The Alignment Problem in a Linear Algebra Framework

Claude G. Diderich[*]
Swiss Federal Inst. of Tech. – Lausanne
Computer Science Department
CH-1015 Lausanne, Switzerland
diderich@di.epfl.ch

Marc Gengler
Ecole Normale Supérieure de Lyon
Labo. de l'Info. du Parallélisme
F-69364 Lyon, France
Marc.Gengler@lip.ens-lyon.fr

Abstract

Two important aspects have to be addressed when automatically parallelizing loop nests for massively parallel distributed memory computers, namely maximizing parallelism and minimizing communication overhead due to non local data accesses. This paper studies the problem of finding a computation mapping and data distributions that minimize the number of remote data accesses for a given degree of parallelism. This problem is called the constant-degree parallelism alignment problem *and is shown to be NP-hard. The algorithm presented uses a linear algebra framework and assumes affine data access functions. It proceeds by enumerating all interesting bases of the set of vectors representing the alignments between computation and data accesses that should be satisfied. It is shown in a comparison with related work how the approach presented allows to express previous results as special cases. The algorithm is applied to benchmark programs and shown superior to more basic mappings.*

1. Introduction

An important problem when parallelizing nested loops for distributed memory parallel computers (DMPC) is how to map the computation and the data onto processors, that is, which processor executes which computation and which processor stores which data element. The optimal solution to this problem is a placement of computation and data onto the processors that minimizes the overall execution time. This problem can be subdivided into two subproblems: i) the *alignment problem* which assigns computation and data to a set of virtual processors, and ii) the *mapping problem* which folds the set of virtual processors onto the physical ones. In this paper we exclusively address the alignment problem. We consider as target architectures massively parallel message passing distributed memory machines in which communication costs are some magnitudes larger than local data accesses or computations. Nevertheless our techniques are also relevant to shared memory machines. Indeed, the problem we consider allows to reduce the interactions between processors due to accesses to a same memory location on a shared memory machine. As a consequence, the number of synchronization barriers can be reduced. Additionally, the global memory bandwidth of interleaved memory systems is increased.

Following the linear algebra formulation of the alignment problem by Huang and Sadayappan in 1991 [14], researchers have primarily focused on finding computation and data alignment functions requiring no remote data accesses [5]. Anderson and Lam [4] have presented a heuristic for minimizing communication. Anderson, Amarasinghe and Lam [3] and Cierniak and Li [7] have studied the alignment problem targeting onto cache coherent distributed shared memory machines using only linear mappings for locality analysis. Lim and Lam [18] presented a set of affine transformations to improve data locality. Darte and Robert [9] reduced uniform alignment constraints to graph theoretic problems. Dion and Robert [12] applied similar techniques to linear access functions. Other researchers have modeled the problem in a graph theoretic framework [6, 17], considered only constant offset data access functions [15, 20] or used stencil-based approaches [8].

This paper is organized as follows. In Sec. 2. we define the alignment problem and show how it can be expressed in a linear algebra framework. Section 3. describes an algorithm for finding a communication-free alignment used as a building block for our algorithm. In Sec. 4. we present a novel algorithm for finding an optimal solution to the constant-degree parallelism alignment problem and discuss various aspects of it. Section 5. features a comparison with related work. In Sec. 6. we describe experimental results when solving the constant-degree parallelism alignment problem for loop nests extracted from various programs, before concluding in Sec. 7.

[*]Research supported by a grant from the Swiss Federal Institute of Technology – Lausanne.

2. The alignment problem

We consider the problem of generating efficient parallel code for single or multiple loop nests. To do so we map the iterations from the sequential loop nests onto different processors. We address the problem of assigning the different array elements accessed to different processors.

The *alignment problem* is the problem of finding an alignment of loop iterations with the array elements accessed, that is, mappings of the loop iterations, called computations, and array elements to a multidimensional grid of virtual processors. The alignment should address the following needs:

i) maximize the degree of parallelism, that is, use as many processors as possible,

ii) minimize the number of non local data accesses, that is, distribute the array elements such that a processor owns a maximal number of the elements it accesses.

Clearly the needs i) and ii) depend on each other. Depending on how the needs i) and ii) are satisfied, various subproblems of the alignment problem can be defined. When allowing only local data accesses, we talk about the *communication-free alignment problem*. Another subproblem is defined by minimizing the number of remote data accesses for a given degree of parallelism. This subproblem is called the *constant-degree parallelism alignment problem*. Other variants are of course possible and interesting. Depending on how the needs are to be taken into account, the problem may or may not be easy to solve. For instance, the communication-free alignment problem is solvable in polynomial time, whereas the constant-degree parallelism alignment problem is NP-hard.

In this paper we restrict ourselves to array access functions that are linear or affine. This permits us to use a linear algebra framework, and in particular the approach presented by Bau *et al.* [5], for expressing the alignment problem. Furthermore we do not consider data replication explicitly as it can be seen as an optimized form of communication. However, programs that use more complicated access patterns may still be dealt with. The principle consists in substituting several affine accesses to, for example, an indirect access, in such a way that the solution will be trivial in the array dimensions concerned. If this trick does not allow to compute any clever data alignment it has the advantage of making our technique applicable to a larger class of programs. In this linear algebra framework, access l to array k is defined by the function

$$\begin{array}{rccl} F_k^l\colon & \mathbb{I} & \longrightarrow & \mathbb{D}_k \\ & \mathbf{i} & \longmapsto & F_k^l(\mathbf{i}) = \mathbf{F}_k^l\,\mathbf{i} + \mathbf{f}_k^l \end{array}$$

where $\mathbb{I}$ represents the index domain defined by the loop bounds and $\mathbb{D}_k$ the array access domain. For example the first access `A(i+j+1,2*j-k+2)` to array `A` in a loop nest of depth three and indices i, j and k, also written as $F_A^1(i,j,k) = (\,i+j+1 \quad 2\times j - k + 2\,)^{\mathsf{T}}$, is encoded by

$$\mathbf{F}_A^1 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & -1 \end{pmatrix} \qquad \text{and} \qquad \mathbf{f}_A^1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

The unknown computation and data mappings can also be written as matrix functions.

$$\begin{array}{rccl} & \multicolumn{3}{c}{\text{Computation map:}} \\ C_j\colon & \mathbb{I} & \longrightarrow & \mathbb{P} \\ & \mathbf{i} & \longmapsto & C_j(\mathbf{i}) = \mathbf{C}_j\,\mathbf{i} + \mathbf{c}_j \\ & \multicolumn{3}{c}{\text{Data map:}} \\ D_k\colon & \mathbb{D}_k & \longrightarrow & \mathbb{P} \\ & \mathbf{a} & \longmapsto & D_k(\mathbf{a}) = \mathbf{D}_k\,\mathbf{a} + \mathbf{d}_k \end{array}$$

where $\mathbb{P}$ represents the virtual multi-dimensional grid of processors. Although $\mathbb{P}$ may be viewed as an unbounded multi-dimensional grid, its size is limited by the number of iteration points in the iteration space $\mathbb{I}$. We are only considering affine functions and are not interested in the exact size of the underlying polytope. $C_j(\mathbf{i})$ represents the processor on which iteration $\mathbf{i}$ of assignment instruction j is executed. Similarly, the function D_k indicates on which processors the elements of array k are located. The order in which each processor is allowed to execute the assigned iterations is given by the data dependences among the data elements accessed and updated. The on-processor iteration scheduling problem is not addressed in this paper.

