1024
               C(i, j) = C(i, j)+A(i, k)*B(k, j)
            enddo
         enddo
      enddo

      end
```

Fig. 17. Matrix multiplication.

$$\begin{aligned}
&\Delta_p(A) = \Big\{ \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \Big| \exists \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \text{ such that} \\
&\Big\lceil \max\Big(0, -\frac{511}{1024} - \frac{p_1}{2}, \frac{-p_1}{2}\Big) \Big\rceil \le c_1 \le \Big\lfloor \min\Big(0, \frac{1023}{1024} - \frac{p_1}{2}\Big) \Big\rfloor, \\
&\Big\lceil \max\Big(0, \frac{-p_2}{2}\Big) \Big\rceil \le c_2 \le \Big\lfloor \min\Big(0, \frac{1023}{1024} - \frac{p_2}{2}\Big) \Big\rfloor, \\
&\max(1, 1 + 1024c_1 + 512p_1) \le a_1 \\
&\quad \le \min\big(1024, 512(1 + 2c_1 + p_1)\big), \\
&1 \le a_2 \le 1024 \Big\}.
\end{aligned} \tag{15}$$

There are two constraints in (15) that contain p_1 in their bound expressions and another constraint that contains p_2 in its bound expressions. Hence, by separately considering two disjoint inequality groups, we can determine in polynomial time that $\Delta_r(A) \neq \emptyset$ for all $\boldsymbol{0} \le \boldsymbol{r} < \hat{P}\boldsymbol{1}$. Further, since $\hat{\nu} \le \hat{\mu}$, the second requirement of the mapping test is also satisfied. In addition,

$$\lfloor \hat{C}_T^{-1} \boldsymbol{\zeta} \rfloor = \begin{pmatrix} \lfloor \frac{i-1}{512} \rfloor \\ \lfloor \frac{j-1}{512} \rfloor \end{pmatrix}, \qquad \lfloor \hat{C}_S^{-1} \boldsymbol{\xi} \rfloor = \begin{pmatrix} \lfloor \frac{i-1}{512} \rfloor \\ 0 \end{pmatrix}.$$

Therefore, the third requirement is also satisfied since,

$$\hat{\nu}(\lfloor \hat{C}_T^{-1} \boldsymbol{\zeta} \rfloor \bmod \hat{P}\boldsymbol{1}) = \begin{pmatrix} \lfloor \frac{i-1}{512} \rfloor \bmod 2 \\ 0 \end{pmatrix},$$

$$\lfloor \hat{C}_S^{-1} \boldsymbol{\xi} \rfloor \bmod \hat{P}\boldsymbol{1} = \begin{pmatrix} \lfloor \frac{i-1}{512} \rfloor \bmod 2 \\ 0 \end{pmatrix}.$$

We can thus conclude that `A(i, k)` is identically mapped onto the same processor that owns `C(i, j)`, for all values

```
      REAL X(682, 4:521, 512), Y(1019, 2:120, 508, 32)
      INTEGER i, j, k

!HPF$ TEMPLATE T(2:2048, 8:1042)
!HPF$ TEMPLATE S(0:1023, 0:10, 0:250, 0:512, 0:100)
!HPF$ PROCESSORS P(8, 9)
!HPF$ ALIGN X(i, j, *) WITH T(3*i, 2*j)
!HPF$ ALIGN Y(i, j, k, l) WITH S(i+3, *, 2*j+2, k+3, 3*l+4)
!HPF$ DISTRIBUTE T(CYCLIC(32), CYCLIC(16)) ONTO P
!HPF$ DISTRIBUTE S(BLOCK, *, *, CYCLIC(16), *) ONTO P

      do k = ...
         do j = ...
            do i = ...
               X(470, j+i+7, 3k+2i) = Y(509, i+7k, 2j+2i+3, 7k)
            enddo
         enddo
      enddo
```

Fig. 18. Synthetic code.

of the loop iteration vector $\boldsymbol{\alpha}$. We can similarly show that `B(k, j)` is also identically mapped onto the same processor that owns `C(i, j)`, for all iterations of the loop nest. Since the third right-hand side array reference `C(i, j)` is trivially mapped onto the same processor as the left-hand side array reference, no communication code needs to be generated for this benchmark.

How does the replication test fare on this benchmark? Note that the Boolean predicate $\hat{\pi}_S\hat{\mathcal{R}}_a^T\hat{\mathcal{R}}_a\hat{\pi}_S^T(\boldsymbol{p}-\boldsymbol{q}) \neq \mathbf{0}$ simplifies to $p_1 \neq q_1$. Thus, though the replication test avoids *some* of the communication check overhead in the send and receive actions shown in Fig. 13, the send and receive sets would still be computed and checked whenever p_1 and q_1 are not equal. However, because of the nature of the affine subscript expressions of the array references involved (i.e., `C(i, j)` and `A(i, k)`), the send and receive sets evaluate to empty sets even when p_1 and q_1 are not equal. Thus, in the case of this benchmark, the entire overhead of performing communication checks at runtime can be avoided which is exactly what the mapping test detects.

6.3 A Synthetic Example

Consider the synthetic code shown in Fig. 18. For this fragment, the various matrices and vectors required by the mapping test are

$$\boldsymbol{\zeta} = \begin{pmatrix} 1408 \\ 2i+2j+6 \end{pmatrix}, \qquad \boldsymbol{\xi} = \begin{pmatrix} 512 \\ 2i+2j+6 \end{pmatrix},$$

$$\hat{\mu} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \hat{\nu} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

$$\hat{\nu}(\lfloor \hat{C}_T^{-1}\boldsymbol{\zeta} \rfloor \bmod \hat{P}\mathbf{1}) = \begin{pmatrix} 4 \\ \lfloor \frac{2i+2j+6}{16} \rfloor \bmod 9 \end{pmatrix},$$

$$\lfloor \hat{C}_S^{-1}\boldsymbol{\xi} \rfloor \bmod \hat{P}\mathbf{1} = \begin{pmatrix} 4 \\ \lfloor \frac{2i+2j+6}{16} \rfloor \bmod 9 \end{pmatrix}.$$

Since $\hat{\nu} = \hat{\mu}$ and $\hat{\nu}(\lfloor \hat{C}_T^{-1}\boldsymbol{\zeta} \rfloor \bmod \hat{P}\mathbf{1}) = \lfloor \hat{C}_S^{-1}\boldsymbol{\xi} \rfloor \bmod \hat{P}\mathbf{1}$, the second and third requirements of the mapping test are satisfied. By applying the FME solver with $(c_1, c_2, a_1, a_2, a_3, a_4)$ as the elimination order, we find the integer FME solution system for $\Delta_p(Y)$ to be

$$\begin{aligned}
\Delta_p(Y) = \Bigg\{ &\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} \Bigg| \exists \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \text{ such that} \\
&\lceil \max(0, -\frac{127}{1024} - \frac{p_1}{8}, -\frac{123}{1024} - \frac{p_1}{8}, \frac{-p_1}{8}) \rceil \leq c_1 \\
&\quad \leq \lfloor \min(0, \frac{511}{512} - \frac{p_1}{8}, \frac{1023}{1024} - \frac{p_1}{8}) \rfloor, \\
&\lceil \max(-\frac{5}{48} - \frac{p_2}{9}, -\frac{11}{144} - \frac{p_2}{9}, \frac{-p_2}{9}) \rceil \leq c_2 \\
&\quad \leq \lfloor \min(\frac{511 - 16p_2}{144}, \frac{32 - p_2}{9}) \rfloor, \\
&\max(1, -3 + 1024c_1 + 128p_1) \leq a_1 \\
&\quad \leq \min\big(1019, 4(31 + 256c_1 + 32p_1)\big), 2 \leq a_2 \leq 120, \\
&\max(1, -3 + 144c_2 + 16p_2) \leq a_3 \\
&\quad \leq \min\big(508, 4(3 + 36c_2 + 4p_2)\big), 1 \leq a_4 \leq 32 \Big\}.
