LACS: a language for affine communication structures

Sanjay Rajopadhye

LACS: A Language for Affine Communication Structures

Sanjay Rajopadhye

N° 2093

Avril 1993

PROGRAMME 1

Architectures parallèles,
bases de données,
réseaux et systèmes distribués

LACS: A Language for Affine Communication Structures[*]

Sanjay Rajopadhye[**]

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués

Projet API

Rapport de recherche n ° 2093 — Avril 1993 — 31 pages

Abstract: We propose a language which provides a unified notation for specifying parallel assignment, communication and reduction operations in massively parallel programs. It is designed around a *Parallel Assignment Statement* (PAS), and *Atomic Communication Events* (ACE's), and can be used to describe most common communication libraries like 1-to-1 transfer, broadcast, personalized communication, reduce, scans, etc., as APL-style one-liners. LACS uses convex polyhedra and affine transformations, and hence, tools from linear algebra can be used for compile time analysis. We can automatically detect when a PAS in LACS is well formed, has write conflicts, etc. We can also detect the presence of communication activities like scatters, reduces, scans, and their generalizations.

Key-words: affine dependencies, broadcast, communication library, convex polyhedra, data partitioning, parallel prefix, reduction, scan, scatter-gather, spread, total exchange.

(Résumé : tsvp)

[*]Supported by NSF Grant No. MIP-910852, by the French Coordinated Research Program C^3, and by the Esprit BRA project NANA 2 (No. 6632).

[**]`rajopadh@irisa.fr`

LACS: Un Langage pour des Structures de Communications Affines

Résumé : On propose un langage qui donne une notation uniforme pour spécifier des affectations parallèles et des opérations de communication et de réduction dans les programmes massivement parallèles. Il est construit autour d'une instruction d'affectation parallèle (PAS), et d'événements atomiques de communication (ACE). En LACS, on peut exprimer la majorité les opérations de communication les plus fréquentes en une seule ligne (comme en APL) : '1-to-1 transfer', diffusion, 'personalized communication', réduction, 'scans', etc. LACS utilise des polyèdres convèxes et des transformations affines, et il est donc possible d'utiliser les outils de l'algèbre linéaire pour l'analyse statique. On peut détecter automatiquement quand une PAS est bien formée, a des conflits d'écritures, etc. On peut aussi détecter automatiquement la présence d'activités de communication comme des 'scatters', réduction, 'scans' et leur géneralizationes.

Mots-clé : dépendences affines, 'broadcast', bibliotheques de communication, polyèdres convèxes, partitionement de données, fonctions d'indéxage linéres, préfix parallèle, reduction, 'scan', 'scatter-gather', 'spread', 'total exchange'.

A Sneak Preview

Consider the problem of specifying communication activity in massively parallel programs (data parallel, nested loop and/or single assignment). Our principal data structure is the multidimensional array, so we have **data** variables such as `X`, `Y` etc. To access them, we have **index** variables such as `i`, `j`, etc. A data variable may be accessed *only* by constructing an appropriate **index expression**—an affine combination (i.e., linear combinations plus a constant) of index variables. An *Atomic Communication Event* (ACE) is specified with a source, a destination, and a transfer operator, `=>`. We can have multiple destinations (broadcast), and/or multiple sources (reduction[*]). All this is a single ACE. We specify collections of *simultaneous* ACE's as a *Parallel Assignment Statement* (PAS). The only way to specify "collections" (both for multiple sources and destinations, and for the ACE's that make up a PAS) is through convex polyhedral regions of index space, whose syntax is: `{\tt List-of-vars '|' List-of constraints}.`

With this brief introduction, look at a few examples (as shown in Fig 1). Let us use the convention that in an array access such as `X[p,q,r]`, the `p` refers to rows, `q` refers to columns and there may be additional indices such as `r`. For some examples, the first two indices are also implicitly the processor id's (although LACS does not impose such an interpretation).

1. A broadcast of `X[N,N]` to `X` at all points (size of `X` is `N`) can be written as an ACE:
   `X[N,N] => {i,j | 0<i,j<=N: Y[i,j]}`

2. A reduction (with addition as the operator) of `X` on the first row, whose result overwrites the value that used to be in `X[1,1]` may be described by the ACE:
   `{i | 0<i<=N: X[1,i]} +=> X[1,1]`

3. The same reduction, but the result is now broadcast back along the row:
   `{i | 0<i<=N: X[1,i]} +=> {i | 0<i<=N: X[1,i]}`

4. A scatter of `N` different `X` values (stored along a third dimension) from point `[1,N]` to all points along the same column (this is a PAS, like all subsequent examples):
   `{i | 0<i<=N: X[1,N,i] => Y[i,N]}`

5. Simultaneous scatter from all points on the main diagonal to their rows:
   `{i,j | 0<i,j<=N: X[i,i,j] => Y[i,j]}`

[*]In this case we need to specify a binary operator.

6. A gather of `X` on the main diagonal whose result goes to `Y` at index `[N,N]`
```
{i | 0<i<=N:      X[i,i] => Y[N,N,i]    }
```

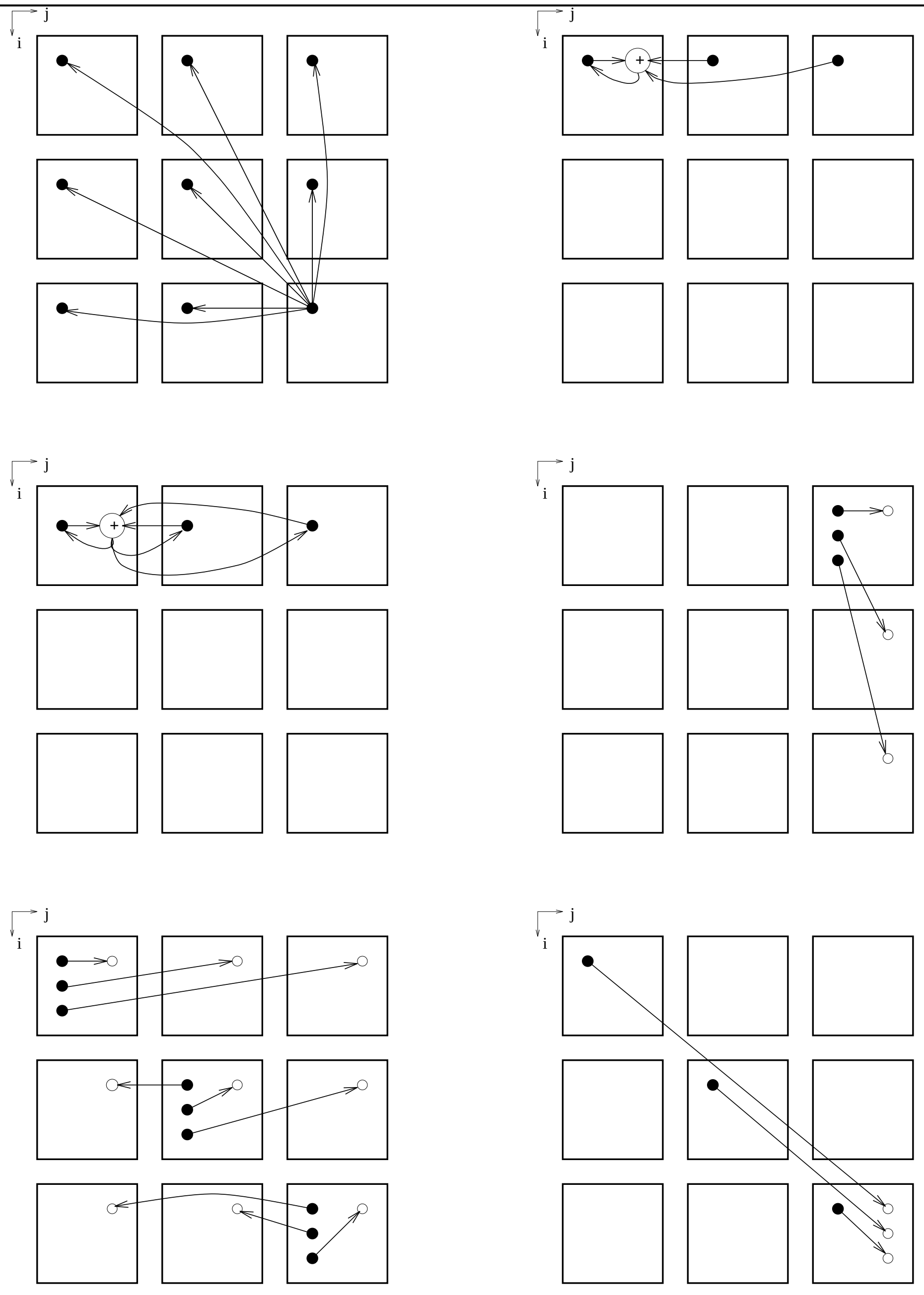

Figure 1: Motivating examples: N is 3, and X and Y are solid and open circles, respectively

Thus, we can claim that LACS has the following features:

- Simplicity: Most of the syntactic features have been summarized in the first paragraph. The important part of the BNF (see Appendix I) is less than a page.
- Expressive Power: A large number of common communication patterns can be written as one-liners. You have already seen a few examples. You will see many more.
- Compile Time Analysis: Many important properties of the communication can be automatically deduced from a LACS program. For this you will have to read the paper.