To have a complete description of the alignment problem, we express the needs i) and ii) in the linear algebra framework. Need i) can be formulated as

$$\max_{\mathbf{C}_j,\mathbf{c}_j}\left(\min_j\left(\operatorname{rank}\left(\mathbf{C}_j^{\mathsf{T}}\right)\right)\right).$$

The actual alignment constraints requiring that the data elements accessed by processor P_q reside on P_q are expressed by the equations

$$\forall\,\mathbf{i}\in\mathbb{I}\colon\ \mathbf{C}_j\,\mathbf{i} + \mathbf{c}_j = \mathbf{D}_k\left(\mathbf{F}_k^l\,\mathbf{i} + \mathbf{f}_k^l\right) + \mathbf{d}_k. \qquad (1)$$

The eqns. (1) are called *alignment constraints* or *locality constraints*. Each eqn. indicates that processor $P_q = \mathbf{C}_j\,\mathbf{i} + \mathbf{c}_j$ executing assignment instruction j at iteration $\mathbf{i}$ owns the data accessed during that iteration which is located on processor $\mathbf{D}_k\left(\mathbf{F}_k^l\,\mathbf{i} + \mathbf{f}_k^l\right) + \mathbf{d}_k$. Need ii) requires that a maximal number of the eqns. (1) be satisfied.

The parallel execution scheme, solution of the alignment problem in the linear algebra framework, is more general than the alignment functions that are implemented by traditional BLOCK or CYCLIC data distribution schemes provided by languages like HPF [16], if abstracting from possible explicit replications. The owner computes restriction is lifted as read and write data accesses are considered at the same level.

We say that an $O(n^q)$ computation has p degrees of *time parallelism* if it executes in $O(n^{q-p})$ time. The same algorithm is said to have p degrees of *processor parallelism* if it uses $O(n^p)$ processors. When talking about degree of parallelism we usually mean processor parallelism. In most cases the degree of time parallelism is equal to the degree of processor parallelism, except in some work-inefficient parallel programs. Consider the following two examples:

```
do i = 2, n
   B(i) = A(i-1)
end do
```

Example a)

```
do i = 2, n
   A(i) = A(i-1)
end do
```

Example b)

It is possible to assign, in both examples, each iteration to a different processor. We say that both examples have one degree of processor parallelism as each of the $O(n)$ processors assigned is used. But only example a) has one degree of time parallelism as it executes in $O(n^{1-1}) = O(1)$ time. This is not the case for example b) in which processor $i+1$ can only compute after processor i has finished its work due to the data dependence between the write of `A(i)` and the read of `A(i-1)`.

3. Solving the communication-free alignment problem

In this section we briefly review the algorithm for solving the communication-free alignment problem in the linear algebra framework presented by Bau *et al.* [5]. Our algorithm, as presented in Sec. 4., uses this algorithm as a building block.

By simple algebraic transformations, the set of eqns. (1) can be rewritten in the following equivalent forms.

$$\forall \mathbf{i} \in \mathbb{I}: (\, \mathbf{C}_j \;\; \mathbf{c}_j \,) \begin{pmatrix} \mathbf{i} \\ 1 \end{pmatrix} = (\, \mathbf{D}_k \;\; \mathbf{d}_k \,) \begin{pmatrix} \mathbf{F}_k^l & \mathbf{f}_k^l \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{i} \\ 1 \end{pmatrix}$$

$$\forall \mathbf{i} \in \mathbb{I}: \begin{pmatrix} \hat{\mathbf{C}}_j & \hat{\mathbf{D}}_k \end{pmatrix} \begin{pmatrix} \mathbf{I} \\ -\hat{\mathbf{F}}_k^l \end{pmatrix} \begin{pmatrix} \mathbf{i} \\ 1 \end{pmatrix} = 0 \qquad (2)$$

where $\hat{\mathbf{C}}_j = (\, \mathbf{C}_j \;\; \mathbf{c}_j \,)$, $\hat{\mathbf{D}}_k = (\, \mathbf{D}_k \;\; \mathbf{d}_k \,)$ and $\hat{\mathbf{F}}_k^l = \begin{pmatrix} \mathbf{F}_k^l & \mathbf{f}_k^l \\ \mathbf{0} & 1 \end{pmatrix}$.

To simplify the problem, as suggested by Bau *et al.* [5], we require that eqns. (2) hold for any vector $\mathbf{i}$, regardless of whether or not it belongs to the iteration domain $\mathbb{I}$. The set of alignment constraints then becomes

$$\begin{pmatrix} \hat{\mathbf{C}}_j & \hat{\mathbf{D}}_k \end{pmatrix} \begin{pmatrix} \mathbf{I} \\ -\hat{\mathbf{F}}_k^l \end{pmatrix} = 0. \qquad (3)$$

Allowing no communication imposes that all locality constraints (3) are satisfied simultaneously. Therefore (3) can be rewritten as

$$\hat{\mathbf{U}}\,\hat{\mathbf{V}} = 0 \qquad (4)$$

where $\hat{\mathbf{U}} = (\, \hat{\mathbf{C}}_1 \cdots \hat{\mathbf{C}}_s \quad \hat{\mathbf{D}}_1 \cdots \hat{\mathbf{D}}_t \,)$, $\hat{\mathbf{V}} = (\, \hat{\mathbf{V}}_{1,1,1} \cdots \hat{\mathbf{V}}_{s,t,u} \,)$ and

$$\hat{\mathbf{V}}_{j,k,l} = \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} & -\hat{\mathbf{F}}_k^l & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix}^{\mathsf{T}}.$$

The sub-matrix $\hat{\mathbf{V}}_{j,k,l}$ represents the alignment constraint of the data access l of array k in instruction j and the processor using that data, that is, the zeros are placed such that $\hat{\mathbf{U}}\hat{\mathbf{V}}_{j,k,l} = \mathbf{0}$ implies $\hat{\mathbf{C}}_j - \hat{\mathbf{D}}_k\hat{\mathbf{F}}_k^l = \mathbf{0}$.