\end{aligned} \tag{16}$$

Equation (16) consists of two disjoint inequality groups; we can therefore determine in polynomial time that $\Delta_r(Y) \neq \emptyset$ for all $\mathbf{0} \leq \boldsymbol{r} < \hat{P}\mathbf{1}$. Therefore, from Theorem 2, the right-hand side array reference is identically mapped onto the same processor that owns the left-hand side array reference. This is indeed the case as can be verified by visualizing the alignments and distributions. Notice that terms unknown at compile time are only manipulated symbolically by the mapping test. For instance, if instead of

$$\mathtt{Y(509, i + 7k, 2j + 2i + 3, 7k)},$$

we had

$$\mathtt{Y(509, i + 7k, 3j + 2i + 3, 7k)},$$

then

$$\lfloor \hat{C}_S^{-1}\boldsymbol{\xi} \rfloor \bmod \hat{P}\mathbf{1} = \begin{pmatrix} 4 \\ \lfloor \frac{3i+2j+6}{16} \rfloor \bmod 9 \end{pmatrix},$$

```
      program adi

      REAL DU1(1024), DU2(1024), DU3(1024)
      REAL AU1(4, 1024, 2), AU2(4, 1024, 2), AU3(4, 1024, 2)
      INTEGER l, kx, ky
      ...
!HPF$ TEMPLATE T(4, 1024, 2)
!HPF$ ALIGN DU1(i) WITH T(*, i, *)
!HPF$ ALIGN AU1(i, j, k) WITH T(i, j, k)
!HPF$ PROCESSORS P(1, 16)
!HPF$ DISTRIBUTE T(BLOCK, CYCLIC(62), *) ONTO P

!C
!C    Initializations.
!C
      ...
!C
!C    The ADI kernel.
!C

      do l = 1, 2048
         do kx = 2, 3
            do ky = 2, 1023
               DU1(ky) = AU1(kx, ky+1, 1)-AU1(kx, ky-1, 1)
               ...
               AU1(kx, ky, 2) = AU1(kx, ky, 1)+a11*DU1(ky)+a12*DU2(ky)+
     1         a13*DU3(ky)+sig*(AU1(kx+1, ky, 1)-2*AU1(kx, ky, 1)+AU1(k
     2         x-1, ky, 1))
               ...
            enddo
         enddo
      enddo

      end
```

Fig. 19. ADI.

and the test would have failed since nothing can be said at compile time about the equality of $(\lfloor(2i+2j+6)/16\rfloor \bmod 9)$ and $(\lfloor(3i+2j+6)/16\rfloor \bmod 9)$. For some values of i and j this may be true (say, $i=3, j=2$) while not for others (say, $i=2, j=2$). Thus, in such a situation, the test fails and, conservatively, communication code is generated.

6.4 The ADI Benchmark

The last example shows how the mapping test makes it possible for the compiler to avoid generating any communication code for the Automatic Differentiation and Integration (ADI) benchmark (from the Livermore Kernel 8 [11]), when the alignments and distributions are suitably chosen.

In essence, the ADI benchmark comprises six arrays of type REAL, of which three are one-dimensional and consist of 1,024 elements each, while the remaining three are three-dimensional arrays consisting of $4 \times 1024 \times 2$ elements each. The only flow dependencies that this benchmark exhibits are three loop-independent ones and all of the source-sink statement pairs are contained in the innermost loop. The right-hand side array references on which these dependencies terminate can be aligned and distributed onto the same processors that own the corresponding left-hand side array references and it is this particular case that the mapping test optimizes.

In the code excerpt shown in Fig. 19, only one source-sink statement pair among the three is indicated. The other two are similar with the array DU1 replaced, respectively, by DU2 and DU3 in the source statements and with the array AU1 replaced, respectively, by AU2 and AU3 in the source and sink statements. We now consider the pair of array references AU1(kx, ky, 2) and DU1(ky) occurring in the second assignment statement of Fig. 19. Thus,

$$\zeta = \begin{pmatrix} kx-1 \\ ky-1 \end{pmatrix}, \xi = \begin{pmatrix} 0 \\ ky-1 \end{pmatrix}, \hat{\mu} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \hat{\nu} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

Since

$$\begin{aligned} \Delta''_p(DU1) = \{(a_1) | \exists \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} &\text{such that} \\ &\max(0, -p_1) \leq c_1 \leq \lfloor \min(0, \frac{3}{4} - p_1) \rfloor, \\ &\lceil \max(-\frac{61}{992} - \frac{p_2}{16}, \frac{-p_2}{16}) \rceil \leq c_2 \leq \lfloor \frac{33 - 2p_2}{32} \rfloor, \\ &\max(1, 1 + 992c_2 + 62p_2) \leq a_1 \\ &\quad \leq \min\big(1024, 62(1 + 16c_2 + p_2)\big)\big\}, \end{aligned}$$

we see that the first requirement is fulfilled. In addition, since $\hat{\nu} \leq \hat{\mu}$, the second requirement is also satisfied. Finally,

$$\lfloor \hat{C}_T^{-1} \zeta \rfloor = \begin{pmatrix} \lfloor \frac{kx-1}{4} \rfloor \\ \lfloor \frac{ky-1}{62} \rfloor \end{pmatrix}, \qquad \lfloor \hat{C}_S^{-1} \xi \rfloor = \begin{pmatrix} 0 \\ \lfloor \frac{ky-1}{62} \rfloor \end{pmatrix}.$$

Hence,

```
!Alignment directives for ADI.
!HPF$ TEMPLATE T(4, 1024, 2)
!HPF$ ALIGN DU1(i) WITH T(*, i, *)
!HPF$ ALIGN DU2(i) WITH T(*, i, *)
!HPF$ ALIGN DU3(i) WITH T(*, i, *)
!HPF$ ALIGN AU1(i, j, k) WITH T(i, j, k)
!HPF$ ALIGN AU2(i, j, k) WITH T(i, j, k)
!HPF$ ALIGN AU3(i, j, k) WITH T(i, j, k)
```

```
!Alignment directives for EFLUX.
!HPF$ TEMPLATE T(0:5001, 34, 4)
!HPF$ ALIGN FS(i, j, k) WITH T(i, j, k)
!HPF$ ALIGN DW(i, j, k) WITH T(i, j, k)
!HPF$ ALIGN W(i, j, k) WITH T(i, j, k)
!HPF$ ALIGN X(i, j, k) WITH T(i, j, k)
!HPF$ ALIGN P(i, j) WITH T(i, j, *)
```

```
!Alignment directives for MATMUL.
!HPF$ TEMPLATE S(1024, 1024)
!HPF$ TEMPLATE T(1024, 1024)
!HPF$ ALIGN A(i, *) WITH S(i, *)
!HPF$ ALIGN B(*, j) WITH S(*, j)
!HPF$ ALIGN C(i, j) WITH T(i, j)
```

Fig. 20. Alignment directives for the ADI, Euler Fluxes, and Matrix Multiplication benchmarks.