1 Introduction

The emergence of practical massively parallel computers such as the Connection Machines, MasPar and others (building on the earlier work on the ILIAC, DAP, MPP) has spurred the development of programming languages that can effectively harness their power. Data parallel languages have become increasingly popular in this role, since they specify the computation in terms of operations on (ideally) independent pieces of data. At this level of abstraction, one can consider languages such as FP, APL2, CMLisp, Crystal [7], Lucid [1], etc., as very high level data parallel languages [16]. Regardless of whether the language model is functional or imperative (nested loops or data parallel) the most important data structure in massive parallelism is the array, and parallelism is achieved when different processes operate on disjoint index subsets of the array. The early data parallel languages such as DAP-FORTRAN, MPP Pascal, *Lisp and MPL were closely tied to the machine architecture, where physical processors were identified explicitly with specific array indices. Recently there has been work on *architecture independent* data parallelism [4, 19], where the programmer specifies the computation in terms of virtual processes, with virtual indices, and the mapping to physical processor indices is isolated to a few directives (or chosen by the compiler).

One of the major factors affecting the performance of parallel programs is the communication cost. Massively parallel computers, such as the Connection Machines, MasPar and others perform SIMD style, local communication much faster than arbitrary point to point communication. Distributed memory machines, perform certain communications faster than others, depending on their topology (and the communication libraries available). In order to achieve the best performance, it is necessary to fully exploit such specific ability, and this has received increasing attention [6, 9, 12, 13, 15] in the context of nested loop and/or data parallel programs. In order to address this problem in a systematic manner, we first need a formal model to describe the communication at an abstract, architecture independent level. In this paper, we present a Language for Affine Communication Structures (LACS) for this purpose [17]. It is based on convex polyhedra and affine transformations, and enables us to concisely specify a large class of practical communications (in APL-style one liners), yielding a generalization of the standard communication libraries such as broadcast, multicast, spread, reduce, scatter-gather, total exchange, etc.

We envisage three main benefits from LACS. In the short term, it provides a separation of concerns: application programmers specify the communication in their program in a high level, declarative, machine independent manner. Parallel machine manufacturers are responsible for providing a correct and efficient implementation of LACS. In the medium term, we expect to infer the LACS specification that achieves the communication of a given program (data parallel or nested loop) provided that data partitioning (allocation of data values to processors) annotations are also given. Another medium term goal is to determine estimates of the costs of implementing a LACS specification, given some of the machine parameters. We thus expect LACS to serve as an intermediate representation for the compilation of communication. In the long term, we anticipate that this will lead to a general theory for automatic data partitioning, which can address non rectangular arrays and also data alignments with skews, replication etc.

As a first step in this direction (both for efficient compilation and for cost estimation), it is necessary to analyze a LACS program and deduce its properties. This is the second subject of this paper, and we focus on the Parallel Assignment Statement (PAS), the principal construct in LACS. The remainder of the paper is organized as follows. In the following section, we describe the language (the syntax is given in the Appendix I). This is followed by a number of exercise problems (Sec 3) for you to get a feel for the expressive power of the language. Then, in Sec 4 we describe the internal representation that is used for the analysis. Sec 5 presents the first part of the static analysis of a PAS—the conditions under which it is well formed. Each test is illustrated with a number of examples. Then, in

Sec 6 we show how using these results, scans, scatters, gathers, etc., can be automatically deduced from a given PAS. Finally, we discuss related work and give our conclusions.

2 Specification of LACS

In parallel processing, communication activity is typically described as a directed graph [5, 20], where nodes represent source processes, and arcs represent communication from a (single) source node to a (single) destination node. LACS uses two generalizations of this. First, we use multi-graphs* in order to permit multiple destinations (broadcasts) and multiple sources (reductions). Second, we explicitly model temporal information—specifying which multi-edges are simultaneously active, and also the order in which these groups become active (this idea was originally used in our earlier language, LaRCS [14]).

Thus, we can model the communication in a massively parallel program by a multi-graph where *each* element of *each* array corresponds to a node. Obviously, such a fine grain description is unacceptable if the compiler has to actually construct the multi-graph. LACS can thus be viewed as a language for specifying such multi-graphs in a compact manner. A LACS program consists of a *sequence* (or a nested loop containing a sequence) of Parallel Assignment Statements (PAS's). Each PAS is a *collection* of Atomic Communication Events (ACE's). Thus, an ACE models a single multi-edge, and a PAS models a group of simultaneous multi-edges, and a complete LACS program is a sequence of PAS's.

2.1 Syntax

Syntactically, LACS has two types of variables, **data** variables and **index** variables. Data variables are simply the names of multidimensional arrays. Index variables are integer variables which are used in order to access elements of the data variables. Certain index variables are declared to be **size parameters**. In general, index variables are used to

*A directed multi-graph is defined as a pair, $\langle V, M\rangle$. V is the set of nodes, and M is a set of multi-edges. Each multi-edge is an ordered pair of **subsets of nodes**—the **source nodeset** and the **destination nodeset**.

build **index vectors**, and any data variable is accessed only through such index vectors. There are two fundamental constraints in LACS:

- The only way to construct an index vector is through *affine combinations of index variables.*
- The only way to specify collections of index variables is through *convex polyhedral regions.*

It is these constraints that both give LACS its simplicity and expressive power, and also enable the use of linear algebra for our analysis. The main constructs in LACS are as follows (see the Appendix I for the BNF).

2.1.1 Convex polyhedral regions

Convex polyhedra (actually, we are only concerned with polytopes, i.e., bounded polyhedra) are central to LACS because of many important properties: they are closed under intersection, difference and other geometric operations; standard libraries exist for computing these. Most importantly for the static analysis of LACS, they are also closed under unimodular affine maps (and also the pre-images of arbitrary integer affine maps). From a pragmatic perspective, index spaces that occur in practice are often convex polyhedra, since loop bounds are typically affine functions of indices. Furthermore, the data access patterns within the loops are usually affine index functions. The following examples give an idea of the LACS syntax for polyhedra

`i,j,k	0<i<=j<=N; k=0`	read as "the indices, i, j, k such that they satisfy $0 < i \leq j \leq N$ and $k = 0$" (`N` is a parameter).
`i,j	0<i,j<=N`	the comma is a shorthand for four inequalities.
`k	j<=k<=N`	`j` must be a parameter or defined in an outer scope.

2.1.2 Atomic Communication Events

As mentioned above, data variables can be accessed only by an index vector—an affine function of index variables. A data variable plus its index vector, eg, `X[1,i,N-i,3k+i]` and `Y[i,i,j]` constitute a `Data-Variable-Expr`. Taken completely out of its context, it may have some free index variables, and by instantiating (some of) them, we can define a collection of `Data-Variable-Expr`'s, called a `Src-Expr` (or `Dst-Expr`). For example, `{i| 0<i<=10: X[i,2i]}` denotes the set of 10 values of `X` at index points along a particular line. Lexically, the braces declare a local scope of index variables, the `Polyhedron` specifies the values that they may take, and the `Data-Variable-Expr` uses them.

The fundamental construct in LACS is an ACE, and it specifies that data at index points given by a `Src-Expr` is moved to the index points specified by a `Dst-Expr` (so it is nothing but a glorified MOVE instruction). It is called atomic because there is a *single result value.* Recall that an ACE can model a single multi-edge. Examples:

`A[1,2,4] => B[0,0,0]`	a point-to-point transfer (simple)
`A[1,2,4] => {j	0<j<10: B[0,j,0]}`	multiple destinations (broadcast)
`{i	0<i<10: A[i,i,10]} *=> B[0,10]`	multiple sources (reduction)

2.1.3 Parallel Assignment Statements

Since an ACE corresponds to a single value, we need a construct to describe multiple transfers. This is done in LACS by the PAS, whose syntax consists of a polyhedron to declare and define the index variables, and an ACE within this lexical scope. The idea is that for each each point in the polyhedron, we have a specific ACE, and the PAS denotes this simultaneous collection of ACE's. This notion of simultaneity is at the level of the specification (an actual implementation may be serialized). Consider the following PAS (Ex 4 from the preview): `{i | 0<i<=N: X[1,N,i] => Y[i,N]}` It is to be read as "for all i between 1 and N, the collection of ACE's, `X[1,N,i] => Y[i,N]`".

2.1.4 Complete LACS Programs

Finally, a complete LACS program is merely a sequence of PAS's. There are two ways of doing this: a finite sequence, or a nested loop whose body is a finite sequence of PAS's. We do not allow arbitrary loops, but only those that represent a lexicographic traversal of a polyhedron, and so the syntax is as shown in the following examples.

```
{ k | 0<k<=N:    lex      {i | 0<i<=k: X[i,i] => Y[k,k,i]} }
{ k | 0<k<=N:    lex      {i,j | 0<i,j<=N : X[k,i,j] => Y[k,j,i]} }
```

You can check that the first one is a sequence of `N` gathers on the main diagonal, each time to index point `[k,k]`, and the second is a series of total exchanges on the columns one after the other. One important point to note is that the outermost `Polyhedron` is not merely a *set* of index points, but also imposes a specific *order* of traversal. The keyword `lex` is used to emphasize this point (and also to disambiguate the grammar).