Eqn. (4) is equivalent to $\hat{\mathbf{V}}^{\mathsf{T}}\,\hat{\mathbf{U}}^{\mathsf{T}} = 0$. Therefore, the column vectors of the unknown matrix $\hat{\mathbf{U}}^{\mathsf{T}}$ are in the null space of the known $\hat{\mathbf{V}}^{\mathsf{T}}$. The degree of parallelism associated with an alignment $\hat{\mathbf{U}}$ equals $\min_j(\operatorname{rank}(\mathbf{C}_j^{\mathsf{T}}))$. Because the degree of parallelism is not influenced by the constant offset, it only depends on the rank of the matrices $\mathbf{C}_j$. In fact, there always exists a trivial solution to the problem mapping all computation and data elements onto a single processor. Such a solution would have $\hat{\mathbf{C}}_j = (\, 0 \cdots 0 \mid 1 \,)$, $\hat{\mathbf{D}}_k = (\, 0 \cdots 0 \mid 1 \,)$ and $\operatorname{rank}(\hat{\mathbf{U}}) = 1$. This means that $\operatorname{rank}(\mathbf{C}_j) \leq \operatorname{rank}(\hat{\mathbf{U}})$. Because the degree of parallelism is not influenced by the constant offset, it only depends on the rank of the matrices $\mathbf{C}_j$. The maximal degree of parallelism is tightly related to the size of a basis of the null space of $\hat{\mathbf{V}}^{\mathsf{T}}$. Algo. 1, called LINEAR-ALIGNMENT algorithm, finds a communication-free alignment with maximal degree of parallelism. Bau *et al.* [5] distinguish between the linear-alignment and the affine-alignment algorithms, depending on whether the constant offsets are considered or not. As the principle of both algorithms is strictly the same we do not make such a distinction and use the name linear-alignment indifferently. A more in-depth discussion of the LINEAR-ALIGNMENT algorithm can be found in [5].

4. An exact solution to the constant-degree parallelism alignment problem

In most cases the degree of parallelism of a communication-free alignment is quite low, very often even non-existing. This is for instance the case for matrix-multiply, Gaussian elimination or 2-D FFT. It leads to define the *constant-degree parallelism alignment problem* (CDPAP), which is the problem of finding communication and data mappings such that the degree of parallelism obtained is at least equal to the input parameter d and the number of data communications is minimized.

The value of parameter d, that is the degree of parallelism, may be computed as a function of the maximal number of processors installed in the target system. The problem may also be solved for different values of d. The value

```
Input: A set of alignment constraints of the form $\hat{\mathbf{C}}_j = \hat{\mathbf{D}}_k \hat{\mathbf{F}}^l_k$.
Output: Communication-free alignment matrices $\hat{\mathbf{C}}_j$ and $\hat{\mathbf{D}}_k$.
  1. Assemble matrix $\hat{\mathbf{V}}$ as in eqn. (4).
  2. Compute a basis $\hat{\mathbf{U}}^{\mathsf{T}}$ for the null space of $\hat{\mathbf{V}}^{\mathsf{T}}$.
  3. Set the degree of parallelism to $\min_j(\operatorname{rank}(\mathbf{C}_j^{\mathsf{T}}))$.
  4. Extract the solution matrices $\hat{\mathbf{C}}_j$ and $\hat{\mathbf{D}}_k$ from $\hat{\mathbf{U}}$ as
     defined in eqn. (4).
```

Algorithm 1. The LINEAR-ALIGNMENT algorithm. It computes communication-free alignment functions having a maximal degree of parallelism.

retained of d would be such that the ratio between the parallelism obtained and the number of remote data accesses is maximized. In an interactive system or user supported compilation tool, the programmer could provide a value. In this section we present an algorithm, called the *constant-degree parallelism alignment algorithm* (CDPAA), exactly solving the CDPAP by minimizing the number of non local data accesses.

We assume a communication model in which all processors are directly connected, that is, the time to transmit a message is independent of the source and destination processor. This model is exact for bus based architectures. In many recent designs, the architects have taken pains to build fair approximations of this communication model. The context in which communications are seen is identical to the one used by Feautrier [13].

4.1. The constants-degree parallelism alignment algorithm

Assume that it is possible to find a communication-free alignment of parallelism degree d' for a given problem $\hat{\mathbf{V}}$[1]. Our idea is to simplify the problem $\hat{\mathbf{V}}$ by finding a minimal set of alignment constraints from (1) to be left unsatisfied such that the simplified problem has a solution of degree of parallelism d when solved by the LINEAR-ALIGNMENT algorithm. To increase the parallelism introduced by the communication-free LINEAR-ALIGNMENT algorithm by $d'' = d - d'$, we construct a modified problem $\hat{\mathbf{V}}'$ such that the size of the basis of the null space of that problem is increased by d'' compared to the size of the nullspace of the original problem. This is equivalent to transforming the initial problem $\hat{\mathbf{V}}$ into a simpler problem $\hat{\mathbf{V}}'$ such that the rank of the matrix $\hat{\mathbf{V}}'$, that is, the size of a basis of the column vectors of the matrix $\hat{\mathbf{V}}'$, is reduced by d''. Note that any array accessed having fewer dimensions than the degree of parallelism required can be safely ignored as it is impossible to satisfy the associated alignment constraints.

[1]We use interchangeably the notion of alignment problem and the matrix $\hat{\mathbf{V}}$ representing the locality constraints of the problem.

We use the notation $\langle \hat{\mathbf{V}} \rangle$ to represent the vector space spanned by the column vectors of the matrix $\hat{\mathbf{V}}$. We use the term alignment constraint interchangeably with the column vectors representing that alignment constraint in the matrix of the problem.

$\langle \hat{\mathbf{V}} \rangle$ represents the space of all the alignment constraints. To increase the degree of parallelism by at least d'', we need to find a subspace $\langle \hat{\mathbf{V}}' \rangle$ of $\langle \hat{\mathbf{V}} \rangle$ such that $\dim(\langle \hat{\mathbf{V}} \rangle) - \dim(\langle \hat{\mathbf{V}}' \rangle) \geq d''$. Let $\tilde{d} = \dim(\langle \hat{\mathbf{V}} \rangle) - d''$. There exists an infinite number of such subspaces $\langle \hat{\mathbf{V}}' \rangle$ of dimension $\tilde{d}$, but only finitely many are of interest to us. In fact, all subspaces of $\langle \hat{\mathbf{V}} \rangle$ that contain less than $\tilde{d}$ vector columns of $\hat{\mathbf{V}}$ are uninteresting, because we know that there exists at least one subspace containing at least $\tilde{d}$ column vectors of $\hat{\mathbf{V}}$. Furthermore, the set of all the subspaces of degree at most $\tilde{d}$ containing at least $\tilde{d}$ column vectors can be easily enumerated. To do so, we select any $\tilde{d}$ column vectors of $\hat{\mathbf{V}}$. Complexity issues of this enumeration are discussed n Sec. 4.3. Then, for each valid subset of column vectors of $\hat{\mathbf{V}}$, we compute a basis and count the number of alignment constraints that can be expressed in that basis. A subset is called valid if the subproblem formed by all the alignment constraints containing any of the vectors used for constructing the subset does have a communication-free alignment solution having d degrees of parallelism. As indicated previously, the degree of parallelism obtained without this test, may be $d - 1$ due to the constant offsets in $\hat{\mathbf{C}}_j$.

Finally, we select a subspace $\langle \hat{\mathbf{V}}'^* \rangle$ that contains the largest number of alignment constraints. Other cost functions are possible and are discussed in Sec. 4.4. The simplified problem to be solved thereafter by the communication-free alignment algorithm is the one containing only the locality constraints of the initial problem that lie within $\langle \hat{\mathbf{V}}^* \rangle$. If there are several possibilities to choose the subspace $\langle \hat{\mathbf{V}}'^* \rangle$, are possible, we retain the one whose associated alignment functions are the simplest. Formally the CDPAA is described as Algo. 2.