$$\hat{\nu}(\lfloor \hat{C}_T^{-1}\boldsymbol{\zeta} \rfloor \bmod \hat{P}\mathbf{1}) = \begin{pmatrix} 0 \\ \lfloor \frac{ky-1}{62} \rfloor \bmod 16 \end{pmatrix},$$

$$\lfloor \hat{C}_S^{-1}\boldsymbol{\xi} \rfloor \bmod \hat{P}\mathbf{1} = \begin{pmatrix} 0 \\ \lfloor \frac{ky-1}{62} \rfloor \bmod 16 \end{pmatrix}.$$

Thus, the third requirement is also satisfied. We can therefore conclude that `DU1(ky)` gets identically mapped onto the same processor that owns `AU1(kx, ky, 2)` for all values of the loop iteration vector $\boldsymbol{\alpha}$. It can be similarly shown that for the same alignment and distribution directives, the right-hand side array references in the other assignment statements on which flow dependencies terminate in this benchmark are also identically mapped onto the same processor as the respective left-hand side array references. Thus, no communication code needs to be generated for the benchmark.

7 Mapping Test Measurements

Execution times and compilation times were measured for the PARADIGM (version 2.0) system with and without the mapping test optimization. For the sake of comparison, execution times and compilation times for the original sequential sources and the parallelized codes generated by `pghpf` (version 2.4) and `xlhpf` (version 1.03) were also recorded. `pghpf` and `xlhpf` are commercial HPF compilers from the Portland Group Inc., (PGI) and the International Business Machines (IBM), respectively. In the input codes to the `pghpf` compiler, `DO` loops were recast into `FORALL` equivalents where possible and were qualified with the `INDEPENDENT` directive when appropriate. The `FORALL` construct and the `INDEPENDENT` directive were not mixed in the inputs to `pghpf` and the tabulated execution times correspond to the best of the two cases. All of the PARADIGM measurements were done in the presence of the replication test.

7.1 System Specifications

The IBM compilers `xlf` and `mpxlf` were used to handle Fortran 77 and Fortran 77+MPI sources, respectively. The HPF sources were compiled using `xlhpf` and `pghpf`. The `-O` option, which results in the generation of optimized code, was always used during compilations done with `xlf`, `xlhpf`, `mpxlf`, and `pghpf`. Compilation times were obtained by considering the source-to-source transformation effected by PARADIGM, *as well as* the source-to-executable compilation done using `mpxlf` (version 5.01). The source-to-source compilation times for PARADIGM were measured on an HP Visualize C180 with a 180MHz HP PA-8000 CPU running HP-UX 10.20 and having 128MB of RAM. Compilation times for `pghpf`, `xlhpf` as well as `mpxlf` were measured on an IBM E30 running AIX 4.3 and having a 133MHz PowerPC 604 processor and 96MB of main memory. In those tables that tabulate the execution times, the RS6000 column refers to the sequential execution times obtained on the IBM E30. The parallel codes were executed on a 16-node IBM SP2 multicomputer running AIX 4.3 and in which each processor was a 62.5MHz POWER node having 128MB of RAM. Interprocessor communication on the IBM SP2 was across a high performance adapter switch.

7.2 Alignments and Distributions

Measurements for the mapping test were taken across three benchmarks: ADI, Euler Fluxes (from FLO52 in the Perfect Club Suite) and Matrix Multiplication. For all the input samples, fixed templates and alignments were chosen; these are shown in Fig. 20. Note that the most suitable alignments were chosen for the benchmark input samples. For the Matrix Multiplication and ADI benchmarks, these alignments resulted in communication-free programs independent of the distributions. For the Euler Fluxes benchmark, the distributions resulted in varying amounts of communication.

The `PROCESSORS` and the `DISTRIBUTE` directives were changed in every benchmark's input sample. The various distributions were chosen *arbitrarily*, the idea being to demonstrate the ability of the mapping test to handle any given alignment/distribution combination.

TABLE 2
Execution Times in Seconds

Benchmark	Processor Mesh	Distribution	IBM AIX 4.3				
			RS6000	SP2			
			`xlf` v 5.01	`pdm` v 2.0		`pghpf` v 2.4	`xlhpf` v 1.03
				Optimized	Nonoptimized		
ADI	1×2	`(BLOCK, BLOCK, *)`	7.07	1.41	83.57	16.78	3.36
	1×4	`(BLOCK, BLOCK, *)`	6.79	0.51	64.79	12.67	2.35
	1×8	`(BLOCK, CYCLIC(120), *)`	7.90	4.34	609.44	20.94	-[k]
EFLUX	1×2	`(BLOCK, BLOCK, *)`	71.85	19.03	19.44	231.67	33.79
	2×2	`(BLOCK, *, CYCLIC)`	71.40	13.47	13.83	274.93	3113.51
	8×1	`(*, BLOCK, CYCLIC)`	71.49	5.96	6.39	91.83	8.17
MATMUL	2×1	`(BLOCK, BLOCK)`[l]	12.65	2.47	5.83	57.48	25.88
	2×2	`(BLOCK, BLOCK)`[m]	104.96	10.06	17.20	224.29	100.37
	4×2	`(BLOCK, CYCLIC(120))`[m]	104.74	5.71	205.00	123.59	-[k]

k. `xlhpf` does not permit a CYCLIC blocking factor greater than 1.
l. Arrays were of type REAL; array sizes were 512×512.
m. Arrays were of type REAL; array sizes were 1024×1024.

7.3 Analysis

As Table 2 reveals, the benefits of the mapping test were most pronounced for the ADI benchmark, followed by the Matrix Multiplication benchmark. In the case of the Euler Fluxes benchmark, the mapping test eliminated six communication checks per iteration for the first input sample and eight communication checks per iteration for the second and third input samples. In the absence of the mapping test, 10 communication checks per iteration were generated. On account of loop-carried flow dependencies, the associated communication codes were hoisted immediately within the outermost loop. However, since the number of iterations of the outermost loop was a mere 100, the optimized compiled codes did not exhibit any significant improvement in runtimes. For all of the ADI benchmark input samples, the iteration space comprised of $2048 \times 2 \times 1022$ points, and the communication codes generated in the absence of the mapping test were hoisted immediately within the innermost loop. For the three Matrix Multiplication benchmark samples, the number of iteration points were $512 \times 512 \times 512, 1,024 \times 1,024 \times 1,024$, and $1,024 \times 1,024 \times 1,024$, respectively, and the single communication check that was generated in the absence of the mapping test was hoisted within the second innermost loop.

Given a sequential input source written using Fortran 77 and having HPF directives, PARADIGM produces an SPMD output consisting of Fortran 77 statements and procedure calls to the MPI library. The compilation of this SPMD code into the final executable is then performed using `mpxlf`. Since the mapping test eliminates the generation of communication code, where possible, it also exerts an influence on the *overall* compilation times. That is, the application of the mapping test often results in the generation of a smaller intermediate SPMD code, and this improves on the back-end source-to-executable compilation time. In our setup, this was done using `mpxlf`. Note that applying the mapping test does not necessarily mean an increased time for the source-to-source compilation phase performed by PARADIGM. This is because, though compilation in the presence of the mapping test involves the additional effort of identifying the candidate array reference pairs that are identically mapped, it, however, saves on the communication code generation part that would otherwise have to be done for the same array reference pairs. Hence, compilation times for the source-to-source compilation phase may in fact be more in the absence of the mapping test and this was found to be true for nearly all of the benchmark samples tested. However, as Table 3 also reveals, the back-end compilation times were nearly always more in the absence of the mapping test and this was because of the larger intermediate SPMD code sizes handled.