LACS has exactly three levels in its block structure as delimited by the braces. Some index variables are declared at the outermost level (sequence of PAS's); they are similar to index variables in a nested loop program. They are called **temporal indices** and belong to the **temporal polyhedron**. Variables declared at the level of a PAS are called **parallel indices**, and belong to the **parallel polyhedron**. Within an ACE, index variables can be declared either as **source indices** or **destination indices**, and constitute the source or destination polyhedra, respectively. You can see that the scope of a source polyhedron *ends* before the `Transfer-Op`, that of the destination polyhedron can only begin *after* a `Transfer-Op`, and that of a parallel polyhedron extends on both sides. Scope rules are lexical, so that all outer indices are visible in an inner scope. Also note that an index variable may be used in two different ways within its scope—to construct an `Index-Expr`, and to define the linear inequalities of an inner polyhedron.

2.2 Examples

At this point I recommend that you return to the first six examples and study them carefully. Observe why the first three are single ACE's while the others are PAS's, and note the manner in which the index variables are used to construct the index expressions.

For the following examples, we use an operation called **scan**, or parallel prefix, which has been identified as an important class of communications (see the book by Bleloch [3] for an excellent discussion). A scan of a vector, X, is defined as the vector Y, and is given by $Y_i = \sum_{j=1}^{i} X_j$ (the summation represents an arbitrary associative and commutative operator). Like reductions, scans combine some aspects of communication and computation with a binary operator; the difference is that in a scan we also want all the partial results. They can be implemented efficiently on parallel machines (in $\Theta(\log n)$ time), and most vendors now provide library routines for them. Let us now look at a few LACS programs with scans.

Example 1 Let us write a scan on the main diagonal, using the same conventions as before. This must be a PAS (since we need multiple result values), whose ACE's are enumerated by, say, `i` = `1`...`N`. The `i`-th ACE produces the result for `X[i,i]`, which is a reduction of the preceding values on the diagonal, i.e., all values `X[j,j]`, where `0` < `j` ≤ `i`. This gives us the following PAS:

```
    {i | 0<i<=N: {j | 0<j<=i: X[j,j]} +=> X[i,i]}
```

Example 2 Now consider the problem of specifying a scan on the fifth off-diagonal, i.e., the points `[6,1]`, `[7,2]` ... `[N,N-5]`. The destinations are all points of the form, `[i+5,i]`, where `i` ranges from 1 to `N-5`. Hence our PAS will have the form,

```
    {i | 0<i<=N-5: Src-Expr +=> X[i+5,i]}
```

Observing that all the points preceding `[i+5,i]` (and on the same off-diagonal) are of the form `[j+5,j]`, where `0` < `j` ≤ `i`, the solution is

```
    {i | 0<i<=N-5: {j | 0<j<=i: X[j+5,j]} +=> X[i+5,i]}
```

Example 3 Finally, consider the problem of specifying independent scans on all off-diagonals in the lower triangular region. This can be done simply by replacing the 5 in the above PAS by an new index variable, `k` in the parallel polyhedron, which ranges from `1` to `N-1`. Hence, we have the following solution.

```
    {i,k | 0<k<N;0<i<=N-k: {j | 0<j<=i: X[j+k,j]} +=> X[i+k,i]}
```

3 Exercise Problems

Using the same kind of reasoning as above, express the communication actions shown below in LACS (hint: a single PAS is required for each of them). As with the above examples, try to first identify the number of results (i.e., the number of ACE's), then choose indices for them, and then work out the details of the individual ACE's. You will soon get the hang of it and begin to have fun. Pictorial descriptions are given in Fig 2.

1. A total exchange on the main diagonal.

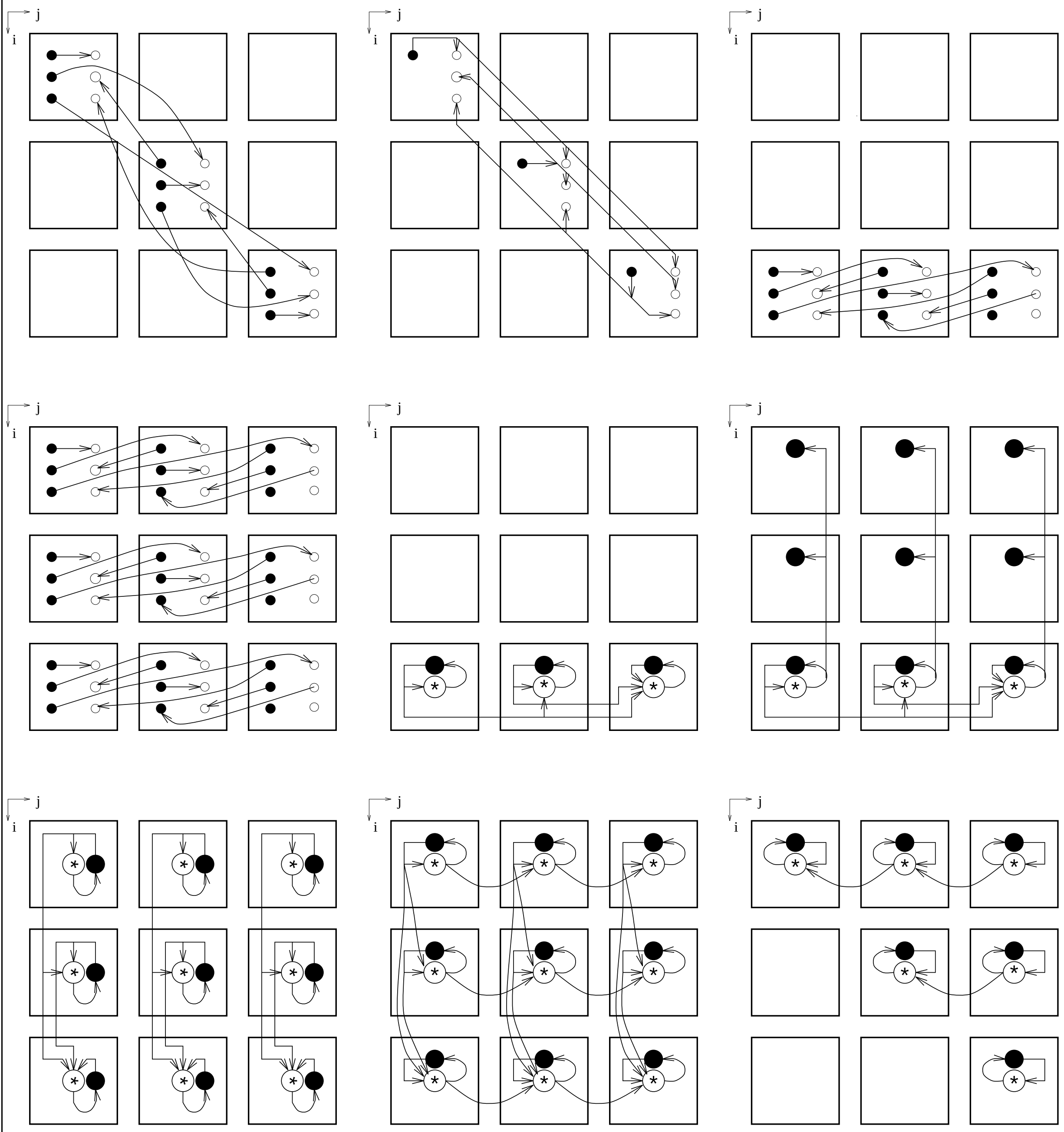

Figure 2: Exercise Problems: A Pictorial View

2. An all-to-all broadcast on the main diagonal.

3. A total exchange on row `N`.

4. Multiple independent total exchanges on all rows.
5. A product scan on row `N`.
6. The same scan, but the results are to be broadcast along the respective columns.
7. Independent scans on all columns.
8. Two dimensional scan (each point gets the product of all points that are above and left of it). In the figure, we show this serialized in order to avoid clutter.
9. Reverse scan on each row in the upper triangular region (each point in the upper triangular region gets the product of all values on the same row and to its right). This too is shown serialized.