The presented alignment framework allows to generate correct SPMD code out of any alignment functions C_j and D_k computed. Because the different data dependences as well as the processors on which the data elements accessed are located are known, it is possible, although not necessarily easy, to insert synchronization primitives such that data dependence constraints are satisfied. This scheduling problem can be solved by using techniques for enumerating discrete points in a polyhedron [2].

$$\hat{\mathbf{V}} = \left(\begin{array}{ccc|ccc|ccc|ccc}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
\hline
-1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 1 & -1 & 0 & -1 \\
0 & -1 & 0 & 0 & -1 & 1 & 0 & -1 & 0 & 0 & -1 & 2 \\
0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 \\
\mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 & \mathbf{v}_5 & \mathbf{v}_6 & \mathbf{v}_7 & \mathbf{v}_8 & \mathbf{v}_9 & \mathbf{v}_{10} & \mathbf{v}_{11} & \mathbf{v}_{12} \\
\multicolumn{3}{c|}{\texttt{A(i,j)}} & \multicolumn{3}{c|}{\texttt{A(i,j-1)}} & \multicolumn{3}{c|}{\texttt{A(i-1,j)}} & \multicolumn{3}{c}{\texttt{A(i+1,j-2)}}
\end{array}\right).$$

Figure 1. Matrix $\hat{\mathbf{V}}$.

4.2. Example

Let us consider the following single loop nest[2]:

```
do i = 1, n
   do j = 1, n
      B(i,j) = A(i,j) + A(i,j-1)
         + A(i-1,j) + A(i+1,j-2)
   end do
end do
```

Computing a communication-free alignment for this problem, by using Algo. 1, yields the trivial solution $C(i,j) = 1$ and $D_A(i,j) = 1$, that is no parallelism. We want to obtain computation and data mappings having $d = 1$ degree of parallelism. Such an alignment is reasonable if the number of processors available is smaller than the loop bound n.

For this example we assemble the matrix $\hat{\mathbf{V}}$ as show in Fig. 1.[3] As $\mathrm{size}(\ker(\hat{\mathbf{V}}^{\mathsf{T}})) = 1$ and $\mathrm{rank}(\mathbf{C}) = 0$, we have $\tilde{d} = 5 - 1 = 4$. By enumerating all subspaces $\langle\hat{\mathbf{V}}'\rangle$ of $\langle\hat{\mathbf{V}}\rangle$ having as basis $\tilde{d}$ vector columns of $\hat{\mathbf{V}}$ and counting for each subspace $\langle\hat{\mathbf{V}}'\rangle$ the number of alignment constraints, that is, sets of columns of $\hat{\mathbf{V}}$, that lie within it, we find that the subspace having as basis $(\mathbf{v}_4, \mathbf{v}_5, \mathbf{v}_6, \mathbf{v}_9)$ contains a maximal number of alignment constraints. As the alignment of the computation with the data access `A(i,j)`, that is vector $\mathbf{v}_3$, cannot be expressed within that basis, the simplified problem is formed of the alignment constraints aligning the accesses `A(i,j-1)`, `A(i-1,j)` and `A(i+1,j-2)` with the computation of iteration `(i,j)`. Solving this simplified problem using Algo. 1, we find the following computation and data alignment functions

$$C(i,j) = i + j - 1 \qquad \text{and} \qquad D_A(i,j) = i + j.$$

Under this alignment, each loop iteration requires three local and one remote memory accesses to array `A`.

[2]The CDPAA is not restricted to single loop nests. It may be applied to multiple loop nests or even whole programs.

[3]To simplify the notation an without loss of generality we only consider data accesses to array `A`.

4.3. Optimality and complexity study

The input size of any CDPAP is characterized by four parameters, which are the number of data accesses n, the maximal dimension of any array accessed or the maximal loop nest depth e, whichever is larger, the number of assignment statements considered c and the number of arrays a.

To study the complexity of the CDPAP, let us consider the following problem. Let $A = \{\mathbf{a}_i \cdot \mathbf{x} = 0\}$ be a set of n linear equations with m variables such that $a_{i,j} \in \{-1, +1\}$. The *homogeneous bipolar maximal feasible linear subsystem problem*, denoted by MAX FLS$^=$, is the problem of finding a non trivial solution vector $\mathbf{x} \in \mathbb{R}^m \backslash \{\mathbf{0}\}$ that satisfies as many of the linear equations in A as possible. Amaldi and Kann [1] have shown the decision version associated with this optimization problem to be NP-complete. We proof the following theorem by reduction from MAX FLS$^=$.
Theorem 4.1 *The decision version of the CDPAP, that is, given a set of alignment constraints $\hat{\mathbf{C}}_j = \hat{\mathbf{D}}_k \hat{\mathbf{F}}_k^l$ and constants d and K, do there exist computation and data mapping functions $\hat{\mathbf{C}}_j$ and $\hat{\mathbf{D}}_k$ having d degrees of processor parallelism such that, at least K alignment constraints are satisfied, is NP-complete.*
Proof. Let A be the decision version of a MAX FLS$^=$ problem and K a decision constant. We transform the problem A into m constant-degree parallelism alignment problems A'_q with required degree of parallelism $d = 1$. The constant-degree parallelism alignment problem q is constructed as follows. Each equation $\mathbf{a}_i \cdot \mathbf{x} = 0$ is transformed into one alignment constraint. Each non satisfied alignment constraint represents one remote data access. Each $\mathbf{a}_i$ is multiplied by 1 or -1 to have $a_{i,q} = 1$. Then alignment constraint i is constructed as i Fig. 2 where $\mathbf{X} = (\, x_q \; y \; x_1 \; \cdots \; x_{q-1} \; x_{q+1} \cdots x_m \; z \,)^{\mathsf{T}}$. The affine data access functions of the obtained alignment problem are

$$F_i(i_1, \ldots, i_{m-1}) = (-a_{i,1}\, i_1, \ldots, -a_{i,q-1}\, i_{q-1}, -a_{i,q+1}\, i_q, \ldots, -a_{i,m}\, i_{m-1}).$$

If there exists a solution to the q-th CDPAP a'_q that satisfies at least K alignment constraints, then $(x_1, \ldots, x_k)$ of X_q^*,

$$\left\{\begin{array}{cccccccccc} a_{i,q}\,x_q & & + & a_{i,1}\,x_1 & + & \cdots & + & a_{i,m} & & = & 0 \\ & y & & & & & & & + & z & = & 0 \end{array}\right.$$

$$\Leftrightarrow \left(\begin{array}{cc|cccccc} 1 & 0 & a_{i,1} & \cdots & a_{i,q-1} & a_{i,q+1} & \cdots & a_{i,m} & 0 \\ 0 & 1 & 0 & \cdots & & & \cdots & 0 & 1 \end{array}\right) \mathbf{X} = \mathbf{0}$$

Figure 2. Construction of an alignment constraint from an affine equation.