8 Summary

The preceding sections have shown certain basic and interesting properties that ownership sets exhibit, even in the presence of arbitrary alignments and distributions. Our approach to solving for the ownership set (and other sets derived from it) is based on integer FME solutions to the systems characterizing these sets. We also showed how the system of equalities and inequalities originally proposed in [1] can be refined to a form requiring the course vector as the only auxiliary vector. This refinement is beneficial to the FME approach. The fundamental property of ownership set equivalence is derived and we demonstrated how it can be used to eliminate redundant communication due to array replication. We also briefly described how to efficiently make decisions regarding the "emptiness" of an ownership set. Finally, we derived a sufficient condition that, when true, ensures that a right-hand side array reference of an assignment statement is available on the same processor that owns the left-hand side array reference, thus, making it possible to avoid generating communication code for the pair.

The mapping test is a very useful optimization. Its positive effect was observable in the case of other benchmarks such as Jacobi, TOMCATV, and 2D Explicit Hydrodynamics (from the Livermore Kernel 18), and

TABLE 3
Compilation Times in Seconds

Benchmark	Processor Mesh	Distribution	pdm v 2.0		mpxlf v 5.01		pghpf v 2.4	xlhpf v 1.03
			Optimized	Nonoptimized	Optimized	Nonoptimized		
ADI	1×2	(BLOCK, BLOCK, *)	6.00	9.00	2.00	7.00	7.00	5.00
	1×4	(BLOCK, BLOCK, *)	7.00	8.00	2.00	7.00	8.00	5.00
	1×8	(BLOCK, CYCLIC(120), *)	7.00	11.00	2.00	17.00	8.00	-[k]
EFLUX	1×2	(BLOCK, BLOCK, *)	11.00	12.00	6.00	8.00	7.00	6.00
	2×2	(BLOCK, *, CYCLIC)	10.00	14.00	4.00	11.00	9.00	12.00
	8×1	(*, BLOCK, CYCLIC)	11.00	12.00	6.00	22.00	9.00	5.00
MATMUL	2×1	(BLOCK, BLOCK)[l]	4.00	5.00	2.00	2.00	5.00	1.00
	2×2	(BLOCK, BLOCK)[m]	4.00	5.00	2.00	3.00	6.00	1.00
	4×2	(BLOCK, CYCLIC(120))[m]	5.00	5.00	2.00	3.00	6.00	-[k]

k. `xlhpf` does not permit a CYCLIC blocking factor greater than 1.
l. Arrays were of type REAL; array sizes were 512×512.
m. Arrays were of type REAL; array sizes were 1024×1024.

was significant in most situations. This was on account of the fact that, typically, suitably chosen `ALIGN` and `DISTRIBUTE` directives perfectly align and distribute at least one pair of left-hand side and right-hand side array references in at least one assignment statement in the Lprogram and such alignments and distributions are often valid independent of the values through which the loop iteration vector ranges. Thus, by efficiently exploiting the ownership set, efficient SPMD code can be generated efficiently at compile time.

APPENDIX

LEMMA PROOFS

Proof for Lemma 3.1. Suppose $\boldsymbol{a}' \in \Delta'_p(X)$. Then, there exists a $\boldsymbol{c}'$, $\boldsymbol{t}'$ such that

$$\hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} \leq \hat{\pi}\boldsymbol{t}' < \hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} + \hat{C}\mathbf{1}.$$

Hence, $\mathbf{0} \leq \hat{\pi}\boldsymbol{t}' - (\hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p}) < \hat{C}\mathbf{1}$. Setting

$$\boldsymbol{l}' = \hat{\pi}\boldsymbol{t}' - (\hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p}),$$

we get $\hat{\pi}\boldsymbol{t}' = \hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} + \boldsymbol{l}'$. Consequently, $\boldsymbol{a}' \in \Delta_p(X)$ because $\boldsymbol{c}'$,$\boldsymbol{l}'$, $\boldsymbol{t}'$, together with $\boldsymbol{a}'$, satisfy the system in (8). Hence,

$$\Delta'_p(X) \subseteq \Delta_p(X). \tag{3.1.1}$$

Similarly, if $\boldsymbol{a} \in \Delta_p(X)$, then there exists a $\boldsymbol{c}$, $\boldsymbol{l}$, and $\boldsymbol{t}$ that, together with $\boldsymbol{a}$, satisfy the system in (8). Since

$$\hat{\pi}\boldsymbol{t} = \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} + \boldsymbol{l},$$
$$\mathbf{0} \leq \boldsymbol{l} < \hat{C}\mathbf{1},$$

we get $\hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} \leq \hat{\pi}\boldsymbol{t} < \hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} + \hat{C}\mathbf{1}$. Thus,

$$\boldsymbol{a} \in \Delta'_p(X)$$

because $\boldsymbol{c}$, $\boldsymbol{a}$, and $\boldsymbol{t}$ satisfy the system in (9). Therefore,

$$\Delta_p(X) \subseteq \Delta'_p(X). \tag{3.1.2}$$

Relations (3.1.1) and (3.1.2), therefore, imply that

$$\Delta'_p(X) = \Delta_p(X).$$

□

Proof for Lemma 3.2. Suppose $\boldsymbol{a}^* \in \Delta^*_p(X)$; assume $\boldsymbol{c}^*$ to be the corresponding course vector for which (10) holds for $\boldsymbol{a}^*$. Consider

$$\boldsymbol{t}^* = \hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}) + (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}).$$

Then,[4]

$$\hat{\mathcal{R}}\boldsymbol{t}^* = \hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}. \tag{3.2.1}$$

Next, from (10), we have, after premultiplying throughout by $\hat{\pi}$,

$$\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}) \leq \hat{\pi}\big(\boldsymbol{t}^* - (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p})\big)$$
$$\leq \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p} + \hat{C}\mathbf{1} - \mathbf{1}).$$

This gives us

$$\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p} \leq \hat{\pi}\boldsymbol{t}^* \leq \hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p} + \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\mathbf{1} - \mathbf{1}).$$

Now, $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T \leq \hat{I}$. Since $\mathbf{0} \leq \hat{C}\mathbf{1} - \mathbf{1}$, we get

$$\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\mathbf{1} - \mathbf{1}) \leq \hat{C}\mathbf{1} - \mathbf{1}.$$

Hence,

$$\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p} \leq \hat{\pi}\boldsymbol{t}^* < \hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p} + \hat{C}\mathbf{1}. \tag{3.2.2}$$

From (10), we also have

$$\mathbf{0} \leq \hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T} \leq \hat{\mathcal{R}}(\boldsymbol{u_T} - \boldsymbol{l_T}).$$

Thus, $\mathbf{0} \leq \hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}) \leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}(\boldsymbol{u_T} - \boldsymbol{l_T})$. That is,

4. Note that $\hat{\mathcal{R}}\hat{\mathcal{R}}^T = \hat{I}$, where $\hat{I}$ is the $m \times m$ identity matrix. However, $\hat{\mathcal{R}}^T\hat{\mathcal{R}}$ is a $y \times y$ matrix in which the only nonzero elements are the principal diagonal elements; the ith principal diagonal element is 1 if and only if the ith template dimension is an aligned dimension. Similarly, $\hat{\pi}\hat{\pi}^T$ is the $z \times z$ identity matrix whereas $\hat{\pi}^T\hat{\pi}$ is a $y \times y$ matrix in which the only nonzero elements are the principal diagonal elements; the ith princial diagonal element is 1 if and only if the ith template dimension is a distributed dimension. Square matrices in which the only nonzero elements are those that reside on the principal diagonal are usually termed *square diagonal*.