The solutions are shown in Fig 3. I have taken the liberty of mixing up the answers and also removing the `*` from all the `Transfer-Op`'s. This has been done for two reasons. I feel that somewhat different skills are needed for writing your own programs, than for reading someone else's' solutions[*]. There will be ample opportunity to develop these latter skills when we come to static analysis, but for now I'd like you to get the synthesis skills. Second, though there are many equivalent solutions, there are certain transformations that you can apply to a PAS. I conjecture that there is a unique normal form for PAS's. If you get substantially different solutions, I would love to hear about them. Can you think of any other interesting communications that *you* want to express? Write to me, and we'll try it. Then I'll give you an excuse for why we can't do it in LACS. :-)

4 Abstract Representation of LACS Programs

Before addressing the problem of analysis of LACS programs, we need to develop an internal representation. First consider, the two ACE's given below, which specify the same communication, namely that `A[0]` is broadcast to all the diagonal values of `X`.

```
A[0]=>{i,j | 0<i,j<=N,i=j: X[i,j]}    and    A[0]=>{i | 0<i<=N: X[i,i]}
```

[*]For this you have to see things exactly *their* way—i.e., put yourself into their frame of reference, and use their conventions, etc.

```
{i,j | 0<i,j<=N :
    X[i,i,j] => Y[j,j,i]
}
```
(a)

```
{i,j,k | 0<i,k,j<=N:
    X[k,i,j] => Y[k,j,i]
}
```
(b)

```
{i | 0<i<=N:
  {j | 0<j<=i: X[N,j]}
    => X[N,i]}
```
(c)

```
{i,j | 0<i<=j<=N:
    {k | j<=k<=N: X[i,k]}
        => X[i,j]
}
```
(d)

```
{i | 0<i<=N:
    {j | 0<j<=i: X[N,j]}
       => {k | 0<k<=N: X[k,i]}
}
```
(e)

```
{
i,j | 0<i,j<=N:
    X[N,i,j] => Y[N,j,i]
}
```
(f)

```
{i | 0<i<=N:
    X[i,i] =>
       {j | 0<j<=N: Y[j,j,i]}
}
```
(g)

```
{i,j | 0<i,j<=N:
    {k,l|0<k<=i,0<l<=j:X[k,l]}
        => X[i,j]
}
```
(h)

```
{i,k | 0<i,k<=N:
    {j | 0<j<=i: X[j,k]}
       => X[i,k]
}
```
(i)

Figure 3: Solutions to the exercises (jumbled)

At first glance, it seems that the first ACE has a two dimensional polyhedron (it has indices, `i` and `j`), while the second one has a one-dimensional polyhedron. However, the constraints in the first ACE indicate that there is a redundancy—although embedded in a two dimensional index space, the polyhedron is really a line. Such redundancies are not allowed. Also, domains such as `i,j | 0<i<10`, as well as `i,j | i>j; i<j+10`, are also not allowed. The first one declares a variable, `j` but does not constrain it in any way, so the domain is unbounded along the $\pm j$ axis. The second one is similar, except that direction in which the domain is unbounded is $\pm[1,1]^T$. Both these polyhedra have what is called a lineality space.

Convex polyhedra have two dual representations. With a *constraint representation*, they can be viewed as the set of points that satisfy a finite number of linear inequalities, $\{x : Ax \geq b\}$. With the *parametric representation*, a polyhedron is the set of (a particular form of) linear combinations of its extremal points, $\{z = R\mu : \mu \geq 0\}$. There are standard libraries for converting between the two representations. In the parametric representation, redundant indices are detected easily by checking if the matrix R has full column rank. The lineality space is detected easily in the constraint representation by testing if the matrix A has full column rank. By appropriate preprocessing, any redundant indices and/or lineality spaces can be removed automatically, and for further analysis we assume that R and A are of full column rank.

We also assume (without loss of generality) that index variable names are distinct, and let z_t, z_p, z_s and z_d denote the vector of temporal, parallel, source and destination index variables, respectively. When necessary, let $z_{tp} = [z_t, z_p]^{\mathrm{T}}$ and $z_1 = [z_t, z_p, z_s]^{\mathrm{T}}$ and similarly, $z_2 = [z_t, z_p, z_d]^{\mathrm{T}}$. The temporal polyhedron, D_t is specified as $Tz_t \geq t$, the parallel polyhedron D_p as $Pz_{tp} \geq p$ (note that temporal as well as parallel indices may be used here), and the source and destination polyhedra (D_s and D_d) as $Sz_1 \geq s$ and $Dz_2 \geq d$ respectively (note how z_1 and z_2 are used here allowing you to specify constraints on all the outer indices).

Example 4 To illustrate the notation, consider, the following LACS program.

```
{i,j | 0 < i <= j <= N : lex
  {k | i <= k <= j :   {l,m | 0 <= l,m <= k : X[i+l,m-k]}
                   +=> {r | 0 <= r <= k : Y[i+j,r,r]    } } }
   }
 }
```

Here, $z_t = [i, j]^{\mathrm{T}}$, $z_p = [k]^{\mathrm{T}}$, $z_s = [l, m]^{\mathrm{T}}$, $z_d = [r]^{\mathrm{T}}$, $z_{tp} = [i, j, k]^{\mathrm{T}}$, $z_1 = [i, j, k, l, m]^{\mathrm{T}}$ and $z_2 = [i, j, k, r]^{\mathrm{T}}$. The temporal, parallel, source and destination polyhedra are, respectively, as shown below.

$$\begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \geq \begin{bmatrix} 1 \\ 0 \\ -N \end{bmatrix}, \qquad \begin{bmatrix} -1 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \\ l \\ m \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \\ r \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Notice how we have treated the parameter `N` as a constant—in general, it is simply another index variable (at yet an outer level). The parameters belong to an unbounded polyhedral region[*]. Also note that in a LACS program, the linear inequalities defining a polyhedron impose constraints *in addition to* those imposed by the outer scope. Hence, at the innermost level, the set of all the constraints on the visible variables is given by what are called the **complete source/destination polyhedra**. Observing that you can easily partition the columns of the above matrices into submatrices of appropriate sizes, $P = [P_1|P_2]$, $S = [S_1|S_2|S_3]$ and $D = [D_1|D_2|D_3]$, the complete polyhedra are given as

$$\begin{bmatrix} T & 0 & 0 \\ P_1 & P_2 & 0 \\ S_1 & S_2 & S_3 \end{bmatrix} \begin{bmatrix} z_t \\ z_p \\ z_s \end{bmatrix} \geq \begin{bmatrix} t \\ p \\ s \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} T & 0 & 0 \\ P_1 & P_2 & 0 \\ D_1 & D_2 & D_3 \end{bmatrix} \begin{bmatrix} z_t \\ z_p \\ z_d \end{bmatrix} \geq \begin{bmatrix} t \\ p \\ d \end{bmatrix}$$

Now consider the two `Data-Variable-Expr`'s that occur in a LACS program. Each one consists of the name of a multidimensional array variable, and a list of `index-expr`'s which yield an affine indexing expression. We denote the source indexing function as $Az_1 + a$ and the destination indexing function as $Bz_2 + b$, where A and B are matrices of appropriate dimensions and a and b are column vectors. A and B may be partitioned into appropriate columns that access the temporal, parallel, source and destination indices respectively, $A = [A_1|A_2|A_3]$ and $B = [B_1|B_2|B_3]$.

Definition 1 Any LACS program may be written as follows (the $\oplus$ is the optional, binary reduction operator, if present).

$$\left\{ \begin{bmatrix} T & 0 & 0 \\ P_1 & P_2 & 0 \\ S_1 & S_2 & S_3 \end{bmatrix} \begin{bmatrix} z_t \\ z_p \\ z_s \end{bmatrix} \geq \begin{bmatrix} t \\ p \\ s \end{bmatrix} : X(A_1 z_t + A_2 z_p + A_3 z_s + a) \right\}$$

[*]One consequence of this is that for any constant vector ρ, there is always a "large enough" problem size—one for which the other polyhedra (temporal, parallel, source and destination) contain at least two points separated by ρ. This will be useful later on.

$$\oplus \Rightarrow \left\{ \begin{bmatrix} T & 0 & 0 \\ P_1 & P_2 & 0 \\ D_1 & D_2 & D_3 \end{bmatrix} \begin{bmatrix} z_t \\ z_p \\ z_d \end{bmatrix} \geq \begin{bmatrix} t \\ p \\ d \end{bmatrix} : Y(B_1 z_t + B_2 z_p + B_3 z_d + b) \right\} \qquad (1)$$

A single PAS may be represented as

$$\left\{ \begin{bmatrix} P_2 & 0 \\ S_2 & S_3 \end{bmatrix} \begin{bmatrix} z_p \\ z_s \end{bmatrix} \geq \begin{bmatrix} p \\ s \end{bmatrix} : X(A_2 z_p + A_3 z_s + a) \right\}$$
$$\oplus \Rightarrow \left\{ \begin{bmatrix} P_2 & 0 \\ D_2 & D_3 \end{bmatrix} \begin{bmatrix} z_p \\ z_d \end{bmatrix} \geq \begin{bmatrix} p \\ d \end{bmatrix} : Y(B_2 z_p + B_3 z_d + b) \right\} \qquad (2)$$

This description may be obtained directly from an abstract syntax tree of the program.

A final bit of notation: for a linear map, $L : Z^n \rightarrow Z^m$, its null space, $\mathrm{N}(L)$ and range space are given by $\mathrm{N}(L) = \{z \in Z^n : Lz = 0\}$ and $\mathrm{R}(L) = \{z \in Z^m : \exists x \in Z^n,\ z = Lx\}$. Also, if the rank of the transformation is r, L is said to be **quasi unimodular** iff the gcd of the determinants of all $r \times r$ square submatrices of L is ± 1. One can show that L is a 1-to-1 function on integral points iff it is quasi unimodular (this is a technicality and is needed because we are not really dealing with vector spaces but lattices—integral points). We assume for the analysis, that the matrices A_1, A_2 and A_3 (and likewise the three B matrices) are quasi unimodular. This reduces the expressive power of the language, but is not a serious limitation; it may be possible to extend LACS to address collections of indices which are *linearly bounded lattices* (LBL's [21]) but this is a matter for future work.