which is different from $\mathbf{0}$ because $x_q \neq 0$, satisfies at least K linear equations. Converse, assume that there exists a vector $\mathbf{x} \neq \mathbf{0}$ satisfying at least K equations of A and none of the m CDPAP A'_q has a solution satisfying at least K alignment constraints. Assume that $x_r \neq 0$. Such an r always exists. Then $X_r = (x_1, 1, x_2, \ldots, x_{r-1}, x_{r+1}, \ldots, x_m, -1)$ is a solution to A'_r satisfying at least K alignment constraints. This is in contradiction to the hypothesis that not such solution exists. Therefore, if no solution exists to any of the m CDPAP satisfying at least K alignment constraints, then there does not exist any solution vector $\mathbf{x}$ verifying at least K linear equations. This concludes the reduction. As the decision version of MAX FLS$^=$ is NP-complete and as the CDPAP belongs to NP, we conclude that the CDPAP is NP-complete. □

Theorem 4.2 *The CDPAA finds communication and data alignment functions that require a minimal number of non local data accesses for a given degree of processor parallelism. Furthermore, if all alignment constraints representing data dependences are satisfied, the degree of processor parallelism equals the degree of time parallelism.*
Proof. Suppose that there exist better communication and data mapping functions, that is, requiring fewer non local data accesses, than the ones computed by the CDPAA. This is not possible because a basis of the subspace representing these better mapping functions would have been enumerated by the CDPAA.

There exist two reasons for the processor and time parallelism being not equal. First, the load may be unevenly balanced. In the CDPAA this is not the case as each virtual processor is assigned the same number of iterations up to a constant factor. Second, any processor may have to wait for the data to become available. But, if all alignment constraints representing data dependences are satisfied, then no data element written by one processor is read by another one. □

If not all alignment constraints representing data dependences are satisfied, the degree of processor parallelism may or may not equal the degree of time parallelism.
Theorem 4.3 *The CDPAA requires $O((e \cdot n)^{4+e(c+a)})$ time to find optimal alignment functions.*
Proof. Each linear algebra operation, like computing the rank of a matrix, testing whether or not a vector lies within a given subspace, can be computed in $O((e \cdot n)^3)$ time as the matrix $\hat{\mathbf{V}}$ is of size $e \cdot n \times e(c+a)$ and $c + a \leq 2\ n$. The most time consuming part of the algorithm is the enumeration of all subsets of $\tilde{d}$ column vectors of $\hat{\mathbf{V}}$. $O(C_{e \cdot n}^{\tilde{d}})$ such vector sets exist. As one can verify $\tilde{d} \leq \mathrm{size}(\mathrm{base}(\langle \hat{\mathbf{V}} \rangle)) \leq e\ (c+a)$. Therefore $C_{e \cdot n}^{\tilde{d}} \leq (e \cdot n)^{e\ (c+a)}$ is obtained. Counting the number of alignment constraints lying within a subspace $\langle \hat{\mathbf{V}}' \rangle$ can be done in $O((e \cdot n)^4)$ time. Combining these results, we obtain the running time complexity of $O((e \cdot n)^{4+e\ (c+a)})$. □

If the complexity of computing the cost function minimized by the CDPAA is $f(n, e, c, a)$, then the result of Theorem 4.3 generalizes to $O((e \cdot n)^{e\ (c+a)} \cdot f(n, e, c, a))$.

Because of the NP-hardness results, it is not astonishing that the running time of the CDPAA is non polynomial. This is not as bad as it may look at first. It is reasonable to assume that n, the number of array accesses is much larger than any of the other three parameters. This is especially true when considering one loop nest at a time. Therefore, when only considering the number of array accesses as variable, the CDPAA execution time is polynomial.

4.4. On the cost function

We use a counting argument based on the number of non local data accesses as optimization function for computing efficient alignment functions. Although some people argue that minimizing only the number of data accesses is not precise enough, we believe that, for a generic message passing DMPC, minimizing the number of communications is essential. It is nevertheless true that a minimal number of remote data accesses is not a sufficient condition to get performance. Among the problems that remain to be addressed are, communication specialization and vectorization to minimize startup overheads [19], data layout on individual processors and per processor iteration scheduling to maximize cache performance. These techniques [3, 7] are usually considered in a second step and can always be applied with our approach.

The cost function used can be generalized by assigning different weights to the different alignment constraints

```
Input: A set of alignment constraints of the form $\hat{\mathbf{C}}_j = \hat{\mathbf{D}}_k \hat{\mathbf{F}}_k^l$ and a constant $d$.
Output: Alignment matrices $\hat{\mathbf{C}}_j$ and $\hat{\mathbf{D}}_k$ representing parallelism of degree at least $d$.
  1. Assemble matrix $\hat{\mathbf{V}}$ as in eqn. (4).
  2. Solve the problem as a communication-free alignment problem. Let $d'$ be the degree of parallelism obtained.
  3. Let $\tilde{d} = \mathrm{rank}(\hat{\mathbf{V}}) - (d - d')$.
4.1. Enumerate all subspaces $\langle \hat{\mathbf{V}}' \rangle$ of dimension $\tilde{d}$ of $\langle \hat{\mathbf{V}} \rangle$ by selecting as a basis $\tilde{d}$ of the vector columns of $\hat{\mathbf{V}}$.
4.2. Check if the subspace is valid, that is, if the subproblem formed of all alignment constraints containing any of the selected basis vectors does have a communication-free alignment solution having at least $d$ degrees of parallelism.
4.3. For each valid subspace $\langle \hat{\mathbf{V}}' \rangle$, count how many alignment constraints are in that subspace.
  5. Select a subspace $\langle \hat{\mathbf{V}}'^* \rangle$ that contains a maximal number of alignment constraints, as computed in 4.3.
  6. Form a new problem $\hat{\mathbf{V}}'$ containing all the alignment constraints that can be expressed in the subspace $\langle \hat{\mathbf{V}}'^* \rangle$ computed in 5.
  7. Solve the new simplified problem $\hat{\mathbf{V}}'$ by using the communication-free alignment algorithm LINEAR-ALIGNMENT.
```

Algorithm 2. The constant-degree parallelism alignment algorithm (CDPAA).

depending on their importance and then minimize the weighted sum of remote data accesses. Such a cost function even allows the user of the algorithm to require some alignment constraints to be satisfied by assigning to them an infinite weight. For example, the owner computes rule can be imposed by assigning a weight of $+\infty$ to the alignment constraint $\hat{\mathbf{C}}_j = \hat{\mathbf{D}}_k \hat{\mathbf{F}}_k^1$, where $\hat{\mathbf{F}}_k^1$ represents the element of array k being modified by instruction j. Any conceivable and efficiently computable cost function can be used to select the d' basis vectors in step 5 of Algo. 2.

The definition and use of specialized cost functions is especially interesting in a heuristic environment.

4.5. Some remarks on data dependence constraints

As long as all alignment constraints are verified, data dependences are as well, and the processor parallelism is identical to the time parallelism. This means that the data and computation distributions are compatible. But, as soon as alignment constraints are dropped this may no longer be true. Removing alignment constraints that represent part of data dependences may increase the processor parallelism without changing the time parallelism. In [11] we characterize the relation between the computed alignment and the iteration scheduling, that is, the relation between processor and time parallelism. Essentially, we show that a sufficient, but not necessary, condition to get a non constant number of active processors during each time step of a linear scheduling is to impose that there be at least two fulfilled alignment constraints that correspond to a data dependence.