$$(\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}) \leq \hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}) + (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}),$$

$$\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}) + (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}) \leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}(\boldsymbol{u_T} - \boldsymbol{l_T}) + (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}).$$

Since $\boldsymbol{0} \leq \hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}$ and because $\hat{0} \leq \hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}}$, we get $\boldsymbol{0} \leq (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p})$. Taking this and (3.2.1) into account, the previous inequalities become

$$\boldsymbol{0} \leq \boldsymbol{t}^* \leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}(\boldsymbol{u_T} - \boldsymbol{l_T}) + (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}).$$

But, from (10),

$$(\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}) \leq (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T\hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}).$$

Thus, $\boldsymbol{0} \leq \boldsymbol{t}^* \leq (\hat{\mathcal{R}}^T\hat{\mathcal{R}} + (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T\hat{\pi})(\boldsymbol{u_T} - \boldsymbol{l_T})$. Again, since

$$\hat{0} \leq \hat{\mathcal{R}}^T\hat{\mathcal{R}} + (\hat{I} - \hat{\mathcal{R}}^T\hat{\mathcal{R}})\hat{\pi}^T\hat{\pi} \leq \hat{I}$$

and because $\boldsymbol{u_T} - \boldsymbol{l_T} \geq \boldsymbol{0}$, the previous constraint becomes:

$$\boldsymbol{0} \leq \boldsymbol{t}^* \leq \boldsymbol{u_T} - \boldsymbol{l_T}. \tag{3.2.3}$$

Also, from (10), we have

$$\hat{\lambda}\boldsymbol{c}^* = \boldsymbol{0}, \tag{3.2.4}$$

$$\boldsymbol{a_l} \leq \boldsymbol{a}^* \leq \boldsymbol{a_u}. \tag{3.2.5}$$

Thus, because (3.2.1), (3.2.4), (3.2.2), (3.2.3), and (3.2.5) satisfy the system in (9), $\boldsymbol{a}^* \in \Delta'_p(X)$. Hence,

$$\Delta^*_p(X) \subseteq \Delta'_p(X). \tag{3.2.6}$$

Similarly, if $\boldsymbol{a}' \in \Delta'_p(X)$, then there exists a $\boldsymbol{c}'$, $\boldsymbol{t}'$ that together with $\boldsymbol{a}'$ satisfy the system in (9). Therefore, $\hat{\mathcal{R}}\boldsymbol{t}' = \hat{\mathcal{A}}\boldsymbol{a}' + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}$. Hence,

$$\hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\boldsymbol{t}' = \hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}' + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}).$$

Since, from (9), $\hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} \leq \hat{\pi}\boldsymbol{t}' < \hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1}$, we get

$$\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p}) \leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{\pi}\boldsymbol{t}' \leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1} - \boldsymbol{1}).$$

Now, $\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{\pi} = \hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}$.[5] Therefore,

$$\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p}) \leq \hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}' + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}) \leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1} - \boldsymbol{1}). \tag{3.2.7}$$

From (9), we also have the following:

$$\hat{\pi}\boldsymbol{t}' - \hat{C}\boldsymbol{1} < \hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} \leq \hat{\pi}\boldsymbol{t}',$$
$$\boldsymbol{0} \leq \hat{\pi}\boldsymbol{t}' \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}).$$

From the above two inequalities, we get

$$-\hat{C}\boldsymbol{1} + \boldsymbol{1} \leq \hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}).$$

Consider the inequality $-\hat{C}\boldsymbol{1} + \boldsymbol{1} \leq \hat{C}(\hat{P}\boldsymbol{c}' + \boldsymbol{p})$. Since $\hat{C}$ always has a well-defined inverse $\hat{C}^{-1}$ and because $\hat{C}^{-1} > \hat{0}$ is always true, we can premultiply

$$-\hat{C}\boldsymbol{1} + \boldsymbol{1} \leq \hat{C}(\hat{P}\boldsymbol{c}' + \boldsymbol{p})$$

throughout by $\hat{C}^{-1}$ to obtain

$$\hat{C}^{-1}\boldsymbol{1} - \boldsymbol{1} - \boldsymbol{p} \leq \hat{P}\boldsymbol{c}'.$$

Since $\boldsymbol{p} < \hat{P}\boldsymbol{1}$ and because $\hat{P}$ also has a well-defined inverse $\hat{P}^{-1}$ that satisfies $\hat{P}^{-1} > \hat{0}$, we get

$$\hat{P}^{-1}(\hat{C}^{-1}\boldsymbol{1} - \boldsymbol{1} - \hat{P}\boldsymbol{1} + \boldsymbol{1}) \leq \hat{P}^{-1}\hat{P}\boldsymbol{c}'.$$

The last inequality simplifies to

$$\hat{P}^{-1}\hat{C}^{-1}\boldsymbol{1} - \boldsymbol{1} \leq \boldsymbol{c}'.$$

This clearly implies that the elements of $\boldsymbol{c}'$ are restricted to nonnegative integers only. Taken along with the fact that $\boldsymbol{p} \geq \boldsymbol{0}$, we get $\hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} \geq \boldsymbol{0}$. Thus,

$$\boldsymbol{0} \leq \hat{C}\hat{P}\boldsymbol{c}' + \hat{C}\boldsymbol{p} \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}). \tag{3.2.8}$$

Since, from (9), $\boldsymbol{0} \leq \boldsymbol{t}' \leq \boldsymbol{u_T} - \boldsymbol{l_T}$, we get

$$\boldsymbol{0} \leq \hat{\mathcal{R}}\boldsymbol{t}' \leq \hat{\mathcal{R}}(\boldsymbol{u_T} - \boldsymbol{l_T}).$$

Therefore,

$$\boldsymbol{0} \leq \hat{\mathcal{A}}\boldsymbol{a}' + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T} \leq \hat{\mathcal{R}}(\boldsymbol{u_T} - \boldsymbol{l_T}). \tag{3.2.9}$$

Again, from (9), we have:

$$\hat{\lambda}\boldsymbol{c}' = \boldsymbol{0}, \tag{3.2.10}$$

$$\boldsymbol{a_l} \leq \boldsymbol{a}' \leq \boldsymbol{a_u}. \tag{3.2.11}$$

Since (32.10), (3.2.7), (3.2.8), (3.2.9), and (3.2.11) fulfill the system in (10), $\boldsymbol{a}' \in \Delta^*_p(X)$. Thus,

$$\Delta'_p(X) \subseteq \Delta^*_p(X). \tag{3.2.12}$$

Relations (3.2.6) and (3.2.12), taken together, imply that

$$\Delta'_p(X) = \Delta^*_p(X).$$

□

Proof for Lemma 3.3. Suppose $\boldsymbol{a}'' \in \Delta''_p(X)$. Therefore, there exists a $\boldsymbol{c}''$ that, together with $\boldsymbol{a}''$, fulfills the system in (11). Since every equality and inequality in (10) occurs in (11), $\boldsymbol{c}''$ and $\boldsymbol{a}''$ also satisfy the system in (10). Therefore, $\boldsymbol{a}'' \in \Delta^*_p(X)$. Hence,

$$\Delta''_p(X) \subseteq \Delta^*_p(X) \tag{3.3.1}$$

Consider an $\boldsymbol{a}^* \in \Delta^*_p(X)$. Thus, there exists a $\boldsymbol{c}^*$ such that

$$\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}) \leq \hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}) \leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1} - \boldsymbol{1}).$$

Premultiplying throughout by $\hat{\pi}$ and after rearranging, we get

5. If $\hat{A}$ and $\hat{B}$ are two conforming square diagonal matrices, then the commutative law for matrix multiplication holds (i.e., $\hat{A}\hat{B} = \hat{B}\hat{A}$).