5 Static Analysis I: Type Checking

We now address the problem of compile time detection of the properties of a given LACS program. As mentioned earlier, we restrict the analysis to the level of a single PAS. A complete analysis involving the temporal components is beyond the scope of this paper, but must necessarily build on the analysis presented here. Note that many important properties of interest such as conflicts, message time estimates, etc., depend on the set of communication activities that are to occur at a single instant of time, and a PAS is the syntactic construct in LACS that contains the relevant information. In order to motivate the discussion, consider the following LACS program, each of which illustrates a particular problem.

Example 5 `A[0] => {i | 0<i<=N: X[0,N]}` This ACE apparently specifies broadcast (the `Dst-Expr` contains a one-dimensional polyhedron), but the destination is really a single point, `X[0,N]`.

Example 6 `{i | 0<i<=N: X[10,0]} *=> A[N]` This is the dual—an apparent reduction, but there is only one point in the `Src-Expr`.

Example 7 `{i | 0<i<=N: X[i,i] => X[0,0] }` This PAS has multiple ACE's which write to the same destination, leading to a write conflict.

Example 8 `{i | 0<i<=N: X[i] => {j | 0<j<=N-i : Y[i+j]} }` This is another, subtle, example of a write conflict (it may be helpful to draw a diagram).

Example 9 `{i | 0<i<=N: X[0,0] => X[i,0] }` This is a PAS where different ACE's have the same source value—the program really specifies a broadcast but as a collection of independent ACE's. It should really be written as `A[0] => {i | 0<i<=N: X[i,0]}`

Definition 2 A PAS is said to have a **false broadcast** (cf **false reduction**) if there is any multi-edge in the corresponding multi-graph, whose destination (cf source) node set contains elements that are repeatedly enumerated.

A PAS is said to have a **write conflict** if there are distinct multi-edges whose destination sets have a non-null intersection.

A PAS has an **undeclared broadcast** if there are distinct multi-edges with the identical source nodesets.

A PAS is said to be **well formed** if it has no false broadcasts no false reductions, no write conflicts and no undeclared broadcasts.

Theorem 1 *A PAS has a false broadcast iff B_3 is not of full column rank.*

Proof: See Appendix II ∎

Theorem 2 *A PAS has a false reduction iff A_3 is not of full column rank.*

Proof: Analogous to Th 1. ∎

Corollary 1 *By projecting the source (cf destination) polyhedron along* $\mathrm{N}(A_3)$ *onto* $\mathrm{R}(A_3)$ *(cf* $\mathrm{N}(B_3)$ *onto* $\mathrm{R}(B_3)$*), and properly rewriting the accessing functions in the* `Src-Expr` *(cf* `Dst-Expr`*), any false reductions and broadcasts may be* automatically *removed.*

Observe that for Ex 5 and Ex 6 above, we have $B_3 = [0,0]^{\mathrm{T}}$ and $A_3 = [0,0]^{\mathrm{T}}$, both of which do not have full column rank, and there is a false broadcast (reduction). Both their null spaces are spanned by `[1]`, so we project the polyhedron in this direction, yielding a polyhedron with a single point. Hence, they may be rewritten as `A[0] => X[0,N]` and `X[10,0] => A[N]` respectively.

Example 10 Consider the PAS, `X[0,0] => {i,j | 0<i,j<=10: X[i+j,i+j]}` Here, the destination polyhedron is a 10×10 square, and $A_3 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, which does not have full column rank, and so we have a false broadcast. Its null space is spanned by $[1,-1]^{\mathrm{T}}$. Projecting the destination polyhedron in this direction (and removing the redundant index) yields `{k | 1<k<=20}`. The equivalent ACE is `X[0,0] => {k| 1<k<=20: X[k,k]}`

At this point, (as you have probably discovered) you will find it very useful to have a graph paper at hand (an engineering pad or something with grid lines will do just as well). A useful trick when you read a PAS, is to first draw out all the points in the parallel polyhedron (usually this is two-dimensional). Then enumerate the simpler one of the Src or Dst (the one whose polyhedron is a singleton). Finally, you can start enumerating the points of the other side.

Theorem 3 *A PAS has a write conflict iff B_2 is not of full column rank or the range spaces of B_2 and B_3 have a non-trivial intersection, (i.e.,* $\mathrm{R}(B_2) \cap \mathrm{R}(B_3) \neq \{0\}$*).*

Proof: See Appendix II ∎

Intuitively, the condition, "B_2 is not of full column rank" corresponds to a case when the *entire* set of destination points for different ACE's is the same[*], whereas the condition, $\mathrm{R}(B_2) \cap \mathrm{R}(B_3) \neq \{0\}$ occurs when there is a non-null intersection among the destination sets of different ACE's, but the sets themselves may be distinct.

An undeclared broadcast occurs (see Def 2) when there are two or more distinct multi-edges (i.e., ACE's), with identical source nodesets (mere intersection will not do—it simply indicates a "read conflict," which is no problem). Hence, our first intuition would be that that this occurs iff A_2 is not of full column rank (a la Th 3). However, this is not strictly true. There are some subtle issues, as we will see.

First, for Ex 9, i.e., `{i | 0<i<=N: X[0,0] => X[i,0]}`, we can check that A_2 is $[0,0]^{\mathrm{T}}$, which does not have full column rank. We can also check that there is an undeclared broadcast—so far, so good. However, consider the following PAS, which has exactly the same A_2. `{i | 0<i<=N: {j | 0<j<=i: X[j,j]} *=> X[i,i]}` We may easily verify that it is a scan on the main diagonal—a perfectly valid program! The difference is that, although the index `i` is not used to *access* `X`, it is used to define the different source polyhedra. We formally characterize this in the following theorem, and we present numerous examples illustrating the result.

Theorem 4 *A PAS has an undeclared broadcast iff* $\exists \begin{bmatrix} \rho_p \\ \rho_s \end{bmatrix}, \rho_p \neq 0, \in \mathrm{N}(A) \cap \mathrm{N}(S)$.

Proof: See Appendix II ▮

Now, let us look at a few more examples. They illustrate many of the subtle issues that come up—some very slight changes to the programs produce very different results. You will find your skills of tracing out the PAS on a graph paper very handy for these examples Note that all of them have the same parallel polyhedron, namely, `i,j | 0<i,j<=10`. Using our internal representation (Def 1) this polyhedron is given by

[*]This is not strictly accurate, but will suffice for an intuitive explanation.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ -10 \\ -10 \end{bmatrix} \quad \text{henceforth denoted as} \quad P \begin{bmatrix} i \\ j \end{bmatrix} \geq p$$

The destination expression in all these examples is simply `X[i,j]`, and for each one, we also show the internal representation (of the source side only).

Example 11 `{i,j | 0<i,j<=10: X[i+j,i+j] => X[i,j]}`

$$\text{or} \left\{ P \begin{bmatrix} i \\ j \end{bmatrix} \geq p : X \left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \right) \right\} \Rightarrow \ldots$$

This PAS has no source polyhedron (S_2, S_3 and A_3 do not exist), and hence the condition becomes $\exists \rho_p \neq 0 \in \mathrm{N}(A_2)$ which is obviously true, and we have an undeclared broadcast. It is instructive to draw a figure and verify that all points in the direction $[1, -1]$ will have the same sources.

Example 12 `{i,j | 0<i,j<=10: {k | 0<k<=i : X[i+j,i+j-k]} *=> X[i,j]}`

$$\text{or} \left\{ \left[\begin{array}{cc|c} \multicolumn{2}{c|}{P} & \mathbf{0} \\ \hline 0 & 0 & 1 \\ 1 & 0 & -1 \end{array} \right] \begin{bmatrix} i \\ j \\ k \end{bmatrix} \geq \left[\begin{array}{c} p \\ \hline 1 \\ 0 \end{array} \right] : X \left(\left[\begin{array}{cc|c} 1 & 1 & 0 \\ 1 & 1 & -1 \end{array} \right] \begin{bmatrix} i \\ j \\ k \end{bmatrix} \right) \right\} * \Rightarrow \ldots$$

Now, we have a new index k in the source polyhedron. A_2 is still the same, but we now have an $S = [S_2 | S_3]$ and A_3. We can check that $\mathrm{N}(A)$ and $\mathrm{N}(S)$ are one dimensional subspaces (spanned by $[1, 1, 0]^{\mathrm{T}}$ and $[0, 1, 0]^{\mathrm{T}}$ respectively). Their intersection is only $\mathbf{0}$, so there is no undeclared broadcast. Use the figures to convince yourself that it really specifies a scan.

Example 13 `{i,j | 0<i,j<=10: {k | 0<k<=i+j : X[i+j,i+j-k]} *=> X[i,j]}`

$$\text{or} \left\{ \left[\begin{array}{cc|c} \multicolumn{2}{c|}{P} & \mathbf{0} \\ \hline 0 & 0 & 1 \\ 1 & 1 & -1 \end{array} \right] \begin{bmatrix} i \\ j \\ k \end{bmatrix} \geq \left[\begin{array}{c} p \\ \hline 1 \\ 0 \end{array} \right] : X \left(\left[\begin{array}{cc|c} 1 & 1 & 0 \\ 1 & 1 & -1 \end{array} \right] \begin{bmatrix} i \\ j \\ k \end{bmatrix} \right) \right\} * \Rightarrow \ldots$$

This is identical to the previous one except that the bounds of the source polyhedron are slightly changed. Observe that the right null spaces of S_2 and A_2 have a non-trivial intersection (this is stronger than the condition in Th 4), and as a result we can state (and also independently verify) that there is an undeclared broadcast.