4.6. A heuristic approach to the problem

As we show in Sec. 4.3., the CDPAP is NP-hard. Even for simple loop nests the running time may be large. In practice, a reasonably good feasible solution found rapidly is sufficient. This lead us to the development of a heuristic for finding a feasible solution to the CDPAP. It proceeds by incrementally constructing a single basis of a subspace containing verified alignment constraints. The alignment constraints are considered one at a time in a predefined order to conveniently express the locality constraints that are considered the most important. The details of this heuristic approach can be found in [10].

5. Comparison with related work

Many techniques for solving the alignment problem proposed by different research teams are closely related to the approach taken in this paper. Huang and Sadayappan [14] were the first to introduce communication-free hyperplane partitioning of loops. Their paper states necessary and sufficient conditions on computation and data mappings to obtain alignments that do not need any communication. Subsequent efforts have focussed on algorithms for computing communication-free alignment functions based on the conditions of Huang and Sadayappan [4, 5]. Others have concentrated their work on finding a subset of data accesses that verify the communication-free alignment conditions for a given degree of parallelism [4, 12, 13]. In this Sec. we review this latter work and compare it to our approach. To do so we use the linear algebra framework of Bau *et al.* [5] presented in Sec. 2.

Anderson and Lam (1993). Anderson and Lam [4] define necessary conditions for the data accesses executed by each processor being local. Using the notation introduced in Sec. 2., these conditions are

$$\begin{aligned}
\ker(\mathbf{D}_k) &\supseteq \mathrm{span}\{\mathbf{s} \mid \mathbf{s} = \mathbf{F}_k^l\,\mathbf{t},\ \mathbf{t} \in \ker(\mathbf{C}_j), \\
&\qquad \text{for all accesses } l \text{ and all instructions } j \\
&\qquad \text{in which array } k \text{ is accessed}\} \qquad (5) \\
\ker(\mathbf{C}_j) &\supseteq \mathrm{span}\{\mathbf{t} \mid \mathbf{t} \in \ker(\mathbf{F}_k^l) \vee
\end{aligned}$$

$$\mathbf{F}_k^l\ t \in (\ker(\mathbf{D}_k) \cap \mathrm{range}(\mathbf{F}_k^l)), \text{ for all accesses } l \text{ and all arrays } k \text{ accessed in instruction } j\} \quad (6)$$

These conditions admit a direct translation into the framework defined by Bau *et al.* [5]. Condition (5) expresses that if iterations $\mathbf{i}_1$ and $\mathbf{i}_2$ are mapped to the same processor, that is $\mathbf{t} = \mathbf{i}_1 - \mathbf{i}_2 \in \ker(\mathbf{C}_j)$, then all accesses l in instruction j to array k are mapped to the same processor, that is, $\mathbf{F}_k^l\ \mathbf{t} \in \ker(\mathbf{D}_k)$. In the framework of Bau *et al.* this means that $\mathbf{D}_k\ \mathbf{F}_k^l\ \mathbf{i}_1 = \mathbf{D}_k\ \mathbf{F}_k^l\ \mathbf{i}_2$. Converse, if all array elements of arrays k are accessed in instruction j, then they are mapped to the same processor, which is expressed by condition (6). More formally, this means that if $\mathbf{t} = \mathbf{i}_1 - \mathbf{i}_2 \in \ker(\mathbf{D}_k\ \mathbf{F}_k^l)$, then $\mathbf{t} \in \ker(\mathbf{C}_j)$. As $\ker(\mathbf{F}_k^l) \subseteq \ker(\mathbf{D}_k\ \mathbf{F}_k^l)$, we conclude that, if $\mathbf{t} \in \ker(\mathbf{F}_k^l)$, then the two iterations must also be mapped onto the same processor. Again, this means that $\mathbf{C}_j\ \mathbf{t} = \mathbf{D}_k\ \mathbf{F}_k^l\ \mathbf{t}$. Note that there is a third case to consider, which reduces to the second. If two iterations $\mathbf{i}_1$ and $\mathbf{i}_2$ access the same data element of array k, that is $\mathbf{F}_k^l\ \mathbf{i}_1 = \mathbf{F}_k^m\ \mathbf{i}_2$, then iterations $\mathbf{i}_1$ and $\mathbf{i}_2$ must be mapped onto the same processor. From $\mathbf{t} = \mathbf{i}_1 - \mathbf{i}_2 \in \ker(C_j)$, we conclude, using $\mathbf{C}_j\ \mathbf{i} = \mathbf{D}_k\ \mathbf{F}_k^l\ \mathbf{i}$, that the condition $\mathbf{t} \in \ker(\mathbf{D}_k\ \mathbf{F}_k^l)$ must also be satisfied. The same holds for $\mathbf{F}_k^m$.

Anderson and Lam present a greedy algorithm to compute the alignment functions $\mathbf{C}_j$ and $\mathbf{D}_k$ that can be satisfied, incrementally adding constraints as long as the conditions (5) and (6) are satisfied, starting with the most frequently used array access functions. Such heuristic techniques are close to the one mentioned in this paper. However, Anderson and Lam base their heuristic exclusively on the number of times the corresponding instruction is executed. Opposed to this, our heuristic function briefly mentioned in Sec. 4.6. and described in detail in [10] takes several aspects into account. These include the number of executions of each data access, the linear and constant offset parts of the data access functions and, most importantly, data dependence information, to yield an alignment that is compatible with a scheduling. Anderson and Lam only consider the linear part of the data access functions, taking care of the constant offsets in a second step.

Feautrier (1994). Feautrier [13] addresses the problem of finding an alignment function that maps the data elements on a one-dimensional grid of virtual processors. The alignment constraints between computation and data accesses are derived from the data-flow graph (DFG) of the program, procedure or loop nest considered. The DFG is a directed graph. Vertices correspond to statements and the arcs to producers and consumers of data. The computation mapping function is defined by the owner computes rule which is imposed. For each statement, the alignment function is assumed to be an affine function of the iteration vectors with unknown parameters. The locality of accesses is imposed by asking that the producer and the consumer of a data item be the same processor. Feautrier defines a distance vectors between any pair of producers and consumers. To any arc of the DFG corresponds a distance vector that expresses the difference of the indices of the processor that computes the data and the one that uses it. Thus, a communication is local if and only if the corresponding distance vector is zero. The edges are hence transformed into affine equations and the problem consists in determining non-trivial parameters for the computation mappings that zero out as many distance vectors as possible. A heuristic is the used to sort the equations in decreasing order of the communication traffic induced. The system of equations, which usually does not have a non trivial solution, is then solved by successive Gauss-Jordan eliminations as long as a feasible solution remains non-trivial.

The approach presented by Feautrier is similar to the heuristic method mentioned in Sec. 4.6., especially as both approaches handle affine access functions. Nevertheless the techniques as well as the heuristics are quite different in the sense that Feautrier uses Gauss-Jordan elimination to construct a feasible solution whereas our approach is based on constructing a subspace in which the alignment constraints can be expressed. Furthermore, Feautrier simplifies the computation mapping problem by using the owner computes rule, whereas our approach allows any affine computation mapping.