$$\hat{C}\hat{P}\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{c}^* + \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{C}\boldsymbol{p} \leq \hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}) \leq \hat{C}\hat{P}\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{c}^* + \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1} - \boldsymbol{1}).$$

Let

$$\boldsymbol{c''} = \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{c}^*. \tag{3.3.2}$$

Therefore, after premultiplying by $\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T$, the previous constraint becomes:

$$\begin{aligned}&\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{C}\hat{P}\boldsymbol{c''} + \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{C}\boldsymbol{p}\\ &\quad\leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}),\\ &\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T})\\ &\quad\leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{C}\hat{P}\boldsymbol{c''} + \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1} - \boldsymbol{1}).\end{aligned}$$

Now,

$$\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T = \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T. \tag{3.3.3}$$

Also,

$$\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T = \hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T. \tag{3.3.4}$$

The last pair of inequalities therefore become:

$$\begin{aligned}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c''} + \hat{C}\boldsymbol{p}) &\leq \hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T})\\ &\leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c''} + \hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1} - \boldsymbol{1}).\end{aligned} \tag{3.3.5}$$

From (10), we have $\boldsymbol{0} \leq \hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}$. Premultiplying throughout by $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T$ and using (3.3.2), we get

$$\boldsymbol{0} \leq \hat{C}\hat{P}\boldsymbol{c''} + \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{C}\boldsymbol{p}.$$

But $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{C}\boldsymbol{p} \leq \hat{C}\boldsymbol{p}$. Hence,

$$\boldsymbol{0} \leq \hat{C}\hat{P}\boldsymbol{c''} + \hat{C}\boldsymbol{p}. \tag{3.3.6}$$

Again, from (10),

$$\hat{C}\hat{P}\boldsymbol{c}^* \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}) - \hat{C}\boldsymbol{p}. \tag{3.3.7}$$

Premultiplying throughout by $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T$ and using (3.3.2), we get

$$\hat{C}\hat{P}\boldsymbol{c''} \leq \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}) - \hat{C}\boldsymbol{p}). \tag{3.3.8}$$

Since $\boldsymbol{0} \leq \hat{C}\hat{P}\boldsymbol{c}^* + \hat{C}\boldsymbol{p}$, we also obtain $-\hat{P}^{-1}\boldsymbol{p} \leq \boldsymbol{c}^*$. But $\boldsymbol{p} \leq \hat{P}\boldsymbol{1} - \boldsymbol{1}$. Therefore,

$$-\boldsymbol{1} + \hat{P}^{-1}\boldsymbol{1} \leq \boldsymbol{c}^*,$$

which means that

$$\boldsymbol{0} \leq \boldsymbol{c}^*. \tag{3.3.9}$$

Using (3.3.9) in (3.3.7), we get

$$\boldsymbol{0} \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}) - \hat{C}\boldsymbol{p}.$$

Because $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T \leq \hat{I}$, $\boldsymbol{0} \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}) - \hat{C}\boldsymbol{p}$ leads to

$$\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}) - \hat{C}\boldsymbol{p}) \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}) - \hat{C}\boldsymbol{p}.$$

Hence, from (3.3.8), $\hat{C}\hat{P}\boldsymbol{c''} \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}) - \hat{C}\boldsymbol{p}$. Combining this with (3.3.6), we therefore get

$$\boldsymbol{0} \leq \hat{C}\hat{P}\boldsymbol{c''} + \hat{C}\boldsymbol{p} \leq \hat{\pi}(\boldsymbol{u_T} - \boldsymbol{l_T}). \tag{3.3.10}$$

Using (3.3.2), we also get

$$\begin{aligned}(\hat{I} - \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\boldsymbol{c''} &= (\hat{I} - \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{c}^*\\ &= (\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T - \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\boldsymbol{c}^* = \boldsymbol{0}.\end{aligned} \tag{3.3.11}$$

From (10), we have $\hat{\lambda}\boldsymbol{c}^* = \boldsymbol{0}$. Premultiplying by $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T$ and by using (3.3.2), we get

$$\hat{\lambda}\boldsymbol{c''} = \boldsymbol{0} \tag{3.3.12}$$

From (10), we get the remaining constraints satisfied by $\boldsymbol{a}^*$:

$$\boldsymbol{a_l} \leq \boldsymbol{a}^* \leq \boldsymbol{a_u}, \tag{3.3.13}$$

$$\boldsymbol{0} \leq \hat{\mathcal{A}}\boldsymbol{a}^* + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T} \leq \hat{\mathcal{R}}(\boldsymbol{u_T} - \boldsymbol{l_T}). \tag{3.3.14}$$

Since (3.3.11), (3.3.12), (3.3.5), (3.3.10), (3.3.13), and (3.3.14) fulfill the system in (11), $\boldsymbol{a}^* \in \Delta''_p(X)$. Therefore,

$$\Delta^*_p(X) \subseteq \Delta''_p(X). \tag{3.3.15}$$

Relations (3.3.1) and (3.3.15), taken together, imply that $\Delta^*_p(X) = \Delta''_p(X)$.

We now need to prove the second part of the lemma: For every $\boldsymbol{a} \in \Delta''_p(X)$, there exists exactly one $\boldsymbol{c}$ for which (11) holds. Suppose that, for an $\boldsymbol{a} \in \Delta''_p(X)$, $\boldsymbol{c}$ satisfies the system in (11). Hence,

$$\begin{aligned}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p}) &\leq \hat{\pi}^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T})\\ &\leq \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c} + \hat{C}\boldsymbol{p} + \hat{C}\boldsymbol{1} - \boldsymbol{1}).\end{aligned}$$

Premultiplying throughout by $\hat{\pi}$ and because

$$(\hat{I} - \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\boldsymbol{c} = \boldsymbol{0},$$

by (11), we get

$$\hat{C}\hat{P}\boldsymbol{c} \leq \kappa \leq \hat{C}\hat{P}\boldsymbol{c} + \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{1} - \boldsymbol{1}), \tag{3.3.16}$$

where

$$\kappa = \hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a} + \boldsymbol{s_0} - \hat{\mathcal{R}}\boldsymbol{l_T}) - \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{C}\boldsymbol{p}.$$

If there exists another course vector $\boldsymbol{c}^*$ that, together with $\boldsymbol{a}$, also fulfills the system in (11), then $\boldsymbol{c}^*$ would also satisfy (3.3.16). That is,

$$\hat{C}\hat{P}\boldsymbol{c}^* \leq \kappa \leq \hat{C}\hat{P}\boldsymbol{c}^* + \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{1} - \boldsymbol{1}). \tag{3.3.17}$$

Using (3.3.16) and (3.3.17), we get

$$-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{1} - \boldsymbol{1}) \leq \hat{C}\hat{P}(\boldsymbol{c}^* - \boldsymbol{c}) \leq \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{1} - \boldsymbol{1}) \tag{3.3.18}$$

Now, $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T$ is a square diagonal matrix in which the principal diagonal elements are either 0 or 1. Hence, from (3.3.18), we can conclude that $\boldsymbol{c} = \boldsymbol{c}^*$. That is, for every $\boldsymbol{a} \in \Delta''_p(X)$, there exists exactly one $\boldsymbol{c}$ for which (11) holds. □

Proof for Lemma 3.4. Since $\Delta''_p(X) \neq \emptyset$, consider an element $\boldsymbol{a''} \in \Delta''_p(X)$. Then, there exists a $\boldsymbol{c}$ such that

$$\hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{p}) \le \pi^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a''}+\boldsymbol{s_0}-\hat{\mathcal{R}}\boldsymbol{l_T}) \le \hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{p}+\hat{C}\boldsymbol{1}-\boldsymbol{1}).$$