Example 14 `{i,j | 0<i,j<=10: {k | 0<k<=i+j : X[i+j,i+k]} *=> X[i,j]}`

$$\text{or}\ \left\{\left[\begin{array}{cc|c} \multicolumn{2}{c|}{P} & \mathbf{0} \\ \hline 0 & 0 & 1 \\ 1 & 1 & -1 \end{array}\right]\left[\begin{array}{c} i \\ j \\ k \end{array}\right] \geq \left[\begin{array}{c} p \\ \hline 1 \\ 0 \end{array}\right] : X\left(\left[\begin{array}{cc|c} 1 & 1 & 0 \\ 1 & 0 & 1 \end{array}\right]\left[\begin{array}{c} i \\ j \\ k \end{array}\right]\right)\right\} * \Rightarrow \ldots$$

Now we have changed A_2, so that it has full column rank, and this makes it a perfectly legal LACS program (it is not exactly a scan, but can be written as the sum of two scans).

Example 15 `{i,j | 0<i,j<=10: {k | i<k<=i+j : X[i+j+k,i+j+k]} *=> X[i,j]}`

$$\text{or}\ \left\{\left[\begin{array}{cc|c} \multicolumn{2}{c|}{P} & \mathbf{0} \\ \hline -1 & 0 & 1 \\ 1 & 1 & -1 \end{array}\right]\left[\begin{array}{c} i \\ j \\ k \end{array}\right] \geq \left[\begin{array}{c} p \\ \hline 1 \\ 0 \end{array}\right] : X\left(\left[\begin{array}{cc|c} 1 & 1 & 1 \\ 1 & 1 & 1 \end{array}\right]\left[\begin{array}{c} i \\ j \\ k \end{array}\right]\right)\right\} * \Rightarrow \ldots$$

Finally, we have an A_2 which is singular, but and S_2 which is not. It is very instructive to study this carefully—it has an undeclared broadcast and illustrates why the condition of Ex 13 above was too strong. The real condition is exactly as given in Th 4

6 Static Analysis II: Inferring the Communication

Once we have a well formed PAS (either automatically using the results of Ths 1-4, or after write conflicts, if any, are removed by the programmer), we can readily infer many communication properties from the PAS. The number of dimensions in the source and destination polyhedra directly tell us if there is a reduction and/or a broadcast (the volume of the polyhedra tell us the number of data values involved). Furthermore, the number of dimensions in the parallel polyhedron and its volume tells us the number of *independent* communication events that take place. All this information can be used in order to determine exactly which routine in the communication library is to be called. A more interesting result is our ability to detect scan operations, as we now show.

6.1 Generalized Scan Operations

Recall that a scan of a vector, X, is Y, where $Y_i = \sum_{j=1}^{i} X_j$. They use the notion of a *prefix* in two crucial ways. First, every element in the vector has a prefix which is also a vector. Second, there is an ordering among the elements of the vector such that their prefixes are monotonic—the prefix of `X[i]` is a proper subset of the prefix of `X[i+1]`. We have already seen numerous examples of how LACS allows you to generalize this to multiple dimensions, by simply extending the idea of prefixes to polyhedra. Let us define this formally.

Definition 3 A PAS is said to have a scan, iff there is a family of multi-edges with an order relation between them such that the corresponding source nodesets are proper subsets.

Now remember that the multi-edges are labeled by index points in the parallel polyhedron, and their source nodesets are simply the images of the corresponding source polyhedra, transformed by the affine map (A, a). As a result, we have the following theorem.

Theorem 5 *A well formed PAS has a scan iff* $\exists \rho = [\rho_p, \rho_s]^{\mathrm{T}}, \rho_p \neq 0, \in \mathrm{N}(A)$.

Proof: The proof is almost identical to that of Th 4. See Appendix II for details. ∎

Let us take a second look at Ex 12. Here, we see that although the right null spaces of A and S did not have a non-trivial intersection, the fact that A was rank deficient implies a scan. Let us study a few more examples.

Example 16 `{i,j | 0<j<=i<=10: {k,l | j<=l<=i<=k<=10: X[k,l]} *=> X[i,j]}`

$$\left\{ \left[\begin{array}{rr|rr} 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ \hline 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{array}\right] \left[\begin{array}{c} i \\ j \\ \hline k \\ l \end{array}\right] \geq \left[\begin{array}{r} 1 \\ 0 \\ -10 \\ \hline 0 \\ 0 \\ 0 \\ -10 \end{array}\right] : X\left(\left[\begin{array}{cc|cc} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} i \\ j \\ k \\ l \end{array}\right] \right) \right\} * \Rightarrow \ldots$$

We can check that since S is of full column rank, there is no undeclared broadcast. But, A_2 is the zero matrix, and so the right null space of A is the two-dimensional subspace spanned by $[1,0,0,0]^T$ and $[0,1,0,0]^T$, and there is a scan. We will call the nullity of A the dimension of the scan, so this is a two-dimensional scan. In fact it corresponds to a scan where every point in the lower triangular region of the `[i,j]` index space (i.e., `i` $\geq$ `j`) gets the product of everything that is below and to its right. The two bases for the scan are `[1,0]` and `[0,1]`.

Example 17 `{i,j | 0<=j<=i<=10: {k | i<=k<=10 : X[k,j]} *=> X[i,j]}`

$$\text{or } \left\{ \left[\begin{array}{rr|r} 0 & 1 & 0 \\ 1 & -1 & 0 \\ -1 & 0 & 0 \\ \hline -1 & 0 & 1 \\ 0 & 0 & -1 \end{array}\right] \left[\begin{array}{c} i \\ j \\ \hline k \end{array}\right] \geq \left[\begin{array}{r} 1 \\ 0 \\ -10 \\ \hline 0 \\ -10 \end{array}\right] : X\left(\left[\begin{array}{cc|c} 0 & 0 & 1 \\ 0 & 1 & 0 \end{array}\right] \left[\begin{array}{c} i \\ j \\ k \end{array}\right] \right) \right\} * \Rightarrow \ldots$$

We again have an S with full column rank, so there is no undeclared broadcast. This time $\mathrm{N}(A)$ is spanned by $[1,0,0]^T$, and hence we only have a one dimensional scan. Again, we can verify that it consists of independent scans on the columns (restricted to the lower triangular region).

Example 18 `{i,j | 0<j<=i<=10: {k,l | 0<=l<=i<=k<=10: X[k,l]} *=> X[i,j]}`

This is identical to Ex 16, except that the lower bound on `l` is now 0 (rather than `j`). Unfortunately, this results in an S matrix that is rank deficient (`j` is not used anymore) and this has an undeclared broadcast. We can remove it by reducing the dimension of the parallel polyhedron, yielding the following:
`{i| 0<=i<=10: {k,l| 0<=l<=i<=k<=10: X[k,l]} *=> {j| 0<=j<=10: X[i,j]}}`

$$\text{or } \left\{ \left[\begin{array}{r|rr} 1 & 0 & 0 \\ -1 & 0 & 0 \\ \hline 0 & 0 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 0 \end{array}\right] \left[\begin{array}{c} i \\ \hline k \\ l \end{array}\right] \geq \left[\begin{array}{r} 1 \\ -10 \\ \hline 1 \\ 0 \\ 0 \\ -10 \end{array}\right] : X\left(\left[\begin{array}{c|cc} 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} i \\ k \\ l \end{array}\right] \right) \right\}$$

$$\Rightarrow \left\{ \left[\begin{array}{r|r} 1 & 0 \\ -1 & 0 \\ \hline 0 & 1 \\ 1 & -1 \end{array}\right] \left[\begin{array}{c} i \\ \hline j \end{array}\right] \geq \left[\begin{array}{r} 1 \\ -10 \\ \hline 1 \\ 0 \end{array}\right] : X\left(\left[\begin{array}{c|c} 1 & 0 \\ 0 & 1 \end{array}\right] \left[\begin{array}{c} i \\ j \end{array}\right] \right) \right\}$$

Here, we have a one dimensional scan (as $\mathrm{N}(A)$ is spanned by $[1,0,0]^{\mathrm{T}}$), but the source polyhedron for each prefix is two dimensional (there are two free variables in the source polyhedron). Hence we have a scan where the result (for any row `i`) is the product of all the data on the rows below `i` (within the lower triangular region, of course); this result is broadcast along the row. Now we see that if we wanted to implement this we would probably call a library to first reduce each row independently, then do a scan of these results and then broadcast the results of the scans back along the row. There is freedom in choosing the exact one dimensional region over which to do the scan. We could do it on the first column, or we could do it on the diagonal.

6.2 Analysis based on Data Allocation

Recall that data variables in a LACS program are merely multidimensional arrays. Their indices do not necessarily correspond to rows or columns on a processor grid. All the analysis presented so far is independent of any such physical interpretation. However, in data parallel programs, the arrays are allocated to physical processors, and this is what ultimately determines the communication. Furthermore, there are certain communication patterns such as scatters, gathers, total exchanges, etc. which rely on such an interpretation. For example, consider the PAS, `{i | 0<i<=N : X[1,i] => Y[i,1]}`

If the two dimensions of `X` and `Y` are interpreted as the two axes of a processor grid, this represents a set of point-to-point transfers. However, if the first dimension is the processor id, and the second is a local address, this is a scatter from processor 1, and if the second dimension represents processor id's we have a gather by processor 1. Similarly, the PAS,

```
{i,j | 0<i,j<=N : X[i,j] => Y[j,i]
```

could be a transpose, or a total exchange (in the light of this, it may be useful to briefly review Exs 4-6 from the sneak preview).