Dion and Robert (1995). The problem considered by Dion and Robert [12] is also close to our interest, although the techniques used and the hypotheses made on the problem instances are quite different. They compute, considering only the linear parts, the largest set of alignment constraints that can be met while yielding a given degree of parallelism d. The constant offsets are considered subsequently, using techniques developed by Darte and Robert [9]. All data access functions must be of full rank and no smaller than d. This is the only way they can assure that the parallelism obtained is indeed as large as wanted. We consider a set of candidate solutions and search for an optimal one that verifies the largest number of constraints while effectively yielding the degree of parallelism desired.

In their approach Dion and Robert consider the following three basic cases.

- If the access matrix $\mathbf{F}_k^l$ is square, then it is invertible. Hence, using the basic condition $\mathbf{C}_j = \mathbf{D}_k\ \mathbf{F}_k^l$, it is possible to either derive a computation mapping $\mathbf{C}_j$ of rank d from a given data mapping $\mathbf{D}_k$ of rank d, that is $\mathbf{C}_j = \mathbf{D}_k\ \mathbf{F}_k^l$, or a data mapping $\mathbf{D}_k$ of rank d from a given computation mapping $\mathbf{C}_j$ of rank d, that is $\mathbf{D}_k = \mathbf{C}_j\ \mathrm{inv}(\mathbf{F}_k^l)$, where inv denotes the inverse.
- If the array access functions considered $\mathbf{F}_k^l$ is narrow,

that is, having fewer columns than lines, then it is possible to consider a left inverse $\mathrm{inv}(\mathbf{F}_k^l)$ and, given a computation mapping $\mathbf{C}_j$ of rank d, compute a data mapping $\mathbf{D}_k$ of rank d as $\mathbf{D}_k = \mathbf{C}_j \; \mathrm{inv}(\mathbf{F}_k^l)$.

- Symmetrically, if $\mathbf{F}_k^l$ is a flat data access function, that is, with fewer rows than columns, then it is possible to determine a computation mapping $\mathbf{C}_j = \mathbf{D}_k \; \mathbf{F}_k^l$ of rank d from a given data mapping $\mathbf{D}_k$ of rank d. Additionally, as $\mathbf{F}_k^l$ admits a right inverse, it is possible to derive a data mapping $\mathbf{D}_k = \mathbf{C}_j \; \mathrm{inv}(\mathbf{F}_k^l)$ of degree d from a given computation mapping $\mathbf{C}_j$ of degree d, provided that the equality $\mathbf{C}_j = \mathbf{C}_j \; \mathrm{inv}(\mathbf{F}_k^l) \; \mathbf{F}_k^l$ holds.

Next, Dion and Robert build a directed graph defined as follows. Vertices correspond either to statements or arrays. There is an arc from vertex p to vertex q if and only if a mapping of rank d can be computed for q from a given mapping of rank d for p according to the basic cases enumerated previously. In this graph they search for a tree containing a maximal number of arcs. Obviously, choosing a mapping of rank d for the root of the computed tree implicitly determines mappings of rank d for all other vertices.

The way Dion and Robert compute one mapping given another mapping is a particular case of the framework defined by Bau *et al.* [5]. This is immediate for square access functions. For a narrow access function $\mathbf{F}_k^l$, Dion and Robert define $\mathbf{D}_k = \mathbf{C}_j \; \mathrm{inv}(\mathbf{F}_k^l)$. Multiplying this equality by $\mathbf{F}_k^l$, we obtain $\mathbf{C}_j = \mathbf{D}_k \; \mathbf{F}_k^l$, that is $(\mathbf{C}_j \; \mathbf{D}_k) \; (\mathbf{I} \; - \mathbf{F}_k^l)^\mathsf{T} = \mathbf{0}$. There are two cases for flat access functions. The case $\mathbf{C}_j = \mathbf{D}_k \; \mathbf{F}_k^l$ is obvious. Let us consider the case $\mathbf{D}_k = \mathbf{C}_j \; \mathrm{inv}(\mathbf{F}_k^l)$, provided that the condition $\mathbf{C}_j = \mathbf{C}_j \; \mathrm{inv}(\mathbf{F}_k^l) \; \mathbf{F}_k^l$ holds. Multiplying by $\mathbf{F}_k^l$, we get $\mathbf{D}_k \; \mathbf{F}_k^l = \mathbf{C}_j \; \mathrm{inv}(\mathbf{F}_k^l) \; \mathbf{F}_k^l$ which reduces to $\mathbf{C}_j = \mathbf{D}_k \; \mathbf{F}_k^l$, taking into account the condition. Thus, the graph built by Dion and Robert can alternatively be expressed exactly in the framework by Bau *et al.* As the technique proposed in this paper allows us to compute, for any given degree of parallelism, a solution that satisfies as many constraints as possible, this solution is equivalent to the one computed by the algorithm of Dion and Robert, when restricted to access functions of full rank.

General considerations. The framework presented in this paper allows to take into account non-local accesses with only constant offsets $F(\mathbf{i}) = \mathbf{I} \; \mathbf{i} + \mathbf{f}_l^k$, linear access functions of the form $F(\mathbf{i}) = \hat{\mathbf{F}}_l^k \; \mathbf{i}$ as well as any general affine access function $F(\mathbf{i}) = \hat{\mathbf{F}}_l^k \; \mathbf{i} + \mathbf{f}_l^k$. Dion and Robert [12], as well as Anderson *et al.* [3], for instance, only consider the linear part of the data access functions in their first step and introduce the constant offsets subsequently.

Finally cache optimizations like the one considered by Anderson *et al.* [3] and Cierniak and Li [7], can be introduced into our framework. Basically, the data is distributed first and the cache reuse for the data local to a given processor is optimized second, using techniques like strip-mining.

6. Experimental results

To show the quality of the alignment functions computed by the CDPAA, we have applied the algorithm to various loop nests of different depths extracted from various programs and benchmarks. In each example[4] we search for alignment functions having at least one degree of parallelism. In Table 1 we list the results of the CDPAA and the heuristic constant degree parallelism alignment algorithm (HCDPAA) [10] as well as the simple data mapping function $D(\mathbf{i}) = i_m$, for the best value of m, that is, the one minimizing the number of remote data accesses. The last data mapping function can be implemented in HPF by using a `BLOCK` distribution scheme. Although the HCDPAA is not described in detail in this paper, it is interesting to compare its performance to the one of the exact CDPAA to show that it is possible to develop efficient heuristics.

Problems `calc1` and `calc2` are the main loop nests in the subroutines of the same name from the program `SHALOW`, a weather prediction benchmark program. `psinv` and `resid` are loop nests extracted from the NAS 2.0 benchmark MG. `jacobi` is the iterative relaxation method of Jacobi. `example` is the small example from [11] and `diag` corresponds to the example described in Sec. 4.2. All problems marked with a star ($\star$) use data access functions of the form $F(\mathbf{i}) = \mathbf{I} \; \mathbf{i} + \mathbf{c}$ where $\mathbf{I}$ is the identity matrix and $\mathbf{c}$ a constant vector.