Premultiplying the above with $\hat{\pi}$ and, since

$$\hat{\pi}\hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T\boldsymbol{c}=\boldsymbol{c},$$
$$\hat{\pi}\hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T\boldsymbol{p}=\hat{\pi}\hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T\boldsymbol{q},$$

we get, after some rearrangement,

$$\hat{C}\hat{P}\boldsymbol{c}+\hat{\pi}\hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T\hat{C}\boldsymbol{q} \le \hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a''}+\boldsymbol{s_0}-\hat{\mathcal{R}}\boldsymbol{l_T}) \le \hat{C}\hat{P}\boldsymbol{c}+\hat{\pi}\hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T(\hat{C}\boldsymbol{q}+\hat{C}\boldsymbol{1}-\boldsymbol{1}).$$

Premultiplying by $\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T$ and by using (3.3.3) and (3.3.4), we can simplify the above to

$$\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{q}) \le \pi^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a''}+\boldsymbol{s_0}-\hat{\mathcal{R}}\boldsymbol{l_T}) \le \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{q}+\hat{C}\boldsymbol{1}-\boldsymbol{1}). \quad (3.4.1)$$

We also have $\boldsymbol{0} \le \hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{q} \le \hat{\pi}(\boldsymbol{u_T}-\boldsymbol{l_T})$. Premultiplying this by $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T$ and by using $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{\pi}=\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}$, this becomes

$$\boldsymbol{0} \le \hat{C}\hat{P}\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{c}+\hat{C}\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{p} \le \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}(\boldsymbol{u_T}-\boldsymbol{l_T}).$$

Substituting $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{c}$ by $\boldsymbol{c}$ and $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{p}$ by $\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{q}$, we get,

$$\boldsymbol{0} \le \hat{C}\hat{P}\boldsymbol{c}+\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{C}\boldsymbol{q} \le \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}(\boldsymbol{u_T}-\boldsymbol{l_T}). \quad (3.4.2)$$

Now, since $\Delta''_q(X) \neq \emptyset$, there exists at least one $\boldsymbol{b''}$ in $\Delta''_q(X)$. Let $\boldsymbol{c^*}$ be the corresponding course vector that, together with $\boldsymbol{b''}$, satisfies the system in (11). We then have

$$\boldsymbol{0} \le \hat{C}\hat{P}\boldsymbol{c^*}+\hat{C}\boldsymbol{q} \le \hat{\pi}(\boldsymbol{u_T}-\boldsymbol{l_T}).$$

Premultiplying the above throughout by $(\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)$ and by using $(\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\boldsymbol{c^*}=\boldsymbol{0}$, we get

$$\boldsymbol{0} \le (\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\hat{C}\boldsymbol{q} \le (\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\hat{\pi}(\boldsymbol{u_T}-\boldsymbol{l_T}). \quad (3.4.3)$$

From (3.4.2), we also get

$$(\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\hat{C}\boldsymbol{q} \le \hat{C}\hat{P}\boldsymbol{c}+\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\hat{C}\boldsymbol{q}+(\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\hat{C}\boldsymbol{q} \le \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}(\boldsymbol{u_T}-\boldsymbol{l_T})+(\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\hat{C}\boldsymbol{q}.$$

By using (3.4.3), the last constraint leads to the following:

$$\boldsymbol{0} \le \hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{q} \le \hat{\pi}(\boldsymbol{u_T}-\boldsymbol{l_T}). \quad (3.4.4)$$

We also have

$$(\hat{I}-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T)\boldsymbol{c}=\boldsymbol{0}, \quad (3.4.5)$$

$$\hat{\lambda}\boldsymbol{c}=\boldsymbol{0}, \quad (3.4.6)$$

$$\boldsymbol{a_l} \le \boldsymbol{a''} \le \boldsymbol{a_u}, \quad (3.4.7)$$

$$\boldsymbol{0} \le \hat{\mathcal{A}}\boldsymbol{a''}+\boldsymbol{s_0}-\hat{\mathcal{R}}\boldsymbol{l_T} \le \hat{\mathcal{R}}(\boldsymbol{u_T}-\boldsymbol{l_T}). \quad (3.4.8)$$

Since (3.4.5), (3.4.6), (3.4.1), (3.4.4), (3.4.7), and (3.4.8) fulfill the system in (11) for $\Delta''_q(X)$, we have $\boldsymbol{a''} \in \Delta''_q(X)$. Therefore,

$$\Delta''_p(X) \subseteq \Delta''_q(X). \quad (3.4.9)$$

Beginning with $\Delta''_q(X)$, we can similarly show that

$$\Delta''_q(X) \subseteq \Delta''_p(X). \quad (3.4.10)$$

Relations (3.4.9) and (3.4.10) together imply that

$$\Delta''_p(X) = \Delta''_q(X).$$

□

Proof for Lemma 3.5. Let $\boldsymbol{a''} \in \Delta''_p(X) \cap \Delta''_q(X)$. Since $\boldsymbol{a''} \in \Delta''_p(X)$, there exists a $\boldsymbol{c}$ such that

$$\hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{p}) \le \pi^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a''}+\boldsymbol{s_0}-\hat{\mathcal{R}}\boldsymbol{l_T}) \le \hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c}+\hat{C}\boldsymbol{p}+\hat{C}\boldsymbol{1}-\boldsymbol{1}).$$

Also, since $\boldsymbol{a''} \in \Delta''_q(X)$, there exists a $\boldsymbol{c^*}$ such that

$$\hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c^*}+\hat{C}\boldsymbol{q}) \le \pi^T\hat{\pi}\hat{\mathcal{R}}^T(\hat{\mathcal{A}}\boldsymbol{a''}+\boldsymbol{s_0}-\hat{\mathcal{R}}\boldsymbol{l_T}) \le \hat{\mathcal{R}}^T\mathcal{R}\hat{\pi}^T(\hat{C}\hat{P}\boldsymbol{c^*}+\hat{C}\boldsymbol{q}+\hat{C}\boldsymbol{1}-\boldsymbol{1}).$$

Thus,

$$-\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{1}-\boldsymbol{1}) \le \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\Big(\hat{C}\hat{P}(\boldsymbol{c^*}-\boldsymbol{c})+\hat{C}(\boldsymbol{q}-\boldsymbol{p})\Big) \le \hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{1}-\boldsymbol{1}).$$

Premultiplying the above with $\hat{\pi}$ and since

$$\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{c}=\boldsymbol{c},$$
$$\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T\boldsymbol{c^*}=\boldsymbol{c^*},$$

we get

$$-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{1}-\boldsymbol{1}) \le \hat{C}\big(\hat{P}(\boldsymbol{c^*}-\boldsymbol{c})+\varphi\big) \le \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{C}\boldsymbol{1}-\boldsymbol{1}), \quad (3.5.1)$$

where $\varphi=\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\boldsymbol{q}-\boldsymbol{p})$. From (3.5.1), we thus get

$$\hat{P}(\boldsymbol{c^*}-\boldsymbol{c})+\varphi=\boldsymbol{0},$$

or

$$\varphi=\hat{P}(\boldsymbol{c}-\boldsymbol{c^*}). \quad (3.5.2)$$

Since $\hat{0} \le \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T \le \hat{I}$ and $\boldsymbol{0} \le \boldsymbol{p}, \boldsymbol{q} < \hat{P}\boldsymbol{1}$, we also have the following constraint for φ:

$$-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{P}\boldsymbol{1}-\boldsymbol{1}) \le \varphi \le \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{P}\boldsymbol{1}-\boldsymbol{1}).$$

Using the last constraint and (3.5.2), we therefore get

$$-\hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{P}\boldsymbol{1}-\boldsymbol{1}) \le \hat{P}(\boldsymbol{c^*}-\boldsymbol{c}) \le \hat{\pi}\hat{\mathcal{R}}^T\hat{\mathcal{R}}\hat{\pi}^T(\hat{P}\boldsymbol{1}-\boldsymbol{1}).$$

From the above, $\boldsymbol{c^*}-\boldsymbol{c}=\boldsymbol{0}$; consequently, from (3.5.2), $\varphi=\boldsymbol{0}$. □

ACKNOWLEDGMENTS

This research was partially supported by the US National Science Foundation under Grant NSF CCR-9526325 and in part by DARPA under Contract F30602-98-2-0144.