We adopt the standard notation [16] that the first m (typically 2) components of an index expression are the processor id, and the l remaining ones correspond to local memory address. Thus, the array reference $X[z]$ or $X[z_m|z_l]$ refers to the z-th element element of array X which resides on processor z_m and has a local address of z_l.

Definition 4 Given a LACS program, we define its **processor projection** as the LACS program obtained by discarding all except the first z_m indices of each data variable. Along the lines of Def 1, the internal representation of the processor projection is as follows:

$$\left\{ \begin{bmatrix} T & 0 & 0 \\ P_1 & P_2 & 0 \\ S_1 & S_2 & S_3 \end{bmatrix} \begin{bmatrix} z_t \\ z_p \\ z_s \end{bmatrix} \geq \begin{bmatrix} t \\ p \\ s \end{bmatrix} : X(A'_1 z_t + A'_2 z_p + A'_3 z_s + a') \right\}$$

$$\oplus \Rightarrow \left\{ \begin{bmatrix} T & 0 & 0 \\ P_1 & P_2 & 0 \\ D_1 & D_2 & D_3 \end{bmatrix} \begin{bmatrix} z_t \\ z_p \\ z_d \end{bmatrix} \geq \begin{bmatrix} t \\ p \\ d \end{bmatrix} : Y(B'_1 z_t + B'_2 z_p + B'_3 z_d + b') \right\} \qquad (3)$$

where $A' = [A'_1|A'_2|A'_3]$ and $B' = [B'_1|B'_2|B'_3]$ (and a' and b') are simply the first m rows of the A and B matrices (and a and b), respectively.

We now have the following results directly from Ths 1-4 when applied to the processor projection of a PAS.

Result 1 If the processor projection of a well formed PAS has a false broadcast, it implies that there are multiple destinations of the broadcast that reside on the same processor. Furthermore, the number of dimensions in $\mathrm{N}(A'_3)$ gives the number of indices that can be eliminated from the destination expression in the original ACE—the result is simply copied by the destination processor.

We can analogously determine the true nature of reductions.

Result 2 If the processor projection of a well formed PAS has an undeclared broadcast, it implies that a single source processor is sending to multiple destinations and this is a scatter (since the original PAS is well formed). Again, the dimensionality of $\mathrm{N}(A') \cap \mathrm{N}(S)$ gives the number of dimensions in the scatter.

The dual is that there is a gather iff $\exists \rho = [\rho_p, \rho_s]^{\mathrm{T}}, \rho_p \neq 0 : \begin{bmatrix} B' \\ S' \end{bmatrix} \rho = 0$. As before, the dimensionality of $\mathrm{N}(B') \cap \mathrm{N}(S)$ gives the number of dimensions in the scatter.

7 Conclusions

In the past, researchers identified common communication patterns such as scatters, gathers, reductions, etc., and developed optimal libraries for specific topologies [2, 10]. The problem of deciding which library routine to use was left to the user. Li and Chen extended this in two main directions [13]. Their work was in the context of compiling massively parallel programs to hypercube. They assumed that the multidimensional meshes of the virtual index space was first embedded in the cube, and the program was compiled to this virtual mesh. They first specialized the libraries so that only a subset (corresponding to one or more dimension of the virtual mesh) could participate in a particular communication, and used accurate cost measures for each library call. Their second contribution was to show how the library calls could be generated automatically through a dependency analysis. Gupta and Bannerjee [9] used these results to address the optimal data partitioning problem. Essentially they formulated the choice of the data alignment (which was assumed to be given in Li and Chens' work) as an optimization problem.

Knobe, Lukas et al [11, 12, 15] have addressed the problem of array allocation to reduce residual communication. Their work is based on alignment preferences which are determined from the access patterns in the program. They characterize an allocation as good or bad depending on whether it causes communication or not—a simple binary metric. Similarly, Ramanujam and Sadayappan [18] present an analysis of affine dependencies in a program, and show that there is no communication if a certain system of diophantine equations has a solution (of a particular form).

Recently, Chatterjee et al [6] have addressed the problem of optimal array alignment. They use earlier results by Gilbert and Schreiber [8] showing how to minimize the cost of evaluation of an expression tree. They also use certain metrics of the architecture, and they classify the communication into three kinds—nearest neighbors (shifts), tree (reductions, scans, spreads) and unstructured (transpose, stride changes, etc.). They show how to solve the problem of optimal array alignment (i.e., choosing the axis, strides and offsets that minimizes communication cost).

I'm sure that there has been other work on compiling for communication, and if you know of any related work, I would appreciate if you drop me a line.

The work presented here is complementary. We have (for the moment) looked at the problem of merely *specifying* the communication in an abstract, architecture independent manner. The background of this was in our language LaRCS [14] which was developed in the context of mapping. We noted that most of the previous work in mapping used a graph theoretic model of the computation [5, 20], and did not explicitly describe temporal information. LaRCS was developed to address this limitation, and includes the notion of collecting edges into groups that are simultaneously active, and of giving an order for this activity.

LACS extends this by using multi-graphs (which lets us describe broadcasts and reductions), and in using convex polyhedra and affine transformations. We found LACS is surprisingly simple yet powerful—many practical communication patterns can be expressed as APL-style one-liners. In this paper, we have shown how properties of a Parallel Assignment Statement in a LACS program can be automatically inferred, by using standard results from linear algebra, convex polyhedra and affine transformations. We anticipate that this method of static analysis can be used to first develop an accurate cost model of communication. Rather than simply using the *number* of non-local accesses as in the past, we expect that we can develop a true cost metric based on the machine architecture, distance, volume, presence of conflicts, etc. Using these accurate cost measures, we expect that the data partitioning problem can be formulated as an optimization problem. We believe that this will address the short, medium and long term goals as stated in Sec 1.

Acknowledgments Thanks to Rumen Andonov, Patrice Quinton and especially, Doran Wilde, for many helpful discussions.

References

[1] E. A. Ashcroft and W. W. Wadge. *Lucid, the Data-Flow Programming Language.* Academic Press, 1985.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation – Numerical Methods.* Prentice Hall, 1989.

[3] G. Bleloch. *Vector Models for Data Parallel Computing.* The MIT Press, 1990.

[4] G. Bleloch and G. W. Sabot. Compiling collection oriented languages onto massively parallel processors. *Journal of Parallel and Distributed Computing*, 8(2), February 1990.

[5] S. H. Bokhari. *Assignment Problems in Parallel and Distributed Computing.* Kluwer Academic Publishers, 1987.

[6] S. Chatterjee, J. Gilbert, R. Schreiber, and S-H. Teng. Automatic array alignment in data-parallel programs. In *Principles of Programming Languages (POPL)*, pages 16–28, ACM, Charleston, S.C., Jan 1993.

[7] Marina C. Chen. A parallel language and its compilation to multiprocessor machines for VLSI. In *Principles of Programming Languages*, ACM, 1986.

[8] J. R. Gilbert and R. Schreiber. Optimal expression evaluation for data-parallel architectures. *Journal of Parallel and Distributed Computing*, 13(1):58–64, September 1991.

[9] M. Gupta and P. Bannerjee. Demonstration of automatic data partitioning techniques for parallelizing compilers on multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, ?(?):179–193, Mar 1991.

[10] S. L. Johnsson and C. T. Ho. Optimal broadcasting and personal communication in the hypercube. *IEEE Transactions on Computers*, C-38(10):1249–1268, September 1989.

[11] K. Knobe, J. D. Lukas, and W. J. Dally. Dynamic memory alignment on distributed memory systems. In *Proc. Workshop on Compilers for Parallel Computers*, pages 394–404, Austrian Center for Parallel Computation, Vienna, Austria, July 1992.

[12] K. Knobe, J. D. Lukas, and G. L. Steele. Data optimization: allocation of arrays to reduce communication on simd machines. *Journal of Parallel and Distributed Computing*, 8(?):102–118, ? 1990.

[13] J. Li and M. C. Chen. Compiling communication efficient programs for massively parallel machines. *IEEE Transactions on Parallel and Distributed Systems*, 2(3):361–376, July 1991.

[14] V. M. Lo, S. Rajopadhye, M. A. Mohamed, S. Gupta, D. Keldsen, W. Nitzberg, J. Telle, and X. Zhong. LaRCS: a language for describing parallel computations. *IEEE Transactions on Parallel and Distributed Systems*, 1993. to appear.

[15] J. D. Lukas. Data locality for shared memory. In *SIAM Conference on Parallel Processing for Scientific Computing*, pages 836–839, SIAM, Norfolk, VA, March 1993.

[16] Jan Prins. A framework for efficient execution of array-based languages on SIMD computers. In *Frontiers of Massively Parallel Computing*, 1990.

[17] S. V. Rajopadhye and M. Muddarangegowda. Parallel assignment, reduction and communication. In *SIAM Conference on Parallel Processing for Scientific Computing*, pages 849–853, SIAM, Norfolk, VA, March 1993.