The notation xL/yR ($z\%$) indicates that each virtual processor executes x local and y remote data accesses, z being the percentage of local accesses. When mapping virtual processors to physical ones by blocking consecutive iterations onto one processor, the number of remote data accesses only applies to iterations within the boundary regions. Such a blocking operation is always possible. When talking about massively parallel machines, as we assume in this paper, the number of virtual processors mapped to one physical one is small, and therefore the number of remote data access becoming local on the physical machine is small. Furthermore, the number of boundaries does not change considerably between different alignment functions. This means that if a given percentage of the remote data accesses become local for an alignment α, then about the same percentage of the remote data accesses become local for an alignment β.

As can be seen from Table 1, the CDPAA and the HCDPAA outperform the simple alignment function $D_m(\mathbf{i})$ on all but one example. Except for the loop nest from `SHALOW` the heuristic presented find optimal computation and data mappings on all tested examples.

[4]We only consider loop nests that do not admit a communication-free alignment.

Pb.name	(1)	(2)	(3)	$D(\mathbf{i}) = i_m$	CDPAA	HCDPAA
`SHALOW`						
`calc1`$\star$	3	7	27	16L/11R (60%)	22L/5R (81%)	20L/7R (74%)
`calc2`$\star$	3	10	27	16L/11R (60%)	21L/6R (78%)	19L/8R (70%)
`NAS2-MG`						
`psinv`$\star$	3	4	21	15L/6R (71%)	19L/2R (90%)	19L/2R (90%)
`resid`$\star$	3	5	21	15L/6R (71%)	19L/2R (90%)	19L/2R (90%)
`jacobi`$\star$	1	2	5	3L/2R (60%)	3L/2R (60%)	3L/2R (60%)
`diag`$\star$	1	2	5	3L/2R (60%)	4L/1R (80%)	4L/1R (80%)
`example`	1	2	4	2L/2R (50%)	3L/1R (75%)	3L/1R (75%)

Table 1. Experimental results for different loop nests. (1) number of instructions, (2) number of different arrays, (3) number of array accesses, xL/yR (z%) number of local and remote data accesses as well as the percentage of local data referenced per iteration step.

7. Conclusion

We have shown how he linear algebra framework can be used to solve the CDPAP. The approach chosen is general, in the sense that previous work can be expressed as special cases. Different aspects of the problem have been addressed. We proved the decision version of the problem to be NP-complete and introduced an exact algorithm for finding a solution.

In future work we are investigating into incorporating into the linear algebra framework, or an extension of it, the notion of scheduling vector. This would allow to optimize a singe function when solving both the scheduling and alignment problems.

References

[1] E. Amaldi and V. Kann, *The complexity and approximability of finding maximum feasible subsystems of linear relations*, Theoret. Comput. Sci. **147** (1995), no. 1–2, 181–210.

[2] C. Ancourt and F. Irigoin, *Scanning polyhedra with DO loops*, Proc. PPoPP '91, April 1991, pp. 39–50.

[3] J. M. Anderson, S. P. Amarasinghe, and M. S. Lam, *Data and computation transformations for multiprocessors*, Proc. PPoPP '95 (Santa Barbara, CA), July 1995.

[4] J. M. Anderson and M. S. Lam, *Global optimizations for parallelism and locality on scalable parallel machines*, Proc. PLDI '93 (Albuquerque, NM), June 1993, pp. 112–125.

[5] D. Bau, I. Kodukula, V. Kotlyar, K. Pingali, and P. Stodghill, *Solving alignment using elementary linear algebra*, Proc. LCPC '94 (Ithaca, NY) (K. Pingali *et al.*, ed.), LNCS, vol. 892, Springer Verlag, August 1994, Also as technical report, TR95-1478, Cornell University, Ithaca, NY, pp. 46–60.

[6] S. Chatterjee, J. R. Gilbert, and R. Schreiber, *The alignment-distribution graph*, Proc. LCPC '93 (Portland, OR) (U. Banerjee *et al.*, ed.), LNCS, vol. 768, Springer Verlag, August 1993, pp. 234–252.

[7] M. Cierniak and W. Li, *Unifying data and control transformations for distributed shared-memory machines*, Proc. PLDI '95 (La Jolla, CA), June 1995, pp. 205–217.

[8] P. Crooks and R. H. Perrott, *An automatic data distribution generator for distributed memory MIMD machines*, Proc. Workshop on Comp. for Par. Comp. (Delft, The Netherlands) (H. J. Sips, ed.), Univ. of Delft, December 1993, pp. 33–44.

[9] A. Darte and Y. Robert, *Mapping uniform loop nests onto distributed memory architectures*, Par. Comp. **20** (1994), no. 5, 679–719.

[10] C. G. Diderich and M. Gengler, *A heuristic approach for finding a solution to the constant-degree parallelism alignment proble*, Proc. of PACT '96 (Boston, MA), October 1996.

[11] ———, *Solving the constant-degree parallelism alignment problem*, Tech. Report DI-96/195, Swiss Fed. Inst. of Tech. – Lsn., Comp. Sci. Dept., Lausanne, Switzerland, June 1996.

[12] M. Dion and Y. Robert, *Mapping affine loop nests: New results*, Proc. HPCN '95 (B. Hertzberger and G. Serazzi, eds.), LNCS, no. 919, Springer-Verlag, 1995, pp. 184–189.

[13] P. Feautrier, *Toward automatic distribution*, Par. Proc. Letters **4** (1994), no. 3, 233–244.

[14] C.-H. Huang and P. Sadayappan, *Communication-free hyperplane partitioning of nested loops*, Proc. LCPC '91 (Santa Clara, CA) (U. Banerjee *et al.*, ed.), LNCS, vol. 589, Springer Verlag, August 1991, pp. 186–200.

[15] K. Knobe, J. D. Lukas, and G. L. Steele, Jr., *Data optimization: Allocation of arrays to reduce communication on SIMD machines*, J. on Par. and Dist. Comp. **8** (1990), no. 2, 102–118.

[16] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele Jr., and M. E. Zosel, *The High Performance Fortran handbook*, MIT Press, Cambridge, MA, 1994.

[17] J. Li and M. Chen, *Index domain alignment: Minimizing cost of cross-referencing between distributed arrays*, Proc. FRONTIERS '90 (Maryland, MA) (J. JaJa, ed.), IEEE Computer Society Press, October 1990, pp. 424–733.

[18] A. W. Lim and M. S. Lam, *Communication-free parallelization via affine transformations*, Proc. LCPC '94 (Ithaca, NY) (K. Pingali *et al.*, ed.), LNCS, no. 589, Springer Verlag, August 1994, pp. 92–106.

[19] A. Platonoff, *Automatic data distribution for massively parallel computers*, Proc. Workshop on Comp. for Par. Comp. (Malaga, Spain), June 1995, pp. 555–570.

[20] B. Sinharoy and B. K. Szymanski, *Data and task alignment in distributed memory architectures*, J. on Par. and Dist. Comp. **21** (1994), no. 1, 61–74.