REFERENCES

[1] C. Ancourt, F. Coelho, F. Irigoin, and R. Keryell, "A Linear Algebra Framework for Static HPF Code Distribution," Technical Report A-278-CRI, Centre de Recherche en Informatique, École Nationale Supérieure des Mines de Paris, 35, rue Saint-Honoroé, F-77305 Fontainebleau cedex, France, Nov. 1995.

[2] S.P. Amarasinghe and M.S. Lam, "Communication Optimization and Code Generation for Distributed Memory Machines," *Proc. ACM SIGPLAN Programming Language Design and Implementation,* pp. 126-138, June 1993.

[3] V. Adve and J. Mellor-Crummey, "Using Integer Sets for Data-Parallel Program Analysis and Optimization," *Proc. ACM SIGPLAN Programming Language Design and Implementation,* pp. 186-198, June 1998.

[4] V. Balasundaram, "A Mechanism for Keeping Useful Internal Information in Parallel Programming Tools—The Data Access Descriptor," *J. Parallel and Distributed Computing,* vol. 9, no. 2, pp. 154-170, June 1990.

[5] U. Banerjee, *Loop Transformations for Restructuring Compilers: The Foundations.* Norwell, Mass.: Kluwer Academic, Jan. 1993.

[6] S. Chatterjee, J.R. Gilbert, F.J.E. Long, R.S. Schreiber, and S.-H. Teng, "Generating Local Addresses and Communication Sets for Data-Parallel Programs," *J. Parallel and Distributed Computing,* vol. 26, no. 1, pp. 72-84, Apr. 1995.

[7] D. Callahan and K. Kennedy, "Analysis of Interprocedural Side Effects in a Parallel Programming Environment," *Proc. First ACM Int'l Conf. Supercomputing,* pp. 138-171, May 1987.

[8] S.K.S. Gupta, S.D. Kaushik, S. Mufti, S. Sharma, C.-H. Huang, and P. Sadayappan, "On Compiling Array Expressions for Efficient Execution on Distributed-Memory Machines," *Proc. 22nd Int'l Conf. Parallel Processing,* vol. II (Software), pp. 301-305, Aug. 1993.

[9] P.G. Joisha and P. Banerjee, "PARADIGM (Version 2. 0): A New HPF Compilation System," *Proc. 13th Int'l Parallel Processing Symp. and 10th Symp. Parallel and Distributed Processing,* pp. 609-615, Apr. 1999.

[10] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steele Jr., and M.E. Zosel, *The High Performance Fortran Handbook, Scientific and Engineering Computation Series.* Cambridge, Mass.: MIT Press, 1994.

[11] F.M. McMahon, "The Livermore FORTRAN Kernels: A Computer Test of the Numerical Performance Range," Technical Report UCRL-55745, Lawrence Livermore National Laboratory, Livermore, Calif., Dec. 1986.

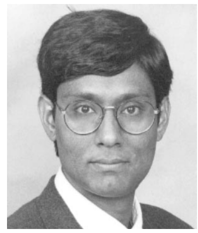
[12] S.P. Midkiff, "Local Iteration Set Computation for Block-Cyclic Distributions," *Proc. 24th Int'l Conf. Parallel Processing,* vol. II (Software), pp. 77-84, Aug. 1995.

[13] E.Y.-H. Su, D.J. Palermo, and P. Banerjee, "Processor Tagged Descriptors: A Data Structure for Compiling for Distributed-Memory Multicomputers," *Proc. 1994 Int'l Conf. Parallel Architectures and Compilation Techniques,* pp. 123-132, Aug. 1994.

[14] A. Thirumalai and J. Ramanujam, "Efficient Computation of Address Sequences in Data-Parallel Programs Using Closed Forms for Basis Vectors," *J. Parallel and Distributed Computing,* vol, 38, no. 2, pp. 188-203, Nov. 1996.

[15] M.J. Wolfe, *High Performance Compilers for Parallel Computing,* Redwood City, Calif.: Addison Wesley, 1996.

Pramod G. Joisha received the BTech degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, in 1996 and the MS degree in electrical and computer engineering from Northwestern University, Evanston, Illinois in 1998. He is currently pursuing the PhD degree in the Department of Electrical and Computer Engineering at the same university. His research interests include parallel computing, compilers, and programming languages. He is a student member of the IEEE.



Prithviraj Banerjee received the BTech degree in electronics and electrical engineering from the Indian Institute of Technology, Kharagpur,India, in 1981 and the MS and PhD degrees in electrical engineering from the University of Illinois at Urbana-Champaign in 1982 and 1984, respectively. Dr. Banerjee is currently the Walter P. Murphy Professor, chairman of the Department of Electrical and Computer Engineering, and director of the Center for Parallel and Distributed Computing at Northwestern University, Evanston, Illinois, prior to which he was the director of the Computational Science and Engineering program and a professor in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. He has received numerous awards and honors during his career, including the 1981 President of India Gold Medal from the Indian Institute of Technology, Kharagpur, the 1986 IBM Young Faculty Development Award, the 1987 National Science Foundation's Presidential Young Investigator's Award, the 1992 Senior Xerox Research Award, the 1993 University Scholar Award from the University of Illinois, the 1996 Frederick Emmons Terman Award of ASEE's Electrical Engineering Division sponsored by Hewlett-Packard, and the 2001 IEEE Taylor L. Booth Education Award from the IEEE Computer Society. He served as the program chair of the 1990 High-Perfomance Computing Conference and of the 1995 International Conference on Parallel Processing, as general chairman of the 1997 International Conference on Parallel and Distributed Computing Systems and the 1989 International Workshop on Hardware Fault Tolerance in Multiprocessors, on the program and organizing committees of the 1988, 1989, 1993, and 1996 Fault-Tolerant Computing Symposium on Computer Architecture, the 1998 International Conference on Architectural Support for Programming Languages and Operating Systems, the 1990, 1993, 1994, 1995, 1996, 1997, and 1998 International Symposium on VLSI Design, the 1994, 1995, 1996, 1997, 1998, and 2000 International Conference on Parallel Processing, and the 1995, 1996, and 1997 International Conference on High-Performance Computing. Dr. Banerjee's research interests are in parallel algorithms for VLSI design automation, distributed-memory parallel compilers, and compilers for adaptive computing. He is the author of over 300 papers. He leads the PARADIGM Compiler Project for compiling programs for distributed memory multicomputers, the ProperCAD project for portable parallel VLSI CAD applications, the MATCH project on a MATLAB compilation environment for adaptive computing, and the PACT project on power aware compilation and architectural techniques. He is the author of *Parallel Algorithms for VLSI CAD* (Prentice Hall 1994) and has supervised 27 PhD and 30 MS student theses so far. He has served as an associate editor for the *Journal of Parallel and Distributed Computing*, the *IEEE Transactions on VLSI Systems*, and the *Journal of Circuits, Systems and Computers* and is currently an associate editor for the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He has been a consultant to many companies and was on the Technical Advisory Board of Ambit Design Systems. Dr. Banerjee is a fellow of the ACM, a member of the IEEE Computer Society, and a fellow of the IEEE.