[18] Ramanujam and P. Sadayappan. Compile time techniques for data distribution in distributed memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):472–482, Oct 1991.

[19] G. W. Sabot. *The Paralation Model: Architecture Independent Programming.* The MIT Press, 1988.

[20] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, SE-3(1):85–93, January 1977.

[21] J. Teich and L. Thiele. Partitioning of processor arrays: a piecewise regular approach. *INTEGRATION: The VLSI Journal*, 14(3):297–332, Feb 1993.

Appendix I: LACS Syntax Highlights

```
Polyhedron ::= Index-var-list '|' Constraint-list
Index-var-list ::=  (a comma separated list of Index-Variable)
Constraint-List ::= (a semicolon separated list of linear inequalities)
```

Some additional shorthand is permitted, as shown in the examples.

```
Data-Variable-Expr ::= Data-Variable '[' Index-Vector ']'
Index-Vector ::=  (a comma separated list of Index-expr)
Index-Expr ::=  (an add-op separated list of Term)
Term ::= Constant | Index-Variable | Constant Index-Variable
```

```
Transfer-op ::= '=>' | op '=>'                      (op is a reduction operator)
ACE ::= Src-Expr Transfer-Op Dst-Expr
Src-Expr ::= Dst-Expr
Dst-Expr ::= '{' Polyhedron ':' Data-Variable-Expr '}'
Dst-Expr ::= Data-Variable-Expr                 (Polyhedron has 0 dimensions)
```

The previous rule is a shorthand for the case when polyhedron is a single point, as also the second rule below (the ACE consists of a single source and/or destination point, or th PAS consists of a single ACE):

```
PAS ::= '{' Polyhedron ':' ACE '}'
PAS ::= ACE
```

```
PAS-list ::= (a semicolon separated list of PAS)
PROG ::= '{' Polyhedron ':' 'lex' PAS-list '}'
PROG ::= PAS-list
```

Appendix II: Proofs

Proof of Th 1:

$\boxed{\Rightarrow}$ If B_3 is not of full column rank, it has a non-trivial right null space. Hence, $\exists \rho \neq 0 : B_3\rho = 0$. Hence, for some large enough problem (see footnote, pp 14), $\exists z_d \in D_d : z'_d = z_d + \rho \in D_d$. Thus, for any z_p, $Y[B_2z_p + B_3z_d + b] = Y[B_2z_p + B_3z'_d + b]$, and the same destination index is repeatedly specified. So we have a false broadcast.

$\boxed{\Leftarrow}$ If the PAS has a false broadcast, the same destination index must be multiply specified within the destination polyhedron of some ACE, i.e., $\exists z_d, z'_d \in D_d, z_d \neq z'_d$, such that for some $z_p \in D_p$, $B_2z_p + B_3z_d + b = B_2z_p + B_3z'_d + b$. But this implies that $B_3(z_d - z'_d) = 0$, i.e., B_3 does not have full column rank.

∎

Proof of Th 3:

$\boxed{\Rightarrow_1 \text{ (B_2 not full column rank implies write conflict)}}$. If B_2 is not of full column rank, it has a non-trivial right null space, i.e. $\exists \rho : B_2\rho = 0$. So $\exists z_p \in D_p$, st, for some large enough problem, $z'_p = z_p + \rho \in D_p$. Hence for all z_d, $Y[B_2z_p + B_3z_d + b] = Y[B_2z'_p + B_3z_d + b]$. Thus the same destination index is repeatedly specified in *different* ACE's within the PAS, and we have a write conflict.

$\boxed{\Rightarrow_2 \text{ ($\mathrm{R}(B_2) \cap \mathrm{R}(B_3) \neq \{0\}$ implies write conflict)}}$. Let $B_2x = B_3y \neq 0$ for some non-zero x, y. Consider the destination points, $B_2z_p + B_3z_d + b$ and $B_2z'_p + B_3z'_d + b$, where $z'_p = z_p + x$ and $z'_d = z_d - y$. Now

$$\begin{aligned} B_2z'_p + B_3z'_d + b &= B_2(z_p + x) + B_3(z_d - y) + b \\ &= B_2z_p + B_2x + B_3z_d - B_3y + b \\ &= B_2z_p + B_3z_d + b \end{aligned}$$

Hence the two destination points are the same and we have a write conflict.

$\boxed{\Leftarrow \text{ (write conflict implies one of the two conditions above)}}$ If a write conflict exits, we have z_p, z'_p, z_d, z'_d, where $z_p \neq z'_p$ such that they write to the same destination, i.e., $B_2z_p + B_3z_d + b = B_2z'_p + B_3z'_d + b$. Hence, $B_2(z_p - z'_p) = B_3(z'_d - z_d) = \rho$ (say). So ρ belongs to the range space of both B_2 and B_3. Now if $\rho = 0$, then $z_p - z'_p$ is a non-zero vector in the right null space of B_2, and so it does not have full column rank. Otherwise, the second condition is satisfied. ∎

Proof of Th 4: Note that the condition can be written as $\exists \rho = [\rho_p, \rho_s]^{\mathrm{T}}, \rho_p \neq 0 :$ $\begin{bmatrix} A_2 & A_3 \\ S_2 & S_3 \end{bmatrix} \rho = 0$. From Def 2, we know that there is an undeclared broadcast iff the source nodesets for two distinct multi-edges are identical.

$\boxed{\Rightarrow}$ Let there be an undeclared broadcast. So $\exists z_p, z'_p \in D_p$, for $z'_p - z_p = \rho_p \neq 0$, such that the images, under the affine map (A, a), of the two polyhedra, $S_2 z_p + S_3 z_s \geq s$ and $S_2(z_p + \rho_p) + S_3 z_s \geq s$, are the same. Since, from Th 2, A_3 must be of full column rank (otherwise there is a false reduction which can be automatically removed) the two images are identical iff there is a bijection between the source polyhedra. Hence, $\forall z_s : S_3 z_s \geq s - S_2 z_p$, $\exists z'_s = f(z_s)$, such that $S_2(z_p + \rho_p) + S_3 f(z_s) \geq s$ and

$$\begin{aligned} & A_2 z_p + A_3 z_s + a = A_2(z_p + \rho_p) + A_3 f(z_s) + a \\ \text{i.e.,} \quad & A_3(z_s - f(z_s)) = A_2 \rho_p \end{aligned}$$

Since A_3 is of full column rank, this implies that $f(z_s) - z_s$ is a constant, say ρ_s and the bijection is really a translation. If we let $\rho = [\rho_p, \rho_s]^{\mathrm{T}}$, we see that $\rho \in \mathrm{N}(A)$. Now, the polyhedron, $S_3 z_s \geq s - S_2 z_p$, translated by ρ_s, is given by $S_3 z_s \geq s - S_2 z_p + S_3 \rho_s$, and this must be the same polyhedron as $S_3 z'_s \geq s - S_2 z_p - S_2 \rho_p$. Thus, $S_2 \rho_p + S_3 \rho_s = 0$, i.e., $\rho \in \mathrm{N}(S)$.

$\boxed{\Leftarrow}$ Let $\rho = [\rho_p, \rho_s]^{\mathrm{T}} \in \mathrm{N}(A) \cap \mathrm{N}(S)$, and $\rho_p \neq 0$. For any z_s in the source polyhedron of some ACE, z_p, the point $z_s + \rho_s$ belongs to the source polyhedron of the ACE, $z_p + \rho_p$ (since $\rho \in \mathrm{N}(S)$). Conversely, for any point, z'_s in the source polyhedron of ACE, z'_p, $z'_s - \rho_s$ belongs to the source polyhedron of ACE, $z'_p - \rho_p$. Hence the two polyhedra are identical, except for translation by ρ_s. Now, since $\rho \in \mathrm{N}(A)$, they also have the same image under the transformation, (A, a), so we have an undeclared broadcast. ∎

Proof of Th 5:

$\boxed{\Rightarrow}$ If there is a scan operation, $\exists z_p, z'_p \in D_p$, for $z'_p - z_p = \rho_p \neq 0$, such that the image of the polyhedron, $S_2 z_p + S_3 z_s \geq s$ under the map (A, a), is a proper subset of the image of $S_2(z_p + \rho_p) + S_3 z_s \geq s$. This implies an *injective* (not bijective, as in Th 4) translation by some constant ρ_s, and hence, $[\rho_p, \rho_s]^{\mathrm{T}} \in \mathrm{N}(A)$.

$\boxed{\Leftarrow}$ Let $\rho = [\rho_p, \rho_s]^{\mathrm{T}} \in \mathrm{N}(A)$, and $\rho_p \neq 0$. Since there in so undeclared broadcast, we know that $S\rho > 0$, (we can choose the sign of ρ appropriately). Now, any z_s in the source polyhedron of some ACE, z_p satisfies $S_2 z_p + S_3 z_s \geq s$. Subtracting $S\rho$ from the RHS and simplifying, $S_2(z_p + \rho_p) + S_3(z_s + \rho_s) \geq s$, i.e, $z_s + \rho_s$ is in the source polyhedron of the ACE $z_p + \rho_p$. Also, since $A\rho = 0$, the nodeset of the first ACE is a proper subset of the second, and we have a scan. ∎

Éditeur

INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)

ISSN 0249-6399